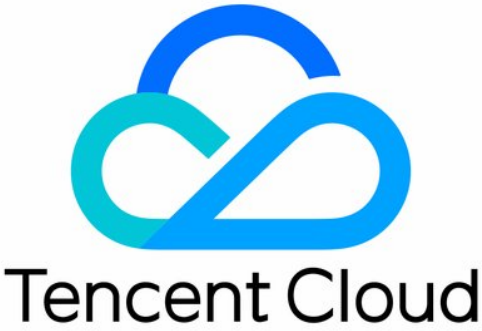
Tencent Kubernetes Engine

Practical Tutorial

Product Documentation



©2013-2022 Tencent Cloud. All rights reserved.

Copyright Notice

©2013-2024 Tencent Cloud. All rights reserved.

Copyright in this document is exclusively owned by Tencent Cloud. You must not reproduce, modify, copy or distribute in any way, in whole or in part, the contents of this document without Tencent Cloud's the prior written consent.

Trademark Notice

Tencent Cloud

All trademarks associated with Tencent Cloud and its services are owned by Tencent Cloud Computing (Beijing) Company Limited and its affiliated companies. Trademarks of third parties referred to in this document are owned by their respective proprietors.

Service Statement

This document is intended to provide users with general information about Tencent Cloud's products and services only and does not form part of Tencent Cloud's terms and conditions. Tencent Cloud's products or services are subject to change. Specific products and services and the standards applicable to them are exclusively provided for in Tencent Cloud's applicable terms and conditions.

Contents

Practical Tutorial
- Cluster
  - Cluster Model Recommendations
  - Enabling Disaster Recovery for Masters of Self-Deployed Clusters
  - Using Private DNS to Implement Automatic Domain Name Resolution When Accessing Cluster via Private Network
- Cluster Migration
  - Using COS as Velero Storage to Implement Backup and Restoration of Cluster Resources
  - Using Velero to Replicate Cluster Resources in TKE
  - Using Velero to Migrate Cluster Resources to TKE Across Cloud Platforms
  - Guide on Migrating Resources in a TKE Managed Cluster to an Serverless Cluster
- Serverless Cluster
  - Accessing Internet through NAT Gateway
  - Using EIP to Access Public Network
  - Mastering Deep Learning in Serverless Cluster
    - Building Deep Learning Container Image
    - Running Deep Learning in TKE Serverless
    - FAQs
      - Public Network Access
      - Log Collection
  - Customized DNS Service of Serverless Cluster
- Edge Cluster
  - TKE Edge ServiceGroup Feature
    - Using ServiceGroup via YAML File
  - TKE Edge Distributed Node Status Determination Mechanism
- Security
  - Pod Security Group
  - Container Image Signature and Verification
  - how to use CAM to authenticate databases for workloads running in TKE
- Service Deployment
  - Proper Use of Node Resources
    - Overview
    - Setting Request and Limit
    - Proper Resource Allocation
    - Auto Scaling
  - Application High Availability Deployment

©2013-2022 Tencent Cloud. All rights reserved.

- Smooth Workload Upgrade
- Parameter Adaptation for docker run
- Solve the inconsistent time zone problem in the container
- Container coredump Persistence
- Using a Dynamic Admission Controller in TKE

Hybrid Cloud
- Elastic Scaling with EKS for IDC-Based Cluster

Network
- DNS
  - Best Practices of TKE DNS
  - Using NodeLocal DNS Cache in a TKE Cluster
  - Implementing Custom Domain Name Resolution in TKE
  - Configuring ExternalDNS in TKE
- Using Network Policy for Network Access Control
- Deploying NGINX Ingress on TKE
- Nginx Ingress High-Concurrency Practices
- Nginx Ingress Best Practices
- Self-Built Nginx Ingress Practice Tutorial
  - Quick Start
  - Custom Load Balancer
  - Enabling CLB Direct Connection
  - Optimization for High Concurrency Scenarios
  - High Availability Configuration Optimization
  - Observability Integration
  - Access to Tencent Cloud WAF
  - Installing Multiple Nginx Ingress Controllers
  - Migrating from TKE Nginx Ingress Plugin to Self-Built Nginx Ingress
  - Complete Example of values.yaml Configuration
- Limiting the bandwidth on pods in TKE
- Directly connecting TKE to the CLB of pods based on the ENI
- Use CLB-Pod Direct Connection on TKE
- Obtaining the Real Client Source IP in TKE
- Using Traefik Ingress in TKE

Release
- Using CLB to Implement Simple Blue-Green Deployment and Grayscale Release
- Using Nginx Ingress to Implement Canary Release

Logs
- Best Practice in TKE Log Collection

©2013-2022 Tencent Cloud. All rights reserved.

- Implementing Multi-line Log Merging for TKE Serverless Log Collection
- Custom Nginx Ingress Log
- Monitoring
  - Using Prometheus to monitor Java applications
  - Using Prometheus to Monitor MySQL and MariaDB
  - Migrating Self-built Prometheus to Cloud Native Monitoring
- OPS
  - Removing and Re-adding Nodes from and to Cluster
  - Using Ansible to Batch Operate TKE Nodes
  - Using Cluster Audit for Troubleshooting
  - Renewing a TKE Ingress Certificate
  - Using cert-manager to Issue Free Certificates
  - Using cert-manager to Issue Free Certificate for DNSPod Domain Name
  - Using the TKE NPDPlus Plug-In to Enhance the Self-Healing Capability of Nodes
  - Using kubecm to Manage Multiple Clusters kubeconfig
  - Quick Troubleshooting Using TKE Audit and Event Services
  - Customizing RBAC Authorization in TKE
  - Clearing De-registered Tencent Cloud Account Resources
- Terraform
  - Managing TKE Clusters and Node Pools with Terraform
- DevOps
  - Construction and Deployment of Jenkins Public Network Framework Appications based on TKE
    - Example
    - Step 1: Configure the TKE cluster and Jenkins
    - Step 2:Slave pod build configuration
    - Build test
  - Using Docker as an image building service in a containerd cluster
  - Deploying Jenkins on TKE
- Auto Scaling
  - Cluster Auto Scaling Practices
  - Using tke-autoscaling-placeholder to Implement Auto Scaling in Seconds
  - Installing metrics-server on TKE
  - Using Custom Metrics for Auto Scaling in TKE
  - Utilizing HPA to Auto Scale Businesses on TKE
  - Using VPA to Realize Pod Scaling up and Scaling down in TKE
  - Adjusting HPA Scaling Sensitivity Based on Different Business Scenarios
  - Implementing elasticity based on traffic prediction with EHPA
  - Implementing Horizontal Scaling based on CLB monitoring metrics using KEDA in TKE

©2013-2022 Tencent Cloud. All rights reserved.

Storage
Backing up and Restoring PVC via CBS-CSI Add-on
Static Mounting of CFS-Turbo File System
Mounting of CFS-Turbo for TKE Clusters
Static Mounting of CFS-Turbo for TKE Serverless Clusters
Containerization
Accelerated Pull of Images Outside the Chinese Mainland
Image Layering Best Practices
Microservice
Hosting Dubbo to TKE
Hosting SpringCloud to TKE
Cost Management
Tools for Resource Utilization Improvement

©2013-2022 Tencent Cloud. All rights reserved.

Practical Tutorial
Cluster
Cluster Model Recommendations

Last updated：2022-04-21 17:22:14

When you create a Kubernetes cluster by using TKE, you must select models from various configuration options. This document describes and compares available feature models and gives suggestions to help you select models that are most applicable to your services.

- Kubernetes Versions
- Container Network Plugins: GlobalRouter and VPC-CNI
- Runtime Components: Docker and Containerd (Under Beta Testing)
- Service Forwarding Modes: iptables and IPVS
- Cluster Types: Managed Cluster and Self-Deployed Cluster
- Node Operating Systems
- Node Pool
- Launch Script

Kubernetes Versions

Kubernetes versions are iterated quickly. New versions usually include many bug fixes and new features. Meanwhile, earlier versions will be phased out. We recommend that you select the latest version that is supported by the current TKE when creating a cluster. Subsequently, you can upgrade existing master components and nodes to the latest versions generated during iteration.

Container Network Plugins: GlobalRouter and VPC-CNI

Network modes

TKE supports the following two network modes. For more information, see How to Choose TKE Network Mode.

- **GlobalRouter mode**:
  - In this mode, container network capabilities are implemented based on container networking interfaces (CNIs) and network bridges, whereas container routing is implemented based on the underlying VPC layer.

©2013-2022 Tencent Cloud. All rights reserved.

- Containers are located on the same network plane as nodes. IP ranges of containers cover abundant IP addresses and do not overlap those of VPC instances.

- **VPC-CNI mode**:
  - In this mode, container network capabilities are implemented based on CNIs and VPC ENIs, whereas container routing is implemented based on ENIs. The performance of this mode is approximately 10% higher than that of the GlobalRouter mode.
  - Containers are located on the same network plane as nodes. The IP ranges of containers fall within those of VPC instances.
  - Pods can use static IP addresses.

How to use

TKE allows you to specify network modes in the following ways:

- Specify the GlobalRouter mode when creating a cluster.
- Specify the VPC-CNI mode when creating a cluster. Subsequently, all pods must be created in VPC-CNI mode.
- Specify the GlobalRouter mode when creating a cluster. You can enable the VPC-CNI mode for the cluster when needed. In this case, the two modes are mixed.

Recommendations

- In general cases, we recommend that you select the GlobalRouter mode, because IP ranges of containers cover abundant IP addresses, allow high scalability, and support large-scale services.
- If a subsequent service needs to run in VPC-CNI mode, you can enable the VPC-CNI mode for the GlobalRouter cluster. In this case, the GlobalRouter mode is mixed with the VPC-CNI mode, but only some services run in VPC-CNI mode.
- If you fully understand and accept the use limits of the VPC-CNI mode and all pods in the cluster need to run in VPC-CNI mode, we recommend that you select the VPC-CNI mode when creating the cluster.

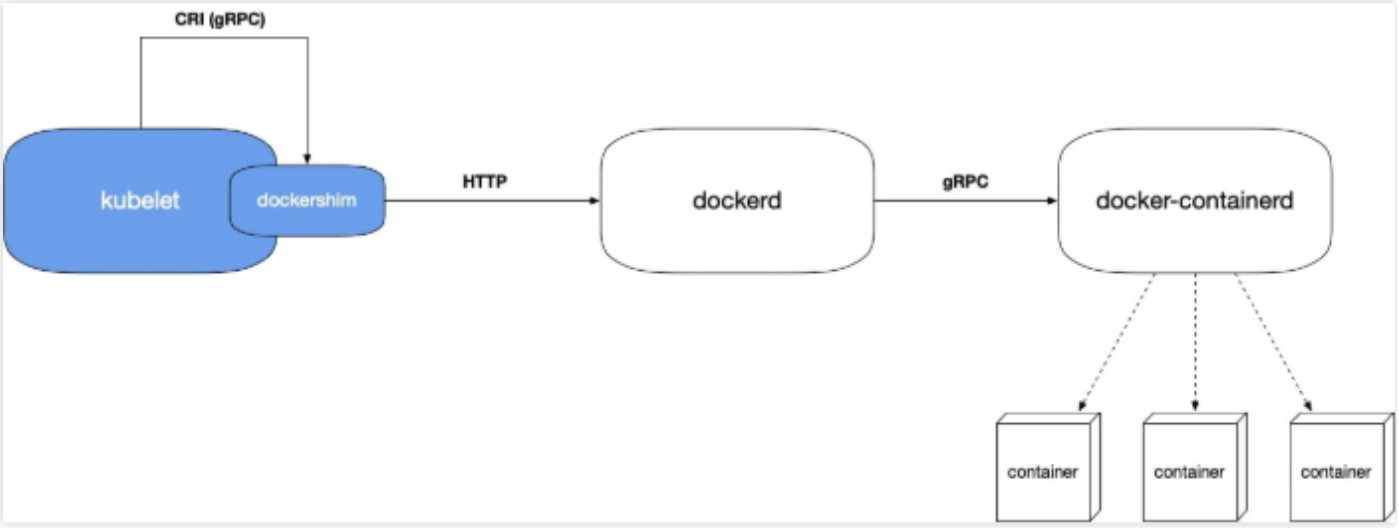
Runtime Components: Docker and Containerd (Under Beta Testing)

Runtime components

TKE supports two types of runtime components: Docker and containerd. For more information, see How to Choose Between containerd and Docker.

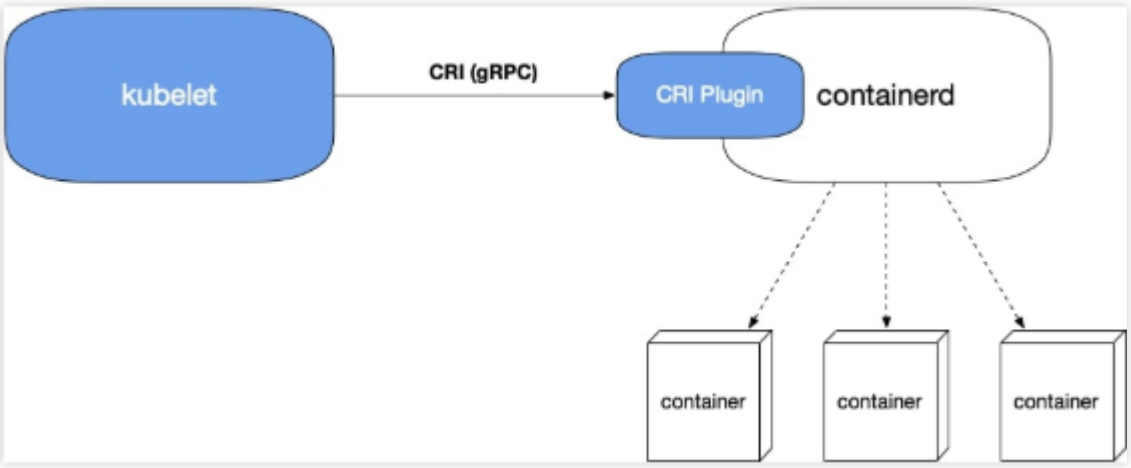
©2013-2022 Tencent Cloud. All rights reserved.

- **Using Docker as a container runtime**:



  - The call chain is as follows:
    i. The dockershim module in kubelet adapts the container runtime interface (CRI) for the Docker runtime.
    ii. The kubelet component calls dockershim by using a socket file.
    iii. The dockershim module calls the API of the dockerd component, that is, the Docker HTTP API.
    iv. The dockerd component calls the docker-containerd gRPC API to create or terminate the container.
  - Reasons for a long call chain:
    Kubernetes initially supported Docker only. Later, the CRI was introduced and runtime was abstracted so that multiple types of runtimes were supported. Docker and Kubernetes are competitors, and therefore Docker did not implement the CRI in dockerd, and Kubernetes had to implement the CRI in dockerd itself. Internal components of Docker were modularized to adapt to the CRI.

- **Using containerd (under beta testing) as a container runtime**:



  - Containerd has supported the CRI plugin since containerd 1.1. That is, containerd can adapt to the CRI.

©2013-2022 Tencent Cloud. All rights reserved.

- The call chain of the containerd runtime does not include dockershim and dockerd, which exist in the call chain of the Docker runtime.

Comparison between the two runtimes

- The call chain of the containerd runtime bypasses dockerd and therefore is shorter. Accordingly, the containerd solution requires fewer components, occupies fewer node resources, and bypasses dockerd bugs. However, containerd has some bugs that need to be fixed. Currently, containerd is under beta testing and has fixed some bugs.
- Having been used for a long time, the Docker solution is more mature, supports the Docker API, and provides abundant features. This solution is friendly to most users.

Recommendations

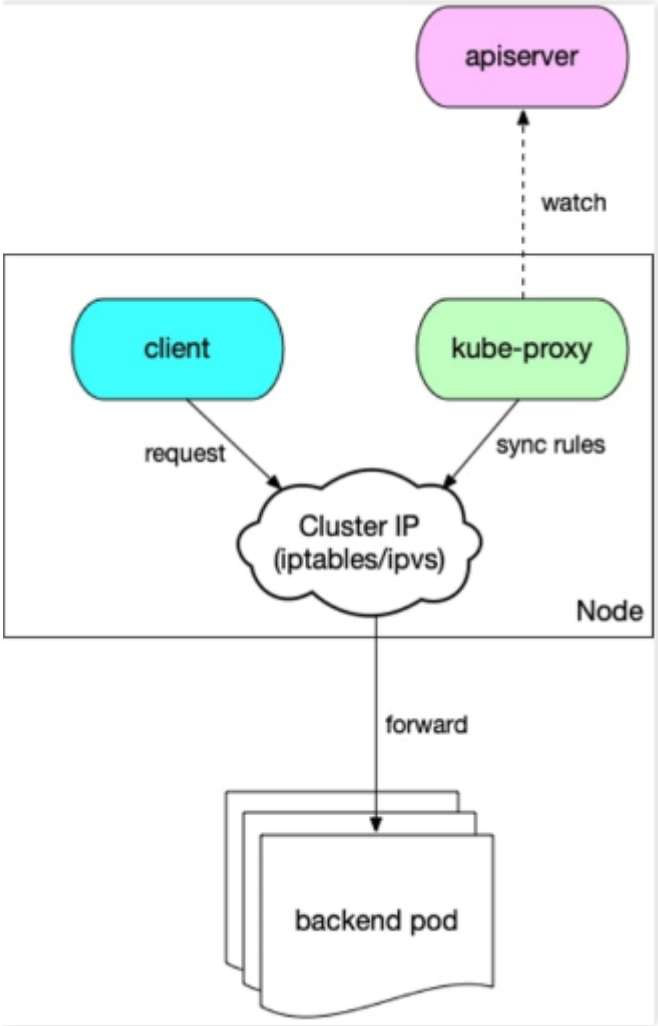
- The Docker solution is more mature than the containerd solution. If you require high stability, we recommend that you select the Docker solution.
- In the following scenarios, you can select the Docker solution only:
- You need to run a Docker host inside of another Docker host (Docker-in-Docker). This requirement usually occurs during continuous integration (CI).
- You need to use Docker commands on a node.
- You need to call the Docker API.

In other scenarios, we recommend that you select the containerd solution.

Service Forwarding Modes: iptables and IPVS

©2013-2022 Tencent Cloud. All rights reserved.

The following figure shows how a Service is forwarded.



1. The kube-proxy component on the node watches API Server to obtain the Service and the Endpoint. Then, the kube-proxy component converts the Service to an iptables or IPVS rule based on the forwarding mode and writes the rule to the node.
2. The client in the cluster gains access to the Service through the cluster IP address. Then, according to the iptable or IPVS rule, the client is load-balanced to the backend pod corresponding to the Service.

Comparison between the two forwarding modes

- The IPVS mode provides higher performance but has some outstanding bugs.
- The iptables mode is more mature and stable.

Recommendations

If you require extremely high stability with less than 2,000 Services running in the cluster, we recommend that you select iptables. In other scenarios, we recommend that you preferably select IPVS.

©2013-2022 Tencent Cloud. All rights reserved.

Cluster Types: Managed Cluster and Self-Deployed Cluster

TKE supports the following types of clusters:

- **Managed clusters**:
- Master components are invisible to you but are managed by Tencent Cloud.
- Clusters with most new features are preferably managed.
- Computing resources of master components are automatically scaled up based on the cluster scale.
- You do not need to pay for master components.
- **Self-deployed clusters**:
  - Master components are fully under your control.
  - You need to purchase master components.

    Recommendations

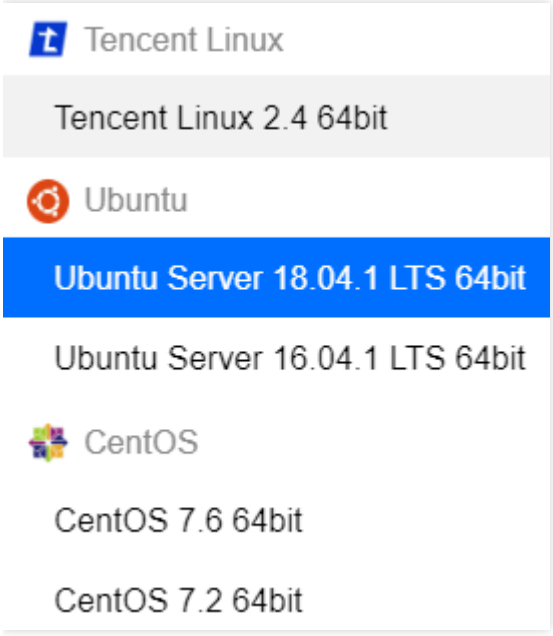
    In regular cases, we recommend that you select managed clusters. If you need to fully control master components, such as specifying custom features to implement advanced features, you can select self-deployed clusters.

Node Operating System

TKE supports three distributions of operating systems, Tencent Linux, Ubuntu and CentOS. The Tencent Linux operating system uses TencentOS-kernel that is a customized kernel maintained by Tencent Cloud team. Other operating systems use the open-source kernel provided by the official Linux community. The following shows the

©2013-2022 Tencent Cloud. All rights reserved.

supported operating systems.



Note：
TKE-Optimized series images is once used for improving the stability of the image and providing more features, but it is not available for the clusters in TKE console after the Tencent Linux public image is launched. For more information, see TKE-Optimized Series Images.

Recommendations

We recommend that you use the Tencent Linux operating system. It is a public image of Tencent Cloud that contains the TencentOS-kernel. TKE now supports this image and uses it as the default image.

Node Pool

The node pool is mainly used to batch manage nodes with the following items:

- Label and Taint properties of nodes
- Startup parameters of node components
- Custom launch script for nodes

  For more information, see Node Pool Overview.

Application scenarios

©2013-2022 Tencent Cloud. All rights reserved.

- Manage heterogeneous nodes by group to reduce management costs.
- Use the Label and Taint properties to enable a cluster to support complex scheduling rules.
- Frequently scale out and in nodes to reduce operation costs.
- Routinely maintain nodes, such as upgrade node versions.

Examples

Some I/O-intensive services require models with high I/O throughput. You can create a node pool for a service of these kinds, configure a model, centrally specify Label and Taint properties for the nodes, and configure affinity with I/O-intensive services. You can select Labels to schedule the service to a node with a high I/O model. To avoid other service pods from being scheduled to the node, you can select specific Taints.

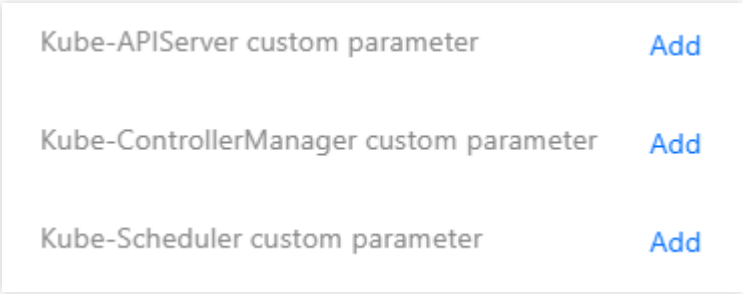
When the service traffic increases, the I/O-intensive service needs more computing resources. During peak hours, the HPA feature automatically scales out pods for the service, and the computing resources of the node become insufficient. In this case, the auto scaling feature of the node pool automatically scales out nodes to withstand the traffic spike.

Launch Script

Custom parameters for components

> Note：
> To use this feature, submit a ticket to apply for it.

- When creating a cluster, you can customize some startup parameters of master components in **Advanced Settings** under **Cluster Information**.

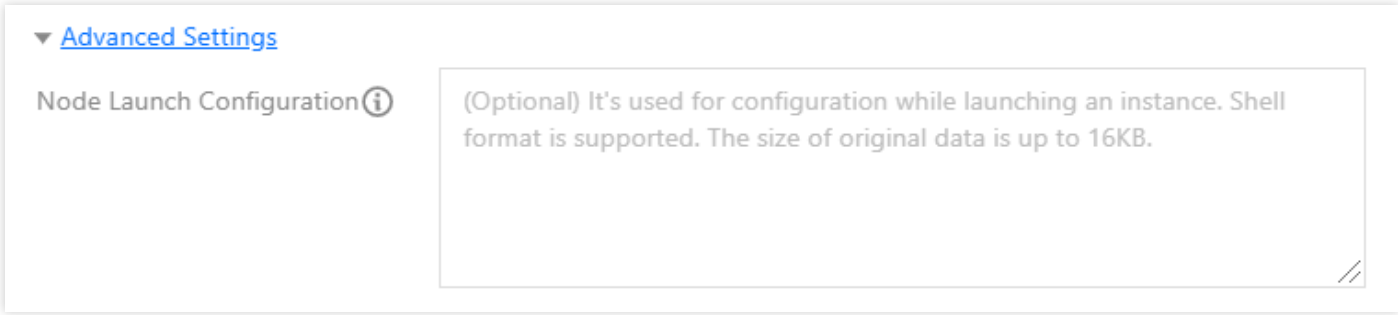


- In **Select Model** step, you can customize some startup parameters of kubelet in **Advanced Settings** under **Worker Configurations**.

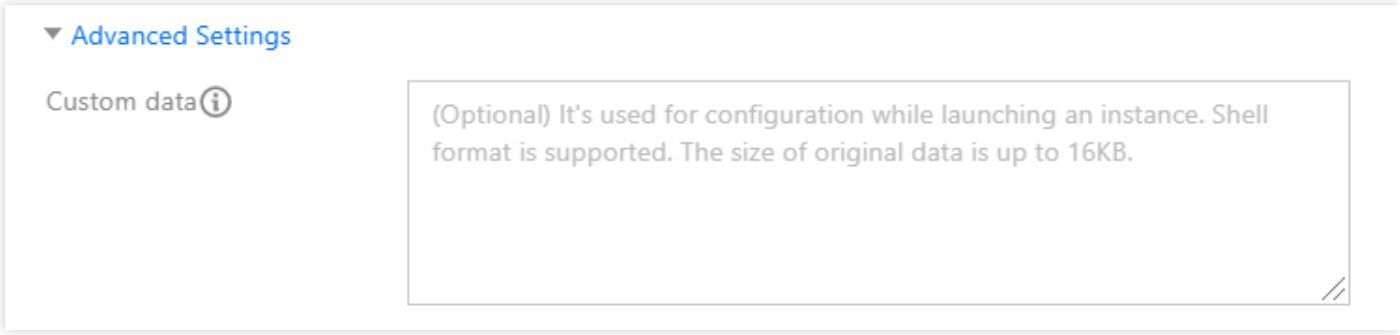
©2013-2022 Tencent Cloud. All rights reserved.

Node launch configuration

- When creating a cluster, in **Advanced Settings** under **CVM Configuration**, you can specify custom data to configure the node launch script. In the script, you can modify component startup parameters and kernel parameters, as shown in the following figure:



- When adding a node, in **Advanced Settings** under **CVM Configuration**, you can specify custom data to configure the node launch script. In the script, you can modify component startup parameters and kernel parameters, as shown in the following figure:



©2013-2022 Tencent Cloud. All rights reserved.

Enabling Disaster Recovery for Masters of Self-Deployed Clusters

Last updated：2020-11-11 15:45:38

Overview

TKE includes managed clusters and self-deployed clusters. If you use a managed cluster, you do not need to be concerned about disaster recovery. The masters of managed clusters are internally maintained by TKE. If you use a self-deployed cluster, you need to manage and maintain the master nodes yourself.
To enable disaster recovery for a self-deployed cluster, you need to first plan a disaster recovery scheme based on your needs and then complete the corresponding configuration during cluster creation. This document introduces how to enable disaster recovery for the masters of a TKE self-deployed cluster for your reference.

How to Enable Disaster Recovery

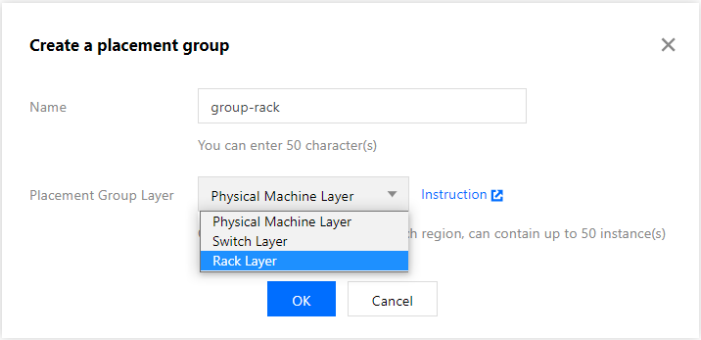
To enable disaster recovery, you need to start from physical deployment. To prevent a fault in the physical layer from causing exceptions on multiple masters, you need to widely distribute master nodes. You can use a placement group to choose the CPM, exchange, or rack dimension to distribute master nodes, thus preventing underlying hardware or software faults from causing exceptions on multiple masters. If you have high requirements for disaster recovery, you can consider deploying masters across availability zones, so as to prevent situations where a large-scale fault causes the entire IDC to become unavailable, leading to multiple master exceptions.

Using a Placement Group to Distribute Masters

1. Log in to the Placement Group Console to create a placement group. For more information, see Spread Placement Group. See the figure below:

> **⚠ Note：**
>
> The placement group and the TKE self-deployed cluster need to be in the same region.

©2013-2022 Tencent Cloud. All rights reserved.

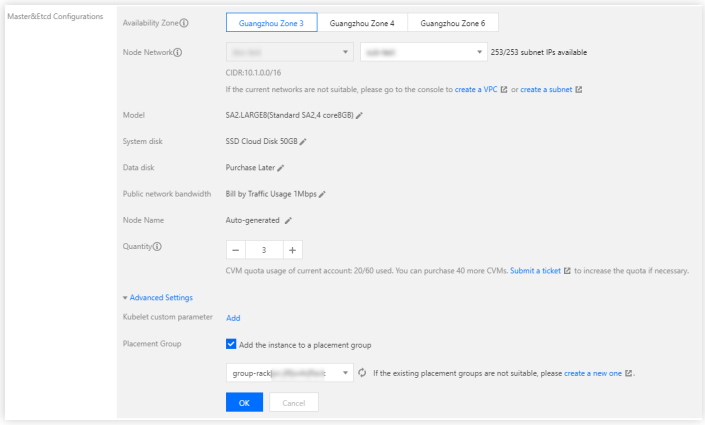


The placement group layers are as follows. In this document, the "rack layer" is selected as an example:

Placement Group Layer	Description
CPM layer	A master node of a self-deployed cluster is deployed on a CVM, which is a virtual machine running on a CPM. Multiple virtual machines may run on one CPM. If the CPM is faulty, all virtual machines running on it will be affected. By using this layer, you can distribute master nodes to different CPMs to prevent one faulty CPM from causing exceptions on multiple nodes.
Exchange layer	Multiple different CPMs may be connected to the same exchange. If the exchange is faulty, multiple CPMs will be affected. By using this layer, you can distribute master nodes to CPMs connected to different exchanges, thereby preventing one faulty exchange from causing exceptions on multiple master nodes.
Rack layer	Multiple different CPMs may be placed on the same rack. If a rack-level fault occurs, multiple CPMs on the rack will become faulty. By using this layer, you can distribute master nodes to CPMs on different racks, thereby preventing rack-level faults from causing exceptions on multiple master nodes.

2. Refer to Creating a Cluster to create a TKE self-deployed cluster. Choose **Master&Etcd Configuration** > **Advanced Configuration**, check **Add Instance to Spread Placement Group**, and select the created placement group. See the figure below:

©2013-2022 Tencent Cloud. All rights reserved.



After configuration is completed, the corresponding master nodes will be distributed to different racks to enable rack-level disaster recovery.

Disaster Recovery with Masters Deployed Across Availability Zones

If you have high requirements for disaster recovery and want to prevent situations where a large-scale fault causes the entire IDC to become unavailable, causing exceptions on all master nodes, you can choose to deploy masters in different availability zones. The configuration method is as follows:
During cluster creation, in **Master&Etcd Configuration**, add models to multiple availability zones. See the figure below:

©2013-2022 Tencent Cloud. All rights reserved.

Using Private DNS to Implement Automatic Domain Name Resolution When Accessing Cluster via Private Network

Last updated：2022-06-10 16:48:45

Overview

After private network access is enabled for the current cluster, TKE will access the cluster through the domain name by default. You need to configure `Host` on the access server to perform DNS queries on the private network. If no DNS rules (`Host`) are configured, an error "no such host" will be reported when you access the cluster on the access server (by running `kubectl get nodes`) as shown below:

In practice, configuring `Host` will increase your management labor costs. Therefore, we recommend you use Tencent Cloud's newly launched Private DNS service, which helps you get things done in just three steps.

Billing description

Private DNS is billed on a pay-as-you-go basis, where the number of private domains and that of DNS requests are billed on a natural day basis. For more information, see Billing Overview.

Available regions

Currently, Private DNS is not available in all the available regions of TKE. For the list of its available regions, see Use Limits.

To access clusters over the private network in regions not covered by Private DNS, you need to manually configure the `Host` . To use Private DNS in those regions, submit a ticket for application.

Prerequisites

A container cluster has been created and private network access has been enabled. For details, see Creating a Cluster.

©2013-2022 Tencent Cloud. All rights reserved.

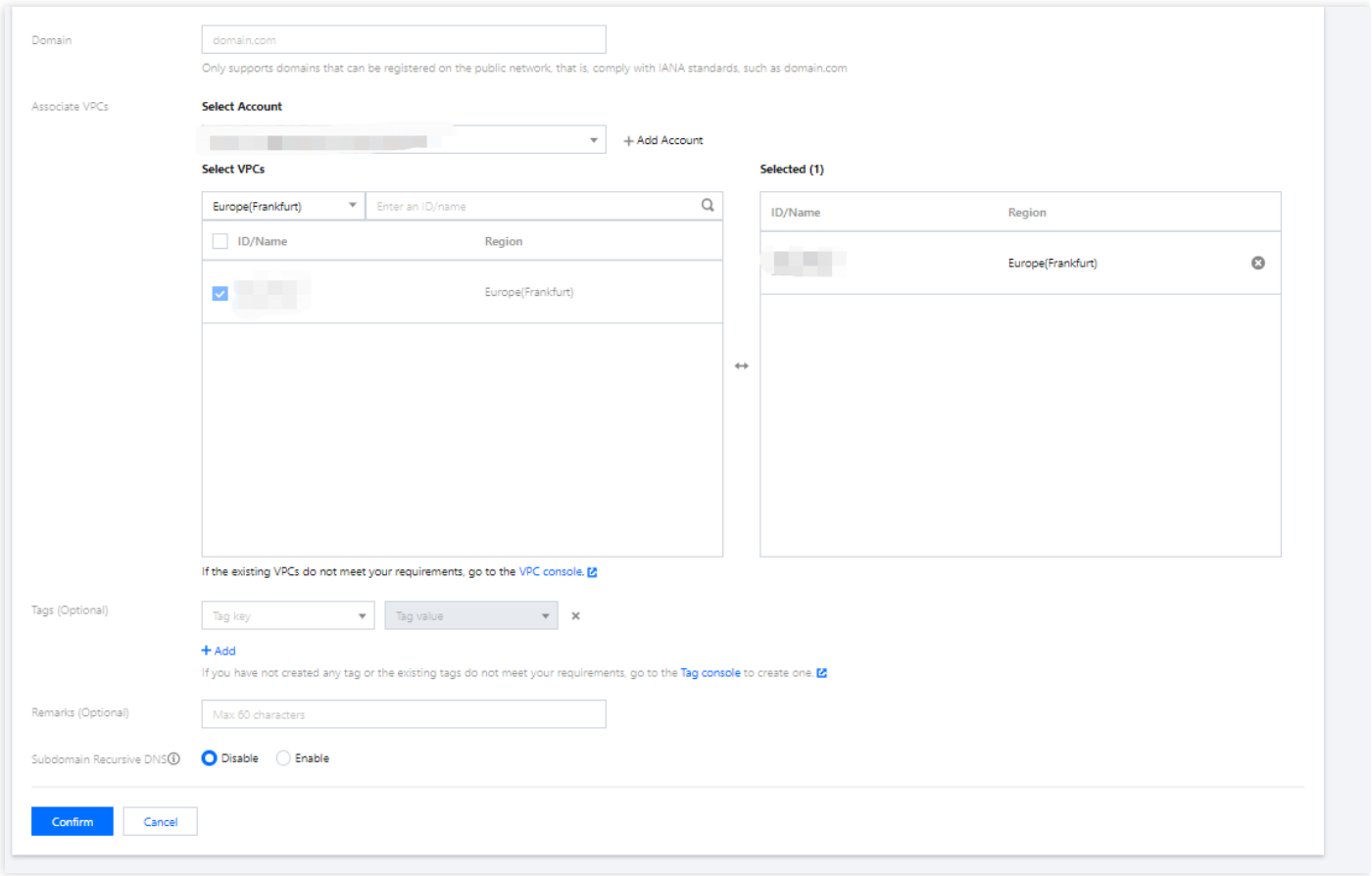
Directions

Activating Private DNS

See Activating Private DNS.

Creating private domain

1. Log in to the Private DNS console.
2. Click **Create Private Domain** and configure the following options (just use the default values for other parameters). For more information, see Creating Private Domain.



- **Domain**: Enter `tencent-cloud.com` (domain name allocated by TKE for accessing the cluster).
- **Associated VPC**: Select the node VPC that needs to access the cluster.

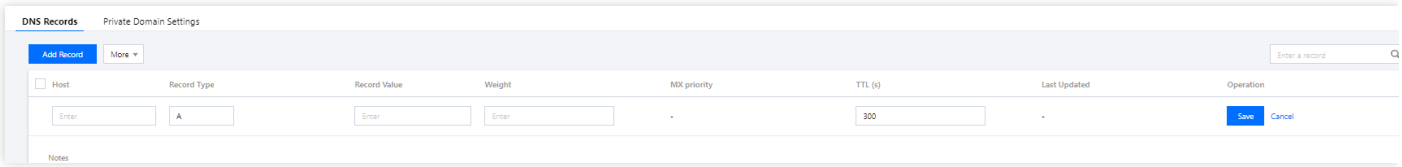
3. Click **OK**.

Configuring DNS records

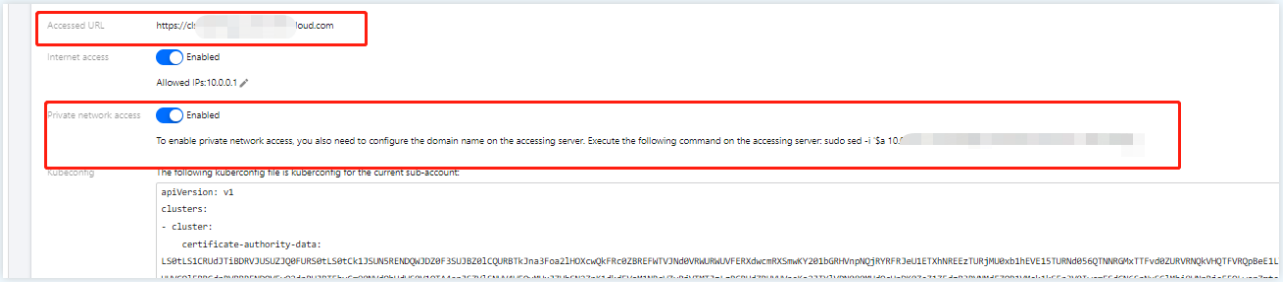
1. Click the private domain name created above to enter the **DNS Records** page.

©2013-2022 Tencent Cloud. All rights reserved.

2. Click **Add Records** and configure the following options:



- **Host Record**: Enter the secondary domain name for accessing the TKE cluster, for example, `cls-{{clsid}}.css` .
- **Record Type**: Enter `A` .
- **Record Value**: Enter the private IP for accessing the TKE cluster.

> Note：
> **You can get the **Host Record** and **Record Value** from **Cluster Management** > **Cluster** > **Basic Info**. Here, **Host Record** is the domain name in **Access Address**, and **Record Value** is the IP address in **Private Network Access**, as shown below:
>
> 
>
>
> c. Click **Save** in the **Operation** column on the right.

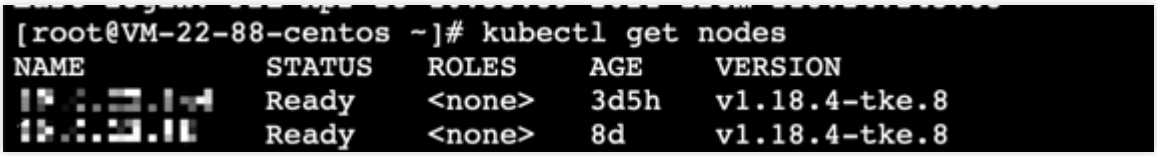
Verifying effect

1. Run the following command to access the cluster again.

```
kubectl get nodes
```

2. When the following result is displayed, the cluster has been successfully accessed, and the node list has been pulled.

©2013-2022 Tencent Cloud. All rights reserved.



©2013-2022 Tencent Cloud. All rights reserved.

Cluster Migration
Using COS as Velero Storage to Implement Backup and Restoration of Cluster Resources

Last updated：2023-05-23 16:25:26

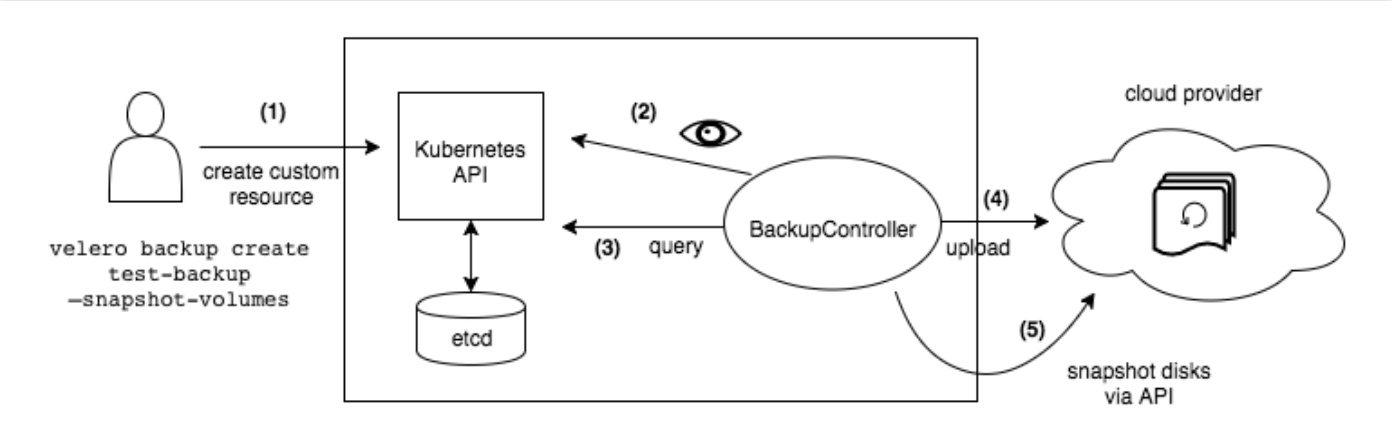
Overview

Velero (the previous version is called Heptio Ark), an open-source tool, can safely back up and restore, perform disaster recovery, and migrate Kubernetes cluster resources and persistent volumes. Deploying Velero in the TKE cluster or self-built Kubenetes cluster can achieve the following features:

- Back up cluster resources and restore in case of loss.
- Migrate cluster resources to other clusters.
- Replicate the production cluster resources to the development and test clusters.

Working principles of Velero is shown in the figure below (from Velero official website). When the user runs the backup command, the backup process is described as follows:

1. Call the custom resource API to create a backup object, as shown in (1).
2. When BackupController detects the generated backup object, as shown in (2), it executes the backup operation, as shown in (3).
3. Upload and store the backup cluster resources and storage volume snapshots to Velero's backend, as shown in (4) and (5).



©2013-2022 Tencent Cloud. All rights reserved.

In addition, when performing a restoration operation, Velero will synchronize the data of the specified backup object from the backend storage to the Kubernetes cluster.
For more information about Velero, see Velero official document. This document describes how to use COS as the Velero backend storage to implement cluster backup and restoration.

Prerequisites

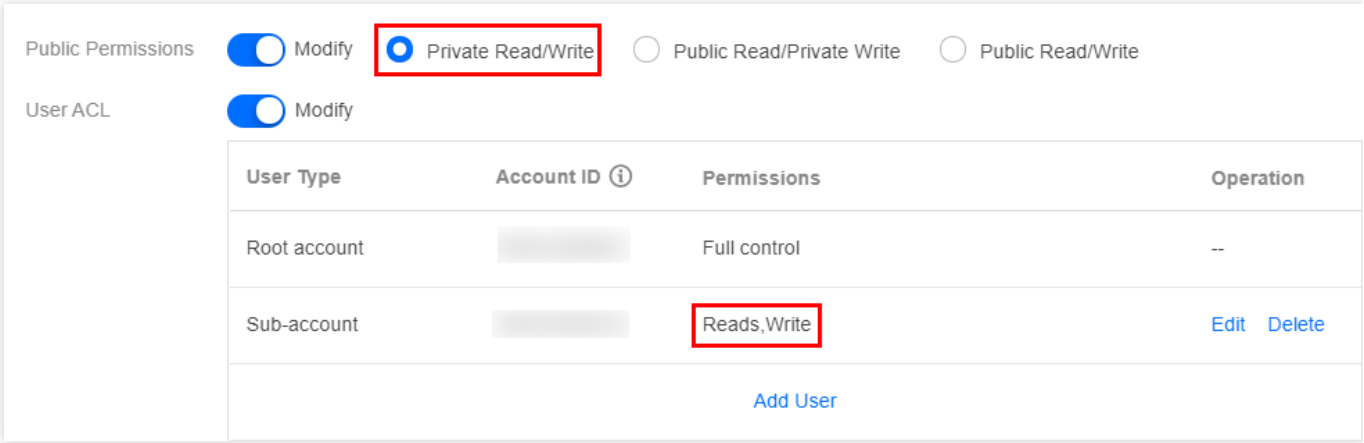
- You have registered a Tencent Cloud account.
- Activate the COS service.
- Create a Kubernetes cluster of v1.10 or later version, and the cluster can use DNS and Internet services normally. For more information, see Creating a Cluster.

Directions

Configuring COS

Creating a bucket

1. Log in to the COS console to create a bucket for Velero to store backups. For more information, see Creating Buckets.
2. Set Access Permission for the bucket. COS supports two permission types:
   - **Public permissions**: for the sake of security, the permission of private read/write is recommended for the bucket. For more information, see **Types of Permission** under Bucket Overview.
   - **User permissions**: the root account has all bucket permissions (full control) by default. You can add sub-accounts and grant them permissions including read/write, read/write ACL, and even **full control**.
     The sample sub-account has been granted the permissions of read/write for performing read/write on the bucket, as shown in the figure below:



©2013-2022 Tencent Cloud. All rights reserved.

Obtaining the bucket access credentials

Velero uses an API compatible with AWS S3 to access COS. It needs to use a pair of access key ID and a signature created by the key for authentication. In the S3 API parameters:

- `access_key_id` : access key ID
- `secret_access_key` : key

1. Log in to CAM console to create and obtain the keys `SecretId` and `SecretKey` of the COS authorized sub-account. Among them:

- The value of `SecretId` corresponds to the `access_key_id` field.
- The value of `SecretKey` corresponds to the `secret_access_key` field.

2. According to the above corresponding relationship, create the credential configuration file `credentials-velero` required by Velero in the local directory. The content is as follows:

```
[default]
aws_access_key_id=<SecretId>
aws_secret_access_key=<SecretKey>
```

Installing Velero

1. Download the latest version of Velero to the cluster environment. This document uses Velero v1.5.2 as an example, as shown below:

```
wget https://github.com/vmware-tanzu/velero/releases/download/v1.5.2/velero-v1.5.2-linux-amd64.tar.gz
```

2. Run the following command to decompress the installation package. The installation package provides Velero command lines and some sample files, as shown below:

```
tar -xvf velero-v1.5.2-linux-amd64.tar.gz
```

3. Run the following command to migrate the Velero executable file from the decompressed directory to the system environment variable directory for direct use. This document takes the migration to the `/usr/bin` directory as an example, as shown below:

```
mv velero-v1.5.2-linux-amd64/velero /usr/bin/
```

©2013-2022 Tencent Cloud. All rights reserved.

4. Run the following commands to install Velero, create Velero and Restic workloads and other necessary resource objects (for installation parameter descriptions, please see table below), as shown below:

```
velero install --provider aws --plugins velero/velero-plugin-for-aws:v1.1.0 --b
ucket <BucketName> \
```

```
--secret-file ./credentials-velero
--use-restic
--default-volumes-to-restic
--backup-location-config
region=ap-guangzhou,s3ForcePathStyle="true",s3Url=https://cos.ap-
guangzhou.myqcloud.com
```

Installation parameter description:

Parameter	Description
--provider	Declare to use the plugin type provided by `aws` .
--plugins	Use AWS S3 compatible API plugin "velero-plugin-for-aws".
--bucket	The name of the bucket created in COS
--secret-file	The access credential file for accessing COS. For more information, see "credentials-velero" credential file created above.
--use-restic	Velero supports using the free and open-source backup tool Restic to back up and restore Kubernetes storage volume data (`hostPath` volume is not supported, For more information, see Restic Limitations). It is recommended to enable this integration, which is a supplement to Velero's backup feature.
--default-volumes-to-restic	Enable Restic to back up all Pod volumes, provided that the `--use-restic` parameter is enabled.
--backup-location-config	Back up bucket access related configuration, including region, s3ForcePathStyle, s3Url, etc.
region	COS bucket region is compatible with S3 API, for example, the creation region is Guangzhou, and the value of parameter “region” is "ap-guangzhou".

©2013-2022 Tencent Cloud. All rights reserved.

Parameter	Description
s3ForcePathStyle	Use S3 file path format.
s3Url	S3 API access address compatible with COS. Please note that the domain name in the access address is not the public domain name used to create the COS bucket. It must be a URL in the format https://cos.<region>.myqcloud.com. For example, if the region is Guangzhou, then The parameter value is `https://cos.ap-guangzhou.myqcloud.com` .

You can use the command `velero install --help` to view other installation parameters. For example, if you do not need to back up storage volume data, you can set `--use-volume-snapshots=false` to disable storage volume snapshot backup.
Check the installation process after executing the installation command, as shown in the figure below:

```
root@VM-0-8-ubuntu:~# velero install  --provider aws --plugins velero/velero-plugin-for-aws:v1.1.0 --bucket jokeyli-        --secret-file ./credentials-velero --use-restic --default-volumes-to-restic --backup-location
-config region=ap-guangzhou,s3ForcePathStyle="true",s3Url=https://cos.ap-guangzhou.myqcloud.com
CustomResourceDefinition/backups.velero.io: attempting to create resource
CustomResourceDefinition/backups.velero.io: created
CustomResourceDefinition/backupstoragelocations.velero.io: attempting to create resource
CustomResourceDefinition/backupstoragelocations.velero.io: created
CustomResourceDefinition/deletebackuprequests.velero.io: attempting to create resource
CustomResourceDefinition/deletebackuprequests.velero.io: created
CustomResourceDefinition/downloadrequests.velero.io: attempting to create resource
CustomResourceDefinition/downloadrequests.velero.io: created
CustomResourceDefinition/podvolumebackups.velero.io: attempting to create resource
CustomResourceDefinition/podvolumebackups.velero.io: created
CustomResourceDefinition/podvolumerestores.velero.io: attempting to create resource
CustomResourceDefinition/podvolumerestores.velero.io: created
CustomResourceDefinition/resticrepositories.velero.io: attempting to create resource
CustomResourceDefinition/resticrepositories.velero.io: created
CustomResourceDefinition/restores.velero.io: attempting to create resource
CustomResourceDefinition/restores.velero.io: created
CustomResourceDefinition/schedules.velero.io: attempting to create resource
CustomResourceDefinition/schedules.velero.io: created
CustomResourceDefinition/serverstatusrequests.velero.io: attempting to create resource
CustomResourceDefinition/serverstatusrequests.velero.io: created
CustomResourceDefinition/volumesnapshotlocations.velero.io: attempting to create resource
CustomResourceDefinition/volumesnapshotlocations.velero.io: created
Waiting for resources to be ready in cluster...
Namespace/velero: attempting to create resource
Namespace/velero: created
ClusterRoleBinding/velero: attempting to create resource
ClusterRoleBinding/velero: created
ServiceAccount/velero: attempting to create resource
ServiceAccount/velero: created
Secret/cloud-credentials: attempting to create resource
Secret/cloud-credentials: created
BackupStorageLocation/default: attempting to create resource
BackupStorageLocation/default: created
VolumeSnapshotLocation/default: attempting to create resource
VolumeSnapshotLocation/default: created
Deployment/velero: attempting to create resource
Deployment/velero: created
DaemonSet/restic: attempting to create resource
DaemonSet/restic: created
Velero is installed! ⚠ Use 'kubectl logs deployment/velero -n velero' to view the status.
```

5. After the installation, wait for Velero and Restic workloads to be ready. Run the following command to check whether the configured storage location is available. If "Available" is displayed, it means that the cluster can access the COS normally, as shown in the figure below:

```
root@VM-0-8-ubuntu:~# velero backup-location get
NAME      PROVIDER   BUCKET/PREFIX        PHASE       LAST VALIDATED                  ACCESS MODE
default   aws        j[redacted]          Available   2020-11-13 22:13:59 +0800 CST   ReadWrite
```

   At this point, the Velero installation is complete. For more information about Velero installation, see Velero documentation on the official website.

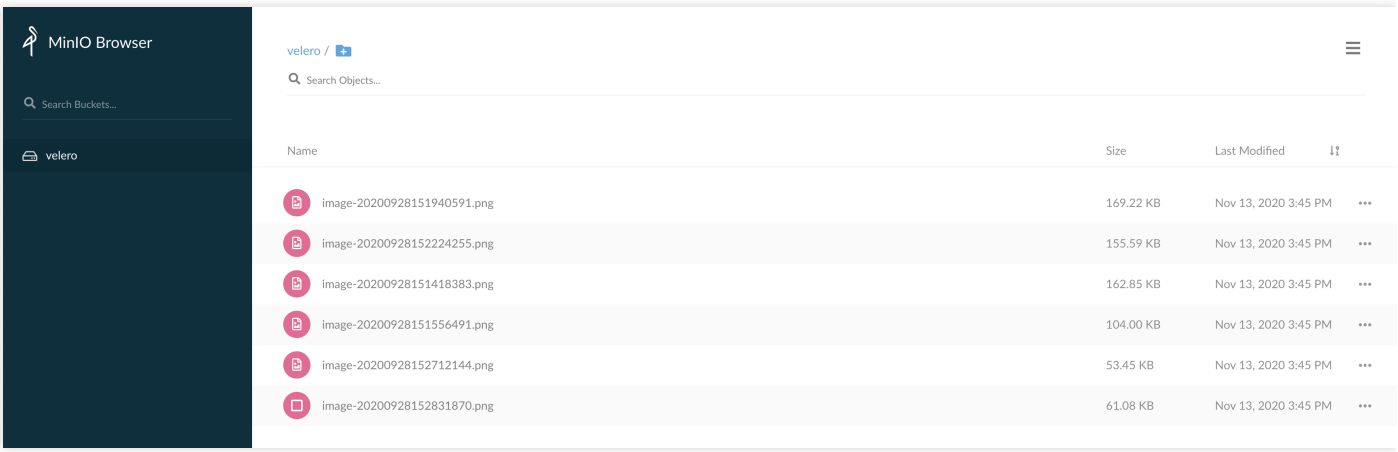
Velero backup and restoration testing

©2013-2022 Tencent Cloud. All rights reserved.

1. Use the Helm tool in the cluster to create a MinIO test service with persistent volumes. For the MinIO installation method, see MinIO Installation. In this example, the MinIO service has bound a load balancer. You can use the public network address to access the management page in the browser.

```
[root@VM-0-28-tlinux ~]# kubectl get pod | grep minio
minio-1605249781-66c4cbdfc-d7r4b   1/1     Running   0          30h
[root@VM-0-28-tlinux ~]# kubectl get svc | grep minio
minio-1605249781   LoadBalancer                                  9000:30252/TCP   31h
[root@VM-0-28-tlinux ~]#
```

2. Log in to the MinIO Web management page and upload the image for testing, as shown in the figure below:



3. Using Velero backup, you can directly back up all objects in the cluster, or filter objects by type, namespace, / or tag. You can run the following command to only backup all resources in the default namespace, as shown below:

```
velero backup create default-backup --include-namespaces default
```

4. Run the following command to check whether the backup task is completed. When the status of the backup task is "Completed" and "ERRORS" is 0, it means that the backup task is completed without any errors.

```
velero backup get
```

©2013-2022 Tencent Cloud. All rights reserved.

The backup process is shown in the figure below:

```
[root@VM-0-28-tlinux ~]# velero backup create default-backup --include-namespaces default
Backup request "default-backup" submitted successfully.
Run `velero backup describe default-backup` or `velero backup logs default-backup` for more details.
[root@VM-0-28-tlinux ~]# velero backup get
NAME             STATUS       ERRORS   WARNINGS   CREATED                         EXPIRES   STORAGE LOCATION   SELECTOR
default-backup   InProgress   0        0          2020-11-14 22:34:18 +0800 CST   29d       default            <none>
[root@VM-0-28-tlinux ~]# velero backup get
NAME             STATUS       ERRORS   WARNINGS   CREATED                         EXPIRES   STORAGE LOCATION   SELECTOR
default-backup   InProgress   0        0          2020-11-14 22:34:18 +0800 CST   29d       default            <none>
[root@VM-0-28-tlinux ~]# velero backup get
NAME             STATUS       ERRORS   WARNINGS   CREATED                         EXPIRES   STORAGE LOCATION   SELECTOR
default-backup   Completed    0        0          2020-11-14 22:34:18 +0800 CST   29d       default            <none>
```

5. Run the following command to delete all resources under MinIO, including PVC persistent volumes, as shown below:

```
[root@VM-0-28-tlinux ~]# helm list
WARNING: Kubernetes configuration file is group-readable. This is insecure. Location: /root/.kube/config
WARNING: Kubernetes configuration file is world-readable. This is insecure. Location: /root/.kube/config
NAME                NAMESPACE       REVISION        UPDATED                                 STATUS          CHART           APP VERSION
minio-1605249781    default         1               2020-11-13 14:43:02.437981822 +0800 CST deployed        minio-8.0.3     master
[root@VM-0-28-tlinux ~]#  helm uninstall minio-1605249781
WARNING: Kubernetes configuration file is group-readable. This is insecure. Location: /root/.kube/config
WARNING: Kubernetes configuration file is world-readable. This is insecure. Location: /root/.kube/config
release "minio-1605249781" uninstalled
[root@VM-0-28-tlinux ~]# kubectl get pvc | grep minio
No resources found in default namespace.
[root@VM-0-28-tlinux ~]# kubectl get pod | grep minio
[root@VM-0-28-tlinux ~]#
```

6. After deleting the MinIO resource, you can use the previous backup to test whether the deleted MinIO resource can be successfully restored. Run the following command to temporarily update the backup storage location to read-only mode (to prevent Velero from creating or deleting backup objects in the backup storage location during the restoration process).

```
kubectl patch backupstoragelocation default --namespace velero \
--type merge \
--patch '{"spec":{"accessMode":"ReadOnly"}}'
```

The execution process is shown in the figure below:

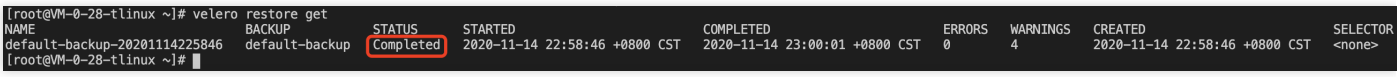
```
[root@VM-0-28-tlinux ~]# kubectl patch backupstoragelocation default --namespace velero \
>     --type merge \
>     --patch '{"spec":{"accessMode":"ReadOnly"}}'
backupstoragelocation.velero.io/default patched
[root@VM-0-28-tlinux ~]# velero backup-location get
NAME      PROVIDER   BUCKET/PREFIX        PHASE       LAST VALIDATED                  ACCESS MODE
default   aws        jo[redacted]         Available   2020-11-14 22:55:41 +0800 CST   ReadOnly
```

©2013-2022 Tencent Cloud. All rights reserved.

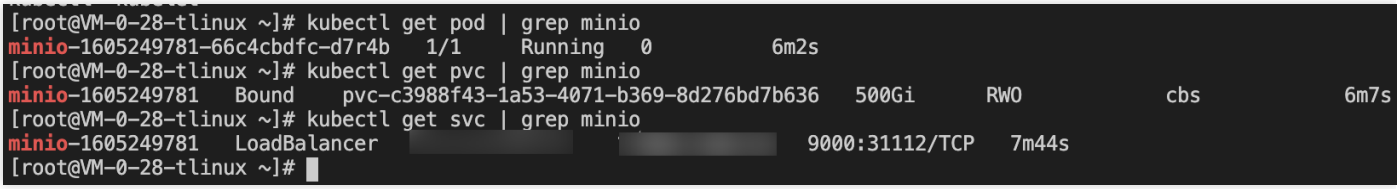
7. Run the following command to create a restoration task using the backup "default-backup" created by Velero in Step 3 above, as shown below:

```
velero restore create --from-backup default-backup
```

Use the command `velero restore get` to view the status of the restoration task. If the restoration status is "Completed" and "ERRORS" is 0, it means that the restoration task is completed, as shown in the figure below:



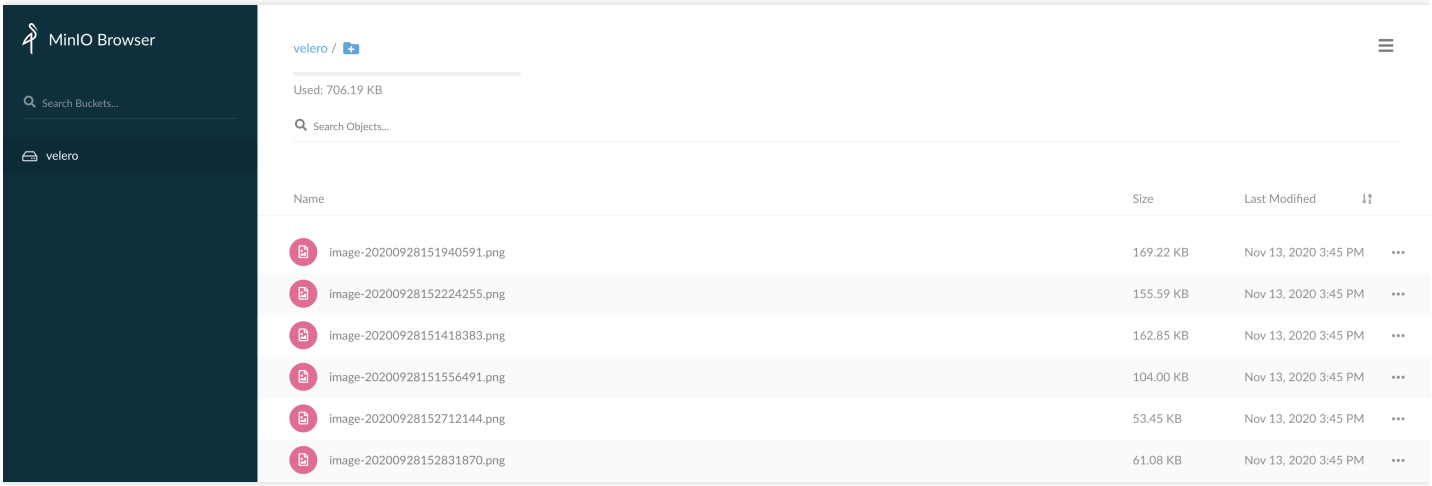
8. After the restoration, run the following command. You can find that the related resources of the previously deleted MinIO have been restored successfully, as shown below:



9. Log in to the MinIO management page on the browser. You can find the previously uploaded image, indicating that the data of the persistent volume is restored successfully, as shown below:

> Note：
> This document describes how to use Restic to backup and restore persistent volumes, but Restic does not support `hostPath` type volumes. For more information, see Restic Limitations.

©2013-2022 Tencent Cloud. All rights reserved.



10. In addition, after the restoration, you can run the following command to restore the backup storage location to read/write mode, so that you can back up normally next time, as shown below:

```
kubectl patch backupstoragelocation default --namespace velero \
--type merge \
--patch '{"spec":{"accessMode":"ReadWrite"}}'
```

Uninstalling Velero

Run the following command to uninstall Velero in the cluster, as shown below:

```
kubectl delete namespace/velero clusterrolebinding/velero
kubectl delete crds -l component=velero
```

Summary

This document mainly introduces Velero, a Kubernetes cluster resource backup tool, and shows how to configure COS as Velero's backend storage, and successfully practices the backup and restoration operations of MinIO service resources and data.

References

- Velero Official Website
- Restic Introduction
- Minio Installation
- Restic Limitations

©2013-2022 Tencent Cloud. All rights reserved.

Using Velero to Replicate Cluster Resources in TKE

Last updated：2020-12-21 09:54:37

Overview

Velero (the previous version is called Heptio Ark), an open-source tool, can safely back up and restore, perform disaster recovery, and migrate Kubernetes cluster resources and persistent volumes. Deploying Velero in the TKE cluster or self-built Kubenetes cluster, it can achieve the following features:

- Back up cluster resources and restore in case of loss.
- Migrate cluster resources to other clusters.
- Replicate the production cluster resources to development and test clusters.

For more information about Velero, see Velero official document. This document describes how to use Velero to seamlessly migrate and replicate cluster resources among TKE clusters.

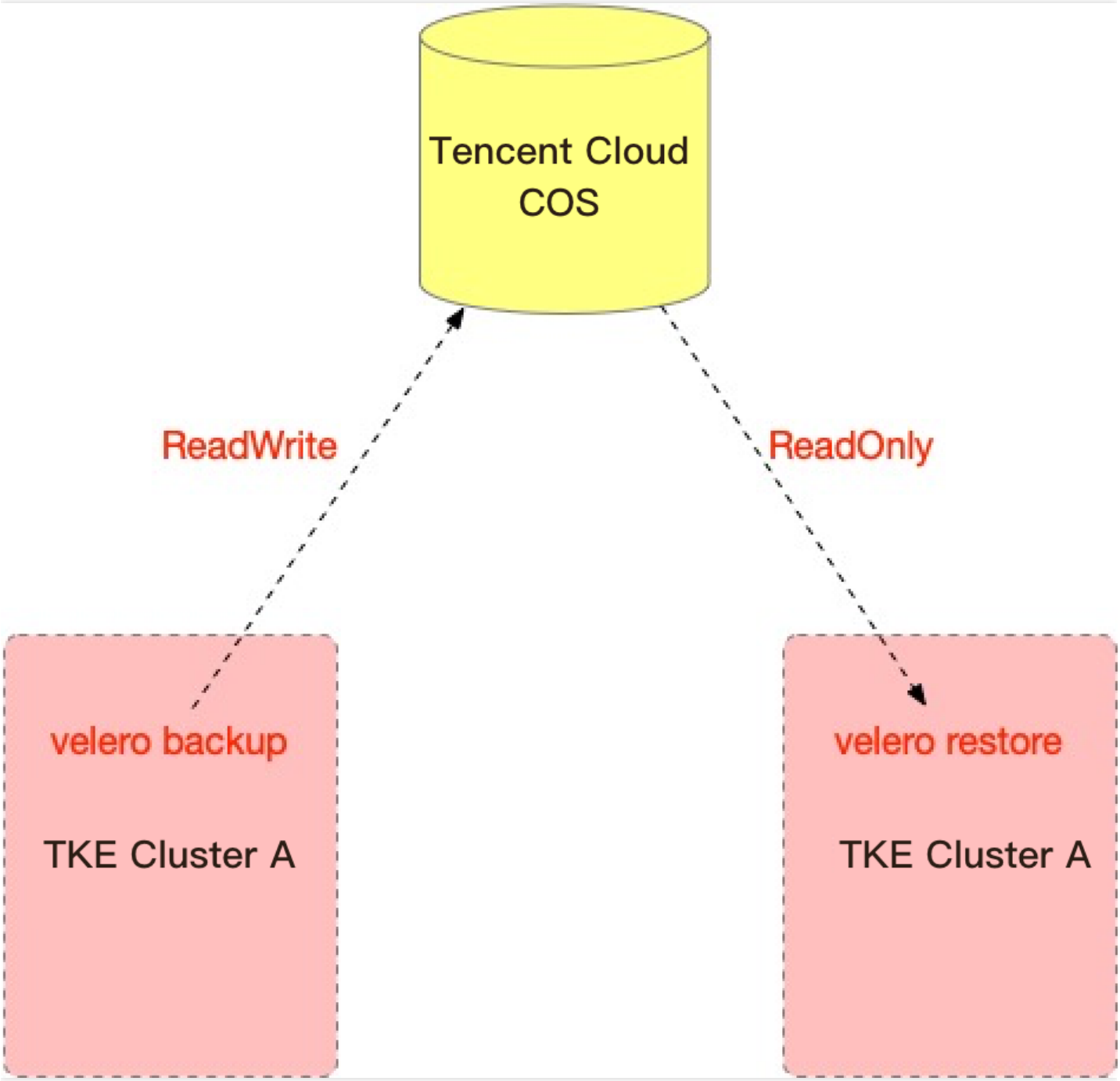
Migration Principle

Install Velero instances on both the cluster to migrate and the target cluster. The Velero instances of the two clusters point to the same COS location. The process is as follows:

1. Use Velero to perform backup operations on the cluster that needs to migrate. Generate backup data and store it in the COS.
2. Use Velero to perform data restoration on the target cluster to implement migration.

©2013-2022 Tencent Cloud. All rights reserved.

The migration principle is shown as follows:



Prerequisites

- You have registered a Tencent Cloud account.
- You have activated the COS service.
- There are already two TKE clusters: cluster A that needs to migrate and cluster B that has created a migration target. For how to create a TKE cluster, see Creating a Cluster.

©2013-2022 Tencent Cloud. All rights reserved.

- Both cluster A and cluster B need to install Velero instances (v1.5 or later version), and share the same COS bucket as Velero backend storage. For the installation steps, see Configuring Storage and Installing Velero.

Notes

- Starting from Velero v1.5, Velero can use Restic to back up all Pod volumes without annotating each Pod individually. By default, users are allowed to use Restic to back up all Pod volumes, except for the following volumes:
  - The volumes that mount the default `Service Account Secret`
  - The type volumes that mount `hostPath`
  - The volumes that mount Kubernetes `secrets` and `configmaps`

    This example requires Velero v1.5 or later version and Restic enabled to back up persistent volume data. Please ensure that parameters `--use-restic` and `--default-volumes-to-restic` are enabled during the Velero installation. For the installation steps, see Configuring Storage and Installing Velero.
- During the migration, it is not allowed to perform any CRUD operations on the resources of the clusters on both sides, so as to avoid data differences during the migration and data inconsistence after the final migration.
- Please try to make sure that the CPU, memory and other specifications of cluster A and Cluster B are the same or do not differ too much, so as to avoid the situation that the migrated Pods cannot be scheduled due to resource reasons, resulting in Pending.

Directions

Creating a backup in cluster A

Checking cluster A resources before backup

Before backing up cluster A, you can view the resources and services of cluster A for Migration Result Verification after cluster restoration.

1. This document will compare and verify the resources of the “default” and “default2” namespaces. Run the following commands to view the Pods and PVC resources in the two namespaces in cluster A, as shown in the figure below:

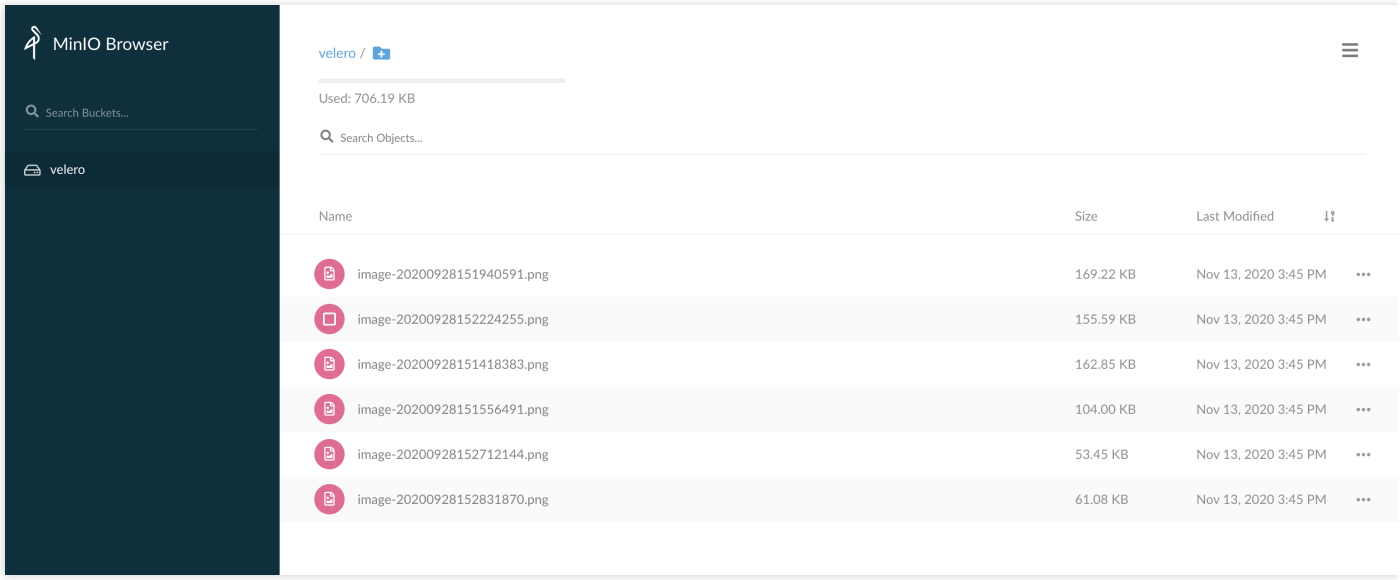
©2013-2022 Tencent Cloud. All rights reserved.

```
[root@VM-0-28-tlinux ~]# kubectl get pod
NAME                              READY   STATUS    RESTARTS   AGE
migrate-test-6bf6d577f4-7g7b9     1/1     Running   0          5d1h
minio-1605249781-66c4cbdfc-d7r4b  1/1     Running   0          21h
[root@VM-0-28-tlinux ~]# kubectl get pod -n default2
NAME                         READY   STATUS    RESTARTS   AGE
default2-58856dc878-kprhk    1/1     Running   0          21h
default2-58856dc878-zgnrw    1/1     Running   0          21h
[root@VM-0-28-tlinux ~]# kubectl get pvc
NAME              STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
minio-1605249781  Bound    pvc-d9b13f5f-6478-4c13-ae4e-942c1a7bf175   500Gi      RWO            cbs            21h
[root@VM-0-28-tlinux ~]#
```

Note：

You can specify some custom Hook operations during the backup. For example, the data in the memory of the running application needs to be persisted to disk before backup. For more information about Hook, see Backup Hooks.

2. The MinIO COS in cluster A uses persistent volumes, and some images has been uploaded, as shown in the figure below:



Cluster backup

1. Run the following command to back up all resources except the Velero namespace (the default namespace installed by Velero) in the cluster. If you need to customize the scope of the backup cluster resource, you can use the command `velero create backup -h` to view the available resource filter parameters.

```
velero backup create <BACKUP-NAME> --exclude-namespaces <NAMESPACE>
```

©2013-2022 Tencent Cloud. All rights reserved.

This document takes the creation of a “default-all” cluster backup as an example. The backup process is shown in the figure below. If the status of the backup task is "Completed", the backup is successful.

```
[root@VM-0-28-tlinux ~]# velero backup create default-all --exclude-namespaces velero
Backup request "default-all" submitted successfully.
Run `velero backup describe default-all` or `velero backup logs default-all` for more details.
[root@VM-0-28-tlinux ~]# velero backup get
NAME             STATUS       ERRORS   WARNINGS   CREATED                         EXPIRES   STORAGE LOCATION   SELECTOR
backup-default   Completed    0        0          2020-11-18 16:18:21 +0800 CST   29d       default            <none>
default-all      InProgress   0        0          2020-11-18 16:20:40 +0800 CST   29d       default            <none>
[root@VM-0-28-tlinux ~]# velero backup get
NAME             STATUS      ERRORS   WARNINGS   CREATED                         EXPIRES   STORAGE LOCATION   SELECTOR
backup-default   Completed   0        0          2020-11-18 16:18:21 +0800 CST   29d       default            <none>
default-all      Completed   0        0          2020-11-18 16:20:40 +0800 CST   29d       default            <none>
[root@VM-0-28-tlinux ~]#
```

Note：

You can also set regular automatic backups for Velero. Run the command `velero schedule -h` to view the setting method.

2. Run the following command to check if there is any errors in the backup operation. If the command does not produce any output, it means that no error occurred during the backup, as shown below:

```
velero backup logs <BACKUP-NAME> | grep error
```

Note：

Please make sure that no errors occur during the backup. If Velero makes any errors during the backup, please re-perform the backup after troubleshooting.

3. After the backup is completed, run the following command to temporarily update the backup storage location to read-only mode (optional, to prevent Velero from creating or deleting backup objects in the backup storage location during the restoration), as shown below:

```
kubectl patch backupstoragelocation default --namespace velero \
--type merge \
--patch '{"spec":{"accessMode":"ReadOnly"}}'
```

Performing a restoration in cluster B

Viewing cluster B resources before restoration

Before performing a restoration in cluster B, you can view the resources and services of cluster B for Migration Result Verification after cluster restoration.

©2013-2022 Tencent Cloud. All rights reserved.

Before the restoration, there are no workload resources under the “default” and “default2” namespaces in cluster B. Run the following commands to view the Pods and PVC resources in the two namespaces in cluster B, as shown below:

```
[root@VM-1-14-tlinux ~]# kubectl  get pod
No resources found in default namespace.
[root@VM-1-14-tlinux ~]# kubectl  get pod  -n default2
No resources found in default2 namespace.
[root@VM-1-14-tlinux ~]# kubectl  get pvc
No resources found in default namespace.
[root@VM-1-14-tlinux ~]#
```

Cluster restoration

1. Run the following command to temporarily update Velero backup storage location in cluster B to read-only mode (optional, to prevent Velero from creating or deleting backup objects in the backup storage location during the restoration), as shown below:

```
kubectl patch backupstoragelocation default --namespace velero \
--type merge \
--patch '{"spec":{"accessMode":"ReadOnly"}}'
```

> **Note：**
>
> You can specify a custom Hook operation to perform during the restoration or after the resource restoration. For example, you need to perform a custom database restoration operation before the database application container starts. For more information about Hook, see Restore Hooks.

2. Before the restoration, you need to ensure that the Velero resource in cluster B is synchronized with the backup file in COS. You can use `--backup-sync-period` to configure the synchronization interval, which is 1 minute by default. You can run the following command to check whether the backup of cluster A has been synchronized.

```
velero backup get <BACKUP-NAME>
```

3. After the backup is successfully obtained and checked, run the following command to restore all contents to cluster B.

```
velero restore create --from-backup <BACKUP-NAME>
```

©2013-2022 Tencent Cloud. All rights reserved.

The restoration process is shown in the figure below:

```
[root@VM-1-14-tlinux ~]# velero backup-location get
NAME      PROVIDER   BUCKET/PREFIX        PHASE       LAST VALIDATED                  ACCESS MODE
default   aws        jo[redacted]         Available   2020-11-18 16:49:43 +0800 CST   ReadOnly
[root@VM-1-14-tlinux ~]# velero backup get
NAME             STATUS      ERRORS   WARNINGS   CREATED                         EXPIRES   STORAGE LOCATION   SELECTOR
backup-default   Completed   0        0          2020-11-18 16:18:21 +0800 CST   29d       default            <none>
default-all      Completed   0        0          2020-11-18 16:20:40 +0800 CST   29d       default            <none>
[root@VM-1-14-tlinux ~]# velero restore create --from-backup default-all
Restore request "default-all-20201118165141" submitted successfully.
Run `velero restore describe default-all-20201118165141` or `velero restore logs default-all-20201118165141` for more details.
[root@VM-1-14-tlinux ~]#
```

4. After the restoration is completed, check the restoration log, and run the following command to check whether there are any errors and skips during restoration, as shown below:

```
# Check whether there are restoration errors during migration
velero restore logs <BACKUP-NAME> | grep error
# View the skipped restoration operations during migration
velero restore logs <BACKUP-NAME> | grep skip
```

As shown in the figure below, you can find that no errors occurred, but some "skipped" steps occurred during restoration. Because when backing up cluster resources, all cluster resources that did not contain the Velero namespace were backed up. Some cluster resources of the same type and name already existed. For example, cluster resources under kube-system. When there is a resource conflict during the restoration, Velero will skip the restoration step. In fact, the restoration process is normal and the “skipped” logs can be ignored (you can analyze these logs as needed).

```
[root@VM-1-14-tlinux ~]# velero restore logs default-all-20201118165141 | grep error
[root@VM-1-14-tlinux ~]# velero restore logs default-all-20201118165141 | grep skip
time="2020-11-18T08:51:43Z" level=info msg="Restore of ConfigMap, coredns skipped: it already exists in the cluster and is the same as the backed up version" logSource="pkg/restore/restore.go:1164" restore=velero/default-all-20201118165141
time="2020-11-18T08:51:43Z" level=info msg="Restore of ConfigMap, tke-cni-agent-conf skipped: it already exists in the cluster and is the same as the backed up version" logSource="pkg/restore/restore.go:1164" restore=velero/default-all-20201118165141
time="2020-11-18T08:51:44Z" level=info msg="Restore of ConfigMap, tke-ingress-controller-config skipped: it already exists in the cluster and is the same as the backed up version" logSource="pkg/restore/restore.go:1164" restore=velero/default-all-20201118165141
time="2020-11-18T08:51:52Z" level=info msg="Restore of ClusterRoleBinding, lb-ingress-clusterrole-nisa-binding skipped: it already exists in the cluster and is the same as the backed up version" logSource="pkg/restore/restore.go:1164" restore=velero/default-all-20201118165141
time="2020-11-18T08:51:52Z" level=info msg="Restore of ClusterRoleBinding, system:controller:attachdetach-controller skipped: it already exists in the cluster and is the same as the backed up version" logSource="pkg/restore/restore.go:1164" restore=velero/default-all-20201118165141
time="2020-11-18T08:51:52Z" level=info msg="Restore of ClusterRoleBinding, system:controller:certificate-controller skipped: it already exists in the cluster and is the same as the backed up version" logSource="pkg/restore/restore.go:1164" restore=velero/default-all-20201118165141
time="2020-11-18T08:51:52Z" level=info msg="Restore of ClusterRoleBinding, system:controller:clusterrole-aggregation-controller skipped: it already exists in the cluster and is the same as the backed up version" logSource="pkg/restore/restore.go:1164" restore=velero/default-all-20201118165141
time="2020-11-18T08:51:52Z" level=info msg="Restore of ClusterRoleBinding, system:controller:cronjob-controller skipped: it already exists in the cluster and is the same as the backed up version" logSource="pkg/restore/restore.go:1164" restore=velero/default-all-20201118165141
time="2020-11-18T08:51:52Z" level=info msg="Restore of ClusterRoleBinding, system:controller:daemon-set-controller skipped: it already exists in the cluster and is the same as the backed up version" logSource="pkg/restore/restore.go:1164" restore=velero/default-all-20201118165141
time="2020-11-18T08:51:52Z" level=info msg="Restore of ClusterRoleBinding, system:controller:deployment-controller skipped: it already exists in the cluster and is the same as the backed up version" logSource="pkg/restore/restore.go:1164" restore=velero/default-all-20201118165141
time="2020-11-18T08:51:52Z" level=info msg="Restore of ClusterRoleBinding, system:controller:disruption-controller skipped: it already exists in the cluster and is the same as the backed up version" logSource="pkg/restore/restore.go:1164" restore=velero/default-all-20201118165141
time="2020-11-18T08:51:52Z" level=info msg="Restore of ClusterRoleBinding, system:controller:endpoint-controller skipped: it already exists in the cluster and is the same as the backed up version" logSource="pkg/restore/restore.go:1164" restore=velero/default-all-20201118165141
time="2020-11-18T08:51:52Z" level=info msg="Restore of ClusterRoleBinding, system:controller:endpointslice-controller skipped: it already exists in the cluster and is the same as the backed up version" logSource="pkg/restore/restore.go:1164" restore=velero/default-all-20201118165141
time="2020-11-18T08:51:52Z" level=info msg="Restore of ClusterRoleBinding, system:controller:expand-controller skipped: it already exists in the cluster and is the same as the backed up version" logSource="pkg/restore/restore.go:1164" restore=velero/default-all-20201118165141
time="2020-11-18T08:51:52Z" level=info msg="Restore of ClusterRoleBinding, system:controller:generic-garbage-collector skipped: it already exists in the cluster and is the same as the backed up version" logSource="pkg/restore/restore.go:1164" restore=velero/default-all-20201118165141
time="2020-11-18T08:51:52Z" level=info msg="Restore of ClusterRoleBinding, system:controller:horizontal-pod-autoscaler skipped: it already exists in the cluster and is the same as the backed up version" logSource="pkg/restore/restore.go:1164" restore=velero/default-all-20201118165141
time="2020-11-18T08:51:52Z" level=info msg="Restore of ClusterRoleBinding, system:controller:job-controller skipped: it already exists in the cluster and is the same as the backed up version" logSource="pkg/restore/restore.go:1164" restore=velero/default-all-20201118165141
time="2020-11-18T08:51:52Z" level=info msg="Restore of ClusterRoleBinding, system:controller:namespace-controller skipped: it already exists in the cluster and is the same as the backed up version" logSource="pkg/restore/restore.go:1164" restore=velero/default-all-20201118165141
time="2020-11-18T08:51:52Z" level=info msg="Restore of ClusterRoleBinding, system:controller:node-controller skipped: it already exists in the cluster and is the same as the backed up version" logSource="pkg/restore/restore.go:1164" restore=velero/default-all-20201118165141
```

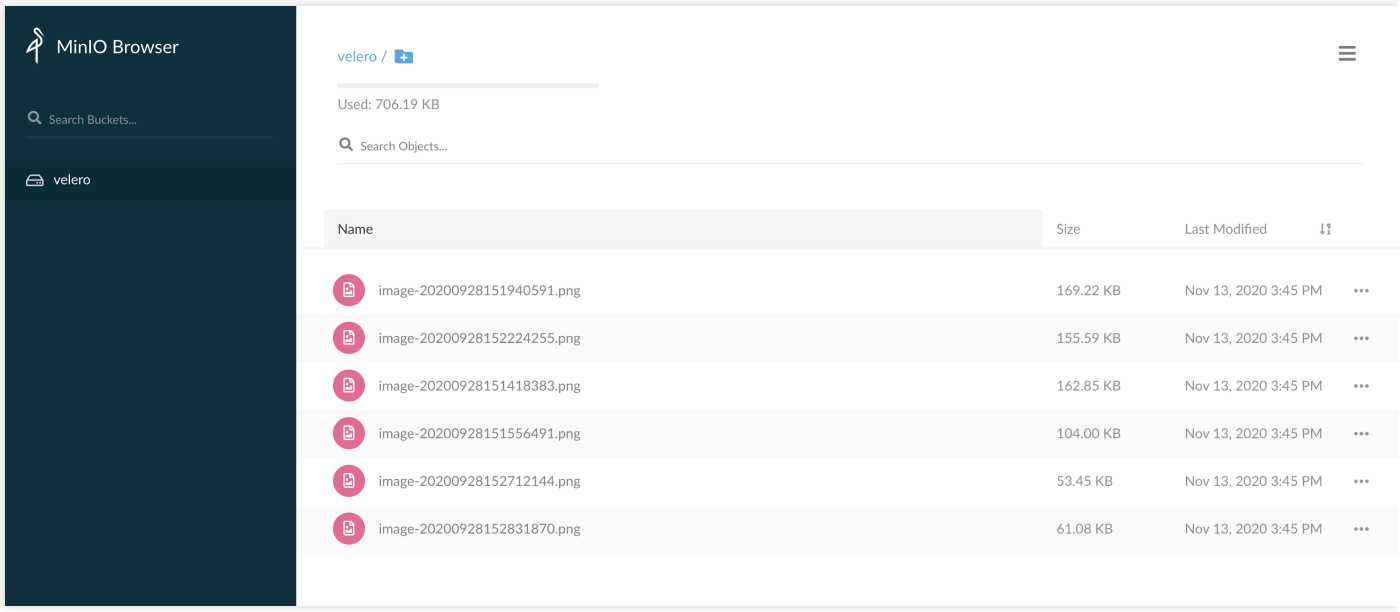
©2013-2022 Tencent Cloud. All rights reserved.

Verifying migration result

1. Run the following command to verify the cluster resources of cluster B after the migration. You can find that the Pods and PVC resources in the “default” and “default2” namespaces have been successfully migrated as expected, as shown below:

```
[root@VM-1-14-tlinux ~]# kubectl get pod
NAME                              READY   STATUS    RESTARTS   AGE
migrate-test-6bf6d577f4-7g7b9     1/1     Running   0          43m
minio-1605249781-66c4cbdfc-d7r4b  1/1     Running   0          43m
[root@VM-1-14-tlinux ~]# kubectl get pod -n default2
NAME                        READY   STATUS    RESTARTS   AGE
default2-58856dc878-kprhk   1/1     Running   0          44m
default2-58856dc878-zgnrw   1/1     Running   0          44m
[root@VM-1-14-tlinux ~]# kubectl get pvc
NAME                STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
minio-1605249781    Bound    pvc-3da9c96a-99aa-4367-b9d3-cb953ba4a57d   500Gi      RWO            cbs            44m
[root@VM-1-14-tlinux ~]#
```

2. Log in to the MinIO service in cluster B, and you can find that the images in the MinIO service are not lost, indicating that the persistent volume data has been successfully migrated as expected.



3. At this point, the migration of resources between TKE clusters has been completed.
   After the migration is completed, run the following command to restore the backup storage locations of cluster A and cluster B to read/write mode, so that the next backup task can be performed normally, as shown below:

```
kubectl patch backupstoragelocation default --namespace velero \
--type merge \
--patch '{"spec":{"accessMode":"ReadWrite"}}'
```

Summary

©2013-2022 Tencent Cloud. All rights reserved.

This document mainly introduces the principle, notes and operation methods of using Velero to migrate cluster resources among TKE clusters. The cluster resources in cluster A are successfully migrated seamlessly to cluster B. The whole migration process is simple and fast, and it is a very friendly cluster resource migration solution.

©2013-2022 Tencent Cloud. All rights reserved.

Using Velero to Migrate Cluster Resources to TKE Across Cloud Platforms

Last updated：2021-12-06 11:19:29

Overview

The open source tool Velero (formerly known as the Heptio Ark) can safely back up and restore, perform disaster recovery, and migrate Kubernetes cluster resources and persistent volumes. TKE supports using Velero to back up, restore and migrate cluster resources. For more information, see Using COS as Velero Storage to Implement Backup and Restoration of Cluster Resources and Using Velero to Migrate and Replicate Cluster Resources in TKE. This document describes how to use Velero to seamlessly migrate self-built or other cloud platform Kubernetes clusters to TKE.

Migration Principle

The principle of using Velero to migrate self-built or other cloud platform cluster is similar to the principle of Using Velero to Replicate Cluster Resources in TKE. Both the source cluster and the target cluster for migration need to install Velero instances and specify the same Tencent Cloud COS bucket. According to the actual needs, the source cluster performs backup, and the target cluster restores cluster resources, so as to implement resource migration. The difference is that when you migrate cluster resources from self-built or other cloud platforms to TKE, you need to consider and solve the problem of cluster environment differences caused by cross-platform. You can refer to the practical backup and restore strategies provided by Velero to solve the problems.

Prerequisites

- There is a self-built or other cloud platform Kubernetes cluster (cluster A), and the cluster version must be v1.10 or later.
- There is a target TKE cluster (cluster B). For how to create a TKE cluster, see Creating a Cluster.
- Both cluster A and cluster B need to install Velero instances (v1.5 or later version), and share the same COS bucket as Velero backend storage. For the installation steps, see Configuring COS and Installing Velero.
- Ensure that image resources can be pulled normally after migration.
- It is recommended that the two clusters use the same Kubernetes version to ensure the APIs are compatible.

©2013-2022 Tencent Cloud. All rights reserved.

Migration Guide

Before migration, it is recommended that you make a detailed migration plan, and consider the following points during the migration process:

Show All

Analyzing

展开&收起

Filter and classify the resource inventories that need migration and that do not need migration based on actual needs.

Considering

展开&收起

- When backing up cluster resources, you need to consider whether to perform Backup Hooks during the backup. For example, the memory data of the running application needs to be stored in disk.
- When restoring (migrating) cluster resources, you need to consider whether to perform Restoring Hooks during the restoration. For example, some initialization work needs to be prepared before restoration.

Writing

展开&收起

Write backup and restoration strategies based on the filtered and classified resource inventories. It is recommended to use the method of creating resource inventories to perform backup and restoration in complex scenarios. The YAML resource inventory is intuitive and easy to maintain. For simple migration or test scenarios, you can specify parameters to implement backup and restoration.

Processing

展开&收起

Due to the cross-cloud platform migration, the relationships of the dynamic storage classes for creating PVC may be different. You need to plan in advance whether the relationships of dynamic PVC/PV storage classes need to be remapped. And you need to create the ConfigMap configuration of the relevant mapping before the restoration. To solve more personalized differences, you can manually modify the backup resource inventory.

Checking

展开&收起

Check whether the migrated cluster resources meet expectations and the data is complete and available.

Directions

©2013-2022 Tencent Cloud. All rights reserved.

The following describes the detailed steps of migrating resources from a cloud platform cluster A to TKE cluster B. For the involved the basic knowledge of Velero backup and restoration, please refer to Practical Velero backup/restoration Knowledge.

Creating the resources of cluster A

Deploy an Nginx workload with PVC in Velero instance in cluster A. For convenience, you can directly use dynamic storage class to create PVC and PV.

1. Run the following command to view the dynamic storage class information supported by the current cluster, as shown below:

```
# Get the storage class information supported by the current cluster, where xxx-StorageClass is the storage class code name, and xxx-Provider is the provider code name (the same below).
$ kubectl get sc
NAME PROVISIONER RECLAIMPOLICY VOLUMEBINDINGMODE ALLOWVOLUMEEXPANSION AGE
xxx-StorageClass xxx-Provider Delete Immediate true 3d3h
...
```

2. Modify the PVC resource inventory in the with-pv.yaml file, and use the storage class named "xxx-StorageClass" in the cluster to dynamically create, as shown below:

```
...
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
name: nginx-logs
namespace: nginx-example
labels:
app: nginx
spec:
# Optional: modify the value of the PVC storage class to the cloud platform of cluster A.
storageClassName: xxx-StorageClass
accessModes:
- ReadWriteOnce
resources:
requests:
storage: 20Gi # Since the minimum storage of this cloud platform is 20 Gi, you need to modify the storage to 20 Gi in this sample.
...
```

©2013-2022 Tencent Cloud. All rights reserved.

3. Run the following command to apply with-pv.yaml in the sample to create the following cluster resources (nginx-example namespace), as shown below:

```
$ kubectl apply -f with-pv.yaml
namespace/nginx-example created
persistentvolumeclaim/nginx-logs created
deployment.apps/nginx-deployment created
service/my-nginx created
```

4. The created PVC “nginx-logs” has been mounted to the `/var/log/nginx` directory of the Nginx container as the log storage of service. The sample here will test and access the Nginx service in the browser to generate log data for the mounted PVC for data comparison after restoration, as shown below:

```
$ kubectl exec -it nginx-deployment-5ccc99bffb-6nm5w bash -n nginx-example
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future ver
sion. Use kubectl kubectl exec [POD] -- [COMMAND]
Defaulting container name to nginx.
Use 'kubectl describe pod/nginx-deployment-5ccc99bffb-6nm5w -n nginx-example'
to see all of the containers in this pod

$ du -sh /var/log/nginx/
84K /var/log/nginx/

# View the first two logs of accss.log and error.log.
$ head -n 2 /var/log/nginx/access.log
192.168.0.73 - - [29/Dec/2020:03:02:31 +0000] "GET /?spm=5176.2020520152.0.0.2
2d016ddHXZumX HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10
_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.3
6" "-"
192.168.0.73 - - [29/Dec/2020:03:02:32 +0000] "GET /favicon.ico HTTP/1.1" 404
555 "http://47.242.233.22/?spm=5176.2020520152.0.0.22d016ddHXZumX" "Mozilla/5.
0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) C
hrome/87.0.4280.88 Safari/537.36" "-"

$ head -n 2 /var/log/nginx/error.log
2020/12/29 03:02:32 [error] 6#6: *597 open() "/usr/share/nginx/html/favicon.ic
o" failed (2: No such file or directory), client: 192.168.0.73, server: localh
ost, request: "GET /favicon.ico HTTP/1.1", host: "47.242.233.22", referrer: "h
ttp://47.242.233.22/?spm=5176.2020520152.0.0.22d016ddHXZumX"
2020/12/29 03:07:21 [error] 6#6: *1172 open() "/usr/share/nginx/html/0bef" fai
led (2: No such file or directory), client: 192.168.0.73, server: localhost, r
equest: "GET /0bef HTTP/1.0"
```

©2013-2022 Tencent Cloud. All rights reserved.

Confirming the resource inventories to migrate

1. Run the following command to output all resource inventories in cluster A.

```
kubectl api-resources --verbs=list -o name | xargs -n 1 kubectl get --show-kind
--ignore-not-found --all-namespaces
```

You can also run the following commands to distinguish namespaces based on resources and narrow the scope of output resources:

- View the resource inventories that do not distinguish namespaces:

```
kubectl api-resources --namespaced=false --verbs=list -o name | xargs -n 1 ku
bectl get --show-kind --ignore-not-found
```

- View the resource inventories that distinguish namespaces:

```
kubectl api-resources --namespaced=true --verbs=list -o name | xargs -n 1 kub
ectl get --show-kind --ignore-not-found --all-namespaces
```

2. You can filter the resource inventories that need to migrate based on the actual needs. The sample here will directly migrate Nginx workload-related resources under the "nginx-example" namespace from this cloud platform to TKE. The resources involved are as follows:

```
$ kubectl get all -n nginx-example
NAME READY STATUS RESTARTS AGE
pod/nginx-deployment-5ccc99bffb-tn2sh 2/2 Running 0 2d19h

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
service/my-nginx LoadBalancer 172.21.1.185 x.x.x.x 80:31455/TCP 2d19h

NAME READY UP-TO-DATE AVAILABLE AGE
deployment.apps/nginx-deployment 1/1 1 1 2d19h

NAME DESIRED CURRENT READY AGE
replicaset.apps/nginx-deployment-5ccc99bffb 1 1 1 2d19h

$ kubectl get pvc -n nginx-example
NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE
nginx-logs Bound d-j6ccrq4k1moziu1l6l5r 20Gi RWO xxx-StorageClass 2d19h
```

©2013-2022 Tencent Cloud. All rights reserved.

```
$ kubectl get pv
NAME CAPACITY ACCESS MODES RECLAIM POLICY STATUS CLAIM STORAGECLASS REASON AGE
d-j6ccrq4k1moziu1l6l5r 20Gi RWO Delete Bound nginx-example/nginx-logs xxx-Stor
ageClass 2d19h
```

Confirming Hook strategy

The sample has configured a Hook strategy that is "setting the file system to read-only before backing up the Nginx workload and restoring it to read/write after the backup" in with-pv.yaml. The YAML file is as follows:

```
...
annotations:
# The annotation of the backup hook strategy indicates that the nginx log direct
ory is set to read-only mode before starting the backup, and is restored to rea
d/write mode after the backup is completed.
pre.hook.backup.velero.io/container: fsfreeze
pre.hook.backup.velero.io/command: '["/sbin/fsfreeze", "--freeze", "/var/log/ngi
nx"]'
post.hook.backup.velero.io/container: fsfreeze
post.hook.backup.velero.io/command: '["/sbin/fsfreeze", "--unfreeze", "/var/log/
nginx"]'
spec:
volumes:
- name: nginx-logs
persistentVolumeClaim:
claimName: nginx-logs
containers:
- image: nginx:1.17.6
name: nginx
ports:
- containerPort: 80
volumeMounts:
- mountPath: "/var/log/nginx"
name: nginx-logs
readOnly: false
- image: ubuntu:bionic
name: fsfreeze
securityContext:
privileged: true
volumeMounts:
- mountPath: "/var/log/nginx"
name: nginx-logs
...
```

©2013-2022 Tencent Cloud. All rights reserved.

Starting migration

Write a backup and restoration strategy based on the actual situation, and begin to migrate the Nginx workload related resources of the cloud platform.

Performing backup in cluster A

1. Create the following YAML file to back up the resources that need to migrate.

```
apiVersion: velero.io/v1
kind: Backup
metadata:
name: migrate-backup
# Must be the namespace installed by velero.
namespace: velero
spec:
# The resources that only contains the nginx-example namespace.
includedNamespaces:
- nginx-example
# The resources that do not distinguish namespace.
includeClusterResources: true
# Specify the storage location of the backup data.
storageLocation: default
# Specify the storage location of the volume snapshot.
volumeSnapshotLocations:
- default
# Use restic to back up the volume.
defaultVolumesToRestic: true
```

2. The backup process is shown below. When the backup status is "Completed" and the number of errors is 0, the backup process is complete and correct.

```
$ kubectl apply -f backup.yaml
backup.velero.io/migrate-backup created
$ velero backup get
NAME STATUS ERRORS WARNINGS CREATED EXPIRES STORAGE LOCATION SELECTOR
migrate-backup InProgress 0 0 2020-12-29 19:24:12 +0800 CST 29d default <none>
$ velero backup get
NAME STATUS ERRORS WARNINGS CREATED EXPIRES STORAGE LOCATION SELECTOR
migrate-backup Completed 0 0 2020-12-29 19:24:28 +0800 CST 29d default <none>
```

3. After the backup is complete, run the following command to temporarily update the backup storage location to read-only mode, as shown below:

©2013-2022 Tencent Cloud. All rights reserved.

> Note
> This can prevent Velero from creating or deleting backup objects in the backup storage location during the restoration. (Optional)

```
kubectl patch backupstoragelocation default --namespace velero \
--type merge \
--patch '{"spec":{"accessMode":"ReadOnly"}}'
```

Processing resource differences across cloud platforms

1. Due to differences in the dynamic storage classes used, you need to create a dynamic storage class name mapping for the persistent volume "nginx-logs" through the ConfigMap shown below.

```
apiVersion: v1
kind: ConfigMap
metadata:
name: change-storage-class-config
namespace: velero
labels:
velero.io/plugin-config: ""
velero.io/change-storage-class: RestoreItemAction
data:
# Storage class name is mapped to Tencent cloud dynamic storage class cbs.
xxx-StorageClass: cbs
```

2. Run the following command to apply the above ConfigMap configuration, as shown below:

```
$ kubectl apply -f cm-storage-class.yaml
configmap/change-storage-class-config created
```

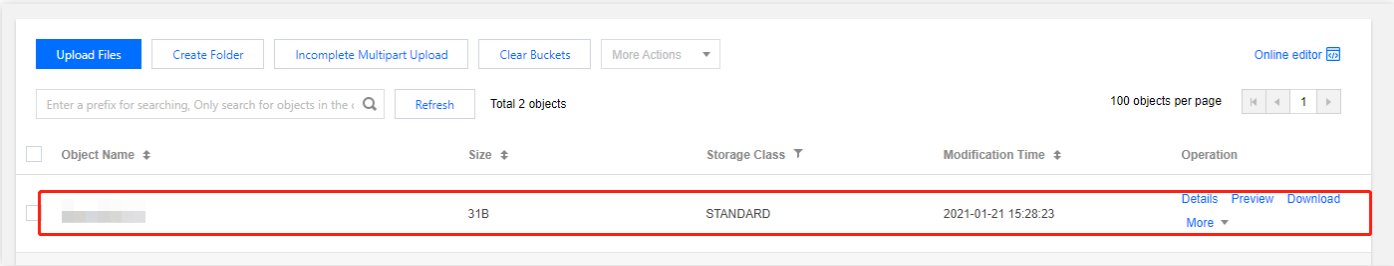
3. The resource inventories backed up by Velero is stored in COS in JSON format. If you have a more personalized migration requirement, you can directly download the backup file and customize it. The sample below will add a "jokey-test:jokey-test" annotation to the Deployment resource of Nginx. The modification process is as follows:

```
$ Downloads % mkdir migrate-backup
# Decompress the backup file.
$ Downloads % tar -zxvf migrate-backup.tar.gz -C migrate-backup
```

©2013-2022 Tencent Cloud. All rights reserved.

```
# Edit the resources that need to be customized. In the sample below, "jokey-test" is added to the Deployment resource of Nginx: "jokey-test" annotation.
$ migrate-backup % cat resources/deployments.apps/namespaces/nginx-example/nginx-deployment.json
{"apiVersion":"apps/v1","kind":"Deployment","metadata":{"annotations":{"jokey-test":"jokey-test",...
# Repack the modified backup files.
$ migrate-backup % tar -zcvf migrate-backup.tar.gz *
```

4. Complete the custom modification and repackage, and log in to COS console to upload the backup file and replace the original backup file.



Performing the restoration in cluster B

1. The sample uses the resource inventories shown below to perform the restoration (migration).

```
apiVersion: velero.io/v1
kind: Restore
metadata:
name: migrate-restore
namespace: velero
spec:
backupName: migrate-backup
includedNamespaces:
- nginx-example
# Fill in the resource type to be restored as needed. There is no resource to be excluded under the nginx-example namespace, so enter '*' here.
includedResources:
- '*'
includeClusterResources: null
# Resources not included in the restoration. Here storageClasses resource types are excluded.
excludedResources:
- storageclasses.storage.k8s.io
# Use the labelSelector selector to select the resource with a specific label. Since there is no need to use the label selector to filter in this sample, please make an annotation here.
```

©2013-2022 Tencent Cloud. All rights reserved.

```
# labelSelector:
# matchLabels:
# app: nginx
# Set the relationship mapping strategy of the namespace.
namespaceMapping:
nginx-example: default
restorePVs: true
```

2. The execution of the restoration process is shown below. When the restoration status is "Completed" and the number of "errors" is 0, it means the restoration process is complete and correct.

```
$ kubectl apply -f restore.yaml
restore.velero.io/migrate-restore created
$ velero restore get
NAME BACKUP STATUS STARTED COMPLETED ERRORS WARNINGS CREATED SELECTOR
migrate-restore migrate-backup Completed 2021-01-12 20:39:14 +0800 CST 2021-01-
12 20:39:17 +0800 CST 0 0 2021-01-12 20:39:14 +0800 CST <none>
```

Checking the migrated resources

1. Run the following command to check whether the running status of the migrated resource is normal, as shown below:

```
# Since the "nginx-example" namespace is specified to map to the "default" name
space when restoration, the restored resource will run under the "default" name
space.
$ kubectl get all -n default
NAME READY STATUS RESTARTS AGE
pod/nginx-deployment-5ccc99bffb-6nm5w 2/2 Running 0 49s
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
service/kube-user LoadBalancer 172.16.253.216 10.0.0.28 443:30060/TCP 8d
service/kubernetes ClusterIP 172.16.252.1 <none> 443/TCP 8d
service/my-nginx LoadBalancer 172.16.254.16 x.x.x.x 80:30840/TCP 49s
NAME READY UP-TO-DATE AVAILABLE AGE
deployment.apps/nginx-deployment 1/1 1 1 49s
NAME DESIRED CURRENT READY AGE
replicaset.apps/nginx-deployment-5ccc99bffb 1 1 1 49s
```

From the command execution result, you can find that the running status of the migrated resource is normal.

2. Check whether the set restoration strategy is successful.

©2013-2022 Tencent Cloud. All rights reserved.

i. Run the following command to check whether the mapping of the dynamic storage class name is correct, as shown below:

```
# You can find that the storage class of PVC/PV is already "cbs", indicating that the storage class mapping is successful.
$ kubectl get pvc -n default
NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE
nginx-logs Bound pvc-bcc17ccd-ec3e-4d27-bec6-b0c8f1c2fa9c 20Gi RWO cbs 55s
$ kubectl get pv
NAME CAPACITY ACCESS MODES RECLAIM POLICY STATUS CLAIM STORAGECLASS REASON AGE
pvc-bcc17ccd-ec3e-4d27-bec6-b0c8f1c2fa9c 20Gi RWO Delete Bound default/nginx-logs cbs 57s
```

If the storage class of PVC/PV is "cbs", the storage class mapping is successful. From the execution result of the above command, you can find that the storage class mapping is successful.

ii. Run the following command to check whether the custom-added "jokey-test" annotation for "deployment.apps/nginx-deployment" before restoration is successful, as shown below:

```
# Obtain the annotation "jokey-test" successfully, indicating that the custom modification of the resource is successful.
$ kubectl get deployment.apps/nginx-deployment -o custom-columns=annotations:.metadata.annotations.jokey-test
annotations
jokey-test
```

If the annotations can be obtained normally, the custom resource has been modified successfully. From the execution result of the above command, you can find that the namespace mapping configuration is successful.

3. Run the following command to check whether the PVC data mounted by the workload is successfully migrated.

```
# Check the data size in the mounted PVC data directory. The data size is 88K, which is more than the size before the migration. The reason is that Tencent Cloud CLB actively initiated a health check and generated some logs.
$ kubectl exec -it nginx-deployment-5ccc99bffb-6nm5w -n default -- bash
Defaulting container name to nginx.
Use 'kubectl describe pod/nginx-deployment-5ccc99bffb-6nm5w -n default' to see all of the containers in this pod.

$ du -sh /var/log/nginx
88K /var/log/nginx
```

©2013-2022 Tencent Cloud. All rights reserved.

```
# Check the first two log information, which is the same as the log before the migration, indicating that the PVC data is not lost.
$ head -n 2 /var/log/nginx/access.log
192.168.0.73 - - [29/Dec/2020:03:02:31 +0000] "GET /?spm=5176.2020520152.0.0.22d016ddHXZumX HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36" "-"
192.168.0.73 - - [29/Dec/2020:03:02:32 +0000] "GET /favicon.ico HTTP/1.1" 404 555 "http://47.242.233.22/?spm=5176.2020520152.0.0.22d016ddHXZumX" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36" "-"

$ head -n 2 /var/log/nginx/error.log
2020/12/29 03:02:32 [error] 6#6: *597 open() "/usr/share/nginx/html/favicon.ico" failed (2: No such file or directory), client: 192.168.0.73, server: localhost, request: "GET /favicon.ico HTTP/1.1", host: "47.242.233.22", referrer: "http://47.242.233.22/?spm=5176.2020520152.0.0.22d016ddHXZumX"
2020/12/29 03:07:21 [error] 6#6: *1172 open() "/usr/share/nginx/html/0bef" failed (2: No such file or directory), client: 192.168.0.73, server: localhost, request: "GET /0bef HTTP/1.0"
```

From the result of the above command, you can find that the PVC data mounted by the workload is successfully migrated. So far, the Nginx (nginx-example namespace) workload-related resources and data in the cluster A have been successfully migrated to TKE cluster B (default namespace).

Summary

This document mainly describes the ideas and methods of using Velero to migrate the resources of the self-built or other cloud platform clusters to TKE, and shows the sample that the cluster resources in cluster A are successfully migrated to cluster B seamlessly. If you encounter scenarios that are not covered in this document during the actual migration, please submit a ticket.

Appendix: Practical Velero Backup/Restoration Knowledge

Velero provides many useful backup and restoration strategies, as shown below:

Resource filtering

Velero includes all objects in a backup or restoration when no filtering options are used. You can specify parameters to filter resources as needed during **backup and restoration**. For details, please refer to Resource Filtering.

©2013-2022 Tencent Cloud. All rights reserved.

- **Includes:**

Parameters	Description
`--include-resources`	Specify a list of resource objects to include.
`--include-namespaces`	Specify a list of namespaces to include.
`--include-cluster-resources`	Specify whether to include resources of the cluster.
`--selector`	Specify to include the resources that match the label selector.

- **Excludes:**

Parameters	Description
`--exclude-namespaces`	Specify a list of namespaces to be excluded.
`--exclude-resources`	Specify a list of resource objects to be excluded.
`velero.io/exclude-from-backup=true`	This configuration item will configure this label attribute for the resource object, and the resource object with this label will be excluded.

Hook operation

- Execute Hook operation during **backup**, for example, you need to store the memory data in disk before backup. For details, see Backup Hooks.
- Execute Hook operation during **restoration**, for example, you need to determine whether component dependencies are available before restoration. For details, see Restore Hooks.
- For configuring the mapping relationship between PVC/PV volumes during **restoration**, please refer to the following documents. For more details, see Restore Reference.
  - Changing PV/PVC Storage Classes
  - Changing PVC selected-node

Configuration of using Restic to back up volume

Starting from Velero v1.5, Velero uses Restic to back up all Pod volumes by default instead of annotating each Pod separately. **Velero v1.5 or later is recommended**

For the Velero version that is earlier than v1.5, when Velero uses Restic to back up volumes, Restic provides the following two ways to find the Pod volumes that need to be backed up:

- The used Pod volume backup selects to contain an annotation (default):

©2013-2022 Tencent Cloud. All rights reserved.

```
kubectl -n <YOUR_POD_NAMESPACE> annotate <pod/YOUR_POD_NAME> backup.velero.io/backup-volumes=<YOUR_VOLUME_NAME_1,YOUR_VOLUME_NAME_2,...>
```

- The used Pod volume backup selects to not contain an annotation:

```
kubectl -n <YOUR_POD_NAMESPACE> annotate <pod/YOUR_POD_NAME> backup.velero.io/backup-volumes-excludes=<YOUR_VOLUME_NAME_1,YOUR_VOLUME_NAME_2,...>
```

Related commands

- After the backup is complete, run the following command to view the backup volume information:

```
kubectl -n velero get podvolumebackups -l velero.io/backup-name=<YOUR_BACKUP_NAME> -o yaml
```

- After the restoration is complete, run the following command to view the restore volume information:

```
kubectl -n velero get podvolumerestores -l velero.io/restore-name=<YOUR_RESTORE_NAME> -o yaml
```

Other operations

- In addition to using the Velero command to perform the backup, it can also be triggered by **creating a backup resource (recommended)**. For the configuration example, see [Backup Example](https://velero.io/docs/v1.5/api-types/backup/#definition). For detailed API field definitions, see Backup API Definition.
- In addition to using the Velero command to perform the restoration, it can also be triggered by **creating a restoration resource (recommended)**. For the configuration example, see Restoration Example. For detailed API field definitions, see Restore API Definition.
- If there are other personalized resource configurations such as annotations and label, you can manually edit the backup JSON resource inventory file before restoration.

©2013-2022 Tencent Cloud. All rights reserved.

Guide on Migrating Resources in a TKE Managed Cluster to an Serverless Cluster

Last updated：2022-12-13 18:23:37

Prerequisites

- A TKE managed cluster on v1.18 or later (cluster A) exists.
- A migration target TKE Serverless cluster on v1.20 or later (cluster B) is created. For how to create a TKE Serverless cluster, see Connecting to a Cluster.
- Both cluster A and cluster B need to share the same COS bucket as Velero backend storage. For how to configure a COS bucket, see Configuring COS.
- We recommend that Clusters A and B be under the same VPC, so that you can back up data in the PVC.
- Make sure that image resources can be pulled properly after migration. For how to configure an image repository in a TKE Serverless cluster, see Image Repository FAQs.
- Make sure that the Kubernetes versions of both clusters are compatible. We recommend you use the same version. If cluster A is on a lower version, upgrade it before migration.

Limitations

- After workloads with a fixed IP are enabled in a TKE cluster, their IPs will change after the migration to a TKE Serverless cluster. You can specify an IP to create a Pod in the Pod template, for example `eks.tke.cloud.tencent.com/pod-ip: "xx.xx.xx.xx"` .
- TKE Serverless clusters with containerd as the container runtime are not compatible with images from Docker Registry v2.5 or earlier, or Harbor v1.10 or earlier.
- In a TKE Serverless cluster, each Pod comes with 20 GiB temporary disk space for image storage by default, which is created and terminated along the lifecycle of the Pod. If you need larger disk space, mount other types of volumes, such as PVC volumes, for data storage.
- When deploying DaemonSet workloads on a TKE Serverless cluster, you need to deploy them on business pods in sidecar mode.
- When deploying NodePort services on a TKE Serverless cluster, you cannot access the services through `NodeIP:Port` . Instead, you need to use `ClusterIP:Port` to access the services.
- Pods deployed on a TKE Serverless cluster expose monitoring data via port 9100 by default. If your business Pod requires listening on port 9100, you can avoid conflicts by using other ports to collect monitoring data when creating

©2013-2022 Tencent Cloud. All rights reserved.

a Pod. For example, you can configure as follows: `eks.tke.cloud.tencent.com/metrics-port: "9110"` .

- In addition to the preceding limitations, other points for attention of TKE Serverless clusters are described here.

Migration Directions

The following describes how to migrate resources from TKE cluster A to TKE Serverless cluster B.

Configuring COS

For operation details, see Creating a bucket.

Downloading Velero

1. Download the latest version of Velero to the cluster environment. Velero v1.8.1 is used as an example in this document.

```
wget https://github.com/vmware-tanzu/velero/releases/download/v1.8.1/velero-v1.8.1-linux-amd64.tar.gz
```

2. Run the following command to decompress the installation package, which contains Velero command lines and some sample files.

```
tar -xvf velero-v1.8.1-linux-amd64.tar.gz
```

3. Run the following command to migrate the Velero executable file from the decompressed directory to the system environment variable directory, that is, `/usr/bin` in this document, as shown below:

```
cp velero-v1.8.1-linux-amd64/velero /usr/bin/
```

Installing Velero in clusters A and B

1. Configure the Velero client and enable CSI.

```
velero client config set features=EnableCSI
```

©2013-2022 Tencent Cloud. All rights reserved.

2. Run the following command to install Velero in clusters A and B and create Velero workloads as well as other necessary resource objects.

- Below is an example of using CSI for PVC backup:

```
velero install --provider aws \
--plugins velero/velero-plugin-for-aws:v1.1.0,velero/velero-plugin-for-csi:v0.2.0 \
--features=EnableCSI \
--features=EnableAPIGroupVersions \
--bucket <BucketName> \
--secret-file ./credentials-velero \
--use-volume-snapshots=false \
--backup-location-config region=ap-guangzhou,s3ForcePathStyle="true",s3Url=https://cos.ap-guangzhou.myqcloud.com
```

Note：
TKE Serverless clusters do not support DaemonSet deployment, so none of the samples in this document support the restic add-on.

- If you don't need to back up the PVC, see the following installation sample:

```
./velero install --provider aws --use-volume-snapshots=false --bucket gtest-1251707795 --plugins velero/velero-plugin-for-aws:v1.1.0 --secret-file ./credentials-velero --backup-location-config region=ap-guangzhou,s3ForcePathStyle="true",s3Url=https://cos.ap-guangzhou.myqcloud.com
```

For installation parameters, see Using COS as Velero Storage to Implement Backup and Restoration of Cluster Resources or run the `velero install --help` command.
Other installation parameters are as described below:

Parameter	Configuration
--plugins	Use the AWS S3 API-compatible add-on `velero-plugin-for-aws`; use the CSI add-on velero-plugin-for-csi to back up `csi-pv`. We recommend you enable it.
--features	Enable optional features:Enable the API group version feature. This feature is used for compatibility with different API group versions and we recommend you enable it.Enable the CSI snapshot feature. This feature is used to back up the CSI-supported PVC, so we recommend you enable it.

©2013-2022 Tencent Cloud. All rights reserved.

Parameter	Configuration
--use-restic	Velero supports the restic open-source tool to back up and restore Kubernetes storage volume data (`hostPath` volumes are not supported. For details, see here). It's used to supplement the Velero backup feature. During the migration to a TKE Serverless cluster, enabling this parameter will fail the backup.
--use-volume-snapshots=false	Disable the default snapshot backup of storage volumes.

3. After the installation is complete, wait for the Velero workload to be ready. Run the following command to check whether the configured storage location is available. If `Available` is displayed, the cluster can access the COS bucket.

```
velero backup-location get
NAME PROVIDER BUCKET/PREFIX PHASE LAST VALIDATED ACCESS MODE DEFAULT
default aws <BucketName> Available 2022-03-24 21:00:05 +0800 CST ReadWrite true
```

At this point, you have completed the Velero installation. For more information, see Velero Documentation.

(Optional) Installing `VolumeSnapshotClass` in clusters A and B

Note：

- Skip this step if you don't need to back up the PVC.
- For more information on storage snapshot, see Backing up and Restoring PVC via CBS-CSI Add-on.

1. Check that you have installed the CBS-CSI add-on.

2. You have granted related permissions of CBS snapshot for `TKE_QCSRole` on the Access Management page of the console. For details, see CBS-CSI.

3. Use the following YAML to create a VolumeSnapshotClass object, as shown below:

```
apiVersion: snapshot.storage.k8s.io/v1beta1
kind: VolumeSnapshotClass
metadata:
labels:
velero.io/csi-volumesnapshot-class: "true"
name: cbs-snapclass
driver: com.tencent.cloud.csi.cbs
deletionPolicy: Delete
```

©2013-2022 Tencent Cloud. All rights reserved.

4. Run the following command to check whether the VolumeSnapshotClass has been created successfully, as shown below:

```
$ kubectl get volumesnapshotclass
NAME DRIVER DELETIONPOLICY AGE
cbs-snapclass com.tencent.cloud.csi.cbs Delete 17m
```

(Optional) Creating sample resource for cluster A

> Note：
> Skip this step if you don't need to back up the PVC.

Deploy a MinIO workload with the PVC in a Velero instance in cluster A. Here, the `cbs-csi` dynamic storage class is used to create the PVC and PV.

1. Use `provisioner` in the cluster to dynamically create the PV for the `com.tencent.cloud.csi.cbs` storage class. A sample PVC is as follows:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
annotations:
volume.beta.kubernetes.io/storage-provisioner: com.tencent.cloud.csi.cbs
name: minio
spec:
accessModes:
- ReadWriteOnce
resources:
requests:
storage: 10Gi
storageClassName: cbs-csi
volumeMode: Filesystem
```

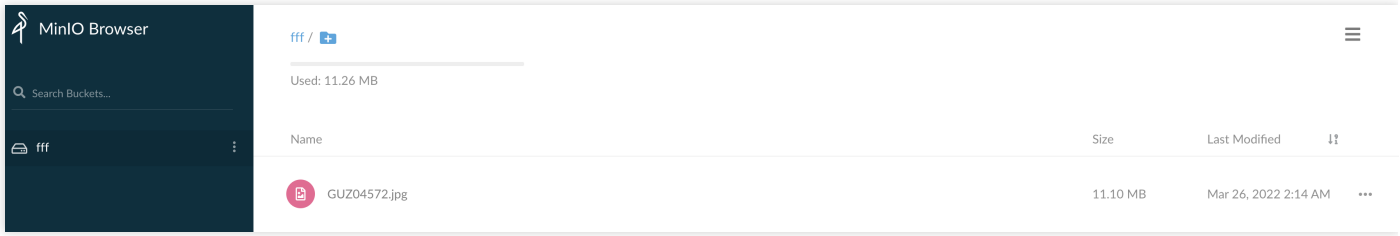
2. Use the Helm tool to create a MinIO testing service that references the above PVC. For more information on MinIO installation, see here. In this sample, a load balancer has been bound to the MinIO service, and you can access the

©2013-2022 Tencent Cloud. All rights reserved.

management page by using a public network address.

```
[root@VM-0-28-tlinux ~]# kubectl get pod | grep minio
minio-1605249781-66c4cbdfc-d7r4b   1/1     Running   0          30h
[root@VM-0-28-tlinux ~]# kubectl get svc | grep minio
minio-1605249781   LoadBalancer                                  9000:30252/TCP   31h
[root@VM-0-28-tlinux ~]#
```

3. Log in to the MinIO web management page and upload the images for testing as shown below:



Backup and restoration

1. To create a backup in cluster A, see Creating a backup in cluster A in the **Cluster Migration** directions.
2. To perform a restoration in cluster B, see Performing a restoration in cluster B in the **Cluster Migration** directions.
3. Verify the migration result:
   - If you don't need to back up the PVC, see Verifying migration result in the **Cluster Migration** directions.
   - If you need to back up the PVC, perform a verification as follows:

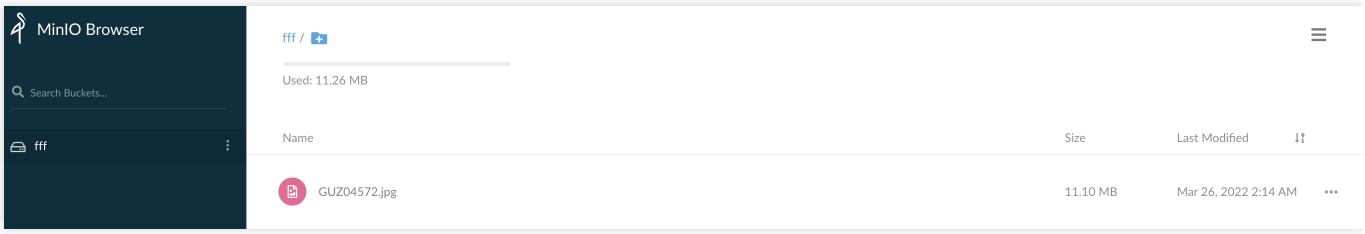
     a. Run the following command to verify the resources in cluster B after migration. You can see that the Pods, PVC, and Service have been successfully migrated as shown below:

```
~]$ k get po
NAME                     READY   STATUS    RESTARTS   AGE
minio-                   1/1     Running   0          59s
~]$ k get pvc
NAME    STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
minio   Bound    pvc-b608cce1-                              10Gi       RWO            cbs            5d16h
~]$ k get svc
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
kubernetes   ClusterIP      192.168.0.1     <none>        443/TCP          5d19h
minio        LoadBalancer   192.168.1.103                 9000:    /TCP    52s
```

     b. Log in to the MinIO service in cluster B. You can see that the images in the MinIO service are not lost,

©2013-2022 Tencent Cloud. All rights reserved.

indicating that the persistent volume data has been successfully migrated as expected.



4. Now, resource migration from the TKE cluster to the TKE Serverless cluster is completed.
   After the migration is complete, run the following command to restore the backup storage locations of clusters A and B to read/write mode as shown below, so that the next backup task can be performed normally:

```
kubectl patch backupstoragelocation default --namespace velero \
```

```
  --type merge
--patch '{"spec":{"accessMode":"ReadWrite"}}'
```

Serverless Cluster FAQs

- Failed to pull an image: See Image Repository.
- Failed to perform a DNS query: This type of failure often takes the form of failing to pull a Pod image or deliver logs to a self-built Kafka cluster. For more information, see Customized DNS Service of Serverless Cluster.
- Failed to deliver logs to CLS: When you use a TKE Serverless cluster to deliver logs to CLS for the first time, you need to authorize the service as instructed in Enabling Log Collection.
- By default, up to 100 Pods can be created for each cluster. If you need to create more, see Default Quota.
- When Pods are frequently terminated and recreated, the `Timeout to ensure pod sandbox` error is reported: The add-ons in TKE Serverless cluster Pods communicate with the control plane for health checks. If the network remains disconnected for six minutes after Pod creation, the control plane will initiate the termination and recreation. In this case, you need to check whether the security group associated with the Pod has allowed access to the 169.254 route.
- Pod port access failure/not ready:
  - Check whether the service container port conflicts with the TKE Serverless cluster control plane port as instructed in Port Limits.
  - If the Pod can be pinged succeeded, but the telnet failed, check the security group.

©2013-2022 Tencent Cloud. All rights reserved.

- When creating an instance, you can use the following features to speed up image pull: Mirror cache and Mirror reuse.
- Failed to dump business logs: After a TKE Serverless job business exits, the underlying resources are repossessed, and container logs can't be viewed by using the `kubectl logs` command, adversely affecting debugging. You can dump the business logs by delaying the termination or setting the `terminationMessage` field as instructed in How to set container's termination message?.
- The Pod restarts frequently, and the `ImageGCFailed` error is reported: A TKE Serverless cluster Pod has 20 GiB disk size by default. If the disk usage reaches 80%, the TKE Serverless cluster control plane will trigger the container image repossession process to try to repossess the unused images and free up the space. If it fails to free up any space, `ImageGCFailed: failed to garbage collect required amount of images` will be reported to remind you that the disk space is insufficient. Common causes of insufficient disk space include:
  - The business has a lot of temporary output.
  - The business holds deleted file descriptors, so some space is not freed up.

Learn More

- Container Storage Interface Snapshot Support in Velero
- Enable API Group Versions Feature
- Install MinIO via Application Market

©2013-2022 Tencent Cloud. All rights reserved.

Serverless Cluster
Accessing Internet through NAT Gateway

Last updated：2023-05-04 10:44:34

Overview

TKE Serverless allows services in a cluster to access the internet by configuring the NAT Gateway and route table. This document guides you through the configuration.

Directions

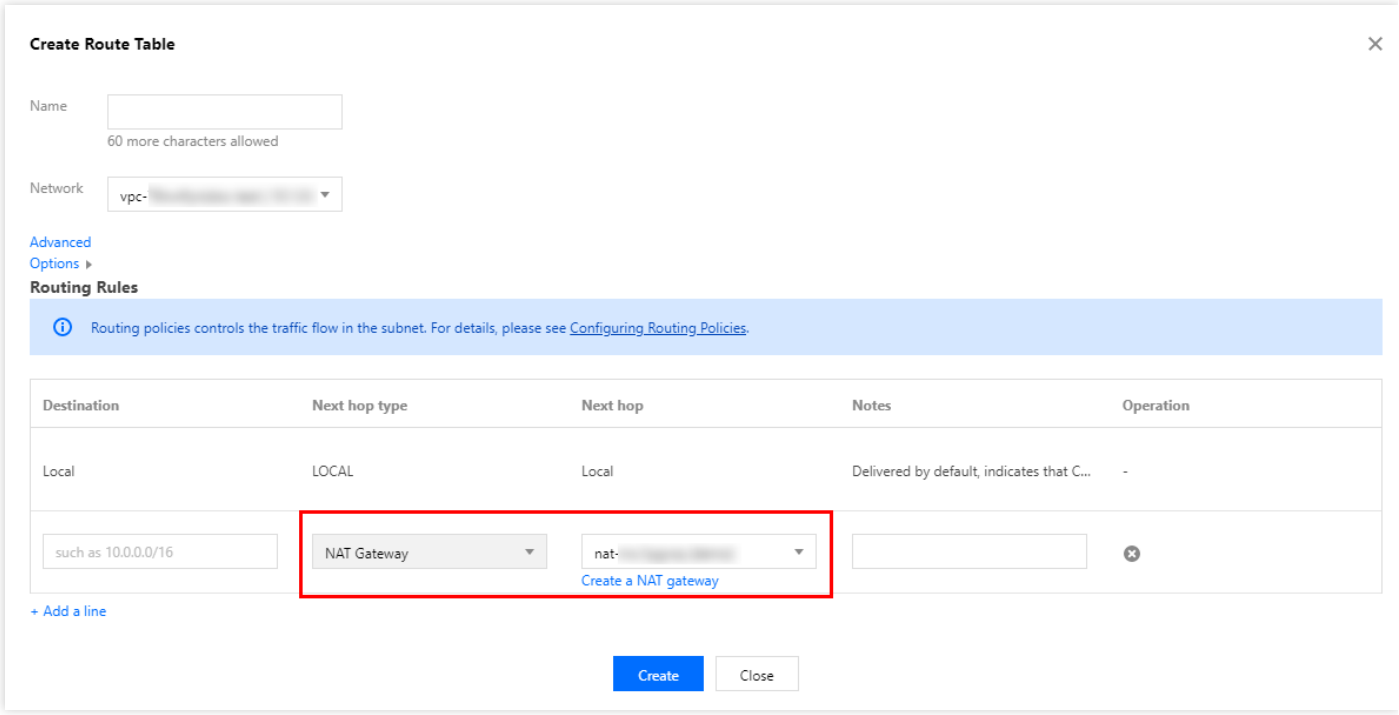
Creating a NAT gateway

1. Log in to the Tencent Cloud VPC console and click **NAT Gateway** in the left sidebar.
2. On the **NAT gateway** page, click **+Create**.
3. In the pop-up **Create NAT gateway** window, create a NAT gateway in the same region and same VPC as the TKE Serverless cluster. For more information, see Getting Started.

Creating a route table pointing to the NAT gateway

1. On the left sidebar, click **Route Table** to go to the **Route table** management page.
2. On the **Route table** management page, click **+Create**.
3. In the pop-up **Create route table** window, create a route table in the same region and same VPC as the TKE Serverless cluster.

©2013-2022 Tencent Cloud. All rights reserved.



The main parameters are described as follows:

- **Destination**: Select the public IP address to be accessed. You can configure a CIDR block for this parameter. For example, if you enter `0.0.0.0/0`, all traffic will be forwarded to the NAT gateway.
- **Next hop type**: Select **NAT gateway**.
- **Next hop**: Select the NAT gateway created in Creating a NAT gateway.

4. Click **Create**.

Associating subnets with the route table

After configuring routes, you need to select subnets and associate them with the route table. Then, traffic from the selected subnets to internet will be routed to the NAT gateway.

1. On the **Route table** page, find the route table created in the Creating a route table for the NAT gateway step and click **Associate subnets** on the right.
2. In the pop-up **Associate subnets** window, select the subnets to be associated and click **OK**.

> Note
> This subnet is not a Service CIDR block but a container network.

©2013-2022 Tencent Cloud. All rights reserved.

After associating the route table with the subnets, resources in the same VPC can access internet through the public IP address of the NAT gateway.

Configuration Verification

1. On the cluster management page, click the ID of the target Serverless cluster to enter the cluster details page.
2. Click **Remote login** for the target container and run a ping command to check whether its Pods can access the internet. If the results in the figure below are returned, it means the Pods have successfully accessed the internet.

```
bash-4.2$ ping qq.com
PING qq.com (203.205.254.157) 56(84) bytes of data.
64 bytes from 203.205.254.157 (203.205.254.157): icmp_seq=1 ttl=45 time=318 ms
64 bytes from 203.205.254.157 (203.205.254.157): icmp_seq=4 ttl=45 time=314 ms
64 bytes from 203.205.254.157 (203.205.254.157): icmp_seq=5 ttl=45 time=311 ms
64 bytes from 203.205.254.157 (203.205.254.157): icmp_seq=6 ttl=45 time=315 ms
```

Notes

The NAT gateway does not automatically adjust the bandwidth of the bound EIP anymore. When the problems (such as image pulling timeout) occur, and the upper limit for bandwidth of the NAT gateway is not reached, you can check the EIP bandwidth and set the upper limit as needed.

©2013-2022 Tencent Cloud. All rights reserved.

Using EIP to Access Public Network

Last updated：2023-03-14 18:19:11

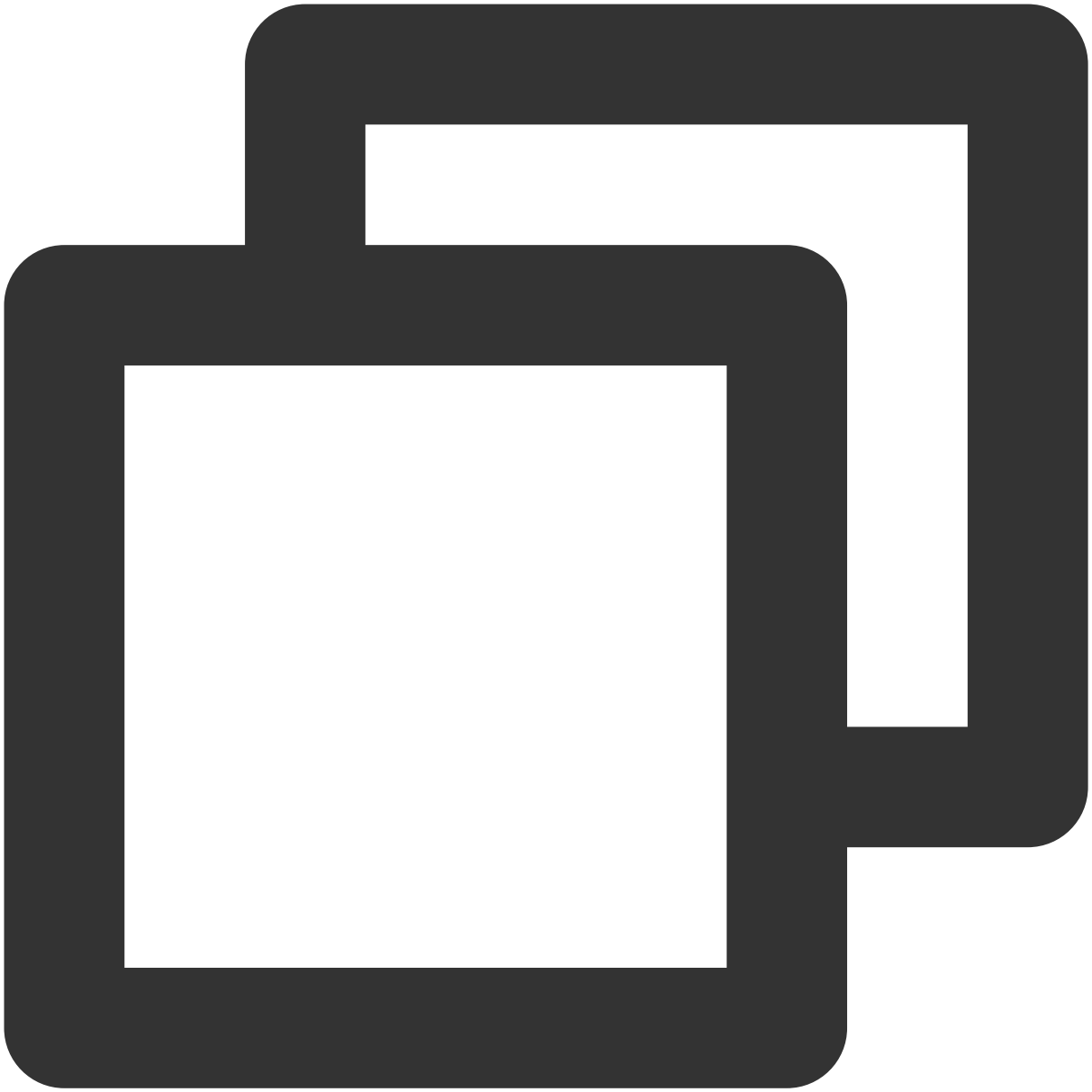
Currently, TKE Serverless allows you to bind an EIP to a Pod simply by declaring it in the template annotations. For more information, see Annotation.

Annotations related to EIP are described as follows:

Annotation Key	Annotation Value and Description	Required
eks.tke.cloud.tencent.com/eip-attributes	It indicates that the workload's Pod needs to be bound to an EIP. If the value is "", the binding will be created with the default EIP configuration. You can enter the EIP's TencentCloud API parameter JSON string in "" to customize the configuration.	Yes if you want to bind an EIP
eks.tke.cloud.tencent.com/eip-claim-delete-policy	It indicates whether to repossess the EIP after the Pod is deleted. `Never` indicates not to repossess. The default value is to repossess.	No
eks.tke.cloud.tencent.com/eip-id-list	It indicates that an existing EIP will be used, and only StatefulSets are supported. After the Pod is terminated, its EIP will not be repossessed by default. Note that the number of StatefulSet Pods cannot exceed the number of `eipId` values specified in this annotation.	No

1. If you want to bind an EIP to a workload or Pod for public network access, the simplest way is to add the `eks.tke.cloud.tencent.com/eip-attributes: ""` flag under the `annotation` of the corresponding workload or Pod as follows:

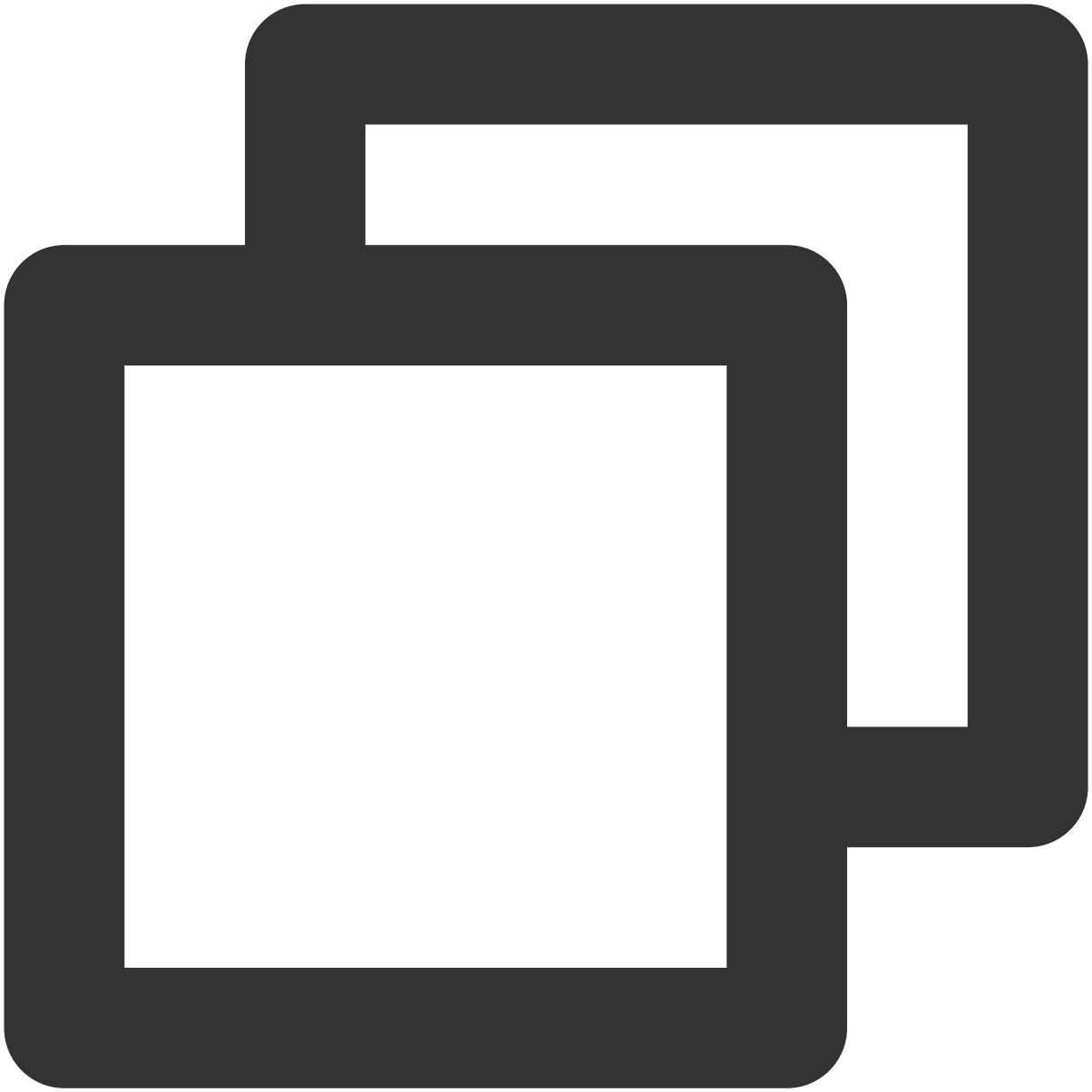
©2013-2022 Tencent Cloud. All rights reserved.



```
metadata:
  name: tf-cnn
  annotations:
    eks.tke.cloud.tencent.com/eip-attributes: ""  # EIP is required, and all are th
```

2. Run the following command to view the relevant events:

©2013-2022 Tencent Cloud. All rights reserved.

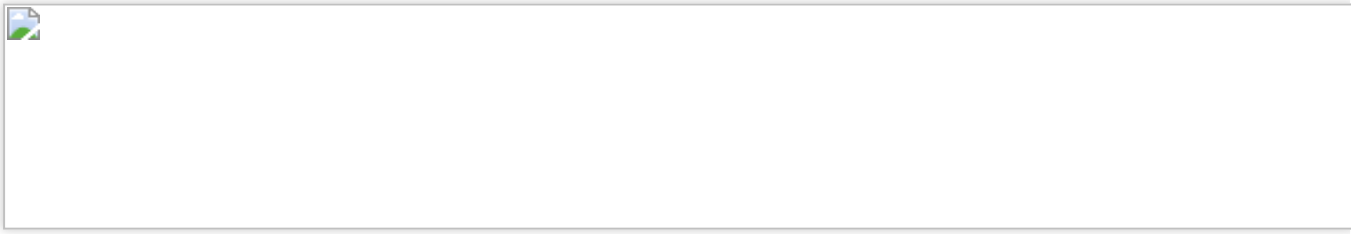


```
kubectl describe pod [name]
```

You can see that there are two new events related to the EIP as shown below, which indicate a success.

©2013-2022 Tencent Cloud. All rights reserved.

3. View the log file, and you can see that the datasets can be downloaded normally as shown below:



Note

The daily number of EIPs that can be applied for is limited, so EIP is not suitable for tasks that need to run multiple times every day.

©2013-2022 Tencent Cloud. All rights reserved.

Mastering Deep Learning in Serverless Cluster
Building Deep Learning Container Image

Last updated：2023-05-06 17:36:46

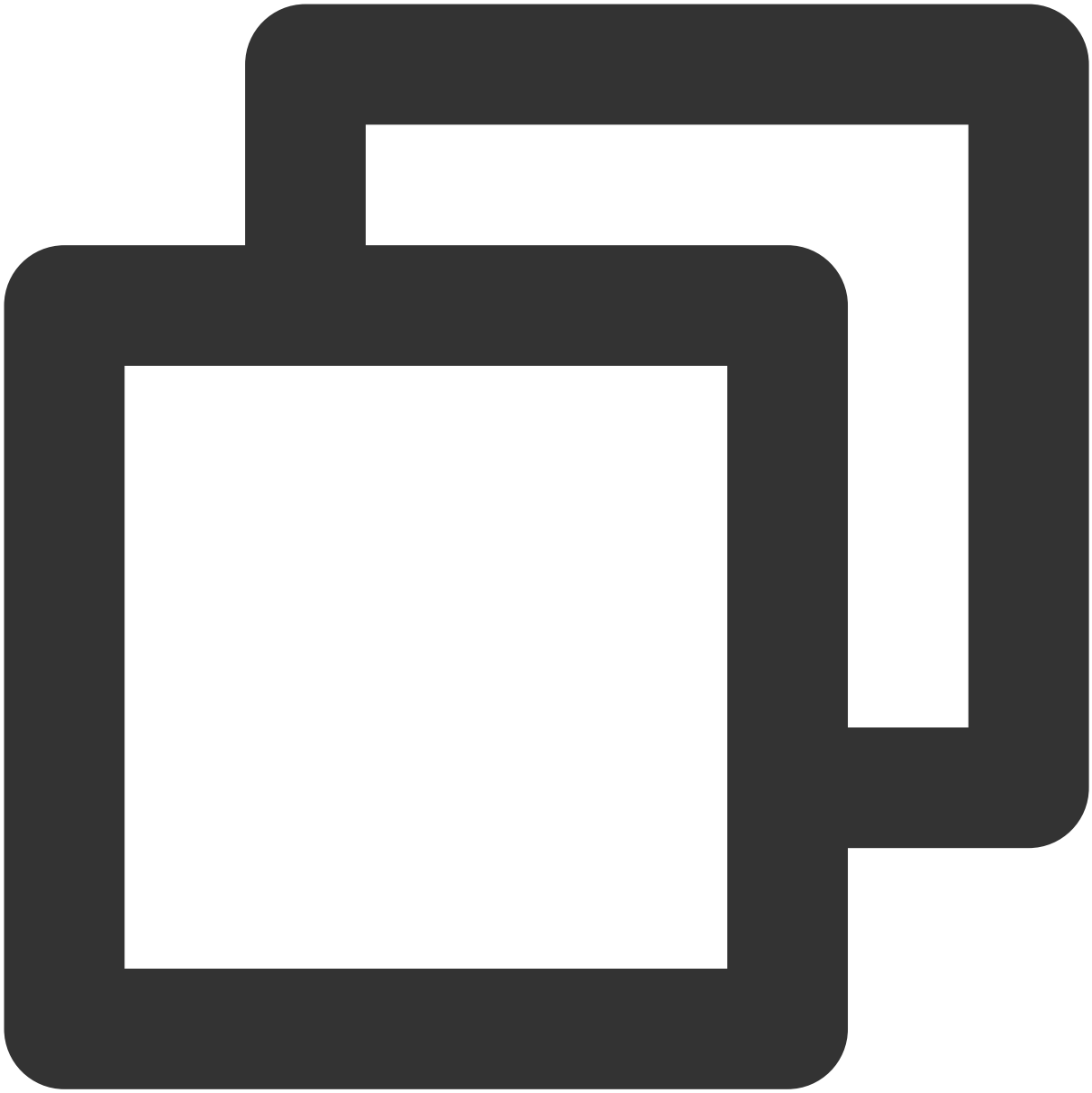
Overview

This series of documents describe how to deploy deep learning in TKE serverless clusters from direct TensorFlow deployment to subsequent Kubeflow deployment and are intended to provide a comprehensive scheme for implementing container-based deep learning. This document focuses on how to create a deep learning container image, which offers an easier and quicker method to deploy deep learning.
Public images cannot meet the requirements for deep learning deployment in this document. Therefore, a self-built image is used.
In addition to the deep learning framework TensorFlow-gpu, this image contains Compute Unified Device Architecture (CUDA) and CUDA Deep Neural Network library (cuDNN), which are required by GPU-based training. This image also integrates official TensorFlow deep learning models, including SOTA models for fields such as computer vision (CV), natural language processing (NLP), and recommender system (RS). For more information on the models, see TensorFlow Model Garden.

Directions

1. This example uses a Docker container to create an image. Prepare a Dockerfile as follows:

©2013-2022 Tencent Cloud. All rights reserved.

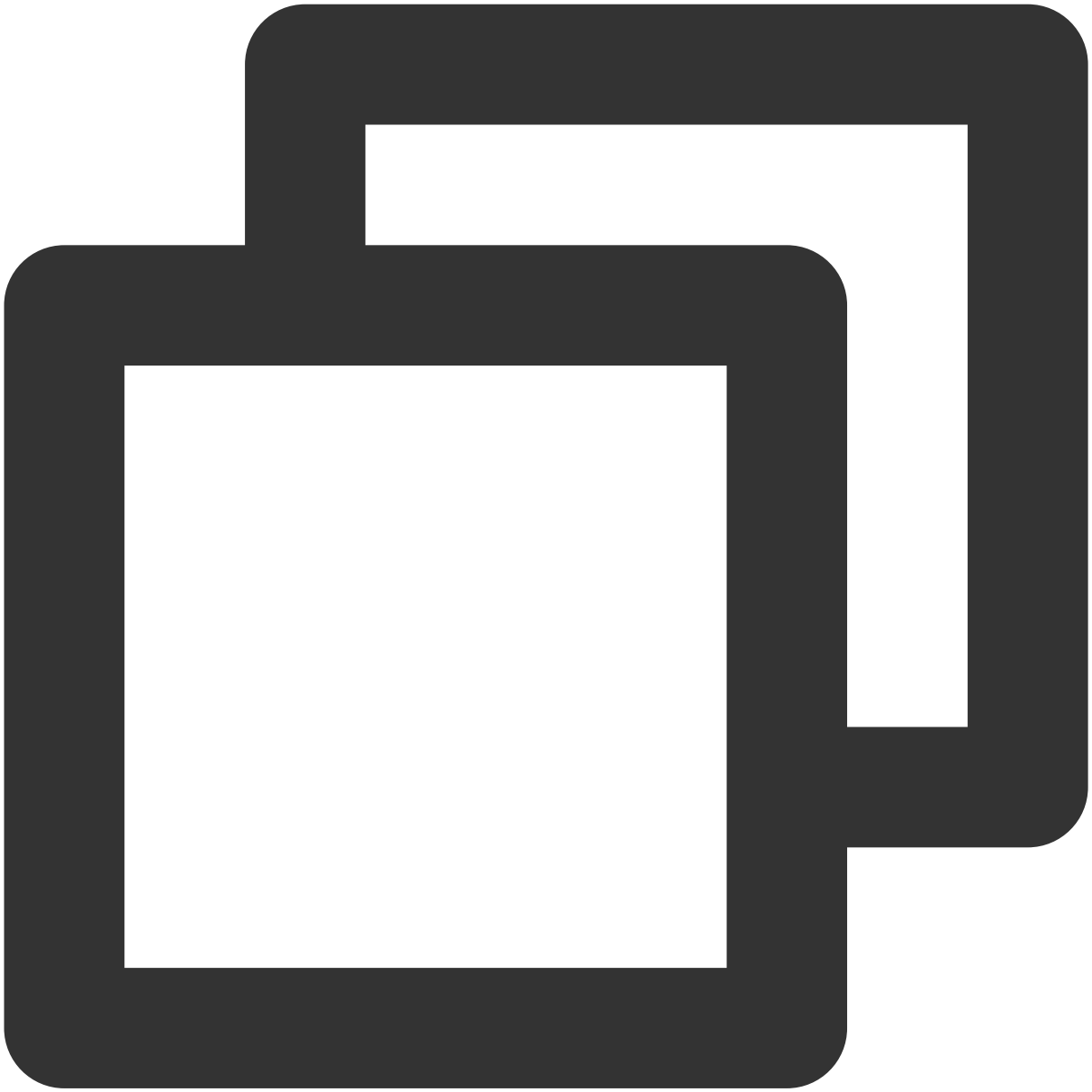


```
FROM nvidia/cuda:11.3.1-cudnn8-runtime-ubuntu20.04
RUN apt-get update -y \\
 && apt-get install -y python3 \\
    python3-pip \\
    git \\
 && git clone git://github.com/tensorflow/models.git \\
 && apt-get --purge remove -y git \\   # Promptly uninstall unneeded components (op
 && rm -rf /var/lib/apt/lists/*       # Delete the package for installation through
 && mkdir /tf /tf/models /tf/data     # Create storage models and data paths, which
ENV PYTHONPATH $PYTHONPATH:/models
ENV LD_LIBRARY_PATH $LD_LIBRARY_PATH:/usr/local/cuda-11.3/lib64:/usr/lib/x86_64-lin
```

©2013-2022 Tencent Cloud. All rights reserved.

```
RUN pip3 install --user -r models/official/requirements.txt \\
 && pip3 install tensorflow
```

2. Run the following command for deployment:



```
docker build -t [name]:[tag] .
```

Note

The steps to install required components such as Python, TensorFlow, CUDA, cuDNN, and model library are not detailed in this document.

©2013-2022 Tencent Cloud. All rights reserved.

Note

Image issues

For the base image nvidia/cuda, the CUDA container image provides an easy-to-use distribution for CUDA-supported platforms and architectures. Here, CUDA 11.3.1 and cuDNN 8 are selected. For more supported tags, see Supported tags.

Environment Variables

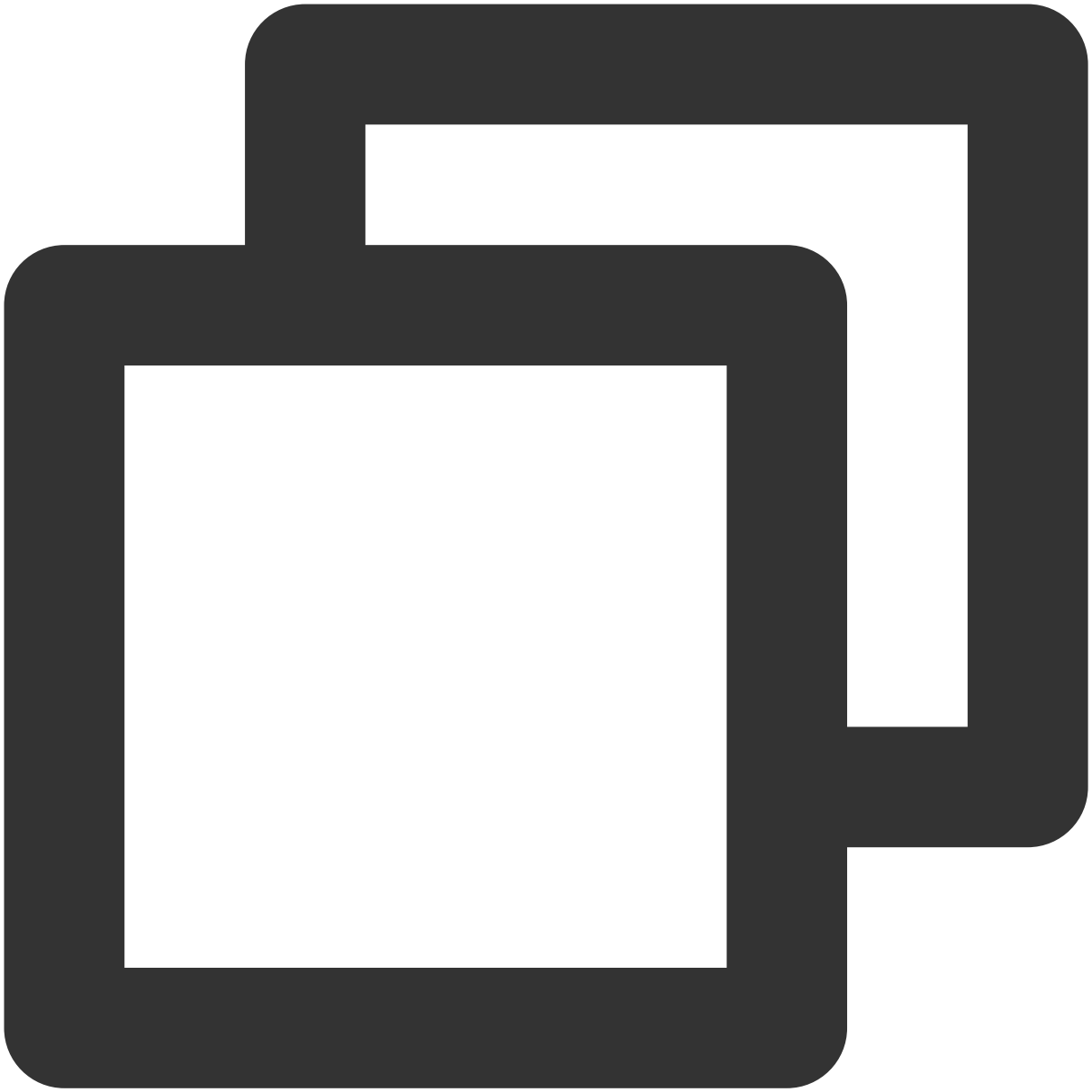
Before implement the best practice in this document, you need to pay special attention to the `LD_LIBRARY_PATH` environment variable.

`LD_LIBRARY_PATH` lists the installation paths of dynamic link libraries usually in the format of `libxxxx.so` , such as `libcudart.so.[version]` , `ibcusolver.so.[version]` , and `libcudnn.so.[version]` , and is used to link CUDA and cuDNN in this example. You can run the `ll` command to view the paths as shown below:

```
root@5a949761c669:/usr/local/cuda-11.3/lib64# ll
total 1534336
drwxr-xr-x 1 root root      4096 Jul  2 03:57 ./
drwxr-xr-x 1 root root      4096 Jul  2 03:57 ../
lrwxrwxrwx 1 root root        16 May  4 02:30 libOpenCL.so.1 -> libOpenCL.so.1.0
lrwxrwxrwx 1 root root        18 May  4 02:30 libOpenCL.so.1.0 -> libOpenCL.so.1.0.0
-rw-r--r-- 1 root root     30856 May  4 02:30 libOpenCL.so.1.0.0
lrwxrwxrwx 1 root root        23 May 13 23:26 libcublas.so.11 -> libcublas.so.11.5.1
-rw-r--r-- 1 root root 121866104 May 13 23:26 libcublas.so.11.5.1.109
lrwxrwxrwx 1 root root        25 May 13 23:26 libcublasLt.so.11 -> libcublasLt.so.11
-rw-r--r-- 1 root root 263770264 May 13 23:26 libcublasLt.so.11.5.1.109
lrwxrwxrwx 1 root root        21 May  4 02:30 libcudart.so.11.0 -> libcudart.so.11.3
-rw-r--r-- 1 root root    619192 May  4 02:30 libcudart.so.11.3.109
lrwxrwxrwx 1 root root        22 May 13 23:30 libcufft.so.10 -> libcufft.so.10.4.2.1
-rw-r--r-- 1 root root 190417864 May 13 23:30 libcufft.so.10.4.2.109
lrwxrwxrwx 1 root root        23 May 13 23:30 libcufftw.so.10 -> libcufftw.so.10.4.2
-rw-r--r-- 1 root root    631888 May 13 23:30 libcufftw.so.10.4.2.109
```

Run the following command based on the Dockerfile source code of the official image:

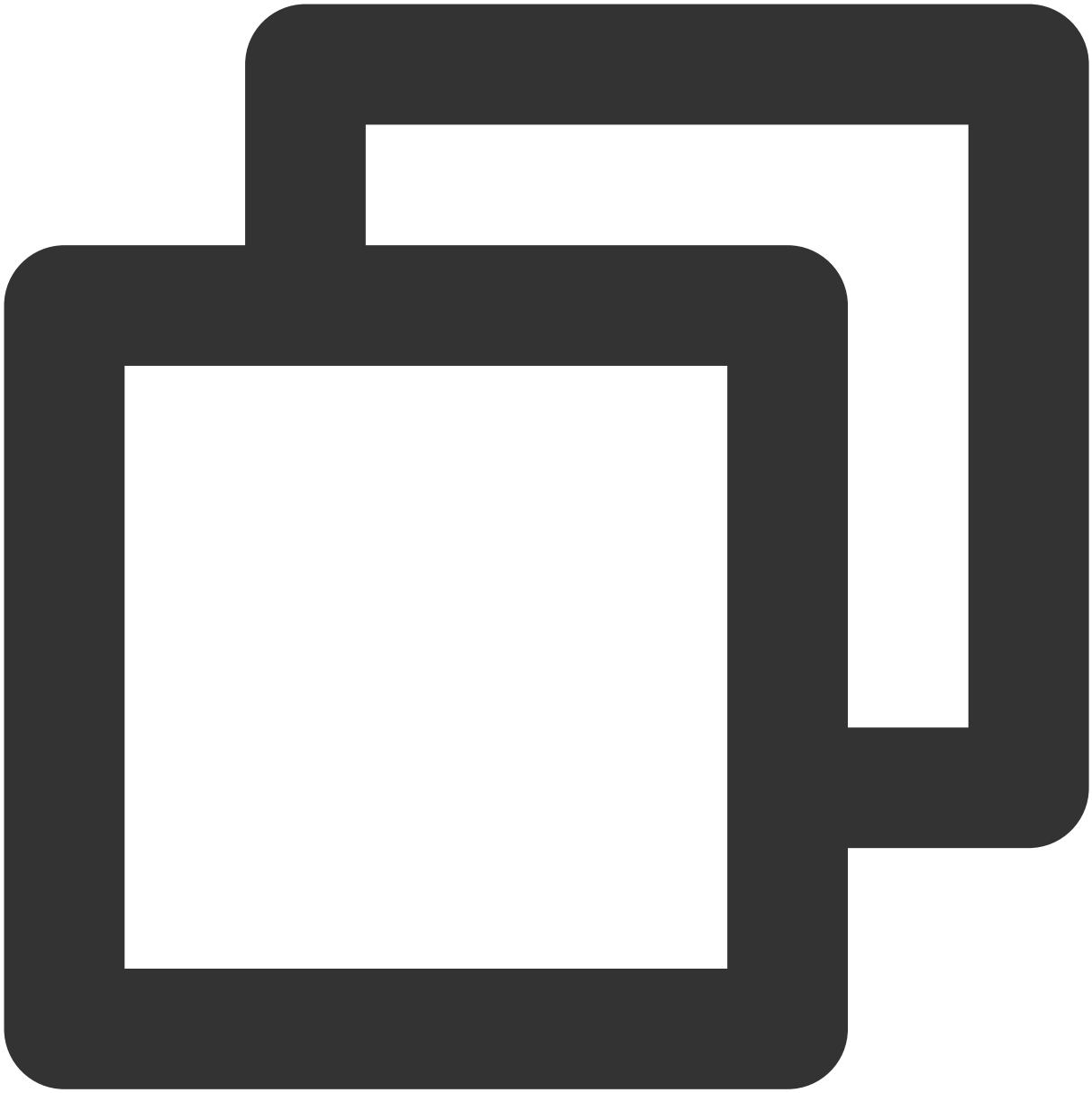
©2013-2022 Tencent Cloud. All rights reserved.



```
ENV LD_LIBRARY_PATH /usr/local/nvidia/lib:/usr/local/nvidia/lib64
```

Here, `/usr/local/nvidia/lib` points to the soft link of the CUDA path and is prepared for CUDA. However, in the tag with cuDNN, only cuDNN is installed, and `LD_LIBRARY_PATH` is not specified for cuDNN, which may report a warning and make GPU resources unavailable. The error is as shown below:

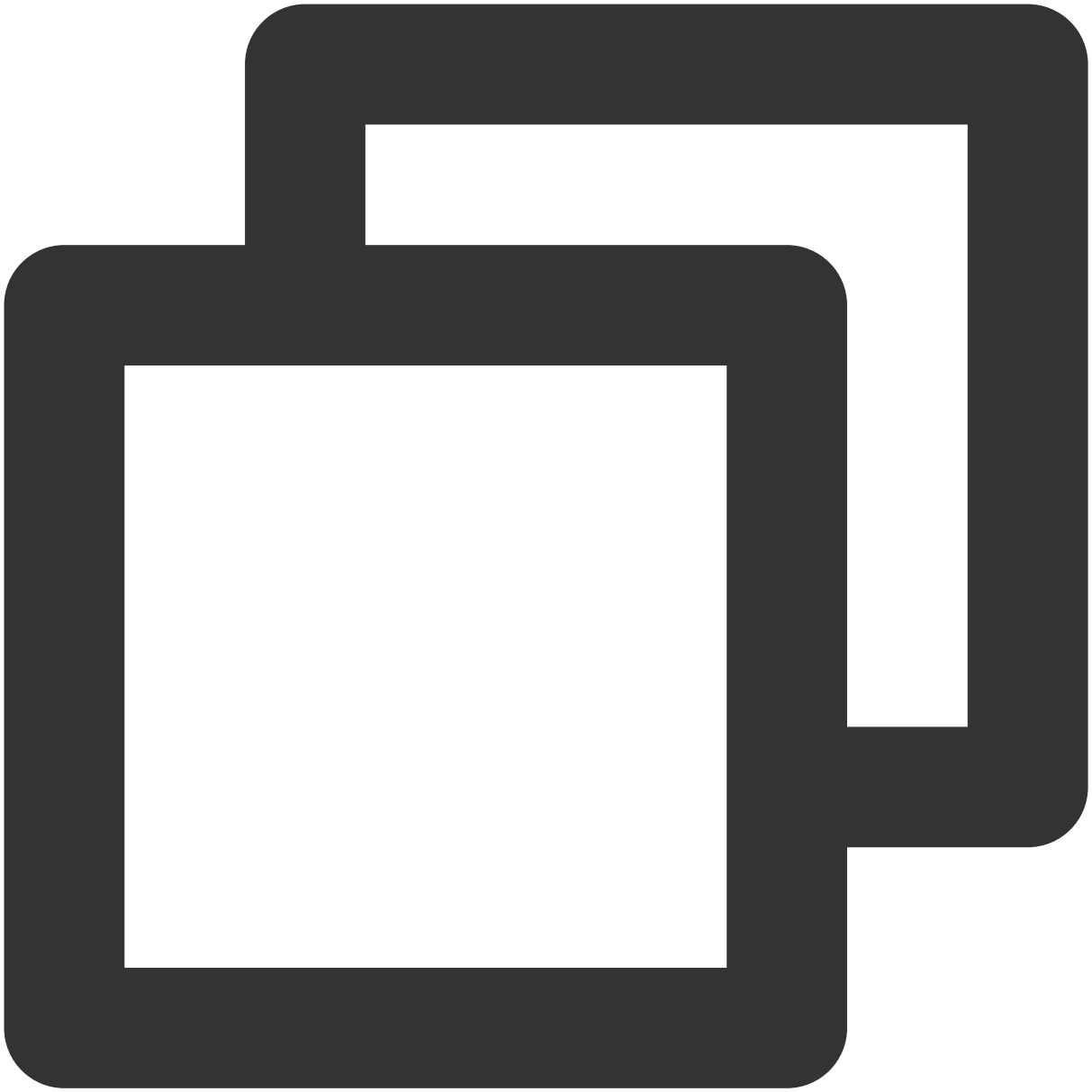
©2013-2022 Tencent Cloud. All rights reserved.



```
Could not load dynamic library 'libcudnn.so.8'; dlerror: libcudnn.so.8: cannot open
Cannot dlopen some GPU libraries. Please make sure the missing libraries mentioned
```

If such an error is reported, you can manually add the cuDNN path. Here, you can run the following command to run the image and view the path of `libcudnn.so` :

©2013-2022 Tencent Cloud. All rights reserved.



```
docker run -it nvidia/cuda:[tag] /bin/bash
```

As shown in the source code, cuDNN is installed under `/usr/lib` by default with the `apt-get install` command. In this example, the actual path of `libcudnn.so.8` is under `/usr/lib/x86_64-linux-gnu#` , and you need to add the path to the end after the colon.

The actual path may vary by tag and system. The path in the source code and what you actually see shall prevail.

Related Operations

©2013-2022 Tencent Cloud. All rights reserved.

For information about the related operations, see Running Deep Learning in TKE Serverless Cluster.

FAQs

If you encounter any problems when performing this practice, see FAQs for troubleshooting.

©2013-2022 Tencent Cloud. All rights reserved.

Running Deep Learning in TKE Serverless

Last updated：2023-05-22 15:48:34

Overview

This series of documents describe how to deploy deep learning in TKE Serverless from direct TensorFlow deployment to subsequent Kubeflow deployment and are intended to provide a comprehensive scheme for implementing container-based deep learning.

Prerequisites

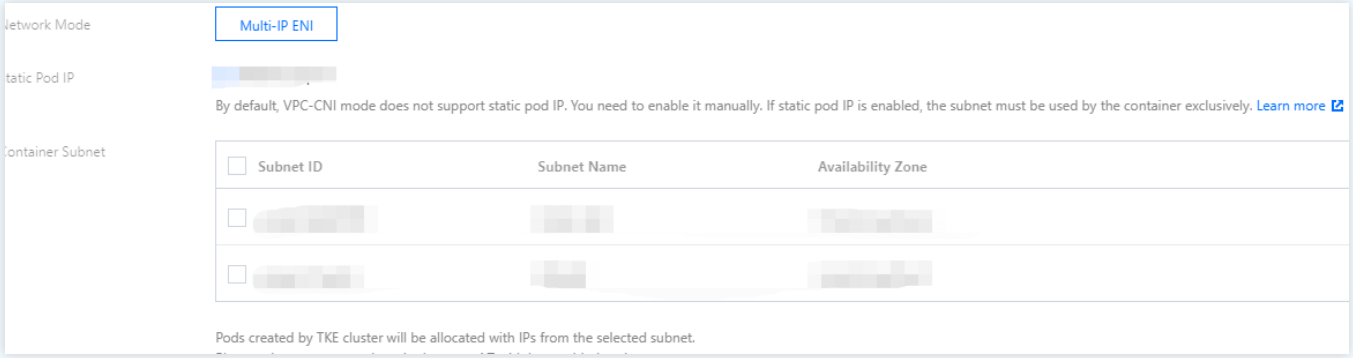
This document proceeds to run a deep learning task in TKE Serverless by using a self-built cluster after the steps in Building Deep Learning Container Image are completed.The self-built image has been uploaded to the image repository `ccr.ccs.tencentyun.com/carltk/tensorflow-model` , which can be directly pulled for use with no rebuild required.

Directions

Creating TKE Serverless cluster

Please create an TKE Serverless cluster as instructed in Connecting to a Cluster.

Note：
As you need to run a GPU-based training task, when creating a cluster, please pay attention to the supported resources in the AZ of the selected container network and be sure to select an AZ that supports GPU as shown below:



©2013-2022 Tencent Cloud. All rights reserved.

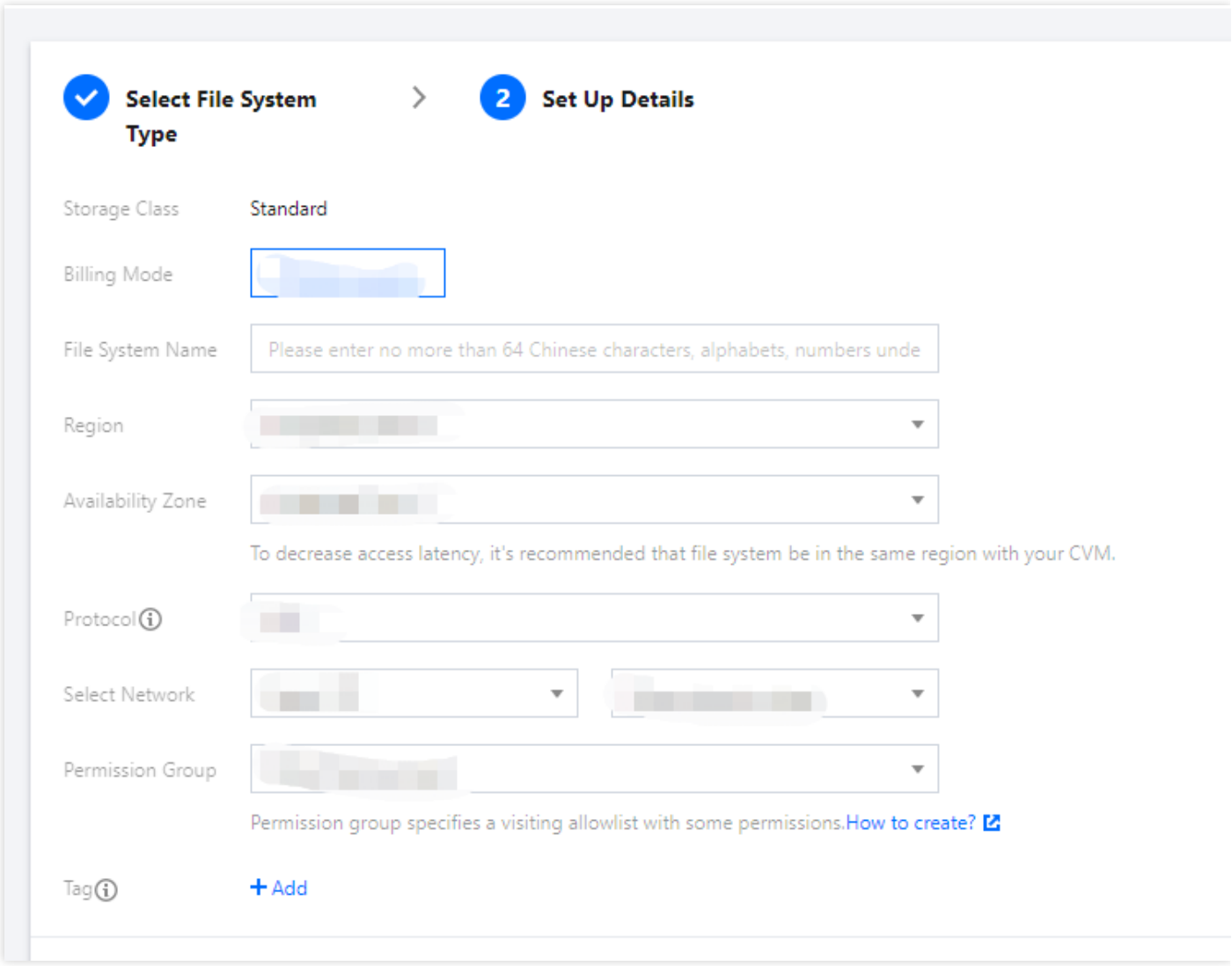
Creating CFS file system (optional)

The container will be automatically deleted, and the resources will be automatically released after the task ends. Therefore, to persistently store models and data, we recommend you mount an external storage service such as CBS, CFS, and COS.

In this example, CFS is used as an NFS disk to persistently store data with frequent reads and writes.

Creating CFS file system

1. Log in to the CFS console and enter the **File System** page.
2. Click **Create**. On the **Create File System** page that pops up, select the file system type and click **Next: Detailed Settings**.
3. On the **Detailed Settings** page, set the relevant configuration items. For more information on CFS types and configurations, please see Creating File Systems and Mount Targets.



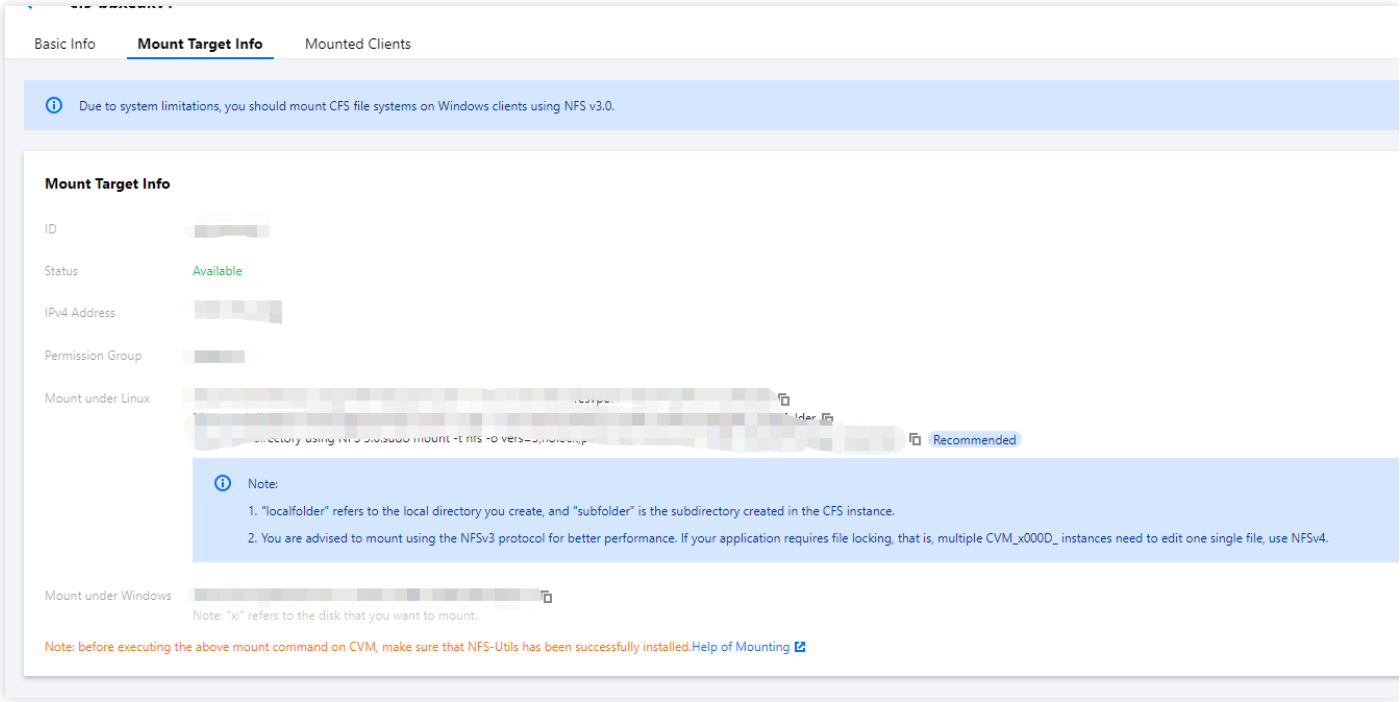
©2013-2022 Tencent Cloud. All rights reserved.

> Note：
> The CFS file system must be created in the region of the cluster.

4. After confirming that everything is correct, click **Buy Now** and make the payment to create a file system.

Getting file system mount information

1. On the **File System** page, click the ID of the file system whose sub-target path needs to be obtained to enter the file system details page.
2. Select the **Mount Target Info** tab and get the file system mount information next to **Mount to Linux** as shown below:



> Note：
> Note down the IPv4 address in the mount target details, such as `10.0.0.161:/`, which will be used as the NFS path in subsequent mount configuration.

Creating training task

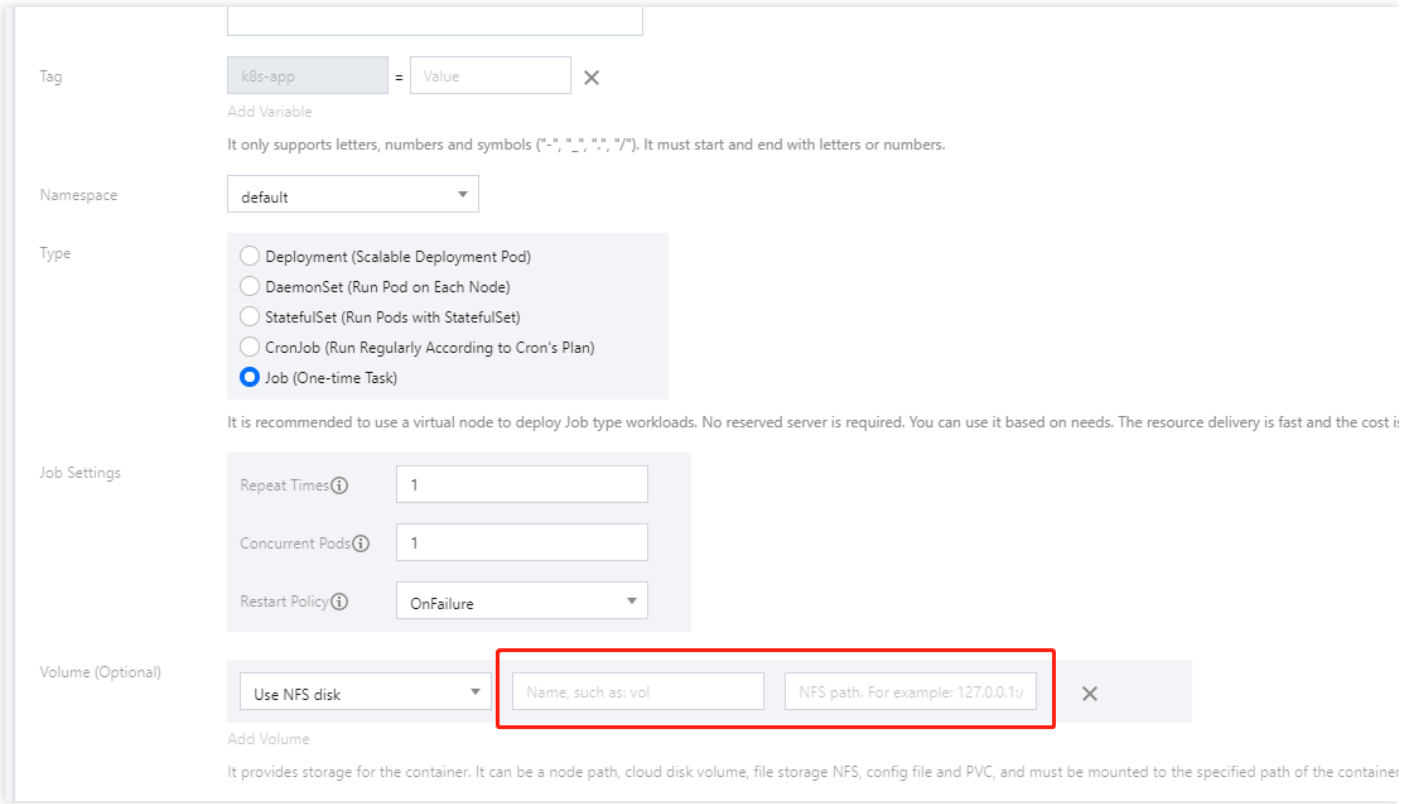
This task uses the MNIST handwritten digit recognition dataset and two-layer CNN as an example. The sample image is the self-built image created in the previous chapter. If you need to use a custom image, please see Creating Deep Learning Container Image. Two task creation methods are provided below:

©2013-2022 Tencent Cloud. All rights reserved.

- Console
- kubectl

Taking the essence of the deep learning task into account, Job node deployment is used as an example in this document. For more information on how to deploy a Job, please see Job Management.The following is the example of deployment in the console:

1. In the **Volume (optional)** configuration item, select **Using NFS disk** and enter the name and IPv4 address of the CFS file system created previously as shown below:



2. In the **Mount Target** configuration item in **Containers in the Pod**, select the volume and configure the mount target as shown below:

Note

- As the dataset may need to be downloaded online, you need to configure the public network access for the cluster. For more information, please see Public Network Access.
- After selecting a GPU model, when setting the request and limit, you need to assign the container CPU and memory resources meeting the resource specifications. The actual values do not need to be accurate down to the ones place.When configuring in the console, you can also delete the default configuration and leave it empty to configure "unlimited" resources, which also have the corresponding billing specifications. This approach is recommended.

©2013-2022 Tencent Cloud. All rights reserved.

- The container running command is inherited from Docker's `CMD` field, whose preferred form is `exec`. If you do not call the `shell` command, there will be no normal shell processing. Therefore, if you want to run a command in the `shell` form, you need to add `"sh"` and `"-c"` at the beginning.When you enter multiple commands and parameters in the console, each command should take a line (subject to the line break)

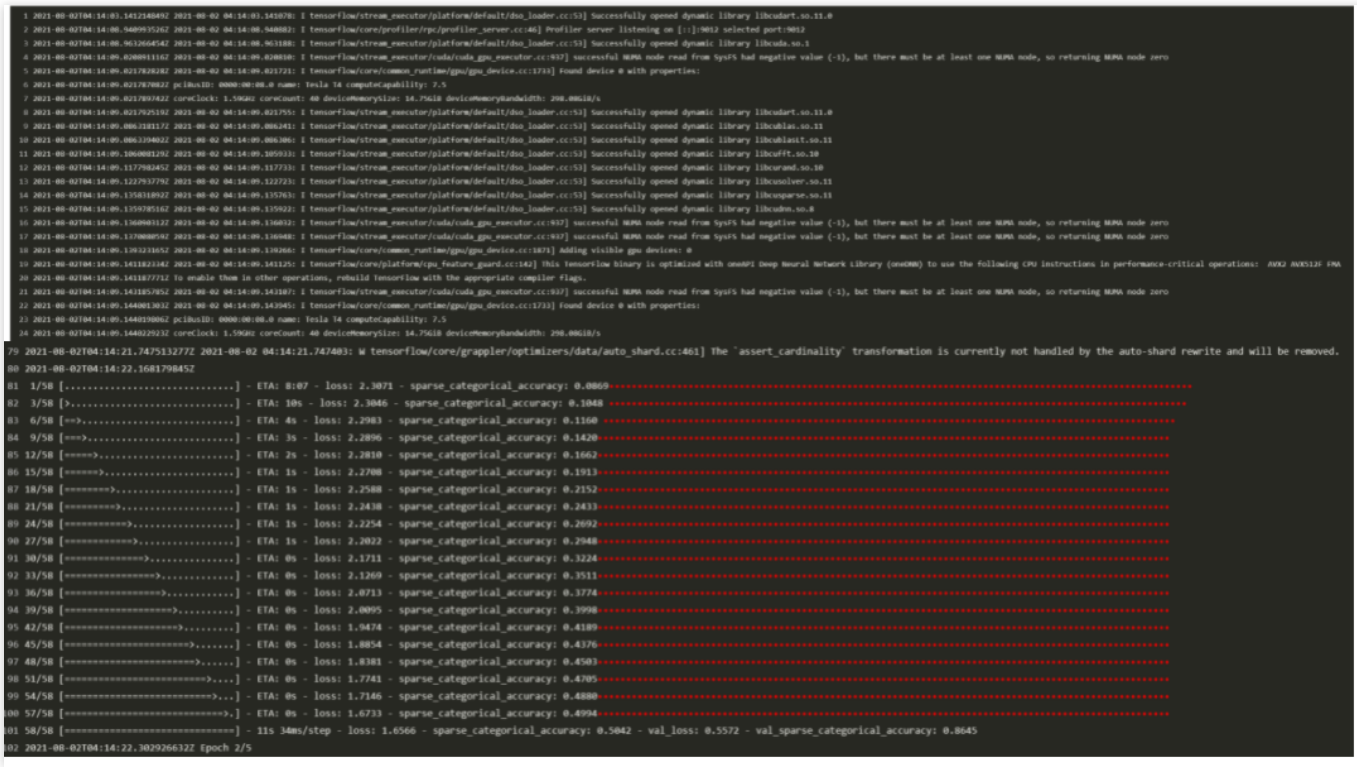
Viewing running result

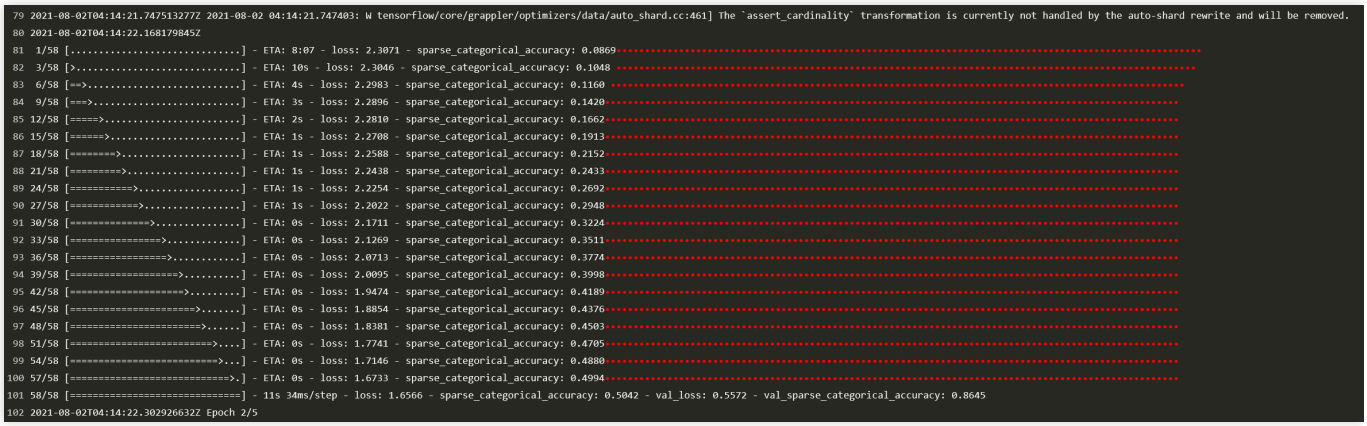
You can view the running result either in the console or on the command line:

- Console
- Command line

After creating a Job, you will be redirected to the Job management page by default. You can also enter the page as follows:

1. Log in to the TKE console and click **Cluster** on the left sidebar.
2. In the elastic cluster list, click the ID of the cluster whose events you want to view to enter the cluster management page.
3. Select **Workload** > **Job** and click the newly created Job in the Job list.
   - Select the **Log** tab to view logs as shown below:





Relevant Operations

Using GPU to deploy deep learning task in TKE

Deployment in TKE is almost the same as that in TKE Serverless. Taking deployment through kubectl with a YAML file as an example, TKE has the following differences:

- When creating a TKE node, you should select a node with GPU. For more information, please see Using a GPU Node.
- As the node has built-in GPU resources, `annotations` and `resources` are not needed. Practically, you can reserve `annotations`, which TKE will not process. We recommend you comment out `resources`, as it may cause unreasonable resource requirements.

FAQs

If you encounter any problems when performing this practice, please see FAQs for troubleshooting.

©2013-2022 Tencent Cloud. All rights reserved.

FAQs
Public Network Access

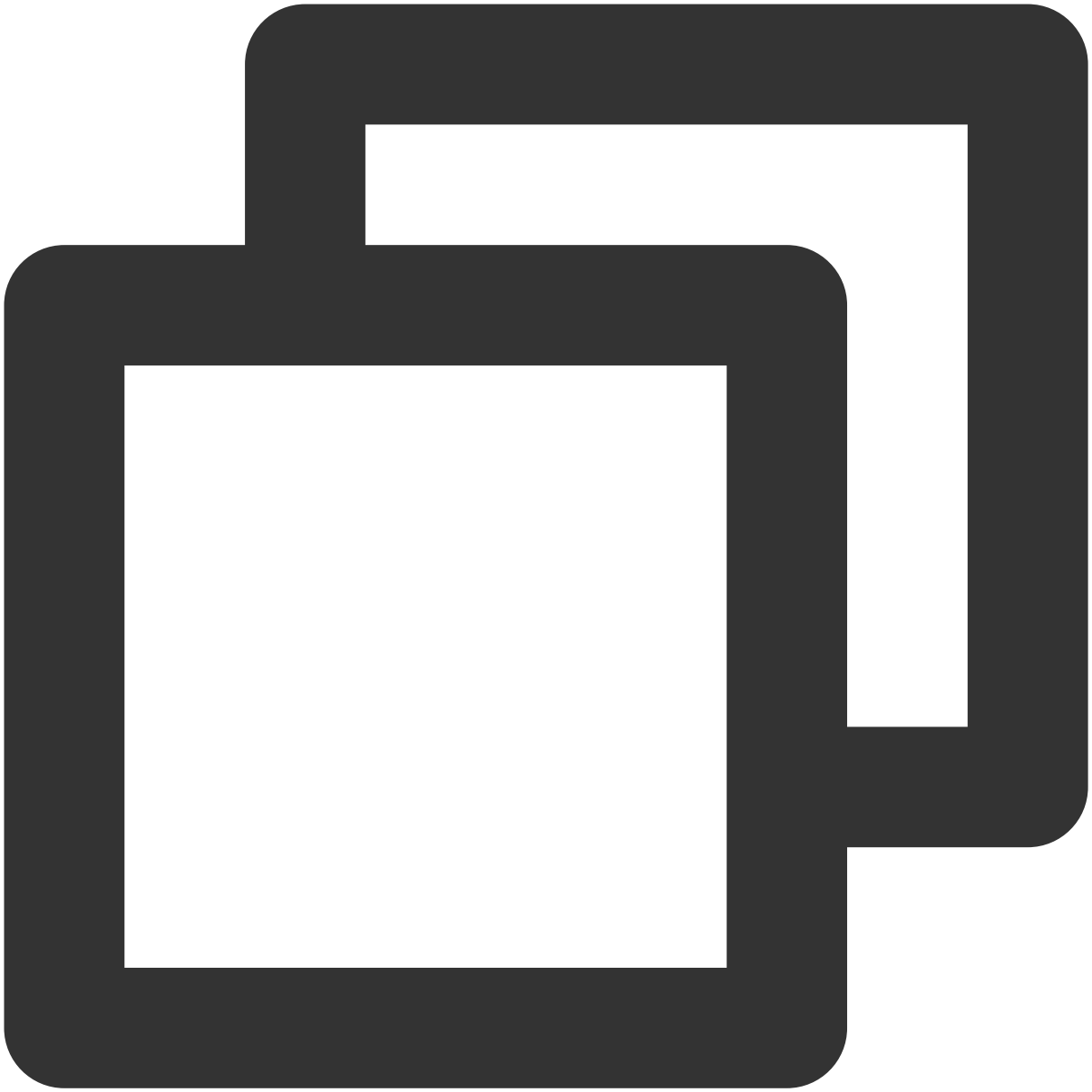
Last updated：2023-05-06 17:36:46

This document offers answers to some questions that you may have when building a deep learning container image and running deep learning in TKE Serverless Cluster.

How does a container access the public network?

As you may need to download training datasets during a task, access to the public network may be required. However, a container in its initial status cannot access the public network, and if you directly run a command with dataset download, the following error will be reported:

©2013-2022 Tencent Cloud. All rights reserved.



```
W tensorflow/core/platform/cloud/google_auth_provider.cc:184] All attempts to get a
E tensorflow/core/platform/cloud/curl_http_request.cc:614] The transmission  of req
```

For the above problem, two public network access methods are provided:

NAT Gateway: It is suitable for scenarios where multiple Pods in the same virtual private cloud (VPC) need to interconnect with the public network. For more information, see Accessing Internet through NAT Gateway.

Note

The created NAT gateway and route table need to be in the same region and VPC as the TKE serverless cluster.

©2013-2022 Tencent Cloud. All rights reserved.

Elastic IP (EIP): It is suitable for scenarios where one or a few Pods need to interconnect with the public network. For more information, see Using EIP to Access Public Network.

©2013-2022 Tencent Cloud. All rights reserved.

Log Collection

Last updated：2023-05-22 15:18:41

This document offers answers to questions that you may have when building a deep learning container image and running deep learning in TKE Serverless.

How do I persistently store logs?

As TKE Serverless containers will be terminated after use, you can view logs only when the Pod is in **Running** status. Once the Pod status becomes **Completed**, the following error will be reported:

```
Error from server (InternalError): Internal error occurred: can not found connect
ion to pod ***
```

The following describes persistent log storage methods:

- Redirect
- Log collection configuration

Redirect

The redirect method is simpler. You only need to change the terminal `stdout` to which `kubectl logs` are output to a file for persistent storage. To do so, run the following command:

```
kubectl logs -f tf-cnn >> info.log
```

However, when using the redirect method, you should note that the output stream will not flow to the terminal; that is, you cannot view the log output progress on the terminal. If you want to output the content to the screen while storing the command output to a file, you can do so in the following two methods:

- Use a pipe and the `tee` command. Run the following command:
  ```
  kubectl logs -f tf-cnn |tee info.log
  ```
- You can also run the `logsave` command to output the content to the screen while storing the command output to the file as follows:
  ```
  logsave [-asv] info.log kubectl logs -f tf-cnn
  ```

Note：

©2013-2022 Tencent Cloud. All rights reserved.

> The advantage of `logsave` over `tee` is that with `logsave`, the time will be recorded for each input, and there is a certain spacing between logs, which makes it easier for you to find logs.

The above three commands all have a shortcoming: as their redirect is based on the `kubectl logs` output, they must be used when the Pod is in **Running** status, and they are only used to view logs after the Pod is in **Completed** status.The redirect method is applicable to scenarios with only a small number of logs and with no requirements for outputting and searching for a high number of logs. If your requirements are not high, we recommend you use the redirect method.

Log collection configuration

In TKE Serverless, you can configure log collection either through environment variables or CRDs.

- Using environment variables to configure log collection
- Using CRD to configure log collection (recommended)

1. Configure log collection as instructed in Using Environment Variables to Configure Log Collection
2. If you want to use keys for authorization, you can create a `Secret` in `Opaque` type and create two keys (`SecretId` and `SecretKey`). The values of `SecretId` and `SecretKey` can be obtained in API Key.
3. You can find the created `Secret` after enabling log collection and associate `SecretId` with `SecretKey`
4. Get the raw logs in the console, switch to the table view, and format the JSON strings

This method has a problem: the log collection feature of TKE Serverless works by sending the collected logs as JSON strings to the specified consumer, but the timestamps of the collected JSON strings are at the second level

In this case, logs are displayed in the console at the second level, and the logs displayed on the search and analysis page can be sorted only by second but cannot be output sequentially at a finer time granularity. However, sometimes a large number of logs are output in a short while, for which a millisecond granularity is often required. Therefore, we recommend the CRD-based configuration method.

©2013-2022 Tencent Cloud. All rights reserved.

Customized DNS Service of Serverless Cluster

Last updated：2022-09-26 17:12:57

Note：
The entry for DNS Forward configuration is no longer available. The parameters of DNS Forward configured previously will be synced and updated in the Corefile of CoreDNS. If you want to modify the DNS service of the cluster, please refer to the following instructions or the directions of native Kubernetes CoreDNS.

Overview

This document describes how to modify the DNS service of a cluster through modifying the CoreDNS configuration file.

Prerequisites

You have created an serverless cluster. You need to select **Deploy CoreDNS to allow the service discovery in the cluster** in the advanced configuration at the time of creation.

Directions

Default Corefile configuration

When a CoreDNS is deployed in an serverless cluster, a Configmap is mounted by default to act as the CoreDNS configuration file (i.e. Corefile).
The default configuration of Corefile is as follows:

```
apiVersion: v1
kind: ConfigMap
metadata:
name: coredns
namespace: kube-system
data:
Corefile: |
```

©2013-2022 Tencent Cloud. All rights reserved.

```
.:53 {
errors
health :8081
kubernetes cluster.local in-addr.arpa ip6.arpa {
pods insecure
fallthrough in-addr.arpa ip6.arpa
ttl 30
}
prometheus :9153
forward . 183.60.83.19 183.60.82.98
cache 30
loop
reload
loadbalance
}
```

Each configuration item adopts the configuration of native Kubernetes. For details, see CoreDNS. Please note:

- `forward` : 183.60.83.19, 183.60.82.98 is the default DNS address of Tencent Cloud.

Customize configuration of Corefile

You can modify ConfigMap of CoreDNS (i.e. Corefile) to modify relevant configuration of service discovery. The use method is consistent with that of the native kubernetes. For details, see Customizing DNS Service.

©2013-2022 Tencent Cloud. All rights reserved.

Edge Cluster
TKE Edge ServiceGroup Feature
Using ServiceGroup via YAML File

Last updated：2022-06-10 16:48:45

Overview

TKE Edge provides the ServiceGroup feature, which only needs two YAML files to implement service deployment in hundreds of regions, without application adaptation or transformation. This document describes how to deploy Nginx services separately within multiple node groups.

Directions

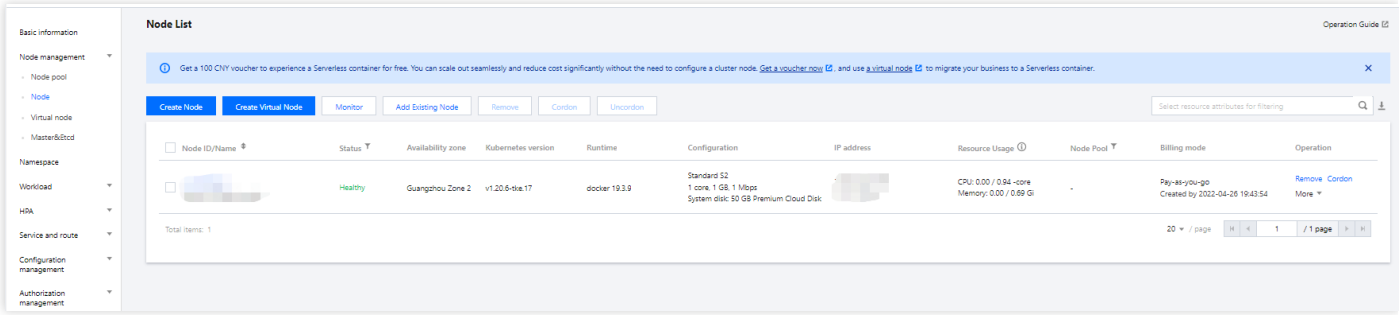
Determining the unique key of ServiceGroup

This step performs logic planning without involving any actual operations. TKE Edge sets the `UniqKey` used as the logical flag of the ServiceGroup to be created to `zone`.

Grouping edge nodes by label

Label the edge nodes in the TKE Edge console or by using kubectl in the TKE Edge console as instructed below:

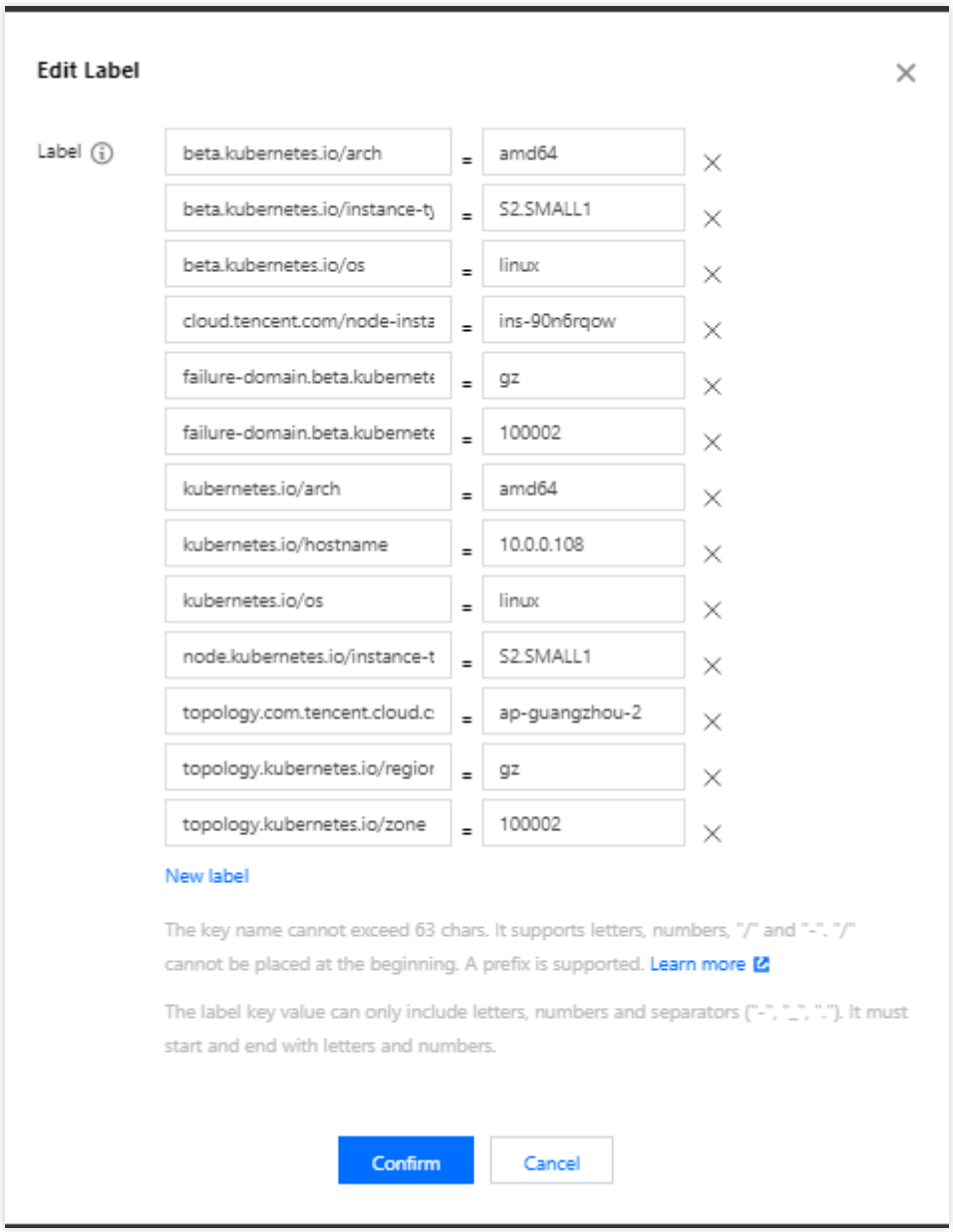
1. Log in to the TKE console and click **Edge Clusters** on the left sidebar.
2. Select the target cluster ID to enter the cluster management page.
3. Click **Node Management** > **Node** to enter the node list page as shown below:



4. Select **More** > **Edit Label** on the right of the target node.

©2013-2022 Tencent Cloud. All rights reserved.

5. In the **Edit Label** pop-up window, add a label as instructed below:



- See the overall architecture chapter. Select `zone=nodeunit1` for nodes 12 and 14 and `zone=nodeunit2` for nodes 21 and 23.
- The label `key` needs to be the same as the `UniqKey` of the ServiceGroup. The `value` is a unique key of the NodeUnit. Nodes with the same `value` belong to the same NodeUnit.
- If there are multiple ServiceGroups in the same cluster, assign different Uniqkeys to different ServiceGroups.

6. Click **OK**.

Deploying DeploymentGrid

```
apiVersion: superedge.io/v1
kind: DeploymentGrid
```

©2013-2022 Tencent Cloud. All rights reserved.

```
metadata:
name: deploymentgrid-demo
namespace: default
spec:
gridUniqKey: zone
template:
selector:
matchLabels:
appGrid: nginx
replicas: 2
template:
metadata:
labels:
appGrid: nginx
spec:
containers:
- name: nginx
image: nginx:1.7.9
ports:
- containerPort: 80
protocol: TCP
```

Deploying ServiceGrid

```
apiVersion: superedge.io/v1
kind: ServiceGrid
metadata:
name: servicegrid-demo
namespace: default
spec:
gridUniqKey: zone
template:
selector:
appGrid: nginx
ports:
- protocol: TCP
port: 80
targetPort: 80
```

Note：

As shown above, the `gridUniqKey` field is set to `zone`. Therefore, you should also set the label `key` to `zone` when grouping edge nodes by label. If there are three node groups, add three labels respectively: `zone: zone-0`, `zone: zone-1`, and `zone: zone-2`.

©2013-2022 Tencent Cloud. All rights reserved.

At this point, each node group contains the Deployment and corresponding Pod of Nginx. For access to the same `service-name` on a node, the requests will be sent to the node in the target group. The verification method is as follows:

```
[root@VM_1_34_centos ~]# kubectl get deploy
NAME READY UP-TO-DATE AVAILABLE AGE
deploymentgrid-demo-zone-0 2/2 2 2 85s
deploymentgrid-demo-zone-1 2/2 2 2 85s
deploymentgrid-demo-zone-2 2/2 2 2 85s
[root@VM_1_34_centos ~]# kubectl get svc
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
kubernetes ClusterIP 172.19.0.1 <none> 443/TCP 87m
servicegrid-demo-svc ClusterIP 172.19.0.177 <none> 80/TCP 80s
```

For node groups added to a cluster after the deployment of DeploymentGrid and ServiceGrid, this feature will automatically create the specified Deployment and Service in the new node groups.

©2013-2022 Tencent Cloud. All rights reserved.

TKE Edge Distributed Node Status Determination Mechanism

Last updated：2022-06-10 19:32:52

Poor edge network conditions will trigger the Kubernetes eviction mechanism to evict Pods that do not meet the expectations. In edge computing scenarios where the network environments of the edge nodes and the cloud are complex, and the network quality cannot be guaranteed, problems such as API server and node disconnection tend to occur. If native Kubernetes is used without modification, the node status will often become abnormal. This causes the Kubernetes eviction mechanism to take effect, where Pods are evicted and the Endpoint is lost, thereby causing service interruption and fluctuation.

To solve this problem, TKE Edge offers the innovative distributed node status determination mechanism. This mechanism can better identify the eviction timing, guarantee system running under poor network conditions, and avoid service interruption and fluctuation.

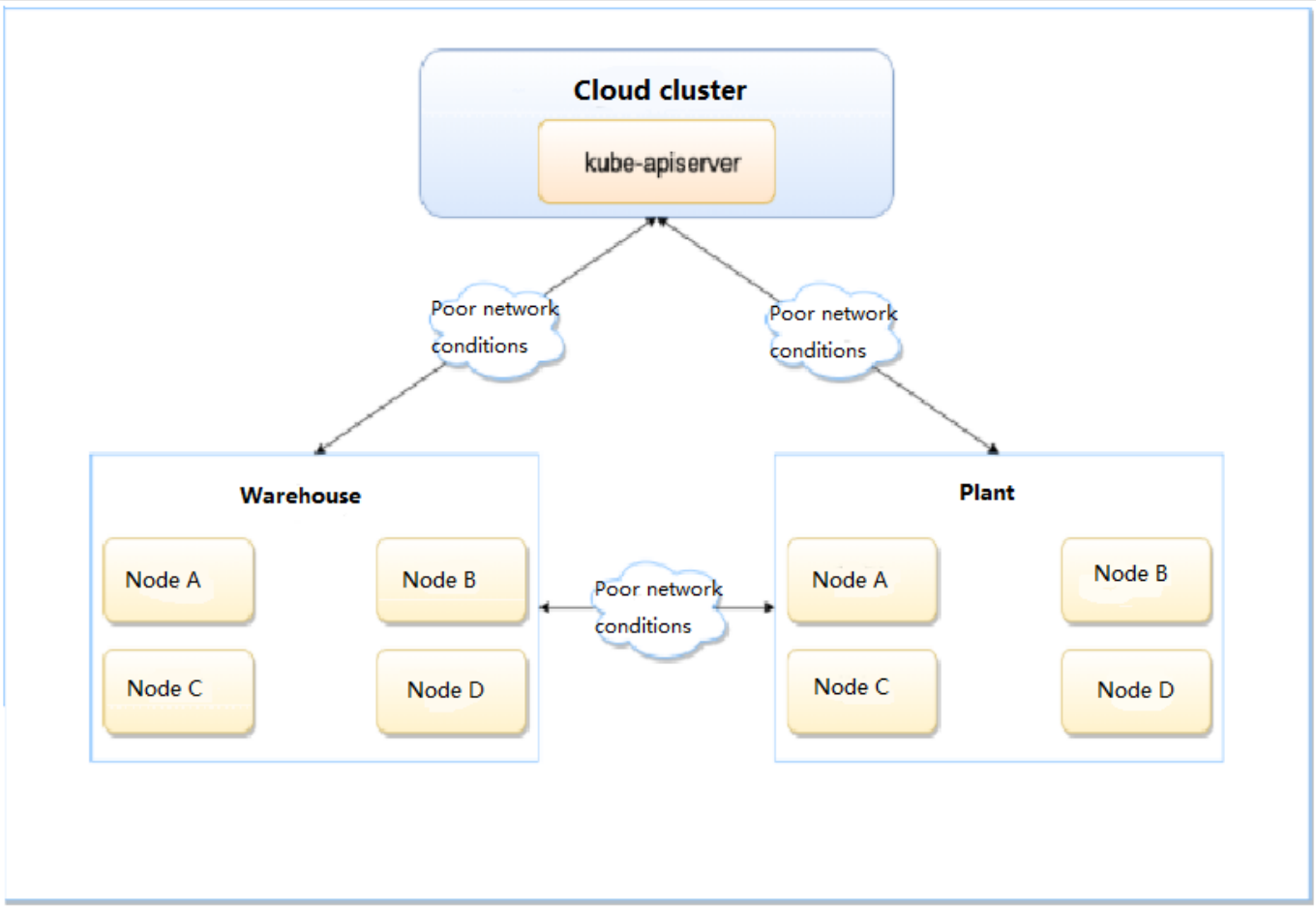
Use Cases

An edge use case is subject to poor network conditions in the cloud. Edge devices are located in edge cloud data centers and mobile edge sites, facing complex network environments for their cloud connectivity, such as unreliable environments at the cloud (console) and edge as well as between edge nodes.

Smart factory

In a smart factory, edge nodes are located in the warehouse and plant, and the master node in the console is in the central data center of Tencent Cloud, as shown below:

©2013-2022 Tencent Cloud. All rights reserved.

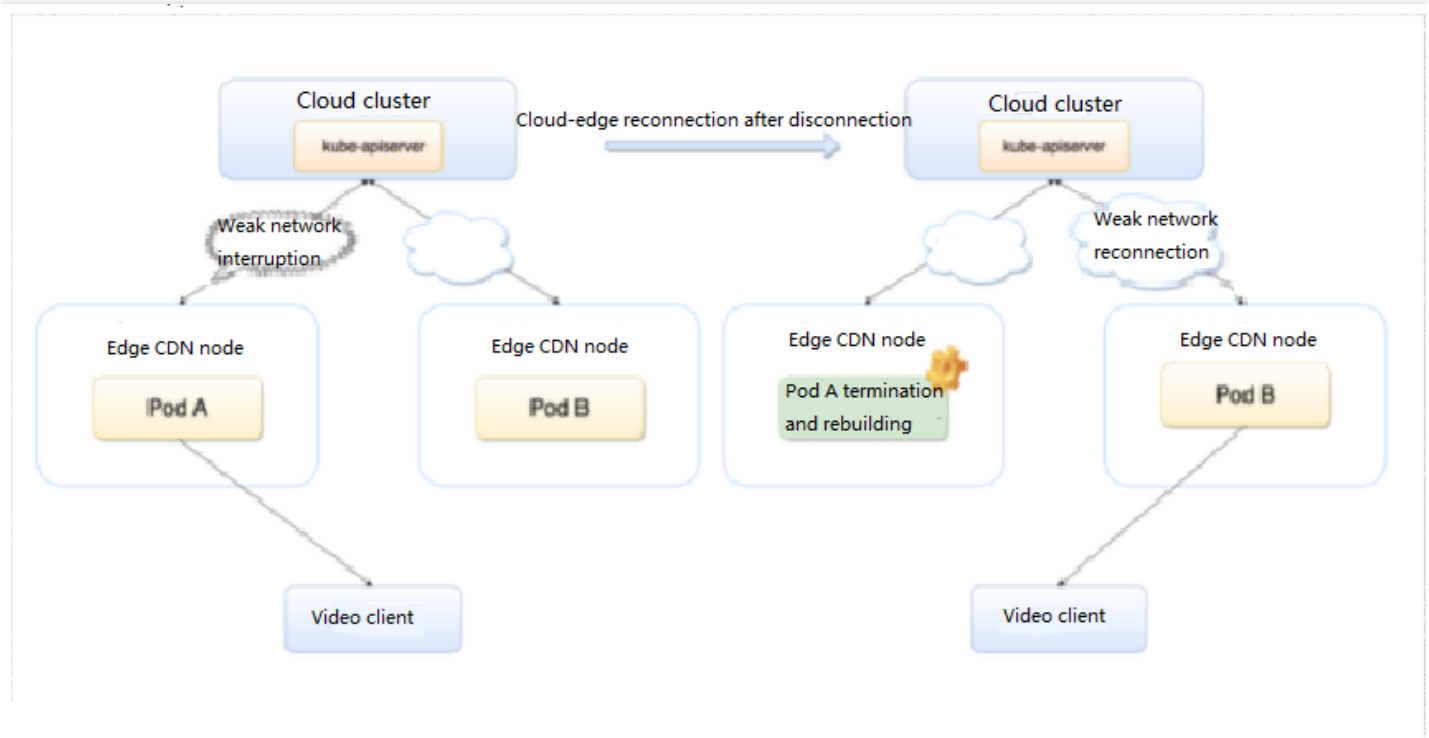


- The edge devices in the warehouse and plant and the cloud cluster can be connected over the internet, 5G, and Wi-Fi, with uneven and unguaranteed network quality.
- The edge devices in the warehouse and plant are connected over the local network, which is better and more reliable than that used to connect to cloud clusters.

Audio/Video pull

©2013-2022 Tencent Cloud. All rights reserved.

Audio/Video pull is as shown below:



Considering user experience and enterprise cost, you often need to improve the edge cache hit rate for audio/video pull to reduce the origin-pull traffic and schedule the same file requested by users to the same service instance and its cache file.

In the case of native Kubernetes, if Pods are frequently rebuilt due to network fluctuation, the service instance caching performance will be compromised, and the scheduling system will schedule user requests to other service instances. Both may have a significant or even unacceptable impact on the CDN performance.

In fact, if edge nodes are running normally, it is unnecessary to evict or rebuild Pods. To solve this problem and ensure the service continuity, the TKE Edge team proposed a distributed node status determination mechanism.

Challenges

Native Kubernetes processing method

Poor network conditions at the cloud edge affect the communication between the kubelet running on the edge node and the cloud API server, where the latter cannot receive the kubelet heartbeat or get a renewal and then cannot accurately get the running statuses of the node and its Pods. If the duration exceeds the set threshold, the API server will consider the node unavailable and perform the following operations:

©2013-2022 Tencent Cloud. All rights reserved.

- Set the status of the missing node to `NotReady` or `Unknown` and add the taints of `NoSchedule` and `NoExecute`.
- Evict the Pods on the missing node and rebuild them on other nodes.
- Remove the Pods on the missing node from the Service's Endpoint list.

Solution

Design principle

In edge computing, it is unreasonable to determine whether a node is normal solely based on the connection between the edge and the API server. You need an additional determination mechanism to make the system more robust.

The network between edge nodes is more stable than that between cloud and edge nodes. You can use a more stable infrastructure to improve the accuracy. TKE Edge adopts an innovative distributed mechanism to determine the node status, which considers edge nodes in addition to the connection between nodes and the API server. Tests and practices have shown that this mechanism has improved the accuracy to determine the node status in a system under poor network conditions at the cloud edge, safeguarding service running. It works as shown below:

- Each node regularly checks the health status of other nodes.
- All nodes in the cluster regularly vote on the status of each node.
- Cloud and edge nodes determine the node status together.

First, nodes check and vote for each other to decide whether a node is in abnormal status, and the decision will be made only when most of the nodes agree on the same determination. Second, even though the network between nodes is usually in a better condition than the cloud edge network, the complex situation at the edge node should be considered, as networks are not 100% reliable. Therefore, the network between nodes is not the only standard, and the node status should be decided by both nodes and the cloud edge. In this regard, the following design is made:

Final Node Status	Normal to the Cloud	Abnormal to the Cloud
Normal to other nodes	Normal	No more Pods are scheduled to this node.
Abnormal to other nodes	Normal	Existing Pods are evicted and removed from the Endpoint list. No more Pods are scheduled to this node.

Solution features

When the cloud determines that the node status is abnormal, but other nodes consider it normal, although existing Pods will not be evicted, new Pods will not be scheduled to the node to ensure the stability of the additional services.

©2013-2022 Tencent Cloud. All rights reserved.

Existing nodes will run normally thanks to the edge autonomy capability of the edge cluster.

Due to the particularity of the edge network and topology, there are often single points of failure between node groups. In a smart factory, although the warehouse and plant are in the same region, they are connected only through a key linkage. Once the linkage is broken, the network will be disconnected. The solution provided in this document ensures that the node group with more nodes will not be considered abnormal when two node groups are disconnected from each other. Therefore, Pods will always be scheduled to the one with more nodes, avoiding excessive node loads.

Edge devices may be in different regions and not be connected. The solution provided in this document supports the determination of statuses of nodes in multiple regions. It allows you to easily group nodes by region or other criteria for intra-group checks. Even if nodes are regrouped, you do not need to redeploy or re-initialize detection add-ons to adapt them to the network conditions of edge computing. After grouping, nodes will only determine the statuses of nodes within their group.

Prerequisites

To use this feature, you need to open port 51005 of the node for distributed, smart health checks between nodes.

Directions

> Note：
> It takes some time to deploy and configure edge and multi-region checks. They do not take effect immediately.

Enabling edge health

Edge Health is disabled by default. It can be manually enabled as instructed below:

1. Log in to the TKE console.
2. On the cluster list page, select the target edge cluster ID to enter the cluster details page.
3. Select **Basic Information** on the left sidebar to enter the **Basic Information** page.
4. On the **Basic Information** page, click **Enable Edge Health**.

Enabling multi-region

Under **Multi-region**, nodes are grouped by region. Node regions are identified by the `tencent.tkeedgehealth/topology-zone` label on the node. For example, `tencent.tkeedgehealth/topology-zone: zone0` indicates to group the node to `zone0`. Nodes with the

©2013-2022 Tencent Cloud. All rights reserved.

same label value are considered to be in the same region. After **Multi-region** is enabled, nodes in the same region will check and vote for each other.

Note：

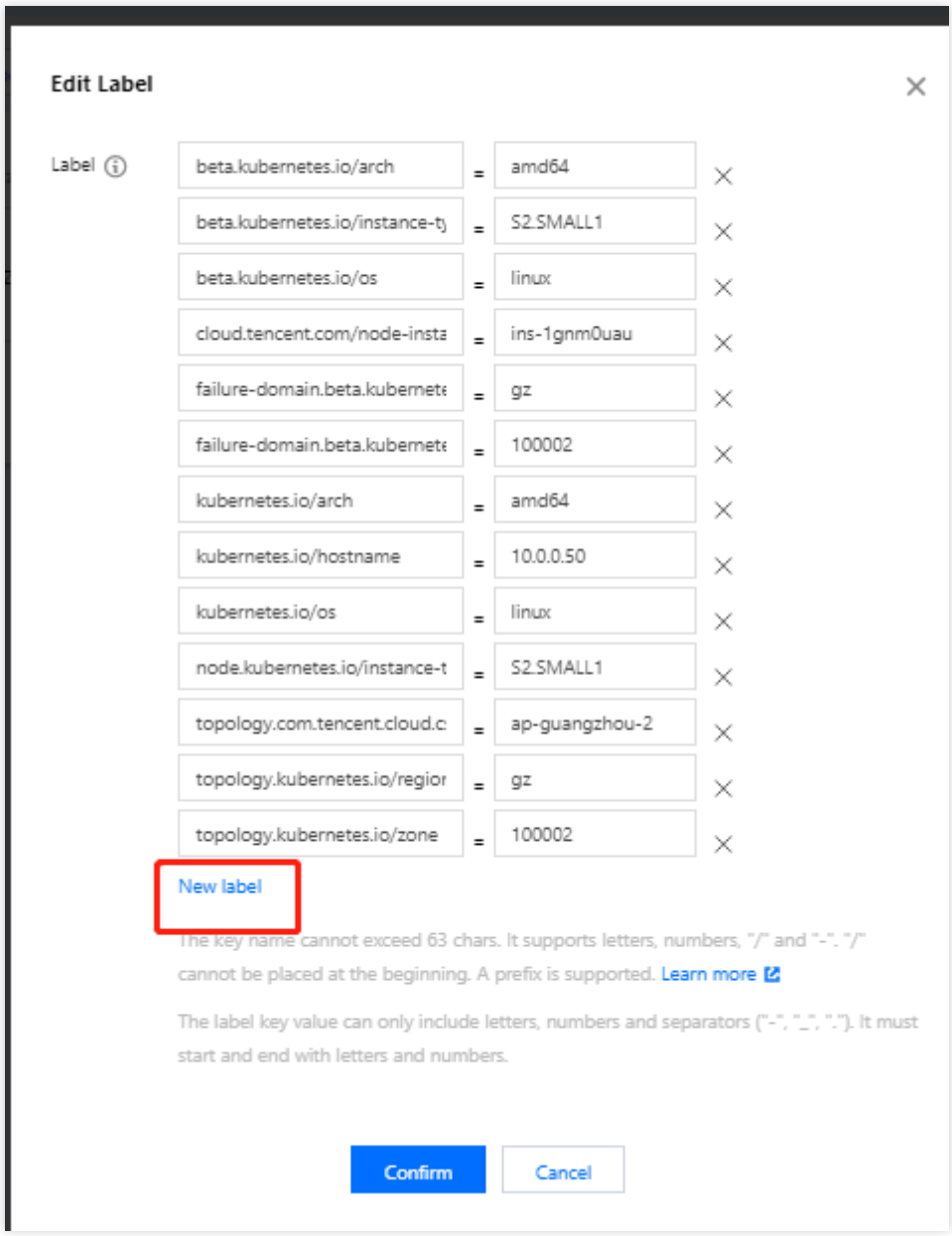
- If this feature is enabled without the `tencent.tkeedgehealth/topology-zone` label, the node will only check its own health status.
- If this feature is not enabled, all nodes in a cluster will check each other, even if the node is labeled `tencent.tkeedgehealth/topology-zone` .

Setting node region label in the console

1. Log in to the TKE console.
2. On the cluster list page, select the target edge cluster ID to enter the cluster details page.
3. Select **Node Management** > **Node** on the left sidebar to enter the **Node List** page.
4. Select **More** > **Edit Label** on the right of the target node.

©2013-2022 Tencent Cloud. All rights reserved.

5. In the **Edit Label** pop-up window, edit the label and click **Submit** as shown below:



Enabling multi-region

After enabling Edge Health, click **Enable Multi-region**.

©2013-2022 Tencent Cloud. All rights reserved.

Security
Pod Security Group

Last updated：2022-11-02 11:54:52

Pod security groups integrate CVM security groups and Kubernetes Pods. You can use CVM security groups to define rules, so as to allow the inbound and outbound network traffic of Pods running on different TKE nodes (currently, only super nodes are supported, and general nodes will be supported).

Limits

Consider the following limits before using security groups for Pods:

- Pods must run in TKE clusters on v1.20 or later.
- Only super nodes are supported for Pod security groups, and more node types will be released.
- Pod security groups cannot be used together with dual-stack clusters.
- Super nodes are only supported in some regions. For more information, see Regions and Availability Zones.

Enabling Security Group Capabilities for Pods

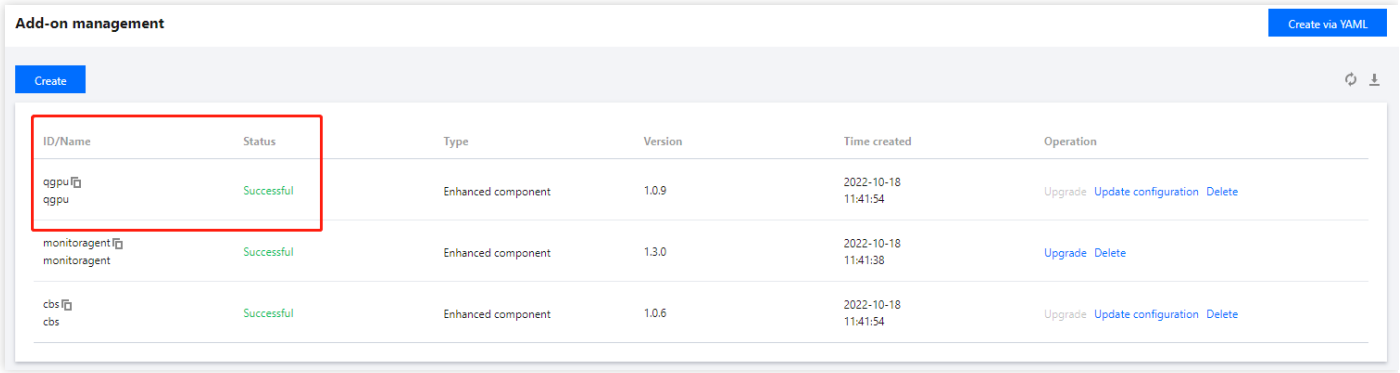
Installing the add-on

1. Log in to the TKE console.
2. Install the `SecurityGroupPolicy` add-on for the cluster.

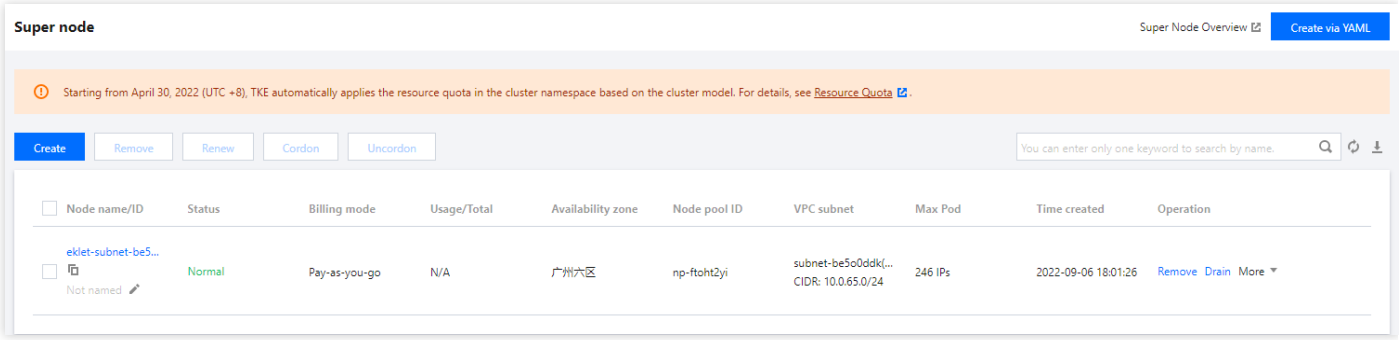
- If you haven't created a cluster yet, you can install the `SecurityGroupPolicy` add-on during creation. For detailed directions, see Add-On Lifecycle Management.
- To enable security group capabilities for Pods in a created cluster, install the `SecurityGroupPolicy` add-on on the **Add-On Management** page. For detailed directions, see Add-On Lifecycle Management.

3. On the **Add-On Management** page, view the add-on status. If the status is **Success**, the add-on has been deployed, as shown below:

©2013-2022 Tencent Cloud. All rights reserved.



4. On the super node page, verify that your TKE general cluster contains a super node. Currently, you can enable security group capabilities only for Pods scheduled to a super node.



Deploying the Sample Application

To use security groups for Pods, you must deploy SecurityGroupPolicy in your cluster. The following describes how to use the security group policy for a Pod via CloudShell. Unless otherwise stated, the steps should be performed on the same terminal, as the variables involved don't apply to different terminals.

Deploying the sample Pod with a security group

1. Create a security group to be used with the Pod. The following describes how to create a simple security group and is for reference only. The rules may differ in a production cluster.
   a. Search for the VPC and security group ID of the cluster. Replace `my-cluster` with the actual value.

```
my_cluster_name=my-cluster
my_cluster_vpc_id=$(tccli tke DescribeClusters --cli-unfold-argument --ClusterIds $my_cluster_name --filter Clusters[0].ClusterNetworkSettings.VpcId | sed 's/\"//g')
my_cluster_security_group_id=$(tccli vpc DescribeSecurityGroups --cli-unfold-argument --Filters.0.Name security-group-name --Filters.0.Values tke-worker-secur
```

©2013-2022 Tencent Cloud. All rights reserved.

```
ity-for-$my_cluster_name --filter SecurityGroupSet[0].SecurityGroupId | sed 's/
\"//g')
```

b. Create a security group for your Pod. Replace `my-pod-security-group` with the actual value. Record the security group ID returned by the command for further use.

```
my_pod_security_group_name=my-pod-security-group
tccli vpc CreateSecurityGroup --GroupName "my-pod-security-group" --GroupDescri
ption "My pod security group"
my_pod_security_group_id=$(tccli vpc DescribeSecurityGroups --cli-unfold-argume
nt --Filters.0.Name security-group-name --Filters.0.Values my-pod-security-grou
p --filter SecurityGroupSet[0].SecurityGroupId | sed 's/\"//g')
echo $my_pod_security_group_id
```

c. Allow the traffic over TCP and UDP on port 53 from the Pod security group created in the previous step to the cluster security group, so that the Pod can access the application through the domain name.

```
tccli vpc CreateSecurityGroupPolicies --cli-unfold-argument --SecurityGroupId
$my_cluster_security_group_id --SecurityGroupPolicySet.Ingress.0.Protocol UDP -
-SecurityGroupPolicySet.Ingress.0.Port 53 --SecurityGroupPolicySet.Ingress.0.Se
curityGroupId $my_pod_security_group_id --SecurityGroupPolicySet.Ingress.0.Acti
on ACCEPT
tccli vpc CreateSecurityGroupPolicies --cli-unfold-argument --SecurityGroupId
$my_cluster_security_group_id --SecurityGroupPolicySet.Ingress.0.Protocol TCP -
-SecurityGroupPolicySet.Ingress.0.Port 53 --SecurityGroupPolicySet.Ingress.0.Se
curityGroupId $my_pod_security_group_id --SecurityGroupPolicySet.Ingress.0.Acti
on ACCEPT
```

d. Allow the inbound traffic over any protocol and port from the Pod associated with the security group to the Pod associated with any security group, and allow the outbound traffic over any protocol and port from the Pod associated with the security group.

```
tccli vpc CreateSecurityGroupPolicies --cli-unfold-argument --SecurityGroupId
$my_pod_security_group_id --SecurityGroupPolicySet.Ingress.0.Protocol ALL --Sec
urityGroupPolicySet.Ingress.0.Port ALL --SecurityGroupPolicySet.Ingress.0.Secur
ityGroupId $my_pod_security_group_id --SecurityGroupPolicySet.Ingress.0.Action
ACCEPT
tccli vpc CreateSecurityGroupPolicies --cli-unfold-argument --SecurityGroupId
$my_pod_security_group_id --SecurityGroupPolicySet.Egress.0.Protocol ALL --Secu
rityGroupPolicySet.Egress.0.Port ALL --SecurityGroupPolicySet.Egress.0.Action A
CCEPT
```

©2013-2022 Tencent Cloud. All rights reserved.

2. Create a Kubernetes namespace to deploy resources.

```
kubectl create namespace my-namespace
```

3. Deploy the `SecurityGroupPolicy` in your cluster.

a. Save the following sample security policy as `my-security-group-policy.yaml`. If you prefer to select a Pod by service account tag, you can replace `podSelector` with `serviceAccountSelector`, and you must specify a selector. If you specify multiple security groups, all their rules will take effect for the selected Pod. Replace `$my_pod_security_group_id` with the security group ID recorded in the previous step.

```
apiVersion: vpcresources.tke.cloud.tencent.com/v1beta1
kind: SecurityGroupPolicy
metadata:
name: my-security-group-policy
namespace: my-namespace
spec:
podSelector:
matchLabels:
app: my-app
securityGroups:
groupIds:
- $my_pod_security_group_id
```

Note

Consider the following limits when specifying one or multiple security groups for the Pod:

- They must exist.
- They must allow inbound requests from cluster security groups (for kubelet) and health checks configured for the Pod.
- Your CoreDNS Pod security groups must allow the inbound traffic over TCP and UDP on port 53 from Pod security groups.
- They must have necessary inbound and outbound rules to communicate with other Pods.

A security group policy applies only to newly scheduled Pods and doesn't affect running Pods. To make it effective for existing Pods, you need to verify that the existing Pods meet the above limits before manually recreating it.

©2013-2022 Tencent Cloud. All rights reserved.

b. Deploy the policy.

```
kubectl apply -f my-security-group-policy.yaml
```

4. To deploy the sample application, use the `my-app` match tag specified by using the `podSelector` in the previous step.

   a. Save the following content as `sample-application.yaml` .

```
apiVersion: apps/v1
kind: Deployment
metadata:
name: my-deployment
namespace: my-namespace
labels:
app: my-app
spec:
replicas: 2
selector:
matchLabels:
app: my-app
template:
metadata:
labels:
app: my-app
spec:
terminationGracePeriodSeconds: 120
containers:
- name: nginx
image: nginx:latest
ports:
- containerPort: 80
nodeSelector:
node.kubernetes.io/instance-type: eklet
tolerations:
- effect: NoSchedule
key: eks.tke.cloud.tencent.com/eklet
operator: Exists
---
apiVersion: v1
kind: Service
metadata:
name: my-app
namespace: my-namespace
labels:
app: my-app
```

©2013-2022 Tencent Cloud. All rights reserved.

```
spec:
selector:
app: my-app
ports:
- protocol: TCP
port: 80
targetPort: 80
```

b. Run the following command to deploy the application. During deployment, Pods will be preferably scheduled to super nodes, and the security group specified in the previous step will be applied to the Pod.

```
kubectl apply -f sample-application.yaml
```

Note：
If you don't use `nodeSelector` to preferably schedule the Pod to a super node, when it is scheduled to another node, the security group will not take effect, and `kubectl describe pod` will output "security groups is only support super node, node 10.0.0.1 is not super node".

5. View the Pod deployed by using the sample application. So far, the involved terminal is `TerminalA` .

```
kubectl get pods -n my-namespace -o wide
```

Below is the sample output:

```
NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES
my-deployment-866ffd8886-9zfrp 1/1 Running 0 85s 10.0.64.10 eklet-subnet-q21ras
u6-8bpgyx9r <none> <none>
my-deployment-866ffd8886-b7gzb 1/1 Running 0 85s 10.0.64.3 eklet-subnet-q21rasu
6-8bpgyx9r <none> <none>
```

6. Go to any Pod on another terminal (`TerminalB`) and replace the Pod ID with the one returned in the previous step.

```
kubectl exec -it -n my-namespace my-deployment-866ffd8886-9zfrp -- /bin/bash
```

7. Verify that the sample application works normally on `TerminalB` .

©2013-2022 Tencent Cloud. All rights reserved.

```
curl my-app
```

Below is the sample output:

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...
```

You receive a response, as all Pods of the running application are associated with the security group you create, which contains the following rules:

i. Allow all traffic between all Pods associated with the security group.
ii. Allow the DNS traffic from the security group to the cluster security group associated with your node. CoreDNS Pods are running on these nodes, and your Pod will search for `my-app` by domain name.

8. On `TerminalA` , delete the security group rule that allows DNS communication from the cluster security group.

```
tccli vpc DeleteSecurityGroupPolicies --cli-unfold-argument --SecurityGroupId
$my_cluster_security_group_id --SecurityGroupPolicySet.Ingress.0.Protocol UDP -
-SecurityGroupPolicySet.Ingress.0.Port 53 --SecurityGroupPolicySet.Ingress.0.Se
curityGroupId $my_pod_security_group_id --SecurityGroupPolicySet.Ingress.0.Acti
on ACCEPT
tccli vpc DeleteSecurityGroupPolicies --cli-unfold-argument --SecurityGroupId
$my_cluster_security_group_id --SecurityGroupPolicySet.Ingress.0.Protocol TCP -
-SecurityGroupPolicySet.Ingress.0.Port 53 --SecurityGroupPolicySet.Ingress.0.Se
curityGroupId $my_pod_security_group_id --SecurityGroupPolicySet.Ingress.0.Acti
on ACCEPT
```

9. On `TerminalB` , try accessing the application again.

```
curl my-app
```

The trial will fail, as the Pod cannot access the CoreDNS Pod, and the cluster security group no longer allows DNS communication from Pods associated with the security group.
If you try using an IP to access the application, you will receive a response, as all ports allow the communication between Pods associated with the security group, and no domain name search is required.

©2013-2022 Tencent Cloud. All rights reserved.

0. After the trial, run the following command to delete the sample security group policy, application, and security group.

```
kubectl delete namespace my-namespace
tccli vpc DeleteSecurityGroup --cli-unfold-argument --SecurityGroupId $my_pod_security_group_id
```

©2013-2022 Tencent Cloud. All rights reserved.

Container Image Signature and Verification

Last updated：2023-05-18 17:04:29

Image signature and signature verification can avoid man-in-the-middle attacks and the update and running of invalid images, ensuring image consistency across the entire linkage ranging from distribution to deployment.

Container image signature

TCR Enterprise Edition supports namespace-level automatic image signature. When an image is pushed to the registry, it will be automatically signed according to the matched signature policy to ensure image content trustworthiness in your registry.

Image signature verification

TKE provides the image signature verification add-on Cerberus, which verifies signed images for trustworthiness. This is to ensure that only container images signed by trusted authorizing parties are deployed in TKE clusters, thereby reducing the risks to image security in the container environment.

©2013-2022 Tencent Cloud. All rights reserved.

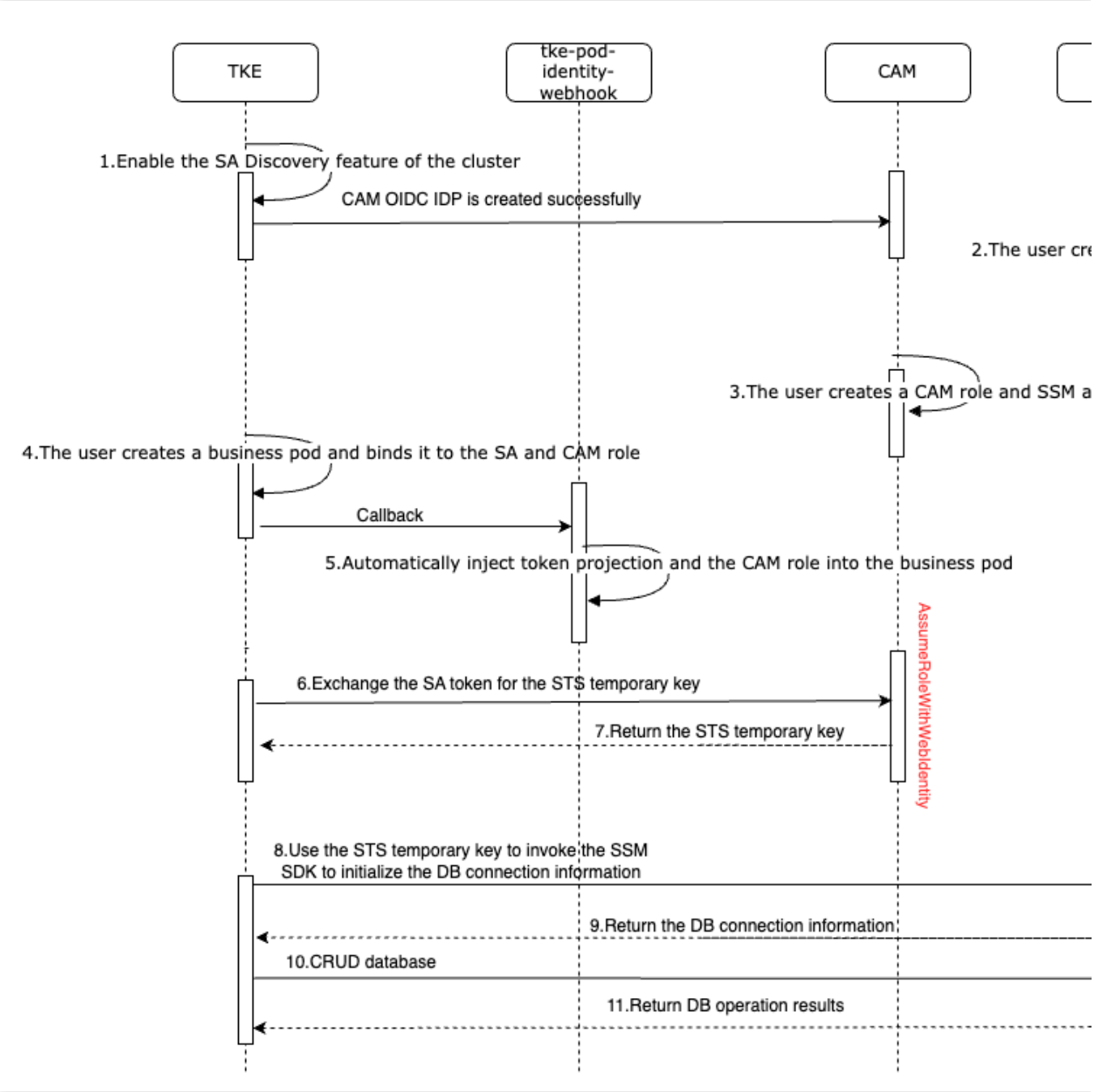
how to use CAM to authenticate databases for workloads running in TKE

Last updated：2024-02-05 17:22:47

Background

Running containerized workloads in a Tencent Cloud managed cluster involves accessing SQL or NoSQL databases outside the cluster. To use SQL databases together with Kubernetes, we need to consider the issues of secret rotation and sensitive data transfer. With Secrets Manager (SSM) and Cloud Access Management (CAM), we can eliminate risks arising from verifying databases with username and key. In addition, SSM features scheduled secret rotation to reduce labor efforts.
This document describes how to use CAM to authenticate databases for workloads running in TKE. In the example below, we create a TencentDB instance and create a secret for it in SSM. Then, enable the resource access control feature of OIDC, make the created CAM OIDC provider as the carrier for role creation, and associate with the policies of accessing TencentDB and SSM. Finally, we securely connect to the TencentDB database through the Kubernetes service account with CAM and SSM. The entire architecture is as follows:

©2013-2022 Tencent Cloud. All rights reserved.



Limits

Only TKE managed clusters are supported.

Supports cluster version ≥ v1.20.6-tke.27/v1.22.5-tke.1

Directions

©2013-2022 Tencent Cloud. All rights reserved.

Step 1. Create a managed cluster

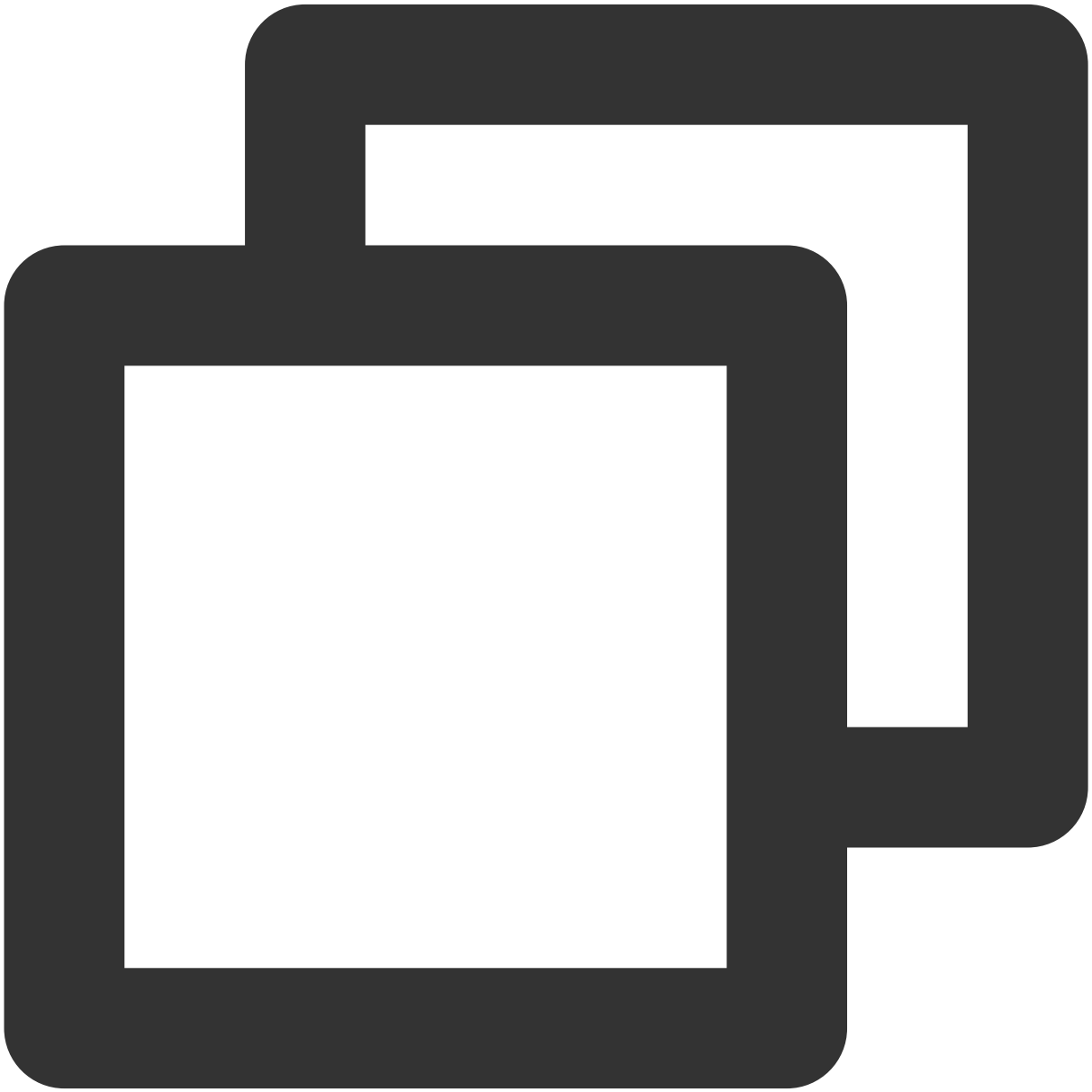
1. Log in to the TKE console to create a cluster.

Notes

You can create a managed cluster as instructed in Creating a cluster.

To use an existing managed cluster, check the cluster version on the details page, and upgrade the version if necessary. See Upgrading a Cluster.

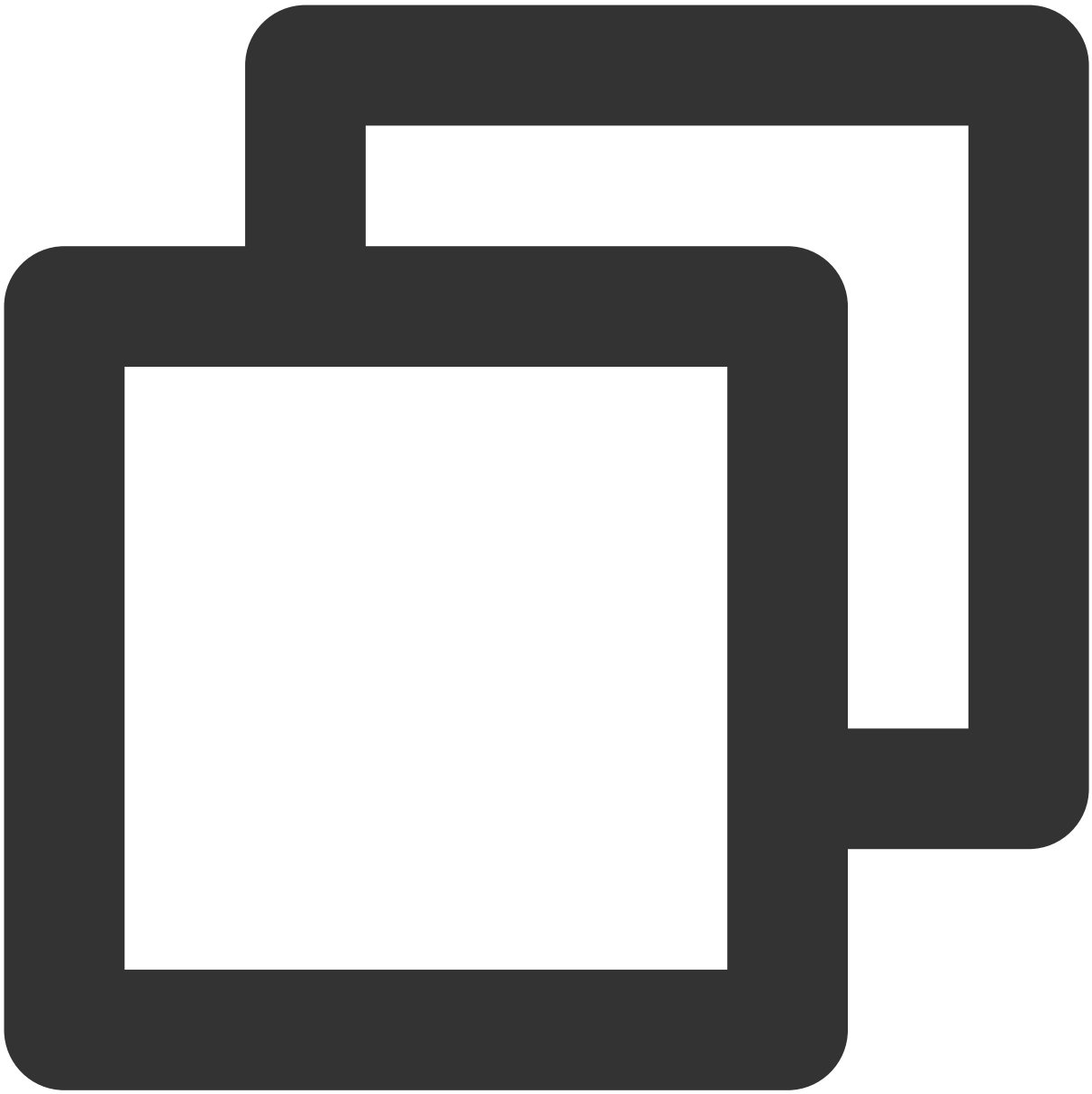
2. Run the following command to access the managed cluster through the kubectl client.



```
kubectl get node
```

©2013-2022 Tencent Cloud. All rights reserved.

The following message indicates that the cluster can be accessed.



```
kubectl get node
NAME          STATUS    ROLES    AGE    VERSION
10.0.4.144    Ready     <none>   24h    v1.22.5-tke.1
```

Notes

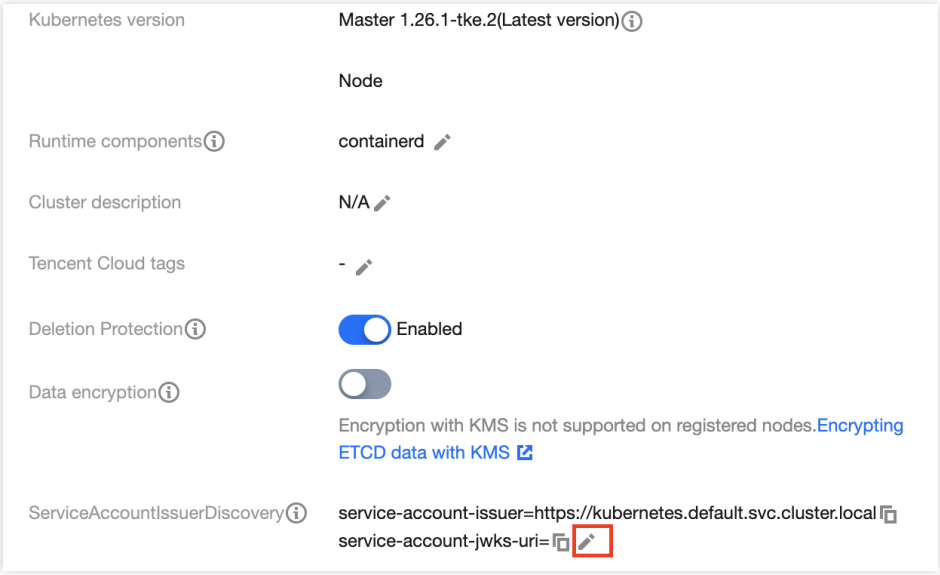
You can connect to a TKE cluster from a local client using kubectl, the Kubernetes command line tool. For details, see Connecting to a Cluster.

Step 2. Enable resource access control of OIDC

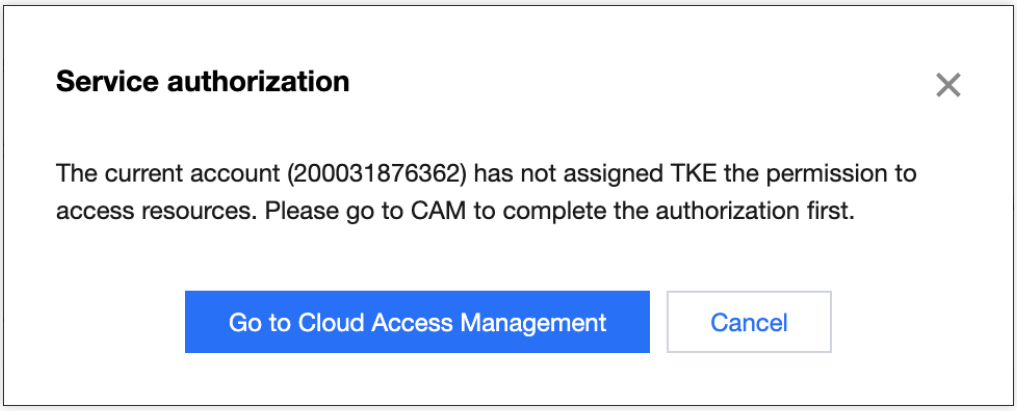
©2013-2022 Tencent Cloud. All rights reserved.

1. On the cluster details page, click

on the right of ServiceAccountIssuerDiscovery.

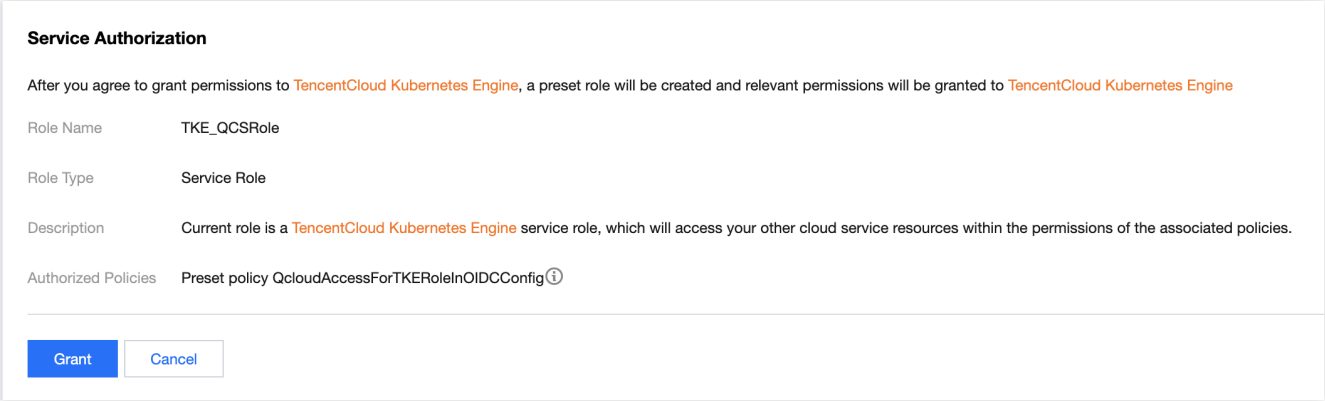


2. On the **Modify ServiceAccountIssuerDiscovery parameters** page, if you are prompted that you do not have permission to modify the parameters, please obtain permission first.



View the authorization policy QcloudAccessForTKERoleInOIDCConfig on the role management page, and click **Grant**.

©2013-2022 Tencent Cloud. All rights reserved.

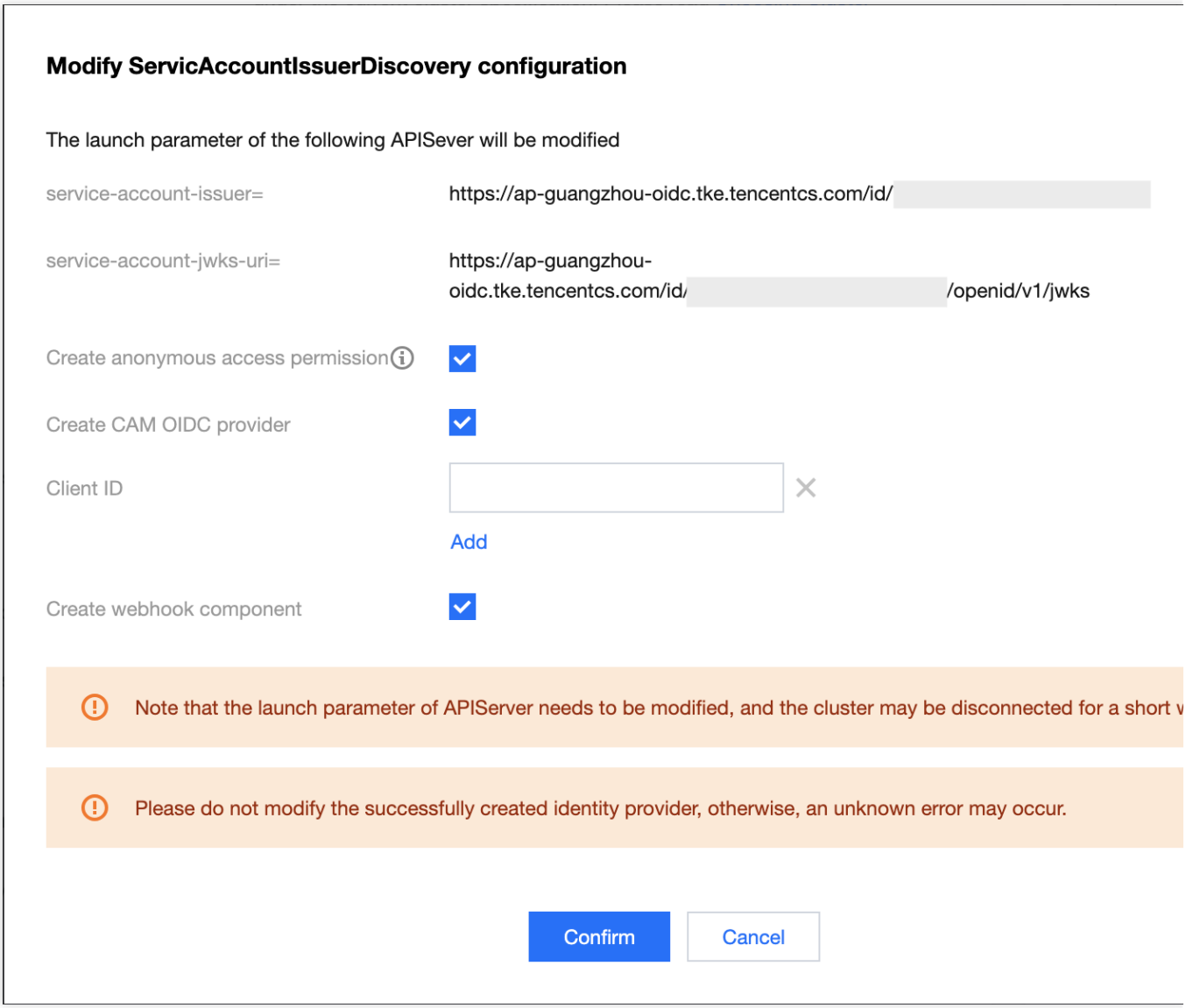


3. Select **Create CAM OIDC provider** and **Create WEBHOOK component**, and enter the client ID. Then click **OK**.

Notes

Client ID is optional. The default value "sts.cloud.tencent.com" is entered when it is not specified. In this example, we use the default value.

©2013-2022 Tencent Cloud. All rights reserved.



4. Go back to the cluster details page. When ServiceAccountIssuerDiscovery is available for modification again, the resource access control is enabled successfully.

Notes

The values of "service-account-issuer" and "service-account-jwks-uri" are default and cannot be modified.

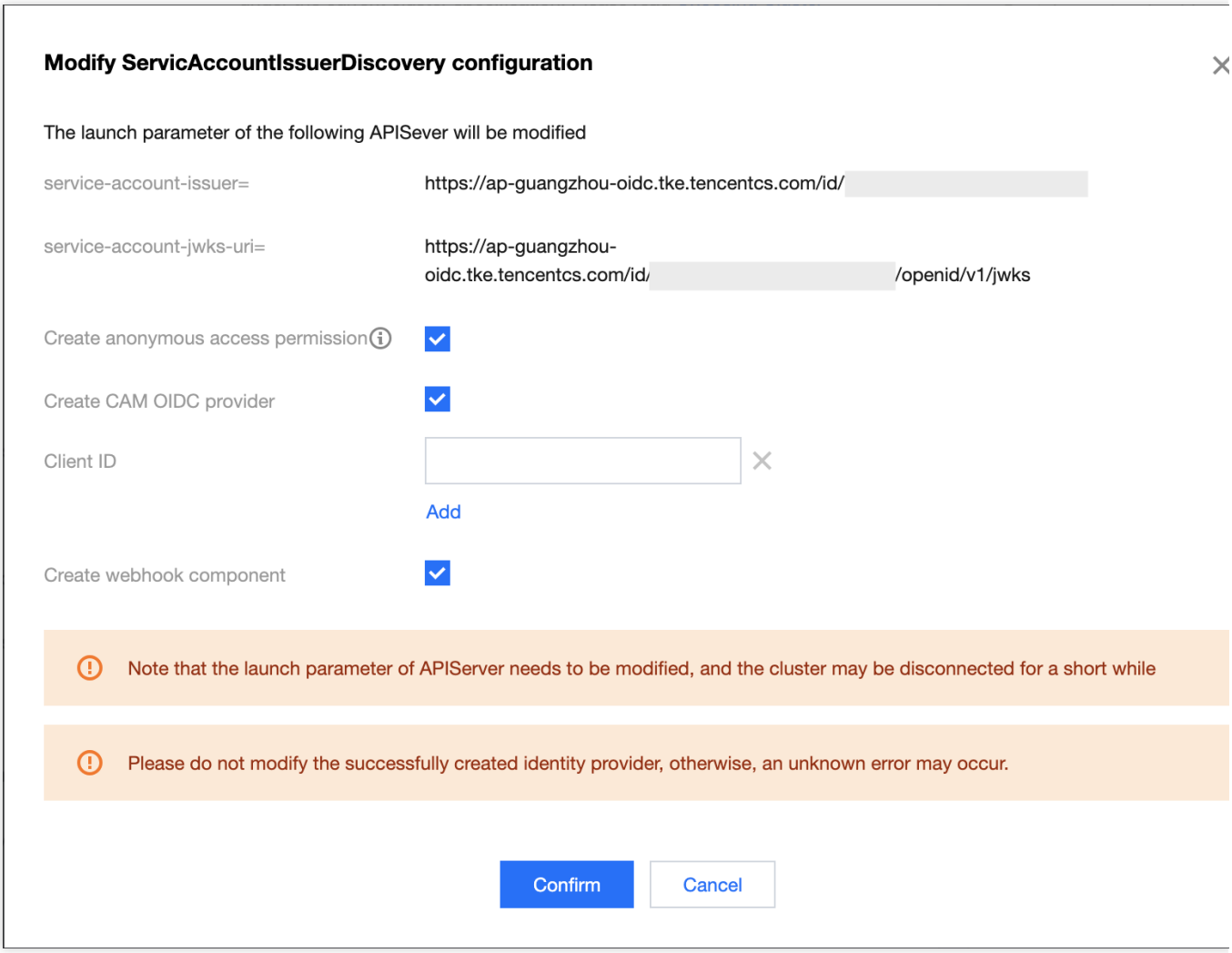
Step 3. Check if the CAM OIDC provider and WEBHOOK component are created successfully

1. Click

on the right of ServiceAccountIssuerDiscovery on the cluster details page.

2. On the **Modify ServiceAccountIssuerDiscovery parameters** page, you can see the prompt: "You have created the identity provider. Check details". Click **Check details**.

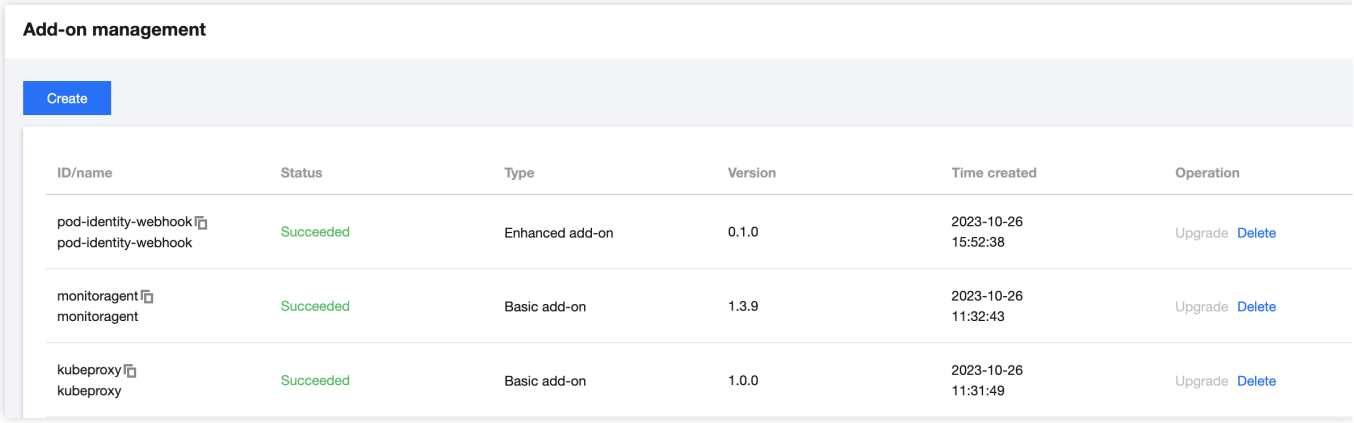
©2013-2022 Tencent Cloud. All rights reserved.



3. Check details of the CAM OIDC provider you created.

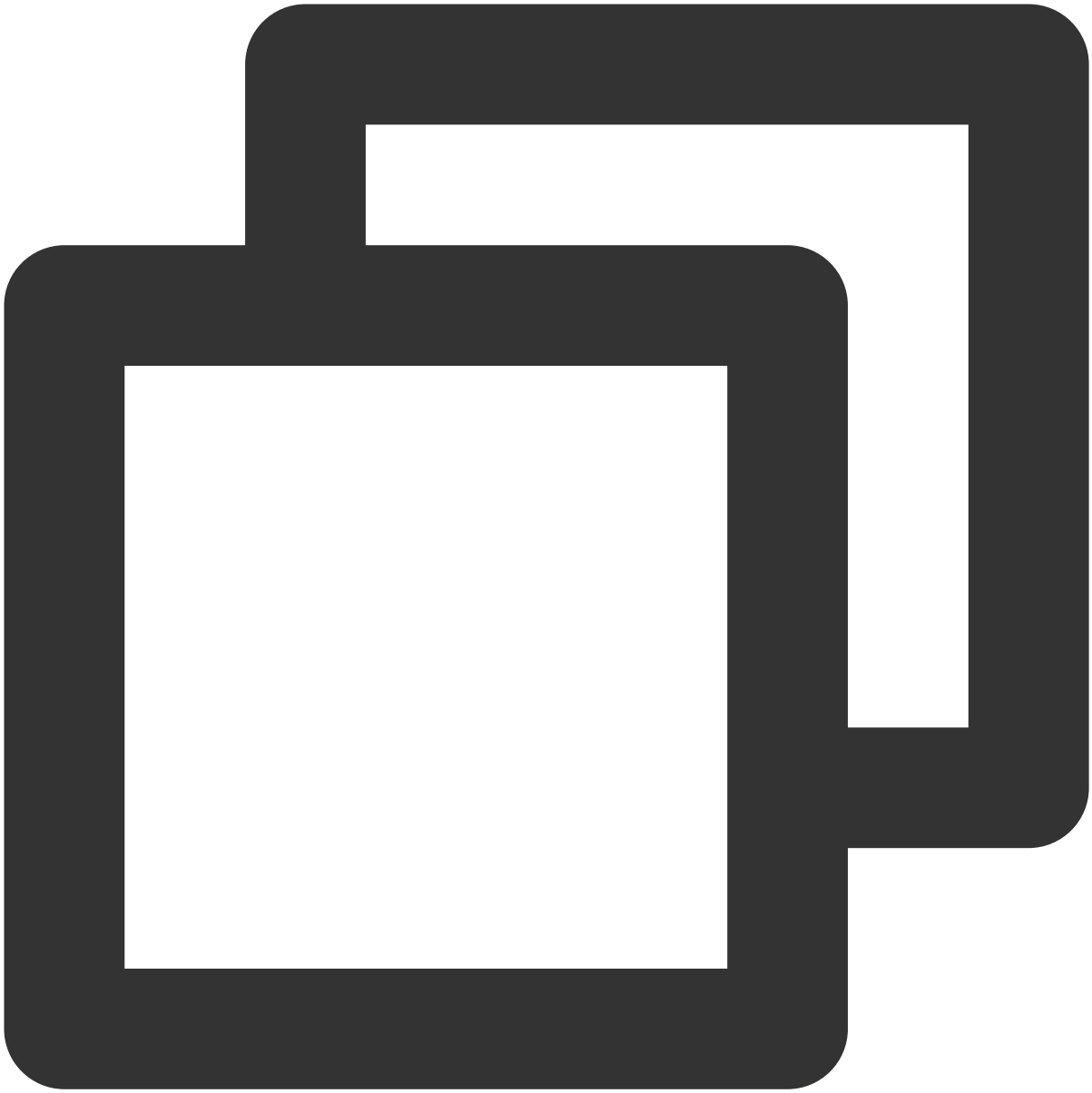
4. Go to **Cluster information > Add-on management**. If the status of the pod-identity-webhook component is "Succeeded", the component is installed successfully.

©2013-2022 Tencent Cloud. All rights reserved.



You can also run the command to check the installation status. If the status of the Pod with the prefix of "pod-identity-webhook" is "Running", the component is installed successfully.

©2013-2022 Tencent Cloud. All rights reserved.



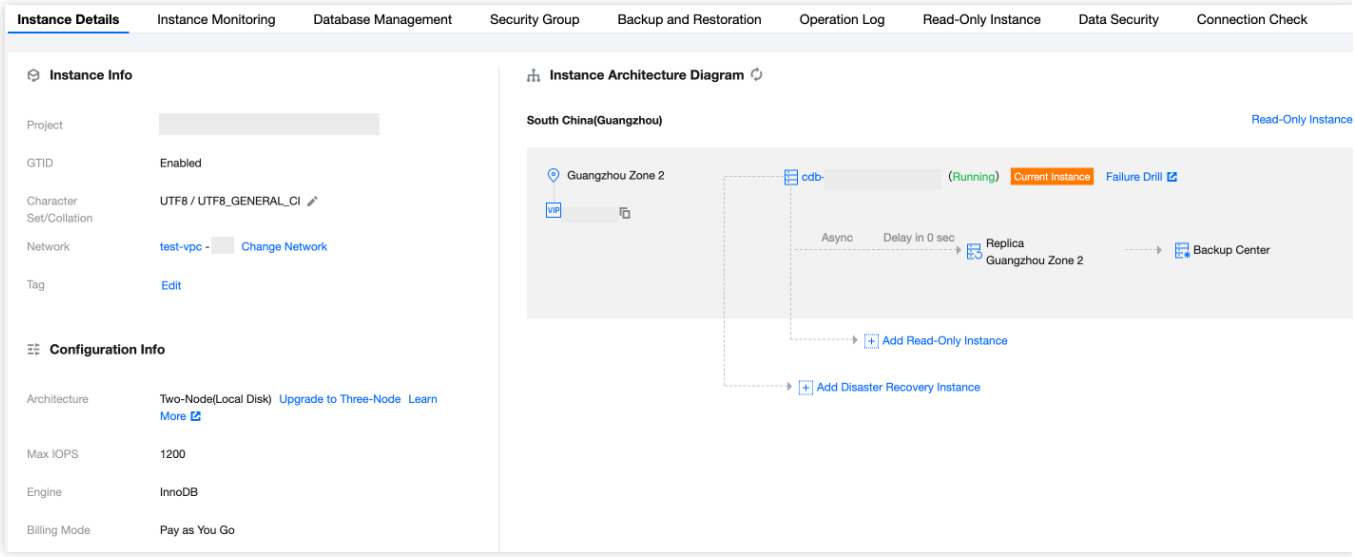
```
kubectl get pod -n kube-system
NAMESPACE     NAME                                       READY   STATUS    RESTARTS
kube-system   pod-identity-webhook-78c76****-9qrpj   1/1     Running   0
```

Step 4. Confirm the TencentDB instance

You need to confirm whether a TencentDB instance exists. If not exists, please create an instance first, and create a database in the instance. Skip the database creation if a TencentDB instance exists.

©2013-2022 Tencent Cloud. All rights reserved.

In this example, a TencentDB for MySQL instance is used, and the public network access is enabled for the instance. For details on instance creation, see Creating MySQL Instance.



Notes

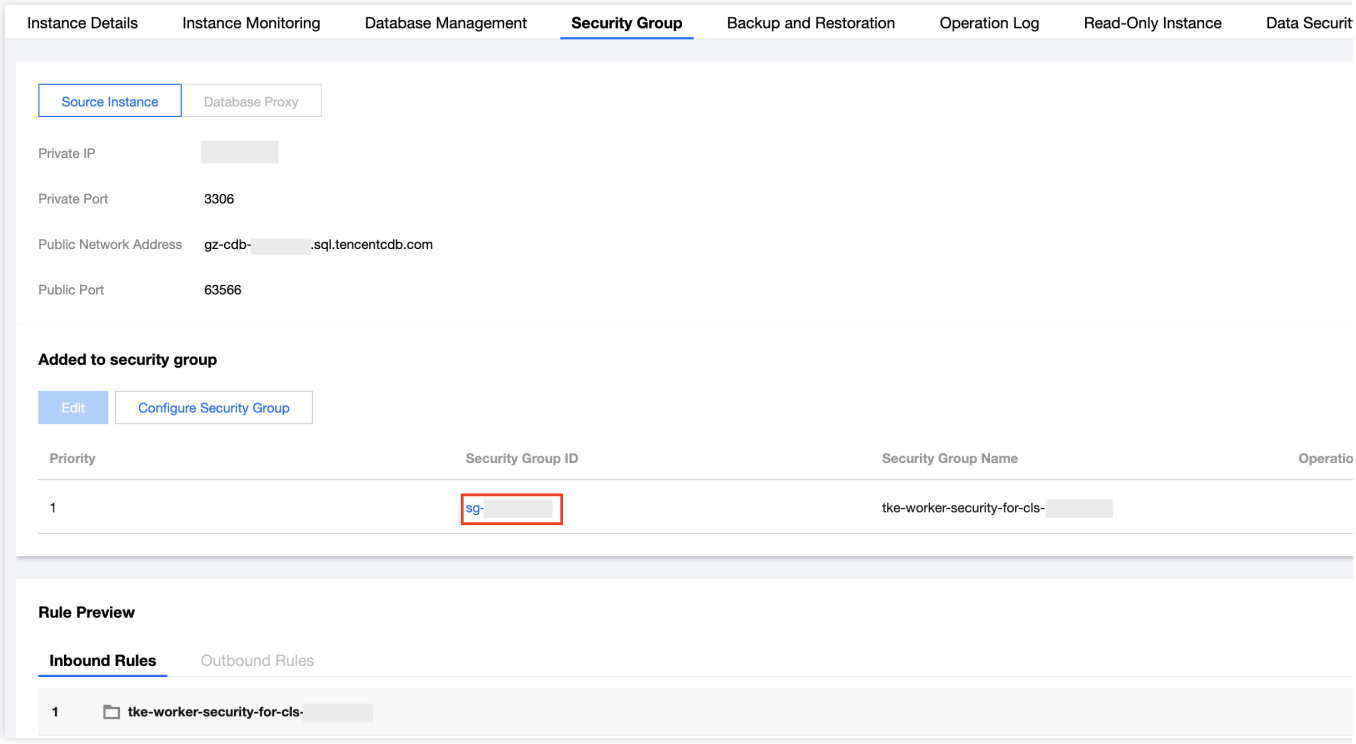
The value of **Public network address** is identified as `$db_address` .

The value of **Port** is identified as `$db_port` .

Step 5. Update security group of the database

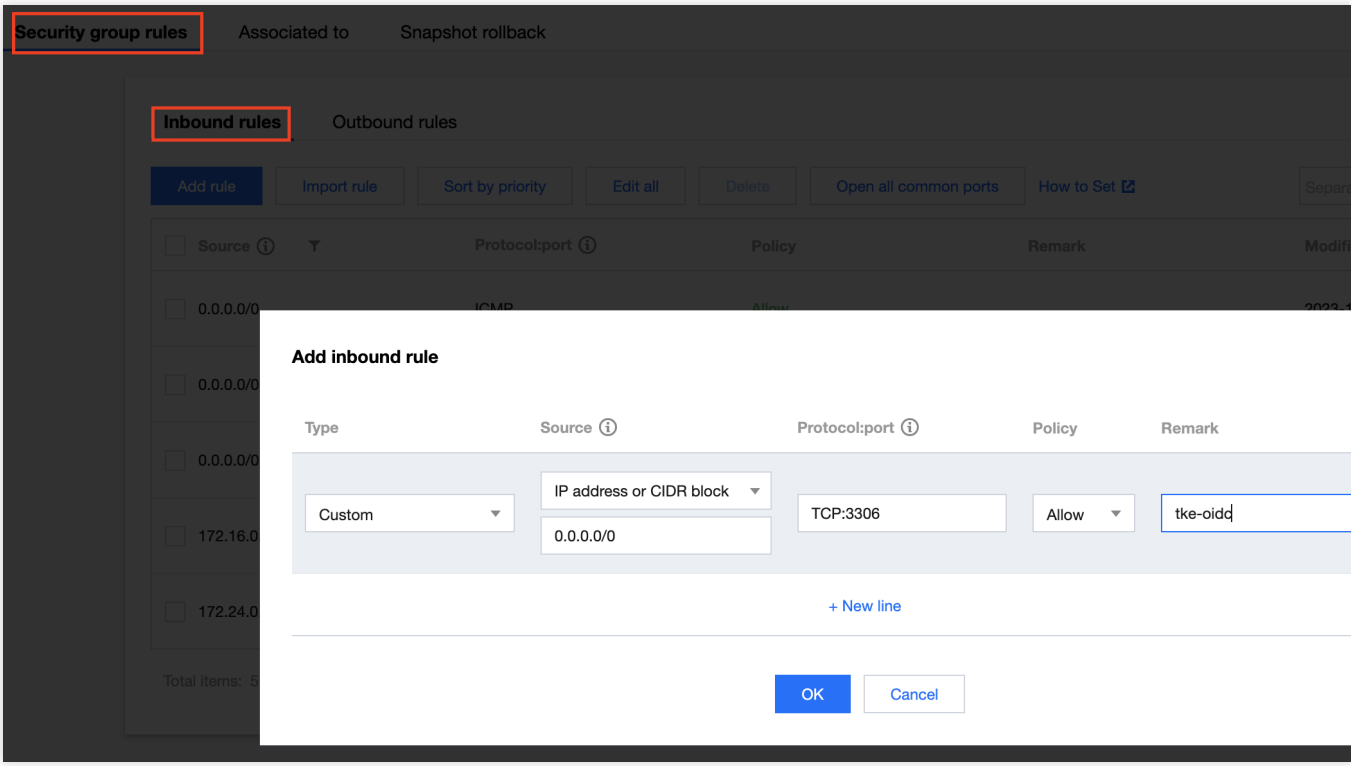
To allow the Pods in a managed cluster to access the TencentDB for MySQL database, you need to configure the security group rules for the database. Modify the security group rules on the security group management page.

©2013-2022 Tencent Cloud. All rights reserved.



To create inbound rules for Kubernetes Pods, please click **Security group ID** to go to the security group instance page. On the security group instance details page, click **Security group rules > Inbound rules > Add rule**. In the **Add inbound rules** window, create inbound rules. In this example, the **Source** is `0.0.0.0/0` , and the **Protocol port** is `TCP:3306` .

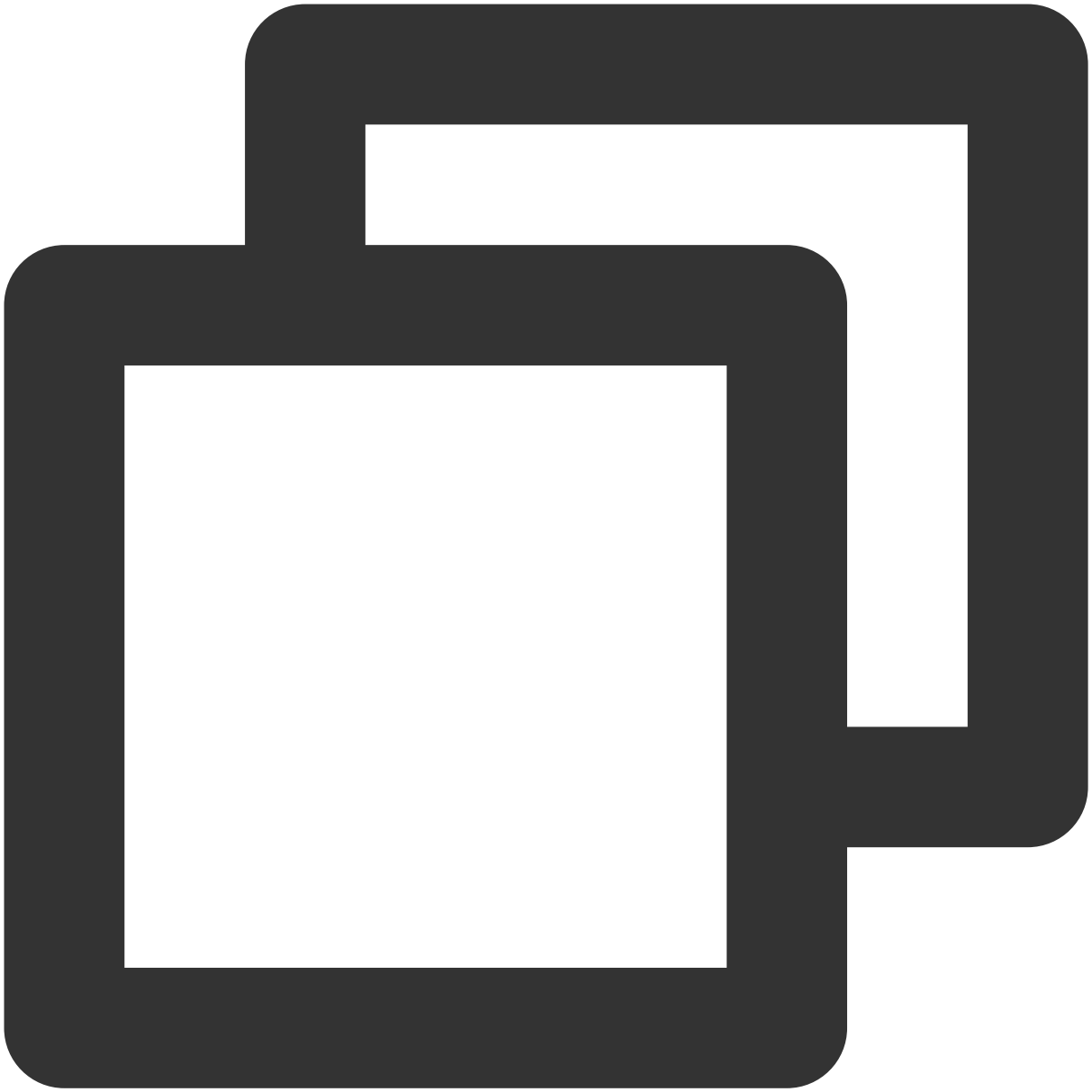
©2013-2022 Tencent Cloud. All rights reserved.



Step 6. Test connectivity of the database

In the instance where a MySQL client is installed, connect to the database with the username "root" and the database password you set at the time of creation. If the connection is failed, please go back to check if the public network is enabled and the security group is correctly configured.

©2013-2022 Tencent Cloud. All rights reserved.

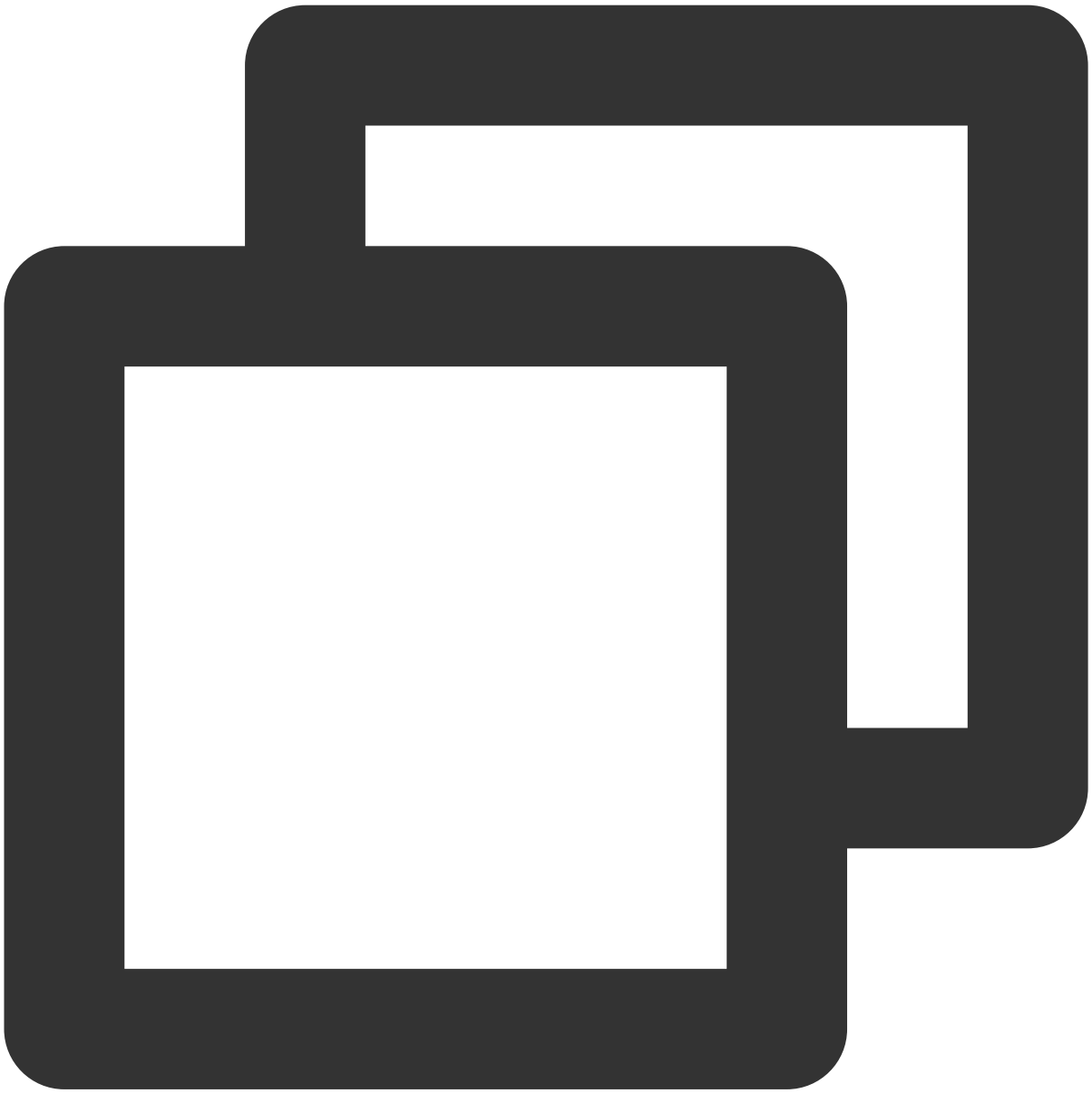


```
mysql -h $db_address  -P $db_port -uroot -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \\g.
Your MySQL connection id is 4238098
Server version: 5.7.36-txsql-log 20211230

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\\h' for help. Type '\\c' to clear the current input statement.

MySQL [(none)]>
```

©2013-2022 Tencent Cloud. All rights reserved.

Step 7. Create a database

To verify the connectivity and operation permissions for the database, please create a database first.

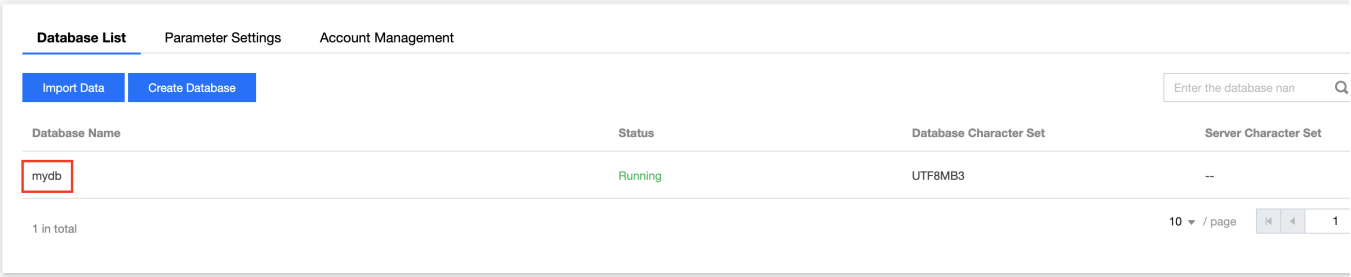


```
MySQL [(none)]> CREATE DATABASE mydb;
Query OK, 1 row affected (0.00 sec)

MySQL [(none)]> CREATE TABLE mydb.user (Id VARCHAR(120), Name VARCHAR(120));
Query OK, 0 rows affected (0.00 sec)
```

©2013-2022 Tencent Cloud. All rights reserved.

```
MySQL [(none)]> INSERT INTO mydb.user (Id,Name) VALUES ('123','tke-oidc');
Query OK, 1 row affected (0.01 sec)

MySQL [(none)]> SELECT * FROM mydb.user;
+------+----------+
| Id   | Name     |
+------+----------+
| 123  | tke-oidc |
+------+----------+
1 row in set (0.01 sec)
```

After the database is created, you can view it in the console.



Notes

The value of the **Database name** is identified as `$db_name` .

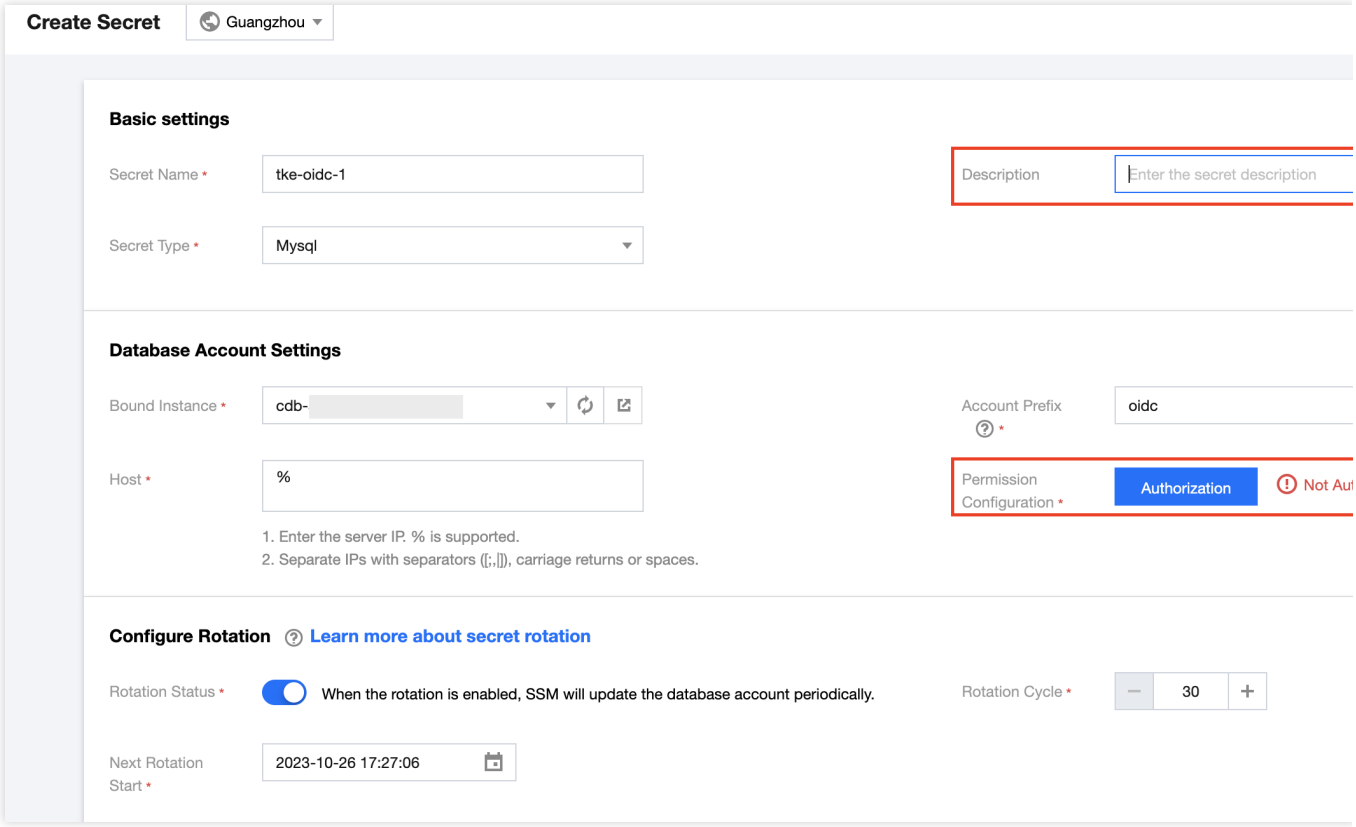
Step 8. Create a database secret instance in SSM

Check whether you have a database secret. If not, please create a database secret, enable secret rotation and select encryption in the SSM console to reduce disclosure risks and security threats to your account. In this example, we will create two database secrets, one of which has the "select" permission to the database. The two secrets are distinguished by the value of **Description**.

1. Log in to the SSM console.

2. On the **Create secret** page, configure the database account as instructed below. For details, see Creating Database Secret.

©2013-2022 Tencent Cloud. All rights reserved.



Bound instance: You can select an existing database instance or create a new one.

Server: Client IP. Enter `%` if you don't want to specify

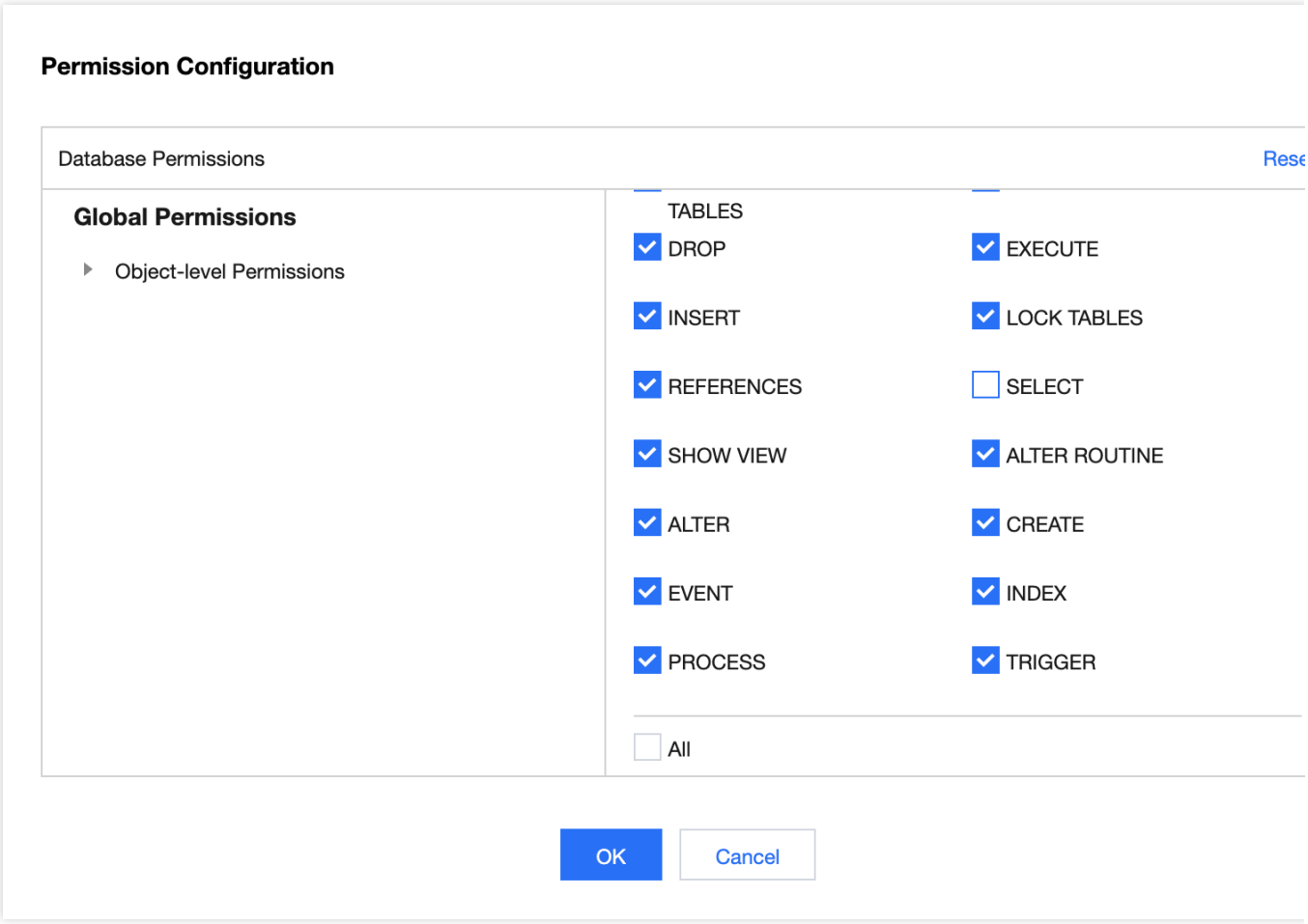
Permission configuration: Grant permissions as needed.

Create the first database secret

Create the second database secret

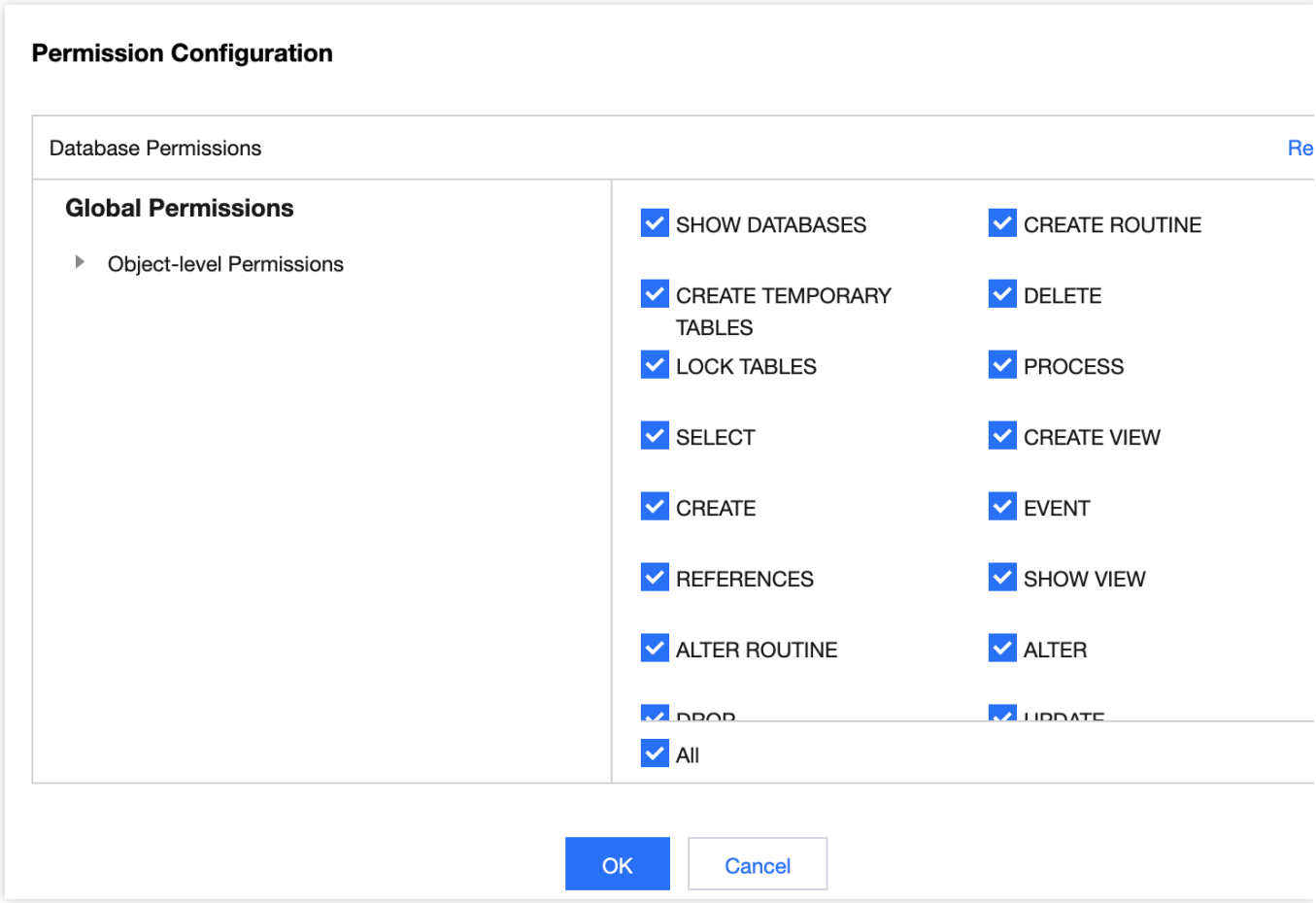
Click **Authorization**. Select the following permissions on the **Permission configuration** page.

©2013-2022 Tencent Cloud. All rights reserved.



Click **Authorization**. Select **All** on the **Permission configuration** page.

©2013-2022 Tencent Cloud. All rights reserved.

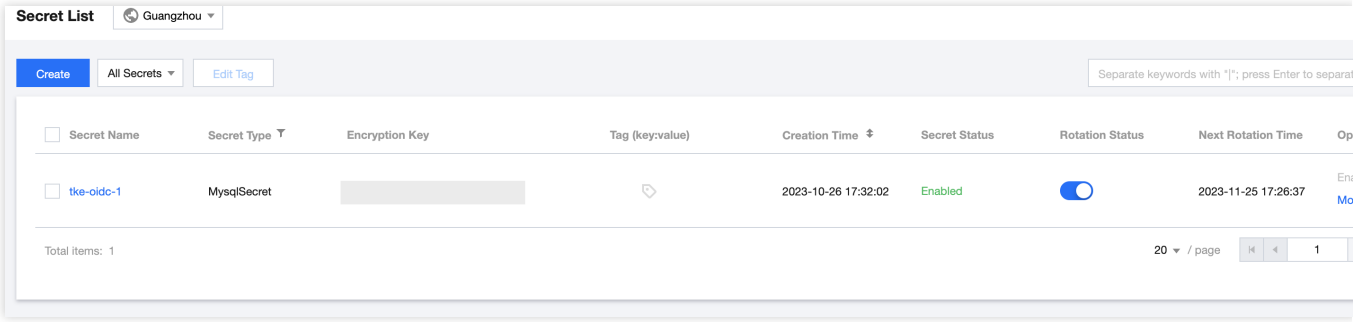


Notes

The value of **Secret name** is identified as `$ssm_name` .

The value of **Secret region** is identified as `$ssm_region_name` .

3. Click **Create**. View the created secrets on the secret list page.



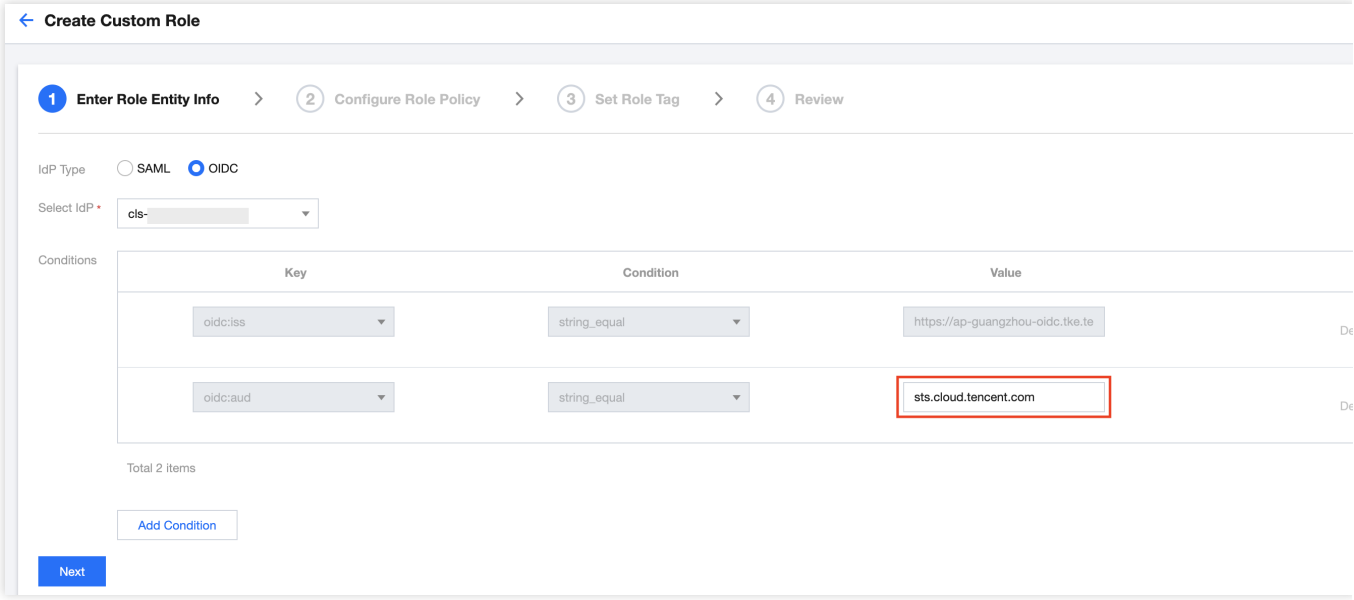
Step 9. Create a CAM role and associate with the access policies

1. Log in to the CAM console.

2. On the Role page, click **Create role > Identity provider**.

3. On the **Create custom role** page, complete the configuration with the following information.

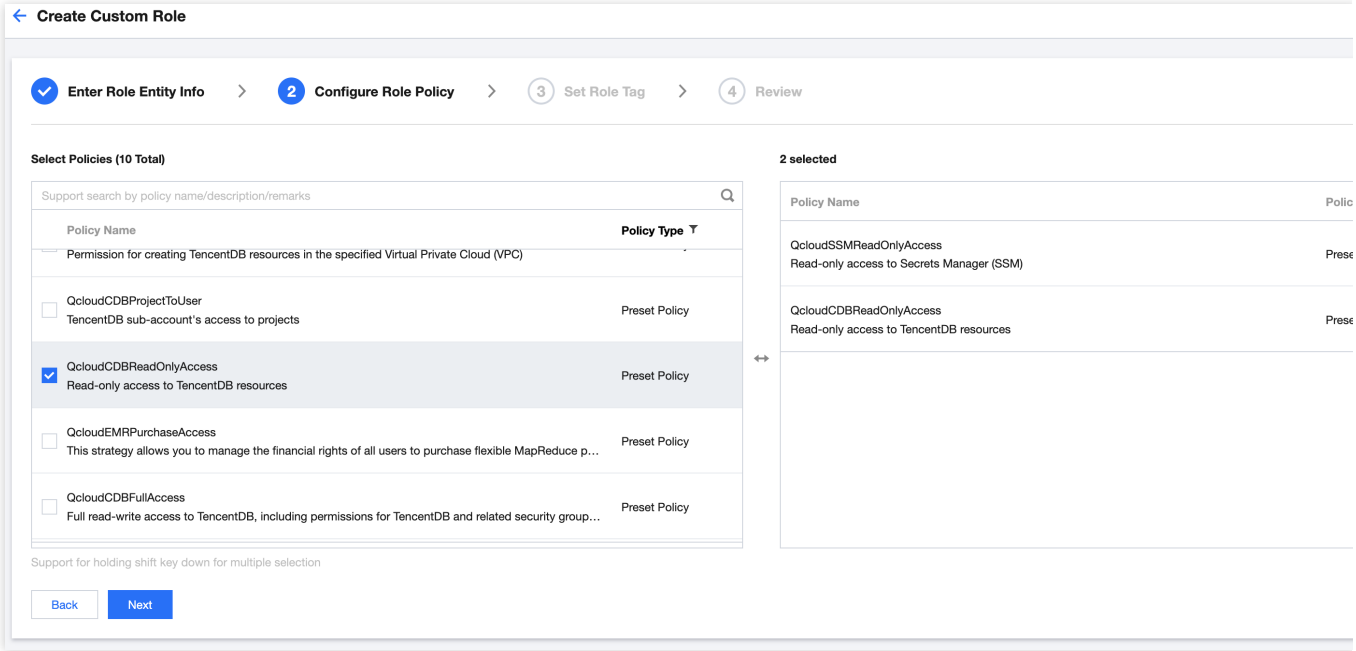
©2013-2022 Tencent Cloud. All rights reserved.



Notes

The value of **odic:aud** must be consistent with the value of **Client ID** of the CAM OIDC provider.

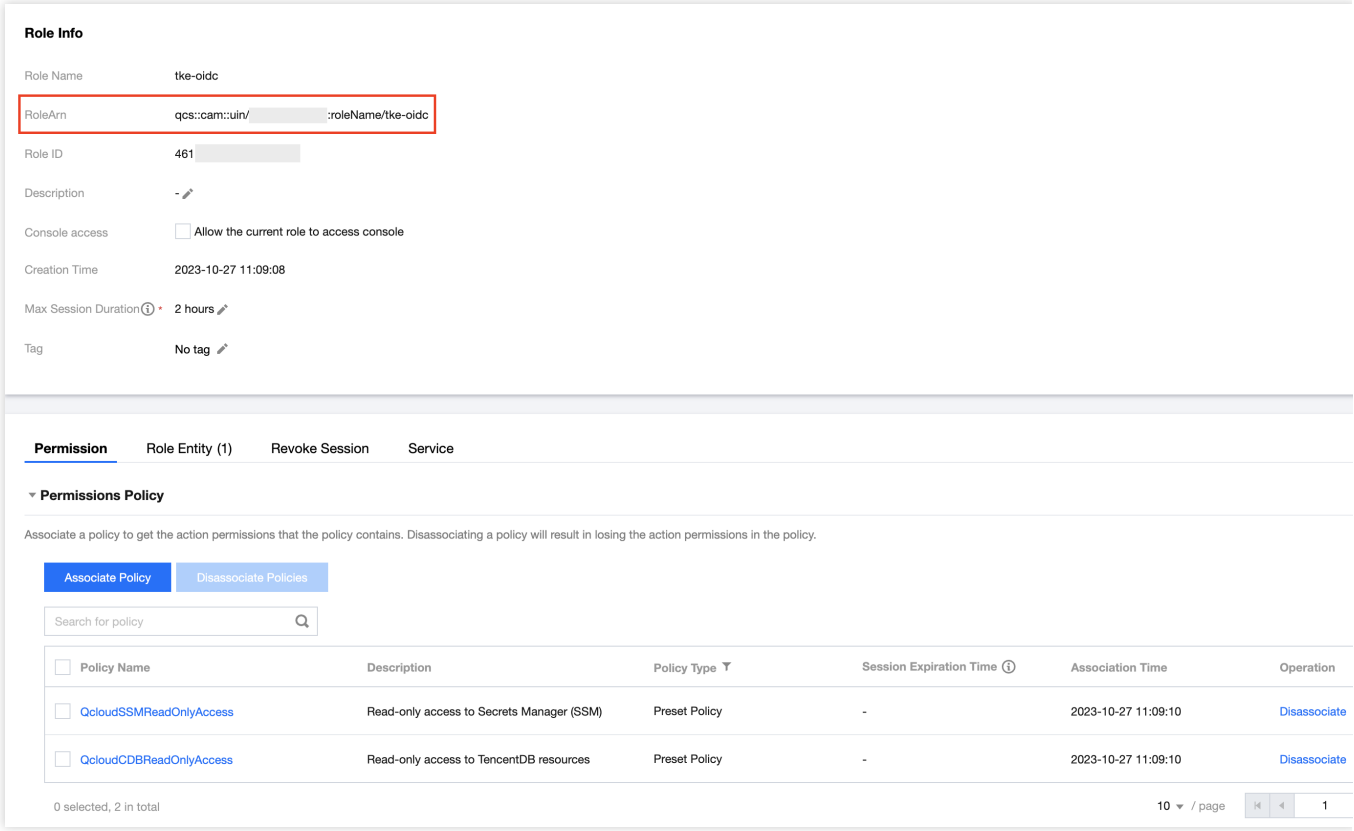
The value of **odic:aud** is identified as `$my_pod_audience` . When multiple values are available to **odic:aud**, select any one of them.



Notes

You can select an existing custom policy or create a new one. In this example, we use QcloudSSMReadOnlyAccess and QcloudCDBReadOnlyAccess

©2013-2022 Tencent Cloud. All rights reserved.



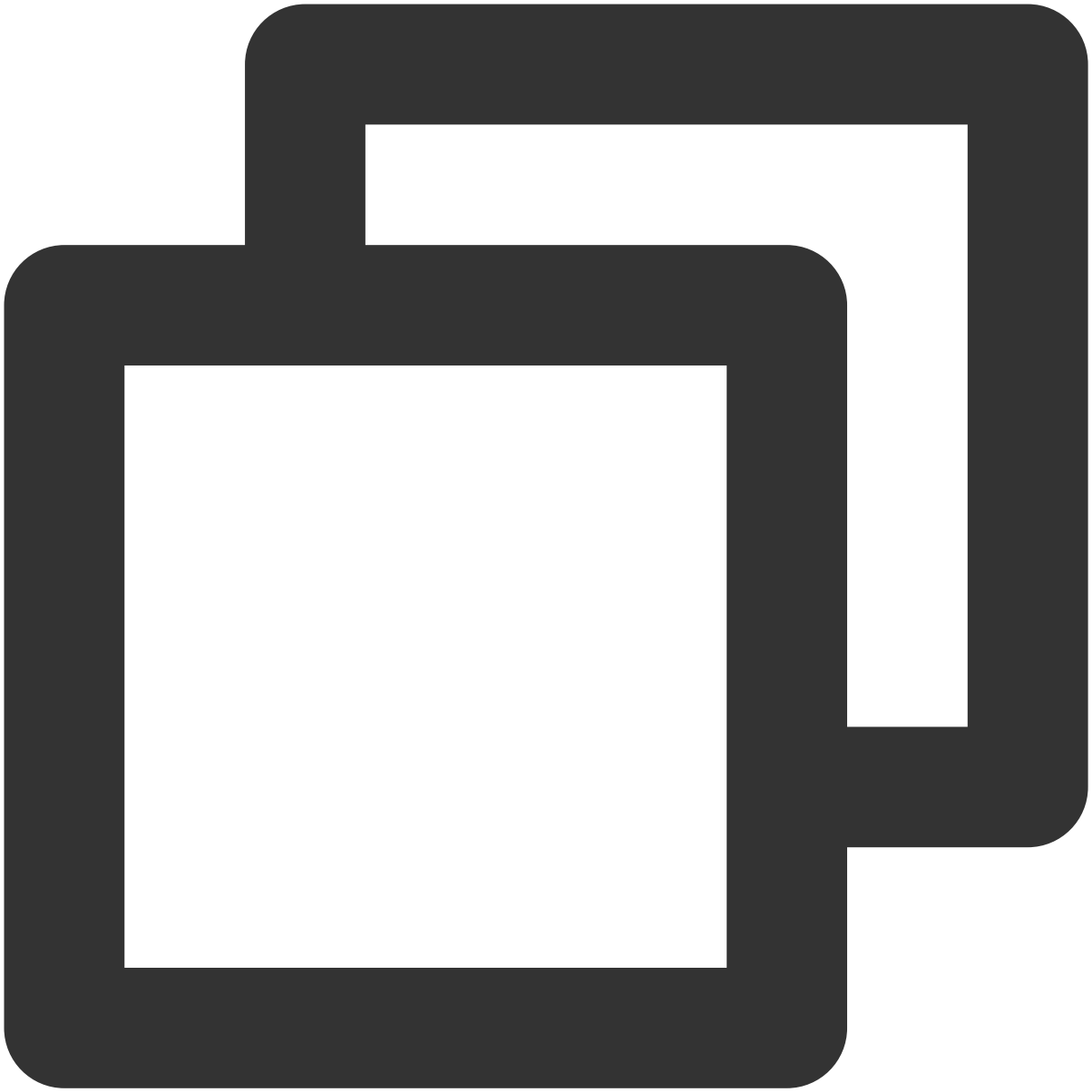
Notes

The value of **RoleArn** is identified as `$my_pod_role_arn` .

Step 10. Deploy the sample application

1. Create a Kubernetes namespace to deploy resources.

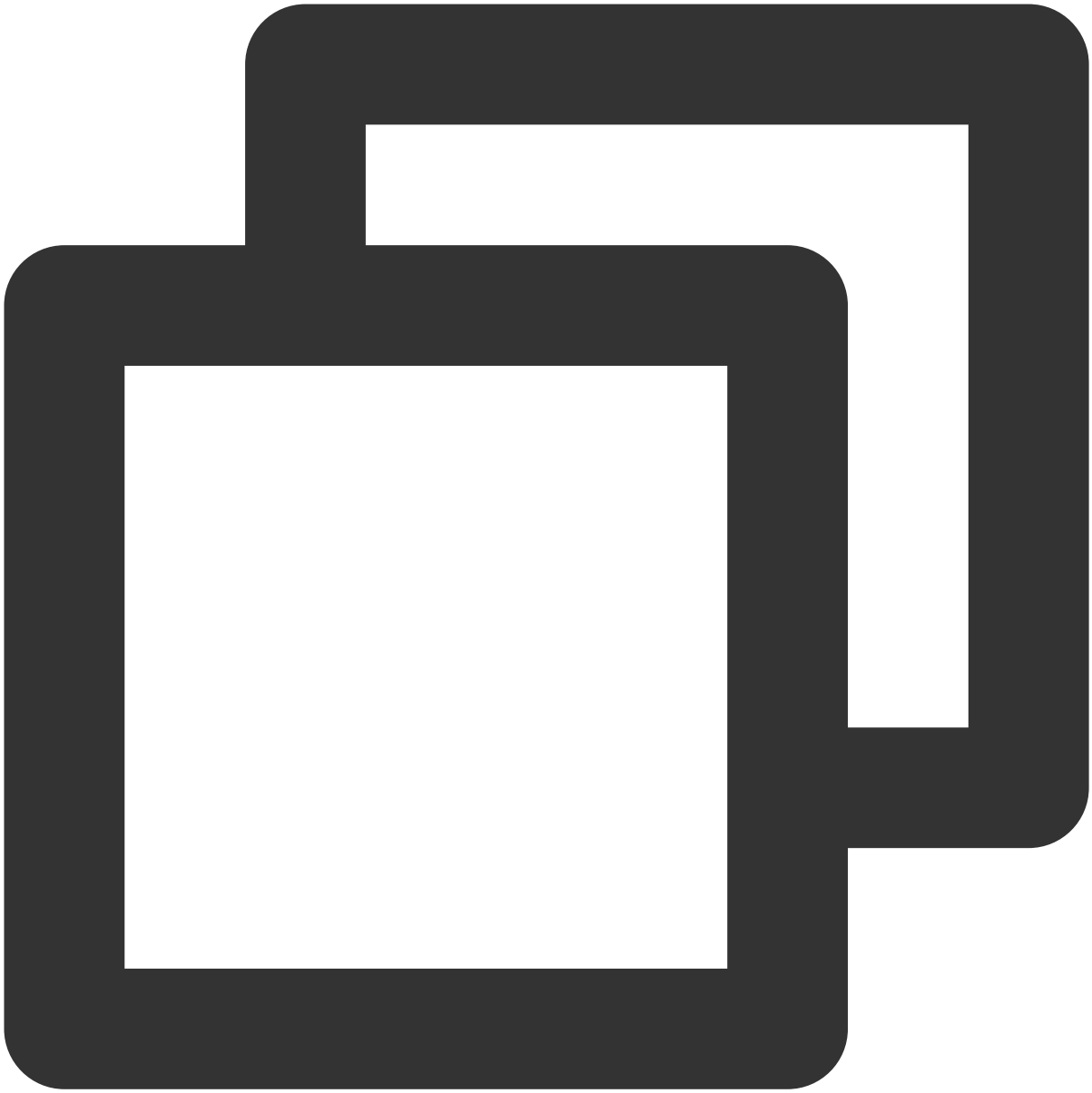
©2013-2022 Tencent Cloud. All rights reserved.



```
kubectl create namespace my-namespace
```

2. Save the following contents to **my-serviceaccount.yaml**. Replace `$my_pod_role_arn` with the value of RoleArn, and replace `$my_pod_audience` with the value of odic:aud.

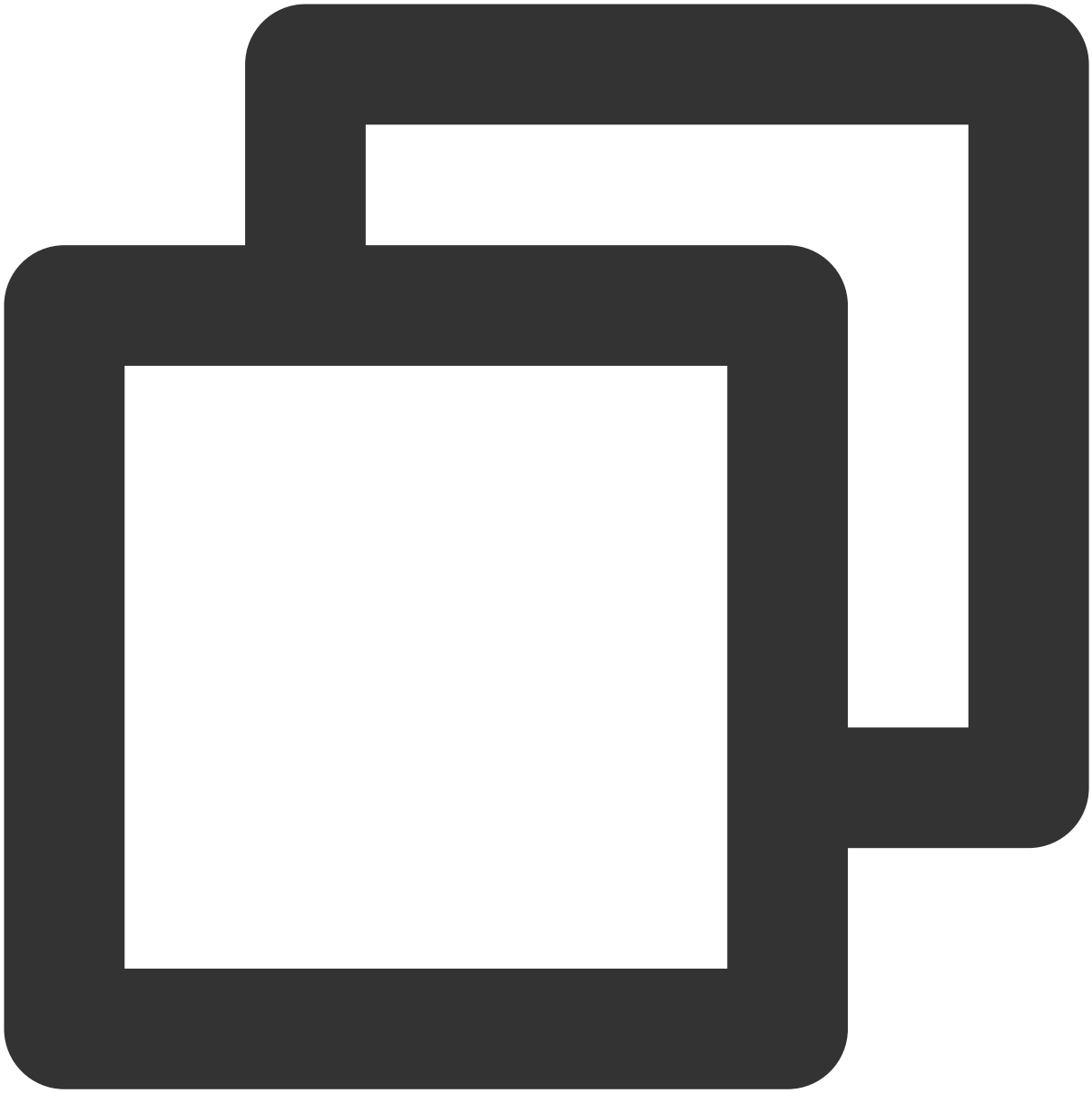
©2013-2022 Tencent Cloud. All rights reserved.



```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: my-serviceaccount
  namespace: my-namespace
  annotations:
    tke.cloud.tencent.com/role-arn: $my_pod_role_arn
    tke.cloud.tencent.com/audience: $my_pod_audience
    tke.cloud.tencent.com/token-expiration: "86400"
```

3. Save the following contents to **sample-application.yaml**.

©2013-2022 Tencent Cloud. All rights reserved.



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  namespace: my-namespace
spec:
  selector:
    matchLabels:
      app: my-app
  replicas: 1
  template:
```

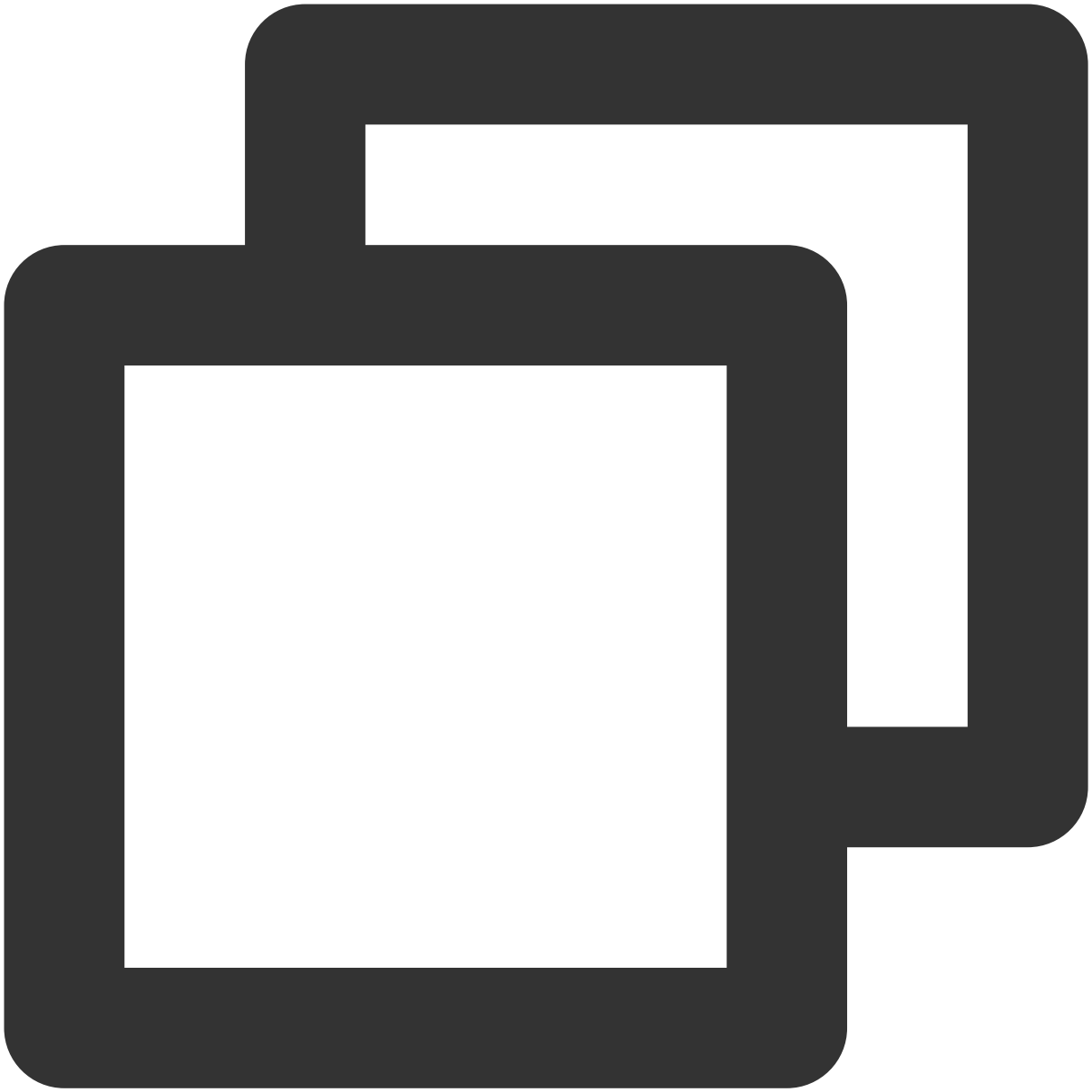
©2013-2022 Tencent Cloud. All rights reserved.

```
    metadata:
      labels:
        app: my-app
    spec:
      serviceAccountName: my-serviceaccount
      containers:
        - name: nginx
          image: $image
          ports:
            - containerPort: 80
```

Note that in this example, `ccr.ccs.tencentyun.com/tkeimages/sample-application:latest` is selected for `$image`, which integrates with the compiled demo file. You can enter it as needed.

4. Deploy the sample application.

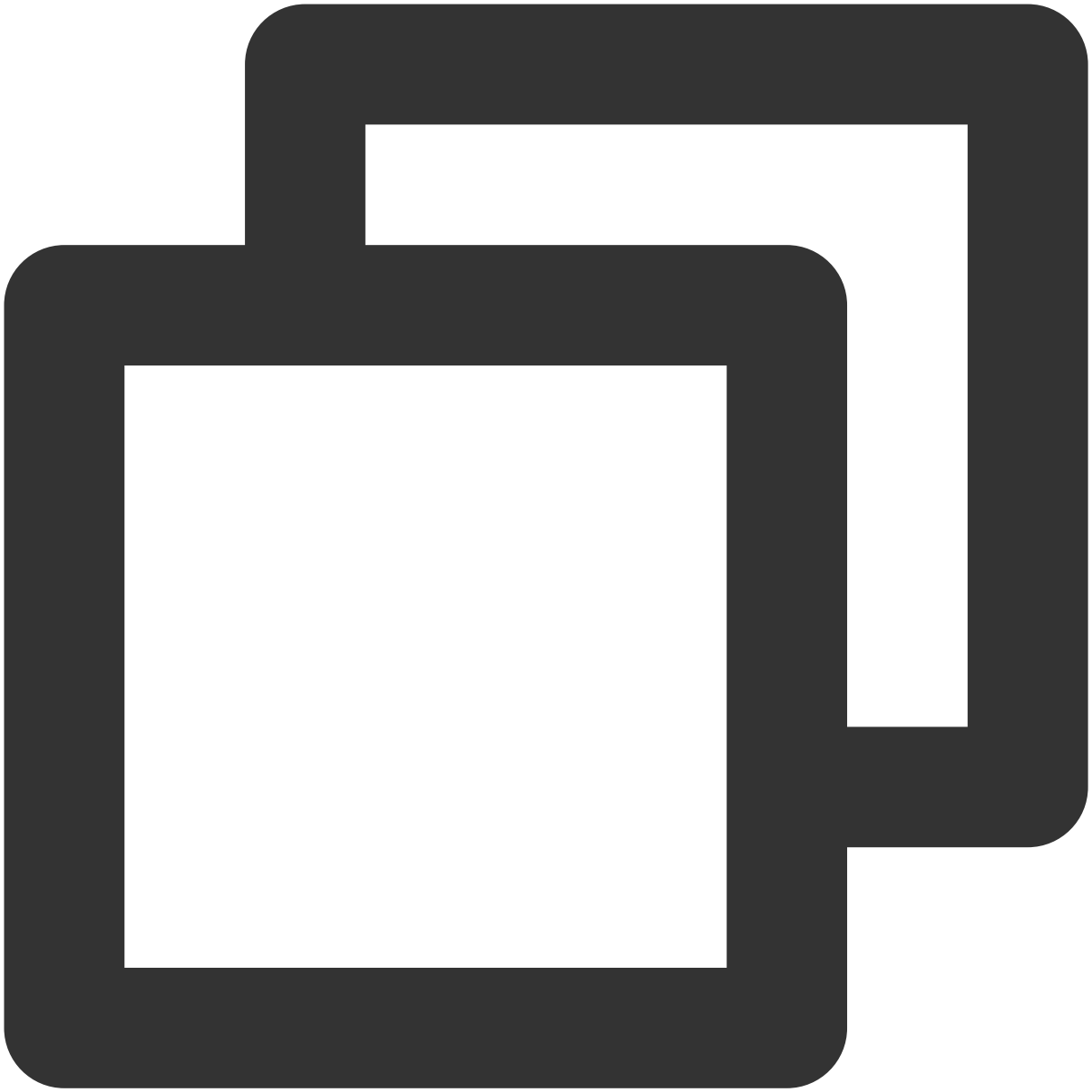
©2013-2022 Tencent Cloud. All rights reserved.



```
kubectl apply -f my-serviceaccount.yaml
kubectl apply -f sample-application.yaml
```

5. Check the Pod deploying with the sample application.

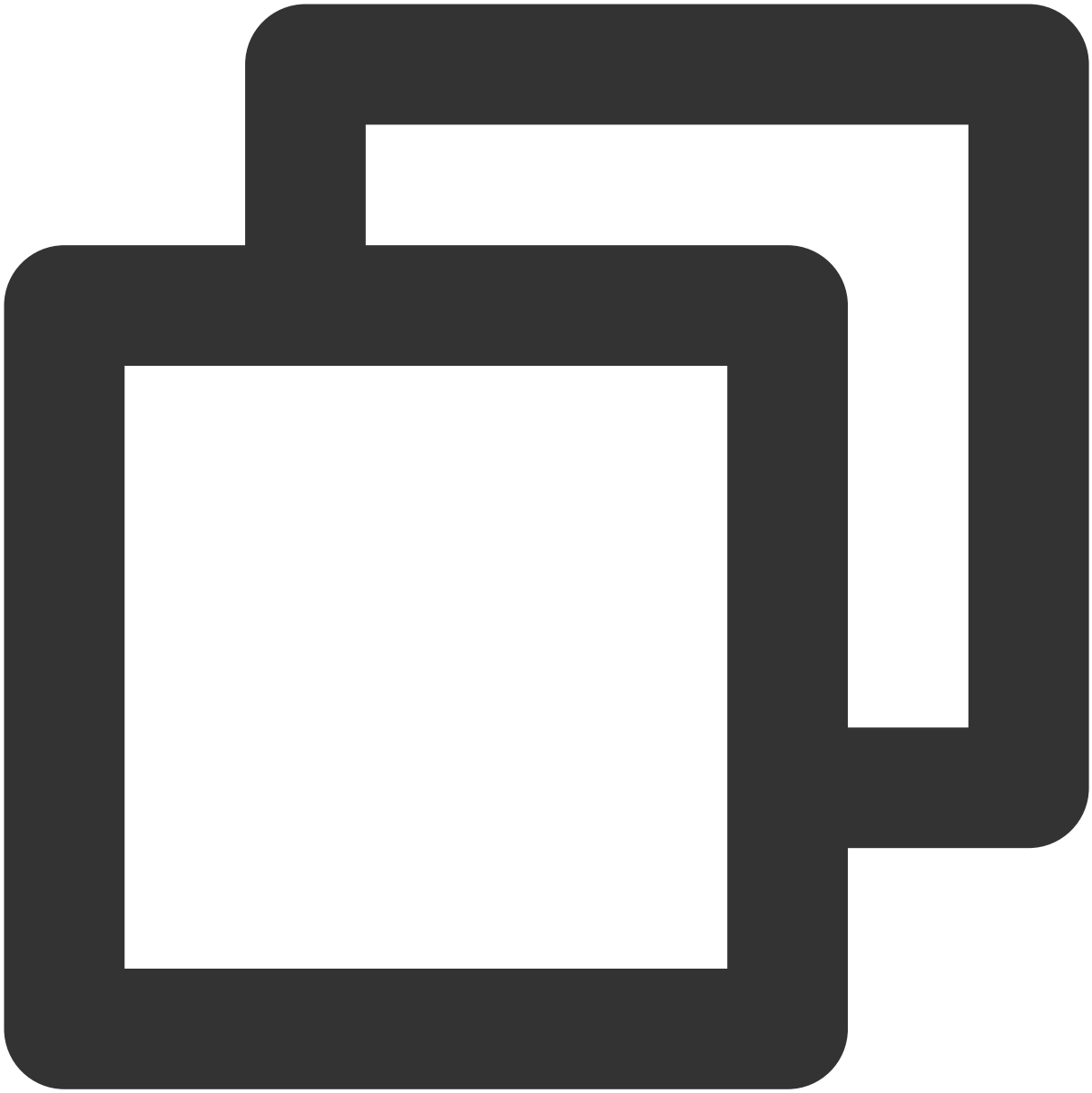
©2013-2022 Tencent Cloud. All rights reserved.



```
kubectl get pods -n my-namespace
```

Below is the sample output:

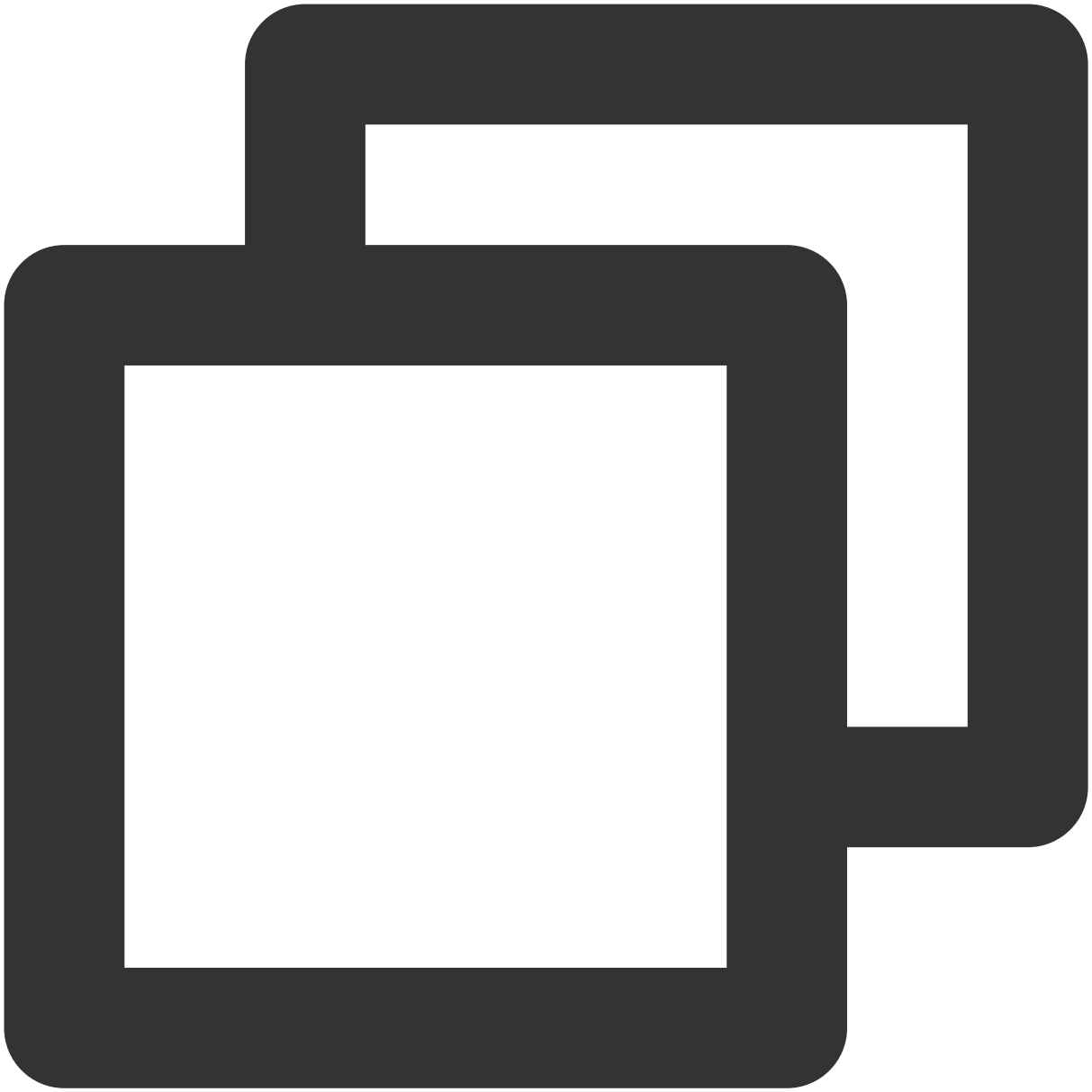
©2013-2022 Tencent Cloud. All rights reserved.



```
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-6bfd845f47-9zxld   1/1     Running   0          67s
```

6. Check workload environment variable information.

©2013-2022 Tencent Cloud. All rights reserved.



```
kubectl describe pod nginx-deployment-6bfd845f47-9zxld -n my-namespace
```

Below is the sample output:

©2013-2022 Tencent Cloud. All rights reserved.

```
[root@VM-32-127-centos ~]# kubectl describe pod nginx-deployment-6bfd845f47-9zxld -n my-namespace
Name:         nginx-deployment-6bfd845f47-9zxld
Namespace:    my-namespace
Priority:     0
Node:         10.0.32.127/10.0.32.127
Start Time:   Thu, 22 Sep 2022 17:58:55 +0800
Labels:       app=nginx
              pod-template-hash=6bfd845f47
Annotations:  tke.cloud.tencent.com/networks-status:
                [{
                    "name": "tke-bridge",
                    "interface": "eth0",
                    "ips": [
                        "172.24.0.79"
                    ],
                    "mac": "66:16:9c:92:28:08",
                    "default": true,
                    "dns": {}
                }]
Status:       Running
IP:           172.24.0.79
IPs:
  IP:           172.24.0.79
Controlled By:  ReplicaSet/nginx-deployment-6bfd845f47
Containers:
  nginx:
    Container ID:   docker://7b2cfb15cc4fe9b262c24e6fef45191c7628e3765513cb60f68d37332f3afd81
    Image:          ccr.ccs.tencentyun.com/alantlliu/nginx:latest
    Image ID:       docker-pullable://ccr.ccs.tencentyun.com/alantlliu/nginx@sha256:30e1d52127fe952b044d5606952eaa07fb20536cf195620497
    Port:           80/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Thu, 22 Sep 2022 17:58:57 +0800
    Ready:          True
    Restart Count:  0
    Environment:
      TKE_DEFAULT_REGION:           ap-guangzhou
      TKE_REGION:                   ap-guangzhou
      TKE_PROVIDER_ID:              cls-37hhvfem
      TKE_ROLE_ARN:                 qcs::cam::uin/3321337994:roleName/tke-oidc
      TKE_WEB_IDENTITY_TOKEN_FILE:  /var/run/secrets/cloud.tencent.com/serviceaccount/token
    Mounts:
      /var/run/secrets/cloud.tencent.com/serviceaccount from tke-cam-token (ro)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-7vdtm (ro)
```

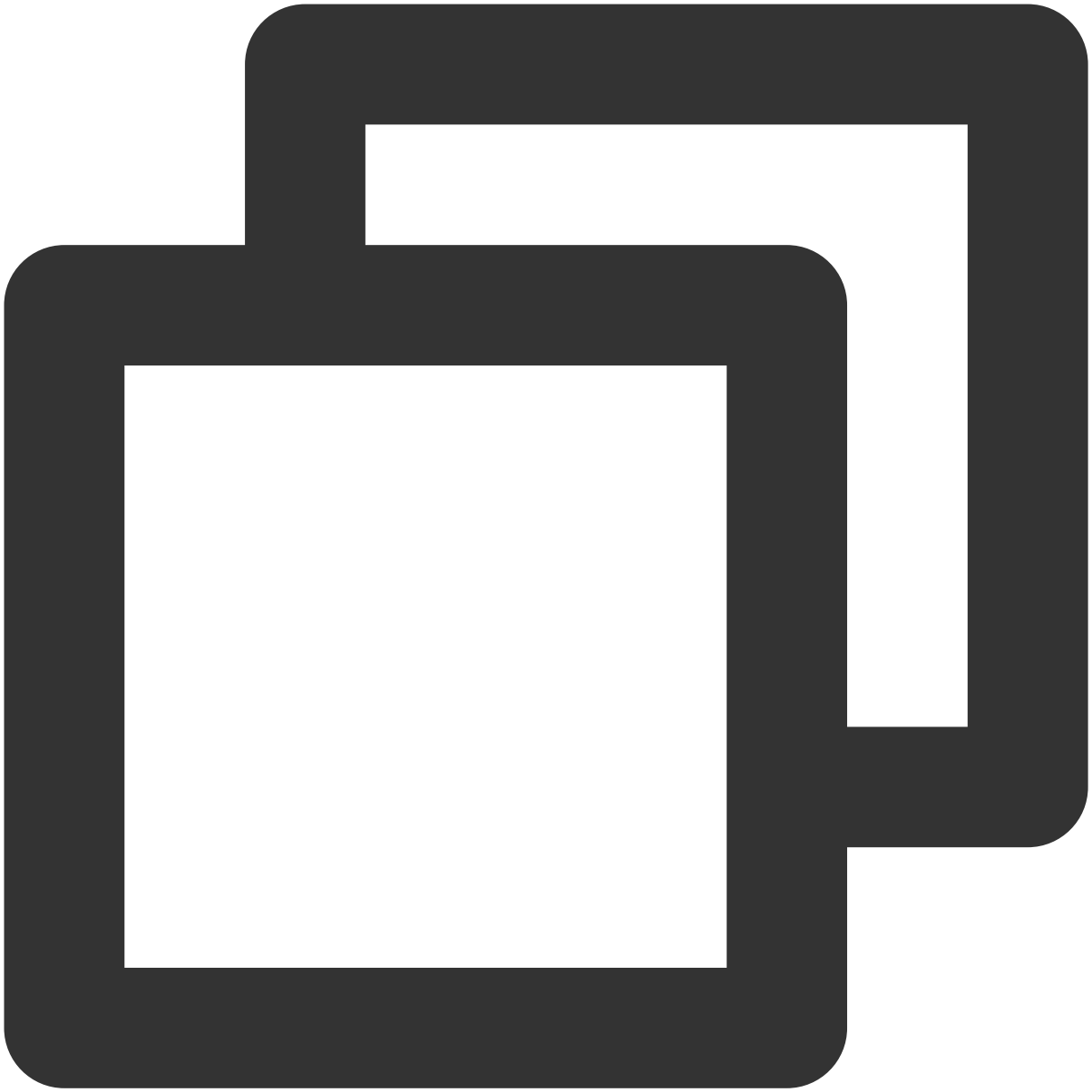
Step 11. Access the pseudo-code implementation of the database demo

1. Confirm that the sub-account has permission to access the AssumeRoleWithWebIdentity API. If not, contact the admin to add the permission.

2. If the sub-account has permission to access the AssumeRoleWithWebIdentity API, obtain the temporary key to access DB + SSM with reference to step 5 in Secret Management.

3. Clone ssm-rotation-sdk-golang code.

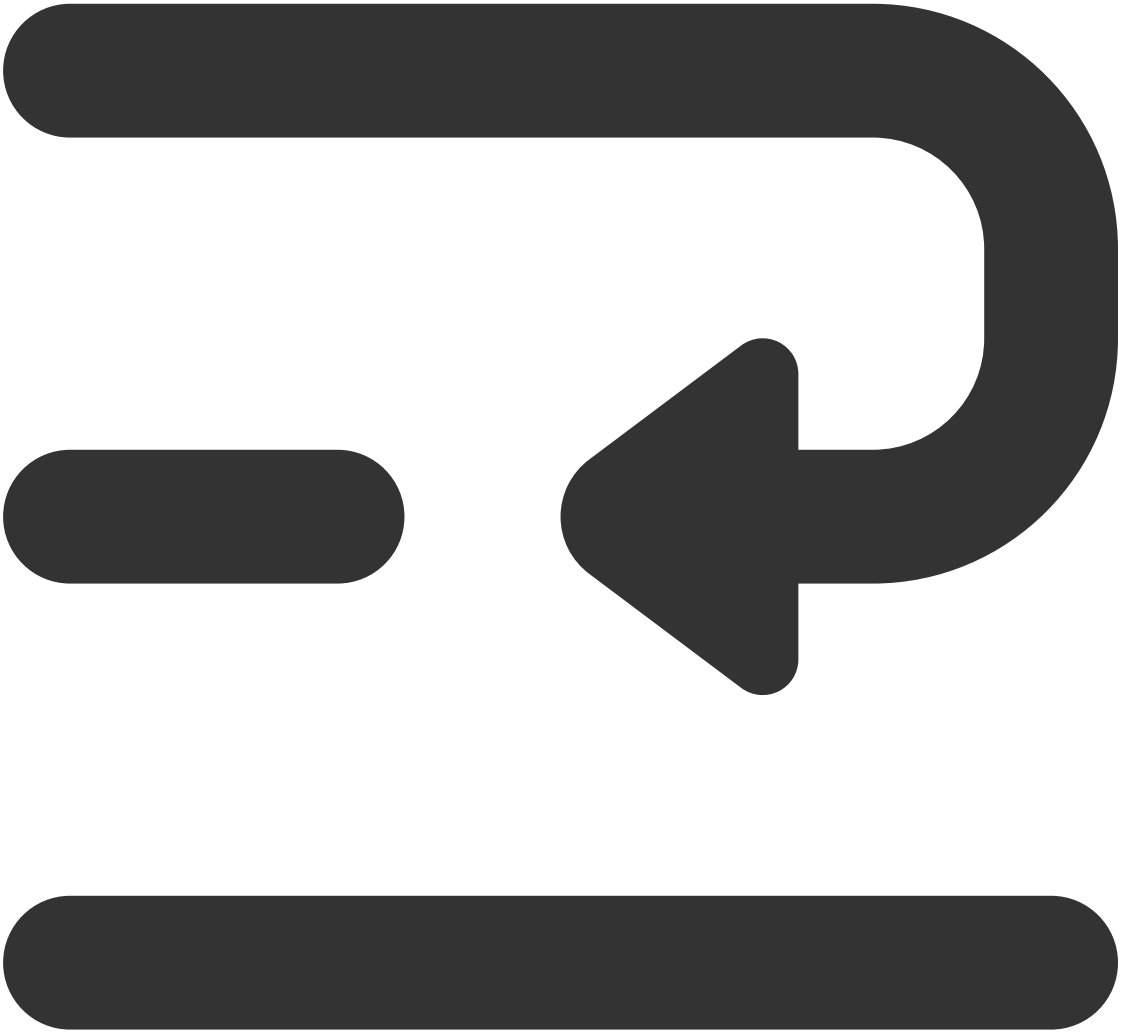
©2013-2022 Tencent Cloud. All rights reserved.



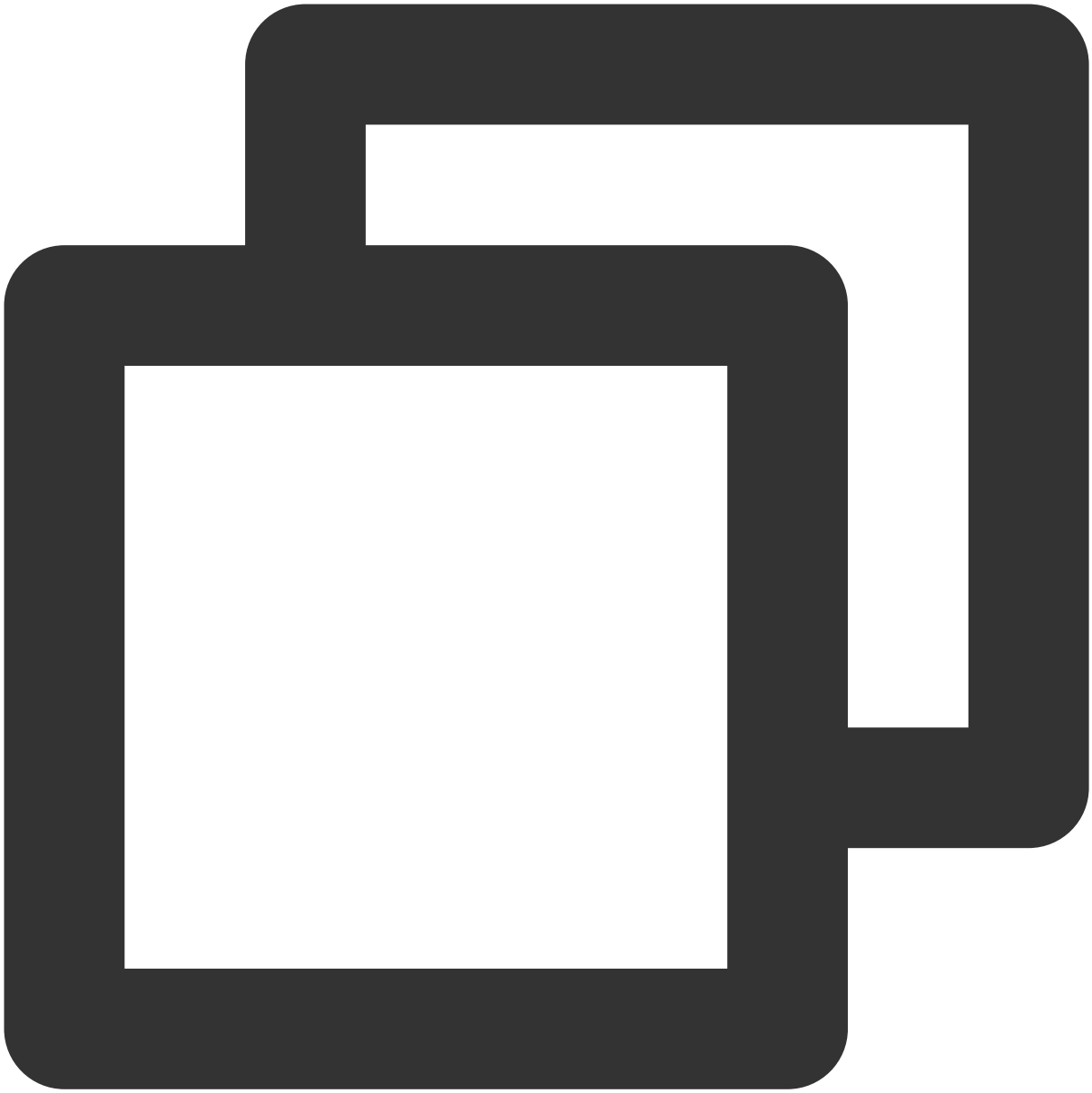
```
shell git clone https://github.com/TencentCloud/ssm-rotation-sdk-golang.git
```

4. Replace the pseudo-code implementation in the demo:

©2013-2022 Tencent Cloud. All rights reserved.



©2013-2022 Tencent Cloud. All rights reserved.



```
package main

import (
        "flag"
        "fmt"
        _ "github.com/go-sql-driver/mysql"
        "github.com/tencentcloud/ssm-rotation-sdk-golang/lib/db"
        "github.com/tencentcloud/ssm-rotation-sdk-golang/lib/ssm"
        "github.com/tencentcloud/tencentcloud-sdk-go/tencentcloud/common"
        "log"
        "time"
```

©2013-2022 Tencent Cloud. All rights reserved.

```
)

var (
        roleArn, tokenPath, providerId, regionName, saToken string
        secretName, dbAddress, dbName, ssmRegionName        string
        dbPort                                              uint64
        dbConn                                              *db.DynamicSecretRotati
        Header                                              = map[string]string{
                "Authorization":      "SKIP",
                "X-TC-Action":        "AssumeRoleWithWebIdentity",
                "Host":               "sts.internal.tencentcloudapi.com",
                "X-TC-RequestClient": "PHP_SDK",
                "X-TC-Version":       "2018-08-13",
                "X-TC-Region":        regionName,
                "X-TC-Timestamp":     "1659944952",
                "Content-type":       "application/json",
        }
)

type Credentials struct {
        TmpSecretId  string
        TmpSecretKey string
        Token        string
        ExpiredTime  uint64
}

func main() {
        flag.StringVar(&secretName, "ssmName", "", "ssm name")
        flag.StringVar(&ssmRegionName, "ssmRegionName", "", "ssm region")
        flag.StringVar(&dbAddress, "dbAddress", "", "database address")
        flag.StringVar(&dbName, "dbName", "", "database name")
        flag.Uint64Var(&dbPort, "dbPort", 0, "database port")
        flag.Parse()

        provider, err := common.DefaultTkeOIDCRoleArnProvider()
        if err != nil {
                log.Fatal("failed to assume role with web identity, err:", err)
        }
        assumeResp, err := provider.GetCredential()
        if err != nil {
                log.Fatal("failed to assume role with web identity, err:", err)
        }

        var credential Credentials
        if assumeResp != nil {
                credential = Credentials{
                        TmpSecretId:  assumeResp.GetSecretId(),
```

©2013-2022 Tencent Cloud. All rights reserved.

```
                        TmpSecretKey: assumeResp.GetSecretKey(),
                        Token:        assumeResp.GetToken(),
                }
        }
        log.Printf("secretId:%v,secretey%v,token%v\\n", credential.TmpSecretId, cre
        DB(credential)
}

func DB(credential Credentials) {
        // Initialize the database connection
        dbConn = &db.DynamicSecretRotationDb{}
        err := dbConn.Init(&db.Config{
                DbConfig: &db.DbConfig{
                        MaxOpenConns:        100,
                        MaxIdleConns:        50,
                        IdleTimeoutSeconds:  100,
                        ReadTimeoutSeconds:  5,
                        WriteTimeoutSeconds: 5,
                        SecretName:          secretName, // Secret name
                        IpAddress:           dbAddress,  // Database address
                        Port:                dbPort,     // Database port
                        DbName:              dbName,     // Leave it empty or speci
                        ParamStr:            "charset=utf8&loc=Local",
                },
                SsmServiceConfig: &ssm.SsmAccount{
                        SecretId:  credential.TmpSecretId,  // Fill in the actual a
                        SecretKey: credential.TmpSecretKey, // Fill in the actual a
                        Token:     credential.Token,
                        Region:    ssmRegionName, // Select the region where the se
                },
                WatchChangeInterval: time.Second * 10, // Interval to check the sec
        })
        if err != nil {
                fmt.Errorf("failed to init dbConn, err:%v\\n", err)
                return
        }
        // In the simulation process, you need to get a db connection to operate th
        t := time.Tick(time.Second)
        for {
                select {
                case <-t:
                        accessDb()
                        queryDb()
                }
        }
}
```

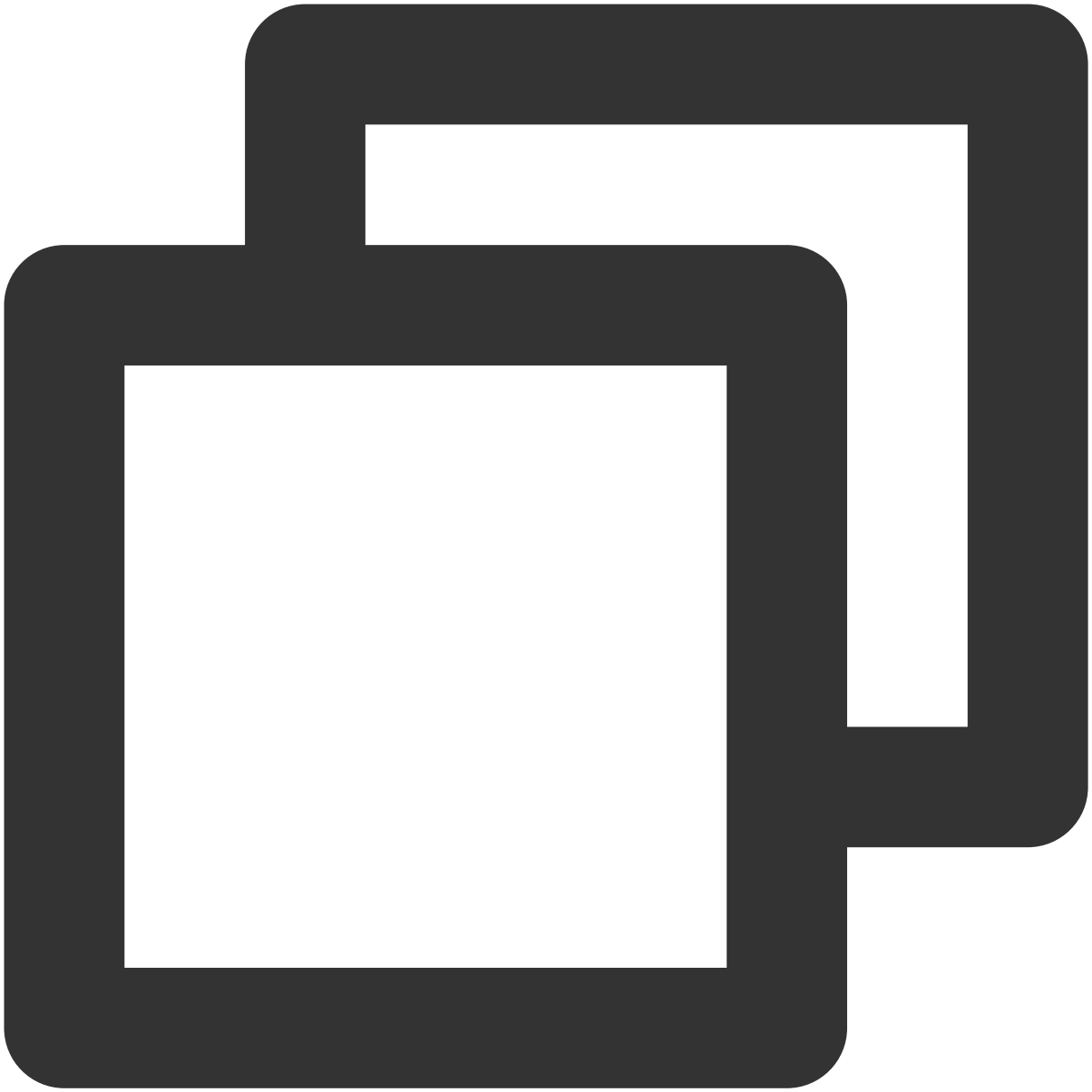
©2013-2022 Tencent Cloud. All rights reserved.

```
func accessDb() {
        fmt.Println("--- accessDb start")
        c := dbConn.GetConn()
        if err := c.Ping(); err != nil {
                log.Fatal("failed to access db with err:", err)
        }
        log.Println("--- succeed to access db")
}

func queryDb() {
        var (
                id   int
                name string
        )
        log.Println("--- queryDb start")
        c := dbConn.GetConn()
        rows, err := c.Query("select id, name from user where id = ?", 1)
        if err != nil {
                log.Printf("failed to query db with err: ", err)
                log.Fatal(err)
        }
        defer rows.Close()
        for rows.Next() {
                err := rows.Scan(&id, &name)
                if err != nil {
                        log.Fatal(err)
                }
                log.Println(id, name)
        }
        err = rows.Err()
        if err != nil {
                log.Fatal(err)
        }
        log.Println("--- succeed to query db")
}
```

Step 12. Test the demo sample

Go to the nginx container based on the result in the step of Deploy the sample.

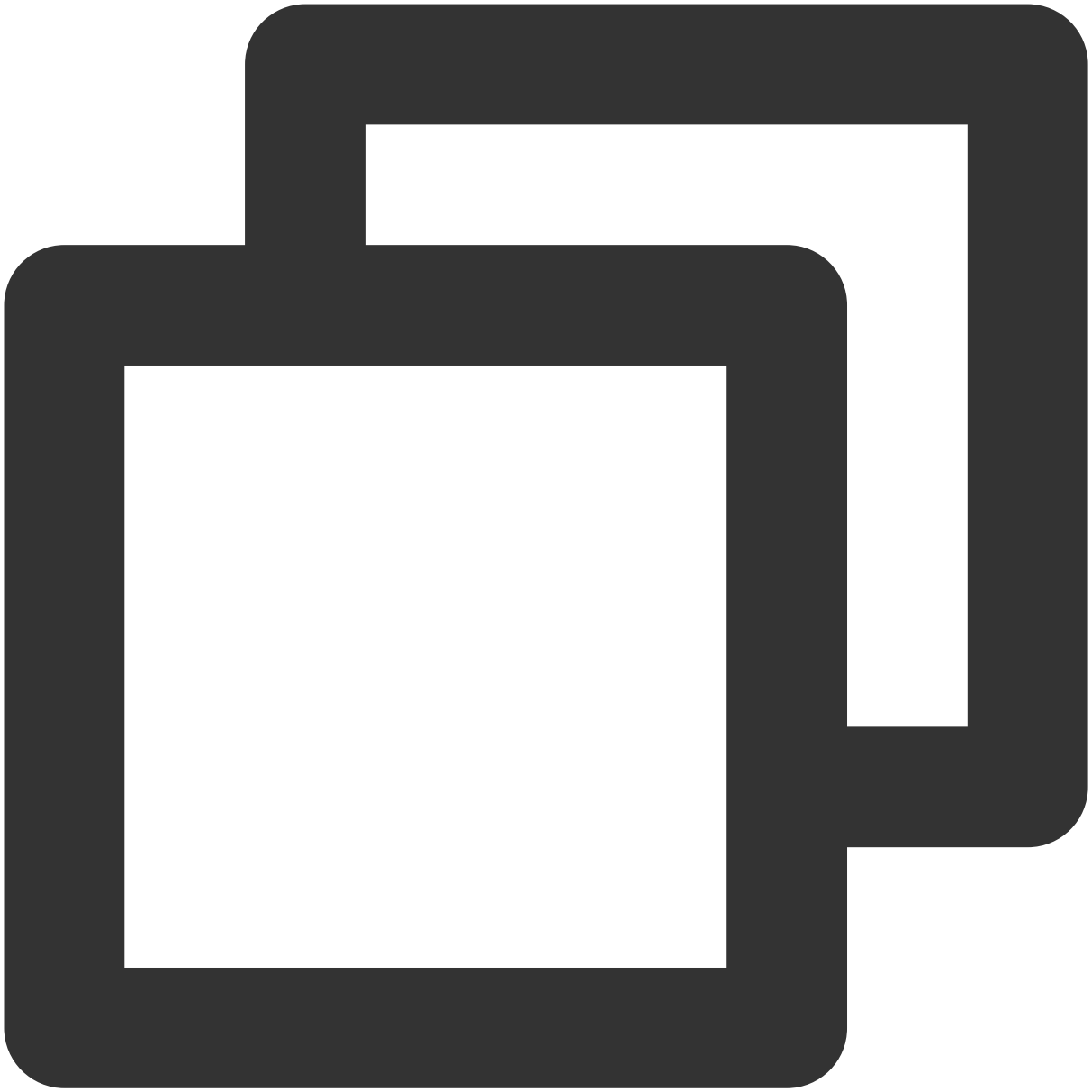
©2013-2022 Tencent Cloud. All rights reserved.



```
kubectl exec -ti  nginx-deployment-6bfd845f47-9zxld   -n my-namespace  -- /bin/bash
cd /root/
```

Replace the values of **$ssm_name** and **$ssm_region_name** according to SSM Instance. Replace the values of **$db_address**, **$db_name** and **$db_port** according to Database Instance.

©2013-2022 Tencent Cloud. All rights reserved.



```
./demo  --ssmName=$ssm_name --ssmRegionName=$ssm_region_name --dbAddress=$db_addres
```

In this example, when $ssm_name=tke-oidc-1, the "select" permission to database is not available.

©2013-2022 Tencent Cloud. All rights reserved.

In this example, when $ssm_name=tke-oidc-2, the "select" permission to database is available.

Test conclusion

The test shows that the expected effect is achieved. Validating the authentication tokens for workloads in a managed cluster through CAM ensures the security of authentication. With the rotation and encryption of database usernames and passwords by SSM, you don't need to worry about the storage and lifecycle of database secrets, and it is no need to use usernames and passwords when connecting managed clusters to the database.

pod-identity-webhook Permission Description

Permission Description

The permission of this component is the minimal dependency required for the current feature to operate.

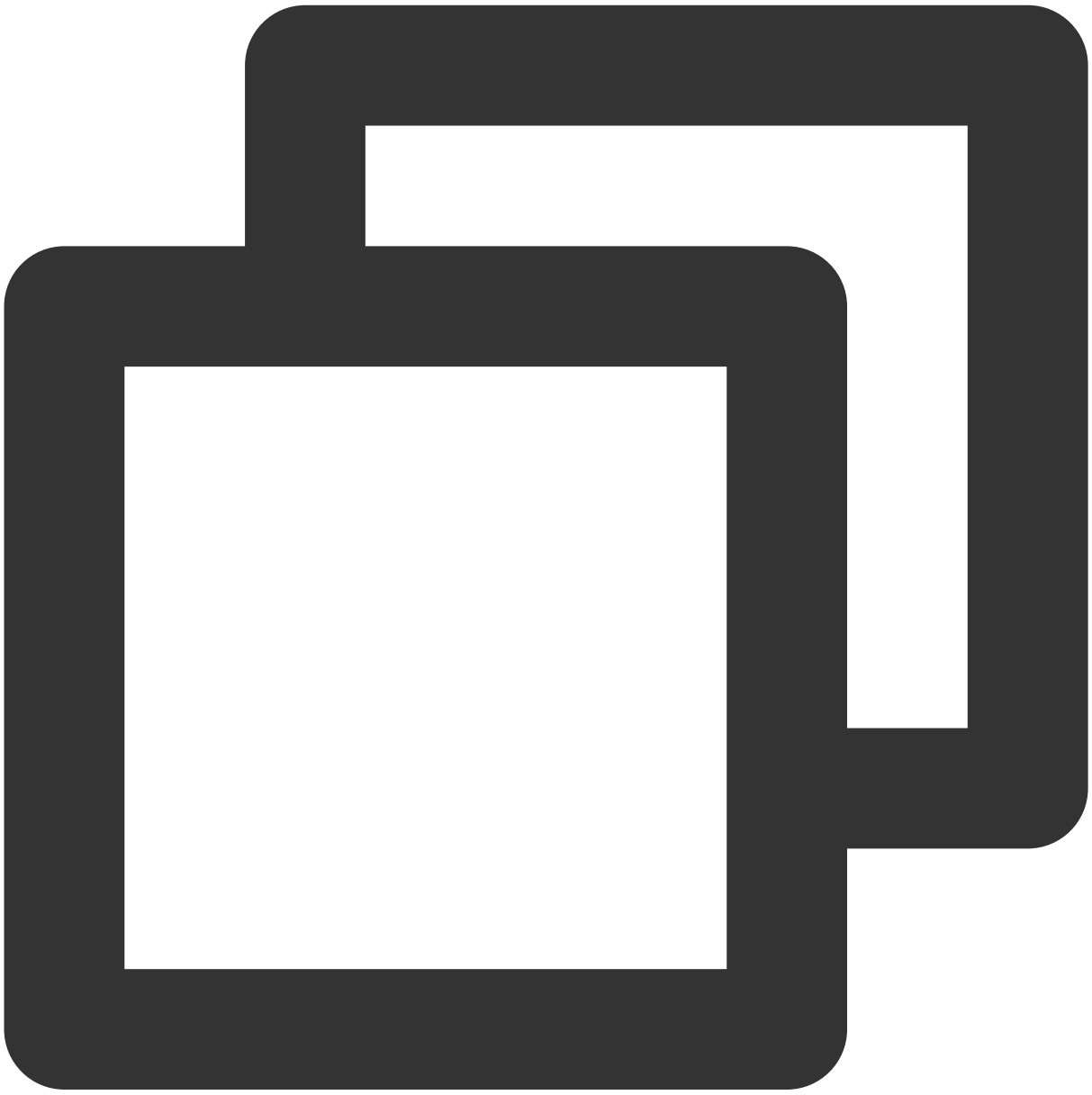
Permission Scenarios

Feature	Involved Object	Involved Operation Permission
It is required to inquire about the resource status of the specified serviceaccounts on the created pod.	serviceaccount	list/watch/get

©2013-2022 Tencent Cloud. All rights reserved.

When creating components, it is required to inject the client's certificate in the resource of mutatingwebhookconfigurations.	mutatingwebhookconfigurations	get/update

Permission Definition



```
rules:
  - apiGroups:
      - ""
    resources:
```

©2013-2022 Tencent Cloud. All rights reserved.

```
      - serviceaccounts
    verbs:
      - get
      - watch
      - list
  - apiGroups:
      - ""
    resources:
      - events
    verbs:
      - patch
      - update
  - apiGroups:
      - "admissionregistration.k8s.io"
    resources:
      - "mutatingwebhookconfigurations"
    verbs:
      - get
```

©2013-2022 Tencent Cloud. All rights reserved.

Service Deployment
Proper Use of Node Resources
Overview

Last updated：2022-04-18 10:42:27

It is easy to deploy containerized services to a Kubernetes cluster. If a service is used in a formal production environment, you need to select a solution and adjust the configuration based on the service scenario and deployment environment. For example, you need to set the container request and limit to ensure high availability of the deployed service, configure health check and auto scaling to better schedule resources, and select persistent storage and external service disclosure.

You can refer to the following documents to deploy Kubernetes services and adjust configurations based on actual requirements:

- Setting Request and Limit
- Proper Resource Allocation
- Auto Scaling

©2013-2022 Tencent Cloud. All rights reserved.

Setting Request and Limit

Last updated：2022-04-18 10:48:34

The request and limit parameters of a container need to be flexibly set based on the service type, your requirements, and the relevant scenario. This document describes how to set request and limit based on actual production experience. You can adjust your configurations based on this document.

How Request Works

The request value does not represent the size of the resources actually assigned to the container, but is a reference value provided to the scheduler. The scheduler detects the resources on each node that can be assigned (assignable node resources = total amount of node resources - sum of requests scheduled to containers in all Pods on the node) and records the assigned resources on each node (sum of requests scheduled to containers defined in all Pods on the node). If the amount of assignable node resources is smaller than the sum of requests in a Pod that needs to be scheduled, the Pod will not be scheduled to the node; otherwise, it will be scheduled to the node.

If request is not configured, the scheduler cannot perceive node resource usage to make correct scheduling decisions. As a result, scheduling may not be rational, resulting in chaotic node statuses. We recommend that you set request for all containers to enable the scheduler to perceive node resource usage and make proper scheduling decisions. In this way, node resources in a cluster can be properly allocated, and faults caused by uneven resource allocation can be prevented.

Setting Default Request and Limit Values

You can use LimitRange to set the default, minimum, and maximum request and limit values for a namespace, as shown below:

```
apiVersion: v1
kind: LimitRange
metadata:
name: mem-limit-range
namespace: test
spec:
limits:
- default:
memory: 512Mi
cpu: 500m
defaultRequest:
```

©2013-2022 Tencent Cloud. All rights reserved.

```
memory: 256Mi
cpu: 100m
type: Container
```

Setting Request and Limit Values for Important Online Applications

When node resources are insufficient, pods of low priorities will be deleted automatically to release node resources. The following lists pods with priorities in ascending order:

1. Pods with no request or limit values
2. Pods with different request and limit values
3. Pods with the same request and limit values

We recommend that you set the same request and limit values for important online applications to ensure a high pod priority. When a node fault occurs, these applications will not be affected because the pods used for these applications are generally not deleted.

Improving Resource Utilization

If a large request value is set for an application but the occupied resource amount of the application is much less than the preset value, resource utilization of the node is low.

Except for services that are sensitive to latency, we recommend that you lower the request value for non-core applications that do always need resources in order to improve resource utilization. Services that are sensitive to latency do not expect high node resource utilization because it affects the packet sending and receiving speeds. If your service supports horizontal scale-out, the request value for a single replica is usually set to less than one core, except for CPU-intensive applications. For example, the request value of CoreDNS can be set to 0.1 core, which indicates 100 MB.

Preventing Large Request and Limit Values

If your service uses a single replica or a few replicas and the request and limit values are large, sufficient resources will be allocated to your service. However, when a replica encounters a fault, your service will be greatly affected. When the node where the pod resides is faulty, other nodes do not have sufficient resources to meet the pod request because the request value is large and cluster resources are allocated in a fragmented manner. As a result, the pod cannot be shifted or recovered.

©2013-2022 Tencent Cloud. All rights reserved.

We recommend that you set small request and limit values and scale out replicas to ensure that your service is more flexible and reliable.

Preventing High Resource Consumption by the Test Namespace

If a production cluster contains a test namespace and the request and limit values of the namespace are not restricted, the cluster may be overloaded and production services could be affected. You can use `ResourceQuota` to restrict the request and limit values of the test namespace, as shown below:

```
apiVersion: v1
kind: ResourceQuota
metadata:
name: quota-test
namespace: test
spec:
hard:
requests.cpu: "1"
requests.memory: 1Gi
limits.cpu: "2"
limits.memory: 2Gi
```

©2013-2022 Tencent Cloud. All rights reserved.

Proper Resource Allocation

Last updated：2020-07-31 15:44:03

You can set request to schedule pods to nodes with sufficient resources but cannot ensure refined control. This document describes how to use node affinity, taint, and toleration to schedule pods to suitable nodes in order to make full use of resources.

Using node affinity

- You can use node affinity to deploy services that have special node requirements to nodes that meet these requirements. For example, you can enable MySQL to schedule a model with high I/O to improve data reading and writing efficiency.
- You can use node affinity to deploy services that need to be associated. For example, you can ensure the web service and Redis cache service are deployed in the same availability zone to ensure a low latency.
- You can use node affinity to schedule separated pods to prevent issues caused by a single point of failure (SPOF) or centralized traffic.

Using taint and toleration

Taint and toleration can help optimize cluster resource scheduling.

- You can add taints to nodes reserved for certain applications, which prevents other pods from being scheduled to these nodes.
- You can add tolerations to pods that need to use reserved resources. Tolerations work with node affinity, to ensure pods can be scheduled even when their affinity settings cannot be matched.

©2013-2022 Tencent Cloud. All rights reserved.

Auto Scaling

Last updated：2020-07-31 15:44:03

This document describes how to use auto scaling, so that services can make full use of available resources based on actual production experience. You can adjust your configurations based on this document.

Coping Abrupt Traffic Spikes

Typically, services have peak and off-peak hours of resource usage. To properly use resources, you can define a Horizontal Pod Autoscaler (HPA) for services to automatically scale out the number of pods during peak hours and scale in the number of pods during off-peak hours. For example, when the traffic of online services is low at night, the HPA can automatically release resources of online services and use them for big data offline tasks.

To use the HPA, you need to install resource metrics (metrics.k8s.io) or custom metrics (custom.metrics.k8s.io) in advance. The HPA controller can then query related APIs to obtain resource use information for services. In this way, Kubernetes obtains resource usage data (metric data) of services in advance.
Previously, the HPA used resource metrics to obtain metric data. After custom metrics became available, the HPA used more flexible metrics to control scaling. To implement HPA, Kubernetes uses metrics-server, communities use prometheus-adapter, and cloud vendors that manage Kubernetes clusters usually use their own APIs. For example, TKE uses HPA to implement CPU, memory, hard disk, and network metrics. You can create an HPA on the web client and convert the metrics to a Kubernetes YAML file, as shown below:

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
name: nginx
spec:
scaleTargetRef:
apiVersion: apps/v1beta2
kind: Deployment
name: nginx
minReplicas: 1
maxReplicas: 10
metrics:
- type: Pods
pods:
metric:
name: k8s_pod_rate_cpu_core_used_request
target:
```

©2013-2022 Tencent Cloud. All rights reserved.

```
  averageValue: "100"
  type: AverageValue
```

Reducing costs

HPA implements horizontal pod scaling. When node resources are insufficient, scaled-out pods are in the pending state. If a large number of nodes are prepared in advance, pending pods will not occur, but the cost will be high. Typically, Kubernetes clusters managed by cloud vendors support cluster-autoscaler. This means nodes can be dynamically added or deleted based on resource usage to maximize computing resource utilization. In addition, pay-as-you-go is used to reduce the cost. For example, TKE uses scaling groups and extended features that contain scaling groups (node pools).

Using vertical scaling

For applications that do not support horizontal scaling or applications with uncertain optimal request and limit ratios, you can use VPA for vertical scaling. In this case, the request and limit values are automatically updated, and pods are restarted. This feature may cause service unavailability for a short period. We do not recommend you use it on a large scale in the production environment.

©2013-2022 Tencent Cloud. All rights reserved.

Application High Availability Deployment

Last updated：2022-08-02 17:19:05

High availability (HA) refers to the ability of an application system to maintain uninterrupted operation, which is usually achieved by improving the fault tolerance of the system. In general, the application fault tolerance can be improved by configuring `replicas` to create multiple replicas of the application, but this does not necessarily mean that the application will have high availability.
This document describes best practices for deploying application high availability. You can choose from them based on your situation.

- Distributing and scheduling business workloads
- Using a placement group to achieve disaster recovery in the physical layer
- Using PodDisruptionBudget to avoid service unavailability caused by node draining
- Using preStopHook and readinessProbe to ensure smooth and uninterrupted service update

Distributing and Scheduling Business Workloads

1. Using anti-affinity to prevent single-point failures

Kubernetes assumes that nodes are unreliable, so the more nodes there are, the higher the probability of nodes being unavailable due to software or hardware failures will be. Therefore, we usually have to deploy multiple replicas of applications and adjust the `replicas` value based on the actual situation. If its value is 1, there must be risks of single-point failures. Even if its value is greater than 1 but all replicas are scheduled to the same node, the single-point failure risks will still be there.

To prevent single-point failures, we need to have an appropriate number of replicas, and we also need to make sure different replicas are scheduled to different nodes. We can do so with anti-affinity. See the example below:

```
affinity:
podAntiAffinity:
requiredDuringSchedulingIgnoredDuringExecution:
- weight: 100
labelSelector:
matchExpressions:
- key: k8s-app
operator: In
values:
- kube-dns
topologyKey: kubernetes.io/hostname
```

©2013-2022 Tencent Cloud. All rights reserved.

The relevant configurations in this example are shown below:

- **requiredDuringSchedulingIgnoredDuringExecution**
  This sets anti-affinity as a required condition that must be met when Pods are scheduled. If no node meets the condition, Pods will not be scheduled to any node (pending).
  If you do not want to set anti-affinity as a required condition, you can use `preferredDuringSchedulingIgnoredDuringExecution` to instruct the scheduler to always try to meet the anti-affinity condition. If no node meets the condition, Pods can still be scheduled to certain nodes.
- **labelSelector.matchExpressions**
  This marks the keys and values of the labels in the service's corresponding Pod.
- **topologyKey**
  This example uses `kubernetes.io/hostname` to indicate that Pods are prevented from being scheduled to the same node.
  If you have higher requirements, such as preventing Pods from being scheduled to nodes in the same availability zone to achieve remote multi-site active-active disaster tolerance, you can use `failure-domain.beta.kubernetes.io/zone` . Generally, all the nodes in the same cluster are in one region. If there are cross-region nodes, there will be considerable latency even if direct connect is used. If Pods have to be scheduled to nodes in the same region, you can use `failure-domain.beta.kubernetes.io/region` .

2. Using topologySpreadConstraints

The topologySpreadConstraints feature defaults to be enabled in K8s v1.18. It is recommended that you use `topologySpreadConstraints` to distribute Pods in clusters of v1.18 or later versions to improve the service availability.

Widely distribute and schedule Pods to each node:
For example, widely distribute and schedule all Pods of nginx to different nodes as evenly as possible. The max allowed number variance of nginx copies on different nodes is `1` . If no more Pods can be scheduled to a node due to reasons such as insufficient resources of the node, the remaining nginx copies are pending.

```
apiVersion: apps/v1
kind: Deployment
metadata:
labels:
k8s-app: nginx
qcloud-app: nginx
name: nginx
namespace: default
spec:
replicas: 1
selector:
matchLabels:
```

©2013-2022 Tencent Cloud. All rights reserved.

```
k8s-app: nginx
qcloud-app: nginx
template:
metadata:
labels:
k8s-app: nginx
qcloud-app: nginx
spec:
topologySpreadConstraints:
- maxSkew: 1
whenUnsatisfiable: DoNotSchedule
topologyKey: topology.kubernetes.io/region
labelSelector:
matchLabels:
k8s-app: nginx
containers:
- image: nginx
name: nginx
resources:
limits:
cpu: 500m
memory: 1Gi
requests:
cpu: 250m
memory: 256Mi
dnsPolicy: ClusterFirst
```

- topologyKey: It is similar to configurations in podAntiAffinity.
- labelSelector: It is similar to configurations in podAntiAffinity. It supports selecting labels of multiple Pods.
- maxSkew: It must be an integer larger than 0, indicating the max allowed variation of Pod number in different topological domain. `1` means the max allowed variation of Pod number is one.
- whenUnsatisfiable: It indicates how to deal with the situations where the conditions are not met.
  `DoNotSchedule` means do not schedule (keep pending), and it is similar to strong anti-affinity.
  `ScheduleAnyway` means widely distribute and schedule Pods on node as evenly as possible, and it is similar to weak anti-affinity (change `DoNotSchedule` to `ScheduleAnyway`).

```
spec:
topologySpreadConstraints:
- maxSkew: 1
whenUnsatisfiable: ScheduleAnyway
topologyKey: topology.kubernetes.io/region
labelSelector:
matchLabels:
k8s-app: nginx
```

©2013-2022 Tencent Cloud. All rights reserved.

If the cluster node supports cross-AZ scheduling, you can widely distribute and schedule Pods to the AZs as evenly as possible to achieve higher levels of high availability (change `topologyKey` to `topology.kubernetes.io/zone`).

```
spec:
topologySpreadConstraints:
- maxSkew: 1
topologyKey: topology.kubernetes.io/zone
whenUnsatisfiable: ScheduleAnyway
labelSelector:
matchLabels:
k8s-app:: nginx
```

Moreover, you can widely distribute the Pods within each AZ when you schedule the Pods to the AZs.

```
spec:
topologySpreadConstraints:
- maxSkew: 1
whenUnsatisfiable: ScheduleAnyway
topologyKey: topology.kubernetes.io/zone
labelSelector:
matchLabels:
k8s-app: nginx
- maxSkew: 1
whenUnsatisfiable: ScheduleAnyway
topologyKey: kubernetes.io/hostname
labelSelector:
matchLabels:
k8s-app: nginx
```

Using a Placement Group to Achieve Disaster Recovery in the Physical Layer

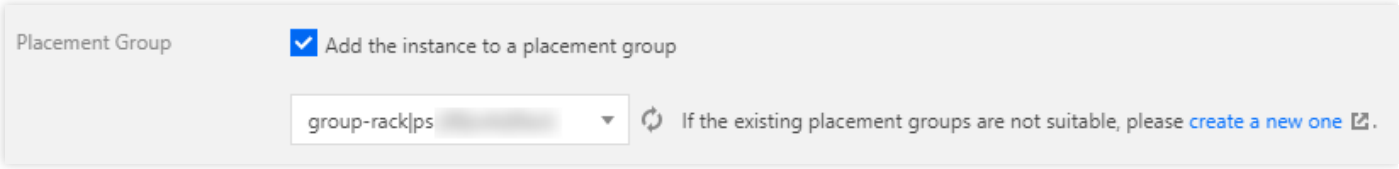
When the underlying hardware or software of a CVM is faulty, multiple nodes may have exceptions at the same time. Even if anti-affinity is used to distribute Pods to different nodes, business exceptions may still be unavoidable. You can use a placement group to distribute nodes in a physical layer, such as the CPM, exchange, or rack layer, to prevent underlying hardware or software faults from causing multiple node exceptions. The steps are as follows:

1. Log in to the Placement Group console to create a placement group and select a layer (CPM layer, exchange layer, or rack layer) as the node distribution policy. For more information, see Spread Placement Group.

©2013-2022 Tencent Cloud. All rights reserved.

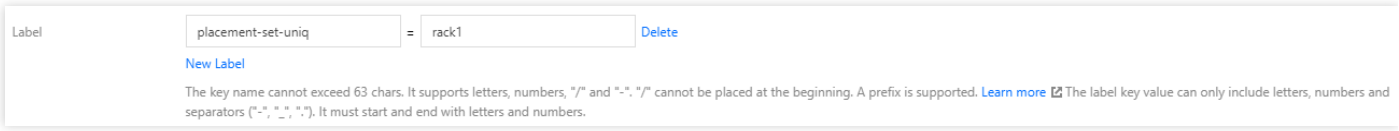
> Note：
> The placement group and the TKE self-deployed cluster need to be in the same region.

2. Add a batch of nodes, check **Add the instance to a placement group** in **Advanced configuration**, and select the created placement group. For more information, see Adding Nodes.



3. On the "Node list" page, edit the same label for this batch of nodes to mark them. These nodes are simultaneously added to the placement group as a single batch.

> Note：
> The placement group policy takes effect only for nodes of the same batch. Therefore, you need to add a label for each batch of nodes and specify different values to mark different batches.



4. Specify node affinity for Pods where workloads need to be deployed. In this way, the Pods will be deployed on the same batch of nodes. Meanwhile, specify Pod anti-affinity so that the Pods will be widely distributed among the batch of nodes. The YAML sample is as follows:

```
affinity:
nodeAffinity:
requiredDuringSchedulingIgnoredDuringExecution:
nodeSelectorTerms:
- matchExpressions:
- key: "placement-set-uniq"
operator: In
values:
- "rack1"
podAntiAffinity:
preferredDuringSchedulingIgnoredDuringExecution:
- weight: 100
podAffinityTerm:
```

©2013-2022 Tencent Cloud. All rights reserved.

```
labelSelector:
matchExpressions:
- key: app
operator: In
values:
- nginx
topologyKey: kubernetes.io/hostname
```

Using PodDisruptionBudget to Avoid Service Unavailability Caused by Node Draining

Node draining involves negative impacts. The following describes the process of draining a node:

1. Cordon the node by setting it as unschedulable to prevent new Pods from being scheduled to it.
2. Delete Pods from the node.
3. Once detecting that the number of Pods decreases, ReplicaSet controller will create a new Pod to be scheduled to a new node.

Such a process first deletes the Pods and then creates new Pods instead of using rolling update. Therefore, if all replicas of a service are on the drained node, the service may become unavailable during the updating process. Normally, the service may become unavailable for two reasons:

1. The service is exposed to single-point failure risks with all the replicas on the same node. Once the node is drained, the service may become unavailable.
   In such a case, you can refer to using anti-affinity to prevent single-point failures.
2. The service is deployed on multiple nodes, but these nodes are drained at the same time. All the replicas of the service are deleted simultaneously, which may cause the service to become unavailable.
   In such a case, you can configure PDB (PodDisruptionBudget) to prevent the simultaneous deletion of all replicas. See the example below:
   - Example 1
   - Example 2

   Ensure that zookeeper has at least two available replicas at the time of node draining.

```
apiVersion: policy/v1beta1
kind: PodDisruptionBudget
metadata:
name: zk-pdb
spec:
minAvailable: 2
selector:
```

©2013-2022 Tencent Cloud. All rights reserved.

```
matchLabels:
app: zookeeper
```

For more details, please read Kubernetes documentation: Specifying a Disruption Budget for your Application.

Using preStopHook and readinessProbe to Ensure Smooth and Uninterrupted Service Update

If configuration is not optimized for a service, some traffic errors may occur during the service update with the default configuration. Please refer to the following steps when making deployment.

Service update scenarios

Some service update scenarios include:

- Manually adjusting the number of service replicas.
- Manually deleting Pods to trigger re-scheduling.
- Draining nodes voluntarily or involuntarily, where Pods are deleted from the drained nodes and then recreated on other nodes.
- Triggering rolling update, such as modifying the image tag to upgrade the program version.
- HPA (HorizontalPodAutoscaler) automatically scales out or scale in services.
- VPA (VerticalPodAutoscaler) automatically scales up or scale down services.

Reasons for connection errors during service update

During a rolling update, the Pods corresponding to the service being updated will be created or terminated, and the endpoints of the service will also add and remove `Pod IP:Port` corresponding to the Pods. Then kube-proxy will update the forwarding rules according to the updated `Pod IP:Port` list, but such rules are not updated immediately.

The forwarding rules are not updated immediately because Kubernetes components are decoupled from each other. Each component uses the controller mode to ListAndWatch the resources it is interested in and responds with actions. Therefore, all the steps in the process, including Pod creation or termination, endpoint update, and forwarding rules update, happen in an asynchronous manner.

When forwarding rules are not immediately updated, some connection errors could occur during the service update. The following describes two possible scenarios to analyze the reasons behind the connection errors:

- Scenario 1: Pods have been created but have not fully started yet. Endpoint controller adds the Pods to the `Pod IP:Port` list of the service. kube-proxy watches the update and updates the service forwarding rules

©2013-2022 Tencent Cloud. All rights reserved.

(iptables/ipvs). If there is a request made at this point, it could be forwarded to a Pod that has not fully started yet. A connection error may occur because the Pod is not able to properly process the request yet.

- Scenario 2: Pods have been terminated, but since all the steps in the process are asynchronous, the forwarding rules have not been updated when the Pods have been fully terminated. In such a case, new requests can still be forwarded to the terminated Pods, leading to connection errors.

Smooth update

- To address problems in scenario 1, you can add readinessProbe to the containers in the Pods. After a container fully starts, it will listen to an HTTP port to which kubelet will send readiness probe packets. If the container can respond normally, it means the container is ready, and the container's status will be modified to Ready. Only when all the containers in a Pod are ready will the Pod be added by the endpoint controller to the `IP:Port` list in the corresponding endpoint of the Service. Then, kube-proxy will update the forwarding rules. In this way, even if a request is immediately forwarded to the new Pod, it will be able to normally process the request, thereby avoiding connection errors.

- To address problems in scenario 2, you can add preStop hook to the containers in the Pods so that, before the Pods are fully terminated, they will sleep for some time during which the endpoint controller and kube-proxy can update the endpoints and the forwarding rules. During that time, the Pods will be in the Terminating status. Even if a request is forwarded to a terminating Pod before the forwarding rules are fully updated, the Pod can still normally process the request because it has not been terminated yet.

Below is a YAML sample:

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
name: nginx
spec:
replicas: 1
selector:
matchLabels:
component: nginx
template:
metadata:
labels:
component: nginx
spec:
containers:
- name: nginx
```

©2013-2022 Tencent Cloud. All rights reserved.

```
image: "nginx"
ports:
- name: http
hostPort: 80
containerPort: 80
protocol: TCP
readinessProbe:
httpGet:
path: /healthz
port: 80
httpHeaders:
- name: X-Custom-Header
value: Awesome
initialDelaySeconds: 15
timeoutSeconds: 1
lifecycle:
preStop:
exec:
command: ["/bin/bash", "-c", "sleep 30"]
```

For more information, please see Kubernetes documentation: Container probes and Container Lifecycle Hooks.

©2013-2022 Tencent Cloud. All rights reserved.

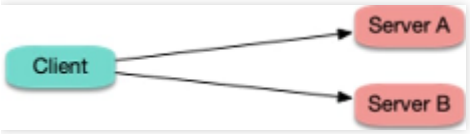
Smooth Workload Upgrade

Last updated：2022-10-12 16:05:09

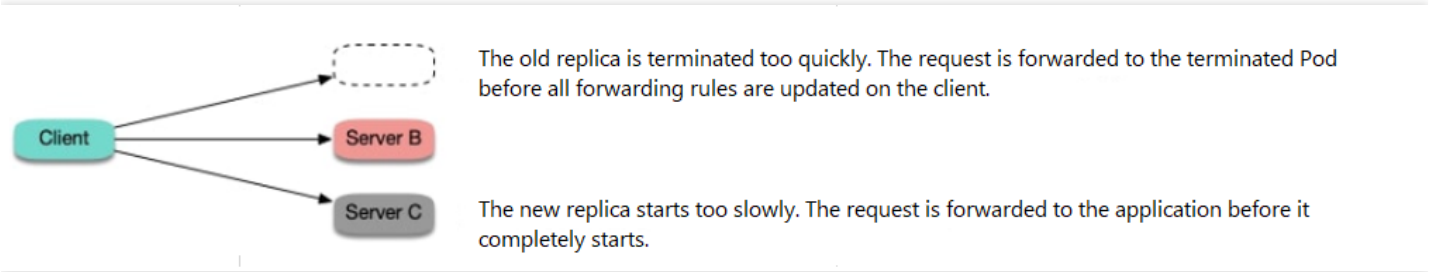
After the problem of decreased availability caused during a Service's single point of failure or node draining is solved, still another scenario that may cause availability decrease needs to be considered, that is, rolling update. A normal rolling update of a Service may affect the Service availability due to the following causes:

Lossy rolling update of the business

If there is a call between Services in the cluster:



When a rolling update is performed on the server:



Either of the following cases may occur:

- **Case 1.** The old replica is immediately terminated, but kube-proxy on the client node hasn't updated all the forwarding rules and still schedules the new connection to the old replica. This will result in a connection exception, and the error "connection refused" (the process is being stopped and no longer receives new requests) or "no route to host" (the container is completely terminated, and its ENI and IP no longer exist) may be reported.
- **Case 2.** The new replica starts, and kube-proxy on the client node immediately watches the new replica, updates the forwarding rules, and schedules the new connection to the new replica. However, a process, such as a Java process like Tomcat, starts slowly in the container, the port is not listened on, and thus the connection cannot be processed during startup, which also results in a connection exception, and the error "connection refused" will be reported generally.

©2013-2022 Tencent Cloud. All rights reserved.

Best practices

- For **case 1**, you can add `preStop` to the container to make the Pod sleep for a while before being truly terminated, during which kube-proxy on the client node will update all the forwarding rules, and then the container will be terminated. In this case, the Pod can still run for a while after being terminated, during which it can still process requests normally if new requests are forwarded to it as forwarding rules are not updated promptly on the client, so as to avoid connection exceptions. This method sounds ungraceful but has a good effect. There is no silver bullet in a distributed architecture, and you can only try to find and implement the best solution under the current design.

- For **case 2**, you can add `ReadinessProbe` to the container to make the Service Endpoint be updated only after all processes in the container are truly started. Then, kube-proxy on the client node will update the forwarding rules to forward the incoming traffic. This ensures that the traffic will be forwarded only after the Pod is completely ready and thus avoids connection exceptions.
  Sample YAML configuration:

```
readinessProbe:
httpGet:
path: /healthz
port: 80
httpHeaders:
- name: X-Custom-Header
value: Awesome
initialDelaySeconds: 10
timeoutSeconds: 1
lifecycle:
preStop:
exec:
command: ["/bin/bash", "-c", "sleep 10"]
```

©2013-2022 Tencent Cloud. All rights reserved.

Parameter Adaptation for docker run

Last updated：2020-07-22 09:34:26

This document describes how to match parameters of docker run and the TKE console when you try to migrate a container that has been debugged in the local Docker to the TKE platform. The following section uses the creation of a simple GitLab service as an example.

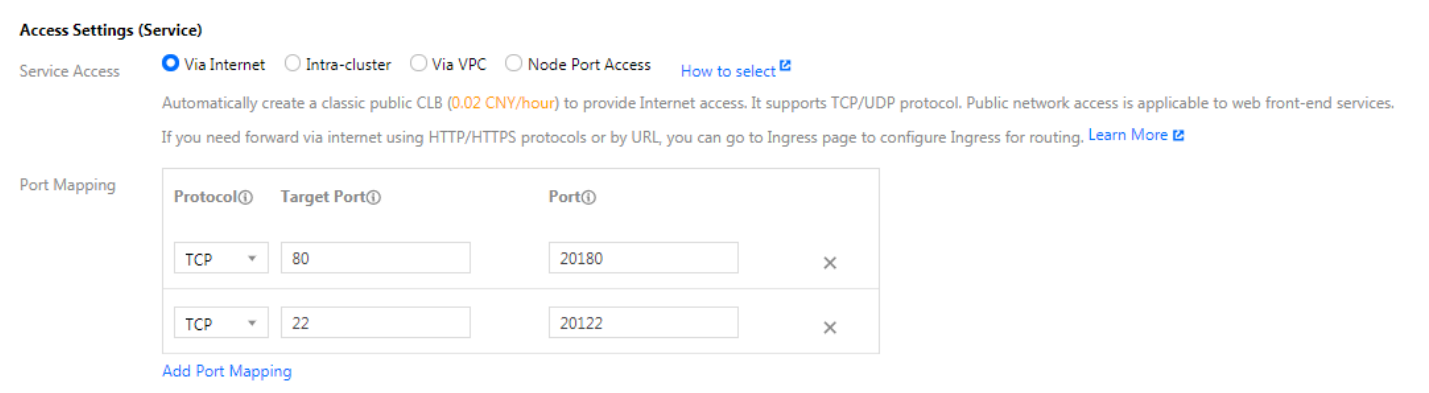
Parameters of a GitLab Container

You can create a simple GitLab container by running the following docker run command:

```
docker run \
-d \
-p 20180:80 \
-p 20122:22 \
--restart always \
-v /data/gitlab/config:/etc/gitlab \
-v /data/var/log/gitlab:/var/log/gitlab \
-v /data/gitlab/data:/var/opt/gitlab \
--name gitlab \
gitlab/gitlab-ce:8.16.7-ce.0
```

`-d` : indicates that the container runs at the backend. You do not need to specify this parameter in the TKE console because containers always run at the backend on the TKE platform.

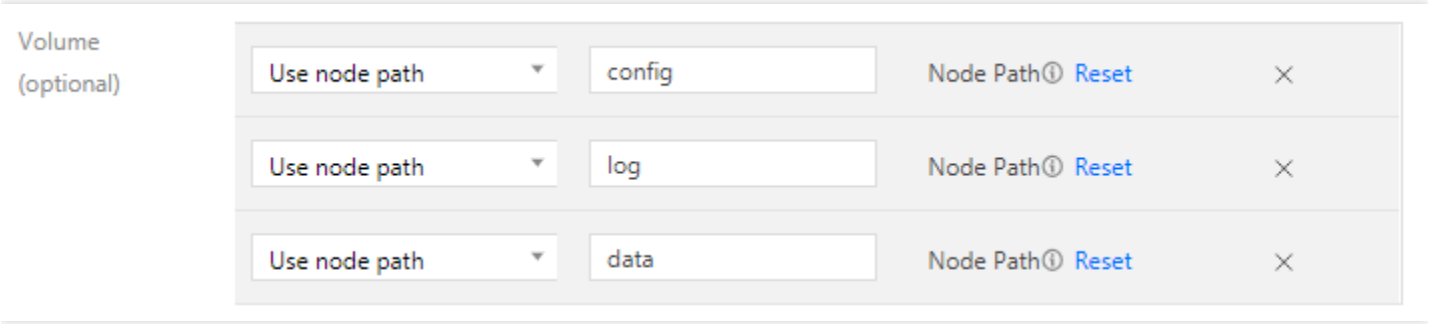
`-p` : specifies port mapping. Two ports are mapped here, that is, container ports 80 and 22, which are mapped to open ports 20180 and 20122 respectively. To take these mappings into effect, you need to add two port mapping rules in the console and specify the corresponding container ports and service ports. As GitLab needs to allow access from the public network, select **Via Internet** as the access method, as shown in the following figure.



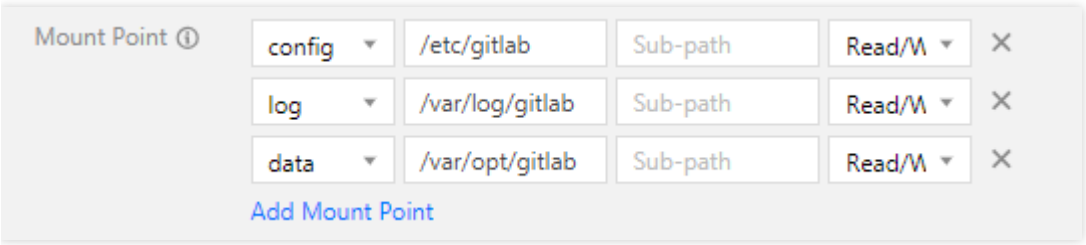
`--restart` : specifies whether to restart the container when it exits. You do not need to specify this parameter in the TKE console because all containers created on the TKE platform will restart upon exit.

©2013-2022 Tencent Cloud. All rights reserved.

`-v` : specifies container volumes. In the preceding command, three volumes are specified. Accordingly, you need to add three **data volumes** in the TKE console and mount them to the container in **Containers in the pod**.
To do this, create three volumes first, as shown in the following figure.



Mount the three volumes to the container in "Containers in the pod", as shown in the following figure.



Note that **Use node path** is selected as the data volume type. In this case, data generated during the running process of the container will be stored to the node where the container is located. If the container is scheduled to another node, the data will be lost. Alternatively, you can select **Use Tencent Cloud CBS**. In this case, the container data will be stored to the CBS instance and will not be lost even if the container is scheduled to other nodes.
`--name` : specifies the container name. This parameter corresponds to the service name in the TKE console. The container name and service name can be the same.

Other Parameters

The following describes other common parameters for executing the docker run command:
`-i` : specifies the interactive container execution mode. This parameter is not supported because the TKE console only allows containers to run at the backend.
`-t` : assigns virtual terminals. This parameter is not supported.
`-e` : specifies environment variables for container running. For example, you can run the following docker run command:

```
docker run -e FOO='foo' -e BAR='bar' --name=container_name container_image
```

Running this command adds two environment variables for the container. You can add environment variables for a container in advanced settings when creating a service in the TKE console. The names and values of the variables are

©2013-2022 Tencent Cloud. All rights reserved.

as follows:

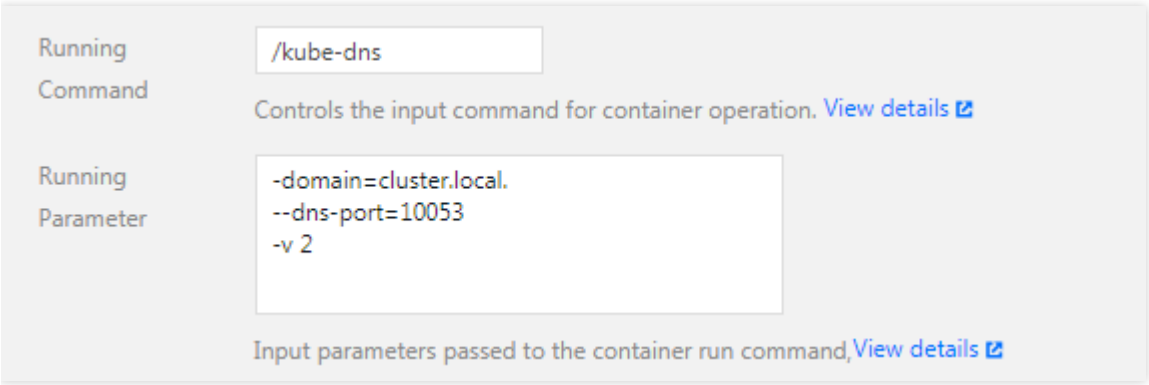
- Variable: FOO, value: foo.
- Variable: BAR, value: bar.

Command and Arguments

You can specify the command name and arguments of a container process in docker run. For example:

```
docker run --name=kubedns gcr.io/google_containers/kubedns-amd64:1.7 /kube-dns --domain=cluster.local. --dns-port=10053 -v 2
```

In this case, the command name of the container process is `/kube-dns`, and the arguments are `-domain=cluster.local.`, `--dns-port=10053`, and `-v 2`. The following figure shows how to set these arguments in the TKE console.



©2013-2022 Tencent Cloud. All rights reserved.

Solve the inconsistent time zone problem in the container

Last updated：2020-04-29 17:40:27

Introduction

The default system time of containers in TKE clusters is Universal Time Coordinated (UTC), which may be different with the local time zone of your nodes. During the use of containers, time zone inconsistency in containers will cause trouble when the system time is used for operations, such as log records and database storage. In this document, we will use "Asia/Shanghai" as the local time zone.

You cannot modify the default time of the cluster but the container. This document provides multiple solutions to time zone inconsistencies in containers. You can choose the solution that works for you.

- Solution 1: create a time zone file in Dockerfile (recommended)
- Solution 2: mount the time zone configuration of the CVM to the container

Operation Environment

All operations described in this document are completed on TKE cluster nodes. The relevant operation environment is shown below. Please use this document to solve problems based on your actual situation.

Role	Region	Specifications	OS	Kubernetes Version
Node	South China (Guangzhou)	CPU: 1 core, memory: 1 GB, bandwidth: 1 Mbps System disk: 50 GB (HDD cloud disk)	CentOS Linux 7 (Core)	1.16.3

Cause Locating

1. Log in to the target node by referring to Log in to Linux Instance Using Standard Login Method (Recommended).
2. Run the following command to query the local time:

```
date
```

©2013-2022 Tencent Cloud. All rights reserved.

The following information appears:

```
[root@VM_6_12_centos ~]# date
Tue Mar  3 16:23:53 CST 2020
```

3. Run the following commands in sequence to query the default time zone of CentOS in the container:

```
docker run -it centos /bin/sh
```

```
date
```

The following information appears:

```
[root@VM_6_12_centos ~]# docker run -it centos /bin/sh
Unable to find image 'centos:latest' locally
latest: Pulling from library/centos
8a29a15cefae: Pull complete
Digest: sha256:fe8d824220415eed5477b63addf40fb06c3b049404242b31982106ac204f6700
Status: Downloaded newer image for centos:latest
sh-4.4# date
Tue Mar  3 08:24:29 UTC 2020
sh-4.4#
```

By comparison, it is clear that the local time zone and the time zone in the container are inconsistent.

4. Run the following command to exit the container:

```
exit
```

Directions

Solution 1: create a time zone file in Dockerfile (recommended)

When creating a basic image or customizing an image based on a basic image, you can create a time zone file in Dockerfile to solve time zone inconsistency within a container. After this, you will no longer be troubled by time zone issues when using the image.

1. Run the following command to create the Dockerfile.txt file:

```
vim Dockerfile.txt
```

2. Press **i** to switch to the editing mode, and write the following information to configure the time zone file.

```
FROM centos
RUN rm -f /etc/localtime \
```

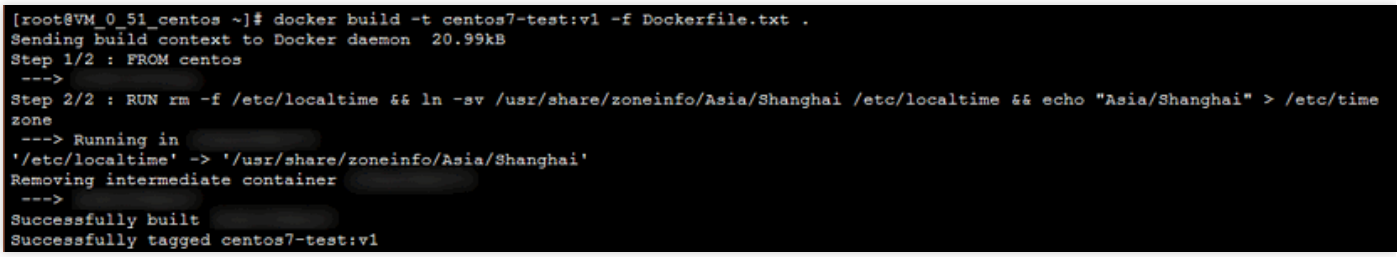
©2013-2022 Tencent Cloud. All rights reserved.

```
&& ln -sv /usr/share/zoneinfo/Asia/Shanghai /etc/localtime \
&& echo "Asia/Shanghai" > /etc/timezone
```

3. Press **Esc**, enter **:wq**, and save and close the file.
4. Run the following command to create a container image:

```
docker build -t centos7-test:v1 -f Dockerfile.txt .
```

The following information appears:



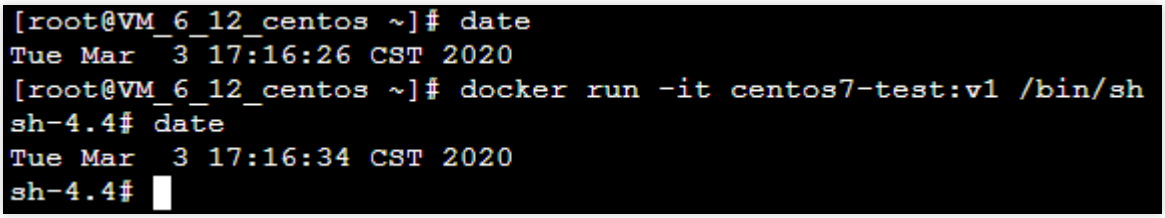
5. Run the following commands in sequence to launch the container image and query the time zone in the container:

```
date
```

```
docker run -it centos7-test:v1 /bin/sh
```

```
date
```

The time zone in the container is consistent with the local time. See the figure below:



6. Run the following command to exit the container:

```
exit
```

Solution 2: mount the time zone configuration of the CVM to the container

You can also solve time zone inconsistency in a container by mounting the time configuration of the CVM to the container. This solution can be set when the container is started, or you can use the CVM path in the YAML file to mount volumes to the container.

Mounting CVM time configuration to the container when the container is started

©2013-2022 Tencent Cloud. All rights reserved.

When mounting the CVM time configuration to the container to overwrite the original configuration, there are two options:

- Mount local `/etc/localtime` : you need to ensure that the CVM time zone configuration file exists and the time zone is correct.
- Mount local `/usr/share/zoneinfo/Asia/Shanghai` : when the local `/etc/localtime` does not exist or the time zone is incorrect, you can directly mount the configuration file.

Choose one of the following methods based on your situation to mount the CVM time configuration to the container:

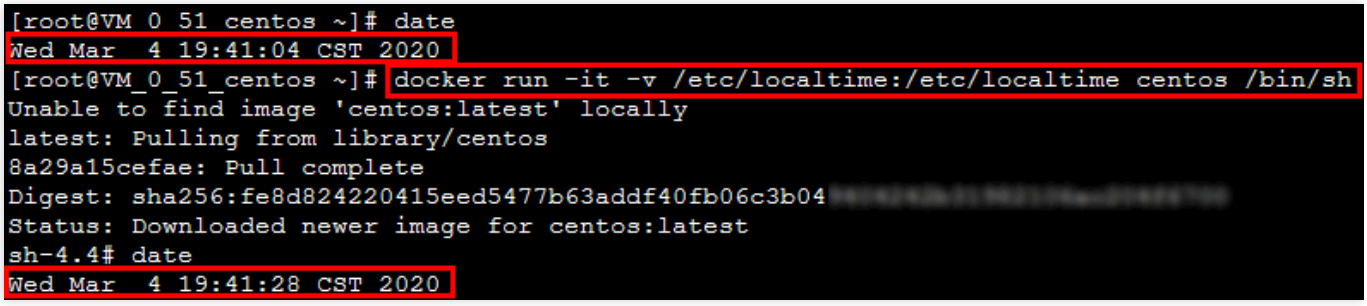
- Method 1: mount local `/etc/localtime` ;
  i. Run the following commands in sequence to query the local time and mount the local `/etc/localtime` into the container:

```
date
```

```
docker run -it -v /etc/localtime:/etc/localtime centos /bin/sh
```

```
date
```

If the following information appears, the time zone in the container is consistent with the local time:



  ii. Run the following command to exit the container:

```
exit
```

- Method 2: mount local `/usr/share/zoneinfo/Asia/Shanghai` :
  i. Run the following commands in sequence to query the local time and mount local `/usr/share/zoneinfo/Asia/Shanghai` into the container:

```
date
```

©2013-2022 Tencent Cloud. All rights reserved.

```
docker run -it -v /usr/share/zoneinfo/Asia/Shanghai:/etc/localtime centos /bi
n/sh
```

```
date
```

If the following information appears, the time zone in the container is consistent with the local time:

ii. Run the following command to exit the container:

```
exit
```

Using data volumes in the YAML file to mount the CVM time zone configuration to the container

This section uses `mountPath:/etc/localtime` as an example to illustrate how to mount the CVM time zone configuration to the container using volumes in the YAML file. This will solve time zone inconsistency in the container.

1. Run the following command on the node to create the pod.yaml file:

```
vim pod.yaml
```

2. Press **i** to switch to the editing mode and enter the following.

```
apiVersion: v1
kind: Pod
metadata:
name: test
namespace: default
spec:
restartPolicy: OnFailure
containers:
- name: nginx
image: nginx-test
imagePullPolicy: IfNotPresent
volumeMounts:
- name: date-config
mountPath: /etc/localtime
command: ["sleep", "60000"]
volumes:
- name: date-config
hostPath:
path: /etc/localtime
```

©2013-2022 Tencent Cloud. All rights reserved.

3. Press **Esc**, enter **:wq**, and save and close the file.
4. Run the following command to create a pod:

```
kubectl create -f pod.yaml
```

The following information appears:

```
[root@VM_6_5_centos ~]# kubectl create -f  pod.yaml
pod/test created
```

5. Run the following commands in sequence to query the time zone in the container:

```
date
```

```
kubectl exec -it test date
```

If the following information appears, the time zone is consistent with the local system time zone.

```
[root@VM_6_5_centos ~]# date
Wed Mar  4 11:56:27 CST 2020
[root@VM_6_5_centos ~]# kubectl exec -it test date
Wed Mar  4 11:56:31 CST 2020
[root@VM_6_5_centos ~]#
```

©2013-2022 Tencent Cloud. All rights reserved.

Container coredump Persistence

Last updated：2020-12-10 11:29:37

Operation Scenarios

Sometimes, containers may fail to work properly after an exception occurs. If there is no sufficient information in the business log to help you identify the cause, you need to use coredump for further analysis. This document how to generate and save coredump for containers.

> ⚠ **Note：**
>
> This document only applies to TKE clusters.

Prerequisites

You have logged in to the TKE console.

Directions

Enabling coredump

1. Run the following command on the node to set the storage path format of the core file for the node:

```
# Run the following command on the node:
echo "/tmp/cores/core.%h.%e.%p.%t" > /proc/sys/kernel/core_pattern
```

Main parameters are described as follows:

- **%h**: host name (in a Pod, the host name is the Pod name) (recommended).
- **%e**: program file name (recommended).
- **%p**: process ID (optional).
- **%t**: coredump time (optional).

The complete path where the core file is generated is as follows:

©2013-2022 Tencent Cloud. All rights reserved.

```
/tmp/cores/core.nginx-7855fc5b44-p2rzt.bash.36.1602488967
```

2. After configuring the node, you need not modify the existing container configuration. The container configuration will automatically take effect through inheritance. If you require batch execution on multiple nodes, perform the corresponding operation accordingly:
   - For existing nodes, see Performing batch operations on TKE nodes by using Ansible.
   - For new nodes, see Configuring the launch script of a node.

Enabling the COS add-on

To prevent the loss of the core file after the container restarts, you need to mount a volume for the container. As the cost of mounting an independent cloud disk for each pod is too high, you need to mount the component to COS. For the directions, see Installing the COS add-on.

Creating a bucket

Log in to the COS console, and manually create a COS bucket for storing the core file generated by the container coredump. In this document, a custom bucket named coredump is created as an example. For directions, see Creating a bucket.

Creating a Secret

You can choose any of the following three methods to create a Secret for accessing COS based on your needs:

- To use COS via the console, see Creating a secret that can access COS.
- To use COS via a YAML file, see Creating a secret that can access COS.
- To create a Secret by using the kubectl command line tool, refer to the following code snippet:

```
kubectl create secret generic cos-secret -n kube-system --from-literal=SecretId
=AKI*****************lV --from-literal=SecretKey=paQ9***************sZF
```

> **Note：**
>
> Remember to replace SecretId, SecretKey, and namespace.

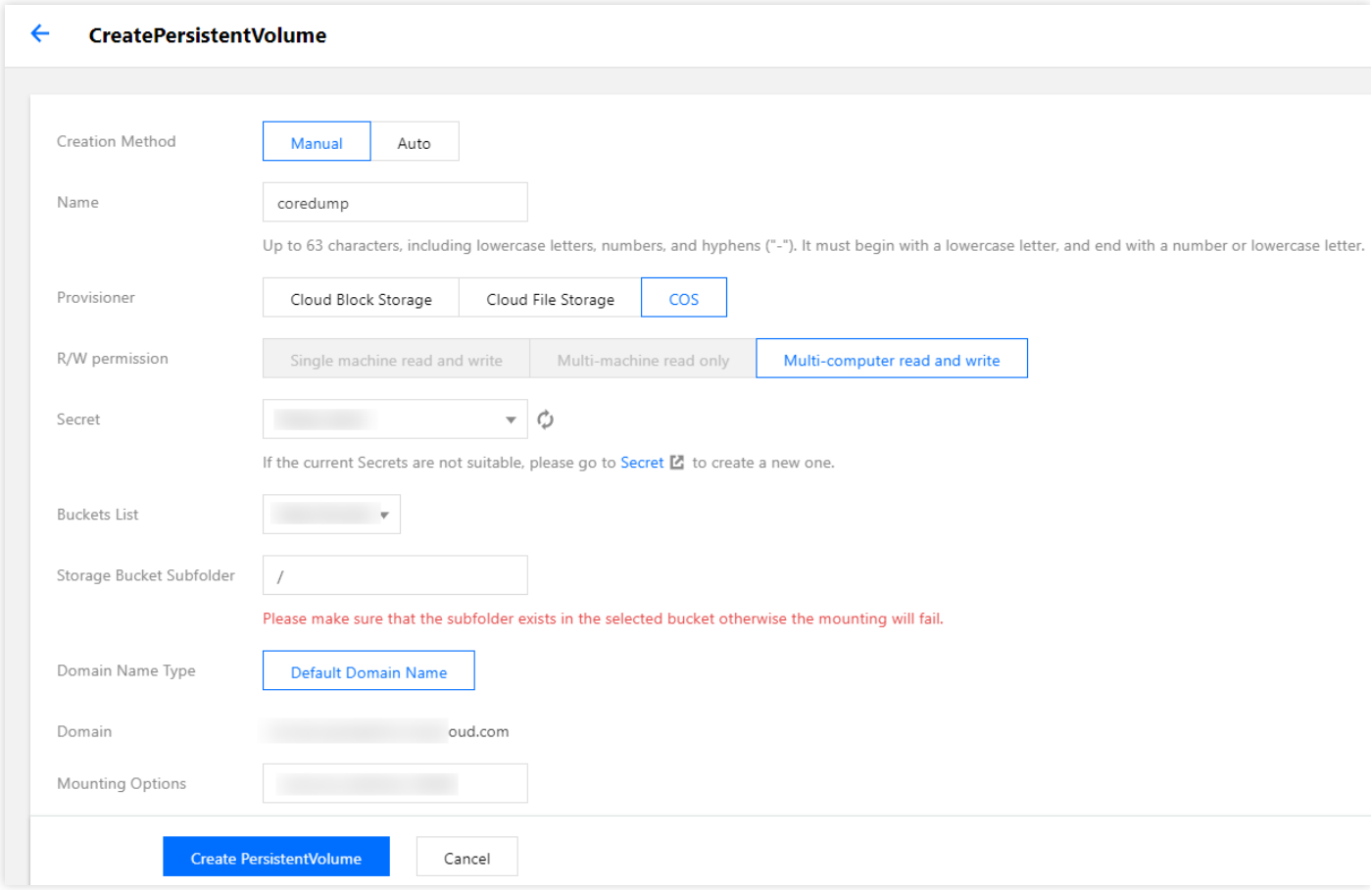
Creating a PV and PVC

To use the COS plug-in, you need to manually create a PV and PVC and then bind them.

Creating a PV

©2013-2022 Tencent Cloud. All rights reserved.

1. On the details page of the target cluster, choose **Storage** > **PersistentVolume** in the left sidebar to go to the "PersistentVolume" page.
2. Click **Create** to go to the "Create a PersistentVolume" page and set the PV parameters as required, as shown in the figure below:



Main parameters are described as follows:

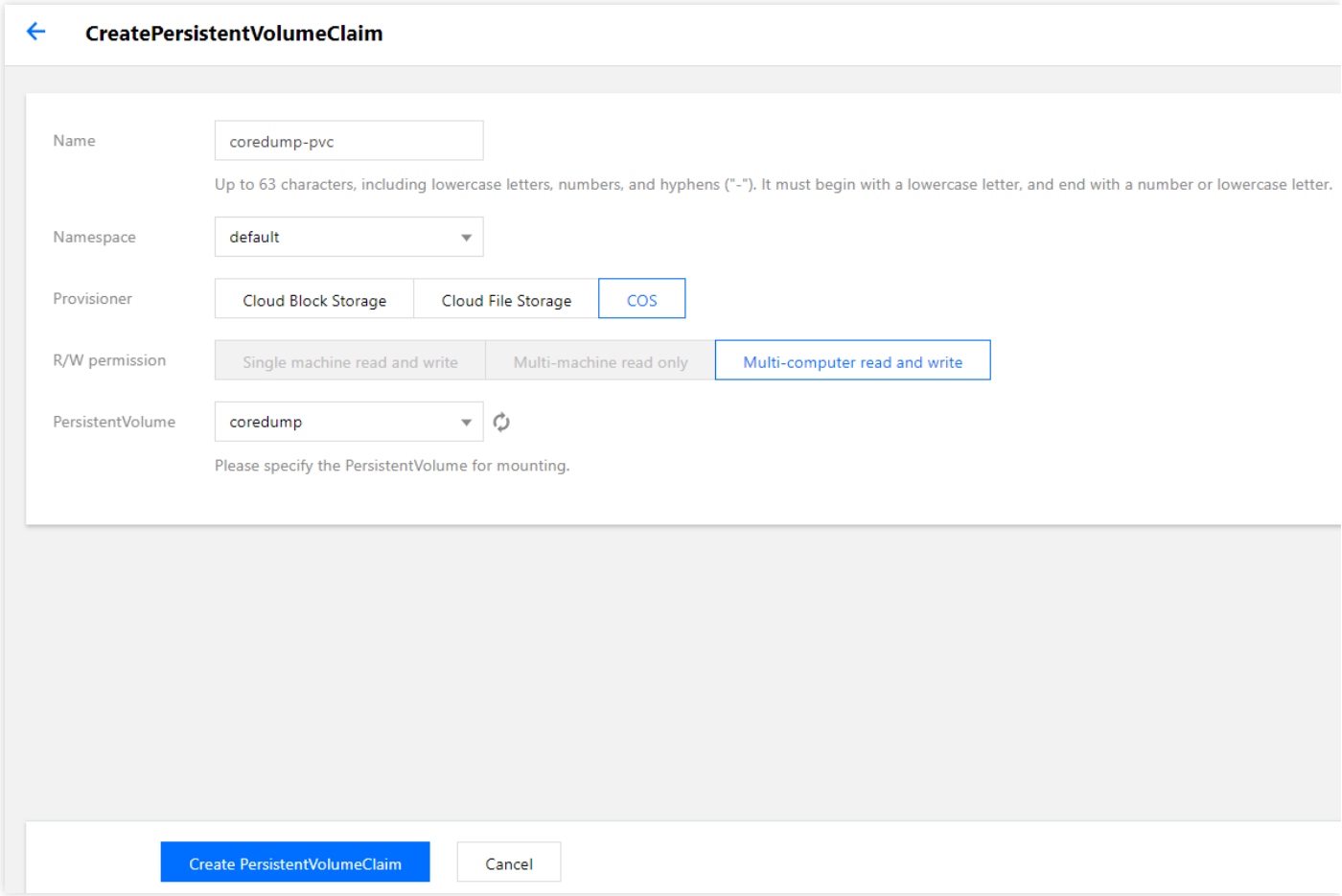
- **Creation Method**: select **Static**.
- **Secret**: select the Secret created in Creating a Secret. In this document, coredump is used as an example (under the kube-system namespace).
- **Bucket List**: select the bucket created for storing the coredump file.
- **Bucket Sub-directory**: specify the root directory here. If you need to specify a sub-directory, please create one in the bucket in advance.

3. Click **Create a PersistentVolume** to complete the process.

Creating a PVC

1. On the details page of the target cluster, choose **Storage** > **PersistentVolumeClaim** in the left sidebar to go to the "PersistentVolumeClaim" page.
2. Click **Create** to go to the "Create a PersistentVolumeClaim" page and set the PVC parameters as required, as shown in the figure below:

©2013-2022 Tencent Cloud. All rights reserved.



Main parameters are described as follows:

- **Namespace**: the namespace must be the same as the namespace where the container of the PVC for mounting COS belongs. If there are multiple namespaces, you can create multiple pairs of PVs and PVCs.
- **PersistentVolume**: select the PV created in Creating a PV.

3. Click **Create a PersistentVolumeClaim** to complete the process.

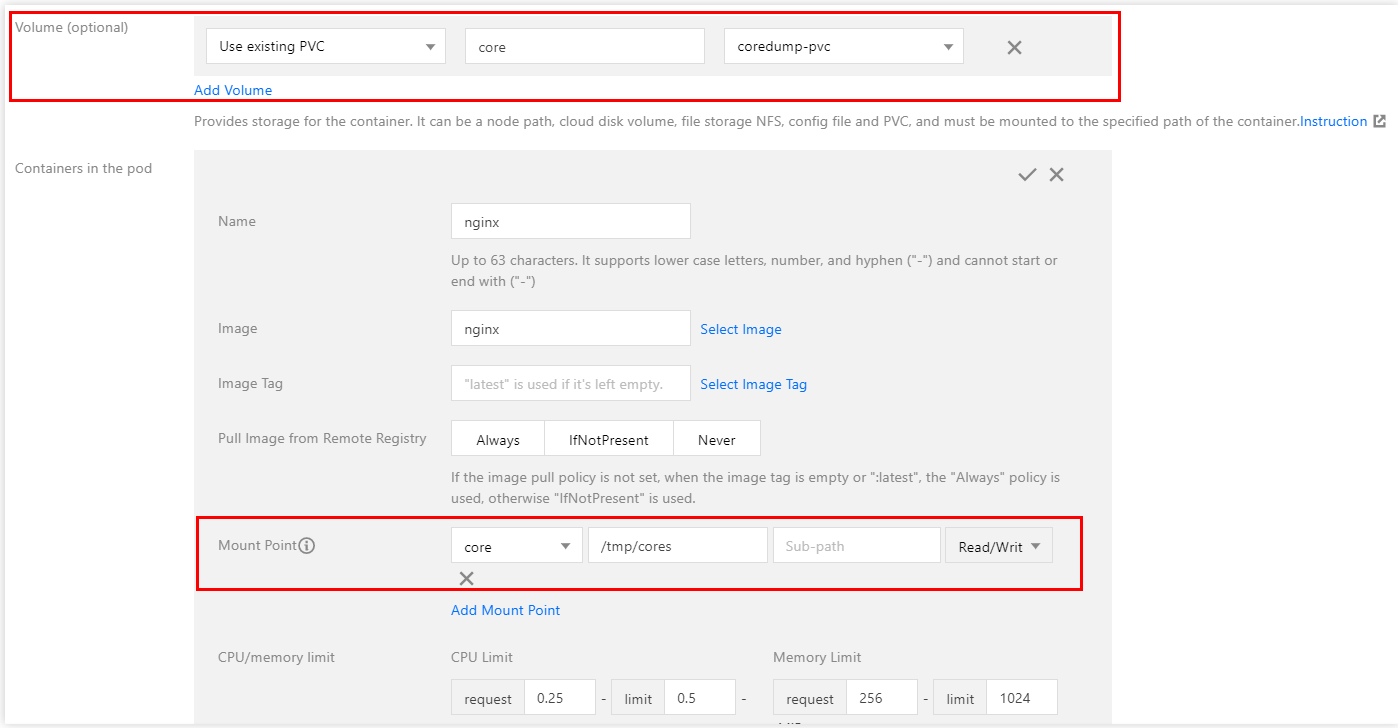
Mounting COS

Using the console to create a Pod to use the PVC

> **Note：**
>
> This step creates a Deployment workload as an example.

1. On the details page of the target cluster, choose **Workload** > **Deployment** to go to the "Deployment" page.
2. Click **Create** to go to the "Create a Workload" page. For more information, see Creating a Deployment. Then, mount a volume as required, as shown in the figure below:

©2013-2022 Tencent Cloud. All rights reserved.



Main parameters are described as follows:

- **Volume**: add the PVC created in Creating a PVC.
- **Mount Target**: click **Add a mount target** to set a mount target. Here, select the added volume "core". Import the PVC specified in **Volume**, and mount it to the destination path. In this document, `/tmp/cores` is used as an example.

3. Click **Create a Workload** to complete the process.

Using a YAML file to create a Pod to use the PVC

You can create a Pod by using a YAML file. Below is a sample:

```
containers:
- name: pod-cos
command: ["tail", "-f", "/etc/hosts"]
image: "centos:latest"
volumeMounts:
- mountPath: /tmp/cores
name: core
volumes:
- name: core
persistentVolumeClaim:
# Replaced by your pvc name.
claimName: coredump
```

©2013-2022 Tencent Cloud. All rights reserved.

Reference

Using COS

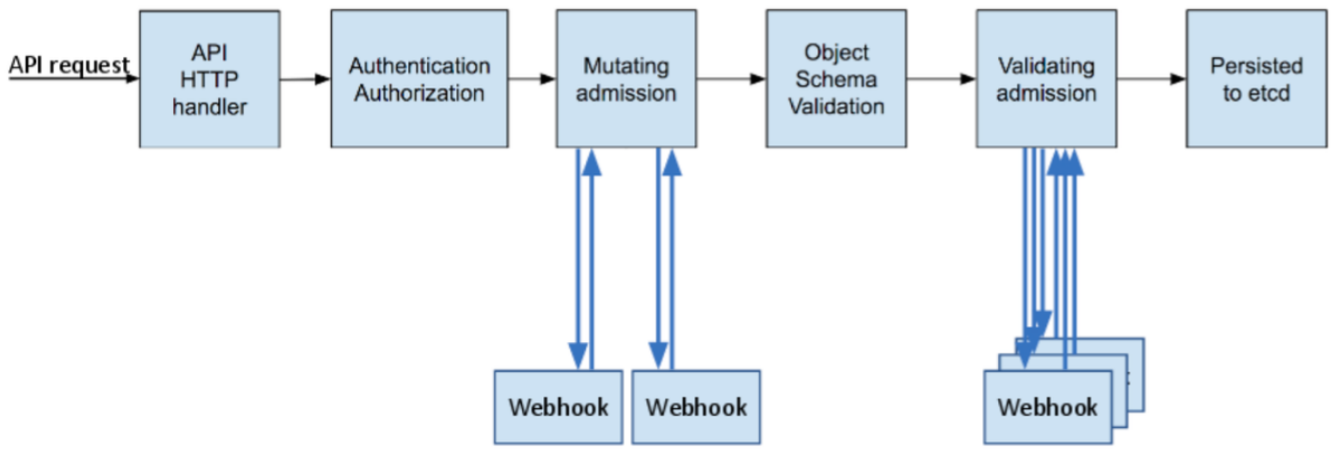
©2013-2022 Tencent Cloud. All rights reserved.

Using a Dynamic Admission Controller in TKE

Last updated：2021-05-24 14:28:39

Operation Scenario

The dynamic admission controller Webhook can change the request object or completely reject a request during access authentication. The way it calls the Webhook service makes it independent of cluster components.
The dynamic admission controller has a high degree of flexibility and allows you to configure various custom admission control settings. The following figure shows the position of dynamic admission control in the API request call chain. For more information, visit the official Kubernetes website.



As shown in the figure, dynamic admission control is divided into two phases: Mutating and Validating. During the Mutating phase, incoming requests can be modified. Subsequently, during the Validating phase, the dynamic admission controller validates incoming requests to determine whether to allow them to pass. These two phases can be used independently or in combination.

This document introduces a simple use case for calling the dynamic admission controller in TKE. You can refer to this document and take your actual requirements into consideration when performing the relevant operations.

Directions

Viewing and verifying the plug-in

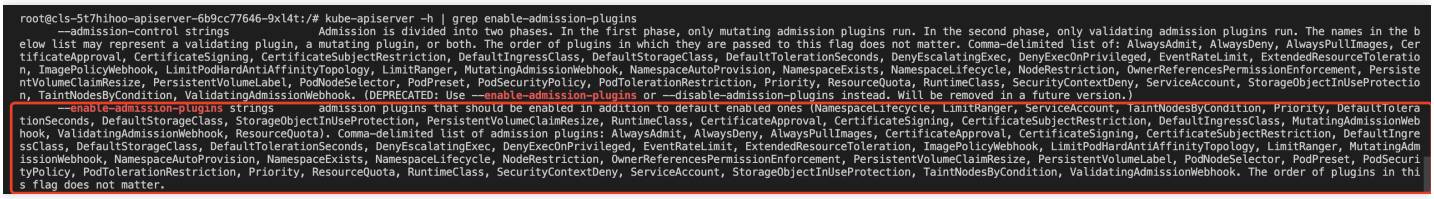
The existing TKE cluster versions (1.10.5 and later) enable the validating admission webhook and mutating admission webhook APIs by default. If your cluster version is earlier than 1.10.5, you can run the following command to check

©2013-2022 Tencent Cloud. All rights reserved.

whether the plug-in has been enabled in your current cluster.

```
kube-apiserver -h | grep enable-admission-plugins
```

If the returned result includes `MutatingAdmissionWebhook` and `ValidatingAdmissionWebhook`, the dynamic admission controller is already enabled in the cluster, as shown in the figure below:



Certificate issuance

To ensure that the dynamic admission controller calls a trustworthy Webhook server, it needs to call the Webhook service (TLS certification) via HTTPS. Therefore, you need to issue a certificate to the Webhook server. During registration of the dynamic admission controller Webhook, you need to bind the `caBundle` field (`caBundle` field in the resource list of `ValidatingWebhookConfiguration` and `MutatingAdmissionWebhook`) with a trustworthy certificate authority (CA) to verify whether the Webhook server certificate is trustworthy. This document introduces two recommended methods for issuing certificates: making a self-signed certificate and using the K8S CSR API to issue a certificate.

> Note：
> When `ValidatingWebhookConfiguration` and `MutatingAdmissionWebhook` use the `clientConfig.service` configuration (and the Webhook service is in the cluster), the domain name of the certificate issued to the server must be `<svc_name>.<svc_namespace>.svc` .

Method 1: making a self-signed certificate

This method is not dependent on Kubernetes clusters and is relatively independent. It's similar to the way in which websites make their own self-signed certificates. Currently, many tools can be used to make a self-signed certificate. This document uses OpenSSL as an example. The procedure is as follows:

1. Run the following command to generate a `ca.key` with 2048 key digits.

```
openssl genrsa -out ca.key 2048
```

©2013-2022 Tencent Cloud. All rights reserved.

2. Run the following command to generate a `ca.crt` based on the `ca.key`.
"webserver.default.svc" is the domain name of the Webhook server in the cluster. The `-days` parameter is used to specify the validity period of the certificate.

```
openssl req -x509 -new -nodes -key ca.key -subj "/CN=webserver.default.svc" -da
ys 10000 -out ca.crt
```

3. Run the following command to generate a `server.key` with 2048 key digits.

```
openssl genrsa -out server.key 2048
```

4. Create the configuration file `csr.conf` used to generate a certificate signature request (CSR). See the sample below:

```
[ req ]
default_bits = 2048
prompt = no
default_md = sha256
distinguished_name = dn
[ dn ]
C = cn
ST = shaanxi
L = xi'an
O = default
OU = websever
CN = webserver.default.svc
subjectAltName = @alt_names
[ alt_names ]
DNS.1 = webserver.default.svc
[ v3_ext ]
authorityKeyIdentifier=keyid,issuer:always
basicConstraints=CA:FALSE
keyUsage=keyEncipherment,dataEncipherment
extendedKeyUsage=serverAuth,clientAuth
subjectAltName=@alt_names
```

5. Run the following command to generate a CSR based on the configuration file `csr.conf`.

```
openssl req -new -key server.key -out server.csr -config csr.conf
```

©2013-2022 Tencent Cloud. All rights reserved.

6. Run the following commands to use `ca.key`, `ca.crt`, and `server.csr` to issue the generated server certificate (x509 signature).

```
openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key \
-CAcreateserial -out server.crt -days 10000 \
-extensions v3_ext -extfile csr.conf
```

7. Run the following command to view the Webhook server certificate.

```
openssl x509 -noout -text -in ./server.crt
```

The generated certificates and key files are described as follows:

- `ca.crt`: the CA certificate
- `ca.key`: the CA certificate key, used to issue a server certificate
- `server.crt`: the issued server certificate
- `server.key`: the issued server certificate key

Method 2: using the K8S CSR API to issue a certificate

You can also use the Kubernetes CA system to issue a certificate. You can execute the following script to use the Kubernetes cluster root certificate and root key to issue a trustworthy certificate user.

> Note：
> The username must be the domain name of the Webhook service in the cluster.

```
USERNAME='webserver.default.svc' # Set the username to be created to the domain n
ame of the Webhook service in the cluster
# Use OpenSSL to generate a self-signed certificate key
openssl genrsa -out ${USERNAME}.key 2048
# Use OpenSSL to generate a self-signed CSR file, with CN indicating the user nam
e and O indicating the group name
openssl req -new -key ${USERNAME}.key -out ${USERNAME}.csr -subj "/CN=${USERNAME}
/O=${USERNAME}"
# Create a Kubernetes CSR
cat <<EOF | kubectl apply -f -
apiVersion: certificates.k8s.io/v1beta1
kind: CertificateSigningRequest
metadata:
name: ${USERNAME}
```

©2013-2022 Tencent Cloud. All rights reserved.

```
spec:
request: $(cat ${USERNAME}.csr | base64 | tr -d '\n')
usages:
- digital signature
- key encipherment
- server auth
EOF
# Approve the certificate as trustworthy
kubectl certificate approve ${USERNAME}
# Obtain the self-signed certificate CRT
kubectl get csr ${USERNAME} -o jsonpath={.status.certificate} > ${USERNAME}.crt
```

- `${USERNAME}`.crt: the Webhook server certificate
- `${USERNAME}`.key: the Webhook server certificate key

Use Cases

This document uses `ValidatingWebhookConfiguration` resources to illustrate how to call the dynamic admission controller Webhook.
To ensure accessibility, the sample code is forked from the original code library to implement a simple API for dynamic admission Webhook requests and responses. For the detailed API format, see Webhook request and response. The sample code can be obtained in Sample Code. This document uses it as the Webhook server code.

1. Prepare the `caBundle` content corresponding to the actual certificate issuance method.
   - If you use method 1 to issue a certificate, run the following command to use `base64` to encode `ca.crt` and generate the `caBundle` field content.

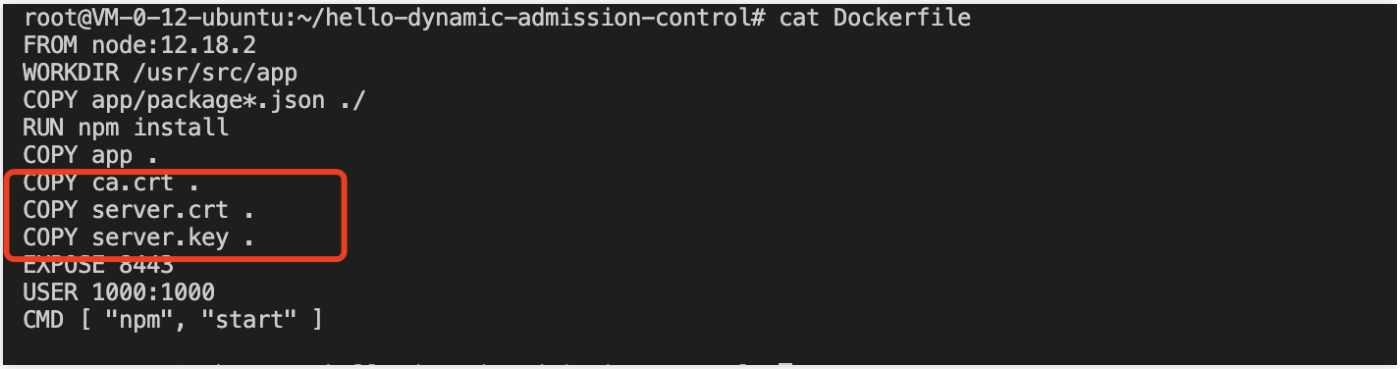
     ```
     cat ca.crt | base64 --wrap=0
     ```

   - If you use method 2 to issue a certificate, the cluster root certificate is the `caBundle` field content. The procedure for obtaining it is as follows:
     a. Log in to the TKE console and click **Clusters** in the left sidebar.
     b. On the "Cluster Management" page, click the ID of the target cluster.
     c. On the cluster details page, click **Basic Information** on the left.
     d. On the "Basic Information" page, obtain the `clusters.cluster[].certificate-authority-data` field in "Kubeconfig" in the "Cluster APIServer Info" module. This field has been encoded in `base64`, and no further processing is needed.

©2013-2022 Tencent Cloud. All rights reserved.

2. Copy the generated `ca.crt` (CA certificate), `server.crt` (HTTPS certificate), and `server.key` (HTTPS key) to the main directory of the project, as shown in the figure below:

3. Modify the Dockerfile in the project and add three certificate files to the container working directory, as shown in the figure below:



4. Run the following command to build a Webhook server image.

```
docker build -t webserver .
```

5. Deploy a Webhook backend service with the domain name of "weserver.default.svc" and modify the adapted `controller.yaml`, as shown in the figure below:

©2013-2022 Tencent Cloud. All rights reserved.

```
root@VM-0-12-ubuntu:~/hello-dynamic-admission-control# cat controller.yaml
apiVersion: v1
kind: Service
metadata:
  name: webserver
spec:
  ports:
  - port: 443
    protocol: TCP
    targetPort: 8443
  selector:
    run: webserver
---
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: webserver
  name: webserver
spec:
  containers:
  - image: webserver
    name: webserver
    ports:
    - containerPort: 8443
    imagePullPolicy: IfNotPresent
    livenessProbe:
      httpGet:
        port: 8443
        path: /hc
        scheme: HTTPS
    readinessProbe:
      httpGet:
        port: 8443
        path: /hc
        scheme: HTTPS
    resources:
      limits:
        cpu: 100m
        memory: 128Mi
      requests:
        cpu: 10m
        memory: 12Mi
    securityContext:
      runAsNonRoot: true
      readOnlyRootFilesystem: true

root@VM-0-12-ubuntu:~/hello-dynamic-admission-control#
```

6. Register and create resources of the `ValidatingWebhookConfiguration` type, and modify the `admission.yaml` file in the adapted project, as shown in the figure below:
The Webhook triggering rule configured in this sample is as follows: when an API of `pods` type and version "v1" is created, Webhook is triggered. The configuration of `clientConfig` corresponds to the above Webhook backend service created in the cluster. The `caBundle` field content is the content of the `ca.crt` obtained in method 1.

```
root@VM-0-12-ubuntu:~/hello-dynamic-admission-control# cat admission.yaml
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
metadata:
  name: "webserver.default.svc"
webhooks:
- name: "webserver.default.svc"
  rules:
  - apiGroups:   [""]
    apiVersions: ["v1"]
    operations:  ["CREATE"]
    resources:   ["pods"]
    scope:       "Namespaced"
  clientConfig:
    service:
      namespace: "default"
      name: "webserver"
    caBundle: "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURFekNDQWZ1Z0F3SUJBZ0lKQU5ZMkg3WlYxb1lFTUEwR0NTcUdTSWIzRFFFQkN3VUFNQ0F4SGpBY0JnTlYKQkFNTUZYZGxZbk5sY25abGNpNWtaV1poZFd4MExuTjJZekFlRncweU1ERXhNVGN3TkRVeU1URmFGdzAwT
0RBMApNRFF3TkRVeU1URmFNQ0F4SGpBY0JnTlZCQU1NRlhkbFluTmxjblpsY2k1aXWmhkV3gwTG5OMll6Q0NBU0l3CkRRWUpLb1pJaHZjTkFRRUJCUUFEZ2dFUEFEQ0NBUW9DZ2dFQkFNRnl5aXpjYU1uZ3EvanRTWVhwdDQwY2dzK0EKQ3pKTnl6a0VTT2l2ZEJ0alB6WHpKTEFCYlh0UkE3U
XYwWkNJakt2OTZLZ1dGUmluVFRZVjBHVVV2T3BsWEh1aApiUnBzZXRVUGFwZUp1ZDRuOFozY09xUTk2T0JRM2ZiNEt1SFdCYXI5UU1sYjFjVDZ3amk0bTZWSklUVmhiQzhjCkFJRWpKSG5uL2pyMkVwTkVIdjhkL1J4S3kyVFVrc3UvYVBhU0I2Sm84Ui9JVkFzNmNHeWwzbnFZdEJ3NFBQSDcKeTltUXJsQko3VHNsazR2aW1xMEVOUXhDWDA2ZVBjVFlyendwTGtqZHMyWU5pSmRiM3pVVEhpZWJTUzVWWkRxWApPT0NWWUt4VnNSUkRwRWFNMDFxR0NiNWJnc2xwTXpHZk5FRUF1K2dLSVM1VGFsTkRhVVhacWRHT0xRRUNBd0VBCkFhTlFNRTR3SFFZRFZSME9CQllFRkZiWkVzeEhaSGdseXVqaWFPZ2w5eS9rTlNrck1COEdBMVVkSXdRWU1CYUEKRkZiWkVzeEhaSGdseXVqaWFPZ2w5eS9rTlNrck1Bd0dBMVVkRXdRRk1BTUJBZjh3RFFZSktvWklodmNOQVFFTApCUUFEZ2dFQkFIVzI4c01RWFNFUnZYMFJDbGlvT3oydFFhVkVhaWd2aU1SR09SZith3duUWpWdktNRHhvRETRCnhRUHh6enhhWjFNRDd4QmRUTlQzVFArZWNTeC9qdEh2RkpPQk10NHQrdHRlb3h5Tl1YlVWZjdNTHRyb2dUYW0KeVE4b0JZN3ppOWtkaGtSZnhmanEraFdZWis5d0hxWFE2U2NFaktzekwxRMjVsdllGWG1FOVB0ZFNXcFpURWtmeggFSklFUjBPZnZxd2F4VHBISEN1NkpucjNDM2NtMDJpWkgyYStMWWVDZ
k1EOTU2YTJqbU4xWCs2cWg0WGVGaHRmCjJCZzNHVUtyVkpJOGUwc01PcUVUVzYrbEtTZ3VQS0FzWStKRXhRbmhzbWJHcjZNNk5MNjR6Zzh5dVl6aVY4UnYKRDc3M0V4a0tzVXdrMU9SQ1hySm4rWWtXWk53VlJpQT0KLS0tLS1FTkQgQ0VSVElGSUNBVEUtLS0tLQo="
    admissionReviewVersions: ["v1"]
    sideEffects: None
    timeoutSeconds: 5
```

©2013-2022 Tencent Cloud. All rights reserved.

7. After registration, create test resources of the Pod type and the API version of "v1", as shown in the figure below:

```
root@VM-0-12-ubuntu:~/hello-dynamic-admission-control# cat pod.yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
  name: hello-pod
spec:
  containers:
  - name: hello
    image: alpine
    command: ['tail', '-f', '/dev/null']
```

8. The test code prints the request log. You can view the Webhook server log to see that the dynamic admission controller has triggered a webhook call, as shown in the figure below:

```
{
  kind: 'AdmissionReview',
  apiVersion: 'admission.k8s.io/v1',
  request: {
    uid: '31ce0418-ba2e-4daf-a6f4-7e97454d06d1',
    kind: { group: '', version: 'v1', kind: 'Pod' },
    resource: { group: '', version: 'v1', resource: 'pods' },
    requestKind: { group: '', version: 'v1', kind: 'Pod' },
    requestResource: { group: '', version: 'v1', resource: 'pods' },
    name: 'hello-pod',
    namespace: 'default',
    operation: 'CREATE',
    userInfo: { username: '100015757548-1600947194', groups: [Array] },
    object: {
      kind: 'Pod',
      apiVersion: 'v1',
      metadata: [Object],
      spec: [Object],
      status: [Object]
    },
    oldObject: null,
    dryRun: false,
    options: { kind: 'CreateOptions', apiVersion: 'meta.k8s.io/v1' }
  }
}
```

9. At this moment, you can see that the test pod has been created successfully. As the test Webhook server code includes the `allowed: true` configuration item, the test pod has been created successfully, as shown in the figure below:

©2013-2022 Tencent Cloud. All rights reserved.

```
root@VM-0-12-ubuntu:~/hello-dynamic-admission-control# cat app/app.js
const bodyParser = require('body-parser');
const express = require('express');
const fs = require('fs');
const https = require('https');

const app = express();
app.use(bodyParser.json());
const port = 8443;

const options = {
  ca: fs.readFileSync('ca.crt'),
  cert: fs.readFileSync('server.crt'),
  key: fs.readFileSync('server.key'),
};

app.get('/hc', (req, res) => {
  res.send('ok');
});

app.post('/', (req, res) => {
  if (
    req.body.request === undefined ||
    req.body.request.uid === undefined
  ) {
    res.status(400).send();
    return;
  }
  console.log(req.body); // DEBUGGING
  const { request: { uid } } = req.body;
  res.send({
    apiVersion: 'admission.k8s.io/v1',
    kind: 'AdmissionReview',
    response: {
      uid,
      allowed: true,
    },
  });
});

const server = https.createServer(options, app);

server.listen(port, () => {
  console.log(`Server running on port ${port}/`);
});

root@VM-0-12-ubuntu:~/hello-dynamic-admission-control#
```

For further verification, you can change "allowed" to "false" and then repeat the above steps to rebuild a Webserver server image and redeploy `controller.yaml` and `admission.yaml` resources. If the request of your

©2013-2022 Tencent Cloud. All rights reserved.

reattempt to create pods resources is intercepted by the dynamic admission controller, then the configured dynamic admission policy has taken effect, as shown in the figure below:

Summary

This document mainly introduces the concept and functionality of the dynamic admission controller Webhook, as well as how to issue certificates needed by the dynamic admission controller in a TKE cluster. This document also describes a simple use case for configuring and using the dynamic admission Webhook feature.

References

- Kubernetes Dynamic Admission Control by Example
- Dynamic Admission Control

©2013-2022 Tencent Cloud. All rights reserved.

Hybrid Cloud
Elastic Scaling with EKS for IDC-Based Cluster

Last updated：2022-10-19 16:39:53

Use Cases

As your IDC resources may be limited, if you need to handle business traffic surges, the computing resources in your IDC may be insufficient to meet the requirements. In this case, you can use public cloud resources to handle temporary traffic. Based on custom scheduling policies and by leveraging TKE Serverless Container Service, TKE Resilience Chart adds supernodes to elastically migrate workloads in your IDC cluster to the cloud, so your cluster can get greater elastic scalability and enjoy the following benefits:

1. The hardware and maintenance costs of your IDC/private cloud do not increase.
2. You can implement high availability for applications at the IDC/private cloud grade and public cloud grade.
3. You can use public cloud resources as needed in a pay-as-you-go manner.

Notes

1. You have activated TKE Serverless cluster.
2. Your IDC is connected to a VPC through Direct Connect over the private network.
3. The address of the API server in the IDC cluster can be accessed over the VPC.
4. Your own IDC cluster can access the public network, as it needs to call TencentCloud APIs over the public network.

TKE Resilience Chart Feature Description

Component description

TKE Resilience Chart mainly consists of a supernode manager, scheduler, and toleration controller as detailed below:

Alias	Component Name	Description

©2013-2022 Tencent Cloud. All rights reserved.

Alias	Component Name	Description
eklet	Supernodes manager	It manages the lifecycle of PodSandboxes and provides APIs related to native kubelet and nodes.
tke-scheduler	Scheduler	It migrates workloads to the cloud elastically according to scheduling policies and is only installed in non-TKE Kubernetes Distro K8s clusters. TKE Kubernetes Distro is a K8s distribution released by TKE to help you create exactly the same K8s cluster in TKE. Currently, it has been open sourced at GitHub. For more information, please see TKE Kubernetes Distro.
admission-controller	Toleration controller	It adds a toleration to a Pod in `pending` status to make it able to be scheduled to a supernode.

Main features

1. If you want to connect an TKE Serverless Pod to a Pod in your local cluster, the local cluster should be in an underlay network model (where a CNI plugin based on BGP routing instead of SDN encapsulation, such as Calico, is used), and you need to add the local Pod's CIDR block routing information in the VPC. For more information, see Interconnection Between Cluster in GlobalRouter Mode and IDC
2. The workload resilience feature switch `AUTO_SCALE_EKS=true|false` is available in global and local dimensions respectively to control whether workloads in `pending` status should be elastically scheduled to EKS as detailed below:

- Global switch: `AUTO_SCALE_EKS` in `kubectl get cm -n kube-system eks-config` is enabled by default.
- Local switch: `spec.template.metadata.annotations ['AUTO_SCALE_EKS']`

Global Switch	Local Switch	Behavior
AUTO_SCALE_EKS=true	AUTO_SCALE_EKS=false	Successfully scheduled
AUTO_SCALE_EKS=true	Undefined	Successfully scheduled
AUTO_SCALE_EKS=true	AUTO_SCALE_EKS=true	Successfully scheduled
AUTO_SCALE_EKS=false	AUTO_SCALE_EKS=false	Failed to be scheduled
AUTO_SCALE_EKS=false	Undefined	Failed to be scheduled
AUTO_SCALE_EKS=false	AUTO_SCALE_EKS=true	Successfully scheduled
Undefined	AUTO_SCALE_EKS=false	Successfully scheduled

©2013-2022 Tencent Cloud. All rights reserved.

Global Switch	Local Switch	Behavior
Undefined	Undefined	Successfully scheduled
Undefined	AUTO_SCALE_EKS=true	Successfully scheduled

3. If you use K8s community edition, you need to specify the scheduler as `tke-scheduler` in workloads. In TKE Kubernetes Distro, you don't need to specify the scheduler.
4. In the workloads, set the number of retained replicas in the local cluster through `LOCAL_REPLICAS: N` .
5. Workload scale-out:

- If the local cluster resources are insufficient and the settings of the global and local switches for the **"successfully scheduled"** behavior are satisfied, workloads in `pending` status will be scaled out to EKS.
- If the number of actually created workload replicas reaches N and the settings of the global and local switches for the **"successfully scheduled"** behavior are satisfied, workloads in `pending` status will be scaled out to TKE Serverless cluster.

6. Workload scale-in:

- For TKE Kubernetes Distro, instances in TKE Serverless cluster will be scaled in preferentially.
- For K8s community edition, workloads will be scaled in randomly.

7. Scheduling rule restrictions:

- DaemonSet Pods cannot be scheduled to supernodes. This feature is available only in TKE Kubernetes Distro. In K8s community edition, DaemonSet Pods will be scheduled to supernodes but `DaemonsetForbidden` will be displayed.
- Pods in `kube-system` and `tke-eni-ip-webhook` namespaces cannot be scheduled to supernodes.
- Ports whose `securityContext.sysctls ["net.ipv4.ip_local_port_range"]` value includes 61000–65534 cannot be scheduled.
- Pods in `Pod.Annotations [tke.cloud.tencent.com/vpc-ip-claim-delete-policy]` cannot be scheduled.
- Ports whose `container (initContainer).ports [].containerPort (hostPort)` value includes 61000–65534 cannot be scheduled.
- Ports with a `container (initContainer)` where the probe points to 61000–65534 cannot be scheduled.
- PersistentVolumes (PVs) except nfs, Cephfs, hostPath, and qcloudcbs cannot be scheduled.
- Pods with fixed IP enabled cannot be scheduled to supernodes.

©2013-2022 Tencent Cloud. All rights reserved.

8. Supernodes support custom DNS configuration: after you add the `eks.tke.cloud.tencent.com/resolv-conf` annotation to a supernode, `/etc/resolv.conf` in the generated CVM instance will be updated to the custom content.

> Note：
> The original DNS configuration on the supernodes will be overwritten, and your custom configuration will prevail.

```
eks.tke.cloud.tencent.com/resolv-conf: |
nameserver 4.4.4.4
nameserver 8.8.8.8
```

Directions

Getting `tke-resilience helm chart`

```
git clone https://github.com/tkestack/charts.git
```

Configuring relevant information

Edit `charts/incubator/tke-resilience/values.yaml` and configure the following information:

```
cloud:
appID: "{Tencent Cloud account APPID}"
ownerUIN: "{Tencent Cloud account ID}"
secretID: "{Tencent Cloud account secretID}"
secretKey: "{Tencent Cloud account secretKey}"
vpcID: "{ID of the VPC where the EKS Pod resides}"
regionShort: "{Abbreviation of the region where the EKS Pod resides}"
regionLong: "{Full name of the region where the EKS Pod resides}"
subnets:
- id: "{ID of the subnet where the EKS Pod resides}"
zone: "{AZ where the EKS Pod resides}"
eklet:
PodUsedApiserver: "{API server address of the current cluster}"
```

> Note：

©2013-2022 Tencent Cloud. All rights reserved.

For more information on the regions and AZs where TKE Serverless container service is available, please see Regions and AZs.

Installing TKE Resilience Chart

You can use the local Helm client to connect to the cluster.

Run the following command to use a Helm chart to install TKE Resilience Chart in a third-party cluster:

```
helm install tke-resilience --namespace kube-system ./tke-resilience --debug
```

Run the following command to check whether the required components in the Helm application are installed. This document uses a TKE Kubernetes Distro cluster with no tke-scheduler installed as an example.

```
# kubectl get Pod -n kube-system | grep resilience
eklet-tke-resilience-5f9dcd99df-rgsmc 1/1 Running 0 43h
eks-admission-tke-resilience-5bb588dc44-9hvhs 1/1 Running 0 44h
```

You can see that one supernode has been deployed in the cluster.

```
# kubectl get node
NAME STATUS ROLES AGE VERSION
10.0.1.xx Ready <none> 2d4h v1.20.4-tke.1
10.0.1.xx Ready master 2d4h v1.20.4-tke.1
eklet-subnet-xxxxxxxx Ready <none> 43h v2.4.6
```

Creating test case

Create a demo application `nginx-deployment`, which has four replicas (three in TKE Serverless cluster and one in the local cluster). Below is the sample YAML configuration:

```
apiVersion: apps/v1
kind: Deployment
metadata:
name: nginx-deployment
labels:
app: nginx
spec:
replicas: 4
strategy:
type: RollingUpdate
selector:
matchLabels:
app: nginx
```

©2013-2022 Tencent Cloud. All rights reserved.

```
template:
metadata:
annotations:
AUTO_SCALE_EKS: "true"
LOCAL_REPLICAS: "1" # Set the number of running replicas in the local cluster to
1
labels:
app: nginx
spec:
#schedulerName: tke-scheduler If it is a third-party cluster, you need to run the
scheduler as `tke-scheduler`
containers:
- name: nginx
image: nginx
imagePullPolicy: IfNotPresent
```

Check whether the replica status and distribution meet the expectations.

```
# kubectl get Pod -owide
NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES
nginx-deployment-77b9b9bc97-cq9ds 1/1 Running 0 27s 10.232.1.88 10.0.1.xxx <none>
<none>
nginx-deployment-77b9b9bc97-s9vzc 1/1 Running 0 27s 10.0.1.118 eklet-subnet-xxxxx
xxx <none> <none>
nginx-deployment-77b9b9bc97-sd4z5 1/1 Running 0 27s 10.0.1.7 eklet-subnet-xxxxxxx
x <none> <none>
nginx-deployment-77b9b9bc97-z86tx 1/1 Running 0 27s 10.0.1.133 eklet-subnet-xxxxx
xxx <none> <none>
```

Check the scale-in feature. As a TKE Kubernetes Distro cluster is used, TKE Serverless cluster instances will be scaled in preferentially. Here, the number of application replicas is adjusted from 4 to 3.

```
# kubectl scale deployment nginx-deployment --replicas=3
```

As shown below, replicas in Tencent Cloud are scaled in first, which meets the expectation:

```
# kubectl get Pod -owide
NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES
nginx-deployment-77b9b9bc97-cq9ds 1/1 Running 0 7m38s 10.232.1.88 10.0.1.xxx <non
e> <none>
nginx-deployment-77b9b9bc97-s9vzc 1/1 Running 0 7m38s 10.0.1.118 eklet-subnet-xxx
xxxxx <none> <none>
nginx-deployment-77b9b9bc97-sd4z5 1/1 Running 0 7m38s 10.0.1.7 eklet-subnet-xxxxx
xxx <none> <none>
```

©2013-2022 Tencent Cloud. All rights reserved.

Network
DNS
Best Practices of TKE DNS

Last updated：2023-05-06 17:36:46

Overview

As DNS is the first step in service access in a Kubernetes cluster, its stability and performance are of great importance. How to configure and use DNS in a better way involves many aspects. This document describes the best practices of DNS.

Selecting the Most Appropriate CoreDNS Version

The following table lists the default CoreDNS versions deployed in TKE clusters on different versions.

TKE Version	CoreDNS version
v1.22	v1.8.4
v1.20	v1.8.4
v1.18	v1.7.0
v1.16	v1.6.2
v1.14	v1.6.2

Due to historical reasons, CoreDNS v1.6.2 may still be deployed in clusters on v1.18 or later. If the current CoreDNS version doesn't meet your requirements, you can manually upgrade it as follows:
Upgrade to v1.7.0
Upgrade to v1.8.4

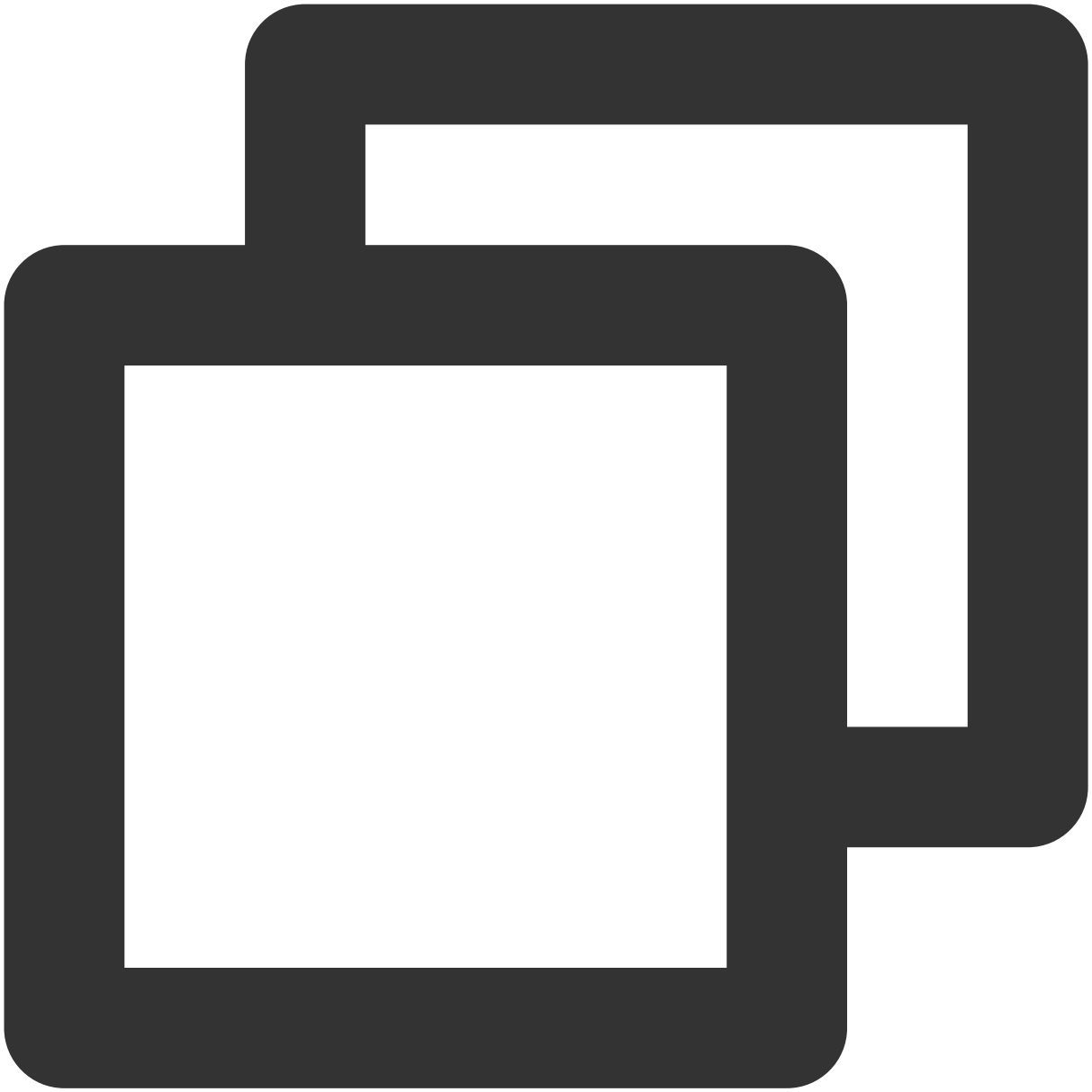
Configuring an Appropriate Number of CoreDNS Replicas

1. The default number of CoreDNS replicas in TKE is 2, and `podAntiAffinity` is configured to deploy the two replicas on different nodes.

©2013-2022 Tencent Cloud. All rights reserved.

2. If your cluster has more than 80 nodes, we recommend you install NodeLocal DNSCache as instructed in Using NodeLocal DNS Cache in a TKE Cluster.
3. Generally, you can determine the number of CoreDNS replicas based on the QPS of business access to DNS, number of nodes, or total number of CPU cores. After you install NodeLocal DNSCache, we recommend you use up to ten CoreDNS replicas. You can configure the number of replicas as follows:
Number of replicas = min (max (ceil (QPS/10000), ceil (number of cluster nodes/8)), 10)
Example:
If the cluster has ten nodes and the QPS of DNS service requests is 22,000, configure the number of replicas to 3.
If the cluster has 30 nodes and the QPS of DNS service requests is 15,000, configure the number of replicas to 4.
If the cluster has 100 nodes and the QPS of DNS service requests is 50,000, configure the number of replicas to 10 (NodeLocal DNSCache has been deployed).
4. You can install the DNSAutoScaler add-on in the console to automatically adjust the number of CoreDNS replicas (smooth upgrade should be configured in advance). Below is its default configuration:

©2013-2022 Tencent Cloud. All rights reserved.



```
data:
  ladder: |-
 {
   "coresToReplicas":
   [
     [ 1, 1 ],
     [ 128, 3 ],
     [ 512,4 ],
   ],
   "nodesToReplicas":
   [
```

©2013-2022 Tencent Cloud. All rights reserved.

```
      [ 1, 1 ],
      [ 2, 2 ]
    ]
  }
```

Using NodeLocal DNSCache

NodeLocal DNSCache can be deployed in a TKE cluster to improve the service discovery stability and performance. It improves cluster DNS performance by running a DNS caching agent on cluster nodes as a DaemonSet.
For more information on NodeLocal DNSCache and how to deploy NodeLocal DNSCache in a TKE cluster, see Using NodeLocal DNS Cache in a TKE Cluster.

Configuring CoreDNS Smooth Upgrade

During node restart or CoreDNS upgrade, some CoreDNS replicas may be unavailable for a period of time. You can configure the following items to maximize the DNS service availability and implement smooth upgrade.

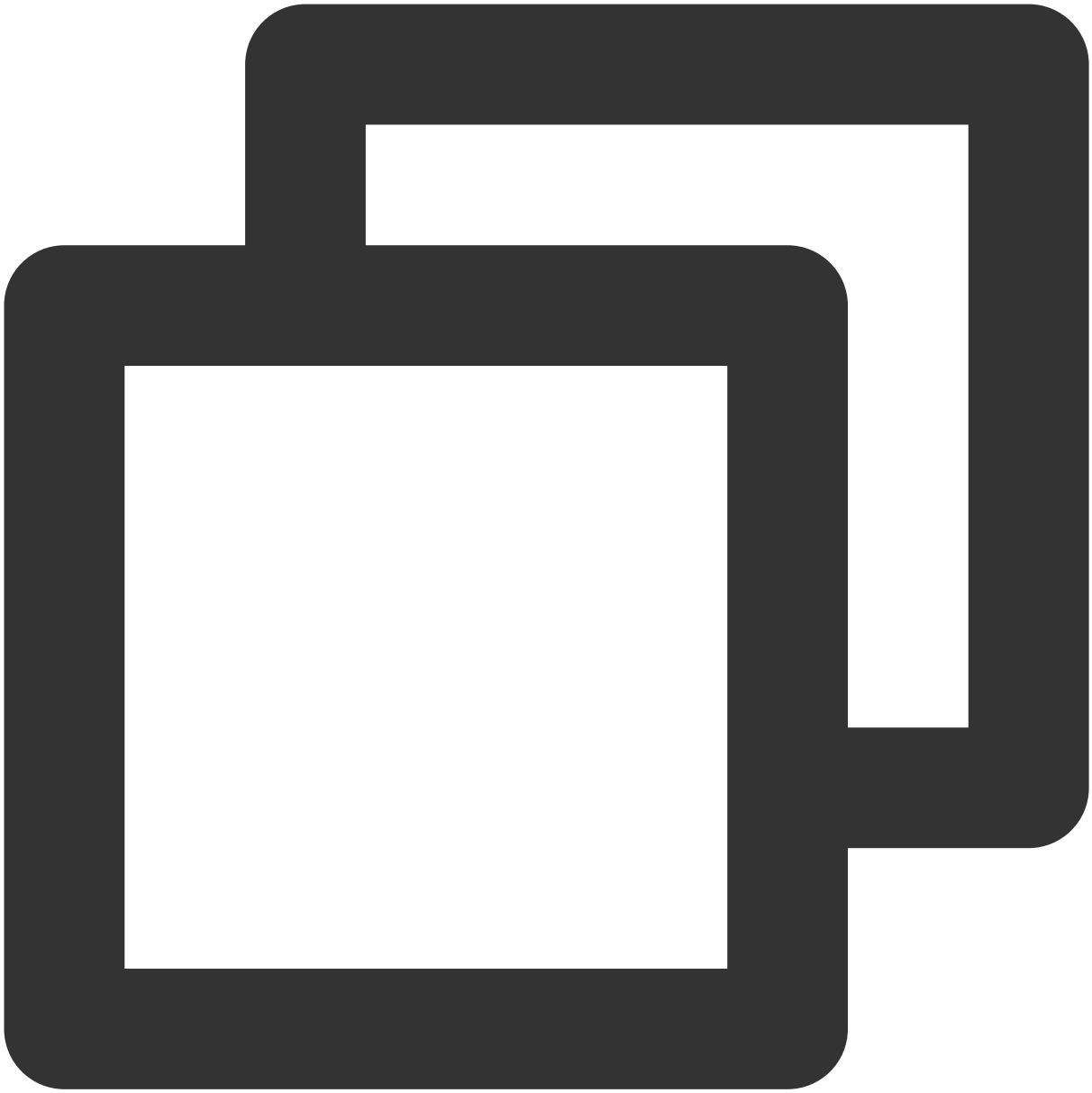
No configuration required in iptables mode

If kube-proxy adopts the iptables mode, kube-proxy clears conntrack entries after iptables rule synchronization, leaving no session persistence problem and requiring no configuration.

Configuring the session persistence timeout period of the IPVS UDP protocol in IPVS mode

If kube-proxy adopts the IPVS mode and the business itself doesn't provide the UDP service, you can reduce the session persistence timeout period of the IPVS UDP protocol to minimize the service unavailability.
1. If the cluster is on v1.18 or later, kube-proxy provides the `--ipvs-udp-timeout` parameter with the default value of `0s`, or the system default value `300s` can be used. We recommend you specify `--ipvs-udp-timeout=10s`. Configure the kube-proxy DaemonSet as follows:

©2013-2022 Tencent Cloud. All rights reserved.

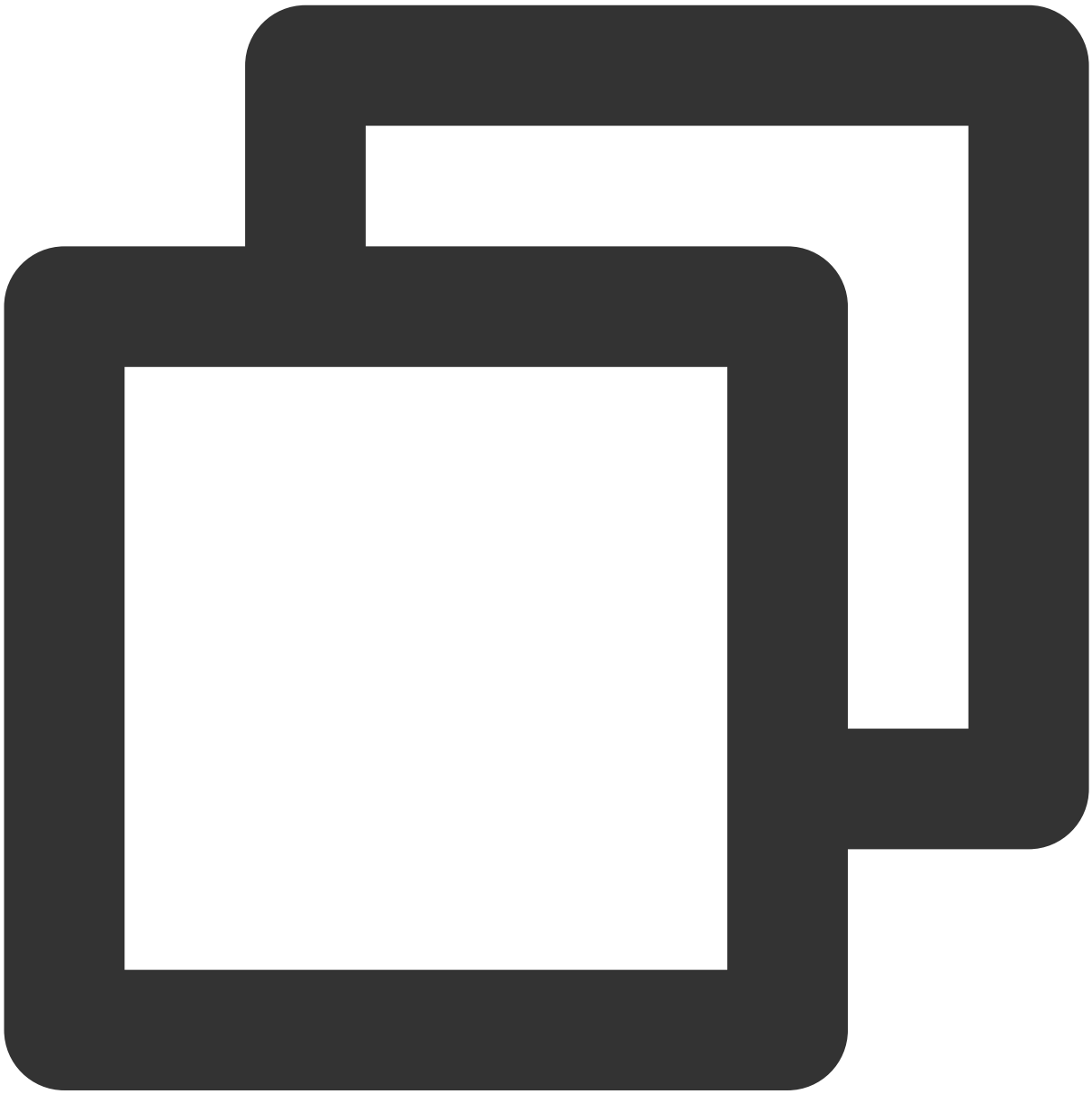


```
spec:
  containers:
  - args:
    - --kubeconfig=/var/lib/kube-proxy/config
    - --hostname-override=$(NODE_NAME)
    - --v=2
    - --proxy-mode=ipvs
    - --ipvs-scheduler=rr
    - --nodeport-addresses=$(HOST_IP)/32
    - --ipvs-udp-timeout=10s
    command:
```

©2013-2022 Tencent Cloud. All rights reserved.

```
      - kube-proxy
      name: kube-proxy
```

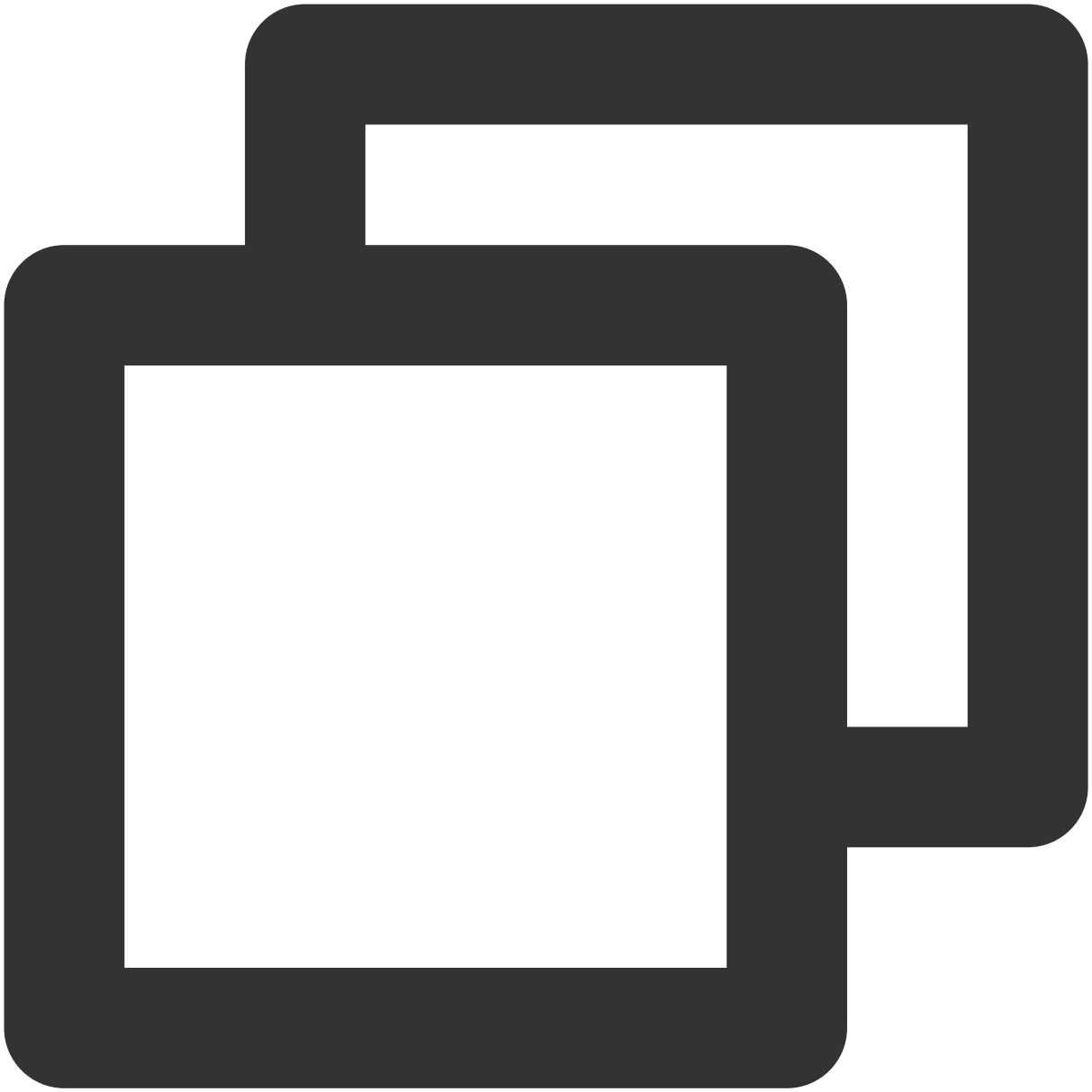
2. If the cluster is on v1.16 or earlier, kube-proxy doesn't support this parameter, and you can use the `ipvsadm` tool to batch modify the information on nodes as follows:



```
yum install -y ipvsadm
ipvsadm --set 900 120 10
```

3. After completing the configuration, verify the result as follows:

©2013-2022 Tencent Cloud. All rights reserved.



```
ipvsadm -L --timeout
Timeout (tcp tcpfin udp): 900 120 10
```

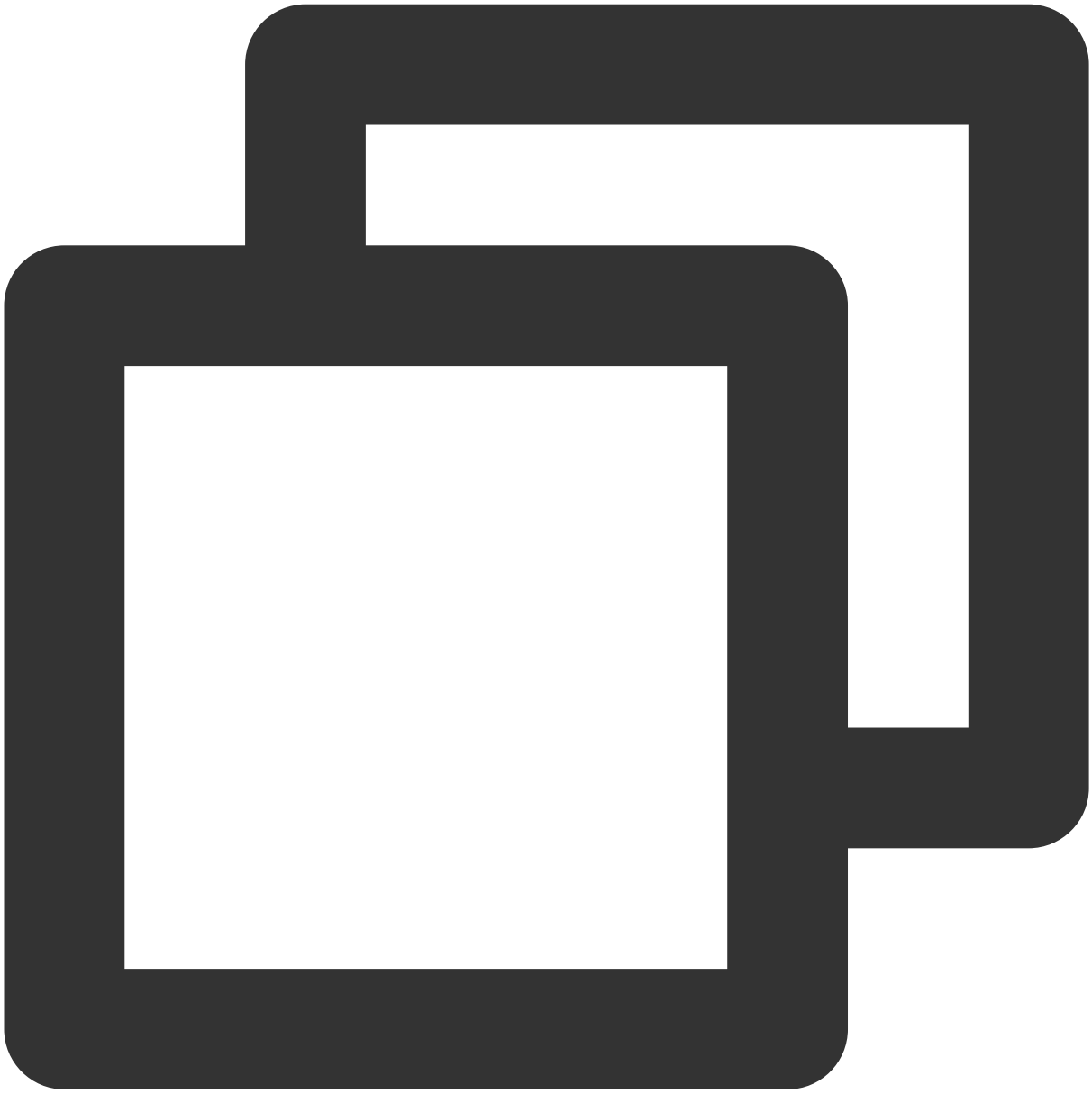
Note

After completing the configuration, you need to wait for five minutes before proceeding to the subsequent steps. If your business uses the UDP service, submit a ticket for assistance.

Configuring graceful shutdown for CoreDNS

©2013-2022 Tencent Cloud. All rights reserved.

You can configure `lameduck` to make replicas that have already received a shutdown signal continue providing the service for a certain period of time. Configure the CoreDNS ConfigMap as follows. Below is only a part of the configuration of CoreDNS v1.6.2. For information about the configuration of other versions, see Manual Upgrade.



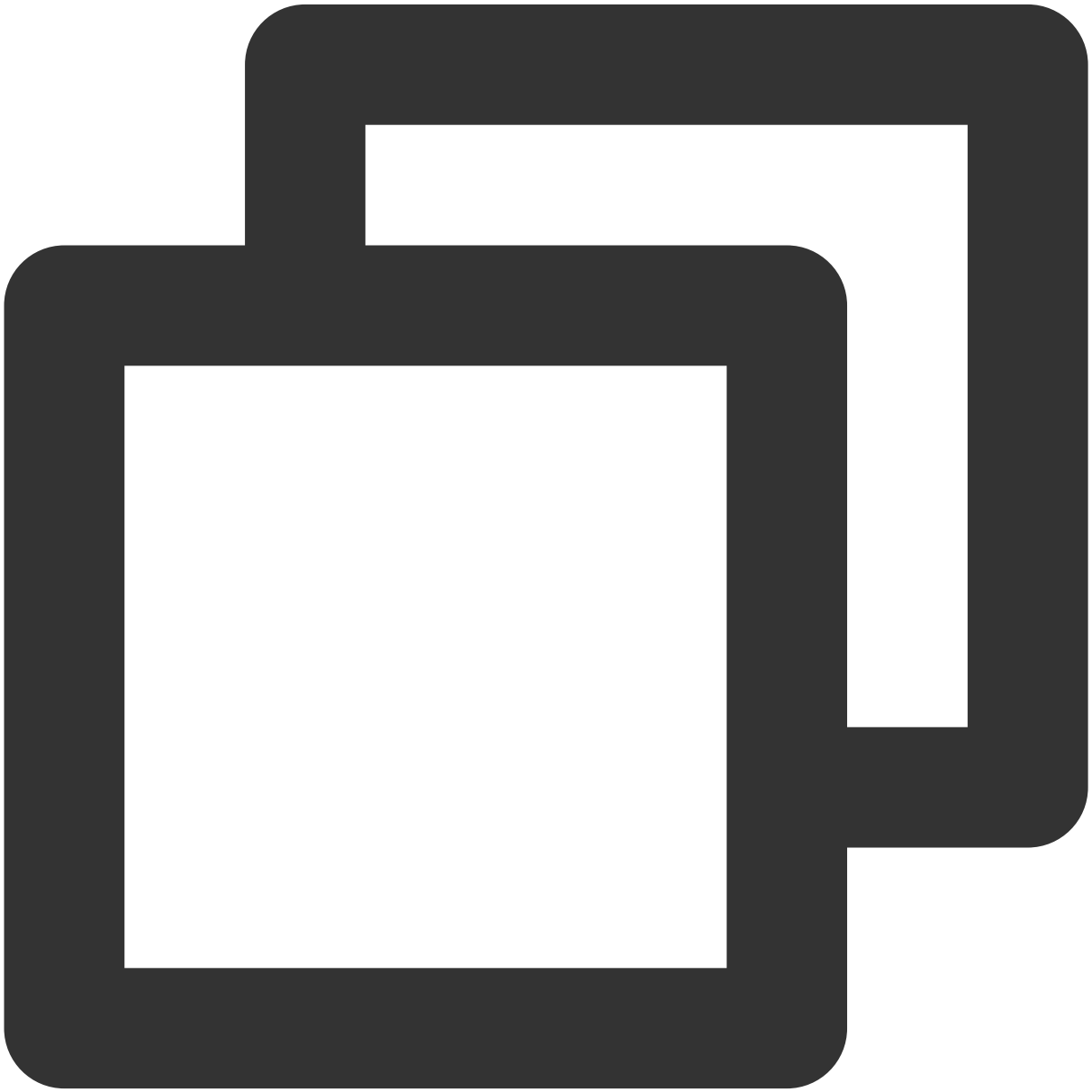
```
.:53 {
    health {
        lameduck 30s
    }
    kubernetes cluster.local. in-addr.arpa ip6.arpa {
        pods insecure
        upstream
```

©2013-2022 Tencent Cloud. All rights reserved.

```
                fallthrough in-addr.arpa ip6.arpa
            }
        }
```

Configuring CoreDNS service readiness confirmation

After a new replica starts, you need to check its service readiness and add it to the backend list of the DNS service.
1. Open the `ready` plugin and configure the CoreDNS ConfigMap as follows. Below is only a part of the configuration of CoreDNS v1.6.2. For information about the configuration of other versions, see Manual Upgrade.

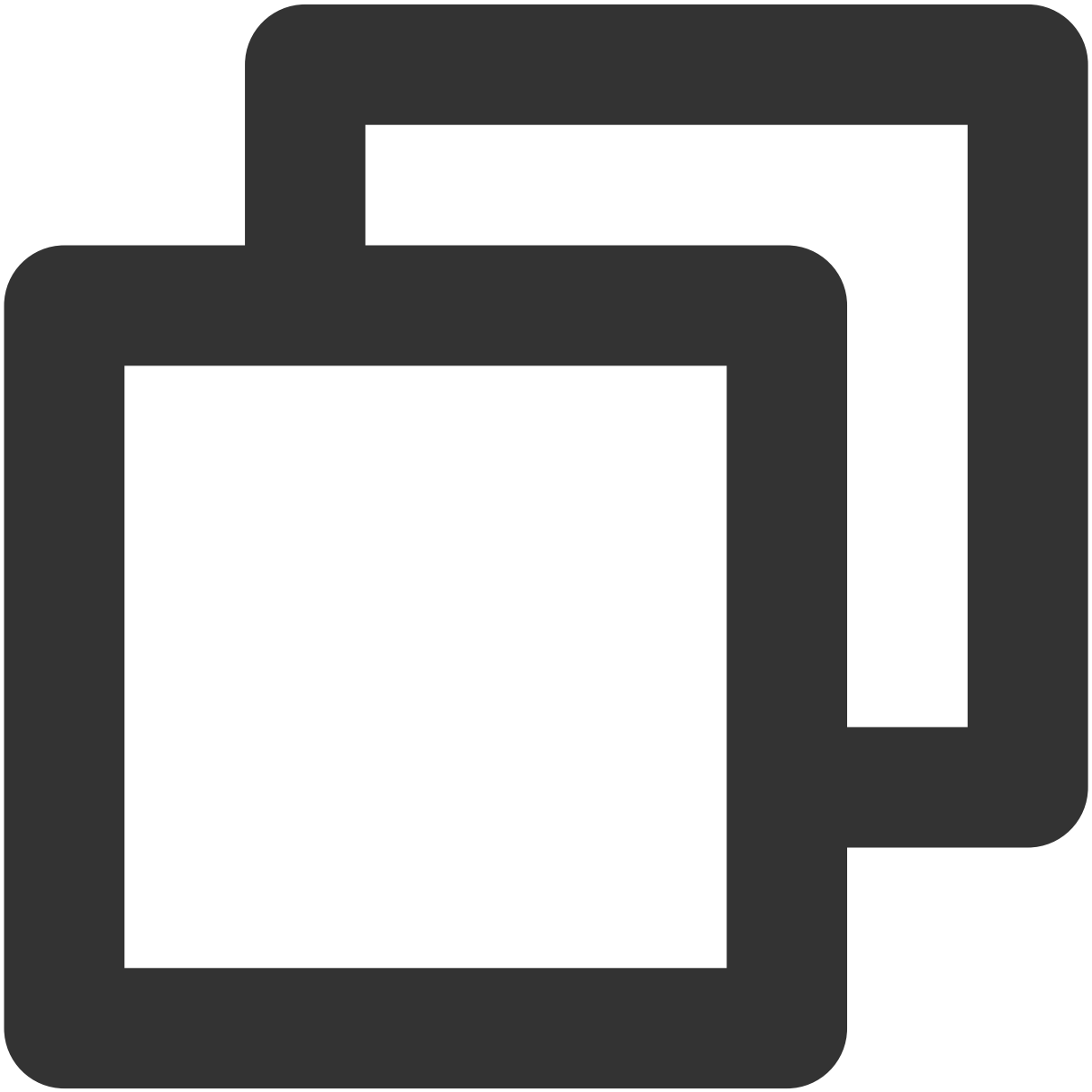


```
        .:53 {
```

©2013-2022 Tencent Cloud. All rights reserved.

```
            ready
            kubernetes cluster.local. in-addr.arpa ip6.arpa {
                pods insecure
                upstream
                fallthrough in-addr.arpa ip6.arpa
            }
        }
```

2. Add the `ReadinessProbe` configuration for CoreDNS:



```
readinessProbe:
```

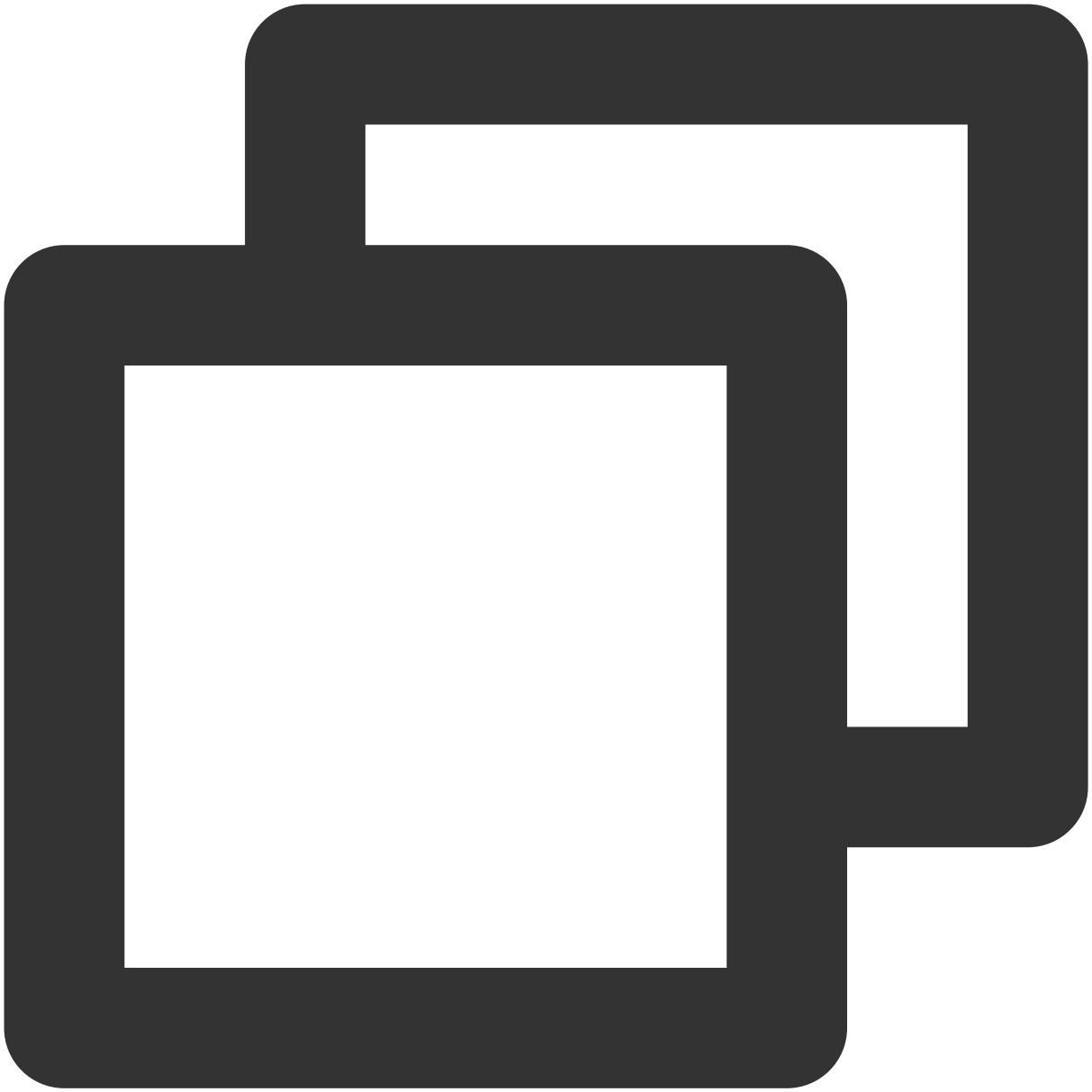
©2013-2022 Tencent Cloud. All rights reserved.

```
    failureThreshold: 5
    httpGet:
      path: /ready
      port: 8181
      scheme: HTTP
    initialDelaySeconds: 30
    periodSeconds: 10
    successThreshold: 1
    timeoutSeconds: 5
```

Configuring CoreDNS to Access Upstream DNS over UDP

If CoreDNS needs to communicate with the DNS server, it will use the client request protocol (UDP or TCP) by default. However, in TKE, the upstream service of CoreDNS is the DNS service in the VPC by default, which offers limited support for TCP. Therefore, we recommend you configure using UDP as follows (especially when NodeLocal DNSCache is installed):

©2013-2022 Tencent Cloud. All rights reserved.

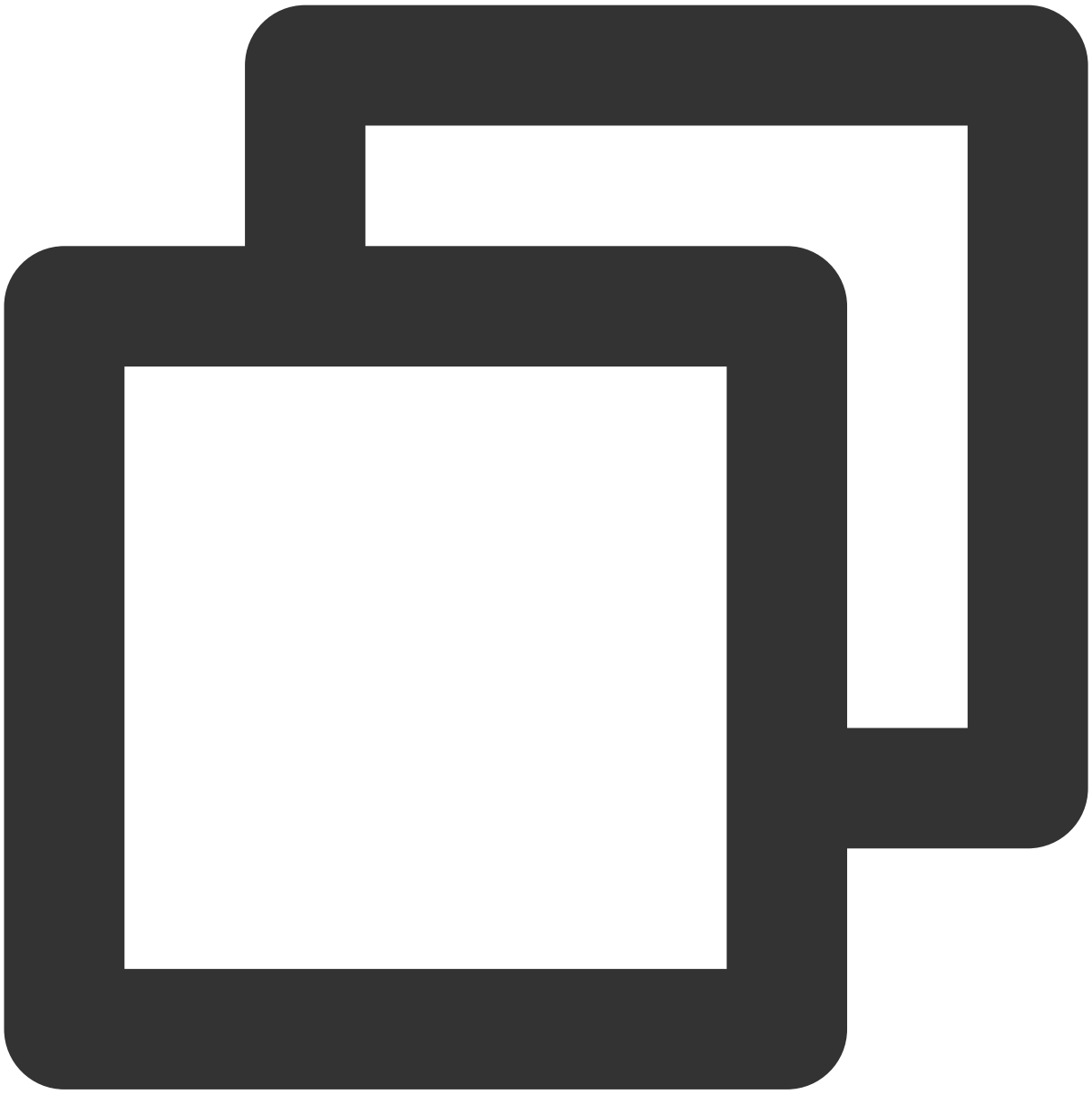


```
.:53 {
    forward . /etc/resolv.conf {
        prefer_udp
    }
}
```

Configuring CoreDNS to Filter HINFO Requests

©2013-2022 Tencent Cloud. All rights reserved.

As the DNS service in the VPC doesn't support DNS requests of the HINFO type, we recommend you configure as follows to filter such requests on the CoreDNS side (especially when NodeLocal DNSCache is installed):

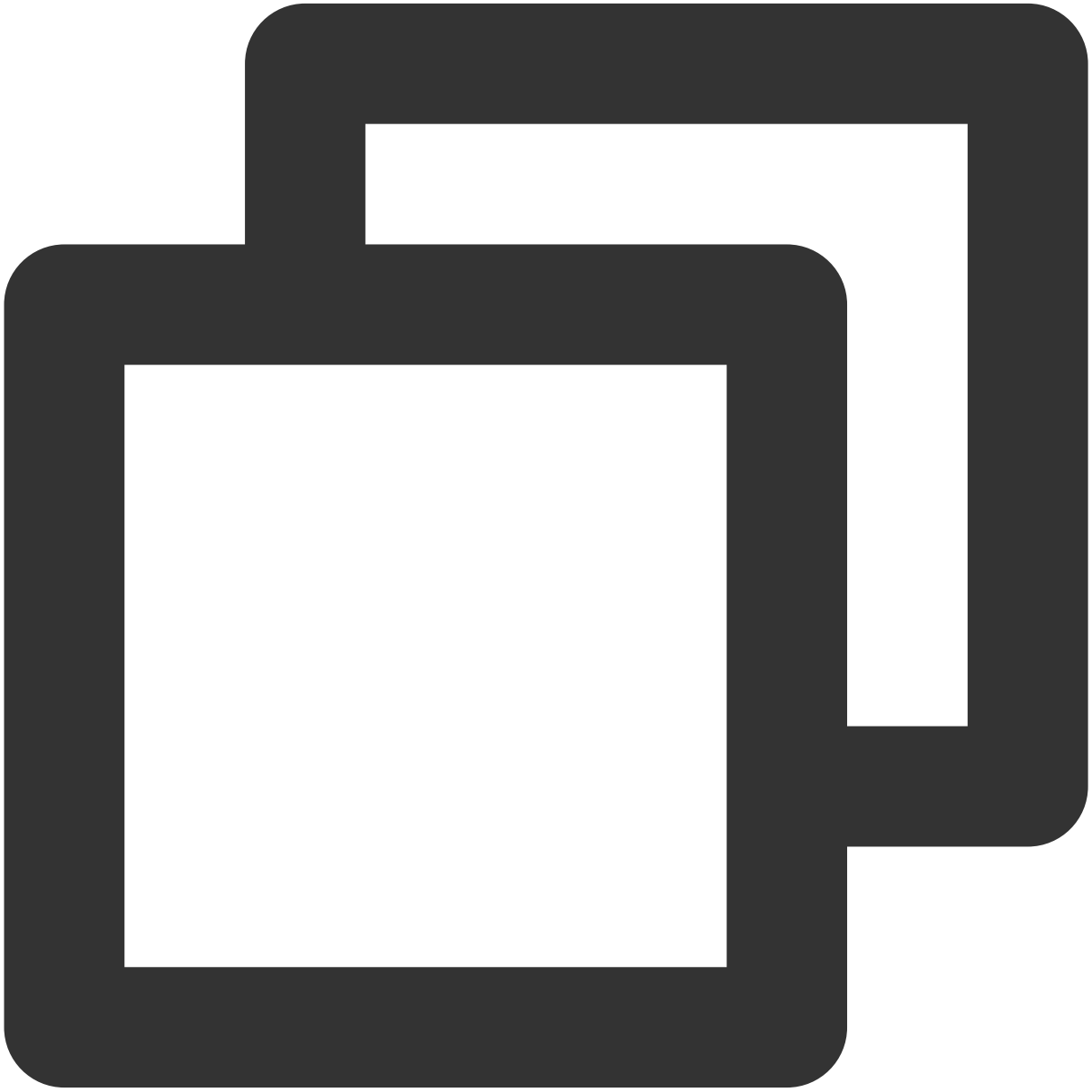


```
.:53 {
    template ANY HINFO . {
        rcode NXDOMAIN
    }
}
```

©2013-2022 Tencent Cloud. All rights reserved.

Configuring CoreDNS to Return "The domain name doesn't exist" for IPv6 AAAA Record Queries

If the business doesn't need to resolve IPv6 domain names, you can configure as follows to reduce the communication costs:



```
.:53 {
    template ANY AAAA {
        rcode NXDOMAIN
    }
```

©2013-2022 Tencent Cloud. All rights reserved.

```
            }
```

Note

Do not use this configuration in IPv4/IPv6 dual-stack clusters.

Configuring Custom Domain Name Resolution

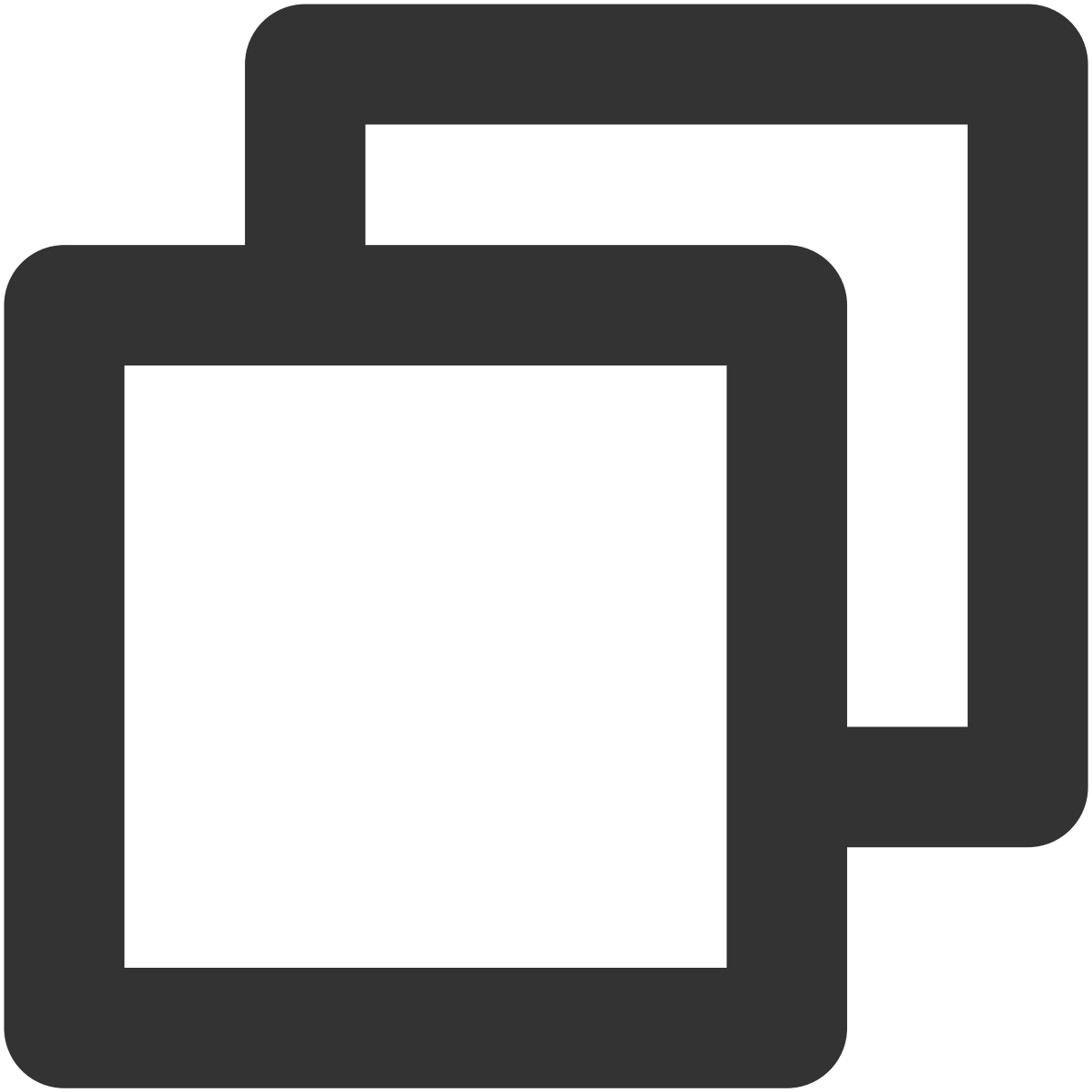
For more information, see Implementing Custom Domain Name Resolution in TKE.

Manual Upgrade

Upgrading to v1.7.0

1. Edit the `coredns` ConfigMap.

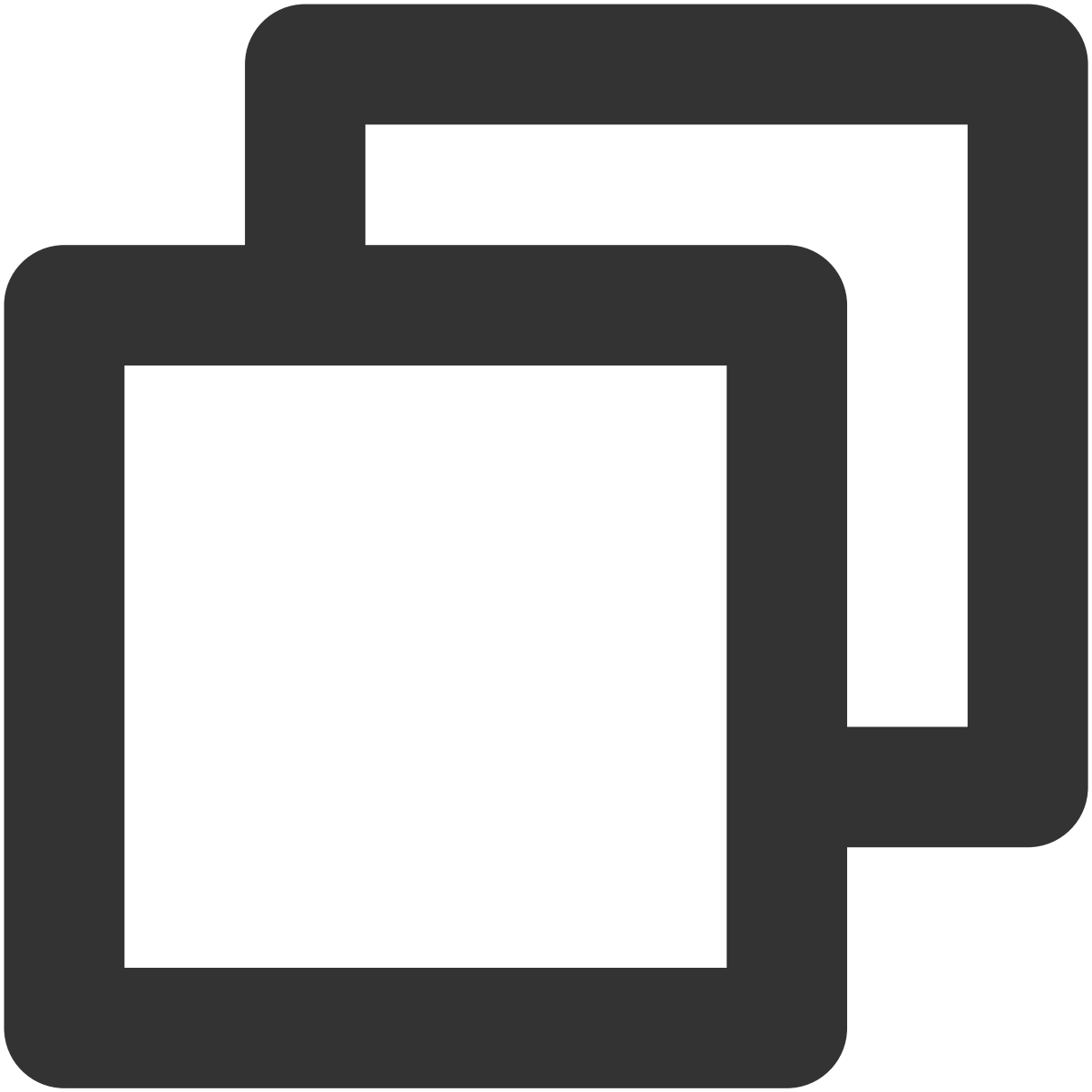
©2013-2022 Tencent Cloud. All rights reserved.



```
kubectl edit cm coredns -n kube-system
```

Modify the content as follows:

©2013-2022 Tencent Cloud. All rights reserved.



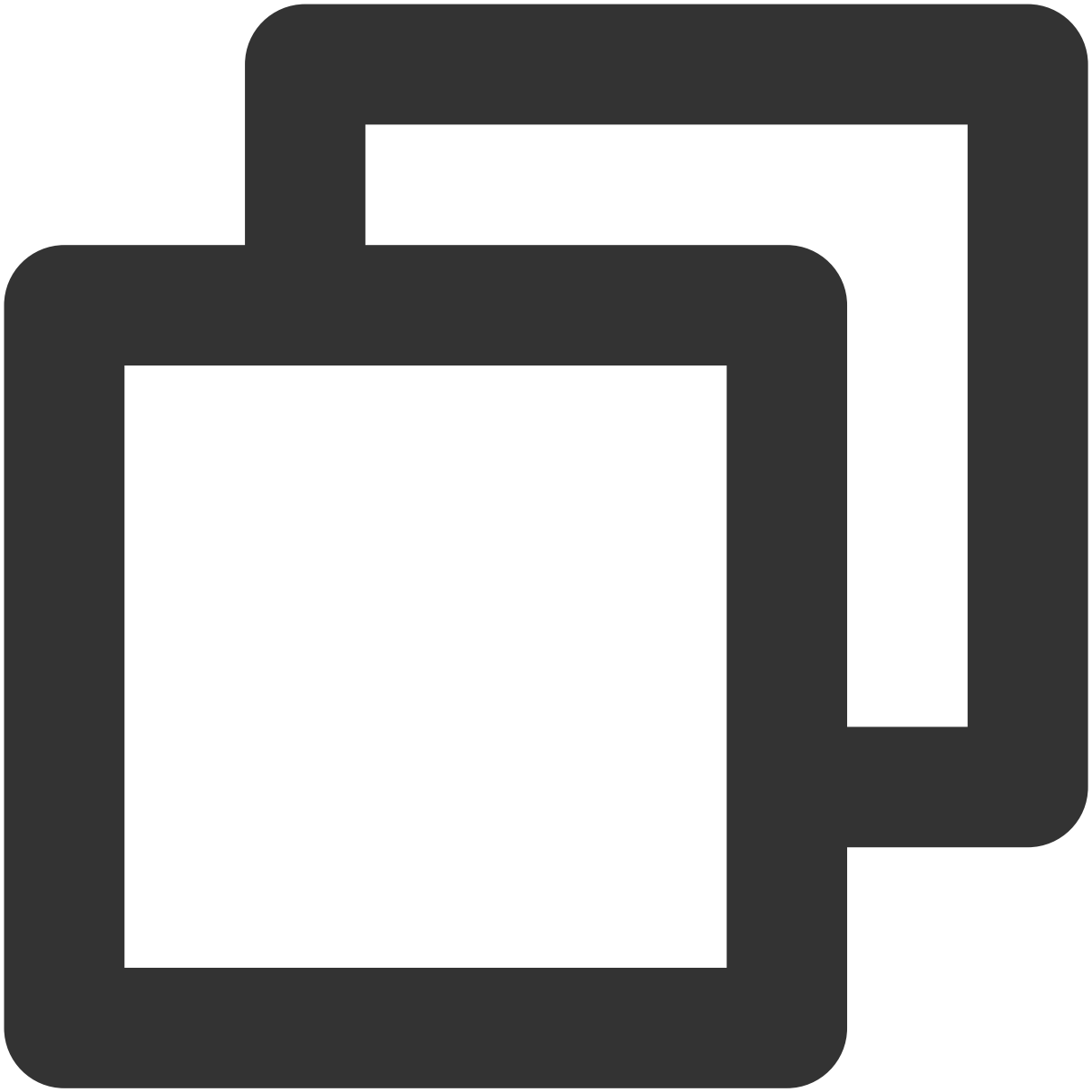
```
.:53 {
    template ANY HINFO . {
        rcode NXDOMAIN
    }
    errors
    health {
        lameduck 30s
    }
    ready
    kubernetes cluster.local. in-addr.arpa ip6.arpa {
        pods insecure
```

©2013-2022 Tencent Cloud. All rights reserved.

```
            fallthrough in-addr.arpa ip6.arpa
        }
        prometheus :9153
        forward . /etc/resolv.conf {
            prefer_udp
        }
        cache 30
        reload
        loadbalance
    }
```

2. Edit the `coredns` Deployment.

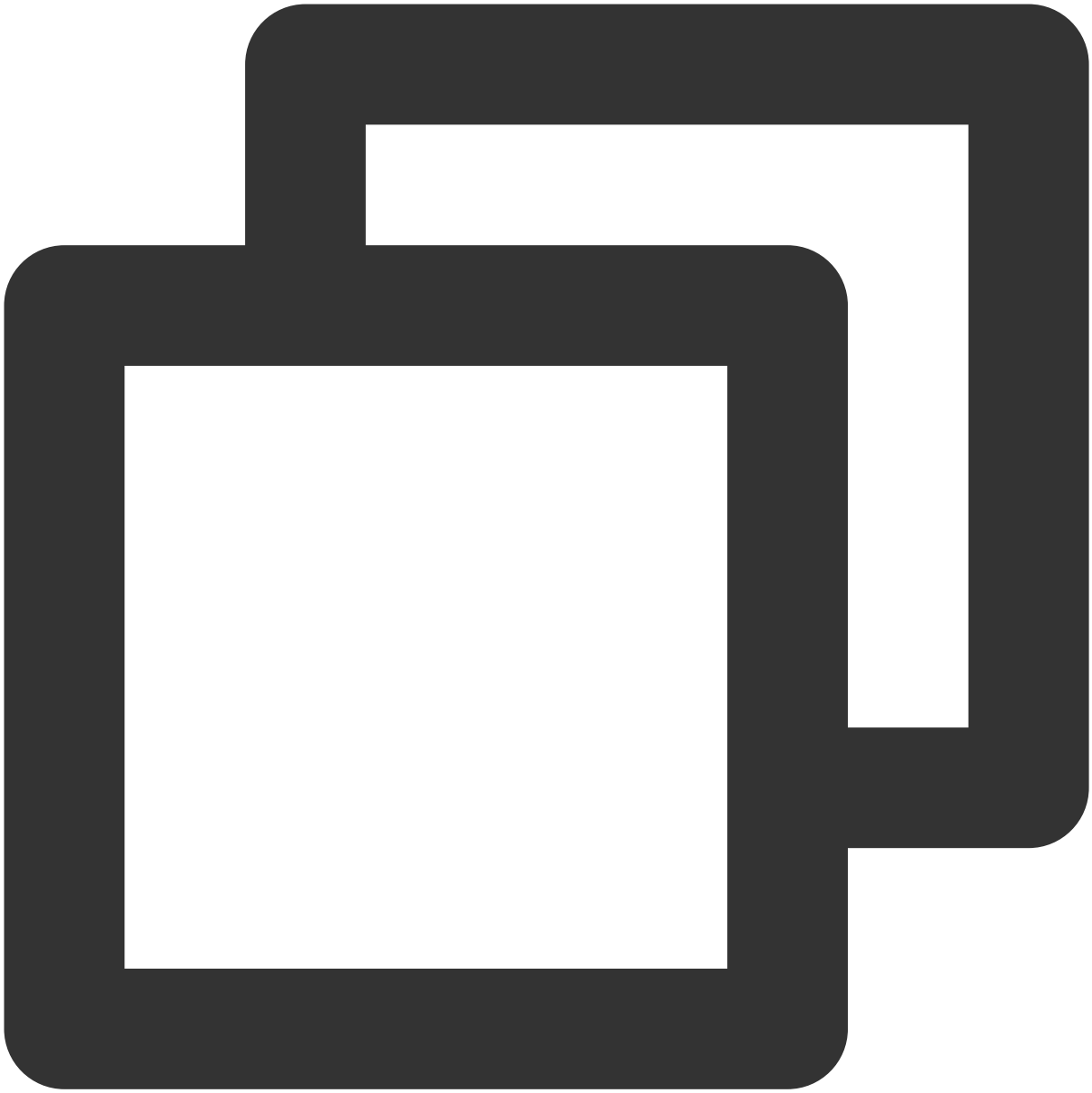
©2013-2022 Tencent Cloud. All rights reserved.



```
kubectl edit deployment coredns -n kube-system
```

Replace the image as follows:

©2013-2022 Tencent Cloud. All rights reserved.

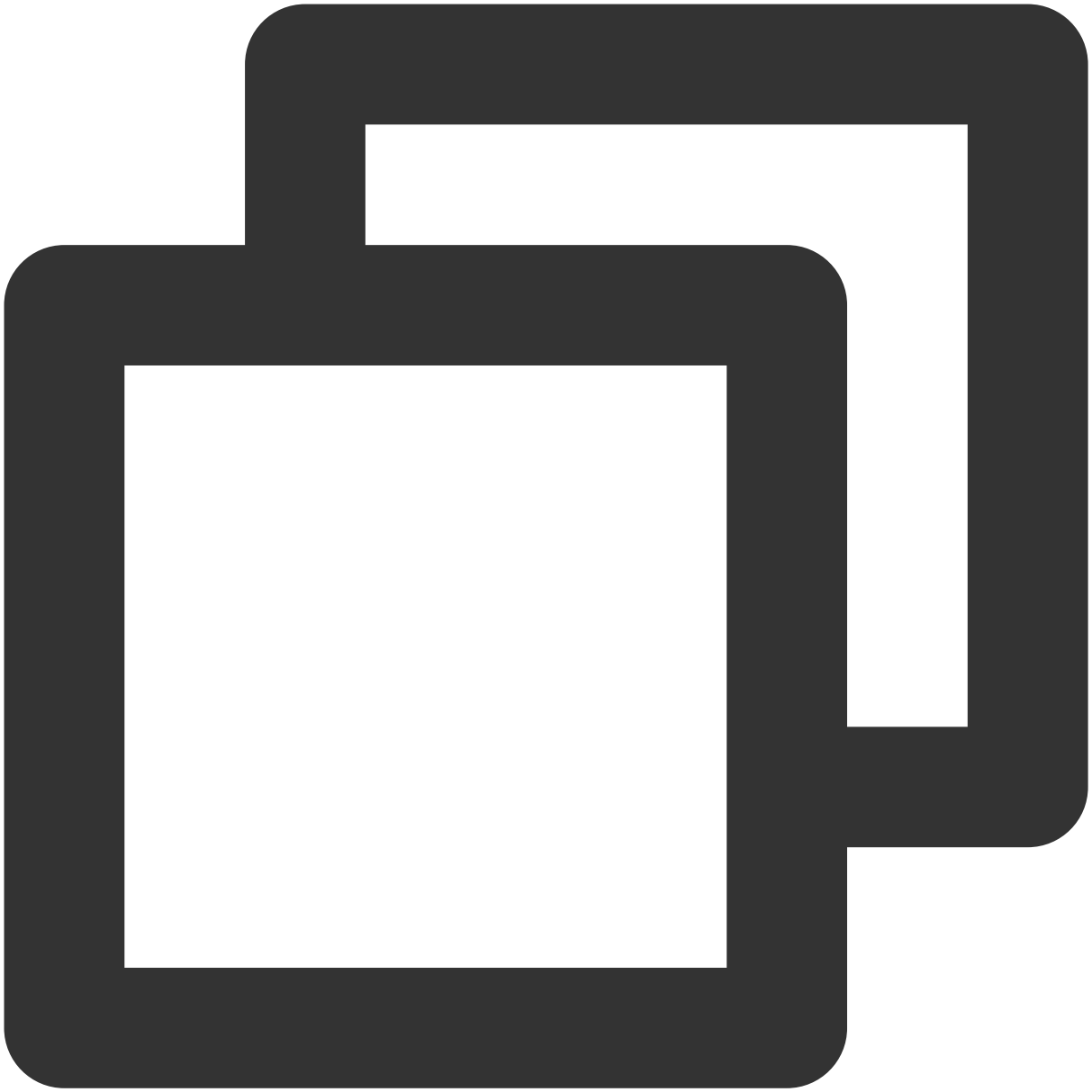


```
image: ccr.ccs.tencentyun.com/tkeimages/coredns:1.7.0
```

Upgrading to v1.8.4

1. Edit the `coredns` ClusterRole.

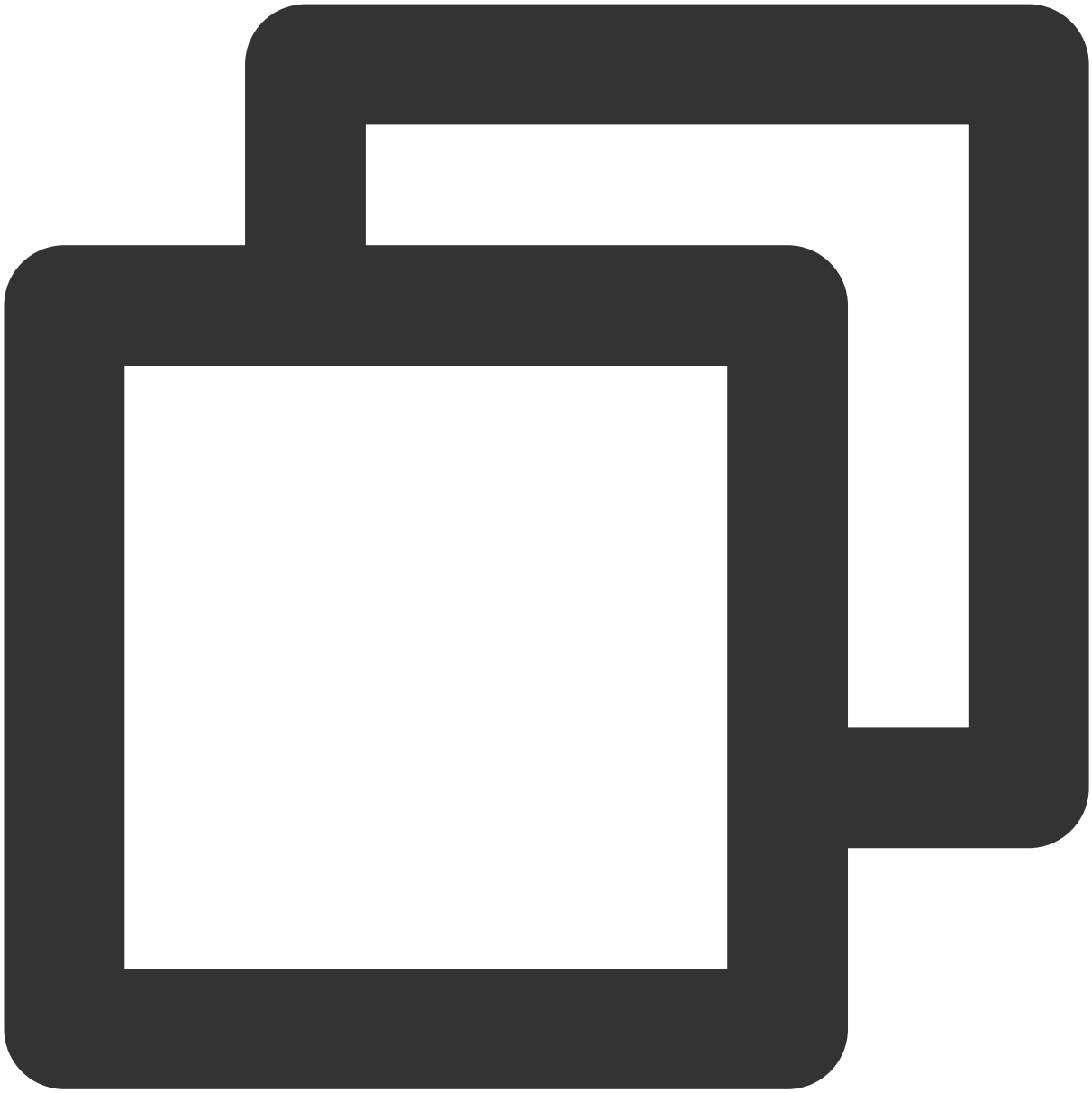
©2013-2022 Tencent Cloud. All rights reserved.



```
kubectl edit clusterrole system:coredns
```

Modify the content as follows:

©2013-2022 Tencent Cloud. All rights reserved.

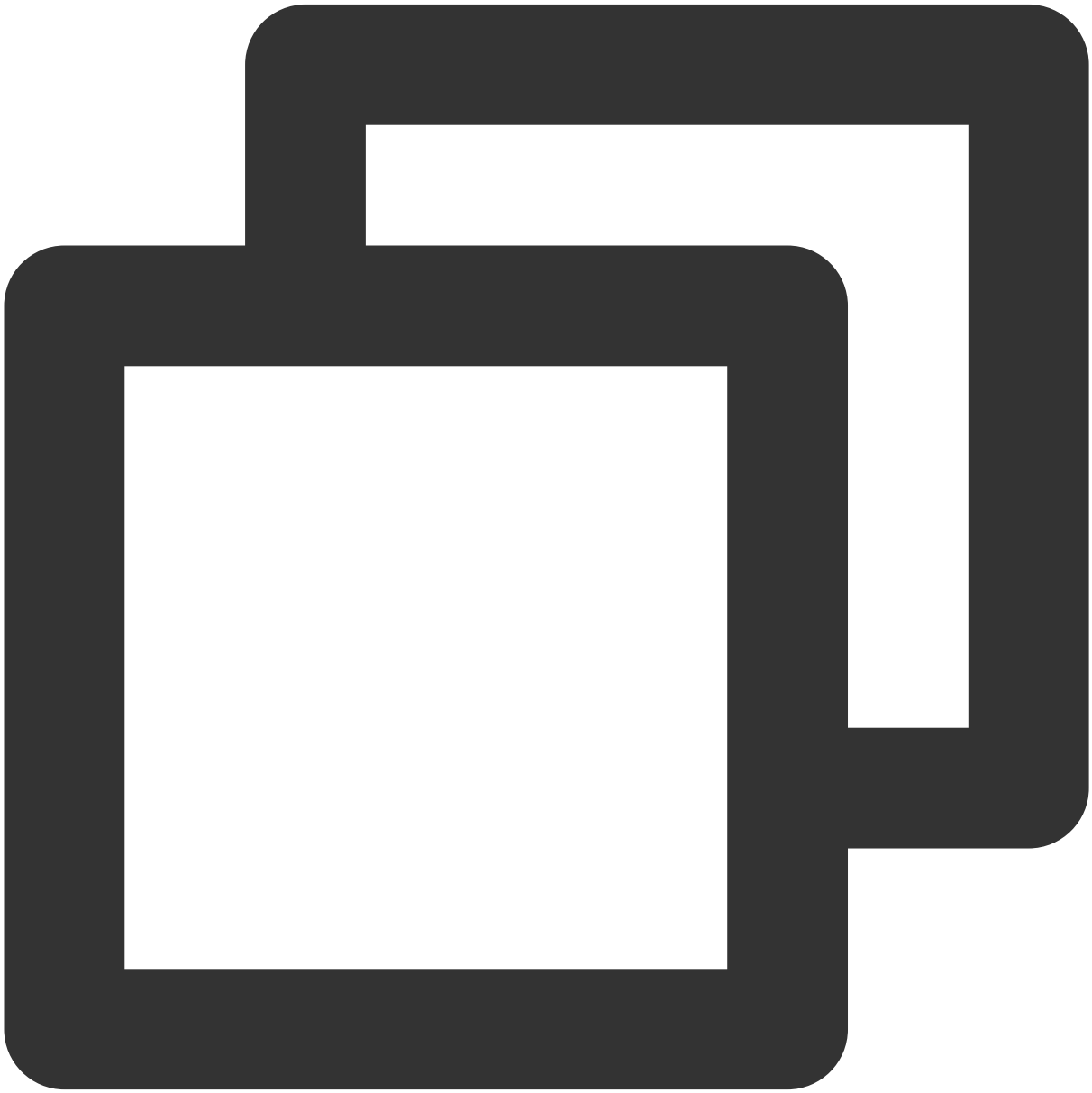


```
rules:
- apiGroups:
  - '*'
  resources:
  - endpoints
  - services
  - pods
  - namespaces
  verbs:
  - list
  - watch
```

©2013-2022 Tencent Cloud. All rights reserved.

```
- apiGroups:
  - discovery.k8s.io
  resources:
  - endpointslices
  verbs:
  - list
  - watch
```

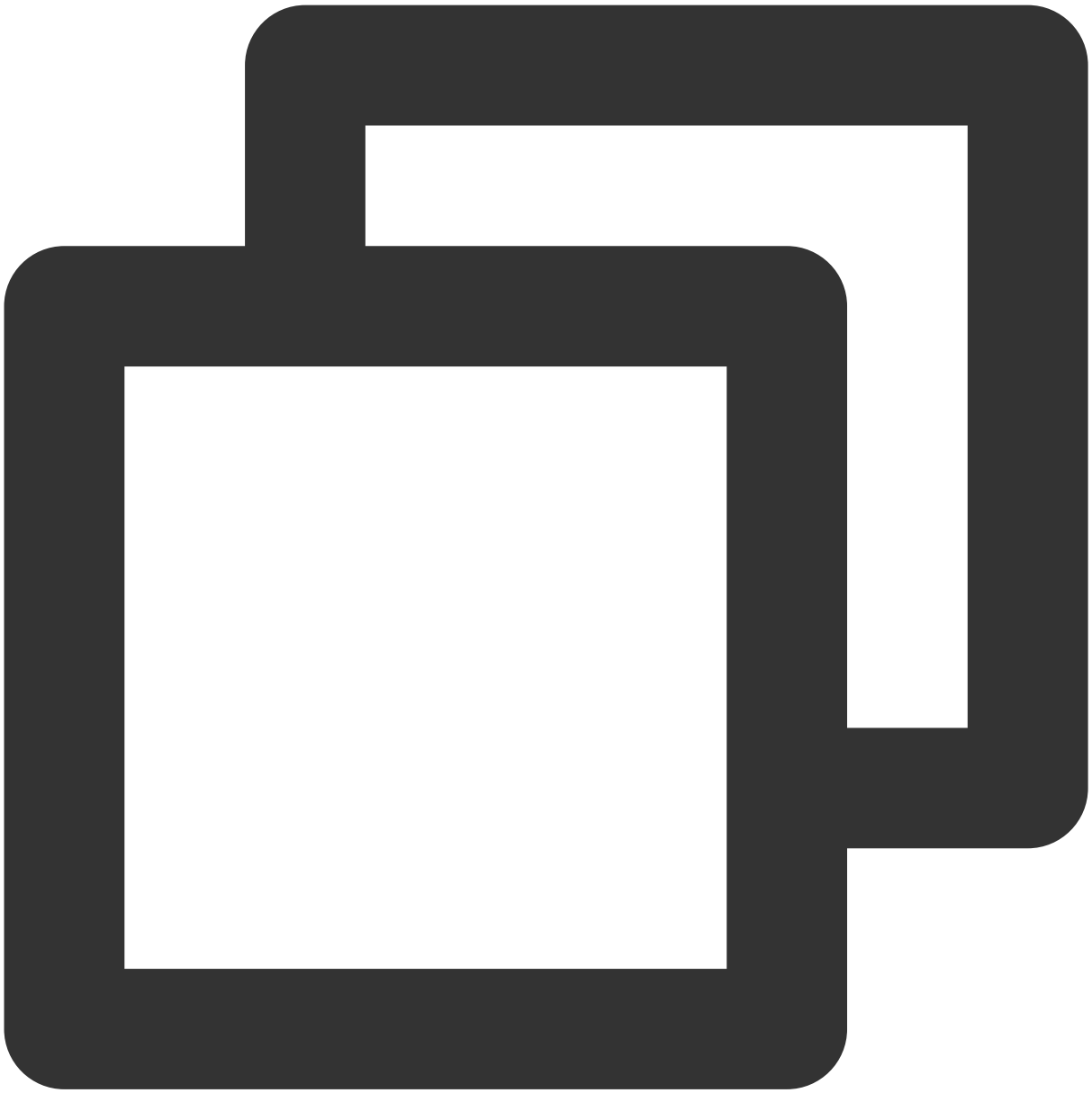
2. Edit the `coredns` ConfigMap.



```
kubectl edit cm coredns -n kube-system
```

©2013-2022 Tencent Cloud. All rights reserved.

Modify the content as follows:



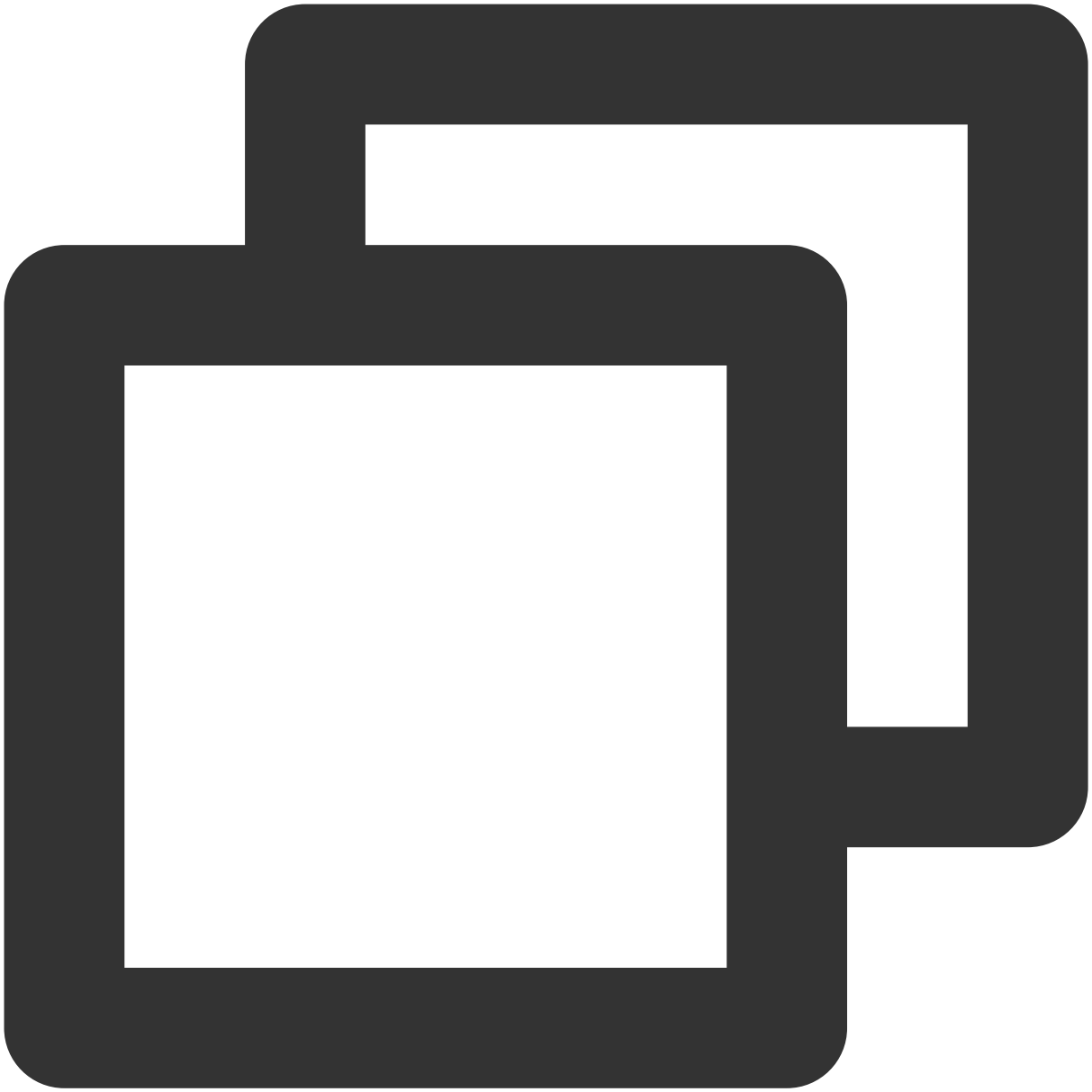
```
.:53 {
    template ANY HINFO . {
        rcode NXDOMAIN
    }
    errors
    health {
        lameduck 30s
    }
    ready
```

©2013-2022 Tencent Cloud. All rights reserved.

```
        kubernetes cluster.local. in-addr.arpa ip6.arpa {
            pods insecure
            fallthrough in-addr.arpa ip6.arpa
        }
        prometheus :9153
        forward . /etc/resolv.conf {
            prefer_udp
        }
        cache 30
        reload
        loadbalance
    }
```

3. Edit the `coredns` Deployment.

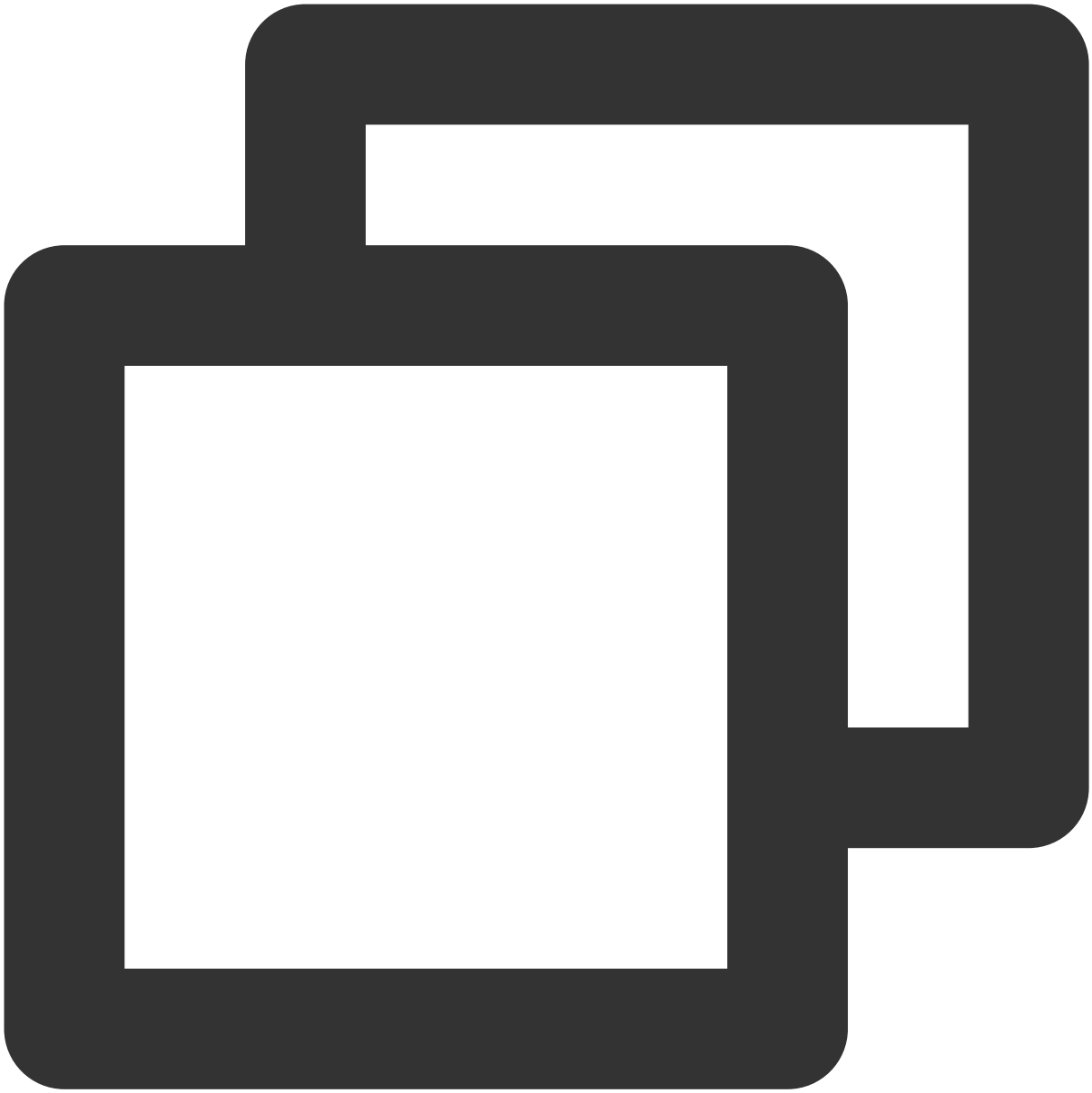
©2013-2022 Tencent Cloud. All rights reserved.



```
kubectl edit deployment coredns -n kube-system
```

Replace the image as follows:

©2013-2022 Tencent Cloud. All rights reserved.



```
image: ccr.ccs.tencentyun.com/tkeimages/coredns:1.8.4
```

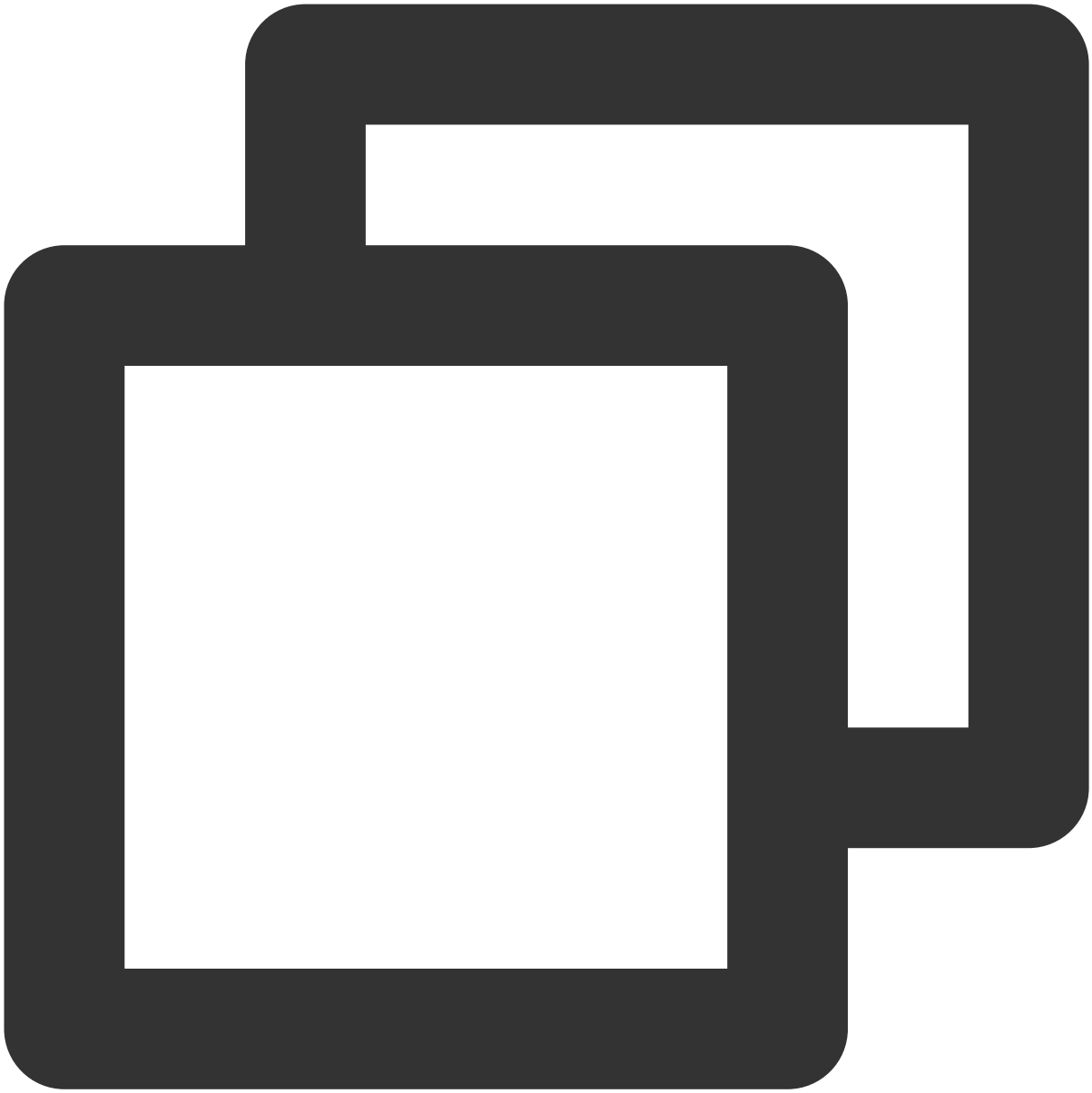
Suggestions on Business Configuration

In addition to the best practices of the DNS service, you can also perform appropriate optimization configuration on the business side to improve the DNS user experience.

©2013-2022 Tencent Cloud. All rights reserved.

1. By default, a domain name in a Kubernetes cluster generally can be resolved after multiple resolution requests. By viewing `/etc/resolv.conf` in a Pod, you will see that the default value of `ndots` is `5`. For example, when the `kubernetes.default.svc.cluster.local` Service in the `debug` namespace is queried:
The domain name has four dots (`.`), so the system tries adding the first `search` to use `kubernetes.default.svc.cluster.local.debug.svc.cluster.local` for query, but cannot find the domain name.
The system continues to use `kubernetes.default.svc.cluster.local.svc.cluster.local` for query, but still cannot find the domain name.
The system continues to use `kubernetes.default.svc.cluster.local.cluster.local` for query, but still cannot find the domain name.
The system tries using `kubernetes.default.svc.cluster.local` without adding the extension. The query succeeds, and the responding `ClusterIP` is returned.
2. The above simple Service domain name can be resolved successfully after four resolutions, and there are a large number of useless DNS requests in the cluster. Therefore, you need to set an appropriate `ndots` value based on the access type configured for the business to reduce the number of queries:

©2013-2022 Tencent Cloud. All rights reserved.



```
spec:
  dnsConfig:
    options:
    - name: ndots
      value: "2"
  containers:
  - image: nginx
    imagePullPolicy: IfNotPresent
    name: diagnosis
```

3. In addition, you can optimize the domain name configuration for your business to access Services:

©2013-2022 Tencent Cloud. All rights reserved.

The Pod should access a Service in the current namespace through `<service-name>`.
The Pod should access a Service in another namespace through `<service-name>.<namespace-name>`.
The Pod should access an external domain name through a fully qualified domain name (FQDN) with a dot (`.`) added at the end to reduce useless queries.

Related Content

Configuration description

errors

It outputs an error message.

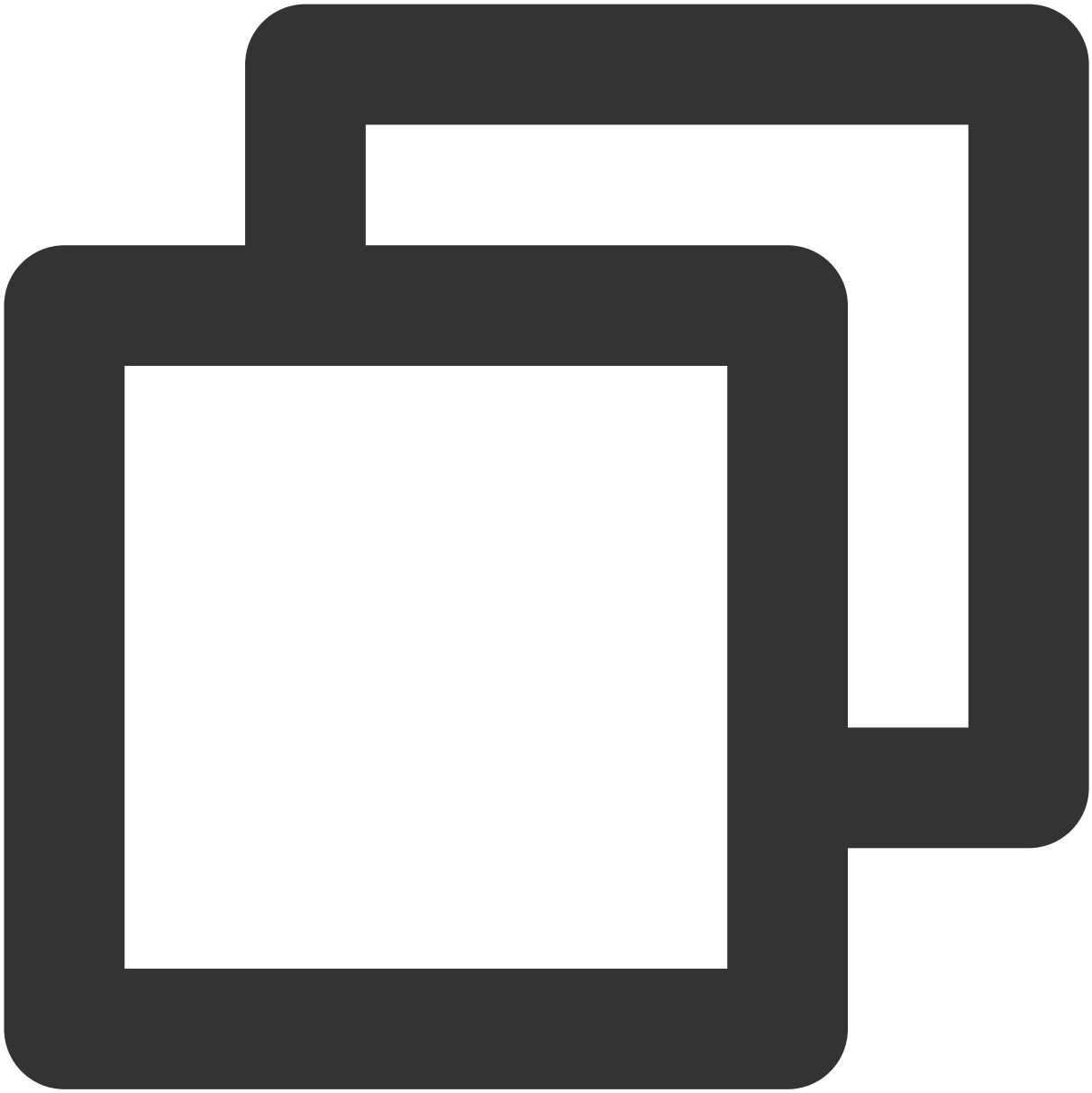
health

It reports the health status and is used for health check configuration such as `livenessProbe`. It listens on port 8080 by default and uses the path `http://localhost:8080/health`.

Note

If there are multiple server blocks, `health` can be configured only once or configured for different ports.

©2013-2022 Tencent Cloud. All rights reserved.



```
com {
  whoami
  health :8080
}

net {
  erratic
  health :8081
}
```

©2013-2022 Tencent Cloud. All rights reserved.

lameduck

It is used to configure the graceful shutdown duration. It is implemented as follows: the hook executes `sleep` when CoreDNS receives a shutdown signal to ensure that the service can continue to run for a certain period of time.

ready

It reports the plugin status and is used for service readiness check configuration such as `readinessProbe`. It listens on port 8181 by default and uses the path `http://localhost:8181/ready`.

kubernetes

It is a Kubernetes plugin that can resolve Services in the cluster.

prometheus

It is a `metrics` data API used to get the monitoring data. Its path is `http://localhost:9153/metrics`.

forward（proxy）

It forwards requests failed to be processed to an upstream DNS server and uses the `/etc/resolv.conf` configuration of the host by default.

According to the configuration of `forward aaa bbb`, the upstream DNS server list `[aaa,bbb]` is maintained internally.

When a request arrives, an upstream DNS server will be selected from the `[aaa,bbb]` list to forward the request according to the preset policy (`random|round_robin|sequential`, where `random` is the default policy). If forwarding fails, another server will be selected for forwarding, and regular health check will be performed on the failed server until it becomes healthy.

If a server fails the health check multiple times (twice by default) in a row, its status will be set to `down`, and it will be skipped in subsequent server selection.

If all servers are down, the system randomly selects a server for forwarding.

Therefore, CoreDNS can intelligently switch between multiple upstream servers. As long as there is an available server in the forwarding list, the request can succeed.

cache

It is the DNS cache.

reload

It hotloads the Corefile. It will reload the new configuration in two minutes after the ConfigMap is modified.

loadbalance

It provides the DNS-based load balancing feature by randomizing the order of records in the answer.

Resource usage of CoreDNS

MEM

CPU

It is subject to the number of Pods and Services in the cluster.

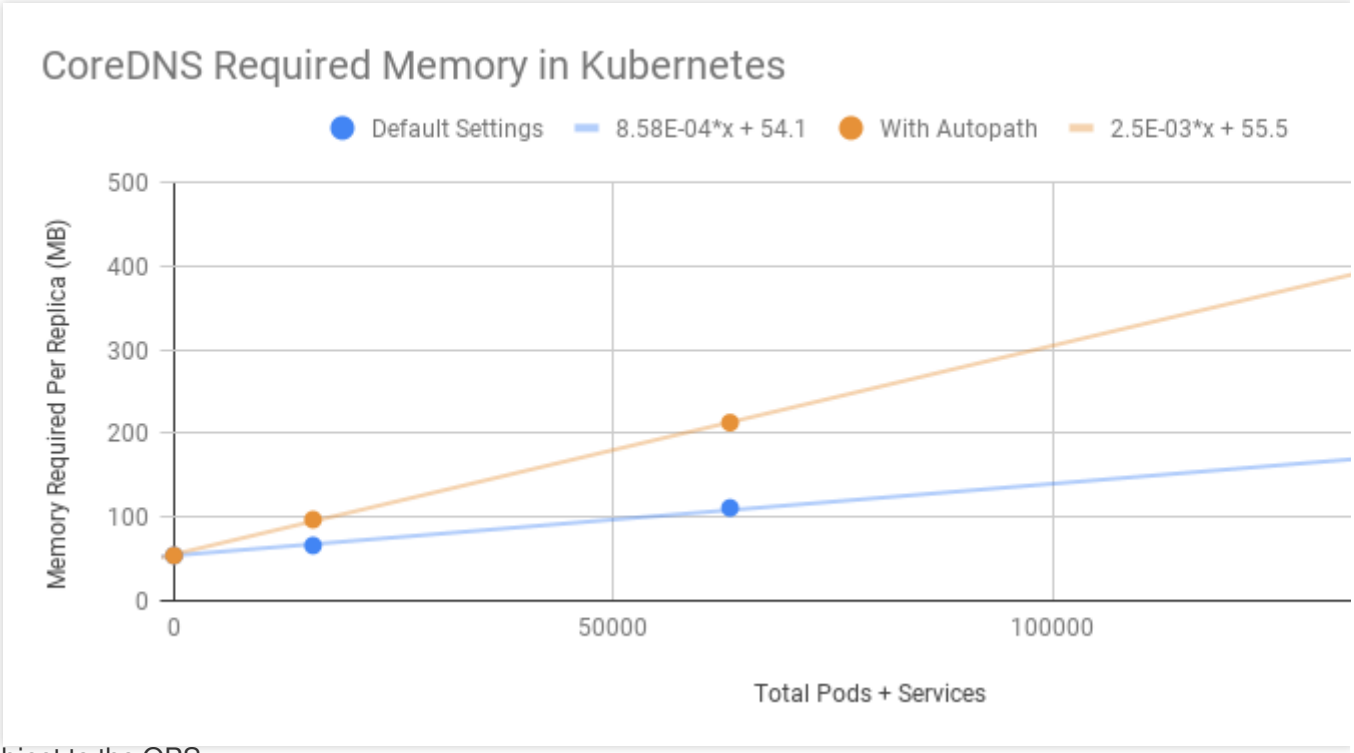
It is affected by the size of enabled cache.

It is affected by the QPS.

©2013-2022 Tencent Cloud. All rights reserved.

Below is the official data of CoreDNS:

MB required (default settings) = (Pods + Services) / 1000 + 54



It is subject to the QPS.

Below is the official data of CoreDNS:

Single-replica CoreDNS with a running node specification of two vCPUs and 7.5 GB MEM:

Query Type	QPS	Avg Latency (ms)	Memory Delta (MB)
external	6733	12.02	+5
internal	33669	2.608	+5

©2013-2022 Tencent Cloud. All rights reserved.

Using NodeLocal DNS Cache in a TKE Cluster

Last updated：2024-08-16 15:25:31

Use Cases

In scenarios where users adopt the Kubernetes standard service discovery mechanism, if the queries per second (QPS) of CoreDNS requests is too high, it may lead to increased DNS query latency and uneven load, adversely affecting business performance and stability.
For this scenario, you can deploy NodeLocal DNS Cache to reduce the pressure of CoreDNS requests, improving the DNS resolution performance and stability within the cluster. This document will detail how to install and use NodeLocal DNS Cache in a Tencent Kubernetes Engine (TKE) cluster.

Use Limits

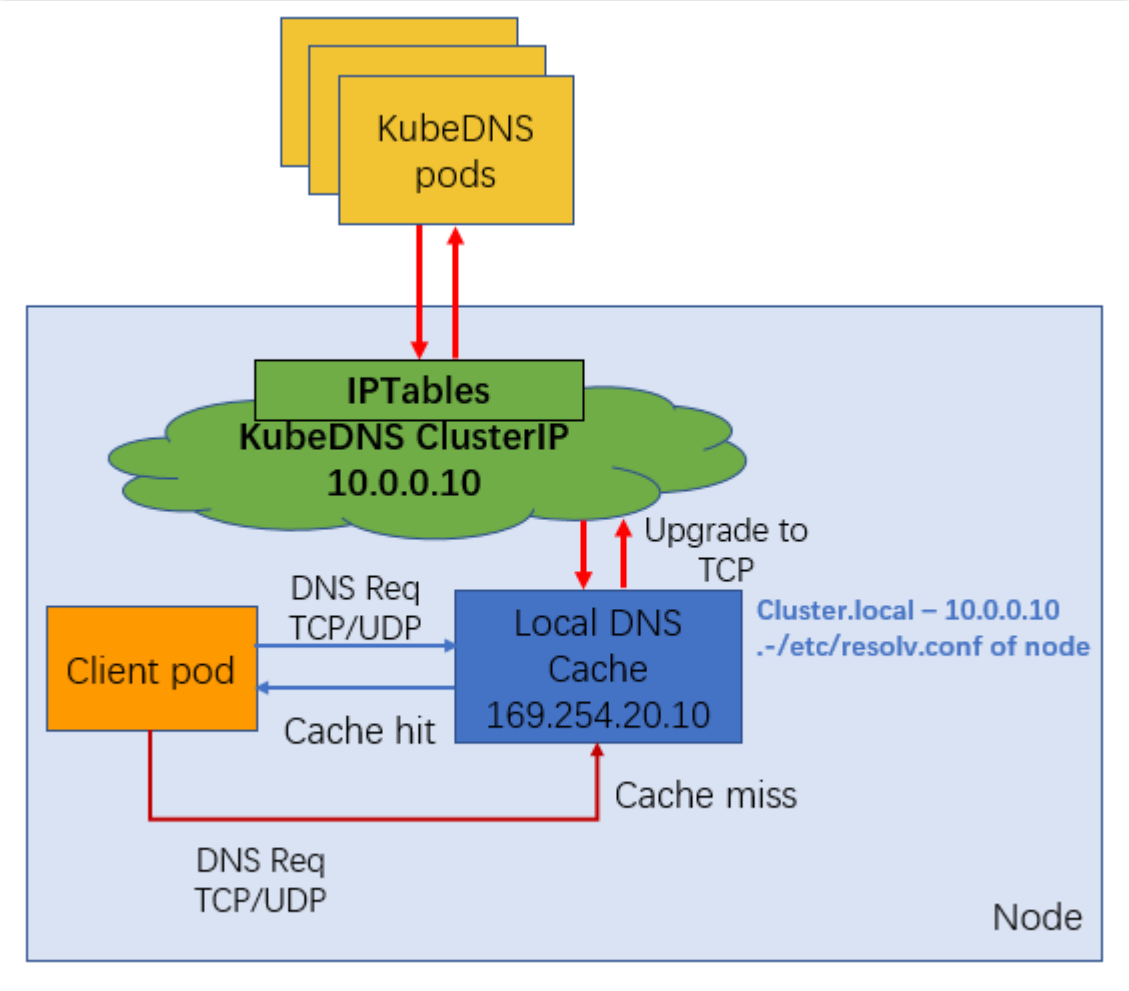
Pods deployed on the super node are not currently supported.
Pods with the network mode of Cilium Overlay and the independent ENI mode are not currently supported.
NodeLocal DNS Cache is currently used only as a CoreDNS cache proxy and does not support configuration of other plugins. If needed, configure CoreDNS directly.

How It Works

Community Solutions

NodeLocal DNS Cache of the community version deploys a hostNetwork pod on each node in the cluster using DaemonSet. The pod is named node-local-dns, which can cache DNS requests for the pod on this node. In case of cache misses, this pod will make a request to the upstream kube-dns service using a TCP connection to fetch the information. The principle diagram is as follows:

©2013-2022 Tencent Cloud. All rights reserved.



The effect can vary with different forwarding modes of kube-proxy.

In iptables mode, after NodeLocal DNS Cache is deployed, both existing pods and incremental pods can seamlessly switch to access the local DNS cache.

In IPVS mode, neither existing pods nor incremental pods can seamlessly switch to access the DNS cache. To use the NodeLocal DNS Cache service in IPVS mode, you can use the following two methods:

Method 1: Modify the kubelet parameter `--cluster-dns` to point to `169.254.20.10`, and then restart the kubelet service. This method carries a risk of business interruption.

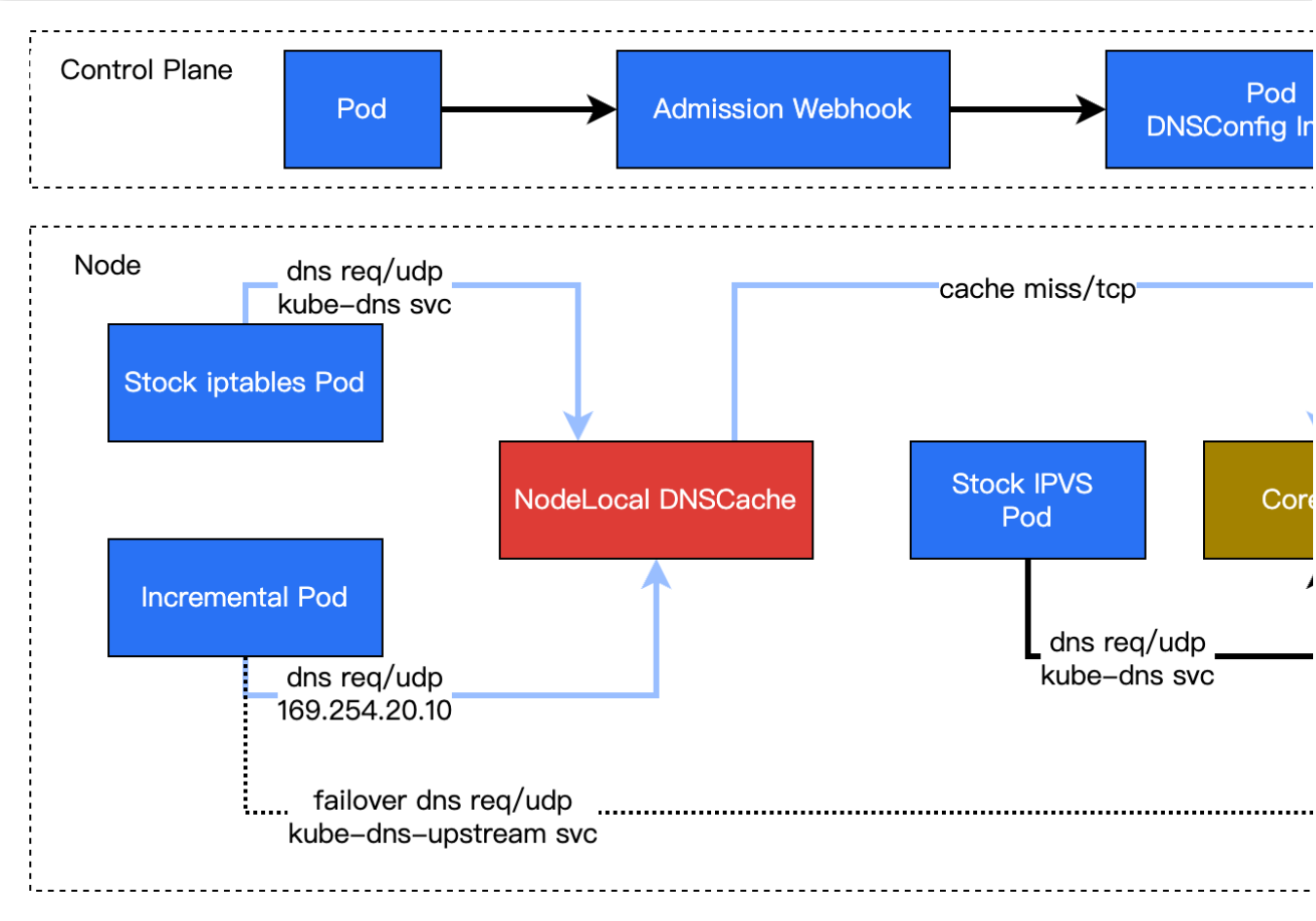
Method 2: Modify DNSConfig of the pod to point to the new address `169.254.20.10`, and use the local DNS cache to handle DNS resolution.

NodeLocal DNS Cache Solution in TKE

The NodeLocal DNS Cache solution in TKE is enhanced for deficiencies of the community version in IPVS mode. For incremental pods, DNSConfig will be automatically configured for the local DNS caching capability. **However, automatic switching for existing pods is still unavailable. Explicit operations (rebuilding pods or manually configuring DNSConfig) of the user are required.**

How it works:

©2013-2022 Tencent Cloud. All rights reserved.

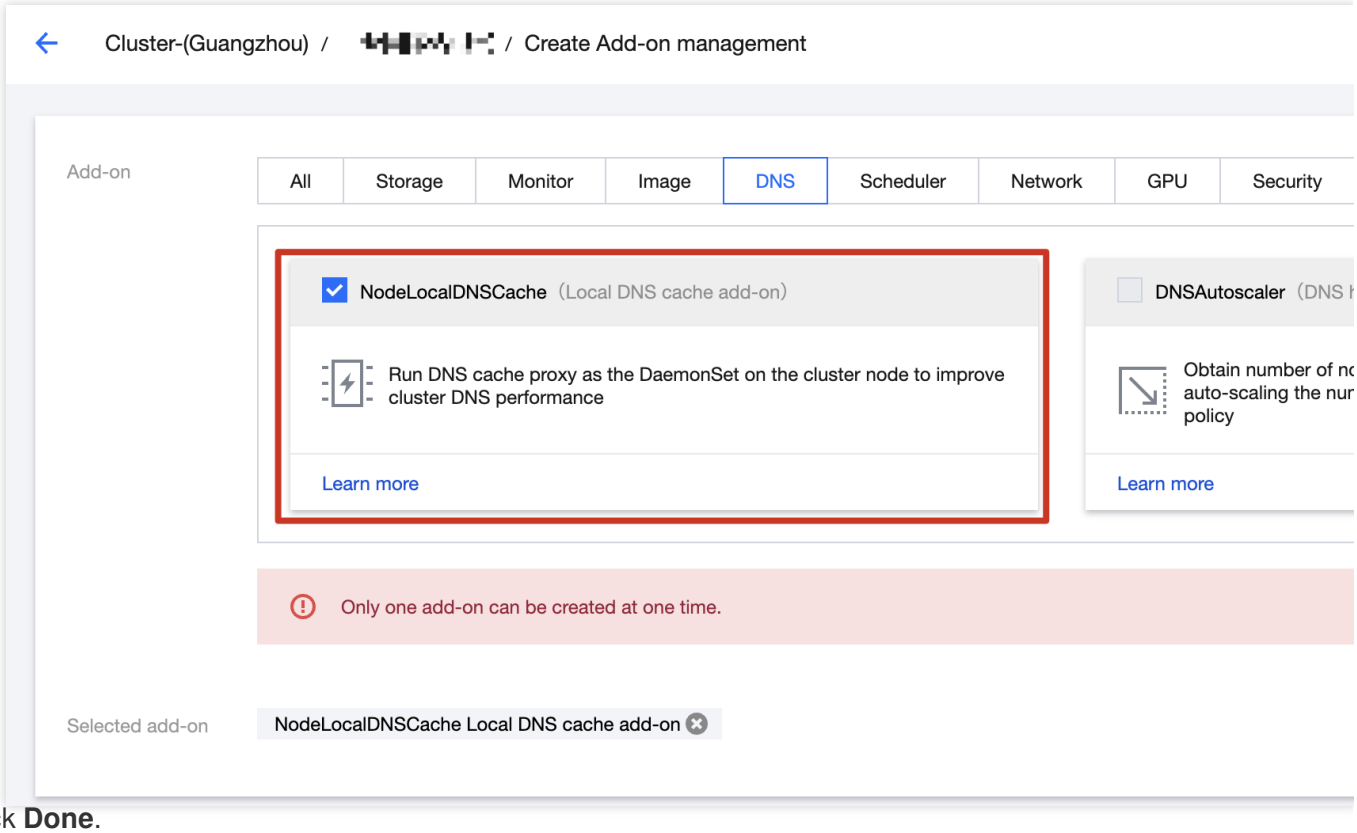


Installing NodeLocal DNS Cache in the Console

You can deploy and install NodeLocal DNS Cache using component management of TKE as follows:

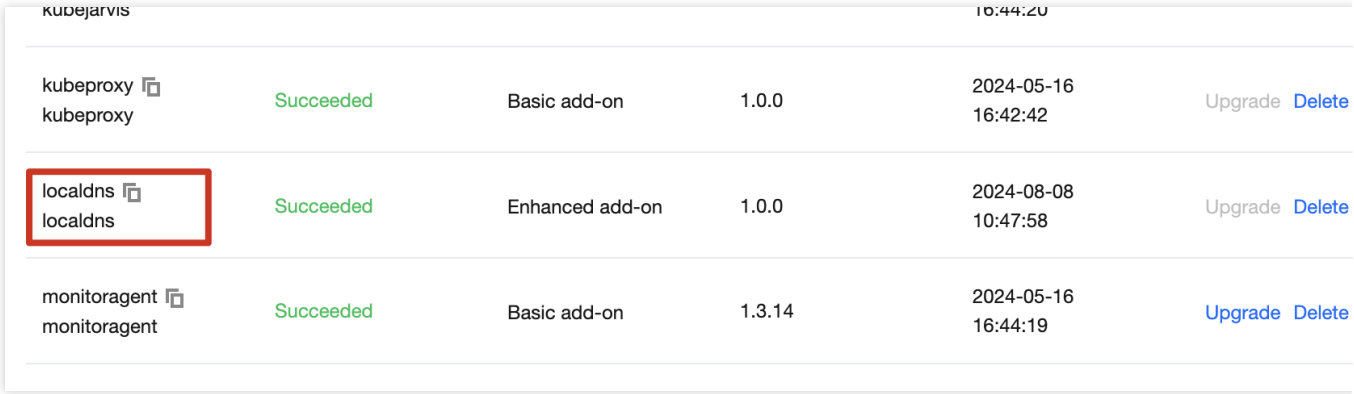
1. Log in to the TKE console, and choose **Cluster** from the left navigation bar.
2. In the Cluster list, click the target cluster ID to access the cluster details page.
3. Select **Add-on Management** from the left-side menu, and click **Create** on the component management page.
4. On the **Create Add-on Management** page, check the **NodeLocalDNSCache** box, as shown in the figure below.

©2013-2022 Tencent Cloud. All rights reserved.



5. Click **Done**.

6. Go back to the **Add-on Management** list page, and check that the localdns component is set to the **Succeeded** state, as shown in the figure below.



Using NodeLocal DNS Cache

In iptables clusters and IPVS clusters, NodeLocal DNS Cache is used in different ways. The specific descriptions are as follows:

iptables Clusters

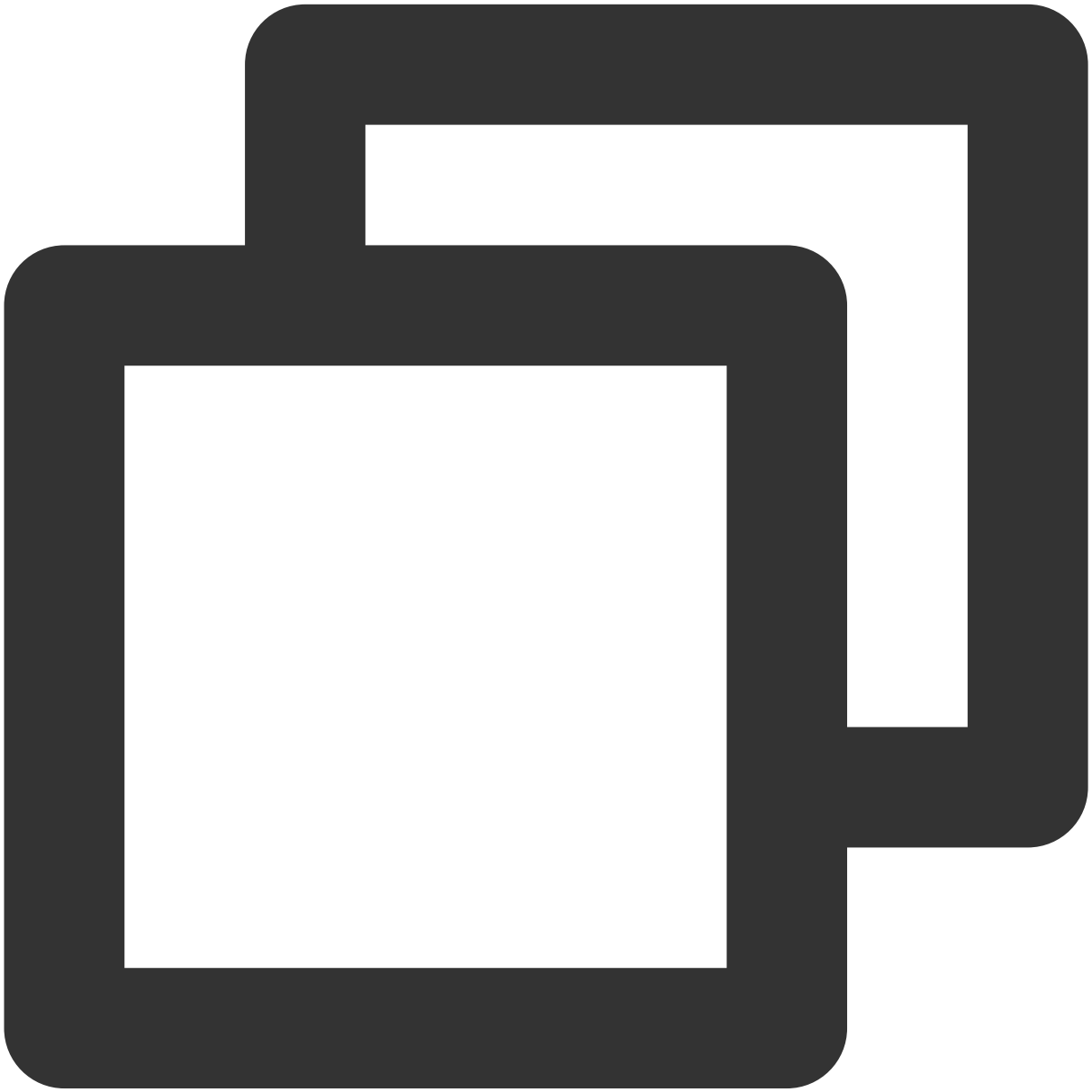
Existing pods: Users do not need to do anything. Existing pods can directly use the local DNS caching capability to resolve DNS requests.

©2013-2022 Tencent Cloud. All rights reserved.

Incremental pods: Users do not need to do anything. Incremental pods can directly use the local DNS caching capability to resolve DNS requests.

IPVS Clusters

For IPVS clusters, TKE will dynamically inject the DNSConfig configuration into newly created pods and set dnsPolicy to None to avoid manual pod YAML configuration. The automatically injected configuration is as follows:



```
dnsConfig:
  nameservers:
  - 169.254.20.10
```

©2013-2022 Tencent Cloud. All rights reserved.

```
    - 10.23.1.234
    options:
    - name: ndots
      value: "3"
    - name: attempts
      value: "2"
    - name: timeout
      value: "1"
    searches:
    - default.svc.cluster.local
    - svc.cluster.local
    - cluster.local
  dnsPolicy: None
```

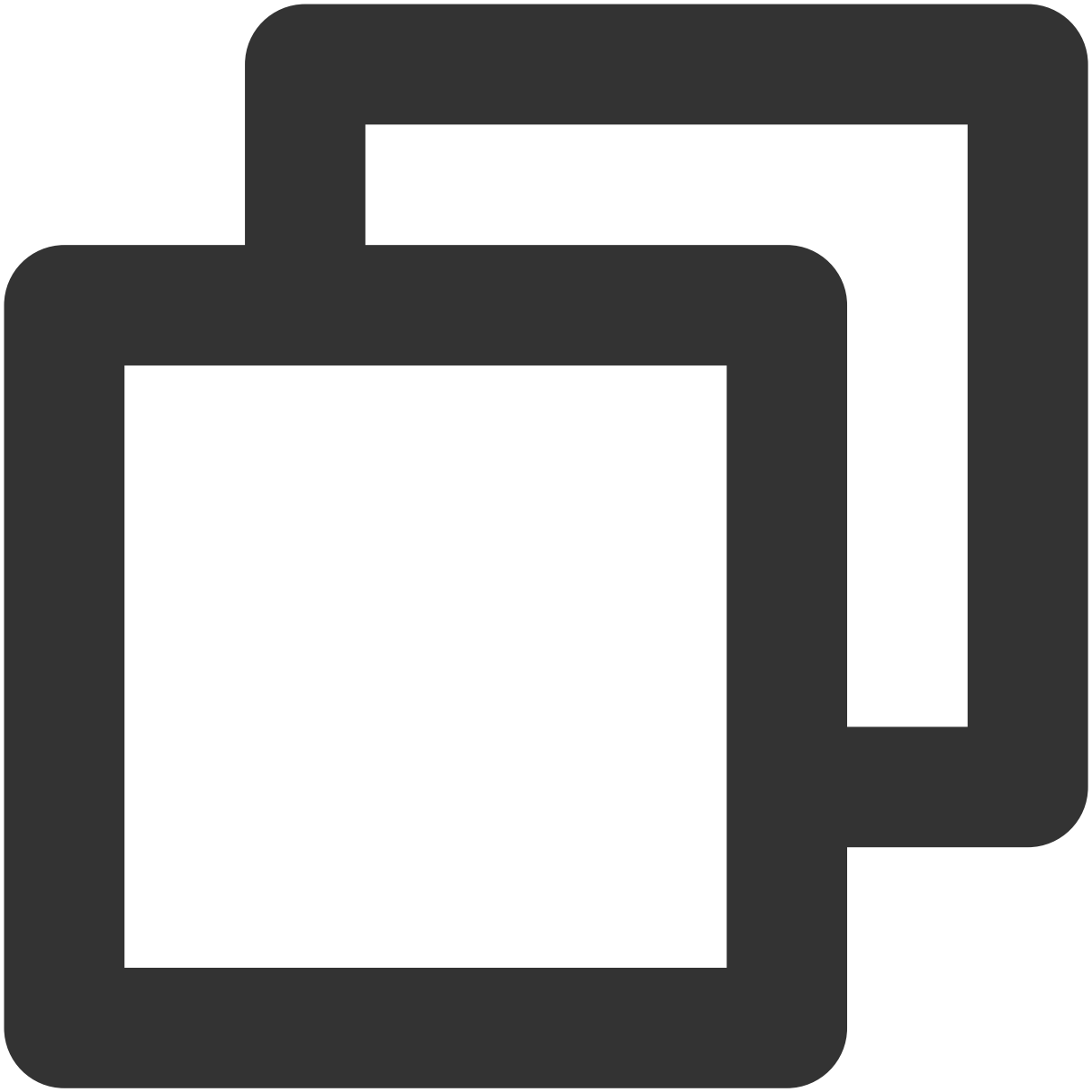
Note:

To automatically inject DNSConfig into the corresponding pod, ensure the following conditions are met:

1. Label the namespace where the pod is located with the tag `localdns-injector=enabled` .

For example, to automatically inject DNSConfig into the newly created pods in the default namespace, perform the following configuration:

©2013-2022 Tencent Cloud. All rights reserved.



```
kubectl label namespace default localdns-injector=enabled
```

2. Ensure the pod is not in the **kube-system** and **kube-public** namespaces. DNSConfig will not be automatically injected into pods in these two namespaces.

3. Ensure the pod label does not contain ** `localdns-injector=disabled` **. Pods with this label will not be injected with DNSConfig.

4. If the network of the newly created pod is configured to non-hostNetwork, configure DNSPolicy to ClusterFirst. If the pod network is hostNetwork, configure DNSPolicy to ClusterFirstWithHostNet.

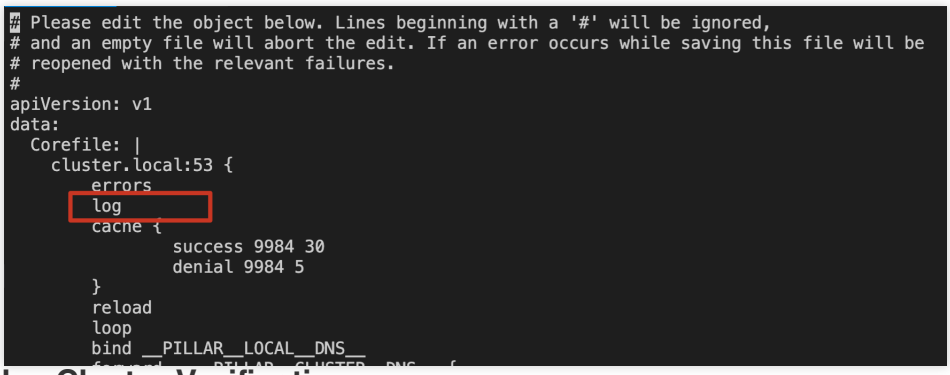
5. The GR network mode is not currently supported.

©2013-2022 Tencent Cloud. All rights reserved.

Existing pods: Existing pods cannot be seamlessly switched for now. To enable the local DNS caching proxy capability for existing pods, users need to rebuild pods. After rebuilding, pods will be automatically injected with DNSConfig to use the local DNS caching capability for DNS request resolution.
Incremental pods: Once the above precautions are met, incremental pods will be automatically injected with DNSConfig to access the node `169.254.20.10:53` and use the local DNS caching capability for DNS request resolution.

Verifying NodeLocal DNS Cache

After NodeLocal DNS Cache is successfully enabled, you can verify on the node whether pod access to CoreDNS services has been resolved using the local DNS cache. Below are the methods to verify the effect of enabling NodeLocal DNS Cache in both the iptables cluster and the IPVS cluster.
Note:
If you want to verify whether NodeLocal DNS Cache on the node is the proxy for DNS requests of this node according to logs, you need to modify the ConfigMap configuration of node-local-dns in the kube-system namespace and add the log capability to the corresponding Corefile configuration, as shown in the figure below.



iptables Cluster Verification

In the iptables cluster, it is necessary to verify whether existing pods and incremental pods can automatically have their DNS requests proxied by local NodeLocal DNS Cache.

Existing Pods

1. Log in to an existing pod.
2. Run the nslookup command to resolve the SVC file of kube-dns, as shown in the figure below.

©2013-2022 Tencent Cloud. All rights reserved.

```
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec
nstead.
/ # nslookup kube-dns.kube-system.svc
nslookup: can't resolve '(null)': Name does not resolve

Name:      kube-dns.kube-system.svc
Address 1: 10.23.1.110 kube-dns.kube-system.svc.cluster.local
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
3: eth0@if8: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether 3a:4c:b7:ad:1c:5b brd ff:ff:ff:ff:ff:ff
    inet 10.99.21.3/32 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 2402:4e00:1207:5916::9b8f:90ce:846a/128 scope global
       valid_lft forever preferred_lft forever
    inet6 fe80::384c:b7ff:fead:1c5b/64 scope link
       valid_lft forever preferred_lft forever
/ #
```

3. Check the logs of the node-cache pod on this node, as shown in the figure below.

```
[INFO] 10.99.21.3:50709 - 62853 "A IN kube-dns.kube-system.svc.default.svc.cluster.local. udp 68 false
a,rd 161 0.002424868s
[INFO] 10.99.21.3:50709 - 63675 "AAAA IN kube-dns.kube-system.svc.default.svc.cluster.local. udp 68 fal
r,aa,rd 161 0.0037218s
[INFO] 10.99.21.3:48989 - 38909 "A IN kube-dns.kube-system.svc.svc.cluster.local. udp 60 false 512" NXD
 0.00215277s
[INFO] 10.99.21.3:48989 - 39939 "AAAA IN kube-dns.kube-system.svc.svc.cluster.local. udp 60 false 512"
153 0.003375326s
[INFO] 10.99.21.3:49403 - 2811 "A IN kube-dns.kube-system.svc.cluster.local. udp 56 false 512" NOERROR
135301s
[INFO] 10.99.21.3:49403 - 3664 "AAAA IN kube-dns.kube-system.svc.cluster.local. udp 56 false 512" NOERR
```

You can confirm that the DNS resolution requests from existing pods to kube-dns are proxied by the NodeLocal DNS Cache service on this node.

Incremental Pods

1. Log in to a newly created pod.
2. Run the nslookup command to resolve the SVC file of kube-dns, as shown in the figure below.

©2013-2022 Tencent Cloud. All rights reserved.

```
/ # nslookup kube-dns.kube-system.svc
nslookup: can't resolve '(null)': Name does not resolve

Name:      kube-dns.kube-system.svc
Address 1: 10.23.1.110 kube-dns.kube-system.svc.cluster.local
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
3: eth0@if7: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether fa:e9:bd:9e:f3:df brd ff:ff:ff:ff:ff:ff
    inet 10.99.10.29/32 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 2402:4e00:1207:5958::9b67:446c:2ab9/128 scope global
       valid_lft forever preferred_lft forever
    inet6 fe80::f8e9:bdff:fe9e:f3df/64 scope link
       valid_lft forever preferred_lft forever
/ #
```

3. Check the logs of the node-cache pod on this node, as shown in the figure below.

```
[INFO] 10.99.10.29:35234 - 52850 "AAAA IN kube-dns.kube-system.svc.svc.cluster.local. udp 60 false 512"
 153 0.000454866s
[INFO] 10.99.10.29:53379 - 11046 "AAAA IN kube-dns.kube-system.svc.cluster.local. udp 56 false 512" NOE
0.000248217s
[INFO] 10.99.10.29:53379 - 10135 "A IN kube-dns.kube-system.svc.cluster.local. udp 56 false 512" NOERRO
0029001 9s
[INFO] 10.99.10.29:35234 - 52022 "A IN kube-dns.kube-system.svc.svc.cluster.local. udp 60 false 512" NX
3 0.009849625s
```

You can confirm that the DNS resolution requests from incremental pods to kube-dns are proxied by the NodeLocal DNS Cache service on this node.

IPVS Cluster Verification

In the IPVS cluster, existing pods cannot automatically switch to use the local DNS cache temporarily. It is necessary to verify whether the incremental pods can automatically proxy their DNS requests using the local NodeLocal DNS Cache. The steps are as follows:

1. Add the label `localdns-injector=enabled` to the required namespace.

2. In the required namespace, create an incremental pod, and confirm the pod is injected with DNSConfig, as shown in the figure below.

©2013-2022 Tencent Cloud. All rights reserved.

```
  dnsConfig:
    nameservers:
    - 169.254.20.10
    - 10.0.3.209
    options:
    - name: ndots
      value: "3"
    - name: attempts
      value: "2"
    - name: timeout
      value: "1"
    searches:
    - dodia.svc.cluster.local
    - svc.cluster.local
    - cluster.local
  dnsPolicy: None
```

3. Log in to a newly created pod.

4. Run the nslookup command to resolve the SVC file of kube-dns, as shown in the figure below.

```
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [
nstead.
/ # nslookup kube-dns.kube-system.svc
nslookup: can't resolve '(null)': Name does not resolve

Name:      kube-dns.kube-system.svc
Address 1: 10.0.3.106 kube-dns.kube-system.svc.cluster.local
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
3: eth0@if17: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether a2:ea:db:b2:ed:08 brd ff:ff:ff:ff:ff:ff
    inet 10.99.10.2/32 scope global eth0
       valid_lft forever preferred_lft forever
/ #
```

5. Check the logs of the node-cache pod on this node, as shown in the figure below.

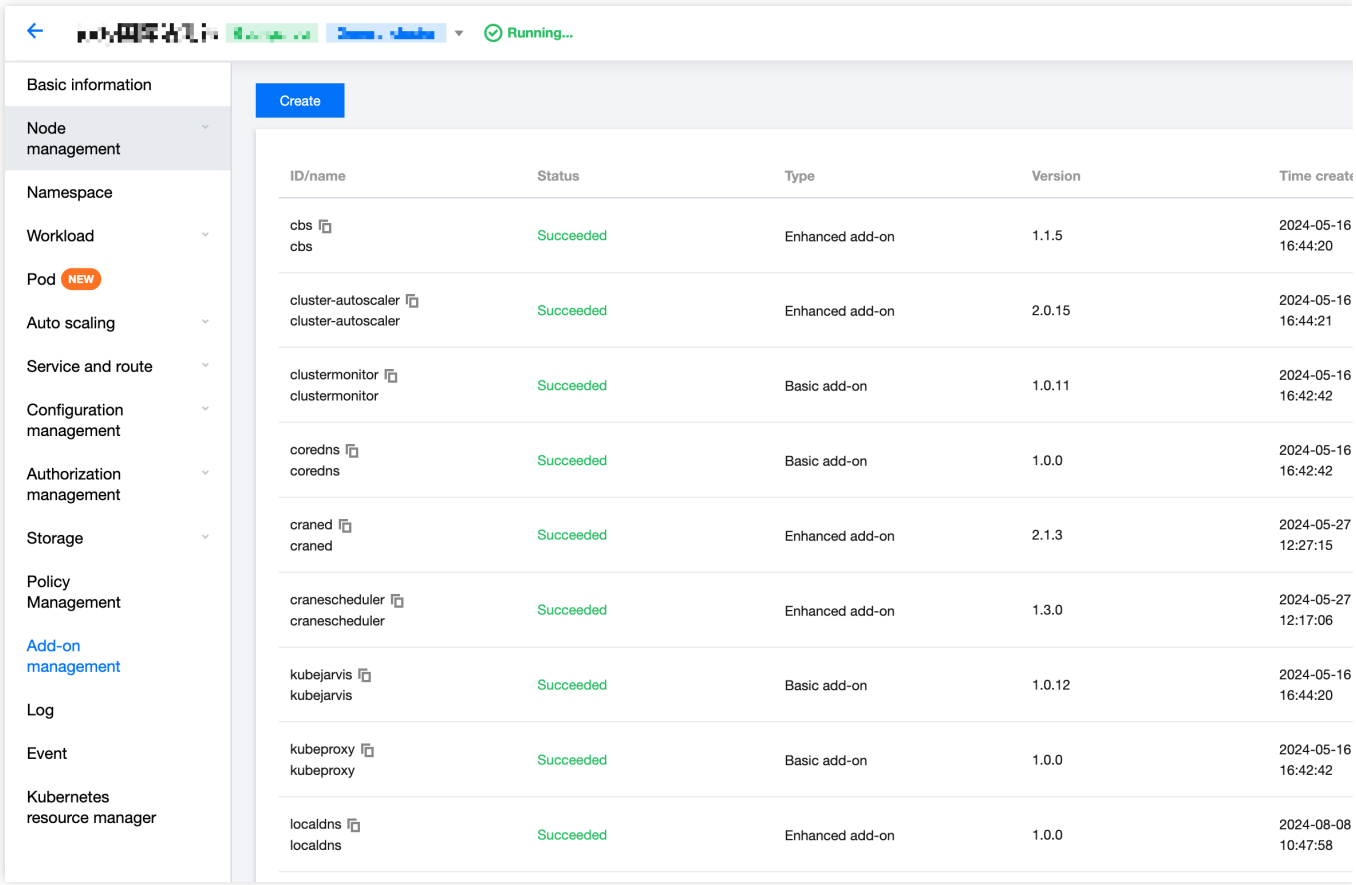
```
[INFO] 10.99.10.2:50375 - 45281 "AAAA IN kube-dns.kube-system.svc.dodia.svc.cluster.local. udp 66 false
aa,rd 159 0.000953497s
[INFO] 10.99.10.2:50375 - 44477 "A IN kube-dns.kube-system.svc.dodia.svc.cluster.local. udp 66 false 512
rd 159 0.001436562s
[INFO] 10.99.10.2:50240 - 24282 "A IN kube-dns.kube-system.svc.svc.cluster.local. udp 60 false 512" NXDO
 0.000706419s
[INFO] 10.99.10.2:50240 - 25235 "AAAA IN kube-dns.kube-system.svc.svc.cluster.local. udp 60 false 512" N
153 0.000955811s
[INFO] 10.99.10.2:35072 - 55625 "A IN kube-dns.kube-system.svc.cluster.local. udp 56 false 512" NOERROR
0513704s
[INFO] 10.99.10.2:35072 - 56505 "AAAA IN kube-dns.kube-system.svc.cluster.local. udp 56 false 512" NOERR
.00060428s
```

You can confirm that the DNS resolution requests from incremental pods in the IPVS cluster to kube-dns are proxied by the NodeLocal DNS Cache service on this node.

Uninstalling NodeLocal DNS Cache

©2013-2022 Tencent Cloud. All rights reserved.

1. Log in to the TKE console, and choose **Cluster** from the left navigation bar.
2. In the Cluster list, click the target cluster ID to access the cluster details page.
3. Select **Add-on Management** from the left-side menu. On the Add-on Management page, click **Delete** on the right side of the row where the component to be deleted is located, as shown in the figure below.



FAQs

prefer_udp Related Configurations

Problem Description

In the TKE cluster, CoreDNS uses the default DNS service (183.60.83.19/183.60.82.98) of Tencent Cloud as the upstream DNS. The default DNS service of Tencent Cloud supports DNS requests for private domain resolution in the VPC. Currently, only the UDP protocol but not the TCP protocol is supported. However, NodeLocal DNS connects to CoreDNS using TCP by default. If CoreDNS is not configured with prefer_udp, it will default to accessing the upstream Tencent Cloud DNS service using TCP, which can lead to occasional domain name resolution failures.

Solution

1. **New TKE clusters**: CoreDNS has been configured with prefer_udp by default, so users do not need to take any action.

©2013-2022 Tencent Cloud. All rights reserved.

2. **Existing clusters**: If users have deployed the NodeLocal DNS Cache component, it is recommended to configure the relevant Corefile of the CoreDNS service to add prefer_udp and reload the configuration, as shown in the figure below.

```
apiVersion: v1
data:
  Corefile: |2-
        .:53 {
            template ANY HINFO . {
                rcode NXDOMAIN
            }
            log
            errors
            health {
                lameduck 30s
            }
            ready
            kubernetes cluster.local. in-addr.arpa ip6.arpa {
                pods insecure
                fallthrough in-addr.arpa ip6.arpa
            }
            prometheus :9153
            forward . /etc/resolv.conf {
                prefer_udp
            }
            cache 30
            reload
            loadbalance
        }
kind: ConfigMap
```

3. **Clusters without NodeLocal DNS Cache installed**: Before installing NodeLocal DNS Cache components, check whether the prefer_udp configuration is added to the CoreDNS Corefile. Users need to manually configure it before continuing with the installation of the NodeLocal DNS Cache components.

kube-proxy Version Compatibility Issues

Problem Description

In TKE clusters, earlier versions of kube-proxy have the multi-backend issue for iptables (legacy/nftable). The triggering conditions are as follows:

1. The proxy mode of kube-proxy in the cluster is iptables.

2. In Kubernetes clusters of different versions, the versions of kube-proxy in the clusters are earlier than the version numbers below.

TKE Cluster Version	Fix Policy
1.24	Upgrade kube-proxy to v1.24.4-tke.5 or later
1.22	Upgrade kube-proxy to v1.22.5-tke.11 or later
1.20	Upgrade kube-proxy to v1.20.6-tke.31 or later

©2013-2022 Tencent Cloud. All rights reserved.

1.18	Upgrade kube-proxy to v1.18.4-tke.35 or later
1.16	Upgrade kube-proxy to v1.16.3-tke.34 or later
1.14	Upgrade kube-proxy to v1.14.3-tke.28 or later
1.12	Upgrade kube-proxy to v1.12.4-tke.32 or later
1.10	Upgrade kube-proxy to v1.10.5-tke.20 or later

At this time, if the customer deploys the NodeLocal DNS Cache component, a multi-backend issue may be triggered, causing the service access failure in the cluster.

Solution

1. If the user's existing cluster configuration meets the above triggering conditions, it is recommended to upgrade the kube-proxy version to the latest version.
2. When the NodeLocal DNS Cache component needs to be installed in the current TKE cluster, the kube-proxy version will be checked. If the version does not meet the requirements, the user will be prohibited from installing the component. In this case, upgrade the kube-proxy version to the latest version.
For the latest kube-proxy version, refer to TKE Kubernetes Revision Version History.

©2013-2022 Tencent Cloud. All rights reserved.

Implementing Custom Domain Name Resolution in TKE

Last updated：2023-03-27 11:08:16

Overview

When using a TKE or TKE Serverless cluster, you may need to resolve the custom internal domain names in the following scenarios:
You build an external centralized storage service, and want to send the monitoring or log collection data in the cluster to the external storage service through a fixed internal domain name.
During the containerization of traditional services, the code of some services is configured to call other internal services through a fixed domain name, and the configuration cannot be modified, that is, the Service name of Kubernetes cannot be used for calling.

Solutions

This document describes the following three solutions for using custom domain name resolution in a cluster:

Solution	Benefits
Solution 1: Using the CoreDNS hosts plugin to configure arbitrary domain name resolution	This solution is simple and intuitive. You can add arbitrary resolution records.
Solution 2: Using the CoreDNS rewrite plugin to map a domain name to a service in the cluster	You do not need to know the IP address of a resolution record in advance, but the IP address mapped by the resolution record must be deployed in the cluster.
Solution 3: Using the CoreDNS forward plugin to set the external DNS as the upstream DNS	You can manage a large number of resolution records. As all records are managed in the external DNS, you do not need to modify the CoreDNS configuration when adding or deleting records.

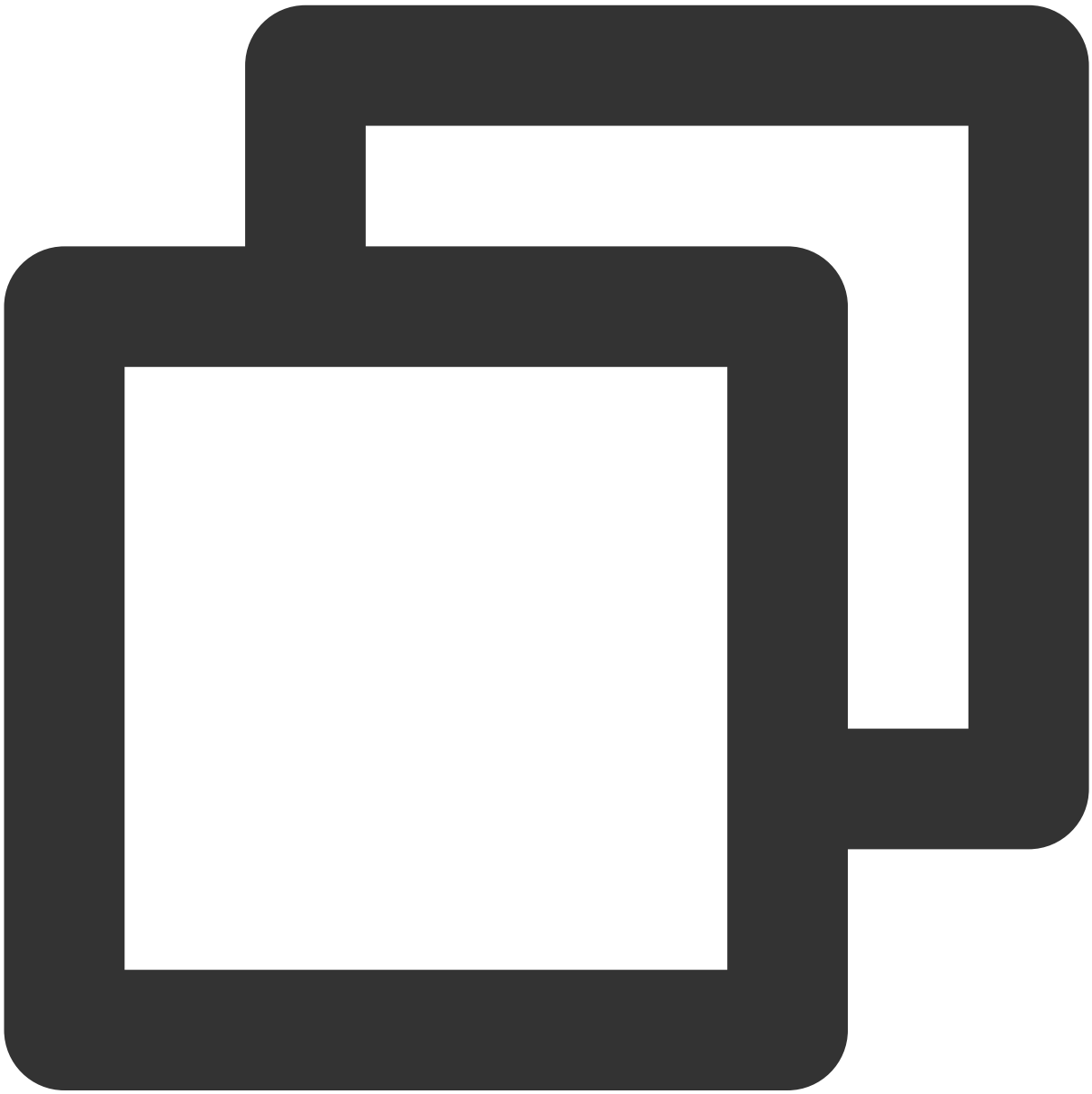
Note
In the first two solutions, you need to modify the CoreDNS configuration file each time you add a resolution record. The modification takes effect without restart. Select a solution based on your actual needs.

©2013-2022 Tencent Cloud. All rights reserved.

Examples

Solution 1: Using the CoreDNS hosts plugin to configure arbitrary domain name resolution

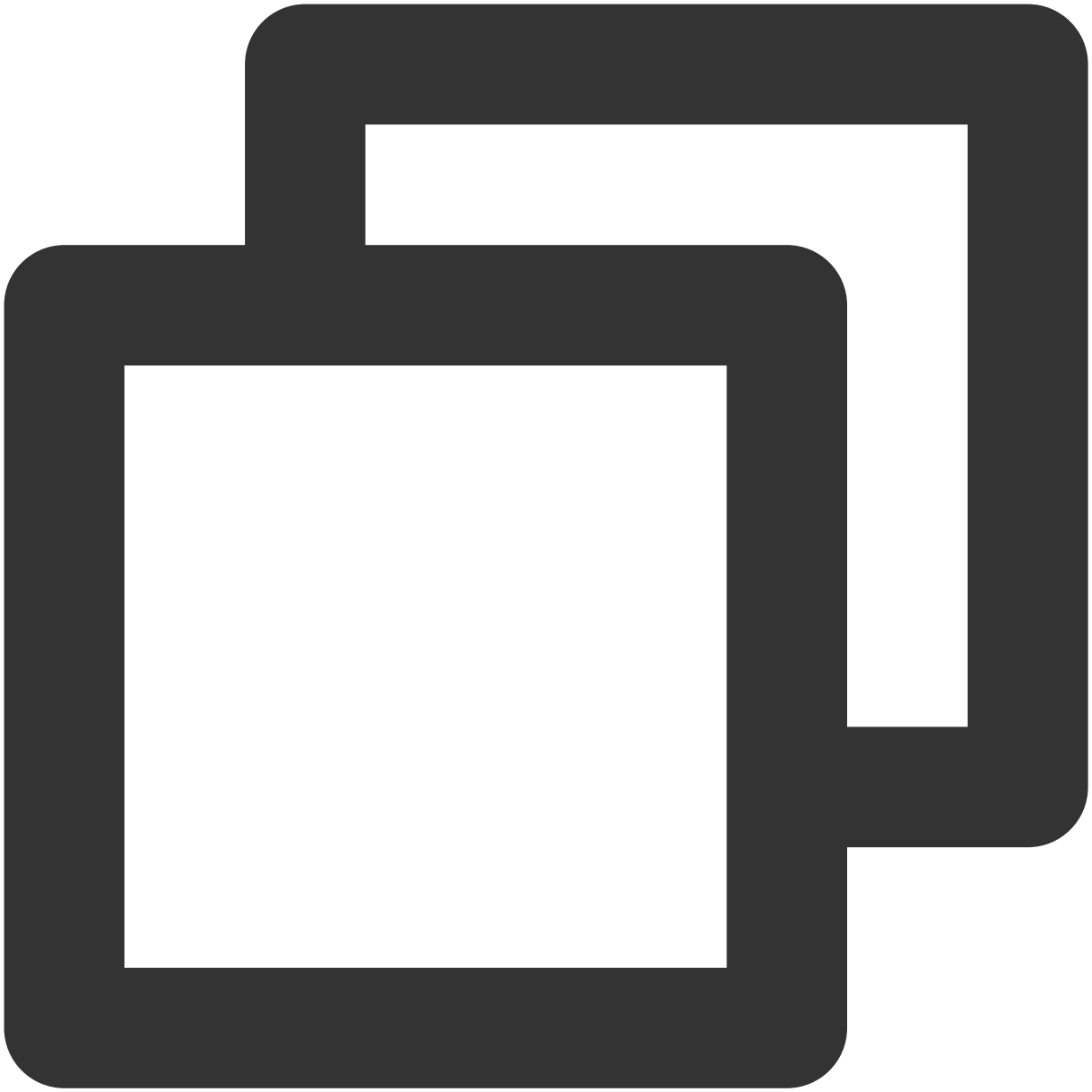
1. Run the following command to modify the `configmap` of `CoreDNS`, as shown below:



```
kubectl edit configmap coredns -n kube-system
```

2. Modify the `hosts` configuration by adding the relevant domain names, as shown below:

©2013-2022 Tencent Cloud. All rights reserved.



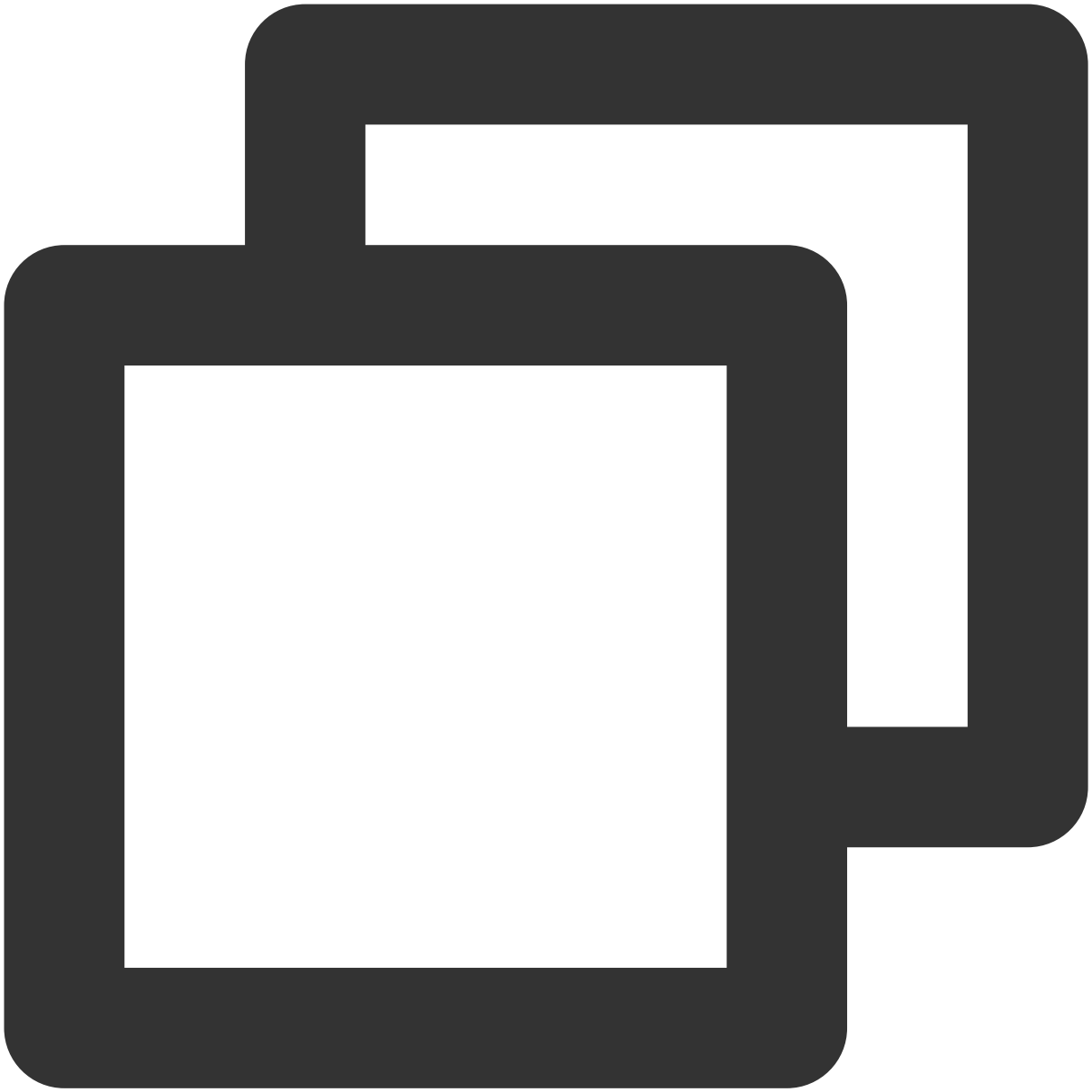
```
hosts {
      192.168.1.6     harbor.example.com
      192.168.1.8     es.example.com
      fallthrough
}
```

Description

Map `harbor.example.com` to 192.168.1.6 and `es.example.com` to 192.168.1.8.

The complete configurations are as follows:

©2013-2022 Tencent Cloud. All rights reserved.



```
apiVersion: v1
data:
      Corefile: |2-
        .:53 {
            errors
            health
            kubernetes cluster.local. in-addr.arpa ip6.arpa {
                pods insecure
                upstream
                fallthrough in-addr.arpa ip6.arpa
            }
```

©2013-2022 Tencent Cloud. All rights reserved.

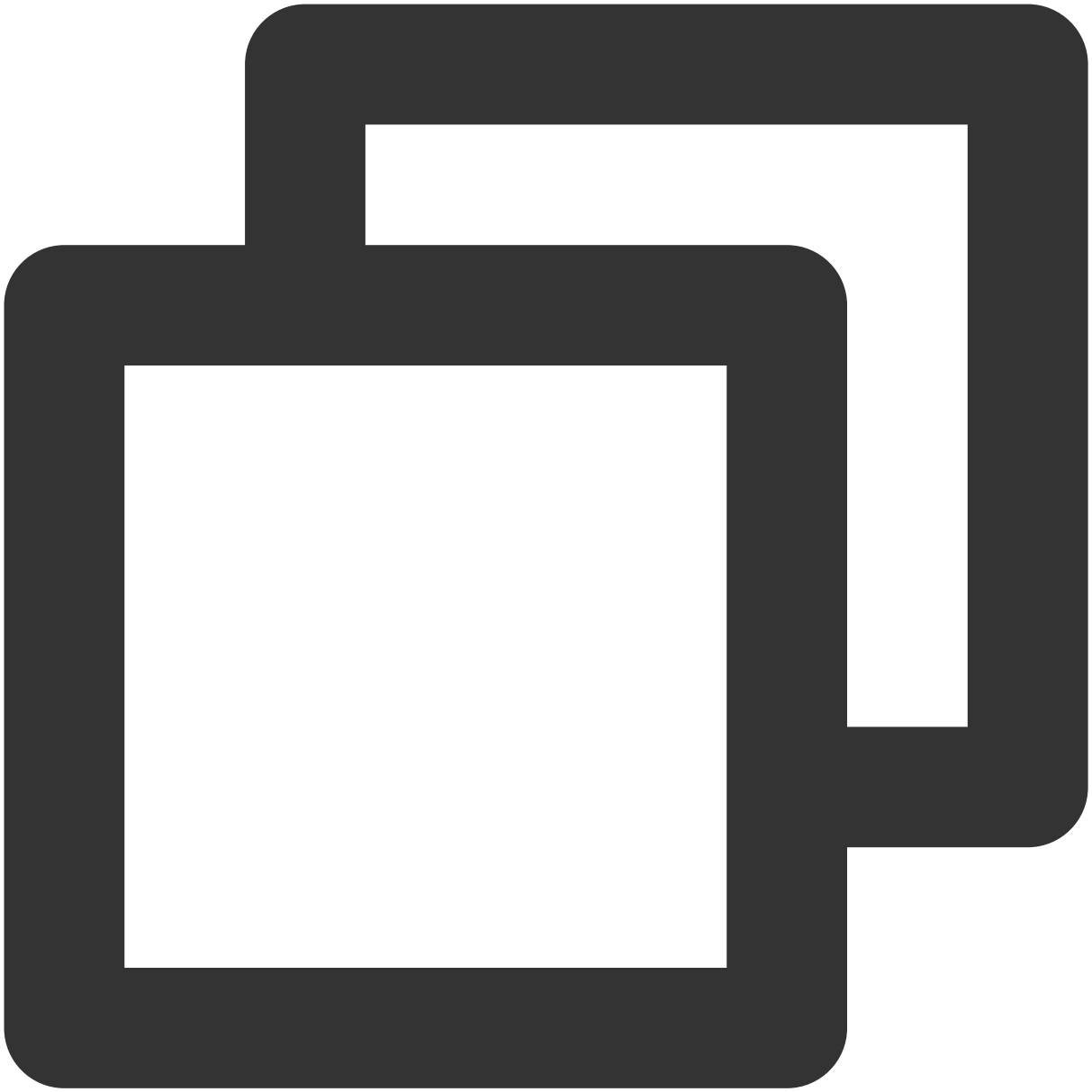
```
            hosts {
                192.168.1.6     harbor.example.com
                192.168.1.8     es.example.com
                fallthrough
            }
            prometheus :9153
            forward . /etc/resolv.conf
            cache 30
            reload
            loadbalance
        }
kind: ConfigMap
metadata:
      labels:
        addonmanager.kubernetes.io/mode: EnsureExists
      name: coredns
      namespace: kube-system
```

Solution 2: Using the CoreDNS rewrite plugin to map a domain name to a service in the cluster

If you need to deploy a service with a custom domain name in a cluster, you can use the Rewrite plugin of CoreDNS to resolve the specified domain name to the ClusterIP of a Service.

1. Run the following command to modify the `configmap` of `CoreDNS`, as shown below:

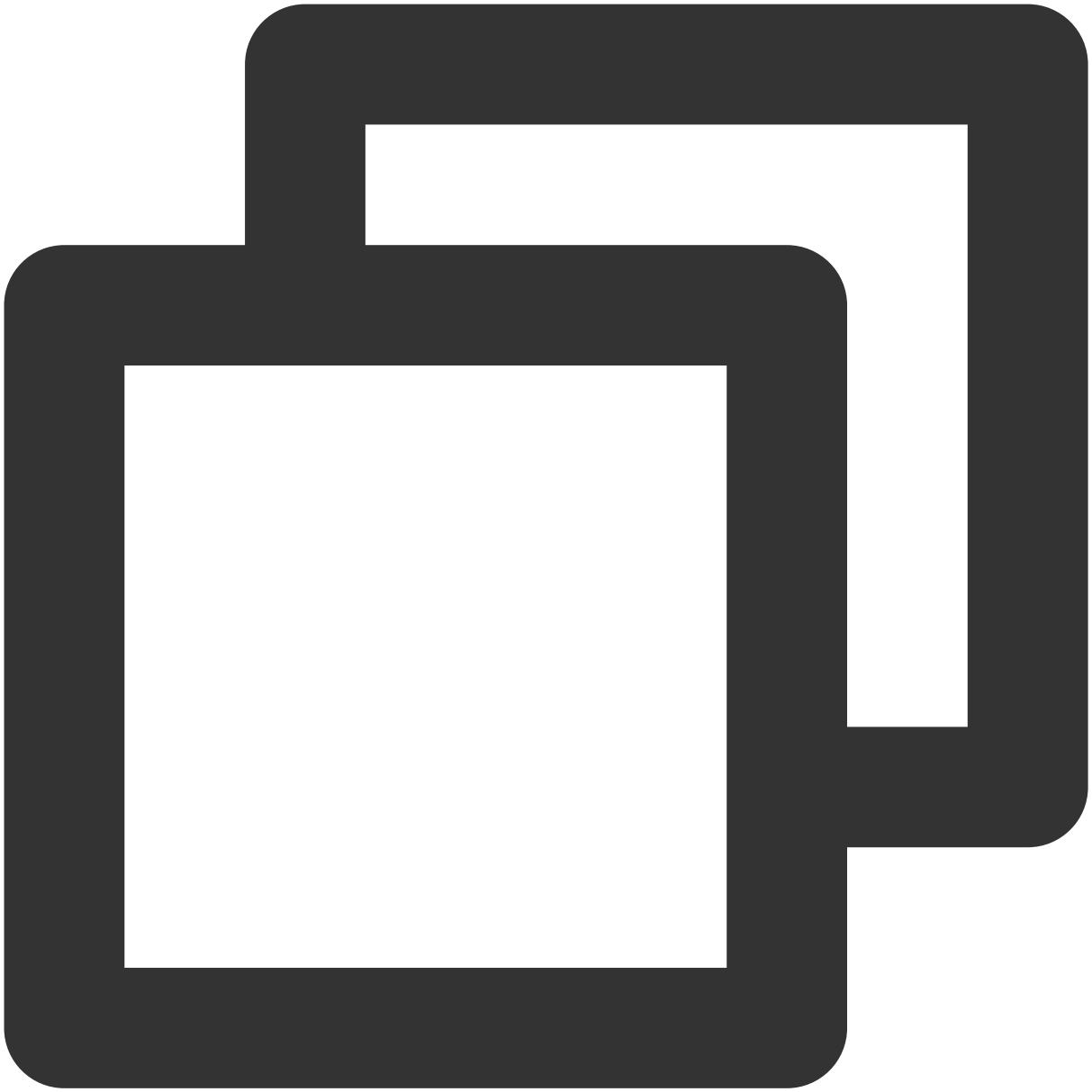
©2013-2022 Tencent Cloud. All rights reserved.



```
kubectl edit configmap coredns -n kube-system
```

2. Run the following command to add the rewrite configuration, as shown below:

©2013-2022 Tencent Cloud. All rights reserved.



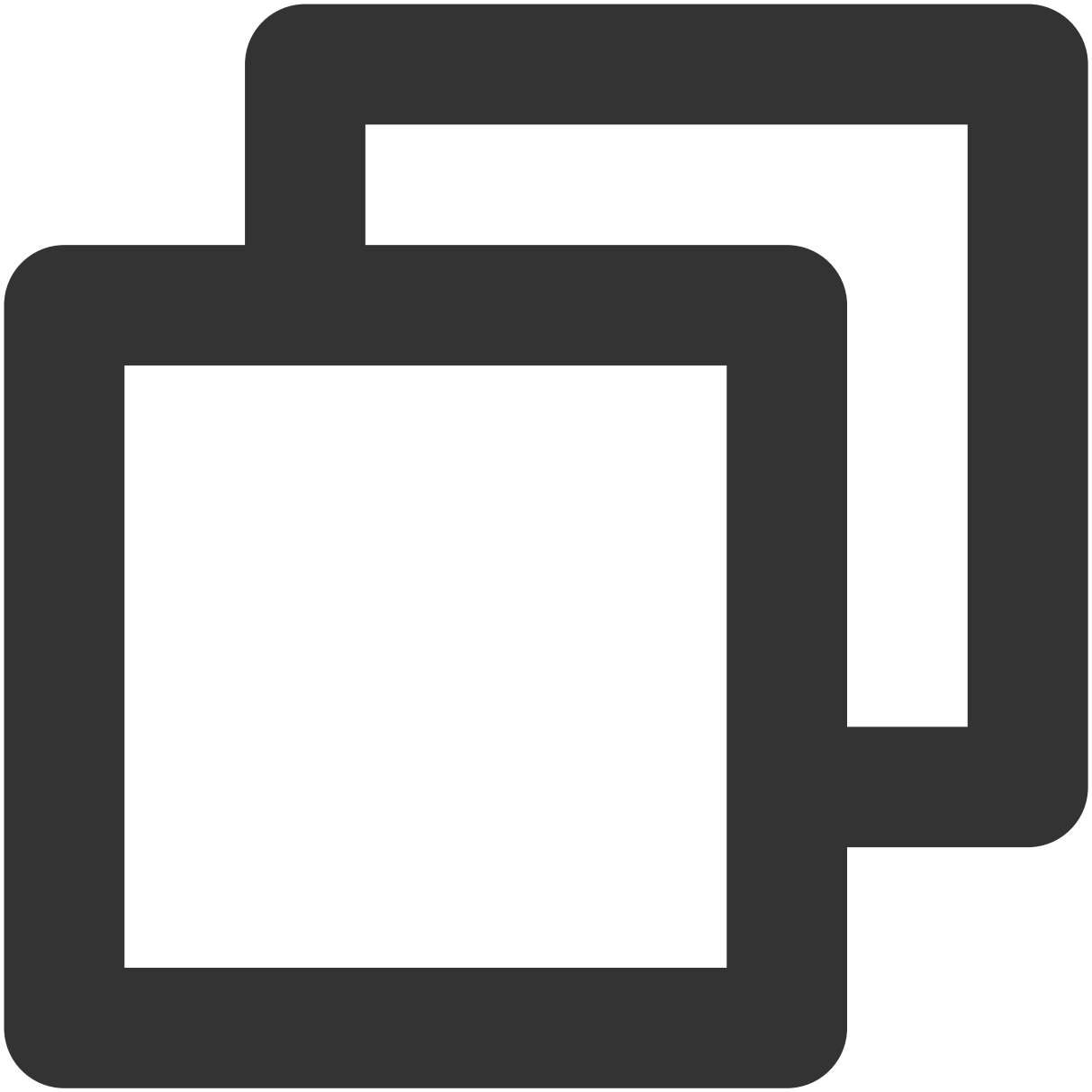
```
rewrite name es.example.com es.logging.svc.cluster.local
```

Description

Map the `es.example.com` domain name to the `es` service deployed in the `logging` namespace. Separate multiple domain names with carriage returns.

The complete configurations are as follows:

©2013-2022 Tencent Cloud. All rights reserved.



```
apiVersion: v1
data:
      Corefile: |2-
        .:53 {
            errors
            health
            kubernetes cluster.local. in-addr.arpa ip6.arpa {
                pods insecure
                upstream
                fallthrough in-addr.arpa ip6.arpa
            }
```

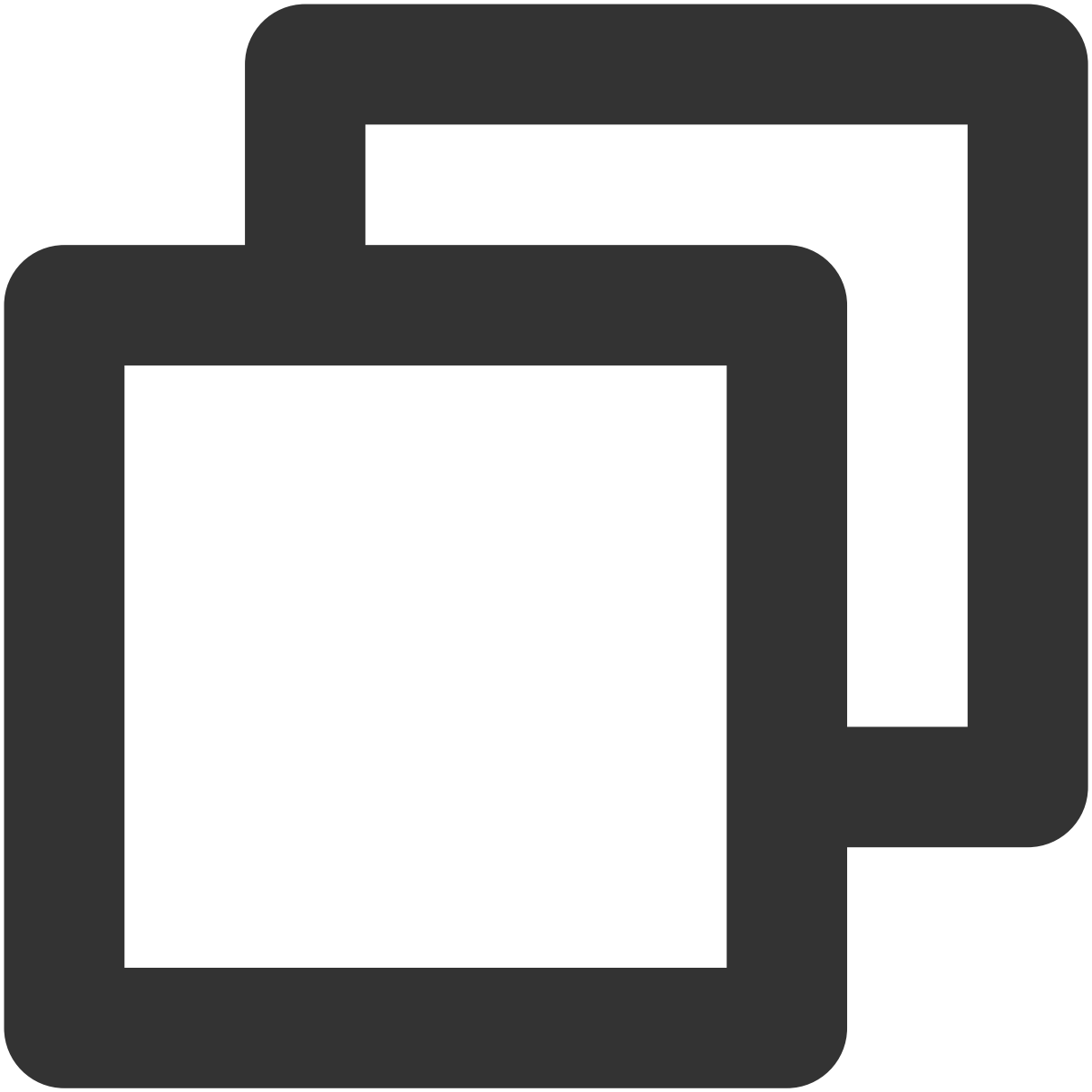
©2013-2022 Tencent Cloud. All rights reserved.

```
            rewrite name es.example.com es.logging.svc.cluster.local
            prometheus :9153
            forward . /etc/resolv.conf
            cache 30
            reload
            loadbalance
        }
kind: ConfigMap
metadata:
      labels:
        addonmanager.kubernetes.io/mode: EnsureExists
      name: coredns
      namespace: kube-system
```

Solution 3: Using the CoreDNS forward plugin to set the external DNS as the upstream DNS

1. Check the `forward` configuration. The default configuration of `forward` is as follows, which means that the domain name that is not in the cluster is resolved by the `nameserver` configured in the `/etc/resolv.conf` file of the node where CoreDNS is located.

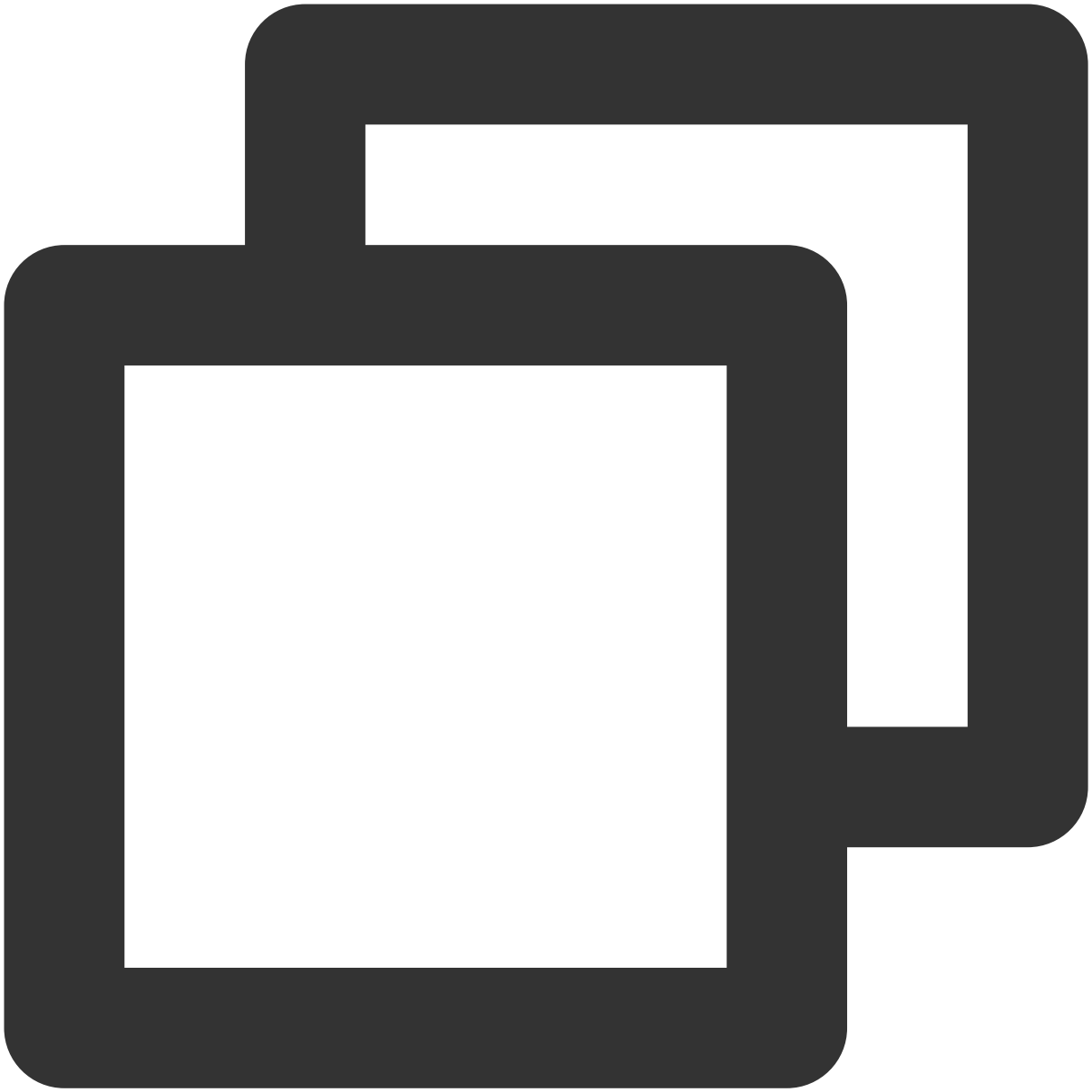
©2013-2022 Tencent Cloud. All rights reserved.



```
forward . /etc/resolv.conf
```

2. Configure `forward` by replacing `/etc/resolv.conf` explicitly with the IP address of the external DNS server, as shown below:

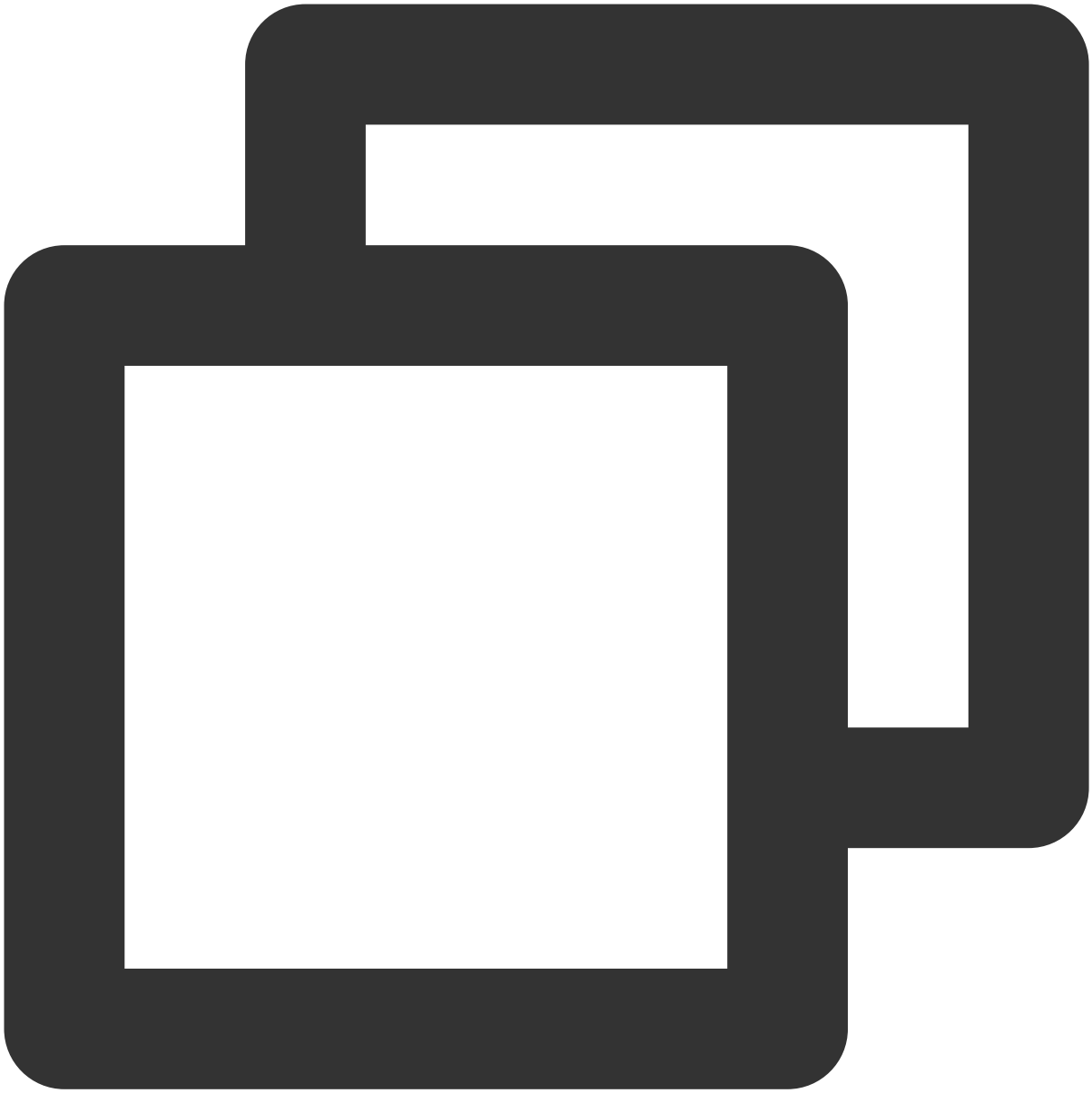
©2013-2022 Tencent Cloud. All rights reserved.



```
forward . 10.10.10.10
```

The complete configurations are as follows:

©2013-2022 Tencent Cloud. All rights reserved.



```
apiVersion: v1
data:
   Corefile: |2-
     .:53 {
         errors
         health
         kubernetes cluster.local. in-addr.arpa ip6.arpa {
             pods insecure
             upstream
             fallthrough in-addr.arpa ip6.arpa
         }
```

©2013-2022 Tencent Cloud. All rights reserved.

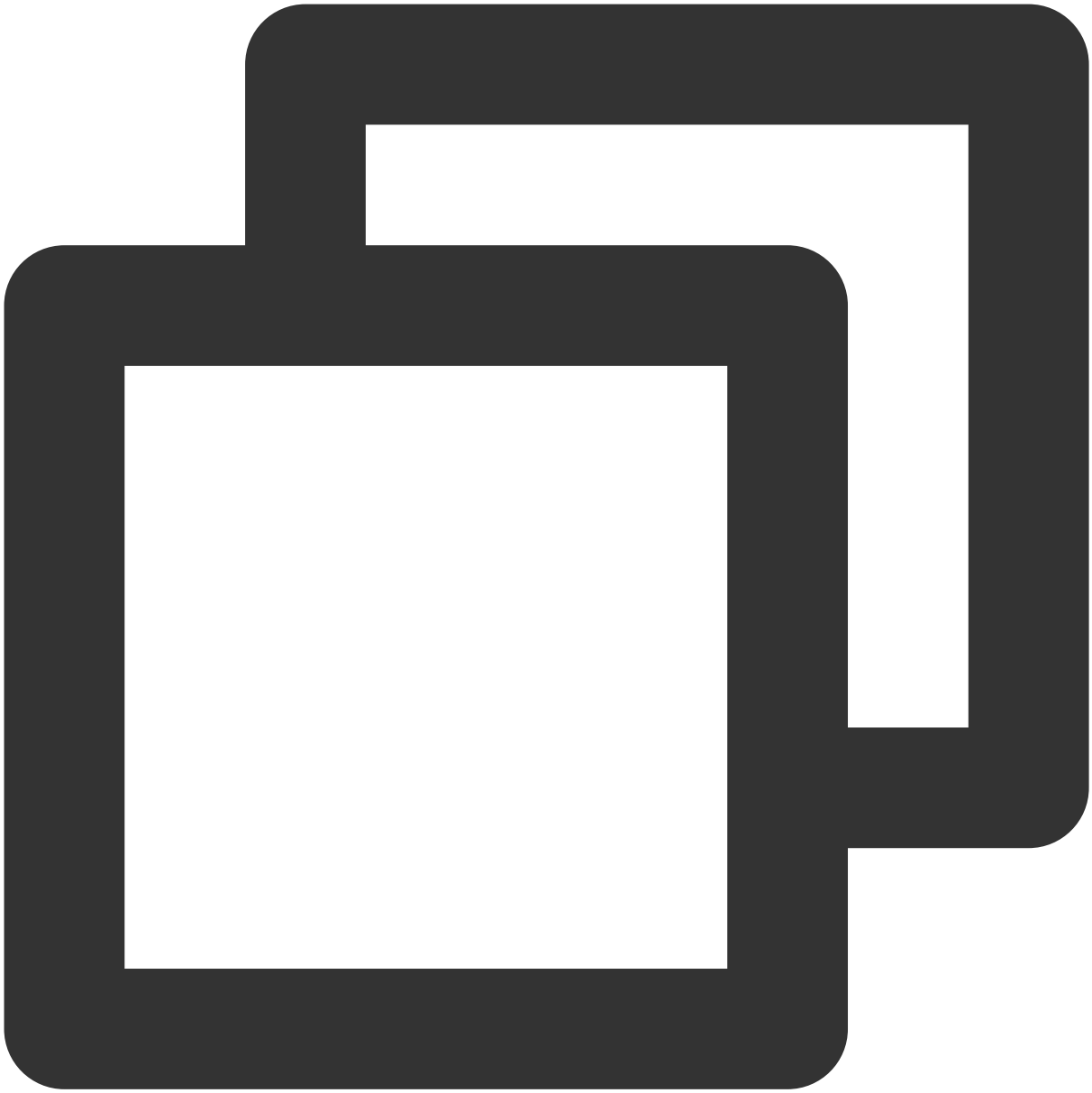
```
        prometheus :9153
        forward . 10.10.10.10
        cache 30
        reload
        loadbalance
    }
kind: ConfigMap
metadata:
  labels:
    addonmanager.kubernetes.io/mode: EnsureExists
  name: coredns
  namespace: kube-system
```

3. Configure the resolution records of the custom domain names to the external DNS. We recommend that you set the nameserver in `/etc/resolv.conf` on the node as the upstream of the external DNS. If it is not set as the upstream of the external DNS, some services may not work properly because the services rely on internal DNS resolution of Tencent Cloud. This document takes BIND 9 as an example to modify the configuration file and write the upstream DNS address into `forwarders`, as shown below:

Note

If the external DNS Server and the request source are not in the same Region, some Tencent domain names that do not support cross-region access may become invalid.

©2013-2022 Tencent Cloud. All rights reserved.



```
options {
     forwarders {
             183.60.83.19;
             183.60.82.98;
     };
     ...
```

Learn More

©2013-2022 Tencent Cloud. All rights reserved.

CoreDNS hosts plugin

CoreDNS rewrite plugin

CoreDNS forward plugin

©2013-2022 Tencent Cloud. All rights reserved.

Configuring ExternalDNS in TKE

Last updated：2023-11-21 10:52:40

This document introduces how to configure ExternalDSN in a Tencent Cloud TKE cluster.

What is ExternalDNS?

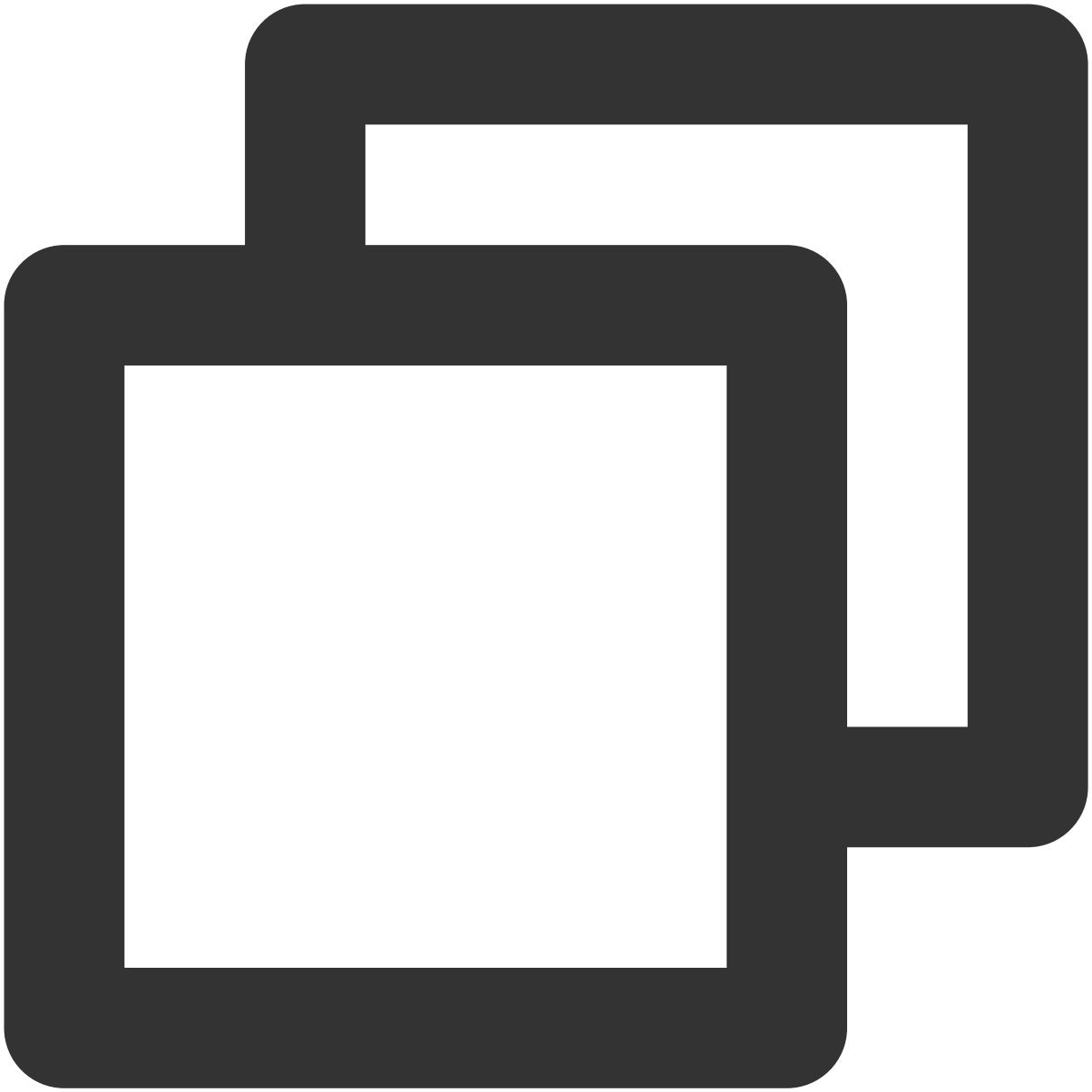
ExternalDNS can sync the public Kubernetes Services and Ingress to the DNS provider.
Inspired by Kubernetes DNS, Kubernetes' cluster-internal DNS server, ExternalDNS makes Kubernetes resources discoverable via public DNS servers. Like KubeDNS, it retrieves a list of resources (Services, Ingresses, etc.) from the Kubernetes API to determine a desired list of DNS records. Unlike KubeDNS, however, it's not a DNS server itself, but merely configures other DNS providers accordingly. For more information, see ExternalDNS Readme.

Directions

Configuring CAM Permissions for the API Key

Go to the Tencent Cloud CAM console and get the SecretId and SecretKey of the API key. Make sure the current user is assigned with the following permissions.

©2013-2022 Tencent Cloud. All rights reserved.



```
{
    "version": "2.0",
    "statement": [
        {
            "effect": "allow",
            "action": [
                "dnspod:ModifyRecord",
                "dnspod:DeleteRecord",
                "dnspod:CreateRecord",
                "dnspod:DescribeRecordList",
```

©2013-2022 Tencent Cloud. All rights reserved.

```
                "dnspod:DescribeDomainList"
            ],
            "resource": [
                "*"
            ]
        },
        {
            "effect": "allow",
            "action": [
                "privatedns:DescribePrivateZoneList",
                "privatedns:DescribePrivateZoneRecordList",
                "privatedns:CreatePrivateZoneRecord",
                "privatedns:DeletePrivateZoneRecord",
                "privatedns:ModifyPrivateZoneRecord"
            ],
            "resource": [
                "*"
            ]
        }
    ]
}
```

Deploying ExternalDNS Service

Configuring PrivateDNS or DNSPod

Tencent Cloud DNSPod provides free intelligent resolution services to all types of domain names. It features massive processing capability, flexible scalability and superior security, providing stable, fast and secure domain name resolution for your sites.

Tencent Cloud Private DNS is a private domain resolution and management service based on Tencent Cloud Virtual Private Cloud (VPC), providing you with safe, stable, and efficient private network resolution service. It supports quick building of a DNS system in VPCs to fulfill your needs.

To use private network DNS in Tencent Cloud environment:

Add the following parameter in the YAML file: `--tencent-cloud-zone-type=private`

Create a DNS domain in the PrivateDNS console. The DNS records are included in the DNS domain name records.

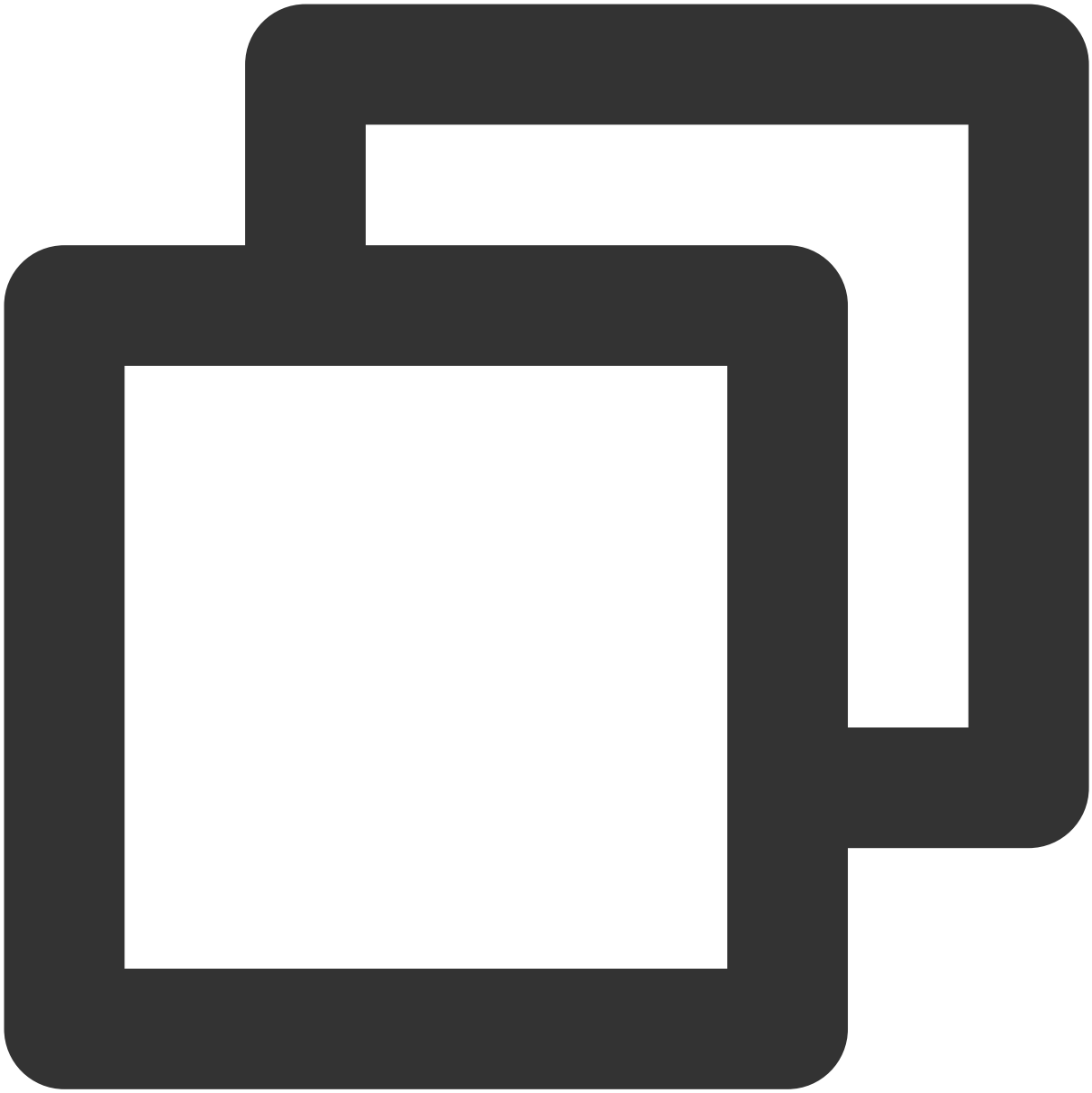
To use public network DNS in Tencent Cloud environment:

Add the following parameter in the YAML file: `--tencent-cloud-zone-type=public`

Create a DNS domain in the DNSPod console. The DNS records are included in the DNS domain name records.

Deploying resource objects in the Kuberentes cluster

©2013-2022 Tencent Cloud. All rights reserved.



```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: external-dns
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: external-dns
rules:
- apiGroups: [""]
```

©2013-2022 Tencent Cloud. All rights reserved.

```
  resources: ["services","endpoints","pods"]
  verbs: ["get","watch","list"]
- apiGroups: ["extensions","networking.k8s.io"]
  resources: ["ingresses"]
  verbs: ["get","watch","list"]
- apiGroups: [""]
  resources: ["nodes"]
  verbs: ["list"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: external-dns-viewer
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: external-dns
subjects:
- kind: ServiceAccount
  name: external-dns
  namespace: default
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: external-dns
data:
  tencent-cloud.json: |
    {
      "regionId": "ap-shanghai", # (Required) ID of the region where the cluster lo
      "secretId": "******",
      "secretKey": "******",
      "vpcId": "vpc-******" (Required), ID of the VPC where the cluster is deployed
    }
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: external-dns
spec:
  strategy:
    type: Recreate
  selector:
    matchLabels:
      app: external-dns
  template:
    metadata:
```

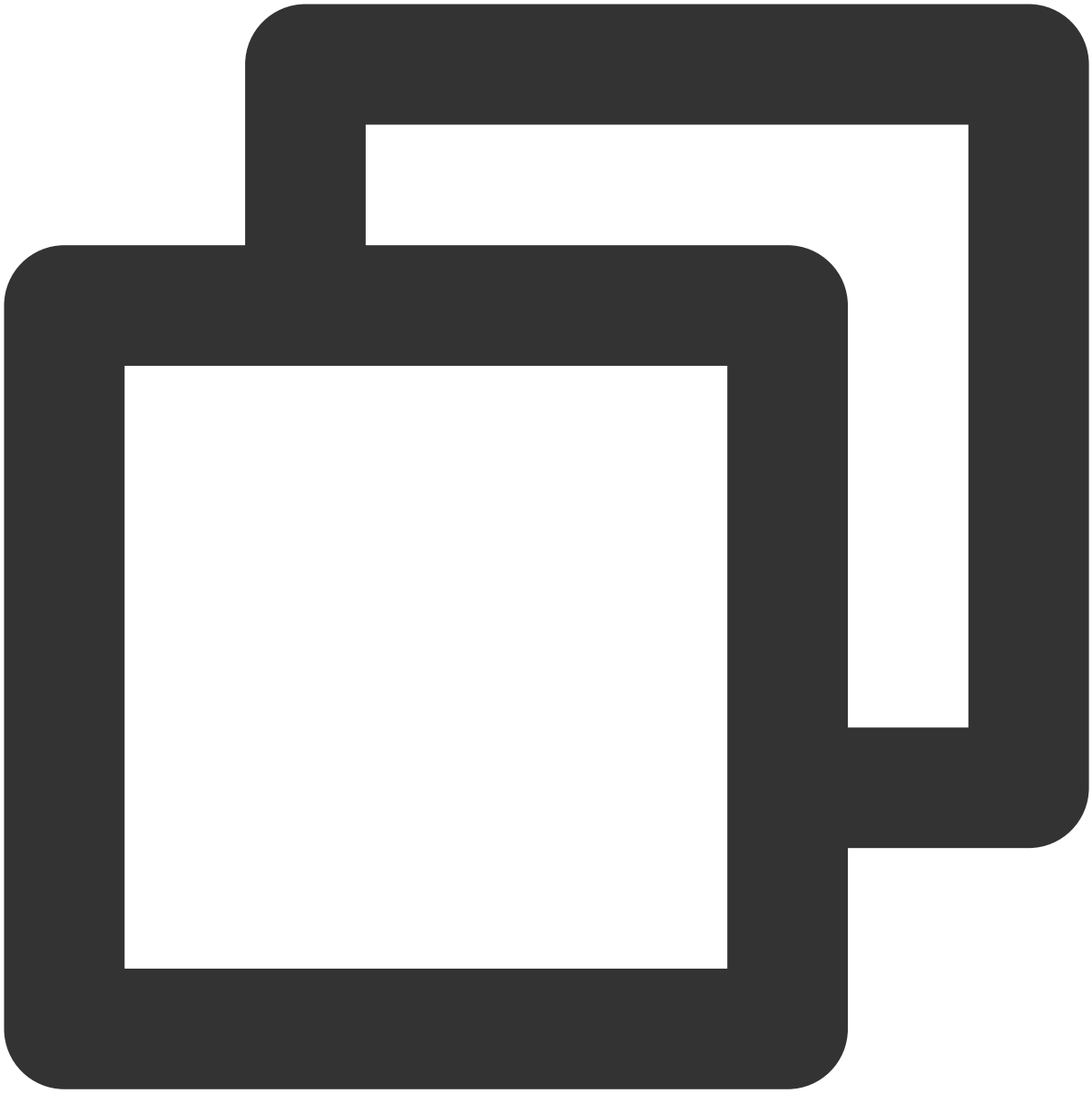
©2013-2022 Tencent Cloud. All rights reserved.

```
      labels:
        app: external-dns
    spec:
      containers:
      - args:
        - --source=service
        - --source=ingress
        - --domain-filter=external-dns-test.com # Make ExternalDNS see only the hos
        - --provider=tencentcloud
        - --policy=sync # Set it to `upssert-only` to prevent ExternalDNS from dele
        - --tencent-cloud-zone-type=private # Only look at private hosted zones. To
        - --tencent-cloud-config-file=/etc/kubernetes/tencent-cloud.json
        image: ccr.ccs.tencentyun.com/tke-market/external-dns:v1.0.0
        imagePullPolicy: Always
        name: external-dns
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
        volumeMounts:
        - mountPath: /etc/kubernetes
          name: config-volume
          readOnly: true
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      serviceAccount: external-dns
      serviceAccountName: external-dns
      terminationGracePeriodSeconds: 30
      volumes:
      - configMap:
          defaultMode: 420
          items:
          - key: tencent-cloud.json
            path: tencent-cloud.json
          name: external-dns
        name: config-volume
```

Example

Creating a Service named “nginx”

©2013-2022 Tencent Cloud. All rights reserved.



```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  annotations:
    external-dns.alpha.kubernetes.io/hostname: nginx.external-dns-test.com # Public
    external-dns.alpha.kubernetes.io/internal-hostname: nginx-internal.external-dns
    external-dns.alpha.kubernetes.io/ttl: "600"
spec:
  type: LoadBalancer
  ports:
```

©2013-2022 Tencent Cloud. All rights reserved.

```
    - port: 80
      name: http
      targetPort: 80
    selector:
      app: nginx
  ---
  apiVersion: apps/v1
  kind: Deployment
  metadata:
    name: nginx
  spec:
    selector:
      matchLabels:
        app: nginx
    template:
      metadata:
        labels:
          app: nginx
      spec:
        containers:
        - image: nginx
          name: nginx
          ports:
          - containerPort: 80
            name: http
```

`nginx.external-dns-test.com` will record the service's loadbalancer VIP.

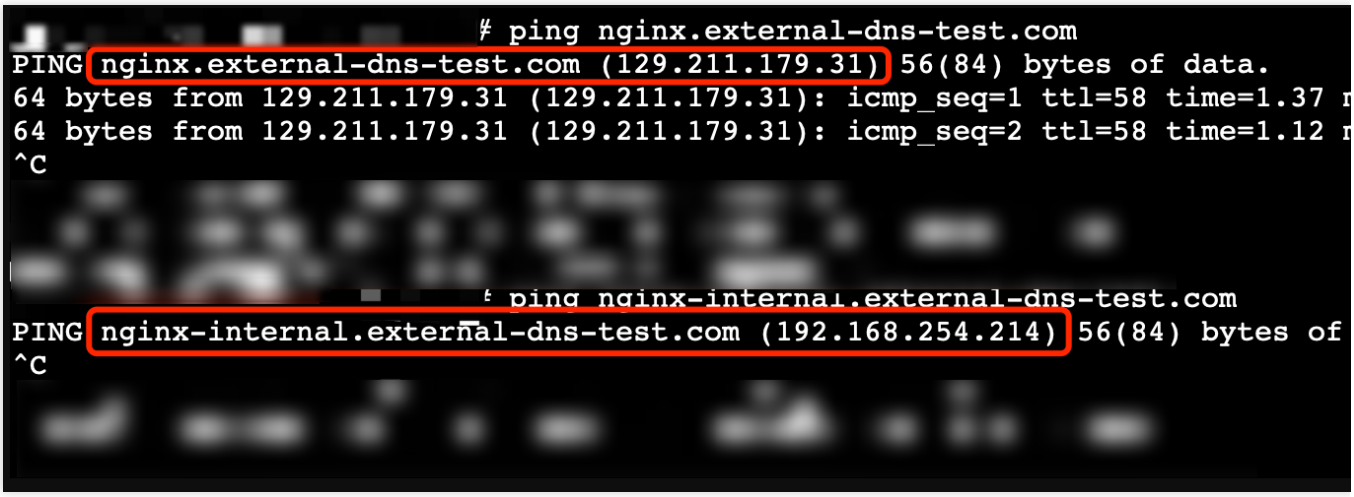
`nginx-internal.external-dns-test.com` will record the service's ClusterIP. The TTL of all DNS records is 600.

Verification

A Service named “nginx” is created with the ClusterIP `192.168.254.214` and Loadbalancer VIP `129.211.179.31`. As shown below:

Log in to a node in the same VPC as the cluster. PING the domain name in the annotation of nginx service. The domain name will be resolved to the ClusterIP and Loadbalancer VIP. As shown below:

©2013-2022 Tencent Cloud. All rights reserved.



©2013-2022 Tencent Cloud. All rights reserved.

Using Network Policy for Network Access Control

Last updated：2022-12-12 17:25:13

Network Policy Introduction

A network policy is a resource provided by Kubernetes to define the Pod-based network isolation policy. It specifies whether a group of Pods can communicate with other groups of Pods and other network endpoints.

Scenarios

In TKE, Pod Networking is implemented by a high-performance Pod network based on the VPC at the IaaS layer, and service proxy is provided by the ipvs and iptables modes supported by kube-proxy. TKE provides network isolation through the Network Policy add-on.

Enabling NetworkPolicy in TKE

The NetworkPolicy add-on is available for TKE now. You can install it with a few steps. For directions, see Network Policy.

NetworkPolicy Configuration Example

> Note
> The apiVersion of the resource object varies based on the cluster Kubernetes version. You can run the command `kubectl api-versions` to view the apiVersion of the current resource object.

- The Pods in the nsa namespace can access one another and cannot be accessed by any other Pods.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
name: npa
```

©2013-2022 Tencent Cloud. All rights reserved.

```
namespace: nsa
spec:
ingress:
- from:
- podSelector: {}
podSelector: {}
policyTypes:
- Ingress
```

- The Pods in nsa namespace cannot be accessed by any Pods.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
name: npa
namespace: nsa
spec:
podSelector: {}
policyTypes:
- Ingress
```

- Pods in the nsa namespace can only be accessed by Pods in the namespace with the app: nsb tag on port 6379 or the TCP port.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
name: npa
namespace: nsa
spec:
ingress:
- from:
- namespaceSelector:
matchLabels:
app: nsb
ports:
- protocol: TCP
port: 6379
podSelector: {}
policyTypes:
- Ingress
```

©2013-2022 Tencent Cloud. All rights reserved.

- Pods in the nsa namespace can access port 5978 or the TCP port of the network endpoint with a CIDR block of 14.215.0.0/16 but cannot access any other network endpoints. This method can be used to configure an allowlist to allow in-cluster services to access external network endpoints.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
name: npa
namespace: nsa
spec:
egress:
- to:
- ipBlock:
cidr: 14.215.0.0/16
ports:
- protocol: TCP
port: 5978
podSelector: {}
policyTypes:
- Egress
```

- Pods in the default namespace can only be accessed by the network endpoint with a CIDR block of 14.215.0.0/16 on port 80 or the TCP port and cannot be accessed by any other network endpoints.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
name: npd
namespace: default
spec:
ingress:
- from:
- ipBlock:
cidr: 14.215.0.0/16
ports:
- protocol: TCP
port: 80
podSelector: {}
policyTypes:
- Ingress
```

Feature Testing of NetworkPolicy

©2013-2022 Tencent Cloud. All rights reserved.

Run the K8s community's e2e test for `NetworkPolicy`. The results are as follows:

NetworkPolicy Feature	Supported
should support a `default-deny` policy	Yes
should enforce policy to allow traffic from pods within server namespace based on PodSelector	Yes
should enforce policy to allow traffic only from a different namespace, based on NamespaceSelector	Yes
should enforce policy based on PodSelector with MatchExpressions	Yes
should enforce policy based on NamespaceSelector with MatchExpressions	Yes
should enforce policy based on PodSelector or NamespaceSelector	Yes
should enforce policy based on PodSelector and NamespaceSelector	Yes
should enforce policy to allow traffic only from a pod in a different namespace based on PodSelector and NamespaceSelector	Yes
should enforce policy based on Ports	Yes
should enforce multiple, stacked policies with overlapping podSelectors	Yes
should support allow-all policy	Yes
should allow ingress access on one named port	Yes
should allow ingress access from namespace on one named port	Yes
should allow egress access on one named port	No
should enforce updated policy	Yes
should allow ingress access from updated namespace	Yes
should allow ingress access from updated pod	Yes
should deny ingress access to updated pod	Yes
should enforce egress policy allowing traffic to a server in a different namespace based on PodSelector and NamespaceSelector	Yes
should enforce multiple ingress policies with ingress allow-all policy taking precedence	Yes
should enforce multiple egress policies with egress allow-all policy taking precedence	Yes

©2013-2022 Tencent Cloud. All rights reserved.

NetworkPolicy Feature	Supported
should stop enforcing policies after they are deleted	Yes
should allow egress access to server in CIDR block	Yes
should enforce except clause while egress access to server in CIDR block	Yes
should enforce policies to check ingress and egress policies can be controlled independently based on PodSelector	Yes

Feature Testing of NetworkPolicy (legacy)

A large number of Nginx services are deployed in the Kubernetes cluster, and a fixed service is measured with ApacheBench (ab). The QPS values in Kube-router-enabled and Kube-router-disabled scenarios are compared to measure the performance loss caused by Kube-router.

Test environment

- VM quantity: 100
- VM configuration: 2 CPU cores, 4 GB memory
- VM OS: Ubuntu
- Kubernetes version: 1.10.5
- kube-router version: 0.2.0

Test process

1. Deploy one service corresponding to two Pods (Nginx) as the test group.

2. Deploy 1,000 services with each of them corresponding to 2/6/8 Pods (Nginx) as the interference group.

3. Deploy a NetworkPolicy rule to ensure that all Pods are selected to produce a sufficient number of iptables rules.

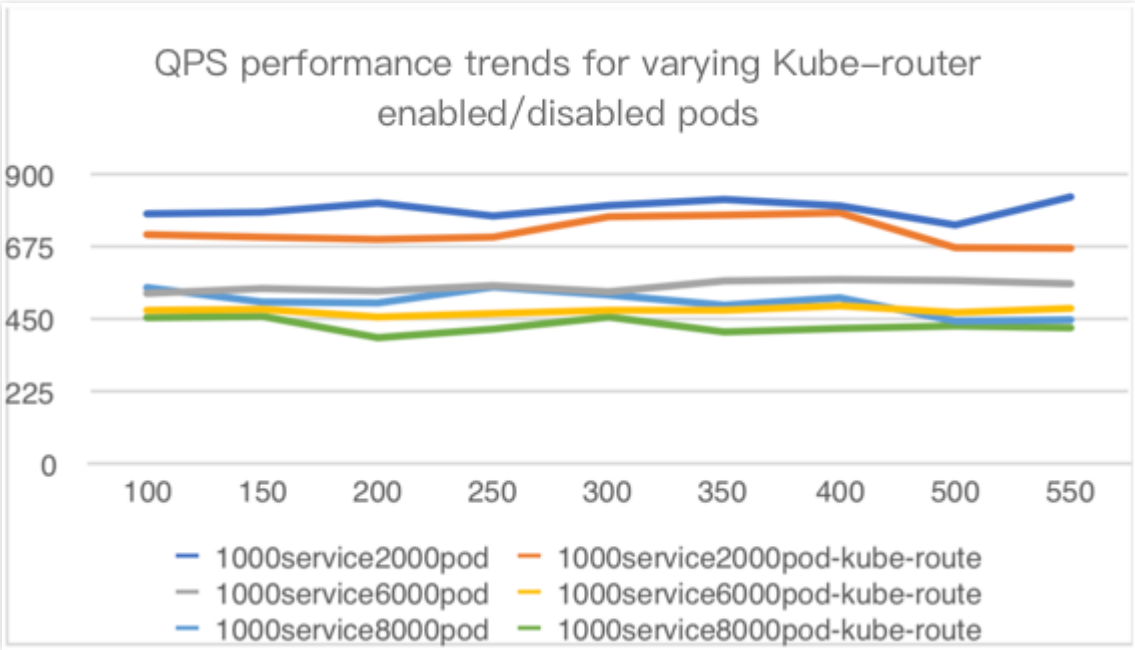
```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
name: npd
namespace: default
spec:
ingress:
- from:
- ipBlock:
```

©2013-2022 Tencent Cloud. All rights reserved.

```
cidr: 14.215.0.0/16
ports:
- protocol: TCP
port: 9090
- from:
- ipBlock:
cidr: 14.215.0.0/16
ports:
- protocol: TCP
port: 8080
- from:
- ipBlock:
cidr: 14.215.0.0/16
ports:
- protocol: TCP
port: 80
podSelector: {}
policyTypes:
- Ingress
```

4. Perform an A/B test to test the service in the test group and record the QPS.

   The following figure shows the obtained performance curve.



- In the legend:
  - 1000service2000pod, 1000service6000pod, and 1000service8000pod are the performances when kube-route is not enabled for Pod.

©2013-2022 Tencent Cloud. All rights reserved.

  - 1000service2000pod-kube-route, 1000service6000pod-kube-route, and 1000service8000pod-kube-route are the performances when kube-route is enabled for Pod.
- X axis: A/B concurrency
- Y axis: QPS

Test conclusion

As the number of Pods increases from 2,000 to 8,000, the performance when Kube-router is enabled is 10% to 20% lower than when it is disabled.

Notes

Kube-router versions provided by Tencent Cloud

The NetworkPolicy add-on is based on the community's Kube-Router project. During the development of this add-on, the Tencent Cloud PaaS team actively built a community, provided features, and fixed bugs. The PRs we committed that were incorporated into the community are listed as follows:

- processing k8s version for NPC #488
- Improve health check for cache synchronization #498
- Make the comments of the iptables rules in NWPLCY chains more accurate and reasonable #527
- Use ipset to manage multiple CIDRs in a network policy rule #529
- Add support for 'except' feature of network policy rule#543
- Avoid duplicate peer pods in npc rules variables #634
- Support named port of network policy #679

©2013-2022 Tencent Cloud. All rights reserved.

Deploying NGINX Ingress on TKE

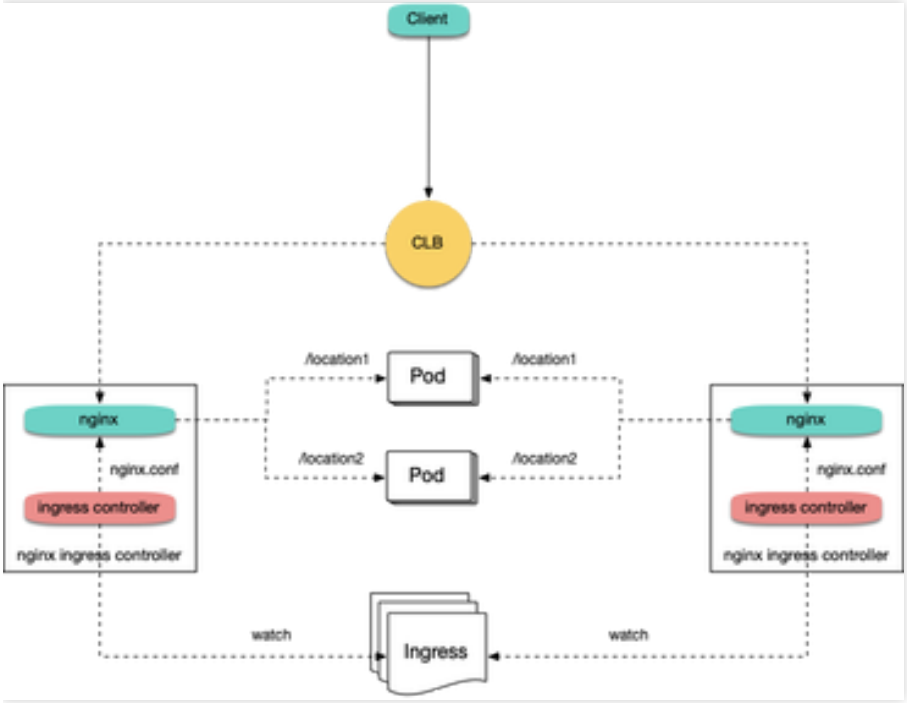
Last updated：2020-12-16 12:18:07

Overview

Nginx Ingress provides robust features and extremely high performance as well as multiple deployment modes. This document introduces the three deployment schemes of Nginx Ingress on Tencent Kubernetes Engine (TKE): Deployment + LB, Daemonset + HostNetwork + LB, and Deployment + LB directly connected to Pod and their deployment methods.

Nginx Ingress Introduction

Nginx Ingress is an implementation of Kubernetes Ingress. By watching the Ingress resources of Kubernetes clusters, it converts Ingress rules into an Nginx configuration to enable Nginx to perform Layer-7 traffic forwarding, as shown in the figure below:



Nginx Ingress can be implemented in the following two modes. This document mainly introduces the implementation of Kubernetes in the open-source community:

- Implementation of Kubernetes in the Open-Source Community
- Official Implementation of Nginx

©2013-2022 Tencent Cloud. All rights reserved.

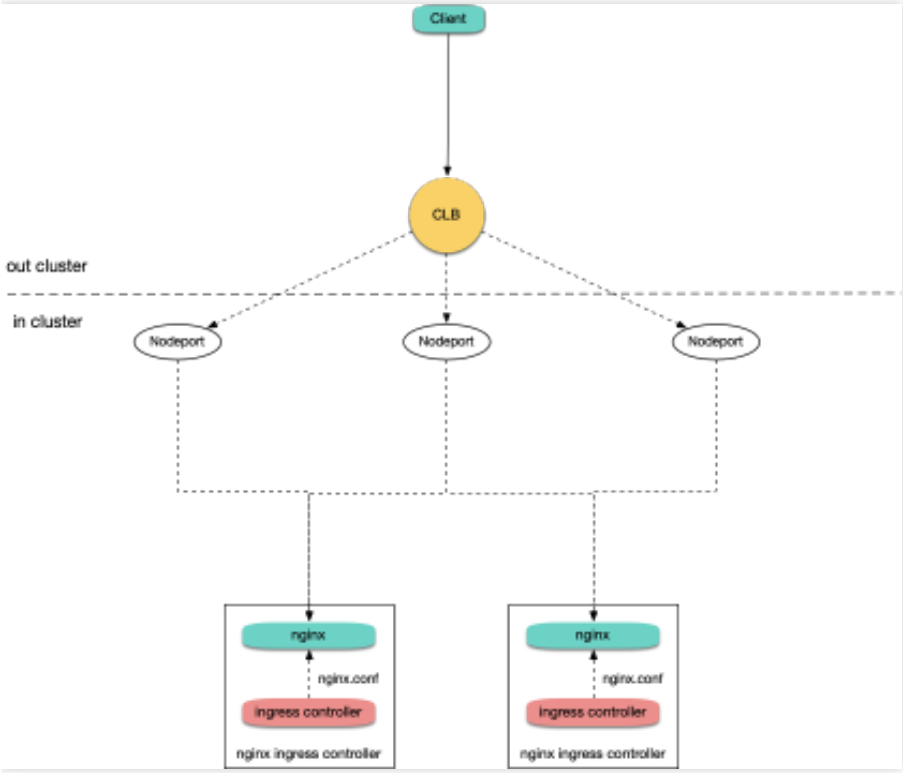
Suggestions for deployment solution selection

Based on a comparison of the three deployment solutions for Nginx Ingress on TKE, this document offers the following selection suggestions:

1. Deployment + LB: this solution is relatively simple and applicable to general scenarios, but performance issues may arise in large-scale and high-concurrency scenarios. If your performance requirements are low, you can consider adopting this solution.
2. Daemonset + HostNetwork + LB: the use of hostNetwork offers good performance, but manual maintenance of CLBs and Nginx Ingress nodes is required and auto scaling cannot be implemented. Therefore, we do not recommend this solution.
3. Deployment + LB directly connected to pod: this solution offers good performance, without the need for manual CLB maintenance, making this the ideal solution. However, in this solution, clusters need to support VPC-CNI. If the existing clusters use the VPC-CNI network plug-in or the Global Router network plug-in and have enabled support for VPC-CNI (mixed use of two modes), we recommend that you adopt this solution.

Solution 1: Deployment + LB

The simplest way to deploy Nginx Ingress on TKE is to deploy Nginx Ingress Controller in Deployment mode and create a LoadBalancer-type Service for it (automatically creating a CLB or binding an existing CLB) to enable the CLB to receive external traffic and forward it into Nginx Ingress, as shown in the figure below:



©2013-2022 Tencent Cloud. All rights reserved.

Currently, by default, a LoadBalancer-type Service on TKE is implemented based on NodePort: the CLB binds the NodePort of each node as the RS (Real Server) and forwards traffic to the NodePort of each node. Then through Iptables or IPVS, nodes route requests to the corresponding backend pod of the Service (namely the pod of Nginx Ingress Controller). Subsequently, if nodes are added or deleted, the CLB will automatically update the node NodePort binding.
Run the following commands to install Nginx Ingress:

```
kubectl create ns nginx-ingress
```

```
kubectl apply -f https://raw.githubusercontent.com/TencentCloudContainerTeam/manifest/master/nginx-ingress/nginx-ingress-deployment.yaml -n nginx-ingress
```

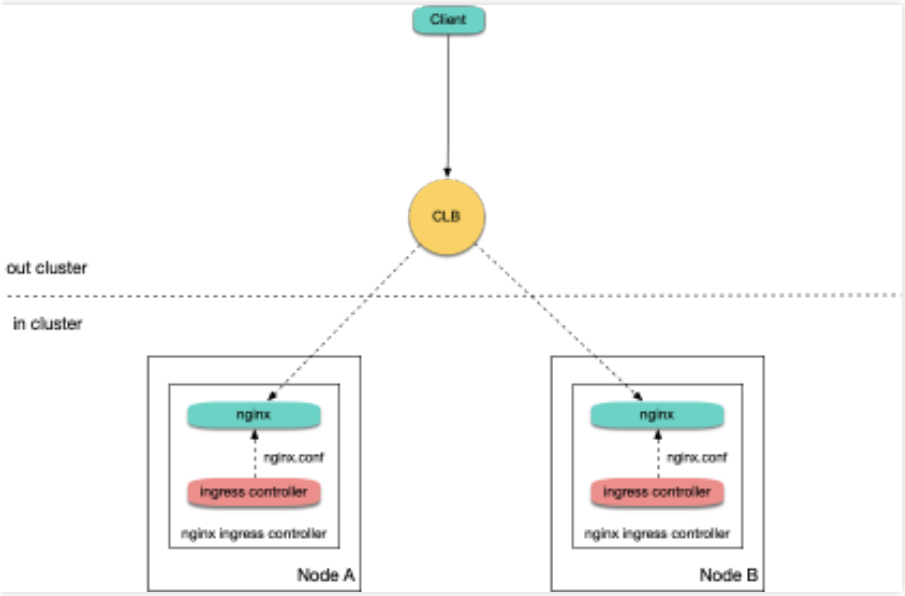
Solution 2: Daemonset + HostNetwork + LB

In solution 1, traffic passes through a NodePort layer, introducing one more layer for forwarding, which leads to the following issues:

- The forwarding path is relatively long: after reaching NodePort, traffic goes through the LB within Kubernetes and is then forwarded through Iptables or IPVS to Nginx. This increases network time consumption.
- Passing through NodePort will necessarily cause SNAT. If traffic is too concentrated, port exhaustion or conntrack insertion conflicts can easily occur, leading to packet loss and causing some traffic exceptions.
- The NodePort of each node also serves as a CLB. If the CLB is bound with the NodePorts of a large number of nodes, the LB status is distributed among each node, which can easily cause a global load imbalance.
- The CLB carries out health probes on NodePort, and probe packets are ultimately forwarded to the Pods of Nginx Ingress. If the CLB is bound with too many nodes, and the Nginx Ingress has a small number of pods, the probe packets will put immense pressure on Nginx Ingress.

In solution 2, the following solution is proposed:
Nginx Ingress uses hostNetwork, and the CLB is directly bound with node IP address + port (80,443), without passing through NodePort. With the use of hostNetwork, the pods of Nginx Ingress cannot be scheduled to the same node. To avoid port listening conflicts, you can preselect some nodes as edge nodes dedicated to the deployment of Nginx Ingress and label them. Then, Nginx Ingress can be deployed as a DaemonSet on these nodes. The following figure shows the architecture:

©2013-2022 Tencent Cloud. All rights reserved.



To install Nginx Ingress, perform the following steps:

1. Run the following command to attach a label to the nodes planned for the deployment of Nginx Ingress (be sure to replace the node names):

```
kubectl label node 10.0.0.3 nginx-ingress=true
```

2. Run the following commands to deploy Nginx Ingress on these nodes:

```
kubectl create ns nginx-ingress
```

```
kubectl apply -f https://raw.githubusercontent.com/TencentCloudContainerTeam/manifest/master/nginx-ingress/nginx-ingress-daemonset-hostnetwork.yaml -n nginx-ingress
```

3. Manually create a CLB, create a TCP listener for ports 80 and 443, and bind them with ports 80 and 443 of the nodes where Nginx Ingress has been deployed.

Solution 3: Deployment + LB Directly Connected to Pod

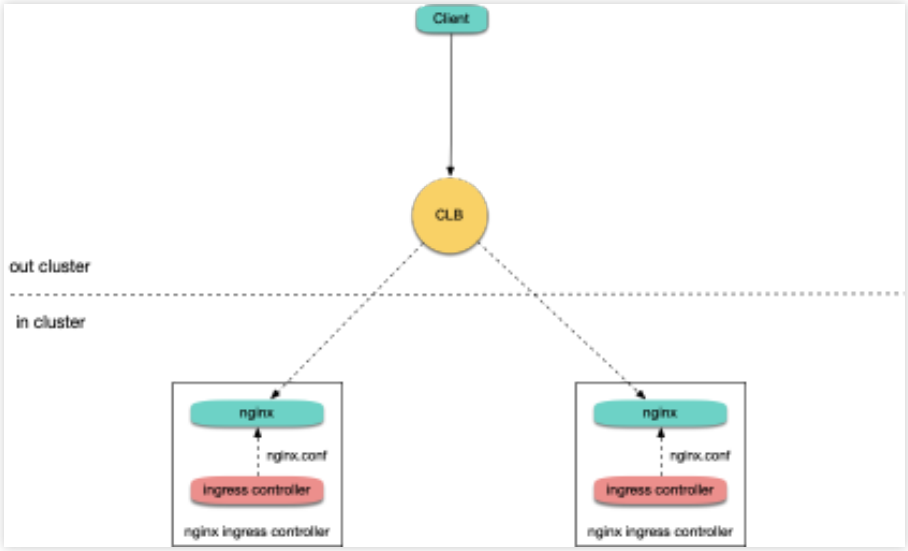
Solution 2 offers more advantages than solution 1, but it has the following issues:

- It increases the OPS cost for manual maintenance of the CLB and Nginx Ingress nodes.
- Nginx Ingress nodes need to be planned in advance. When Nginx Ingress nodes are added or deleted, you need to manually bind or unbind nodes on the CLB console.
- Automatic scaling is not supported.

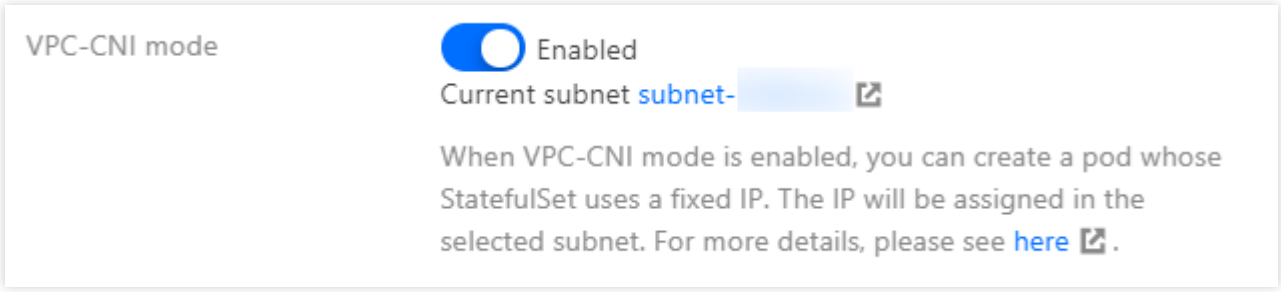
©2013-2022 Tencent Cloud. All rights reserved.

In solution 3, the following solution is proposed:

- If the network mode is VPC-CNI and all pods use ENI, you can directly bind the CLB with the ENI pods, bypassing NodePort. This saves the trouble of manual management of the CLB and enables support for automatic scaling, as shown in the figure below:



- If the network mode is Global Router, you can go to the cluster information page and enable VPC-CNI support for the cluster. This enables the mixed use of the two network modes, as shown in the figure below:



After ensuring that the cluster supports VPC-CNI, run the following commands in sequence to install Nginx Ingress:

```
kubectl create ns nginx-ingress
```

```
kubectl apply -f https://raw.githubusercontent.com/TencentCloudContainerTeam/manifest/master/nginx-ingress/nginx-ingress-deployment-eni.yaml -n nginx-ingress
```

FAQs

How can private network Ingress be supported?

©2013-2022 Tencent Cloud. All rights reserved.

In solution 2: Daemonset + HostNetwork + LB, the CLB is manually managed. When creating a CLB, you can select public network or private network. In solution 1: Deployment + LB and solution 3: Deployment + LB directly connected to pod, public network CLBs are created by default.

To use a private network, you can redeploy YAML and add a key to the Service in nginx-ingress-controller, for example, `service.kubernetes.io/qcloud-loadbalancer-internal-subnetid`, with value set to the annotation of the subnet ID created by the private network CLB. Refer to the following code:

```
apiVersion: v1
kind: Service
metadata:
annotations:
service.kubernetes.io/qcloud-loadbalancer-internal-subnetid: subnet-xxxxxx # valu
e should be replaced with a subnet ID in the VPC where the cluster belongs.
labels:
app: nginx-ingress
component: controller
name: nginx-ingress-controller
```

How can an existing LB be shared?

In solution 1: Deployment + LB and solution 3: Deployment + LB directly connected to Pod, new CLBs are automatically created by default. The traffic entry address of Ingress depends on the IP address of the newly created CLB. If a business is dependent upon the entry address, you can bind Nginx Ingress with an existing CLB.

The specific method is to redeploy YAML and add a key to the Service in nginx-ingress-controller, such as `service.kubernetes.io/tke-existed-lbid`, with value set to the annotation of the CLB ID. Refer to the following code:

```
apiVersion: v1
kind: Service
metadata:
annotations:
service.kubernetes.io/tke-existed-lbid: lb-6swtxxxx # value should be replaced wi
th the CLB ID.
labels:
app: nginx-ingress
component: controller
name: nginx-ingress-controller
```

What's the size of the Nginx Ingress public network bandwidth?

There are two types of Tencent Cloud accounts: bill-by-IP accounts and bill-by-CVM accounts:

©2013-2022 Tencent Cloud. All rights reserved.

> ⚠ **Note：**
>
> You can refer to Distinguishing Between Tencent Cloud Account Types to identify your account type.

- **Bill-by-IP account type:** bandwidth is moved to the CLB or IP address for management.
  If your account is a bill-by-IP account, the Nginx Ingress bandwidth equals the purchased CLB bandwidth, which is 10 Mbps by default (pay-as-you-go) and can be adjusted as needed.
- **Bill-by-CVM account type:** bandwidth is managed on CVMs.
  If your account is a bill-by-CVM account, Nginx Ingress uses a public network CLB, and the public network bandwidth of Nginx Ingress is the sum of the bandwidth of all TKE nodes bound with the CLB. If solution 3: Deployment + LB directly connected to pod is adopted, the CLB is directly connected to pods, which means that the CLB is directly bound with ENI. In that case, the public network bandwidth of Nginx Ingress is the sum of the bandwidth of all nodes where Nginx Ingress Controller Pods are scheduled.

How can I create an Ingress?

When you deploy Nginx Ingress on TKE and need to use Nginx Ingress to manage Ingress, if you cannot create an Ingress on the TKE console, you can use YAML to create an Ingress and you need to specify the annotation of Ingress Class for each Ingress. Refer to the following code:

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
name: test-ingress
annotations:
kubernetes.io/ingress.class: nginx # this is the key part
spec:
rules:
- host: *
http:
paths:
- path: /
backend:
serviceName: nginx-v1
servicePort: 80
```

How can I enable monitoring?

For Nginx Ingress installed through the method in How can I create an Ingress, the metrics port has been opened and can be used for Prometheus collection. If prometheus-operator is installed in the cluster, you can use ServiceMonitor to collect monitoring data for Nginx Ingress. Refer to the following code:

©2013-2022 Tencent Cloud. All rights reserved.

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
name: nginx-ingress-controller
namespace: nginx-ingress
labels:
app: nginx-ingress
component: controller
spec:
endpoints:
- port: metrics
interval: 10s
namespaceSelector:
matchNames:
- nginx-ingress
selector:
matchLabels:
app: nginx-ingress
component: controller
```

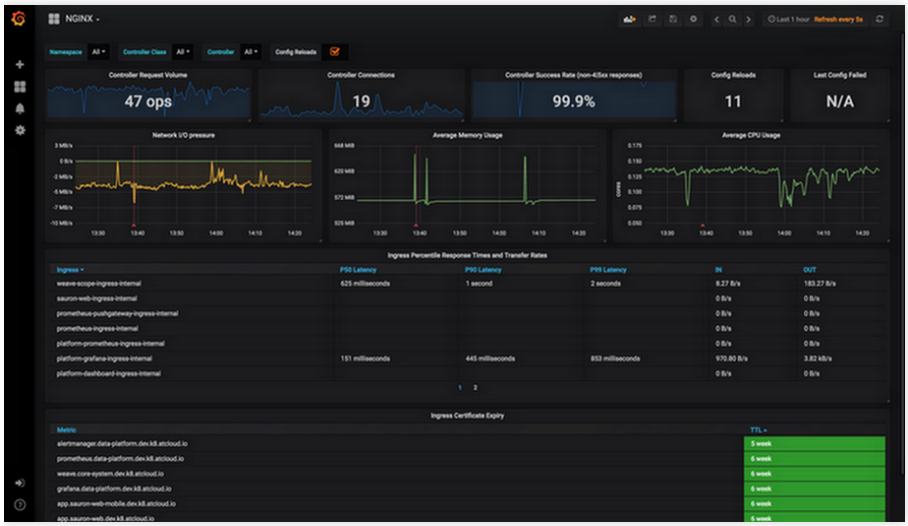
For native Prometheus configuration, refer to the following code:

```
- job_name: nginx-ingress
scrape_interval: 5s
kubernetes_sd_configs:
- role: endpoints
namespaces:
names:
- nginx-ingress
relabel_configs:
- action: keep
source_labels:
- __meta_kubernetes_service_label_app
- __meta_kubernetes_service_label_component
regex: nginx-ingress;controller
- action: keep
source_labels:
- __meta_kubernetes_endpoint_port_name
regex: metrics
```

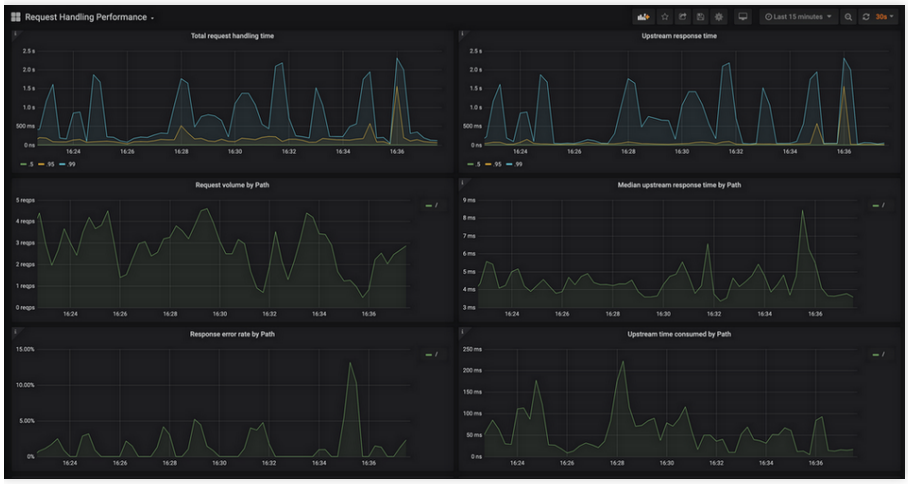
After collecting monitoring data, you can configure the dashboards provided by the Nginx Ingress community for grafana and display data.

In actual operation, you can directly copy JSON data and import it to grafana to import dashboards. `nginx.json` is used to display the various regular monitoring dashboards for Nginx Ingress, as shown in the figure below:

©2013-2022 Tencent Cloud. All rights reserved.



`request-handling-performance.json` is used to display the performance monitoring dashboard of Nginx Ingress, as shown in the figure below:



References

- TKE Service YAML Sample
- TKE Service Using an Existing CLB
- Distinguishing Between Tencent Cloud Account Types

©2013-2022 Tencent Cloud. All rights reserved.

Nginx Ingress High-Concurrency Practices

Last updated：2022-08-02 10:01:36

Overview

Nginx Ingress Controller implements the Kubernetes Ingress API based on Nginx. When Nginx, which is a high-performance gateway, runs in the production environment, you need to optimize its parameters to make full use of its high performance. The deployment YAML file in Deploying Nginx Ingress on TKE has already optimized some performance parameters for Nginx.
This document introduces the methods and principles for optimizing the global configuration and kernel parameters of Nginx Ingress to better adapt to high-concurrency business scenarios.

Optimizing Kernel Parameters

You can use the following methods to optimize the kernel parameters of Nginx Ingress and use the initContainers method to configure the kernel parameters. For more information, see Configuration examples.

- Increasing the size of the connection queue
- Expanding the range of source ports
- Reusing TIME_WAIT
- Increasing the maximum number of file handles
- Configuration examples

Increasing the size of the connection queue

In a high-concurrence environment, queue overflow may occur if the connection queue is too small, failing to establish some connections. The size of the connection queue of the process listener socket is controlled by the `net.core.somaxconn` kernel parameter. By adjusting the value of this parameter, you can enlarge the Nginx Ingress connection queue.

When a process calls the listen system to listen on ports, it passes in the backlog parameter, which determines the size of the socket connection queue. The value of the backlog parameter is not greater than that of somaxconn. When the Go program standard library listens, it reads and uses the somaxconn value as the queue size by default. However, Nginx does not read somaxconn when listening on the socket, but reads `nginx.conf`. In the listening port configuration items in `nginx.conf`, you can configure the backlog parameter to specify a connection queue size for Nginx port listening. The following shows a sample configuration:

©2013-2022 Tencent Cloud. All rights reserved.

```
server {
listen 80 backlog=1024;
...
```

If the value of backlog is not specified, it defaults to 511. The detailed description of the backlog parameter is as follows:

```
backlog=number
sets the backlog parameter in the listen() call that limits the maximum length fo
r the queue of pending connections. By default, backlog is set to -1 on FreeBSD,
DragonFly BSD, and MacOS, and to 511 on other platforms.
```

By default, even if the set value of somaxconn exceeds 511, the maximum size of the connection queue for Nginx port listening is still 511. For this reason, connection queue overflow may occur in a high-concurrency environment.

Nginx Ingress performs the preceding configuration differently. Nginx Ingress Controller can automatically read and use the value of somaxconn as the backlog value and write it to the generated nginx.conf file. Therefore, the connection queue size of Nginx Ingress is determined by somaxconn only, and the size defaults to 4096 in TKE.
In a high-concurrency environment, we recommend that you run the following command to set the somaxconn value to 65535:

```
sysctl -w net.core.somaxconn=65535
```

Expanding the range of source ports

In a high-concurrency environment, Nginx Ingress uses large numbers of source ports to establish connections with the upstream. The range of source ports is randomly selected from the range defined in the `net.ipv4.ip_local_port_range` kernel parameter. In a high-concurrency environment, a small port range can easily exhaust source ports, resulting in abnormal connections.
The default source port range of pods created in a TKE environment is 32768 - 60999. We recommend that you run the following command to expand the range to 1024 - 65535:

```
sysctl -w net.ipv4.ip_local_port_range="1024 65535"
```

Reusing TIME_WAIT

If the concurrency of non-persistent connections is high, the number of connections in the TIME_WAIT state in netns will also be large, By default, connections in the TIME_WAIT state have to wait for a period of 2MSL before being released, and therefore the source ports will be occupied for a long time. When the number of connections in this state exceeds a certain number, new connections may fail to be established.

©2013-2022 Tencent Cloud. All rights reserved.

We recommend that you run the following command to enable TIME_WAIT reuse for Nginx Ingress, which reuses TIME_WAIT connections for new TCP connections:

```
sysctl -w net.ipv4.tcp_tw_reuse=1
```

Increasing the maximum number of file handles

When Nginx is used as a reverse proxy, each request establishes a connection with the client and upstream server respectively, which occpuies two file handles. Therefore, the theoretical maximum number of connections that Nginx can process simultaneously is half the maximum number of file handles set for the system.

The maximum number of file handles of the system is controlled by the `fs.file-max` kernel parameter, which defaults to 838860 in TKE. We recommend that you run the following command to set the maximum number of file handles to 1048576:

```
sysctl -w fs.file-max=1048576
```

Configuration examples

Add initContainers for pods of Nginx Ingress Controller and configure the kernel parameters. The following shows a sample code:

```
initContainers:
- name: setsysctl
image: busybox
securityContext:
privileged: true
command:
- sh
- -c
- |
sysctl -w net.core.somaxconn=65535
sysctl -w net.ipv4.ip_local_port_range="1024 65535"
sysctl -w net.ipv4.tcp_tw_reuse=1
sysctl -w fs.file-max=1048576
```

Optimizing the Global Configuration

In addition to optimizing the kernel parameters, you can optimize the global configuration of Nginx by using the following methods:

©2013-2022 Tencent Cloud. All rights reserved.

- Increasing the maximum number of keepalive connection requests
- Increasing the maximum number of keepalive idle connections
- Increasing the maximum number of connections for a single worker
- Configuration examples

Increasing the maximum number of keepalive connection requests

For keepalive connections between Nginx and the client or upstream server, the keepalive_requests parameter controls the maximum number of requests that can be processed by a single keepalive connection, which defaults to 100. When the number of requests for a keepalive connection exceeds the default, the connection will be disconnected and then re-established.

For Ingress in a private network, the QPS of a single client may be high (for example, 10,000 QPS), and Nginx may frequently disconnect its keepalive connections with the client, resulting in large numbers of connections in the TIME_WAIT state. To prevent this issue in a high-concurrency environment, we recommend that you increase the maximum number of requests for keepalive connections between Nginx and clients. This maximum number is determined by the `keep-alive-requests` parameter in Nginx Ingress, and you can set it to 10000. For more information, see keep-alive-requests.

The number of keepalive connection requests between Nginx and the upstream is determined by `upstream-keepalive-requests`. For more information on the configuration method, see upstream-keepalive-requests.

> Note：
> In non-high-concurrency environments, you do not need to configure this parameter. If you set it to a higher value, load imbalance may occur. This is because, when keepalive connections between Nginx and the upstream are retained too long, the number of connection scheduling times will decrease and the connections will be too "rigid", leading to a traffic load imbalance.

Increasing the maximum number of idle keepalive connections

For connections between Nginx and the upstream, you can configure the keepalive parameter, which determines the maximum number of idle connections and defaults to 320. In a high-concurrency environment, large numbers of requests and connections exist. However, in an actual production environment, requests are not fully balanced, and some connections may be temporarily idle. When the number of idle connections increases and idle connections are removed, Nginx may frequently disconnect from and reconnect to the upstream, significantly increasing the number of TIME_WAIT connections.
In a high-concurrency environment, we recommend that you set keepalive to 1000. For more information, see upstream-keepalive-connections.

©2013-2022 Tencent Cloud. All rights reserved.

Increasing the maximum number of connections for a single worker

The `max-worker-connections` parameter controls the maximum number of connections that can be used by each worker process, which defaults to 16384 in TKE. In a high-concurrency environment, we recommend that you set the value of this parameter to a greater value, for example, 65536, so that Nginx can handle more connections. For more information, see max-worker-connections.

Configuration examples

The global configuration of Nginx is implemented through the configmap configuration (Nginx Ingress Controller will read and automatically load the configuration.) The following shows a sample code:

```
apiVersion: v1
kind: ConfigMap
metadata:
name: nginx-ingress-controller
# Nginx Ingress performance optimization: https://www.nginx.com/blog/tuning-ngin
x/
data:
# The number of requests that can be processed by a persistent connection between
Nginx and the client, which defaults to 100. We recommend that you increase this
number in high-concurrency scenarios.
# Reference: https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configur
ation/configmap/#keep-alive-requests
keep-alive-requests: "10000"
# The maximum number of idle persistent connections (not the maximum number of co
nnections) between Nginx and the upstream, which defaults to 320. We recommend th
at you increase this number in high-concurrency scenarios to prevent the frequent
establishment of connections from significantly increasing the number of TIME_WAI
T connections.
# Reference: https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configur
ation/configmap/#upstream-keepalive-connections
upstream-keepalive-connections: "2000"
# The maximum number of connections that can be used by each worker process, whic
h defaults to 16384
# Reference: https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configur
ation/configmap/#max-worker-connections
max-worker-connections: "65536"
```

References

- Deploying Nginx Ingress on TKE
- ConfigMaps

©2013-2022 Tencent Cloud. All rights reserved.

- Tuning NGINX for Performance
- Module ngx_http_upstream_module

©2013-2022 Tencent Cloud. All rights reserved.

Nginx Ingress Best Practices

Last updated：2023-05-06 17:36:46

Overview

TKE supports the installation of the Nginx-ingress add-on and uses it to access Ingress traffic. For more information about Nginx-ingress, see Nginx-ingress. This document describes the best practices for the Nginx-ingress add-on.

Prerequisites

You have installed the Nginx-ingress add-on.

Directions

Opening multiple Nginx Ingress traffic entries for the cluster

After the Nginx-ingress add-on is installed, there will be an Nginx-ingress operator add-on under `kube-system`. You can use this add-on to create multiple Nginx Ingress instances. Each Nginx Ingress instance uses a different IngressClass and uses a different CLB as a traffic entry, so that different Ingresses can be bound to different traffic entries. You can create multiple Nginx Ingress instances for the cluster based on your actual needs.

1. Log in to the TKE console and select **Cluster** in the left sidebar.
2. On the **Cluster** page, click the ID of the target cluster to go to the cluster details page.
3. In the left sidebar, click **Add-on management** to go to the **Add-on list** page.
4. Click the installed Nginx-ingress add-on to go to the details page.
5. Click **Add Nginx Ingress instance** to configure the Nginx Ingress instances as needed, and specify a different IngressClass name for each instance.

Note

For information about how to install an Nginx Ingress instance, see Installing Nginx-ingress Instance.

6. When creating an Ingress, you can specify a specific IngressClass to bind the Ingress to a specific Nginx Ingress instance. You can create an Ingress via the console or YAML.

Using the console to create an Ingress

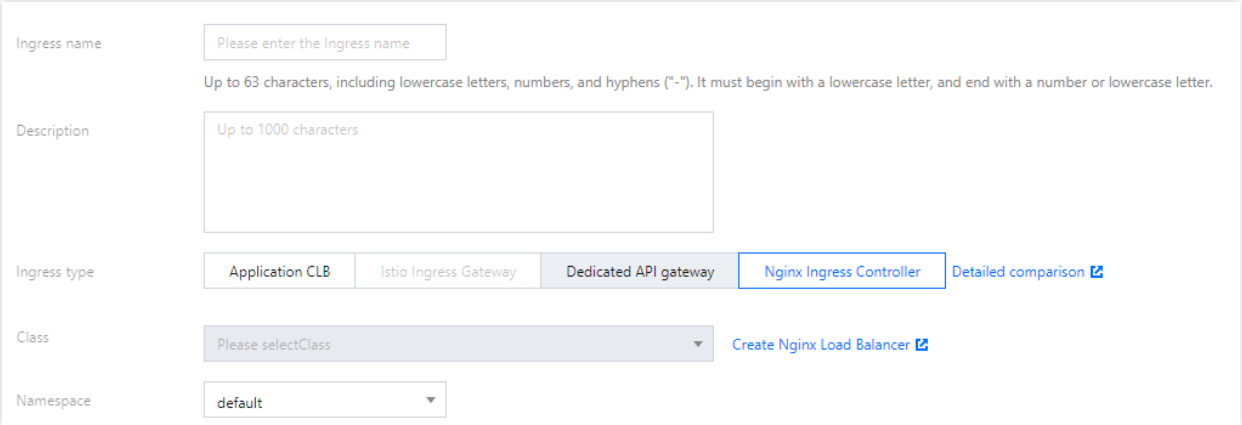
Using YAML to create an Ingress

You can refer to the Managing Ingress in Console > Creating an Ingress section to create an Ingress. Also, take note of the following points:

Ingress type: Select **Nginx Load Balancer**.

©2013-2022 Tencent Cloud. All rights reserved.

Class: Select the newly created Nginx Ingress instance.



You can refer to the Managing Ingresses Using Kubectl > Creating an Ingress section to create an Ingress. Also, specify the annotation (`kubernetes.io/ingress.class`) of ingressClass as shown below:

```
1 apiVersion: networking.k8s.io/v1beta1
2 kind: Ingress
3 metadata:
4   annotations:
5     ingress.cloud.tencent.com/direct-access: "false"
6     kubernetes.io/ingress.class: nginx-external
```

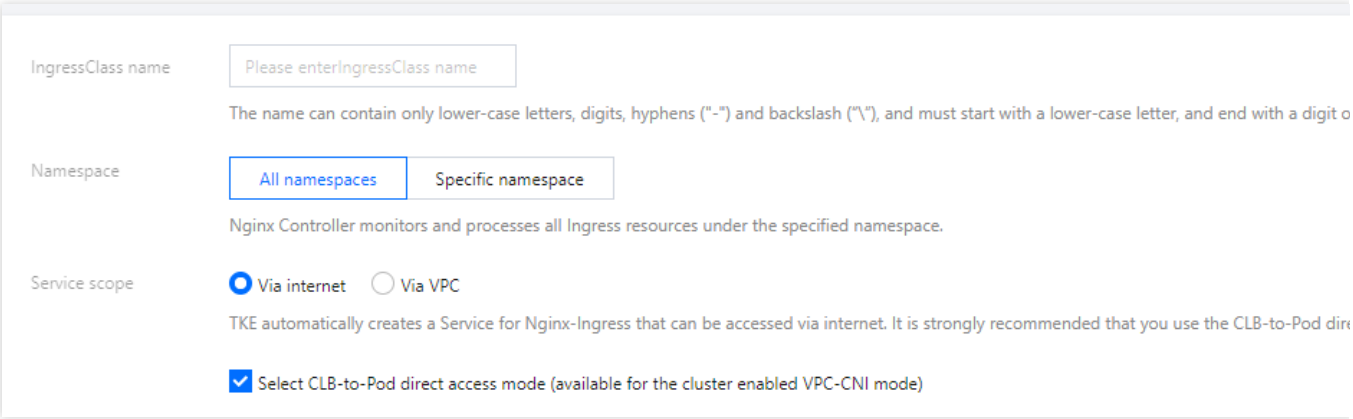
Performance optimization

CLB-to-Pod direct access mode

When the cluster network mode is Global Router, CLB-to-Pod direct access mode is not enabled by default. It is recommended to enable CLB-to-Pod direct access mode based on the following directions:

1. Enable the VPC-CNI mode for the cluster.

2. When creating an Nginx Ingress instance, you can check **Select CLB-to-Pod direct access mode** to enable traffic to bypass the NodePort and reach the Pod directly to improve performance, as shown below:



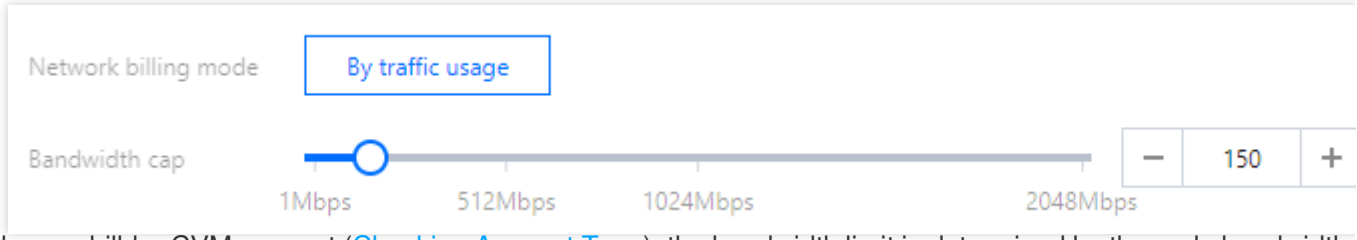
Note

For information about how to install an Nginx Ingress instance, see Installing Nginx-ingress Instance.

©2013-2022 Tencent Cloud. All rights reserved.

Adjusting the LB bandwidth limit

As the traffic entry, if LB needs a higher concurrency or throughput, you can set the bandwidth limit based on the actual needs when creating an Nginx Ingress instance and allocate a higher bandwidth for Nginx Ingress, as shown below:



If you have a bill-by-CVM account (Checking Account Type), the bandwidth limit is determined by the node bandwidth. You can adjust the node bandwidth limit based on the following conditions:

If the CLB-to-Pod direct access mode is enabled, the total LB bandwidth is the sum of the bandwidths of the nodes where the Nginx Ingress instance Pods locate. It is recommended to plan some nodes with a high public network bandwidth to deploy Nginx Ingress instances (specify a node pool as DaemonSet to deploy).

If the CLB-to-Pod direct access mode is not enabled, the total bandwidth of LB is the sum of the public network bandwidths of all nodes.

Nginx Ingress parameter optimization

The Nginx Ingress instance can optimize the kernel parameters and the configuration of Nginx Ingress by default. For more information, see Nginx Ingress High-Concurrency Practices. You can refer to the following directions for customization.

Modifying the kernel parameters

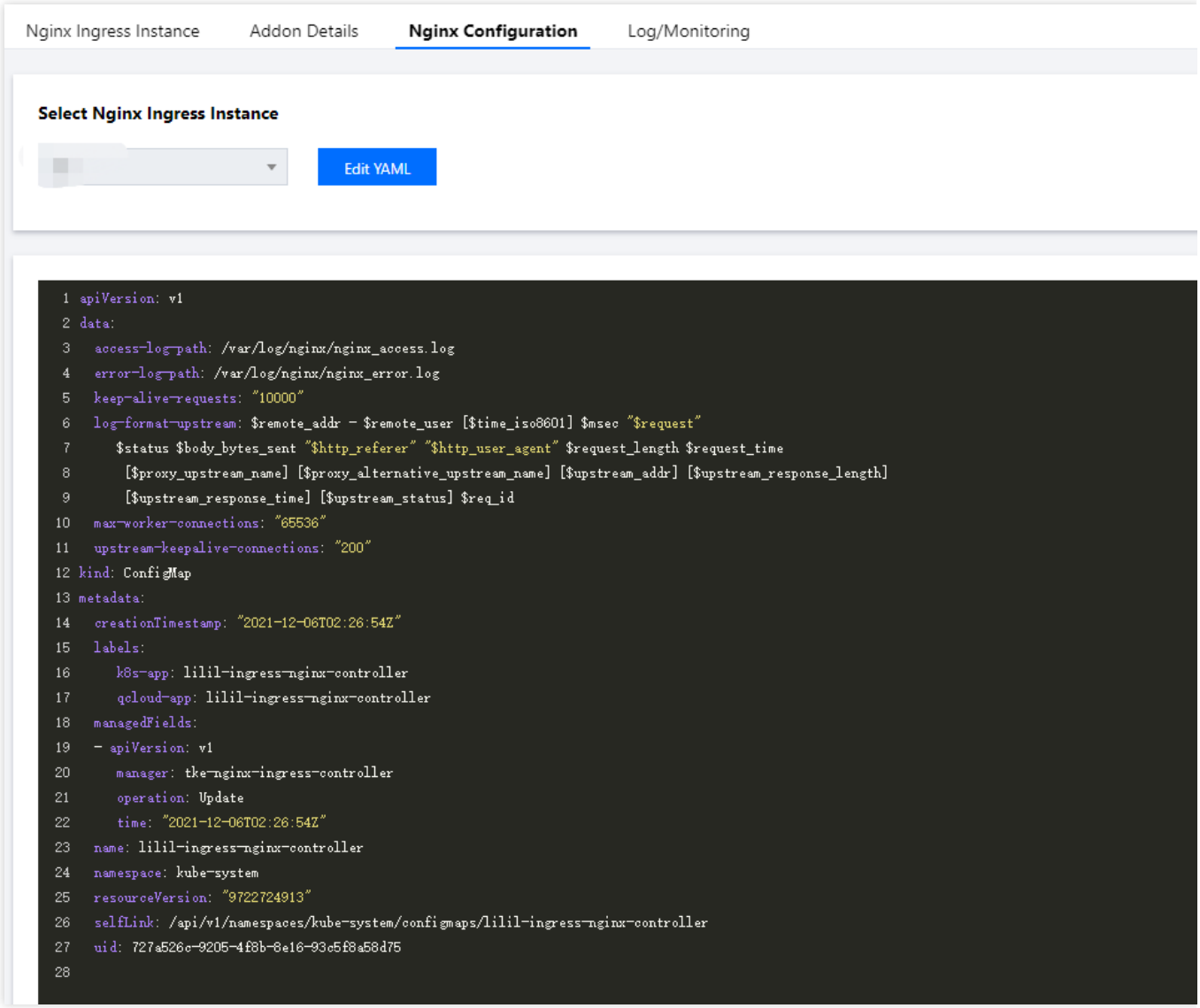
Modifying the configuration of the Nginx Ingress instance

Edit the deployed DaemonSet or Deployment of nginx-ingress-conntroller (depending on the instance deployment options) and modify initContainers as shown below. Note that you cannot modify the resources under kube-system in the console. You need to use kubectl to modify initContainers.

```
initContainers:
- command:
  - sh
  - -c
  - |-
    sysctl -w net.core.somaxconn=65535
    sysctl -w net.ipv4.ip_local_port_range="1024 65535"
    sysctl -w net.ipv4.tcp_tw_reuse=1
    sysctl -w fs.file-max=1048576
```

In the **Nginx Configuration** section, select the Nginx Ingress instance and click **Edit YAML** to modify the ConfigMap configuration of the instance, as shown below:

©2013-2022 Tencent Cloud. All rights reserved.




```
apiVersion: v1
data:
  access-log-path: /var/log/nginx/nginx_access.log
  error-log-path: /var/log/nginx/nginx_error.log
  keep-alive-requests: "10000"
  log-format-upstream: $remote_addr - $remote_user [$time_iso8601] $msec "$request"
    $status $body_bytes_sent "$http_referer" "$http_user_agent" $request_length $request_time
     [$proxy_upstream_name] [$proxy_alternative_upstream_name] [$upstream_addr] [$upstream_response_length]
     [$upstream_response_time] [$upstream_status] $req_id
  max-worker-connections: "65536"
  upstream-keepalive-connections: "200"
kind: ConfigMap
metadata:
  creationTimestamp: "2021-12-06T02:26:54Z"
  labels:
    k8s-app: lilil-ingress-nginx-controller
    qcloud-app: lilil-ingress-nginx-controller
  managedFields:
  - apiVersion: v1
    manager: tke-nginx-ingress-controller
    operation: Update
    time: "2021-12-06T02:26:54Z"
  name: lilil-ingress-nginx-controller
  namespace: kube-system
  resourceVersion: "9722724913"
  selfLink: /api/v1/namespaces/kube-system/configmaps/lilil-ingress-nginx-controller
  uid: 727a526c-9205-4f8b-8e16-93c5f8a58d75

```

Note

For more information about ConfigMap configuration, see Official Document.

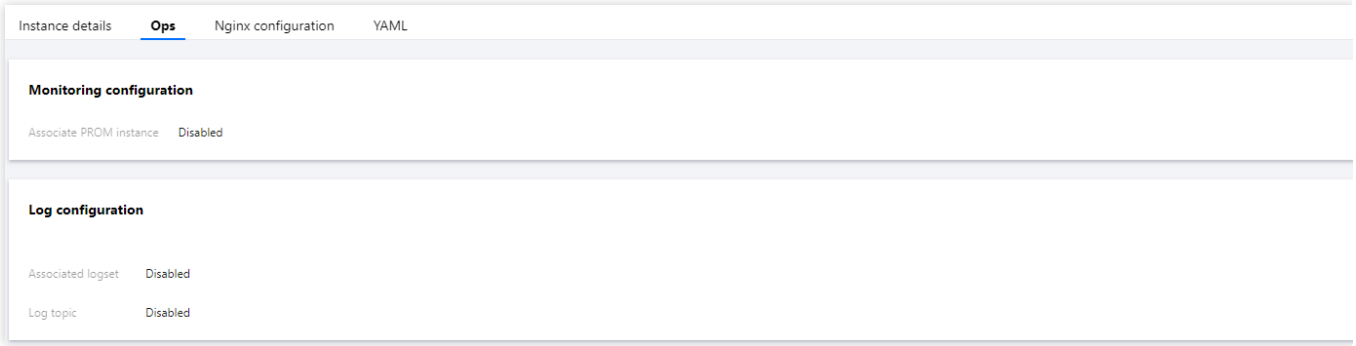
Improving the observability of Nginx Ingress

Enable logging

Note:

The log configuration relies on Cloud Log Service (CLS). For more information, see Nginx-ingress Log Configuration. The logging feature allows you to view the status metrics of an instance and helps you with troubleshooting. After you create an Nginx Ingress instance, go to its details page and enable the logging feature for the instance in the operations section, as shown below:

©2013-2022 Tencent Cloud. All rights reserved.



Note:

For v0.49.3 instances, the indexing configuration file for log collection is located in a custom resource definition (CRD) object named LogConfig. If you disable or re-enable the logging feature after modifying LogConfig, the configuration of LogConfig is reset. Therefore, you must back up the data in the object in a timely manner. The deletion of the Nginx Ingress instance and the upgrade of the Nginx-ingress add-on do not affect the indexing configuration file.

If you need to customize the logging feature, see here for reference.

Log search and log dashboard

After enabling the log configuration, you can click **More** under **Operation** on the right side of an instance in the Nginx Ingress list, and select **Check access logs in CLS** or **View Access Log Dashboard**.

Click **Check access logs in CLS** to go to the CLS console and select the logset and topic corresponding to the instance in **Search and Analyze** to view the access and error logs of Nginx Ingress.

Click **View Access Log Dashboard** to go to the dashboard that displays statistics based on the Nginx Ingress log data.

©2013-2022 Tencent Cloud. All rights reserved.

Self-Built Nginx Ingress Practice Tutorial
Quick Start

Last updated：2024-08-12 17:48:23

Overview

Nginx Ingress Controller is a Kubernetes Ingress controller based on the high-performance NGINX reverse proxy, and it is also one of the most commonly used open-source Ingress implementations. This document explains how to self-build an Nginx Ingress Controller in the TKE environment, mainly using helm for installation and providing some `values.yaml` configuration guidance.

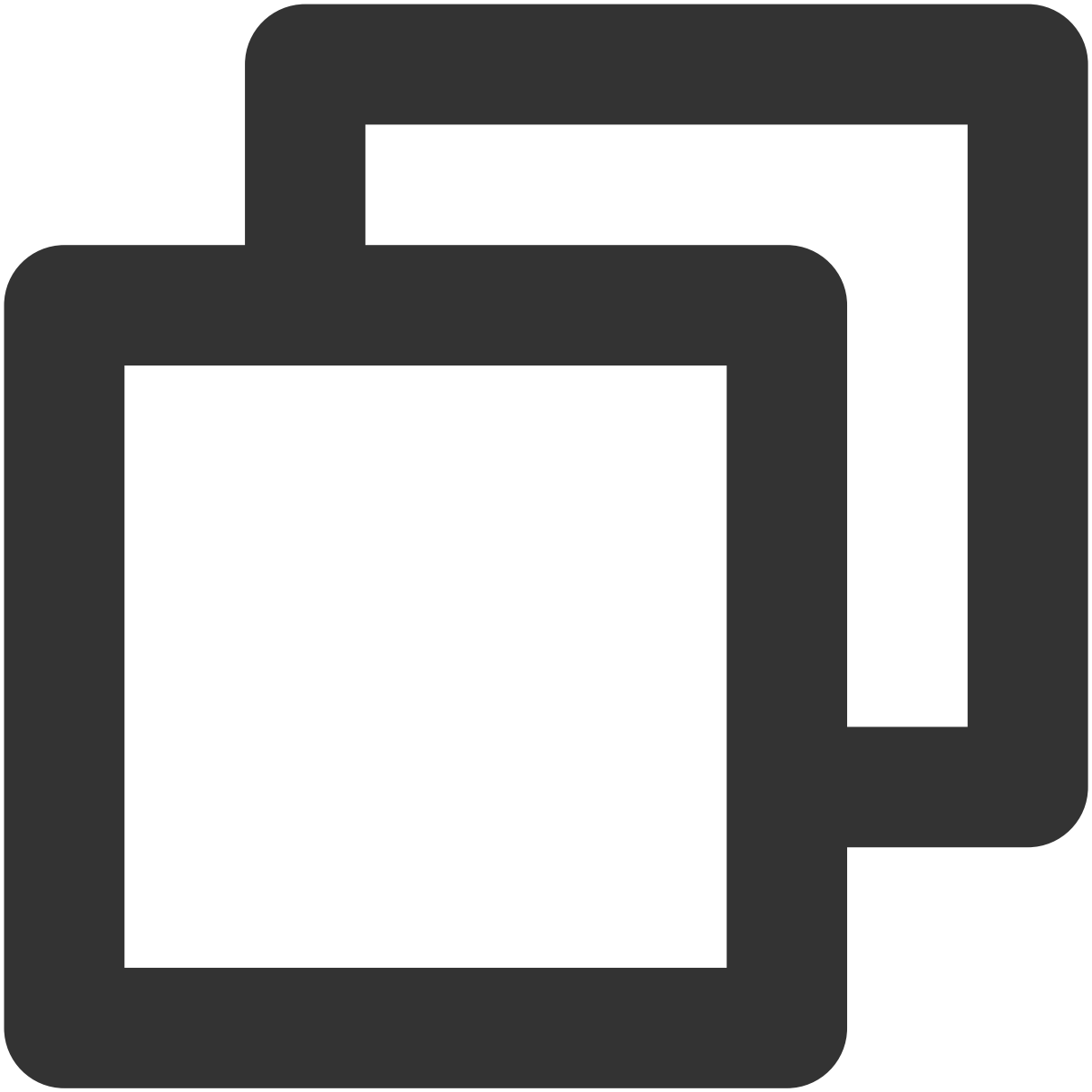
Prerequisites

A TKE cluster is created.
The helm is installed.
The TKE cluster's kubeconfig is configured, with the permissions to operate the TKE cluster. For more details, see connect to the cluster.

Installation with helm

Add helm repo:

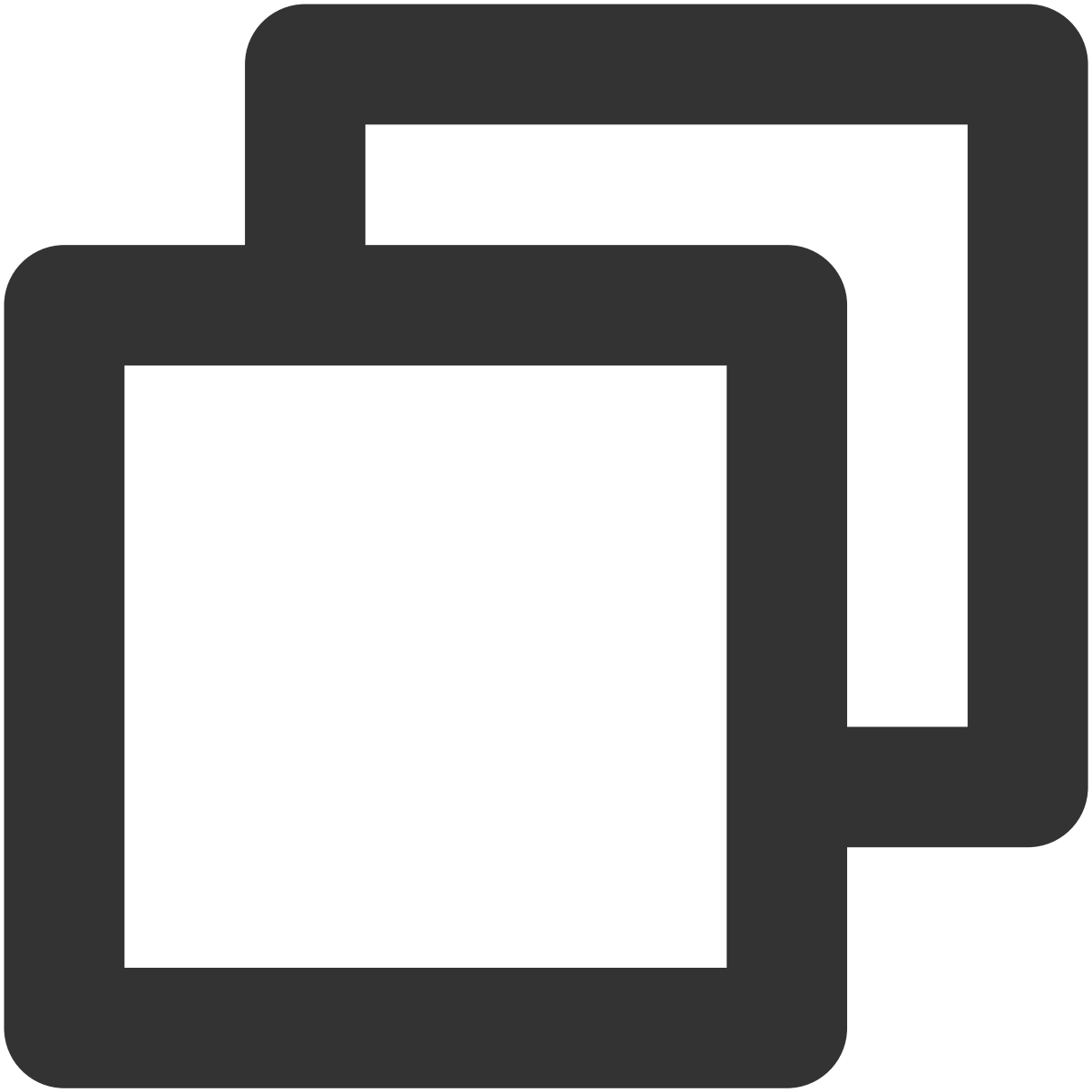
©2013-2022 Tencent Cloud. All rights reserved.



```
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
```

View default configuration:

©2013-2022 Tencent Cloud. All rights reserved.

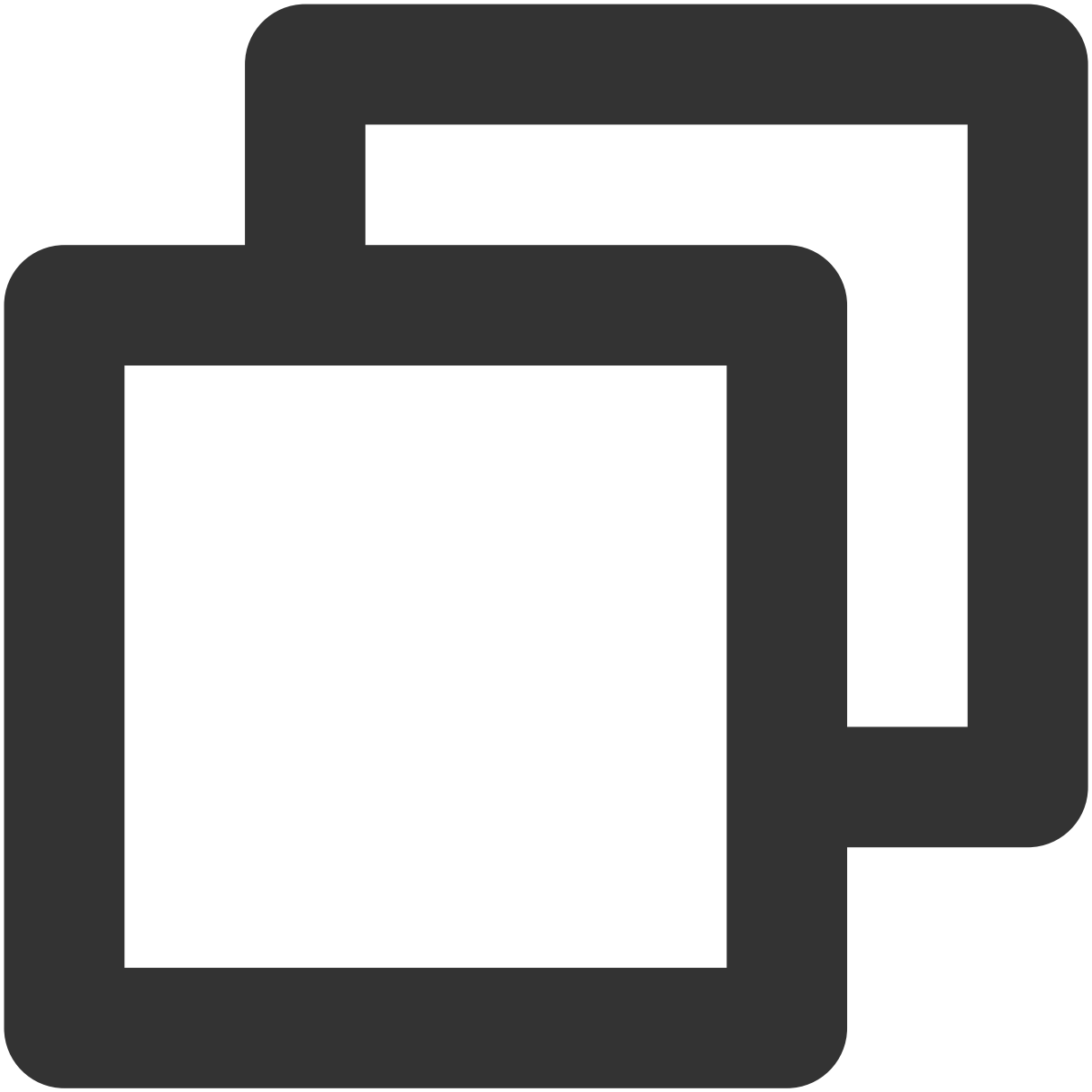


```
helm show values ingress-nginx/ingress-nginx
```

The Nginx Ingress dependent image is under the `registry.k8s.io` registry. In the network environment in Chinese mainland, it cannot be pulled. You can replace it with the mirror image in docker hub.

Prepare `values.yaml` :

©2013-2022 Tencent Cloud. All rights reserved.



```
controller: # The following configuration replaces the dependent image with the mir
  image:
    registry: docker.io
    image: k8smirror/ingress-nginx-controller
  admissionWebhooks:
    patch:
      image:
        registry: docker.io
        image: k8smirror/ingress-nginx-kube-webhook-certgen
  defaultBackend:
    image:
```

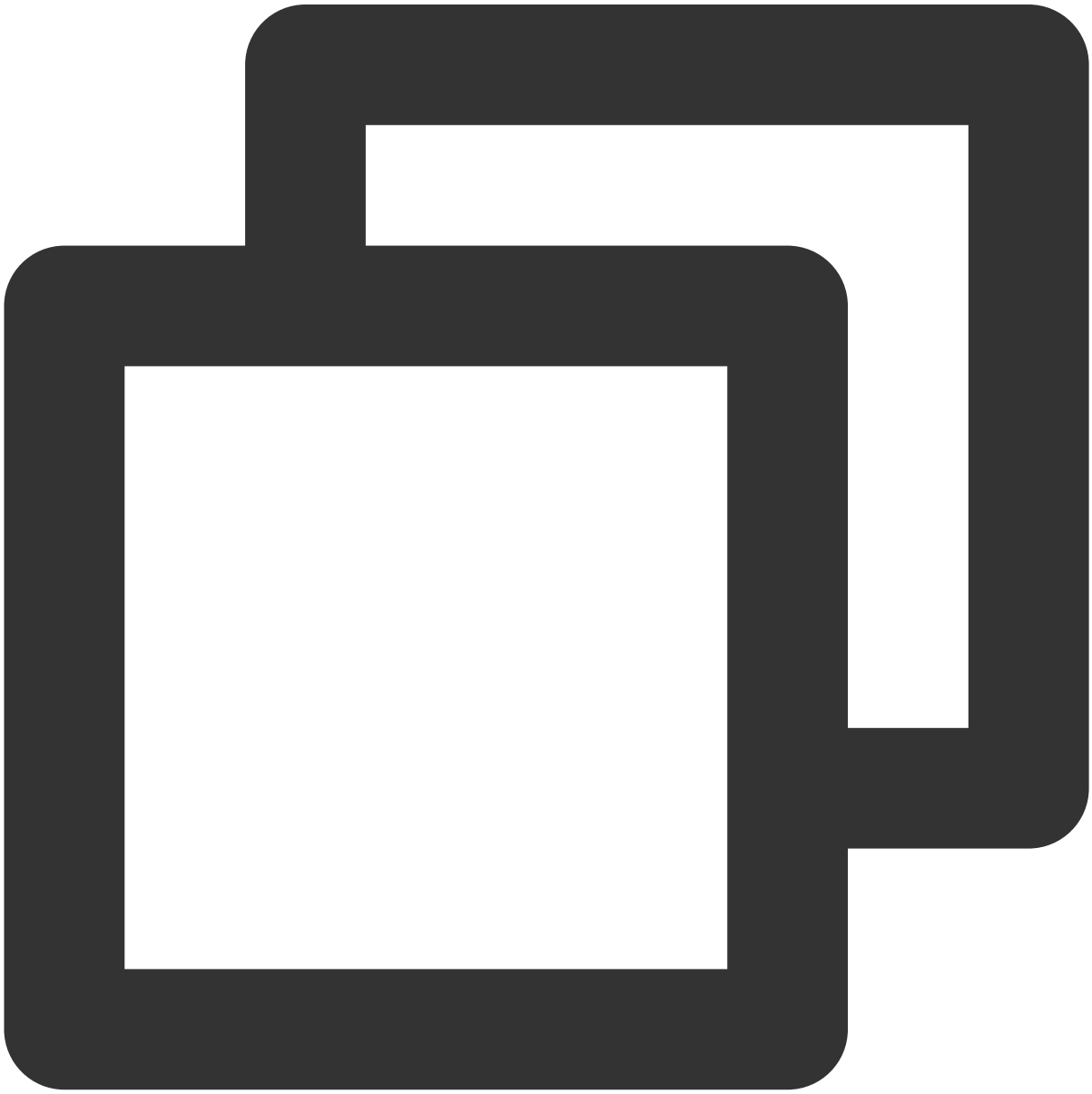
©2013-2022 Tencent Cloud. All rights reserved.

```
      registry: docker.io
      image: k8smirror/defaultbackend-amd64
  opentelemetry:
    image:
      registry: docker.io
      image: k8smirror/ingress-nginx-opentelemetry
```

Note:

All mirror images in the configuration use image-porter for long-term automatic synchronization, so you can perform installation and upgrade with confidence.

Installation:



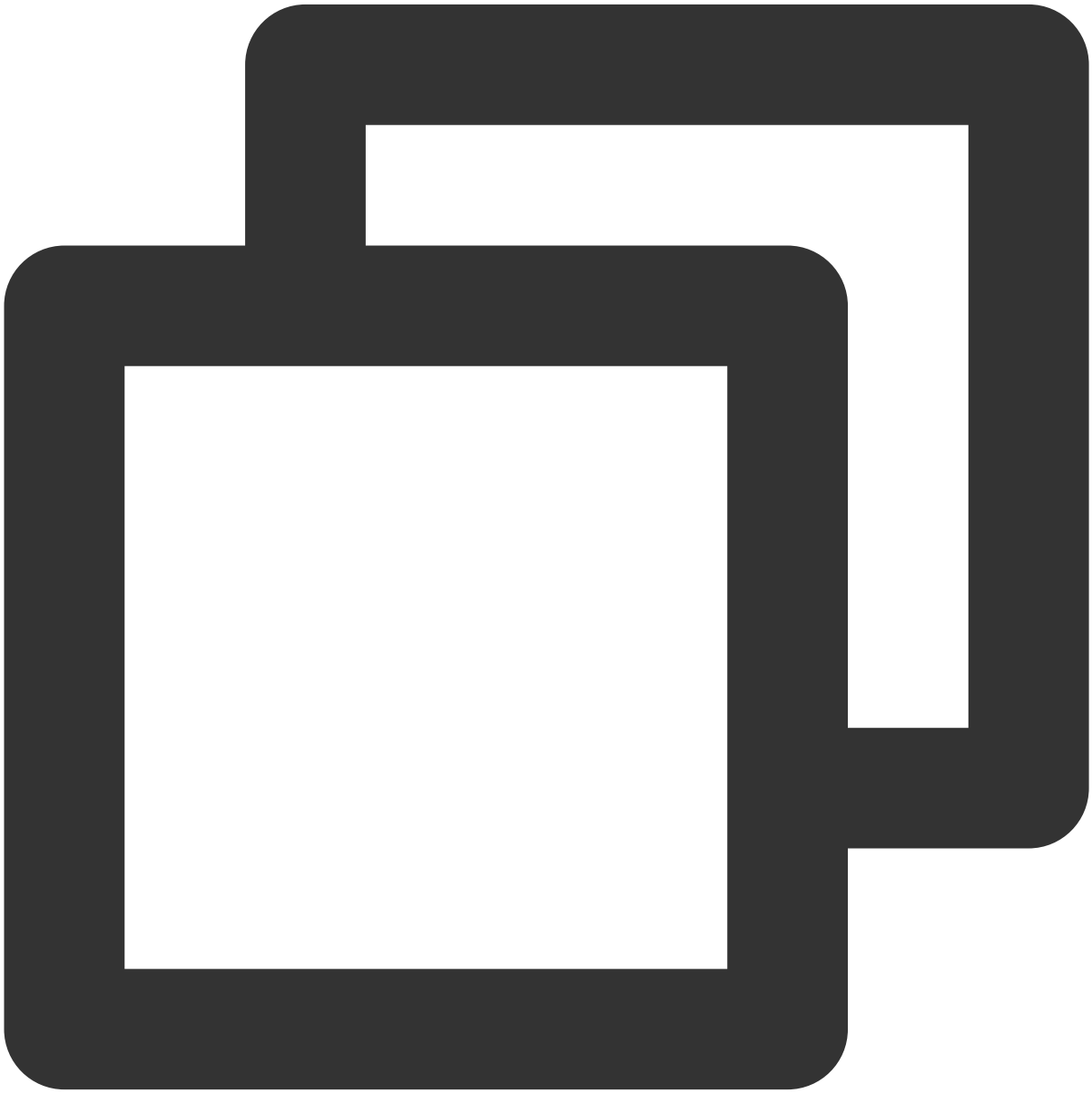
©2013-2022 Tencent Cloud. All rights reserved.

```
helm upgrade --install ingress-nginx ingress-nginx/ingress-nginx \\
  --namespace ingress-nginx --create-namespace \\
  -f values.yaml
```

Note:

If you need to modify the values configuration or upgrade the version in the future, you can update Nginx Ingress Controller by running this command.

Check the traffic entry (CLB VIP or domain name):



```
$ kubectl get services -n ingress-nginx
```

©2013-2022 Tencent Cloud. All rights reserved.

```
NAME                                 TYPE           CLUSTER-IP       EXTERNAL-IP
ingress-nginx-controller             LoadBalancer   xxx.xx.xxx.xxx   xxx.xx.xx.xxx
ingress-nginx-controller-admission   ClusterIP      xxx.xx.xxx.xxx   <none>
```

Note:

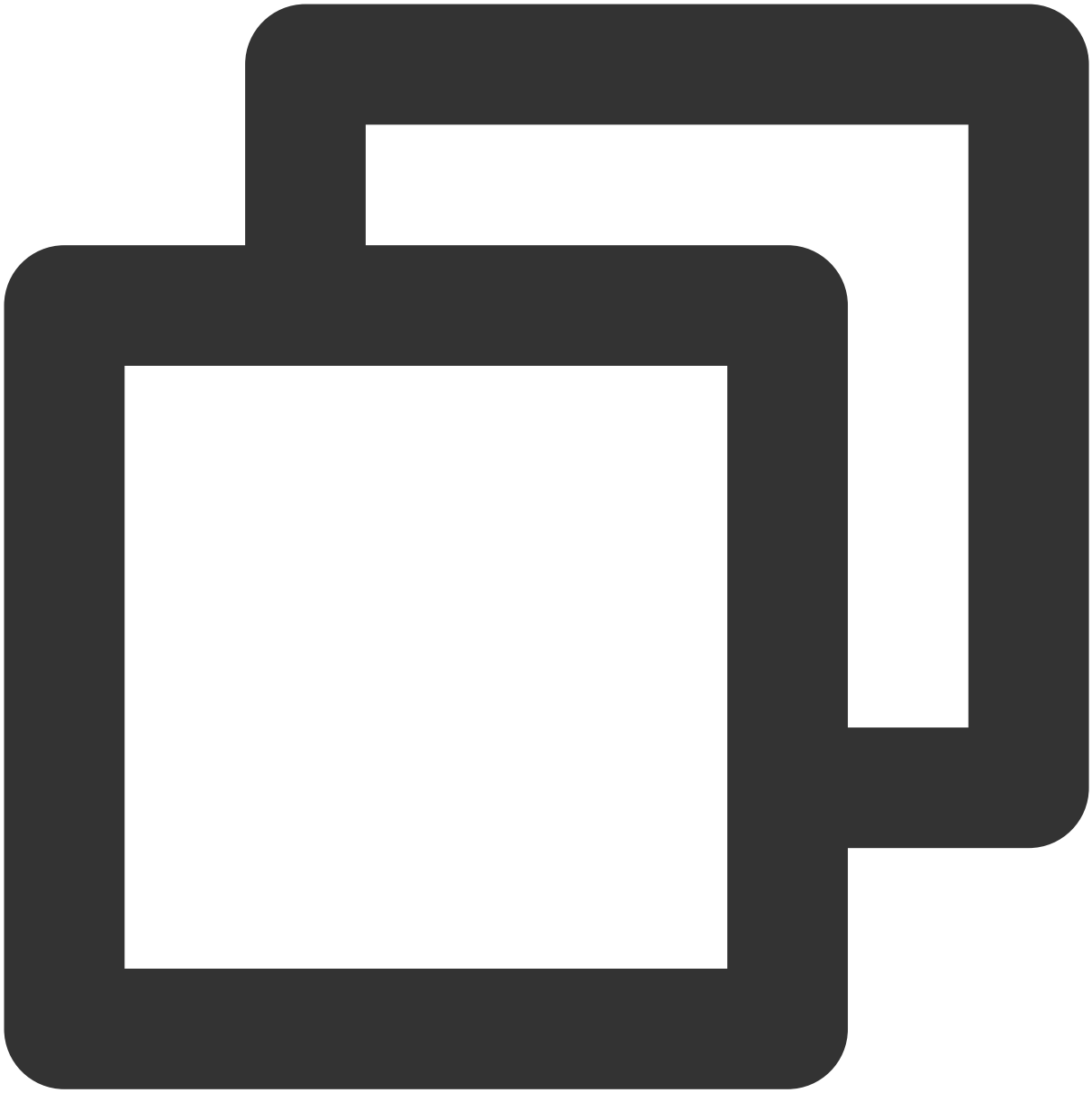
`EXTERNAL-IP` of a `LoadBalancer` service is the VIP or domain name of the CLB. You can configure DNS resolution for it. If it is a VIP, configure an A record; if it is a CLB domain name, configure a CNAME record.

Versions and Upgrades

The version of Nginx Ingress needs to be compatible with the Kubernetes cluster version. Refer to the official Supported Versions table to confirm if the current cluster version supports the latest Nginx Ingress. If not, you need to specify the chart version during installation.

For example, if the current TKE cluster version is 1.24, the latest chart version can only be `4.7.*` . Run the following commands to check available versions:

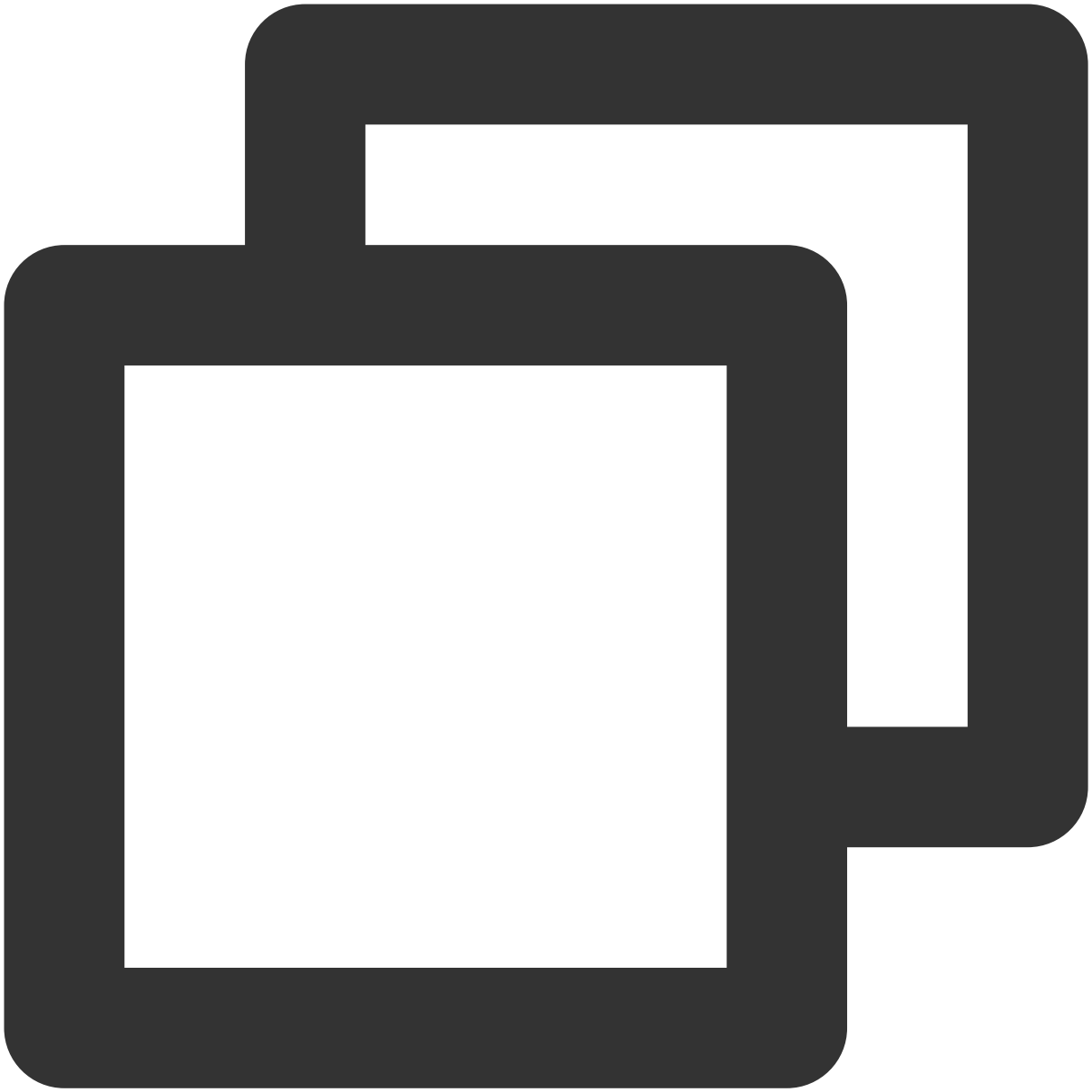
©2013-2022 Tencent Cloud. All rights reserved.



```
$ helm search repo ingress-nginx/ingress-nginx --versions | grep 4.7.
ingress-nginx/ingress-nginx     4.7.5           1.8.5           Ingress controller
ingress-nginx/ingress-nginx     4.7.3           1.8.4           Ingress controller
ingress-nginx/ingress-nginx     4.7.2           1.8.2           Ingress controller
ingress-nginx/ingress-nginx     4.7.1           1.8.1           Ingress controller
ingress-nginx/ingress-nginx     4.7.0           1.8.0           Ingress controller
```

You can see that the latest `4.7.*` version is `4.7.5` . Add the version number during installation:

©2013-2022 Tencent Cloud. All rights reserved.



```
helm upgrade --install ingress-nginx ingress-nginx/ingress-nginx \\
  --version 4.7.5 \\
  --namespace ingress-nginx --create-namespace \\
  -f values.yaml
```

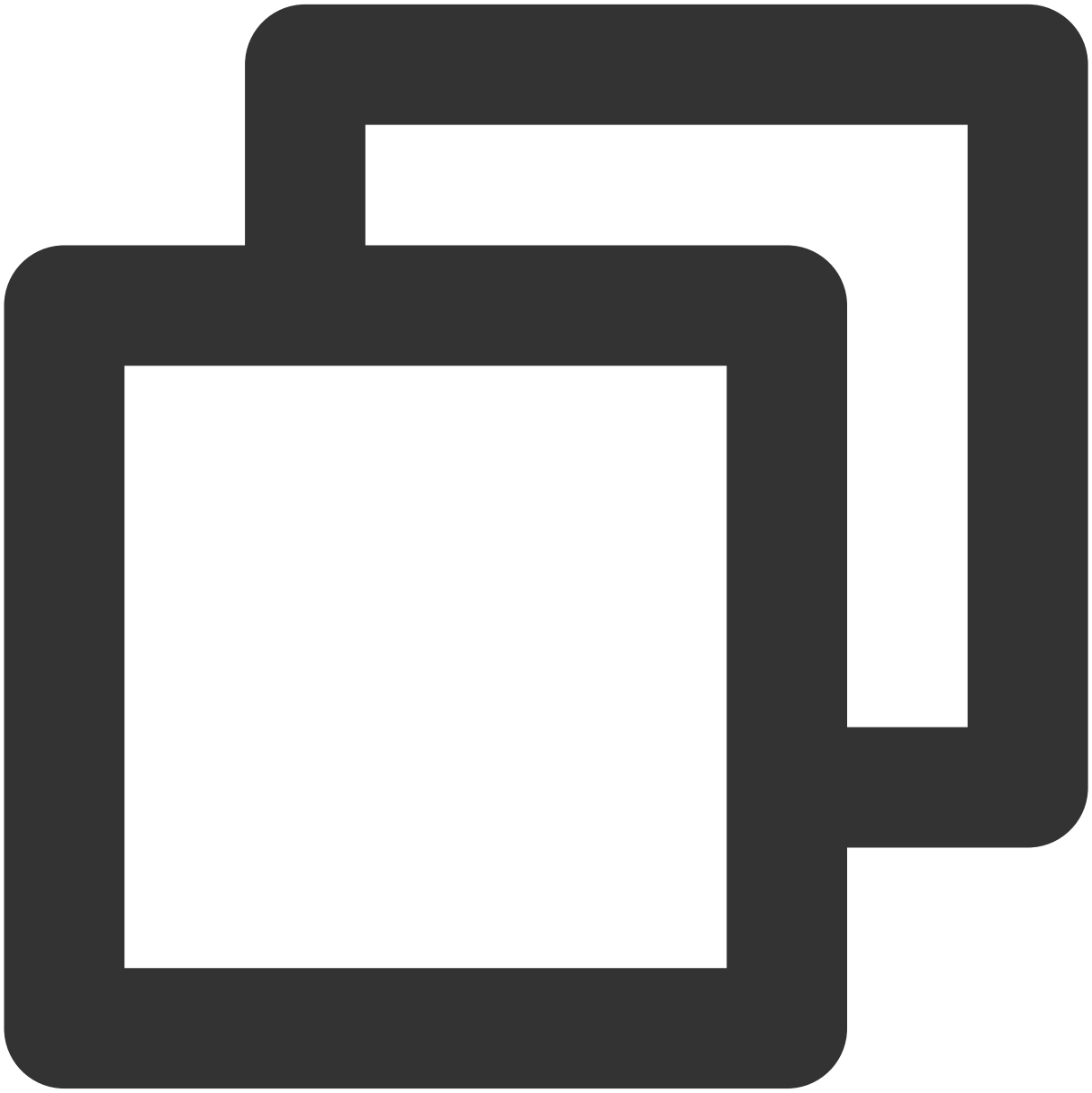
Note:

Before upgrading the TKE cluster, check if the current Nginx Ingress version is compatible with the upgraded cluster version. If not, upgrade Nginx Ingress first (run the preceding command to specify the chart version).

©2013-2022 Tencent Cloud. All rights reserved.

Using Ingress

Nginx Ingress implements the standard capabilities of Kubernetes's Ingress API definition. For basic usage of Ingress, refer to the Kubernetes official documentation.
You must specify `ingressClassName` as IngressClass (`nginx` by default) used by the Nginx Ingress instance:



```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
```

©2013-2022 Tencent Cloud. All rights reserved.

```
    name: nginx
  spec:
    ingressClassName: nginx
    rules:
      - http:
          paths:
            - path: /
              pathType: Prefix
              backend:
                service:
                  name: nginx
                  port:
                    number: 80
```

Additionally, Nginx Ingress has many other unique features. For details on extending the features of Ingress through Ingress annotations, refer to Nginx Ingress Annotations.

More Customization

If you need more customization for Nginx Ingress, refer to the following document and merge the `values.yaml` configuration as needed. The Complete Example of values.yaml Configuration provides a complete example of `values.yaml` configuration after merging.

Custom Load Balancer

Enabling CLB Direct Connection

Optimization for High Concurrency Scenarios

High Availability Configuration Optimization

Observability Integration

Access to Tencent Cloud WAF

Installing Multiple Nginx Ingress Controllers

Migrating from TKE Nginx Ingress Plugin to Self-Built Nginx Ingress

Complete Example of values.yaml Configuration

©2013-2022 Tencent Cloud. All rights reserved.

Custom Load Balancer

Last updated：2024-08-12 17:48:23

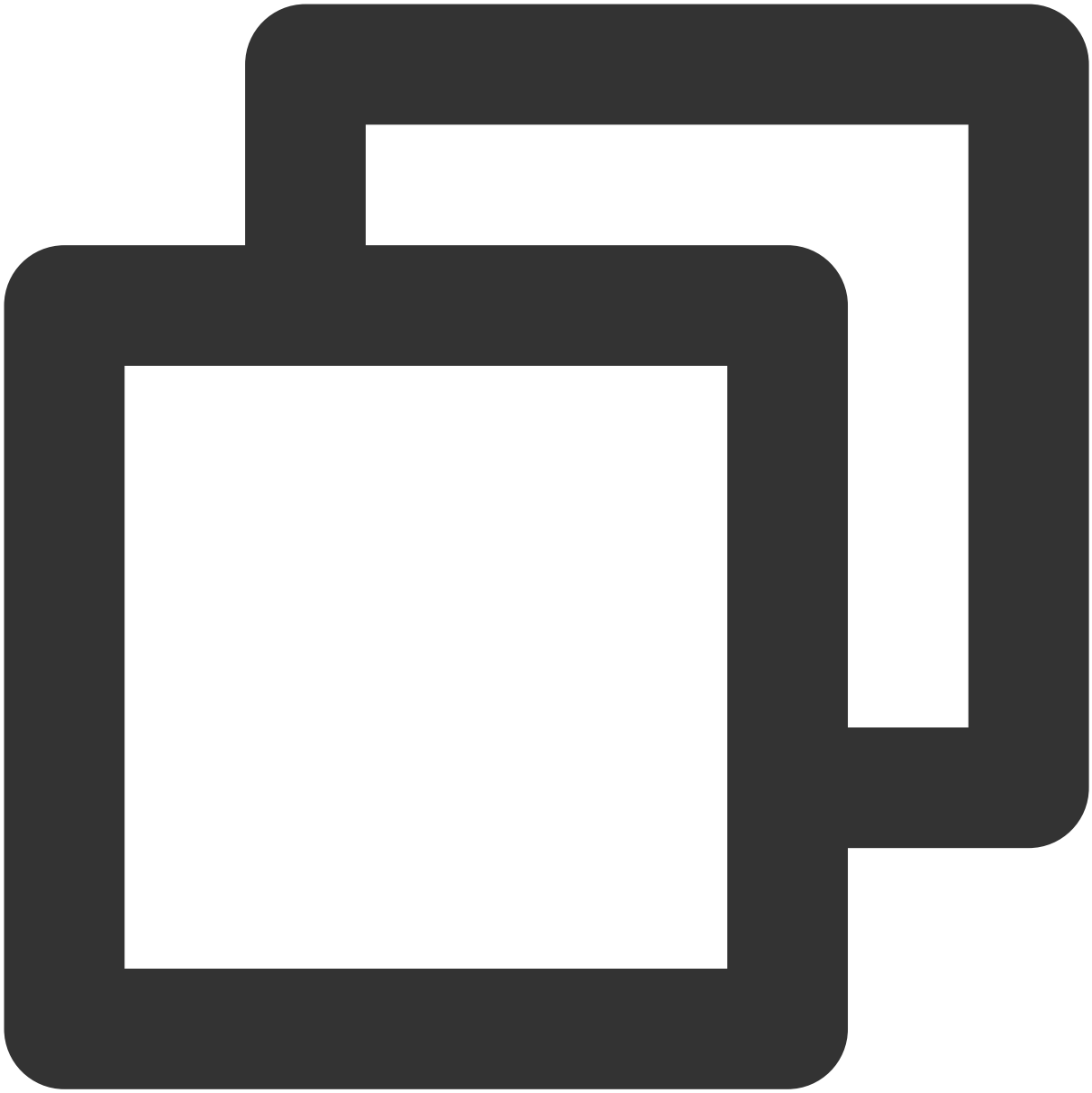
Overview

By default, a public CLB is automatically created during installation to handle traffic, but you can also use the TKE service annotation to customize the CLB for Nginx Ingress Controller. This document introduces the method for customization.

Using Private Network CLB

For example, if you want to change the CLB to a private network CLB, the sample code in `values.yaml` is as follows:

©2013-2022 Tencent Cloud. All rights reserved.

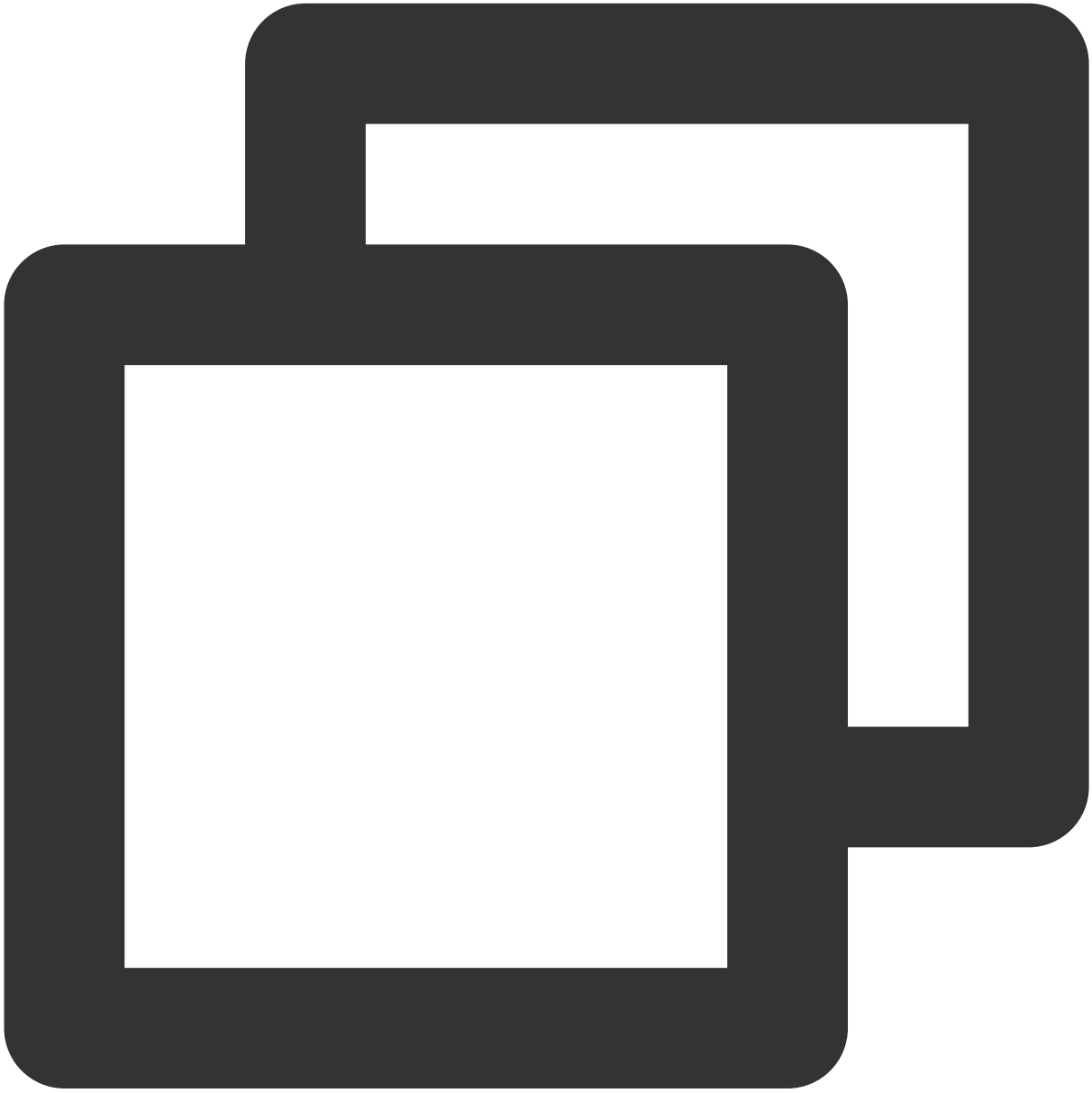


```
controller:
  service:
    annotations:
      service.kubernetes.io/qcloud-loadbalancer-internal-subnetid: 'subnet-xxxxxx'
```

Using Existing CLBs

©2013-2022 Tencent Cloud. All rights reserved.

You can also directly create a CLB in the CLB console according to your own needs (such as the custom instance specifications, ISP type, billing mode, and bandwidth limit), and then reuse this CLB with annotations in `values.yaml`. For details, refer to Using Existing CLBs.



```
controller:
  service:
    annotations:
      service.kubernetes.io/tke-existed-lbid: 'lb-xxxxxxxx' # Instance ID of the ex
```

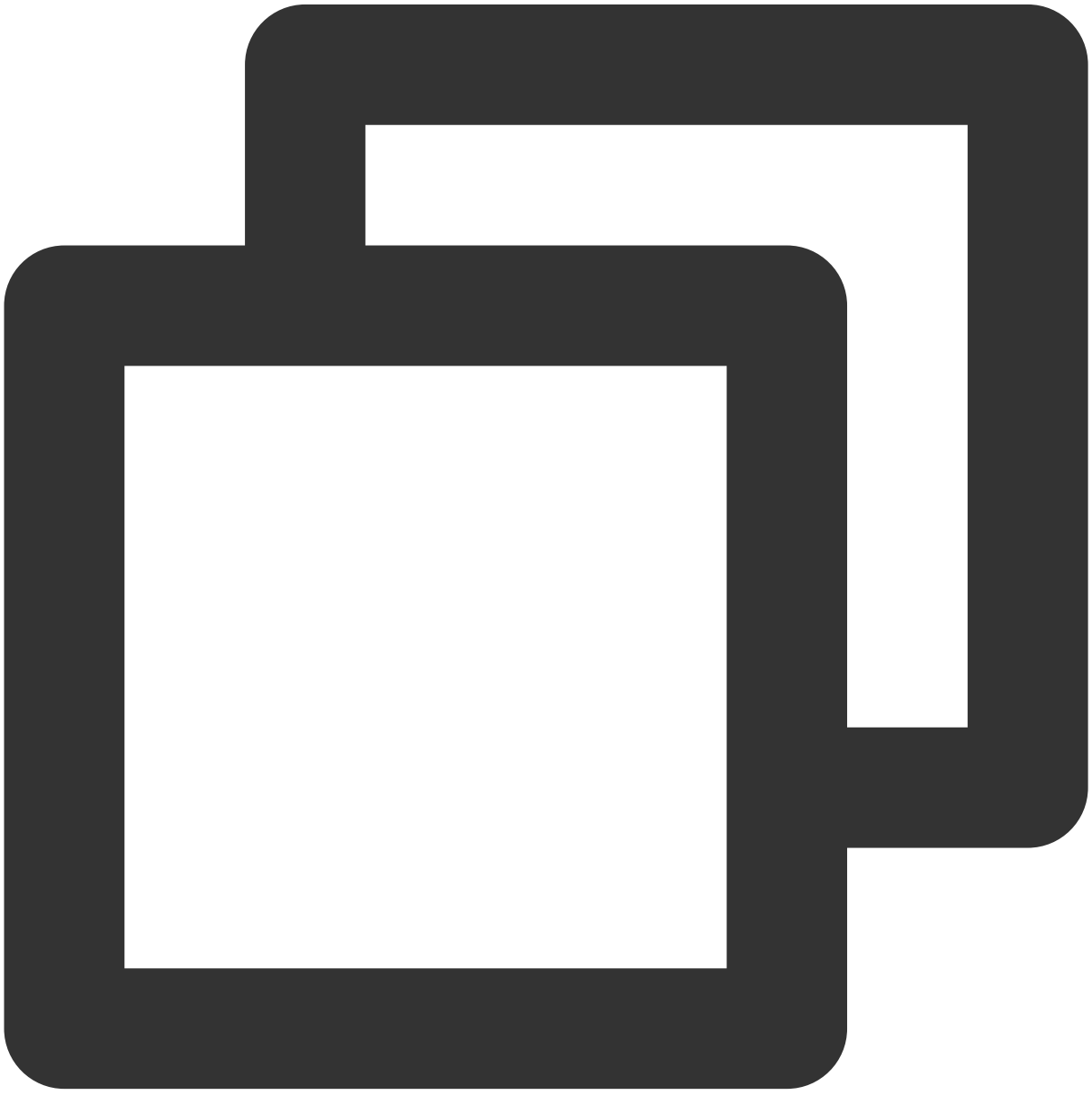
Note:

©2013-2022 Tencent Cloud. All rights reserved.

When a CLB instance is created in the CLB console, the selected VPC network must be consistent with that of the cluster.

Using Private and Public Network CLBs

If you need Nginx Ingress to use both public and private network CLBs to handle traffic simultaneously, you can configure Nginx Ingress to use two services. By default, a public network CLB service will be created. If you also need a private network CLB service, you can configure the internal service by following these steps:



©2013-2022 Tencent Cloud. All rights reserved.

```
controller:
  service:
    internal:
      enabled: true # Create a private network CLB service
      annotations:
        service.kubernetes.io/qcloud-loadbalancer-internal-subnetid: "subnet-xxxxxx
```

©2013-2022 Tencent Cloud. All rights reserved.

Enabling CLB Direct Connection

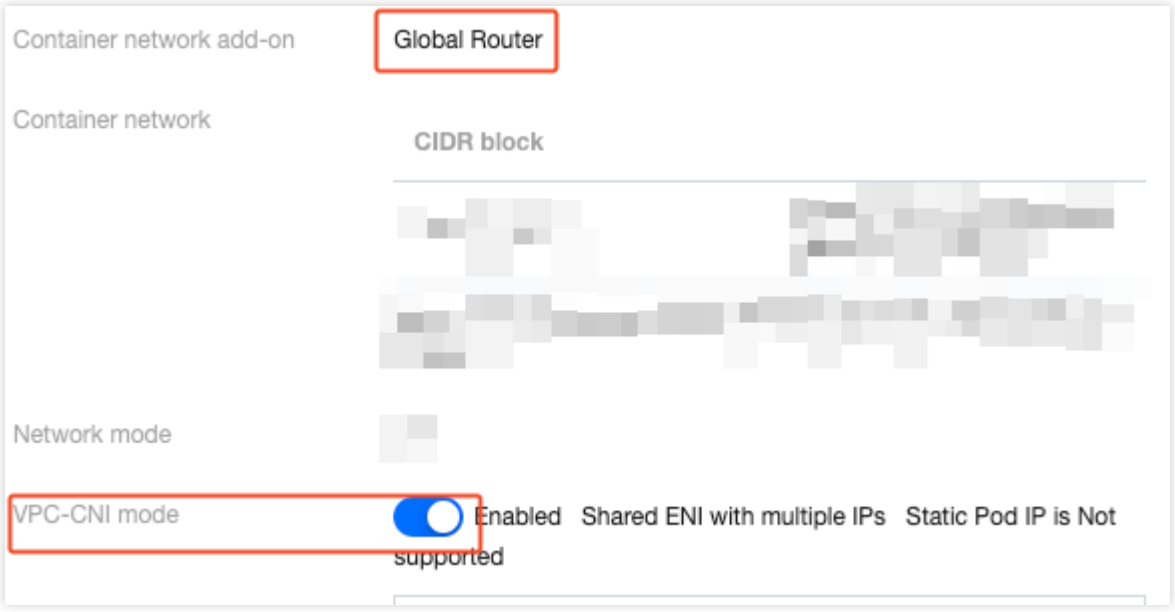
Last updated：2024-08-12 17:48:23

Overview

The traffic forwarded from CLB to Nginx Ingress can be directly connected, bypassing the NodePort communication. This method offers better performance and allows obtaining the real source IP address.
If you are using a TKE serverless cluster, or you can ensure that all Nginx Ingress Pods are scheduled on the super node, then this link is already directly connected and requires no additional action.
In other cases, the link will default to NodePort communication. If you wish to enable direct connection, you can refer to the following steps (choose steps applicable to your cluster environment).
Note:
For details, see Using Services with LoadBalancer-to-Pod Direct Access Mode.

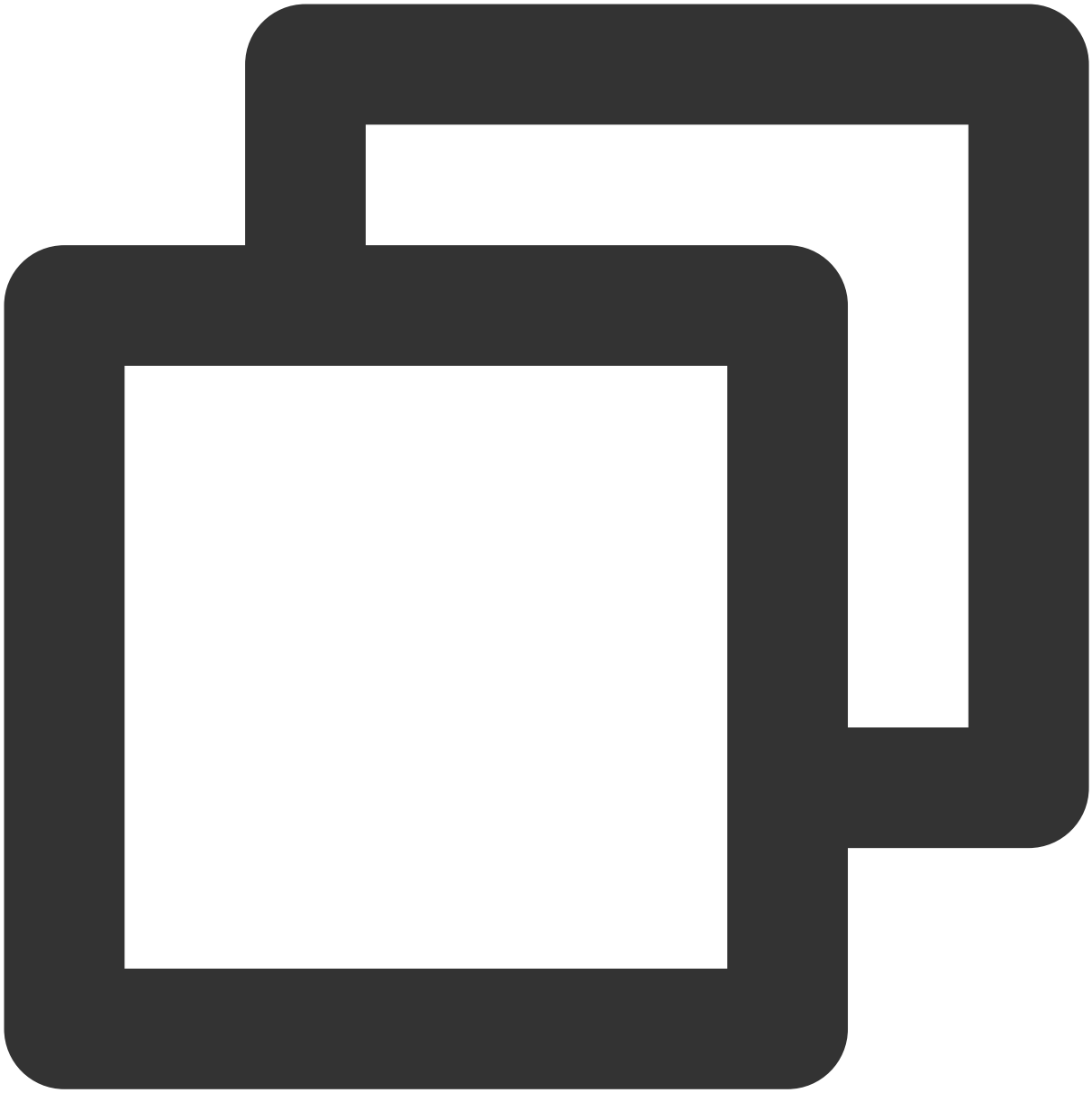
Enabling Direct Connection in GlobalRouter+VPC-CNI Network Mode

If the cluster network mode is GlobalRouter and VPC-CNI is enabled:



It is recommended to declare the use of the VPC-CNI network for Nginx Ingress and enable CLB direct connection.
`values.yaml` configuration method:

©2013-2022 Tencent Cloud. All rights reserved.



```
controller:
  podAnnotations:
    tke.cloud.tencent.com/networks: tke-route-eni # Declare the use of VPC-CNI netw
  resources: # Declare the use of ENI in resources
    requests:
      tke.cloud.tencent.com/eni-ip: "1"
    limits:
      tke.cloud.tencent.com/eni-ip: "1"
  service:
    annotations:
      service.cloud.tencent.com/direct-access: "true" # Enable CLB direct access
```

©2013-2022 Tencent Cloud. All rights reserved.

Enabling Direct Connection in GlobalRouter Network Mode

If the cluster network is GlobalRouter but VPC-CNI is not enabled, it is recommended to enable VPC-CNI for the cluster. For details, see GlobalRouter + VPC-CNI Network Mode Enable Direct Connection to enable CLB direct connection.

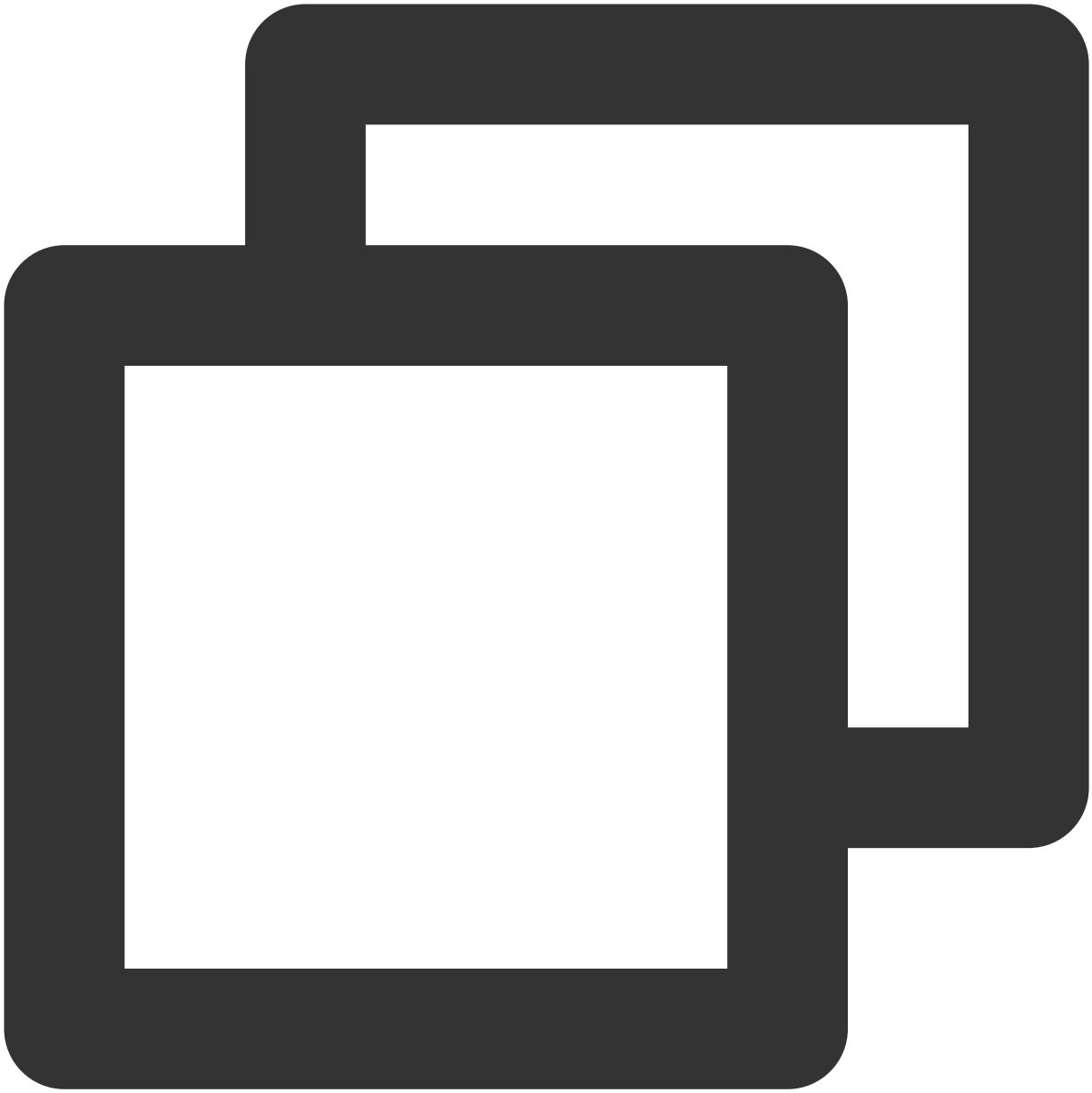
If you do not wish to enable VPC-CNI, you can enable direct connection according to the steps below but must accept the use limit.

Note:

Confirm that your account meets the above conditions and accepts the use limit.

1. Modify the configmap to enable the direct connection capability in GlobalRouter cluster dimensions:

©2013-2022 Tencent Cloud. All rights reserved.



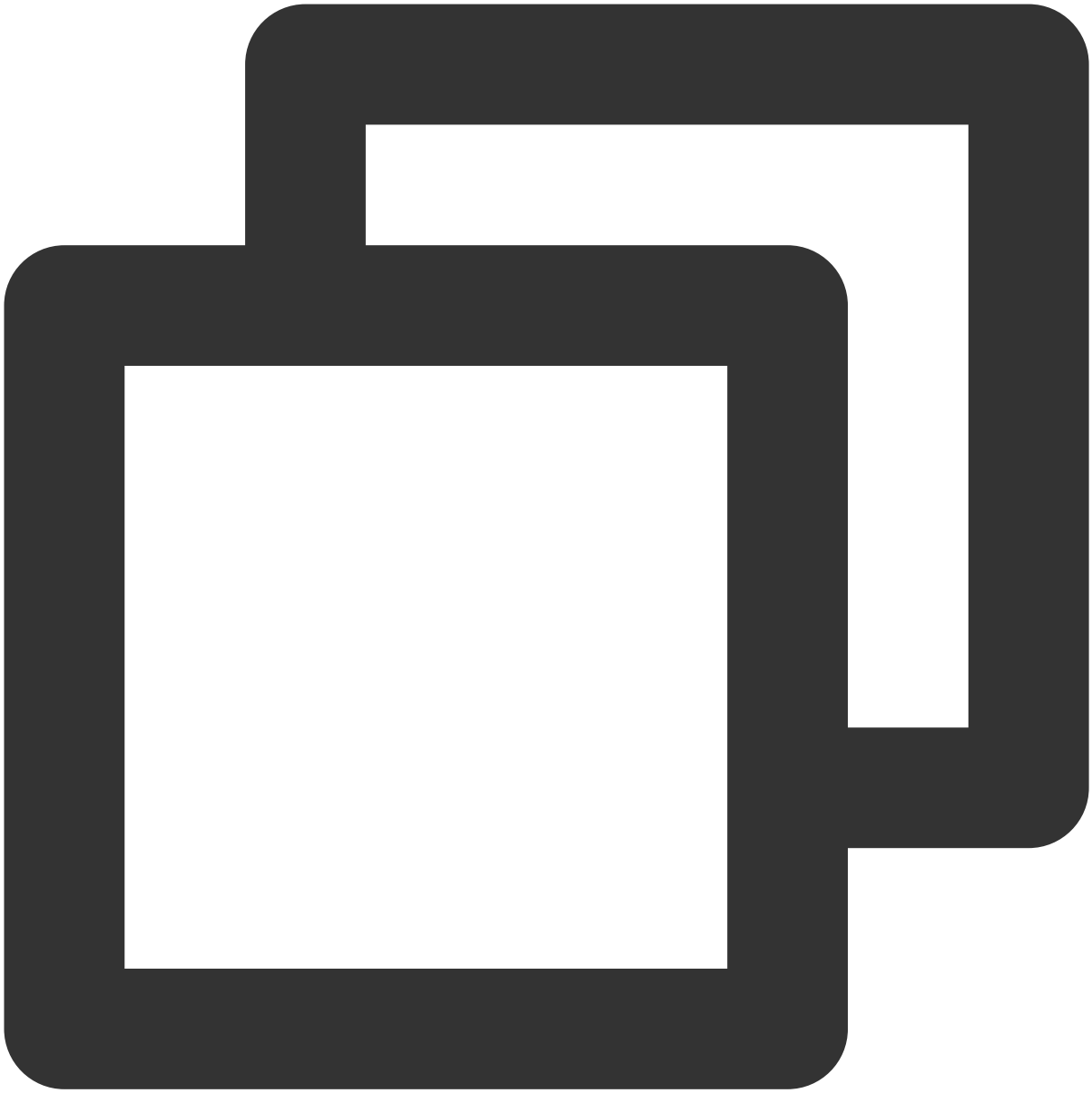
```
kubectl edit configmap tke-service-controller-config -n kube-system
```

Set `GlobalRouteDirectAccess` to true:

©2013-2022 Tencent Cloud. All rights reserved.

2. Configure `values.yaml` to enable CLB direct connection:

©2013-2022 Tencent Cloud. All rights reserved.

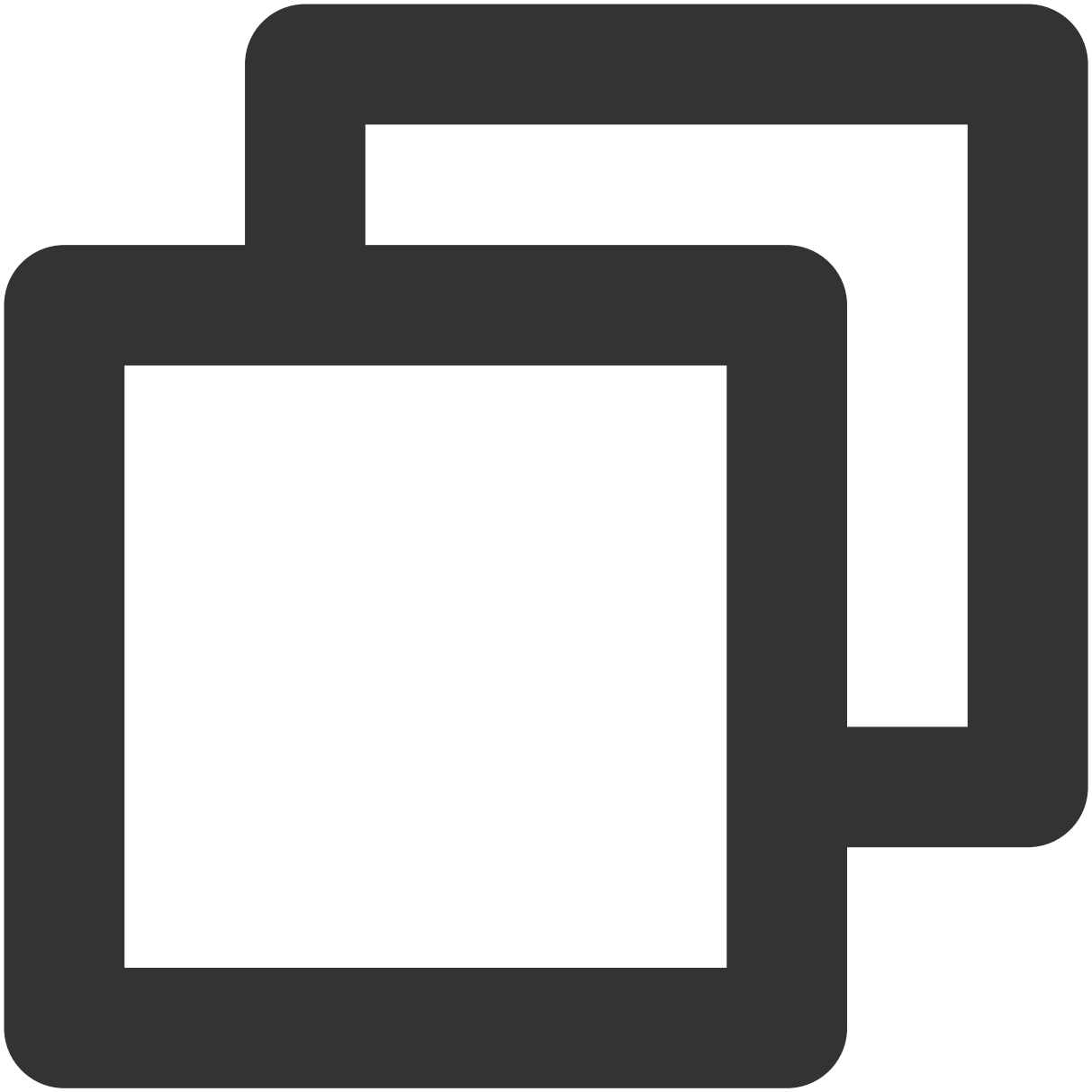


```
controller:
  service:
    annotations:
      service.cloud.tencent.com/direct-access: "true" # Enable CLB direct access
```

Enabling Direct Connection in VPC-CNI Network Mode

If the cluster network is VPC-CNI, directly configure `values.yaml` to enable CLB direct connection:

©2013-2022 Tencent Cloud. All rights reserved.



```
controller:
  service:
    annotations:
      service.cloud.tencent.com/direct-access: "true" # Enable CLB direct access
```

©2013-2022 Tencent Cloud. All rights reserved.

Optimization for High Concurrency Scenarios

Last updated：2024-08-12 17:48:23

Operation Scenarios

This document introduces how to configure and optimize Nginx Ingress for high concurrency scenarios.

Operation Guide

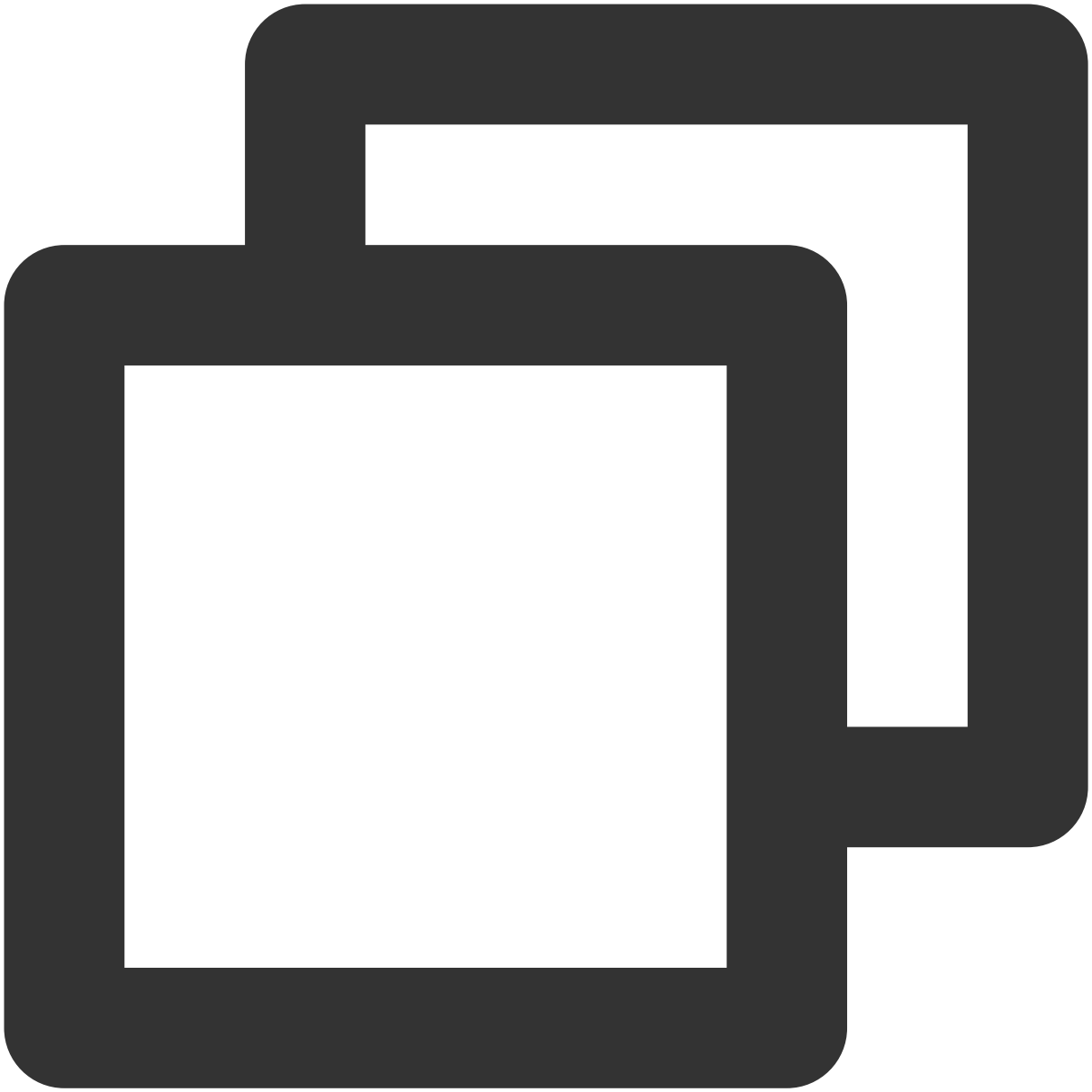
Increasing CLB Specifications and Bandwidth

High concurrency scenarios require high traffic throughput and forwarding performance of CLB. You can manually create a CLB in the CLB Console, select LCU-supported instance specifications, choose the model as needed, and increase the bandwidth limit (ensure the VPC is consistent with that of the TKE cluster).
After the CLB is created, configure Nginx Ingress to reuse this CLB as the traffic entry. For details, refer to Custom Definition CLB.

Tuning Kernel Parameters and Nginx Configuration

Optimize kernel parameters and Nginx configuration for high concurrency scenarios. `values.yaml` configuration method:

©2013-2022 Tencent Cloud. All rights reserved.



```
controller:
  extraInitContainers:
    - name: sysctl
      image: busybox
      imagePullPolicy: IfNotPresent
      securityContext:
        privileged: true
      command:
        - sh
        - -c
        - |
```

©2013-2022 Tencent Cloud. All rights reserved.

```
          sysctl -w net.core.somaxconn=65535 # Increase connection queue to prevent
          sysctl -w net.ipv4.ip_local_port_range="1024 65535" # Expand the source p
          sysctl -w net.ipv4.tcp_tw_reuse=1 # Enable TIME_WAIT reuse to allow new c
          sysctl -w fs.file-max=1048576 # Increase the file handle count to prevent
  config:
    # The number of requests that can be processed by a persistent connection betwe
    # Reference: https://kubernetes.github.io/ingress-nginx/user-guide/nginx-config
    keep-alive-requests: "1000"
    # The maximum number of idle persistent connections (not the maximum number of
    # Reference: https://kubernetes.github.io/ingress-nginx/user-guide/nginx-config
    upstream-keepalive-connections: "2000"
    # The maximum number of connections that each worker process can open is 16384
    # Reference: https://kubernetes.github.io/ingress-nginx/user-guide/nginx-config
    max-worker-connections: "65536"
```

Note:

For details, refer to Nginx Ingress High-Concurrency Practices.

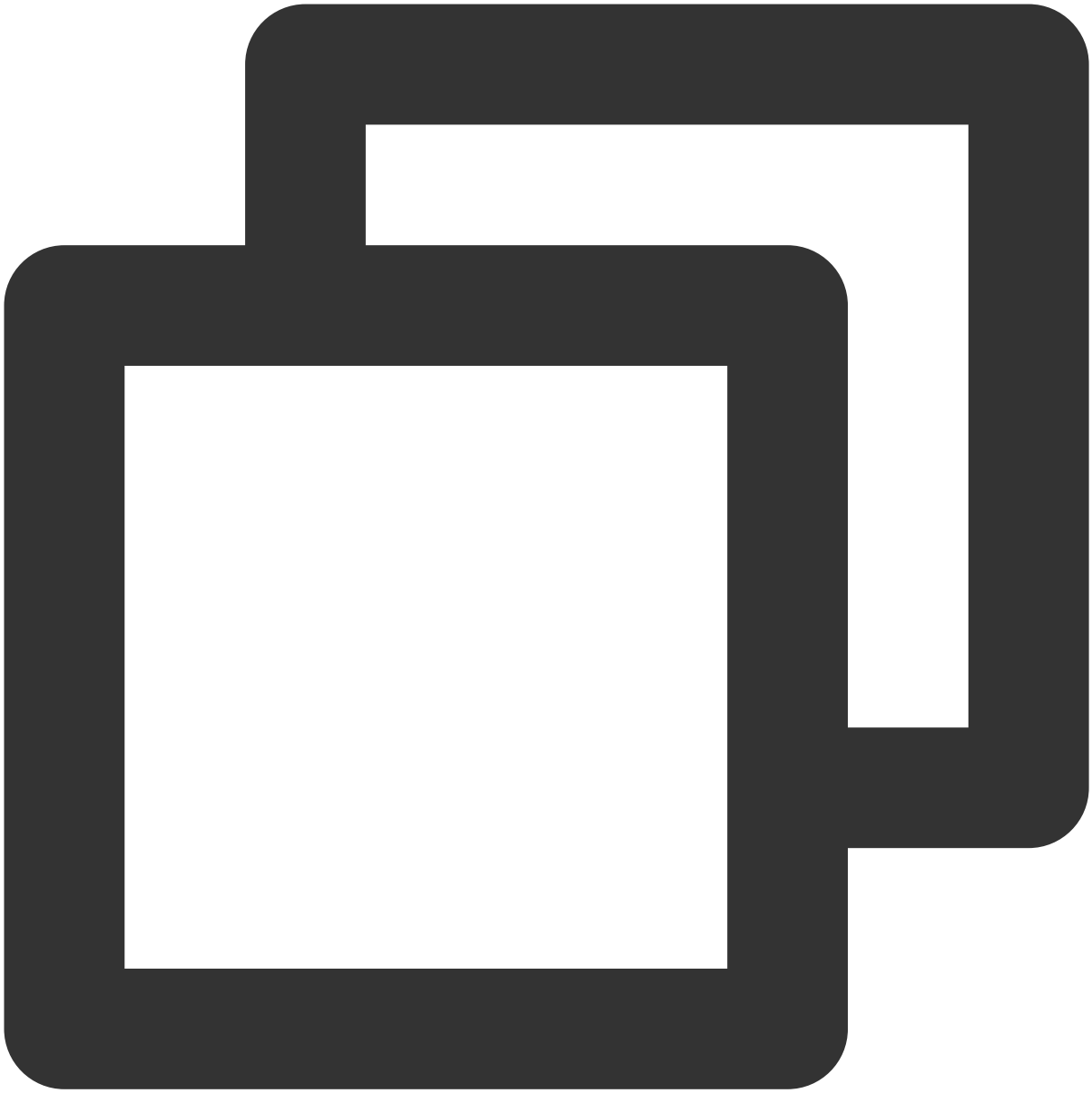
Log Rotation

Nginx Ingress will print logs to the container's standard output by default, which will be managed automatically by the container during running. In high-concurrency scenarios, this may lead to high CPU occupancy.

The solution is to output Nginx Ingress logs to log files and use a sidecar to automatically rotate the log files, preventing the disk space from being filled up with logs.

`values.yaml` configuration method:

©2013-2022 Tencent Cloud. All rights reserved.



```
controller:
  config:
    # Nginx logs are written to log files to avoid high CPU utilization under high
    access-log-path: /var/log/nginx/nginx_access.log
    error-log-path: /var/log/nginx/nginx_error.log
  extraVolumes:
    - name: log # Log mounting directory of the controller
      emptyDir: {}
  extraVolumeMounts:
    - name: log # Log directory shared by the logrotate and controller
      mountPath: /var/log/nginx
```

©2013-2022 Tencent Cloud. All rights reserved.

```
extraContainers: # Logrotate sidecar container for log rotation
  - name: logrotate
    image: imroc/logrotate:latest # https://github.com/imroc/docker-logrotate
    imagePullPolicy: IfNotPresent
    env:
      - name: LOGROTATE_FILE_PATTERN # Pattern of rotated log files, matching the
        value: "/var/log/nginx/nginx_*.log"
      - name: LOGROTATE_FILESIZE # Threshold of log file size for rotation
        value: "100M"
      - name: LOGROTATE_FILENUM # Number of rotations per log file
        value: "3"
      - name: CRON_EXPR # Crontab expression for periodic logrotate running, whic
        value: "*/1 * * * *"
      - name: CROND_LOGLEVEL # Crond log level, ranging from 0 to 8, the smaller
        value: "8"
    volumeMounts:
      - name: log
        mountPath: /var/log/nginx
```

©2013-2022 Tencent Cloud. All rights reserved.

High Availability Configuration Optimization

Last updated：2024-08-12 17:48:23

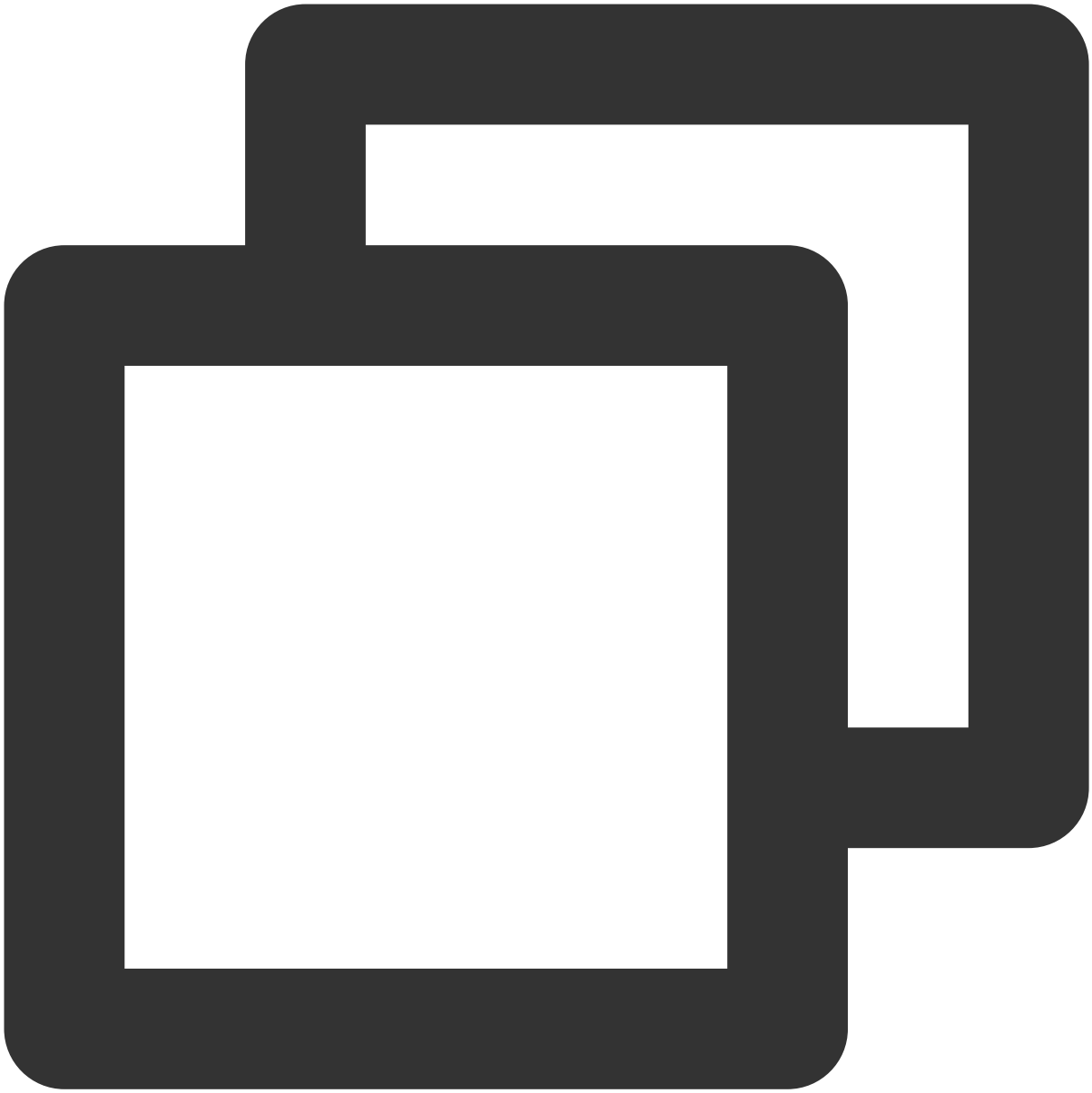
Overview

This document describes the high-availability deployment configuration methods for Nginx Ingress.

Increasing the Number of Replicas

Configure automatic scaling:

©2013-2022 Tencent Cloud. All rights reserved.

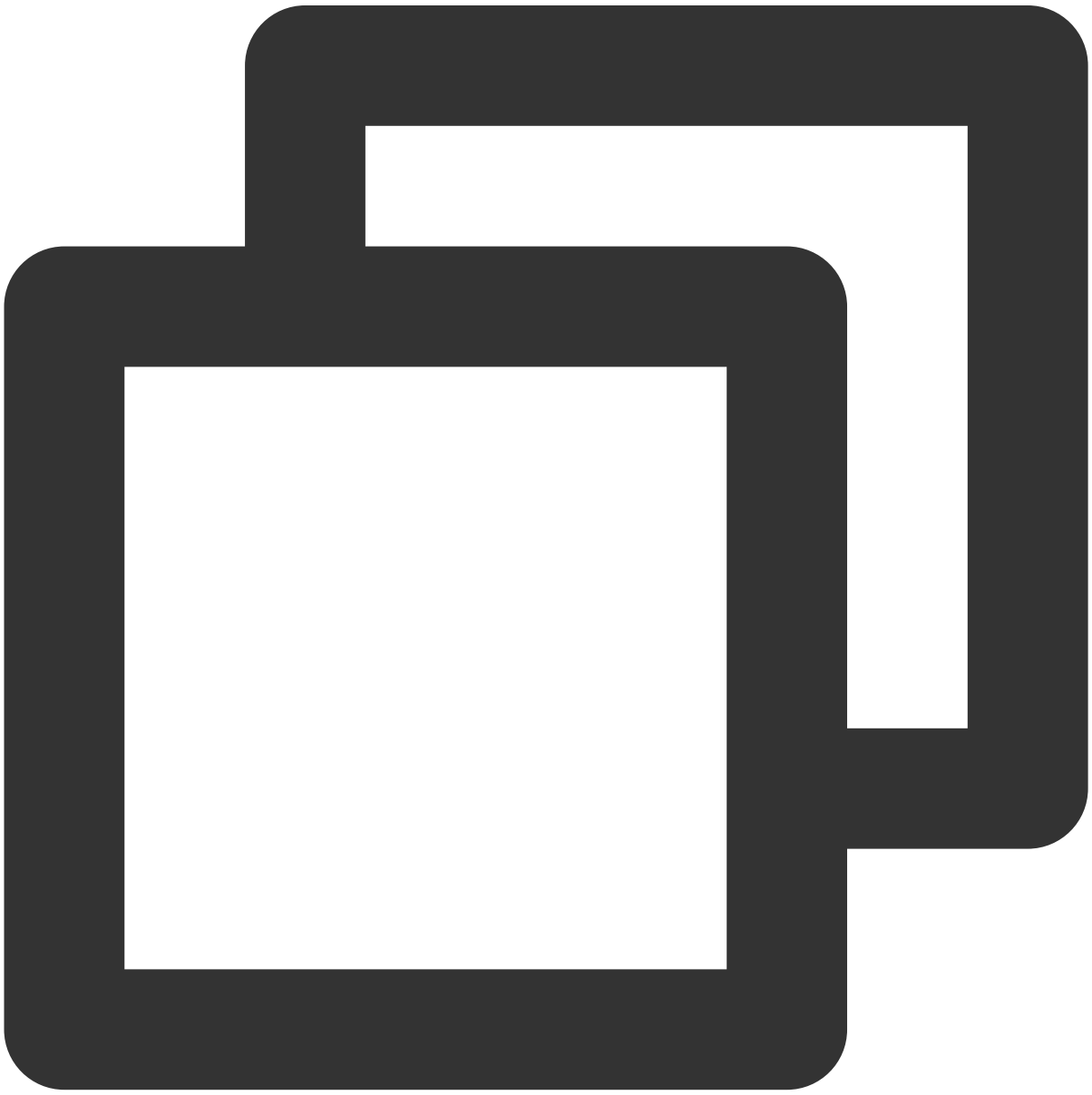


```
controller:
  autoscaling:
    enabled: true
    minReplicas: 10
    maxReplicas: 100
    targetCPUUtilizationPercentage: 50
    targetMemoryUtilizationPercentage: 50
    behavior: # Quick scale-out to handle traffic peaks, slow scale-in to leave a b
      scaleUp:
        stabilizationWindowSeconds: 300
        policies:
```

©2013-2022 Tencent Cloud. All rights reserved.

```
      - type: Percent
        value: 900
        periodSeconds: 15 # Allowing scale-out up to 9 times the current number
  scaleDown:
    stabilizationWindowSeconds: 300
    policies:
      - type: Pods
        value: 1
        periodSeconds: 600 # Allowing scale-in of only one pod at most every 10
```

If you want a fixed number of replicas, directly configure `replicaCount` :

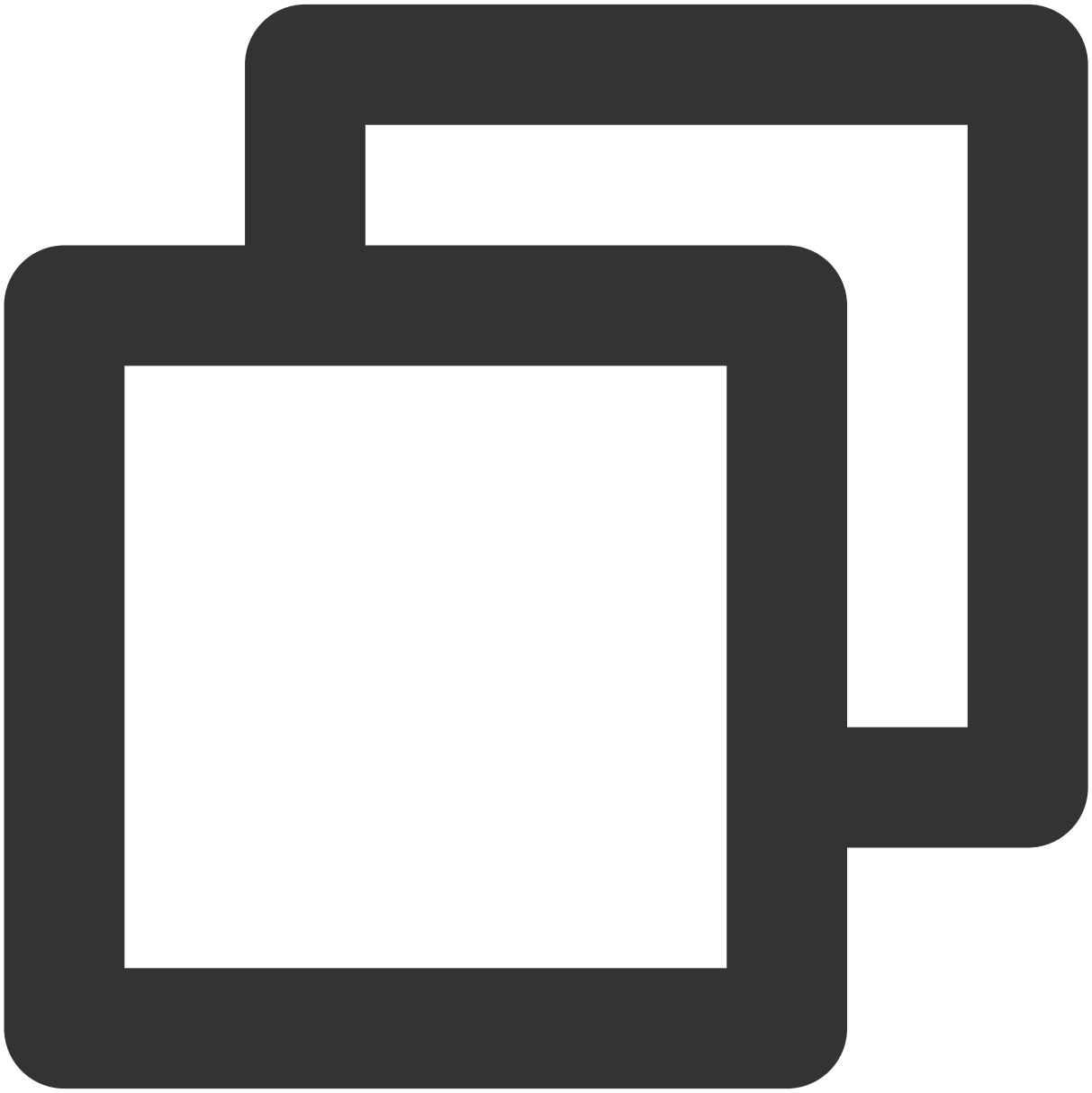


©2013-2022 Tencent Cloud. All rights reserved.

```
controller:
  replicaCount: 50
```

Spreading Scheduling

Use topology distribution constraints to spread out pods for disaster recovery and avoid single points of failure:



```
controller:
  topologySpreadConstraints: # Policy to maximize spreading
```

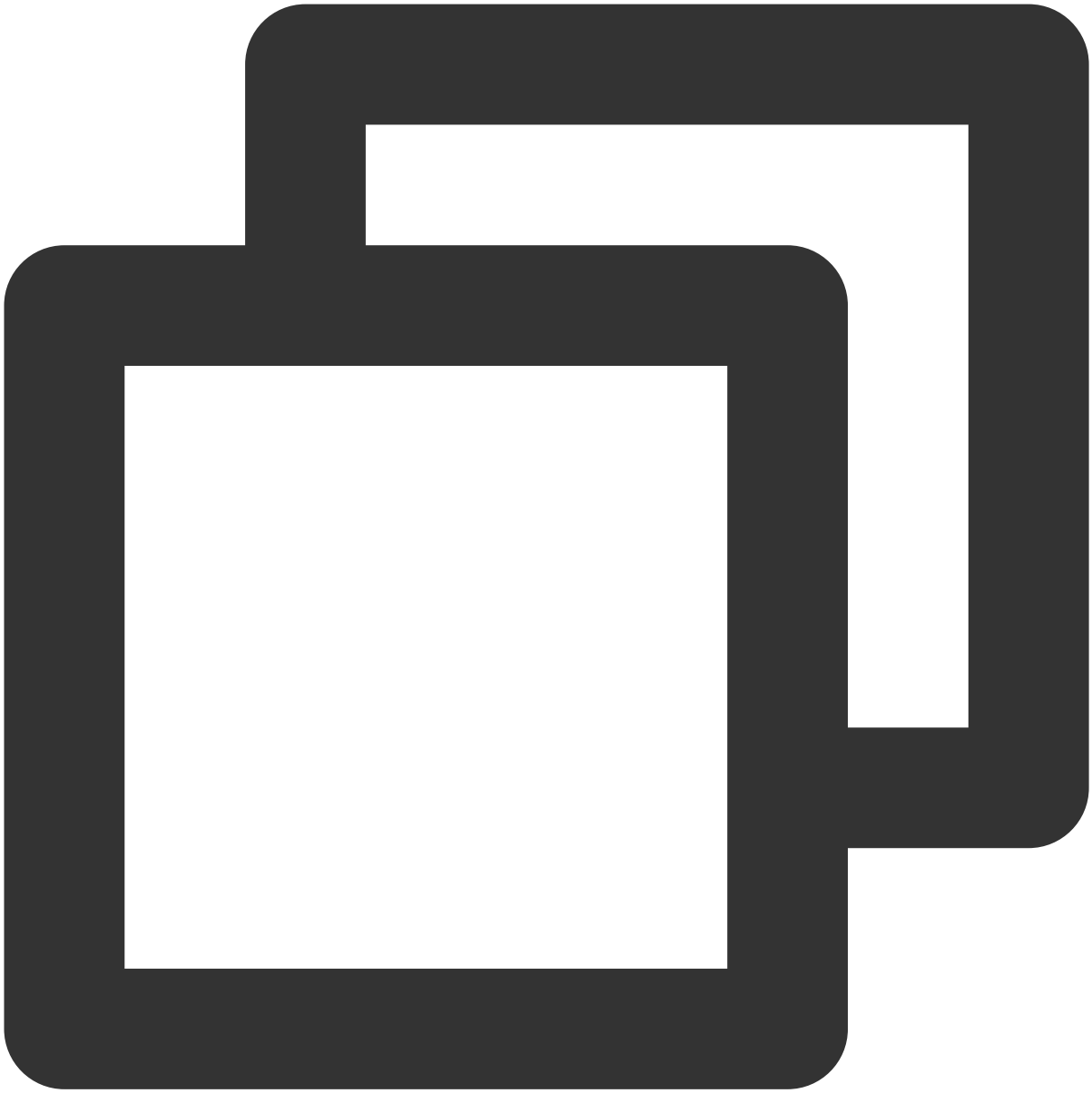
©2013-2022 Tencent Cloud. All rights reserved.

```
    - labelSelector:
        matchLabels:
          app.kubernetes.io/name: '{{ include "ingress-nginx.name" . }}'
          app.kubernetes.io/instance: '{{ .Release.Name }}'
          app.kubernetes.io/component: controller
      topologyKey: topology.kubernetes.io/zone
      maxSkew: 1
      whenUnsatisfiable: ScheduleAnyway
    - labelSelector:
        matchLabels:
          app.kubernetes.io/name: '{{ include "ingress-nginx.name" . }}'
          app.kubernetes.io/instance: '{{ .Release.Name }}'
          app.kubernetes.io/component: controller
      topologyKey: kubernetes.io/hostname
      maxSkew: 1
      whenUnsatisfiable: ScheduleAnyway
```

Scheduling to Dedicated Nodes

Typically, the load of Nginx Ingress Controller is proportional to the traffic. Considering its importance as a gateway, we recommend scheduling it to dedicated nodes or super nodes to avoid interfering with business pods or being interfered by them.
Schedule it to the specified node pool:

©2013-2022 Tencent Cloud. All rights reserved.



```
controller:
  nodeSelector:
    tke.cloud.tencent.com/nodepool-id: np-********
```

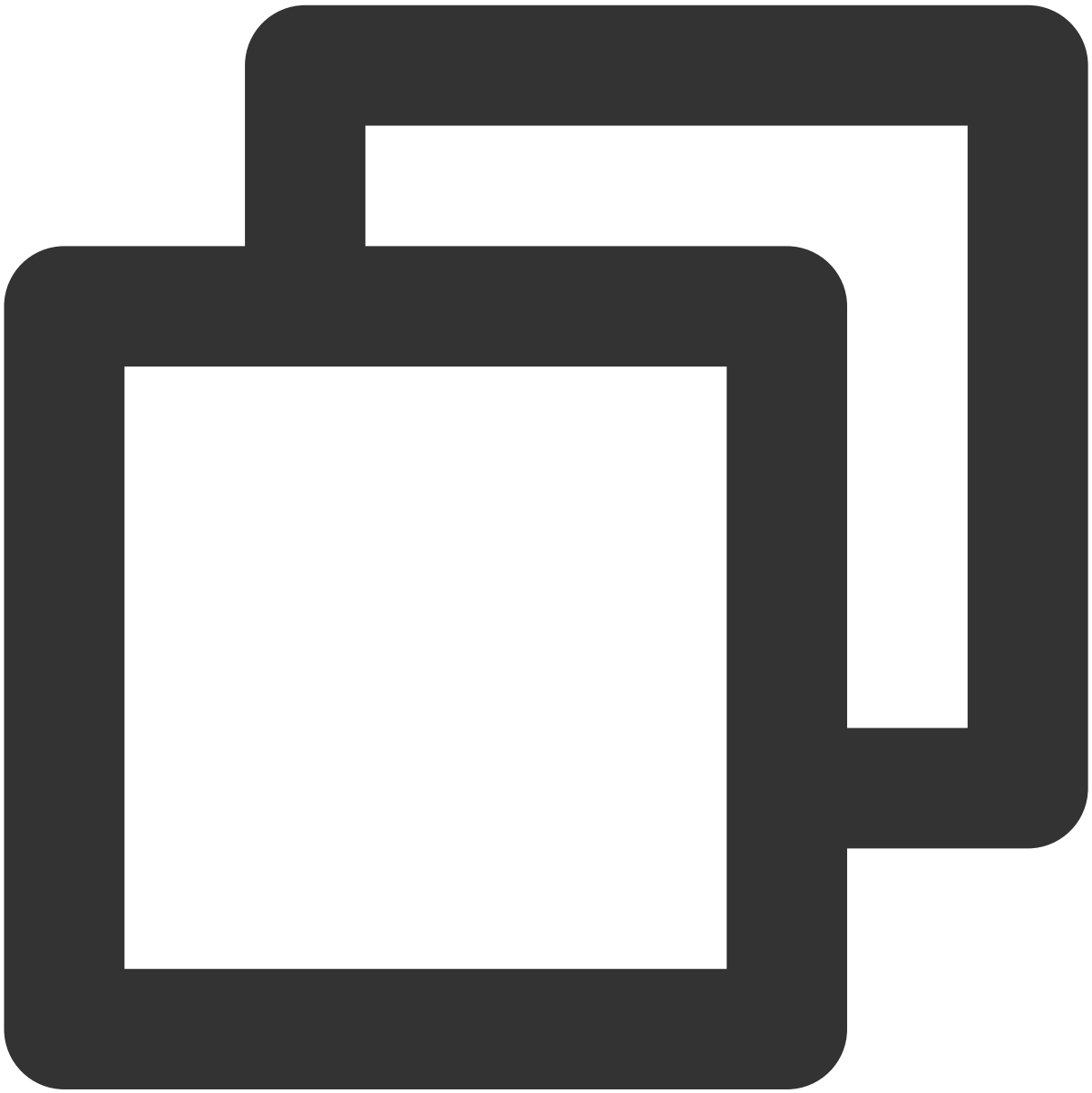
Note:

Super nodes perform better as all pods exclusively occupy the virtual machine without mutual interference. If you are using a serverless cluster, there is no need to configure the scheduling policy here, as it will only be scheduled to super nodes.

©2013-2022 Tencent Cloud. All rights reserved.

Setting Reasonable requests and limits

If Nginx Ingress is not scheduled to super nodes, set requests and limits reasonably to ensure sufficient resources while avoiding excessive resource usage that leads to high node load:

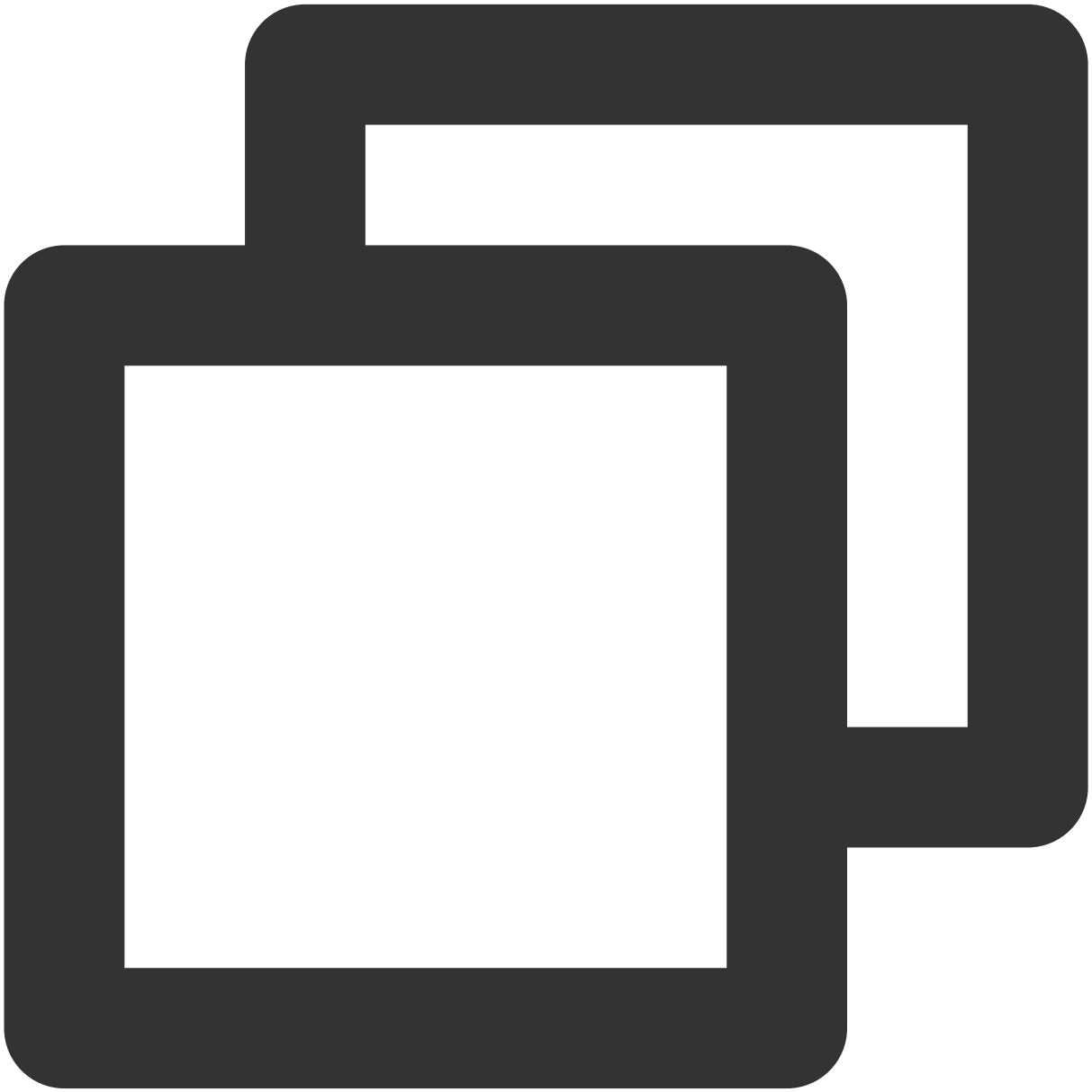


```
controller:
  resources:
    requests:
      cpu: 500m
      memory: 512Mi
```

©2013-2022 Tencent Cloud. All rights reserved.

```
    limits:
      cpu: 1000m
      memory: 1Gi
```

If you are using super nodes or a serverless cluster, you need to define requests only, that is, declare the virtual machine specifications for each pod:



```
controller:
  resources:
    requests:
      cpu: 1000m
```

©2013-2022 Tencent Cloud. All rights reserved.

```
        memory: 2Gi
```

©2013-2022 Tencent Cloud. All rights reserved.

Observability Integration

Last updated：2024-08-12 17:48:23

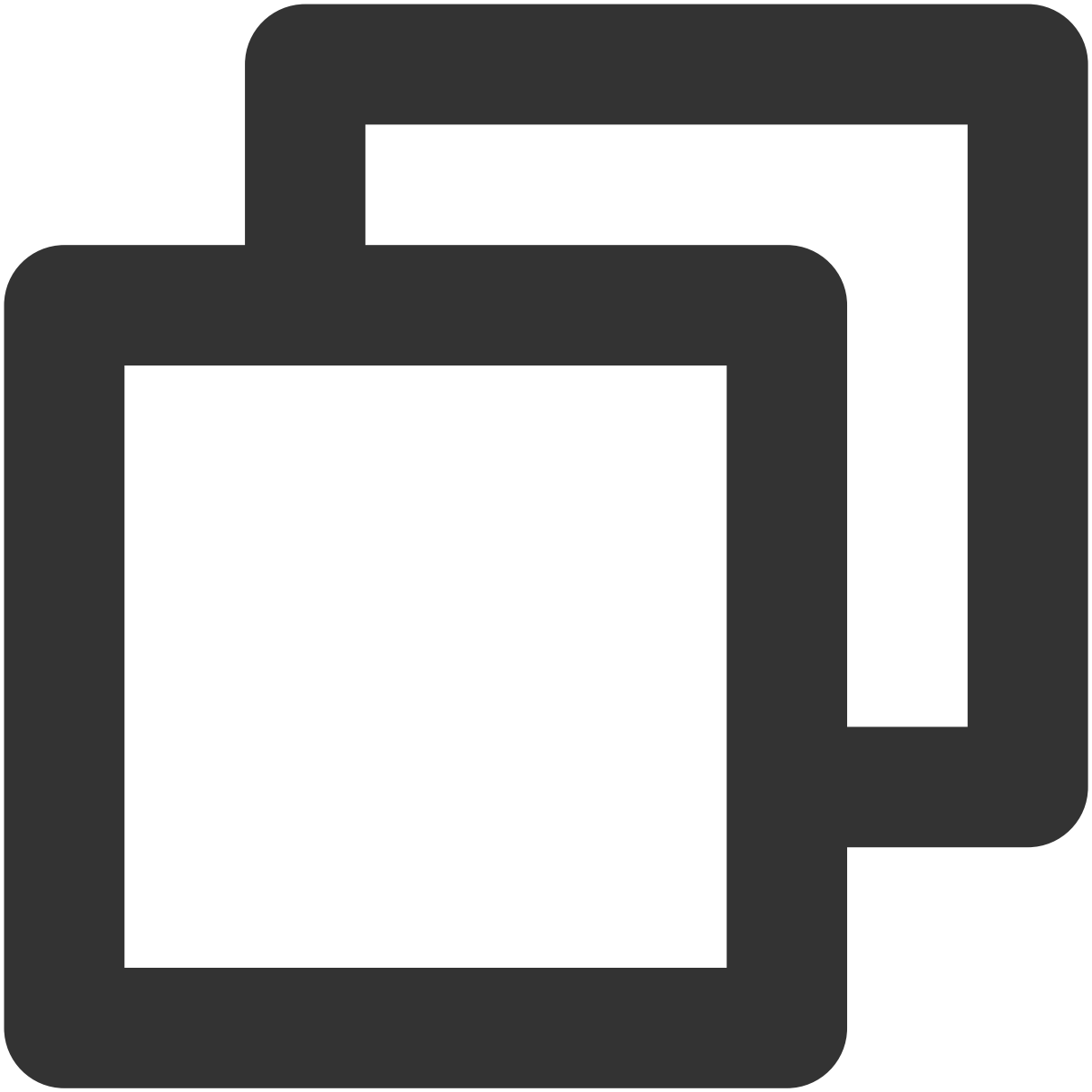
Overview

This document introduces how to configure Nginx Ingress to integrate monitoring and logging systems to enhance observability, including integration with Tencent Cloud hosted products like Prometheus, Grafana, and CLS, as well as with self-built Prometheus and Grafana.

Integrating with Prometheus Monitoring

If you use TKE Cluster Associated with Tencent Cloud Prometheus Monitoring Service, or if you have installed Prometheus Operator to monitor the cluster, you can enable ServiceMonitor to collect monitoring data for Nginx Ingress. `values.yaml` configuration method:

©2013-2022 Tencent Cloud. All rights reserved.



```
controller:
  metrics:
    enabled: true # Specifically create a service for Prometheus for Nginx Ingress
    serviceMonitor:
      enabled: true # Enable monitoring and collection rules when ServiceMonitor cu
```

Integrating with Grafana Monitoring Dashboards

©2013-2022 Tencent Cloud. All rights reserved.

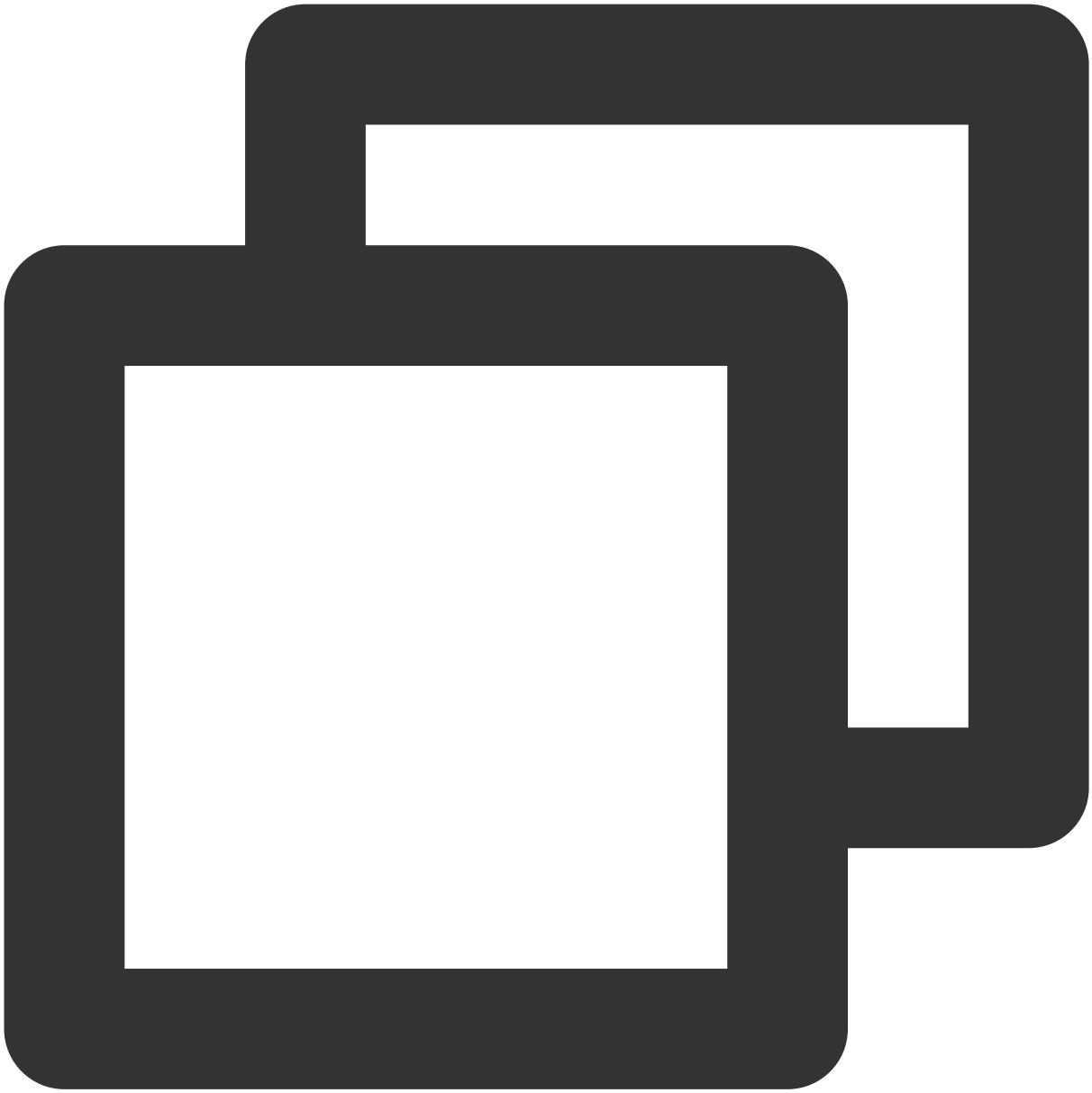
If you are using TKE Cluster Associated with Tencent Cloud Prometheus Monitoring Service and have also linked Tencent Cloud Grafana Service, or If you have your own Grafana, simply import the two monitoring dashboards (json files) provided by the official Nginx Ingress Grafana Dashboards into Grafana.

Integrating with CLS

The following content introduces how to collect the access log of Nginx Ingress Controller to CLS and analyze logs using the CLS dashboard.

1. Configure the format of the Nginx access logs in `values.yaml` and set the timezone to display the local time for better readability:

©2013-2022 Tencent Cloud. All rights reserved.



```
controller:
  config:
    log-format-upstream:
      $remote_addr - $remote_user [$time_local] "$request"
      $status $body_bytes_sent "$http_referer" "$http_user_agent"
      $request_length $request_time [$proxy_upstream_name] [$proxy_alternative_upst
      $upstream_response_length $upstream_response_time $upstream_status $req_id $h
  extraEnvs:
    - name: TZ
      value: Asia/Shanghai
```

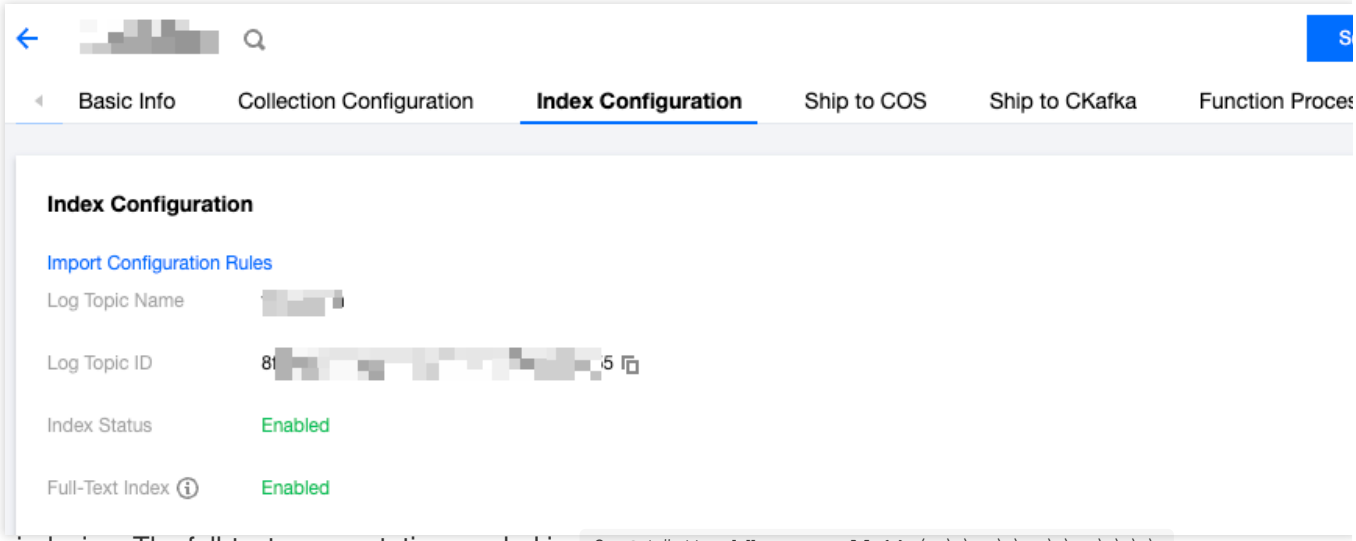
©2013-2022 Tencent Cloud. All rights reserved.

2. Ensure that the log collection feature is enabled for the cluster.

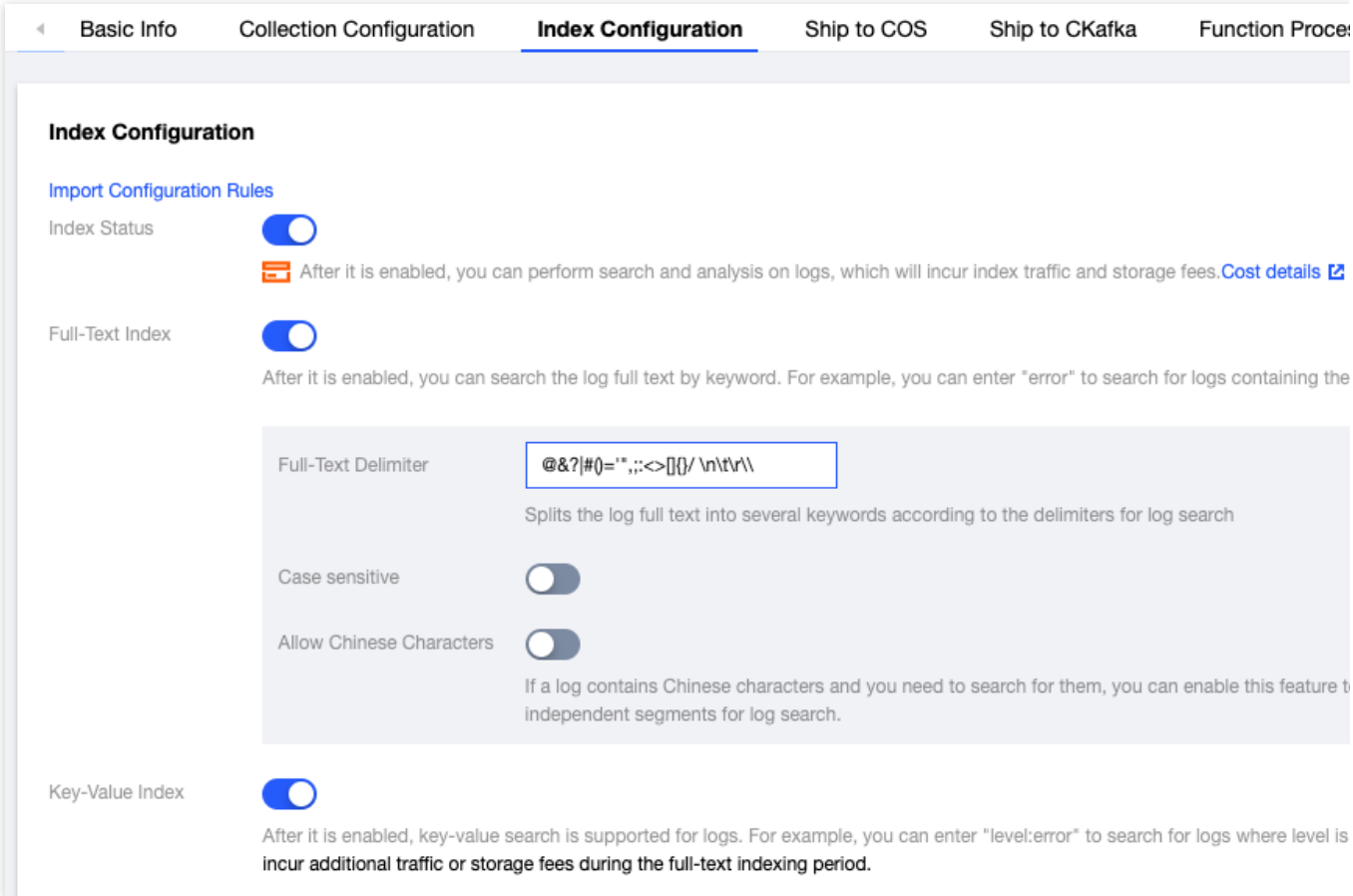
3. Prepare CLS log sets and log topics for Nginx Ingress Controller. If you do not have them, go to the CLS Console, create them as needed, and record the log topic ID.

4. Enable indexing for the log topic:

Go to the **Index Configuration** page of Log Topic, and click **Edit**:



Enable indexing. The full-text segmentation symbol is `@&?|#()='",;:<>[]{}/ \\n\\t\\r\\\\` :

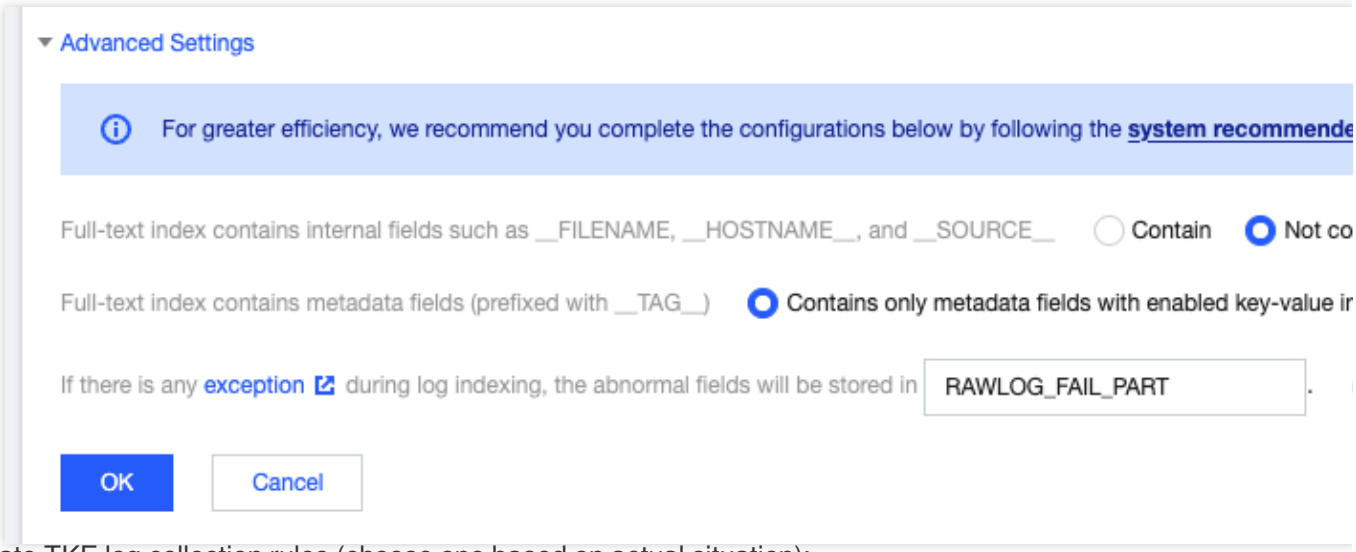


Bulk add index fields (match the configuration shown below):

©2013-2022 Tencent Cloud. All rights reserved.

Perform advanced settings:

©2013-2022 Tencent Cloud. All rights reserved.



5. Create TKE log collection rules (choose one based on actual situation):

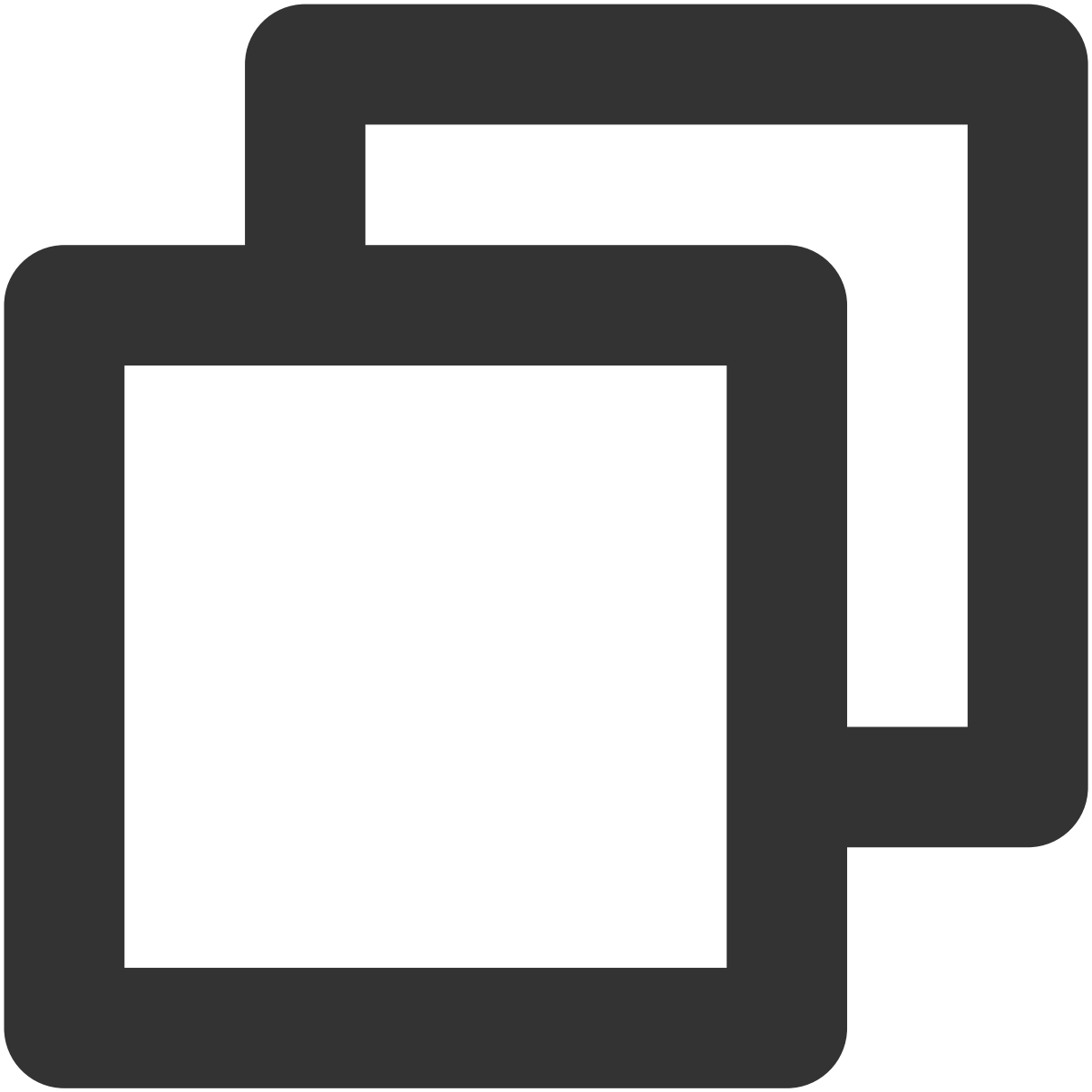
Note:

The configuration item that must be replaced is `topicId`, which is the log topic ID, indicating that the collected logs will be sent to the corresponding CLS log topic.

Depending on your situation, choose whether to collect standard output or log files. By default, Nginx Ingress outputs logs to standard output. You can also choose to save logs to log files. For details, refer to [Log Rotation].

Collect standard output:

©2013-2022 Tencent Cloud. All rights reserved.



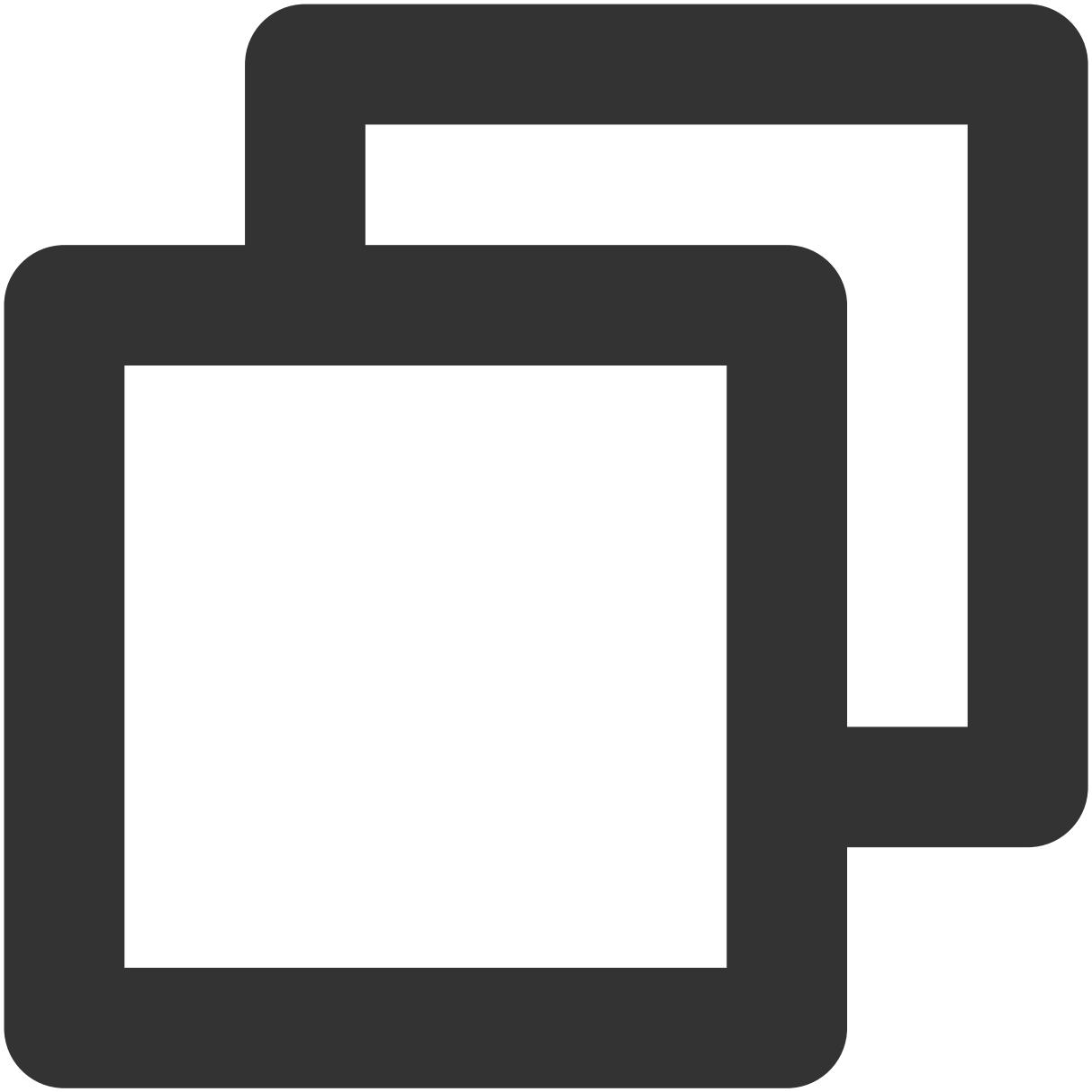
```
apiVersion: cls.cloud.tencent.com/v1
kind: LogConfig
metadata:
  name: ingress-nginx-controller # Name of the log collection rule. If there are mu
spec:
  clsDetail:
    topicId: "********-****-****-****-************" # Log topic ID to be replaced.
    logType: fullregex_log
    extractRule:
      beginningRegex: (\\S+)\\s-\\s(\\S+)\\s\\[([^\\]]+)\\]\\s\\"(\\w+)\\s(\\S+)\\s
      logRegex: (\\S+)\\s-\\s(\\S+)\\s\\[([^\\]]+)\\]\\s\\"(\\w+)\\s(\\S+)\\s([^\\"
```

©2013-2022 Tencent Cloud. All rights reserved.

```
      keys:
        - remote_addr
        - remote_user
        - time_local
        - timestamp
        - method
        - url
        - version
        - status
        - body_bytes_sent
        - http_referer
        - http_user_agent
        - request_length
        - request_time
        - proxy_upstream_name
        - proxy_alternative_upstream_name
        - upstream_addr
        - upstream_response_length
        - upstream_response_time
        - upstream_status
        - req_id
        - sys_address
  inputDetail:
    type: container_stdout
    containerStdout:
      namespace: ingress-nginx # Namespace where Nginx Ingress is located.
      workload:
        kind: deployment
        name: ingress-nginx-controller # Select the deployment name of Nginx Ingres
```

Collecting log files:

©2013-2022 Tencent Cloud. All rights reserved.



```
apiVersion: cls.cloud.tencent.com/v1
kind: LogConfig
metadata:
  name: ingress-nginx-controller # Name of the log collection rule. If there are mu
spec:
  clsDetail:
    topicId: "********-****-****-****-************" # Log topic ID to be replaced.
    logType: fullregex_log
    extractRule:
      beginningRegex: (\\S+)\\s-\\s(\\S+)\\s\\[([^\\]]+)\\]\\s\\"(\\w+)\\s(\\S+)\\s
      logRegex: (\\S+)\\s-\\s(\\S+)\\s\\[([^\\]]+)\\]\\s\\"(\\w+)\\s(\\S+)\\s([^\\"
```

©2013-2022 Tencent Cloud. All rights reserved.

```
      keys:
        - remote_addr
        - remote_user
        - time_local
        - timestamp
        - method
        - url
        - version
        - status
        - body_bytes_sent
        - http_referer
        - http_user_agent
        - request_length
        - request_time
        - proxy_upstream_name
        - proxy_alternative_upstream_name
        - upstream_addr
        - upstream_response_length
        - upstream_response_time
        - upstream_status
        - req_id
        - sys_address
  inputDetail:
    type: container_file
    containerFile:
      namespace: ingress-nginx # Namespace where Nginx Ingress is located.
      workload:
        kind: deployment
        name: ingress-nginx-controller # Select the deployment name of Nginx Ingres
      container: controller
      logPath: /var/log/nginx
      filePattern: nginx_access.log
```

6. Test Ingress requests to generate log data.

7. Go to the Search and Analyze page in the **CLS console**, select the log topic used by Nginx Ingress, and ensure the logs can be retrieved properly.

8. If everything is normal, you can use the Nginx Access Dashboard and Nginx Monitoring Dashboard of **CLS**, and select the log topic used by Nginx Ingress to display the analysis panel of Nginx access logs.

9. In the Nginx Access Dashboard and Nginx Monitoring Dashboard of **CLS**, you can directly set monitoring and alarm rules using the panels. For details, refer to the Monitoring Alarm Overview.

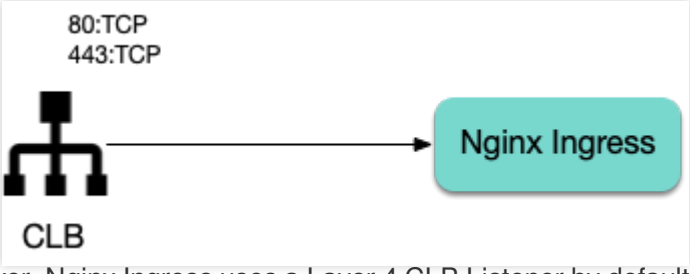
©2013-2022 Tencent Cloud. All rights reserved.

Access to Tencent Cloud WAF

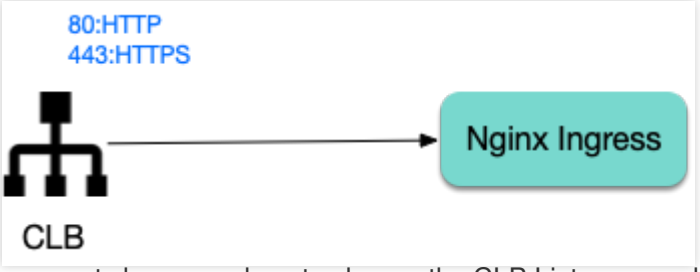
Last updated：2024-08-12 17:48:23

Background

Tencent Cloud WAF (WAF) supports access to Tencent Cloud CLB (CLB), but it requires a Layer 7 Listener (HTTP/HTTPS):



However, Nginx Ingress uses a Layer 4 CLB Listener by default:



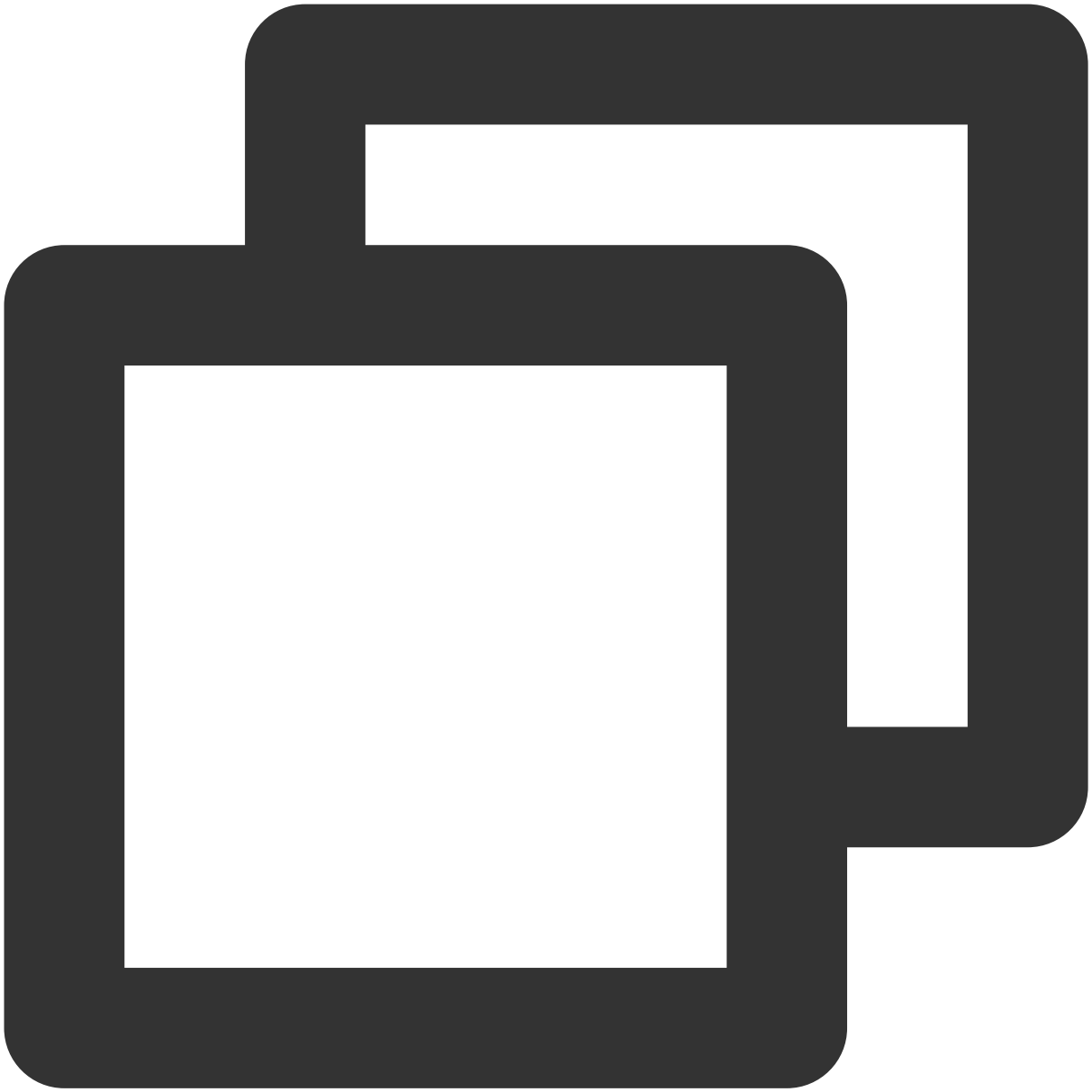
This document shows you how to change the CLB Listener used by Nginx Ingress to a layer-7 listener.

Using the Specify-protocol Annotation

The TKE service supports using the `service.cloud.tencent.com/specify-protocol` annotation to modify the CLB listener protocol. For details, see Service Extension Protocol.

`values.yaml` configuration example:

©2013-2022 Tencent Cloud. All rights reserved.



```
controller:
  service:
    annotations:
      service.cloud.tencent.com/specify-protocol: |
        {
          "80": {
            "protocol": [
              "HTTP"
            ],
            "hosts": {
              "a.example.com": {},
```

©2013-2022 Tencent Cloud. All rights reserved.

```
            "b.example.com": {}
          }
        },
        "443": {
          "protocol": [
            "HTTPS"
          ],
          "hosts": {
            "a.example.com": {
              "tls": "cert-secret-a"
            },
            "b.example.com": {
              "tls": "cert-secret-b"
            }
          }
        }
      }
```

The domain names involved in the actual Ingress rules also need to be configured in the `hosts` field of the annotation.

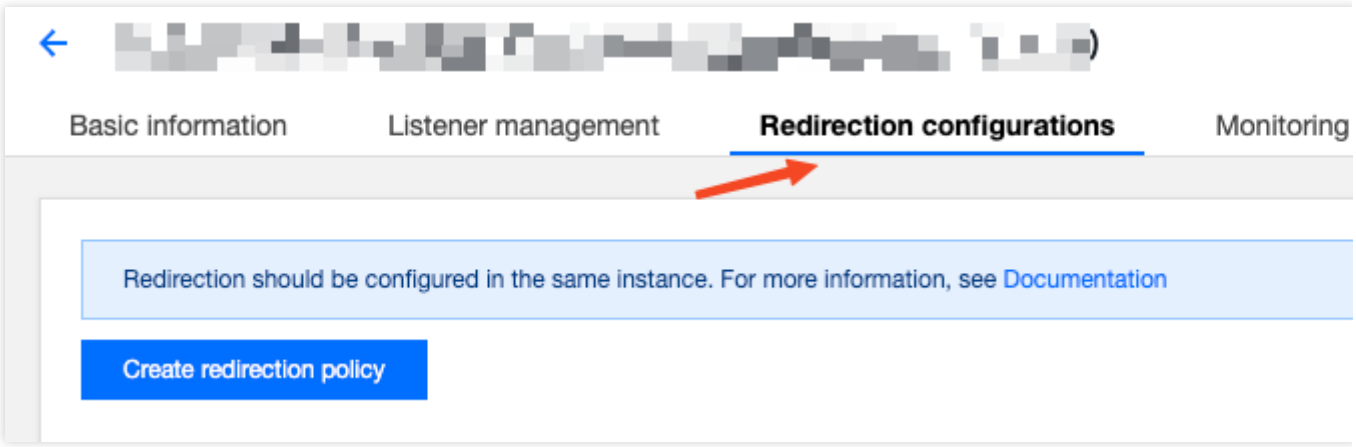
An HTTPS listener requires a certificate. First, create a certificate in My Certificates, and then create a Secret in the TKE cluster (in the namespace where Nginx Ingress is located). The Key of the Secret is `qcloud_cert_id`, and the Value is the corresponding certificate ID. Then refer to the secret name in the annotation.

`targetPorts` needs to direct the HTTPS port to port 80 (HTTP) of Nginx Ingress to avoid CLB's port 443 traffic being forwarded to Nginx Ingress's port 443 (which would lead to double certificates and forward failure).

If HTTP traffic is not needed, set `enableHttp` to false.

Note:

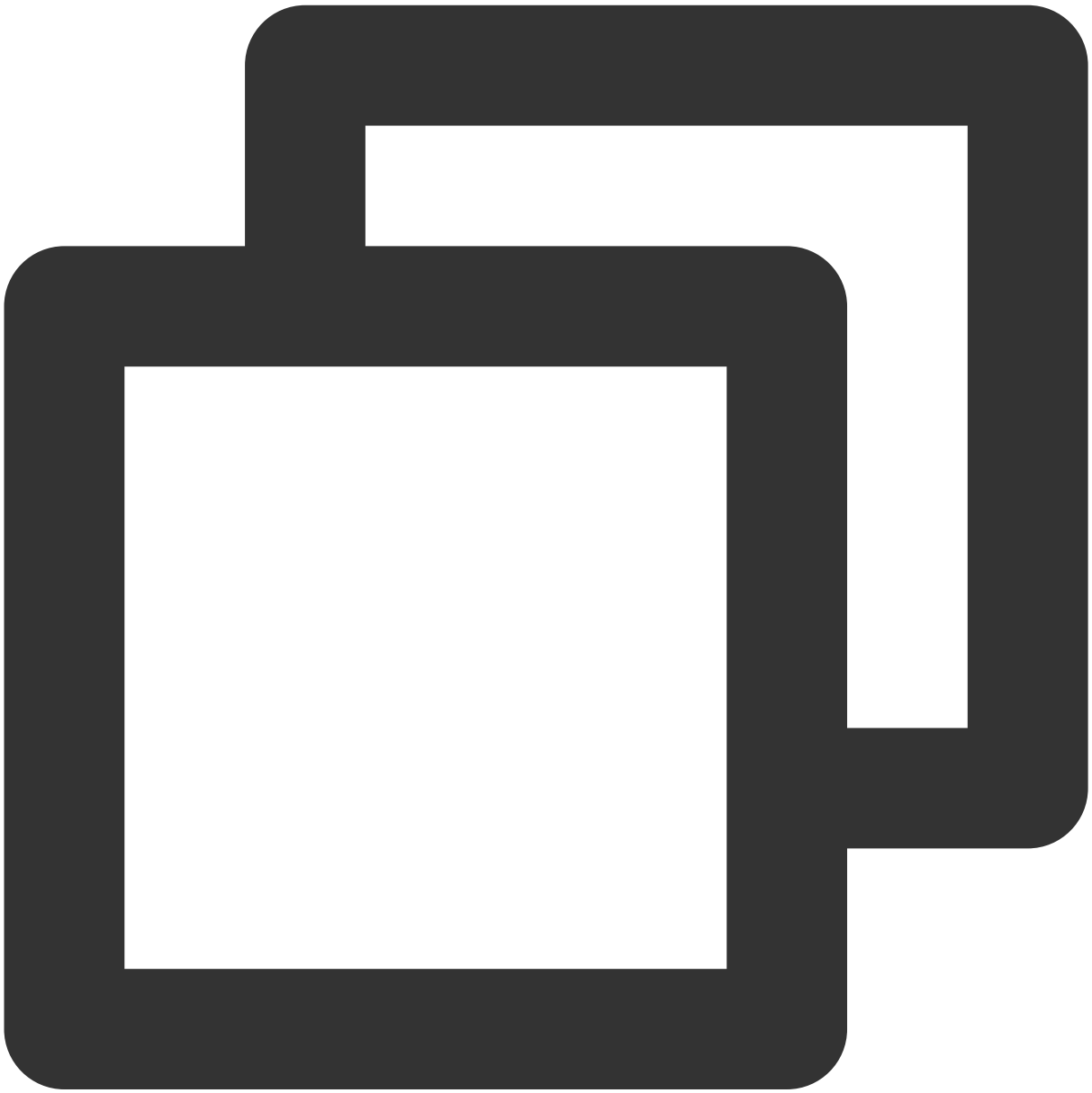
To redirect HTTP traffic to HTTPS, find the CLB instance used by Nginx Ingress in the CLB console (the instance ID can be obtained by viewing the YAML file of Nginx Ingress Controller service), and manually configure the redirection rules on the instance page:



©2013-2022 Tencent Cloud. All rights reserved.

Directions

1. Upload the certificate and copy the certificate ID in My Certificates.
2. Create the corresponding certificate secret in the namespace of Nginx Ingress (referencing the certificate ID):

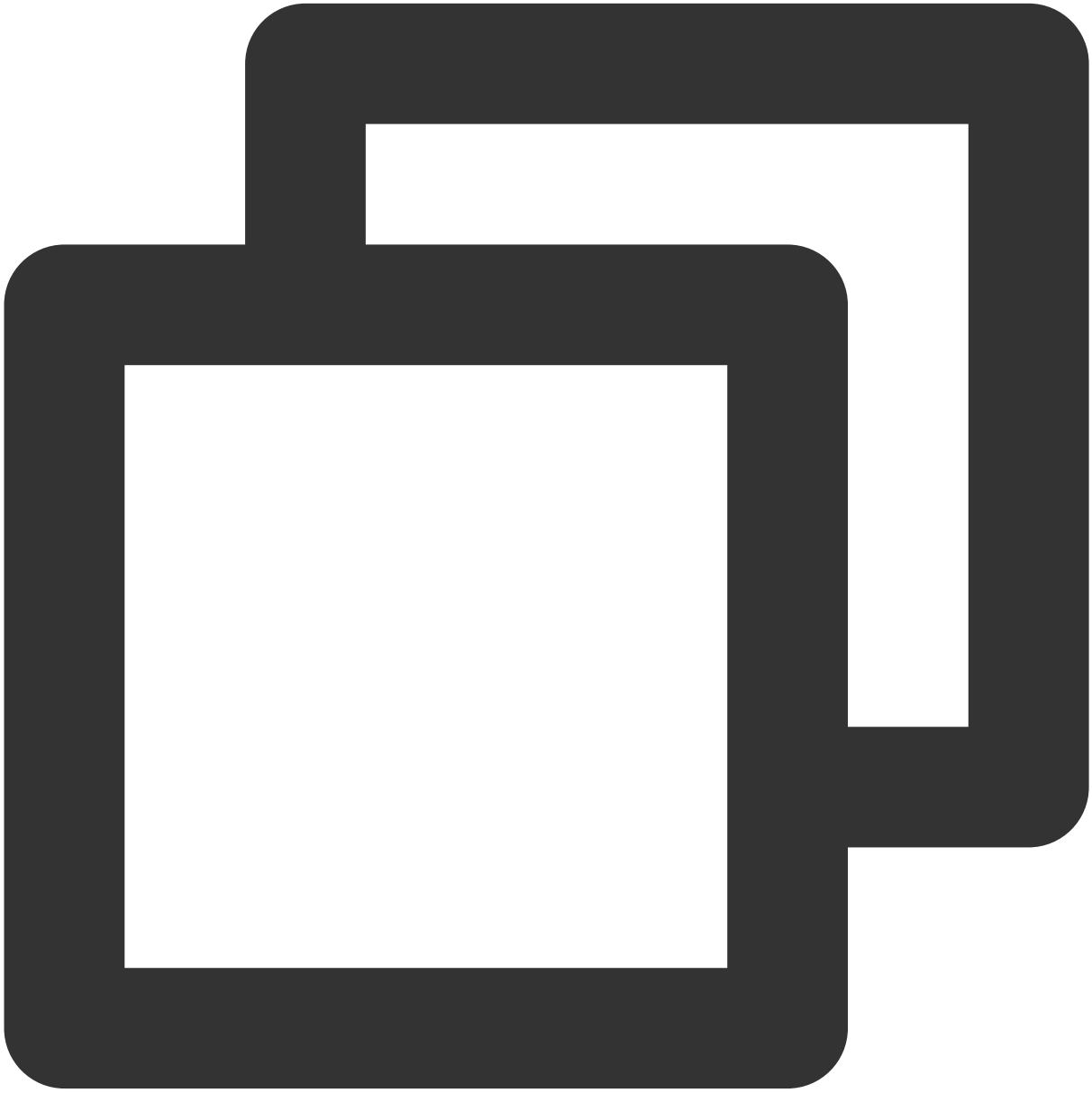


```
apiVersion: v1
kind: Secret
metadata:
  name: cert-secret-test
  namespace: ingress-nginx
```

©2013-2022 Tencent Cloud. All rights reserved.

```
stringData: # Using stringData eliminates the need of manual base64 transcoding
  # highlight-next-line
  qcloud_cert_id: E2pcp0Fy
type: Opaque
```

3. Configure `values.yaml` :



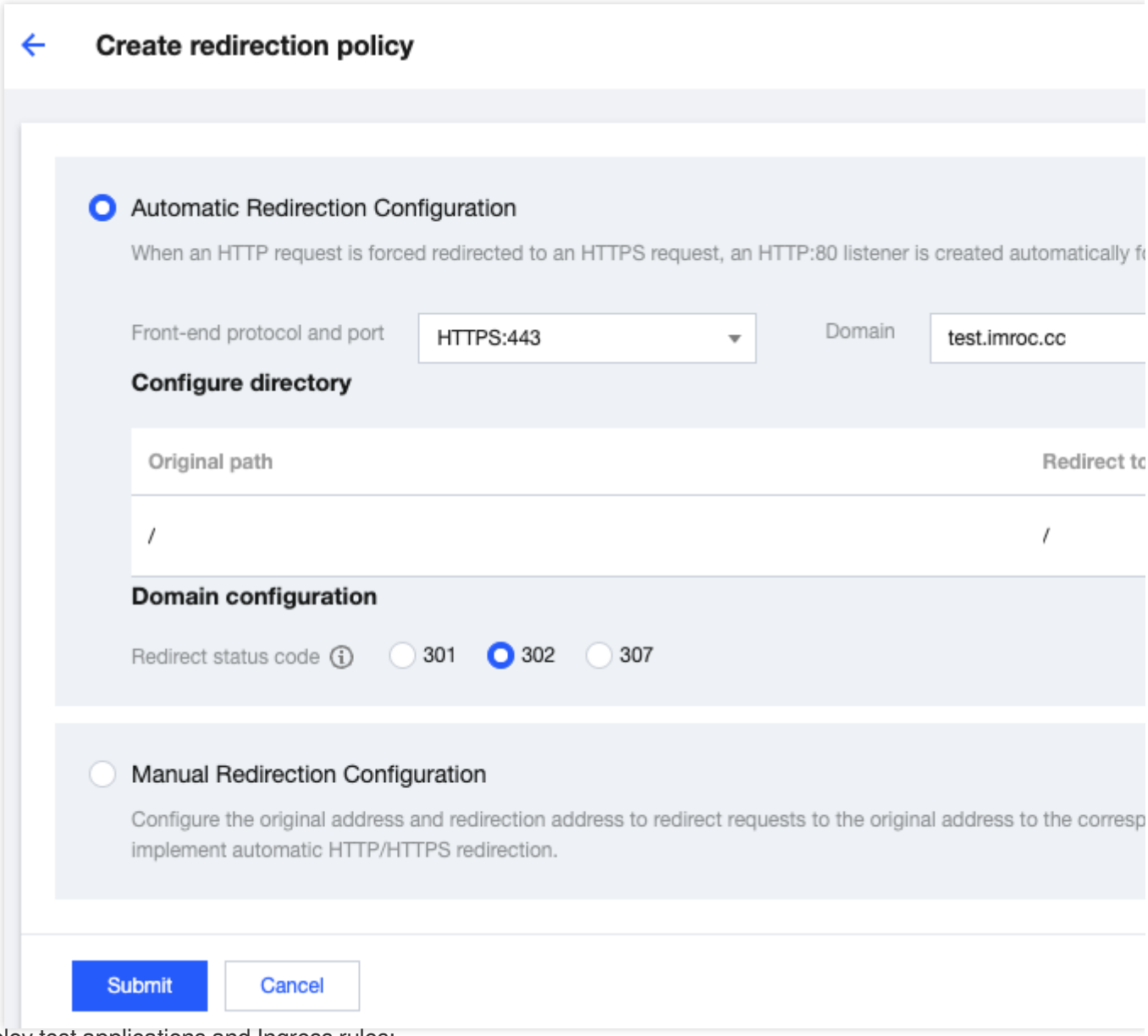
```
controller: # The following configuration replaces the dependent image with the mir
  image:
    registry: docker.io
    image: k8smirror/ingress-nginx-controller
```

©2013-2022 Tencent Cloud. All rights reserved.

```
admissionWebhooks:
  patch:
    image:
      registry: docker.io
      image: k8smirror/ingress-nginx-kube-webhook-certgen
defaultBackend:
  image:
    registry: docker.io
    image: k8smirror/defaultbackend-amd64
opentelemetry:
  image:
    registry: docker.io
    image: k8smirror/ingress-nginx-opentelemetry
service:
  enableHttp: false
  targetPorts:
    https: http
  annotations:
    service.cloud.tencent.com/specify-protocol: |
      {
        "80": {
          "protocol": [
            "HTTP"
          ],
          "hosts": {
            "test.example.com": {}
          }
        },
        "443": {
          "protocol": [
            "HTTPS"
          ],
          "hosts": {
            "test.example.com": {
              "tls": "cert-secret-test"
            }
          }
        }
      }
```

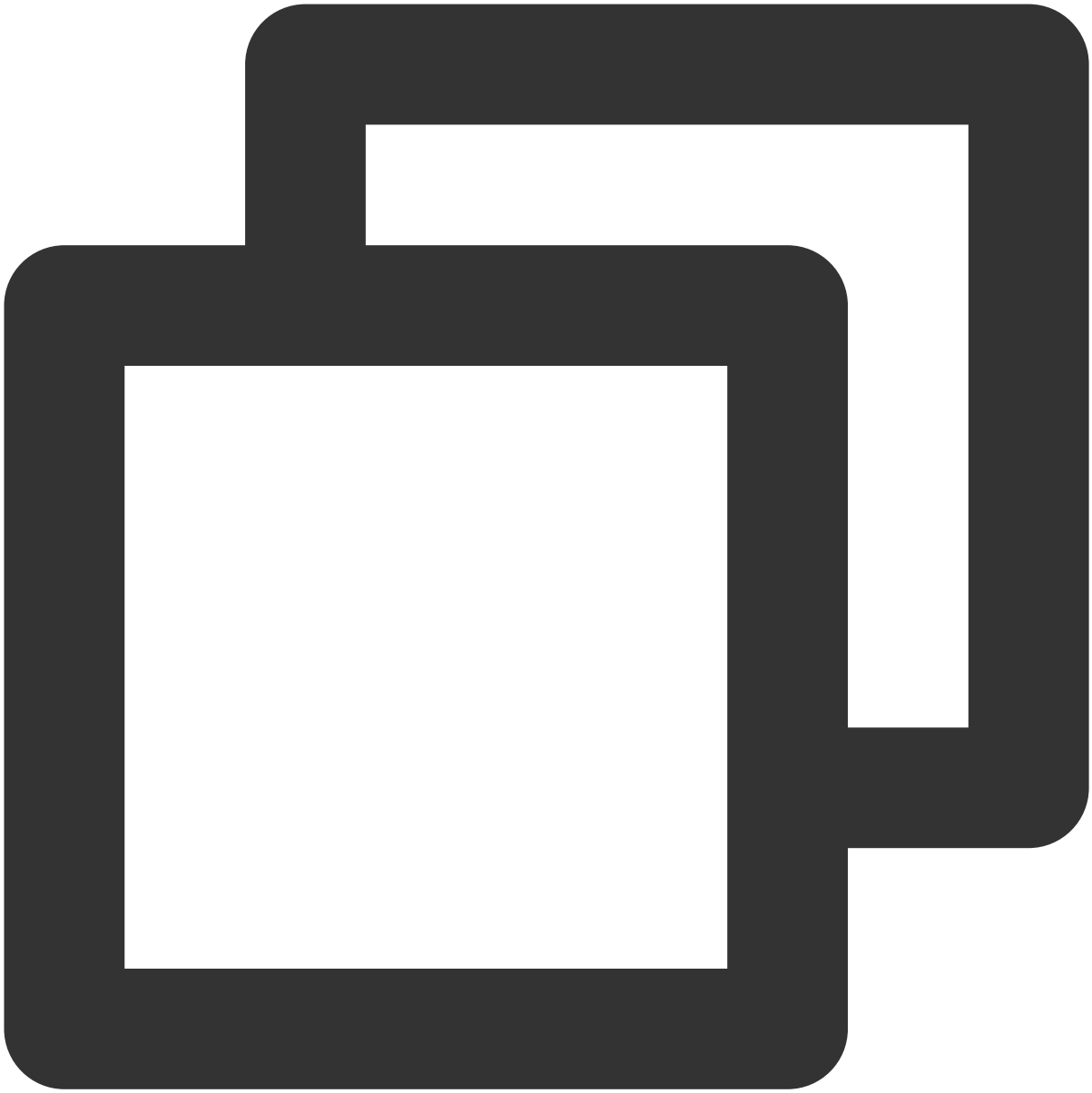
4. If needed, you can redirect HTTP to HTTPS automatically by configuring the redirection rule in the CLB console:

©2013-2022 Tencent Cloud. All rights reserved.



5. Deploy test applications and Ingress rules:

©2013-2022 Tencent Cloud. All rights reserved.



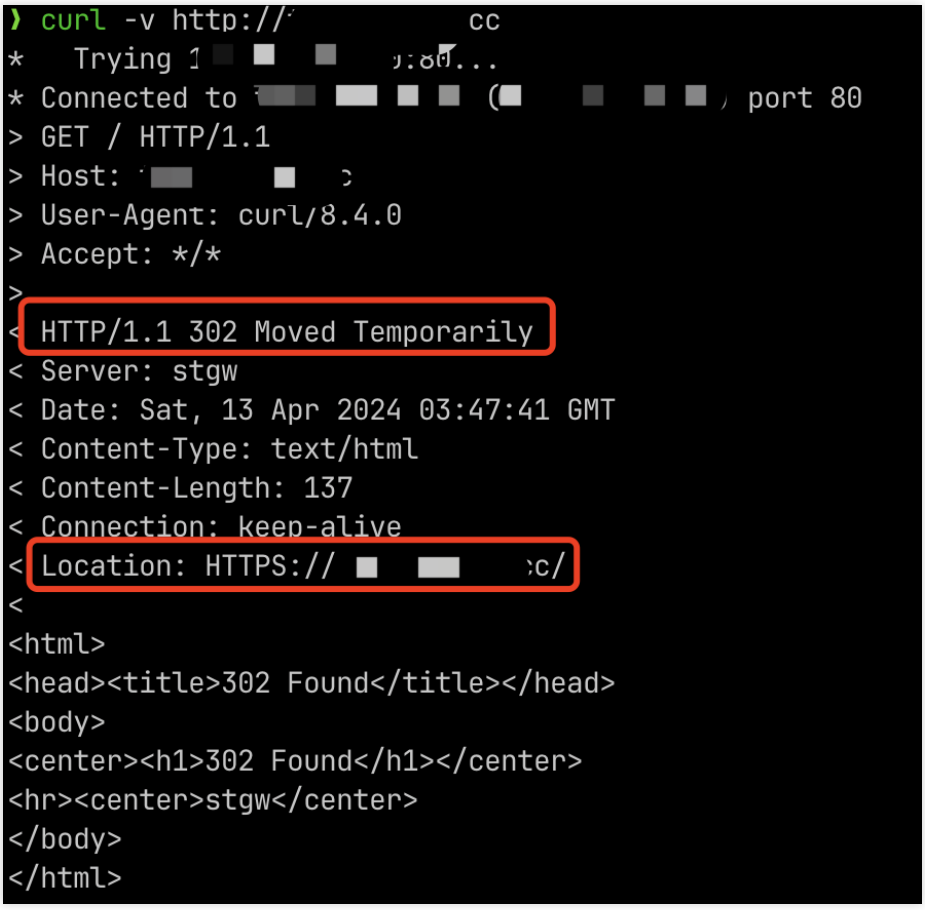
```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: nginx
  name: nginx
spec:
  ports:
    - port: 80
      protocol: TCP
      targetPort: 80
```

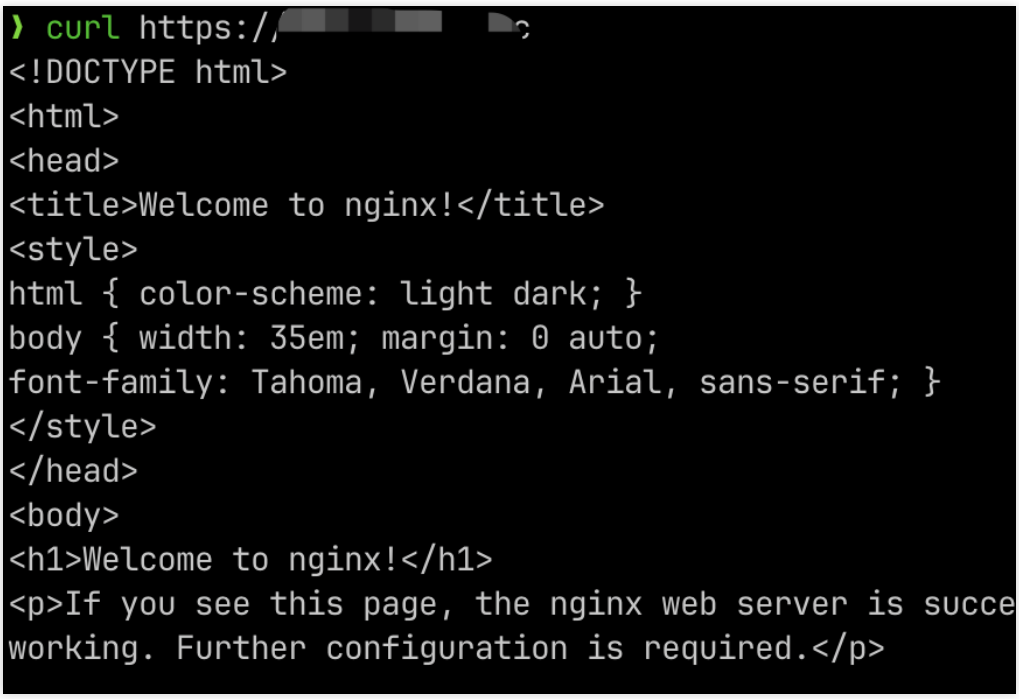
©2013-2022 Tencent Cloud. All rights reserved.

```
  selector:
    app: nginx
  type: NodePort
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - image: nginx:latest
          name: nginx
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: nginx
spec:
  ingressClassName: nginx
  rules:
    - host: test.example.com
      http:
        paths:
          - backend:
              service:
                name: nginx
                port:
                  number: 80
            path: /
            pathType: Prefix
```

6. After configuring hosts or domain name resolution, test if the feature works properly:

©2013-2022 Tencent Cloud. All rights reserved.





Configuring WAF

After Nginx Ingress is configured, if the corresponding CLB listener has been changed to HTTP/HTTPS, it satisfies the prerequisite for Nginx Ingress to connect to WAF. You can then follow the instructions in the WAF Official

©2013-2022 Tencent Cloud. All rights reserved.

Documentation to complete the WAF access to Nginx Ingress.

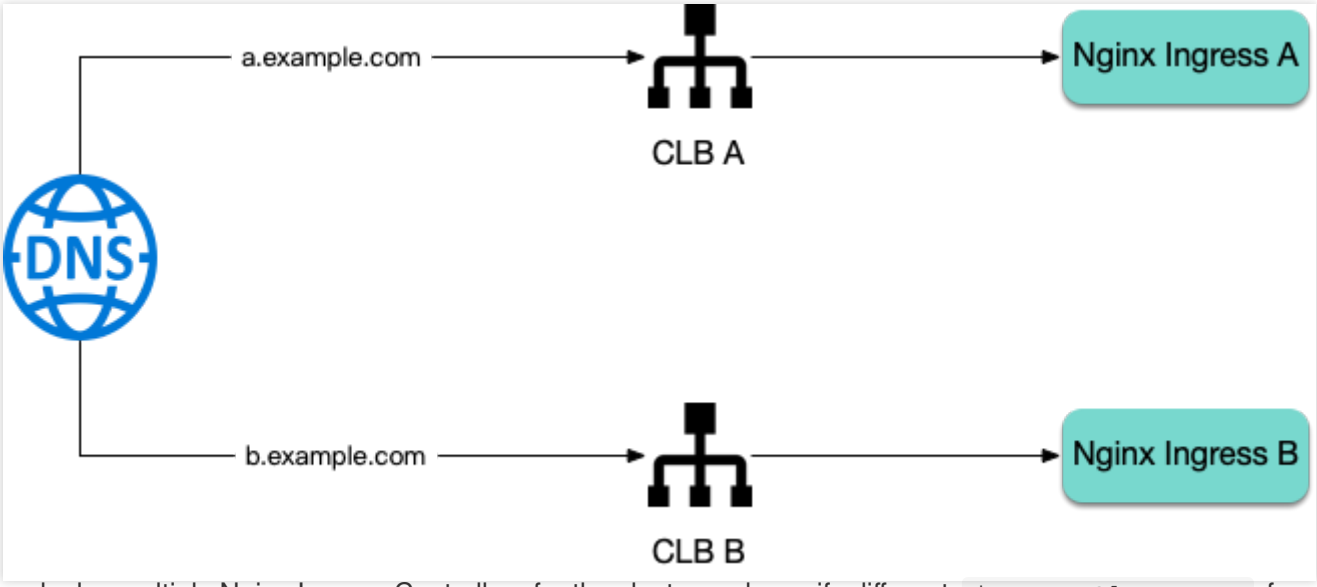
©2013-2022 Tencent Cloud. All rights reserved.

Installing Multiple Nginx Ingress Controllers

Last updated：2024-08-12 17:48:23

Overview

If you need to deploy multiple Nginx Ingress Controllers, that is, you want to use different traffic entries with different Ingress rules:



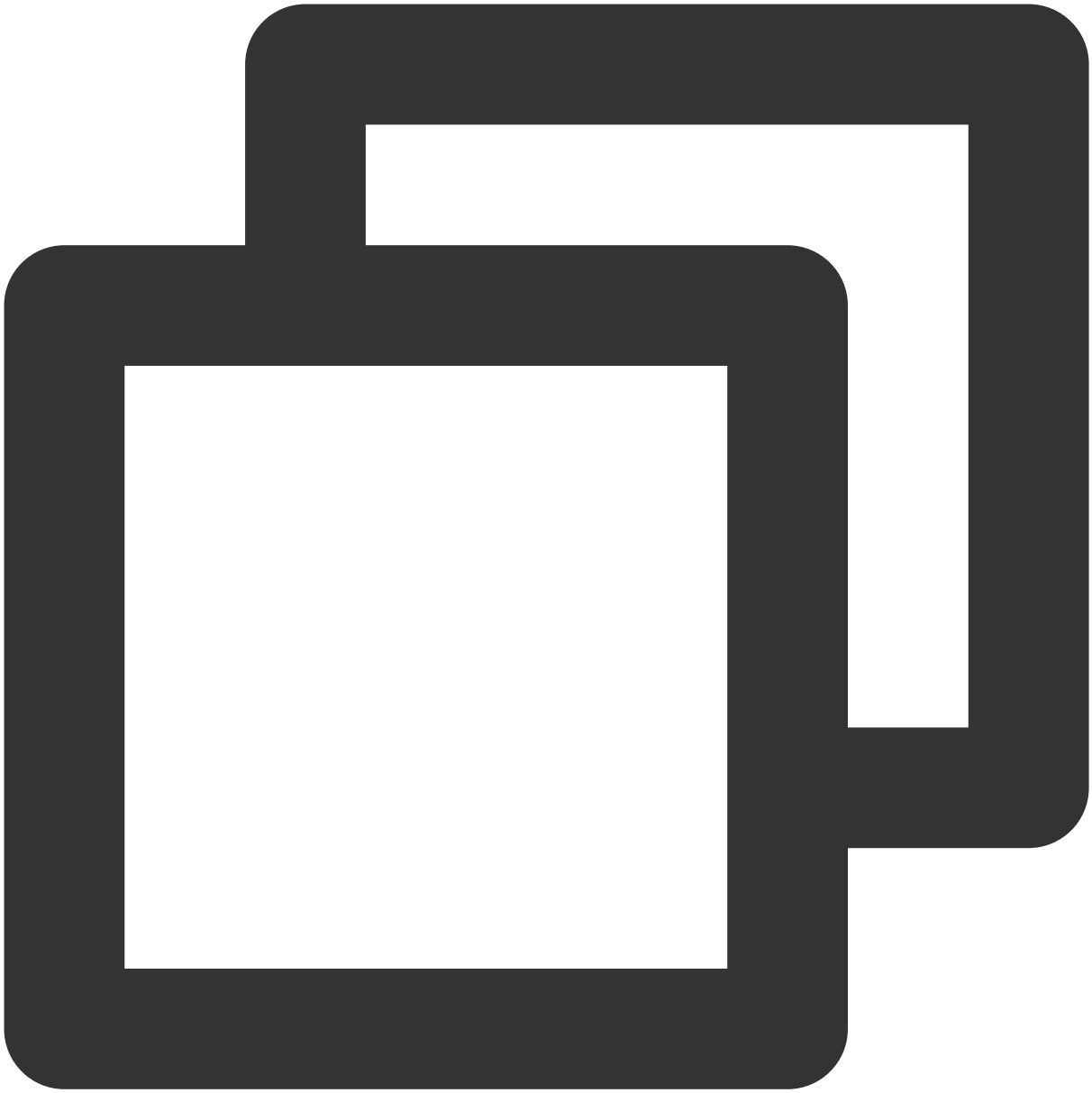
You can deploy multiple Nginx Ingress Controllers for the cluster and specify different `ingressClassName` for different Ingresses.

This document describes the configuration methods for installing multiple Nginx Ingress Controllers.

Configuration Method

To install multiple Nginx Ingress Controllers, you need to specify `ingressClassName` in `values.yaml` (ensure there are no conflicts):

©2013-2022 Tencent Cloud. All rights reserved.



```
controller:
  ingressClassName: prod
  ingressClassResource:
    name: prod
    controllerValue: k8s.io/ingress-prod
```

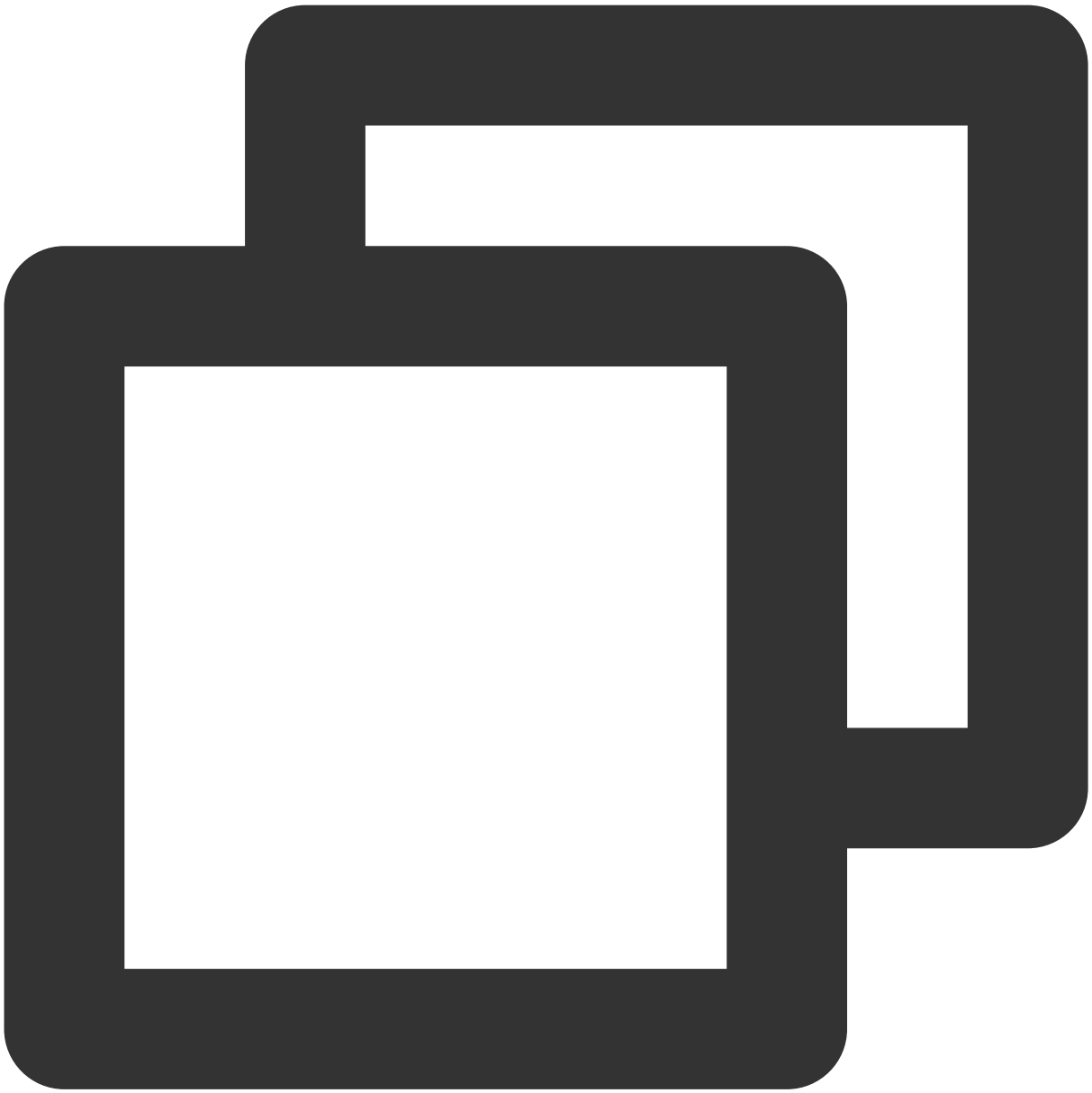
Note:

Three fields need to be changed simultaneously.

Additionally, the release names for multiple instances must be different from those of installed ones. **Even if the namespaces are different, the release names cannot be the same** (to avoid ClusterRole conflicts). The sample

©2013-2022 Tencent Cloud. All rights reserved.

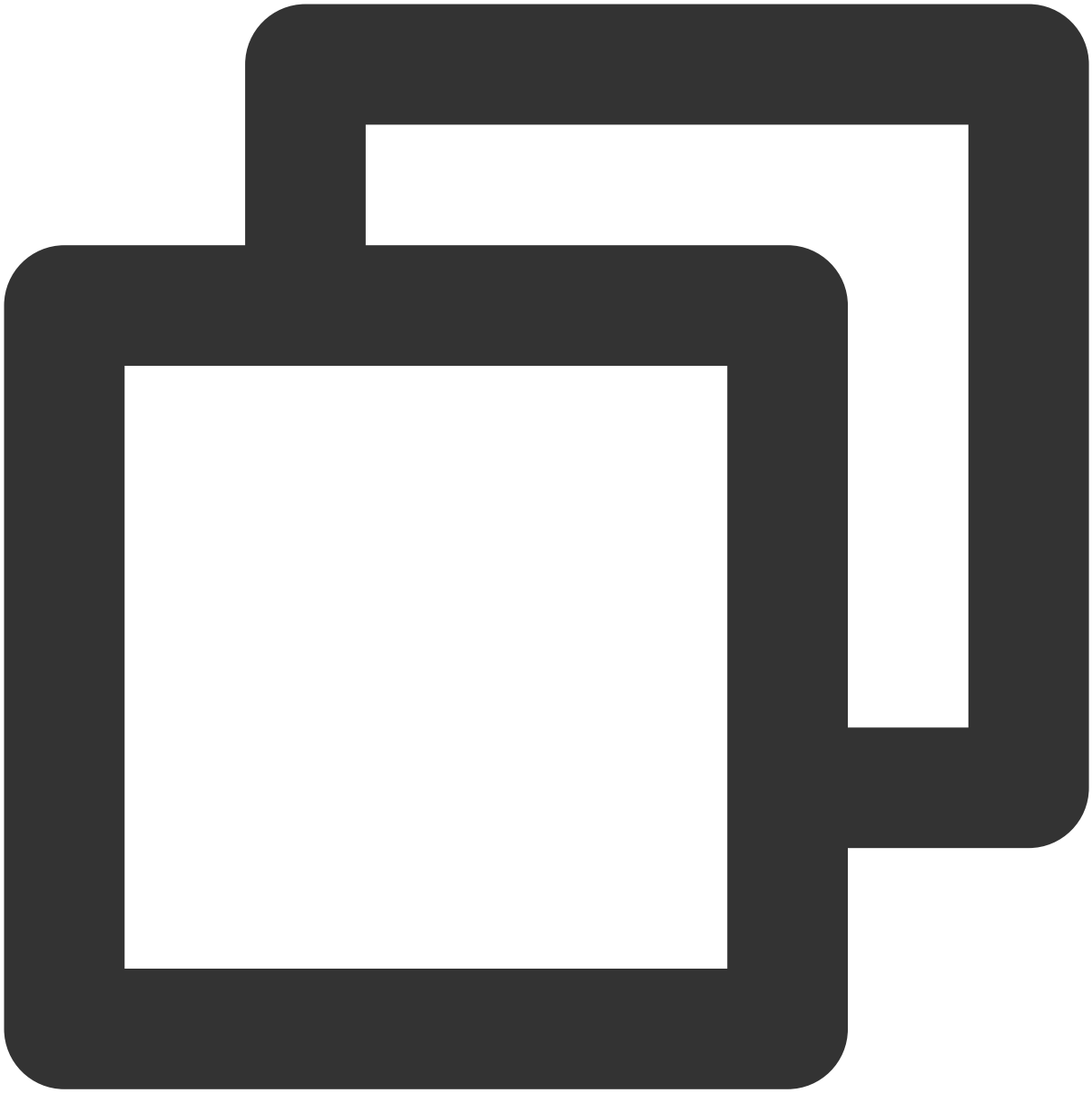
code is as follows:



```
helm upgrade --install prod ingress-nginx/ingress-nginx \\
  --namespace ingress-nginx --create-namespace \\
  -f values.yaml
```

When creating Ingress resources, you also need to specify the corresponding `ingressClassName` :

©2013-2022 Tencent Cloud. All rights reserved.



```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: nginx
spec:
  ingressClassName: prod
  rules:
    - http:
        paths:
          - path: /
            pathType: Prefix
```

©2013-2022 Tencent Cloud. All rights reserved.

```
            backend:
              service:
                name: nginx
                port:
                  number: 80
```

©2013-2022 Tencent Cloud. All rights reserved.

Migrating from TKE Nginx Ingress Plugin to Self-Built Nginx Ingress

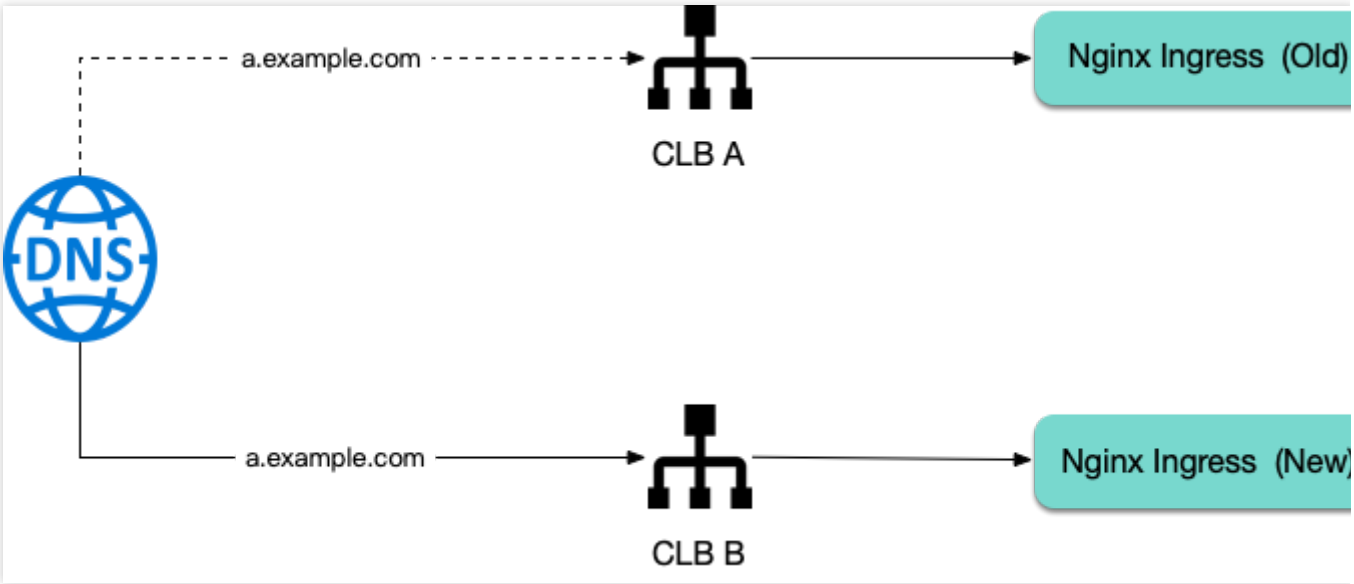
Last updated：2024-08-12 17:48:23

Benefits of Migration

Nginx Ingress offers a vast and flexible range of features and configurations that can cater to various use cases. Self-building allows you to unlock all features of Nginx Ingress, customize configurations as needed, and update versions in a timely manner.

Migration Approach

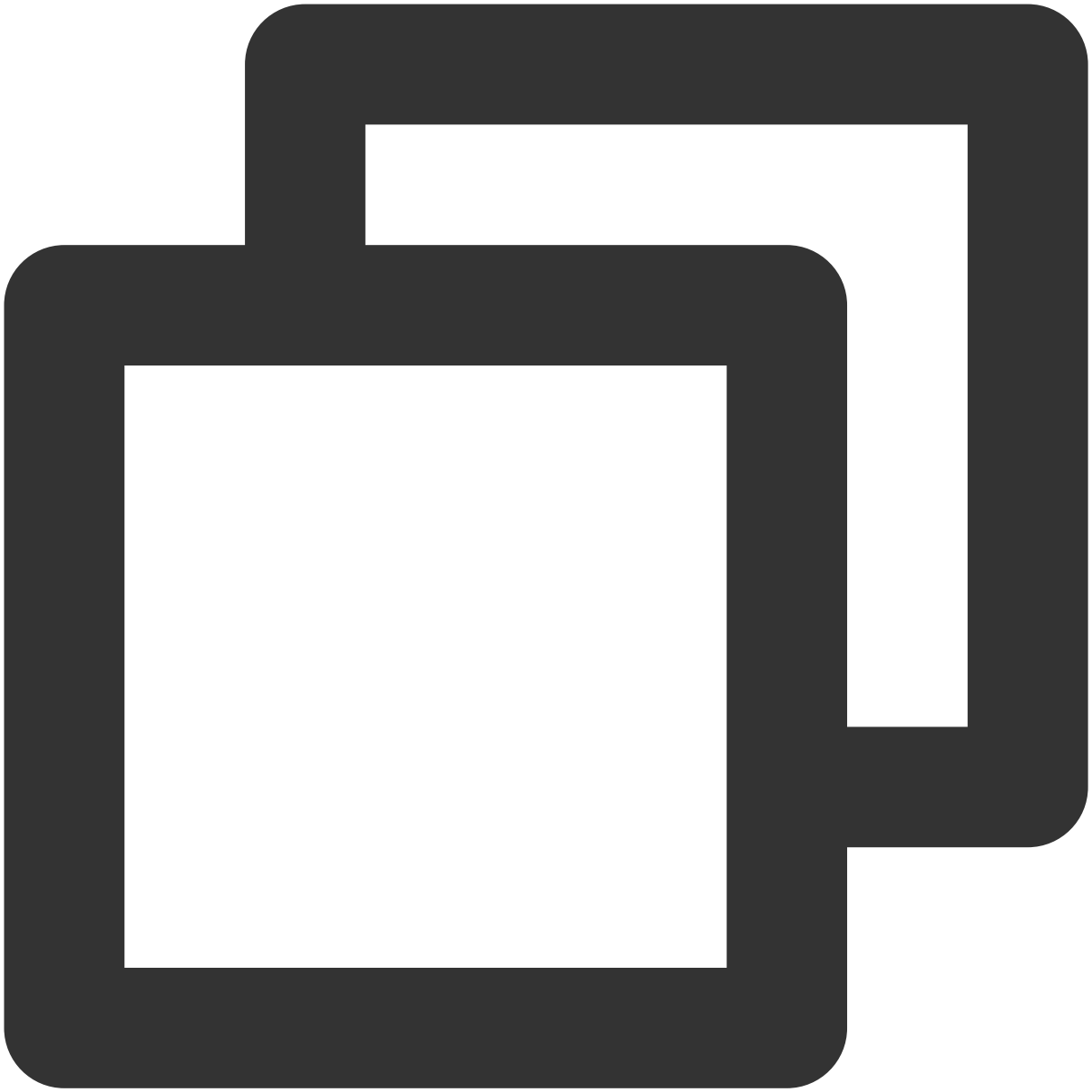
Use the self-built method described in this document to create a new Nginx Ingress instance. Share the same IngressClass with the old instance, sharing the same Ingress forwarding rules. Both traffic entries will coexist. Finally, modify the DNS to point to the new entry address for a smooth migration.



Confirming Information About the Installed Nginx Ingress

1. First, confirm the IngressClass name of the installed Nginx Ingress instance, for example:

©2013-2022 Tencent Cloud. All rights reserved.

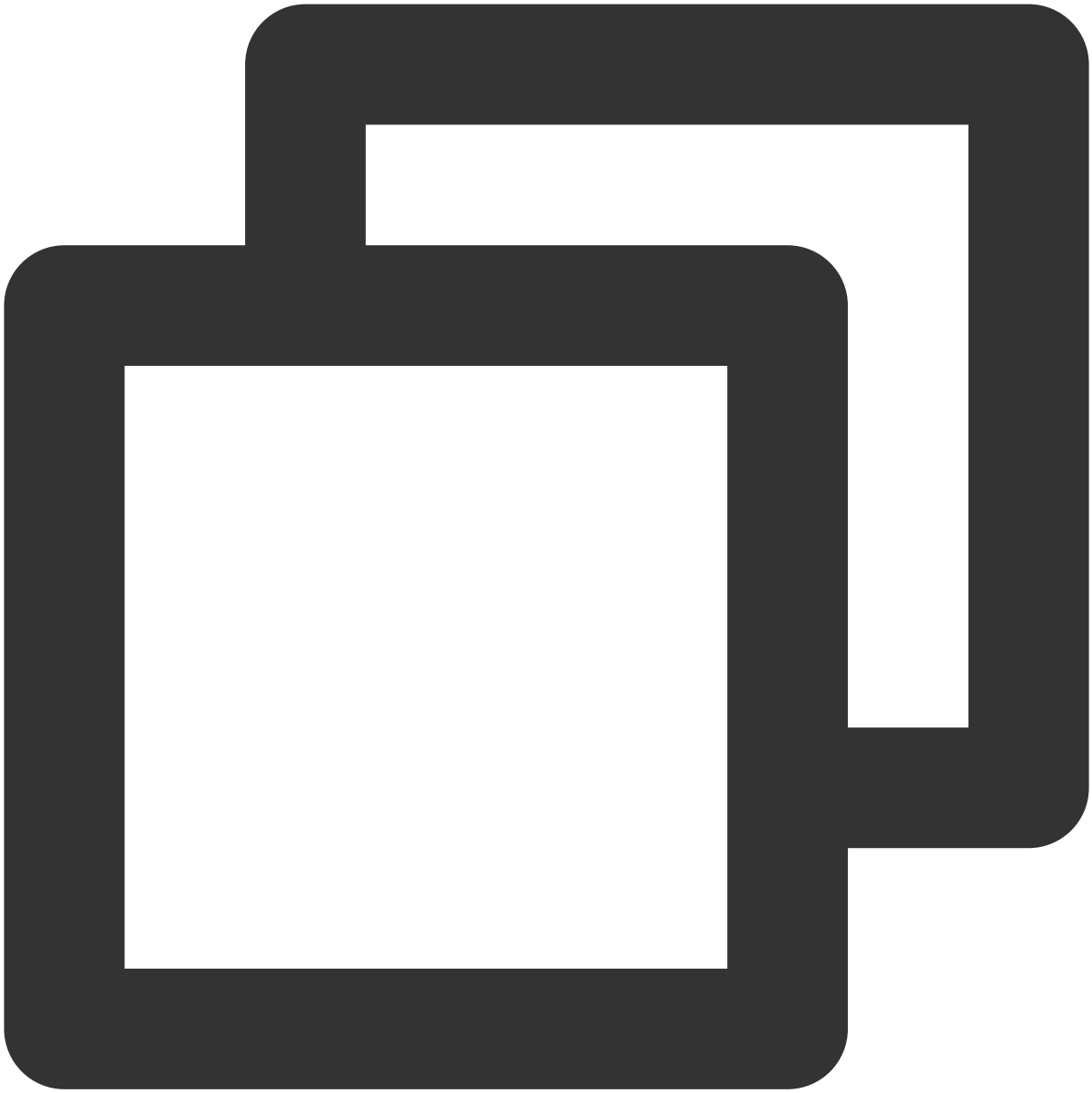


```
$ kubectl get deploy -A | grep nginx
kube-system            extranet-ingress-nginx-controller           1/1     1
```

In this example, there is only one instance. The Deployment name is `extranet-ingress-nginx-controller`, and the part before `-ingress-nginx-controller` is the IngressClass, which is `extranet`.

2. Confirm the current image version of the Nginx Ingress:

©2013-2022 Tencent Cloud. All rights reserved.

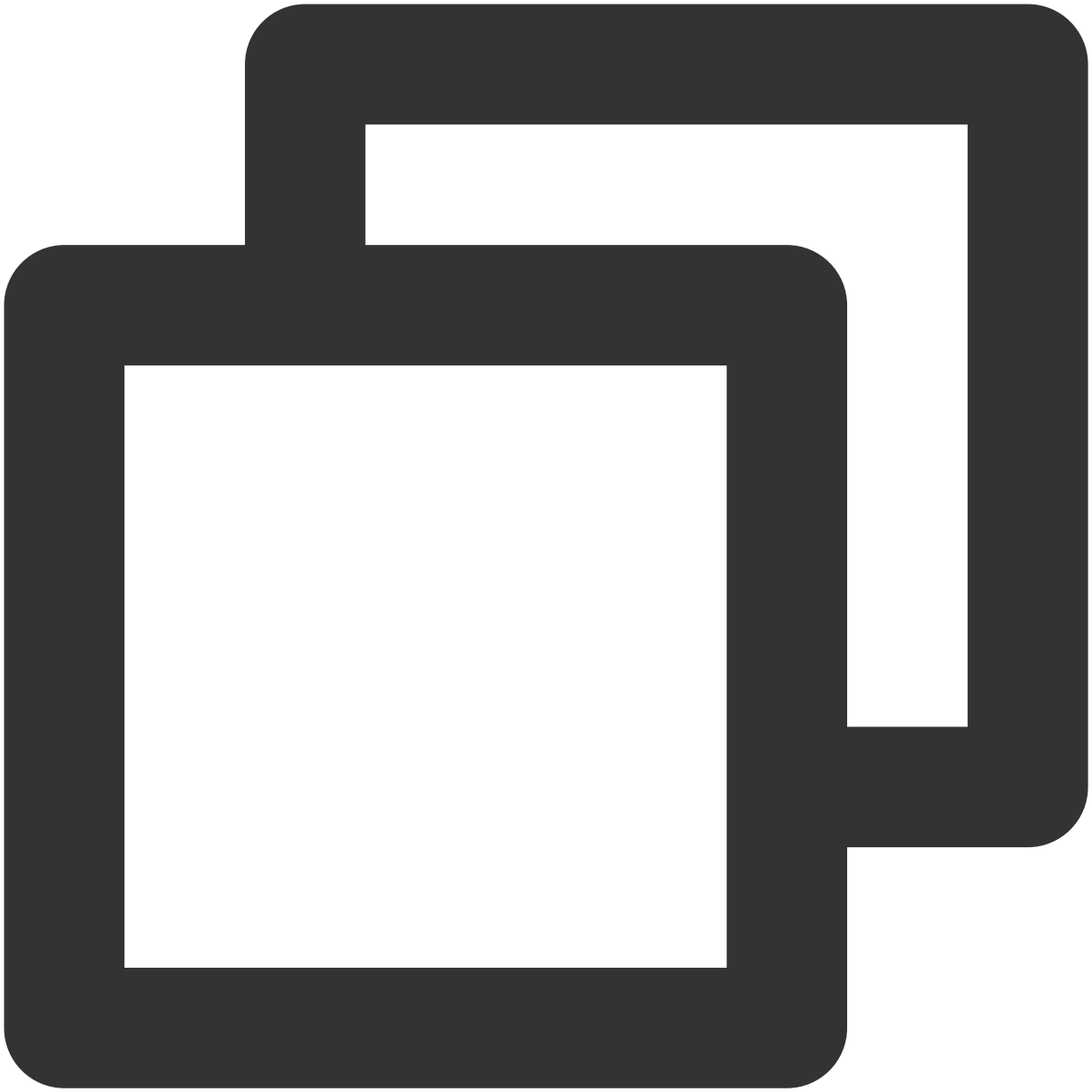


```
$ kubectl -n kube-system get deploy extranet-ingress-nginx-controller -o yaml | gre
        image: ccr.ccs.tencentyun.com/tkeimages/nginx-ingress-controller:v1.9.5
```

In this example, the image version is `v1.9.5` .

3. Confirm the current chart version:

©2013-2022 Tencent Cloud. All rights reserved.



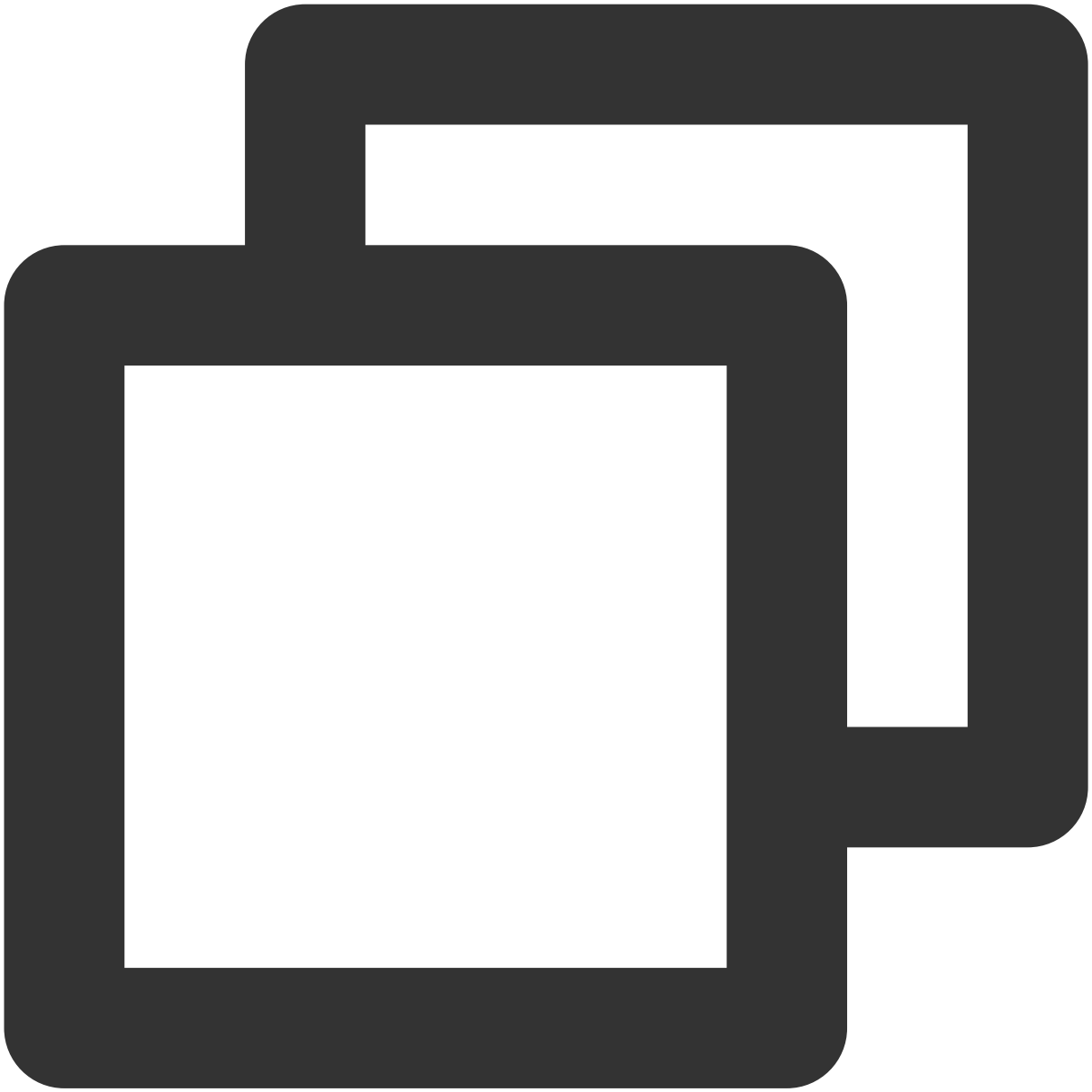
```
$ helm search repo ingress-nginx/ingress-nginx --versions  | grep 1.9.5
ingress-nginx/ingress-nginx     4.9.0           1.9.5           Ingress controller
```

In this example, the chart version is `4.9.0` . Please remember this version, as it needs to be specified when helm is used to install the new rendered version.

Preparing values.yaml

©2013-2022 Tencent Cloud. All rights reserved.

Ensure that the new Nginx Ingress instance created by helm and the Nginx Ingress instance created by the TKE plugin share the same IngressClass, so that the Ingress rules are effective on both sides simultaneously.
Check the current IngressClass definition:

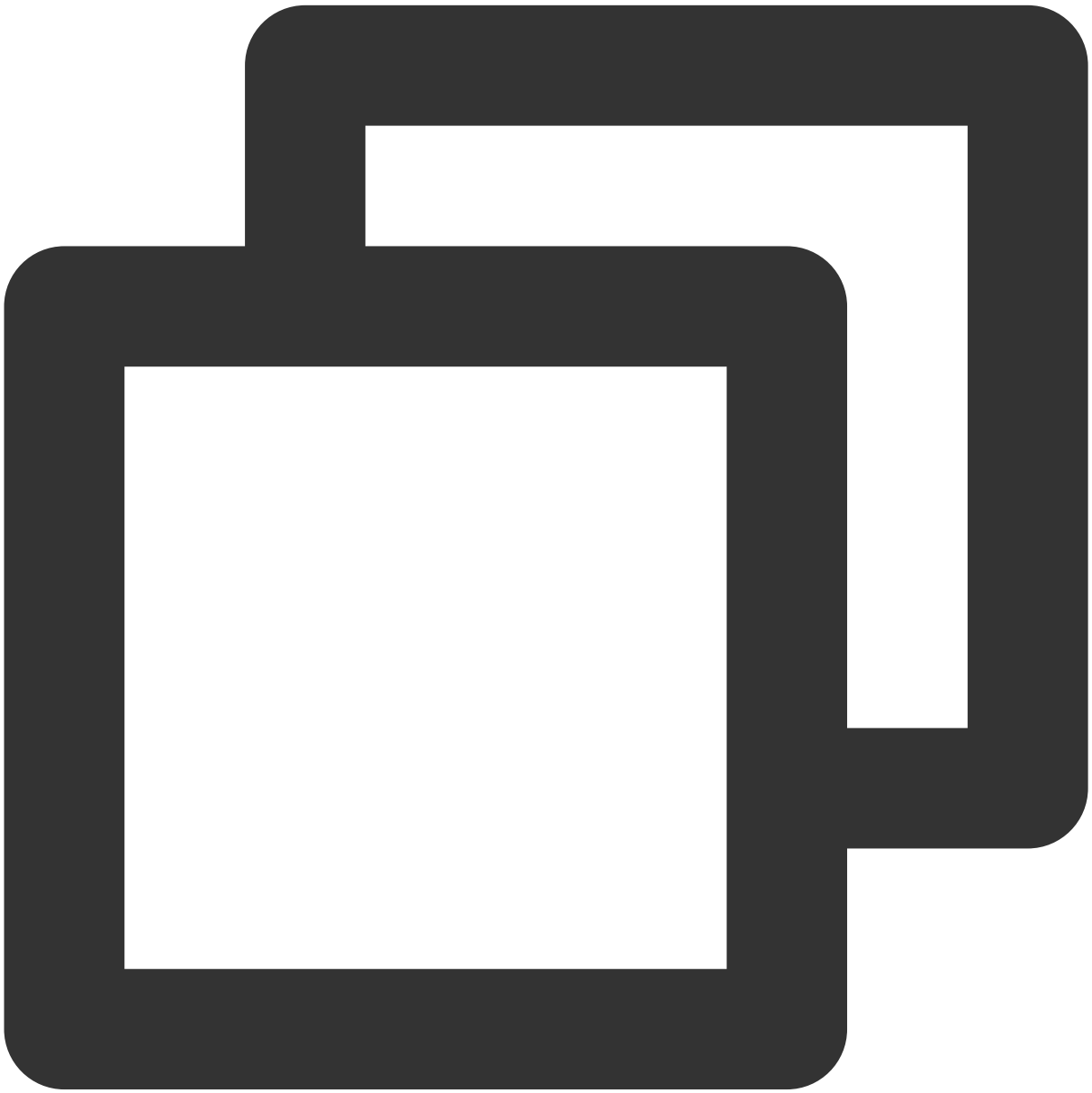


```
$ kubectl get ingressclass extranet -o yaml
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  creationTimestamp: "2024-03-27T10:47:49Z"
  generation: 1
  labels:
```

©2013-2022 Tencent Cloud. All rights reserved.

```
    app.kubernetes.io/component: controller
  name: extranet
  resourceVersion: "27703380423"
  uid: 5e2de0d1-8eae-4b55-afde-25c8fe37d478
spec:
  controller: k8s.io/extranet
```

Obtain the controller value `k8s.io/extranet` and configure it in `values.yaml` along with the IngressClass name:

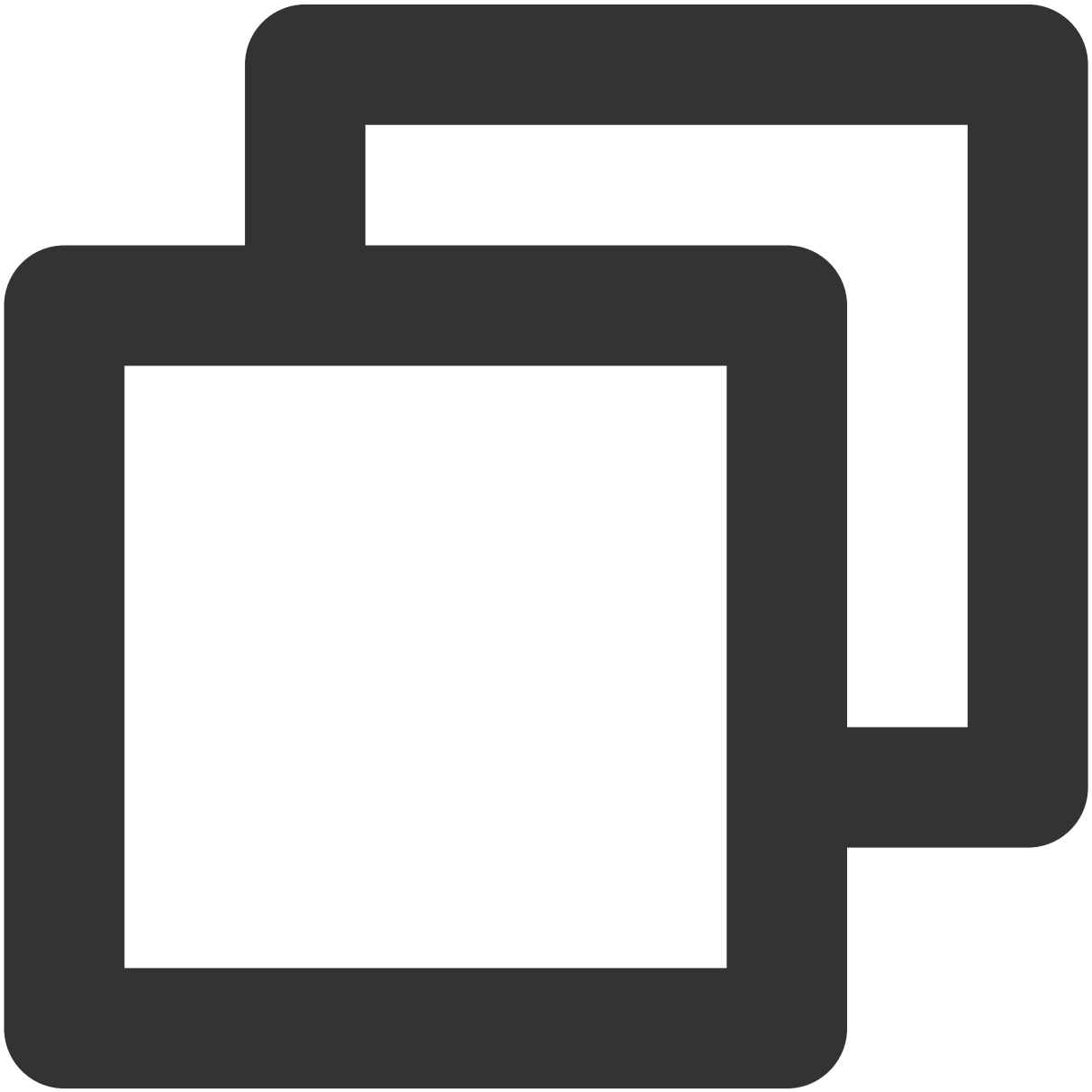


```
controller:
```

©2013-2022 Tencent Cloud. All rights reserved.

```
  ingressClassName: extranet # IngressClass name
  ingressClassResource:
    enabled: false # IngressClass resources are not automatically created to avoid
    controllerValue: k8s.io/extranet # The new Nginx Ingress reuses the existing In
```

Installing a New Nginx Ingress Controller



```
helm upgrade --install new-extranet-ingress-nginx ingress-nginx/ingress-nginx \\
```

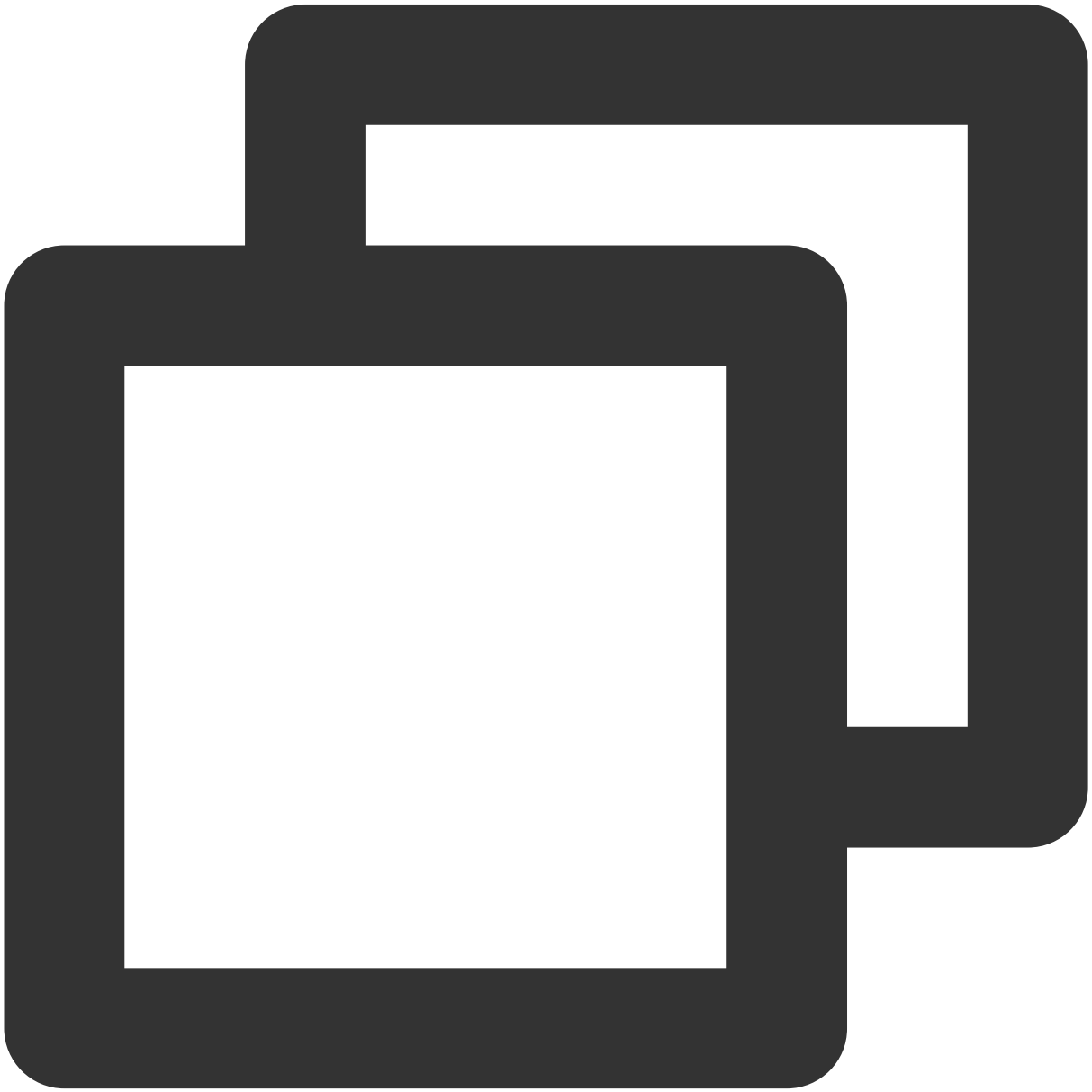
©2013-2022 Tencent Cloud. All rights reserved.

```
    --namespace ingress-nginx --create-namespace \\
    --version 4.9.0 \\
    -f values.yaml
```

Ensure that the release name will not be the same as any existing Nginx Ingress Deployment name after the suffix `-controller` is added to the release name. If a ClusterRole with the same name exists, it will cause the helm installation to fail.

Specify the version to the chart version (that is, the chart version corresponding to the current Nginx Ingress instance version) obtained in the previous steps.

Obtain the traffic entry of the new Nginx Ingress:



©2013-2022 Tencent Cloud. All rights reserved.

```
$ kubectl -n ingress-nginx get svc
NAME                                                TYPE           CLUSTER-IP       E
new-extranet-ingress-nginx-controller               LoadBalancer   172.16.165.100   4
```

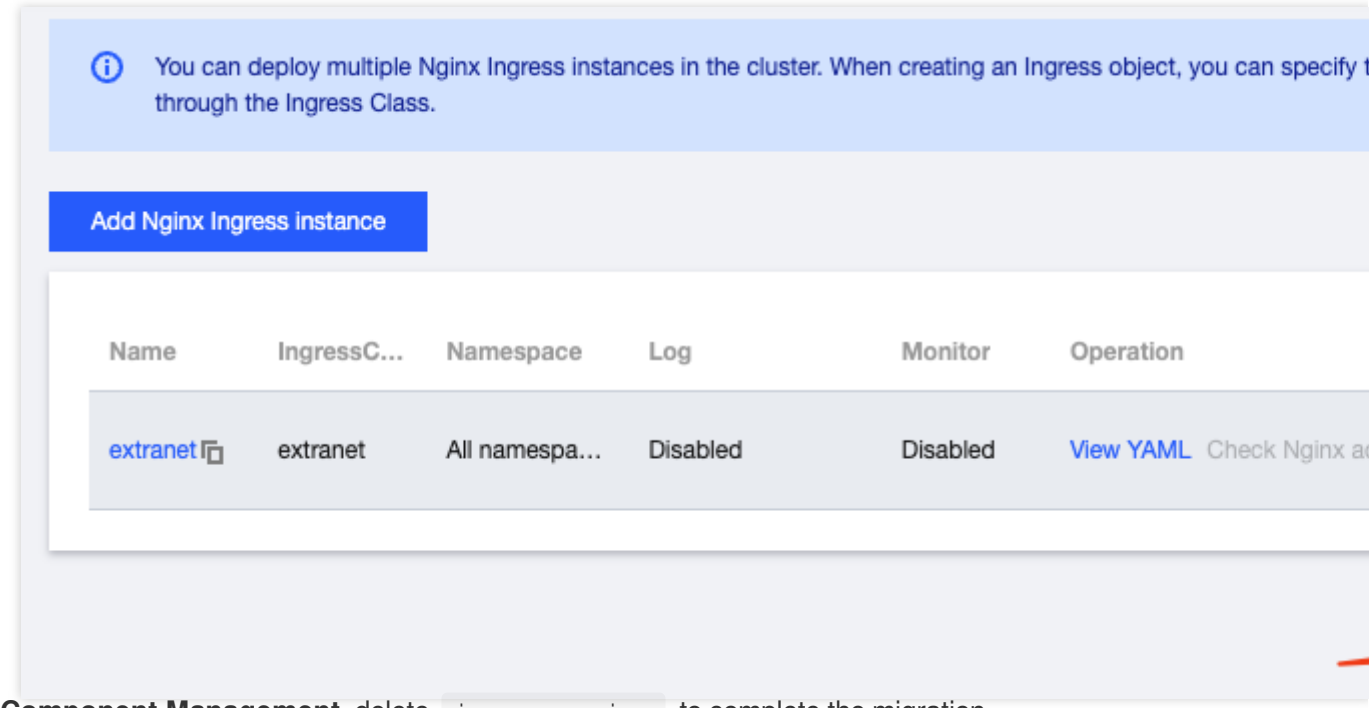
`EXTERNAL-IP` is the new traffic entry. Please verify that it can forward traffic normally.

Switching DNS

At this point, both new and old Nginx Ingress instances coexist, and traffic can be forwarded normally through either traffic entry.
Next, modify the DNS resolution for the domain name to point to the new Nginx Ingress traffic entry. Before the DNS resolution takes full effect, traffic can be forwarded normally through both entries. This process will be very smooth, and the production environment traffic will not be affected.

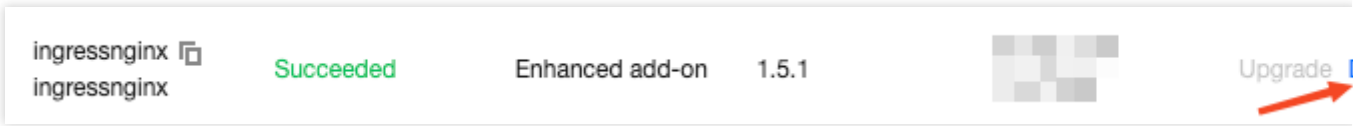
Deleting the Old NginxIngress Instance and Plugin

1. Once all traffic has been completely switched off from the old Nginx Ingress instances, go to the TKE console to delete the Nginx Ingress instance:



2. In **Component Management**, delete `ingressnginx` to complete the migration.

©2013-2022 Tencent Cloud. All rights reserved.

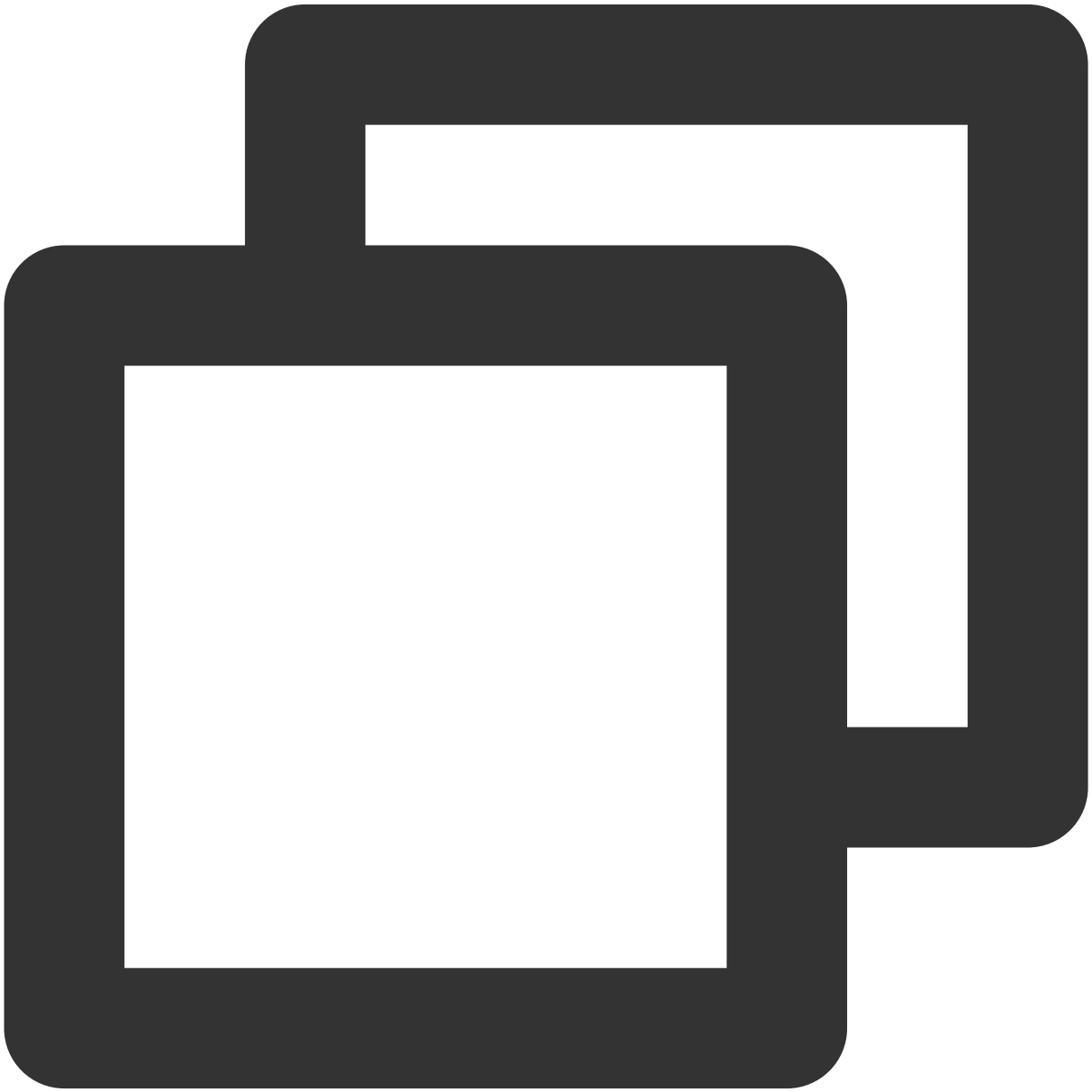


©2013-2022 Tencent Cloud. All rights reserved.

Complete Example of values.yaml Configuration

Last updated：2024-08-12 17:48:23

The following is a relatively complete example of `values.yaml` configuration. You can copy this example and modify it as needed:



```
controller:
```

©2013-2022 Tencent Cloud. All rights reserved.

```
extraInitContainers:
  - name: sysctl
    image: busybox
    securityContext:
      privileged: true
    imagePullPolicy: IfNotPresent
    command:
      - sh
      - -c
      - |
        sysctl -w net.core.somaxconn=65535 # Increase the connection queue to pre
        sysctl -w net.ipv4.ip_local_port_range="1024 65535" # Expand the source p
        sysctl -w net.ipv4.tcp_tw_reuse=1 # Enable TIME_WAIT reuse to allow new c
        sysctl -w fs.file-max=1048576 # Increase the file handle count to prevent
config:
  # The number of requests that can be processed by a persistent connection betwe
  # Reference: https://kubernetes.github.io/ingress-nginx/user-guide/nginx-config
  keep-alive-requests: "1000"
  # The maximum number of idle persistent connections (not the maximum number of
  # Reference: https://kubernetes.github.io/ingress-nginx/user-guide/nginx-config
  upstream-keepalive-connections: "2000"
  # The maximum number of connections that each worker process can open is 16384
  # Reference: https://kubernetes.github.io/ingress-nginx/user-guide/nginx-config
  max-worker-connections: "65536"
  log-format-upstream: $remote_addr - $remote_user [$time_local] "$request"
    $status $body_bytes_sent "$http_referer" "$http_user_agent"
    $request_length $request_time [$proxy_upstream_name] [$proxy_alternative_upst
    $upstream_response_length $upstream_response_time $upstream_status $req_id $h
  # Nginx logs are written to log files to avoid high CPU usage under high concur
  access-log-path: /var/log/nginx/nginx_access.log
  error-log-path: /var/log/nginx/nginx_error.log
extraEnvs:
  - name: TZ
    value: Asia/Shanghai
extraVolumes:
  - name: log
    emptyDir: {}
extraVolumeMounts:
  - name: log
    mountPath: /var/log/nginx
extraContainers:
  - name: logrotate
    image: imroc/logrotate:latest
    imagePullPolicy: Always
    env:
      - name: LOGROTATE_FILE_PATTERN # Pattern of rotated log files, matching the
        value: "/var/log/nginx/nginx_*.log"
```

©2013-2022 Tencent Cloud. All rights reserved.

```
        - name: LOGROTATE_FILESIZE # Threshold of log file size for rotation
          value: "100M"
        - name: LOGROTATE_FILENUM # Number of rotations per log file
          value: "3"
        - name: CRON_EXPR # Crontab expression for periodic logrotate running, whic
          value: "*/1 * * * *"
        - name: CROND_LOGLEVEL # Crond log level, ranging from 0 to 8, the smaller
          value: "8"
      volumeMounts:
        - name: log
          mountPath: /var/log/nginx
  podAnnotations:
    tke.cloud.tencent.com/networks: tke-route-eni # Declaration of the using the VP
  resources: # Declaration of using ENI in resources
    requests:
      tke.cloud.tencent.com/eni-ip: "1"
    limits:
      tke.cloud.tencent.com/eni-ip: "1"
  service:
    annotations:
      service.cloud.tencent.com/direct-access: "true" # Enabling of CLB direct acce
  autoscaling:
    enabled: true
    minReplicas: 1
    maxReplicas: 10
    targetCPUUtilizationPercentage: 50
    targetMemoryUtilizationPercentage: 50
    behavior: # Quick scale-out to handle traffic peaks, slow scale-in to leave a b
      scaleDown:
        stabilizationWindowSeconds: 300
        policies:
          - type: Percent
            value: 900
            periodSeconds: 15 # Allowing scale-out up to 9 times the current number
      scaleUp:
        stabilizationWindowSeconds: 300
        policies:
          - type: Pods
            value: 1
            periodSeconds: 600 # Allowing scale-in of only one pod at most every 10
  topologySpreadConstraints: # Policy to maximize spreading
    - labelSelector:
        matchLabels:
          app.kubernetes.io/name: '{{ include "ingress-nginx.name" . }}'
          app.kubernetes.io/instance: "{{ .Release.Name }}"
          app.kubernetes.io/component: controller
      topologyKey: topology.kubernetes.io/zone
```

©2013-2022 Tencent Cloud. All rights reserved.

```
      maxSkew: 1
      whenUnsatisfiable: ScheduleAnyway
    - labelSelector:
        matchLabels:
          app.kubernetes.io/name: '{{ include "ingress-nginx.name" . }}'
          app.kubernetes.io/instance: "{{ .Release.Name }}"
          app.kubernetes.io/component: controller
      topologyKey: kubernetes.io/hostname
      maxSkew: 1
      whenUnsatisfiable: ScheduleAnyway
  image:
    registry: docker.io
    image: k8smirror/ingress-nginx-controller
  admissionWebhooks:
    patch:
      image: # The default image cannot be pulled in Chinese mainland, which can be
        registry: docker.io
        image: k8smirror/ingress-nginx-kube-webhook-certgen
  defaultBackend:
    image: # The default image cannot be pulled in Chinese mainland, which can be r
      registry: docker.io
      image: k8smirror/defaultbackend-amd64
  opentelemetry:
    image: # The default image cannot be pulled in Chinese mainland, which can be r
      registry: docker.io
      image: k8smirror/ingress-nginx-opentelemetry
```

©2013-2022 Tencent Cloud. All rights reserved.

Limiting the bandwidth on pods in TKE

Last updated：2022-03-23 18:17:29

Overview

This document describes how to restrict the Pod bandwidth in TKE. Currently, TKE does not support Pod speed restriction; however, you can modify the CNI plugin to achieve it based on your actual scenario.

Notes

- TKE supports using the bandwidth plugin of the community to restrict the network speed. Currently, it can be used in GlobalRouter mode and VPC-CNI shared ENI mode.
- Currently, it is not supported for the VPC-CNI dedicated ENI mode.

Directions

Modifying CNI plugin

GlobalRouter mode

The GlobalRouter network mode is a routing policy for communication between the container network and VPC based on the global routing capabilities of the underlying VPC. It is suitable for common scenarios and seamlessly compatible with standard Kubernetes features. For more information, see GlobalRouter Mode.

1. Log in to the Pod node as instructed in Logging in to Linux Instance Using Standard Login Method.
2. Run the following command to view the configuration of `tke-bridge-agent`:

```
kubectl edit daemonset tke-bridge-agent -n kube-system
```

Add args `--bandwidth` to enable the support for the bandwidth plugin.

VPC-CNI shared ENI mode

The VPC-CNI mode is container network capability implemented based on CNI and VPC ENI and is suitable for scenarios with high requirements for latency.
The open-source bandwidth plugin supports traffic shaping at the Pod entry and exit as well as bandwidth control. For more information, see VPC-CNI Mode.

©2013-2022 Tencent Cloud. All rights reserved.

1. Log in to the Pod node as instructed in Logging in to Linux Instance Using Standard Login Method.
2. Run the following command to view the configuration of `tke-eni-agent` :

```
kubectl edit daemonset tke-eni-agent -n kube-system
```

Add args `--bandwidth` to enable the support for the bandwidth plugin.

> Note：
> You can enable this feature simply by adding the above parameters to `tke-eni-agent` and disable it by removing the parameters. Deployment, enablement, and disablement are supported, which take effect only for new Pods.

Specifying annotation in Pod

You can configure in the method provided by the community:

- Use the `kubernetes.io/ingress-bandwidth` annotation to specify the inbound bandwidth cap.
- Use the `kubernetes.io/egress-bandwidth` annotation to specify the outbound bandwidth cap.

Sample:

```
apiVersion: apps/v1
kind: Deployment
metadata:
name: nginx
spec:
replicas: 1
selector:
matchLabels:
app: nginx
template:
metadata:
labels:
app: nginx
annotations:
kubernetes.io/ingress-bandwidth: 10M
kubernetes.io/egress-bandwidth: 20M
spec:
containers:
- name: nginx
image: nginx
```

©2013-2022 Tencent Cloud. All rights reserved.

Configuration Verification

You can verify whether the configuration succeeds in the following two methods:

- **Method 1:** log in to the Pod node and run the following command to check whether the caps have been added:

```
tc qdisc show
```

If a result similar to the following is returned, the caps have been added successfully:

```
qdisc tbf 1: dev vethc09123a1 root refcnt 2 rate 10Mbit burst 256Mb lat 25.0ms
qdisc ingress ffff: dev vethc09123a1 parent ffff:fff1 ----------------
qdisc tbf 1: dev 6116 root refcnt 2 rate 20Mbit burst 256Mb lat 25.0ms
```

- **Method 2:** run the following command to use iperf for testing:

```
iperf -c <service IP> -p <service port> -i 1
```

If a result similar to the following is returned, the caps have been added successfully:

```
------------------------------------------------------------
Client connecting to 172.16.0.xxx, TCP port 80
TCP window size: 12.0 MByte (default)
------------------------------------------------------------
[ 3] local 172.16.0.xxx port 41112 connected with 172.16.0.xx port 80
[ ID] Interval Transfer Bandwidth
[ 3] 0.0- 1.0 sec 257 MBytes 2.16 Gbits/sec
[ 3] 1.0- 2.0 sec 1.18 MBytes 9.90 Mbits/sec
[ 3] 2.0- 3.0 sec 1.18 MBytes 9.90 Mbits/sec
[ 3] 3.0- 4.0 sec 1.18 MBytes 9.90 Mbits/sec
[ 3] 4.0- 5.0 sec 1.18 MBytes 9.90 Mbits/sec
[ 3] 5.0- 6.0 sec 1.12 MBytes 9.38 Mbits/sec
[ 3] 6.0- 7.0 sec 1.18 MBytes 9.90 Mbits/sec
[ 3] 7.0- 8.0 sec 1.18 MBytes 9.90 Mbits/sec
[ 3] 8.0- 9.0 sec 1.18 MBytes 9.90 Mbits/sec
[ 3] 9.0-10.0 sec 1.12 MBytes 9.38 Mbits/sec
[ 3] 0.0-10.3 sec 268 MBytes 218 Mbits/se
```

©2013-2022 Tencent Cloud. All rights reserved.

Directly connecting TKE to the CLB of pods based on the ENI

Last updated：2023-05-24 10:17:46

Overview

Kubernetes designs and provides two types of native resources at the cluster access layer, 'Service' and 'Ingress', which are responsible for the network access layer configurations of layer 4 and layer 7, respectively. The traditional solution is to create an Ingress- or LoadBalancer-type service to bind Tencent Cloud CLBs and open services to the public. In this way, user traffic is loaded on the NodePort of the user node, and then forwarded to the container network through the KubeProxy component. This solution has some limitations in business performance and capabilities.

To address these limitations, the Tencent Cloud TKE team **provides a new network mode for users who use independent or managed clusters. that is, TKE directly connects to the CLB of pods based on the ENI**. This mode provides enhanced performance and business capabilities. This document describes the differences between the two modes and how to use the direct connection mode.

Solution Comparison

Comparison Item	Direct Connection	NodePort Forwarding	Local Forwarding
Performance	Zero loss	NAT forwarding and inter-node forwarding	Minor loss
Pod update	The access layer backend automatically synchronizes updates, so the update process is stable	The access layer backend NodePort remains unchanged	Services may be interrupted without update synchronization
Cluster dependency	Cluster version and VPC-CNI network requirements	-	-
Business capability restriction	Least restriction	Unable to obtain the source IP address or implement session persistence	Conditional session persistence

©2013-2022 Tencent Cloud. All rights reserved.

Analysis of Problems with the Traditional Mode

Performance and features

In a cluster, `KubeProxy` forwards the traffic from user `NodePort` through NAT to the cluster network. This process has the following problems:

- NAT forwarding causes certain loss in request performance.
- NAT operations cause performance loss.
- The destination address of NAT forwarding may cause the traffic to be forwarded across nodes in a container network.
- NAT forwarding changes the source IP address of the request, so the client cannot obtain the source IP address.
- When the CLB traffic is concentrated on several NodePorts, the over-concentrated traffic will cause excessive SNAT forwarding by NodePorts, which will exhaust the traffic capacity of the port. This problem may also lead to conntrack insertion conflicts, resulting in packet loss and performance deterioration.
- Forwarding by `KubeProxy` is random and does not support session persistence.
- Each NodePort of `KubeProxy` has independent load-balancing capabilities. As such capabilities cannot be concentrated in one place, global load balancing is difficult to achieve.

To address the preceding problems, the technical suggestion previously provided to users was to adopt local forwarding to avoid the problems caused by `KubeProxy` NAT forwarding. However, due to the randomness of forwarding, session persistence remains unsupported when multiple replicas are deployed on a node. Moreover, when local forwarding coincides with rolling updates, services can be easily interrupted. This places higher requirements on the rolling update policies and downtime of businesses.

Service availability

When a service is accessed through NodePorts, the design of NodePorts is highly fault-tolerant. The CLB binds the NodePorts of all nodes in the cluster as the backend. When any node of the cluster accesses the service, the traffic will be randomly allocated to the workloads of the cluster. Therefore, the unavailability of NodePorts or pods does not affect the traffic access of the service.

Similar to local access, in cases where the backend of the CLB is directly connected to user pods, if the CLB cannot be promptly bound to the new pod when the service is processing a rolling update, the number of CLB backends of the service entry may be seriously insufficient or even exhausted as a result of rapid rolling updates. Therefore, when the service is processing a rolling update, the security and stability of the rolling update can be ensured if the CLB of the access layer is healthy.

CLB control plane performance

©2013-2022 Tencent Cloud. All rights reserved.

The control plane APIs of the CLB include APIs for creating, deleting, and modifying layer-4 and layer-7 listeners, creating and deleting layer-7 rules, and binding each listener or the rule backend. Most of these APIs are asynchronous APIs, which require the polling of request results, and API calls are time-consuming. When the scale of the user cluster is large, the synchronization of a large amount of access layer resources can impose high latency pressure on components.

Comparison of the New and Old Modes

Performance comparison

TKE has launched the direct pod connection mode, which optimizes the control plane of the CLB. In the overall synchronization process, this new mode mainly optimizes batch calls and backend instance queries where remote calls are relatively frequent. **After the optimization, the performance of the control plane in a typical ingress scenario is improved by 95% to 97% compared with the previous version.** At present, the synchronization time is mainly the waiting time of asynchronous APIs.

Backend node data surge

For cluster scaling, the relevant data is as follows:

Layer-7 Rule Quantity	Cluster Node Quantity	Cluster Node Quantity (Update)	Performance Before Optimization (s)	Optimized Batch Calling Performance (s)	Re-optimized Backend Instance Query Performance (s)	Time Consumption Reduction (%)
200	1	10	1313.056	227.908	31.548	97.597%
200	1	20	1715.053	449.795	51.248	97.011%
200	1	30	2826.913	665.619	69.118	97.555%
200	1	40	3373.148	861.583	90.723	97.310%
200	1	50	4240.311	1085.03	106.353	97.491%

Layer-7 rule data surge

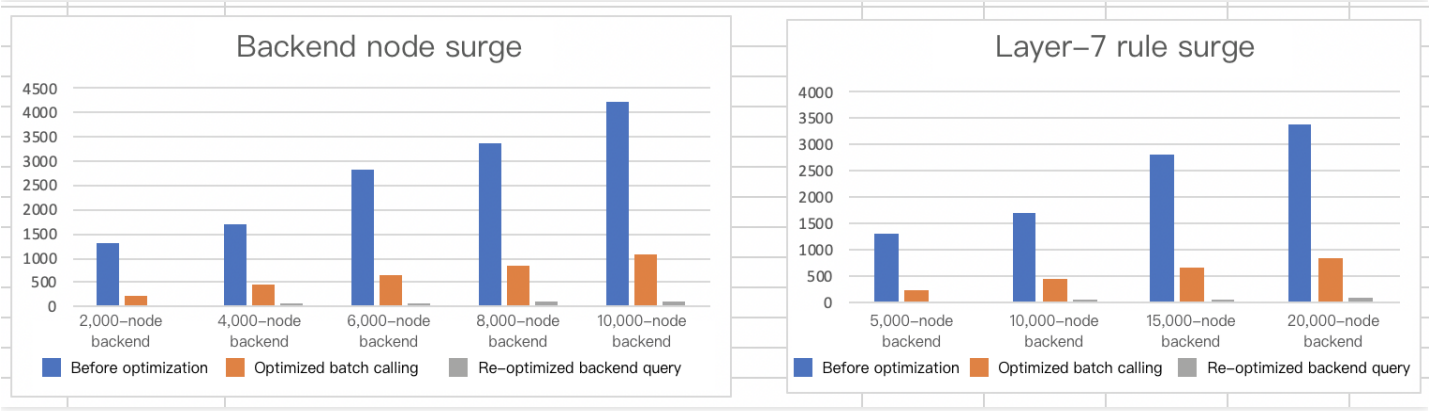
For first-time activation and deployment of services in the cluster, the relevant data is as follows:

Layer-7 Rule Quantity	Layer-7 Rule Quantity (Update)	Cluster Node Quantity	Performance Before Optimization (s)	Optimized Batch Calling Performance (s)	Re-optimized Backend Instance Query Performance (s)	Time Consumption Reduction (%)

©2013-2022 Tencent Cloud. All rights reserved.

Layer-7 Rule Quantity	Layer-7 Rule Quantity (Update)	Cluster Node Quantity	Performance Before Optimization (s)	Optimized Batch Calling Performance (s)	Re-optimized Backend Instance Query Performance (s)	Time Consumption Reduction (%)
1	100	50	1631.787	451.644	68.63	95.79%
1	200	50	3399.833	693.207	141.004	95.85%
1	300	50	5630.398	847.796	236.91	95.79%
1	400	50	7562.615	1028.75	335.674	95.56%

The following figure shows the comparison:



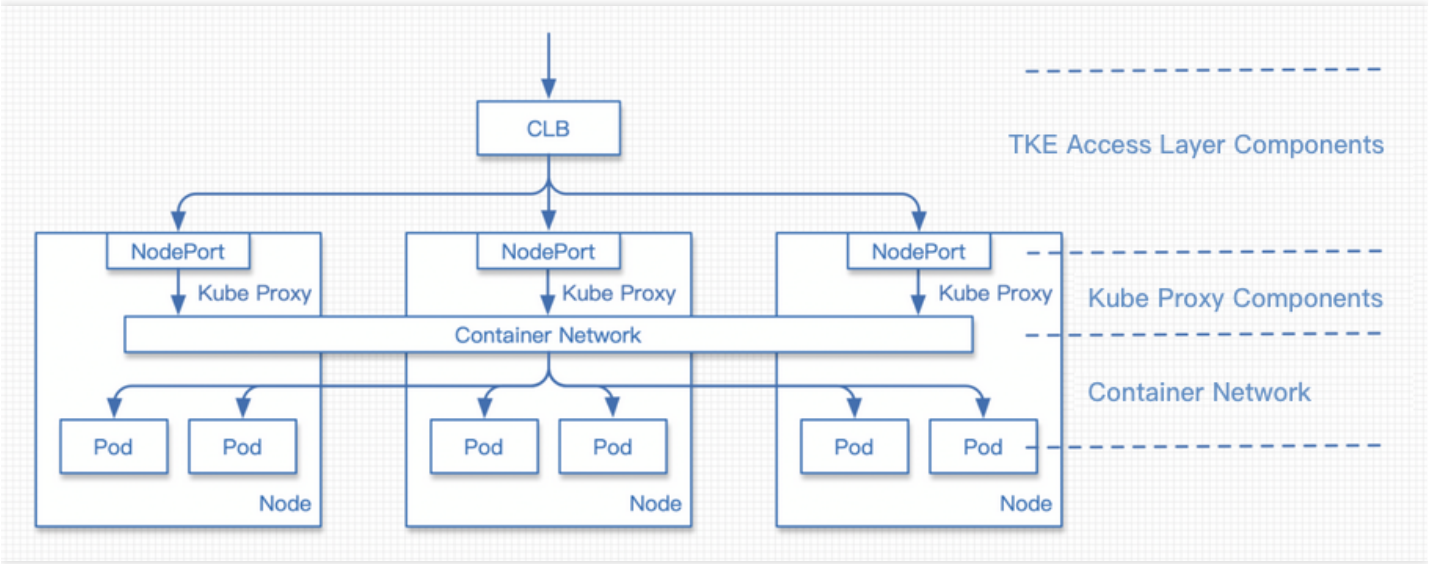
In addition to control plane performance optimization, the CLB can directly access the pods of the container network, which is the integral part of component business capabilities. This not only prevents the loss of NAT forwarding performance, but also eliminates the impact of NAT forwarding on the business features in the cluster. However, the support for optimal access to the container network remains unavailable when the project is launched.
The new mode integrates the feature that allows pods to have an ENI entry under the cluster CNI network mode in order to implement direct access to the CLB. CCN solutions are already available for implementing direct CLB backend access to the container network.

In addition to the capability of direct access, availability during rolling updates must be ensured. To implement this, we use the official feature `ReadinessGate`, which was officially released in version 1.12 and is mainly used to control the conditions of pods.
By default, a pod has three possible conditions: PodScheduled, Initialized, and ContainersReady. When the state of all pods is Ready, Pod Ready also becomes ready. However, in cloud-native scenarios, the status of pods needs to be determined in combination with other factors. `ReadinessGate` allows us to add fences for pod status determination so that the pod status can be determined and controlled by a third party, and the pod status can be associated with a third party.

©2013-2022 Tencent Cloud. All rights reserved.

CLB traffic comparison

Traditional NodePort mode

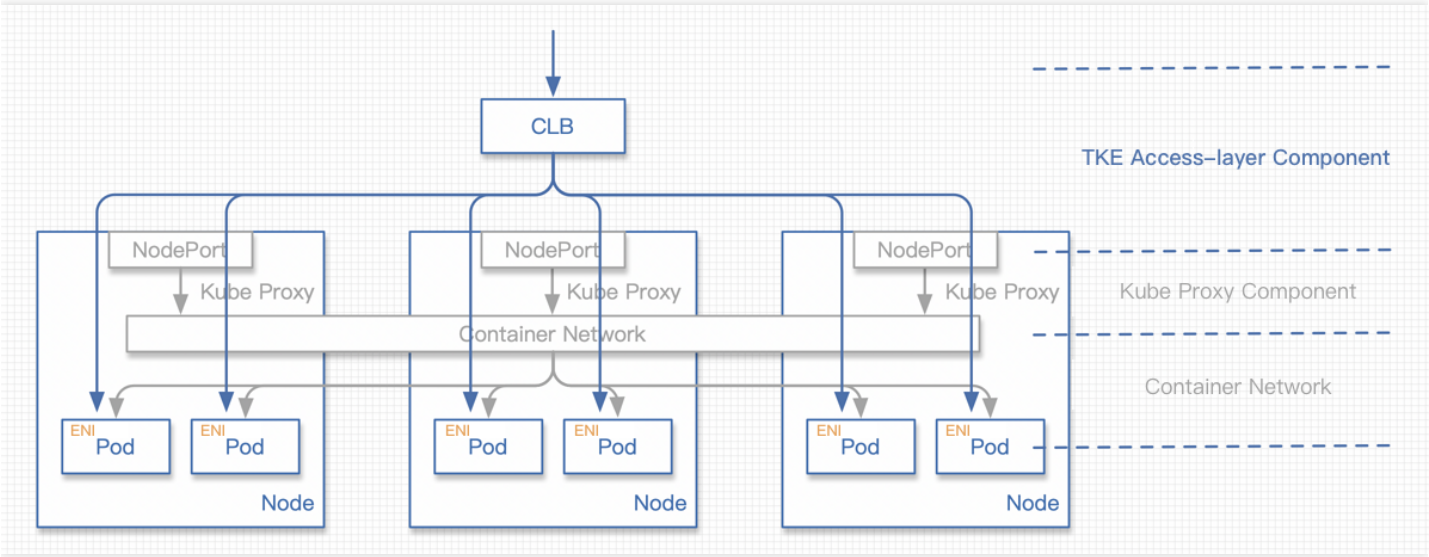


The request process is as follows:

1. The request traffic reaches the CLB.
2. The request is forwarded by the CLB to the NodePort of a certain node.
3. KubeProxy performs NAT forwarding for the traffic from the NodePort, with the destination address being the IP address of a random pod.
4. The request reaches the container network and is then forwarded to the corresponding node based on the pod address.
5. The request reaches the node to which the destination pod belongs and is then forwarded to the pod.

New direct pod connection mode

©2013-2022 Tencent Cloud. All rights reserved.



The request process is as follows:

1. The request traffic reaches the CLB.
2. The request is forwarded by the CLB to the ENI of a certain pod.

Differences between direct connection and local access

- There is little difference in terms of performance. When local access is enabled, traffic is not subject to NAT operations or cross-node forwarding, and only another route to the container network is added.
- The source IP address can be obtained correctly without NAT operations. The session persistence feature may be abnormal in this condition: when multiple pods exist on a node, traffic is randomly allocated to different pods. This mechanism may cause session persistence problems.

Introduction of ReadinessGate

Issues related to rolling updates

To introduce ReadinessGate, the cluster version must be 1.12 or later.
When users start the rolling update of an app, `Kubernetes` performs the rolling update according to the update policy. However, the identifications that it uses to determine whether a batch of pods has started only includes the statuses of the pods, but does not consider whether a health check is configured for the pods in the CLB and the pods have passed the check. If such pods cannot be scheduled in time when the access layer components experience a heavy load, the pods that have successfully completed the rolling update may not be providing services to external users, resulting in service interruption.
In order to associate the backend status of the CLB and rolling update, the new feature `ReadinessGate`, which was introduced in Kubernetes 1.12, was introduced into the TKE access-layer components. With this feature, only after the TKE access-layer components confirm that the backend binding is successful and the health check is

©2013-2022 Tencent Cloud. All rights reserved.

passed, will the state of `ReadinessGate` be configured to enable the pods to enter the Ready state, thus facilitating the rolling update of the entire workload.

Using ReadinessGate in a cluster

Kubernetes clusters provide a service registration mechanism. With this mechanism, you only need to register your services to a cluster as `MutatingWebhookConfigurations` resources. When a pod is created, the cluster will deliver notifications to the configured callback path. At this time, the pre-creation operation can be performed for the pod, that is, `ReadinessGate` can be added to the pod.

> Note：
> This callback process must be based on HTTPS. That is, the CA that issues requests must be configured in `MutatingWebhookConfigurations` , and a certificate issued by the CA must be configured on the server.

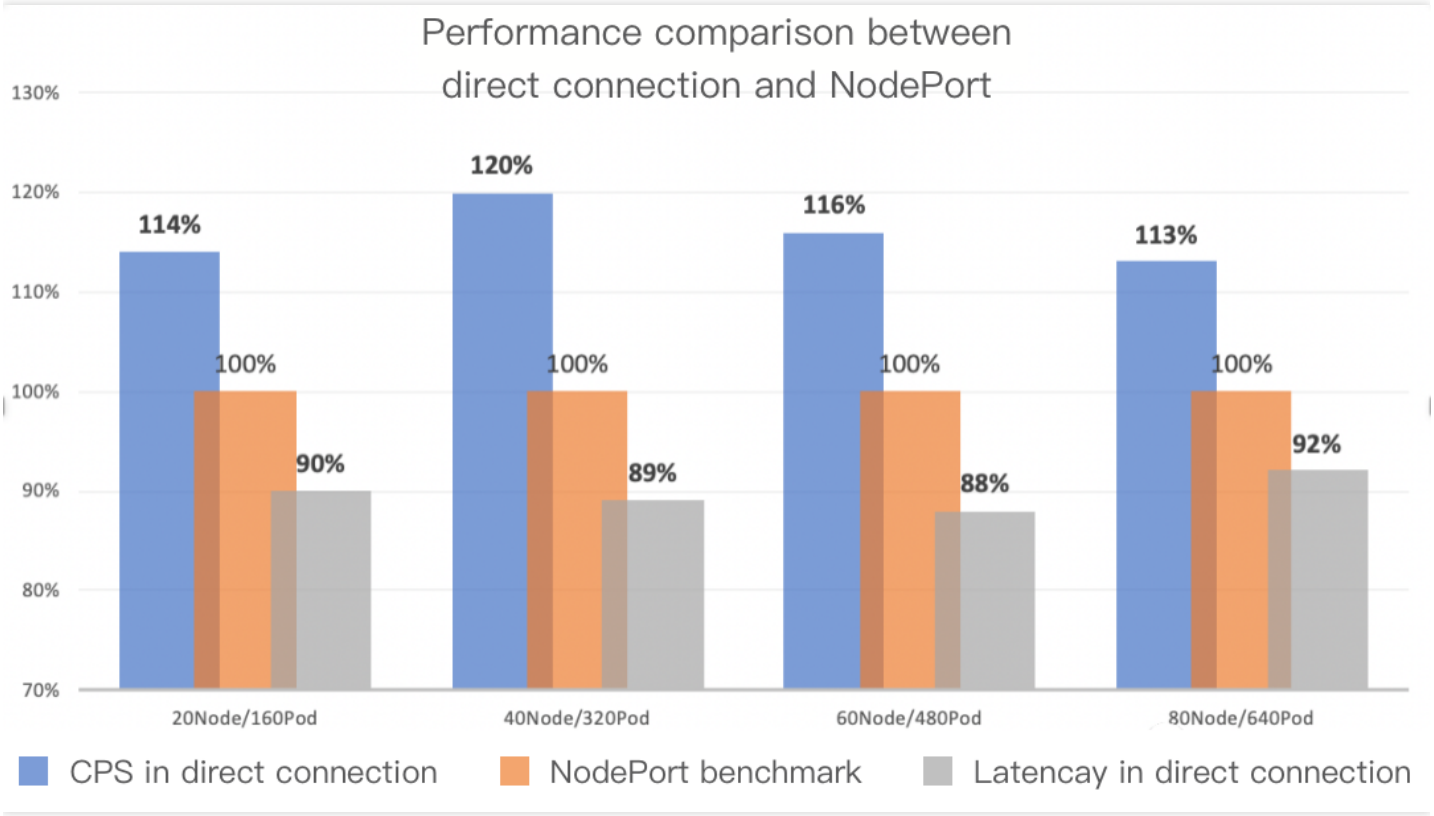
Disaster recovery of the ReadinessGate mechanism

Service registration or certificates in user clusters may be deleted by users, but these system component resources should not be modified or destroyed by users. However, such problems will inevitably occur due to users' exploration of clusters or improper operations.
The access layer components will check the integrity of the resources above during launch. If their integrity is compromised, the components will rebuild these resources to enhance the robustness of the system.

QPS and network latency comparison

Direct connection and NodePorts are the access layer solutions for service applications. In fact, the workloads deployed by users are the ultimate workers, and therefore the capabilities of user workloads directly determine the QPS and other metrics of services.
For these two access-layer solutions, we performed some comparative tests on network link latency under low workload pressure. The latency of direct connection on the network link of the access layer can be reduced by 10%, and traffic in the VPC network was greatly reduced. During the tests, the cluster size was gradually increased from 20 nodes to 80 nodes, and the wrk tool was used to test the network latency of the cluster. The comparison of QPS and network latency between direct connection and NodePorts is shown in the following figure:

©2013-2022 Tencent Cloud. All rights reserved.



KubeProxy design ideas

`KubeProxy` has some disadvantages, but based on the various features of CLB and VPC network, we have a more localized access layer solution. `KubeProxy` offers a universal and fault-tolerant design for the cluster access layer. It is basically applicable to clusters in all business scenarios. As an official component, this design is very appropriate.

New Mode Usage Guide

Prerequisites

- The Kubernetes version of the cluster is 1.12 or later.

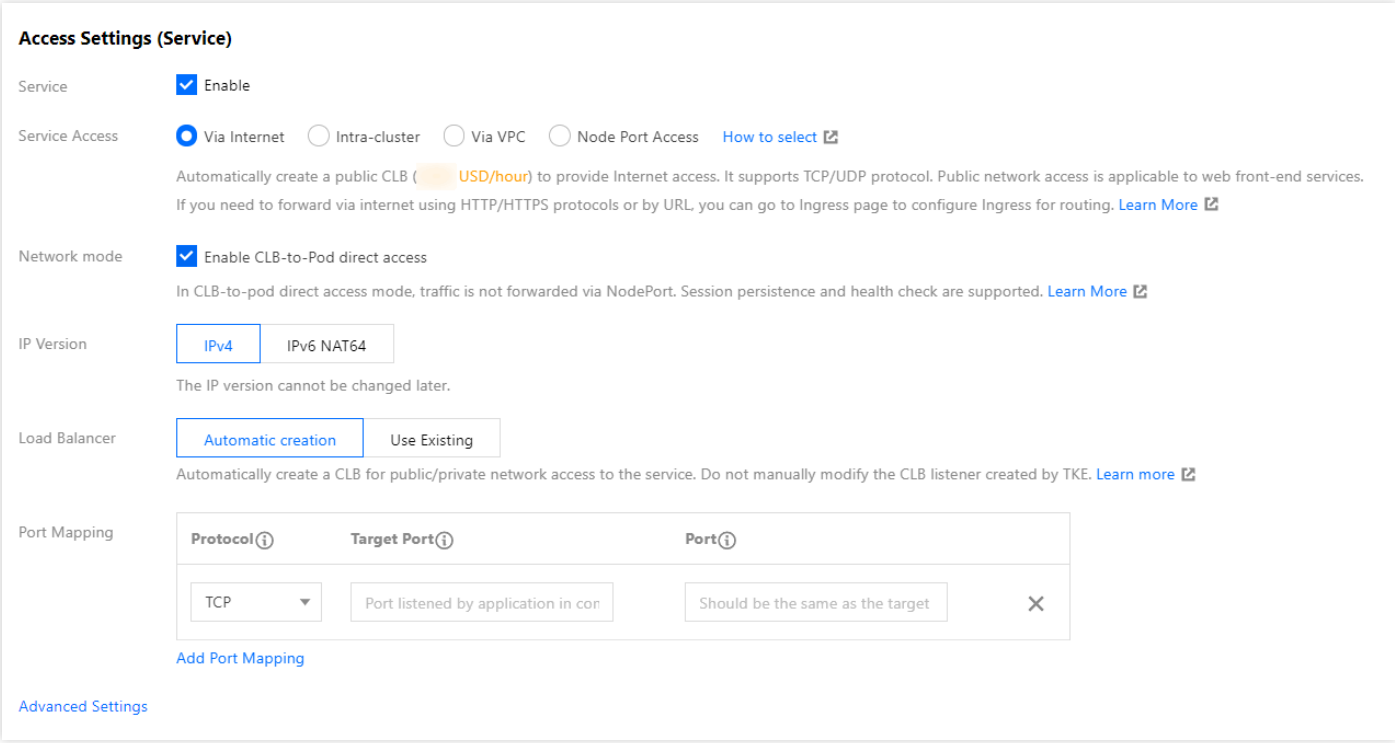
2. The VPC-CNI ENI mode is enabled for the cluster network mode.
3. The workloads used by a service in direct connection mode adopts the VPC-CNI ENI mode.

Console operation instructions

1. Log in to the TKE console.
2. Refer to the steps of creating a service in the console and go to the "Create a Service" page to configure the service parameters as required.

©2013-2022 Tencent Cloud. All rights reserved.

Configure the main parameters, as shown in the following figure:



- **Service Access Mode**: select **Provide Public Network Access** or **VPC Access**.
- **Network Mode**: select **Direct CLB-Pod Connection Mode**.
- **Workload Binding**: select **Import Workload**. In the window that appears, select the backend workload in VPC-CNI mode.

3. Click **Create Service** to complete the creation process.

Kubectl operation instructions

- **Workload example: nginx-deployment-eni.yaml**

> Note：
> Note: `spec.template.metadata.annotations` declares `tke.cloud.tencent.com/networks: tke-route-eni`, meaning that the workload uses the VPC-CNI ENI mode.

```
apiVersion: apps/v1
kind: Deployment
```

©2013-2022 Tencent Cloud. All rights reserved.

```
metadata:
labels:
app: nginx
name: nginx-deployment-eni
spec:
replicas: 3
selector:
matchLabels:
app: nginx
template:
metadata:
annotations:
tke.cloud.tencent.com/networks: tke-route-eni
labels:
app: nginx
spec:
containers:
- image: nginx:1.7.9
name: nginx
ports:
- containerPort: 80
protocol: TCP
```

- **Service example: nginx-service-eni.yaml**

> Note：
> `metadata.annotations` declares `service.cloud.tencent.com/direct-access: "true"`, meaning that, when synchronizing the CLB, the service configures the access backend by using the direct connection method.

```
apiVersion: v1
kind: Service
metadata:
annotations:
service.cloud.tencent.com/direct-access: "true"
labels:
app: nginx
name: nginx-service-eni
spec:
externalTrafficPolicy: Cluster
ports:
- name: 80-80-no
```

©2013-2022 Tencent Cloud. All rights reserved.

```
port: 80
protocol: TCP
targetPort: 80
selector:
app: nginx
sessionAffinity: None
type: LoadBalancer
```

- **Deploying Cluster**

> Note：
> In the deployment environment, you must first connect to a cluster (if you do not have a cluster, create one.)
> You can refer to the Help Document to configure kubectl to connect to a cluster.

```
➜ ~ kubectl apply -f nginx-deployment-eni.yaml
deployment.apps/nginx-deployment-eni created

➜ ~ kubectl apply -f nginx-service-eni.yaml
service/nginx-service-eni configured

➜ ~ kubectl get pod -o wide
NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES
nginx-deployment-eni-bb7544db8-6ljkm 1/1 Running 0 24s 172.17.160.191 172.17.0.
3 <none> 1/1
nginx-deployment-eni-bb7544db8-xqqtv 1/1 Running 0 24s 172.17.160.190 172.17.0.
46 <none> 1/1
nginx-deployment-eni-bb7544db8-zk2cx 1/1 Running 0 24s 172.17.160.189 172.17.0.
9 <none> 1/1
➜ ~ kubectl get service -o wide
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE SELECTOR
kubernetes ClusterIP 10.187.252.1 <none> 443/TCP 6d4h <none>
nginx-service-eni LoadBalancer 10.187.254.62 150.158.221.31 80:32693/TCP 6d1h a
pp=nginx
```

Summary

Currently, TKE uses ENI to implement the direct pod connection mode. We will further optimize this feature, including in the following respects:

©2013-2022 Tencent Cloud. All rights reserved.

- Implement direct pod connection under a common container network, without dependency on the VPC-ENI network mode.
- Support the removal of the CLB backend before pod deletion.

Comparison with similar solutions in the industry:

- AWS has a similar solution that implements direct pod connection through ENI.
- Google Kubernetes Engine (GKE) has a similar solution that integrates the Network Endpoint Groups (NEG) feature of Google Cloud Load Balancing (CLB) to implement direct connection to pods at the access layer.

References

1. Service
2. Ingress
3. Strategy
4. Pod readiness
5. Preserving the client source IP
6. How to Choose TKE Network Mode
7. GlobalRouter VPC-CNI Mode Description
8. Connecting to a Cluster
9. Kubernetes Ingress with AWS ALB Ingress Controller
0. GKE Container-native Load Balancing Through Standalone Zonal NEGs

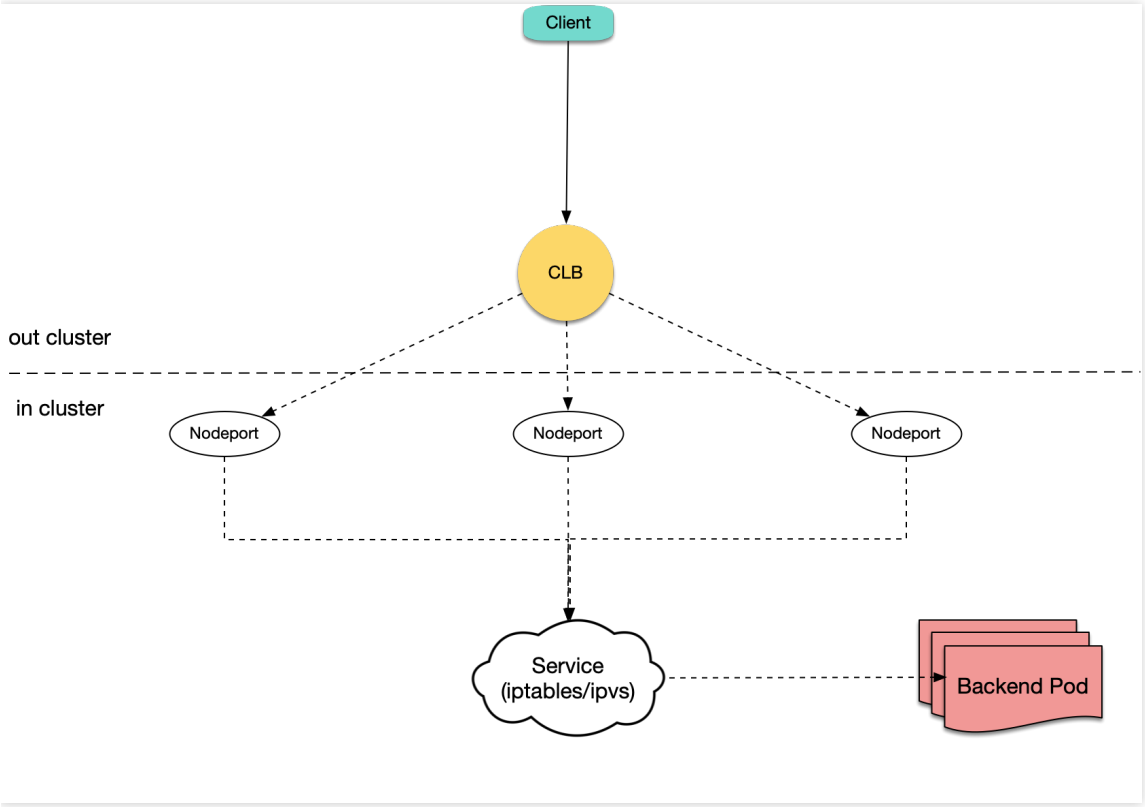
©2013-2022 Tencent Cloud. All rights reserved.

Use CLB-Pod Direct Connection on TKE

Last updated：2023-02-03 15:28:47

Overview

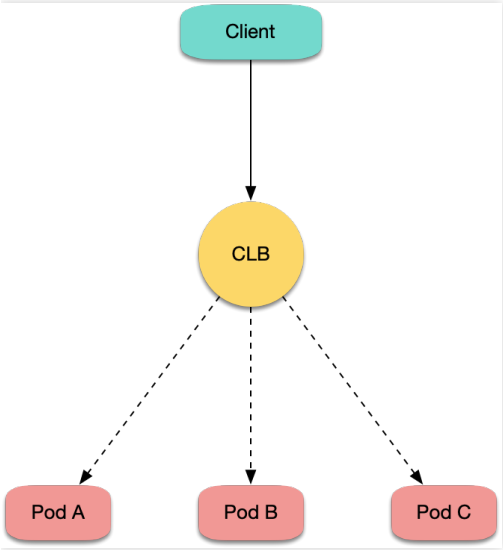
Kubernetes officially provides a NodePort-type Service. This means it provides all nodes with the same port through which a Service can be opened. Traditionally, most Services of the Cloud Load Balancer (CLB) type are implemented based on NodePort. Specifically, the CLB backend is bound with the NodePort of each node. When the CLB receives external traffic, it forwards the traffic to the NodePort of one of the nodes. Then, traffic is forwarded through the CLB within Kubernetes to pods by using iptables or ipvs. See the figure below:



TKE adopts the same approach to implement the default CLB-type Service and Ingress. Currently, however, it also supports the CLB-pod direct connection mode, in which the CLB backend is directly bound with pod IP + Port, without

©2013-2022 Tencent Cloud. All rights reserved.

being bound with the NodePort of nodes. See the figure below:



Analysis of Implementation Methods

Analysis of issues in the traditional NodePort method

Traditionally, users create a cloud Ingress or LB-type Service by using a CLB directly bound to Nodeport. However, the traditional method involves the following issues:

- After traffic is forwarded from the CLB to NodePort, it needs to go through SNAT before being forwarded to pods. This causes additional performance loss.
- If traffic is overly concentrated on a few NodePorts (for example, when gateways are deployed on a few nodes by using nodeSelector), source port exhaustion or conntrack insertion conflicts may occur.
- The NodePort of each node also serves as a CLB. If the CLB is bound with the NodePorts of too many nodes, the CLB status may be overly distributed, leading to a global load imbalance.

Advantages of the CLB-pod direct connection method

The CLB-pod direct connection method not only solves the issues of the traditional NodePort method but also offers the following advantages:

- As there is no SNAT, `externalTrafficPolicy: Local` is no longer needed to obtain the source IP address.
- Session persistence is easier to achieve. You only need to enable session persistence for the CLB, without having to set `sessionAffinity` in the Service.

©2013-2022 Tencent Cloud. All rights reserved.

Operation Scenarios

The CLB-pod direct connection method can be used in the following scenarios:

- You need to obtain the actual source IP address of the client in Layer-4 but do not expect to use the `externalTrafficPolicy: Local` method.
- The network performance needs to be further improved.
- Session persistence needs to be easier to achieve.
- Load imbalance in global connection scheduling needs to be resolved.

Prerequisites

- The Kubernetes version of the cluster must be 1.12 or later.
  For CLB-pod direct connection, you need to check whether pods are Ready. Specifically, check whether Pods are Running and have passed the readinessProbe and the CLB's pod health monitoring. This is dependent on the `ReadinessGate` feature, which is supported in Kubernetes 1.12 and later versions.
- The `VPC-CNI` ENI mode must be enabled for the cluster network mode. You can refer to Confirming whether ENI is enabled to perform confirmation.
  Currently, CLB-pod direct connection is implemented based on ENI and does not support the common network mode.

Directions

Confirming whether ENI is enabled

Perform the following steps based on your actual situation:

- If you have selected **VPC-CNI** for "Container network plugin" during cluster creation, then the pods created use ENI by default and you can skip this step.
- If you have selected **Global Router** for "Container network plugin" during cluster creation and then enabled VPC-CNI support, then the two modes are used at the same time. In that case, created pods do not use ENI by default. In this case, you need to use YAML to create workloads and specify the annotation `tke.cloud.tencent.com/networks: tke-route-eni` for pods to declare the use of ENI. In addition, you need to add requests and limits such as `tke.cloud.tencent.com/eni-ip: "1"` for one of the containers. The YAML sample is as follows:

```
apiVersion: apps/v1
kind: Deployment
metadata:
```

©2013-2022 Tencent Cloud. All rights reserved.

```
labels:
app: nginx
name: nginx-deployment-eni
spec:
replicas: 3
selector:
matchLabels:
app: nginx
template:
metadata:
annotations:
tke.cloud.tencent.com/networks: tke-route-eni
labels:
app: nginx
spec:
containers:
- image: nginx
name: nginx
resources:
requests:
tke.cloud.tencent.com/eni-ip: "1"
limits:
tke.cloud.tencent.com/eni-ip: "1"
```

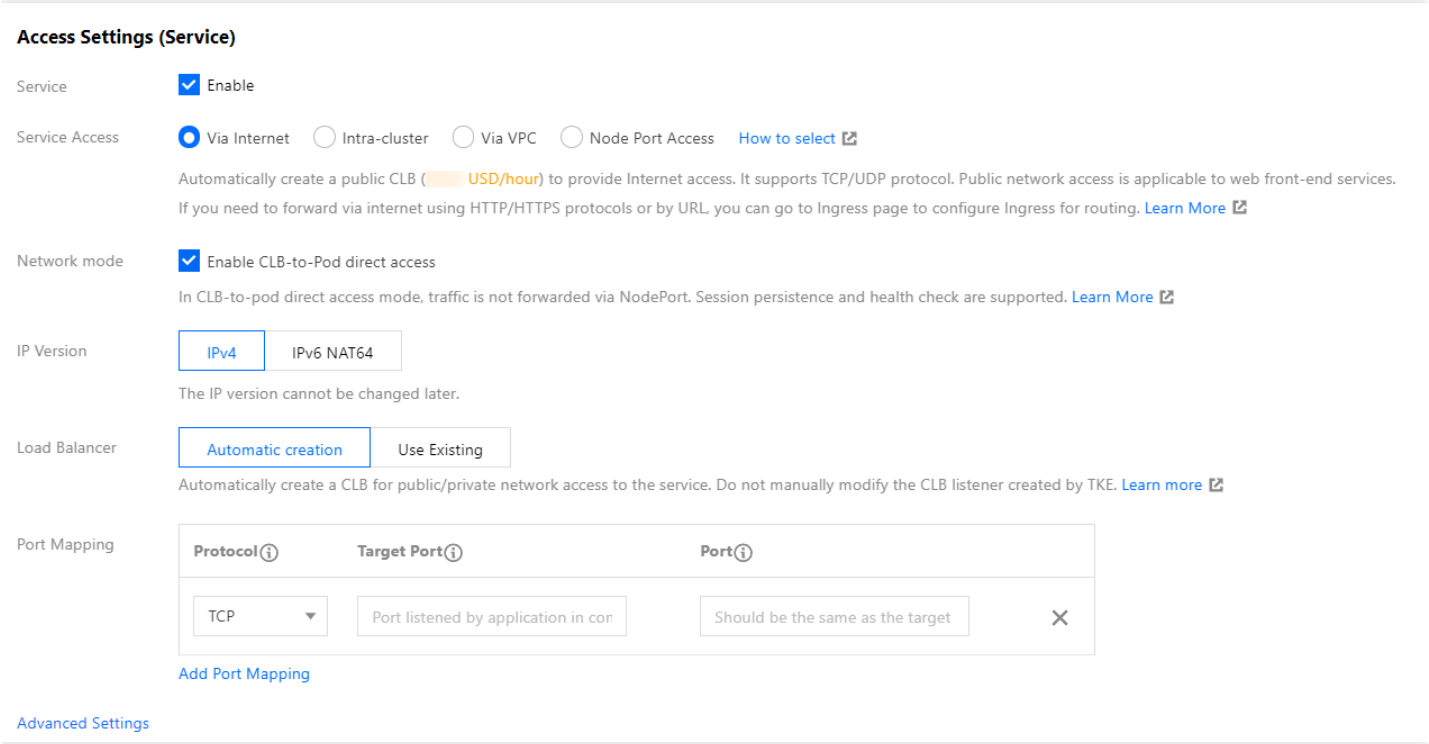
Declaring the direct connection mode during Service creation

When opening services through a CLB Service, you need to declare the use of the direct connection mode. The steps are as follows:

Using the console to create a Service

To use the console to create a Service, select **Direct CLB-Pod Connection Mode**. For more information, see Creating a Service. See the figure below:

©2013-2022 Tencent Cloud. All rights reserved.



Using YAML to create a Service

To use YAML to create a Service, you need to add the annotation `service.cloud.tencent.com/direct-access: "true"` for the Service. A sample is as follows:

> Note：
> For more information on how to use YAML to create a Service, see Creating a Service.

```
apiVersion: v1
kind: Service
metadata:
annotations:
service.cloud.tencent.com/direct-access: "true"
labels:
app: nginx
name: nginx-service-eni
spec:
externalTrafficPolicy: Cluster
ports:
- name: 80-80-no
port: 80
protocol: TCP
```

©2013-2022 Tencent Cloud. All rights reserved.

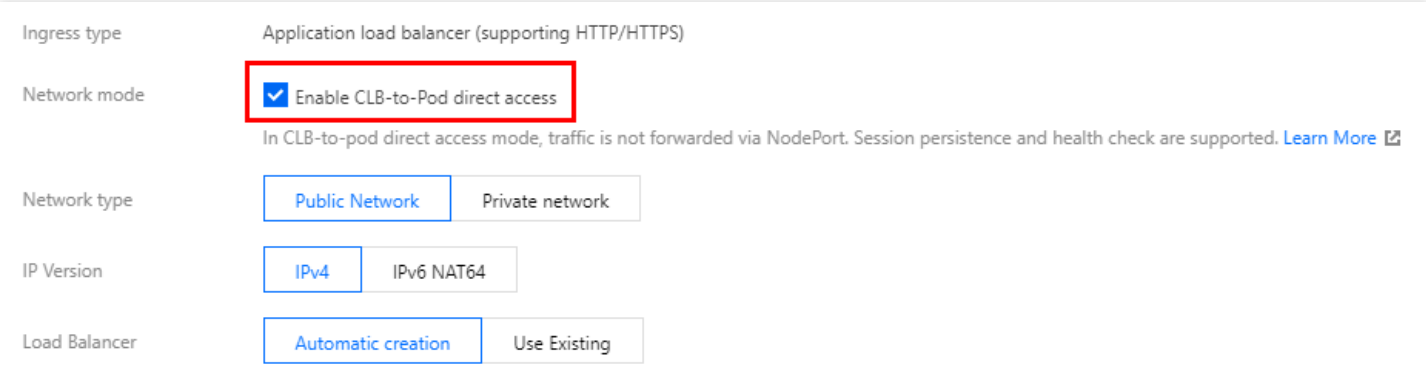
```
targetPort: 80
selector:
app: nginx
sessionAffinity: None
type: LoadBalancer
```

Declaring the direct connection mode during Ingress creation

When opening services through an Ingress, you also need to declare the use of the direct connection mode. The steps are as follows:

Using the console to create an Ingress

To use the console to create an Ingress, select **Direct CLB-Pod Connection Mode**. For more information, see Creating an Ingress. See the figure below:



Using YAML to create an Ingress

To use YAML to create an Ingress, you need to add the annotation `ingress.cloud.tencent.com/direct-access: "true"` for the Ingress. A sample is as follows:

> Note：
> For more information on how to use YAML to create an Ingress, see Creating an Ingress.

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
annotations:
ingress.cloud.tencent.com/direct-access: "true"
kubernetes.io/ingress.class: qcloud
name: test-ingress
namespace: default
```

©2013-2022 Tencent Cloud. All rights reserved.

```
spec:
rules:
- http:
paths:
- backend:
serviceName: nginx
servicePort: 80
path: /
```

References

- TKE in Direct Connection to the CLB of Pods Based on ENI
- Enabling VPC-CNI for a Cluster

©2013-2022 Tencent Cloud. All rights reserved.

Obtaining the Real Client Source IP in TKE

Last updated：2020-12-21 17:05:11

Application Scenarios

When your business requires to know the sources of service requests, the backend server must be able to accurately obtain the real client source IP of the request client. Possible scenarios:

- Audit the source of a service request. For example unusual login location alarms.
- Trace the source of a security attack or security event, such as APT attacks and DDoS attacks.
- Analyze data, such as service traffic region statistics.

Implementation Methods

In TKE, the default external load balancer is Tencent Cloud Load Balancer, which serves as the first access entry for incoming traffic. The CLB forwards request traffic loads to Kubernetes Service (default) of Kubernetes worker nodes. During this load-balancing process, the real client source IP is preservec (pass-through forwarded). However, in Kubernetes Service forwarding scenarios, data packets will go through SNAT during forwarding no matter whether the CLB forwarding mode is iptables or ipvs, which means that the real client source IP will not be preserved. For your reference, this document provides the following four methods for obtaining the real client source IP in TKE use cases. You can choose an appropriate method based on your actual needs.

Preserving the client source IP through Service resource configuration

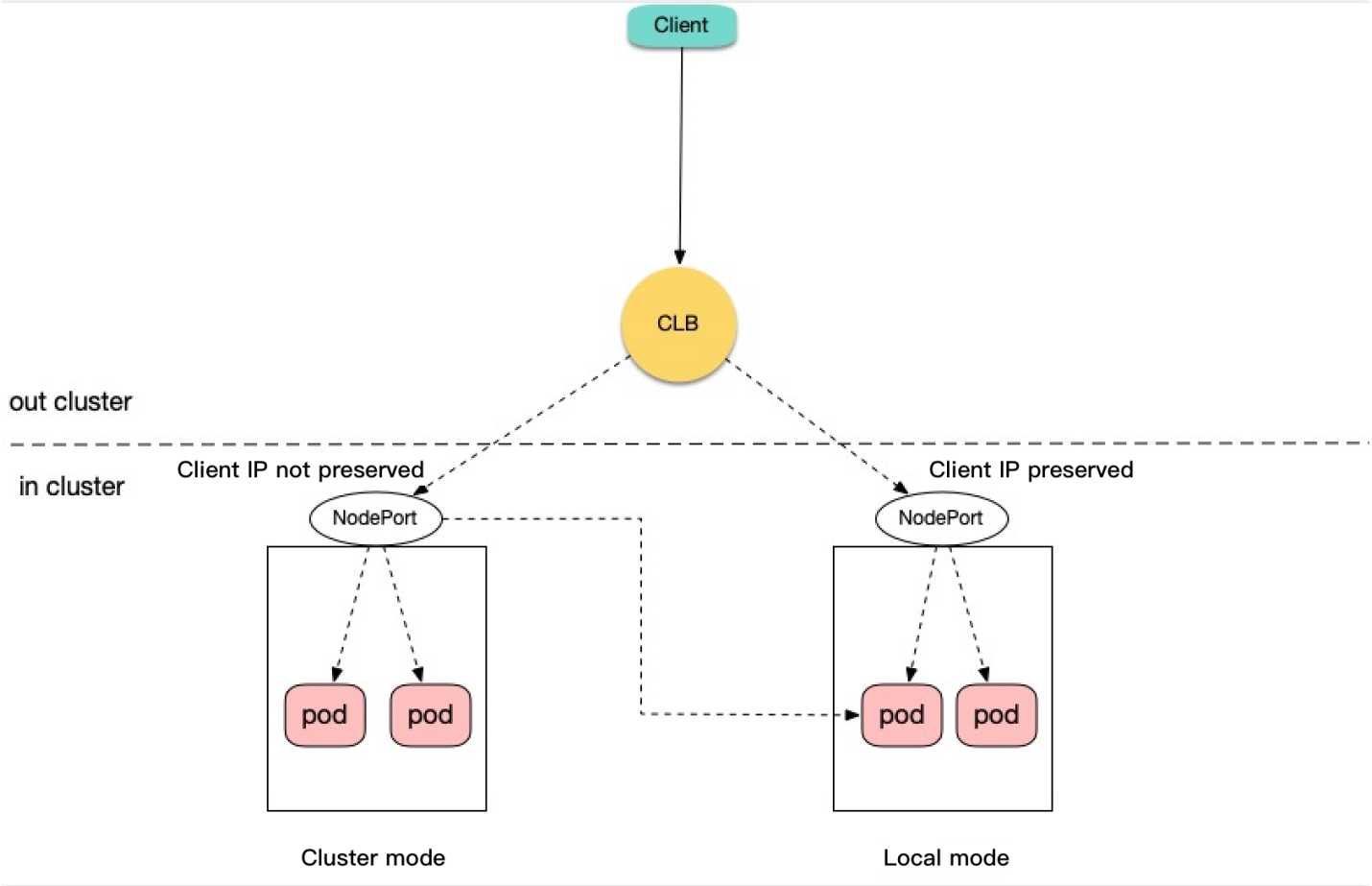
The advantage and disadvantage of this method are as follows:

- **Advantage**: you only need to configure Kubernetes Service resources.
- **Disadvantage**: potential risks of traffic load imbalance across pods (endpoints) may occur.

To enable the feature of preserving the client source IP, you can configure the `Service.spec.externalTrafficPolicy` field in Service resources. This field has two possible values, `Cluster` (default) and `Local`, which respectively indicate whether to route external traffic to the local or cluster

©2013-2022 Tencent Cloud. All rights reserved.

endpoints of nodes, as shown in the figure below:



- `Cluster` : hides the client source IP. Service traffic of the `LoadBalancer` and `NodePort` types may be forwarded to the pods of other nodes.
- `Local` : preserves the client source IP and prevents service traffic of the `LoadBalancer` and `NodePort` types from being forwarded to the pods of other nodes. For more information, see Create an External Load Balancer. The sample YAML configuration is as follows:

```
apiVersion: v1
kind: Service
metadata:
name: example-Service
spec:
selector:
app: example-Service
ports:
- port: 8765
targetPort: 9376
externalTrafficPolicy: Local
type: LoadBalancer
```

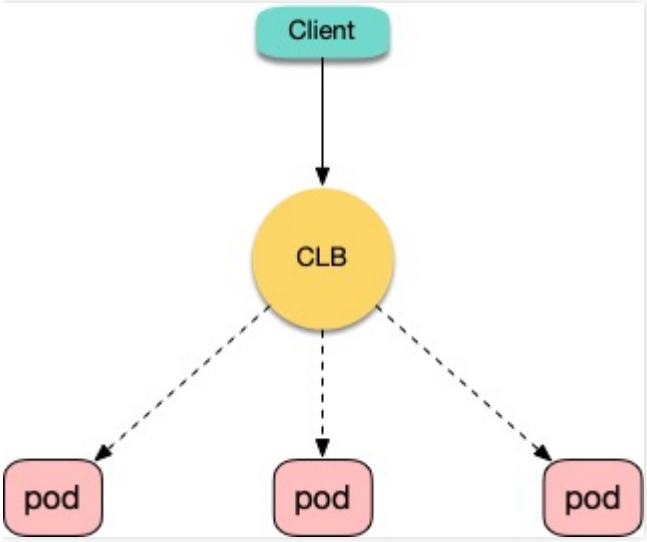
©2013-2022 Tencent Cloud. All rights reserved.

Obtaining the source IP address in the TKE native CLB-to-pod direct connection forwarding mode

The advantage and disadvantage of this method are as follows:

- **Advantage**: this feature is supported by native TKE. You only need to complete configuration in the console based on the corresponding reference document.
- **Disadvantage**: the VPC-CNI network mode needs to be enabled for the cluster.

The CLB-to-pod direct connection forwarding is a TKE native feature, which is actually CLB pass-through forwarding and bypasses Kubernetes Service traffic forwarding) is used, the source IP address of a request received by backend pods is the real source IP address of the client. This method applies to layer-4 and layer-7 service forwarding scenarios. The following figure shows how the forwarding works:



For more information and configuration details, see Using CLB-to-Pod Direct Connection on TKE.

Obtaining the source IP address through the HTTP header

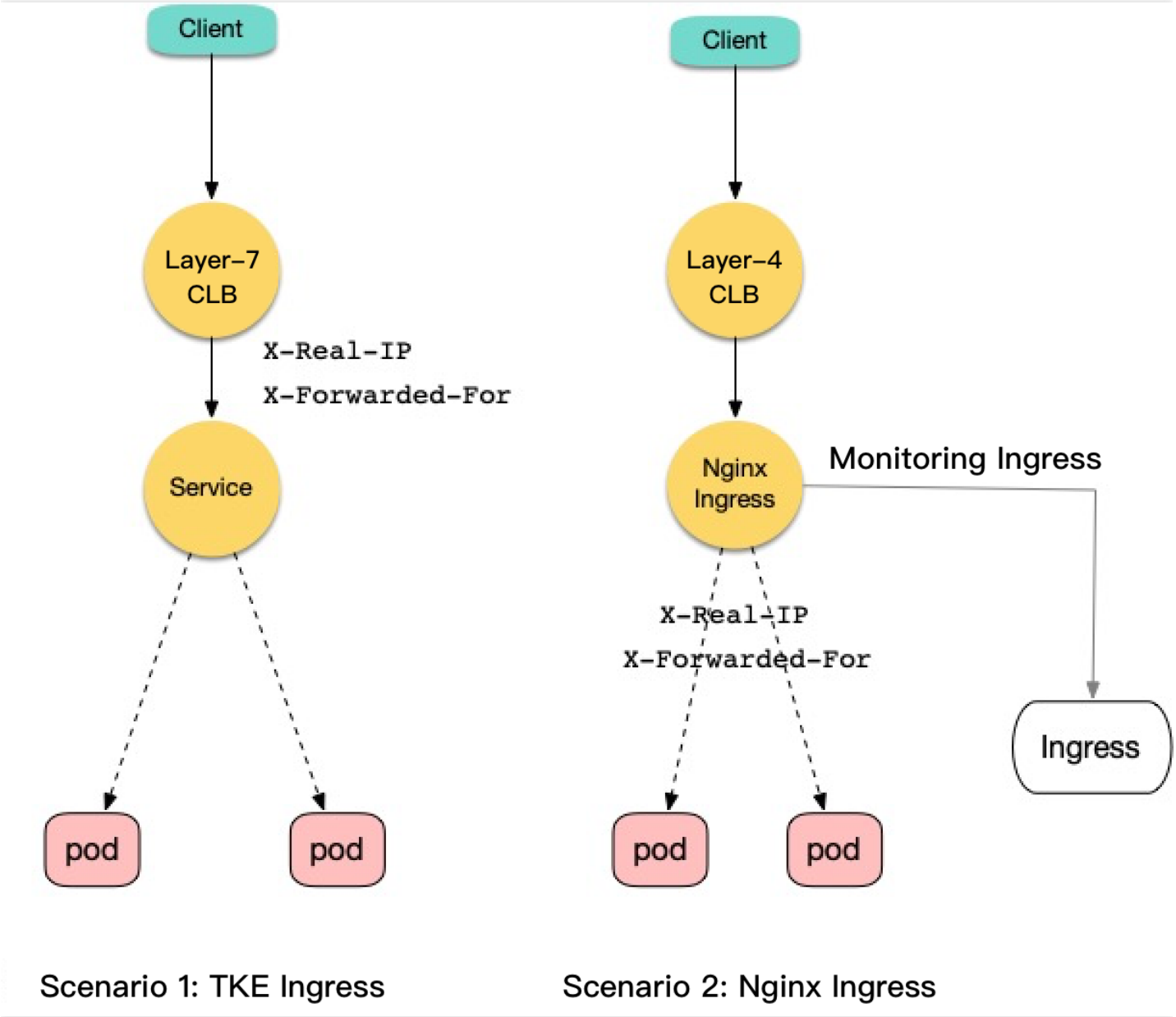
The advantage and disadvantage of this method are as follows:

- **Advantage**: this method is recommended for layer-7 (HTTP/HTTPS) traffic forwarding scenarios. The fields in the HTTP header can be directly obtained through web service proxy configuration or backend application code. In this way, the real source IP address of a client can be obtained easily and efficiently.
- **Disadvantage**: this method only applies to layer-7 (HTTP/HTTPS) traffic forwarding scenarios, not layer-4 forwarding scenarios.

In layer-7 (HTTP/HTTPS) service forwarding scenarios, the real source IP address of a client can be obtained from the `X-Forwarded-For` and `X-Real-IP` fields in the HTTP header. There are two use cases in TKE, as shown

©2013-2022 Tencent Cloud. All rights reserved.

in the figure below:

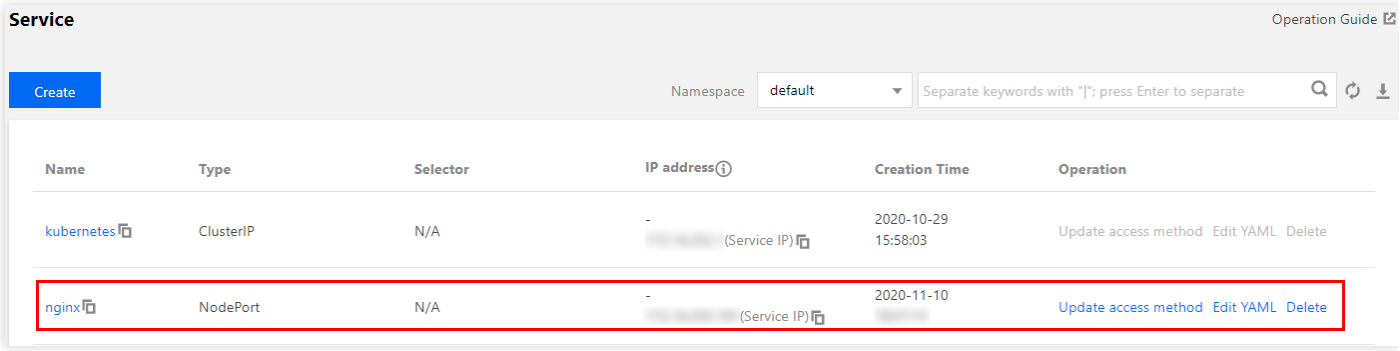


Scenario 1: using TKE Ingress to obtain the real source IP address

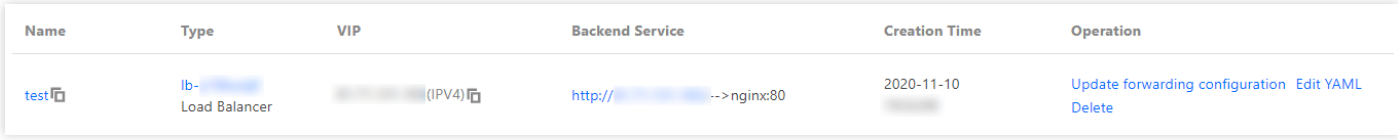
CLB (CLB layer-7) stores the real source IP address of a client in the `X-Forwarded-For` and `X-Real-IP` fields of the HTTP header by default. When service traffic goes through Service layer-4 forwarding, both fields are retained, and the backend can obtain the real source IP address of the client through web server proxy configuration or application code. For more information, see Obtain Acutual IP for Layer 7 Load Balancing. The process for obtaining the source IP address in the TKE console is as follows:

1. Create a NodePort-type Service for workloads. In this document, nginx is used as an example, as shown in the figure below:

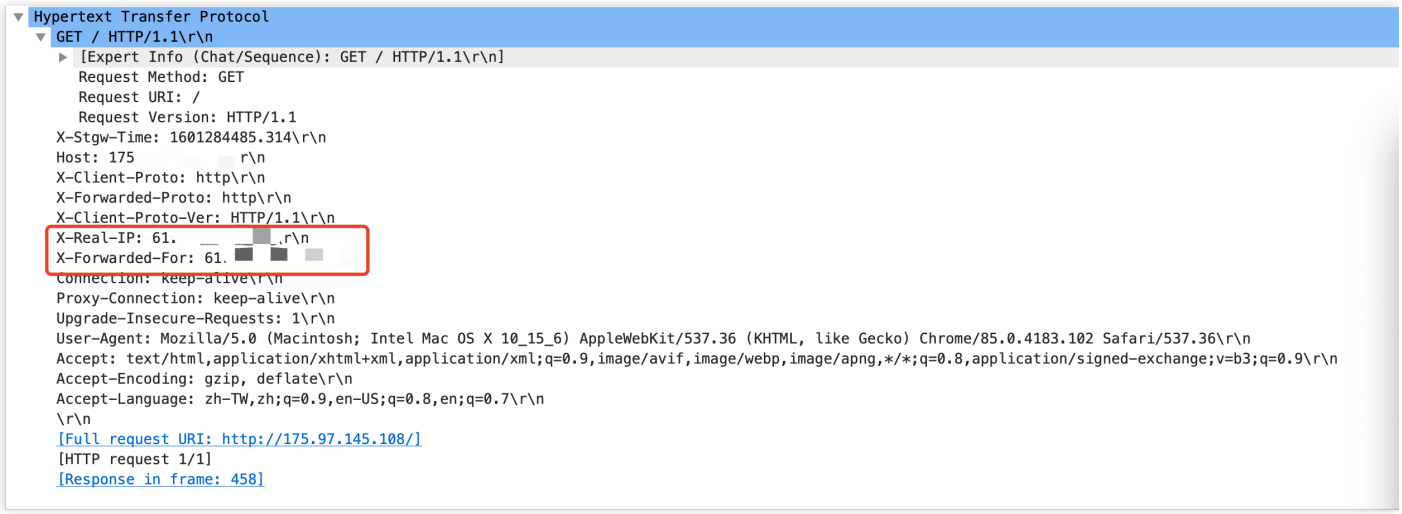
©2013-2022 Tencent Cloud. All rights reserved.



2. Create an Ingress access entry for Service. In this document, test is used as an example, as shown in the figure below:



3. After the configuration takes effect, you can obtain the real source IP address of a client from the `X-Forwarded-For` or `X-Real-IP` field of the HTTP header on the backend. The following figure shows the packet capture test results on the backend:



Scenario 2: using Nginx Ingress to obtain the real source IP address

Nginx Ingress service deployment requires Nginx Ingress to be able to perceive the real source IP address of a client. You can preserve the client source IP by create an external load balancer or using CLB-Pod direct connection on TKE. When forwarding requests, Nginx Ingress uses the `X-Forwarded-For` and `X-Real-IP` fields to store the client source IP, and the backend can obtain the real client source IP from these fields. The configuration process is as follows:

©2013-2022 Tencent Cloud. All rights reserved.

1. Nginx Ingress can be installed through TKE marketplace, custom YAML configuration, or the official (helm) installation method. For more information on its principles and deployment methods, see deployment solution 1 or 3 in Deploying Nginx Ingress on TKE. If you choose solution 1 for deployment, you must change the value of the `externalTrafficPolicy` field of Nginx Ingress Controller Service to `Local`.
   After the installation is completed, a CLB (layer-4) access entry is automatically created for Nginx Ingress Controller Service, which can be checked in the TKE console as shown in the figure below:

2. Create an Ingress for the backend server that requires forwarding, and configure forwarding rules. The sample YAML file is as follows:

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
annotations:
kubernetes.io/ingress.class: nginx # ingressClass is "nginx".
name: example
namespace: default
spec:
rules: # Configure service forwarding rules
- http:
paths:
- backend:
serviceName: nginx
servicePort: 80
path: /
```

3. After the configuration takes effect, you can obtain the real client source IP from the `X-Forwarded-For` or `X-Real-IP` field of the HTTP header on the backend. The following figure shows the packet capture test results on

©2013-2022 Tencent Cloud. All rights reserved.

the backend:

Obtaining the real source IP through TOA kernel component loading

The advantage and disadvantages of this method are as follows:

- **Advantage**: in the TCP transmission mode, only the first TCP connection packet is reconstructed at the kernel layer with almost no performance loss.
- **Disadvantages**:
  - You must load the TOA kernel component on cluster worker nodes and call functions on the server side to obtain the source IP address and port information carried by requests. The configuration and usage are relatively complex.
  - In the UDP transmission mode, each data packet is reconstructed to include option data (the source IP address and source port), which results in performance loss on the network transmission connection.

For the principles and loading method of the TOA kernel component, see Obtaining the Real IPs of Access Users.

References

- How Tencent CLB obtains real client IP addresses: Obtaining Real IP for Layer 7 Load Balancing
- Introduction to Tencent CLB: Cloud Load Balancer
- Deploying Nginx Ingress on TKE
- Introduction to the TKE network mode: GlobalRouter VPC-CNI Mode Description
- Using CLB-to-Pod Direct Connection on TKE
- Introduction to TOA module usage: Obtaining the Real IPs of Access Users
- Description of external load balancer configuration for Kubernetes: Create an External Load Balancer

©2013-2022 Tencent Cloud. All rights reserved.

Using Traefik Ingress in TKE

Last updated：2023-05-22 18:01:11

Operation Scenario

Traefik is an excellent reverse proxy tool. Compared with Nginx, Traefik offers the following advantages:

- Native support for the dynamic configuration of, for example, Kubernetes CRD resources such as Ingress and IngressRoute (Nginx requires reloading of the full configuration each time, which may affect connections in some cases).
- Native support for service discovery. After dynamic configuration, such as by using Ingress and IngressRoute, Traefik will automatically watch the backend endpoint and synchronize it to the backend list of the CLB.
- Elegant Dashboard management page.
- Native support for metrics and seamless integration with Prometheus and Kubernetes.
- More advanced features, such as multi-version canary release, traffic replication, automatic generation of free HTTPS certificates, and middleware.

This document introduces how to install Traefik in a TKE cluster and provides use cases for Ingress and IngressRoute via Traefik.

Prerequisites

- You have created a TKE cluster and can connect to the cluster via Kubectl.
- You have installed Helm.

Directions

Installing Traefik

This document describes the installation of Traefik in a TKE cluster as an example. For the detailed installation method, see the Traefik official documentation.

1. Run the following command to add the Helm chart repo source of Traefik. See the sample below:

```
helm repo add traefik https://helm.traefik.io/traefik
```

©2013-2022 Tencent Cloud. All rights reserved.

2. Prepare the installation configuration file `values-traefik.yaml`. See the sample below:

```
providers:
kubernetesIngress:
publishedService:
enabled: true # Display the external IP address of Ingress as the LB IP address of Traefik.
additionalArguments:
- "--providers.kubernetesingress.ingressclass=traefik" # Indicates the ingress class name.
- "--log.level=DEBUG"
service:
annotations:
service.cloud.tencent.com/direct-access: "true" # For gateway applications, we recommend that you use direct connection between the LB and pods (bypassing NodePort). If you use the VPC-CNI and Global Router network modes at the same time, use this annotation to display the declaration of direct binding of the LB to pods (bypassing NodePort). If you selected the VPC-CNI network mode during cluster creation, the declaration need not be displayed (because, by default, the LB is directly connected to pods). For more information, see the official documentation at https://intl.cloud.tencent.com/document/product/457/38408.
service.kubernetes.io/tke-existed-lbid: lb-lb57hvgl # Use this annotation to bind the LB created in advance, so that even if Traefik is rebuilt in the future, the traffic entry will remain unchanged. If you do not specify this annotation, a new LB will be automatically created by default. For more information, see the official documentation at https://intl.cloud.tencent.com/document/product/457/36835.
ports:
web:
expose: true
exposedPort: 80 # HTTP port number that is externally exposed. To use a standard port number in the Chinese mainland, ICP filing is required.
websecure:
expose: true
exposedPort: 443 # HTTPS port number that is externally exposed. To use a standard port number in the Chinese mainland, ICP filing is required.
deployment:
enabled: true
replicas: 1
podAnnotations:
tke.cloud.tencent.com/networks: "tke-route-eni" # When VPC-CNI and Global Router network modes are used at the same time, display a statement to indicate the ENI to be used by pods. This should be used together with the request and limit of eni-ip below.
resources:
requests:
```

©2013-2022 Tencent Cloud. All rights reserved.

```
tke.cloud.tencent.com/eni-ip: "1"
limits:
tke.cloud.tencent.com/eni-ip: "1"
```

Note：
To view the full default configuration, run `helm show values traefik/traefik`.
3. Run the following command to install Traefik to your TKE cluster. See the sample below:

```
kubectl create ns ingress
helm upgrade --install traefik -f values-traefik.yaml traefik/traefik
```

4. Run the following command to obtain the IP address of the traffic entry (for example, EXTERNAL-IP as shown below). See the sample below:

```
$ kubectl get service -n ingress
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
traefik LoadBalancer 172.22.252.242 49.233.239.84 80:31650/TCP,443:32288/TCP 42
h
```

Using an Ingress

Traefik allows you to use the Ingress resources of Kubernetes as a dynamic configuration. You can directly create Ingress resources in your cluster and use them to externally open your cluster. You need to add the specified Ingress class (defined during Traefik installation). See the sample below:

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
name: test-ingress
annotations:
kubernetes.io/ingress.class: traefik # Indicates the ingress class name.
spec:
rules:
- host: traefik.demo.com
http:
paths:
- path: /test
backend:
serviceName: nginx
servicePort: 80
```

©2013-2022 Tencent Cloud. All rights reserved.

Note：
At present, TKE does not display Traefik as a product, so you cannot use the TKE console to create an Ingress in a visualized manner. Instead, you need to use YAML to create the Ingress.

Using IngressRoute

Traefik not only supports standard Kubernetes Ingress resources but also supports the unique CRD resources of Traefik, such as IngressRoute. It can support more advanced features that an Ingress does not provide. See the IngressRoute usage example below:

```
apiVersion: traefik.containo.us/v1alpha1
kind: IngressRoute
metadata:
name: test-ingressroute
spec:
entryPoints:
- web
routes:
- match: Host(`traefik.demo.com`) && PathPrefix(`/test`)
kind: Rule
services:
- name: nginx
port: 80
```

Note：
For more information on the usage of Traefik, see the Traefik Official Documentation.

©2013-2022 Tencent Cloud. All rights reserved.

Release
Using CLB to Implement Simple Blue-Green Deployment and Grayscale Release

Last updated：2023-02-06 15:15:34

Overview

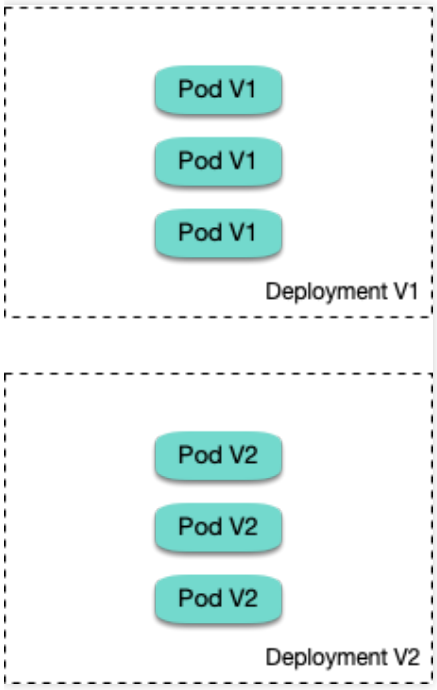
To implement blue-green deployment or Grayscale Release in a Tencent Cloud Kubernetes cluster, you usually need to deploy extra open-source tools in the cluster, such as Nginx Ingress and Traefik or deploy services to Service Mesh to utilize its capabilities. These solutions are relatively difficult to implement. If you only have simple requirements for blue-green deployment or grayscale release, you don’t expect to import too many components into the cluster, and don’t require complex usage, you can refer to this document to utilize the native features of Kubernetes and the LB plug-ins of TKE general clusters and TKE Serverless clusters to implement simple blue-green deployment and grayscale release.

> Note：
> This document is applicable only to TKE general clusters and TKE Serverless clusters.

How It Works

Users usually use Kubernetes workloads, such as Deployment and StatefulSet, to deploy businesses. Each workload manages a group of pods. With Deployment as an example, the following figure shows how it works:

©2013-2022 Tencent Cloud. All rights reserved.



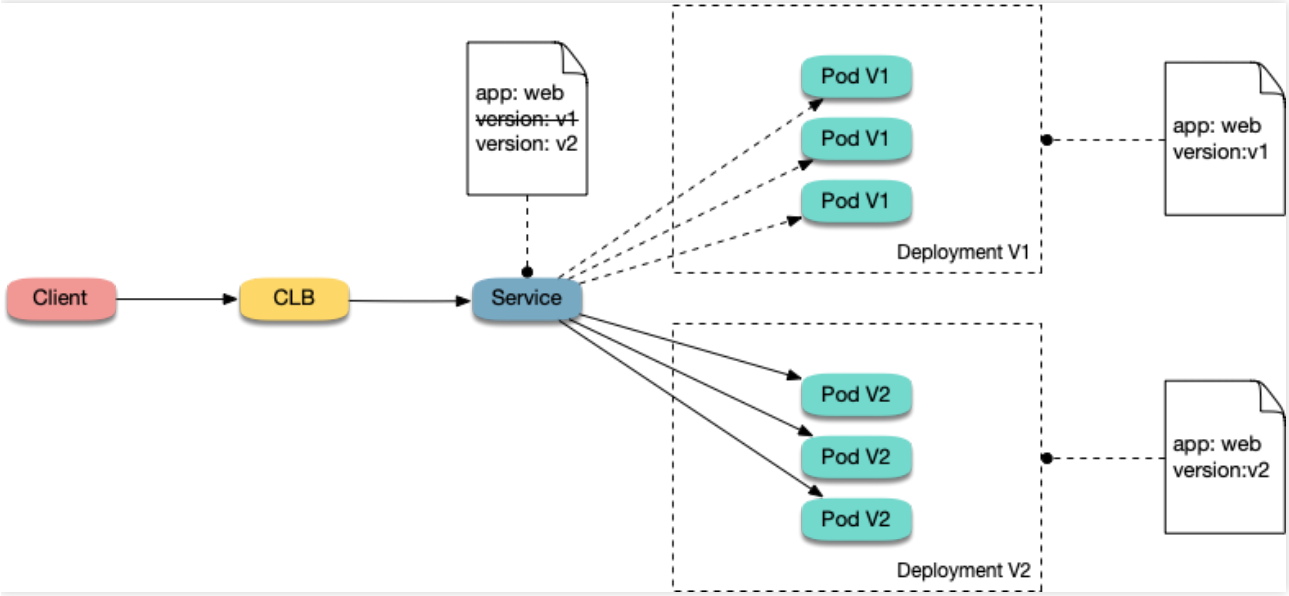
For each workload, a corresponding Service is created, and the Service matches backend pods via a selector. This allows other services or external requests to access the Service and the services provided by backend pods. To open services to external users, you can directly set the Service type to LoadBalancer, and the LB plug-in will automatically create a Tencent CLB as the traffic entry.

How blue-green deployment works

Using Deployment as an example, assume that two different versions of Deployment have been deployed in a cluster, and its pods have the same label, but the two versions correspond to two different label values. In this case, the Service selects the pods of one of the versions via the selector. We can modify the label value, which indicates the

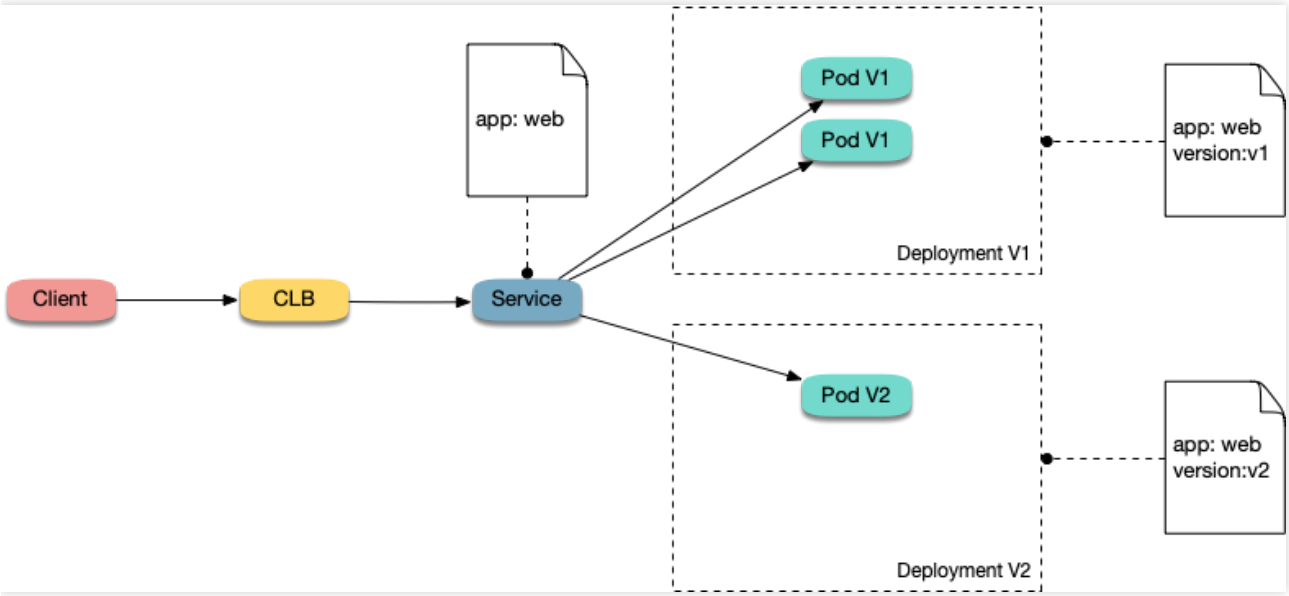
©2013-2022 Tencent Cloud. All rights reserved.

service version, in the selector of the Service to switch services from one version to the other. See the figure below:



How Grayscale Release works

Users usually create a Service for each workload, but Kubernetes does not require Services to have a one-to-one correspondence to workloads. When a Service matches backend pods via a selector, if the pods of different workloads are selected by the same selector, then the Service corresponds to multiple workload versions. By adjusting the number of replicas of different workload versions, you can adjust the weight of different service versions. See the figure below:



Directions

©2013-2022 Tencent Cloud. All rights reserved.

Using YAML to create resources

This document introduces the following two methods for using YAML to deploy workloads and create Services:

- Method 1: log in to the TKE console, click the ID of the target cluster to go to the cluster details page, click **Create via YAML** in the upper right corner, and input the YAML sample file content in this document to the editing interface.
- Method 2: save the sample YAML as a file and use kubectl to specify the YAML file to create resources, for example, `kubectl apply -f xx.yaml` .

Deploying multiple versions of workloads

1. Deploy the first version of Deployment in the cluster. Here nginx is used as an example. The YAML sample is as follows:

```
apiVersion: apps/v1
kind: Deployment
metadata:
name: nginx-v1
spec:
replicas: 3
selector:
matchLabels:
app: nginx
version: v1
template:
metadata:
labels:
app: nginx
version: v1
spec:
containers:
- name: nginx
image: "openresty/openresty:centos"
ports:
- name: http
protocol: TCP
containerPort: 80
volumeMounts:
- mountPath: /usr/local/openresty/nginx/conf/nginx.conf
name: config
subPath: nginx.conf
volumes:
- name: config
configMap:
```

©2013-2022 Tencent Cloud. All rights reserved.

```
name: nginx-v1
---
apiVersion: v1
kind: ConfigMap
metadata:
labels:
app: nginx
version: v1
name: nginx-v1
data:
nginx.conf: |-
worker_processes 1;
events {
accept_mutex on;
multi_accept on;
use epoll;
worker_connections 1024;
}
http {
ignore_invalid_headers off;
server {
listen 80;
location / {
access_by_lua '
local header_str = ngx.say("nginx-v1")
';
}
}
}
```

2. Deploy the second version of Deployment. Here nginx is used as an example. The YAML sample is as follows:

```
apiVersion: apps/v1
kind: Deployment
metadata:
name: nginx-v2
spec:
replicas: 3
selector:
matchLabels:
app: nginx
version: v2
template:
metadata:
```

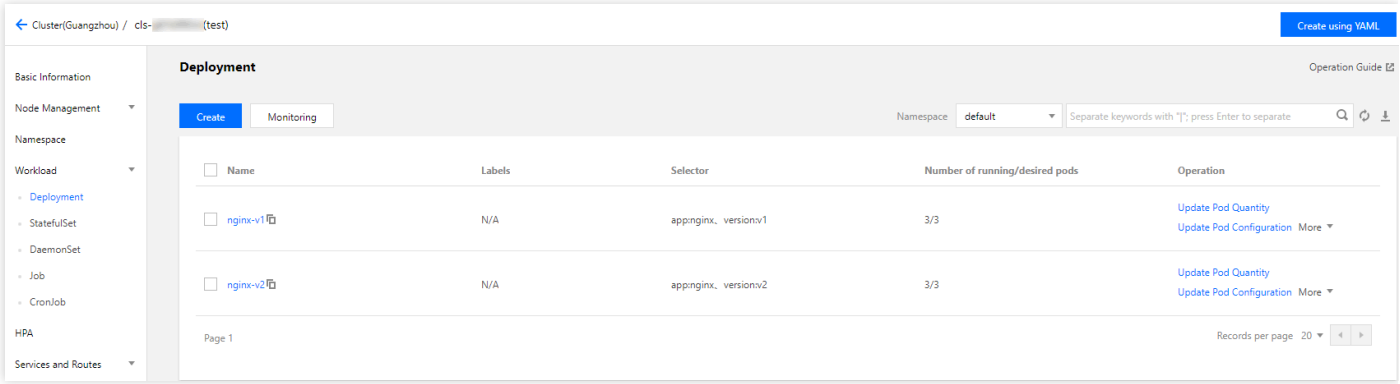
©2013-2022 Tencent Cloud. All rights reserved.

```
labels:
app: nginx
version: v2
spec:
containers:
- name: nginx
image: "openresty/openresty:centos"
ports:
- name: http
protocol: TCP
containerPort: 80
volumeMounts:
- mountPath: /usr/local/openresty/nginx/conf/nginx.conf
name: config
subPath: nginx.conf
volumes:
- name: config
configMap:
name: nginx-v2
---
apiVersion: v1
kind: ConfigMap
metadata:
labels:
app: nginx
version: v2
name: nginx-v2
data:
nginx.conf: |-
worker_processes 1;
events {
accept_mutex on;
multi_accept on;
use epoll;
worker_connections 1024;
}
http {
ignore_invalid_headers off;
server {
listen 80;
location / {
access_by_lua '
local header_str = ngx.say("nginx-v2")
';
}
}
```

©2013-2022 Tencent Cloud. All rights reserved.

```
}
```

3. Log in to the TKE console and go to the **Workload** > **Deployment** page of the cluster to view the deployment information.



Implementing blue-green deployment

1. Create a LoadBalancer-type Service for the deployed Deployment to open services to external users and specify that the v1 version is used. The YAML sample is as follows:

```
apiVersion: v1
kind: Service
metadata:
name: nginx
spec:
type: LoadBalancer
ports:
- port: 80
protocol: TCP
name: http
selector:
app: nginx
version: v1
```

2. Run the following commands to test the access.

```
for i in {1..10}; do curl EXTERNAL-IP; done; # Replace EXTERNAL-IP with the CLB IP address of the Service.
```

©2013-2022 Tencent Cloud. All rights reserved.

The returned results are as follows. All of them are responses from the v1 version.

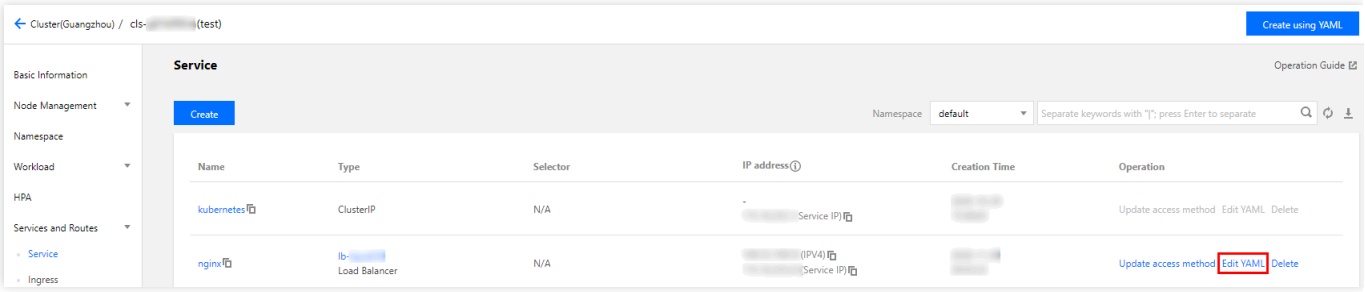
```
nginx-v1
nginx-v1
nginx-v1
nginx-v1
nginx-v1
nginx-v1
nginx-v1
nginx-v1
nginx-v1
nginx-v1
```

3. Use the console or kubectl to modify the selector of the Service to enable the selector to select the v2 service version:

- **Modification via the console**:

  i. Go to the cluster details page, and choose **Services and Routes** > **Service** in the left sidebar.

  ii. On the "Service" page, locate the Service to be modified and click **Edit YAML** to its right, as shown in the figure below:



  Modify the selector content as follows:

```
selector:
app: nginx
version: v2
```

  iii. Click **Done**.

- **Modification via kubectl**:

```
kubectl patch service nginx -p '{"spec":{"selector":{"version":"v2"}}}'
```

©2013-2022 Tencent Cloud. All rights reserved.

4. Run the following commands to test the access again.

```
$ for i in {1..10}; do curl EXTERNAL-IP; done; # Replace EXTERNAL-IP with the C
LB IP address of the Service.
```

The returned results are as follows. All of them are responses from the v2 version. This means you have successfully implemented blue-green deployment.

```
nginx-v2
nginx-v2
nginx-v2
nginx-v2
nginx-v2
nginx-v2
nginx-v2
nginx-v2
nginx-v2
nginx-v2
```

Implementing Grayscale Release

1. Grayscale Release is different from blue-green deployment. You do not need to specify the v1 version to be used by the Service. You only need to delete the `version` label in the selector so that the Service will simultaneously select the pods of the two Deployment versions. The YAML sample is as follows:

```
apiVersion: v1
kind: Service
metadata:
name: nginx
spec:
type: LoadBalancer
ports:
- port: 80
protocol: TCP
name: http
selector:
app: nginx
```

2. Run the following commands to test the access.

©2013-2022 Tencent Cloud. All rights reserved.

```
for i in {1..10}; do curl EXTERNAL-IP; done; # Replace EXTERNAL-IP with the CLB IP address of the Service.
```

The returned results are as follows. Half of them are responses from the v1 version, and the other half from the v2 version.

```
nginx-v1
nginx-v1
nginx-v2
nginx-v2
nginx-v2
nginx-v1
nginx-v1
nginx-v1
nginx-v2
nginx-v2
```

3. Use the console or kubectl to adjust the replicas of Deployment versions v1 and v2. Specifically, set v1 to 1 replica and v2 to 4 replicas.

- **Modification via the console**:
  i. Go to the **Workload** > **Deployment** page of the cluster and choose **More** > **Edit YAML** to the right of the v1 Deployment version.
  ii. On the YAML editing page, change `.spec.replicas` of v1 to 1 and click **Done**.
  iii. Repeat the above steps to change `.spec.replicas` of v2 to 4 and click **Done**.
- **Modification via kubectl**:

```
kubectl scale deployment/nginx-v1 --replicas=1
kubectl scale deployment/nginx-v2 --replicas=4
```

4. Run the following commands to perform an access test again.

```
for i in {1..10}; do curl EXTERNAL-IP; done; # Replace EXTERNAL-IP with the CLB IP address of the Service.
```

The returned results are as follows. In 10 access attempts, the v1 version responded only twice. The ratio between the responses of v1 and those of v2 is consistent with the ratio between their replicas, that is, 1:4. This shows you have implemented Grayscale Release by controlling the number of replicas of different service versions.

©2013-2022 Tencent Cloud. All rights reserved.

```
nginx-v2
nginx-v1
nginx-v2
nginx-v2
nginx-v2
nginx-v2
nginx-v1
nginx-v2
nginx-v2
nginx-v2
```

©2013-2022 Tencent Cloud. All rights reserved.

Using Nginx Ingress to Implement Canary Release

Last updated：2020-11-11 14:52:29

This document introduces the use cases, usage, and practices of implementing Canary Release by using Nginx Ingress.

Note：

For clusters that implement Canary Release by using Nginx Ingress, Nginx Ingress should be deployed as the Ingress Controller, and a unified traffic entry should be opened for external access. For more information, see Deploying Nginx Ingress on TKE.

Use Cases

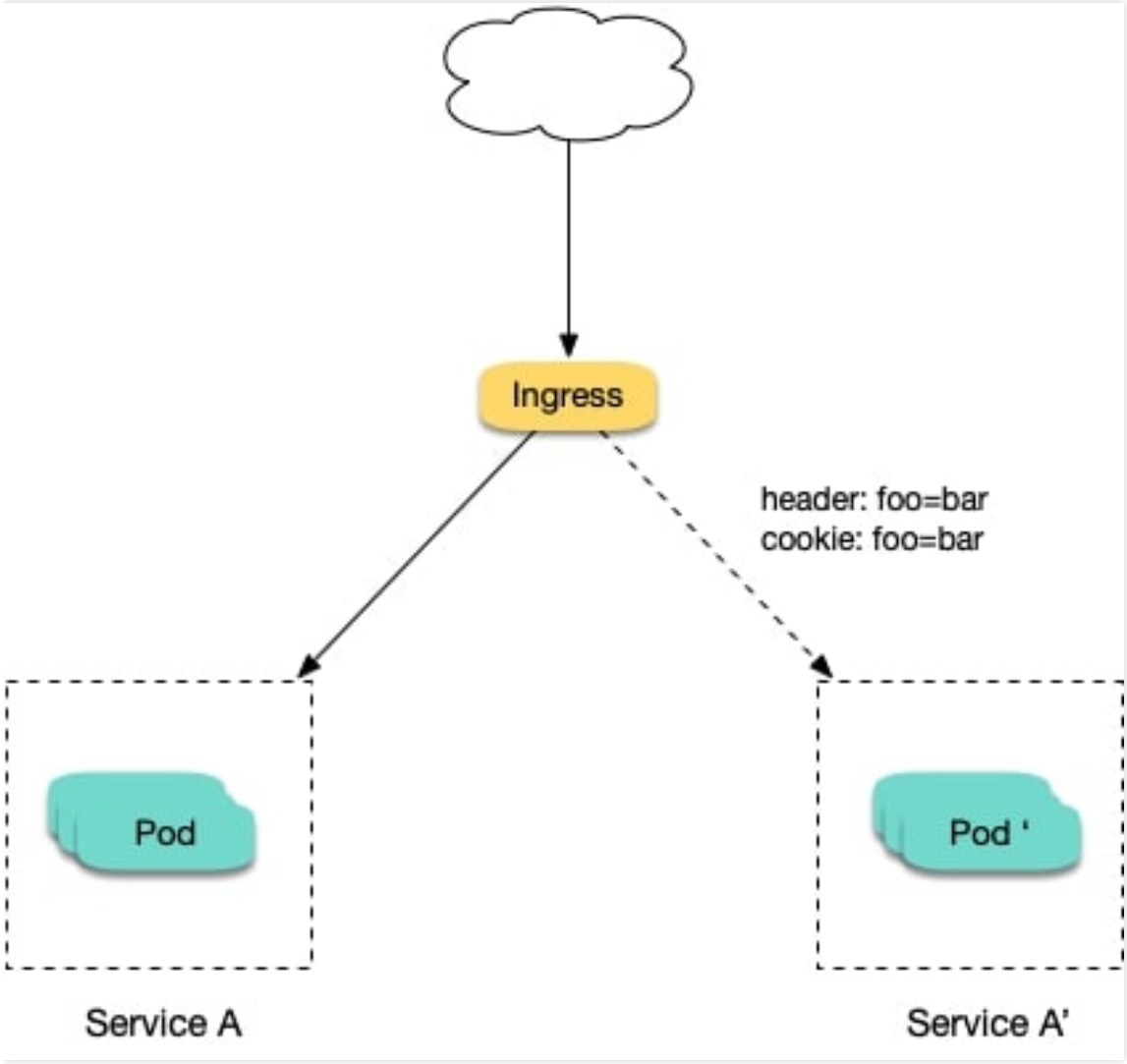
The application scenarios where Canary Release is implemented through Nginx Ingress mainly depend on the business traffic splitting policy. Currently, Nginx Ingress supports three types of traffic splitting policies, which are based on Header, Cookie, and service weight, respectively. Based on these three types of policies, the following two deployment scenarios can be implemented:

Scenario 1: providing some users with a new version for beta testing

Assume that Service A, which provides Layer-7 service to external users, has been running online; you want to activate a newly developed version, Service A', for some users as a beta test, without replacing the existing Service A; you want to let it run stably for some time before gradually and fully activating the new version and smoothly deactivating the old version.
For this scenario, you can use Nginx Ingress to make deployments under traffic split policies based on Header or Cookie. Business uses Header or Cookie to mark different types of users and configures Ingress to enable requests with the specified Header or Cookie to be forwarded to the new version while other requests are still forwarded to the

©2013-2022 Tencent Cloud. All rights reserved.

old version. In this way, the new version is available to some users for beta testing. See the figure below:

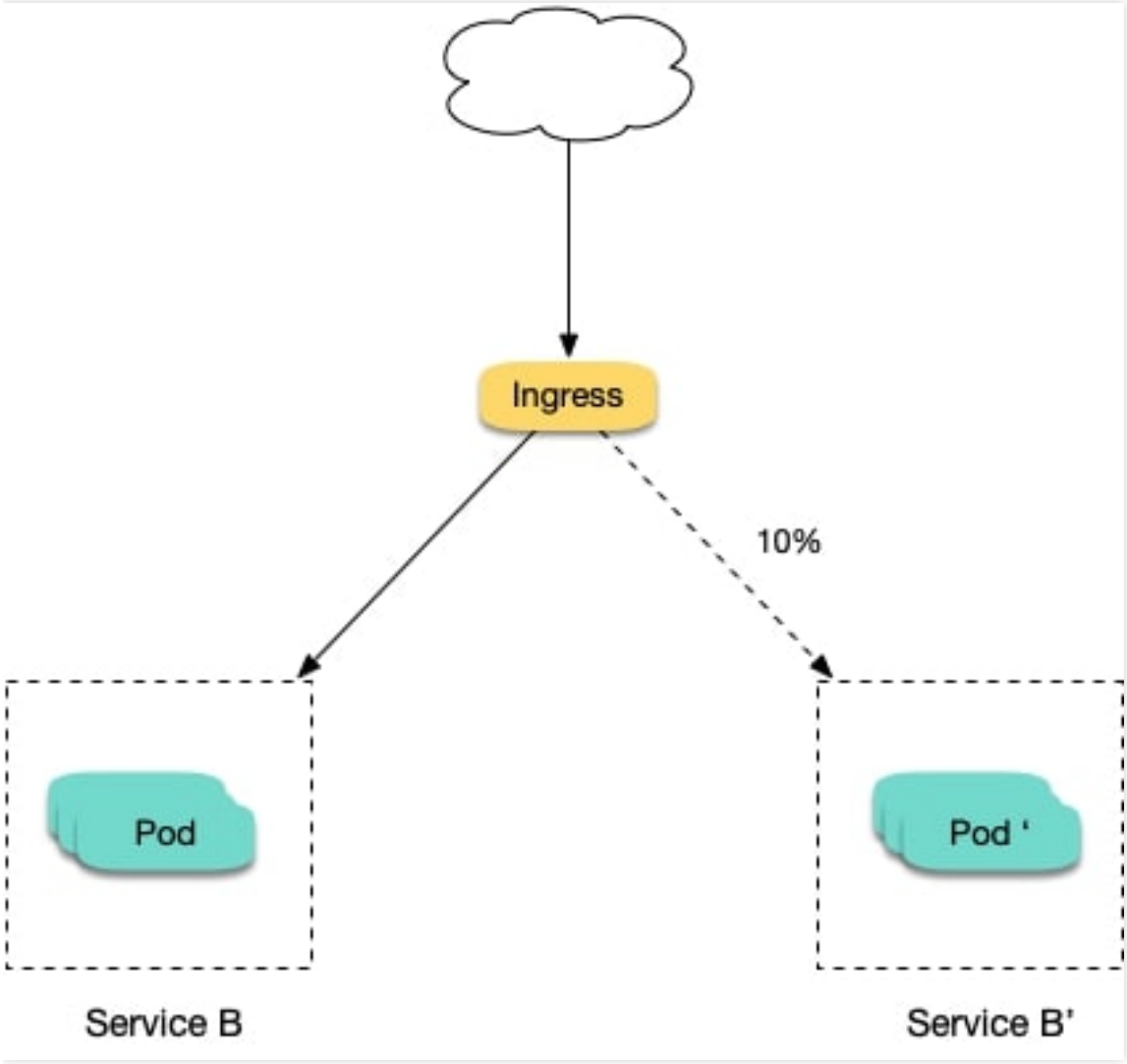


Scenario 2: splitting a certain proportion of traffic to the new version

Assume that Service B, which provides Layer-7 service to external users, has been running online for some time; you have rectified some issues in Service B and to activate a new version, Service B', for some users for beta testing, without replacing the existing Service B; and you want to first split 10% of the traffic to the new version and let it run stably for some time before gradually increasing the proportion of traffic to the new version to ultimately replace and

©2013-2022 Tencent Cloud. All rights reserved.

deactivate the old version smoothly. See the figure below:



Annotation Descriptions

You can implement Canary Release by specifying the annotation supported by Nginx Ingress for Ingress resources. You need to create two Ingresses for the service: one is a regular Ingress, and the other, which carries the fixed annotation of `nginx.ingress.kubernetes.io/canary: "true"` is called a Canary Ingress. The Canary Ingress usually represents the new version of a service. By configuring this and the annotation of the traffic splitting policy, you can implement Canary Release in multiple scenarios. The following section introduces relevant annotations in detail:

- **`nginx.ingress.kubernetes.io/canary-by-header`**
  Indicates that if the request header contains the specified header name and the value is `always` , the request will

©2013-2022 Tencent Cloud. All rights reserved.

be forwarded to the corresponding real server defined by the Ingress. If the value is `never`, it will not be forwarded and can be used for rollback to the old version. In case of other values, this annotation will be ignored.

- **`nginx.ingress.kubernetes.io/canary-by-header-value`**
  This annotation can be a supplement to `canary-by-header`. You can specify a custom value for the request header, including but not limited to `always` or `never`. When the value of the request header matches the specified custom value, the request will be forwarded to the corresponding real server defined by the Ingress. In case of other values, this annotation will be ignored.

- **`nginx.ingress.kubernetes.io/canary-by-header-pattern`**
  This annotation is similar to `canary-by-header-value`. The difference is that this annotation uses a regular expression, instead of a fixed value, to match the value of the request header. If this annotation and `canary-by-header-value` exist at the same time, this annotation will be ignored.

- **`nginx.ingress.kubernetes.io/canary-by-cookie`**
  Similar to `canary-by-header`, this annotation is used for cookie and supports only `always` and `never`.

- **`nginx.ingress.kubernetes.io/canary-weight`**
  Indicates the proportion of the traffic assigned to the Canary Ingress. The value range is [0-100]. For example, the value 10 indicates that 10% of the traffic is assigned to the corresponding real server of the Canary Ingress.

Note：

- The above rules will be assessed according to the priority sequence: `canary-by-header > canary-by-cookie > canary-weight`.
- When an Ingress is marked as the Canary Ingress, all non-Canary annotations, other than `nginx.ingress.kubernetes.io/load-balance` and `nginx.ingress.kubernetes.io/upstream-hash-by`, will be ignored.

Sample

Note：

The following sample uses a TKE cluster as an example. From this sample, you can quickly learn how to implement Canary Release by using Nginx Ingress. Please note:

©2013-2022 Tencent Cloud. All rights reserved.

- For a single service, only one Canary Ingress can be defined, so the real server can only support up to two versions.

2. A domain name must be configured in the Ingress. Otherwise, the Ingress will not work.
3. Even if all traffic is switched to the Canary Ingress, the old version still needs to exist. Otherwise, an error will be reported.

Using YAML to create resources

This document introduces the following two methods for using YAML to deploy workloads and create Services:

- Method 1: on the details page of the TKE or EKS cluster, click **Use YAML to Create Resources** in the upper right corner and input the YAML sample file content in this document to the editing interface.
- Method 2: save the sample YAML as a file and use kubectl to specify the YAML file to create resources, for example, `kubectl apply -f xx.yaml` .

Deploying two versions of a service

1. Deploy the first version of Deployment in the cluster. Here nginx-v1 is used as an example. The YAML sample is as follows:

```
apiVersion: apps/v1
kind: Deployment
metadata:
name: nginx-v1
spec:
replicas: 1
selector:
matchLabels:
app: nginx
version: v1
template:
metadata:
labels:
app: nginx
version: v1
spec:
containers:
- name: nginx
image: "openresty/openresty:centos"
ports:
- name: http
protocol: TCP
containerPort: 80
```

©2013-2022 Tencent Cloud. All rights reserved.

```
volumeMounts:
- mountPath: /usr/local/openresty/nginx/conf/nginx.conf
name: config
subPath: nginx.conf
volumes:
- name: config
configMap:
name: nginx-v1

---

apiVersion: v1
kind: ConfigMap
metadata:
labels:
app: nginx
version: v1
name: nginx-v1
data:
nginx.conf: |-
worker_processes 1;

events {
accept_mutex on;
multi_accept on;
use epoll;
worker_connections 1024;
}

http {
ignore_invalid_headers off;
server {
listen 80;
location / {
access_by_lua '
local header_str = ngx.say("nginx-v1")
';
}
}
}

---

apiVersion: v1
kind: Service
metadata:
name: nginx-v1
```

©2013-2022 Tencent Cloud. All rights reserved.

```
spec:
type: ClusterIP
ports:
- port: 80
protocol: TCP
name: http
selector:
app: nginx
version: v1
```

2. Deploy the second version of Deployment. Here nginx-v2 is used as an example. The YAML sample is as follows:

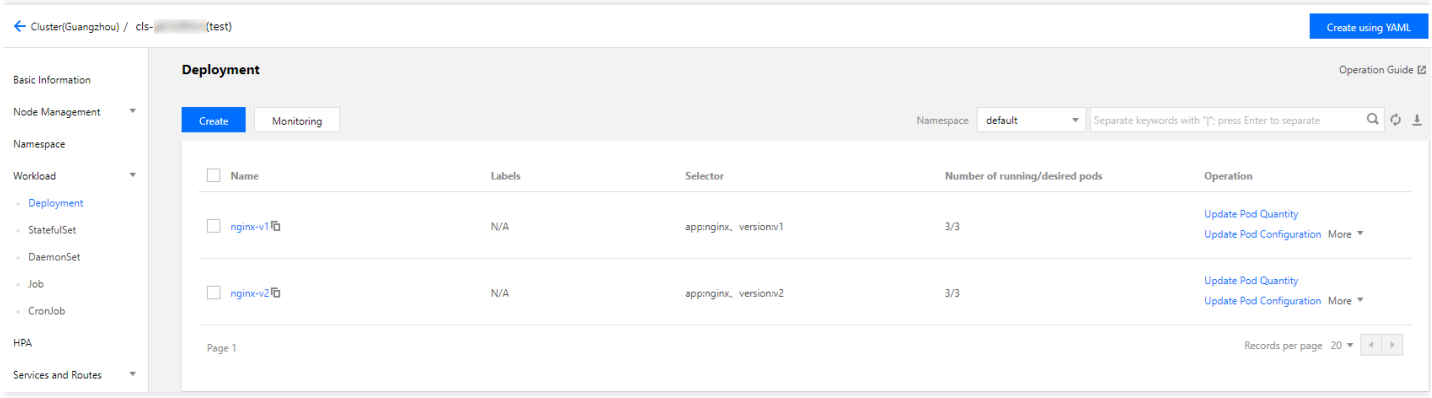
```
apiVersion: apps/v1
kind: Deployment
metadata:
name: nginx-v2
spec:
replicas: 1
selector:
matchLabels:
app: nginx
version: v2
template:
metadata:
labels:
app: nginx
version: v2
spec:
containers:
- name: nginx
image: "openresty/openresty:centos"
ports:
- name: http
protocol: TCP
containerPort: 80
volumeMounts:
- mountPath: /usr/local/openresty/nginx/conf/nginx.conf
name: config
subPath: nginx.conf
volumes:
- name: config
configMap:
name: nginx-v2
---

apiVersion: v1
```

©2013-2022 Tencent Cloud. All rights reserved.

```
kind: ConfigMap
metadata:
labels:
app: nginx
version: v2
name: nginx-v2
data:
nginx.conf: |-
worker_processes 1;

events {
accept_mutex on;
multi_accept on;
use epoll;
worker_connections 1024;
}

http {
ignore_invalid_headers off;
server {
listen 80;
location / {
access_by_lua '
local header_str = ngx.say("nginx-v2")
';
}
}
}

---

apiVersion: v1
kind: Service
metadata:
name: nginx-v2
spec:
type: ClusterIP
ports:
- port: 80
protocol: TCP
name: http
selector:
app: nginx
version: v2
```

You can log in to the TKE Console and go to the workload details page of the cluster to view the deployment information, as shown in the figure below:

©2013-2022 Tencent Cloud. All rights reserved.



3. Create an Ingress, open the service to external access, and point to the v1 service. The YAML sample is as follows:

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
name: nginx
annotations:
kubernetes.io/ingress.class: nginx
spec:
rules:
- host: canary.example.com
http:
paths:
- backend:
serviceName: nginx-v1
servicePort: 80
path: /
```

4. Run the following commands to verify access.

```
curl -H "Host: canary.example.com" http://EXTERNAL-IP # EXTERNAL-IP should be rep
laced with the opened IP address of Nginx Ingress.
```

The returned result is as follows:

```
nginx-v1
```

Traffic splitting based on the header

Create a Canary Ingress, specify the real server of the v2 version, and add an annotation to enable requests with the Region field in the header and the corresponding value of cd or sz to be forwarded to the current Canary Ingress. For

©2013-2022 Tencent Cloud. All rights reserved.

example, if you select users in Chengdu and Shenzhen for the beta test of the new version, the YAML sample is as follows:

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
annotations:
kubernetes.io/ingress.class: nginx
nginx.ingress.kubernetes.io/canary: "true"
nginx.ingress.kubernetes.io/canary-by-header: "Region"
nginx.ingress.kubernetes.io/canary-by-header-pattern: "cd|sz"
name: nginx-canary
spec:
rules:
- host: canary.example.com
http:
paths:
- backend:
serviceName: nginx-v2
servicePort: 80
path: /
```

Run the following commands to perform an access test.

```
$ curl -H "Host: canary.example.com" -H "Region: cd" http://EXTERNAL-IP # EXTERNA
L-IP should be replaced with the opened IP address of Nginx Ingress.
nginx-v2
$ curl -H "Host: canary.example.com" -H "Region: bj" http://EXTERNAL-IP
nginx-v1
$ curl -H "Host: canary.example.com" -H "Region: cd" http://EXTERNAL-IP
nginx-v2
$ curl -H "Host: canary.example.com" http://EXTERNAL-IP
nginx-v1
```

You can see that the v2 service responds only to requests in which the value of the header field `Region` is cd or sz.

Traffic splitting based on cookies

To use cookies, you cannot set a custom value. For example, if you want to select users in Chengdu for the beta test, then only requests with the cookie of `user_from_cd` will be forwarded to the current Canary Ingress. The YAML sample is as follows:

Note：

©2013-2022 Tencent Cloud. All rights reserved.

If you have created a Canary Ingress through the above steps, please delete it and then refer to this step for creation.

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
annotations:
kubernetes.io/ingress.class: nginx
nginx.ingress.kubernetes.io/canary: "true"
nginx.ingress.kubernetes.io/canary-by-cookie: "user_from_cd"
name: nginx-canary
spec:
rules:
- host: canary.example.com
http:
paths:
- backend:
serviceName: nginx-v2
servicePort: 80
path: /
```

Run the following commands to perform an access test.

```
$ curl -s -H "Host: canary.example.com" --cookie "user_from_cd=always" http://EXT
ERNAL-IP # EXTERNAL-IP should be replaced with the opened IP address of Nginx Ing
ress.
nginx-v2
$ curl -s -H "Host: canary.example.com" --cookie "user_from_bj=always" http://EXT
ERNAL-IP
nginx-v1
$ curl -s -H "Host: canary.example.com" http://EXTERNAL-IP
nginx-v1
```

You can view that the v2 service responds only to requests in which the value of the cookie `user_from_cd` is `always`.

Traffic splitting based on service weight

To use a Canary Ingress based on service weight, you only need to specify the proportion of traffic to be imported. For example, to import 10% of traffic to the v2 version, the YAML sample is as follows:

Note：

©2013-2022 Tencent Cloud. All rights reserved.

If you have created a Canary Ingress through the above steps, please delete it and then refer to this step to create a new one.

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
annotations:
kubernetes.io/ingress.class: nginx
nginx.ingress.kubernetes.io/canary: "true"
nginx.ingress.kubernetes.io/canary-weight: "10"
name: nginx-canary
spec:
rules:
- host: canary.example.com
http:
paths:
- backend:
serviceName: nginx-v2
servicePort: 80
path: /
```

Run the following commands to perform an access test.

```
$ for i in {1..10}; do curl -H "Host: canary.example.com" http://EXTERNAL-IP; don
e;
nginx-v1
nginx-v1
nginx-v1
nginx-v1
nginx-v1
nginx-v1
nginx-v2
nginx-v1
nginx-v1
nginx-v1
```

You can see that the chance of the v2 service responding is 10%, which corresponds to the 10% service weight setting.

References

- Official Documentation of Nginx Ingress Canary Annotations
- Deploying Nginx Ingress on TKE

©2013-2022 Tencent Cloud. All rights reserved.

Logs
Best Practice in TKE Log Collection

Last updated：2021-07-15 14:18:35

Overview

This document introduces the log-related features of TKE, including log collection, storage, and query, and provides suggestions based on actual application scenarios.

Note：

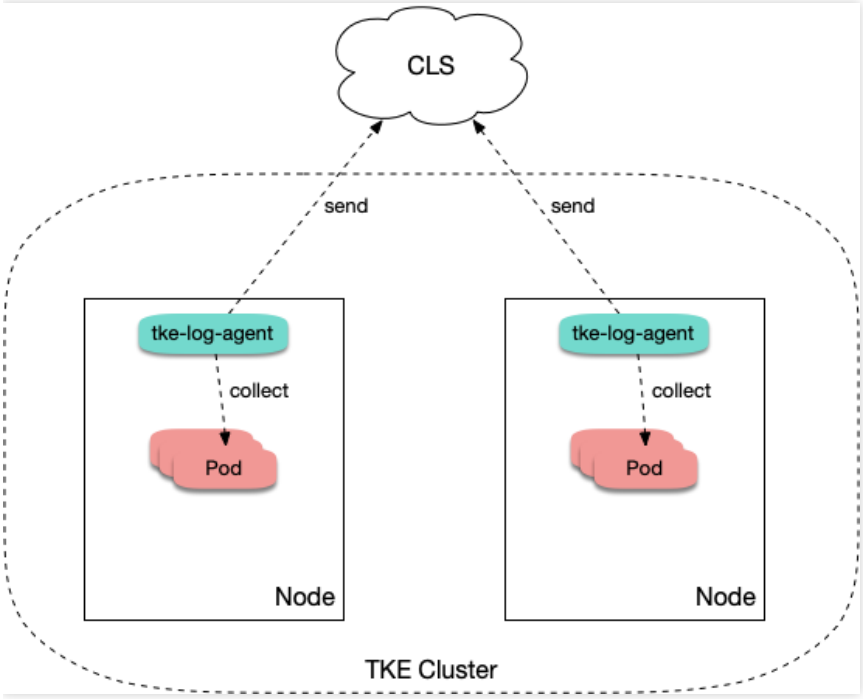
- This document is only applicable to TKE clusters.
- For more information on how to enable log collection for a TKE cluster and its basic usage, see Log Collection.

Architecture

After log collection is enabled for a TKE cluster, tke-log-agent is deployed on each node as a DaemonSet. According to the collection rules, tke-log-agent collects container logs from each node and reports them to CLS for storage,

©2013-2022 Tencent Cloud. All rights reserved.

indexing and analysis. See the figure below:



Use Cases of Collection Types

To use the TKE log collection feature, you need to determine the target data source for collection when creating log collection rules. TKE supports collection of standard output, files in a container and files on a host. See below for more details.

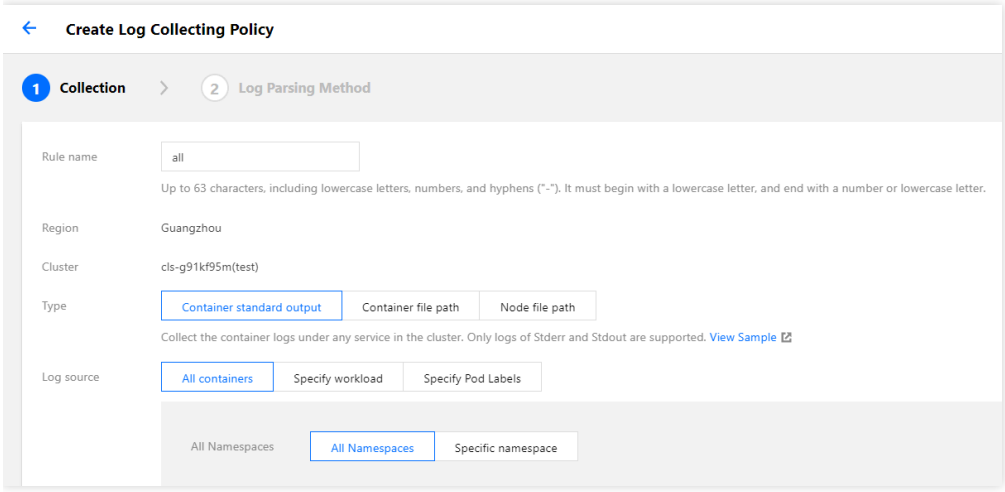
Collecting standard output

If you choose to collect from standard output, logs of containers in a pod are written to the standard output, and the log content will be managed by the container runtime (Docker or Containerd). We recommend using standard out as it is the simplest collection mode. Its advantages are as follows:

1. No extra volume mounting is needed.
2. You can view the log content by simply running `kubectl logs` .
3. No worries about log rotation. The container runtime will perform storage and automatic rotation of logs to prevent situations where the disk capacity is exhausted because some pods write excessive logs.
4. You don't need to worry about the log file path. You can use unified collection rules to cover a wide range of workloads and reduce operation complexity.

The following figure shows a sample collection configuration. For more information on configuration, see Collecting standard output logs of a container.

©2013-2022 Tencent Cloud. All rights reserved.



Collecting log files in container

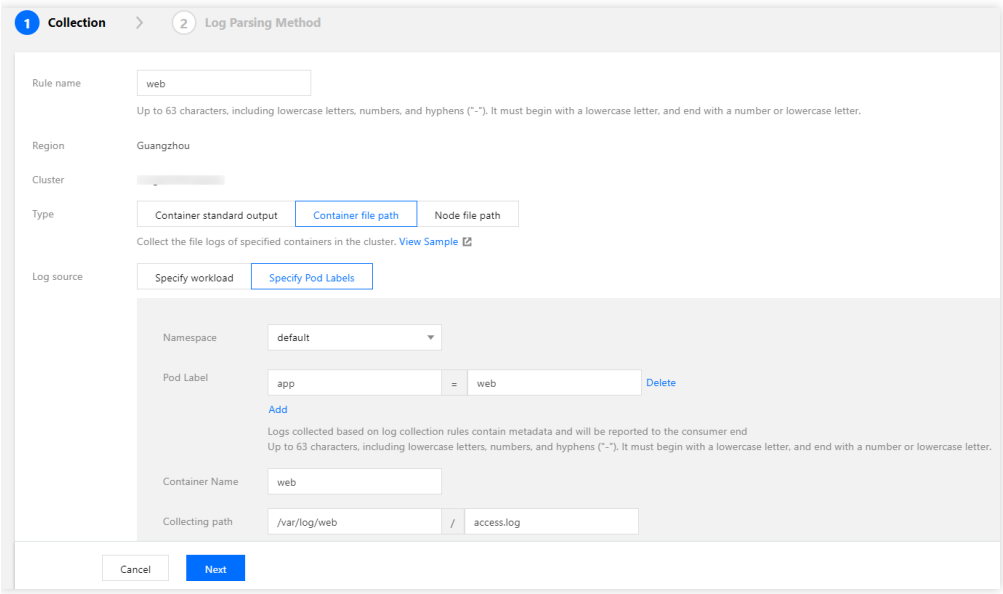
Usually logs are written into log files. When containers are used, log files are written in containers. Please note:

- If no volume is mounted in the log file path:
  Log files will be written to the container writable layer and stored in the container data disk. Usually, the path is `/var/lib/docker`. We recommend that you mount a volume to this path, and the volume should not be used for the system disk. After the container stops, the logs will be cleared.
- If a volume is mounted in the log file path:
  Log files will be stored in the backend storage of the corresponding volume type. Usually, emptydir is used. After the container stops, the logs will be cleared. During runtime, log files will be stored in `/var/lib/kubelet` of the host. This path usually does not have a mounted disk, so it will use the system disk. As unified storage is available when using the log collection feature, you are not advised to mount other persistent storage to store log files (such as CBS, COS, or CFS).

Most open-source log collectors require you to mount a volume to the pod log file path before collection, but TKE log collection does not require mounting. To output logs to files in containers, you do not need to consider whether to mount a volume. The following figure shows a sample collection configuration. For more information on configuration,

©2013-2022 Tencent Cloud. All rights reserved.

see Collecting File Logs in a Container.



Collecting files on the host

If businesses need to write logs into log files and you hope to retain the original log files as a backup after the container stops to avoid complete log loss in the event of collection exceptions, you can mount a hostPath to the log file path. This way, log files will be stored in the specified directory on the host and these log files will not be cleared after the container stops.

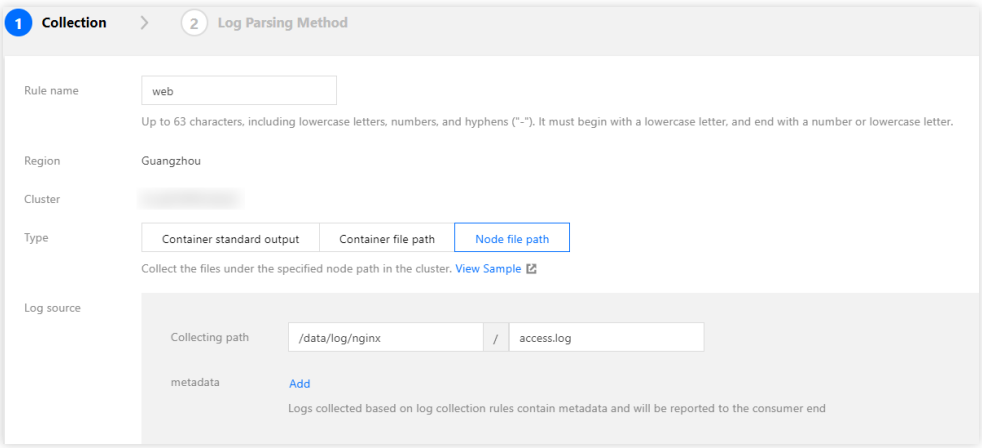
As log files are not automatically cleared, the issue of repeated collection may occur if a pod is scheduled to another container and then scheduled back to the original container causing log files to be written into the same path. In that case, there are two collection scenarios:

- Same file name:
  For example, assume the fixed file path is `/data/log/nginx/access.log`. In this case, repeated collection will not occur, because the collector will remember the time point of previously collected log files and collect only increments.
- Different file names:
  Usually, the log frameworks used by businesses automatically perform log rotation periodically, generally on a daily basis, and automatically rename old log files and add the timestamp suffix. If the collection rules use `*` as the wildcard character to match log file names, repeated collection may occur. After the log framework renames log files, the collector will mistakenly think it has found new log files, so it will collect the files again.

> Note：
> Usually, repeated collection will not occur. If the log framework automatically performs rotation, we recommend that the wildcard character `*` not be used to match log files.

©2013-2022 Tencent Cloud. All rights reserved.

The following figure shows a sample collection configuration. For more information on configuration, see Collecting

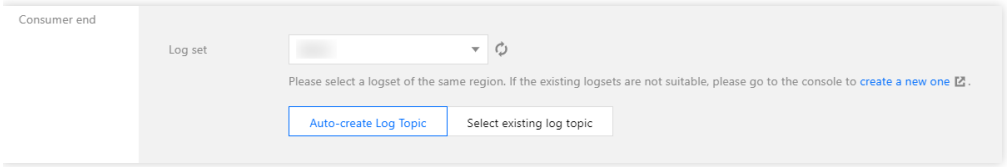


file logs in specified node paths.

Log Output

TKE log collection is integrated with CLS on the cloud, and log data is reported to CLS. CLS manages logs based on logsets and log topics. A logset is a project management unit of CLS and can include multiple log topics. Usually, the logs of the same business are put in the same logset, and applications or services of the same type in the same business use the same log topic.

In TKE, log collection rules have a one-to-one correspondence with log topics. When selecting the consumer during the creation of TKE log collection rules, you need to specify the logset and log topic. Logsets are usually created in advance, and you can choose to automatically create log topics. See the figure below:



After a log topic is automatically created, you can go to Logset Management, open the details page of the corresponding logset, and rename the log topic to make it easier to find in future searches.

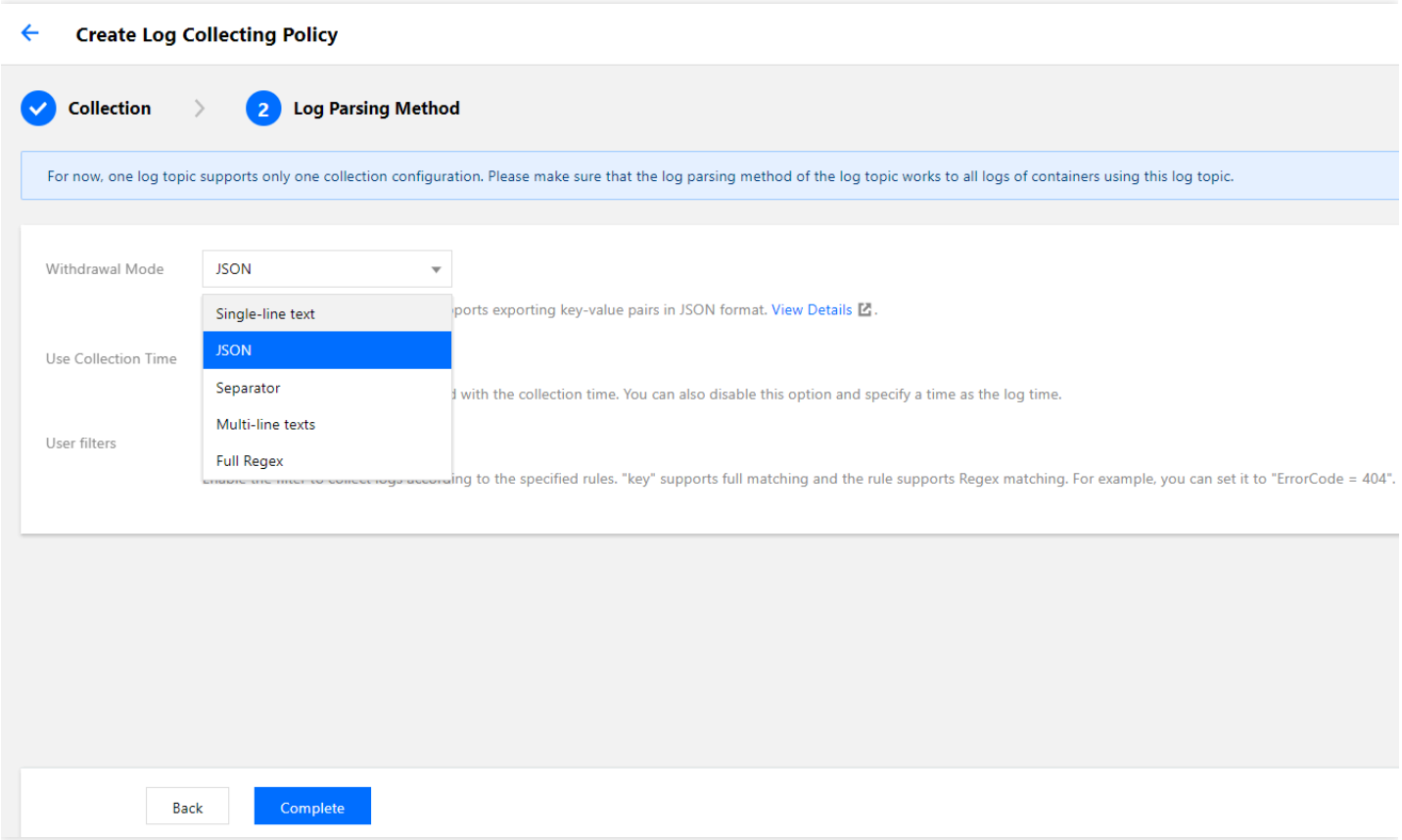
Configuring Log Format Parsing

When creating a log collection rule, you need to configure the log parsing format to facilitate future searches. Please refer to the following sections to complete configuration based on the actual situation.

Selecting an extraction mode

©2013-2022 Tencent Cloud. All rights reserved.

TKE supports five extraction modes: single-line text, JSON, separator, multi-line text, and full RegEx, as shown in the figure below:



- JSON mode
- Single-line text and multi-line text modes
- Separator and full RegEx modes

You can only select JSON mode when logs are output in JSON format, in which case this mode is recommended. In JSON format, the logs are already structured, allowing CLS to extract the JSON key as the field name and value as the corresponding key value. This means you do not have to configure complex matching rules based on the business log output format. A sample of such logs is as follows:

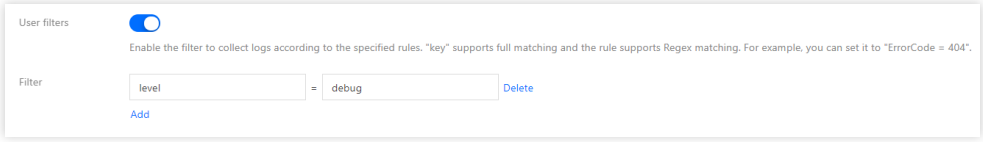
```
{"remote_ip":"10.135.46.111","time_local":"22/Jan/2019:19:19:34 +0800","body_sen
t":23,"responsetime":0.232,"upstreamtime":"0.232","upstreamhost":"unix:/tmp/php-c
gi.sock","http_host":"127.0.0.1","method":"POST","url":"/event/dispatch","reques
t":"POST /event/dispatch HTTP/1.1","xff":"-","referer":"http://127.0.0.1/my/cours
e/4","agent":"Mozilla/5.0 (Windows NT 10.0; WOW64; rv:64.0) Gecko/20100101 Firefo
x/64.0","response_code":"200"}
```

Configuring the content to be filtered out

©2013-2022 Tencent Cloud. All rights reserved.

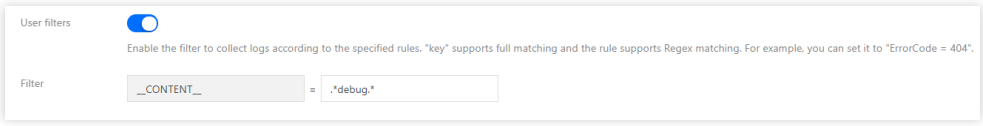
You can choose to filter out useless log information to lower costs.

- If you use the JSON, separator, or full RegEx extraction mode, the log content is structured, and you can specify fields to perform regular expression matching for the log content to be retained, as shown in the figure below:



- If you use the single-line text or multi-line text extraction mode, the log content is not structured, so you cannot specify fields for filtering. Usually, you can use regular expressions to perform fuzzy matching on the full log content to be retained, as shown in the figure below:

> Note：
> The content should be matched using a regular expression, instead of perfect match. For example, to retain only the domain name `a.test.com` in a log, the expression for matching should be `a\.test\.com`, instead of `a.test.com`.



Customizing log timestamps

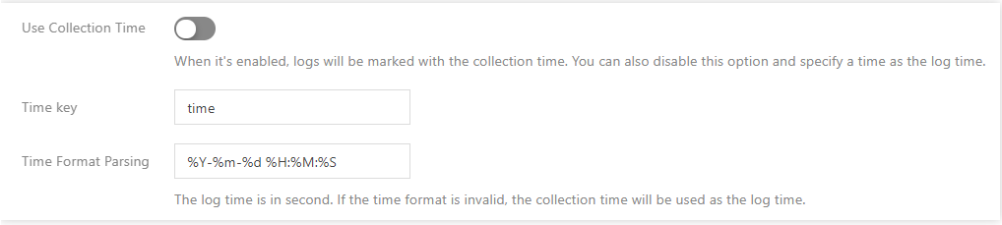
Each log should contain a timestamp used mainly for searching. This allows users to select a time period during searches. By default, the log timestamp is determined by the collection time, but you can customize it by selecting a certain field as the timestamp. This can allow for more precise searches. For example, assume that a service has been running for some time before you create a collection rule. If you do not set a custom time format, the timestamps of old logs will be set to the current time during collection, resulting in inaccurate timestamps.

As the single-line text and multi-line text extraction modes do not structure log content, no field can be specified as the timestamp, which means these two modes do not support this feature. Other extraction modes support this feature. You need to disable "Use collection time", select a field name as the timestamp, and configure the time format. For example, assuming that the `time` field is used as the timestamp and the `time` value of a log is `2020-09-22 18:18:18`, you can set the time format as: `%Y-%m-%d %H:%M:%S`, as shown in the figure below:

> Note：

©2013-2022 Tencent Cloud. All rights reserved.

The CLS timestamps currently support precision to the second. If the timestamp field of a business log is precise to the millisecond, you cannot use custom timestamps and can only use the default timestamp determined by the collection time.



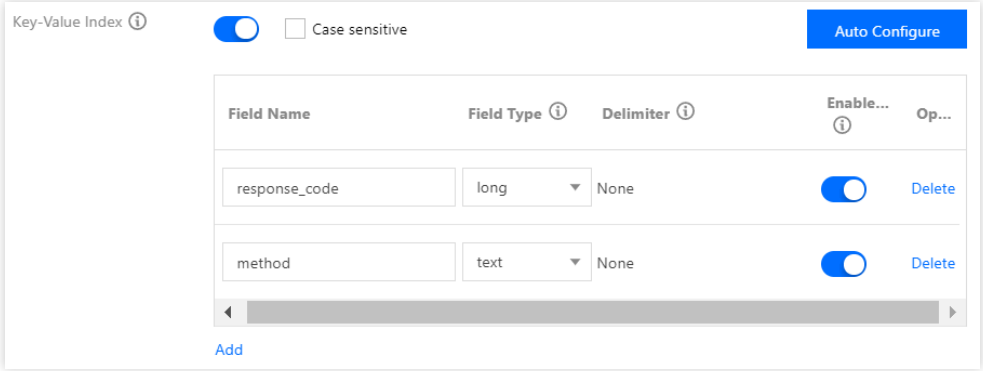
For more information on the time format configuration, see Configuring the Time Format.

Log Query

After log collection rules are configured, the collector will automatically start collecting logs and report them to CLS. You can query logs in **Search Analysis** on the CLS console. After an index is enabled, the Lucene syntax is supported. There are three types of indexes, as follows:

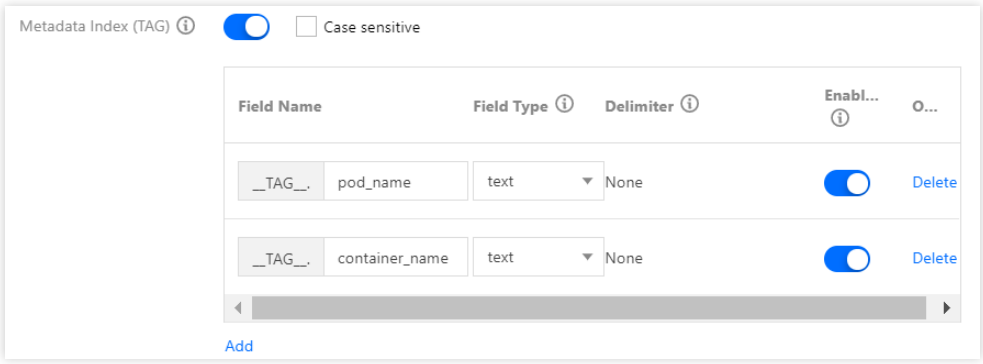
- Full-text index: used for fuzzy search. You do not need to specify a field. See the figure below:

- Key value index: index for structured log content. You can specify log fields to search. See the figure below:



©2013-2022 Tencent Cloud. All rights reserved.

- Metadata field index: when some extra fields, such as pod name and namespace, are automatically attached during log reporting, this index allows you to specify these fields during search. See the figure below:

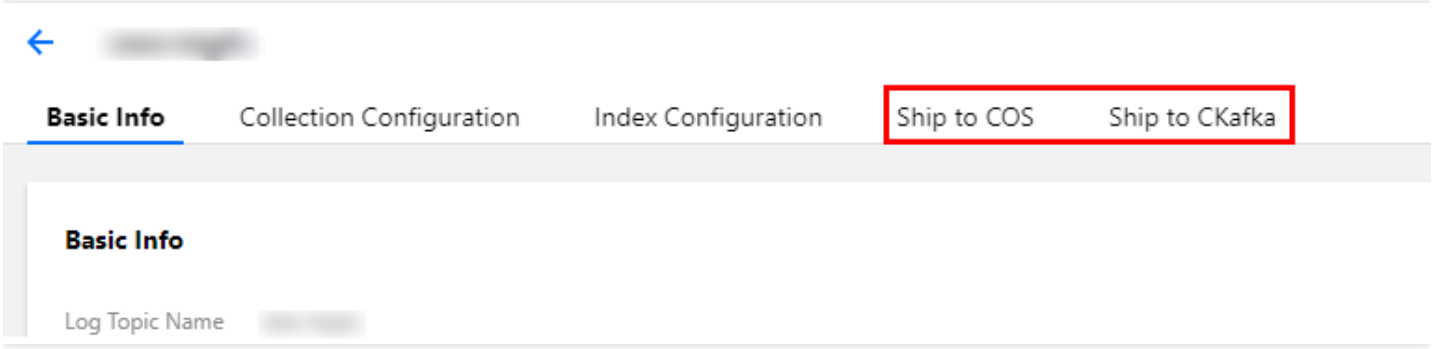


The following figure shows a query sample:

Publishing Logs to COS and Ckafka

CLS allows logs to be published to COS and the message queue CKafka. You can set it in the log topic, as shown in the figure below:

©2013-2022 Tencent Cloud. All rights reserved.



This is applicable to the following scenarios:

- Scenarios where long-term archiving and storage of log data are required. The logset stores log data for seven days by default. You can adjust the duration. The larger the data volume, the higher the cost. Usually, data is stored for a few days. If you need to store logs for a longer period, you can publish log data to COS for low-cost storage.
- Scenarios where further processing (such as offline calculation) of logs is required. You can publish log data to COS or Ckafka to be consumed and processed by other programs.

References

- TKE: Log Collection User Guide
- CLS: Configuring the Time Format
- CLS: Shipping to COS
- CLS: Shipping to Ckafka

©2013-2022 Tencent Cloud. All rights reserved.

Implementing Multi-line Log Merging for TKE Serverless Log Collection

Last updated：2023-05-09 16:53:02

Overview

When the TKE Serverless log collection is enabled via environment variables, the log extraction defaults to the single-line extraction mode. If the log data of the client occupies multiple lines (such as the Java program log), the line break `\n` cannot be used to mark the end of a log. To help the CLS to clearly distinguish each log, it is necessary to configure a configmap with the first-line regular expression. When a log in a line matches the preset regular expression, it is considered as the beginning of a log, and the next matching line will be the end mark of the log. This document describes how to merge multi-line logs when you use environment variables to enable TKE Serverless log collection.

Directions

Raw log sample

```
2020-09-24 16:09:07 ERROR System.out(4844) java.lang.NullPointerException
at com.temp.ttscancel.MainActivity.onCreate(MainActivity.java:43)
at android.app.Activity.performCreate(Activity.java:5248)
at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1110) at
android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2162) at and
roid.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2257)
at android.app.ActivityThread.access$800(ActivityThread.java:139)
at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1210)
```

Creating Configmap

For the sample raw log file, the content of the parser.conf configuration file is:

```
apiVersion: v1
data:
parser.conf: |-
[PARSER]
Name parser_name
Format regex
Regex ^(?<timestamp>[0-9]{2,4}\-[0-9]{1,2}\-[0-9]{1,2} [0-9]{1,2}\:[0-9]{1,2}\:[0
```

©2013-2022 Tencent Cloud. All rights reserved.

```
-9]{1,2}) (?<message>.*)
kind: ConfigMap
metadata:
name: cm
namespace: default
```

Among them, `parser_name` needs to be set in the annotation (spec.template.metadata.annotations) when the workload is created. Run the following command to set:

```
eks.tke.cloud.tencent.com/parser-name: parser_name
```

For more information about `parser.conf` configuration file, see Regular Expression.

Mounting Configmap when creating a Pod

You need to perform the following operations when creating a Pod:

1. Mount the created Configmap as a Volume.
2. For how to enable log collection via environment variables, see Log Collection.
3. Specify two annotations.

```
eks.tke.cloud.tencent.com/parser-name: "parser_name"
eks.tke.cloud.tencent.com/volume-name-for-parser: "volume-name"
```

- `eks.tke.cloud.tencent.com/parser-name` refers to the name of the created Configmap.
- `eks.tke.cloud.tencent.com/volume-name-for-parser` refers to the name of the Volume mounted in the Pod, which can be customized.

Pod yaml template

```
apiVersion: apps/v1
kind: Deployment
metadata:
labels:
k8s-app: multiline
qcloud-app: multiline
name: multiline
namespace: default
spec:
replicas: 1
selector:
matchLabels:
k8s-app: multiline
qcloud-app: multiline
```

©2013-2022 Tencent Cloud. All rights reserved.

```
template:
metadata:
annotations:
eks.tke.cloud.tencent.com/parser-name: parser_name
eks.tke.cloud.tencent.com/volume-name-for-parser: volume-name
labels:
k8s-app: multiline
qcloud-app: multiline
spec:
containers:
- env:
- name: EKS_LOGS_OUTPUT_TYPE
value: cls
- name: EKS_LOGS_LOG_PATHS
value: stdout
- name: EKS_LOGS_TOPIC_ID
value: topic-id
- name: EKS_LOGS_LOGSET_NAME
value: eks
- name: EKS_LOGS_SECRET_ID
valueFrom:
secretKeyRef:
key: SecretId
name: cls
optional: false
- name: EKS_LOGS_SECRET_KEY
valueFrom:
secretKeyRef:
key: SecretKey
name: cls
optional: false
image: nginx
imagePullPolicy: Always
name: ng
resources:
limits:
cpu: 500m
memory: 1Gi
requests:
cpu: 250m
memory: 256Mi
volumeMounts:
- mountPath: /mnt
name: volume-name
imagePullSecrets:
- name: qcloudregistrykey
restartPolicy: Always
```

©2013-2022 Tencent Cloud. All rights reserved.

```
volumes:
- configMap:
defaultMode: 420
name: cm
name: volume-name
```

Structured log sample

```
2020-09-24 16:09:07 ERROR System.out(4844) java.lang.NullPointerException \at com
.temp.ttscancel.MainActivity.onCreate(MainActivity.java:43) \at android.app.Activ
ity.performCreate(Activity.java:5248) \at android.app.Instrumentation.callActivit
yOnCreate(Instrumentation.java:1110) \at android.app.ActivityThread.performLaunch
Activity(ActivityThread.java:2162) \at android.app.ActivityThread.handleLaunchAct
ivity(ActivityThread.java:2257) \at android.app.ActivityThread.access$800(Activit
yThread.java:139) \at android.app.ActivityThread$H.handleMessage(ActivityThread.j
ava:1210)
```

©2013-2022 Tencent Cloud. All rights reserved.

Custom Nginx Ingress Log

Last updated：2023-05-06 17:36:46

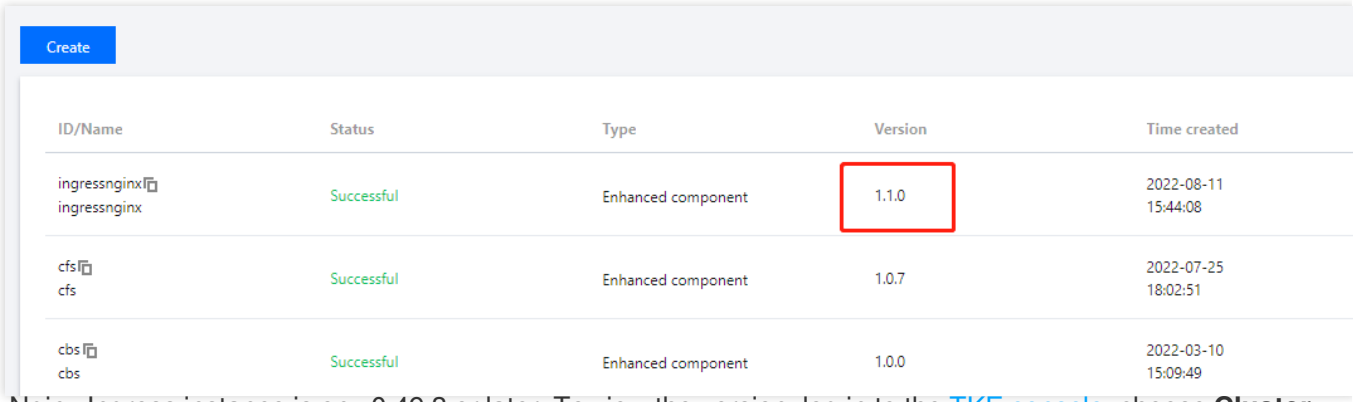
By integrating CLS, TKE provides a complete set of productized capabilities to collect and consume Nginx Ingress logs. For more information, see Nginx-ingress Log Configuration. If the default log index setting does not meet your needs, you can customize the setting. This document describes how to modify the log index setting of Nginx Ingress.

Prerequisites

1. Nginx Ingress v1.1.0 or later is used. To view the version of the Nginx Ingress add-on, log in to the TKE console and choose **Cluster details** > **Add-on management**.

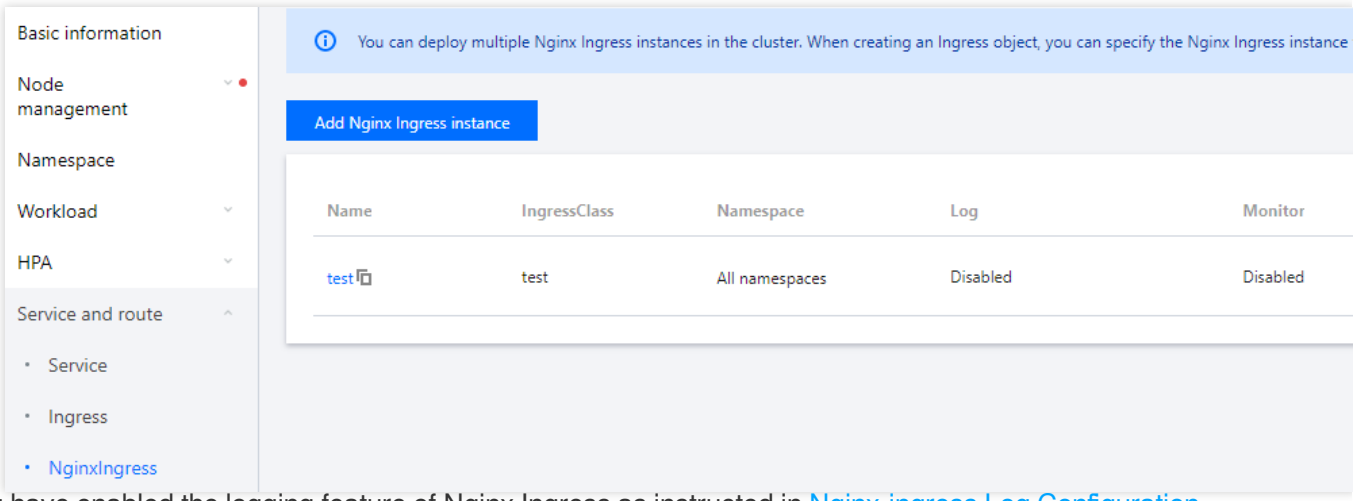
Note:

Only Nginx Ingress v1.1.0 or later supports this feature. For earlier versions, such as Nginx Ingress v1.0.0, the add-on will roll back the log index setting modified by users.



2. The Nginx Ingress instance is on v0.49.3 or later. To view the version, log in to the TKE console, choose **Cluster details** > **Services and Routes** > **NginxIngress**, and click **View YAML** on the right of the target instance. In the YAML file, the `ccr.ccs.tencentyun.com/paas/nginx-ingress-controller` image must be on v0.49.3 or later.

©2013-2022 Tencent Cloud. All rights reserved.



3. You have enabled the logging feature of Nginx Ingress as instructed in Nginx-ingress Log Configuration.

Directions

Note

To modify the log structure, you need to understand the log stream of Nginx Ingress, which consists of log output, collection, indexing, and configuration. Here, if log output or collection is missing or incorrectly configured, log structure modification will fail.

Step 1. Modify the log output format of the Nginx Ingress instance

The log configuration of the Nginx Ingress instance is in the master configuration ConfigMap named in the format of `Instance Name-ingress-nginx-controller`. In the ConfigMap, you need to modify the `log-format-upstream` key.

©2013-2022 Tencent Cloud. All rights reserved.

```
apiVersion: v1
data:
  access-log-path: /var/log/nginx/nginx_access.log
  allow-snippet-annotations: "false"
  error-log-path: /var/log/nginx/nginx_error.log
  keep-alive-requests: "10000"
  log-format-upstream: $remote_addr - $remote_user [$time_iso8601] $msec "$request"
    $status $body_bytes_sent "$http_referer" "$http_user_agent" $request_length $request_time
    [$proxy_upstream_name] [$proxy_alternative_upstream_name] [$upstream_addr] [$upstream_response_
    [$upstream_response_time] [$upstream_status] $req_id $service_name $namespace
  max-worker-connections: "65536"
  upstream-keepalive-connections: "200"
kind: ConfigMap
metadata:
  creationTimestamp: "2022-07-22T02:56:35Z"
  labels:
    k8s-app:       s-ingress-nginx-controller
    qcloud-app:          -ingress-nginx-controller
  managedFields:
  - apiVersion: v1
    fieldsType: FieldsV1
    fieldsV1:
      f:data:
        .: {}
        f:access-log-path: {}
        f:allow-snippet-annotations: {}
        f:error-log-path: {}
        f:keep-alive-requests: {}
        f:max-worker-connections: {}
```

Sample

Add two consecutive strings `$namespace` and `$service_name` to the end of a log.

```
apiVersion: v1
data:
  access-log-path: /var/log/nginx/nginx_access.log
  allow-snippet-annotations: "false"
  error-log-path: /var/log/nginx/nginx_error.log
  keep-alive-requests: "10000"
  log-format-upstream: $remote_addr - $remote_user [$time_iso8601] $msec "$request"
    $status $body_bytes_sent "$http_referer" "$http_user_agent" $request_length $request_time
    [$proxy_upstream_name] [$proxy_alternative_upstream_name] [$upstream_addr] [$upstream_response_le
    [$upstream_response_time] [$upstream_status] $req_id $service_name $namespace
  max-worker-connections: "65536"
  upstream-keepalive-connections: "200"
kind: ConfigMap
```

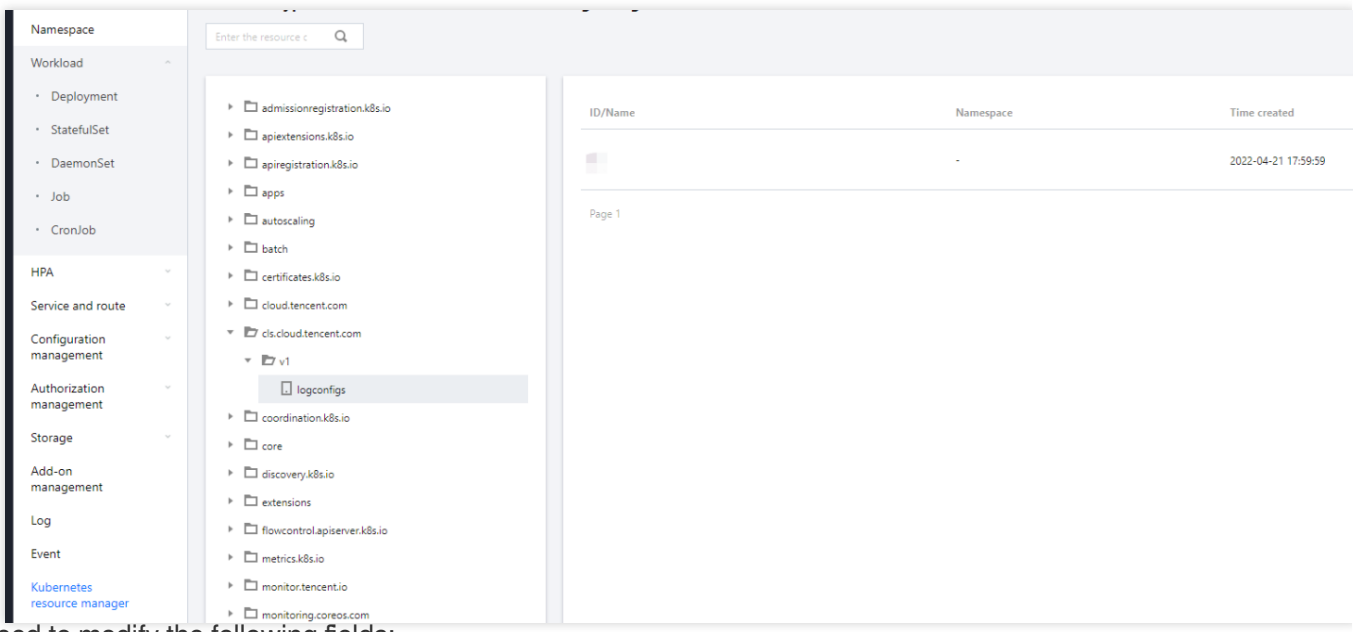
For more information about fields in Nginx Ingress logs, see Log format.

Step 2. Modify the format for collecting and reporting cluster logs to Agent

The cluster log collection rules are in a resource object of the `logconfigs.cls.cloud.tencent.com` type.

Log in to the TKE console, choose **Cluster details** > **Kubernetes resource manager**, find the `Instance Name-ingress-nginx-controller` resource object, and click **Edit YAML** to modify it.

©2013-2022 Tencent Cloud. All rights reserved.



You need to modify the following fields:

beginningRegex: Regular expression to match the start of a log.

keys: Log fields.

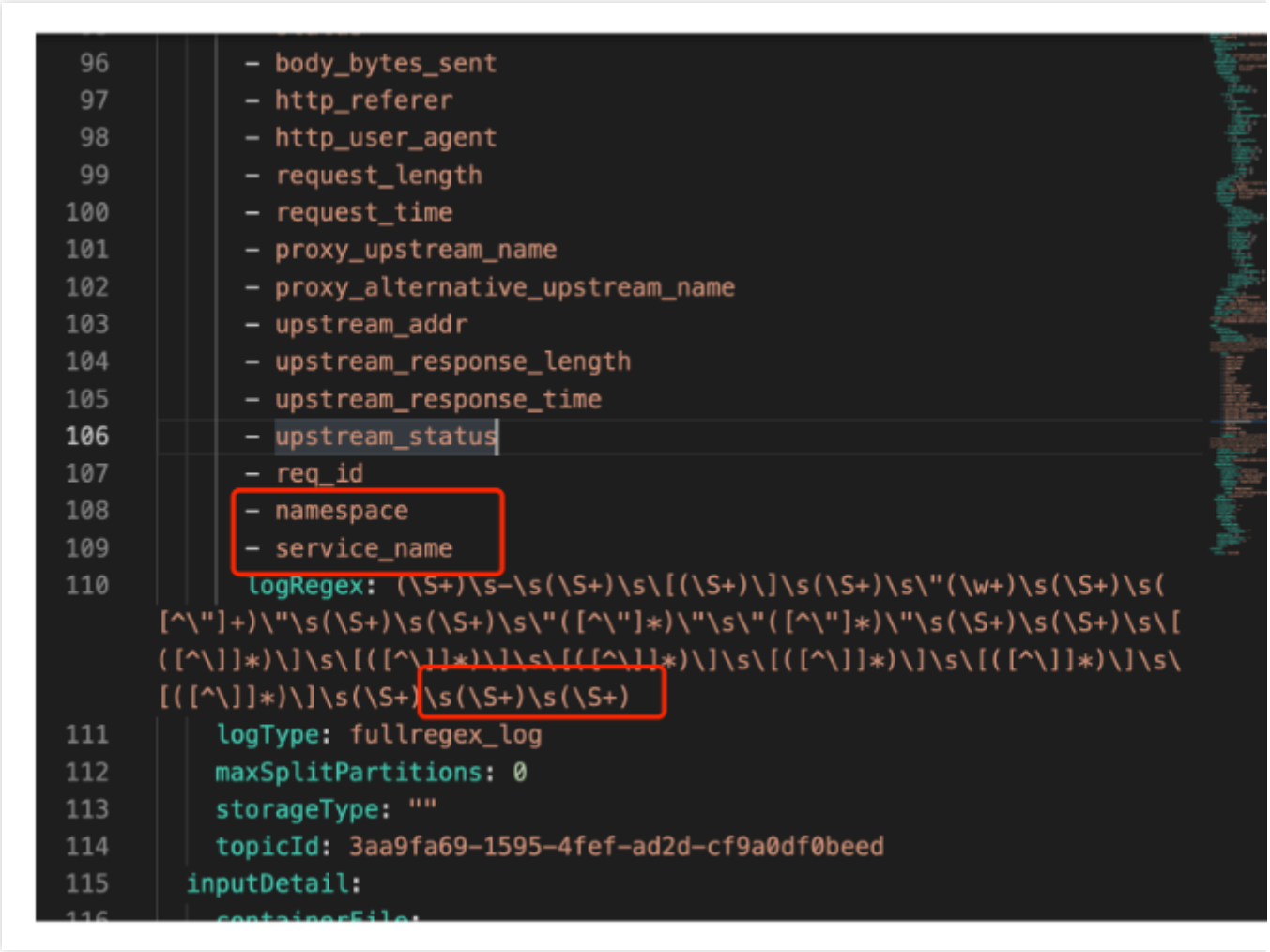
logRegex: Regular expression to match the end of a log.

The regular expressions match the Nginx log row format. We recommend you add the fields to the existing Nginx log format, declare them at the end of `keys`, and add their regular expression parsing results to the end of `beginningRegex` and `logRegex` respectively.

Sample

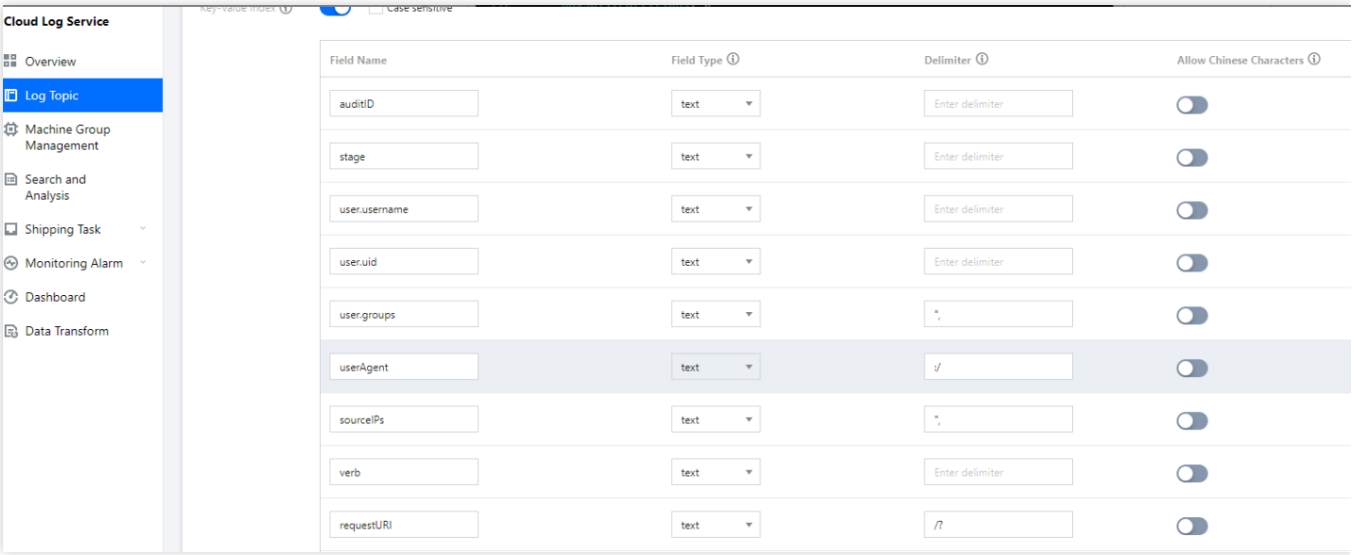
Add two keys in Step 1 to the end of `keys` and add the regular expression strings to the end of `beginningRegex` and `logRegex` respectively:

©2013-2022 Tencent Cloud. All rights reserved.



(Optional) Step 3. Modify the log index format of CLS

To search for a field, you need to add the index of the new field in the corresponding log topic in the CLS console as instructed in Configuring Index. Then, all collected logs can be searched for by the index.



©2013-2022 Tencent Cloud. All rights reserved.

Restoring the Initial Settings

As log rule modification is complicated and involves regular expressions, any incorrect step can cause log collection failure. If a log collection error occurs, we recommend you restore to the initial log collection capabilities by disabling the log collection feature and then enabling it again.

©2013-2022 Tencent Cloud. All rights reserved.

Monitoring
Using Prometheus to monitor Java applications

Last updated：2020-11-23 18:25:51

Overview

The Prometheus community developed JMX Exporter for exporting JVM monitoring metrics so that Prometheus can be used to collect monitoring data. After your Java business is containerized into Kubernetes, you can learn how to use Prometheus and JMX Exporter to monitor Java applications by reading this document.

Introduction to JMX Exporter

Java Management Extensions (JMX) is an extended framework for Java management. Based on this framework, JMX Exporter reads the runtime status of JVMs. JMX Exporter utilizes the JMX mechanism of Java to read JMX runtime monitoring data and then converts the data to metrics that can be recognized by Prometheus. In this way, you can use Prometheus to collect the monitoring data.

JMX Exporter provides two methods for opening JVM monitoring metrics: **independent process launch** and **JVM in-process launch**:

1. Independent process launch
Parameters are specified during JVM launch to open the RMI API of JMX. JMX Exporter calls RMI to obtain JVM runtime status data, converts the data to Prometheus metrics, and opens the port to allow collection by Prometheus.
2. JVM in-process launch
Parameters are specified during JVM launch to run the jar package of JMX Exporter as a javaagent. JVM runtime status data is read in-process and then converted to Prometheus metrics, and the port is opened to allow collection by Prometheus.

Note：

We do not recommend the **independent process launch** method, because it requires complicated configuration and involves an independent process. The monitoring of the process itself can incur new problems. In this document, the **JVM in-process launch** method is used as an example, in which JMX Exporter is used in the Kubernetes environment to open JVM monitoring metrics.

©2013-2022 Tencent Cloud. All rights reserved.

Directions

Opening JVM monitoring metrics by using JMX Exporter

Packaging images

When using the JVM in-process launch method to launch JVM, you need to specify the jar package and configuration files of JMX Exporter. The jar package is a binary file that is difficult to mount with configmap. We recommend that you directly package the jar package and configuration file of JMX Exporter into a business container image. The process is as follows:

1. Create a directory for producing images and place the JMX Exporter configuration file `prometheus-jmx-config.yaml` into the directory.

```
ssl: false
lowercaseOutputName: false
lowercaseOutputLabelNames: false
```

> ⚠ **Note：**
>
> For more configuration items, refer to the official Prometheus document.

2. Prepare a jar package file. To do this, go to the GitHub page of jmx_exporter to obtain the download address of the latest jar package and run the following command to download the package to the created directory.

```
wget https://repo1.maven.org/maven2/io/prometheus/jmx/jmx_prometheus_javaagent/0.13.0/jmx_prometheus_javaagent-0.13.0.jar
```

3. Prepare a Dockerfile. This document uses Tomcat as an example.

```
FROM tomcat:jdk8-openjdk-slim
ADD prometheus-jmx-config.yaml /prometheus-jmx-config.yaml
ADD jmx_prometheus_javaagent-0.13.0.jar /jmx_prometheus_javaagent-0.13.0.jar
```

4. Run the following command to compile the image.

```
docker build . -t ccr.ccs.tencentyun.com/imroc/tomcat:jdk8
```

©2013-2022 Tencent Cloud. All rights reserved.

Now, you have completed image packaging. You can also use the docker multi-stage building feature and skip the step of manually downloading the jar package. The following shows a sample Dockerfile:

```
FROM ubuntu:16.04 as jar
WORKDIR /
RUN apt-get update -y
RUN DEBIAN_FRONTEND=noninteractive apt-get install -y wget
RUN wget https://repo1.maven.org/maven2/io/prometheus/jmx/jmx_prometheus_javaag
ent/0.13.0/jmx_prometheus_javaagent-0.13.0.jar
FROM tomcat:jdk8-openjdk-slim
ADD prometheus-jmx-config.yaml /prometheus-jmx-config.yaml
COPY --from=jar /jmx_prometheus_javaagent-0.13.0.jar /jmx_prometheus_javaagent-
0.13.0.jar
```

Deploying Java applications

When an application is deployed in Kubernetes, you must modify JVM launch parameters in order to load JMX Exporter during launch. During launch, JVM reads the `JAVA_OPTS` environmental variable as an extra launch parameter. During deployment, you can add this environmental variable for the application. The following shows an example:

```
apiVersion: apps/v1
kind: Deployment
metadata:
name: tomcat
spec:
replicas: 1
selector:
matchLabels:
app: tomcat
template:
metadata:
labels:
app: tomcat
spec:
containers:
- name: tomcat
image: ccr.ccs.tencentyun.com/imroc/tomcat:jdk8
env:
- name: JAVA_OPTS
value: "-javaagent:/jmx_prometheus_javaagent-0.13.0.jar=8088:/prometheus-jmx-conf
ig.yaml"

---
```

©2013-2022 Tencent Cloud. All rights reserved.

```
apiVersion: v1
kind: Service
metadata:
name: tomcat
labels:
app: tomcat
spec:
type: ClusterIP
ports:
- port: 8080
protocol: TCP
name: http
- port: 8088
protocol: TCP
name: jmx-metrics
selector:
app: tomcat
```

- Launch parameter format: `-javaagent:<jar>=<port>:<config>`
- In this example, port 8088 is used to open the monitoring metrics of JVM. You can use another port as needed.

Adding a Prometheus monitoring configuration

Configure Prometheus to enable monitoring data collection. The following shows an example:

```
- job_name: tomcat
scrape_interval: 5s
kubernetes_sd_configs:
- role: endpoints
namespaces:
names:
- default
relabel_configs:
- action: keep
source_labels:
- __meta_kubernetes_service_label_app
regex: tomcat
- action: keep
source_labels:
- __meta_kubernetes_endpoint_port_name
regex: jmx-metrics
```

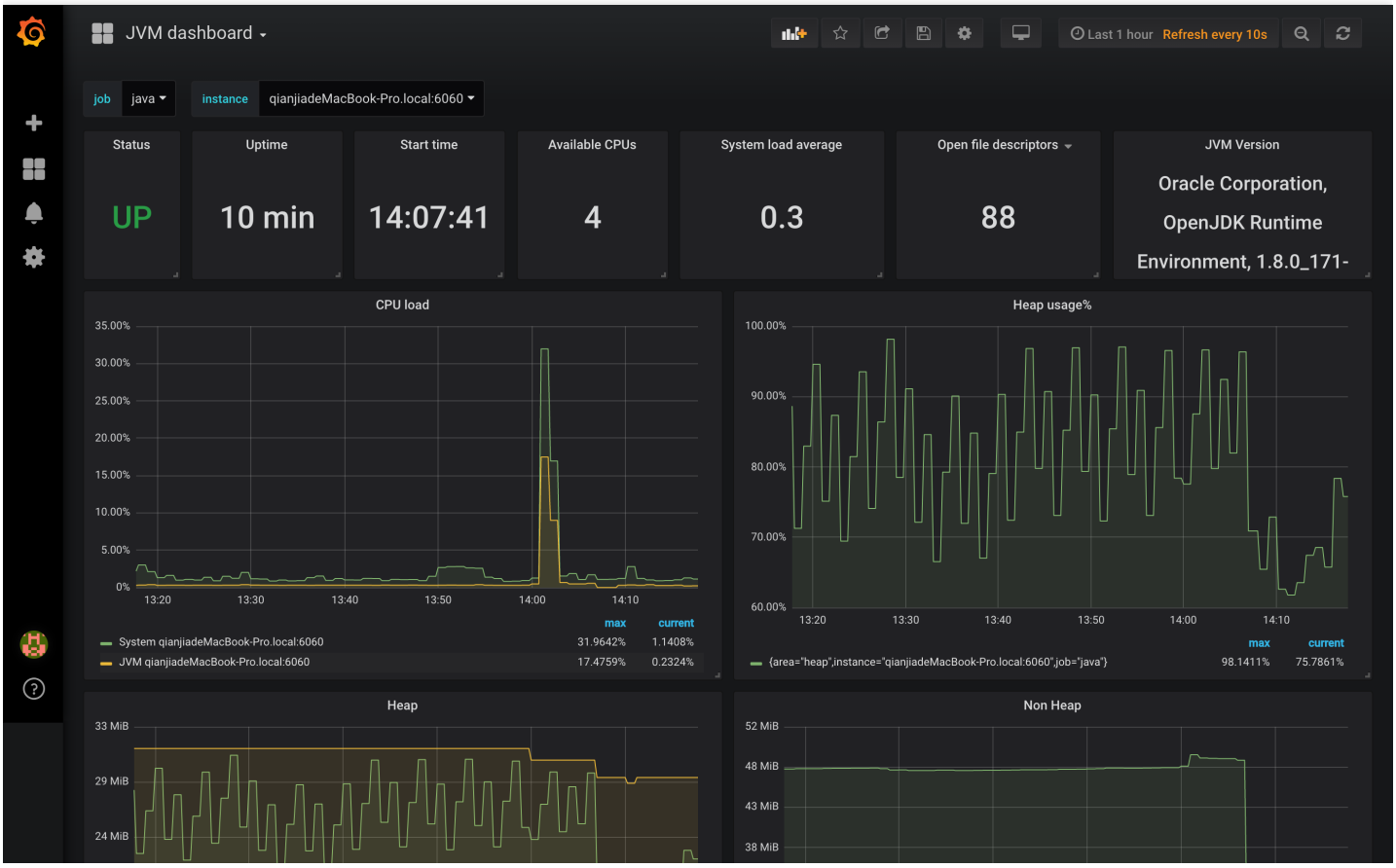
If prometheus-operator has been installed, you can create a CRD object of ServiceMonitor to configure Prometheus. The following shows an example:

©2013-2022 Tencent Cloud. All rights reserved.

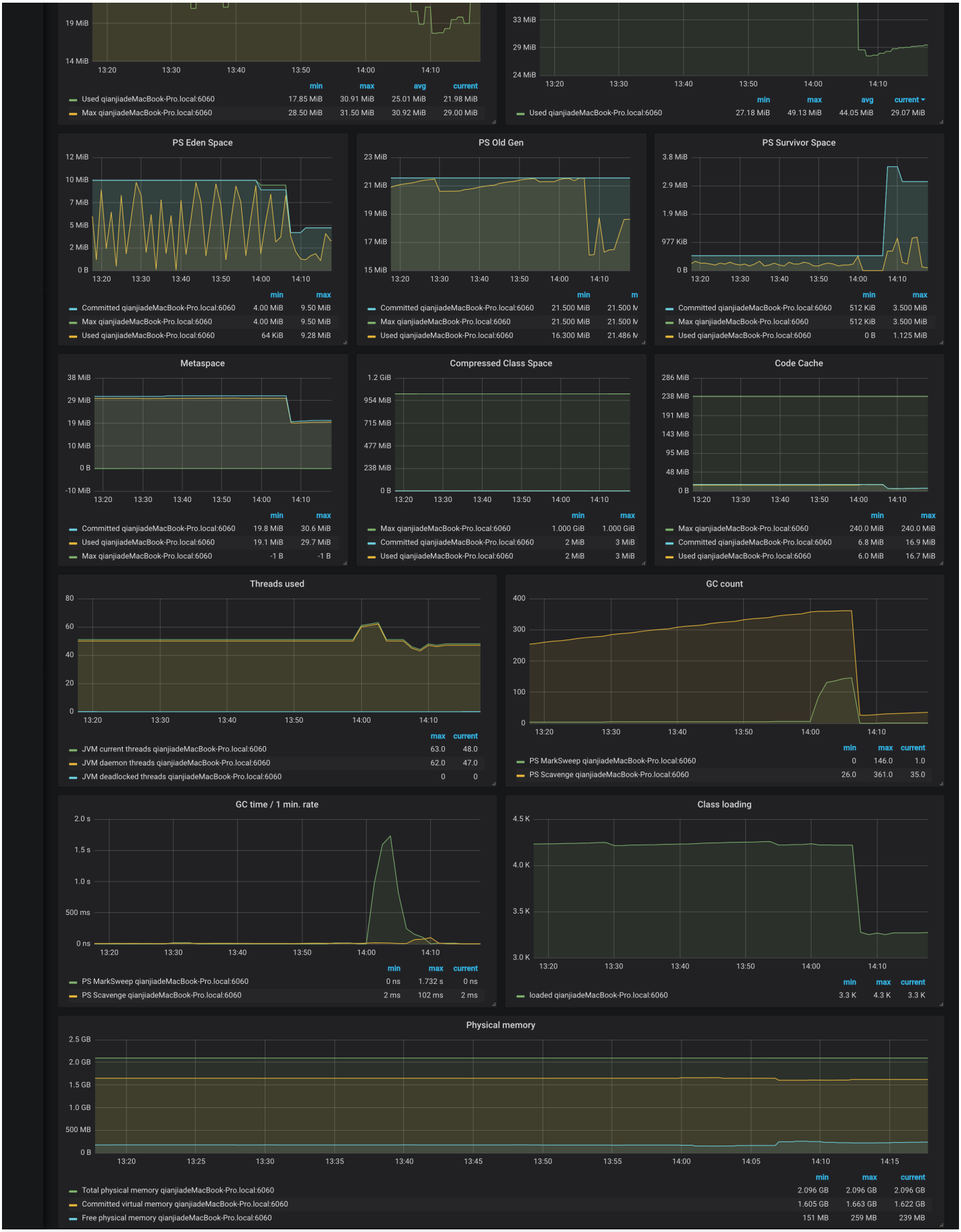
```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
name: tomcat
namespace: default
labels:
app: tomcat
spec:
endpoints:
- port: jmx-metrics
interval: 5s
namespaceSelector:
matchNames:
- default
selector:
matchLabels:
app: tomcat
```

Adding a Grafana monitoring dashboard

Collected data can be displayed. If you are familiar with Prometheus and Grafana, you can design a dashboard based on your specific metrics. Alternatively, you can use the dashboards provided by the community, such as the JVM dashboard. This dashboard can be directly imported for use. The following figure shows the dashboard view:



©2013-2022 Tencent Cloud. All rights reserved.



©2013-2022 Tencent Cloud. All rights reserved.

References

- JMX Exporter
- JVM Monitoring Dashboard

©2013-2022 Tencent Cloud. All rights reserved.

Using Prometheus to Monitor MySQL and MariaDB

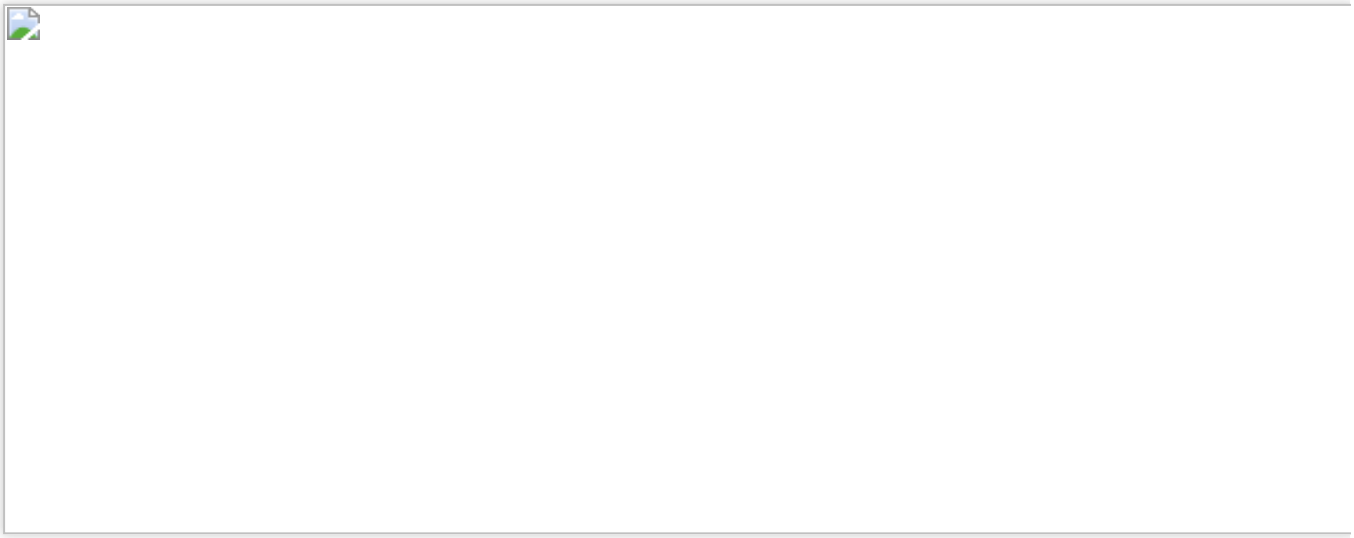
Last updated：2023-03-14 18:19:11

Overview

MySQL is a common relational database management system. As a branch of MySQL, MariaDB is compatible with MySQL and is becoming increasingly popular. In a Kubernetes environment, you can use Prometheus to monitor MySQL and MariaDB database using the open-source MySQL exporter. This document describes how to use Prometheus to monitor MySQL and MariaDB.

Introduction to MySQL Exporter

The MySQL exporter reads database status data from MySQL or MariaDB, converts it to Prometheus metric format, and opens it to the HTTP interface. In this case, Prometheus can collect and monitor these metrics.



Directions

Deploying the MySQL exporter

Note

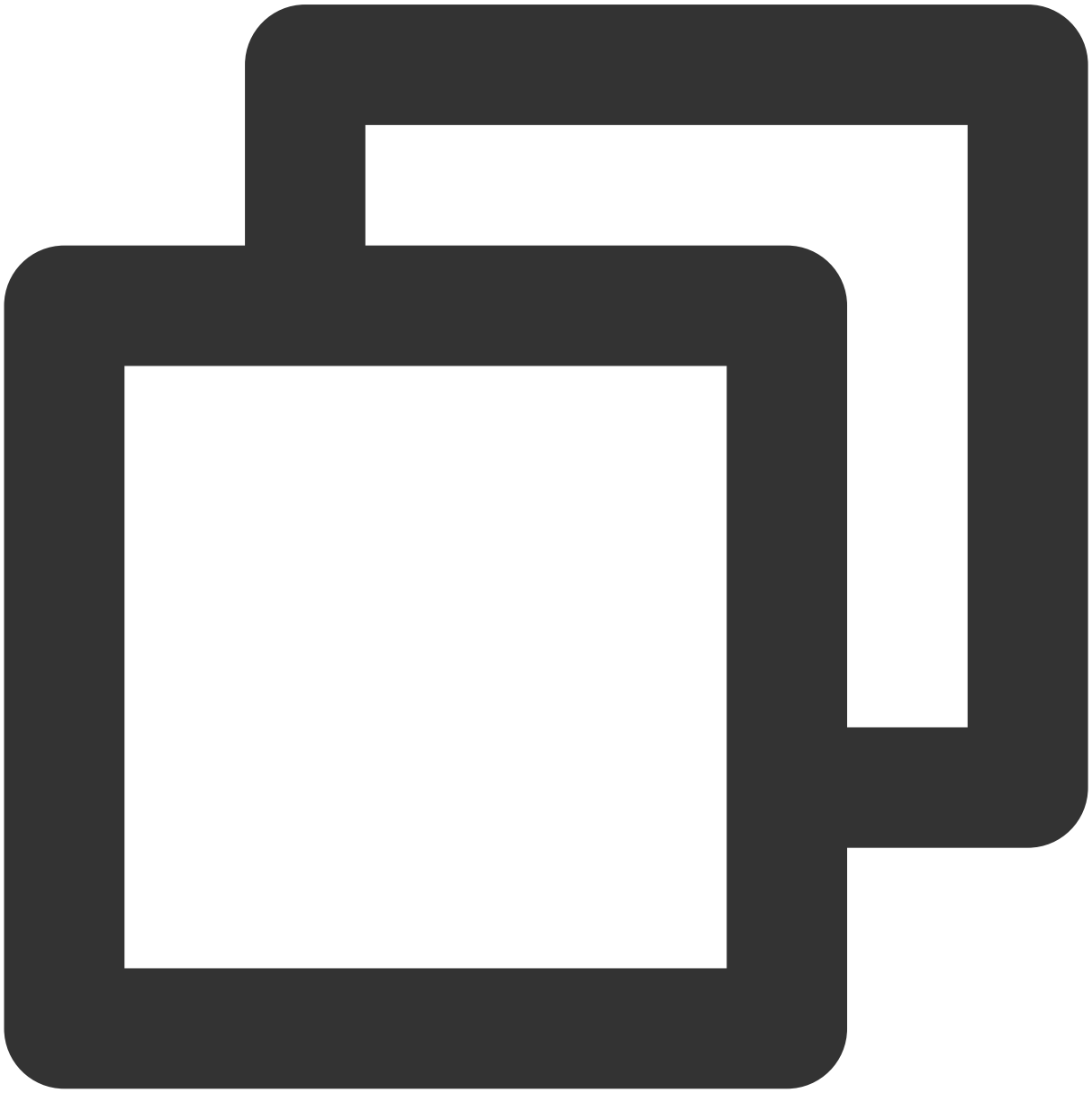
Before deploying the MySQL exporter, ensure that MySQL or MariaDB has been deployed in the cluster, outside the cluster, or in the cloud service used.

©2013-2022 Tencent Cloud. All rights reserved.

Deploying MySQL

The following example shows how to deploy MySQL to a cluster from the Marketplace.

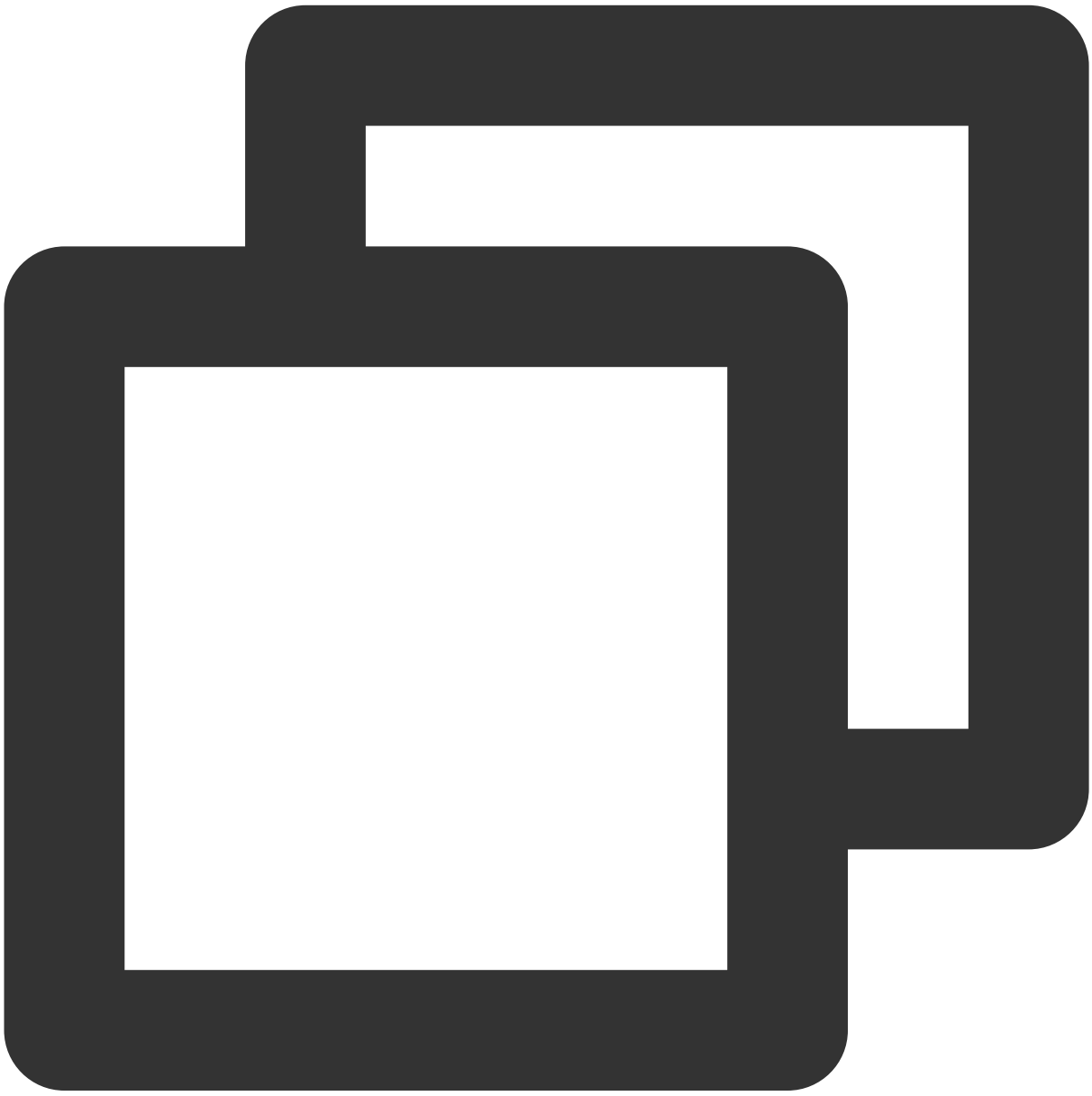
1. Log in to the TKE console and select **Marketplace** in the left sidebar.
2. On the **Marketplace** page, search for and click **MySQL**.
3. On the **Application Details** page, click **Create Application**.
4. On the **Create Application** page, enter the necessary information and click **Create**.
5. After the application is created, select **Application** in the left sidebar and view the details of the application on the page displayed.
6. Run the following command to check whether MySQL runs properly:



©2013-2022 Tencent Cloud. All rights reserved.

```
$ kubectl get pods
NAME                     READY   STATUS         RESTARTS   AGE
mysql-698b898bf7-4dc5k   1/1     Running        0          11s
```

7. Run the following command to obtain the root password:



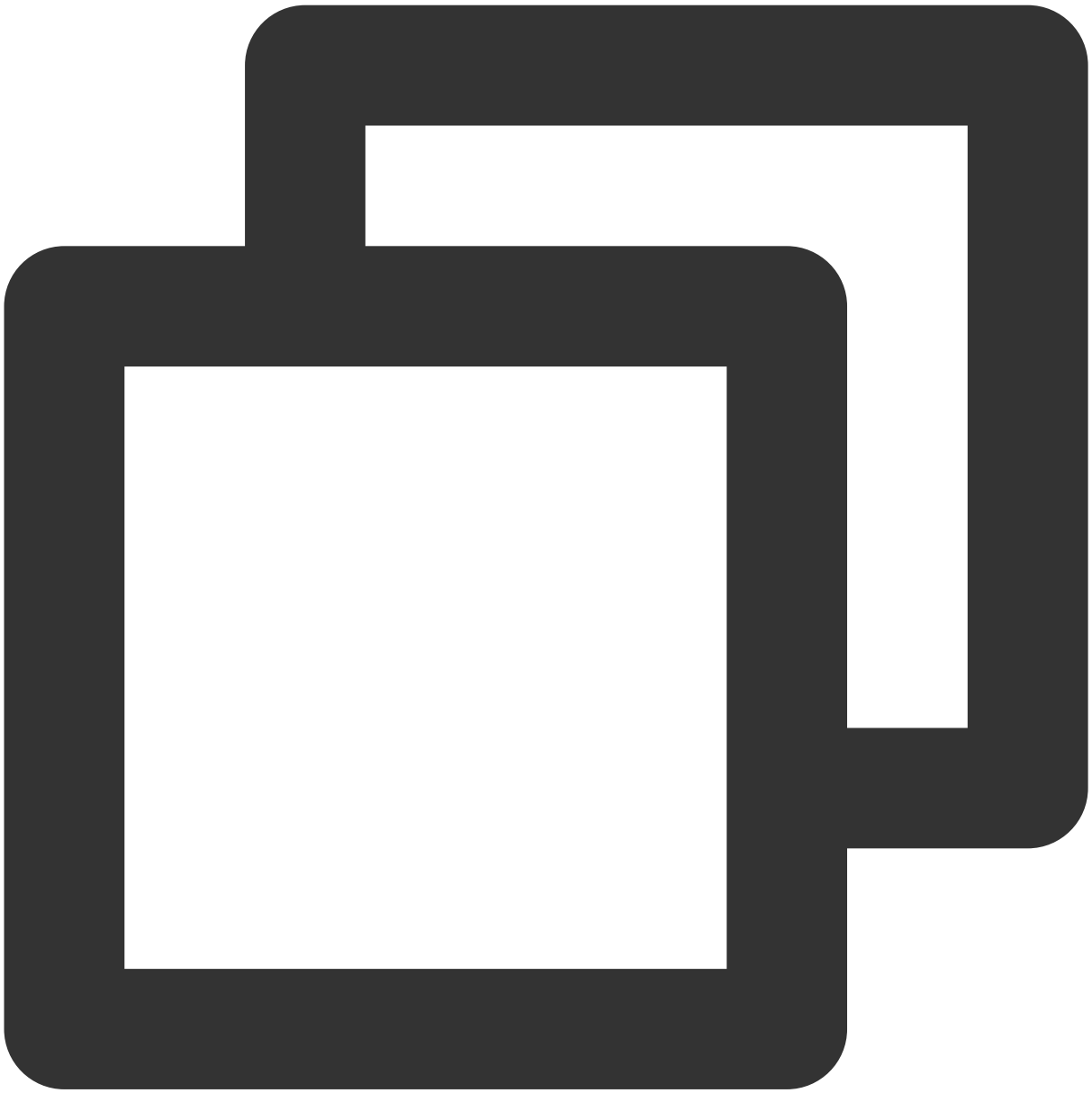
```
$ kubectl get secret -o jsonpath={.data.mysql-root-password} mysql | base64 -d
6ZAj33yLBo
```

Deploying the MySQL exporter

©2013-2022 Tencent Cloud. All rights reserved.

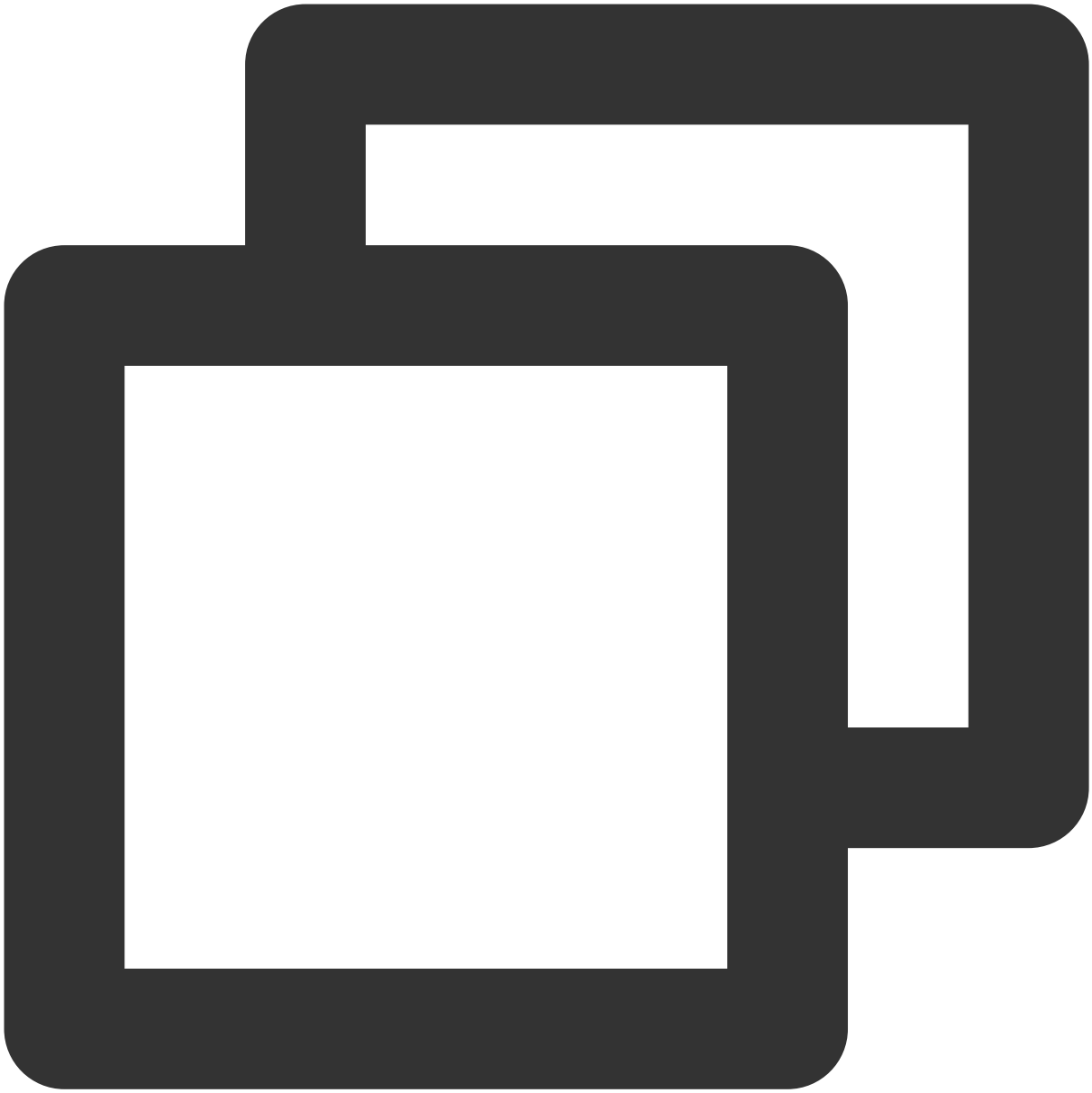
After deploying MySQL, deploy the MySQL exporter as follows:

1. Run the following commands in sequence to create a MySQL exporter account and log in to MySQL:



```
$ kubectl exec -it mysql-698b898bf7-4dc5k bash
```

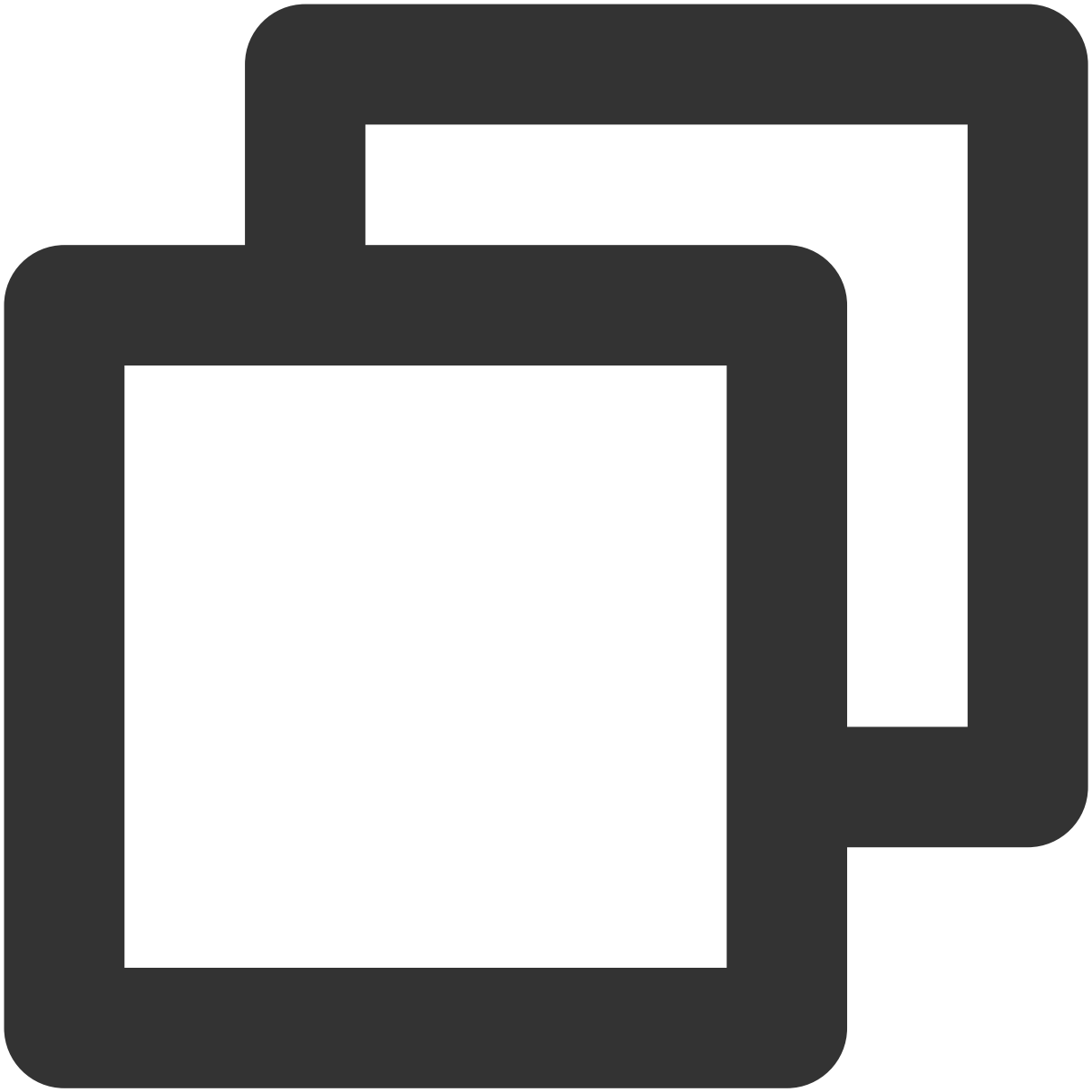
©2013-2022 Tencent Cloud. All rights reserved.



```
$ mysql -uroot -p6ZAj33yLBo
```

2. Run the following command to create an account. `mysqld-exporter/123456` is used as an example.

©2013-2022 Tencent Cloud. All rights reserved.



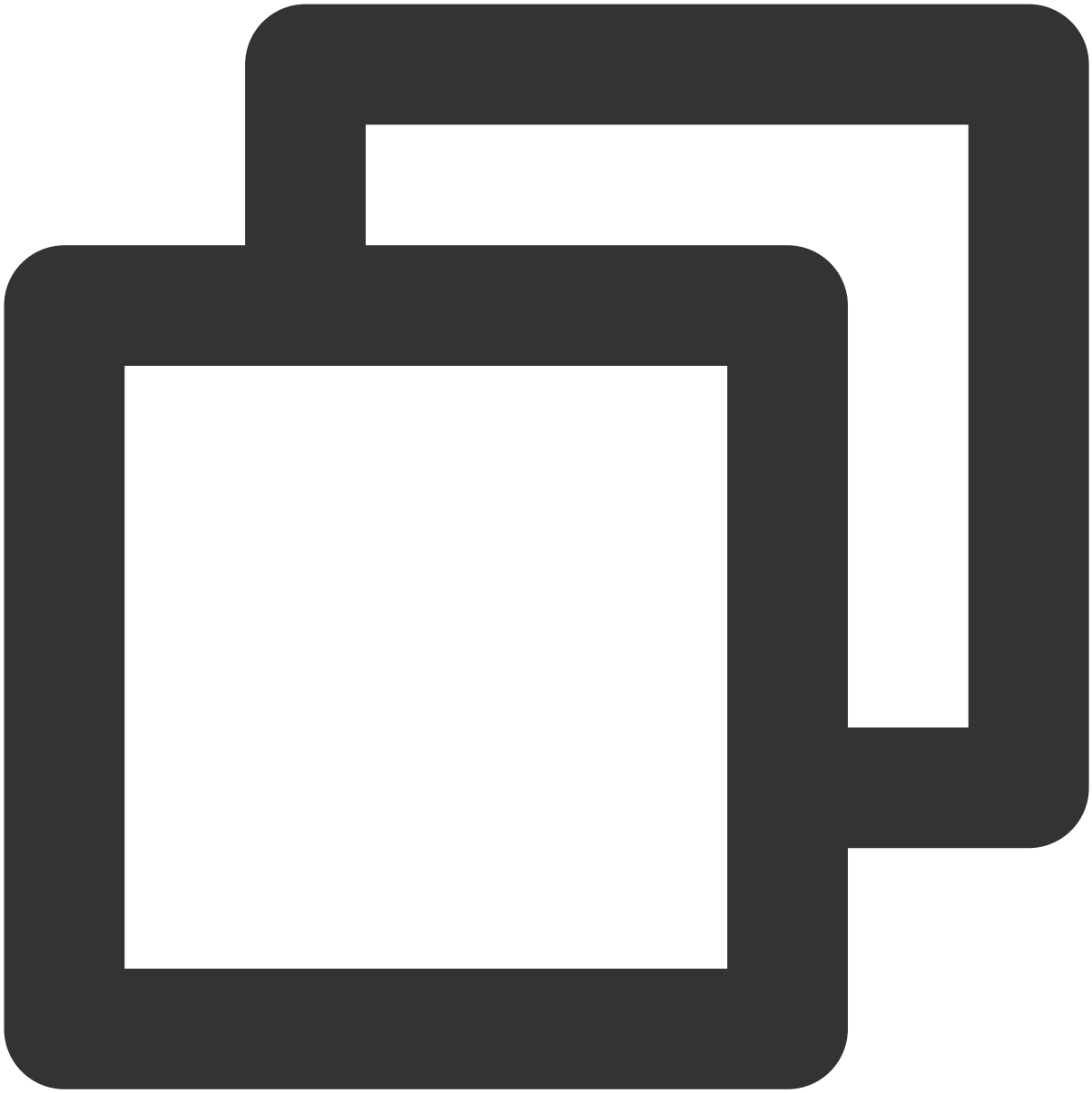
```
CREATE USER 'mysqld-exporter' IDENTIFIED BY '123456' WITH MAX_USER_CONNECTIONS 3;
GRANT PROCESS, REPLICATION CLIENT, REPLICATION SLAVE, SELECT ON *.* TO 'mysqld-expo
flush privileges;
```

3. Use the YAML file to deploy the MySQL exporter. An example is as follows:

Note

Replace the account, password, and MySQL connection address in DATA_SOURCE_NAME with real ones.

©2013-2022 Tencent Cloud. All rights reserved.



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysqld-exporter
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysqld-exporter
  template:
    metadata:
```

©2013-2022 Tencent Cloud. All rights reserved.

```
      labels:
        app: mysqld-exporter
    spec:
      containers:
      - name: mysqld-exporter
        image: prom/mysqld-exporter:v0.12.1
        args:
        - --collect.info_schema.tables
        - --collect.info_schema.innodb_tablespaces
        - --collect.info_schema.innodb_metrics
        - --collect.global_status
        - --collect.global_variables
        - --collect.slave_status
        - --collect.info_schema.processlist
        - --collect.perf_schema.tablelocks
        - --collect.perf_schema.eventsstatements
        - --collect.perf_schema.eventsstatementssum
        - --collect.perf_schema.eventswaits
        - --collect.auto_increment.columns
        - --collect.binlog_size
        - --collect.perf_schema.tableiowaits
        - --collect.perf_schema.indexiowaits
        - --collect.info_schema.userstats
        - --collect.info_schema.clientstats
        - --collect.info_schema.tablestats
        - --collect.info_schema.schemastats
        - --collect.perf_schema.file_events
        - --collect.perf_schema.file_instances
        - --collect.perf_schema.replication_group_member_stats
        - --collect.perf_schema.replication_applier_status_by_worker
        - --collect.slave_hosts
        - --collect.info_schema.innodb_cmp
        - --collect.info_schema.innodb_cmpmem
        - --collect.info_schema.query_response_time
        - --collect.engine_tokudb_status
        - --collect.engine_innodb_status
        ports:
        - containerPort: 9104
          protocol: TCP
        env:
        - name: DATA_SOURCE_NAME
          value: "mysqld-exporter:123456@(mysql.default.svc.cluster.local:3306)/"
--
apiVersion: v1
kind: Service
metadata:
  name: mysqld-exporter
```

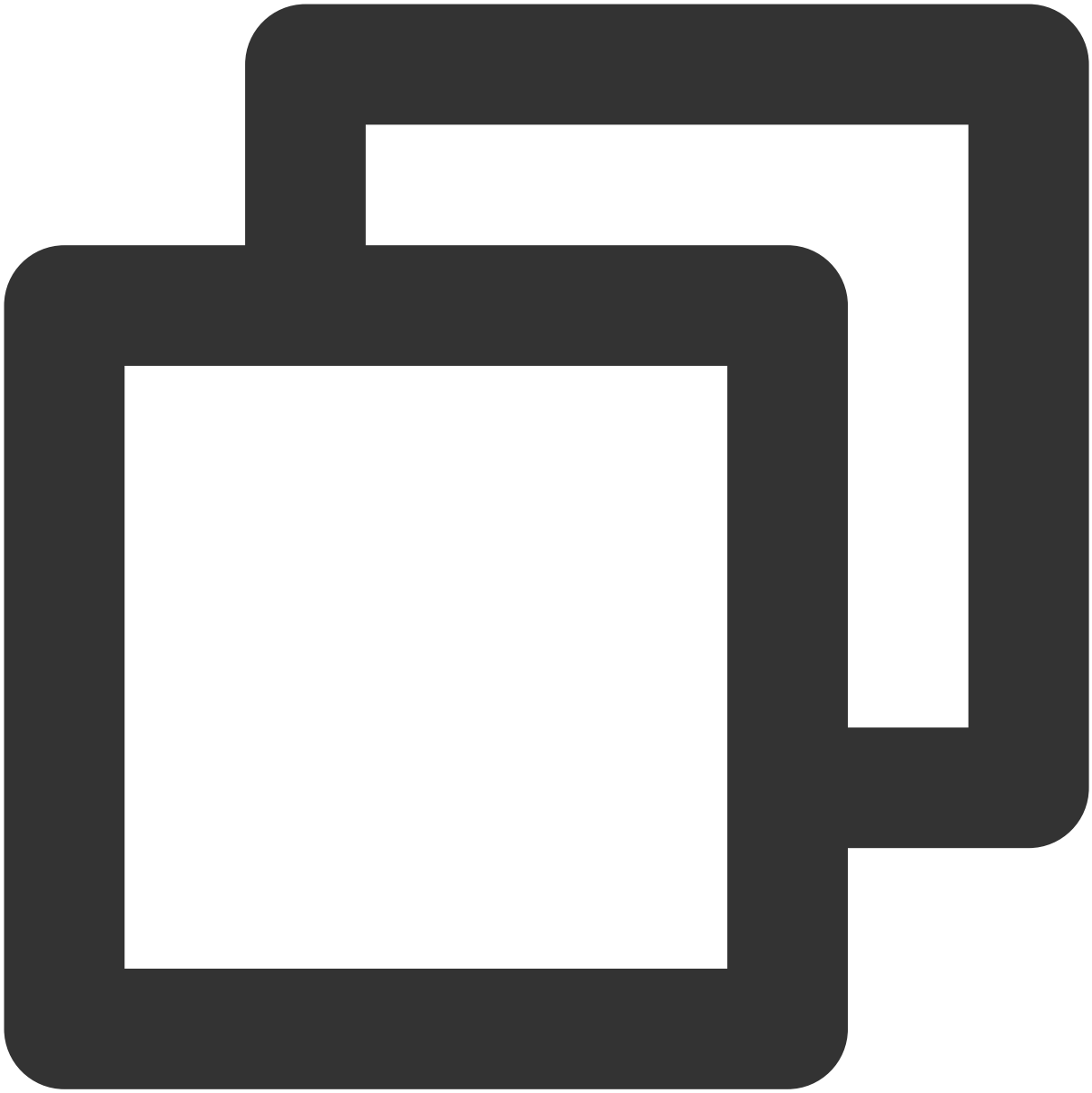
©2013-2022 Tencent Cloud. All rights reserved.

```
    labels:
      app: mysqld-exporter
  spec:
    type: ClusterIP
    ports:
    - port: 9104
      protocol: TCP
      name: http
    selector:
      app: mysqld-exporter
```

Configuring monitoring data collection

After deploying the MySQL exporter, configure monitoring data collection to ensure that data exposed by the MySQL exporter can be collected. The following example shows ServiceMonitor definition (The cluster must support ServiceMonitor definition to configure collection rules):

©2013-2022 Tencent Cloud. All rights reserved.

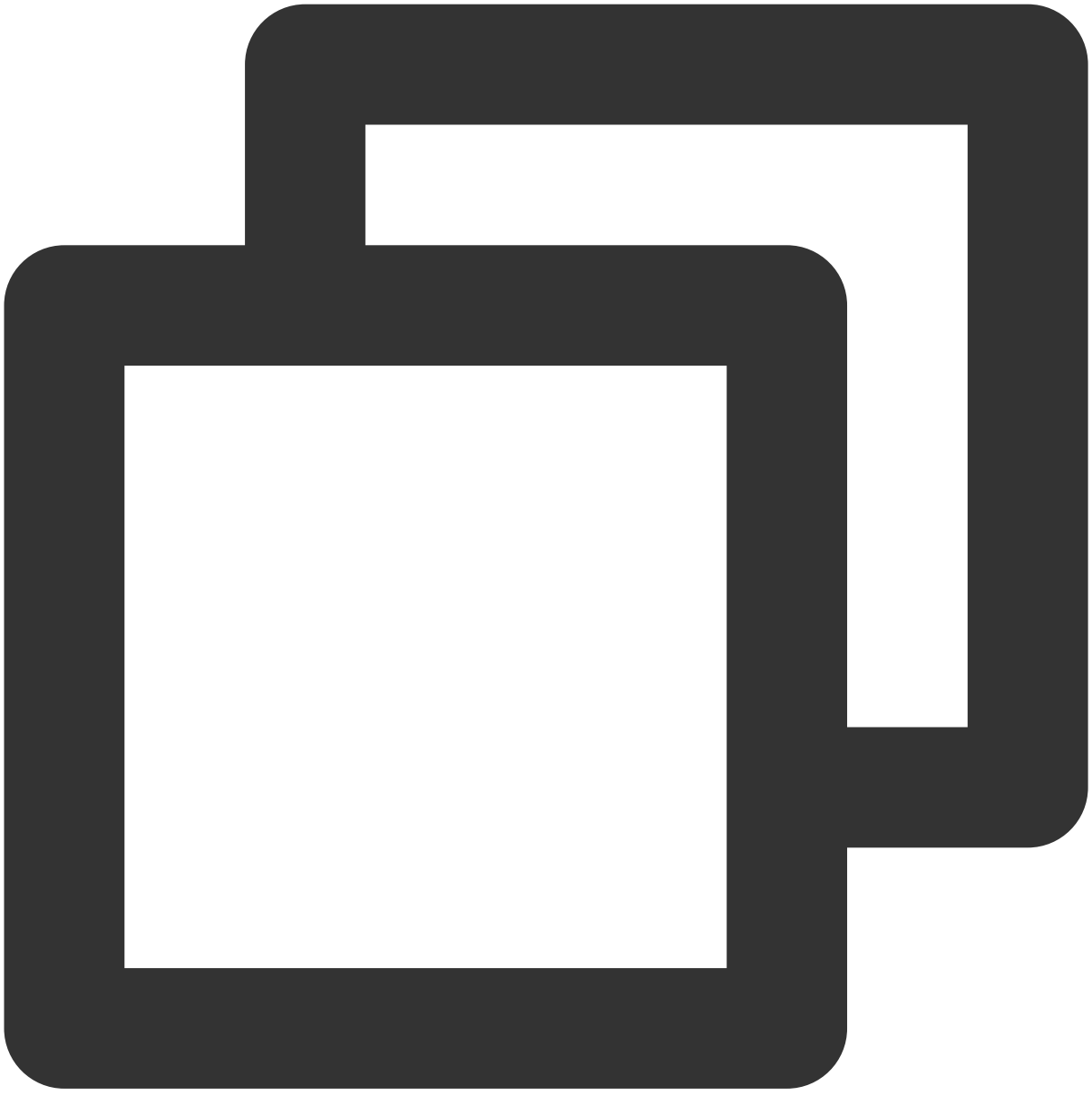


```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: mysqld-exporter
spec:
  endpoints:
    interval: 5s
    targetPort: 9104
  namespaceSelector:
    matchNames:
    - default
```

©2013-2022 Tencent Cloud. All rights reserved.

```
  selector:
    matchLabels:
      app: mysqld-exporter
```

The following example shows a native Prometheus configuration:



```
  - job_name: mysqld-exporter
    scrape_interval: 5s
    kubernetes_sd_configs:
    - role: endpoints
      namespaces:
        names:
```

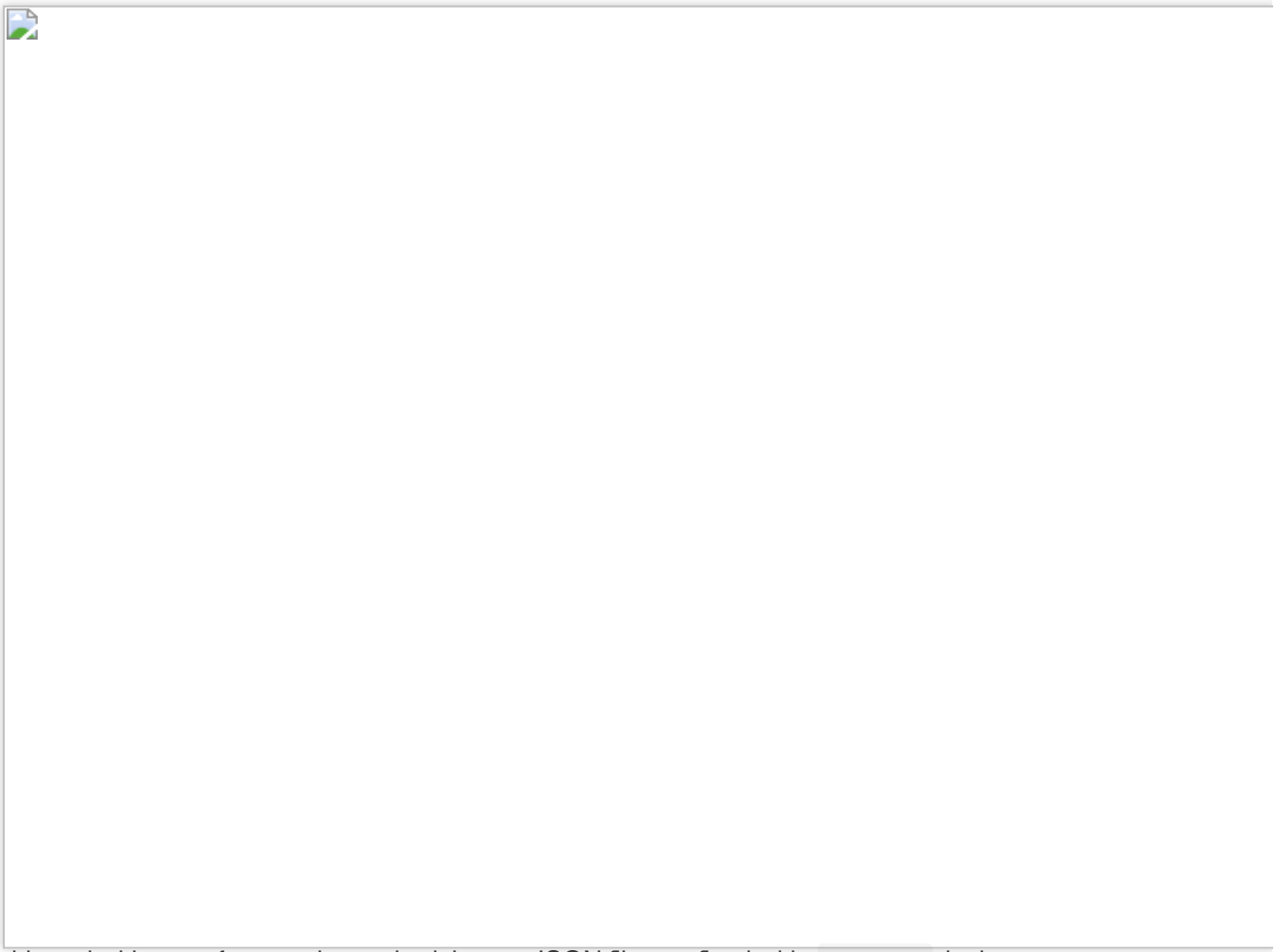
©2013-2022 Tencent Cloud. All rights reserved.

```
      - default
    relabel_configs:
    - action: keep
      source_labels:
      - __meta_kubernetes_service_label_app_kubernetes_io_name
      regex: mysqld-exporter
    - action: keep
      source_labels:
      - __meta_kubernetes_endpoint_port_name
      regex: http
```

Adding a monitoring dashboard

Once data can be collected, add a monitoring dashboard for Grafana to display data.

If you only need to view the MySQL or MariaDB overview information, import the grafana.com dashboard, as shown in the figure below.



If a dashboard with more features is required, import JSON files prefixed with `MySQL_` in the percona open-source dashboard.

©2013-2022 Tencent Cloud. All rights reserved.

Migrating Self-built Prometheus to Cloud Native Monitoring

Last updated：2022-04-18 16:58:49

Overview

Compatible with the APIs of Prometheus and Grafana and the CRD usage of mainstream prometheus-operator, TKE Cloud Native Monitoring is more flexible and extensible. Combined with Prometheus open source tools, it can have more advanced usages.
This document describes how to use auxiliary scripts and migration tools to quickly migrate the self-built Prometheus to cloud native monitoring.

Prerequisites

You have installed Kubectl on the node of the self-built Prometheus cluster and configured Kubeconfig to ensure that you can manage the cluster through Kubectl.

Directions

Migrating the Dynamic Collection Configuration

If the prometheus-operator is used in self-built Prometheus, CRD resources such as ServiceMonitor and PodMonitor are usually used to dynamically add collection configurations. This method also applies to cloud native monitoring. If you only need to migrate the prometheus-operator of the self-built Prometheus cluster to cloud native monitoring, and without migrating the cluster, then there is no need to migrate the dynamic configuration. You only need to use the cloud native monitoring to associate the self-built cluster, and then the ServiceMonitor and PodMonitor resources created by the self-built Prometheus will automatically take effect in cloud native monitoring.

For cross-cluster migration, you can export the CRD resources of self-built Prometheus and selectively reapply them in the associated cloud native monitoring cluster. The following describes how to export ServiceMonitor and PodMonitor in batches in a self-built Prometheus cluster.

1. Create the script `prom-backup.sh` with the following contents:

©2013-2022 Tencent Cloud. All rights reserved.

```
_ns_list=$(kubectl get ns | awk '{print $1}' | grep -v NAME)
count=0
declare -a types=("servicemonitors.monitoring.coreos.com" "podmonitors.monitori
ng.coreos.com")
for _ns in ${_ns_list}; do
## loop for types
for _type in "${types[@]}"; do
echo "Backup type [namespace: ${_ns}, type: ${_type}]."
_item_list=$(kubectl -n ${_ns} get ${_type} | grep -v NAME | awk '{print $1}' )
## loop for items
for _item in ${_item_list}; do
_file_name=./${_ns}_${_type}_${_item}.yaml
echo "Backup kubernetes config yaml [namespace: ${_ns}, type: ${_type}, item:
${_item}] to file: ${_file_name}"
kubectl -n ${_ns} get ${_type} ${_item} -o yaml > ${_file_name}
count=$[count + 1]
echo "Backup No.${count} file done."
done;
done;
done;
```

2. Run the following command to run the `prom-backup.sh` script.

```
bash prom-backup.sh
```

3. The `prom-backup.sh` script will export each ServiceMonitor and PodMonitor resource into a separate YAML file. You can run the `ls` command to view the output file list. The example is as follows:

```
$ ls
kube-system_servicemonitors.monitoring.coreos.com_kube-state-metrics.yaml
kube-system_servicemonitors.monitoring.coreos.com_node-exporter.yaml
monitoring_servicemonitors.monitoring.coreos.com_coredns.yaml
monitoring_servicemonitors.monitoring.coreos.com_grafana.yaml
monitoring_servicemonitors.monitoring.coreos.com_kube-apiserver.yaml
monitoring_servicemonitors.monitoring.coreos.com_kube-controller-manager.yaml
monitoring_servicemonitors.monitoring.coreos.com_kube-scheduler.yaml
monitoring_servicemonitors.monitoring.coreos.com_kube-state-metrics.yaml
monitoring_servicemonitors.monitoring.coreos.com_kubelet.yaml
monitoring_servicemonitors.monitoring.coreos.com_node-exporter.yaml
```

©2013-2022 Tencent Cloud. All rights reserved.

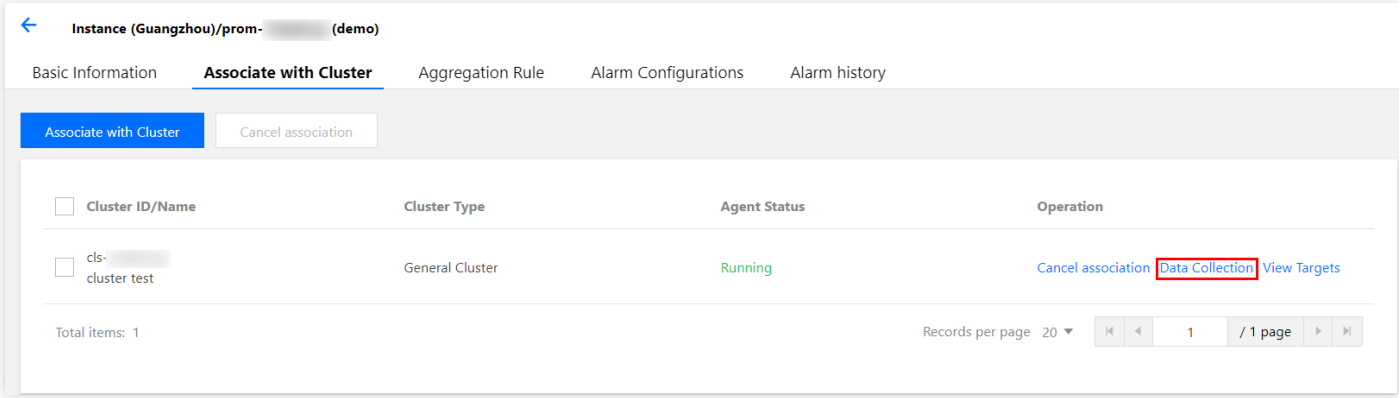
4. You can filter, modify and reapply the YAML file to the associated cloud native monitoring cluster (do not apply the collection rules that already exist or have the same feature). The cloud native monitoring will automatically perceive these dynamic collection rules and perform collection.

> Note：
> If you need to add ServiceMonitor or PodMonitor, you can add it visually on the TKE console, or you can directly create it with YAML. The usage is fully compatible with the CRD of the Prometheus community.

Migrating the static collection configuration

If the self-built Prometheus system directly uses the Prometheus native configuration file, you can convert it into a RawJob of cloud native monitoring with a few steps on the TKE console, making it compatible with the `scrape_configs` configuration item of the Prometheus native configuration file.

1. Log in to the TKE console.
2. Click **Cloud Native Monitoring** in the left sidebar to go to the **Cloud Native Monitoring** page.
3. Click the instance ID/name to configure to go to its basic information page.
4. Select **Associate with Cluster** tab, select the cluster to configure, and click **Data Collection** under the **Operation** column.



5. Select **RawJob** > **Add**. Copy and paste the Job configuration from the native Prometheus configuration file into this configuration window.

©2013-2022 Tencent Cloud. All rights reserved.

6. You can paste all the Job arrays that need to import into the cloud native monitoring, and click **Confirm**. The Job arrays will be automatically split into multiple RawJobs and named as the `job_name` field of each Job.

Migrating the global configuration

You can modify the Prometheus CRD resource of cloud native monitoring to modify the global configuration.

1. Run the following command to obtain the Prometheus information.

```
$ kubectl get ns
prom-fnc7bvu9 Active 13m
$ kubectl -n prom-fnc7bvu9 get prometheus
NAME VERSION REPLICAS AGE
tke-cls-hha93bp9 11m
$ kubectl -n prom-fnc7bvu9 edit prometheus tke-cls-hha93bp9
```

2. Run the following command to modify the Prometheus configuration.

```
$ kubectl -n prom-fnc7bvu9 edit prometheus tke-cls-hha93bp9
```

Modify the following parameters in the pop-up window:

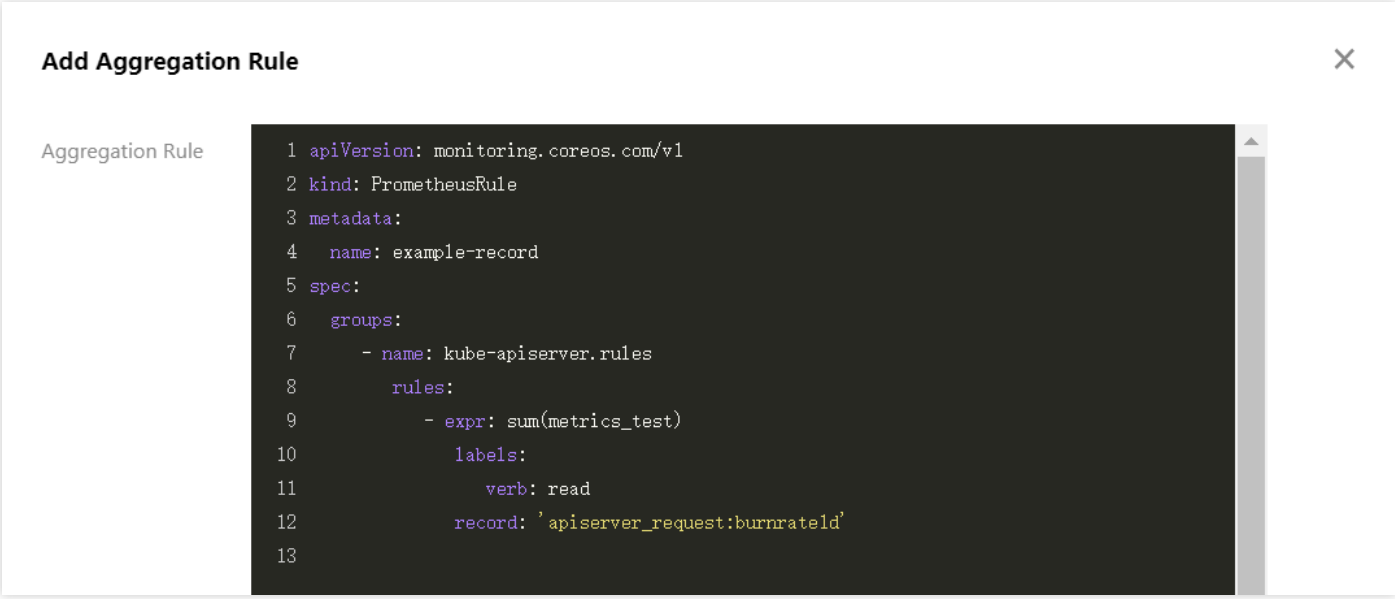
- **scrapeInterval**: the collection capture interval (default value is 15 seconds)
- **externalLabels**: add the default label tag for all time series data.

Migrating the aggregation configuration

©2013-2022 Tencent Cloud. All rights reserved.

The format of each Prometheus aggregation configuration rule is the same no matter it is the original static configuration Recording rules or the dynamic configuration PrometheusRule.

1. Log in to the TKE console.
2. Click **Cloud Native Monitoring** in the left sidebar to go to the **Cloud Native Monitoring** page.
3. Click the instance ID/name to configure to go to its basic information page.
4. Select **Aggregation Rule** > **Create Aggregation Rule**. In the **Add Aggregation Rule** window, paste each rule into the groups array in the PrometheusRule format, as shown in the figure below:



> Note：
> If the self-built Prometheus uses the aggregation rules defined by PrometheusRule, it is recommended to migrate them according to the above steps. If the PrometheusRule resource is created directly in the cluster using YAML, it cannot be displayed in cloud native monitoring on the console currently.

Migrating the alarm configuration

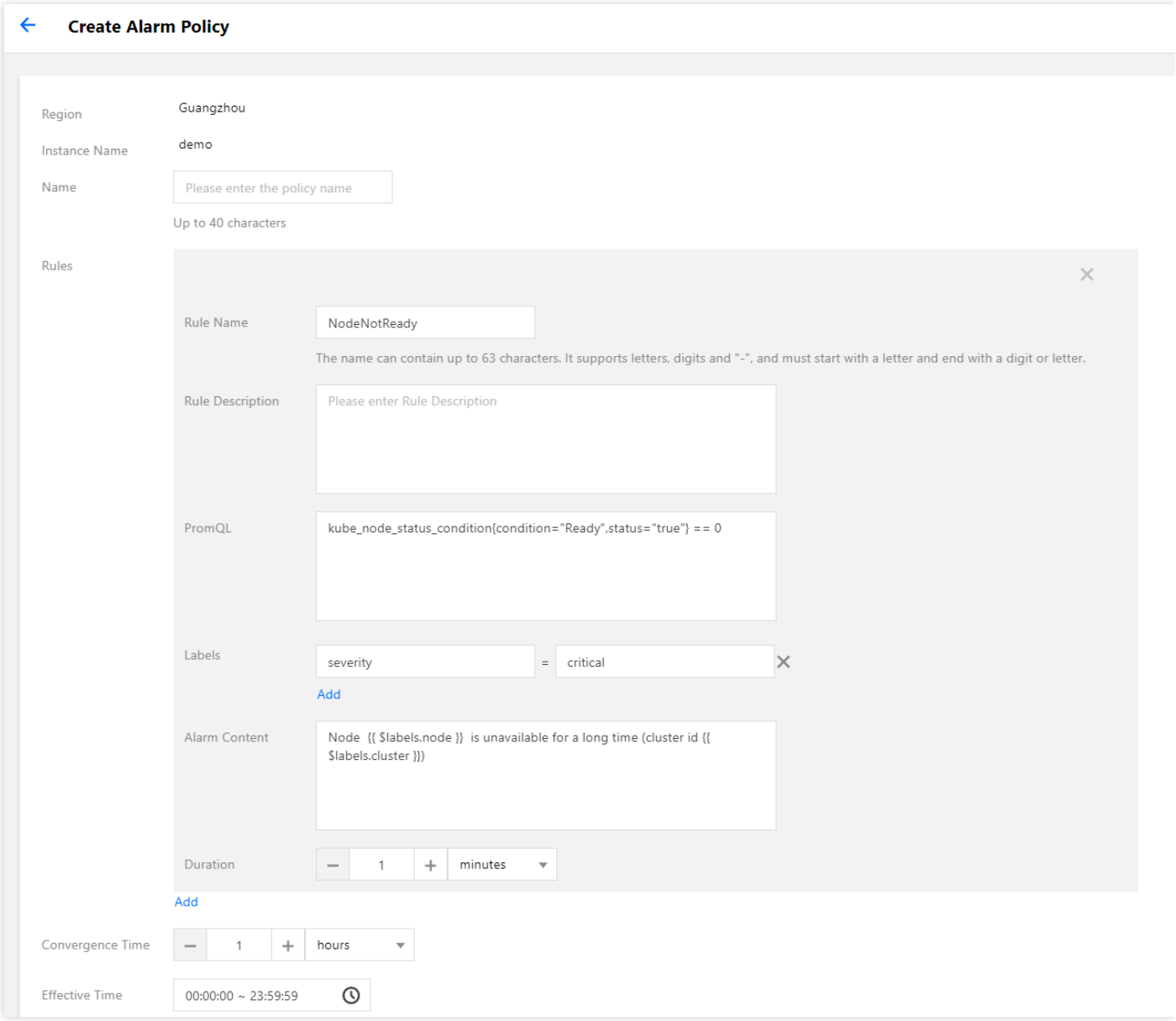
This document provides the self-built Prometheus Alarm original configuration YAML file as an example to describe how to convert it into a monitoring configuration similar to cloud native monitoring.

```
- alert: NodeNotReady
expr: kube_node_status_condition{condition="Ready",status="true"} == 0
for: 5m
labels:
severity: critical
annotations:
```

©2013-2022 Tencent Cloud. All rights reserved.

```
description: node {{ $labels.node }} is not available for a long time (cluster id
{{ $labels.cluster }})
```

1. Log in to the TKE console.
2. Click **Cloud Native Monitoring** in the left sidebar to go to the **Cloud Native Monitoring** page.
3. Click the instance ID/name to configure to go to its basic information page.
4. Select **Alarm Configurations** > **Create Alarm Policy** to configure the alarm policy.



Main parameters are described as follows:

- **PromQL**: the core configuration of the alarm and is the PromQL expression used to indicate the alarm trigger condition, which is equivalent to the “expr” field of the original configuration.
- **Labels**: an extra label added for the alarm, which is equivalent to the labels field of the original configuration.
- **Alarm Content**: the pushed alarm content. You can use a template or a template with variables. It is recommended to add the cluster ID in the alarm content. You can use the variable `{{ $labels.cluster`

©2013-2022 Tencent Cloud. All rights reserved.

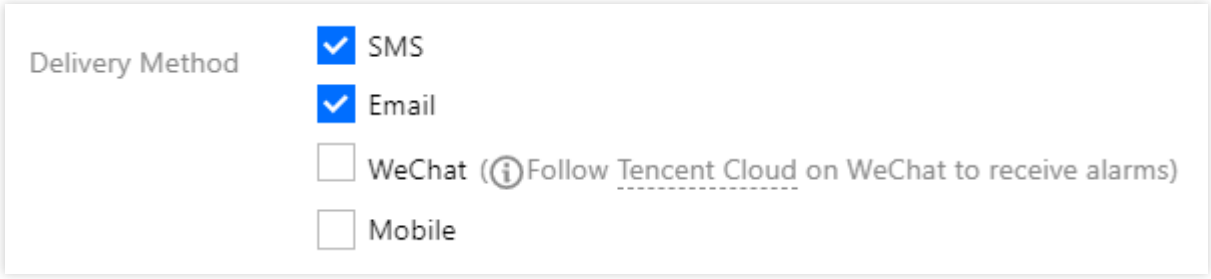
`}}` to represent the cluster ID.

- **Duration**: indicates an alarm will be pushed when the alarm is not restored after the alarm condition is met for how long. It is equivalent to the “for” field of the original configuration. The configuration in the following sample is 5 minutes.
- **Convergence Time**: indicates an alarm will be pushed again when the alarm is not restored after the alarm condition is met for how long, that is, the push interval between the same alarms. It is equivalent to the repeat_interval configuration of AlertManager. The configuration in the following sample is 1 hour.

> Note：
> The above alarm configuration example shows that after the node status changes to NotReady, the alarm will be pushed if it is not restored within 5 minutes. If it has not restored for a long time, the alarm will be pushed again at an interval of 1 hour.

5. Configure the alarm channel. Currently, only Tencent Cloud and WebHook are available.
   - Tencent Cloud alarm channel
   - WebHook alarm channel

   The alarm channels of Tencent Cloud support SMS, Email, WeChat and Mobile. You can select as needed.



Migrating the Grafana dashboard

The self-built Prometheus is usually configured with many custom Grafana monitoring dashboards. If you need to migrate a large number of dashboards to other platforms, it is too inefficient to export and import one by one. You can use the grafana-backup tool to export and import Grafana dashboards in batches. For details, please refer to the following directions.

1. Run the following command to install grafana-backup, as shown below:

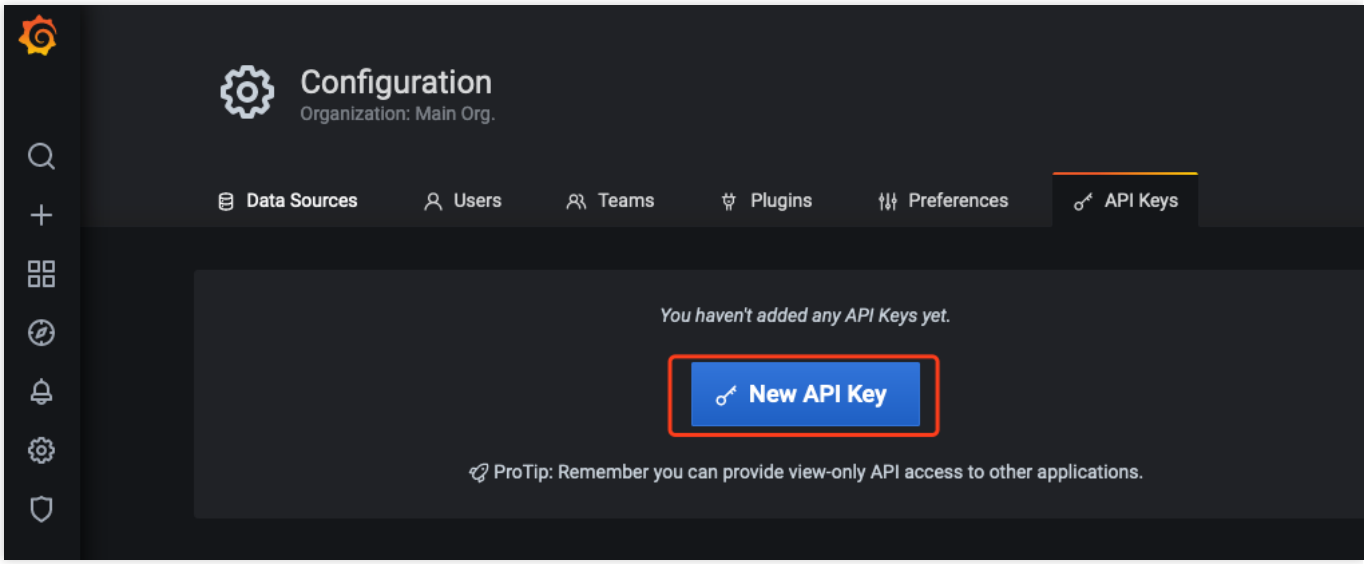
```
pip3 install grafana-backup
```

©2013-2022 Tencent Cloud. All rights reserved.

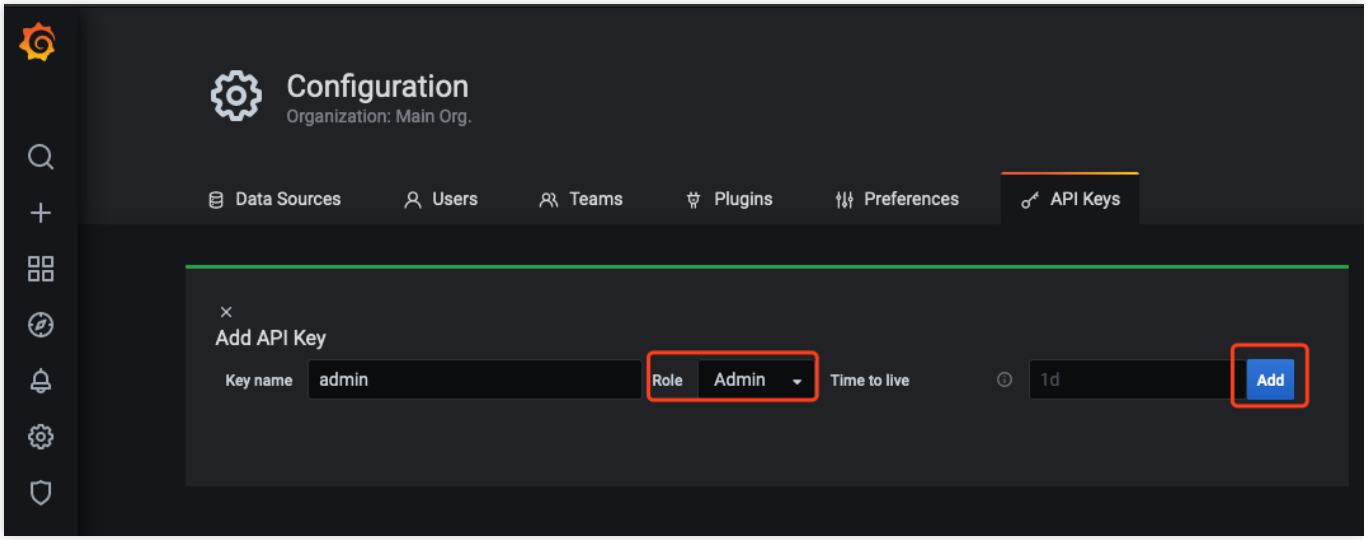
Note：
It is recommended to use Python3 to avoid the compatibility problems.

2. Create API Keys.

i. Enter the configuration page of self-buit Grafana and cloud native monitoring Grafana respectively. Select **API Keys** > **New API Key**, as shown below:



ii. In **Add API Key** window, create an API KEY whose role is Admin, as shown below:



3. Back up the configuration file of the dashboard that you want to export.

©2013-2022 Tencent Cloud. All rights reserved.

i. Run the following command to obtain the access address of the self-built Grafana, as shown below:

```
$ kubectl -n monitoring get svc
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
grafana ClusterIP 172.21.254.127 <none> 3000/TCP 25h
```

Note：
Take the Grafana access address `http://172.21.254.127:3000` in the cluster as an example.

ii. Run the following command to generate the grafana-backup configuration file (with Grafana address and APIKey) as shown below:

```
export TOKEN=<TOKEN>
cat > ~/.grafana-backup.json <<EOF
{
"general": {
"debug": true,
"backup_dir": "_OUTPUT_"
},
"grafana": {
"url": "http://172.21.254.127:3000",
"token": "${TOKEN}"
}
}
EOF
```

Note：
You need to replace <TOKEN> with the APIKey of self-built Grafana, and replace the URL with the actual environment address.

4. Run the following command to export all dashboards, as shown below:

```
grafana-backup save
```

The dashboard will be saved as a compressed file in the `_OUTPUT_` directory. You can run the following command to view the files in this directory, as shown below:

©2013-2022 Tencent Cloud. All rights reserved.

```
$ tree _OUTPUT_
_OUTPUT_
└── 202012151049.tar.gz
0 directories, 1 file
```

5. Run the following command to restore the configuration file, as shown below:

```
export TOKEN=<TOKEN>
cat > ~/.grafana-backup.json <<EOF
{
"general": {
"debug": true,
"backup_dir": "_OUTPUT_"
},
"grafana": {
"url": "http://prom-xxxxxx-grafana.ccs.tencent-cloud.com",
"token": "${TOKEN}"
}
}
EOF
```

> Note：
> You need to replace <TOKEN> with the APIKey of cloud native monitoring Grafana, and replace the URL with the access address of cloud native monitoring Grafana. (The internet access need to be enabled).

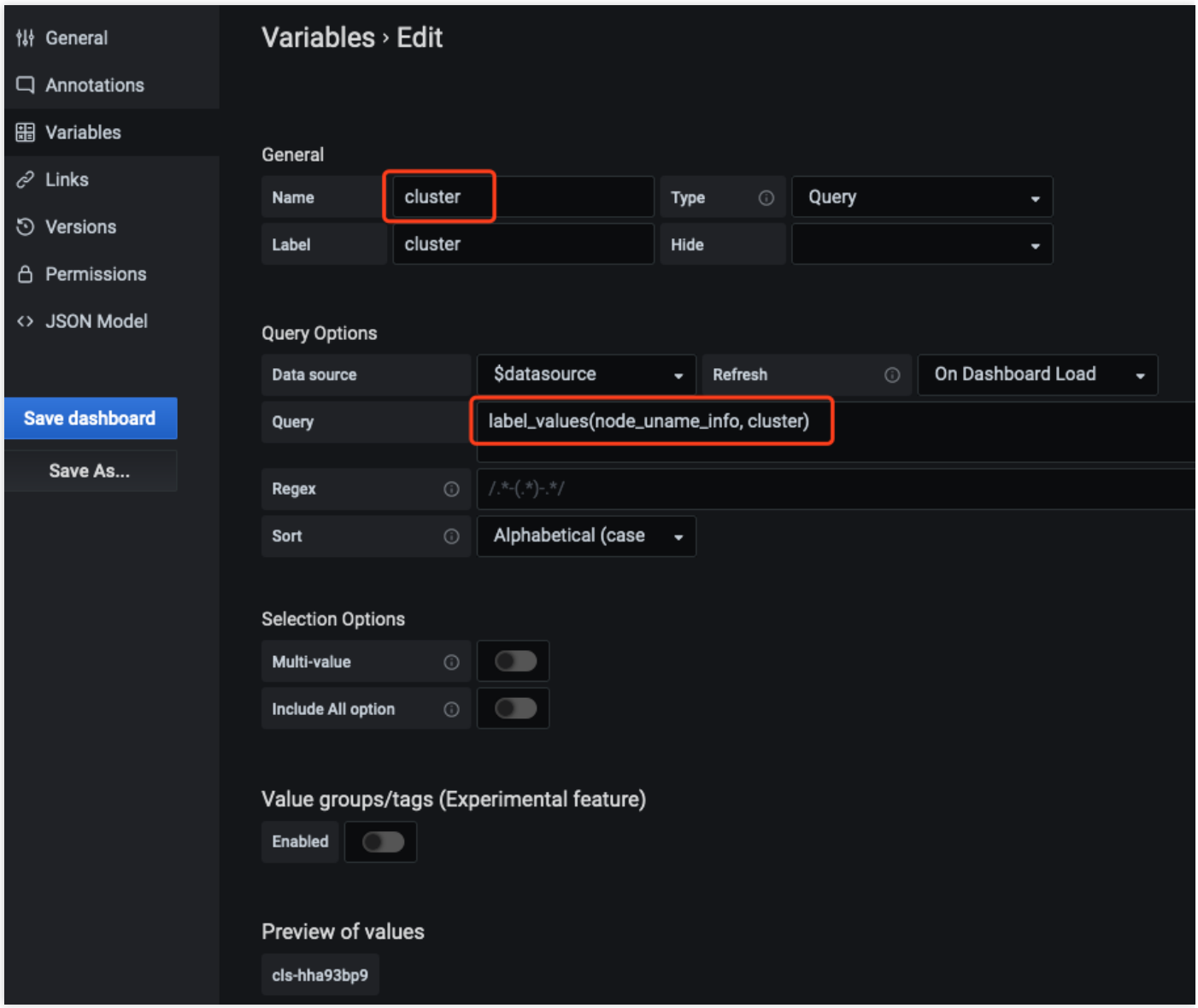
6. Run the following command to import the exported dashboards to the cloud native monitoring Grafana with one click, as shown below:

```
grafana-backup restore _OUTPUT_/202012151049.tar.gz
```

7. In Grafana configuration dashboard, select **Dashboard settings** > **Variables** > **New** to create the cluster field. It is recommended to add the filter field “cluster” for all dashboards. Cloud native monitoring supports multiple clusters. It will add the label “cluster” to the data of each cluster, and use the cluster ID to distinguish different clusters, as

©2013-2022 Tencent Cloud. All rights reserved.

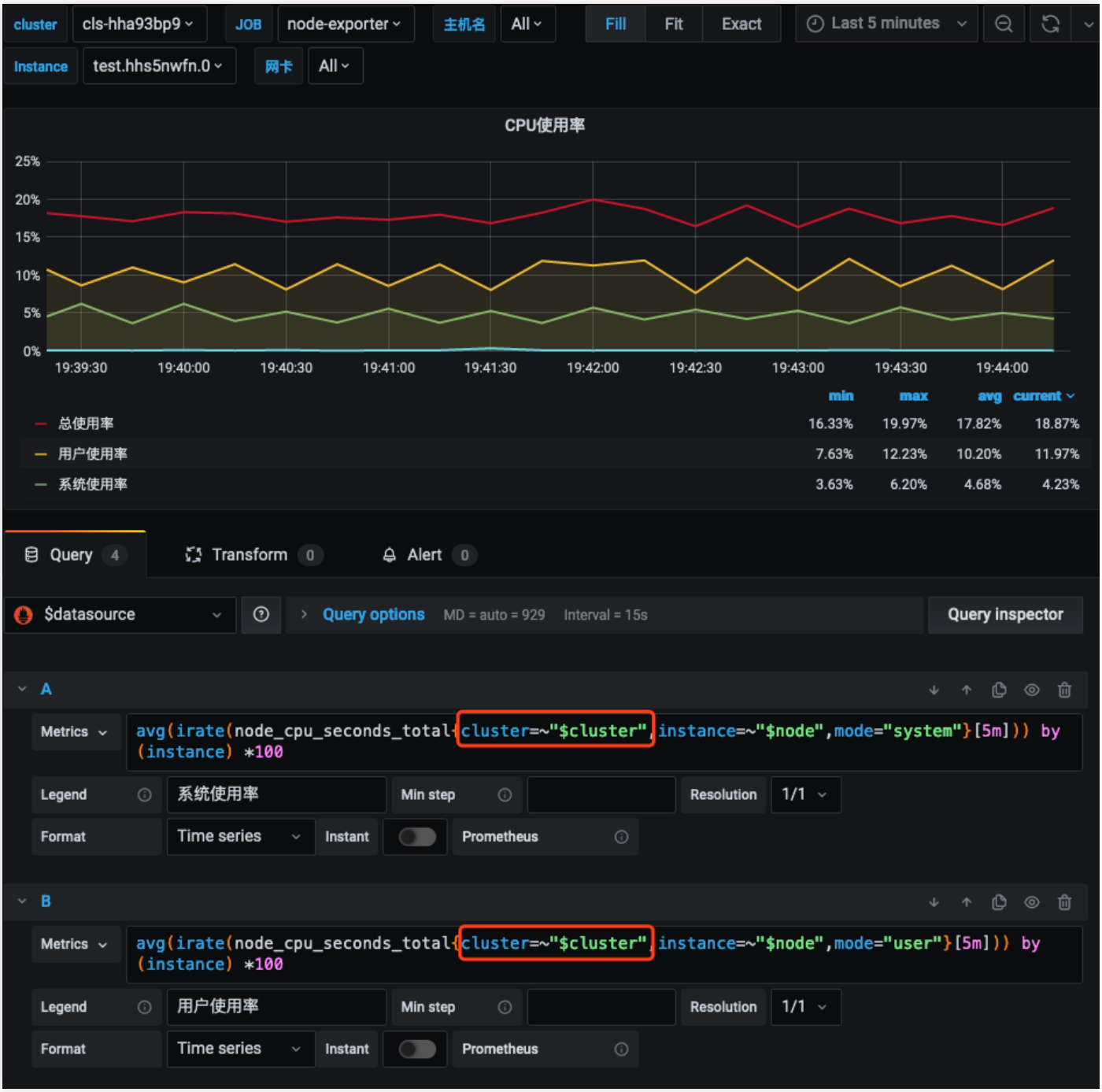
shown below:



Note：
Enter an arbitrary metric name that is involved in the current dashboard in label_values (The example is node_uname_info).

8. Modify the query statements of PromQL in all dashboards and add the filter conditions `cluster=~"$cluster"` , as shown below:

©2013-2022 Tencent Cloud. All rights reserved.



Integrating with the existing systems

Cloud native monitoring supports accessing self-built Grafana and AlertManager systems.

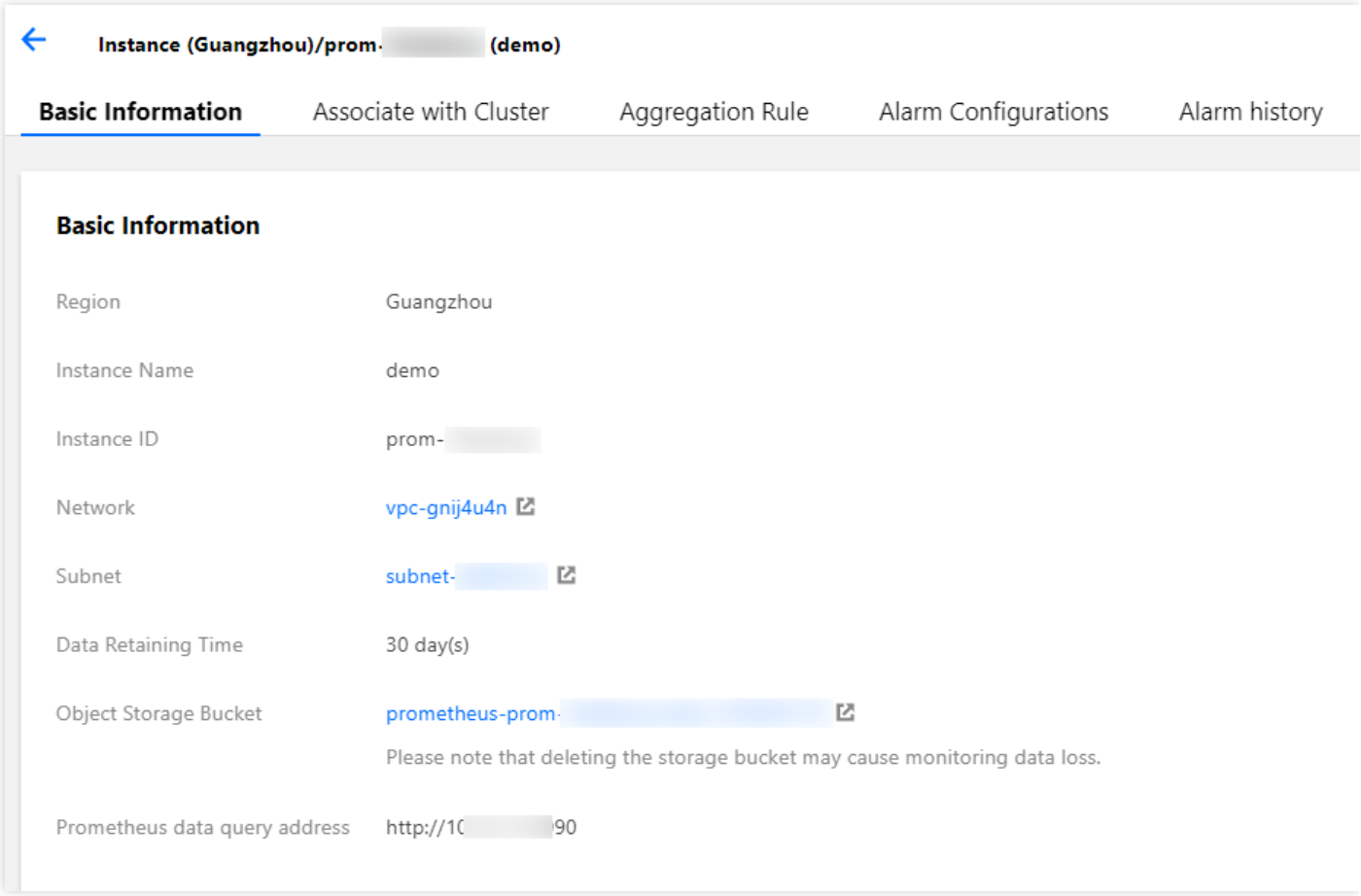
- Accessing self-built Grafana
- Accessing self-built AlertManager

Cloud native monitoring provides Prometheus API. If you need to use self-built Grafana to display monitoring, you can add cloud native monitoring data as a Prometheus data source to self-built Grafana. You can find the Prometheus API

©2013-2022 Tencent Cloud. All rights reserved.

address in the basic information of cloud native monitoring instance on TKE console.

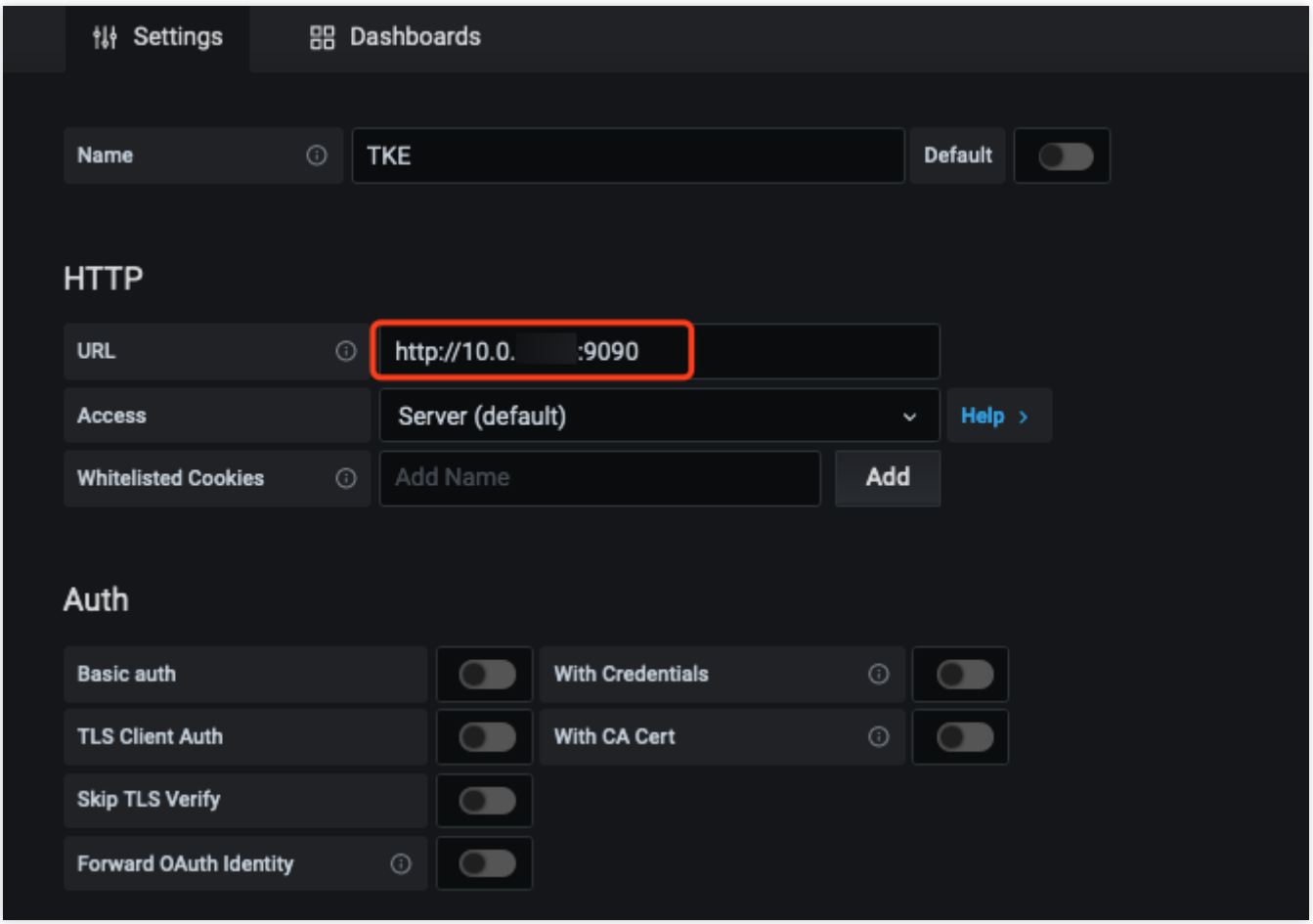
1. Log in to the TKE console.
2. Click **Cloud Native Monitoring** in the left sidebar to go to the **Cloud Native Monitoring** page.
3. Click the instance ID/name to go to its details page to obtain the Prometheus API address.



>?Ensure that the self-built Grafana and cloud native monitoring are in the same VPC or their networks have connected.

©2013-2022 Tencent Cloud. All rights reserved.

4. Add the Prometheus API address in Grafana as the Prometheus data source, as shown below:



©2013-2022 Tencent Cloud. All rights reserved.

OPS
Removing and Re-adding Nodes from and to Cluster

Last updated：2023-02-02 17:05:22

Overview

In many TKE scenarios, such as Kubernetes version upgrade and kernel version upgrade, you must remove the node and then add it back. This document describes the process of removing and re-adding a node in detail. This operation can be divided into the following steps:

1. Drain the Pods running on the node.
2. Remove the node from the cluster, and then re-add it to the cluster. This node will reinstall the system.
3. Remove the cordon.

Notes

- If multiple nodes on a single cluster all need to be removed and added back, it is recommended that you do so node by node. That is, complete the removal and addition of a single node and verify that the service is normal, and then remove the next node and add it back, completing multiple nodes successively.
- If you need to do this for multiple clusters under the same account, we recommend that the operation be executed in batches. After the operation has been completed on each cluster, verify whether the cluster status is normal.

Directions

Step 1: drain Pods

Before performing the removal and re-addition of a node in a cluster, you must first drain the Pods on the node to be removed to have them operate on a different node. The draining process involves deleting the Pods on the node one by one, and then reconstruct them on another node.

Principles of draining

To streamline node maintenance operations, Kubernetes introduced the `drain` command. The use principles are as follows:

©2013-2022 Tencent Cloud. All rights reserved.

For versions after Kubernetes 1.4, the `drain` operation is to first cordon the node and then delete all the Pods on the node. If this Pod is managed by a controller such as Deployment, the controller will re-construct the Pod when it detects that the number of Pod replicas has decreased, and will schedule them to other nodes that meet the conditions. If this Pod is a bare Pod that is not managed by a controller, it will not be re-constructed after it is drained.

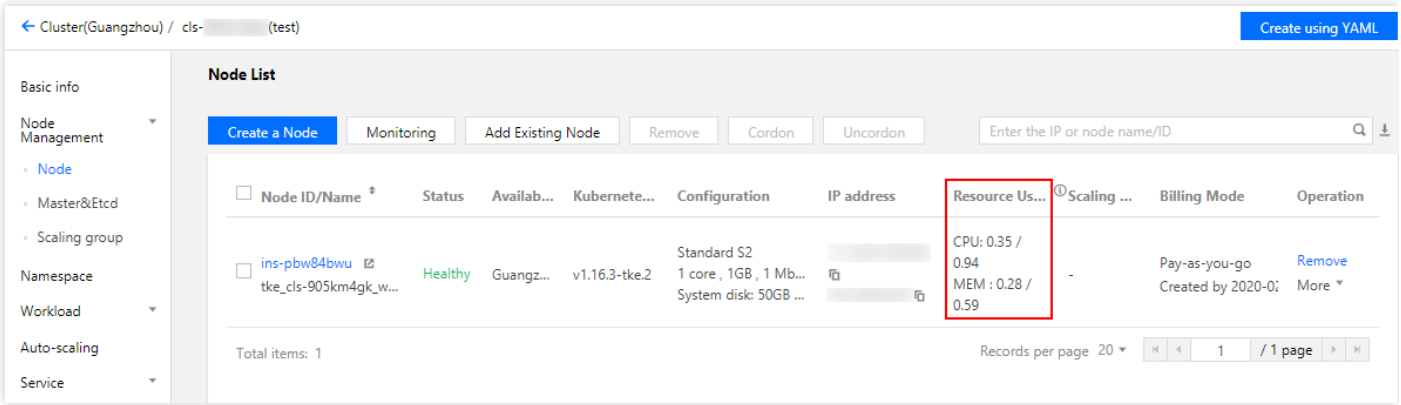
This process involves first deleting, and then re-creation, and is not a rolling update. Therefore, in the update process, some requests for drained services may fail. If all the related Pods of the drained service are on the drained node, the service may become completely unavailable.

To avoid this situation, Kubernetes versions 1.4 and later introduced PodDisruptionBudget (PDB). You only need to select a business (a group of Pods) in the PDB policy file, to declare the minimum number of replicas that this business can tolerate. Now when you execute the `drain` operation, the Pod is no longer deleted directly, but instead whether it meets the PDB policy is checked through `evict api` . The Pods will only be deleted if the PDB policy is satisfied, protecting business availability. Note that the impact of the `drain` operation on businesses can only be controlled if PDB is correctly configured.

Checking before draining

The draining process involves reconstructing Pods, which may affect services in the cluster. Therefore, it is recommended that you perform the following checks before performing draining:

1. Check whether the remaining nodes in the cluster have sufficient resources to run the Pods on the node to be drained.
   You can check the resource allocation of the nodes in the TKE console. On the **Cluster List** page, select the target cluster ID > **Node Management** > **Node**, and check the **Assigned/Total Resources** on the **Node List** page.



   If the remaining resources of the nodes are insufficient, it is recommended that you add new nodes to the cluster to prevent the drained Pod from being unable to run and, as a result, affecting the service.
2. Check whether active drainage protection, PodDisruptionBudget (PDB), is configured in the cluster.
   Active drainage protection interrupts the execution of drainage operations. It is recommended that you first delete the active drainage protection PDB.

©2013-2022 Tencent Cloud. All rights reserved.

3. Check whether all the Pods of a single service are on the node to be drained.
   If all the Pods of a single service are located on the same node, draining the Pods will make the entire service unavailable. Please determine whether the service needs all Pods to be located on the same node.
   - If not, it is recommended that you add anti-affinity scheduling.
   - If so, it is recommended that you perform the operation during periods of low or no traffic.
4. Check if the service uses a local disk (hostpath).
   If the service uses the `hostpath volume` method, when the Pod is scheduled to another node, the data will be lost which may affect the business. If the data is important, back it up before draining.

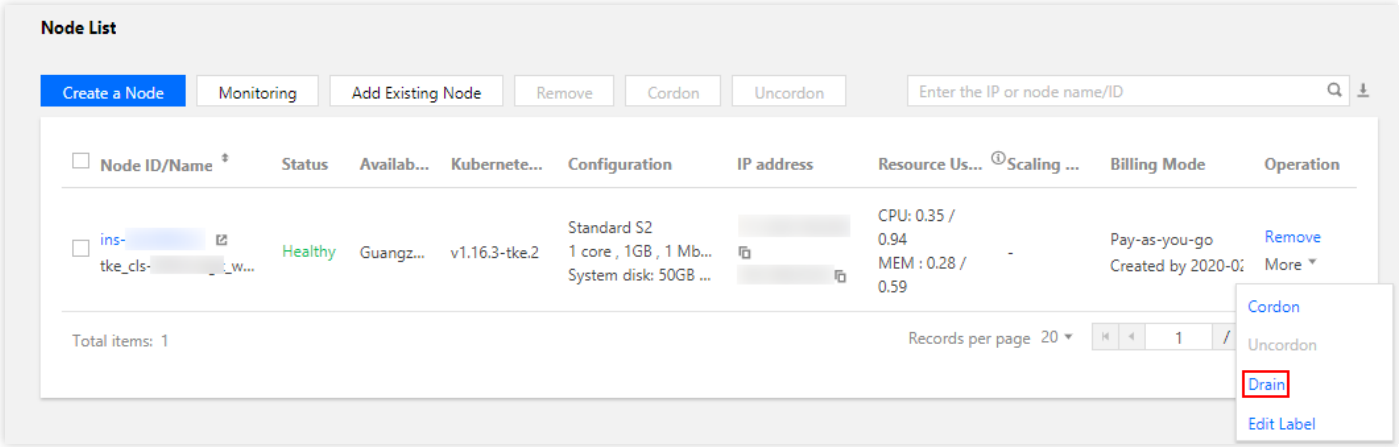
> Note：
> Currently, kubelet's image pull policy is serial. If a large number of Pods is scheduled to the same node in a short period of time, the Pod launch time may be longer.

Details

Currently, there are two ways to complete drainage for TKE clusters:

- Drain in TKE Console
- Use the kubectl drain Command to Drain

1. On the cluster list page, click the target cluster ID.
2. On the cluster details page, select **Node Management** > **Node**.
3. On the **Node** page, click **Drain** in the **Operation** column of the target node.

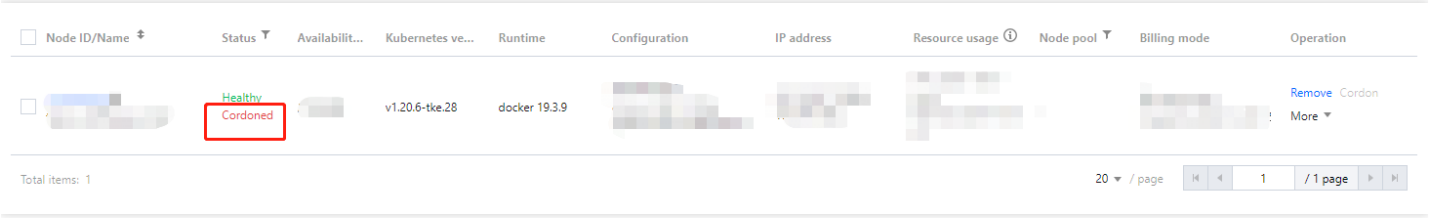


4. In the pop-up window, confirm the node information and click **OK**.

Step 2: remove the node

©2013-2022 Tencent Cloud. All rights reserved.

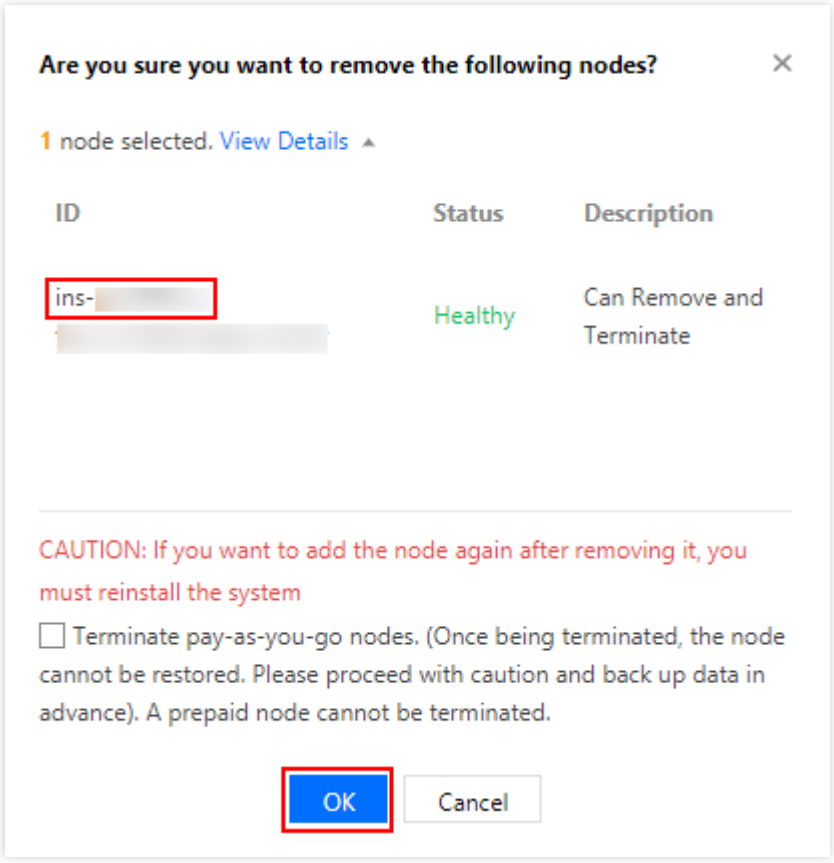
When the Pods running on a node are drained, this node is cordoned.



1. On the **Node List** page, click **Remove** in the **Operation** column of the target node.
2. In the pop-up window, deselect **Terminate pay-as-you-go nodes** and click **OK** to remove the node from the cluster.

Note：

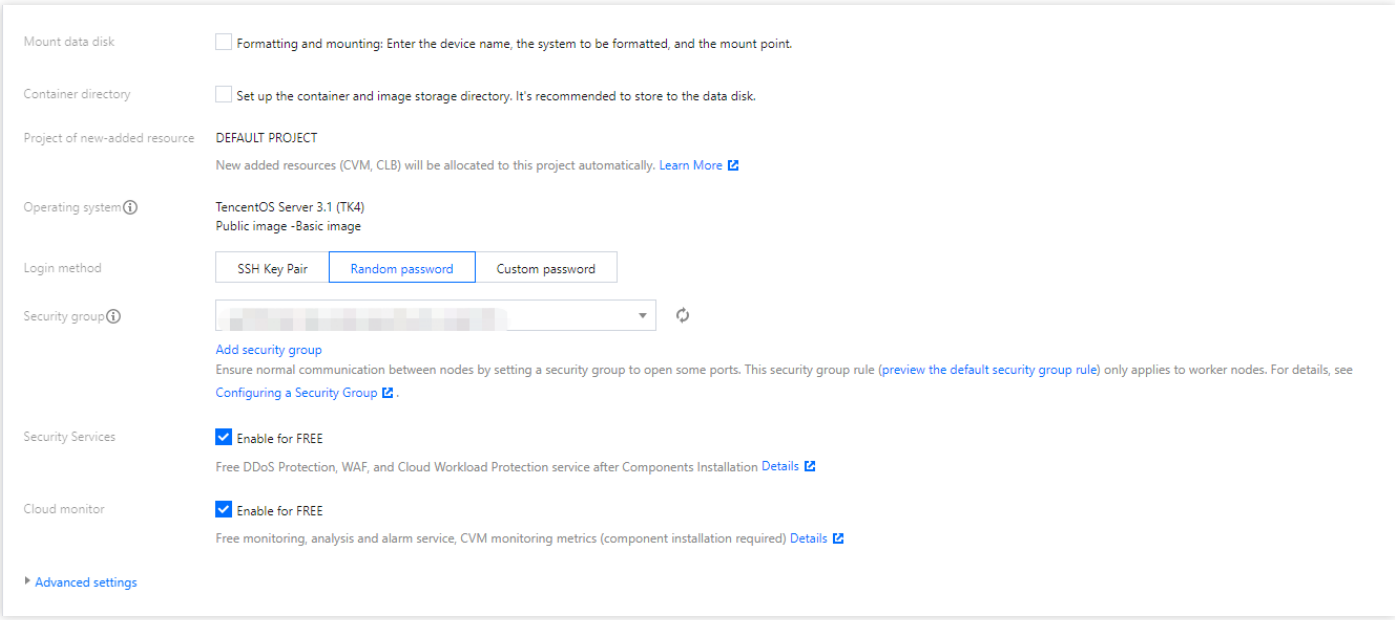
- Note the node ID to be used for re-adding the node to the cluster.
- If the node is pay-as-you-go, make sure not to select **Terminate pay-as-you-go nodes**. Terminated nodes cannot be restored.



©2013-2022 Tencent Cloud. All rights reserved.

Step 3: add the node back to the cluster

1. On the **Node List** page, click **Add Existing Node** on the top of the page.
2. On the **Add Existing Node** page, enter the recorded node ID and click .
3. In the search result list, select the target node and set CVM and other parameters as needed.



Note

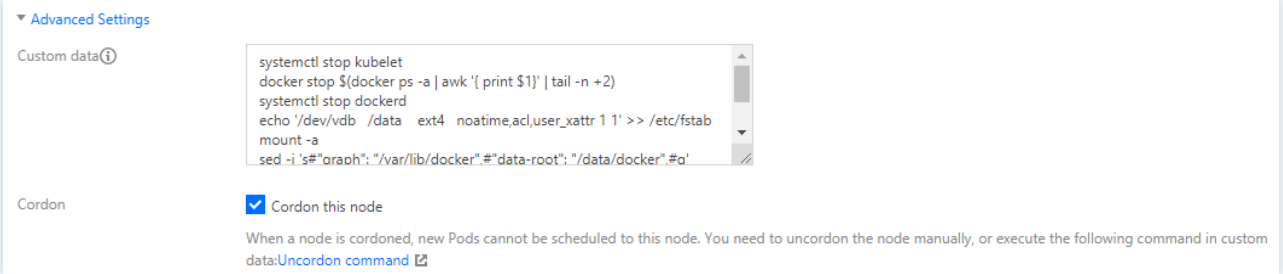
Mount Data Disk and **Container Directory** are not selected by default.

If you need to store the container and image on the data disk, select **Mount Data disk**. When *Mount Data Disk** is selected, formatted system disks of ext3, ext4 or XFS file systems will be mounted directly. Data disks of other file systems or unformatted data disks will be automatically formatted as ext4 and mounted.

If you need to keep the data on the data disk and mount the data disk without formatting it, perform the steps below:

i. On the **CVM Configuration** page, do not select **Mount Data Disk**.

ii. Open **Advanced Settings**. In the **Custom Data** area, enter the following node initialization script and select **Cordon this node**.



©2013-2022 Tencent Cloud. All rights reserved.

```
systemctl stop kubelet
docker stop $(docker ps -a | awk '{ print $1}' | tail -n +2)
systemctl stop dockerd
echo '/dev/vdb /data ext4 noatime,acl,user_xattr 1 1' >> /etc/fstab
mount -a
sed -i 's#"graph": "/var/lib/docker",#"data-root": "/data/docker",#g' /etc/docker/daemon.json
systemctl start dockerd
systemctl start kubelet
```

4. Set the login password and security group according to your actual circumstances, click **Complete**, and wait for the node to be added successfully.

Step 4: remove the cordon

Note：
After the node is added successfully, it is still cordoned.

1. On the **Node List** page, select **More** > **Uncordon** in the **Operation** column of the node.
2. In the pop-up window, click **OK** to remove the cordon.

©2013-2022 Tencent Cloud. All rights reserved.

Using Ansible to Batch Operate TKE Nodes

Last updated：2020-11-11 10:55:16

Overview

When adding nodes to a TKE cluster, you can perform batch operations, such as modification of kernel parameters, by entering a script in **Custom Data**. However, if you need to perform batch operations on existing nodes, you can use the Ansible open-source tool described in this document.

How It Works

Ansible is a popular open-source OPS tool that can be used to directly perform batch operations on devices over SSH protocol, without the need to manually preinstall dependencies. The following figure shows how it works:

Directions

Preparing the Ansible control node

1. Select an instance as the Ansible control node, through which batch operations on existing TKE nodes can be initiated. You can select any instance in the VPC where the cluster is located as the control node (including any TKE node).
2. After selecting the control node, select the installation method:
   - For Ubuntu:

     ```
     sudo apt update && sudo apt install software-properties-common -y && sudo apt
     -add-repository --yes --update ppa:ansible/ansible && sudo apt install ansibl
     e -y
     ```

   - For CentOS:

©2013-2022 Tencent Cloud. All rights reserved.

```
sudo yum install ansible -y
```

Preparing the configuration file

Add private IPs of all target nodes to the `host.ini` file, with one IP address per line, as shown in the example below:

```
10.0.3.33
10.0.2.4
```

To operate on all nodes, you can run the following commands to generate the `host.ini` file:

```
kubectl get nodes -o jsonpath='{.items[*].status.addresses[?(@.type=="InternalIP")].address}' | tr ' ' '\n' > hosts.ini
```

Preparing the batch execution script

Define the batch operations that you want to perform in a script and save it as a script file, as shown in the following example:
A self-built image repository is created, and no certificate has been issued by an authority. It uses the certificate issued by HTTP or HTTPS. By default, an error occurs when dockerd pulls images from this repository. You can perform batch modification of the dockerd configuration on nodes to add the address of the self-built repository to `insecure-registries` in the dockerd configuration. This allows dockerd to ignore the certificate check. The content of the `modify-dockerd.sh` script file is as follows:

```
# yum install -y jq # centos
apt install -y jq # ubuntu
cat /etc/docker/daemon.json | jq '."insecure-registries" += ["myharbor.com"]' > /tmp/daemon.json
cp /tmp/daemon.json /etc/docker/daemon.json
systemctl restart dockerd
```

Using Ansible to perform batch script execution

Usually, when TKE nodes are added, they all point to the same SSH login key or password. Perform the following operations based on your actual situation:

Using a key

1. Prepare a key file, for example, `tke.key` .
2. Run the following command to authorize the key file.

©2013-2022 Tencent Cloud. All rights reserved.

```
chmod 0600 tke.key
```

3. Perform batch script execution.
   - Sample for Ubuntu nodes:

```
ansible all -i hosts.ini --ssh-common-args="-o StrictHostKeyChecking=no -o Us
erKnownHostsFile=/dev/null" --user ubuntu --become --become-user=root --priva
te-key=tke.key -m script -a "modify-dockerd.sh"
```

   - Sample for other operating systems:

```
ansible all -i hosts.ini --ssh-common-args="-o StrictHostKeyChecking=no -o Us
erKnownHostsFile=/dev/null" --user root -m script -a "modify-dockerd.sh"
```

Using a password

1. Run the following command to pass a password into a PASS variable.

```
read -s PASS
```

2. Perform batch script execution.
   - For nodes on Ubuntu, the default SSH username is `ubuntu`. See the sample below:

```
ansible all -i hosts.ini --ssh-common-args="-o StrictHostKeyChecking=no -o Us
erKnownHostsFile=/dev/null" --user ubuntu --become --become-user=root -e "ans
ible_password=$PASS" -m script -a "modify-dockerd.sh"
```

   - For nodes on other operating systems, the default SSH username is `root`. See the sample below:

```
ansible all -i hosts.ini --ssh-common-args="-o StrictHostKeyChecking=no -o Us
erKnownHostsFile=/dev/null" --user root -e "ansible_password=$PASS" -m script
-a "modify-dockerd.sh"
```

©2013-2022 Tencent Cloud. All rights reserved.

Using Cluster Audit for Troubleshooting

Last updated：2023-05-06 17:36:46

Overview

Cluster resources may be deleted or modified in the case of misoperations, application bugs, or apiserver API calls from malicious programs. You can use the cluster audit feature to keep logs of apiserver API calls. In this way, you can search and analyze audit logs to find the causes of problems. This document describes how to use the cluster audit feature for troubleshooting.

Note

This document applies to only TKE clusters.

Prerequisites

You have enabled the cluster audit feature in the TKE console. For more information, see Enabling cluster audit.

Use Cases

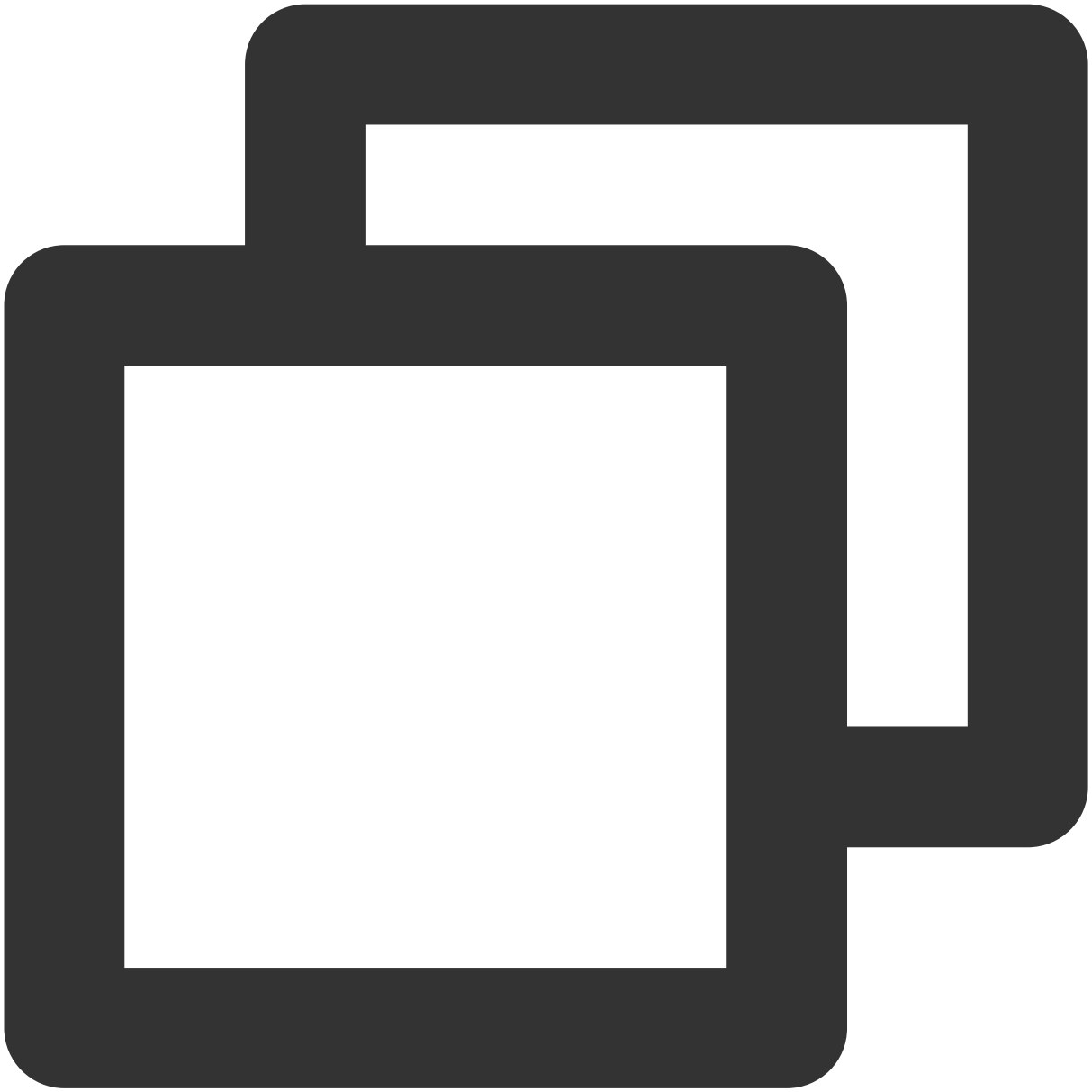
Obtaining the analysis result

1. Log in to the Cloud Log Service (CLS) console. In the left sidebar, click **Search and Analysis**.
2. On the **Search and Analysis** page, select the logset and log topic to search and a time scope.
3. Enter an analysis statement and click **Search and Analysis** to obtain the analysis result.

Example 1: querying the operator who cordoned a node

To query the operator who cordoned a node, run the following command:

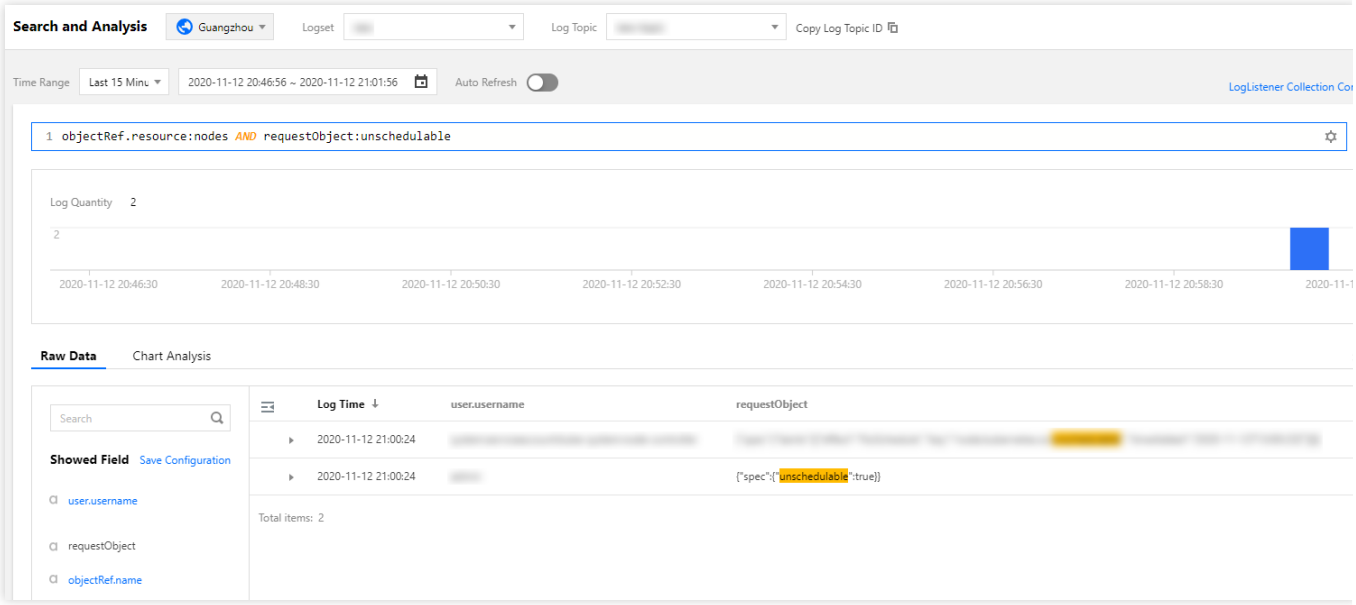
©2013-2022 Tencent Cloud. All rights reserved.



```
objectRef.resource:nodes AND requestObject:unschedulable
```

On the **Search and Analysis** page, select **Default Configuration** for the layout. The following figure shows the query result:

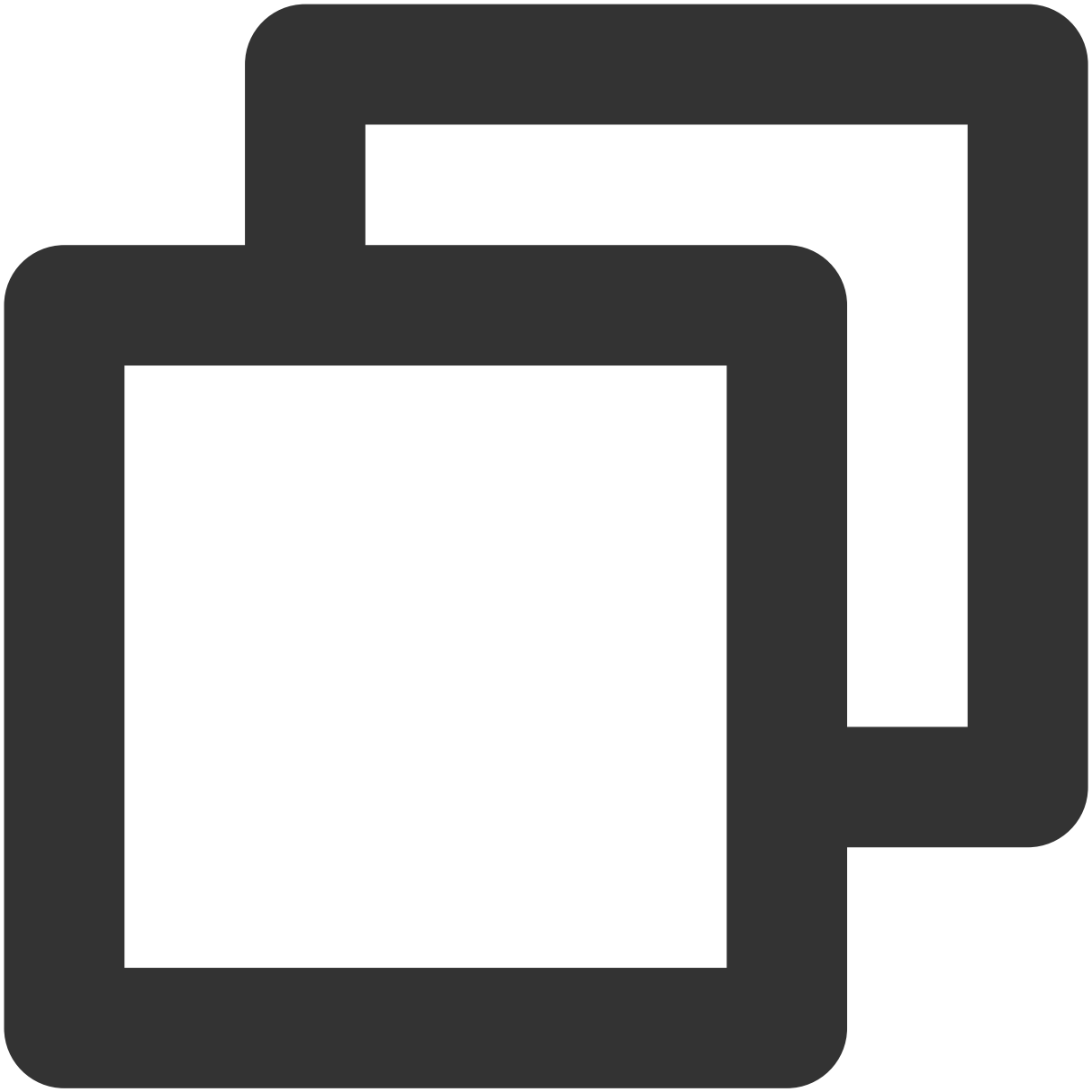
©2013-2022 Tencent Cloud. All rights reserved.



Example 2: querying the operator who deleted a workload

To query the operator who deleted a workload, run the following command:

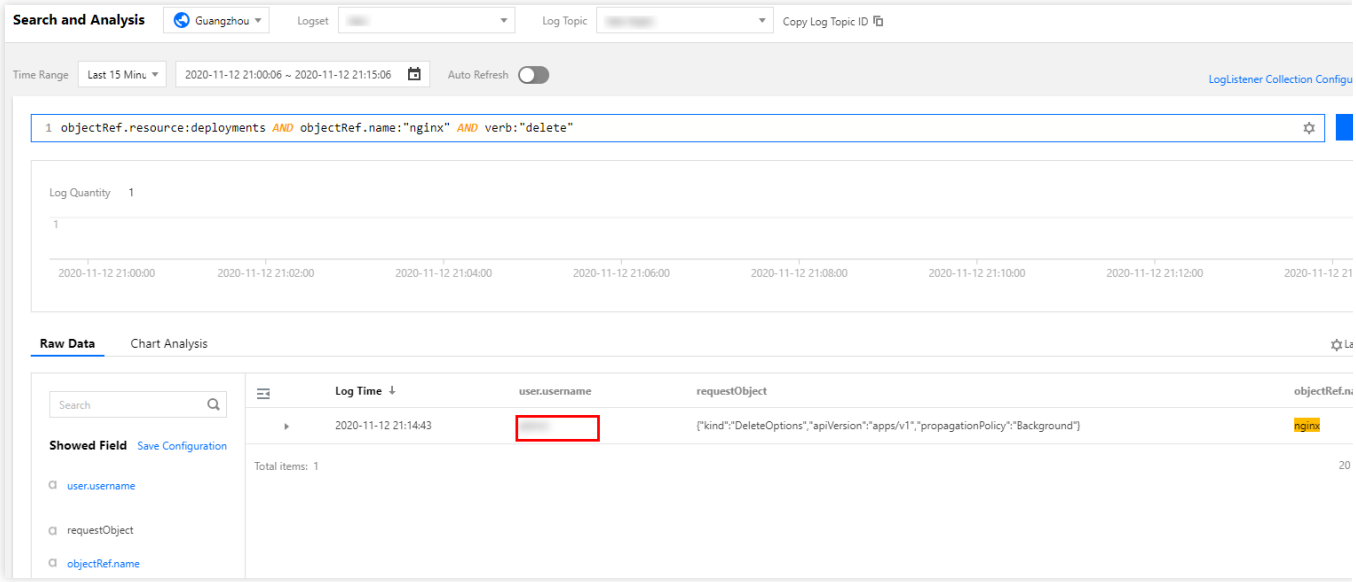
©2013-2022 Tencent Cloud. All rights reserved.



```
objectRef.resource:deployments AND objectRef.name:"nginx" AND verb:"delete"
```

You can obtain detailed information about the operator sub-account from the query result.

©2013-2022 Tencent Cloud. All rights reserved.

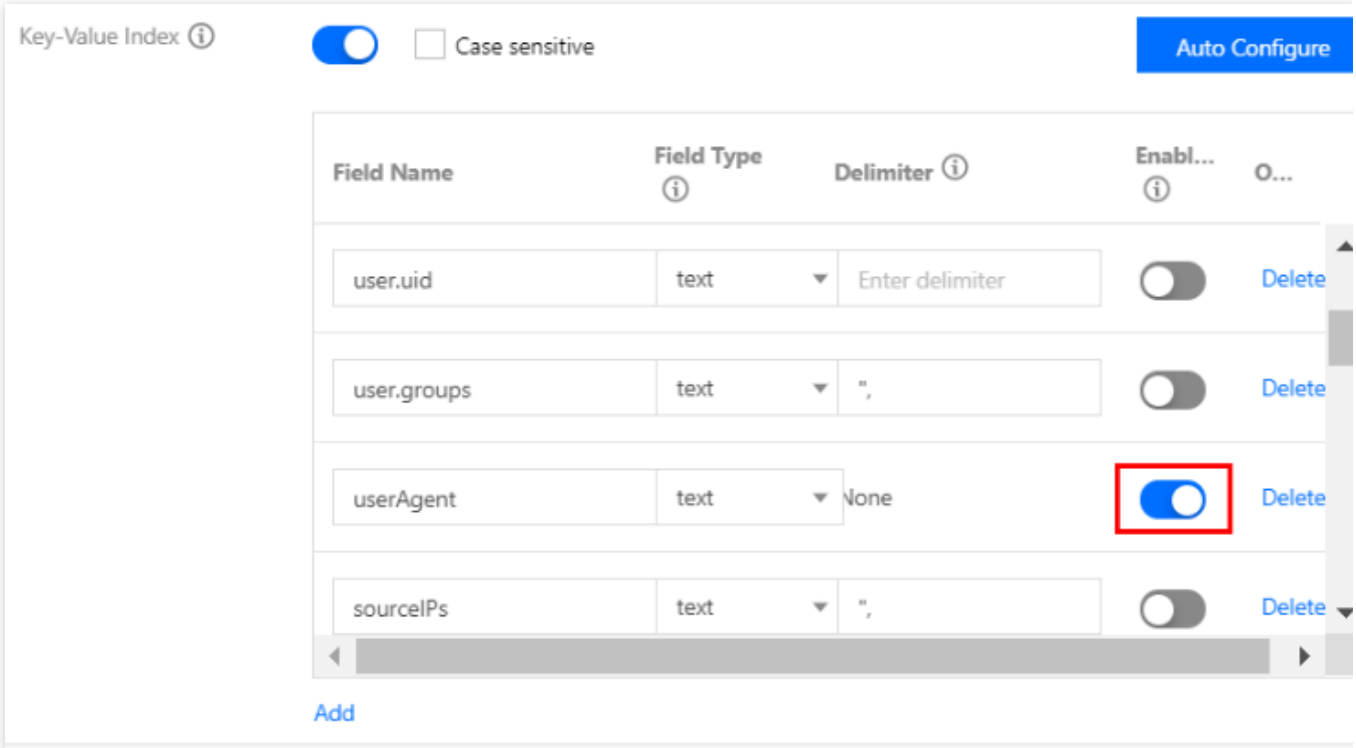


.

Example 3: locating the causes of apiserver access limitation

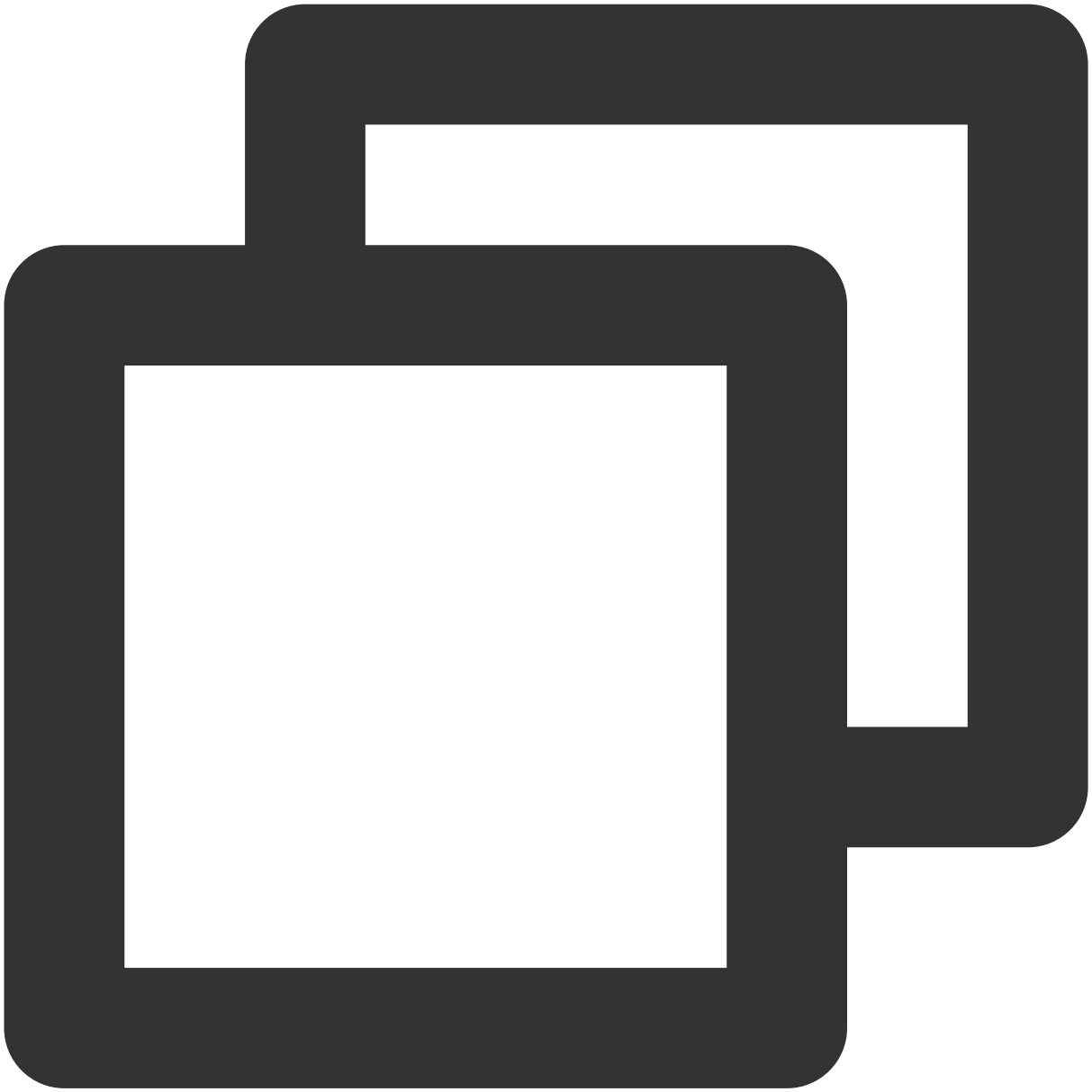
To prevent apiserver/etcd from being overloaded due to frequent apiserver access caused by malicious programs or bugs, apiserver enables an access limit mechanism by default. If the access limit is reached, you can identify the clients that have sent large numbers of requests through audit logs.

1. If you need to analyze clients that send requests based on userAgent, modify the log topic in the **Key-Value Index** window and collect statistics based on the userAgent field, as shown below:



2. Run the following command to collect QPS statistics from each client to the apiserver:

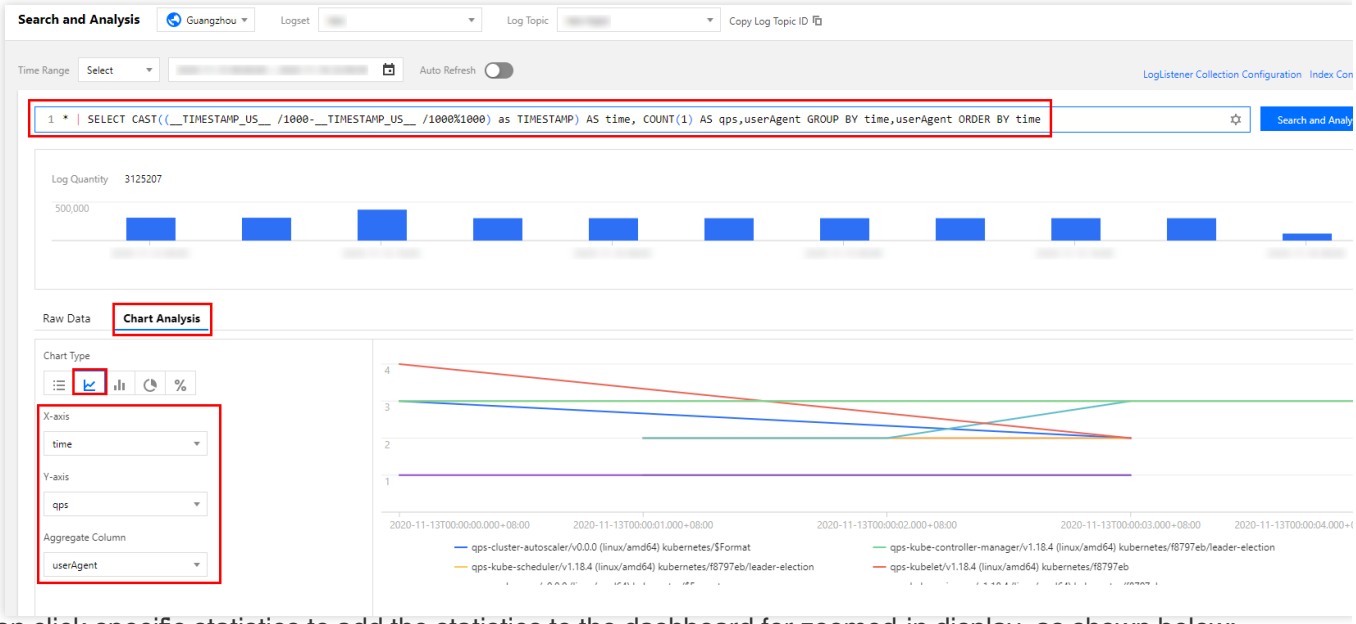
©2013-2022 Tencent Cloud. All rights reserved.



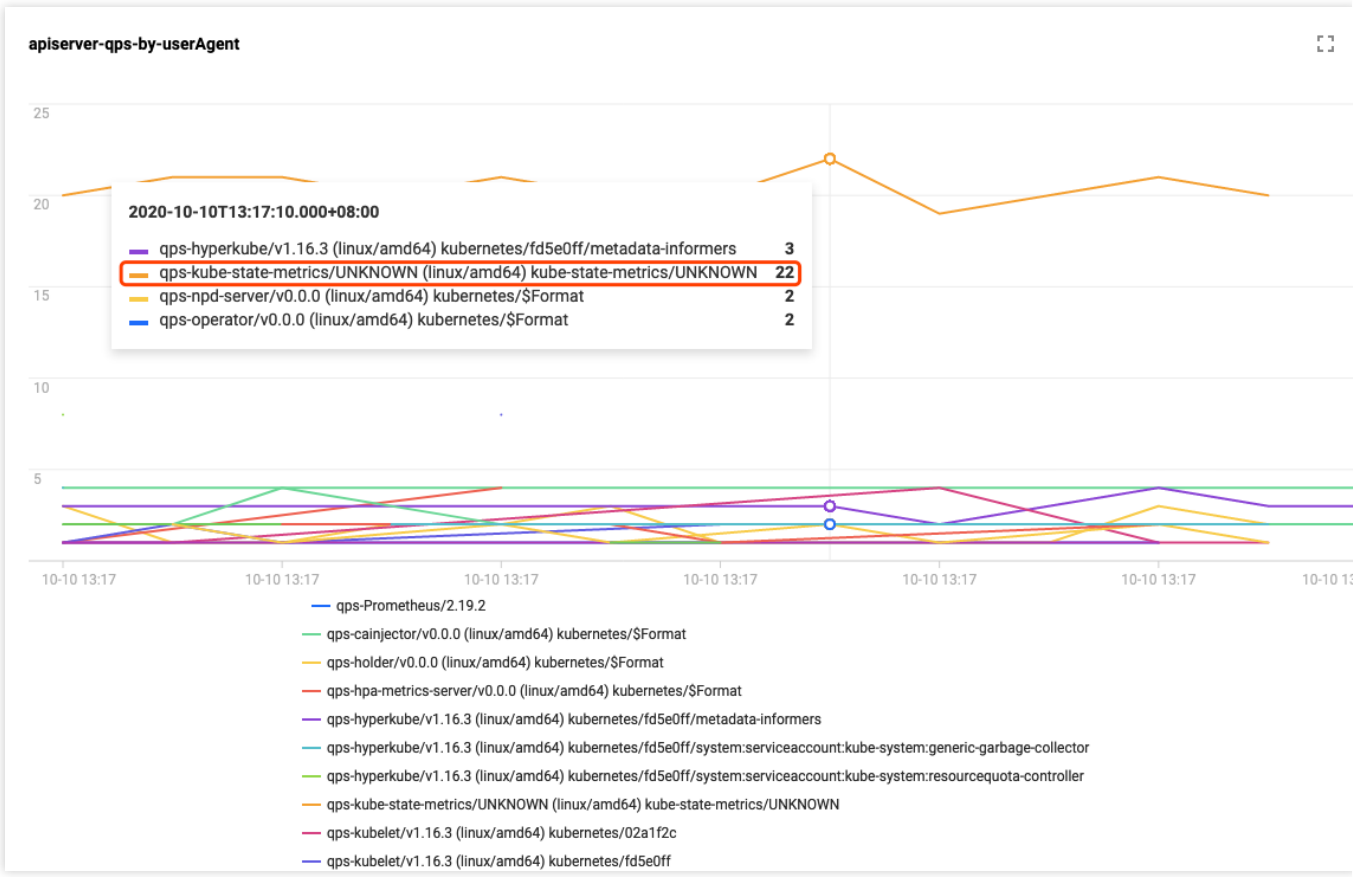
```
* | SELECT histogram( cast(__TIMESTAMP__ as timestamp),interval 1 minute) AS time,
```

3. Switch to the statistical chart and select the sequence diagram. Specify the basic information and coordinate axes, as shown below:

©2013-2022 Tencent Cloud. All rights reserved.

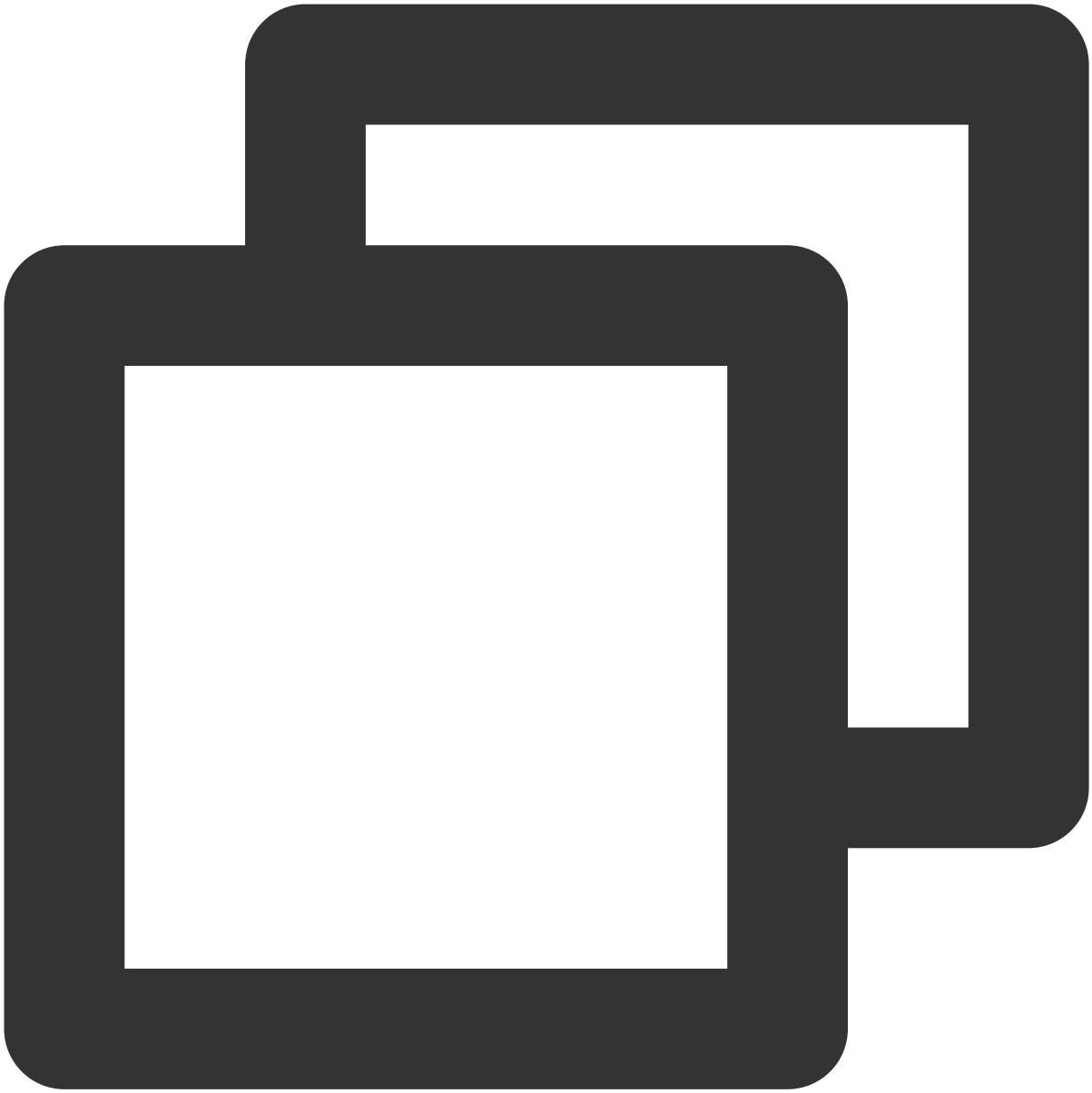


You can click specific statistics to add the statistics to the dashboard for zoomed-in display, as shown below:



As can be seen in the figure above, the client kube-state-metrics sends far more requests than the other clients. According to the logs, kube-state-metrics frequently sends requests to the apiserver due to RBAC permission issues. As a result, the apiserver access limit is triggered. The logs involved are as follows:

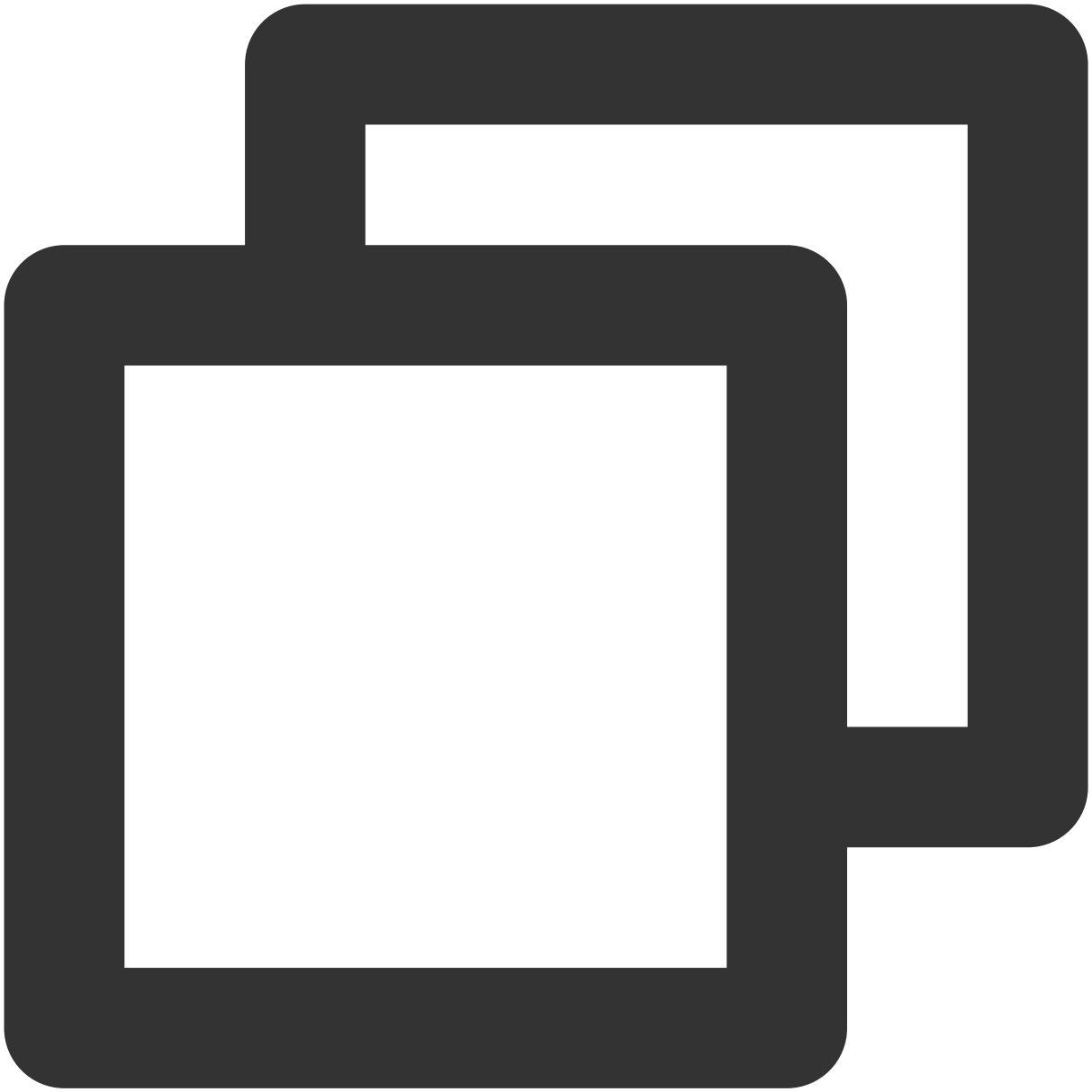
©2013-2022 Tencent Cloud. All rights reserved.



```
I1009 13:13:09.760767       1 request.go:538] Throttling request took 1.393921018s,
E1009 13:13:09.766106       1 reflector.go:156] pkg/mod/k8s.io/client-go@v0.0.0-201
```

To use other fields, such as user.username, to distinguish the clients to collect data on, you can modify the SQL statement as required. An example SQL statement is as follows:

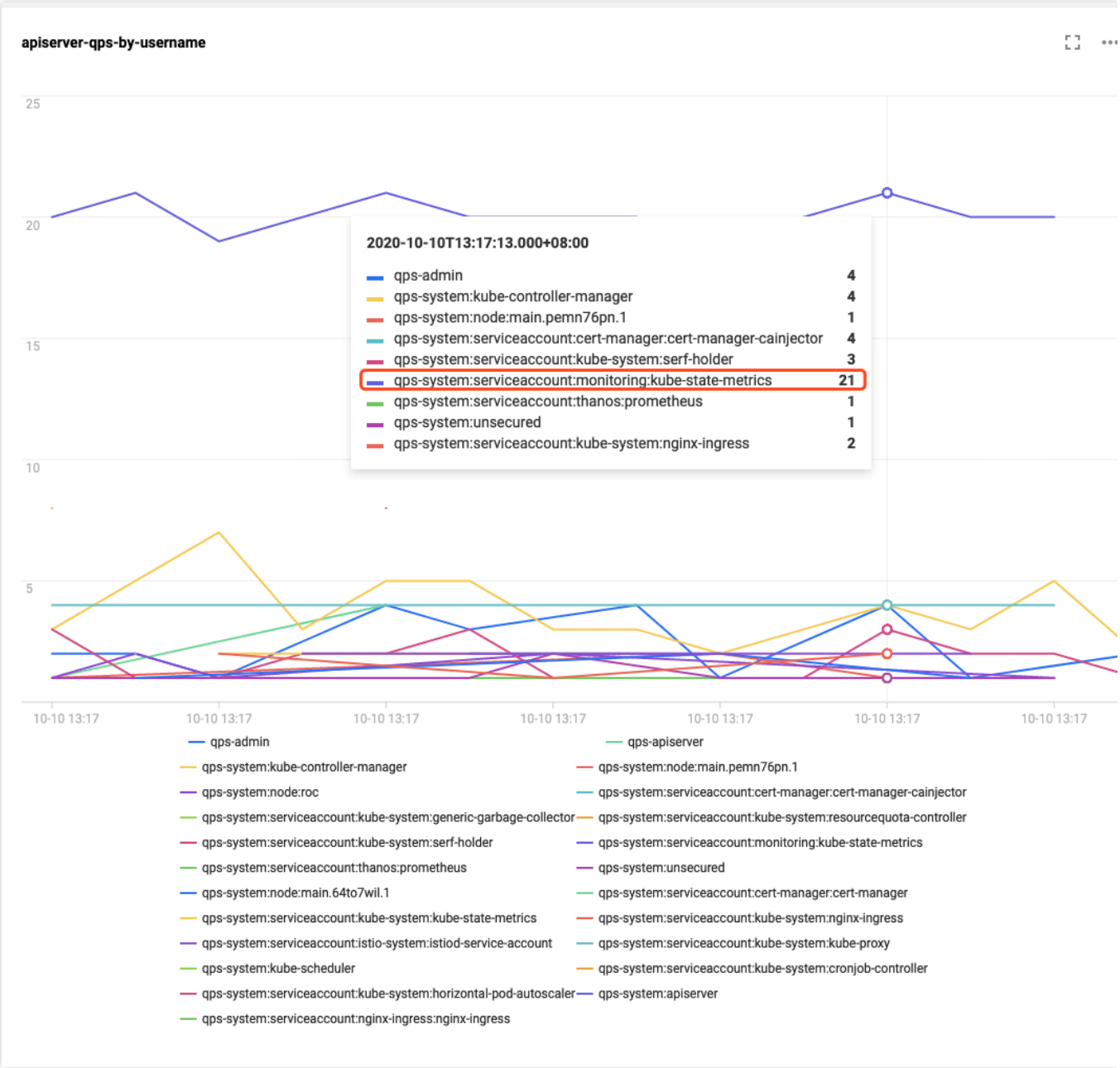
©2013-2022 Tencent Cloud. All rights reserved.



```
* | SELECT histogram( cast(__TIMESTAMP__ as timestamp),interval 1 minute)  AS time,
```

The following figure shows the display result:

©2013-2022 Tencent Cloud. All rights reserved.



References

For more information about the TKE cluster audit feature and basic operations, see Cluster Audit.

Cluster audit data is stored in CLS. To query and analyze audit data in the CLS console, see Syntax Rules for the search syntax.

To analyze audit data, an SQL statement supported by CLS is required. For more information, see Overview.

©2013-2022 Tencent Cloud. All rights reserved.

Renewing a TKE Ingress Certificate

Last updated：2022-05-31 11:53:12

Overview

Ingress certificates created in the Tencent Kubernetes Engine (TKE) console will reference certificates hosted in the SSL Certificate Service. If an Ingress is used for a long time, the Ingress certificate may expire, which will have a major impact on online businesses. This document describes how to renew an Ingress certificate before it expires.

Directions

Querying the certificate expiration time

1. Log in to the SSL Certificate Service console and click **Certificate Management** in the left sidebar.
2. In the certificate list, click **Expiry date** to view certificates that are about to expire.

Adding a certificate

On the **Certificate management** page, you can renew an existing certificate to generate a new certificate. You can **Purchase certificate**, **Apply for free certificate**, or **Upload certificate** to add a certificate.

Viewing Ingresses referencing old certificate

1. Log in to the SSL Certificate Service console and select **Associate cloud resources** next to a certificate to view the load balancer that references this certificate.
2. Click the load balancer ID to redirect to the CLB details page. If the CLB is used for the TKE Ingress, `tke-clusterId` and `tke-lb-ingress-uuid` will appear in the **Tag** section. `tke-clusterId` and `tke-lb-ingress-uuid` indicate the cluster ID and Ingress UID, respectively.
3. On the **Basic info** page of the CLB, click the editing icon in the tag line to enter the **Edit tags** page.
4. Use Kubectl to query the Ingress of the cluster based on the cluster ID and filter out the Ingress resource whose UID is `tke-lb-ingress.uuid`. The sample reference code is as follows:

```
$ kubectl get ingress --all-namespaces -o=custom-columns=NAMESPACE:.metadata.namespace,INGRESS:.metadata.name,UID:.metadata.uid | grep 1a******-****-****-a329-eec697a28b35
api-prod gateway 1a******-****-****-a329-eec697a28b35
```

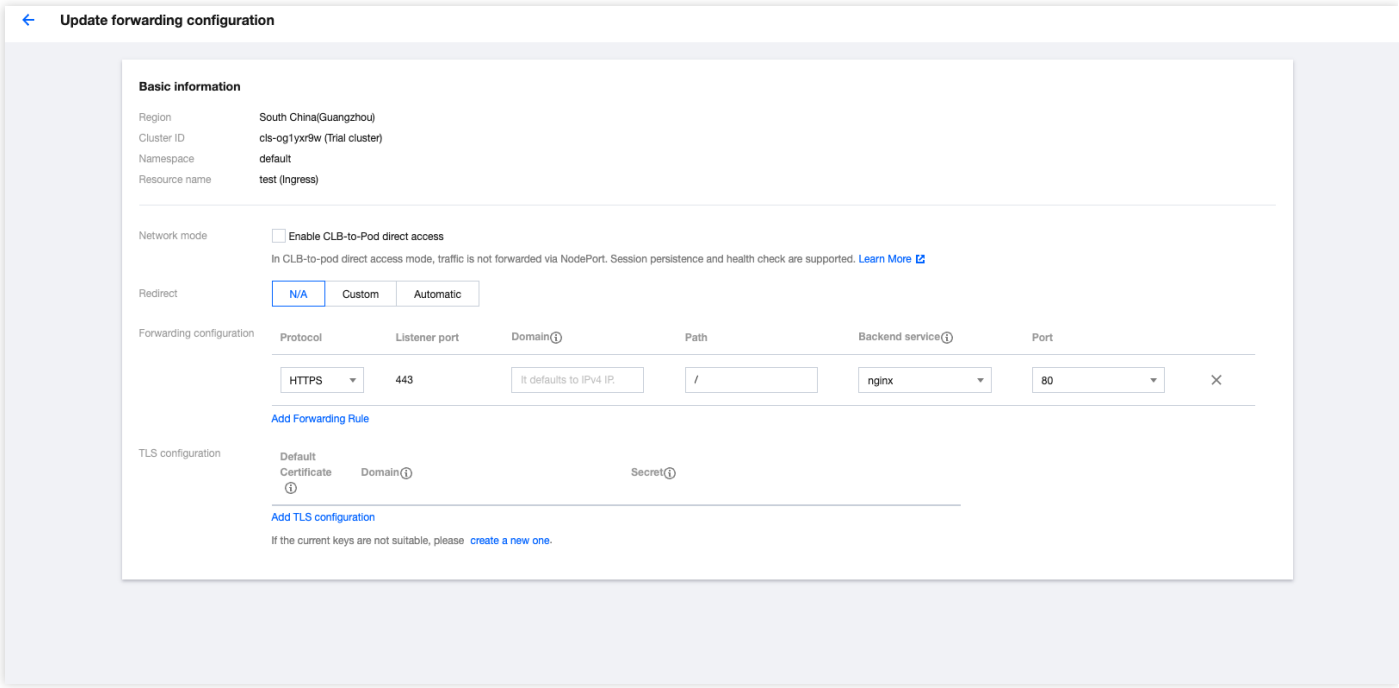
©2013-2022 Tencent Cloud. All rights reserved.

According to the query result, `api-prod/gateway` in this cluster references the certificate. Therefore, this Ingress needs to be updated.

Updating an Ingress

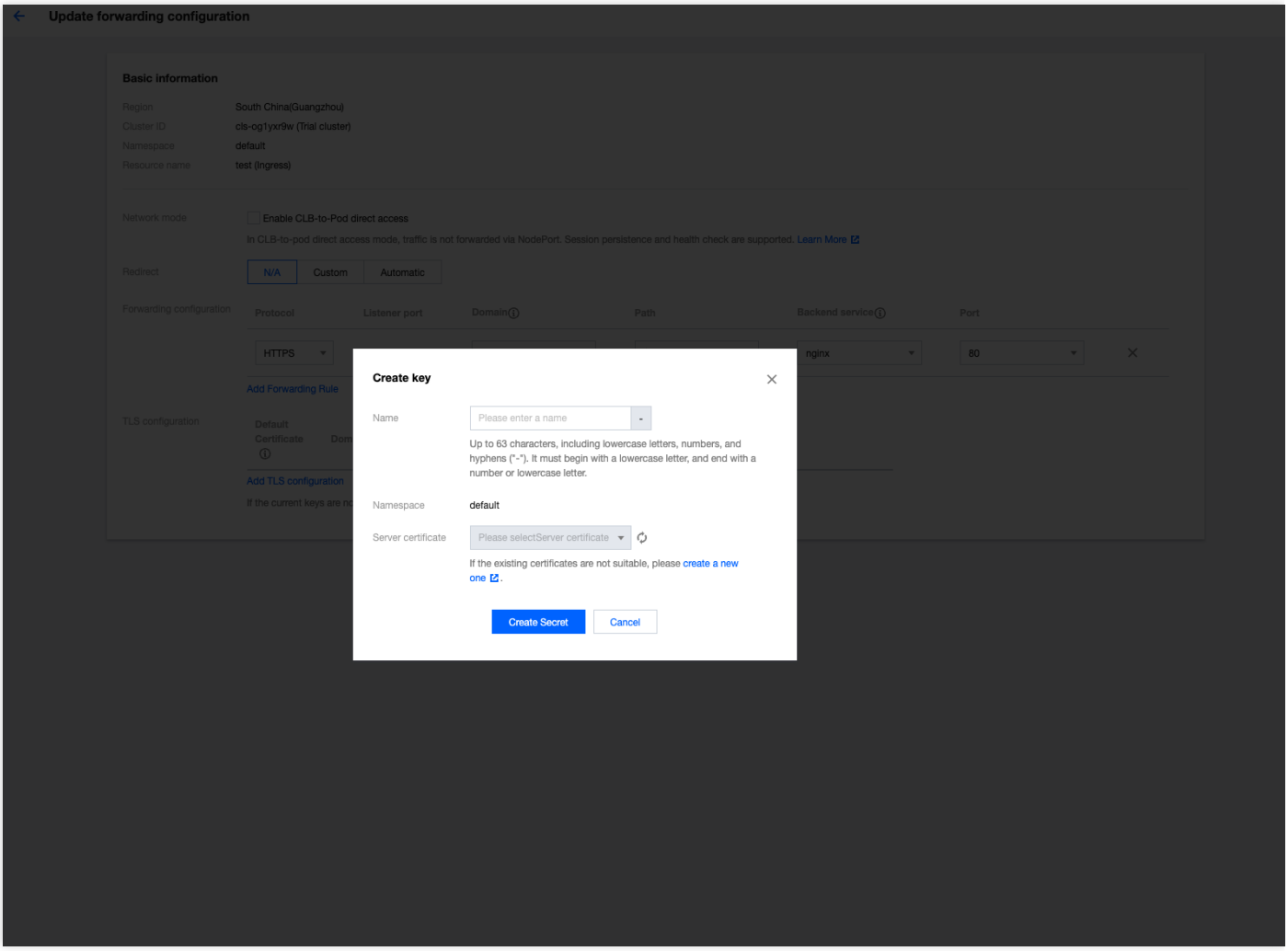
1. In the TKE console, find the Ingress that references the old certificate and click **Update forwarding configuration**.

2. On the **Update forwarding configuration** page, create a secret for the new certificate.



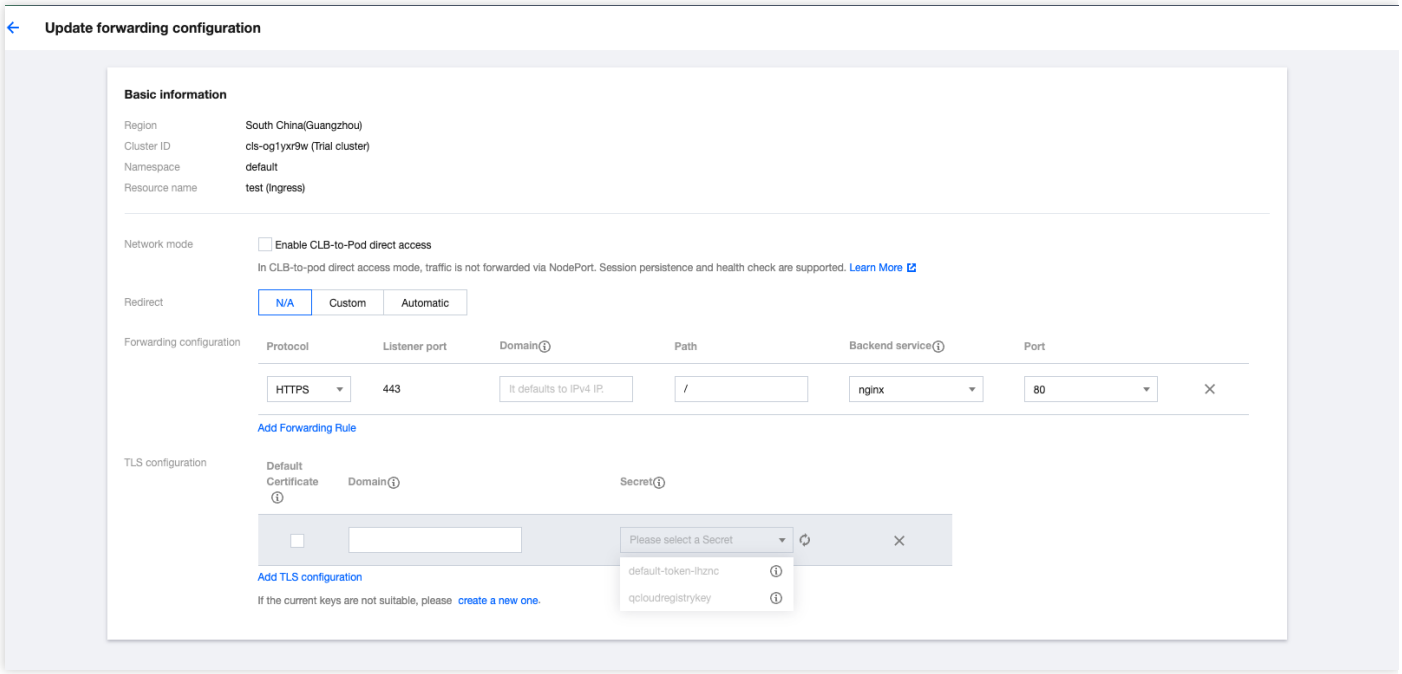
On the **Create key** page, select the new certificate and click **Create secret**.

©2013-2022 Tencent Cloud. All rights reserved.



Return to the **Update forwarding configuration** page, modify the TLS configuration of the Ingress, and add the created certificate secret.

©2013-2022 Tencent Cloud. All rights reserved.



Click **Update forwarding configuration** to renew the Ingress certificate.

©2013-2022 Tencent Cloud. All rights reserved.

Using cert-manager to Issue Free Certificates

Last updated：2023-05-23 17:56:00

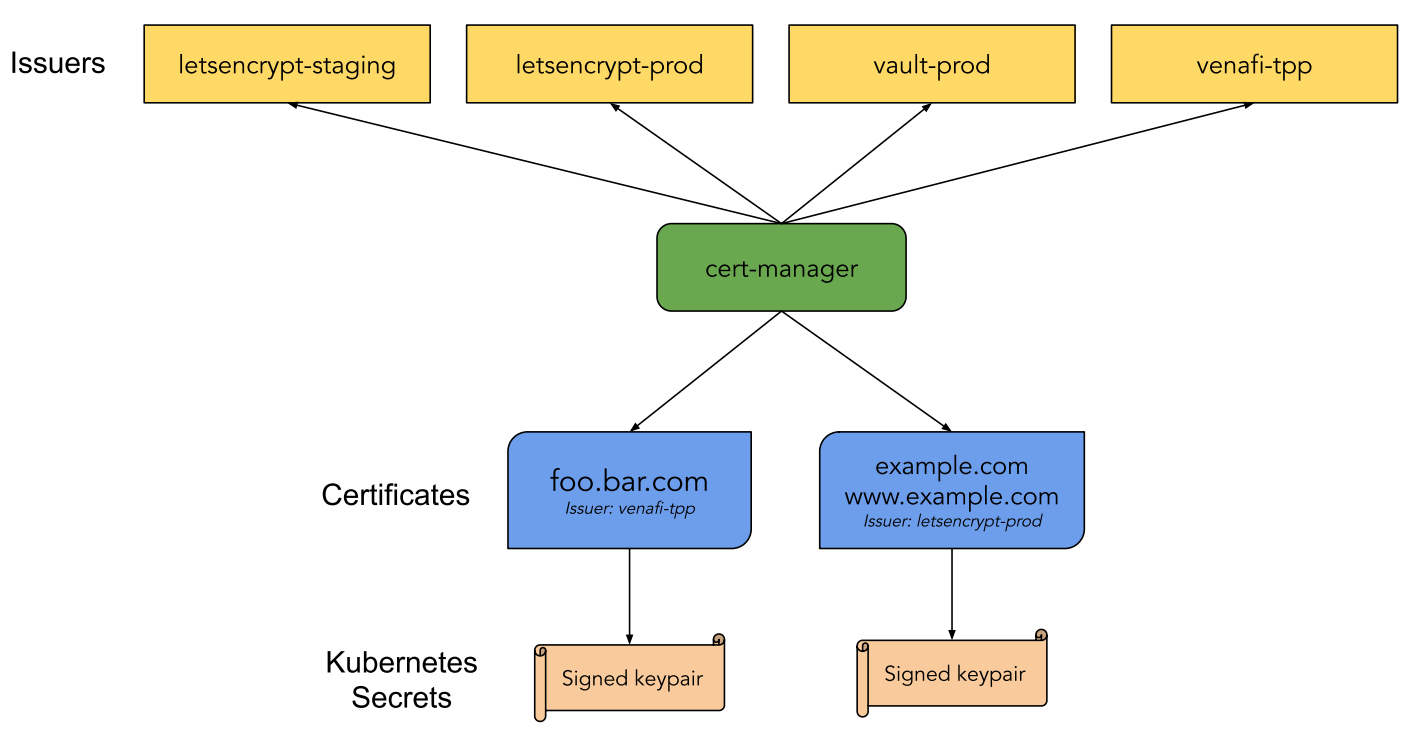
Overview

As HTTPS becomes increasingly popular, most websites have begun to upgrade from HTTP to HTTPS. To use HTTPS, you need to apply for a certificate from an authority and pay a certain cost. The more certificates you apply for, the higher the cost will be. cert-manager is a powerful certificate management tool for Kubernetes. You can use cert-manager based on the ACME protocol and Let's Encrypt to issue free certificates and have certificates automatically renewed. In this way, you can use certificates permanently for free.

Principles

How cert-manager works

After being deployed to a Kubernetes cluster, cert-manager queries custom CRD resources that it supports. You can create CRD resources to instruct cert-manager to issue certificates and automatically renew certificates, as shown in the figure below:



©2013-2022 Tencent Cloud. All rights reserved.

- **Issuer/ClusterIssuer**: indicates the method used by cert-manager to issue certificates. This document mainly describes the ACME method for issuing free certificates.

> Note：
> Issuer differs from ClusterIssuer in that Issuer can only be used to issue certificates under your own namespace, whereas ClusterIssuer can be used to issue certificates under any namespace.

- **Certificate**: is used to pass the domain name certificate information, the configuration required for issuing a certificate, and Issuer/ClusterIssuer references to cert-manager.

Issuing a free certificate

Let's Encrypt uses the ACME protocol to verify the ownership of a domain name. After successful verification, a free certificate is automatically issued. The free certificate is valid for only 90 days, so verification needs to be performed again to renew the certificate before the certificate expires. cert-manager supports automatic renewal of certificates, which allows you to use certificates permanently for free. You can verify the ownership of a certificate by using two methods: **HTTP-01** and **DNS-01**. For more information on the verification process, see How It Works.

- HTTP-01 verification
- DNS-01 verification

HTTP-01 verification adds a temporary location for the HTTP service to which a domain name is directed. This method is only applicable to issuing a certificate for services that use open ingress traffic and does not support wildcard certificates.
For example, Let's Encrypt sends an HTTP request to `http://<your_domain>/.well-known/acme-challenge/<token>`. `YOUR_DOMAIN` indicates the domain name to be verified, and `TOKEN` indicates a file placed by the ACME client. In this case, the ACME client is cert-manager. You can modify or create ingress rules to add temporary verification paths and direct them to the service that provides `TOKEN`. Let's Encrypt will then verify whether `TOKEN` meets the expectation. If the verification succeeds, a certificate is issued.

Verification method comparison
The HTTP-01 methods features simple configuration and extensive applicability. Different DNS providers can use the same configuration method. The disadvantages of this method are that it relies on ingress resources, is applicable only to services that support open ingress traffic, and does not support wildcard certificates.
The advantages of DNS-01 are that it does not rely on ingress resources and supports wildcard domain names. Its disadvantages are that different DNS providers have different configuration methods, and cert-manager Issuer does not support too many different DNS providers. However, you can deploy the cert-manager-enabled webhook service to extend Issuer in order to support more DNS providers, such as DNSPod and Alibaba DNS. For more information on

©2013-2022 Tencent Cloud. All rights reserved.

supported providers, see the webhook list.
This document uses the recommended `DNS-01` method, which offers comprehensive features with few restrictions.

Directions

Installing cert-manager

Usually, you can use YAML to install cert-manager in your cluster with one click. For more information, see this document on the official website: Installing with regular manifests.
The official image used by cert-manager can be pulled from `quay.io` . Alternatively, you can run the following command to use the image synchronized to the mainland China CCR for one-click installation:

Configuring DNS

Log in to a DNS provider backend system, configure the DNS A record of the domain name, and direct it to the opened IP address of the real server that needs the certificate. To do this, see the figure below, where Cloudflare is used as an example.

Issuing a certificate by using the HTTP-01 verification method

HTTP-01 validation can be performed by using Ingress. Cert-manager will automatically modify the Ingress or add an Ingress to expose the temporary HTTP path needed for validation. When HTTP-01 validation is configured for Issuer, if the `name` of an Ingress is specified, the specified Ingress will be modified to expose the HTTP path needed for validation. If `class` is specified, an Ingress will be added automatically. You can refer to the following Example.

Each ingress provided by TKE corresponds to a CLB. If you use an existing ingress provided by TKE to open services while using the HTTP-01 verification method, you can only adopt the automatic ingress modification mode, but not the automatic ingress addition mode. For automatically added ingresses, other CLBs will be automatically created, causing the opened IP address inconsistent with the ingress of the real server. In this case, Let's Encrypt fails to find the temporary path needed for verification in the service ingress, which results in verification failure and the failure to issue a certificate. If you use a user-created ingress, for example, by deploying Nginx Ingress on TKE, and ingresses in the same ingress class share the same CLB, the automatic ingress addition mode is supported.

©2013-2022 Tencent Cloud. All rights reserved.

Example

If you use an ingress provided by TKE to open a service, you cannot use cert-manager to issue and manage free certificates. This is because certificates are referenced in Certificate Management and are not managed in Kubernetes.

If you deploy Nginx Ingress on TKE and the ingress of the real server is `prod/web`, you can create an Issuer by referring to the following sample code:

```
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
name: letsencrypt-http01
namespace: prod
spec:
acme:
server: https://acme-v02.api.letsencrypt.org/directory
privateKeySecretRef:
name: letsencrypt-http01-account-key
solvers:
- http01:
ingress:
name: web # Specifies the name of the ingress for automatic modification.
```

When you use an Issuer to issue a certificate, cert-manager will automatically create an ingress and automatically modify `prod/web` of the ingress to open the temporary path needed for verification. See the following sample code for automatic ingress addition:

```
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
name: letsencrypt-http01
namespace: prod
spec:
acme:
server: https://acme-v02.api.letsencrypt.org/directory
privateKeySecretRef:
name: letsencrypt-http01-account-key
solvers:
- http01:
ingress:
class: nginx # Specifies the ingress class of the automatically created ingress.
```

After successfully creating an Issuer, refer to the following sample code to create a certificate and reference the Issuer to issue the certificate:

©2013-2022 Tencent Cloud. All rights reserved.

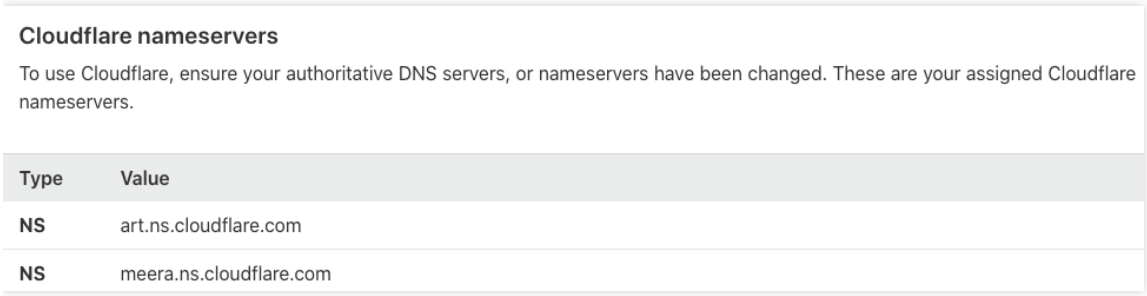
```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
name: test-mydomain-com
namespace: prod
spec:
dnsNames:
- test.mydomain.com # Indicates the domain name for issuing a certificate.
issuerRef:
kind: Issuer
name: letsencrypt-http01 # References Issuer and indicates the HTTP-01 method is used for verification.
secretName: test-mydomain-com-tls # The issued certificate will be saved in this Secret.
```

Issuing a certificate by using the DNS-01 verification method

If you choose to use the DNS-01 verification method, you must select a DNS provider. cert-manager provides built-in support for DNS providers. For the detailed list and usage, see Supported DNS01 providers. If you need to use a DNS provider other than those on the list, refer to the following two schemes:

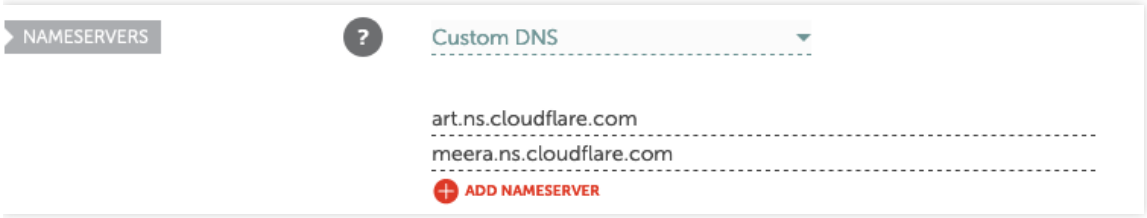
- Scheme 1: Configuring a custom nameserver
- Scheme 2: Using webhooks

On the backend system of the DNS provider, configure a custom nameserver and direct it to the address of a nameserver that can manage other DNS providers' domain names, such as Cloudflare. You can log in to the backend of Cloudflare to view the specific address, as shown in the figure below:



You can configure a custom nameserver for namecheap, as shown in the figure below:

©2013-2022 Tencent Cloud. All rights reserved.



Finally, when configuring the Issuer and specifying the DNS-01 verification method, add the Cloudflare information.

Obtaining and using certificates

After creating a certificate, you can run the kubectl command to check whether the certificate has been issued successfully.

```
$ kubectl get certificate -n prod
NAME READY SECRET AGE
test-mydomain-com True test-mydomain-com-tls 1m
```

- `READY` = `False` : indicates that the certificate failed to be issued. You can run the `describe` command to check the event and analyze the failure cause.

  ```
  $ kubectl describe certificate test-mydomain-com -n prod
  ```

- `READY` = `True` : indicates that the certificate was issued successfully. In this case, the certificate will be stored in the specified Secret, for example, `default/test-mydomain-com-tls` . You can run kubectl to view the certificate, where `tls.crt` indicates the certificate, and `tls.key` indicates the key.

  ```
  $ kubectl get secret test-mydomain-com-tls -n default
  ...
  data:
  tls.crt: <cert>
  tls.key: <private key>
  ```

You can mount the certificate to the app that needs it or directly reference the Secret in an ingress that you created. The following shows a sample YAML file:

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
name: test-ingress
```

©2013-2022 Tencent Cloud. All rights reserved.

```
annotations:
kubernetes.io/Ingress.class: nginx
spec:
rules:
- host: test.mydomain.com
http:
paths:
- path: /web
backend:
serviceName: web
servicePort: 80
tls:
hosts:
- test.mydomain.com
secretName: test-mydomain-com-tls
```

References

- cert-manager official website
- How It Works
- API reference docs
- Certificate

©2013-2022 Tencent Cloud. All rights reserved.

Using cert-manager to Issue Free Certificate for DNSPod Domain Name

Last updated：2021-12-03 16:13:23

Overview

If you use DNSPod to manage your domain names and want to automatically issue free certificates for domain names in Kubernetes, you can use cert-manager to this end:

cert-manager supports many DNS providers but not DNSPod. However, it offers a webhook to support more providers, and support for DNSPod is also implemented in the community. This document describes how to use cert-manager and cert-manager-webhook-dnspod to automatically issue free certificates for domain names in DNSPod.

Basic Knowledge

We recommend you read Using cert-manager to Issue Free Certificates first.

Directions

1. Create a DNSPod key

Log in to the DNSPod console. In Key Management, create a key and copy the automatically generated `ID` and `Token`

2. Install cert-manager

Install cert-manager. For more information, please see Using cert-manager to Issue Free Certificates.

3. Install cert-manager-webhook-dnspod

Use HELM to install cert-manager-webhook-dnspod. You need to prepare the HELM configuration file.
Below is a sample `dnspod-webhook-values.yaml` :

```
groupName: example.your.domain # Enter a custom group name

secrets: # Paste the generated ID and token below
apiID: "<id>"
apiToken: "<token>"
```

©2013-2022 Tencent Cloud. All rights reserved.

```
clusterIssuer:
enabled: true # Automatically create a ClusterIssuer
email: your@email.com # Enter your email address
```

For the complete configuration, please see values.yaml.

Use HELM for installation:

```
git clone --depth 1 https://github.com/qqshfox/cert-manager-webhook-dnspod.git
helm upgrade --install -n cert-manager -f dnspod-webhook-values.yaml cert-manage
r-webhook-dnspod ./cert-manager-webhook-dnspod/deploy/cert-manager-webhook-dnspo
d
```

4. Create a certificate

Use the following YAML file to create a `Certificate` object to issue a free certificate:

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
name: example-com-crt
namespace: istio-system
spec:
secretName: example-com-crt-secret # The certificate is stored in this secret
issuerRef:
name: cert-manager-webhook-dnspod-cluster-issuer # The automatically generated C
lusterIssuer is used here
kind: ClusterIssuer
group: cert-manager.io
dnsNames: # Enter the list of domain names for which to issue certificates. Ensu
re that all the domain names are managed by DNSPod
- example.com
- test.example.com
```

If the status becomes `READY`, the certificate is successfully issued:

```
$ kubectl -n istio-system get certificates.cert-manager.io
NAME READY SECRET AGE
example-com-crt True example-com-crt-secret 25d
```

If the issuance fails, you can run `describe` to view the cause:

```
kubectl -n istio-system describe certificates.cert-manager.io example-com-crt
```

©2013-2022 Tencent Cloud. All rights reserved.

5. Use the certificate

After the certificate is successfully issued, it will be stored in the specified `Secret` as follows:

- Use in Ingress
- Use in Istio ingress gateway

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
name: test-ingress
annotations:
kubernetes.io/ingress.class: nginx
spec:
rules:
- host: test.example.com
http:
paths:
- path: /
backend:
serviceName: web
servicePort: 80
tls:
hosts:
- test.example.com
secretName: example-com-crt-secret # Reference the certificate secret
```

©2013-2022 Tencent Cloud. All rights reserved.

Using the TKE NPDPlus Plug-In to Enhance the Self-Healing Capability of Nodes

Last updated：2020-11-26 15:22:55

When a Kubernetes cluster is running, nodes may become unavailable due to component faults, kernel deadlocks, insufficient resources, and other causes. By default, the kubelet monitors the status of node resources such as PIDPressure, MemoryPressure, and DiskPressure. However, if nodes are already unavailable or the kubelet has started draining the pods when reporting node statuses, the native Kubernetes node health monitoring mechanism may not function properly. To detect node faults proactively, you need to add more specific metrics to describe node health status and adopt corresponding recovery policies to achieve smart OPS, reduce development costs, and migitate the burden on OPS personnel.

node-problem-detector

Node problem detector (NPD) is an open-source Kubernetes addon for node health detection. NPD enables users to set regular expressions to detect node exceptions in system logs or files. Based on the OPS experiences, users can set regular expressions that may generate exception logs and choose the report mode. NPD will parse the configuration file. When a log can match the regular expression rules set by the user, the detected exception status can be reported through NodeCondition, Event, or Prometheus Metric. Except for the log matching function, NPD also allows users to write custom detection addons. Users can develop their own script or executable file and integrate it into the NPD addon. In this way, NPD can execute the detection program periodically.

TKE NPDPlus Add-On

In TKE, NPD is enhanced and integrated as an add-on called NodeProblemDetectorPlus (NPDPlus). You can install this add-on in existing clusters with one click. Alternatively, you can deploy NPDPlus when creating a cluster. TKE extracts metrics that can detect node exceptions in certain ways and integrates these metrics into NPDPlus. For example, NPDPlus can detect the systemd status of the kubelet and Docker in containers as well as the CVM file descriptor and thread pressure.

TKE uses NPDPlus to detect node unavailability proactively, instead of reporting exceptions after nodes become unhealthy. After users deploy NPDPlus in a TKE cluster and run the command `kubectl describe node`, they can view some node conditions. For example, FDPressure indicates whether the number of file descriptors used on the node has reached 80% of the threshold allowed by the CVM, and ThreadPressure indicates whether the number

©2013-2022 Tencent Cloud. All rights reserved.

of threads on the node has reached 90% of the threshold allowed by the CVM. Users can monitor these conditions and configure preventive policies to minimize potential exceptions. For more information, see Node Conditions.

Meanwhile, the current opinion of Kubernetes is that the NotReady mechanism of nodes relies on the parameter settings of kube-controller-manager. Therefore, when a node network connection fails, Kubernetes can hardly detect node exceptions in seconds. In some scenarios (such as livestreaming and online conferences), this is unacceptable. NPDPlus inherits the distributed node health detection feature. It can detect node network status in seconds and check whether nodes can communicate with other nodes without communicating with the Kubernetes master component.

For more information on how to use the TKE NPDPlus add-on, see NodeProblemDetectorPlus Usage.

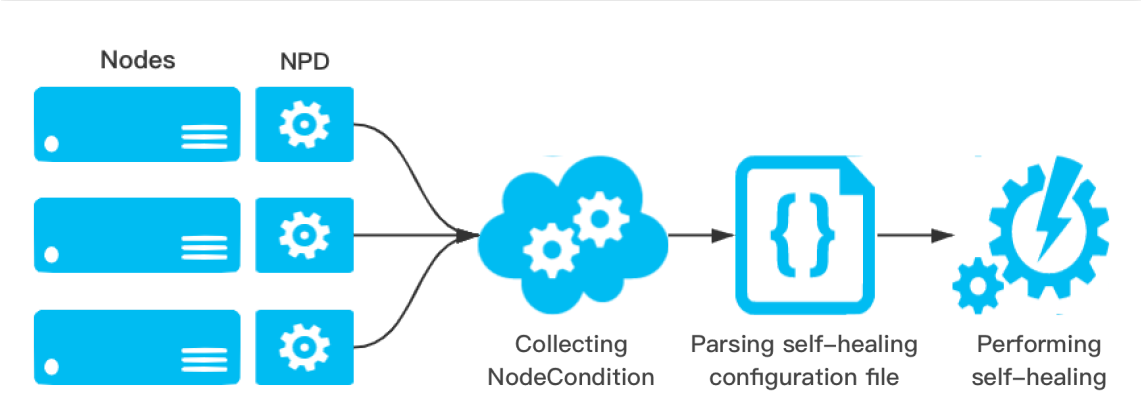
Node Self-Healing

The health status information of nodes is collected to proactively detect node exceptions before business pods become unavailable. This way, OPS or development personnel can correct Docker, the kubelet, or nodes in a timely manner. To reduce the workload of OPS personnel, NPDPlus provides self-healing capabilities based on collected node status information. Cluster admins can configure self-healing capabilities, such as restarting Docker, restarting the kubelet, or restarting CVM nodes, based on different node states. Meanwhile, to prevent node avalanche in clusters, strict throttling must be performed before self-healing to prevent massive numbers of nodes from being restarted. The specific policies are as follows:

- Only one node in the cluster can perform a self-healing action at a time, and the interval between self-healing actions must be no less than one minute.
- When a new node is added to the cluster, the node will be given a 2-minute toleration period to prevent incorrect self-healing from being triggered by the initial instability of the node addition.
- If a node remains abnormal after a CVM restart is triggered, the node will not perform any additional self-healing actions within 3 hours.

NPDPlus records all executed self-healing actions in Node Event, so that cluster admins can monitor the events that occur on nodes, as shown in the figure below:

©2013-2022 Tencent Cloud. All rights reserved.



©2013-2022 Tencent Cloud. All rights reserved.

Using kubecm to Manage Multiple Clusters kubeconfig

Last updated：2023-02-23 18:34:01

Overview

Kubectl is a command line tool provided by Kubernetes for performing operations on clusters. It uses kubeconfig as a configuration file (the default path is `~/.kube/config`) to configure the information of multiple clusters, and manage and operate multiple clusters.

To manage and operate the TKE or TKE Serverless cluster through Kubectl, you need to enable the APIServer's public or private network access on the cluster basic information page to obtain kubeconfig (cluster access credentials). If you need to use kubectl to manage multiple clusters, generally you need to extract the contents of each field in kubeconfig and merge them into the kubeconfig file of the device where kubectl locates. This method is complicated and may easily cause an error.

Through the kubecm tool, you can merge multiple cluster access credentials into kubeconfig more simply and efficiently. This document describes how to use kubecm to efficiently manage the kubeconfig of multiple clusters.

Prerequisites

- You have created a TKE general cluster or TKE Serverless cluster.
- You have installed the kubectl command line tool on the device used for managing multiple clusters.

Directions

Installing kubecm

Install kubecm on the device used for managing multiple clusters.

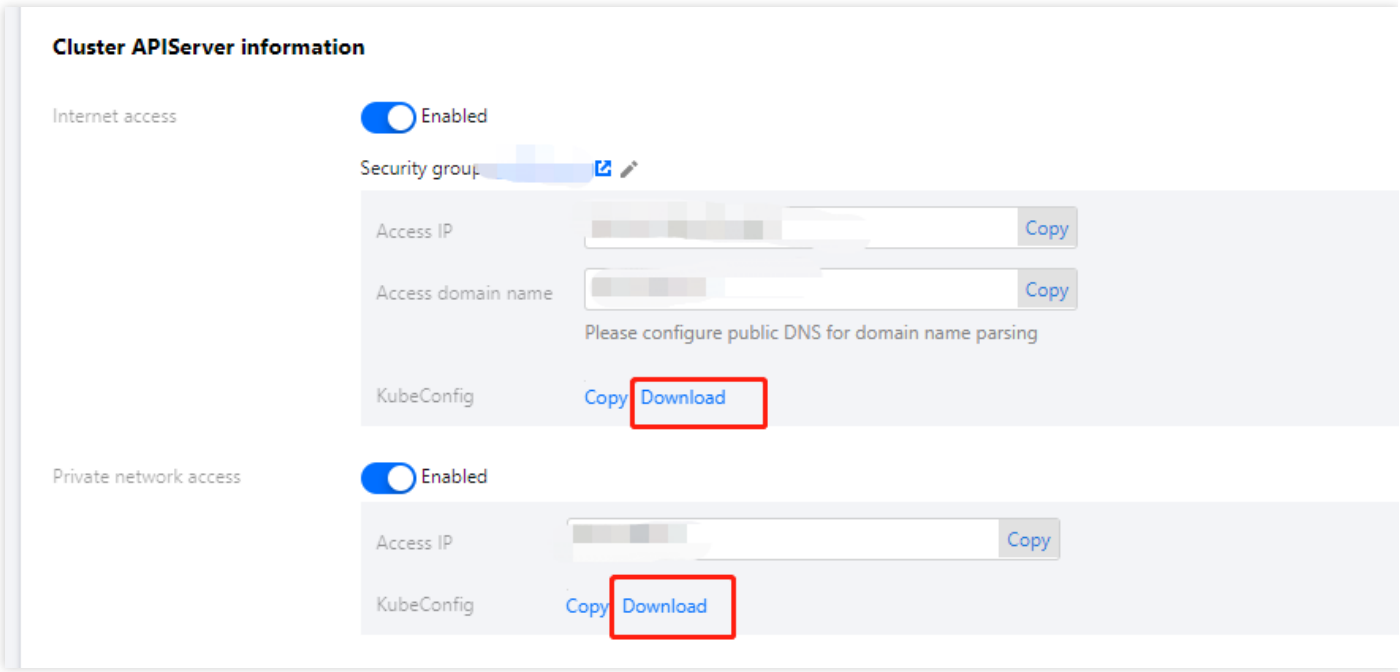
Obtaining cluster access credential

After creating a cluster, you need to follow the steps below to obtain access credential for the cluster:

1. Log in to the TKE console and click **Cluster** in the left sidebar.
2. Click the ID/name of the cluster for which the access credential needs to be obtained to go to the basic information page of the cluster.

©2013-2022 Tencent Cloud. All rights reserved.

3. On the **Basic Information** page, enable **Internet access** and **Private network access** in the **Cluster APIServer Information** section.
4. Click **Download** on the right of **KubeConfig**.



Using kubecm to add access credential to kubeconfig

This document takes the cluster access credential `cls-l6whmzi3-config` as an example. Run the following command to use kubecm to add the access credential to kubeconfig (`-n` means you can specify the context name):

```
kubecm add -f cls-l6whmzi3-config -n cd -c
```

Viewing the cluster list

Run the following `kubecm ls` command to view the cluster list in kubeconfig (the asterisk identifies the cluster under operation):

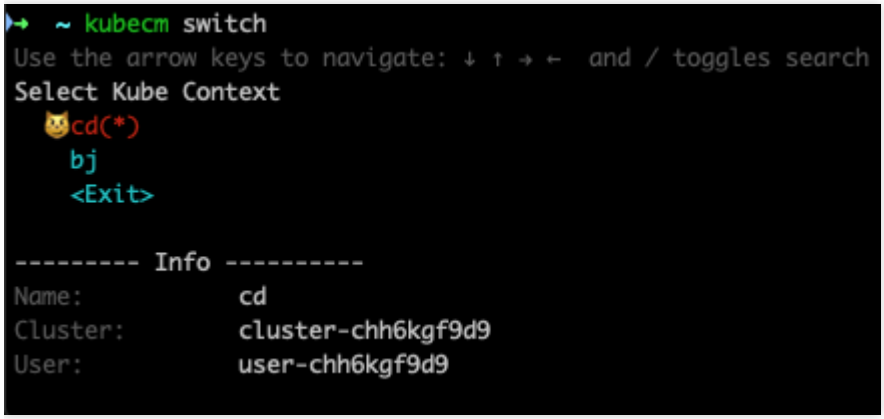
```
$ kubecm ls
+-----------+-----------+---------------------+------------------+---------
-------------------------+------------------+
| CURRENT | NAME | CLUSTER | USER | SERVER | Namespace |
+===========+===========+=====================+==================+=========
=========================+==================+
| * | cd | cluster-chh6kgf9d9 | user-chh6kgf9d9 | https://cls-l6whmzi3.ccs.tence
| default |
| | | | | nt-cloud.com | |
+-----------+-----------+---------------------+------------------+---------
```

©2013-2022 Tencent Cloud. All rights reserved.

```
-------------------------+------------------+
| | bj | cluster-6qaua96n | user-6qaua96n | https://cls-6qaua96n.ccs.tence | kube
-system |
| | | | | nt-cloud.com | |
+------------+-----------+----------------------+------------------+---------
-------------------------+------------------+
```

Switching the cluster

Run the following `kubecm switch` command to interactively switch to another cluster:



Removing the cluster

Run the following `kubecm delete` command to remove a cluster:

```
$ kubecm delete bj
Context Delete:「bj」
「/Users/roc/.kube/config」 write successful!
+------------+---------+----------------------+-------------------+-----------
----------------------+--------------+
| CURRENT | NAME | CLUSTER | USER | SERVER | Namespace |
+============+=========+======================+===================+===========
======================+==============+
| | cd | cluster-chh6kgf9d9 | user-chh6kgf9d9 | https://cls-l6whmzi3.ccs.tence |
default |
| | | | | nt-cloud.com | |
+------------+---------+----------------------+-------------------+-----------
----------------------+--------------+
```

Learn More

©2013-2022 Tencent Cloud. All rights reserved.

- Open-source kubecm
- kubecm official documents

©2013-2022 Tencent Cloud. All rights reserved.

Quick Troubleshooting Using TKE Audit and Event Services

Last updated：2023-02-23 18:34:01

Use Cases

The cluster auditing and event storage features of TKE are configured with rich visual charts to display audit logs and cluster events in multiple dimensions. Their operations are simple, and most common cluster Ops use cases are covered, making it easy for you to find and locate problems, improve the Ops efficiency, and maximize the value of audit and event data.
This document describes how to use audit and event dashboards to quickly locate cluster problems for several use cases.

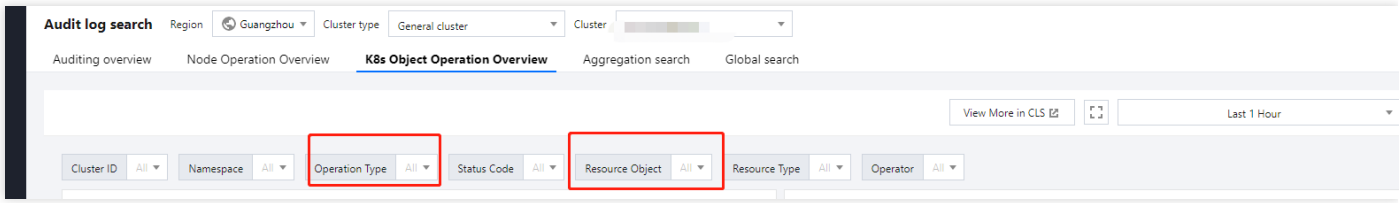
Prerequisites

You have logged in to the TKE console and enabled cluster audit and event storage.

Example

Sample 1. Troubleshooting workload disappearance

1. Log in to the TKE console.
2. Select **Log Management** > **Audit Logs** in the left sidebar to go to the **Audit log search** page.
3. Select the **K8s Object Operation Overview** tab and specify the operation type and resource object to be checked in **Filters** as shown below:



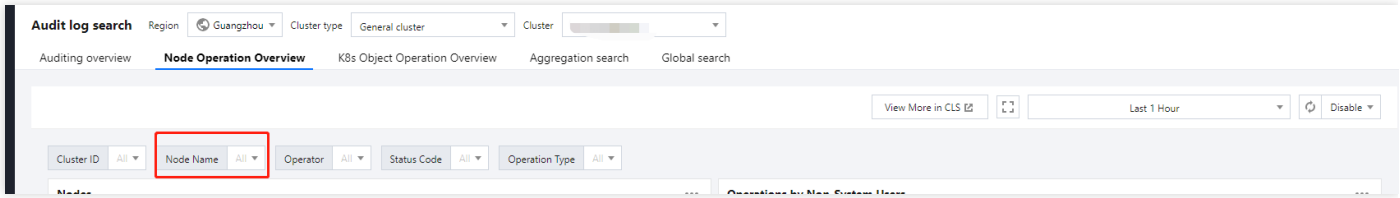
4. The query result is displayed, as shown in the figure below:

©2013-2022 Tencent Cloud. All rights reserved.

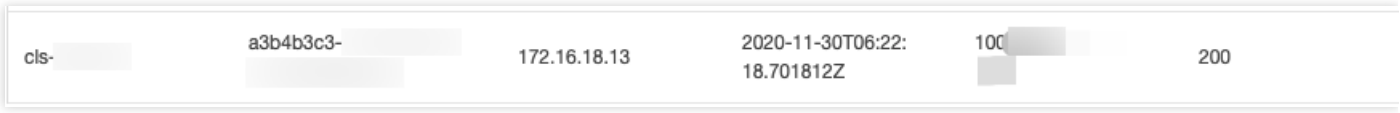
As shown above, the `10001****7138` account deleted the `nginx` application at `2020-11-30T03:37:13`. For more information on the account, select **CAM** > **User List**.

Sample 2. Troubleshooting node cordoning

1. Log in to the TKE console.
2. Select **Log Management** > **Audit Logs** in the left sidebar to go to the **Audit log search** page.
3. Select the **Node Operation Overview** tab and specify the name of the cordoned node in **Filters** as shown below:



4. Click **Filter** to start the query. The result is as shown below:



As shown in the above figure, account `10001****7138` cordoned the node `172.16.18.13` at `2020-11-30T06:22:18`.

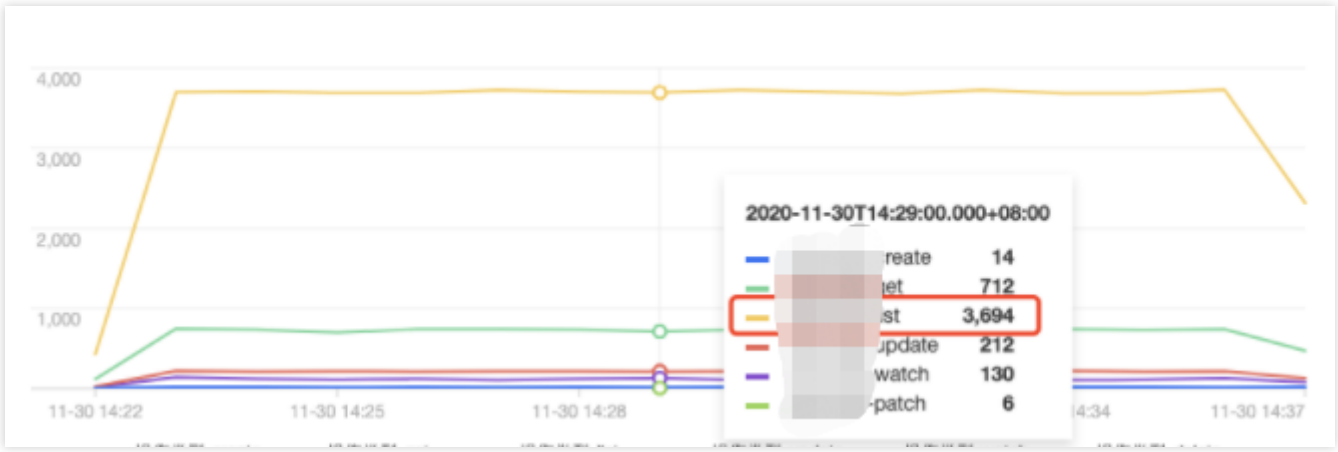
Sample 3. Troubleshooting slow API server response

1. Log in to the TKE console.
2. Select **Log Management** > **Audit Logs** in the left sidebar to go to the **Audit log search** page.
3. Select the **Aggregated Search** tab, which provides trend graphs of API server access requests in multiple dimensions, such as user, operation type, and return status code, as shown below:

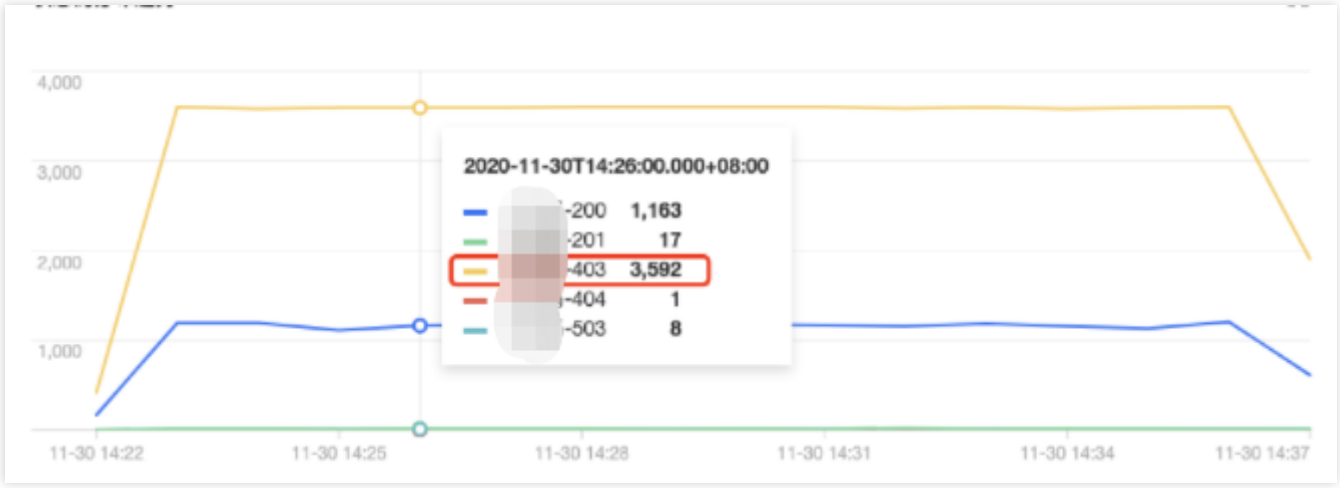
- **Operator distribution trend**:

©2013-2022 Tencent Cloud. All rights reserved.

- **Operation type distribution trend**:



- **Status code distribution trend**:



As shown above, the `tke-kube-state-metrics` user has much more access requests than others. The operation type distribution trend shows that most of the operations are LIST operations, and the status code distribution trend shows that most of the status codes are 403. The business logs show that the `tke-kube-state-metrics` add-on kept requesting API server retries due to the RBAC authentication issue, resulting in a sharp increase in API server access requests. Below is a sample log:

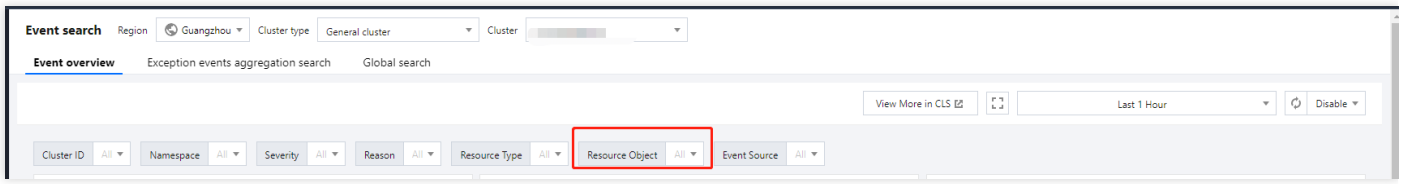
```
E1130 06:19:37.368981 1 reflector.go:156] pkg/mod/k8s.io/client-go@v0.0.0-20191109102209-3c0d1af94be5/tools/cache/reflector.go:108: Failed to list *v1.VolumeAttachment: volumeattachments.storage.k8s.io is forbidden: User "system:serviceaccount:kube-system:tke-kube-state-metrics" cannot list resource "volumeattachments" in API group "storage.k8s.io" at the cluster scope
```

Sample 4. Troubleshooting a node exception

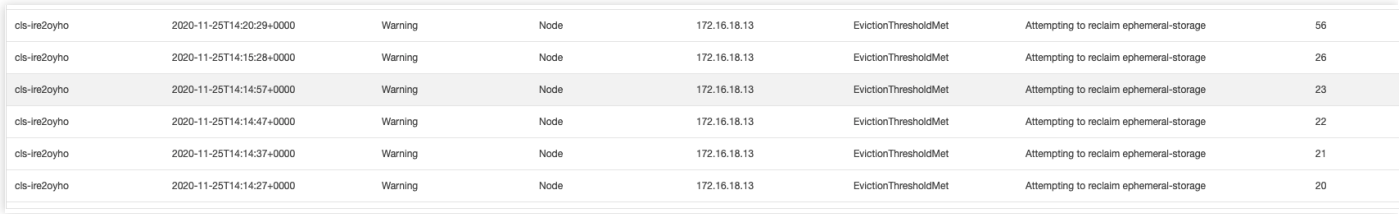
1. Log in to the TKE console.

©2013-2022 Tencent Cloud. All rights reserved.

2. Select **Log Management** > **Event Logs** in the left sidebar to go to the **Event search** page.
3. Select the **Event Overview** tab and enter the abnormal node IP in the **Resource Object** filter as shown below:



4. Click **Filter** to start the query. The results show that there is an event of **Insufficient disk space of the node**.
5. Click the event to further view the trend of the abnormal event.



cls-ire2oyho	2020-11-25T14:20:29+0000	Warning	Node	172.16.18.13	EvictionThresholdMet	Attempting to reclaim ephemeral-storage	56
cls-ire2oyho	2020-11-25T14:15:28+0000	Warning	Node	172.16.18.13	EvictionThresholdMet	Attempting to reclaim ephemeral-storage	26
cls-ire2oyho	2020-11-25T14:14:57+0000	Warning	Node	172.16.18.13	EvictionThresholdMet	Attempting to reclaim ephemeral-storage	23
cls-ire2oyho	2020-11-25T14:14:47+0000	Warning	Node	172.16.18.13	EvictionThresholdMet	Attempting to reclaim ephemeral-storage	22
cls-ire2oyho	2020-11-25T14:14:37+0000	Warning	Node	172.16.18.13	EvictionThresholdMet	Attempting to reclaim ephemeral-storage	21
cls-ire2oyho	2020-11-25T14:14:27+0000	Warning	Node	172.16.18.13	EvictionThresholdMet	Attempting to reclaim ephemeral-storage	20

As shown in the above figure, starting from 2020-11-25, the node `172.16.18.13` was exceptional due to insufficient disk space. Then kubelet began to drain pods on the node to reclaim the node's disk space.

Sample 5. Locating a node scale-out trigger

The cluster auto-scaler (CA) add-on automatically increases or decreases the number of nodes in the cluster according to the load condition when node pool **elastic scaling** is enabled. If a node in the cluster is automatically scaled, you can backtrack the whole scaling process through event search.

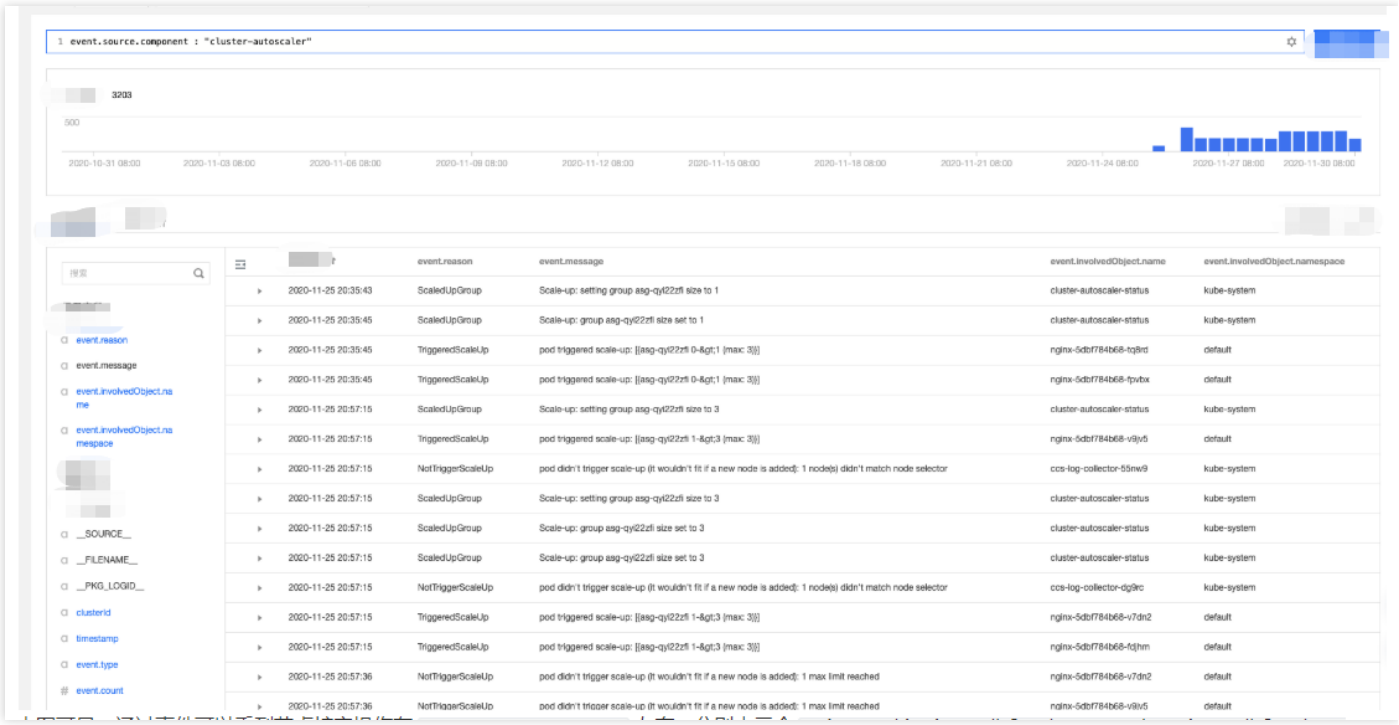
1. Log in to the TKE console.
2. Select **Log Management** > **Event Logs** in the left sidebar to go to the **Event search** page.
3. Select the **Global Search** tab and enter the following search command in the search box:

©2013-2022 Tencent Cloud. All rights reserved.

```
event.source.component : "cluster-autoscaler"
```

4. Select `event.reason` , `event.message` , and `event.involvedObject.name` from the **Hidden Fields** on the left for display. Click **Search and Analysis** and view the results.

5. Sort the search results by **Log Time** in reverse order as shown below:



According to the event flow in the above figure, you can find that the node scaling occurred around `2020-11-25 20:35:45` and was triggered by three Nginx pods (nginx-5dbf784b68-tq8rd, nginx-5dbf784b68-fpvbx, and nginx-5dbf784b68-v9jv5). After three nodes were scaled out, the subsequent scaling was not triggered because the number of nodes in the node pool reached the upper limit.

©2013-2022 Tencent Cloud. All rights reserved.

Customizing RBAC Authorization in TKE

Last updated：2022-06-10 19:32:53

TKE allows you to manage the general authorization of sub-accounts by using the **authorization management** feature in the console and customize your authorization by using a custom YAML (Using RBAC Authorization). Kubernetes RBAC authorization description and principle are as shown below:

- **Permission objects (Role or ClusterRole)**: Use apiGroups, resources, and verbs to define permissions, including:
  - Role permission object: Used for a specific namespace.
  - ClusterRole permission object: It can be reused for authorization in multiple namespaces (RoleBinding) or the entire cluster (ClusterRoleBinding).
- **Authorization object (Subjects)**: The subjects for granting permissions, including three types of subjects: User, Group, and ServiceAccount.
- **Permission binding (RoleBinding or ClusterRoleBinding)**: It combines and binds the permission objects and authorization objects, including:
  - RoleBinding: Used for a specific namespace.
  - ClusterRoleBinding: Used for the entire cluster.

Kubernetes RBAC authorization mainly provides the following four permission binding methods. This document describes how to use them for user authorization management.

Method	Description
Method 1. Bind permissions in a namespace	RoleBinding references a Role object to grant Subjects resource permissions in a namespace.
Method 2. Reuse permission objects for binding in multiple namespaces	Different RoleBinding objects in multiple namespaces can reference the same ClusterRole object template to grant Subjects the same template permissions.
Method 3. Bind permissions in the entire cluster	ClusterRoleBinding references the ClusterRole template to grant Subjects permissions for the entire cluster.
Method 4. Customize permissions	You can customize permissions, for example, grant a user the permission to log in to the TKE cluster in addition to the preset read-only permission.

Note：
In addition to the above methods, you can combine ClusterRole with other ClusterRoles by using aggregationRule on Kubernetes RBAC v1.9 or later. For more information, see Aggregated ClusterRoles.

©2013-2022 Tencent Cloud. All rights reserved.

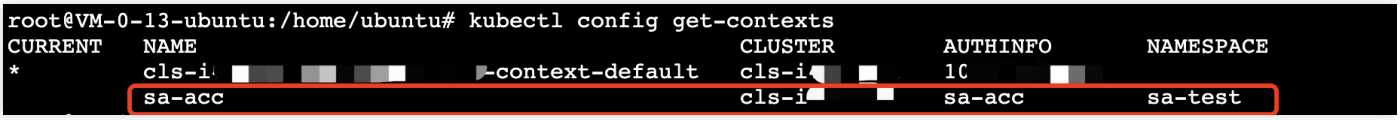
Method 1. Bind permissions in a namespace

This method is mainly used to bind related permissions under a certain namespace for a certain user. It is suitable for scenarios that require refined permissions. For example, developers, testers, and Ops personnel can only manipulate resources in their respective namespaces. The following describes how to implement permission binding for a namespace in TKE.

1. Use the following shell script to create a test namespace and a test user of ServiceAccount type, and set up cluster access credential (token) authentication as shown below:

```
USERNAME='sa-acc' # Set the test account name
NAMESPACE='sa-test' # Set the test namespace name
CLUSTER_NAME='cluster_name_xxx' # Set the test cluster name
# Create the test namespace
kubectl create namespace ${NAMESPACE}
# Create the test ServiceAccount account
kubectl create sa ${USERNAME} -n ${NAMESPACE}
# Obtain the Secret token resource name automatically created by the ServiceAcc
ount account
SECRET_TOKEN=$(kubectl get sa ${USERNAME} -n ${NAMESPACE} -o jsonpath='{.secret
s[0].name}')
# Get the plaintext token of the Secrets
SA_TOKEN=$(kubectl get secret ${SECRET_TOKEN} -o jsonpath={.data.token} -n sa-t
est | base64 -d)
# Set an access credential of token type using the obtained plaintext token inf
ormation
kubectl config set-credentials ${USERNAME} --token=${SA_TOKEN}
# Set the context entries for accessing the cluster
kubectl config set-context ${USERNAME} --cluster=${CLUSTER_NAME} --namespace=
${NAMESPACE} --user=${USERNAME}
```

2. Run the `kubectl config get-contexts` command to view the generated contexts as shown below:



3. Create a Role permission object resource file `sa-role.yaml` as shown below:

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
namespace: sa-test # Specify the namespace
name: sa-role-test
rules: # Set the permission rule
- apiGroups: ["", "extensions", "apps"]
resources: ["deployments", "replicasets", "pods"]
verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
```

4. Create a RoleBinding object resource file `sa-rb-test.yaml`. The following permission binding indicates that the `sa-acc` user of ServiceAccount type has `sa-role-test` (Role type) permissions in the `sa-test` namespace, as shown below:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
name: sa-rb-test
namespace: sa-test
subjects:
- kind: ServiceAccount
name: sa-acc
namespace: sa-test # The namespace of the ServiceAccount
apiGroup: "" # The default apiGroup is `rbac.authorization.k8s.io`.
roleRef:
kind: Role
name: sa-role-test
apiGroup: "" # The default apiGroup is `rbac.authorization.k8s.io`.
```

5. From the verification result as shown below, you can find that when the Context is `sa-context`, the default namespace is `sa-test`, and it has the permissions configured in the `sa-role-test` (Role) object under the `sa-test` namespace, but it has no permissions under the `default` namespace.

```
root@VM-0-13-ubuntu:/home/ubuntu# kubectl get pod --context=sa-acc
No resources found in sa-test namespace.
root@VM-0-13-ubuntu:/home/ubuntu# kubectl run nginx --image=nginx -n sa-test  --context=sa-acc
pod/nginx created
root@VM-0-13-ubuntu:/home/ubuntu# kubectl get pod --context=sa-acc
NAME    READY   STATUS    RESTARTS   AGE
nginx   1/1     Running   0          8s
root@VM-0-13-ubuntu:/home/ubuntu# kubectl run nginx --image=nginx -n default  --context=sa-acc
Error from server (Forbidden): pods is forbidden: User "system:serviceaccount:sa-test:sa-acc" cannot create resource "pods" in API group "" in the namespace "default"
```

©2013-2022 Tencent Cloud. All rights reserved.

Method 2. Reuse permission objects for binding in multiple namespaces

This method is mainly used to grant the same permissions in multiple namespaces to a user. It is suitable for scenarios where a permission template is used to bind permissions in multiple namespaces. For example, you might want to bind the same resource operation permissions for DevOps personnel in multiple namespaces. The following describes how to reuse cluster permissions in multiple namespaces in TKE.

1. Use the following shell script to create an user authenticated with X.509 self-signed certificate, approve the CSR and the certificate as trustworthy, and set the cluster resource access credential Context as shown below:

```
USERNAME='role_user' # Set the username
NAMESPACE='default' # Set the test namespace name
CLUSTER_NAME='cluster_name_xxx' # Set the test cluster name
# Use OpenSSL to generate a self-signed certificate key
openssl genrsa -out ${USERNAME}.key 2048
# Use OpenSSL to generate a self-signed CSR file, where `CN` indicates the user
name and `O` indicates the group name
openssl req -new -key ${USERNAME}.key -out ${USERNAME}.csr -subj "/CN=${USERNAM
E}/O=${USERNAME}"
# Create a Kubernetes CSR
cat <<EOF | kubectl apply -f -
apiVersion: certificates.k8s.io/v1beta1
kind: CertificateSigningRequest
metadata:
name: ${USERNAME}
spec:
request: $(cat ${USERNAME}.csr | base64 | tr -d '\n')
usages:
- digital signature
- key encipherment
- client auth
EOF
# Approve the certificate as trustworthy
kubectl certificate approve ${USERNAME}
# Obtain the self-signed certificate CRT
kubectl get csr ${USERNAME} -o jsonpath={.status.certificate} | base64 --decode
> ${USERNAME}.crt
# Set the cluster resource access credential (X.509 certificate)
kubectl config set-credentials ${USERNAME} --client-certificate=${USERNAME}.crt
--client-key=${USERNAME}.key
# Set the Context cluster and default namespace
```

©2013-2022 Tencent Cloud. All rights reserved.

```
kubectl config set-context ${USERNAME} --cluster=${CLUSTER_NAME} --namespace=
${NAMESPACE} --user=${USERNAME}
```

2. Create a ClusterRole object resource file `test-clusterrole.yaml` as shown below:

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
name: test-clusterrole
rules:
- apiGroups: [""]
resources: ["pods"]
verbs: ["get", "watch", "list", "create"]
```

3. Create a RoleBinding object resource file `clusterrole-rb-test.yaml` . The following permission binding indicates that the `role_user` user with the self-signed certificate authentication has `test-clusterrole` (ClusterRole type) permissions in the `default` namespace, as shown below:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
name: clusterrole-rb-test
namespace: default
subjects:
- kind: User
name: role_user
namespace: default # The namespace of the user
apiGroup: "" # The default apiGroup is `rbac.authorization.k8s.io`.
roleRef:
kind: ClusterRole
name: test-clusterrole
apiGroup: "" # The default apiGroup is `rbac.authorization.k8s.io`.
```

4. From the verification result as shown below, you can find that when the Context is `role_user` , the default namespace is `default` , and it has the permissions configured by the `test-clusterrole` permission

©2013-2022 Tencent Cloud. All rights reserved.

object.

```
root@VM-0-13-ubuntu:/home/ubuntu# kubectl  get pod --context=role_user
No resources found in default namespace.
root@VM-0-13-ubuntu:/home/ubuntu# kubectl run nginx --image=nginx --context=role_user
pod/nginx created
root@VM-0-13-ubuntu:/home/ubuntu# kubectl  get pod --context=role_user
NAME    READY   STATUS    RESTARTS   AGE
nginx   1/1     Running   0          4s
root@VM-0-13-ubuntu:/home/ubuntu# kubectl  delete pod nginx --context=role_user
Error from server (Forbidden): pods "nginx" is forbidden: User "role_user" cannot delete resource "pods" in API group "" in the namespace "default"
```

5. Create the second RoleBinding object resource file `clusterrole-rb-test2.yaml` . The following permission binding indicates that the `role_user` user with the self-signed certificate authentication has `test-clusterrole` (ClusterRole type) permissions in the `default2` namespace.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
name: clusterrole-rb-test
namespace: default2
subjects:
- kind: User
name: role_user
namespace: default # The namespace of the user
apiGroup: "" # The default apiGroup is `rbac.authorization.k8s.io`.
roleRef:
kind: ClusterRole
name: test-clusterrole
apiGroup: "" # The default apiGroup is `rbac.authorization.k8s.io`.
```

6. From the verification result as shown below, you can find that in the `default2` namespace, `role_user` also has the permissions configured by `test-clusterrole` . At this point, you have implemented permission reuse and binding in multiple namespaces.

```
root@VM-0-13-ubuntu:/home/ubuntu# kubectl  create namespace default2
namespace/default2 created
root@VM-0-13-ubuntu:/home/ubuntu# kubectl  get pod -n default2 --context=role_user
Error from server (Forbidden): pods is forbidden: User "role_user" cannot list resource "pods" in API group "" in the namespace "default2"
root@VM-0-13-ubuntu:/home/ubuntu# kubectl apply -f clusterrole-rb-test2.yaml
rolebinding.rbac.authorization.k8s.io/clusterrole-rb-test created
root@VM-0-13-ubuntu:/home/ubuntu# kubectl  get pod -n default2 --context=role_user
No resources found in default2 namespace.
root@VM-0-13-ubuntu:/home/ubuntu# kubectl run nginx --image=nginx -n default2 --context=role_user
pod/nginx created
root@VM-0-13-ubuntu:/home/ubuntu# kubectl  get pod -n default2 --context=role_user
NAME    READY   STATUS    RESTARTS   AGE
nginx   1/1     Running   0          7s
root@VM-0-13-ubuntu:/home/ubuntu# kubectl  delete pod nginx -n default2 --context=role_user
Error from server (Forbidden): pods "nginx" is forbidden: User "role_user" cannot delete resource "pods" in API group "" in the namespace "default2"
```

Method 3. Bind permissions in the entire cluster

©2013-2022 Tencent Cloud. All rights reserved.

This method is mainly used to bind permissions of all namespaces for a user. It is suitable for cluster-wide authorization, such as log collection permission and admin permission. The following directions describe how to use multiple namespaces in TKE to reuse cluster permission for authorization binding.

1. Create a ClusterRoleBinding object resource file `clusterrole-crb-test3.yaml`. The following permission binding indicates that the `role_user` user with the certificate authentication has `test-clusterrole` (ClusterRole type) permissions in the entire cluster.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
name: clusterrole-crb-test
subjects:
- kind: User
name: role_user
namespace: default # The namespace of the user
apiGroup: "" # The default apiGroup is `rbac.authorization.k8s.io`.
roleRef:
kind: ClusterRole
name: test-clusterrole
apiGroup: "" # The default apiGroup is `rbac.authorization.k8s.io`.
```

2. From the verification result as shown below, you can find that after the YAML of permission binding is applied, `role_user` has the cluster-wide `test-clusterrole` permissions.

```
root@VM-0-13-ubuntu:/home/ubuntu# kubectl  apply -f clusterrole-crb-test.yaml
clusterrolebinding.rbac.authorization.k8s.io/clusterrole-crb-test created
root@VM-0-13-ubuntu:/home/ubuntu# kubectl  create namespace default3
namespace/default3 created
root@VM-0-13-ubuntu:/home/ubuntu# kubectl  create namespace default4
namespace/default4 created
root@VM-0-13-ubuntu:/home/ubuntu# kubectl  run nginx --image=nginx -n default3 --context=role_user
pod/nginx created
root@VM-0-13-ubuntu:/home/ubuntu# kubectl  run nginx --image=nginx -n default4 --context=role_user
pod/nginx created
root@VM-0-13-ubuntu:/home/ubuntu# kubectl  get pod  -n default3 --context=role_user
NAME    READY   STATUS    RESTARTS   AGE
nginx   1/1     Running   0          33s
root@VM-0-13-ubuntu:/home/ubuntu# kubectl  get pod  -n default4 --context=role_user
NAME    READY   STATUS    RESTARTS   AGE
nginx   1/1     Running   0          32s
```

Method 4. Customize permissions

This section describes how to grant a user custom permissions as a cluster admin, including preset read-only permission and additional permission to log in to the TKE cluster.

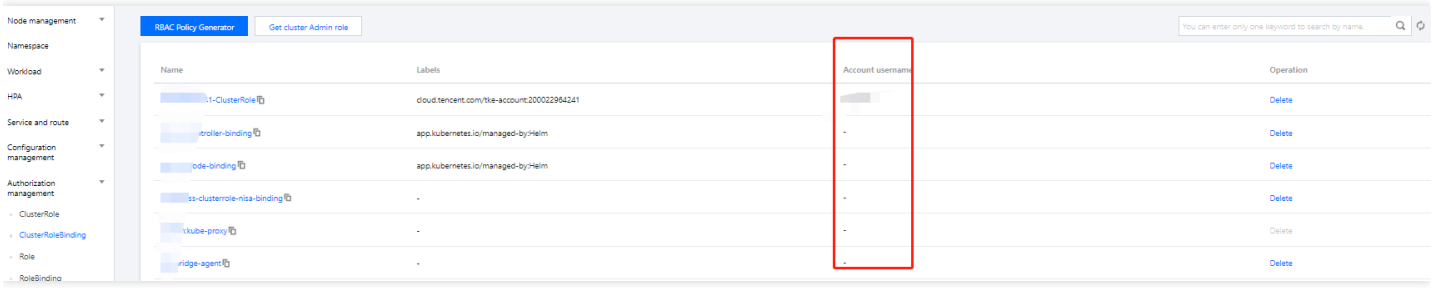
1. Authorize

First, grant read-only permission to a specified user as instructed in Using Preset Identity Authorization.

©2013-2022 Tencent Cloud. All rights reserved.

2. View user information in the RBAC

View the information of the user bound to the read-only ClusterRoleBinding, which is to be bounded to the new ClusterRoleBinding. As shown below, you need to view the details in the ClusterRoleBinding object of the specified user.




```
subjects:
- apiGroup: rbac.authorization.k8s.io
kind: User
name: 700000xxxxxx-1650879262 # The username of the specified user in RBAC. You n
eed to get this information of the specified user.
```

3. Create a ClusterRole

Create a ClusterRole through YAML for a read-only user with TKE login permission as shown below:

```
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRole
metadata:
name: "700000xxxxxx-ClusterRole-ro" # ClusterRole name
rules:
- apiGroups:
- ""
resources:
- pods
- pods/attach
- pods/exec # Pod login permission
- pods/portforward
- pods/proxy
verbs:
- create
- get
- list
- watch
```

4. Create a ClusterRoleBinding

©2013-2022 Tencent Cloud. All rights reserved.

Create the YAML file of the ClusterRoleBinding for the specified user as shown below:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
name: "700000xxxxxx-ClusterRoleBinding-ro"
roleRef:
apiGroup: rbac.authorization.k8s.io
kind: ClusterRole
name: "700000xxxxxx-ClusterRole-ro" # Use the ClusterRole name in step 3
subjects:
- apiGroup: rbac.authorization.k8s.io
kind: User
name: "700000xxxxxx-1650879262" # Use the user information in step 2
```

Summary

Combined with Tencent Cloud access permission management and Kubernetes RBAC authorization mode, the authorization management feature in the TKE console becomes simple and convenient, which can meet the permission management scenarios of most Tencent Cloud sub-accounts. The custom permission binding through YAML is more flexible and suitable for complex and personalized user permission control. You can choose a permission management method as needed.

©2013-2022 Tencent Cloud. All rights reserved.

Clearing De-registered Tencent Cloud Account Resources

Last updated：2022-08-26 17:44:49

Use Cases

If Tencent Cloud accounts in your organization are de-registered due to employee resignation or transfer, you can use TKE to **quickly clear** the accounts or have them **automatically cleared**. This document describes how to use the TKE console to clear the RBAC resource objects of Tencent Cloud accounts that have been de-registered.

Principle

RBAC controls user access to clusters. For more information, see Overview.

Directions

Viewing de-registered Tencent Cloud accounts

You can view de-registered Tencent Cloud accounts in your cluster in the following steps:

1. Log in to the TKE console and select **Cluster** on the left sidebar.
2. On the cluster management page, select the target region.
3. In the cluster list, click a cluster ID to enter the cluster details page.
4. Select **Authorization Management** > **ClusterRoleBinding** or **RoleBinding**. Under the **Account Username** in the list, a de-registered Tencent Cloud account will be red. Hover over it, and you'll be prompted to clear relevant resource objects.

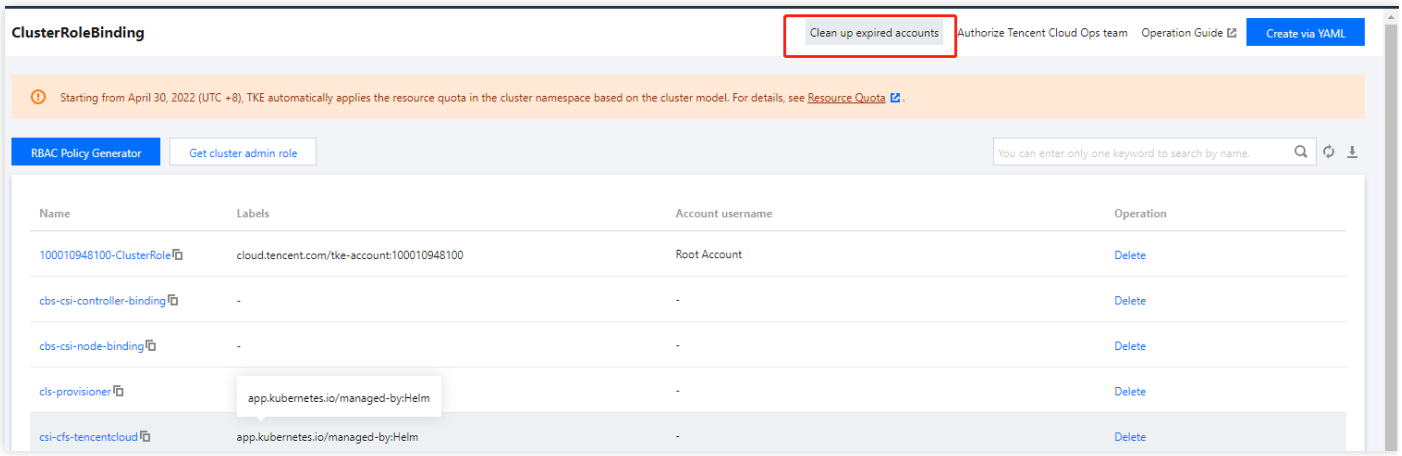
Clearing invalid accounts

You can quickly clear the RBAC resource objects of the de-registered Tencent Cloud accounts **manually** or **automatically** in the following steps:

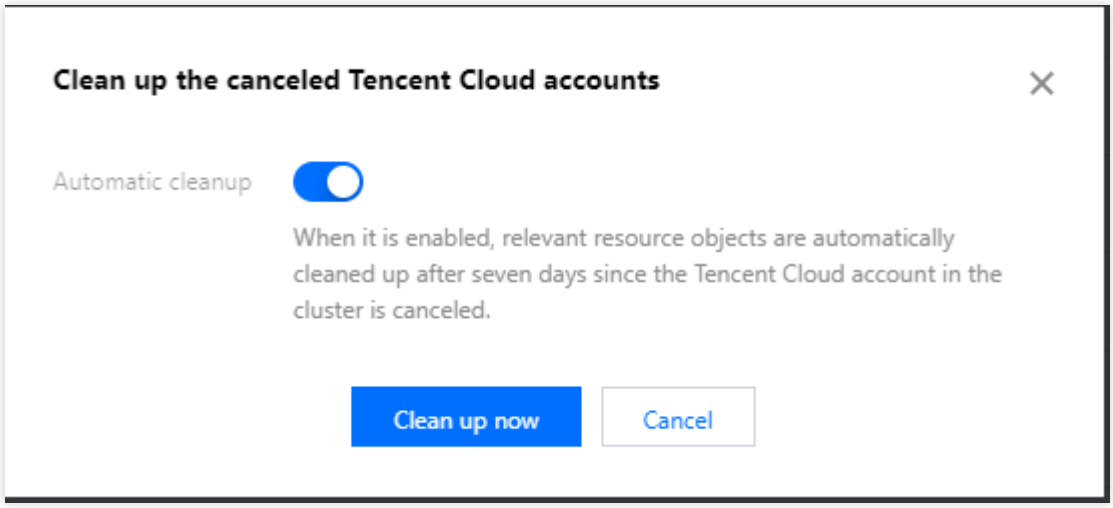
1. Log in to the TKE console and select **Cluster** on the left sidebar.
2. On the cluster management page, select the target region.
3. In the cluster list, click a cluster ID to enter the cluster details page.

©2013-2022 Tencent Cloud. All rights reserved.

4. Select **Authorization Management** > **ClusterRoleBinding** or **RoleBinding**. On the **ClusterRoleBinding** or **RoleBinding** page, click **Clear invalid account** in the top-right corner.



5. In the **Clear De-registered Tencent Cloud Account** pop-up window, click **Clear now** to clear those that haven't been cleared.
   You can also enable **automatic clearing** to have de-registered accounts cleared in an automatic and scheduled manner.



©2013-2022 Tencent Cloud. All rights reserved.

Terraform
Managing TKE Clusters and Node Pools with Terraform

Last updated：2023-09-05 09:40:15

Installing Terraform

Go to the Terraform official website and use the command line to install Terraform directly or download the binary installation package file.

Verification and Authentication

Obtaining credentials

Before using Terraform for the first time, go to the TencentCloud API Key page to apply for `SecretId` and `SecretKey`. If you already have them, skip this step.

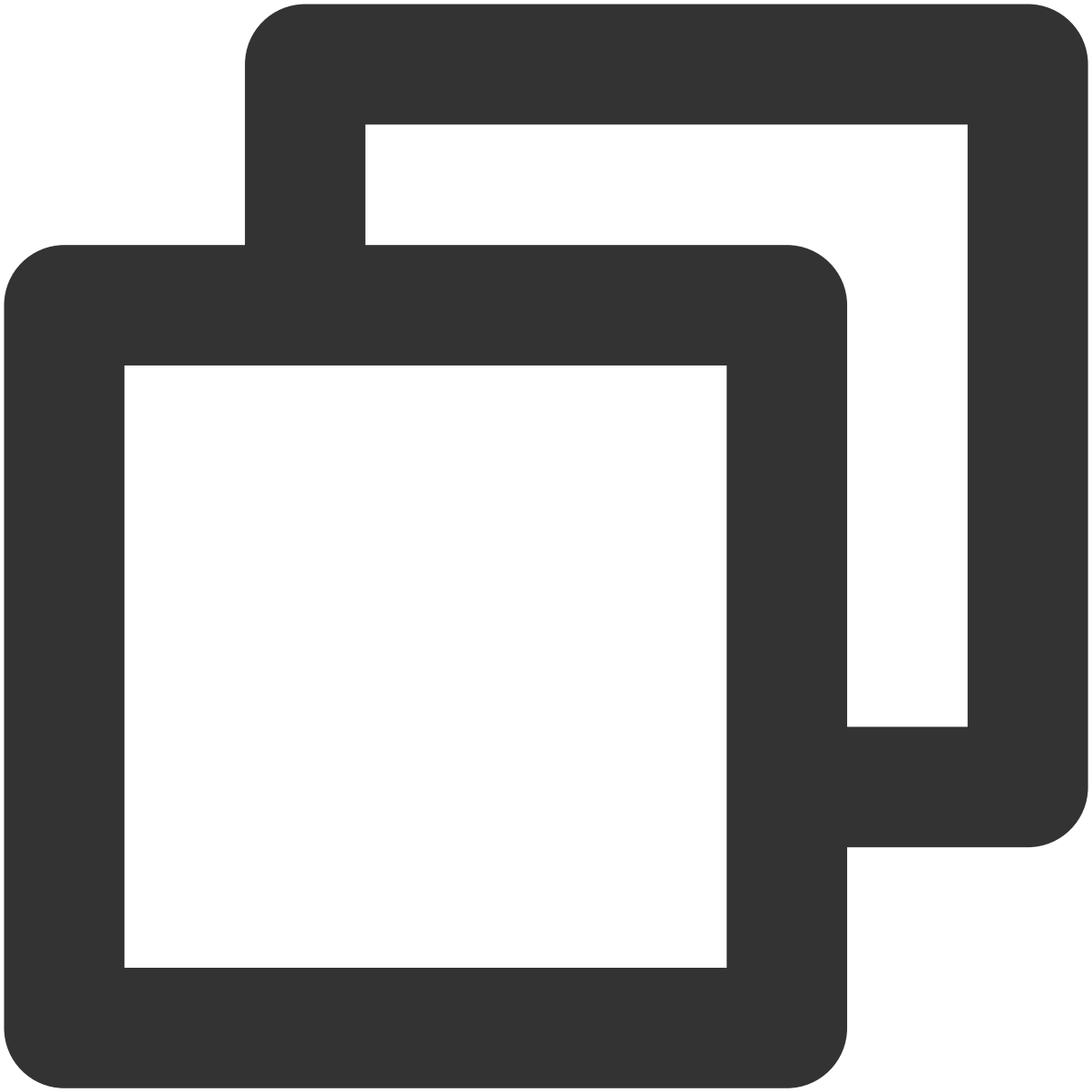
1. Log in to the CAM console and select **Access Key** > **Manage API Key** in the left sidebar.
2. On the **Manage API Key** page, click **Create Key** to create a pair of `SecretId/SecretKey`.

Authentication

Method 1: (Recommended) Inject access key for the account with environment variables

Add the following content to the environment variables:

©2013-2022 Tencent Cloud. All rights reserved.



```
export TENCENTCLOUD_SECRET_ID="xxx"                    # Replace it with the `Secr
export TENCENTCLOUD_SECRET_KEY="xxx"            # Replace it with the `SecretKey` o
```

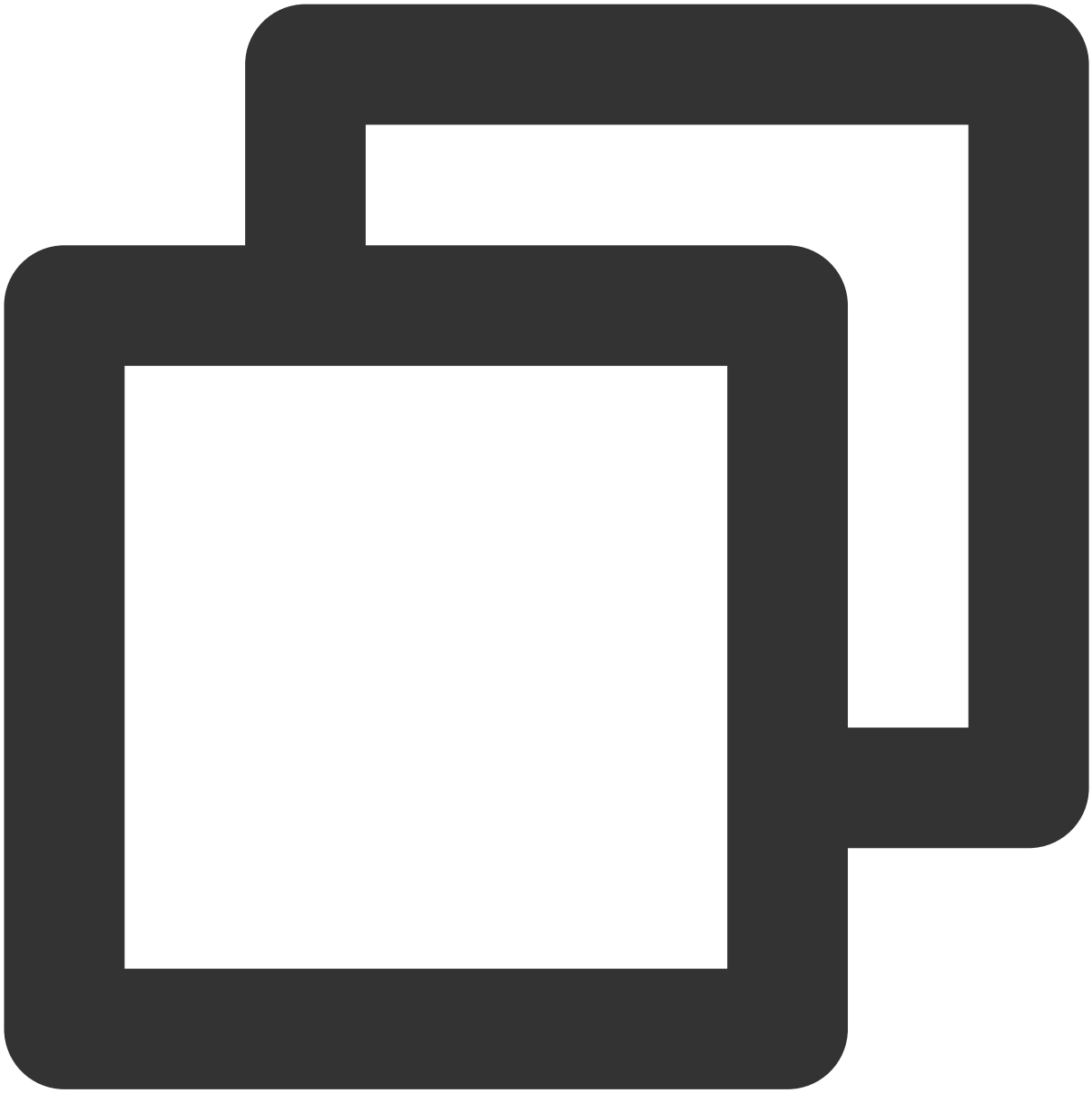
Method 2: Enter the access key for the account in the `provider` block of the Terraform configuration file

Create a `provider.tf` file under the user directory and enter the following content:

Note

Please ensure the security of the access key in the configuration file.

©2013-2022 Tencent Cloud. All rights reserved.



```
provider "tencentcloud" {
  secret_id = "xxx"
  secret_key = "xxx"
}
```

Creating a TKE Cluster with Terraform

1. Create a working directory. Then create a Terraform configuration file named `main.tf` under it.

©2013-2022 Tencent Cloud. All rights reserved.

Notes

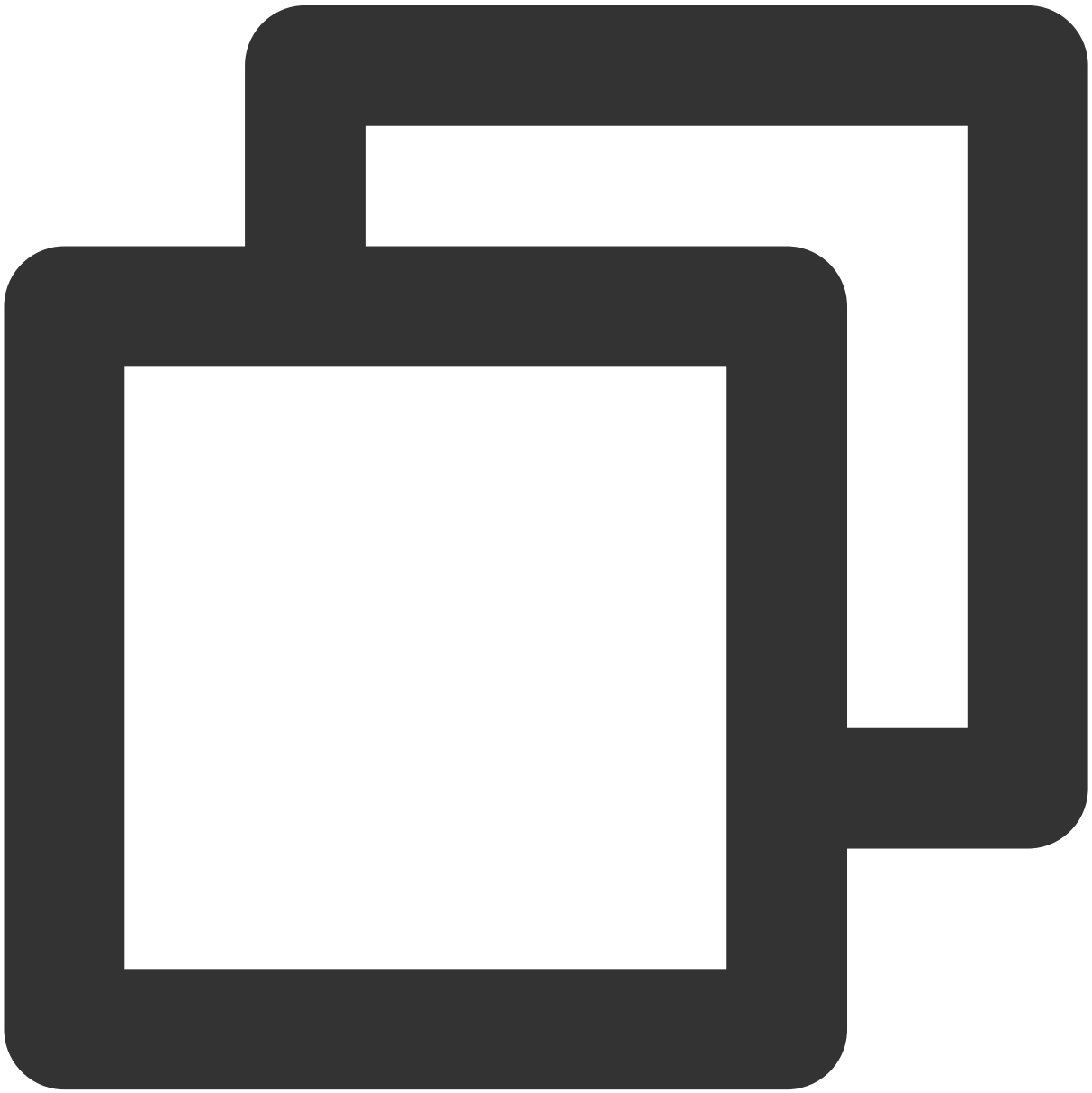
The `main.tf` file describes the following Terraform configurations:

Create a VPC, and create a subnet in the VPC.

Create a managed TKE cluster.

Create a node pool in the cluster.

The content of the `main.tf` file is as follows:



```
# Identify the use of Tencent Cloud Terraform Provider
terraform {
  required_providers {
```

©2013-2022 Tencent Cloud. All rights reserved.

```
    tencentcloud = {
      source = "tencentcloudstack/tencentcloud"
    }
  }
}

# Define local variables and modify the values as needed when using them in subs
locals {
    region = "xxx"                                                          # R
    zone1 = "xxx"
    vpc_name = "xxx"                                                      # Set the V
    vpc_cidr_block = "xxx"                              # CIDR block of the VPC, su
    subnet1_name = "xxx"                                          # Name of subnet 1,
    subnet1_cidr_block = "xxx"                 # CIDR block of subnet 1, such as `
    cluster_name = "xxx"                                          # TKE cluster name,
    network_type = "xxx"                                          # Network mode of t
    cluster_cidr = "xxx"                                          # Container network
    cluster_version = "xxx"                             # Kubernetes version of the
}


# Basic configuration of the Tencent Cloud `provider`
provider "tencentcloud" {
    # Enter the `SecretId` and `SecretKey` if you use the configuration file. It
    # secret_id  = "xxx"
    # secret_key = "xxx"
  region = local.region
}

# Declare VPC resources
resource "tencentcloud_vpc" "vpc_example" {
  name = local.vpc_name
  cidr_block    = local.vpc_cidr_block
}

# Declare subnet resources
resource "tencentcloud_subnet" "subnet_example" {
  availability_zone = local.zone1
  cidr_block        = local.subnet1_cidr_block
  name              = local.subnet1_name
  vpc_id            = tencentcloud_vpc.vpc_example.id                         # T
}

# Declare TKE cluster resources and create a cluster with the network set as Glo
resource "tencentcloud_kubernetes_cluster" "managed_cluster_example" {
  vpc_id = tencentcloud_vpc.vpc_example.id
  cluster_name = local.cluster_name
```

©2013-2022 Tencent Cloud. All rights reserved.

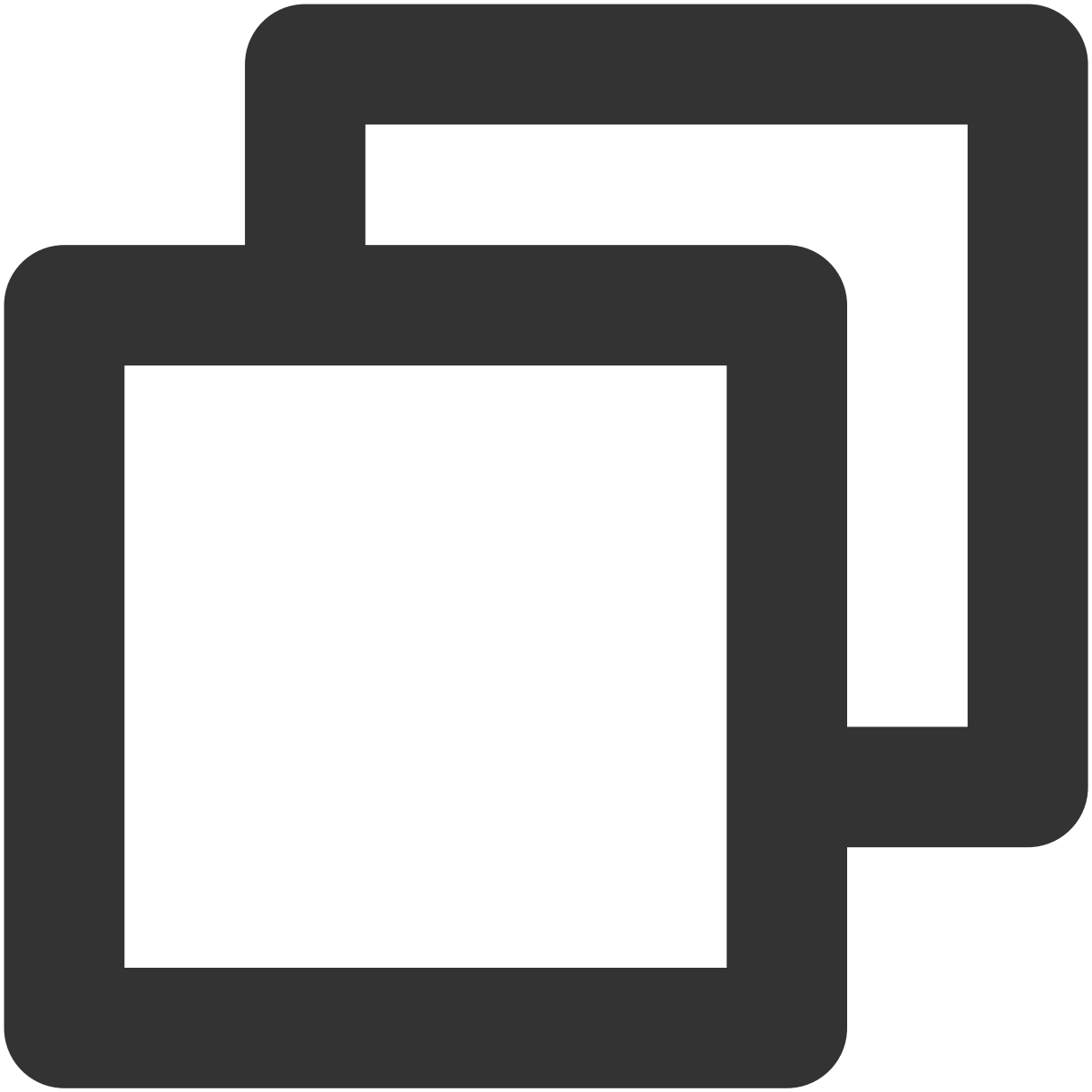
```
  network_type = local.network_type
  cluster_cidr = local.cluster_cidr
  cluster_version = local.cluster_version
}


# You can use the following declaration to create a cluster in VPC-CNI mode.
# resource "tencentcloud_kubernetes_cluster" "managed_cluster_example" {
#   vpc_id = tencentcloud_vpc.vpc_example.id
#   cluster_name = local.cluster_name
#   network_type = "VPC-CNI"
#   eni_subnet_ids = [tencentcloud_subnet.subnet_example.id]
#   service_cidr = "172.16.0.0/24"
#   cluster_version = local.cluster_version
# }
```

2. (Optional) **If you use Tencent Cloud TKE for the first time, please grant TKE permissions to access other cloud service resources.** If you have granted permissions, skip this step.

When you log in to the TKE console for the first time, you need to grant TKE permissions to access CVMs, CLBs, CBS, and other cloud resources. For more information, see Description of Role Permissions Related to Service Authorization.

You can also grant permissions in the Terraform configuration file. To do this, please create a `cam.tf` file with the following content under the working directory.

©2013-2022 Tencent Cloud. All rights reserved.



```
############################ Please add declaration configuration in the Terraform c

# Create the preset role `TKE_QCSRole` for the service
resource "tencentcloud_cam_role" "TKE_QCSRole" {
 name        = "TKE_QCSRole"
 document    = <<EOF
{
 "statement": [
{
  "action":"name/sts:AssumeRole",
  "effect":"allow",
```

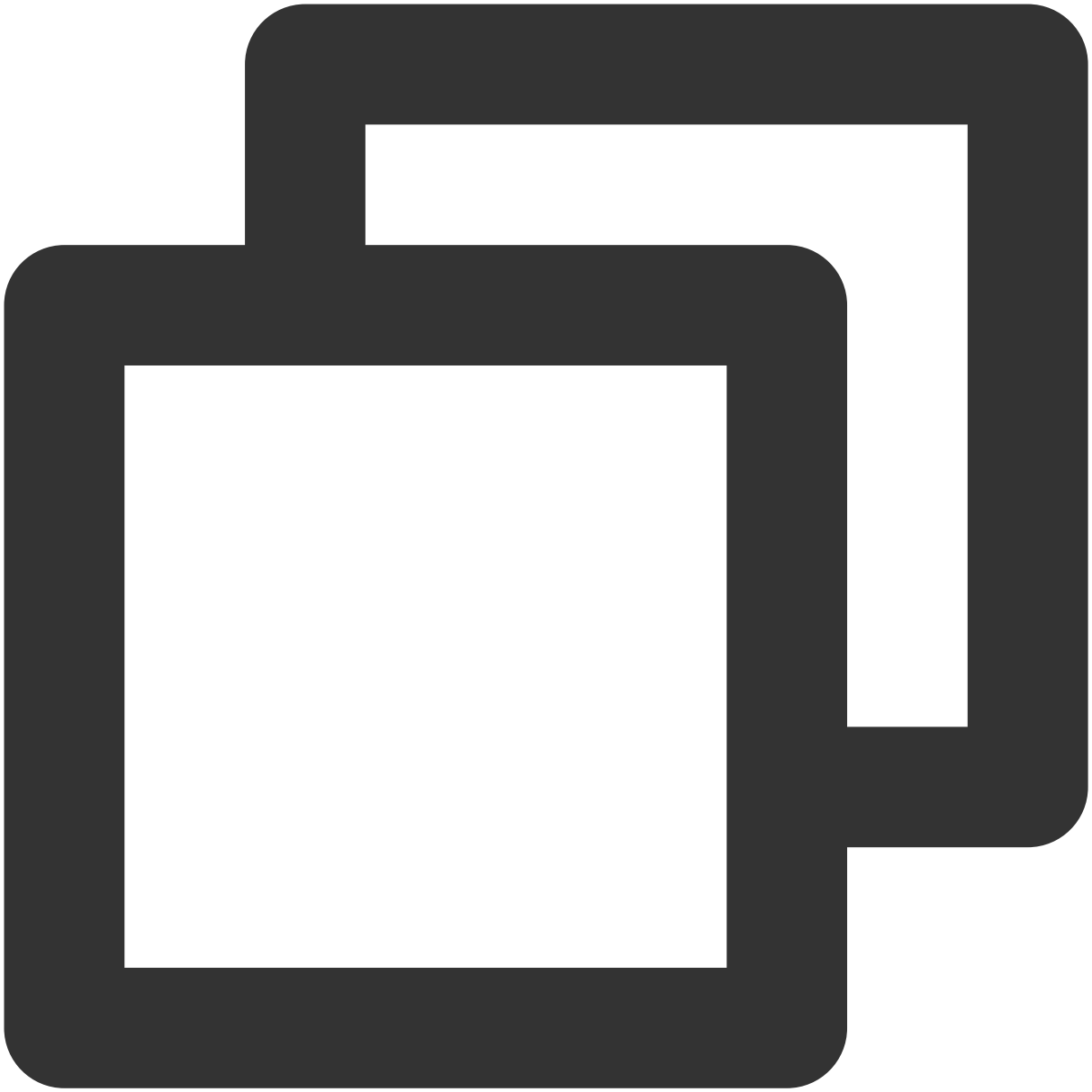
©2013-2022 Tencent Cloud. All rights reserved.

```
  "principal":{
    "service":"ccs.qcloud.com"
  }
}
 ],
 "version":"2.0"
}
EOF
 description = "The current role is the Tencent Cloud TKE service role, and it will
}

# Preset policy `QcloudAccessForTKERole`
data "tencentcloud_cam_policies" "qca" {
 name = "QcloudAccessForTKERole"
}

# Preset policy `QcloudAccessForTKERoleInOpsManagement`
data "tencentcloud_cam_policies" "ops_mgr" {
 name = "QcloudAccessForTKERoleInOpsManagement"
}

# Associate the policy `QcloudAccessForTKERole` with the role `TKE_QCSRole`
resource "tencentcloud_cam_role_policy_attachment" "QCS_QCA" {
 role_id   = lookup(tencentcloud_cam_role.TKE_QCSRole, "id")
 policy_id = data.tencentcloud_cam_policies.qca.policy_list.0.policy_id
}

# Associate the policy `QcloudAccessForTKERoleInOpsManagement` with the role `TKE_Q
resource "tencentcloud_cam_role_policy_attachment" "QCS_OpsMgr" {
 role_id   = lookup(tencentcloud_cam_role.TKE_QCSRole, "id")
 policy_id = data.tencentcloud_cam_policies.ops_mgr.policy_list.0.policy_id
}

############################ Create the role `TKE_QCSRole` and grant permissions to
############################ Create the role `IPAMDofTKE_QCSRole` and grant permissi


# Create the preset role `IPAMDofTKE_QCSRole` for the service
resource "tencentcloud_cam_role" "IPAMDofTKE_QCSRole" {
 name = "IPAMDofTKE_QCSRole"
 document    = <<EOF
{
 "statement": [
{
  "action":"name/sts:AssumeRole",
  "effect":"allow",
  "principal":{
```

```
    "service":"ccs.qcloud.com"
  }
}
 ],
 "version":"2.0"
}
EOF
 description = "The current role is the IPAMD service role, and it will access your
}

# Preset policy `QcloudAccessForIPAMDofTKERole`
data "tencentcloud_cam_policies" "qcs_ipamd" {
 name = "QcloudAccessForIPAMDofTKERole"
}

# Associate the policy `QcloudAccessForIPAMDofTKERole` with the role `IPAMDofTKE_QC
resource "tencentcloud_cam_role_policy_attachment" "QCS_Ipamd" {
 role_id   = lookup(tencentcloud_cam_role.IPAMDofTKE_QCSRole, "id")
 policy_id = data.tencentcloud_cam_policies.qcs_ipamd.policy_list.0.policy_id
}
############################ Create the role `IPAMDofTKE_QCSRole` and grant permissi
############################ Create the role `TKE_QCSLinkedRoleInEKSLog` and grant p
# To enable log collection for super nodes, create the preset role `TKE_QCSLinkedRo
resource "tencentcloud_cam_service_linked_role" "service_linked_role" {
  qcs_service_name = ["cvm.qcloud.com", "ekslog.tke.cloud.tencent.com"]
  description      = "tke log role created by terraform"
  tags = {
    "createdBy" = "terraform"
  }
}
```

3. Run the following command to initialize the environment for Terraform.

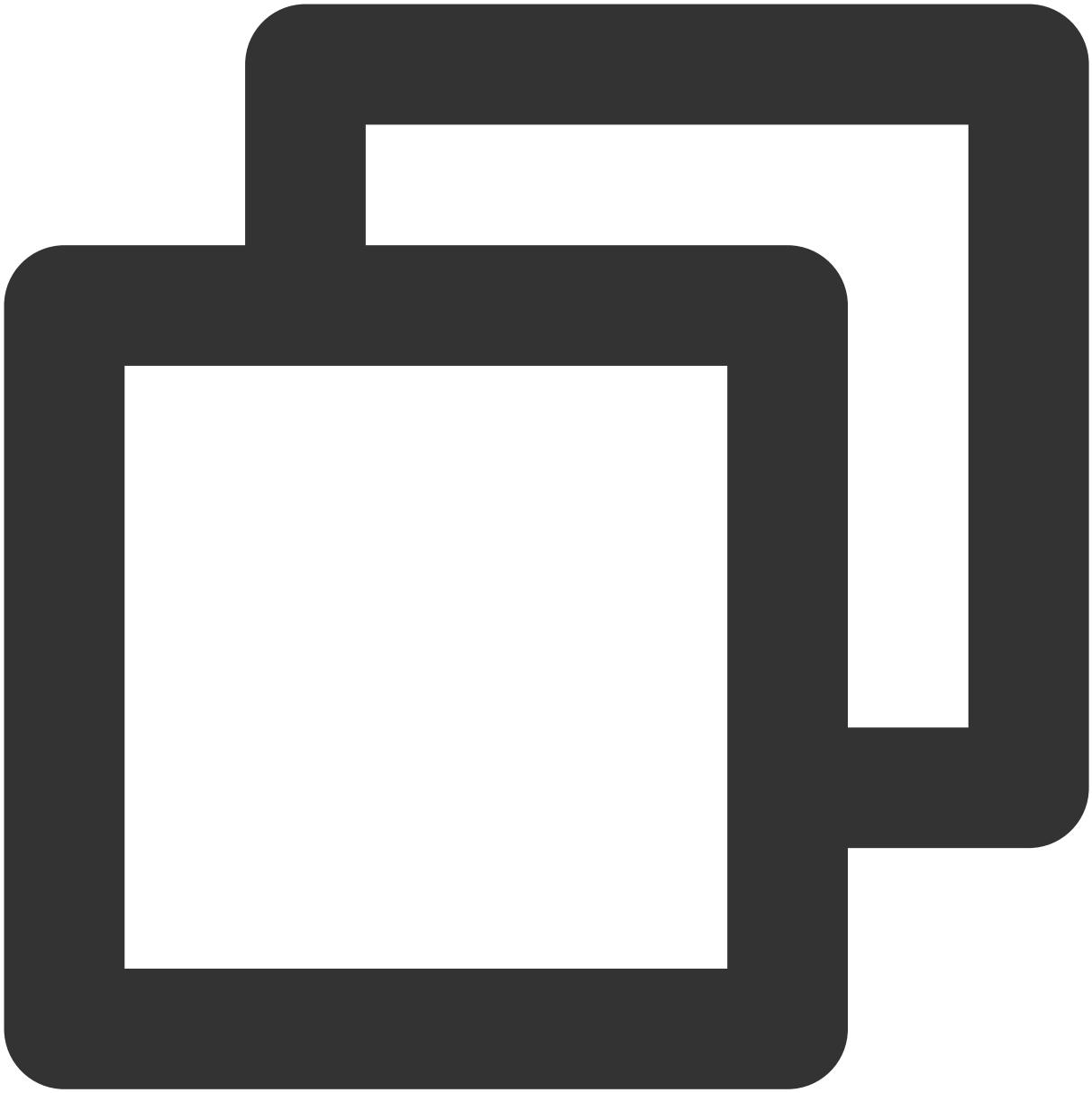
©2013-2022 Tencent Cloud. All rights reserved.



```
terraform init
```

The returned information is as follows:

©2013-2022 Tencent Cloud. All rights reserved.

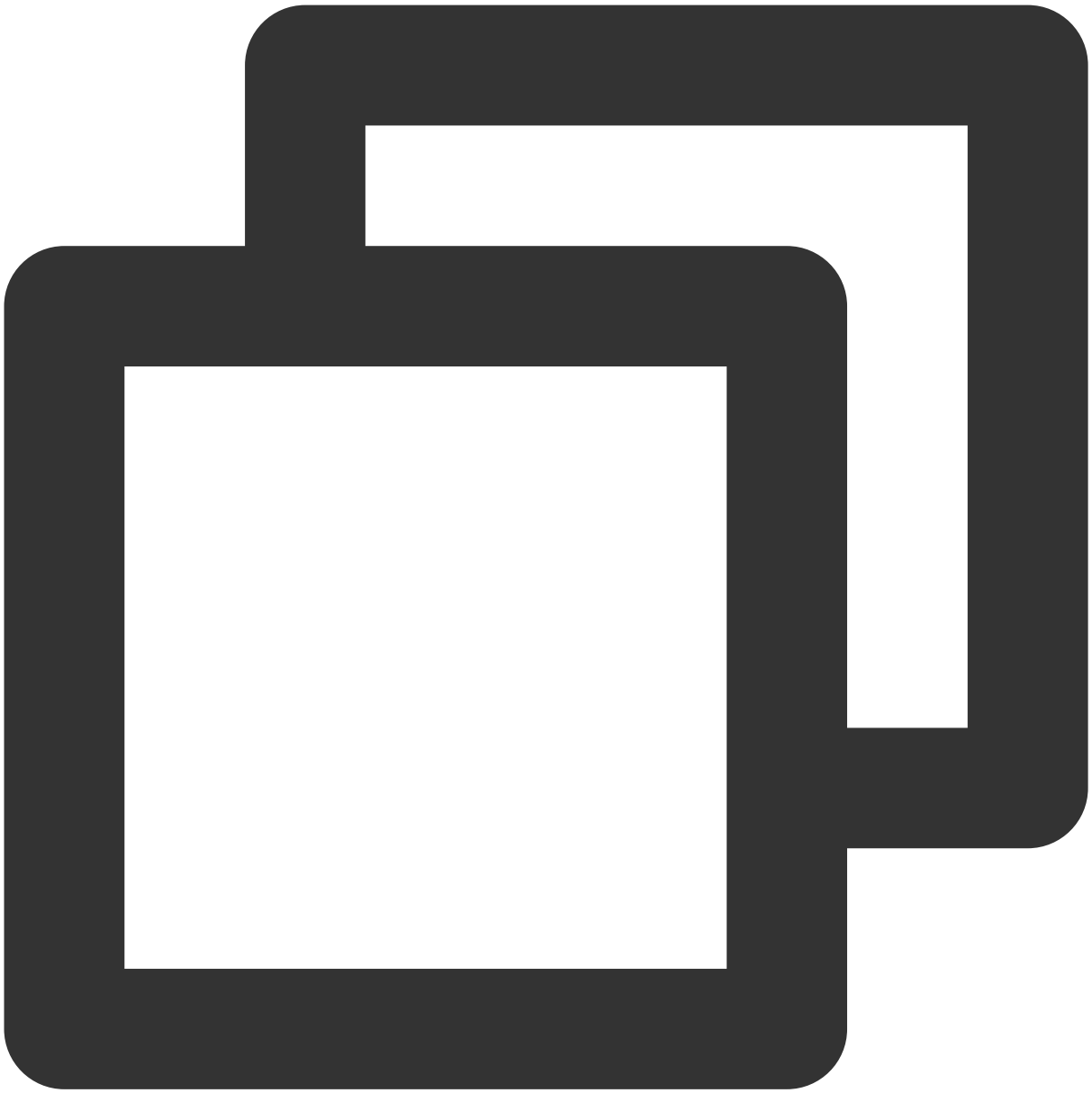


```
Initializing the backend...

Initializing provider plugins...
- Finding tencentcloudstack/tencentcloud versions matching "~> 1.78.13"...
- Installing tencentcloudstack/tencentcloud v1.78.13...
...

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.
```

©2013-2022 Tencent Cloud. All rights reserved.

```
...
```

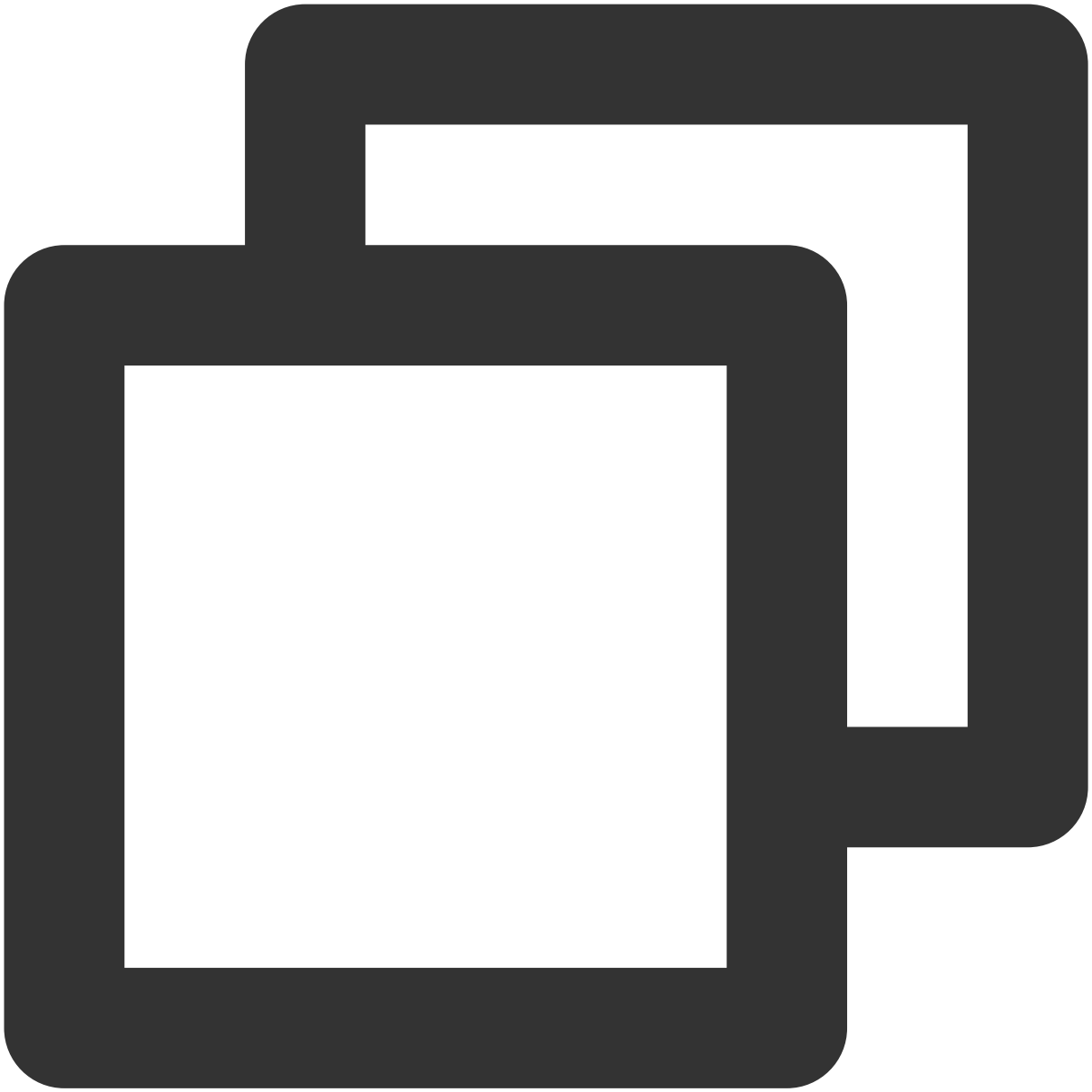
4. Run the following command to view the resource plan generated by Terraform based on the configuration file.



```
terraform plan
```

The returned information is as follows:

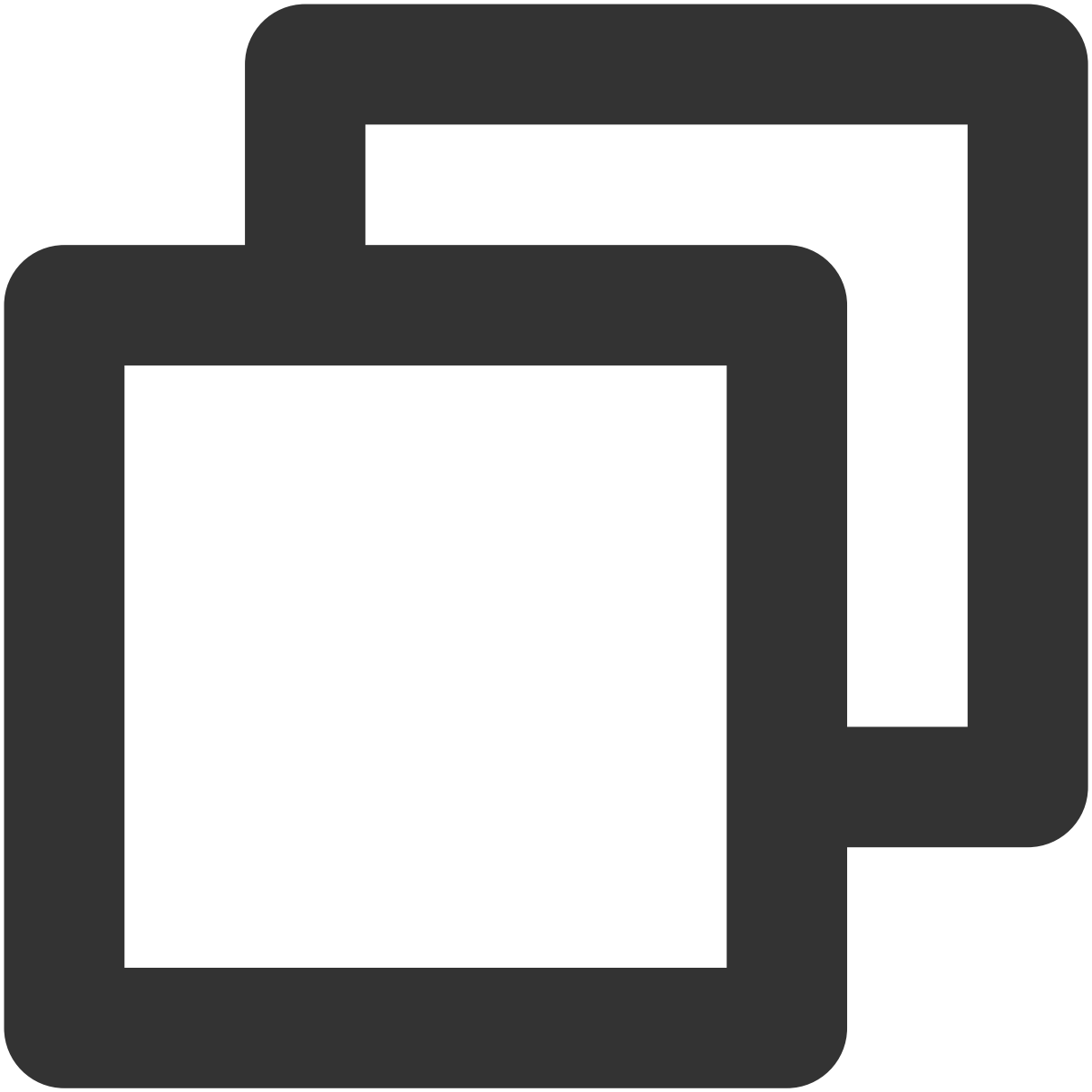
©2013-2022 Tencent Cloud. All rights reserved.



```
Terraform used the selected providers to generate the following execution plan. Res
  + create

Terraform will perform the following actions:
...

Plan: 3 to add, 0 to change, 0 to destroy.
...
```

5. Run the following command to create the resource.

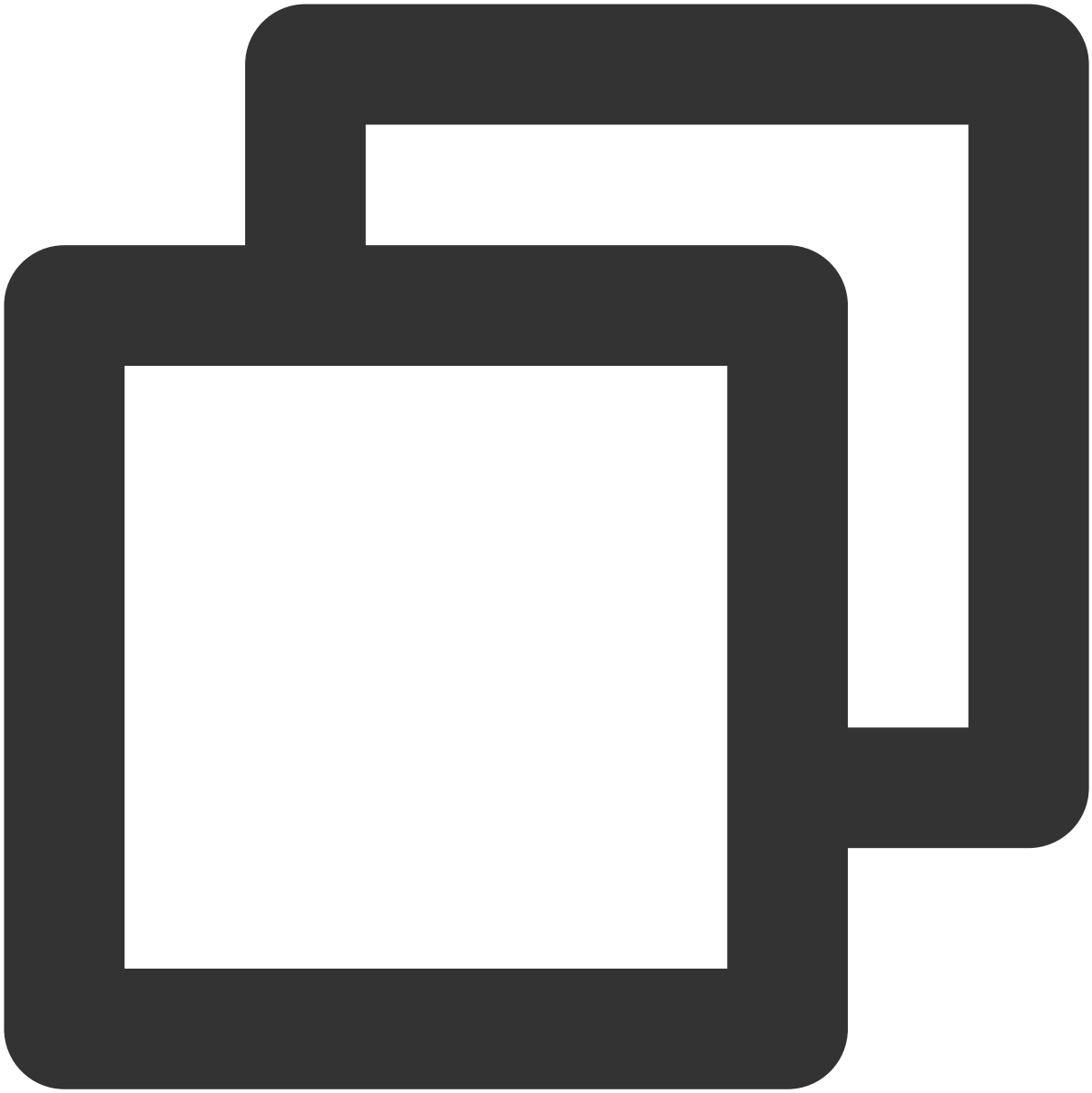
©2013-2022 Tencent Cloud. All rights reserved.



```
terraform apply
```

The returned information is as follows:

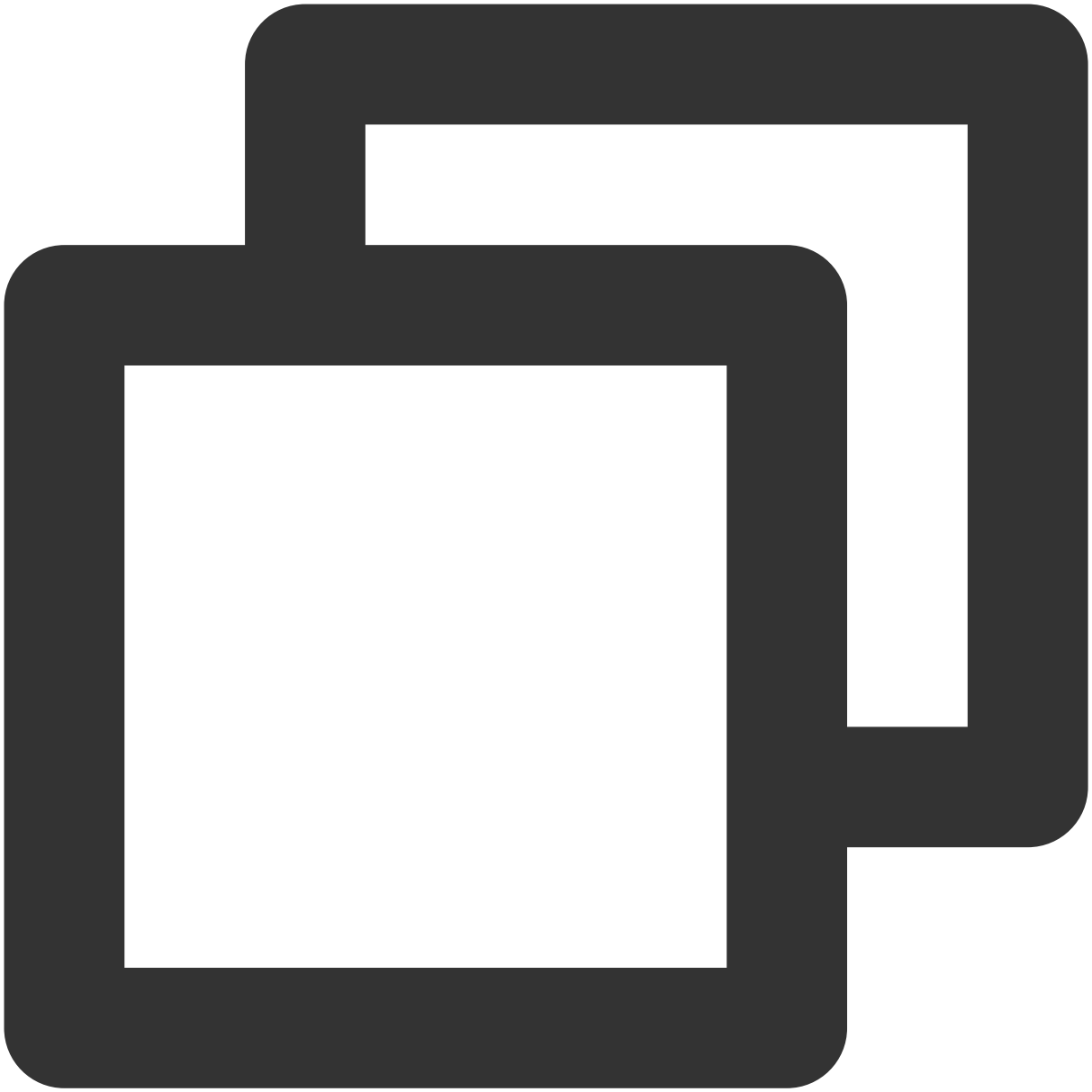
©2013-2022 Tencent Cloud. All rights reserved.



```
...
Plan: 3 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value:
```

Enter `yes` as prompted to create the resource. The following information is returned:

©2013-2022 Tencent Cloud. All rights reserved.



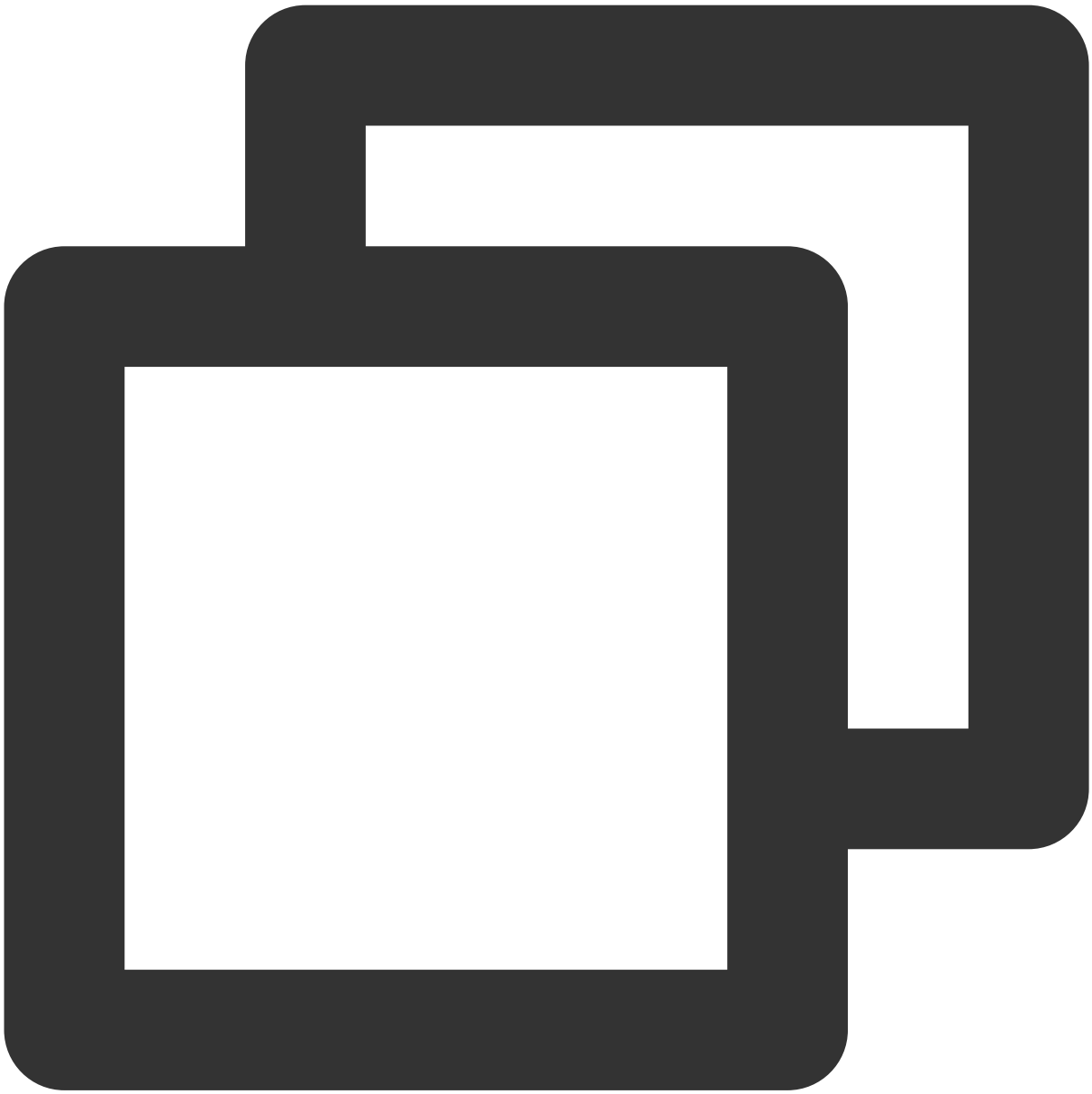
```
...
Apply complete! Resources: 3 added, 0 changed, 0 destroyed.
```

You have completed the creation of the VPC, subnet and managed TKE cluster. You can view these resources in Tencent Cloud console.

Creating a TKE Node Pool with Terraform

©2013-2022 Tencent Cloud. All rights reserved.

1. Create a working directory, under which create a Terraform configuration file named `nodepool.tf`.
The content of the `nodepool.tf` file is as follows:



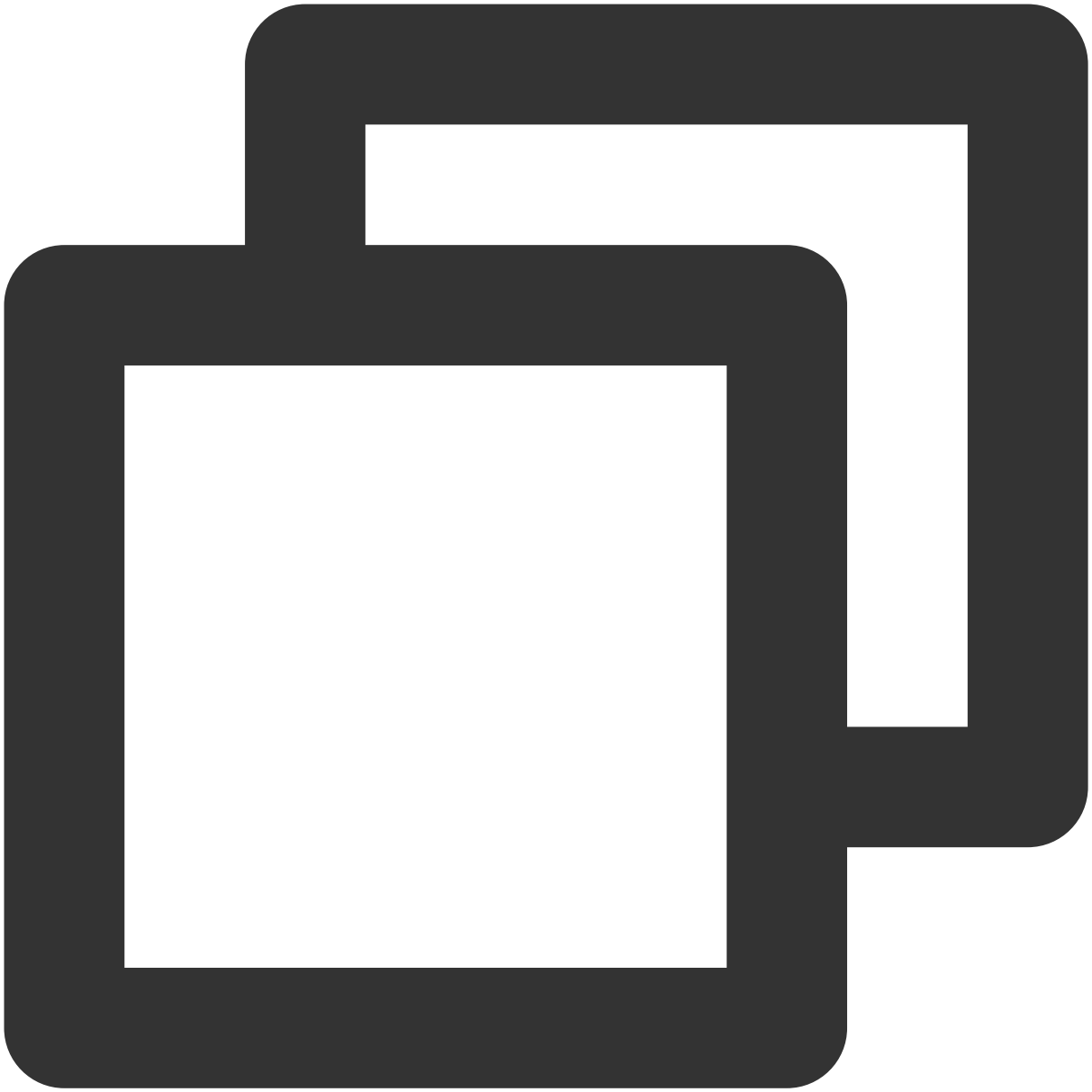
```
# Define local variables and modify the values as needed when using them in subsequ
# You can also reference Terraform related resource instance (such as `tencentcloud
locals {
    node_pool_name = "xxx"                                  # Node pool name, such as `
    max_node_size = xxx                                             # Max number of nod
    min_node_size = xxx                                             # Min number of nod
    cvm_instance_type = "xxx"                       # CVM instance in the node pool. Fo
    cvm_pass_word = "xxx"                                           # Login password fo
```

©2013-2022 Tencent Cloud. All rights reserved.

```
    security_group_ids = ["sg-xxx", "sg-xxx"]
}

# Declare TKE node pool resources
resource "tencentcloud_kubernetes_node_pool" "example_node_pool" {
  cluster_id = tencentcloud_kubernetes_cluster.managed_cluster_example.id # Associa
  delete_keep_instance = false                    # Set it to `false`, which indicat
  max_size   = local.max_node_size
  min_size   = local.min_node_size
  name       = local.node_pool_name
  vpc_id     = tencentcloud_vpc.vpc_example.id
  subnet_ids = [tencentcloud_subnet.subnet_example.id] # Array of IDs of subnets as
  auto_scaling_config {
    instance_type = local.cvm_instance_type
    # key_ids = ["xxx"]                                 # Set the login key for the
    password = local.cvm_pass_word                      # Set the login password for
    security_group_ids = local.security_group_ids
  }
}
```

2. Run the following command to view the resource plan generated by Terraform based on the configuration file.

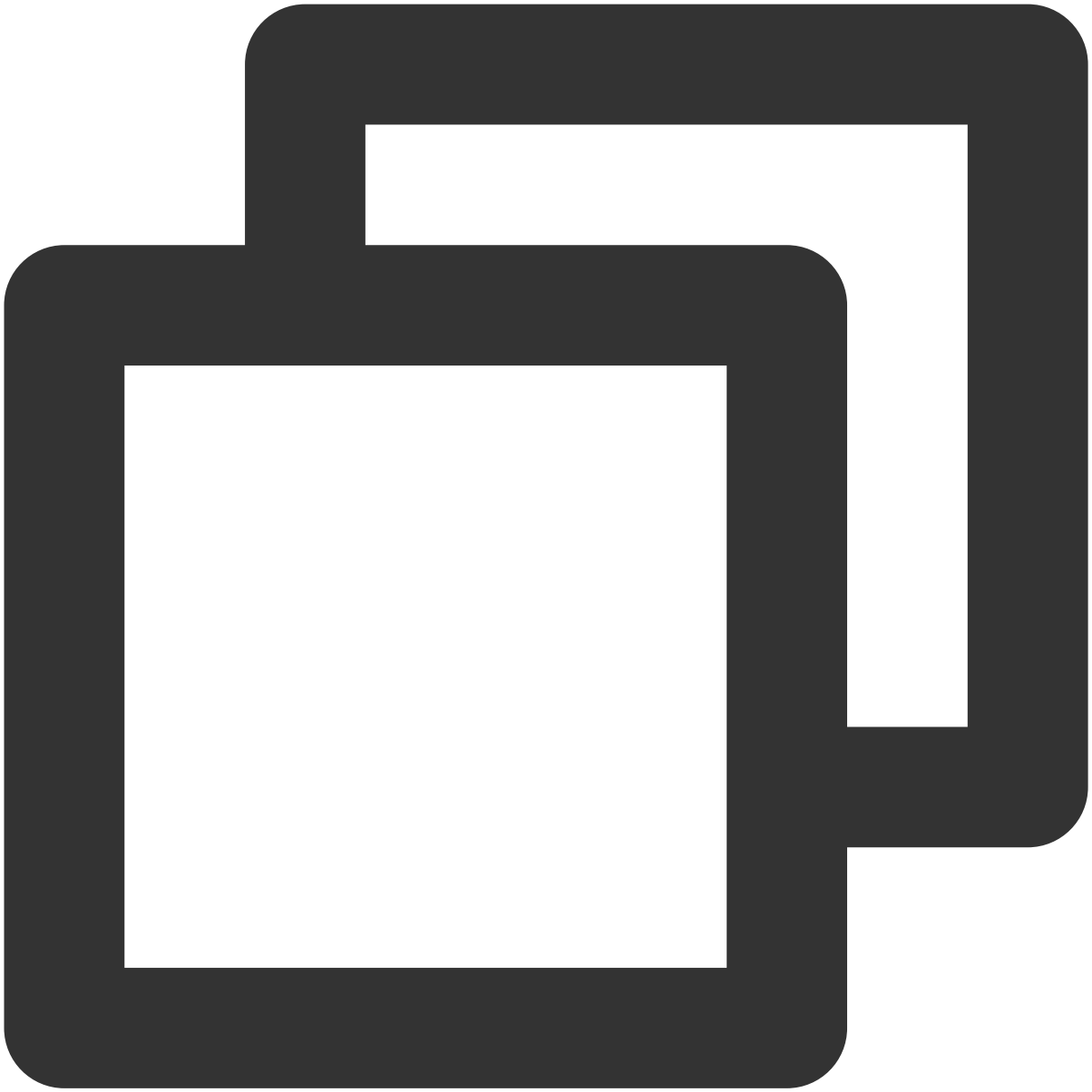
©2013-2022 Tencent Cloud. All rights reserved.



```
terraform plan
```

The returned information is as follows:

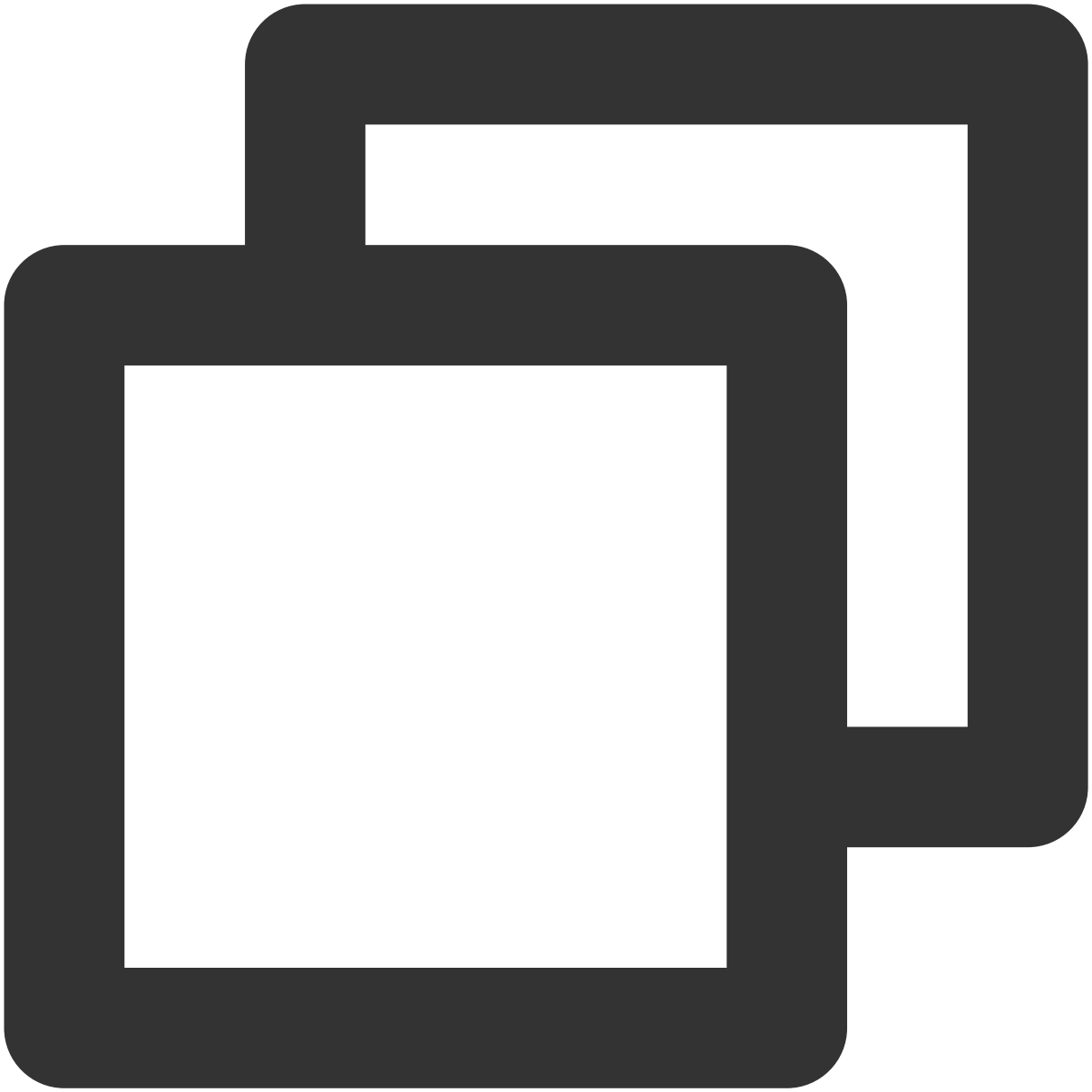
©2013-2022 Tencent Cloud. All rights reserved.



```
Terraform used the selected providers to generate the following execution plan. Res
  + create

Terraform will perform the following actions:
...

Plan: 1 to add, 0 to change, 0 to destroy.
...
```

3. Run the following command to create the resource.

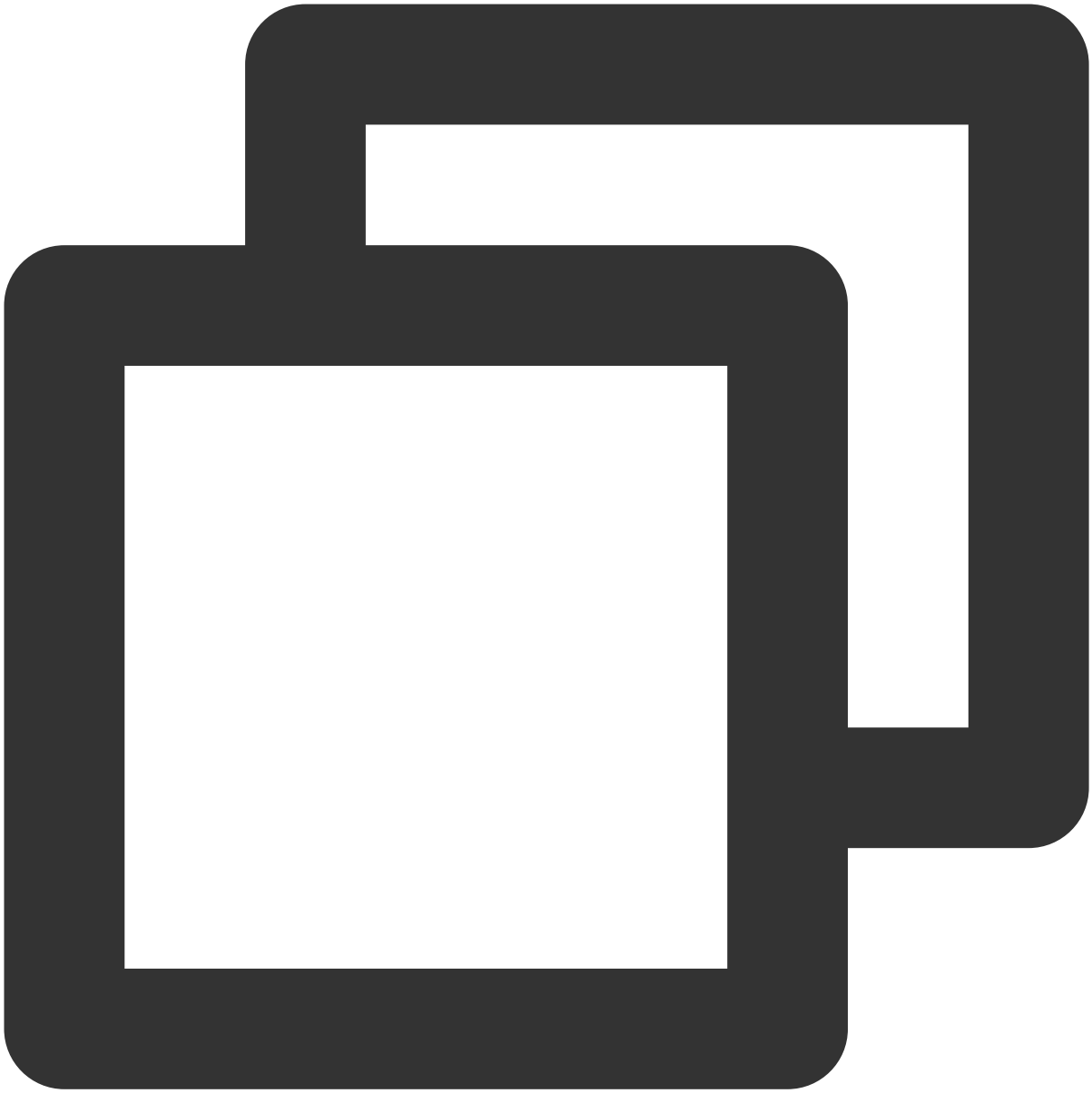
©2013-2022 Tencent Cloud. All rights reserved.



```
terraform apply
```

The returned information is as follows:

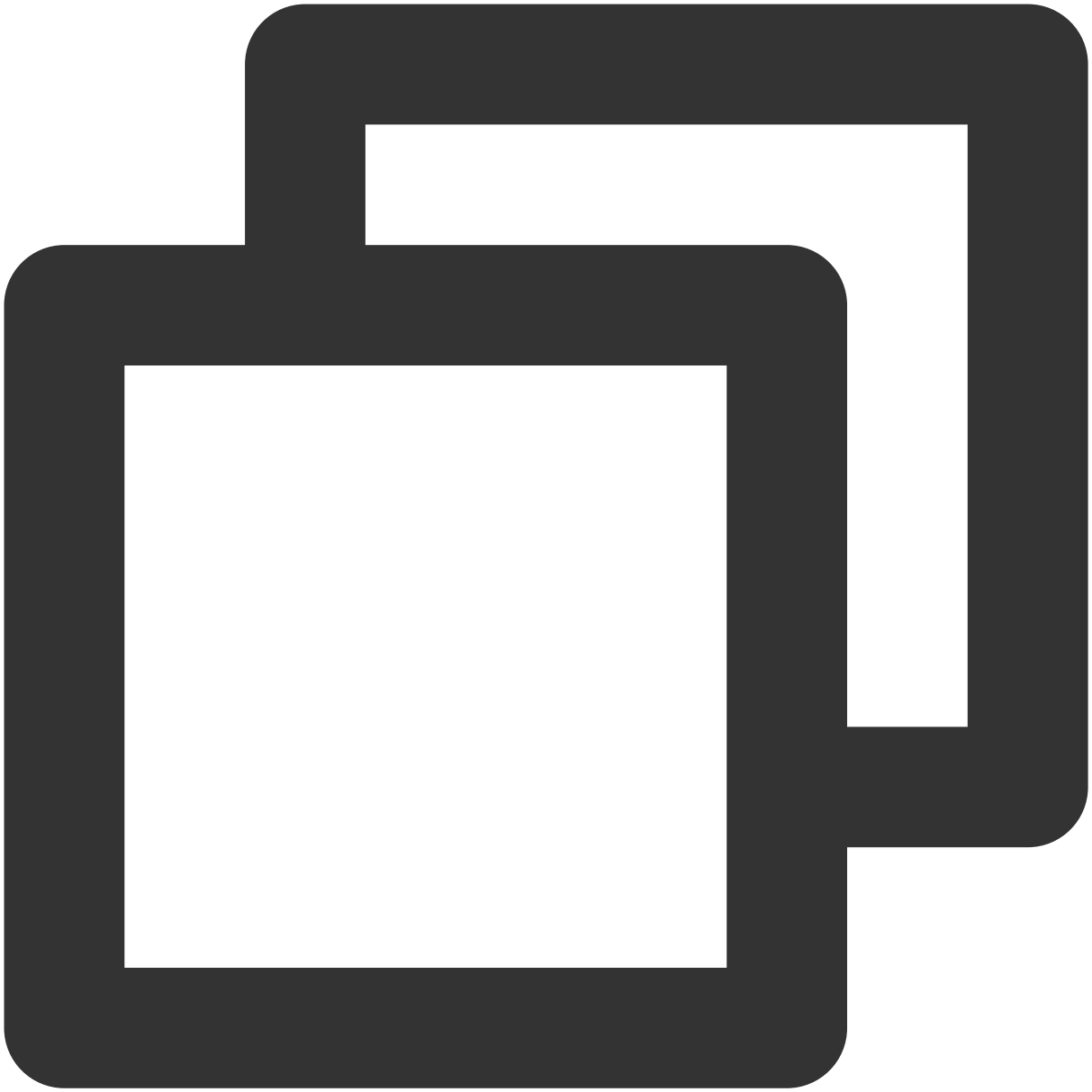
©2013-2022 Tencent Cloud. All rights reserved.



```
...
Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value:
```

Enter `yes` as prompted to create the resource. The following information is returned:

©2013-2022 Tencent Cloud. All rights reserved.



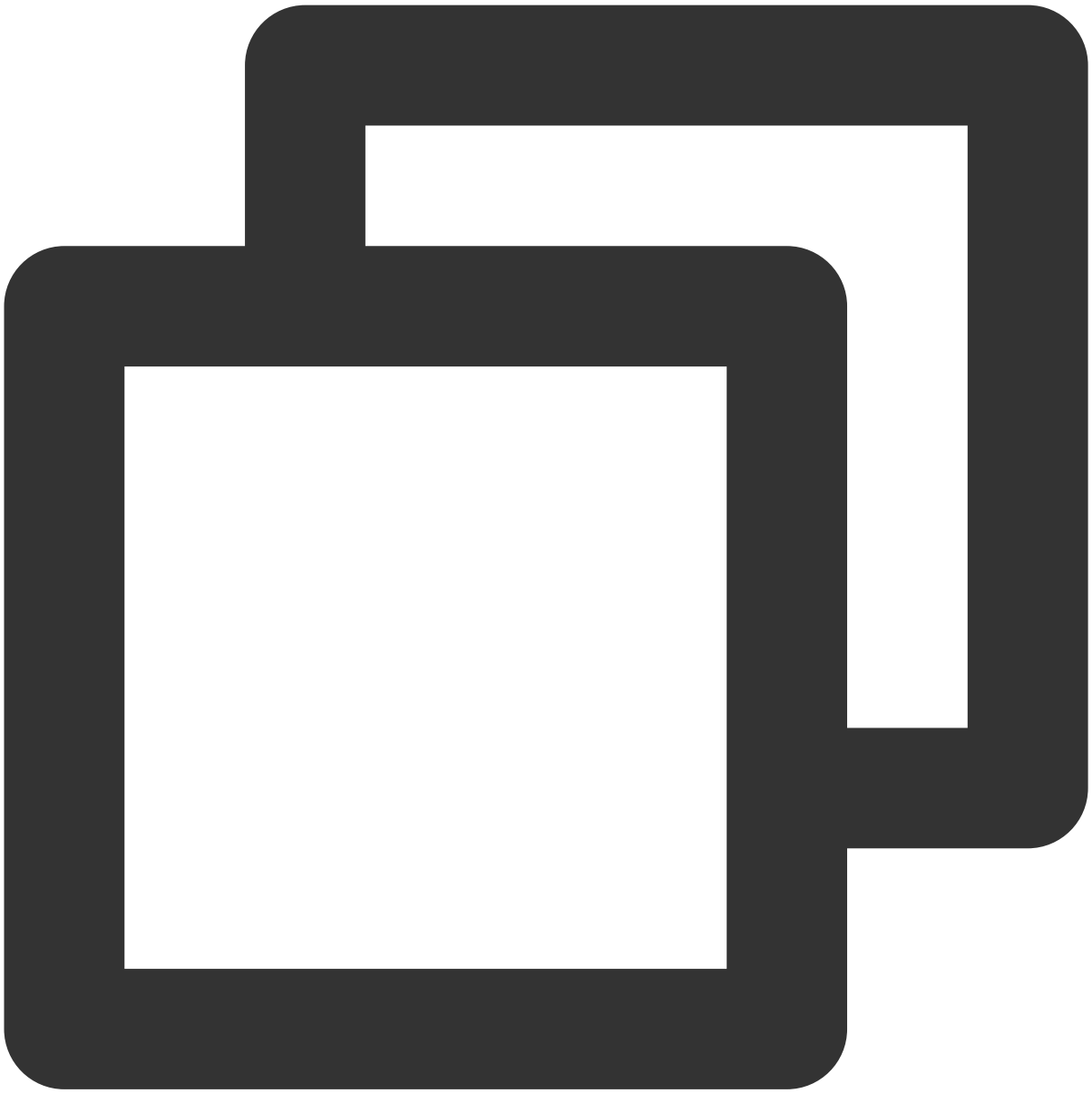
```
...
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

You have completed the creation of the node pool. You can view the resources you have created in Tencent Cloud console.

Cleaning up Resources with Terraform

©2013-2022 Tencent Cloud. All rights reserved.

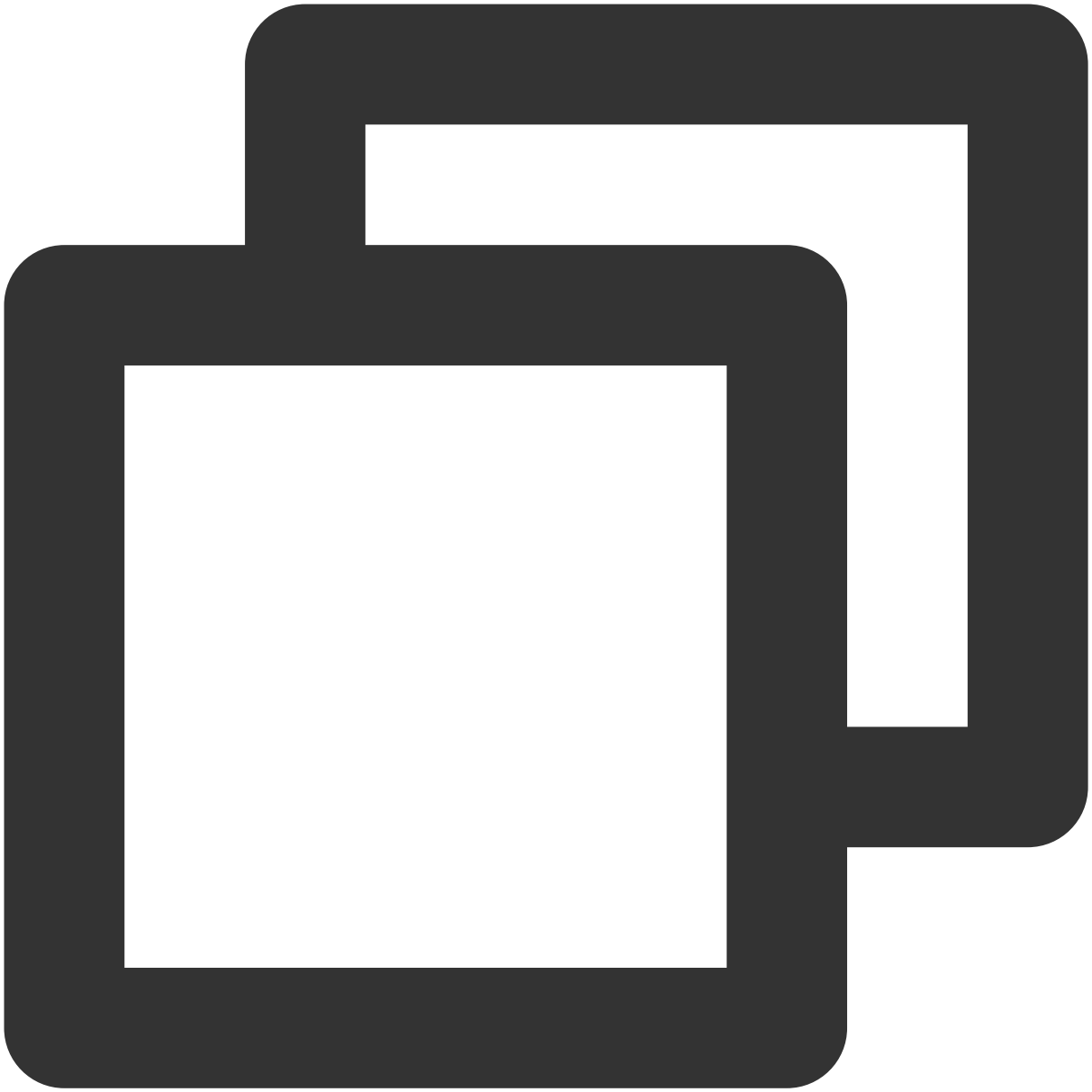
You can run the following command to delete the VPCs, subnets and managed TKE clusters you have created.



```
terraform destroy
```

The returned information is as follows:

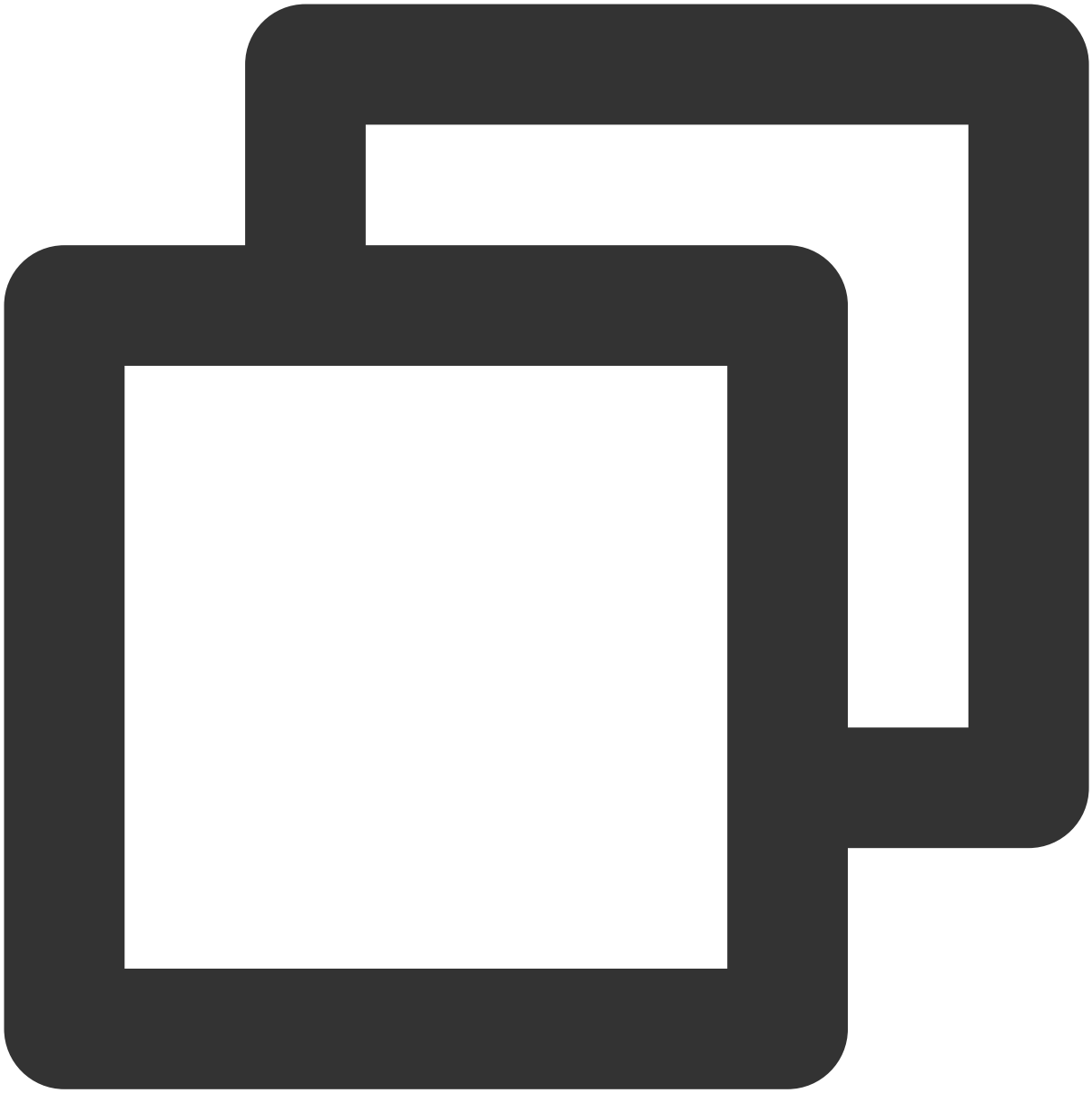
©2013-2022 Tencent Cloud. All rights reserved.



```
...
Plan: 0 to add, 0 to change, 3 to destroy.

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value:
```

Enter `yes` as prompted to confirm the deletion. The following information is returned:

©2013-2022 Tencent Cloud. All rights reserved.



```
...
Destroy complete! Resources: 3 destroyed.
```

References

Terraform documentation

Tencent Cloud Terraform Provider

©2013-2022 Tencent Cloud. All rights reserved.

Tencent Cloud General TKE cluster

Tencent Cloud TKE Node Pool

©2013-2022 Tencent Cloud. All rights reserved.

DevOps
Construction and Deployment of Jenkins Public Network Framework Appications based on TKE
Example

Last updated：2020-05-11 14:36:13

Introduction

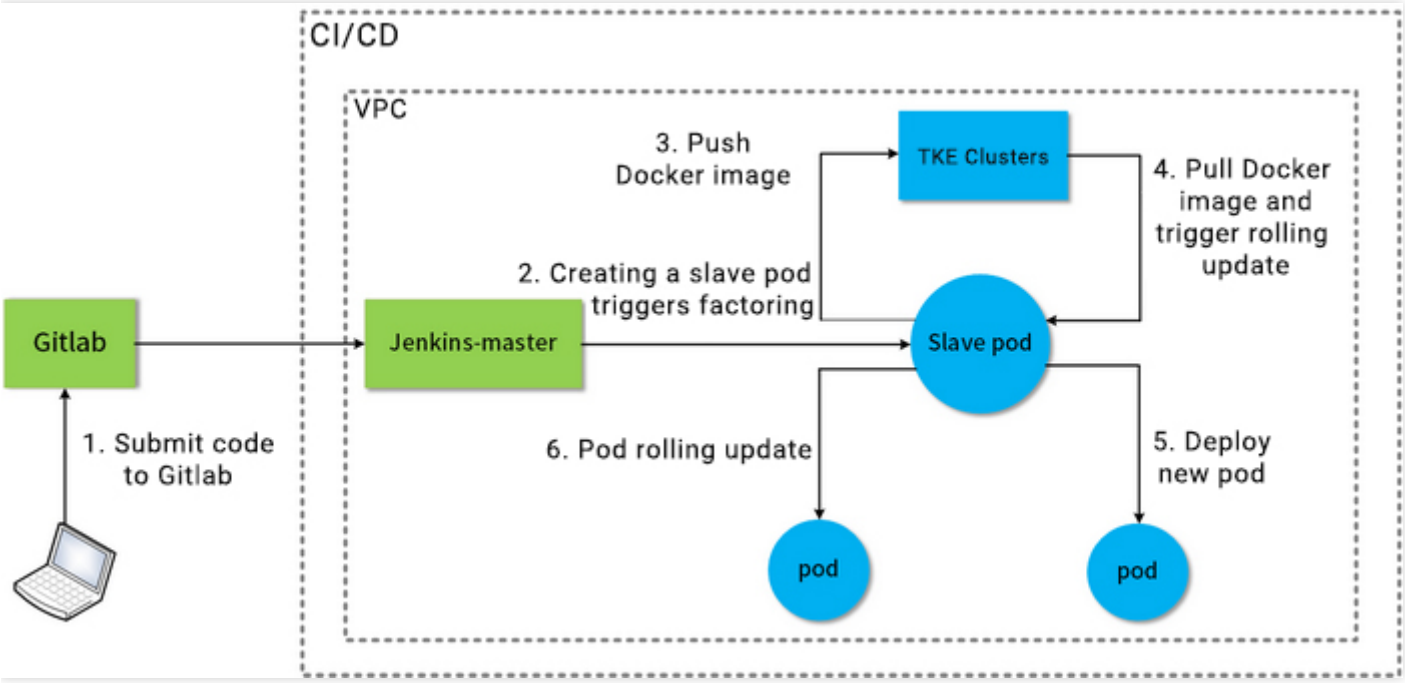
Jenkins helps users set up a continuous integration and continuous delivery environment. The Jenkins Master/Slave pod architecture can solve the pain points of concurrence restriction in batch building for enterprises, implementing continuous integration. This document describes how to use Jenkins in Tencent Cloud TKE to implement rapid and sustainable business delivery and reduce resource and labor costs.

How It Works

The TKE-based Jenkins public network architecture is used as an example in this document. In this architecture, the Jenkins Master is located outside the TKE cluster and the slave pod is located within the TKE cluster. The diagram of

©2013-2022 Tencent Cloud. All rights reserved.

the architecture is shown in the following figure:



- The Jenkins Master and TKE cluster are located in the same VPC network.
- The Jenkins Master is outside the TKE cluster, and the slave pod is in a node of the TKE cluster.
- The user submits code to GitLab, which triggers the Jenkins Master to call the slave pod to build, package, and then publish the image into the TKE image repository. The TKE cluster pulls the image and triggers rolling update for pod deployment.
- Multi-slave-pod building can meet the need of concurrent batch building.

Operation Environment

This section describes the specific environment in this scenario.

TKE cluster

Role	Kubernetes Version	Operating System
TKE managed cluster	1.16.3	CentOS 7.6.0_x64

Jenkins configuration

Role	Version
Jenkins Master	2.190.3

©2013-2022 Tencent Cloud. All rights reserved.

Role	Version
Jenkins Kubernetes plug-in	1.21.3

Nodes

Role	Private IP	Operating System	CPU	Memory	Bandwidth
Jenkins Master	10.0.0.7	CentOS 7.6 64-bit	4 cores	8 GB	3 Mbps
Node	10.0.0.14	CentOS 7.6 64-bit	2 cores	4 GB	1 Mbps

Notes

- Be sure that a Jenkins Master node is available under the same VPC as the TKE cluster, and that Git is installed for the node.
- Be sure that the GitLab code repository used in the steps already contains a Dockerfile file.
- We recommend that you set the TKE cluster and Jenkins Master security group as being fully open to the private network. For more information, see TKE Security Group Settings.

Procedure

Complete the following steps to configure the TKE cluster and Jenkins. Then, use the slave pod to build, package, and publish the image into the TKE image repository. Lastly, use the pulled image for pod deployment in the TKE console.

1. TKE Cluster and Jenkins Configuration
2. Slave Pod Building Configuration
3. Building Test

©2013-2022 Tencent Cloud. All rights reserved.

Step 1: Configure the TKE cluster and Jenkins

Last updated：2023-05-06 17:36:46

TKE Cluster Configuration

This document describes how to customize RBAC authorization ServiceAccount in TKE and get the cluster access address, token, and cluster CA certificate information required during Jenkins configuration.

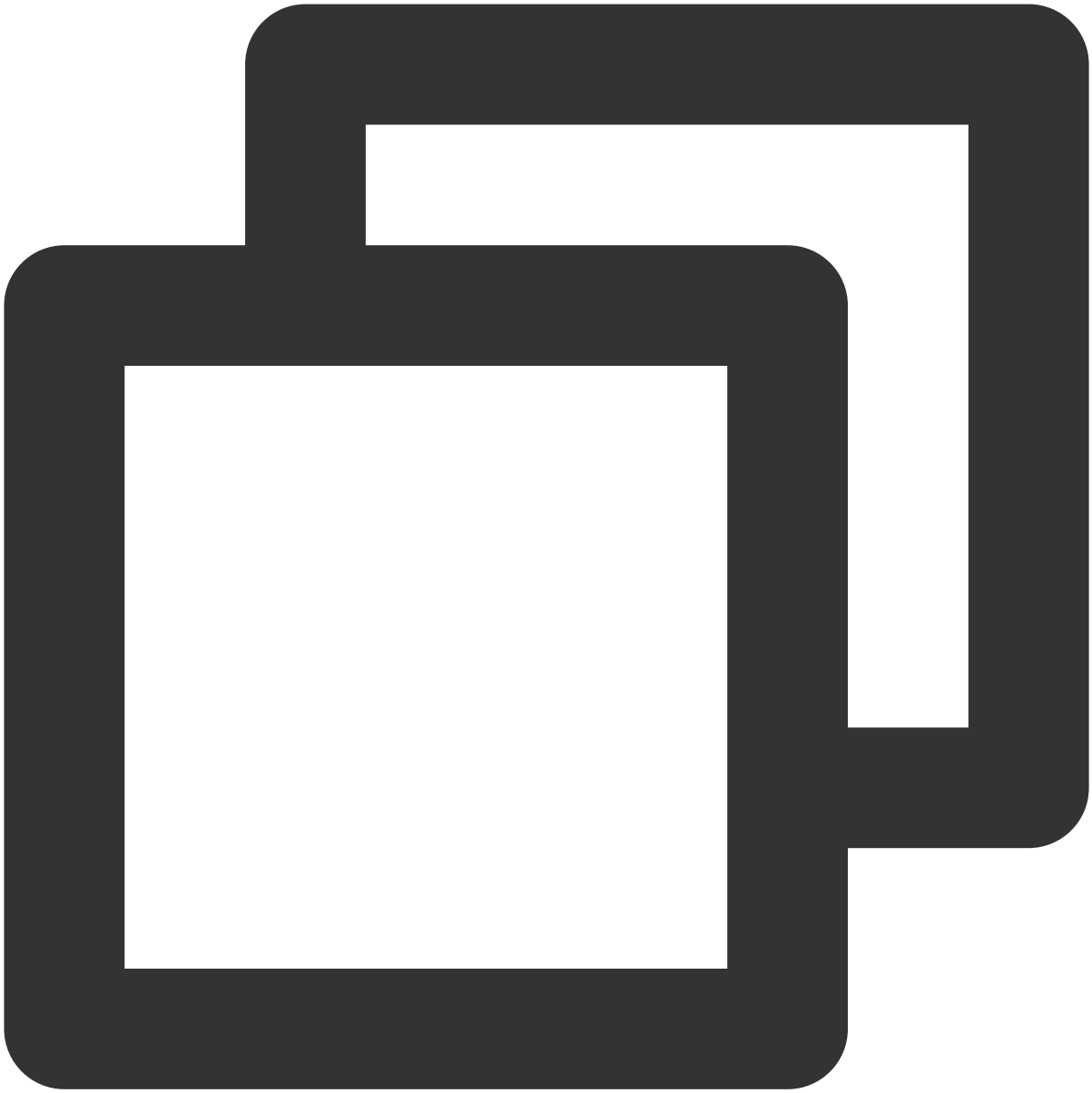
Getting the cluster credential

Note

You need to enable private network access in the current cluster. For more information, see Basic Features.

1. Use the following Shell script to create a test namespace `ci` and a test user `jenkins` of the ServiceAccount type and get the cluster access credential (token):

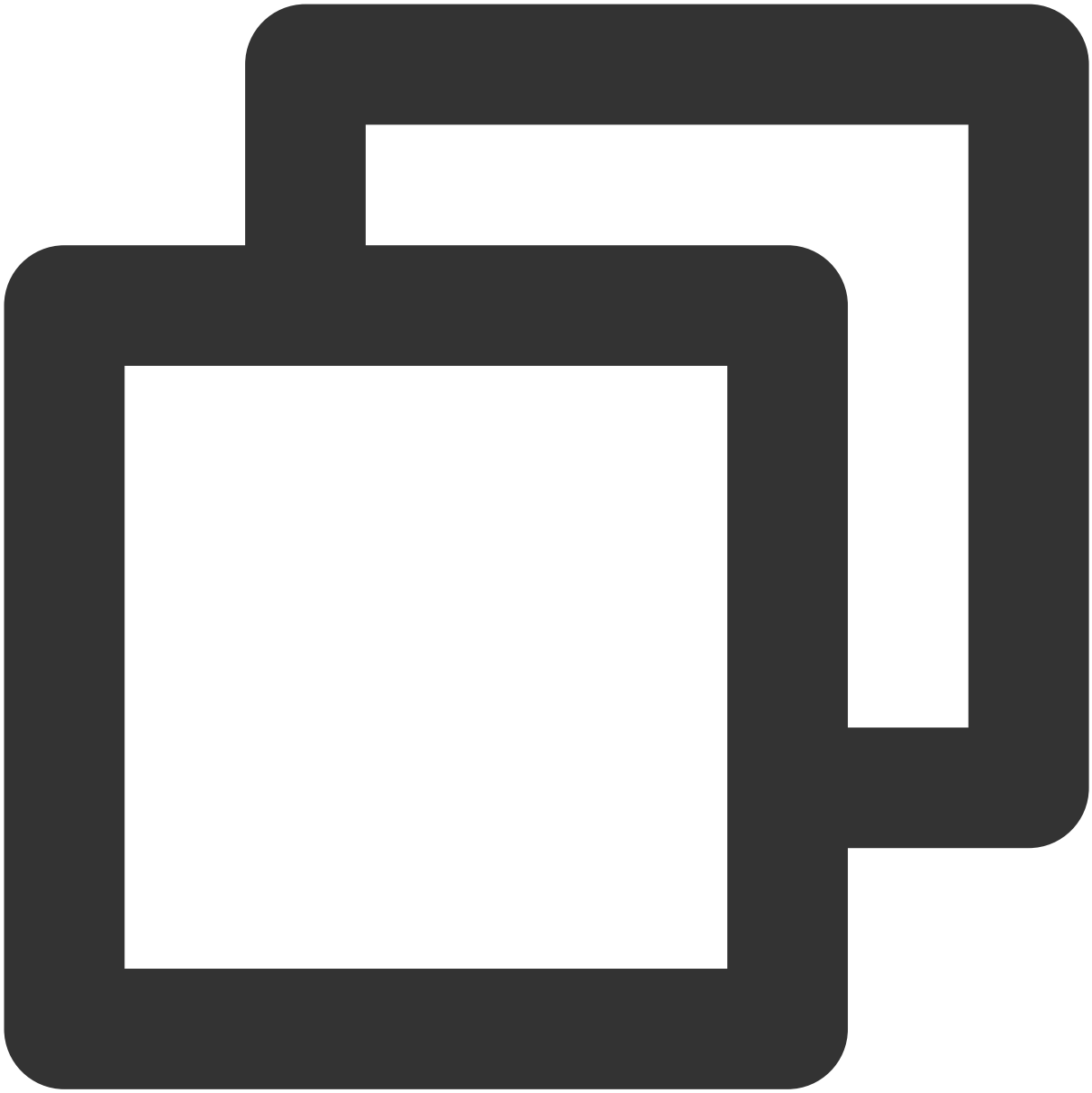
©2013-2022 Tencent Cloud. All rights reserved.



```
# Create the test namespace `ci`
kubectl create namespace ci
# Create the test ServiceAccount account
kubectl create sa jenkins -n ci
# Get the secret token automatically created by the ServiceAccount account
kubectl get secret $(kubectl get sa jenkins -n ci -o jsonpath={.secrets[0].name}) -
```

2. Create a Role permission object resource file `jenkins-role.yaml` in the `ci` test namespace as follows:

©2013-2022 Tencent Cloud. All rights reserved.

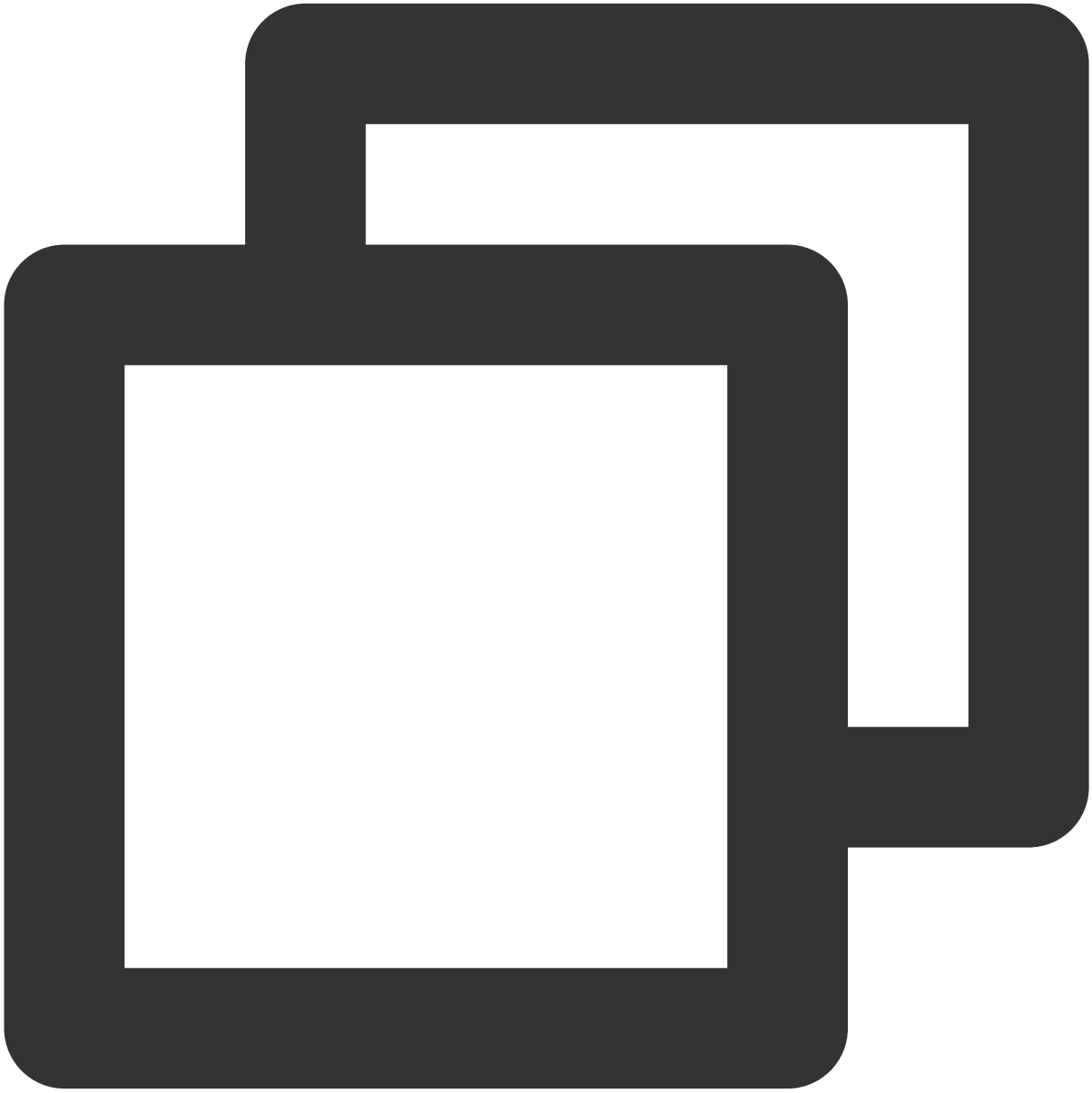


```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: jenkins
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["create","delete","get","list","patch","update","watch"]
- apiGroups: [""]
  resources: ["pods/exec"]
  verbs: ["create","delete","get","list","patch","update","watch"]
```

©2013-2022 Tencent Cloud. All rights reserved.

```
- apiGroups: [""]
  resources: ["pods/log"]
  verbs: ["get","list","watch"]
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get"]
```

3. Create a RoleBinding object resource file `jenkins-rolebinding.yaml` . The following permission binding indicates that the `jenkins` user of the ServiceAccount type has `jenkins` (Role type) permissions in the `ci` namespace, as shown below:



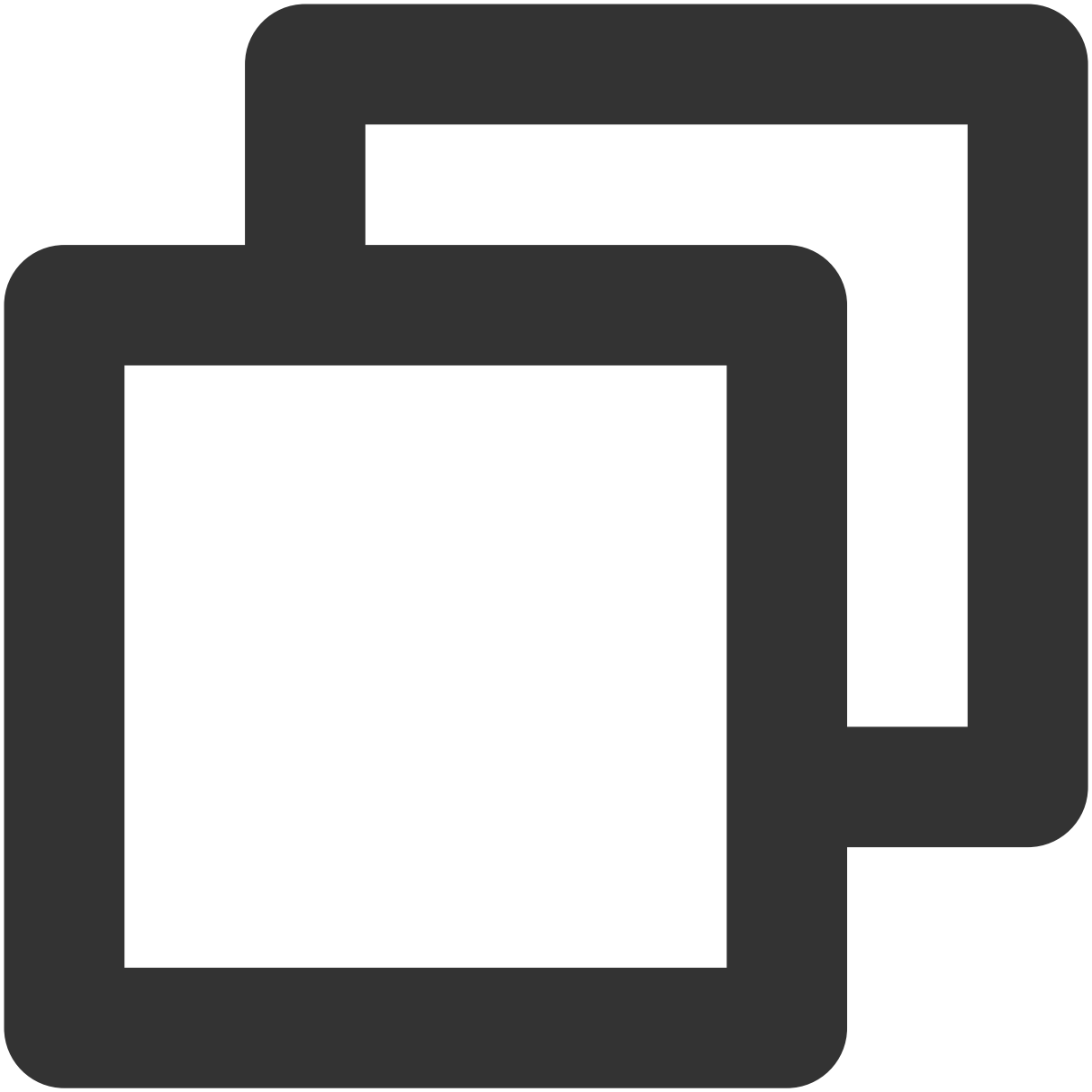
©2013-2022 Tencent Cloud. All rights reserved.

```
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: RoleBinding
metadata:
  name: jenkins
  namespace: ci
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: jenkins
subjects:
- kind: ServiceAccount
  name: jenkins
```

Getting the cluster CA certificate

1. Log in to the node of the cluster as instructed in Logging In To Linux Instance (Web Shell).
2. Run the following command to view the cluster CA certificate:

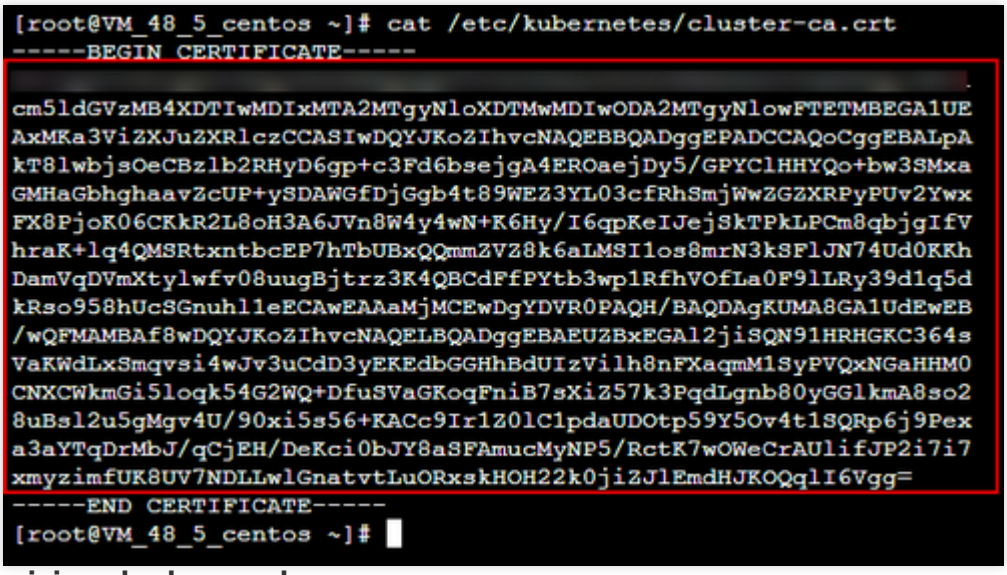
©2013-2022 Tencent Cloud. All rights reserved.



```
cat /etc/kubernetes/cluster-ca.crt
```

3. Record and save the returned certificate information as shown below:

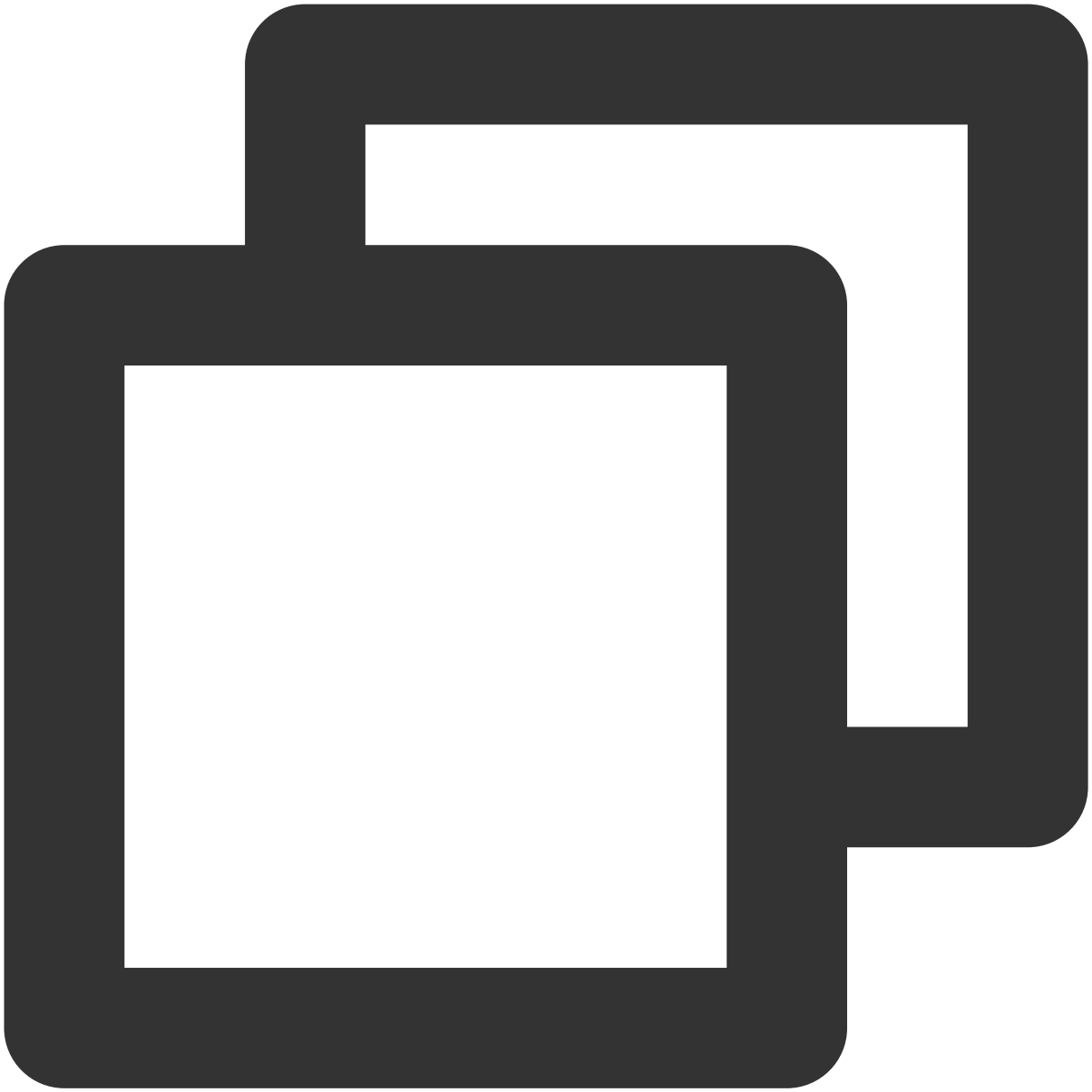
©2013-2022 Tencent Cloud. All rights reserved.



Authorizing docker.sock

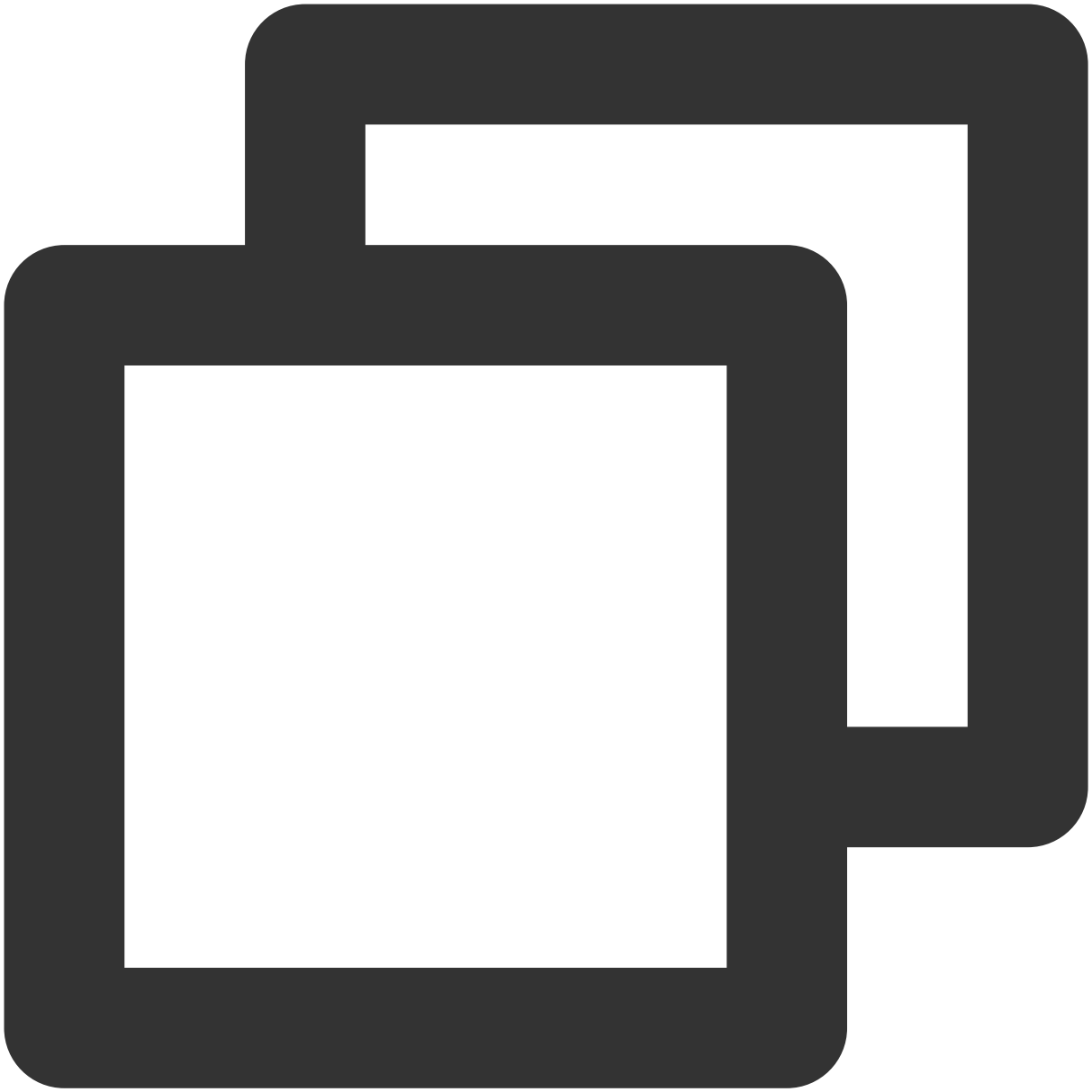
Each node of the TKE cluster has a `docker.sock` file. The slave pod connects to this file when running `docker build`. Before that, you need to log in to each node and run the following commands to authorize `docker build`:

©2013-2022 Tencent Cloud. All rights reserved.



```
chmod 666 /var/run/docker.sock
```

©2013-2022 Tencent Cloud. All rights reserved.



```
ls -l /var/run/docker.sock
```

Configuring Jenkins

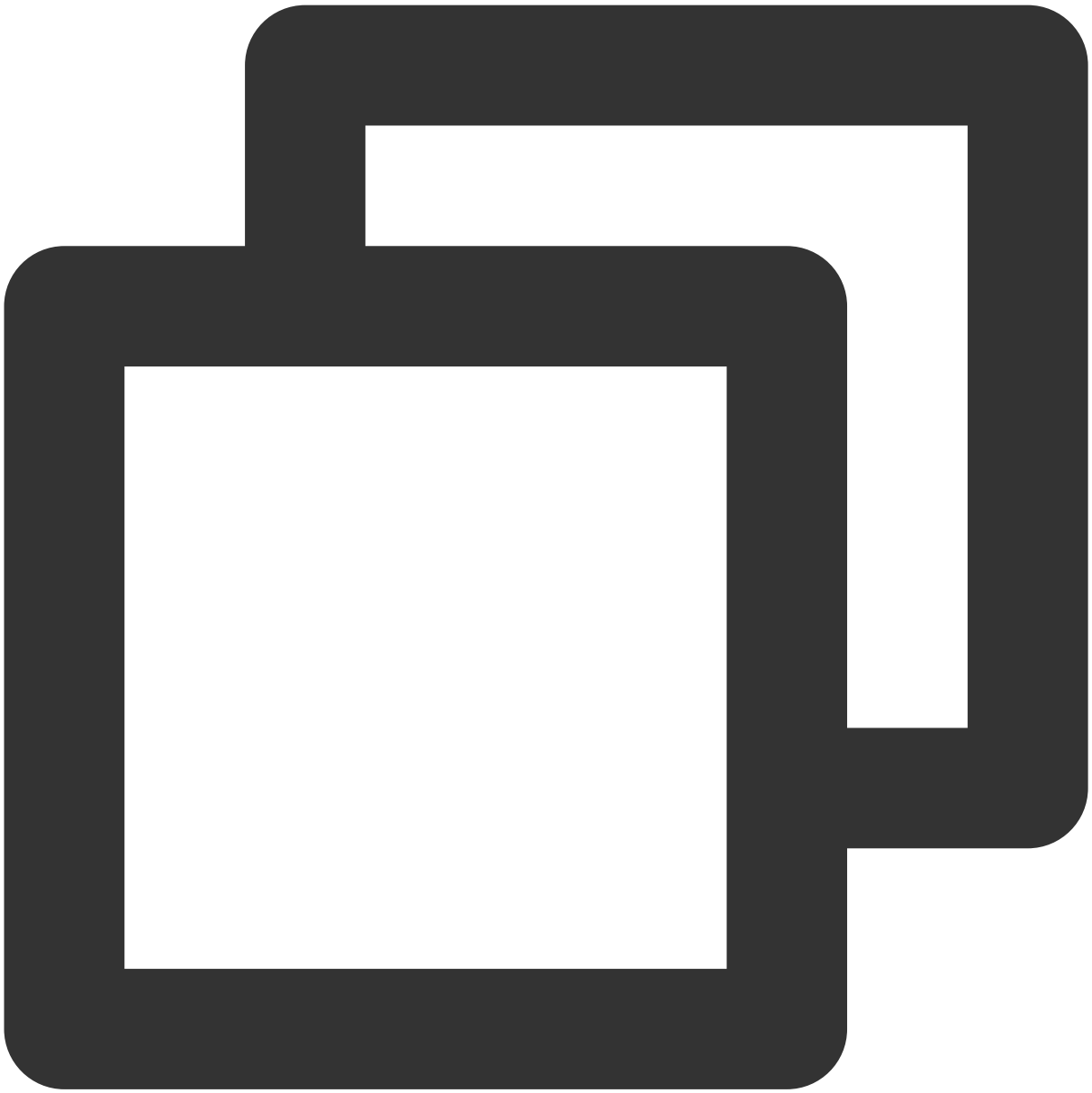
Note

The UI of Jenkins varies with the product version. Select an appropriate version based on your business needs.

Adding a TKE private network access address

©2013-2022 Tencent Cloud. All rights reserved.

1. Log in to the Jenkins master node as instructed in Logging In To Linux Instance (Web Shell).
2. Run the following command to configure the access address (domain name):



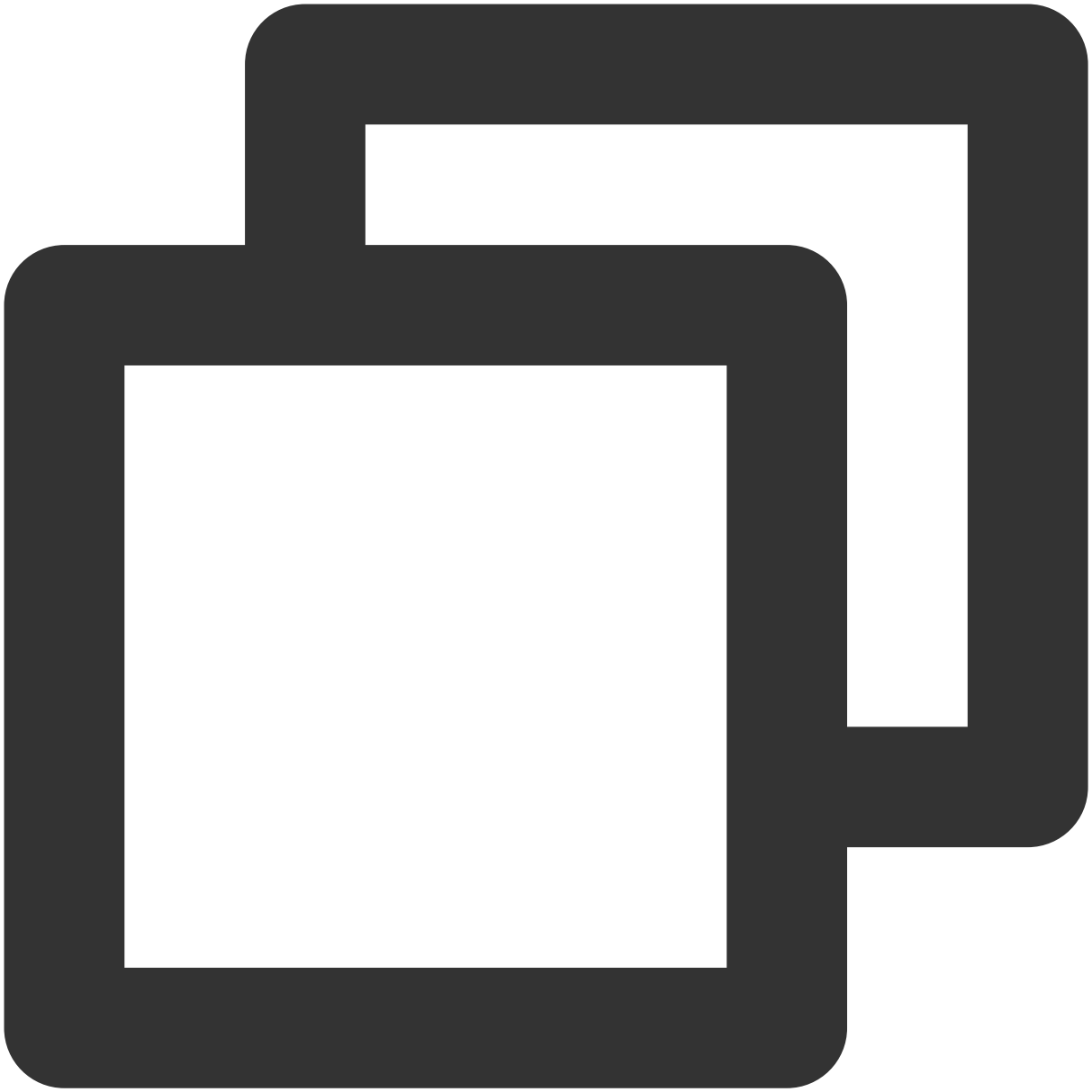
```
sudo sed -i '$a 10.x.x.x cls-ixxxelli.ccs.tencent-cloud.com' /etc/hosts
```

Note

This command can be obtained from **Cluster APIServer Information** on the basic information page of the cluster after private network access is enabled for the cluster. For more information, see Getting the cluster credential.

3. Run the following command to query whether the configuration is successful:

©2013-2022 Tencent Cloud. All rights reserved.



```
cat /etc/hosts
```

If the result shown in the following figure appears, the configuration was successful.

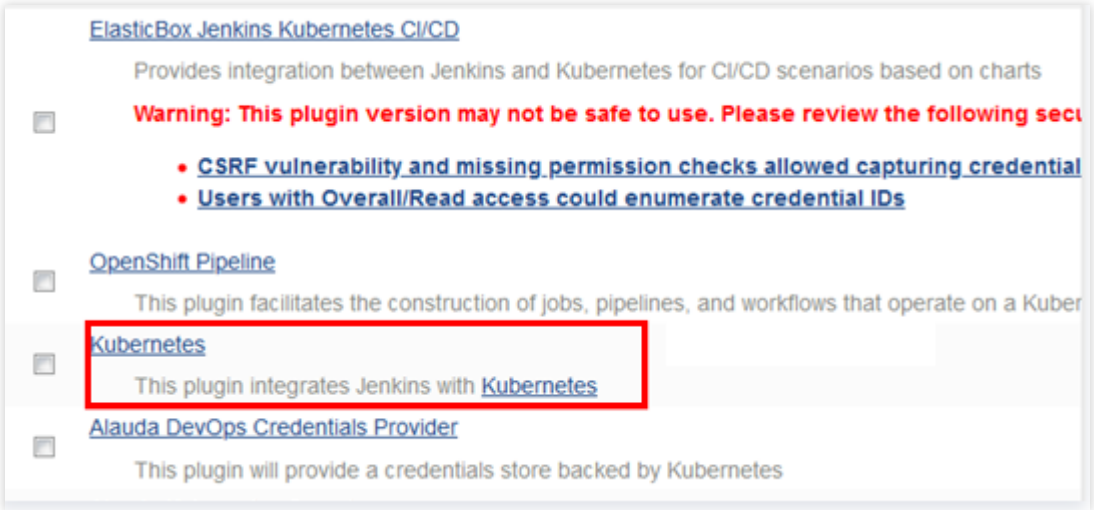
©2013-2022 Tencent Cloud. All rights reserved.

```
[root@VM_0_7_centos ~]# sudo sed -i '$a 10 [redacted].ccs.tencent-clou
[root@VM_0_7_centos ~]# cat /etc/hosts
127.0.0.1 VM_0_7_centos VM_0_7_centos
127.0.0.1 localhost.localdomain localhost
127.0.0.1 localhost4.localdomain4 localhost4

::1 VM_0_7_centos VM_0_7_centos
::1 localhost.localdomain localhost
::1 localhost6.localdomain6 localhost6

10. [redacted].ccs.tencent-cloud.com
```

Required plug-ins for Jenkins installation

1. Log in to the Jenkins backend and click **Manage Jenkins** in the left sidebar.
2. On the **Manage Jenkins** panel, click **Manage plug-ins**.
3. In the **Available** tab, check **Locale**, **Kubernetes**, **Git Parameter**, and **Extended Choice Parameter**.

Locale indicates a Chinese language plug-in. If this plug-in is installed, the Jenkins UI is in Chinese by default.
Kubernetes indicates the Kubernetes plug-in.
Git Parameter and **Extended Choice Parameter** are used to pass parameters during package building. The following figure shows the **Kubernetes** plug-in as an example:



4. After checking the preceding plug-ins, click **Install without restart** and restart Jenkins.

Enabling the jnlp port

1. Log in to the Jenkins backend and click **Manage Jenkins** in the left sidebar.
2. On the **Manage Jenkins** panel, click **Configure global security**.
3. In **TCP port for inbound agents**, check **Fixed** and enter **50000**.
4. Keep other configuration items as their defaults and click **Save** at the bottom of the page.

Adding the TKE cluster credential

1. Log in to the Jenkins backend and choose **Credentials** > **System** in the left sidebar.
2. On the **System** panel, select **Global credentials (unrestricted)**.

©2013-2022 Tencent Cloud. All rights reserved.

3. On the page that appears, click **Add credentials** in the left sidebar, and configure the basic credential information as follows:

Kind: Select **Secret text**.

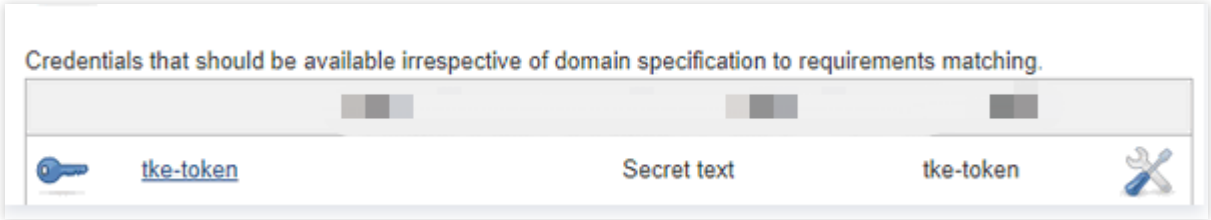
Scope: Use the default option **Global (Jenkins, nodes, items, all child items, etc)**.

Secret: Enter the **token** of ServiceAccount `jenkins` obtained in Getting the cluster credential.

ID: Leave it blank as default.

Description: Complete the information about the credential, which is displayed as the credential name and descriptive information. This document uses `tke-token` as an example.

4. Click **OK** to add the credential. Once successfully added, the credential is displayed in the credential list as shown below:



Adding the GitLab credential

1. On the **Global credentials (unrestricted)** page, click **Add credentials** in the left sidebar, and configure the basic credential information as follows:

Kind: Select **Username with password**.

Scope: Use the default option **Global (Jenkins, nodes, items, all child items, etc)**.

Username: Enter the GitLab username.

Password: Enter the GitLab login password.

ID: Leave it blank as default.

Description: Complete the information about the credential, which is displayed as the credential name and descriptive information. This document uses `gitlab-password` as an example.

2. Click **OK**.

Configuring the slave pod template

1. Log in to the Jenkins backend and click **Manage Jenkins** in the left sidebar.

2. On the **Manage Jenkins** panel, click **Configure system**.

3. At the bottom of the **Configure system** panel, choose **Add a new cloud** > **Kubernetes** in the **Cloud** section.

4. Click **Kubernetes Cloud details...** to configure the following basic information for Kubernetes.

The following describes the main parameters. For other parameters, simply keep them as their defaults:

Name: Enter a custom name. This document uses `kubernetes` as an example.

Kubernetes URL: Specify the TKE cluster access address. For more information, see Getting the cluster credential.

Kubernetes server certificate key: Specify the cluster CA certificate. For more information, see Getting the cluster CA certificate.

©2013-2022 Tencent Cloud. All rights reserved.

Credentials: Select the `tke-token` credential created in the Adding the TKE cluster token step and then click **Test connection**. If the connection succeeds, the "Connection successful" prompt appears.
Jenkins URL: Enter a Jenkins private network address, such as `http://10.x.x.x:8080` .
5. Choose **Pod templates** > **Add pod template** > **Pod template details...** and configure the basic information of the pod template.
The following describes the main parameters. For other parameters, simply keep them as their defaults:
Name: Enter a custom name. This document uses `jnlp-agent` as an example.
Labels: Define the tag name. You can select a pod for building based on the tag. This document uses `jnlp-agent` as an example.
Usage: Select **Use this node as much as possible**.
6. In the **Containers** drop-down list, choose **Add container** > **Container template** and configure the following container information:
Name: Enter a custom container name. This document uses `jnlp-agent` as an example.
Docker image: Enter the image address `jenkins/jnlp-slave:alpine` .
**Working directory*: Keep it as its default. Save the working directory, which will be used for building and packaging shell scripts.
Leave other configuration items as their defaults.
7. In **Volume**, complete the following steps to add a volume and configure the docker command for the slave pod.
7.1 Choose **Add volume** > **Host path volume**. Enter `/usr/bin/docker` for both the host and mount paths.
7.2 Choose **Add volume** > **Host path volume**. Enter `/var/run/docker.sock` for both the host and mount paths.
7.3 Click **Save** at the bottom of the page to finish configuring the slave pod template.

Subsequent Operations

Go to Step 2: Configure Slave Pod Building to create a task and configure task parameters.

©2013-2022 Tencent Cloud. All rights reserved.

Step 2:Slave pod build configuration

Last updated：2023-05-06 17:36:46

This document describes how to build a slave pod in Jenkins by creating and configuring a job.

Note

The UI of Jenkins varies with the product version. Select an appropriate version based on your business needs.

Creating a Job

1. Log in to the Jenkins backend and click **New Item** or **Create an item**.
2. On the page that appears, configure the basic information of the job.

Enter an item name: Enter a custom name. This document uses `test` as an example.

Select **Freestyle project**.

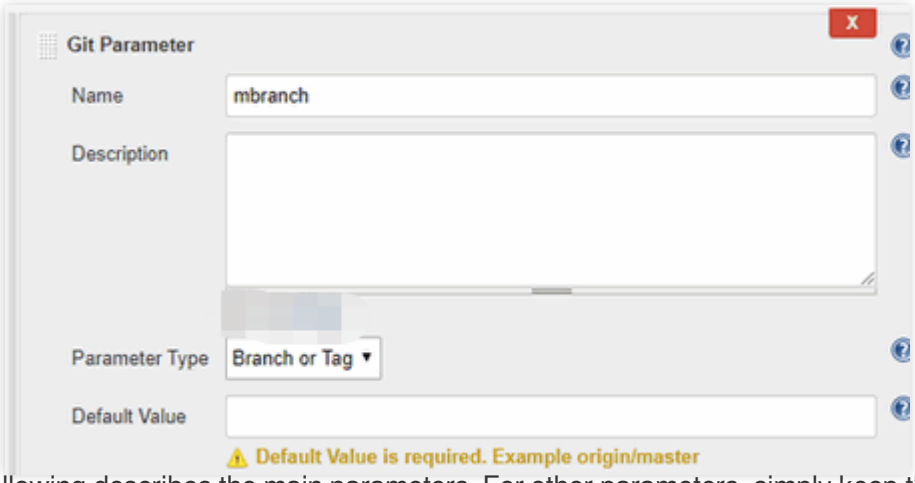
3. Click **OK** to go to the job parameter configuration page.
4. Configure the job basic information.

Description: Enter the descriptive information of the job. This document uses `slave pod test` as an example.

This is project is parameterized: Check this option and choose **Add Parameter** > **Git Parameter**.

Configuring Job Parameters

1. On the **Git Parameter** panel, configure the following parameters as shown below:



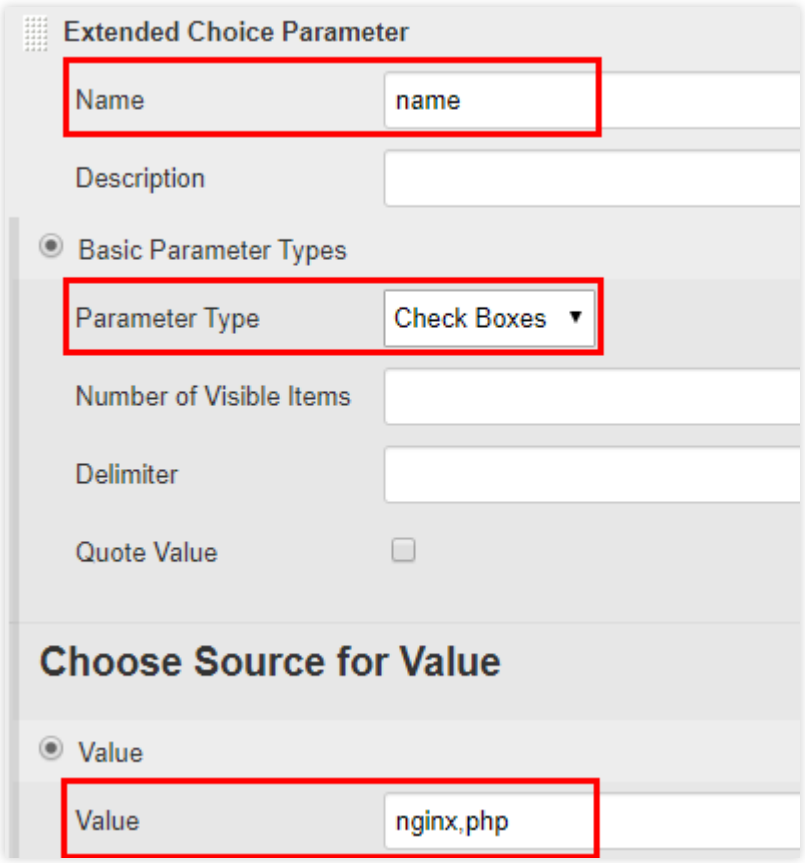
The following describes the main parameters. For other parameters, simply keep them as their defaults:

Name: Enter `mbranch`, which can be used to match and obtain a branch.

Parameter Type: Select **Branch or Tag**.

2. Choose **Add Parameter** > **Extended Choice Parameter**. On the panel that appears, configure the following parameters as shown below:

©2013-2022 Tencent Cloud. All rights reserved.



The following describes the main parameters. For other parameters, simply keep them as their defaults:

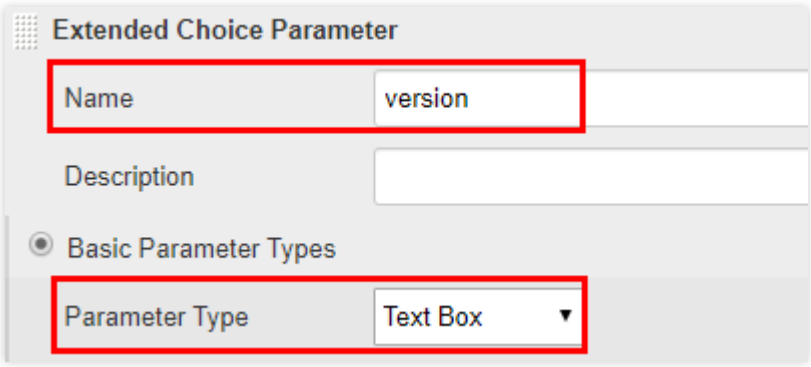
Name: Enter `name`, which can be used to obtain the image name.

Basic Parameter Types: Check this option.

Parameter Type: Select **Check Boxes**.

Value: Check this option and enter a custom image name. The name will be passed to the `name` variable. This document uses `nginx,php` as an example.

3. Choose **Add Parameter** > **Extended Choice Parameter**. On the panel that appears, configure the following parameters as shown below:



The following describes the main parameters. For other parameters, simply keep them as their defaults:

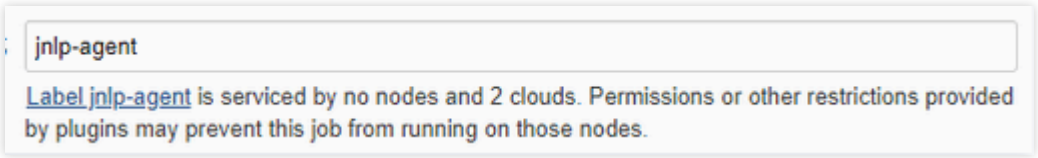
Name: Enter `version`, which can be used to obtain the image tag variable.

©2013-2022 Tencent Cloud. All rights reserved.

Basic Parameter Types: Check this option.

Parameter Type: Select **Text Box** to obtain the image value in text format and pass it to the `version` variable.

4. Check **Restrict where this project can be run**. For **Label Expression**, enter the pod label `jnlp-agent` set in the Configuring the slave pod template step as shown below:



Configuring Source Code Management

In the **Source Code Management** tab, check **Git** and configure the following settings:

Repositories:

Repository URL: Enter your GitLab repository address, such as `https://gitlab.com/user-name/demo.git` .

Credentials: Select the authentication credential created in the Adding the GitLab credential step.

Branches to build:

Branch Specifier (blank for 'any'): Enter `$mbranch` , which is used to dynamically obtain the branch. Its value corresponds to the value of `mbranch` defined in "Git Parameter".

Configuring the Shell Packaging Script

1. In the **Build** tab, choose **Add build step** > **Execute Shell**.

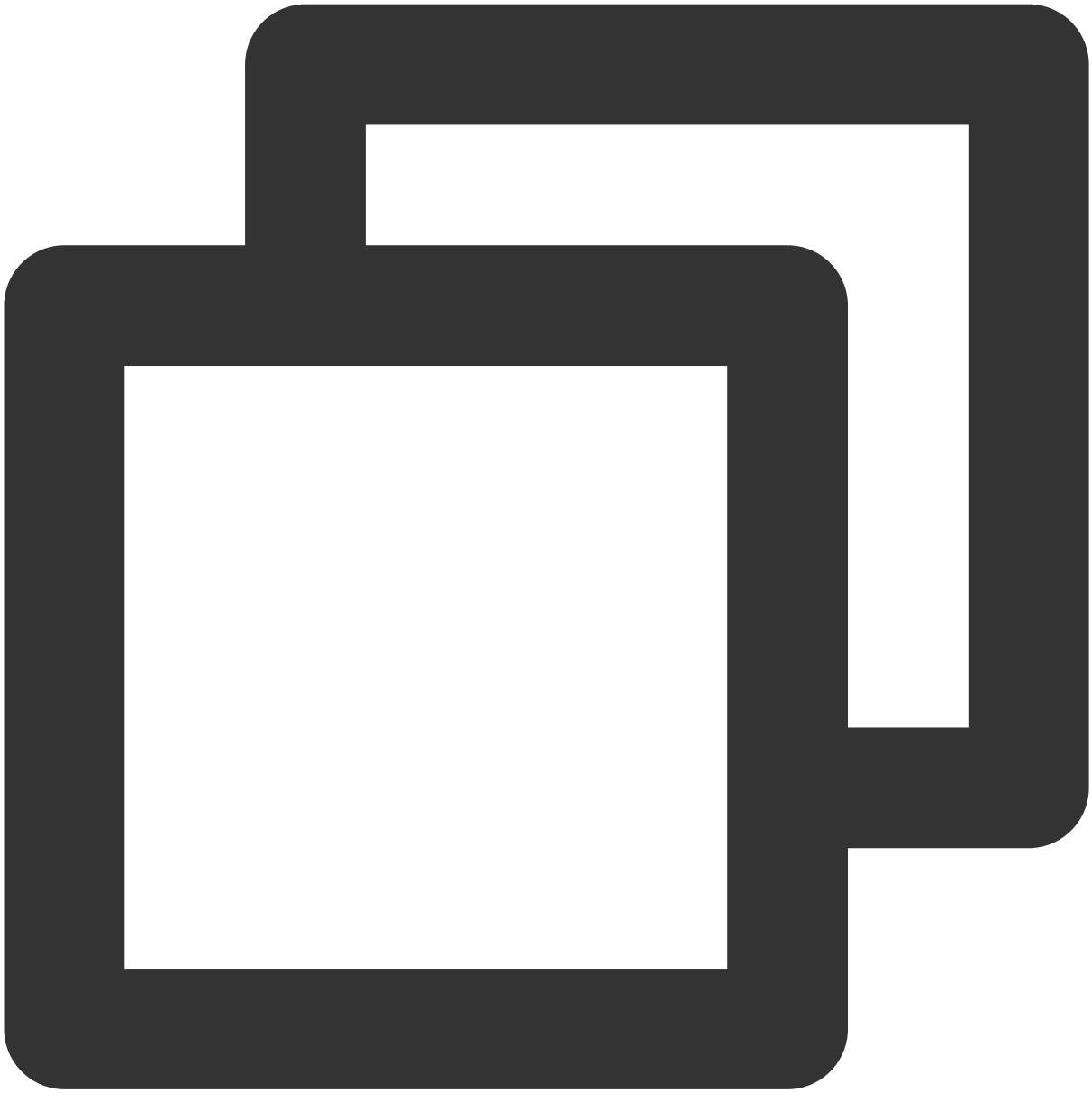
2. Copy and paste the following script to the **Command** entry box. Then, click **Save**.

Note

In this script, information such as the GitLab repository address, TKE image address, and username and password of the image repository are used as examples only. In actual cases, replace them based on your needs.

Make sure that you build the package based on the source code of Docker build. In addition, the working directory `/home/Jenkins/agent` must be consistent with the working directory of the Container Template in the “Containers” list.

©2013-2022 Tencent Cloud. All rights reserved.



```
echo "GitLab address: https://gitlab.com/[user]/[project-name]].git"
echo "Selected branch (image): "$mbranch", set branch (image) version: "$version"
echo "TKE image address: hkccr.ccs.tencentyun.com/[namespace]/[ImageName]"

echo "1. Log in to the TKE image repository"
docker login --username=[username] -p [password] hkccr.ccs.tencentyun.com

echo "2. Build the package based on the source code of Docker build:"
cd /home/Jenkins/agent/workspace/[project-name] && docker build -t $name:$version

echo "3. Upload the Docker image to the TKE repository:"
```

©2013-2022 Tencent Cloud. All rights reserved.

```
docker tag $name:$version hkccr.ccs.tencentyun.com/[namespace]/[ImageName]:$name-$
docker push hkccr.ccs.tencentyun.com/[namespace]/[ImageName]:$name-$version
```

The script provides the following features:

Obtain the selected branch, image name, and image tag.

Publish the docker image combined and built with the code to the TKE image repository.

Subsequent Operations

You have now successfully built the slave pod. Next, go to Building Tests to publish and verify images.

©2013-2022 Tencent Cloud. All rights reserved.

Build test

Last updated：2023-05-24 17:14:24

This step describes how to publish one or more images in the TKE image repository, and how to use an image to create a Deployment in the TKE console.

Building configuration

1. Log in to the Jenkins backend and click the task "test" created in the Slave pod building configuration step from the task list, as shown in the following figure:
2. Click **Build with Parameters** in the left sidebar to open the "Project test" panel and configure the following parameters:

- **mbranch**: select the branch required for building. This document uses `origin/nginx` as an example.
  - **name**: select the name of the image to be built based on your actual needs. This document uses `nginx` as an example.
  - **version**: enter a custom image tag. This document uses `v1` as an example.

3. Click **Start Building**.
   After the building is successfully completed, go to the TKE console and choose **Image Repository** > **My Images** to view the built image.

Publishing in the Console

1. Log in to the TKE console and click **Clusters** in the left sidebar.
2. Select the target cluster ID and go to the cluster management page of the Deployment to be created.
3. Click **Create** to go to the "Create a workload" page. See Creating a Deployment for the configuration of key parameters.
   In "Containers in the pod", choose **Select Image** > **My Images**. Then, select the image that was successfully uploaded during the preceding building process.
4. Click **Save** to finish creating the Deployment.
   On the **Pod Management** page, the nginx pod is running normally if the deployment was successful.

Related Operations: Batch Building Configuration

1. Log in to the Jenkins backend and click **System Management** in the left sidebar. Click **System Configuration** on the "Manage Jenkins" panel that appears.
2. On the "System Configuration" page, customize the "number of executors". This document uses 10 as an example.

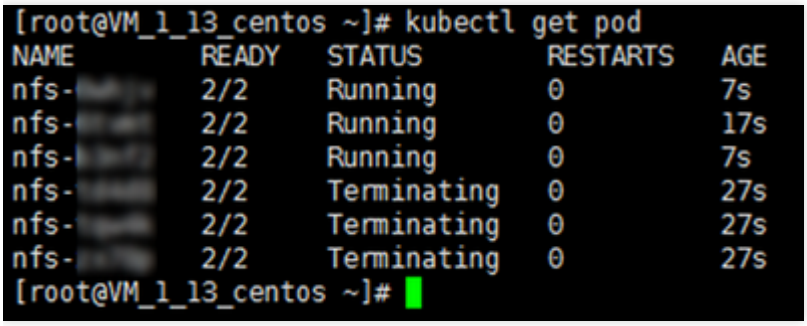
©2013-2022 Tencent Cloud. All rights reserved.

Note：
The number of executors is 10, indicating that 10 jobs can be executed at the same time.

3. For other configuration items, ensure that they are consistent with those in the Configuring the slave pod template step.
4. Create 10 tests by referring to the Slave pod building configuration step, as shown in the following figure.
5. Configure building for multiple tasks by referring to the Building Configuration step.
6. After the building is completed successfully, you can log in to the node and query the job pod by running the following command.

```
kubectl get pod
```

If the result similar to the following is returned, the call was successful.



©2013-2022 Tencent Cloud. All rights reserved.

Using Docker as an image building service in a containerd cluster

Last updated：2021-01-11 10:20:30

Overview

In a Kubernetes cluster, some CI/CD workflow may need to use Docker to provide image packaging services. This can be implemented by the Docker of the host. Mount Docker's UNIX Socket (`/var/run/docker.sock`) as the hostPath to the CI/CD service Pod, and then call the Docker of the host through the UNIX Socket to build image in the container. This method is simple and can save more resources than running a Docker host inside of another Docker host (Docker in Docker). However, this method may encounter the following problems:

- It cannot be performed in a cluster whose Runtime is containerd.
- If it is not controlled, it may overwrite the existing image on the node.
- When you need to modify the Docker Daemon configuration file, it may affect other services.
- It is not safe in the multi-tenancy scenario. After the privileged Pod obtains the UNIX Socket of Docker, the container in the Pod can not only call the host's Docker to build the image, delete the existing image or container, or even operate other containers through `docker exec` interface.

For the first problem above, Kubernetes has officially announced that Docker will be disused after version 1.22. These users may switch their service to containerd. For some clusters that require a containerd, and still use Docker to build the image without changing the CI/CD service process, you can add the DinD container to the original Pod as a sidecar or use DaemonSet to deploy the Docker service dedicated to building the image on the node.
This document describes the following two ways to use Docker to build images on the CI/CD workflow:

- Using DinD as the Sidecar of Pod
- Using DaemOnset to deploy Docker on each Containerd node

Directions

Using DinD as the Sidecar of Pod

For the implementation principle of DinD (Docker in Docker), see DinD Official Document. The following example shows that adding a Sidecar to clean-ci container, and combined with emptyDir, making the clean-ci container can access the DinD container through UNIX sockets.

©2013-2022 Tencent Cloud. All rights reserved.

```
apiVersion: v1
kind: Pod
metadata:
name: clean-ci
spec:
containers:
- name: dind
image: 'docker:stable-dind'
command:
- dockerd
- --host=unix:///var/run/docker.sock
- --host=tcp://0.0.0.0:8000
securityContext:
privileged: true
volumeMounts:
- mountPath: /var/run
name: cache-dir
- name: clean-ci
image: 'docker:stable'
command: ["/bin/sh"]
args: ["-c", "docker info >/dev/null 2>&1; while [ $? -ne 0 ] ; do sleep 3; docke
r info >/dev/null 2>&1; done; docker pull library/busybox:latest; docker save -o
busybox-latest.tar library/busybox:latest; docker rmi library/busybox:latest; whi
le true; do sleep 86400; done"]
volumeMounts:
- mountPath: /var/run
name: cache-dir
volumes:
- name: cache-dir
emptyDir: {}
```

Using DaemOnset to deploy Docker on each containerd node

This method is simple. You just need to directly forward the DaemonSet in the containerd cluster (mounting hostPath). In order not to affect the `/var/run` path on the node, you can specify other paths.

1. Use the following YAML to deploy DaemonSet, as shown below:

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
name: docker-ci
spec:
selector:
matchLabels:
app: docker-ci
```

©2013-2022 Tencent Cloud. All rights reserved.

```
template:
metadata:
labels:
app: docker-ci
spec:
containers:
- name: docker-ci
image: 'docker:stable-dind'
command:
- dockerd
- --host=unix:///var/run/docker.sock
- --host=tcp://0.0.0.0:8000
securityContext:
privileged: true
volumeMounts:
- mountPath: /var/run
name: host
volumes:
- name: host
hostPath:
path: /var/run
```

2. Share the same hostPath between the service Pod and DaemonSet, as shown below:

```
apiVersion: v1
kind: Pod
metadata:
name: clean-ci
spec:
containers:
- name: clean-ci
image: 'docker:stable'
command: ["/bin/sh"]
args: ["-c", "docker info >/dev/null 2>&1; while [ $? -ne 0 ] ; do sleep 3; doc
ker info >/dev/null 2>&1; done; docker pull library/busybox:latest; docker save
-o busybox-latest.tar library/busybox:latest; docker rmi library/busybox:lates
t; while true; do sleep 86400; done"]
volumeMounts:
- mountPath: /var/run
name: host
volumes:
- name: host
hostPath:
path: /var/run
```

©2013-2022 Tencent Cloud. All rights reserved.

Deploying Jenkins on TKE

Last updated：2023-05-06 17:36:46

Overview

Many DevOps requirements need to be implemented with Jenkins. This document describes how to deploy Jenkins in TKE.

Prerequisites

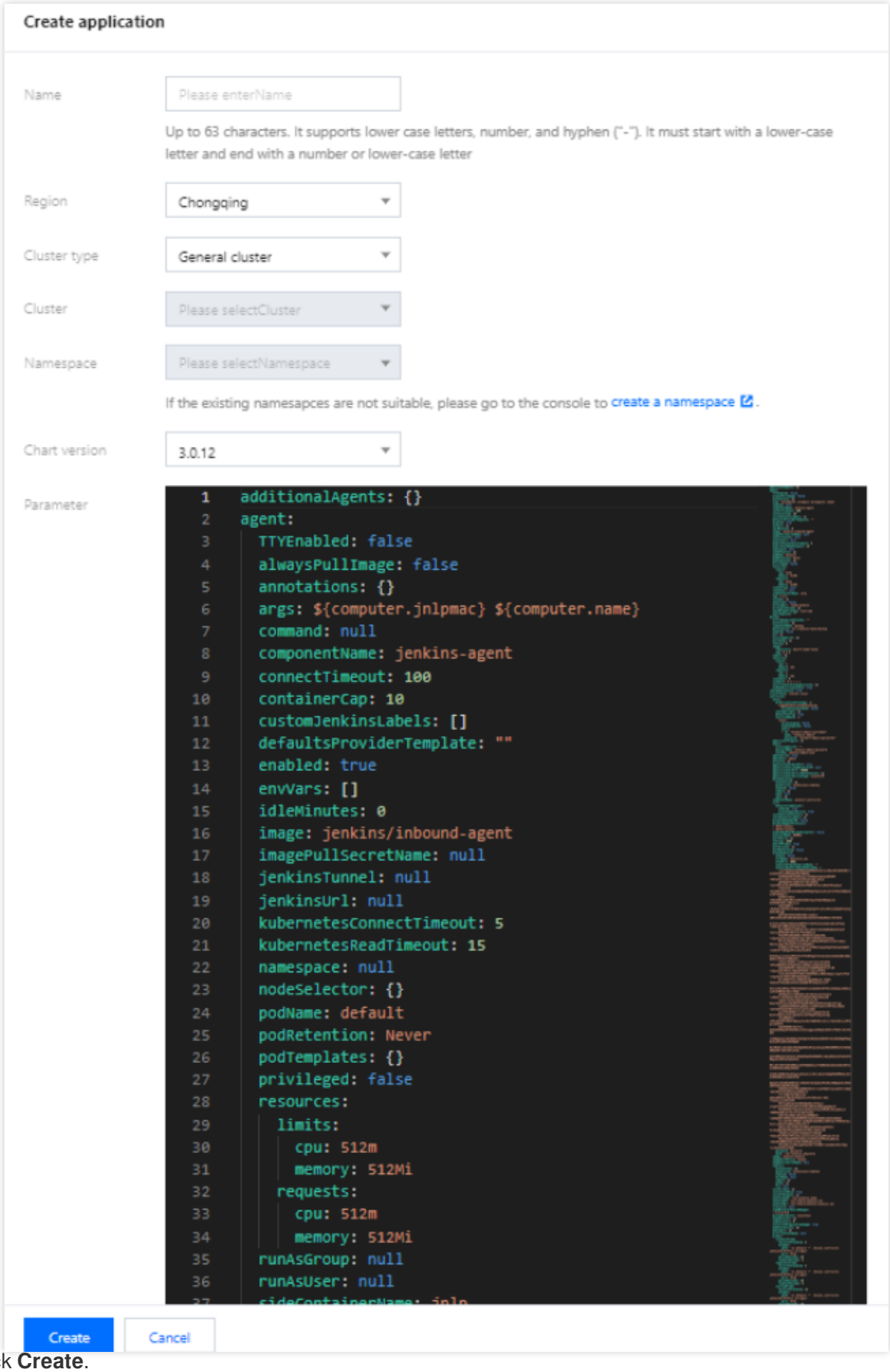
You have created a TKE cluster.

Directions

Installing Jenkins

1. Log in to the TKE console and click Marketplace in the left sidebar.
2. On the **Marketplace** page, search for `Jenkins` and go to the Jenkins application page.
3. Click **Create application** and configure `values.yaml` in **Parameter** as needed.

©2013-2022 Tencent Cloud. All rights reserved.



4. Click **Create**.

©2013-2022 Tencent Cloud. All rights reserved.

Exposing Jenkins UI

By default, you cannot access the Jenkins UI outside the cluster. To access the Jenkins UI, you can use an Ingress. TKE provides CLB-type Ingress and Nginx-type Ingress for your choice.

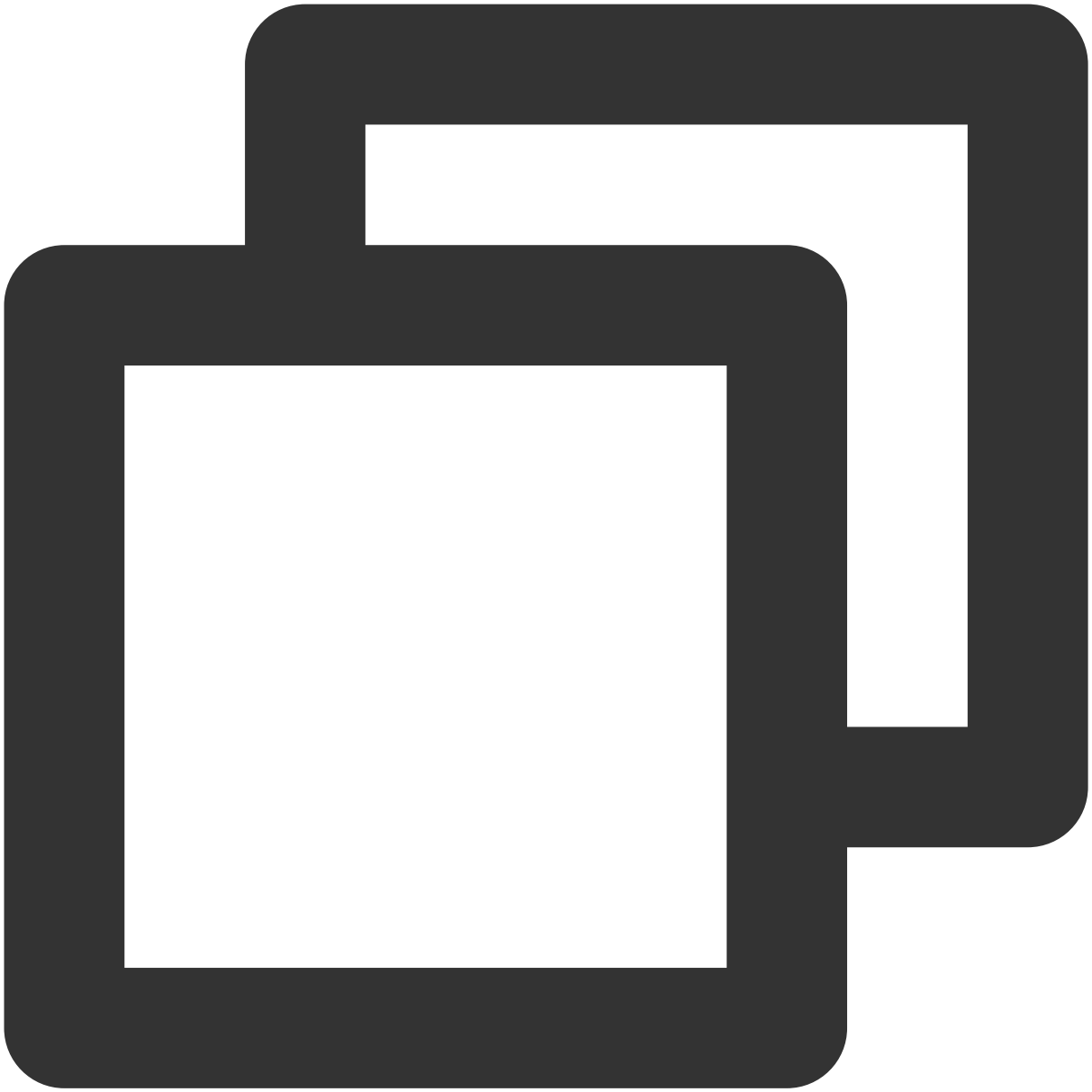
Note

Jenkins v2.263 is used in the following sample. The UI of Jenkins varies with the product version. You can select a version based on your business needs.

Logging in to Jenkins

On the Jenkins UI, enter the initial username and password to log in to the Jenkins backend. The username is `admin` , and the password can be obtained by running the following command.

©2013-2022 Tencent Cloud. All rights reserved.



```
kubectl -n devops get secret jenkins -o jsonpath='{.data.jenkins-admin-password}' |
```

Note

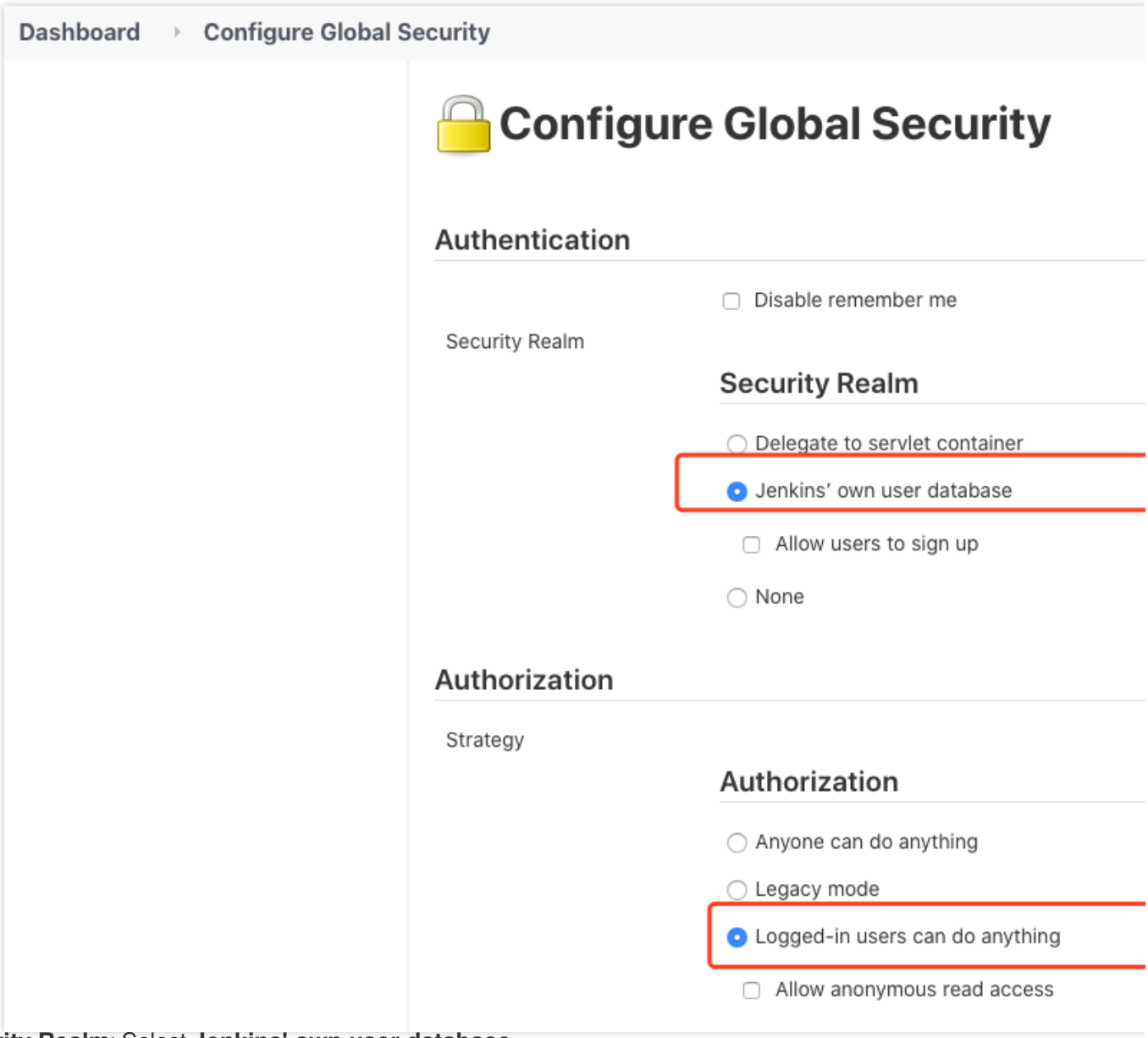
When running the above command, specify your actual namespace.

Creating a user

We recommend you manage Jenkins as a general user. Before creating a general user, you need to configure an authentication and authorization policy.

©2013-2022 Tencent Cloud. All rights reserved.

1. Log in to the Jenkins backend and choose **Dashboard** > **Manage Jenkins** > **Security** > **Configure Global Security** to go to the authentication and authorization policy page as shown below:

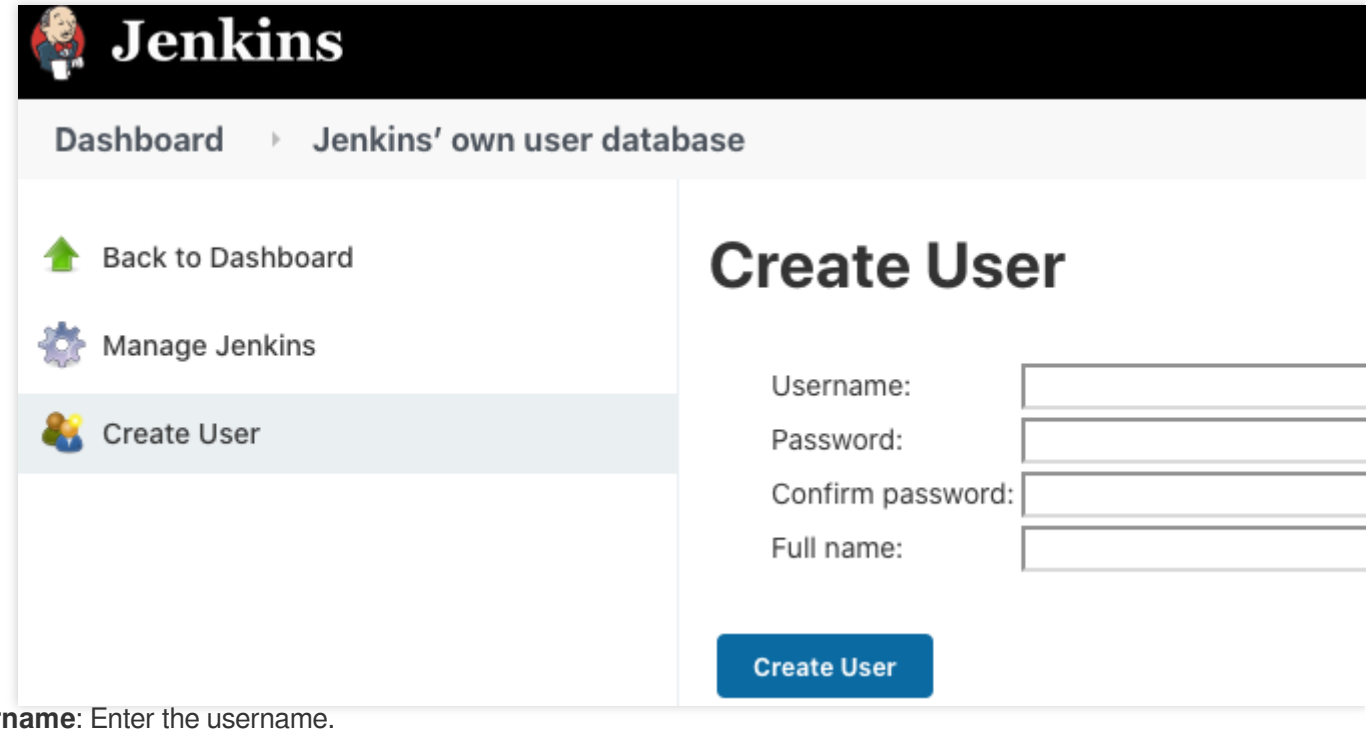


Security Realm: Select **Jenkins' own user database**.

Authorization: Select **Logged-in users can do anything**.

2. Choose **Dashboard** > **Manage Jenkins** > **Security** > **Manage Users** > **Create User** and create a user as prompted as shown below:

©2013-2022 Tencent Cloud. All rights reserved.



Username: Enter the username.

Password: Enter the password.

Confirm password: Confirm the password.

Full name: Enter the full username.

3. Click **Create User**.

Installing the plugin

Log in to the Jenkins backend and choose **Dashboard** > **Manage Jenkins** > **System Configuration** > **Manage Plugins** to go to the plugin management page.

You can install the following commonly used plugins:

Kubernetes

Pipeline

Git

GitLab

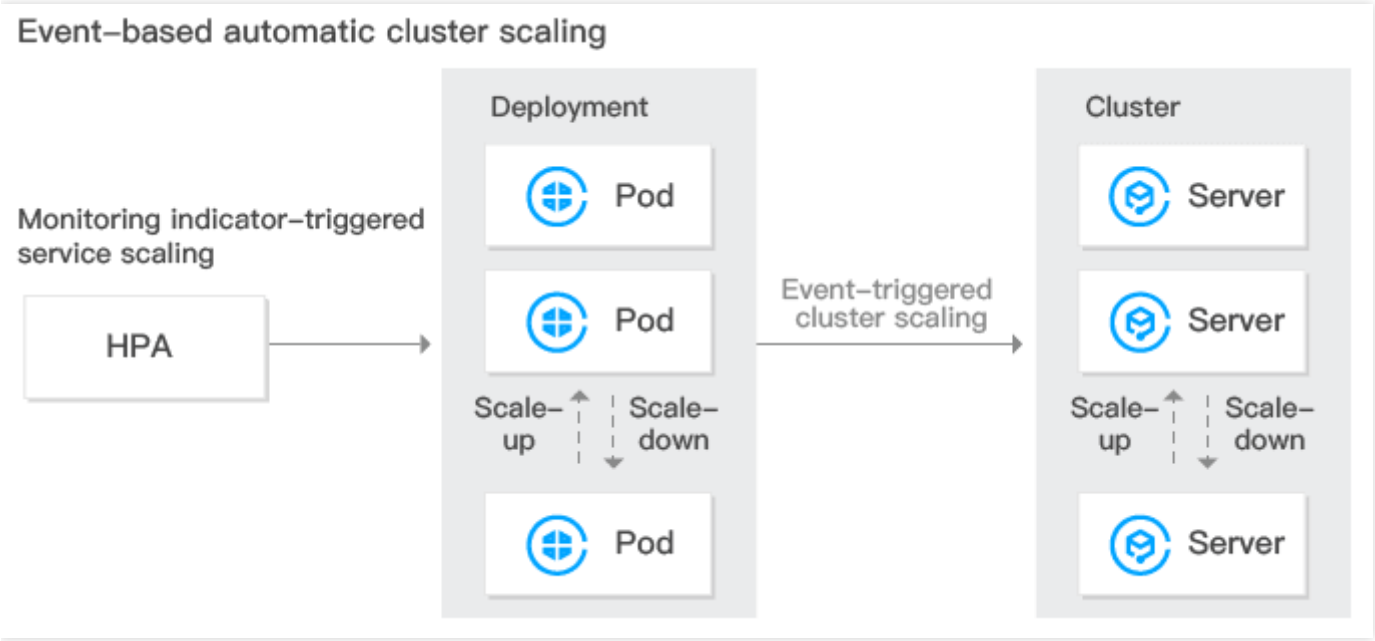
GitHub

©2013-2022 Tencent Cloud. All rights reserved.

Auto Scaling
Cluster Auto Scaling Practices

Last updated：2019-09-25 15:42:29

Tencent Kubernetes Engine (TKE) provides elastic scalability at cluster and service levels. It can monitor the metrics of a container including CPU, memory, and bandwidth and perform auto scaling. At the same time, clusters can be auto scaled if a container does not have sufficient resources or has more resources than necessary. Please see the figure below:



Cluster Auto Scaling Features

TKE allows users to enable auto scaling for clusters, helping users manage their computing resources efficiently. Users can set scaling policies based on their needs. Cluster auto scaling has the following features:

- Cluster auto scaling can dynamically and automatically create and release Cloud Virtual Machines (CVMs) in real time based on the project load situation to help users cope with project situation with the optimal number of instances. No human intervention is needed throughout the whole process, freeing users from manual deployment.
- Cluster auto scaling can help users handle project situation with the optimal amount of node resources. When there are more needs, it seamlessly and automatically adds CVMs to container clusters. When there are fewer needs, it automatically removes unnecessary CVMs to increase device ultilization and reduce the costs of deployment and instances.

Cluster Auto Scaling Feature Description

©2013-2022 Tencent Cloud. All rights reserved.

Basic Features of Kubernetes Cluster Auto Scaling

- Supports setting multiple scaling groups.
- Supports setting scale-in and scale-out policies. For more information, see Cluster Autoscaler.

Advanced TKE Scaling Group Features

- Supports using custom models while creating the scaling groups (recommended).
- Supports using a node in a cluster as a template while creating a scaling group.
- Supports adding spot instances to scaling groups (recommended).
- Supports automatically matching an appropriate scaling group when a model is sold out.
- Supports configuring scaling groups across availability zones.

Cluster Auto Scaling Restrictions

- The number of nodes that can be added by cluster auto scaling is limited by the VPC, container network, TKE cluster node quota, and the quota of CVMs that can be purchased.
- Whether nodes can be scaled out depends on whether the model you want to use is still available. If the model is sold out, nodes cannot be scaled out. It is recommended to configure multiple scaling groups.
- You need to configure the `request` value of the container under the workload. With the `request` value, whether the resources in the cluster are sufficient can be assessed in order to decide whether to trigger automatic scale-out.
- It is not recommended to enable monitoring metric-based auto scaling of nodes.
- Deleting a scaling group will also terminate the CVM instances in it. Please be cautious when doing so.

Configuring Cluster Scaling Groups

- Configuring multiple scaling groups (recommended)
  When there are multiple scaling groups in a cluster, the auto scaling component will select a scaling group for scale-out according to the scaling algorithm you select. The component will only select one scaling group each time. If it fails to scale out the target scaling group for reasons such as CVM model sold-out, the scaling group will be put to sleep for a period of time. At the same time, the second matching scaling group will be selected for scale-out.
  - Random: select a random scaling group for scale-out.
  - Most-Pods: select the scaling group that can schedule the most Pods based on the pending Pods and the models you select for the scaling groups.
  - Least-waste: select the scaling group that can ensure the fewest remaining resources after Pod scheduling based on the pending Pods and the models you select for the scaling groups.

©2013-2022 Tencent Cloud. All rights reserved.

It is recommended to configure multiple scaling groups with different models in the cluster, so as to prevent the scaling failures caused by model sold-out. At the same time, you can use a combination of spot instances and normal instances to reduce costs.

- Configuring a single scaling group
  If you only want to use one specific model for cluster scale-out, we recommend that you configure the scaling group to multiple subnets and availability zones.

©2013-2022 Tencent Cloud. All rights reserved.

Using tke-autoscaling-placeholder to Implement Auto Scaling in Seconds

Last updated：2023-05-23 16:33:44

Operation Scenarios

If a TKE cluster is configured with a node pool and enables Auto Scaling, automatic node scale-out (automatically purchasing of devices and adding them to the cluster) can be triggered when node resources are insufficient. This scale-out process takes some time and may be too slow to ensure normal business operations in some scenarios with sudden traffic increases. `tke-autoscaling-placeholder` can be used to implement scale-out on TKE in seconds, which is suitable for scenarios with sudden traffic increases. This document introduces how to use `tke-autoscaling-placeholder` to implement Auto Scaling in seconds.

How It Works

`tke-autoscaling-placeholder` utilizes low-priority pods to preemptively occupy resources (pause containers with request, consuming only a small amount of resources), reserving some resources as a buffer for high-priority businesses prone to sudden traffic spikes. When pod scale-out is needed, high-priority pods will quickly occupy the resources of low-priority pods for scheduling. In this case, the low-priority pods of `tke-autoscaling-placeholder` will change to the Pending status. If you have configured a node pool and enabled Auto Scaling, node scale-out will be triggered. As some resources are used as a buffer, even if the node scale-out process is slow, some pods can still be quickly scaled out and scheduled, achieving scaling in seconds. You can adjust the amount of resources reserved as the buffer by adjusting request in `tke-autoscaling-placeholder` or the number of replicas based on your needs.

Use limits

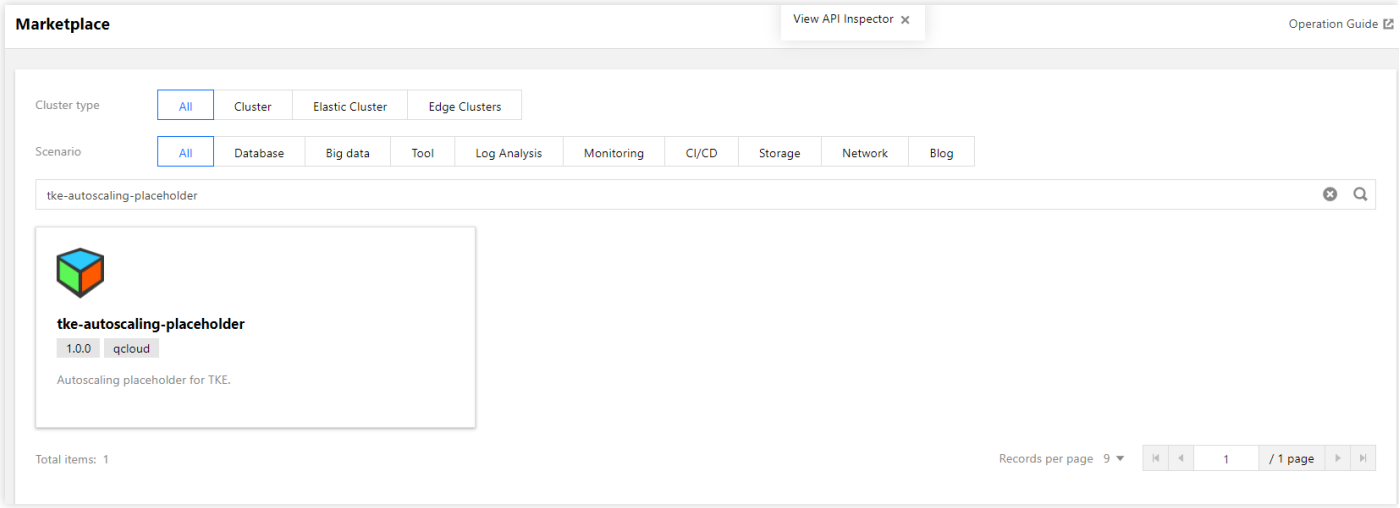
To use the `tke-autoscaling-placeholder` app, the cluster version must be later than 1.18.

Directions

Installing tke-autoscaling-placeholder

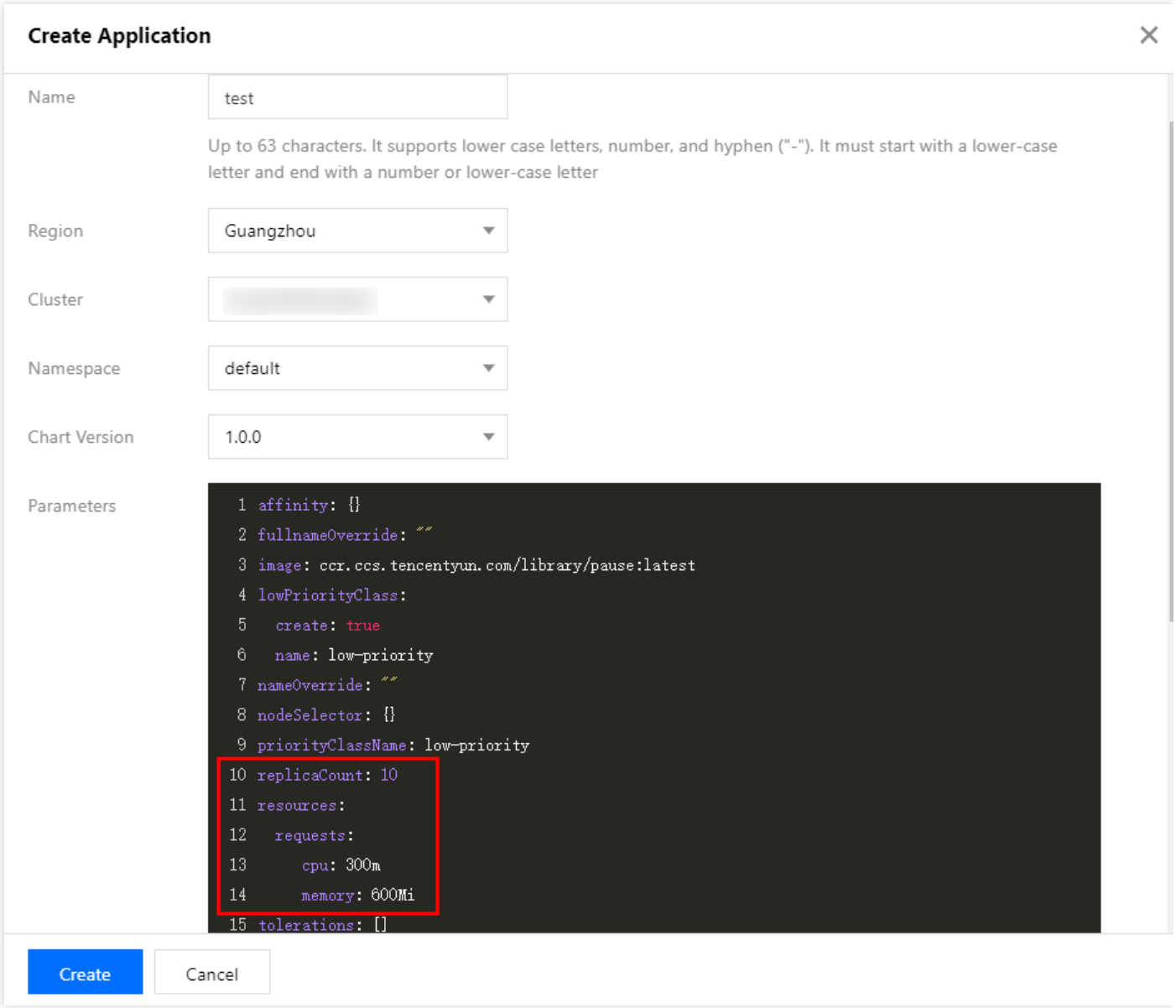
©2013-2022 Tencent Cloud. All rights reserved.

1. Log in to the TKE console.
2. In the left sidebar, click **App Market** to go to the "App Market" management page.
3. In the search box of the "App Market" page, enter `tke-autoscaling-placeholder` to search for the app, as shown in the figure below:



4. On the "App Details Page", click **Create an App** in the "Basic Information" module.
5. In the "Create an App" window that pops up, configure and create an app based on your needs, as shown in the figure below:

©2013-2022 Tencent Cloud. All rights reserved.



Configuration instructions:

- **Name**: enter the app name. It can contain up to 63 characters, including lowercase letters, numbers, and hyphens ("-"). It must begin with a lowercase letter and end with a number or lowercase letter.
- **Region**: select the region for deployment.
- **Cluster Type**: select **Standard Cluster**.
- **Cluster**: select the ID of the cluster for deployment.
- **Namespace**: select the namespace for deployment.
- **Chart Version**: select the chart version for deployment.
- **Parameters**: among the configuration parameters, the most important ones are `replicaCount` and `resources.request`, which indicate the number of replicas of `tke-autoscaling-placeholder` and the amount of resources occupied by each replica, respectively. They collectively determine the size of buffer

©2013-2022 Tencent Cloud. All rights reserved.

resources. You can set them based on the estimated amount of extra resources needed for sudden traffic increases.

For complete parameter configuration descriptions for `tke-autoscaling-placeholder`, see the following table:

Parameter Name	Description	Default Value
replicaCount	Number of placeholder replicas	10
image	placeholder image address	`ccr.ccs.tencentyun.com/library/pause:latest`
resources.requests.cpu	Amount of CPU resources occupied by a single placeholder replica	300m
resources.requests.memory	Size of memory occupied by a single placeholder replica	600Mi
lowPriorityClass.create	Whether to create a low PriorityClass (to be imported by placeholder)	true
lowPriorityClass.name	Name of the low PriorityClass	low-priority
nodeSelector	Specifies the node with a specific label to which placeholder will be scheduled.	{}
tolerations	Specifies the taint to be tolerated by placeholder.	[]
affinity	Specifies the affinity configuration of placeholder.	{}

6. Click **Create** to deploy the tke-autoscaling-placeholder app.
7. Run the following commands to check whether the pod for resource preemptive occupation starts successfully. Below is a sample:

```
$ kubectl get pod -n default
tke-autoscaling-placeholder-b58fd9d5d-2p6ww 1/1 Running 0 8s
tke-autoscaling-placeholder-b58fd9d5d-55jw7 1/1 Running 0 8s
tke-autoscaling-placeholder-b58fd9d5d-6rq9r 1/1 Running 0 8s
tke-autoscaling-placeholder-b58fd9d5d-7c95t 1/1 Running 0 8s
tke-autoscaling-placeholder-b58fd9d5d-bfg8r 1/1 Running 0 8s
tke-autoscaling-placeholder-b58fd9d5d-cfqt6 1/1 Running 0 8s
tke-autoscaling-placeholder-b58fd9d5d-gmfmr 1/1 Running 0 8s
tke-autoscaling-placeholder-b58fd9d5d-grwlh 1/1 Running 0 8s
```

©2013-2022 Tencent Cloud. All rights reserved.

```
tke-autoscaling-placeholder-b58fd9d5d-ph7vl 1/1 Running 0 8s
tke-autoscaling-placeholder-b58fd9d5d-xmrmv 1/1 Running 0 8s
```

Deploying a high-priority pod

By default, the priority of `tke-autoscaling-placeholder` is low. You can specify a high PriorityClass for its business pod to facilitate preemptive resource occupation and implement quick scale-out. If you have not yet created a PriorityClass, you can refer to the following sample to create one:

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
name: high-priority
value: 1000000
globalDefault: false
description: "high priority class"
```

In the business Pod, set `priorityClassName` to a high PriorityClass. Below is a sample:

```
apiVersion: apps/v1
kind: Deployment
metadata:
name: nginx
spec:
replicas: 8
selector:
matchLabels:
app: nginx
template:
metadata:
labels:
app: nginx
spec:
priorityClassName: high-priority # Specify a high PriorityClass here.
containers:
- name: nginx
image: nginx
resources:
requests:
cpu: 400m
MEM: 800Mi
```

When cluster node resources are insufficient, the scaled-out high-priority business pod can occupy the resources of low-priority pods of `tke-autoscaling-placeholder` and schedule the resources. At this time, the status of the

©2013-2022 Tencent Cloud. All rights reserved.

`tke-autoscaling-placeholder` pods changes to Pending. Below is a sample:

```
$ kubectl get pod -n default
NAME READY STATUS RESTARTS AGE
nginx-bf79bbc8b-5kxcw 1/1 Running 0 23s
nginx-bf79bbc8b-5xhbx 1/1 Running 0 23s
nginx-bf79bbc8b-bmzff 1/1 Running 0 23s
nginx-bf79bbc8b-l2vht 1/1 Running 0 23s
nginx-bf79bbc8b-q84jq 1/1 Running 0 23s
nginx-bf79bbc8b-tq2sx 1/1 Running 0 23s
nginx-bf79bbc8b-tqgxg 1/1 Running 0 23s
nginx-bf79bbc8b-wz5w5 1/1 Running 0 23s
tke-autoscaling-placeholder-b58fd9d5d-255r8 0/1 Pending 0 23s
tke-autoscaling-placeholder-b58fd9d5d-4vt8r 0/1 Pending 0 23s
tke-autoscaling-placeholder-b58fd9d5d-55jw7 1/1 Running 0 94m
tke-autoscaling-placeholder-b58fd9d5d-7c95t 1/1 Running 0 94m
tke-autoscaling-placeholder-b58fd9d5d-ph7vl 1/1 Running 0 94m
tke-autoscaling-placeholder-b58fd9d5d-qjrsx 0/1 Pending 0 23s
tke-autoscaling-placeholder-b58fd9d5d-t5qdm 0/1 Pending 0 23s
tke-autoscaling-placeholder-b58fd9d5d-tgvmw 0/1 Pending 0 23s
tke-autoscaling-placeholder-b58fd9d5d-xmrmv 1/1 Running 0 94m
tke-autoscaling-placeholder-b58fd9d5d-zxtwp 0/1 Pending 0 23s
```

If you have configured Auto Scaling for the node pool, node scale-out will be triggered. As the buffer resources have been allocated to the business pod, your business can be scaled out quickly. Therefore, despite the slow node speed, the normal running of your business is not affected.

Summary

This document introduces the `tke-autoscaling-placeholder` tool for implementing scaling in seconds. It takes advantage of pod priorities and the preemptive occupation feature to pre-deploy some low-priority "empty pods" to occupy resources, which become buffer resources. Then, in the event of a traffic spike that results in insufficient cluster resources, the resources of these low-priority "empty pods" can be occupied while triggering node scale-out at the same time. In this way, scaling can be implemented in seconds even in the case of resource shortages, and normal business operation will not be affected.

References

- Pod Priority and Preemption
- Creating a Node Pool

©2013-2022 Tencent Cloud. All rights reserved.

Installing metrics-server on TKE

Last updated：2021-08-17 15:32:42

Operation Scenarios

The metrics-server can realize the Resource Metrics API (metrics.k8s.io) of Kubernetes. Through this API, you can query some monitoring metrics of Pods and Nodes. The monitoring metrics of Pods are used in HPA, VPA, and `kubectl top pods` commands, whereas the Node metrics are currently used only in `kubectl top nodes` commands. TKE itself realizes the Resource Metrics API, pointed towards the hpa-metrics-server, and currently TKE also provides monitoring metrics for Pods.

After installing the metrics-server to the cluster, you can run `kubectl top nodes` to obtain the monitoring overview of nodes to replace the realization of the Resource Metrics API. HPA created on the TKE console does not use Resource Metrics and only uses Custom Metrics. Therefore, installing the metrics-server does not affect HPA created on the TKE console. This document describes how to install the metrics-server on TKE.

Directions

Downloading the YAML deployment file

Run the following commands to download the latest deployment file components.yaml of the metrics-server.

```
wget https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
```

Modifying the metrics-server launch parameter

The metrics-server requests the kubelet API of each node to obtain monitoring data. The API is exposed via HTTPS, but as TKE node kubelet uses a self-signed certificate, if the metrics-server directly requests the kubelet API, an error of certification verification failure will occur. Therefore, you need to add the `--kubelet-insecure-tls` launch parameter in the components.yaml file.
Moreover, as the official image repository of the metrics-server is stored in `k8s.gcr.io`, users in China may not be able to directly pull images from the repository. You need to manually synchronize images to CCR or use the synchronized image `ccr.ccs.tencentyun.com/mirrors/metrics-server:v0.4.0`.

Below is a sample of modification of the components.yaml file:

©2013-2022 Tencent Cloud. All rights reserved.

```
containers:
- args:
- --cert-dir=/tmp
- --secure-port=4443 # Please replace with 4443
- --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
- --kubelet-use-node-status-port
- --kubelet-insecure-tls # Add this launch parameter
image: ccr.ccs.tencentyun.com/mirrors/metrics-server:v0.4.0 # For cluster in the
Chinese mainland, please replace with this image address
ports:
- containerPort: 4443 # Please replace with 4443
name: https
protocol: TCP
```

Deploying the metrics-server

After modifying components.yaml, run the following commands to implement one-click deployment to the cluster via kubectl:

```
kubectl apply -f components.yaml
```

Note：
Through the above step, you can install and deploy the metrics-server. Alternatively, you can run the following commands for one-click installation of the metrics-server, but this method cannot ensure synchronization with the latest version.

```
kubectl apply -f https://raw.githubusercontent.com/TencentCloudContainerTeam/mani
fest/master/metrics-server/components.yaml
```

Checking the running status

1. Run the following commands to check whether the metrics-server starts normally. Below is a sample:

```
$ kubectl get pod -n kube-system | grep metrics-server
metrics-server-f976cb7d-8hssz 1/1 Running 0 1m
```

2. Run the following commands to check the configuration file. Below is a sample:

©2013-2022 Tencent Cloud. All rights reserved.

```
$ kubectl get --raw /apis/metrics.k8s.io/v1beta1 | jq
{
"kind": "APIResourceList",
"apiVersion": "v1",
"groupVersion": "metrics.k8s.io/v1beta1",
"resources": [
{
"name": "nodes",
"singularName": "",
"namespaced": false,
"kind": "NodeMetrics",
"verbs": [
"get",
"list"
]
},
{
"name": "pods",
"singularName": "",
"namespaced": true,
"kind": "PodMetrics",
"verbs": [
"get",
"list"
]
}
]
}
```

3. Run the following commands to check the node usage performance. Below is a sample:

```
$ kubectl top nodes
NAME CPU(cores) CPU% MEMORY(bytes) MEMORY%
test1 1382m 35% 2943Mi 44%
test2 397m 10% 3316Mi 49%
test3 81m 8% 464Mi 77%
```

©2013-2022 Tencent Cloud. All rights reserved.

Using Custom Metrics for Auto Scaling in TKE

Last updated：2023-04-07 20:04:01

Scenario

TKE supports many metrics for elastic scaling based on the custom metrics API, covering CPU, memory, disk, network, and GPU in most HPA scenarios. For more information on the list, see HPA Metrics. For complex scenarios such as automatic scaling based on the QPS per replica, you can install prometheus-adapter to implement auto scaling. Kubernetes provides the custom metrics API and external metrics API for HPA to perform auto scaling based on metrics, allowing users to customize auto scaling as needed. Prometheus-adapter supports the above two APIs. In the actual environment, the custom metrics API can meet most scenarios. This document describes how to use custom metrics for auto scaling through the custom metrics API.

Prerequisites

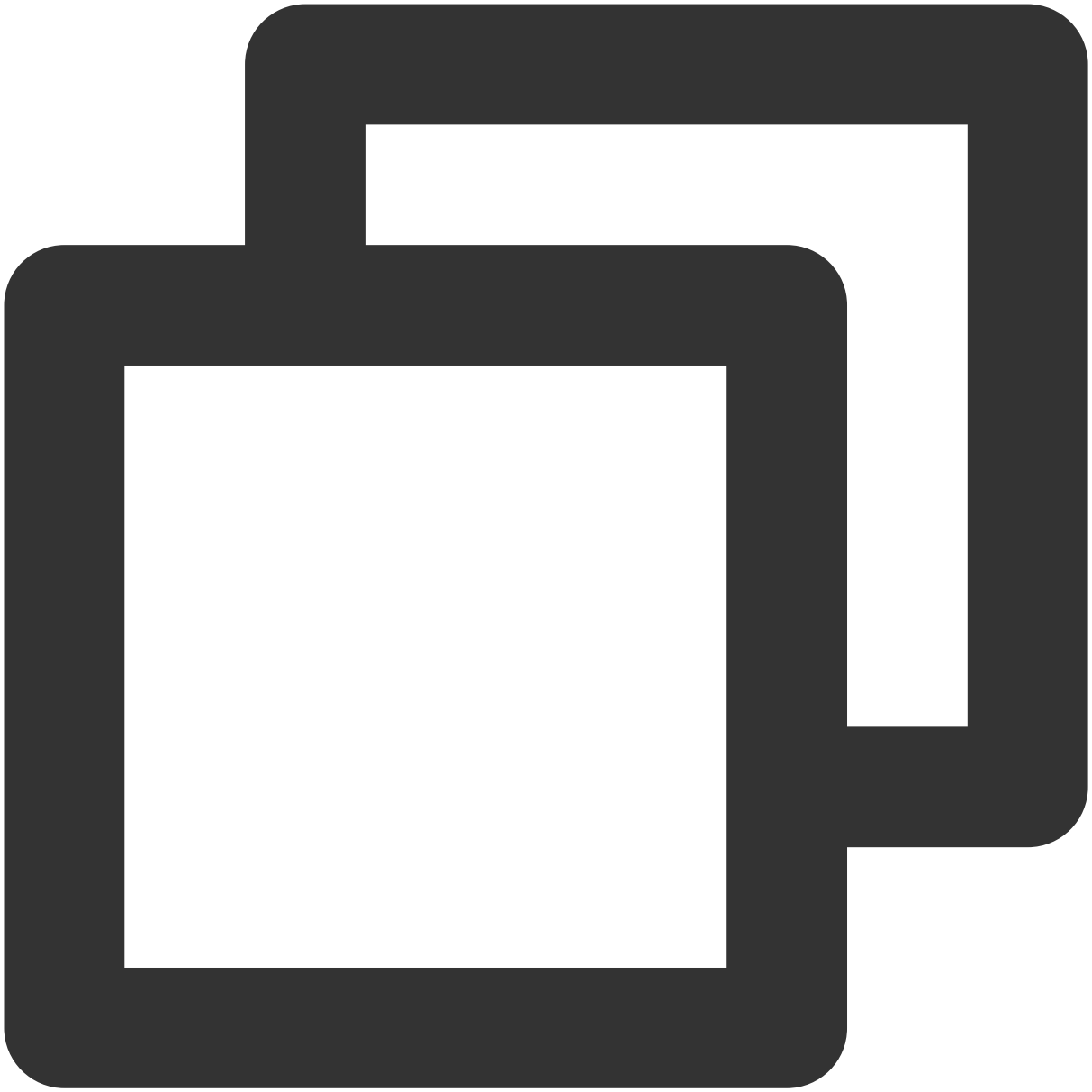
You have created a TKE cluster of v1.12 or later version. For more information, see Creating a Cluster.
You have deployed the PROM instance and collected the corresponding custom metrics.
You have installed Helm.

Directions

Opening the monitoring metric

This document takes the Golang service application as an example, which opens the `httpserver_requests_total` metric and records HTTP requests. This metric can be used to calculate the QPS value of the service application, as shown below:

©2013-2022 Tencent Cloud. All rights reserved.



```
package main

import (
        "github.com/prometheus/client_golang/prometheus"
        "github.com/prometheus/client_golang/prometheus/promhttp"
        "net/http"
        "strconv"
)

var (
HTTPRequests = prometheus.NewCounterVec(
```

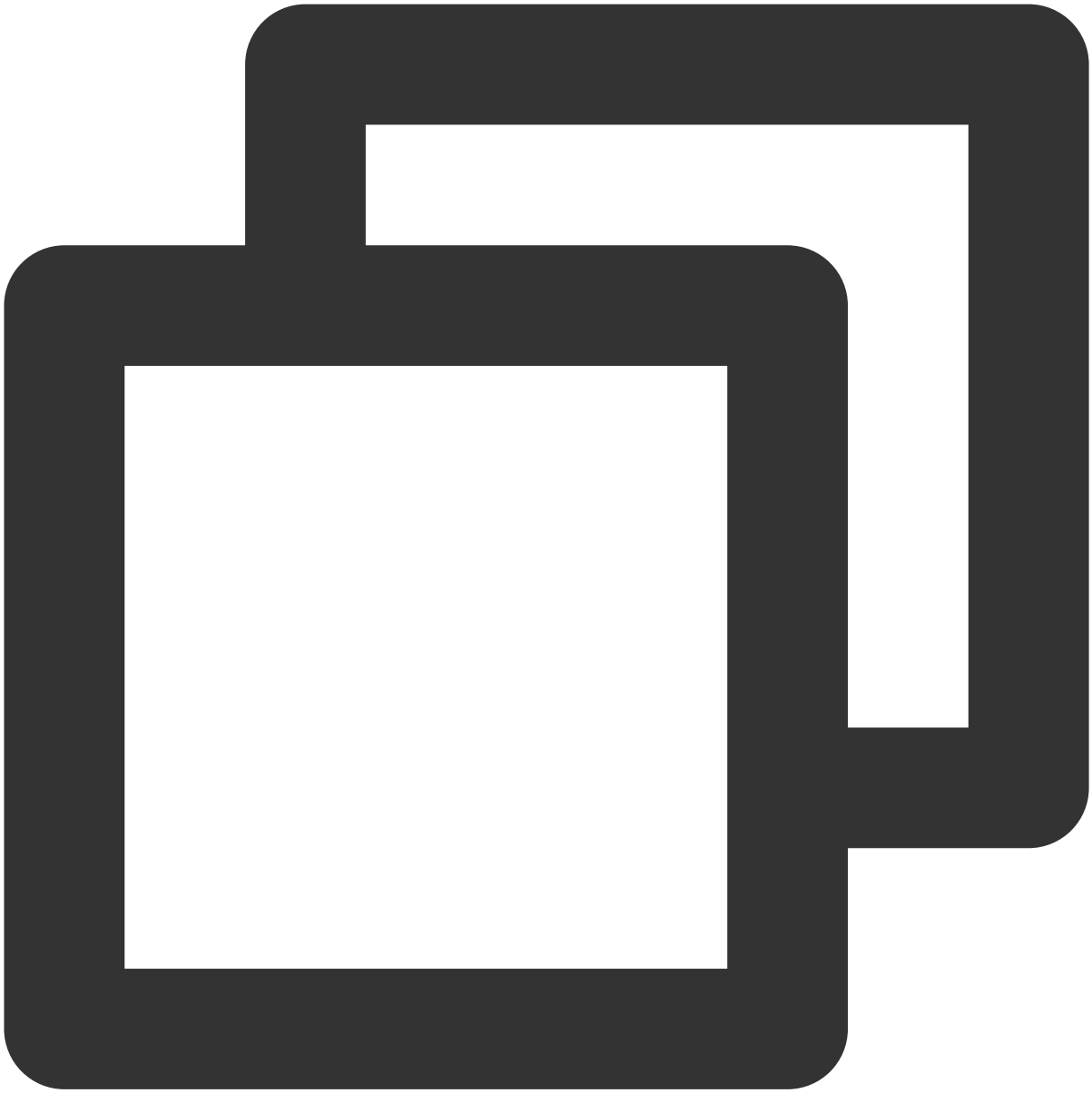
©2013-2022 Tencent Cloud. All rights reserved.

```
        prometheus.CounterOpts{
            Name: "httpserver_requests_total",
            Help: "Number of the http requests received since the server started",
        },
        []string{"status"},
    )
)

func init() {
    prometheus.MustRegister(HTTPRequests)
}

func main(){
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        path := r.URL.Path
        code := 200
        switch path {
        case "/test":
            w.WriteHeader(200)
            w.Write([]byte("OK"))
        case "/metrics":
            promhttp.Handler().ServeHTTP(w, r)
        default:
            w.WriteHeader(404)
            w.Write([]byte("Not Found"))
        }
        HTTPRequests.WithLabelValues(strconv.Itoa(code)).Inc()
    })
    http.ListenAndServe(":80", nil)
}
```

Deploying the service application

Package the application into a container image and deploy it to the cluster. Take the Deployment mode as an example:

©2013-2022 Tencent Cloud. All rights reserved.



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: httpserver
  namespace: httpserver
spec:
  replicas: 1
  selector:
    matchLabels:
      app: httpserver
  template:
```

©2013-2022 Tencent Cloud. All rights reserved.

```
    metadata:
      labels:
        app: httpserver
    spec:
      containers:
      - name: httpserver
        image: registry.imroc.cc/test/httpserver:custom-metrics
        imagePullPolicy: Always

---

apiVersion: v1
kind: Service
metadata:
  name: httpserver
  namespace: httpserver
  labels:
    app: httpserver
  annotations:
    prometheus.io/scrape: "true"
    prometheus.io/path: "/metrics"
    prometheus.io/port: "http"
spec:
  type: ClusterIP
  ports:
  - port: 80
    protocol: TCP
    name: http
  selector:
    app: httpserver
```

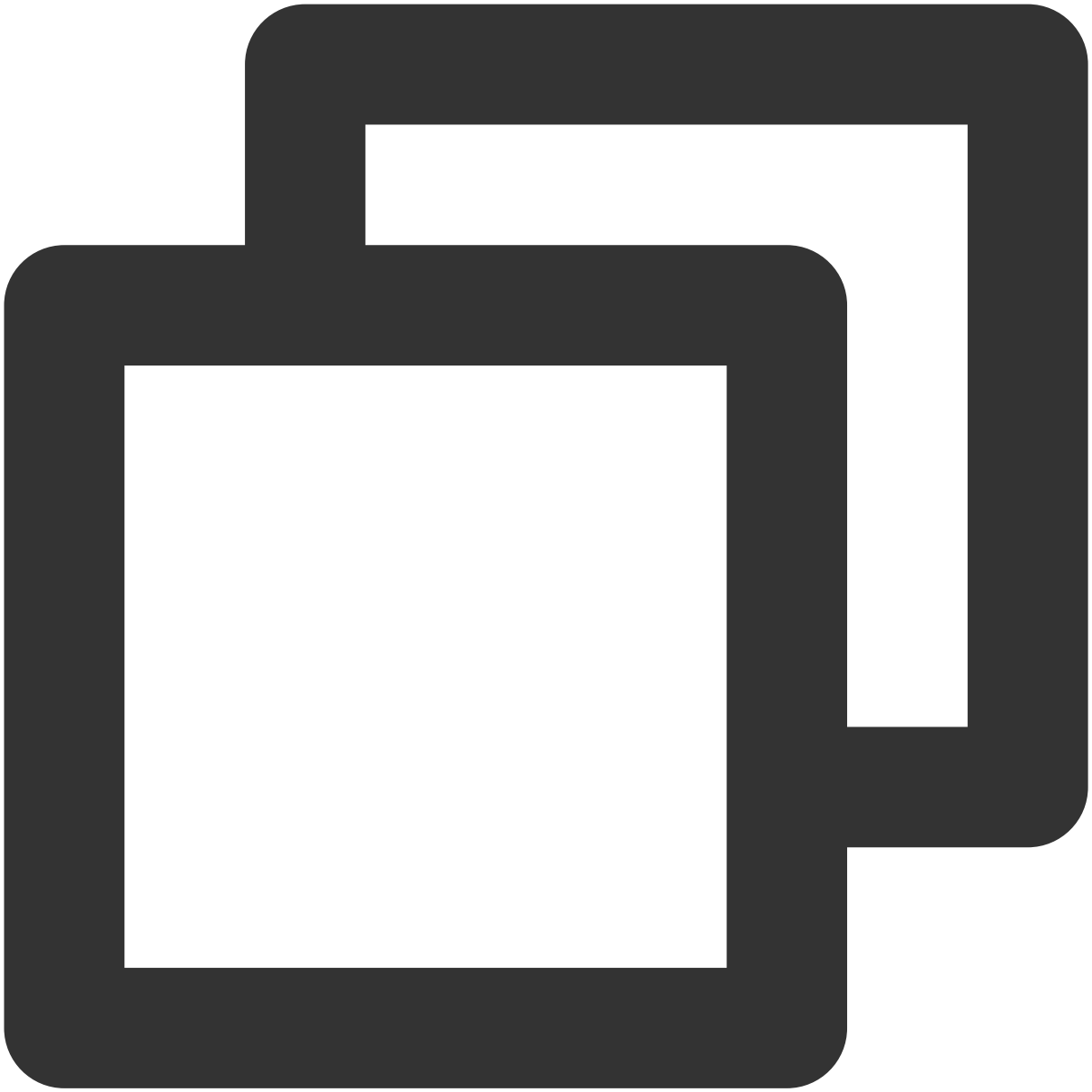
Collecting service monitoring metrics through PROM instance

You can configure a PROM instance to collect the monitoring metrics opened by the service through PROM Instance Collection Rules or ServiceMonitor.

Method 1: Configuring PROM instance collection rules

Add the following collection rules to the configuration file of PROM instance collection rule, as shown below:

©2013-2022 Tencent Cloud. All rights reserved.



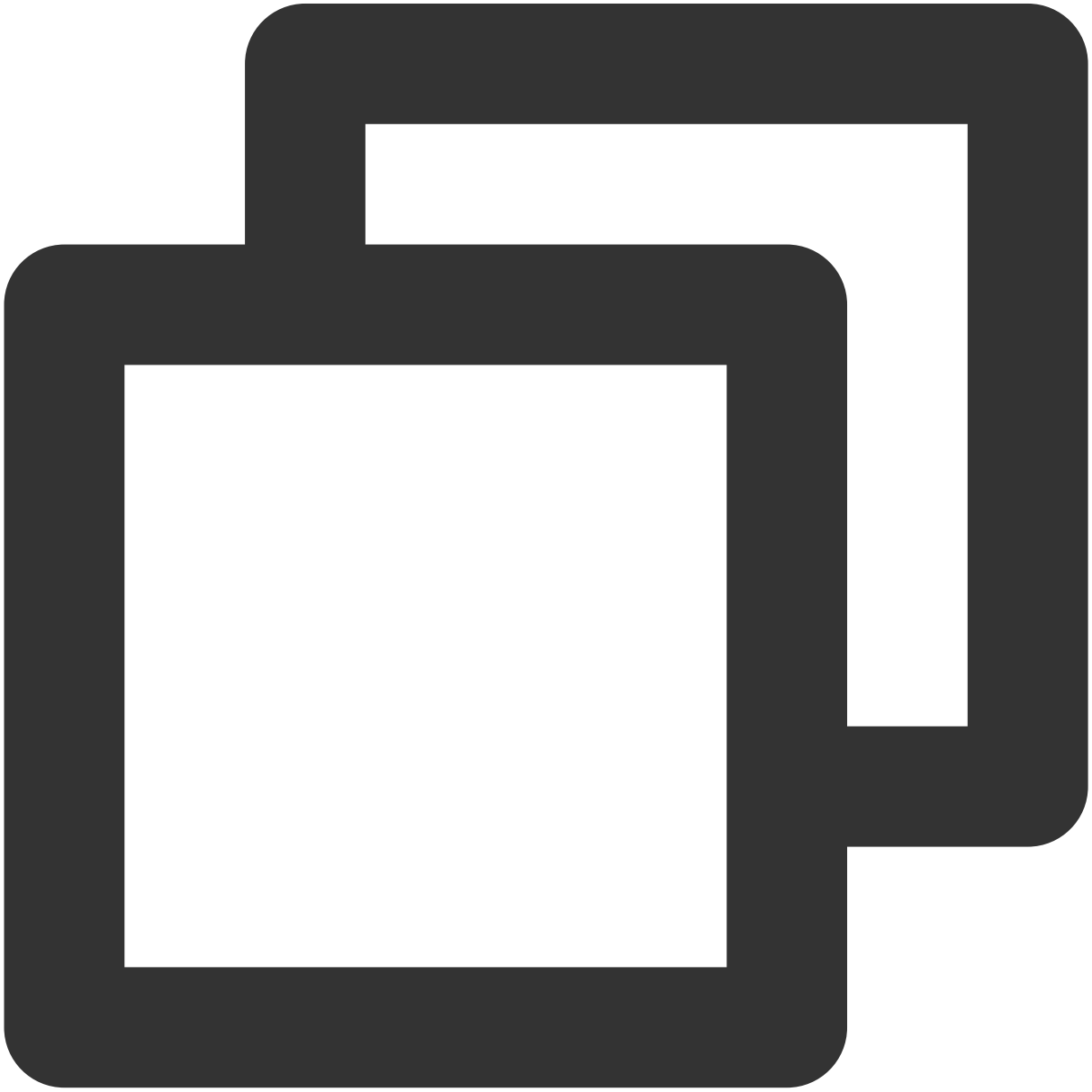
```
- job_name: httpserver
    scrape_interval: 5s
    kubernetes_sd_configs:
    - role: endpoints
        namespaces:
            names:
            - httpserver
    relabel_configs:
    - action: keep
        source_labels:
        - __meta_kubernetes_service_label_app
```

©2013-2022 Tencent Cloud. All rights reserved.

```
            regex: httpserver
        - action: keep
            source_labels:
            - __meta_kubernetes_endpoint_port_name
            regex: http
```

Method 2: Configuring ServiceMonitor

If prometheus-operator has been installed, you can create a CRD object of the ServiceMonitor to configure PROM instance, as shown below:



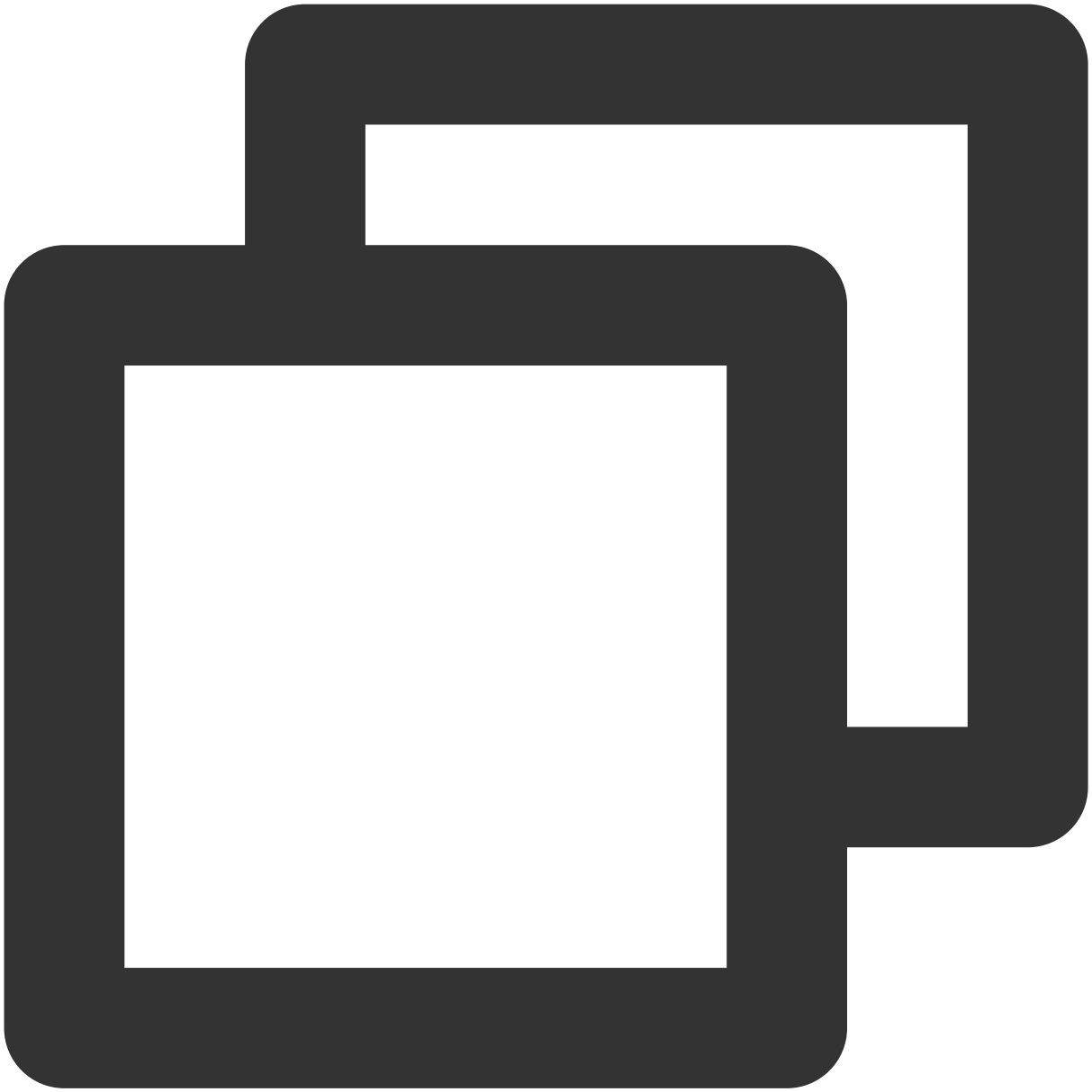
©2013-2022 Tencent Cloud. All rights reserved.

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: httpserver
spec:
  endpoints:
  - port: http
    interval: 5s
  namespaceSelector:
    matchNames:
    - httpserver
  selector:
    matchLabels:
      app: httpserver
```

Installing prometheus-adapter

1. Use Helm to install prometheus-adapter. Please confirm and configure custom metrics before installation. According to the example in Opening the monitoring metric above, the `httpserver_requests_total` metric is used in the service to record HTTP requests, so you can calculate the QPS of each service Pod through the following PromQL, as shown below:

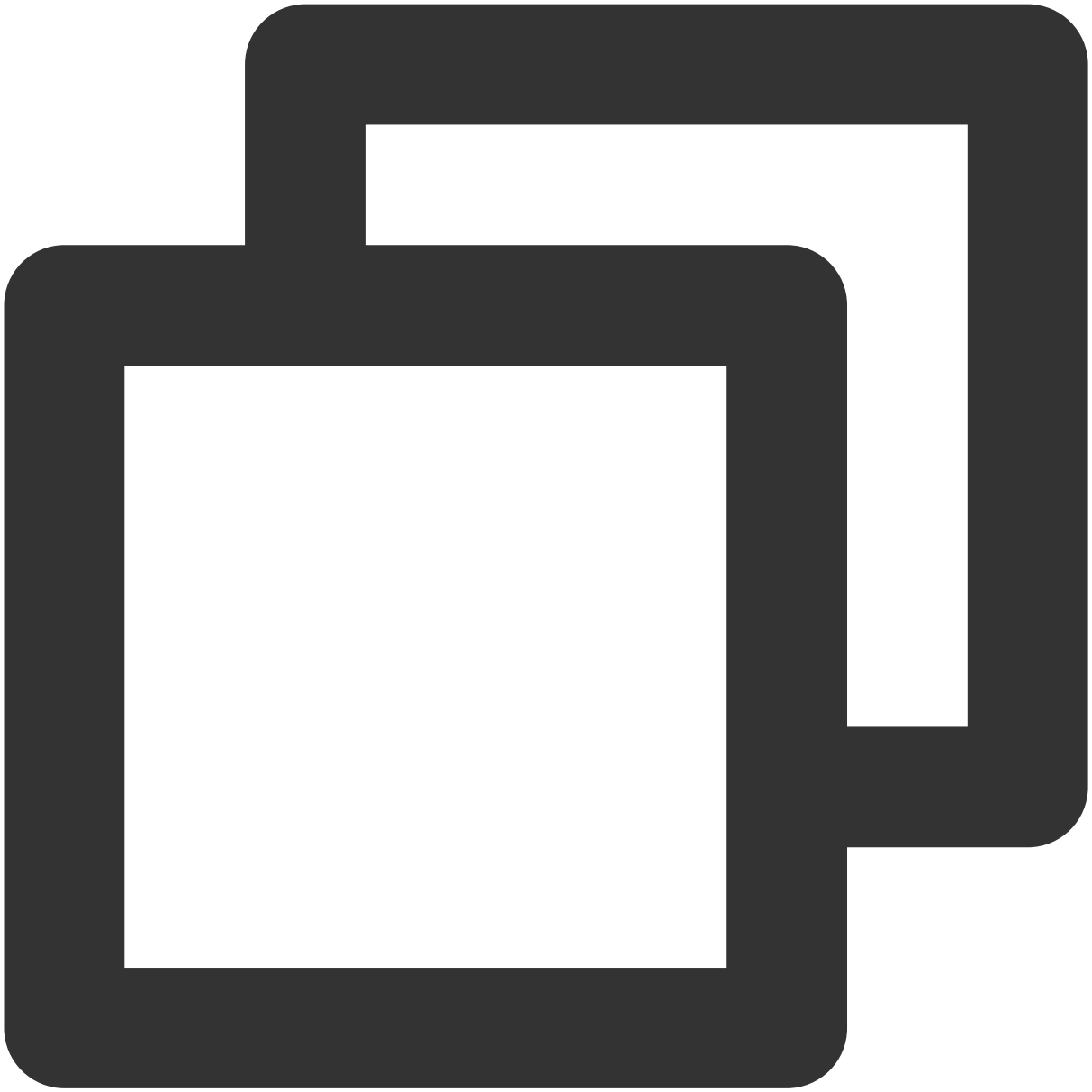
©2013-2022 Tencent Cloud. All rights reserved.



```
sum(rate(http_requests_total[2m])) by (pod)
```

2. Convert it to the configuration of prometheus-adapter. Create `values.yaml` with the following content:

©2013-2022 Tencent Cloud. All rights reserved.



```
rules:
   default: false
   custom:
   - seriesQuery: 'httpserver_requests_total'
     resources:
       template: <<.Resource>>
     name:
       matches: "httpserver_requests_total"
       as: "httpserver_requests_qps" # QPS metric calculated by PromQL
     metricsQuery: sum(rate(<<.Series>>{<<.LabelMatchers>>}[1m])) by (<<.GroupBy>>)
prometheus:
```

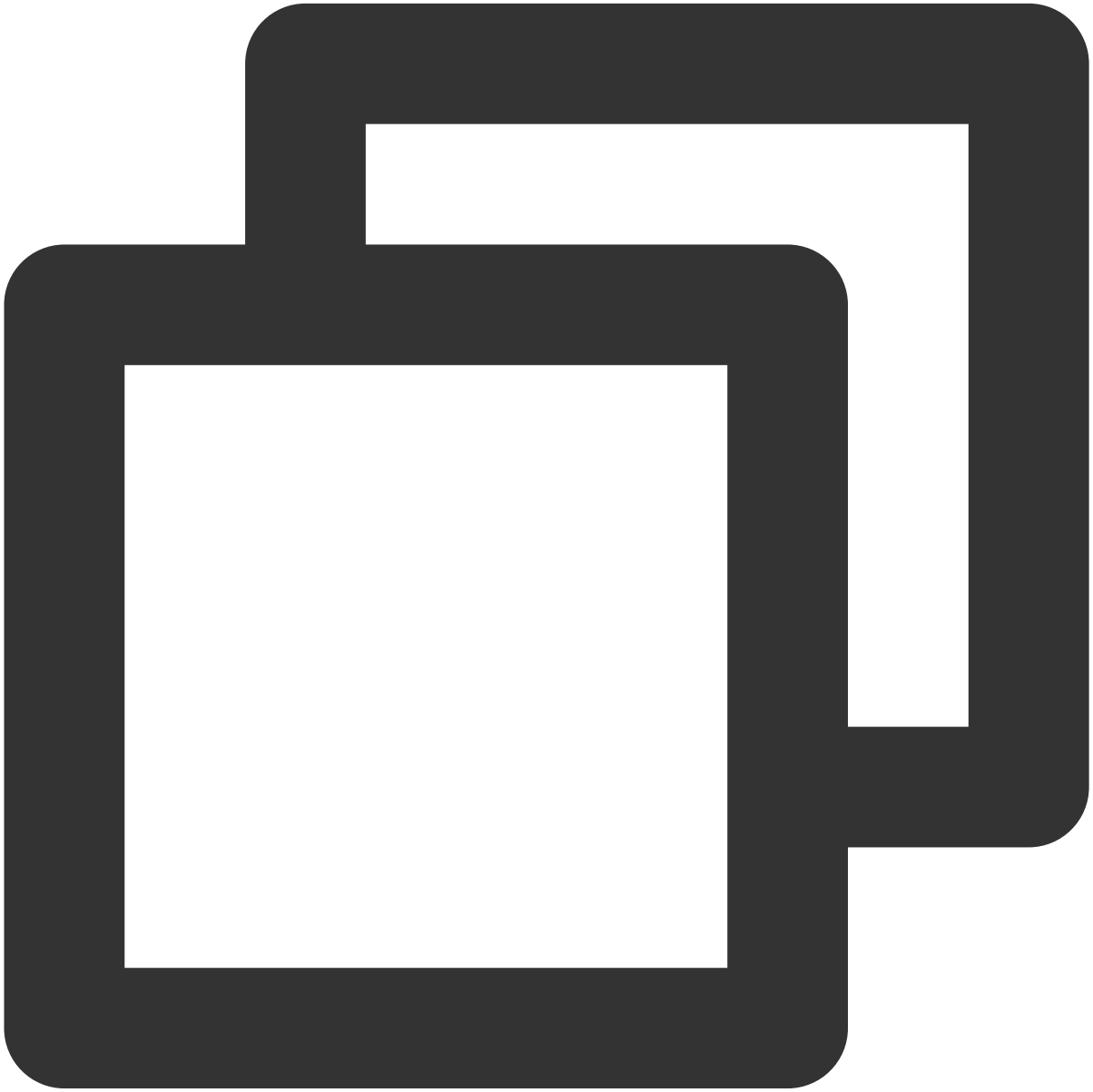
©2013-2022 Tencent Cloud. All rights reserved.

```
    url: http://prometheus.monitoring.svc.cluster.local # Replace PROM instance API
    port: 9090
```

3. Run the following Helm command to install prometheus-adapter, as shown below:

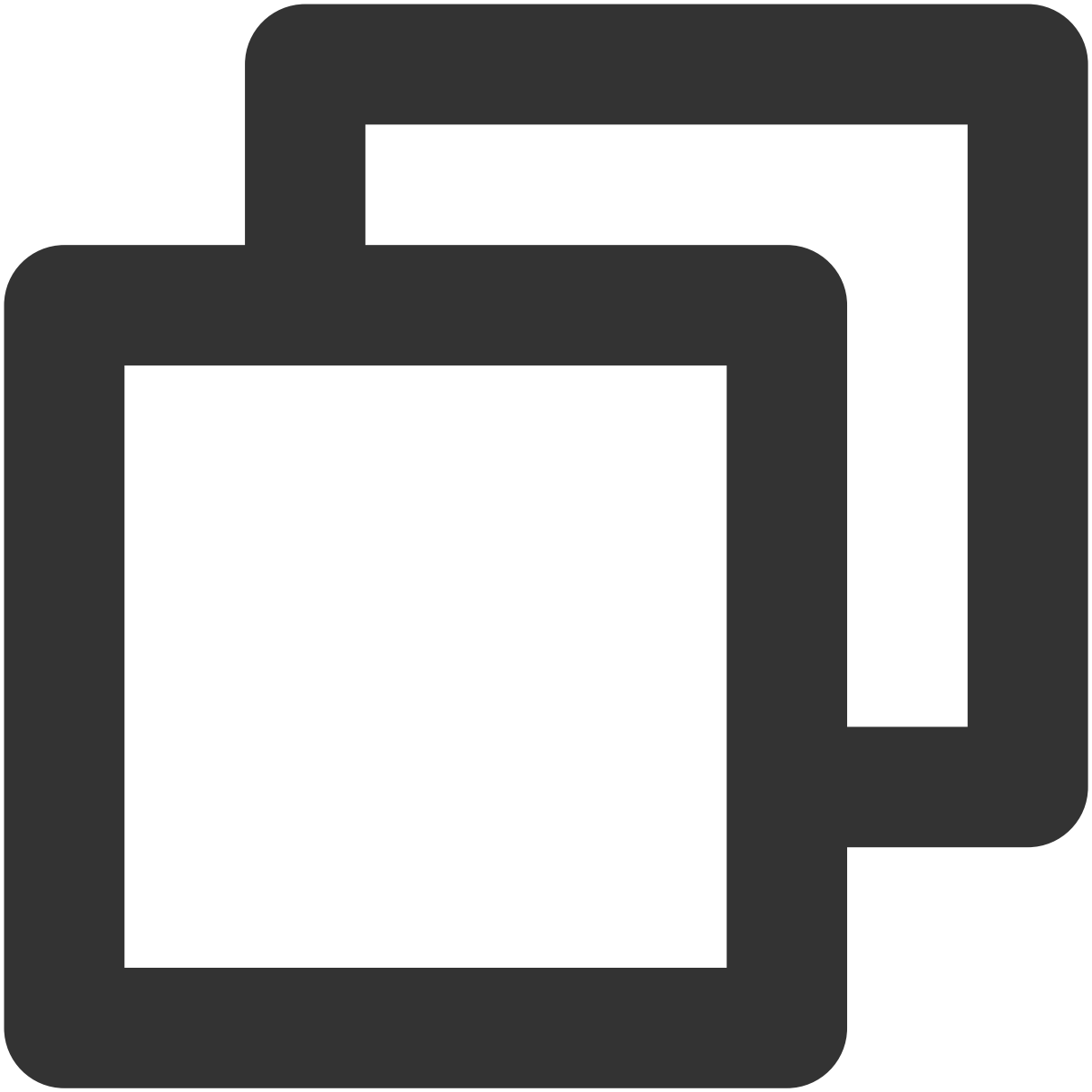
Note

Before installation, you need to delete the TKE's registered Custom Metrics API using the following command:



```
kubectl delete apiservice v1beta1.custom.metrics.k8s.io
```

©2013-2022 Tencent Cloud. All rights reserved.

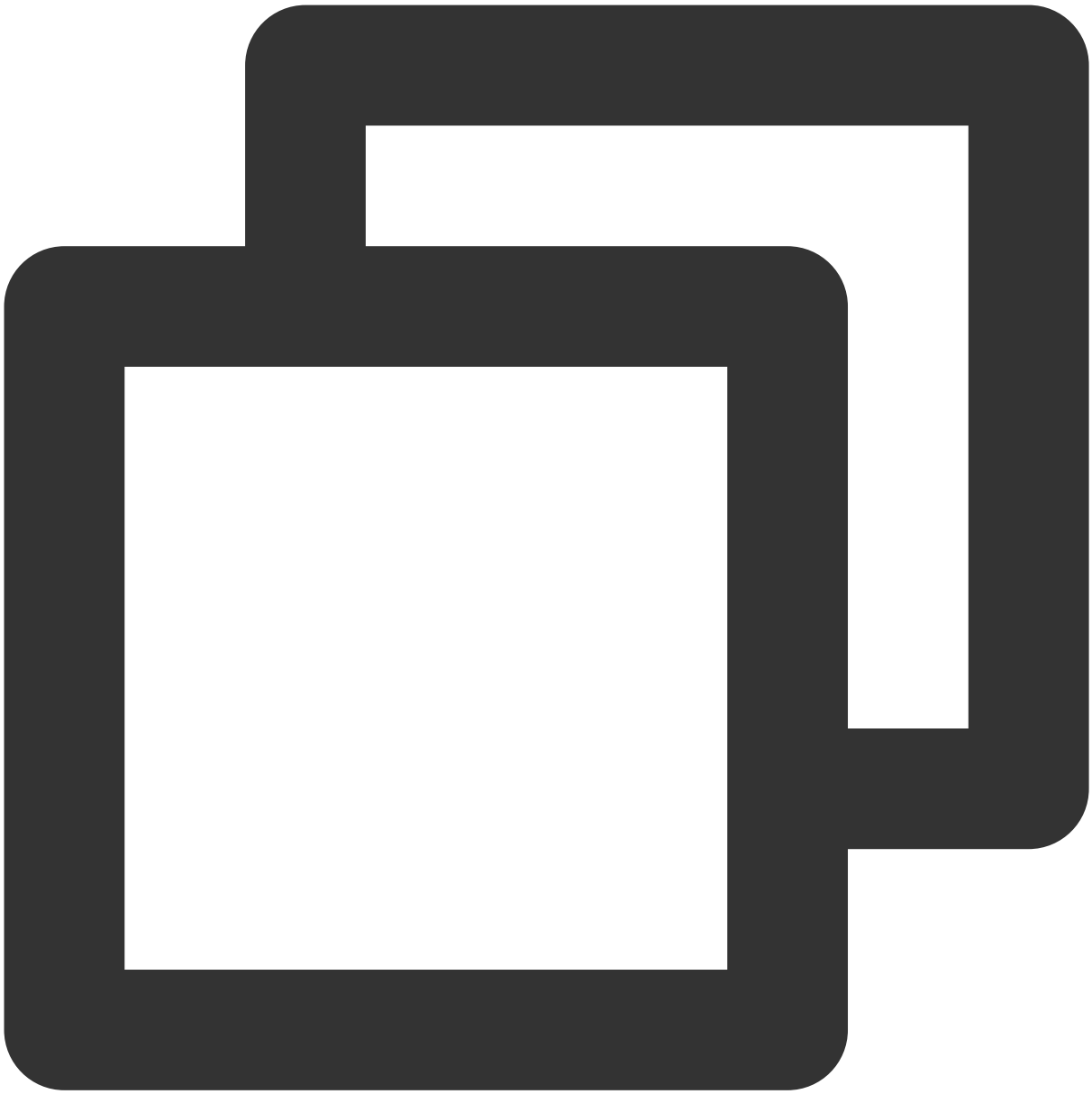


```
helm repo add prometheus-community https://prometheus-community.github.io/helm-char
helm repo update
# Helm 3
helm install prometheus-adapter prometheus-community/prometheus-adapter -f values.y
# Helm 2
# helm install --name prometheus-adapter prometheus-community/prometheus-adapter -f
```

Verifying installation result

©2013-2022 Tencent Cloud. All rights reserved.

If the installation is correct, you can run the following command to view the configured QPS related metrics returned by the Custom Metrics API, as shown below:



```
$ kubectl get --raw /apis/custom.metrics.k8s.io/v1beta1
{
  "kind": "APIResourceList",
  "apiVersion": "v1",
  "groupVersion": "custom.metrics.k8s.io/v1beta1",
  "resources": [
    {
      "name": "jobs.batch/httpserver_requests_qps",
```

©2013-2022 Tencent Cloud. All rights reserved.

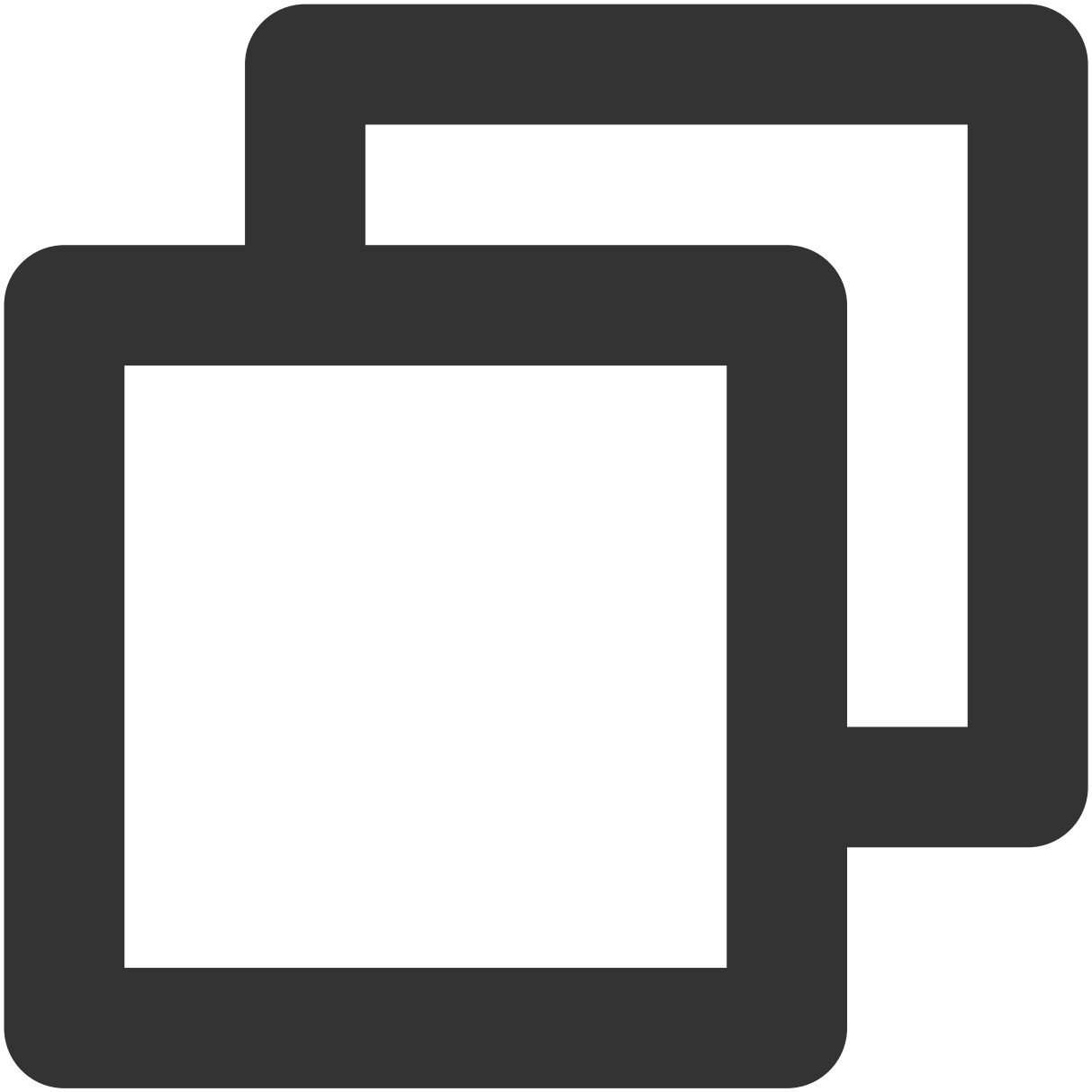
```
      "singularName": "",
      "namespaced": true,
      "kind": "MetricValueList",
      "verbs": [
        "get"
      ]
    },
    {
      "name": "pods/httpserver_requests_qps",
      "singularName": "",
      "namespaced": true,
      "kind": "MetricValueList",
      "verbs": [
        "get"
      ]
    },
    {
      "name": "namespaces/httpserver_requests_qps",
      "singularName": "",
      "namespaced": false,
      "kind": "MetricValueList",
      "verbs": [
        "get"
      ]
    }
  ]
}
```

Run the following command to view the QPS value of the Pod, as shown below:

Note

In the following example, the value is 500m, which means the value of QPS is 0.5 request/second.

©2013-2022 Tencent Cloud. All rights reserved.



```
$ kubectl get --raw /apis/custom.metrics.k8s.io/v1beta1/namespaces/httpserver/pods/
{
  "kind": "MetricValueList",
  "apiVersion": "custom.metrics.k8s.io/v1beta1",
  "metadata":{
    "selfLink": "/apis/custom.metrics.k8s.io/v1beta1/namespaces/httpserver/pods/%2A
  },
  "items": [
    {
      "describedObject": {
        "kind": "Pod",
```

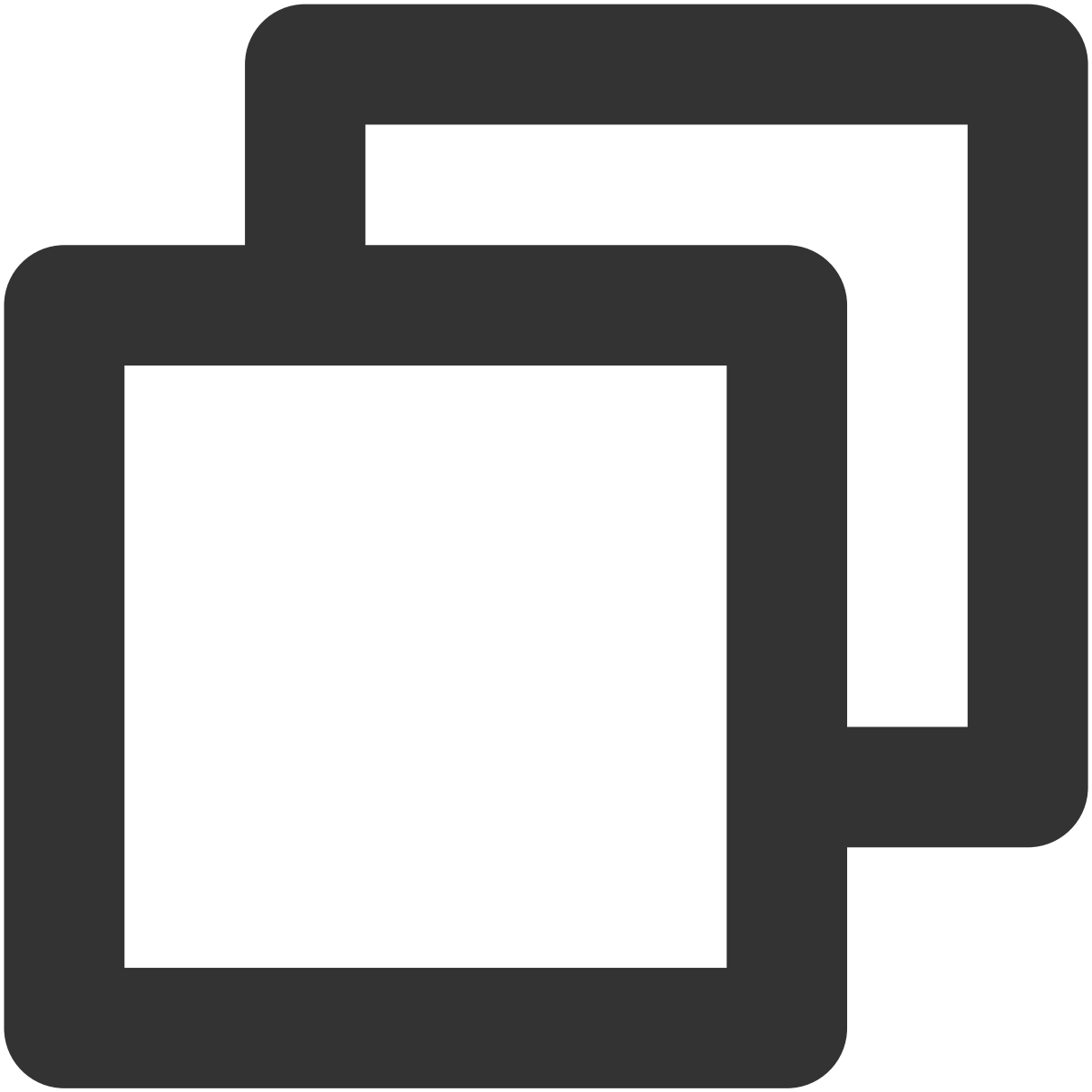
©2013-2022 Tencent Cloud. All rights reserved.

```
        "namespace": "httpserver",
        "name": "httpserver-6f94475d45-7rln9",
        "apiVersion": "/v1"
      },
      "metricName": "httpserver_requests_qps",
      "timestamp": "2020-11-17T09:14:36Z",
      "value": "500m",
      "selector": null
    }
  ]
}
```

Testing HPA

If the scaling out is triggered when the average QPS of each service Pod reaches 50 requests/second, and the minimum and maximum number of replicas are 1 and 1000 respectively, the configuration example will be as follows:

©2013-2022 Tencent Cloud. All rights reserved.

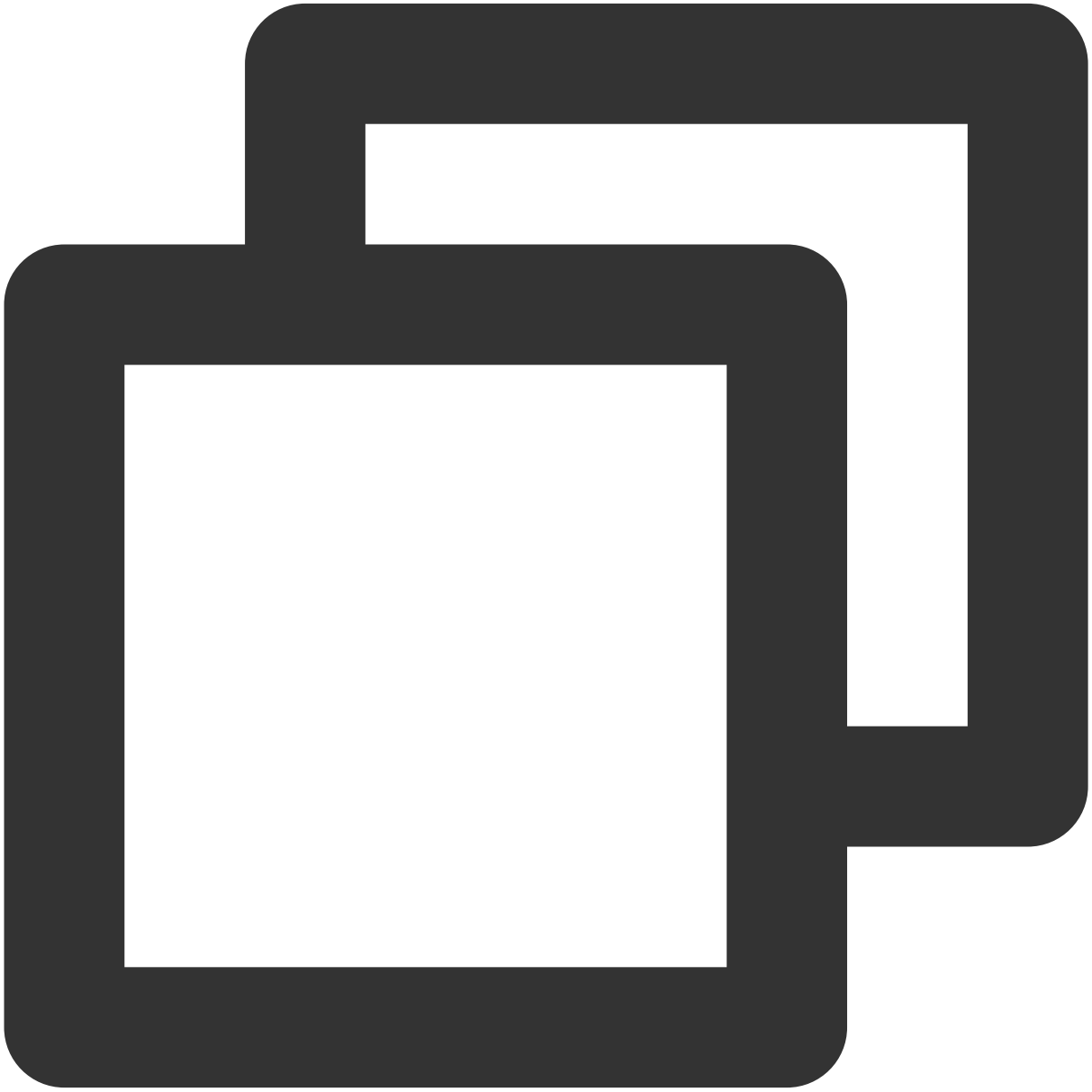


```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: httpserver
  namespace: httpserver
spec:
  minReplicas: 1
  maxReplicas: 1000
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
```

©2013-2022 Tencent Cloud. All rights reserved.

```
    name: httpserver
  metrics:
  - type: Pods
    pods:
      metric:
        name: httpserver_requests_qps
      target:
        averageValue: 50
        type: AverageValue
```

Run the following command to test the service and observe whether the scaling out is triggered, as shown below:



©2013-2022 Tencent Cloud. All rights reserved.

```
$ kubectl get hpa
NAME         REFERENCE                TARGETS     MINPODS   MAXPODS   REPLICAS   AGE
httpserver   Deployment/httpserver   83933m/50   1         1000      2          18h
$ kubectl get pods
NAME                          READY   STATUS              RESTARTS   AGE
httpserver-6f94475d45-47d5w   1/1     Running             0          3m41s
httpserver-6f94475d45-7rln9   1/1     Running             0          37h
httpserver-6f94475d45-6c5xm   0/1     ContainerCreating   0          1s
httpserver-6f94475d45-wl78d   0/1     ContainerCreating   0          1s
```

If the scaling out is triggered normally, it means that HPA has implemented auto scaling based on service custom metrics.

©2013-2022 Tencent Cloud. All rights reserved.

Utilizing HPA to Auto Scale Businesses on TKE

Last updated：2023-05-06 17:36:46

Overview

Horizontal Pod Autoscaler (HPA) for Kubernetes Pods can automatically adjust the number of replicas of Pods based on CPU usage, memory usage, and other custom metrics to make the overall level of workload services match the user-defined target value. This document introduces the HPA feature of TKE and how to use the feature to achieve automatic scaling of Pods.

Overview

The HPA feature provides TKE with a very flexible self-adaptation ability, allowing TKE to quickly increase the number of Pod replicas within the scope of user-defined settings to cope with the sudden rise of service loads and properly scale in when service loads decrease to save computing resources for other services. The entire process is automatic without the need for manual intervention. It’s suitable for service scenarios where service fluctuation is huge, the number of services is large, and frequent scaling is needed, such as e-commerce services, online education, and financial services.

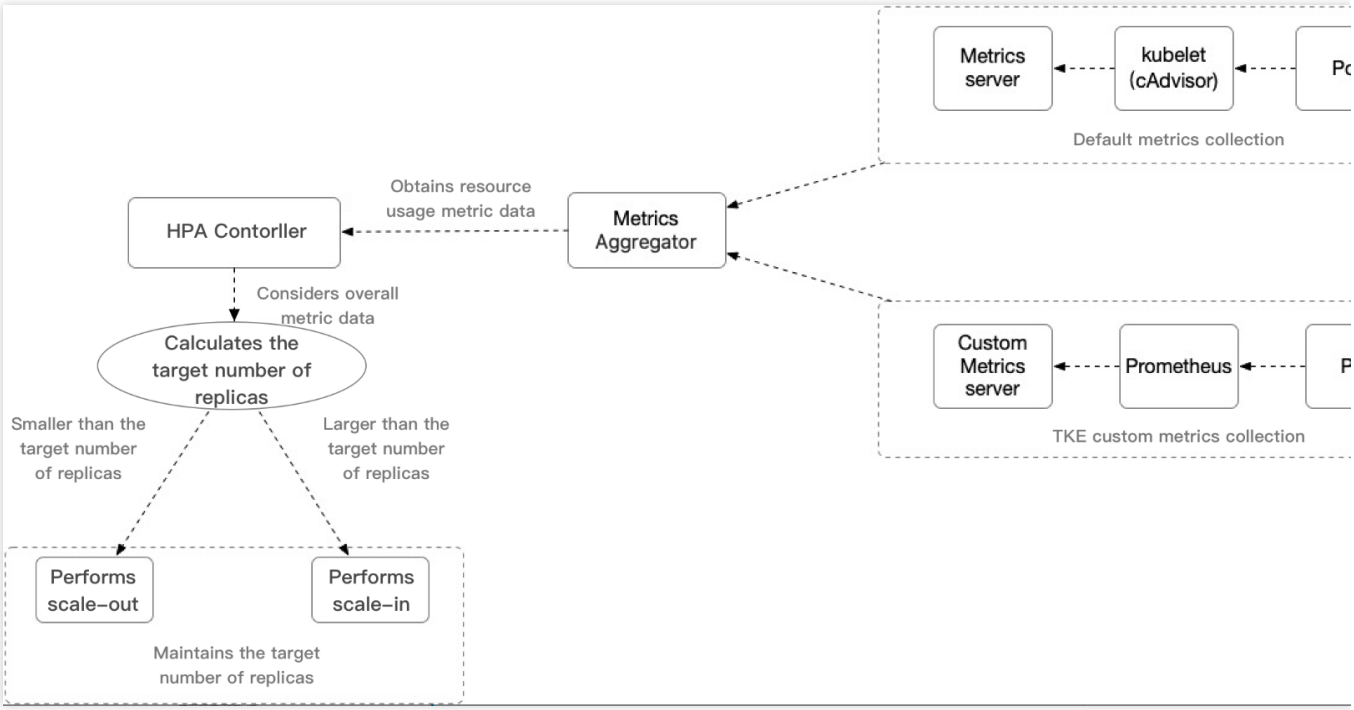
Principle Overview

The HPA feature for Pods is realized by Kubernetes API resources and the controller. Resources use metrics to determine the behavior of the controller, whereas the controller periodically adjusts the number of replicas of service Pods based on Pod resource usage, thus making the level of workloads matches the user-defined target value. The following figure shows the scaling process:

Note

The automatic horizontal scaling of Pods does not apply to objects that cannot be scaled, such as DaemonSet resources.

©2013-2022 Tencent Cloud. All rights reserved.



Key content:

HPA Controller: The control component that controls the HPA scaling logic.

Metrics Aggregator: normally, the controller obtains metric values from a series of aggregation APIs (`metrics.k8s.io` , `custom.metrics.k8s.io` , and `external.metrics.k8s.io`). The `metrics.k8s.io` API is usually provided by the Metrics server. The community edition can provide the basic CPU and memory metric types. Compared with the community edition, the custom Metrics Server collection used by TKE supports a wider range of HPA metric trigger types, providing relevant metrics that include CPU, memory, disk, network, and GPU metrics. For more information, see Autoscaling Metrics.

Note

The controller can also obtain metrics from Heapster. However, starting from Kubernetes 1.11, the controller cannot obtain metrics from Heapster any more.

Calculates the target number of replicas: For information about the TKE HPA scaling algorithm, see How it Works. For more algorithm details, see Algorithm details.

Prerequisites

You have registered a Tencent Cloud account.

You have logged in to the TKE console.

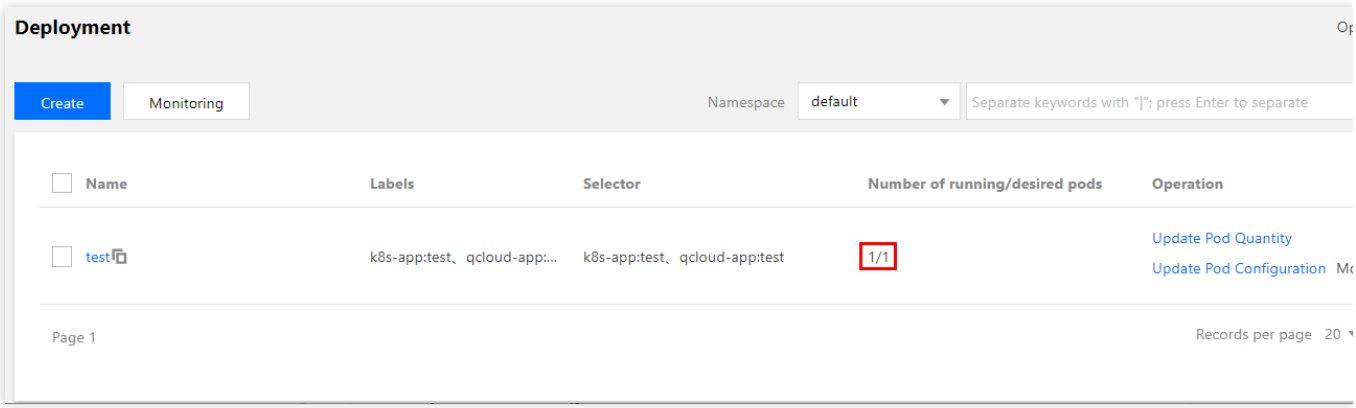
You have created a TKE cluster. For more information, see Creating a Cluster.

©2013-2022 Tencent Cloud. All rights reserved.

Directions

Deploying test workloads

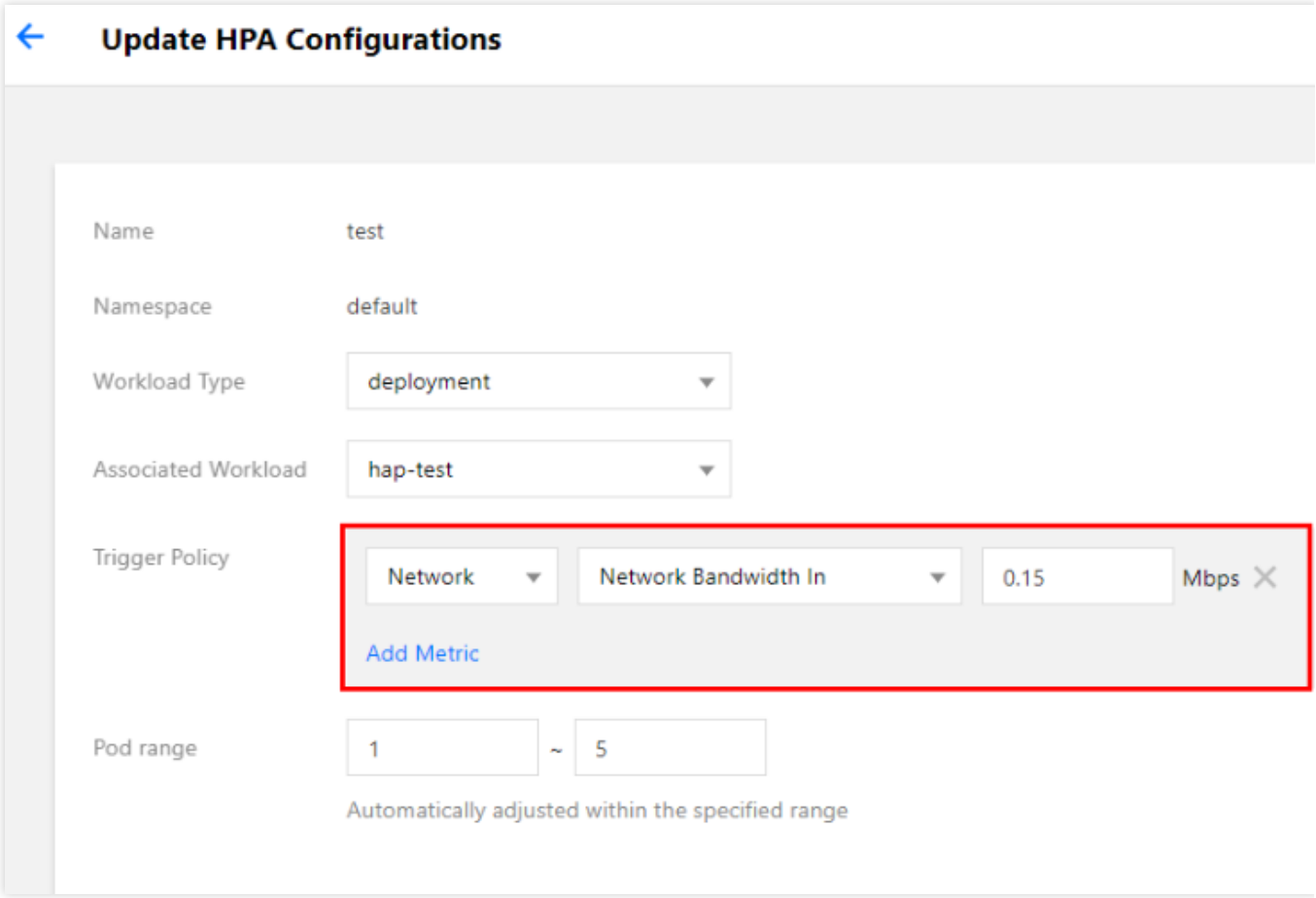
In the TKE console, create a Deployment-type workload. For more information, see Deployment Management. As an example, this document assumes that a Deployment-type workload named “hpa-test” with one replica and a service type of Web service is created, as shown below. Due to console iterations, the following figure may not be exactly the same as the actual display in the console. The actual console display prevails.



Configuring HPA

In the TKE Console, bind the test workload with an HPA configuration. For more information about how to bind an HPA configuration, see Directions. As an example, this document describes the configuration of a policy under which scale-out is triggered when the network egress bandwidth reaches 0.15 Mbps (150 Kbps), as shown below:

©2013-2022 Tencent Cloud. All rights reserved.

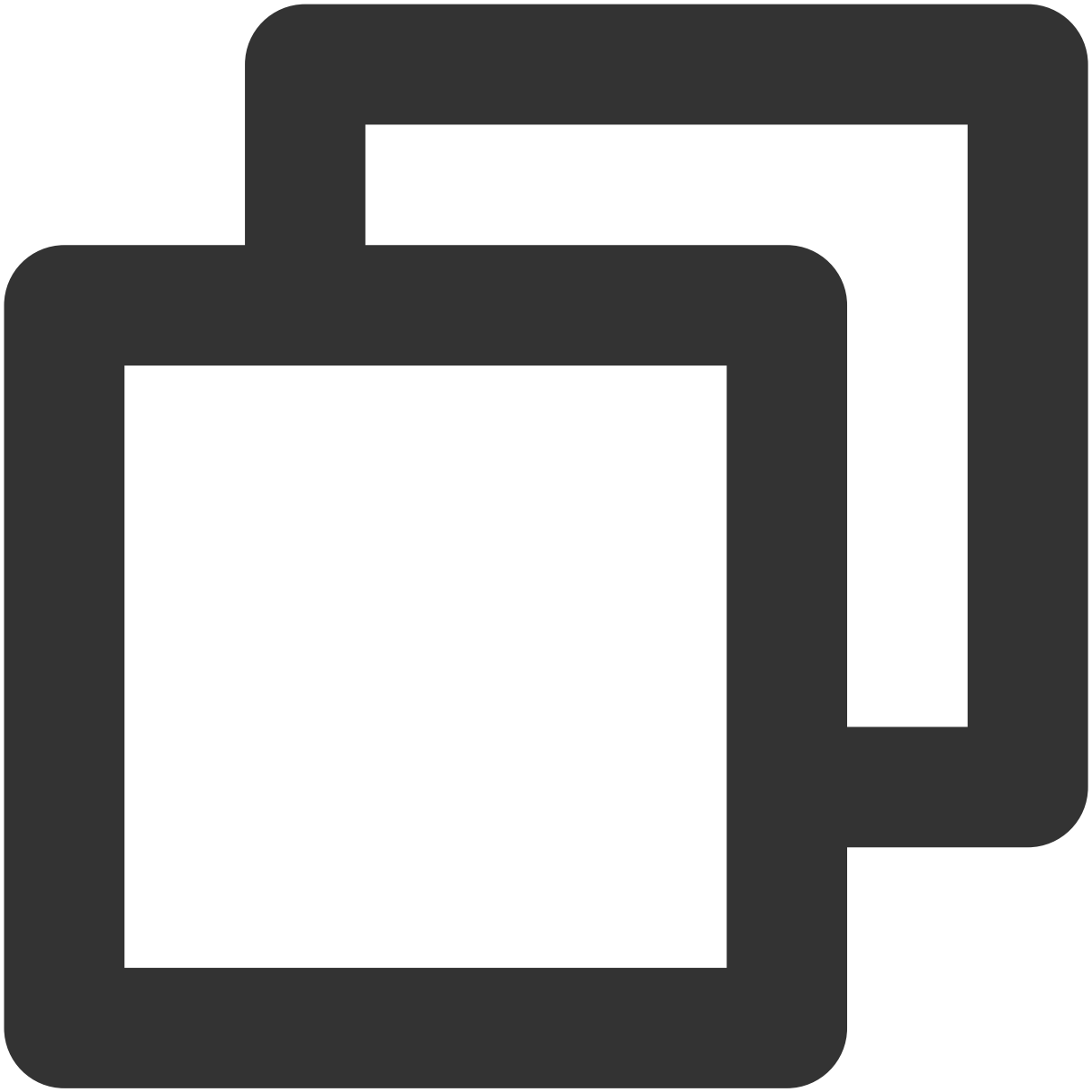


Function verification

Simulating the scale-out process

Run the following command to launch a temporary Pod in the cluster to test the configured HPA feature (simulated client):

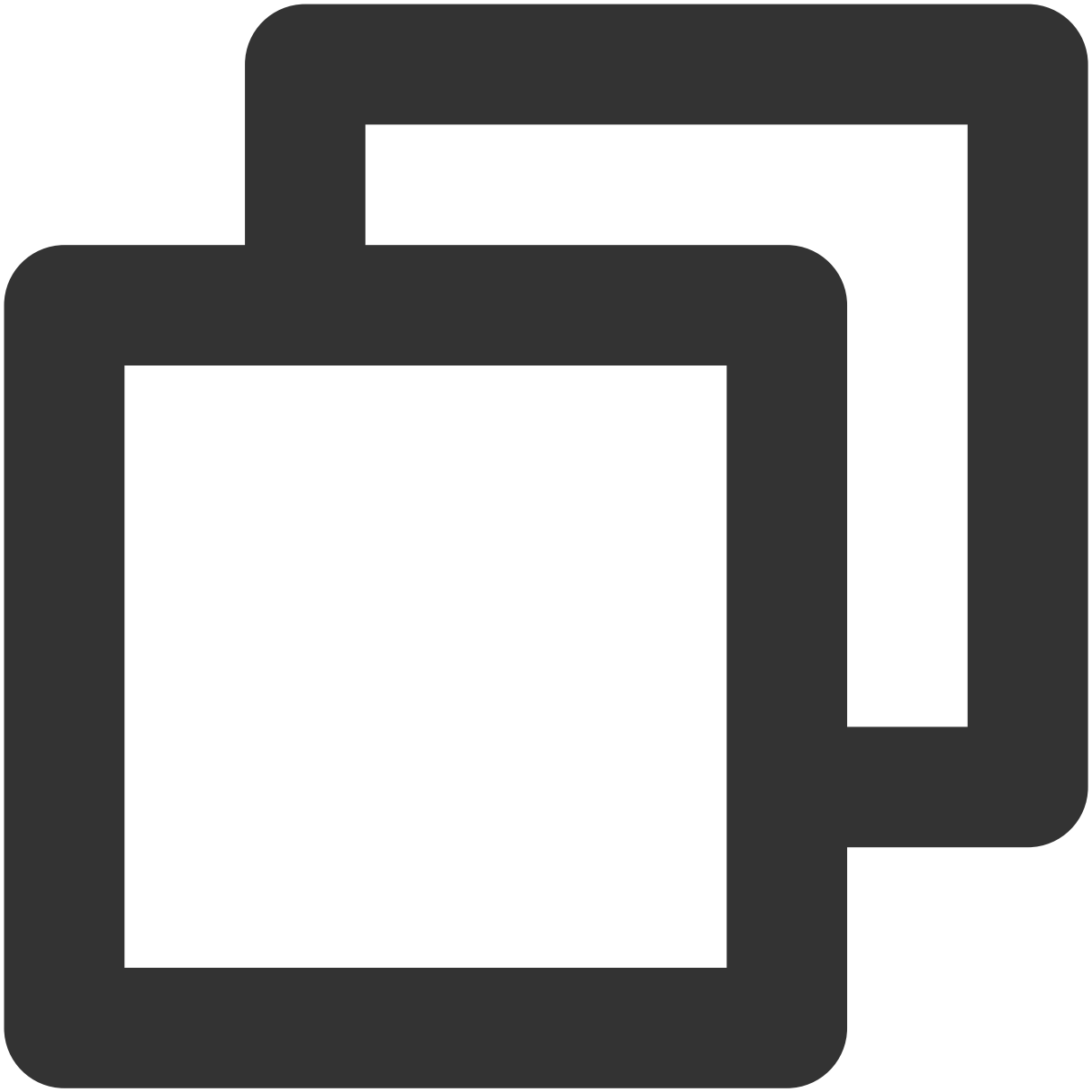
©2013-2022 Tencent Cloud. All rights reserved.



```
kubectl run -it --image alpine hpa-test --restart=Never --rm /bin/sh
```

Run the following command in the temporary Pod to simulate a situation where large numbers of requests accessing the "hpa-test" service in a short period causes the egress traffic bandwidth to increase:

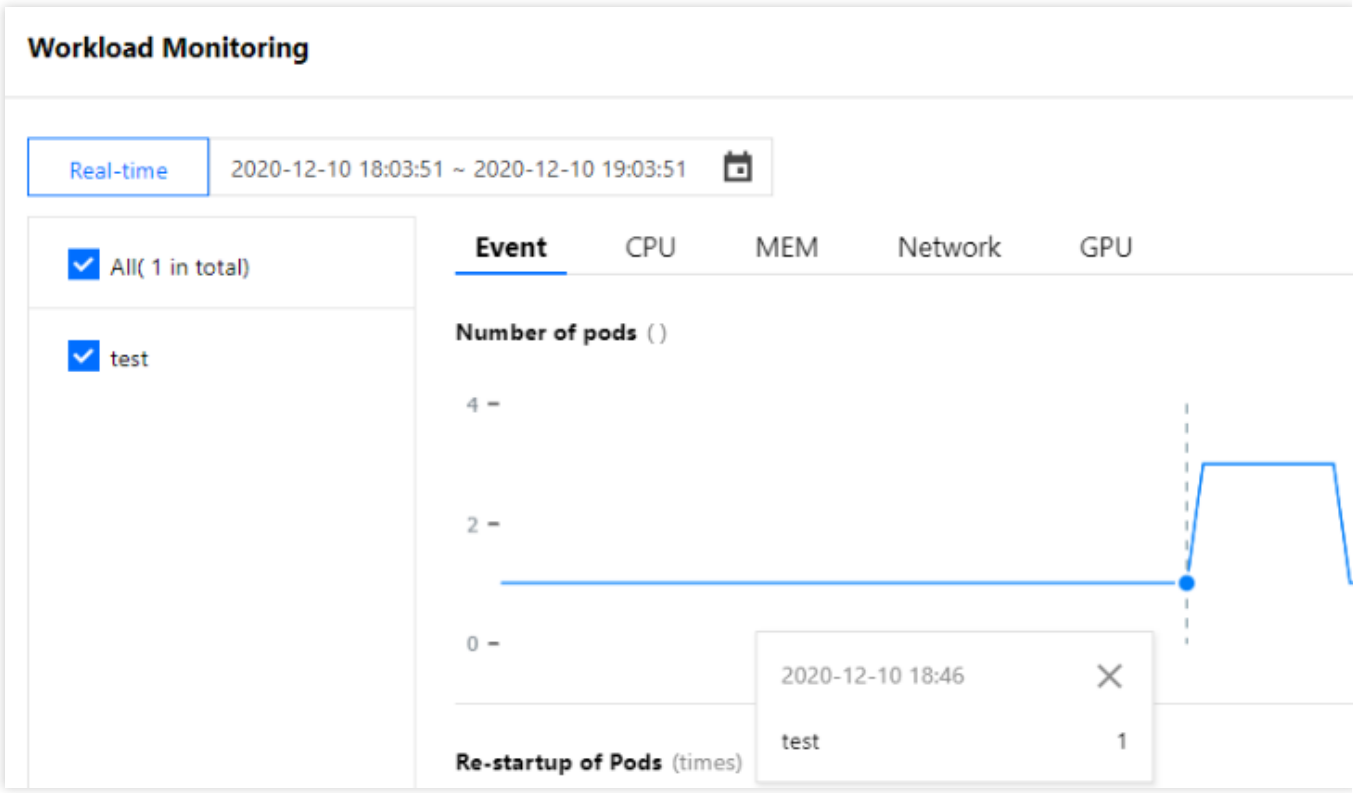
©2013-2022 Tencent Cloud. All rights reserved.



```
# hpa-test.default.svc.cluster.local is the domain name of the service in the clust
while true; do wget -q -O - hpa-test.default.svc.cluster.local; done
```

After running the request simulation command in the test Pod, observe the monitored number of Pods of the workload. You can find that scale-out is triggered because the number of Pod replicas increases to 2 at 16:21, as shown below. Due to console iterations, the following figure may not be exactly the same as the actual display in the console. The actual console display prevails.

©2013-2022 Tencent Cloud. All rights reserved.

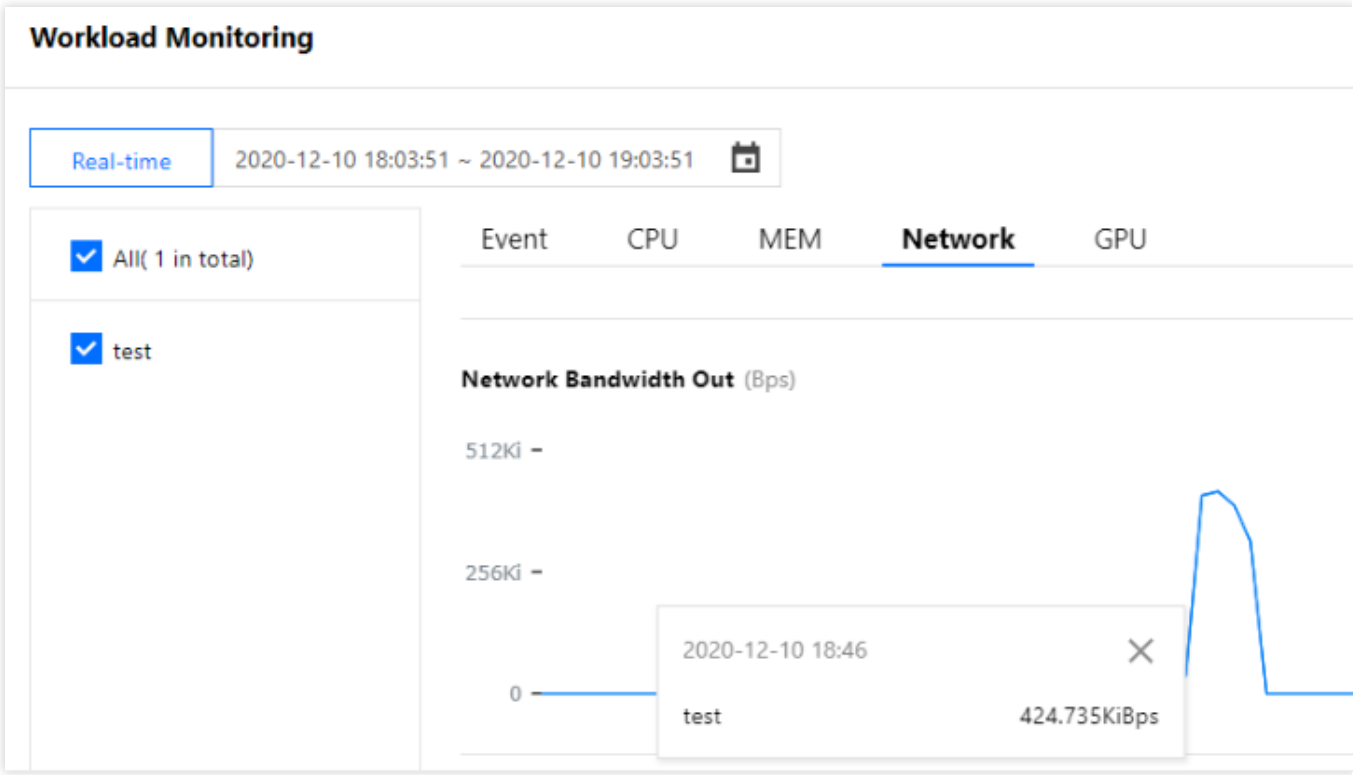


The triggering of the HPA scaling algorithm can also be proved by the fact that in the workload monitoring information, the network egress bandwidth increases to 196 Kbps at 16:21, as shown below. This bandwidth exceeds the set target egress bandwidth value of HPA, so another replica is created to meet the need. Due to console iterations, the following figure may not be exactly the same as the actual display in the console. The actual console display prevails.

Note

The HPA scaling algorithm does not just rely on formula calculation to control the scaling logic but takes multiple dimensions into consideration to decide whether scale-out or scale-in is needed. Therefore, the actual implementation may slightly differ from expectations. For more information, see Algorithm details.

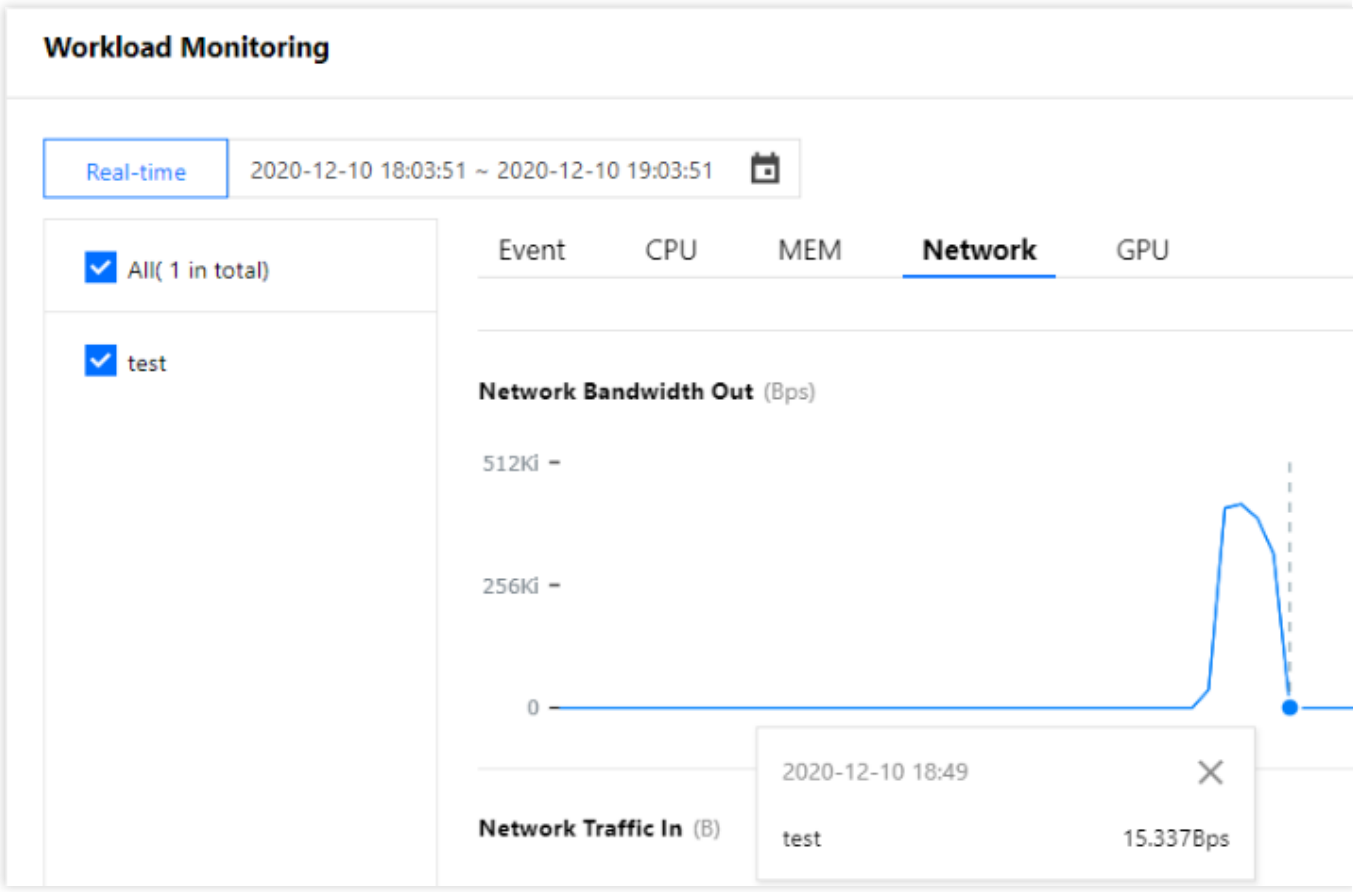
©2013-2022 Tencent Cloud. All rights reserved.



Simulating the scale-in process

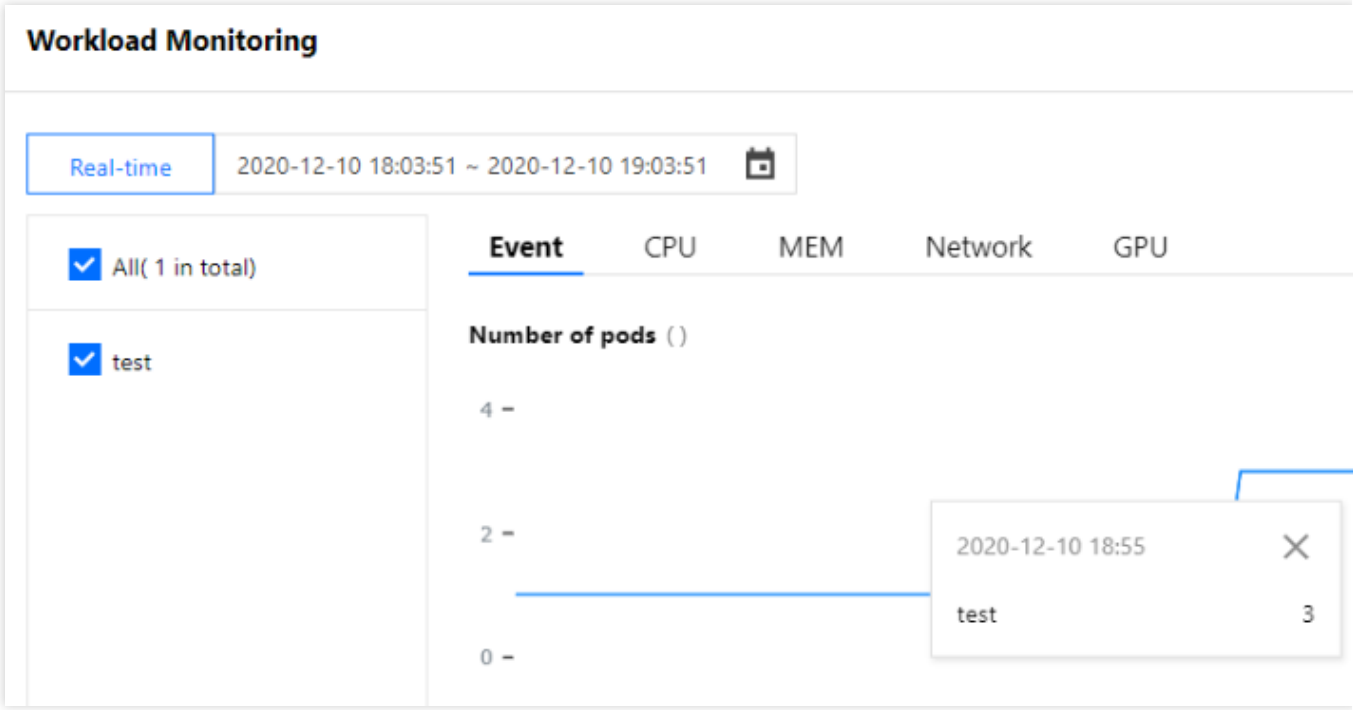
When simulating the scale-in process, manually stop executing the request simulation command at around 16:24. According to the HPA scaling algorithm, the workload scale-in condition is met. In the workload monitoring information, you can find that the network egress bandwidth decreases to the original level before the scale-out, as shown below:

©2013-2022 Tencent Cloud. All rights reserved.

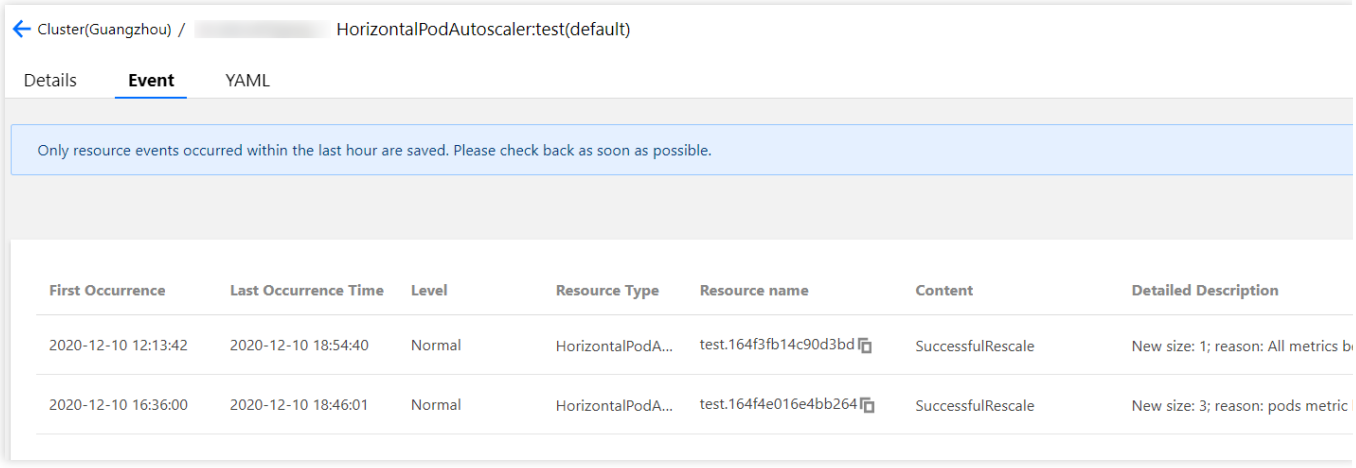


However, scale-in is actually not triggered until 16:30. This is because there is a toleration time of five minutes by default after the triggering condition is met to prevent frequent scaling due to metric value fluctuations within a short time. For more information, see here. As shown below, the number of replicas decreases to 1 based on the HPA scaling algorithm five minutes after the command is stopped. Due to console iterations, the following figure may not be exactly the same as the actual display in the console. The actual console display prevails.

©2013-2022 Tencent Cloud. All rights reserved.

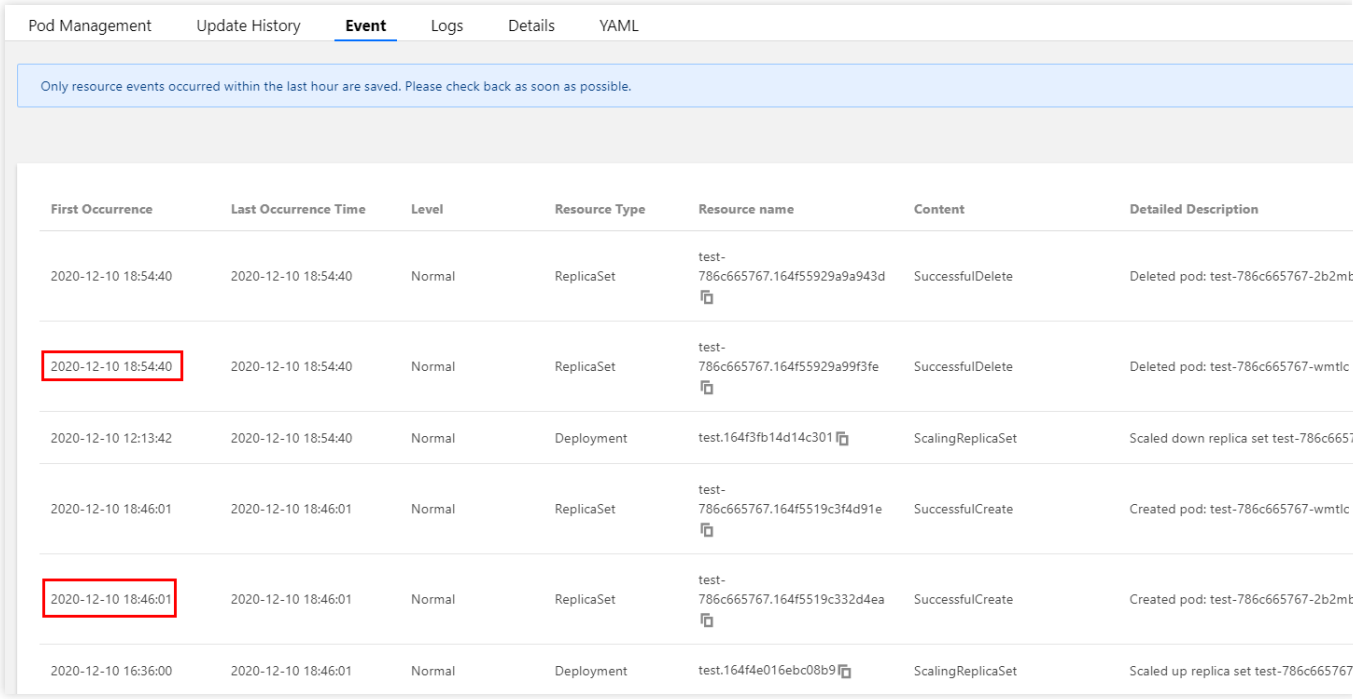


When an HPA scaling event occurs in TKE, the event will be displayed in the event list of the corresponding HPA instance. Note that the time fields on the event notification list include "First occurrence" and "Last occurrence". "First occurrence" indicates the first time when the same event occurred, while "Last occurrence" indicates the latest time when the same event occurred. Therefore, as you can see in the event list shown in the figure below, the "Last occurrence" field displays 16:21:03 for the scale-out event in our example and 16.29:42 for the scale-in event. The points in time displayed here match those in the workload monitoring, as shown below:



Besides, the workload event list also records the events of adding/deleting replicas by workloads when HPA occurs. As you can see in the figure below, the points in time of workload scale-out and scale-in match those displayed in the HPA event list. The point in time when the number of replicas increased is 16:21:03, and the point in time when the number of replicas decreased is 16:29:42.

©2013-2022 Tencent Cloud. All rights reserved.



Summary

This example demonstrates the HPA feature of TKE, describing how to use the TKE custom metric type of network egress bandwidth as the metric for triggering workload HPA scaling.

When the actual metric value of the workload exceeds the target metric value configured by HPA, HPA calculates the proper number of replicas according to its scale-out algorithm to implement scale-out. This ensures that the metric level of the workload meets expectations and that the workload can run healthily and steadily.

When the actual metric value of the workload is far lower than the target metric value configured by HPA, HPA waits until the toleration time expires, and then calculates the proper number of replicas to implement scale-in and release idle resources appropriately, thus achieving the objective of improving resource utilization. Moreover, during the whole process, relevant events are recorded in the HPA and workload event lists, so that the whole scaling process of the workload is traceable.

©2013-2022 Tencent Cloud. All rights reserved.

Using VPA to Realize Pod Scaling up and Scaling down in TKE

Last updated：2021-06-09 17:46:36

Overview

Kubernetes Vertical Pod Autoscaler (VPA) can automatically adjust the reserved CPU and memory of Pod, improve cluster resource utilization and release CPU and memory for other Pods. This document describes how to use the VPA community edition in TKE to implement the scaling up and scaling down of Pods.

Use Cases

The auto-scaling feature of VPA makes the TKE very flexible and adaptive. When the business load increases sharply, VPA can quickly increase the Request of the container within the user's setting range. When the business load decreases, VPA can appropriately reduce the Request based on the actual needs to save computing resources. The entire process is automated without manual intervention. It is suitable for scenarios that require rapid expansion and stateful application expansion. In addition, VPA can be used to recommend a more reasonable Request to user, and improve the resource utilization of the container while ensuring that the container has sufficient available resources.

VPA Strengths

Compared with Horizontal Pod Autoscaler (HPA), VPA has the following advantages:

- VPA does not need to adjust the replicas of Pod for expansion, and the expansion speed is faster.
- VPA can achieve the expansion of the stateful applications, while HPA is not suitable for the scaling out of the stateful applications.
- If the Request is set too large, the cluster resource utilization is still very low when HPA is used to scale in the Pods to a Pod. In this case, you can use VPA to scale down to improve the cluster resource utilization.

VPA Limits

Note：

©2013-2022 Tencent Cloud. All rights reserved.

VPA community edition is in testing. Use this feature with caution. We recommend setting "updateMode" to "Off" to ensure that VPA will not automatically change the value of Request. You can still view the recommended value of request bound to the load in the VPA object.

- You can use the VPA to update the resource configurations of the running Pods. This feature is in testing. The configuration updates will lead to Pod restart and rebuilding, and the Pods may be scheduled to other nodes.
- The VPA does not evict the Pods that are not run under a controller. For these Pods, the `Auto` mode is equivalent to the `Initial` mode.
- You cannot run VPA simultaneously with the HPA that uses the CPU and memory as metrics. If the HPA uses other metrics except CPU and memory, you can run the VPA with the HPA at the same time. For details, see Using Custom Metrics for Auto Scaling in TKE.
- The VPA uses an Admission Webhook as its admission controller. If there are other Admission Webhooks in the cluster, you need to ensure that they do not conflict with the Admission Webhooks of the VPA. The execution sequence of admission controllers is defined in the configuration parameters of the API Server.
- The VPA can react to most Out of Memory (OOM) events.
- The VPA performance has not been tested in large-scale clusters.
- The recommended value of Pod resource Request set by the VPA may exceed the upper limit of the available resources (such as node resources, idle resources, and resource quotas). In this case, the Pod may go to Pending and cannot be scheduled. This can be partly addressed by using the VPA together with the Cluster Autoscaler.
- Multiple VPA resources matching the same pod have undefined behavior.

For more limitations on VPA, see VPA Known limitations.

Prerequisites

- You have created a TKE cluster.
- The cluster has been connected via the command line tool Kubectl. For how to connect to a cluster, see Connecting to a Cluster.

Directions

Deploying VPA

1. Log in to the CVM in the cluster.
2. You can connect to a TKE cluster from a local client using the command line tool kubectl.
3. Run the following command to clone the kubernetes/autoscaler from GitHub Repository.

©2013-2022 Tencent Cloud. All rights reserved.

```
sh
git clone https://github.com/kubernetes/autoscaler.git
```

4. Run the following command to switch to the `vertical-pod-autoscaler` directory.

```
cd autoscaler/vertical-pod-autoscaler/
```

5. (Optional) If you have already deployed another version of VPA, run the following command to remove it. Otherwise an exception may occur.

```
./hack/vpa-down.sh
```

6. Run the following command to deploy VPA related components to your cluster.

```
./hack/vpa-up.sh
```

7. Run the following command to verify whether the VPA component is successfully created.

```
kubectl get deploy -n kube-system | grep vpa
```

After successfully creating the VPA component, you can check the three Deployments in the kube-system namespace, namely vpa-admission-controller, vpa-recommender, and vpa-updater, as shown below:

```
[root@VM-22-114-centos hack]# kubectl get deploy -n kube-system | grep vpa
vpa-admission-controller                1/1     1            1           17s
vpa-recommender                         1/1     1            1           22h
vpa-updater                             1/1     1            1           22h
```

Sample 1: using VPA to obtain the recommended value of Request

Note：

- We do not recommend using VPA to automatically update Request in a production environment.
- You can use VPA to view the recommended value of Request and manually trigger the update as needed.

©2013-2022 Tencent Cloud. All rights reserved.

In this sample, you will create a VPA object with `updateMode` set to `Off` and create a Deployment with two Pods, and each Pod has a container. After the Pod is created, VPA will analyze the CPU and memory requirements of the container and record the recommended value of Request in the `status` field. VPA will not automatically update the resource requests of the running containers.

Run the following command in kubectl to generate a VPA object named `tke-vpa`, pointing to a Deployment named `tke-deployment`:

```
shell
cat <<EOF | kubectl apply -f -
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
name: tke-vpa
spec:
targetRef:
apiVersion: "apps/v1"
kind: Deployment
name: tke-deployment
updatePolicy:
updateMode: "Off"
EOF
```

Run the following command to generate a Deployment object named `tke-deployment`:

```
shell
cat <<EOF | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
name: tke-deployment
spec:
replicas: 2
selector:
matchLabels:
app: tke-deployment
template:
metadata:
labels:
app: tke-deployment
spec:
containers:
- name: tke-container
image: nginx
EOF
```

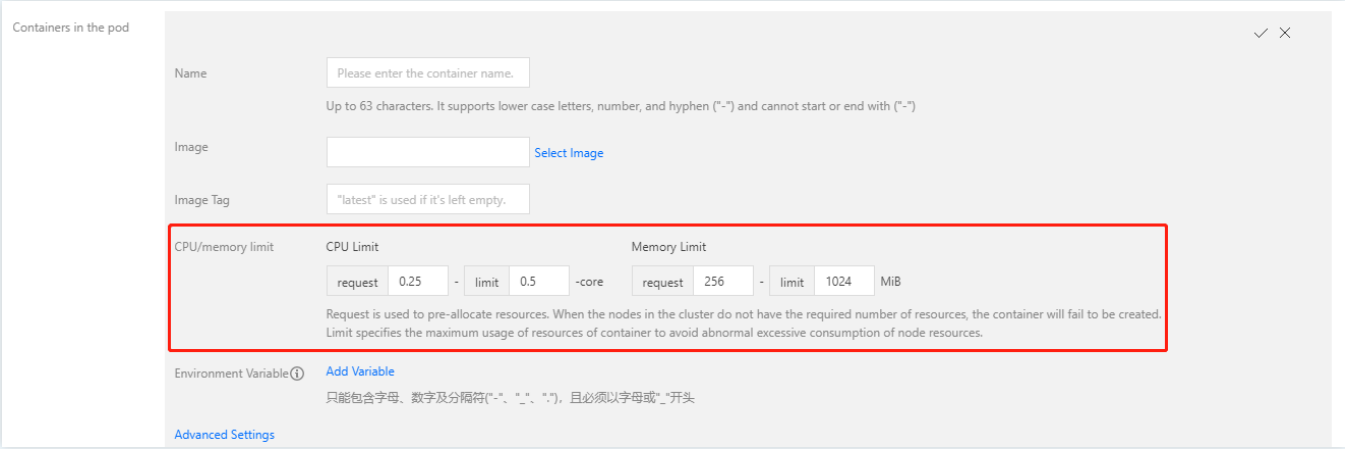
©2013-2022 Tencent Cloud. All rights reserved.

The generated Deployment object is show as follows:

```
[root@VM-22-114-centos ~]# kubectl get deploy,vpa | grep tke
deployment.apps/tke-deployment   2/2     2            2           7m1s
verticalpodautoscaler.autoscaling.k8s.io/tke-vpa   Off   25m   262144k   True      66s
```

Note：

The `tke-deployment` created above does not set the Request of CPU or memory, and the Qos of the Pod is set to BestEffort. In this case, Pod is easy to be evicted. We recommend that you set the Request and Limit when creating the Deployment of the application. If you create a workload via the TKE console, the default Request and Limit of each container will be automatically set.



Run the following command to view the recommended Requests of CPU and memory by VPA:

```shell
kubectl get vpa tke-vpa -o yaml
```

The execution results are as follows:

```yaml
...
recommendation:
containerRecommendations:
- containerName: tke-container
lowerBound:
cpu: 25m
memory: 262144k
```

©2013-2022 Tencent Cloud. All rights reserved.

```
target:# Recommended value
cpu: 25m
memory: 262144k
uncappedTarget:
cpu: 25m
memory: 262144k
upperBound:
cpu: 1771m
memory: 1851500k
```

The CPU and memory corresponding to `target` are the recommended Requests. You can remove the previous Deployment and create a new Deployment with the recommended Request.

Field	Description
lowerBound	The minimum value recommended. The use of a Request smaller than this value may have a major impact on performance or availability.
target	Recommended value. The VPA calculates the most appropriate Request.
uncappedTarget	The latest recommended value. It is only based on the actual resource usage and does not consider the recommended value range of the container set in `.spec.resourcePolicy.containerPolicies`. The uncappedTarget may differ from the recommended `lowerBound` and `upperBound`. This field is only used to indicate the status and will not affect the actual resource allocation.
upperBound	The maximum value recommended. The use of a Request larger than this value may cause a resource waste.

Sample 2: Disabling a specific container

If there are multiple containers in the Pod, for example, one is an application container and the other is a secondary container. You can choose to stop recommending Request for the secondary container to save the cluster resources.

In this sample, you will create a VPA with a specific container disabled, and create a Deployment with a Pod, and the Pod contains two containers. After the Pod is created, VPA only creates and calculates the recommended value for one container, and stops recommending Request for the other container.

Run the following command in the kubectl to generate a VPA object named `tke-opt-vpa`, pointing to a Deployment named `tke-opt-deployment`:

```
shell
cat <<EOF | kubectl apply -f -
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
```

©2013-2022 Tencent Cloud. All rights reserved.

```
name: tke-opt-vpa
spec:
targetRef:
apiVersion: "apps/v1"
kind: Deployment
name: tke-opt-deployment
updatePolicy:
updateMode: "Off"
resourcePolicy:
containerPolicies:
- containerName: tke-opt-sidecar
mode: "Off"
EOF
```

Note：
In the `.spec.resourcePolicy.containerPolicies` of the VPA, the `mode` of `tke-opt-sidecar` is set to "Off", and VPA will not calculate and recommend a new Request for `tke-opt-sidecar`.

Run the following command to generate a Deployment object named `tke-deployment`：

```
sh
cat <<EOF | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
name: tke-opt-deployment
spec:
replicas: 1
selector:
matchLabels:
app: tke-opt-deployment
template:
metadata:
labels:
app: tke-opt-deployment
spec:
containers:
- name: tke-opt-container
image: nginx
- name: tke-opt-sidecar
image: busybox
```

©2013-2022 Tencent Cloud. All rights reserved.

```
command: ["sh","-c","while true; do echo TKE VPA; sleep 60; done"]
EOF
```

The generated Deployment object is show as follows:

```
[root@VM-22-114-centos ~]# kubectl get deploy,vpa | grep opt
deployment.apps/tke-opt-deployment   1/1     1            1           5m12s
verticalpodautoscaler.autoscaling.k8s.io/tke-opt-vpa   Off    25m   262144k   True       14m
```

Run the following command to view the recommended Requests of CPU and memory by VPA:

```
shell
kubectl get vpa tke-opt-vpa -o yaml
```

The execution results are as follows:

```
yaml
...
recommendation:
containerRecommendations:
- containerName: tke-opt-container
lowerBound:
cpu: 25m
memory: 262144k
target:
cpu: 25m
memory: 262144k
uncappedTarget:
cpu: 25m
memory: 262144k
upperBound:
cpu: 1595m
memory: 1667500k
```

In the execution result, there is only the recommended value of `tke-opt-container` , and no recommended value of `tke-opt-sidecar` .

Sample 3: updating the Request automatically

Note：
Automatic updating the resources of the running Pods is an experimental feature of VPA. We recommend that you do not use this feature in a production environment.

©2013-2022 Tencent Cloud. All rights reserved.

In this sample, you will create a VPA that can automatically adjust the CPU and memory Requests, and create a Deployment with two Pods. Each Pod will set the Request and Limit of the resource.

Run the following command in the kubectl to generate a VPA object named `tke-auto-vpa`, pointing to a Deployment named `tke-auto-deployment`:

```
yaml
cat <<EOF | kubectl apply -f -
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
name: tke-auto-vpa
spec:
targetRef:
apiVersion: "apps/v1"
kind: Deployment
name: tke-auto-deployment
updatePolicy:
updateMode: "Auto"
EOF
```

Note：
The `updateMode` field of this VPA is set to `Auto`, which means that the VPA can update the CPU and memory Requests during the life cycle of the Pod. VPA can remove the Pod, adjust the CPU and memory Requests, and then rebuild a Pod.

Run the following command to generate a Deployment object named `tke-auto-deployment`:

```
shell
cat <<EOF | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
name: tke-auto-deployment
spec:
replicas: 2
selector:
matchLabels:
app: tke-auto-deployment
template:
metadata:
labels:
```

©2013-2022 Tencent Cloud. All rights reserved.

```
app: tke-auto-deployment
spec:
containers:
- name: tke-container
image: nginx
resources:
requests:
cpu: 100m
memory: 100Mi
limits:
cpu: 200m
memory: 200Mi
EOF
```

Note：
When the Deployment is created in the above operation, the Request and Limit of the resource have been set. In this case, VPA will not only recommend the Request, but also automatically recommend the Limit based on the initial ratio of Request and Limit. For example, the initial ratio of CPU's Request and Limit in YAML is 100m:200m, namely 1:2, then the value of Limit recommended by VPA is twice the value of Request recommended in the VPA object.

The generated Deployment object is show as follows:

```
[root@VM-22-114-centos ~]# kubectl get deploy,vpa | grep tke-auto
deployment.apps/tke-auto-deployment   2/2     2            2           10m
verticalpodautoscaler.autoscaling.k8s.io/tke-auto-vpa   Auto   25m   262144k   True   7m26s
```

Run the following command to obtain the detailed information of the running Pod:

```
sh
kubectl get pod pod-name -o yaml
```

The execution result is shown below. VPA modified the original Request and Limits to the recommended value of VPA, and maintained the initial ratio of Request and Limits. At the same time, an annotation that recorded the updates is generated:

```
yaml
apiVersion: v1
kind: Pod
```

©2013-2022 Tencent Cloud. All rights reserved.

```
metadata:
annotations:
...
vpaObservedContainers: tke-container
vpaUpdates: Pod resources updated by tke-auto-vpa: container 0: memory request, c
pu request
...
spec:
containers:
...
resources:
limits:# The new Request and Limits will maintain the initial ratio
cpu: 50m
memory: 500Mi
requests:
cpu: 25m
memory: 262144k
...
```

Run the following command to obtain the detailed information of the relevant VPA:

```
sh
kubectl get vpa tke-auto-vpa -o yaml
```

The execution results are as follows:

```
yaml
...
recommendation:
containerRecommendations:
- containerName: tke-container
Lower Bound:
Cpu: 25m
Memory: 262144k
Target:
Cpu: 25m
Memory: 262144k
Uncapped Target:
Cpu: 25m
Memory: 262144k
Upper Bound:
Cpu: 101m
Memory: 262144k
```

`target` means that the container will run in the best state when the Requests of CPU and memory are 25m and 262144k respectively.

©2013-2022 Tencent Cloud. All rights reserved.

VPA uses the recommended values of `lowerBound` and `upperBound` to decide whether to evict a Pod and replace it with a new Pod. If the Pod's Request is smaller than the lower limit or larger than the upper limit, VPA will remove the Pod and replace it with a Pod with a recommended value.

Troubleshooting

1. An error occurs when running the `vpa-up.sh` script.

Errors

```
shell
ERROR: Failed to create CA certificate for self-signing. If the error is "unknown
option -addext", update your openssl version or deploy VPA from the vpa-release-0
.8 branch.
```

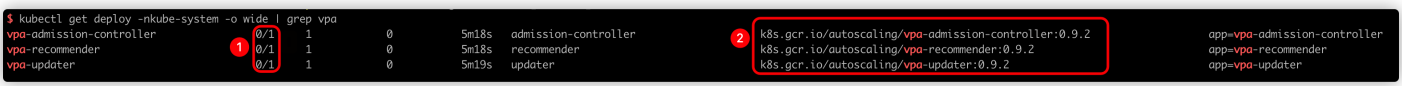
Solutions

1. If you have not run the command through the CVM in the cluster, we recommend that you download the Autoscaler project in the CVM and deploy VPA. If you need to connect the cluster to your CVM, see Connecting to a Cluster.
2. If the errors still exist, please check whether the following problems exist:
   - Check whether the `openssl` version of the cluster CVM is later than v1.1.1.
   - Whether the `vpa-release-0.8` branch of the Autoscaler project is used.

2. The VPA-related load could not be started up.

Errors

If the VPA-related load fails to start up, and the following message is generated:



Message 1: indicates that the Pods in the load fail to run.
Message 2: indicates the address of the image.

Solutions

The VPA-related load could not be started up because the image located in GCR could not be downloaded. You can try the following steps to solve the problem:

1. **Download the image**.
   Visit the "k8s.gcr.io/" image repository and download the images of vpa-admission-controller, vpa-recommender,

©2013-2022 Tencent Cloud. All rights reserved.

and vpa-updater.

2. **Replace the image tags and push the images**.
   Replace the image tags of vpa-admission-controller, vpa-recommender, and vpa-updater and push them to your image repository. For how to push and upload the image, please see TCR Personal Edition.
3. **Change the image address in YAML**.
   In the YAML file, update the image addresses of vpa-admission-controller, vpa-recommender, and vpa-updater to the new addresses you set.

©2013-2022 Tencent Cloud. All rights reserved.

Adjusting HPA Scaling Sensitivity Based on Different Business Scenarios

Last updated：2023-05-23 15:22:02

Support for Scaling Speed Adjustment by HPA v2beta2 and Later

Sensitivity adjustment for HPA scale-out is not supported by versions earlier than K8s 1.18.

- The `--horizontal-pod-autoscaler-downscale-stabilization-window` parameter of `kube-controller-manager` controls the scale-in time window, which is five minutes by default, that is, a scale-in can be performed at least five minutes after the workload reduction.
- The fixed algorithm of the HPA controller and the constant factor of hardware encoding control the scale-out speed, which cannot be customized.

In this design logic, users cannot customize the speed of HPA scaling. However, different business scenarios may have different requirements for scaling sensitivity:

1. For key businesses with traffic surges, a scale-out needs to be fast (if needed), and a scale-in needs to be slow (to avoid another traffic peak).
2. Applications processing key data should be scaled out as soon as possible when the data volume surges, so as to speed up data processing. When the data volume decreases, they should be scaled in as soon as possible to reduce costs. Unnecessary and frequent scaling operations are acceptable when the data volume jitters momentarily.
3. Businesses processing general data/network traffic can be scaled in a general way to reduce jitters.

HPA is updated on K8s 1.18, where scaling sensitivity control is added to v2beta2, but the version number of v2beta2 remains unchanged.

Principles and Misunderstandings

During HPA scaling, the fixed algorithm is first used to calculate the desired number of replicas:

Desired number of replicas = ceil[current number of replicas * (current metric / desired metric)]

Here, if "current metric / desired metric" is close to 1 (which is within the default tolerance of 0.1, that is, the ratio ranges between 0.9 and 1.1), no scaling is performed; otherwise, jitters may cause frequent scaling.

©2013-2022 Tencent Cloud. All rights reserved.

Note：
Tolerance is determined by the `--horizontal-pod-autoscaler-tolerance` parameter of `kube-controller-manager`. It defaults to 0.1, that is, 10%.

Scaling speed adjustment described in this document doesn't mean adjusting the algorithm for calculating the desired number of replicas. It doesn't increase/decrease the scaling ratio or quantity, but only controls the scaling speed. The implementation should deliver the following effect: controlling the maximum custom ratio/number of Pods that can be added/released in a custom time period allowed by HPA.

How to Use

In this update, the `behavior` field is added to HPA Spec, which contains the `scaleUp` and `scaleDown` fields for scaling control. For more information, see HPAScalingRules v2beta2 autoscaling.

Sample code

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
name: web
spec:
minReplicas: 1
maxReplicas: 1000
metrics:
- pods:
metric:
name: k8s_pod_rate_cpu_core_used_limit
target:
averageValue: "80"
type: AverageValue
type: Pods
scaleTargetRef:
apiVersion: apps/v1
kind: Deployment
name: web
behavior: # This is the key point.
scaleDown:
stabilizationWindowSeconds: 300 # When a scale-in is needed, observe for five min
utes first. If it is still needed, perform the scale-in.
policies:
- type: Percent
value: 100 # Allow for releasing all
```

©2013-2022 Tencent Cloud. All rights reserved.

```
periodSeconds: 15
scaleUp:
stabilizationWindowSeconds: 0 # Perform a scale-out when needed
policies:
- type: Percent
value: 100
periodSeconds: 15 # Up to one time the current number of Pods can be added every
15 seconds.
- type: Pods
value: 4
periodSeconds: 15 # Up to four Pods can be added every 15 seconds.
selectPolicy: Max # Use the larger value of the two calculated based on the above
two scale-out policies
```

Notes

- The above `behavior` configuration is default, which means it will be added by default if not specified.
- You can configure one or more policies for `scaleUp` and `scaleDown`. `selectPolicy` determines which policy to use for scaling.
- `selectPolicy` is `Max` by default, that is, different calculation results are evaluated and the largest number of Pods is selected for scaling.
- `stabilizationWindowSeconds` is the stable window period, that is, scaling is performed only when the metric is below or above the threshold for the stable window period. This is to avoid frequent scaling caused by jitters. For a scale-out, the stable window defaults to 0, indicating to perform the scale-out immediately; for a scale-in, it defaults to five minutes.
- `policies` defines the scaling policy. `type` can be `Pods` or `Percent`, indicating the maximum number or ratio of replicas that can be added every `periodSeconds`.

Scenarios and Samples

Fast scale-out

If you need to quickly scale out your application, you can use the following HPA configuration:

```
behavior:
scaleUp:
policies:
- type: Percent
value: 900
periodSeconds: 15 # Up to nine times the current number of replicas can be added
every 15 seconds.
```

©2013-2022 Tencent Cloud. All rights reserved.

The above configuration indicates that nine times the current number of replicas are added immediately, within the `maxReplicas` limit though.
Suppose there is only one Pod, the traffic surges, and the metric constantly exceeds nine times the threshold, a scale-out will be performed quickly, during which the number of Pods will change as follows:

```
1 -> 10 -> 100 -> 1000
```

If no scale-in policy is configured, a scale-in will be performed after the global default time window (which is five minutes by default).

Fast scale-out and slow scale-in

When the traffic peak is over and the concurrent volume drops significantly, if the default scale-in policy is used, the number of Pods will drop a few minutes later. If another traffic peak comes unexpectedly after the scale-in, the scale-out will be fast but still take some time. If the traffic surges to a really high level, the backend may fail to keep up, causing some requests to fail. In this case, you can add a scale-in policy for HPA by configuring `behavior` as follows:

```
behavior:
scaleUp:
policies:
- type: Percent
value: 900
periodSeconds: 15 # Up to nine times the current number of replicas can be added
every 15 seconds.
scaleDown:
policies:
- type: Pods
value: 1
periodSeconds: 600 # Only one Pod can be released every ten minutes.
```

In the above sample, the `scaleDown` configuration is added, specifying that only one Pod can be released every ten minutes. This greatly slows down the scale-in, during which the number of Pods will change as follows:

```
1000 -> ... (10 minutes later) -> 999
```

In this way, key businesses will be able to handle traffic surges, and the requests won't fail.

Slow scale-out

If you want to make scale-outs slow and stable for general applications, add the following `behavior` configuration to HPA:

©2013-2022 Tencent Cloud. All rights reserved.

```
behavior:
scaleUp:
policies:
- type: Pods
value: 1
periodSeconds: 300 # Only one Pod can be added every five minutes.
```

Suppose there is only one Pod and the metric constantly exceeds the threshold, the number of Pods will change as follows during the scale-out:

```
1 -> 2 -> 3 -> 4
```

Disabling automatic scale-in

If you want to prevent key applications from an automatic scale-in after a scale-out and need to determine the scale-in conditions by manual intervention or a self-developed controller, you can use the following `behavior` configuration to disable automatic scale-in:

```
behavior:
scaleDown:
selectPolicy: Disabled
```

Extending the time window for scale-in

By default, the time window for scale-in is five minutes. If you need to extend the time window to avoid exceptions caused by traffic peaks, you can specify the time window for scale-in by configuring `behavior` as follows:

```
behavior:
scaleDown:
stabilizationWindowSeconds: 600 # Perform a scale-in ten minutes later
policies:
- type: Pods
value: 5
periodSeconds: 600 # Up to five Pods can be released every ten minutes.
```

In the above sample, when the load drops, a scale-in will be performed 600 seconds (ten minutes) later, and up to five Pods can be released every ten minutes.

Extending the time window for scale-out

Some applications often undergo frequent scale-outs due to data spikes, and the added Pods may be a waste of resources. In data processing pipelines, the desired number of replicas depends on the number of events in the

©2013-2022 Tencent Cloud. All rights reserved.

queue. When a large number of events are heaped in the queue, a fast but not too sensitive scale-out is desired, as the heap may last only a short time and disappear even if no scale-out is performed.

The default scale-out algorithm executes a scale-out after a short period of time. You can add a time window to avoid resource waste after a scale-out caused by spikes. Below is the sample `behavior` configuration:

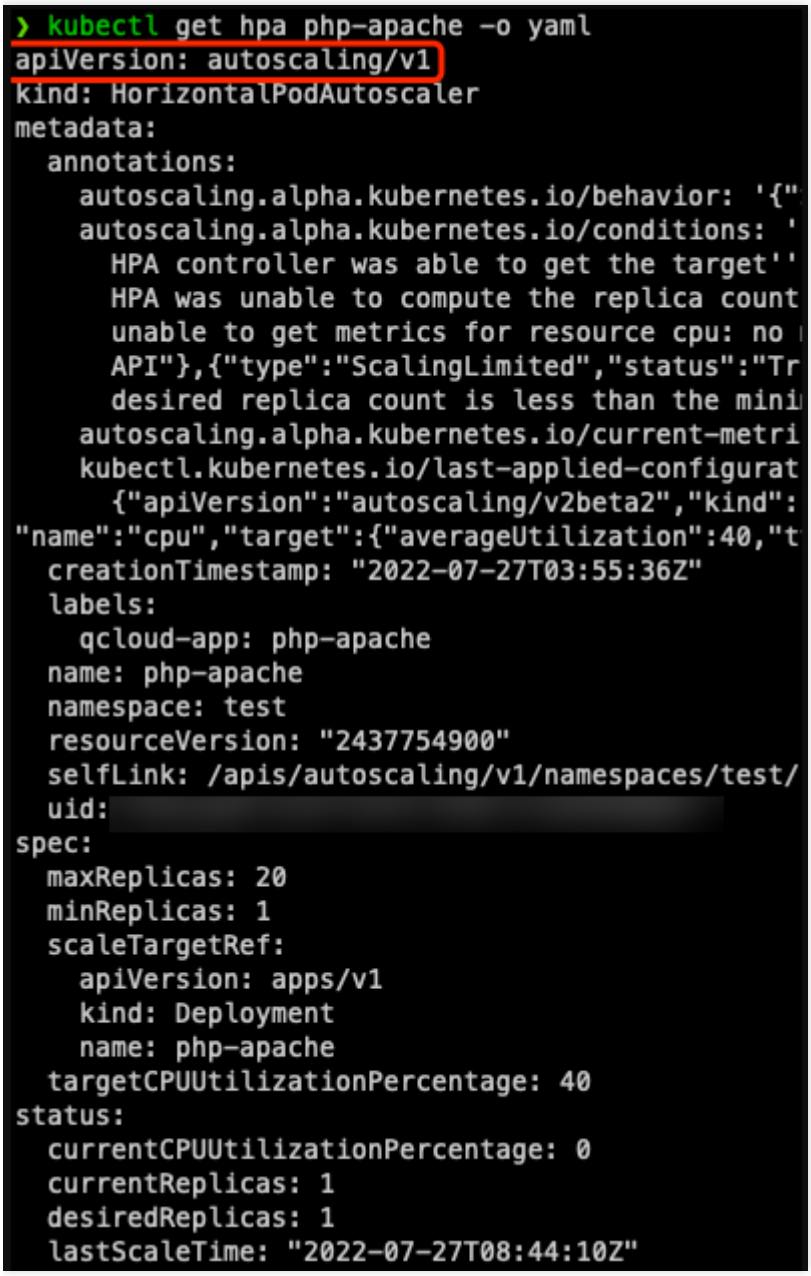
```
behavior:
scaleUp:
stabilizationWindowSeconds: 300 # A scale-out is performed after a 5-minute time window.
policies:
- type: Pods
value: 20
periodSeconds: 60 # Up to 20 Pods can be added every minute.
```

In the above sample, a scale-out is performed after a 5-minute time window. If the metric falls below the threshold during this window, no scale-out is performed. If the metric constantly exceeds the threshold, a scale-out is performed, and up to 20 Pods can be added every minute.

FAQs

Why is YAML on v1 or v2beta1 obtained after a HPA is created by using v2beta2?

©2013-2022 Tencent Cloud. All rights reserved.



This is because HPA has many API versions:

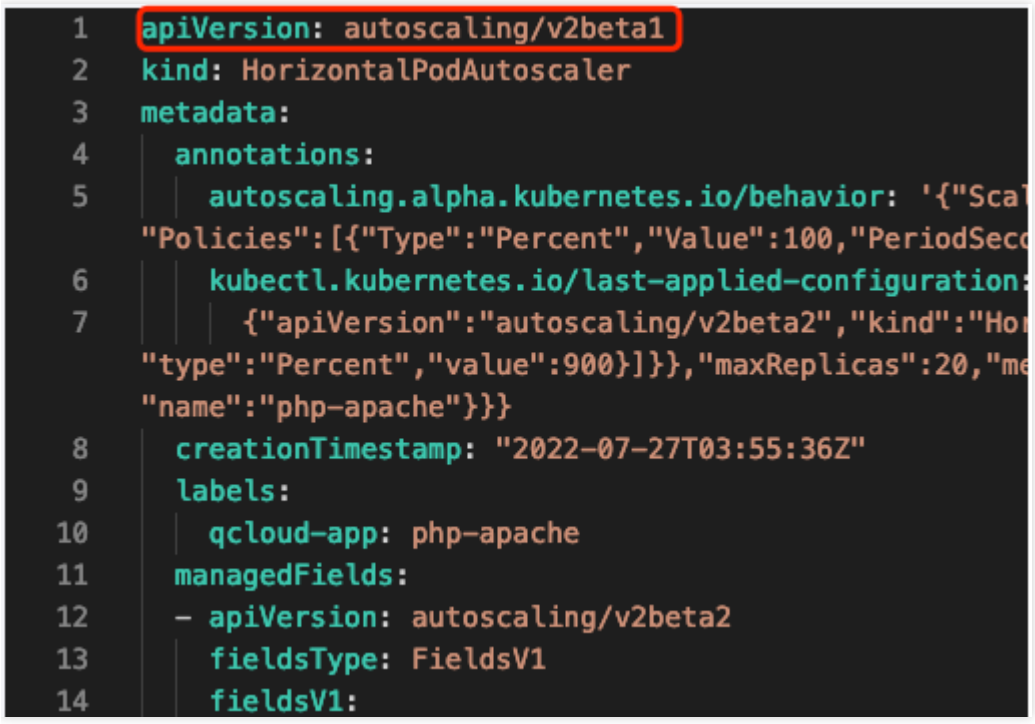
```
kubectl api-versions | grep autoscaling
autoscaling/v1
autoscaling/v2beta1
autoscaling/v2beta2
```

The version number is irrelevant to the version for creation (which is automatically converted).

If kubectl is used, during API discovery, various types of resources and version information returned by the API server will be cached. Some resources are available in multiple versions; if the version to get is not specified, the default

©2013-2022 Tencent Cloud. All rights reserved.

version will be used, which is v1 for HPA. If the operation is performed on some platform UIs, the result will depend on the platform implementation method. In the TKE console, the default version is v2beta1:



How do I use the v2beta2 version to get or edit?

Just specify the complete resource name containing the version information:

```
kubectl get horizontalpodautoscaler.v2beta2.autoscaling php-apache -o yaml
# kubectl edit horizontalpodautoscaler.v2beta2.autoscaling php-apache
```

Why is a scale-out slow when it is configured to be fast?

Add the following configuration:

```
behavior:
scaleUp:
policies:
- type: Percent
value: 900
periodSeconds: 10
```

It indicates that up to nine times the current number of Pods can be added every ten seconds. In actual tests, it happens that the scale-out is slow when the threshold is greatly exceeded.

Generally, it's due to the calculation period and metric latency:

©2013-2022 Tencent Cloud. All rights reserved.

- There is a period for calculating the desired number of replicas, which defaults to 15 seconds (determined by the `--horizontal-pod-autoscaler-sync-period` parameter of `kube-controller-manager`).
- During each calculation, the corresponding metric API is used to get the current monitoring metric value, which is usually not returned in real time. For the TKE service, monitoring data is reported once every minute. For self-built Prometheus and Prometheus Adapter, monitoring data is updated according to the monitoring data scrape interval, and the `--metrics-relist-interval` parameter in Prometheus Adapter determines the monitoring metric refresh period (which can be queried in Prometheus); the sum of the two is the longest period for a monitoring data update.

Generally, extreme HPA sensitivity is not necessary, and a certain latency is acceptable. In highly sensitive scenarios, you can use Prometheus to shorten the monitoring metric scrape interval and `--metrics-relist-interval` of the Prometheus Adapter.

Summary

This document describes how to use new HPA features to control the scaling speed so as to meet the requirements in different scenarios. It also provides some common scenarios and configuration samples that can be used as needed.

References

- Horizontal Pod Autoscaling
- Configurable scale up/down velocity for HPA

©2013-2022 Tencent Cloud. All rights reserved.

Implementing elasticity based on traffic prediction with EHPA

Last updated：2024-05-24 15:26:52

EHPA introduction

Effective Horizontal Pod Autoscaler (EHPA) is an auto scaling product provided by the Crane open source project. It is based on the community HPA for underlying elastic control and supports a richer set of elastic trigger policies (prediction, observation, and cycle) to enhance the efficiency of elastic control and ensure service quality.
Key features of EHPA include:
Preemptive scaling to ensure service quality: By predicting future traffic peaks, it preemptively scales up to avoid avalanches and service stability failures due to untimely scaling.
Reducing invalid scaling-down: By predicting future demand, it minimizes unnecessary scaling-down, stabilizes the resource utilization rate of workloads, and eliminates spike misjudgments.
Support to Cron configuration: It supports elastic configuration based on Cron to cope with abnormal traffic peaks during major promotion activities.
Community compatibility: It uses the community HPA as the execution layer for elastic control, fully compatible with the community.

Installation

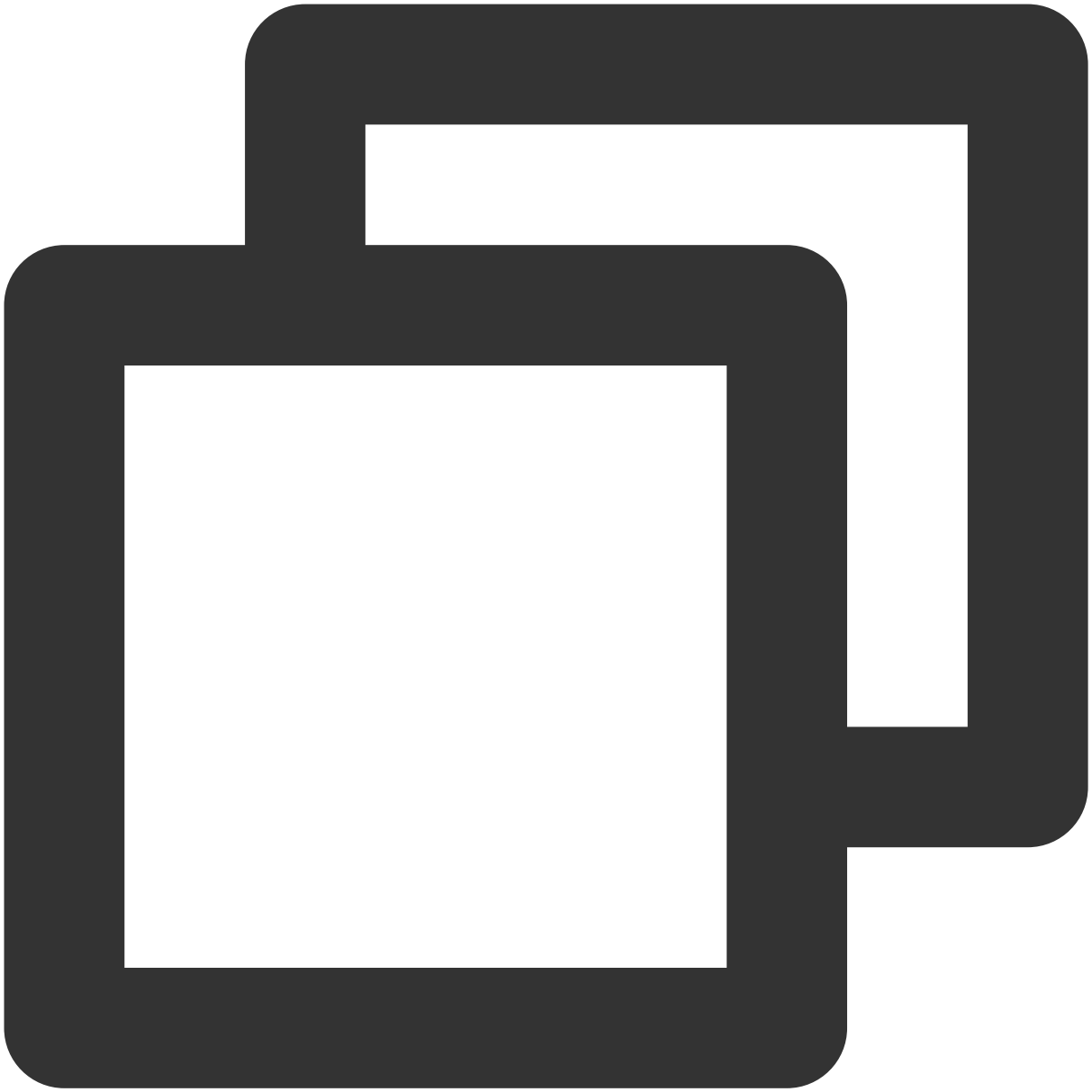
For EHPA installation, refer to the Crane open source project documentation. Refer to Intelligent Autoscaling Practices Based on Effective HPA for Custom Metrics for details.

Product features

Elasticity based on prediction

The load of most online applications is periodic. Users can predict future loads based on daily or weekly trends. EHPA uses DSP algorithms to predict future time series data of applications.
The following code is an example of an EHPA template with predictive capabilities enabled:

©2013-2022 Tencent Cloud. All rights reserved.

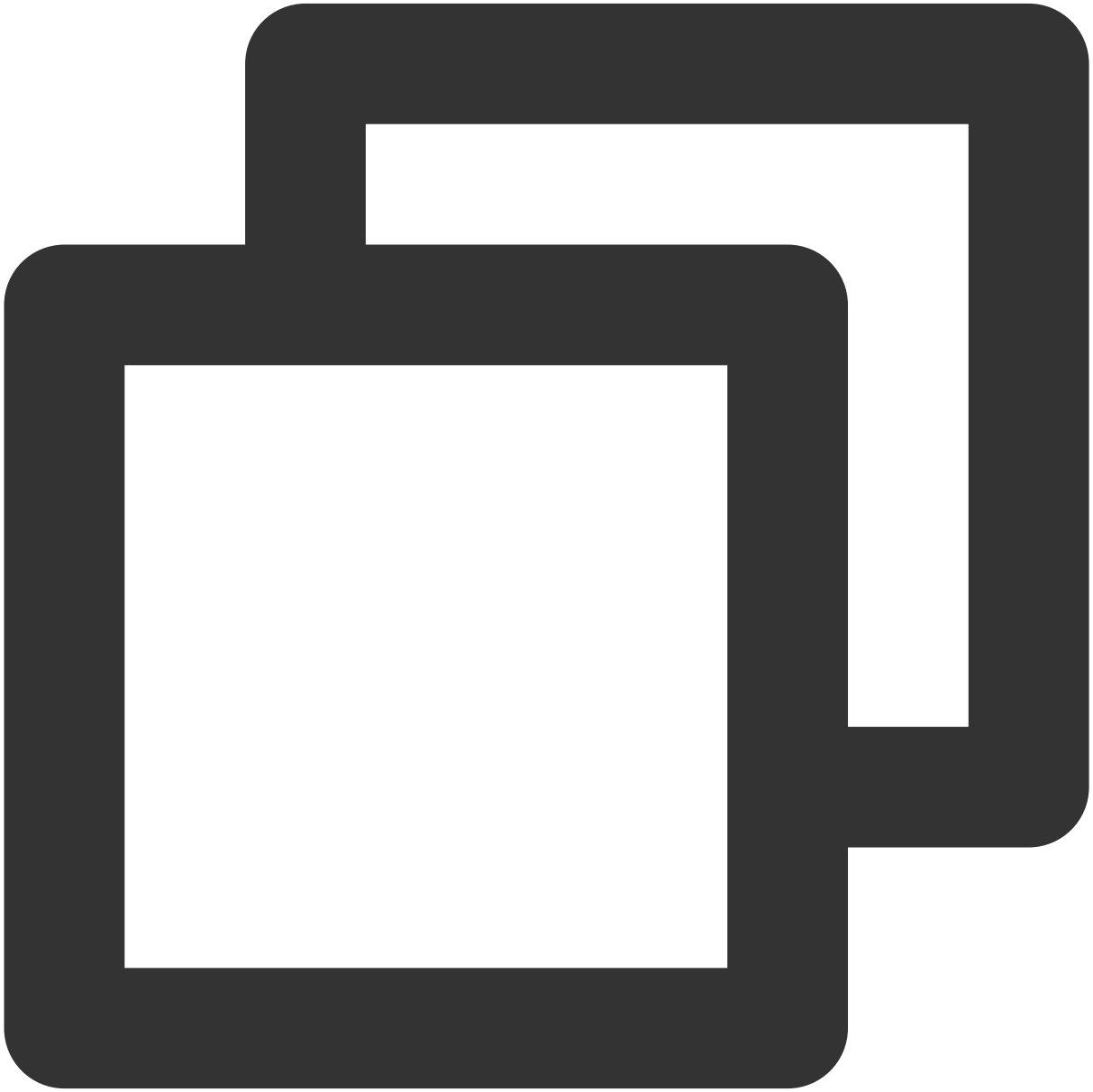


```
apiVersion: autoscaling.crane.io/v1alpha1
kind: EffectiveHorizontalPodAutoscaler
spec:
    prediction:
      predictionWindowSeconds: 3600
      predictionAlgorithm:
        algorithmType: dsp
        dsp:
        sampleInterval: "60s"
        historyLength: "3d"
```

©2013-2022 Tencent Cloud. All rights reserved.

Fallback for monitoring data

When using predictive algorithms for prediction, you may be concerned about the accuracy of the predictive data. Therefore, when calculating the number of replicas, EHPA will not only calculate based on predictive data but also consider actual monitoring data for fallback to enhance the safety of elasticity. The specific implementation principle is, when you define spec.metrics in EHPA and enable elasticity prediction, the EffectiveHPAController will automatically generate multiple Metric Specs to create the underlying management's HPA.
For example, when a user defines the following Metric Spec in an EHPA YAML file:

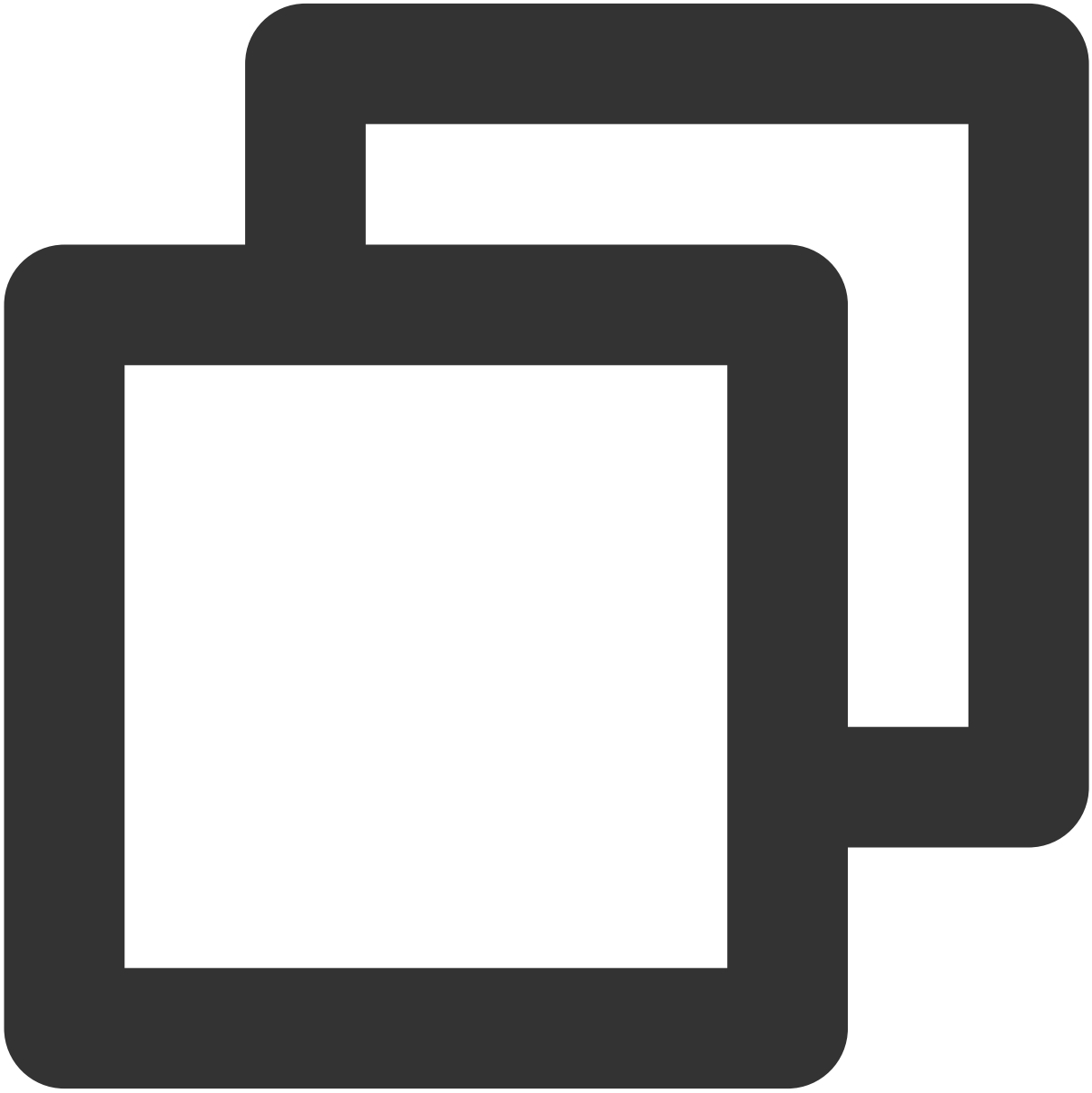


```
apiVersion: autoscaling.crane.io/v1alpha1
kind: EffectiveHorizontalPodAutoscaler
```

©2013-2022 Tencent Cloud. All rights reserved.

```
spec:
    metrics:
    - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
```

The system will be automatically converted to two HPA threshold configurations:



```
apiVersion: autoscaling/v2beta1
```

©2013-2022 Tencent Cloud. All rights reserved.

```
kind: HorizontalPodAutoscaler
spec:
  metrics:
    - pods:
        metric:
          name: crane_pod_cpu_usage
          selector:
              matchLabels:
                autoscaling.crane.io/effective-hpa-uid: f9b92249-eab9-4671-afe0-179
        target:
          type: AverageValue
          averageValue: 100m
      type: Pods
    - resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 50
      type: Resource
```

In the example above, the Metric threshold configurations created by the user in EHPA will automatically be converted into two Metric threshold configurations on the underlying HPA: predictive metric threshold and actual monitored metric threshold.

The predictive metric threshold is a custom metric. Its value is provided by MetricAdapter of Crane.

The actual monitored metric threshold is a resource metric, the same as defined by the user in EHPA. Therefore, HPA will calculate the number of replicas based on the metrics actually monitored by the application.

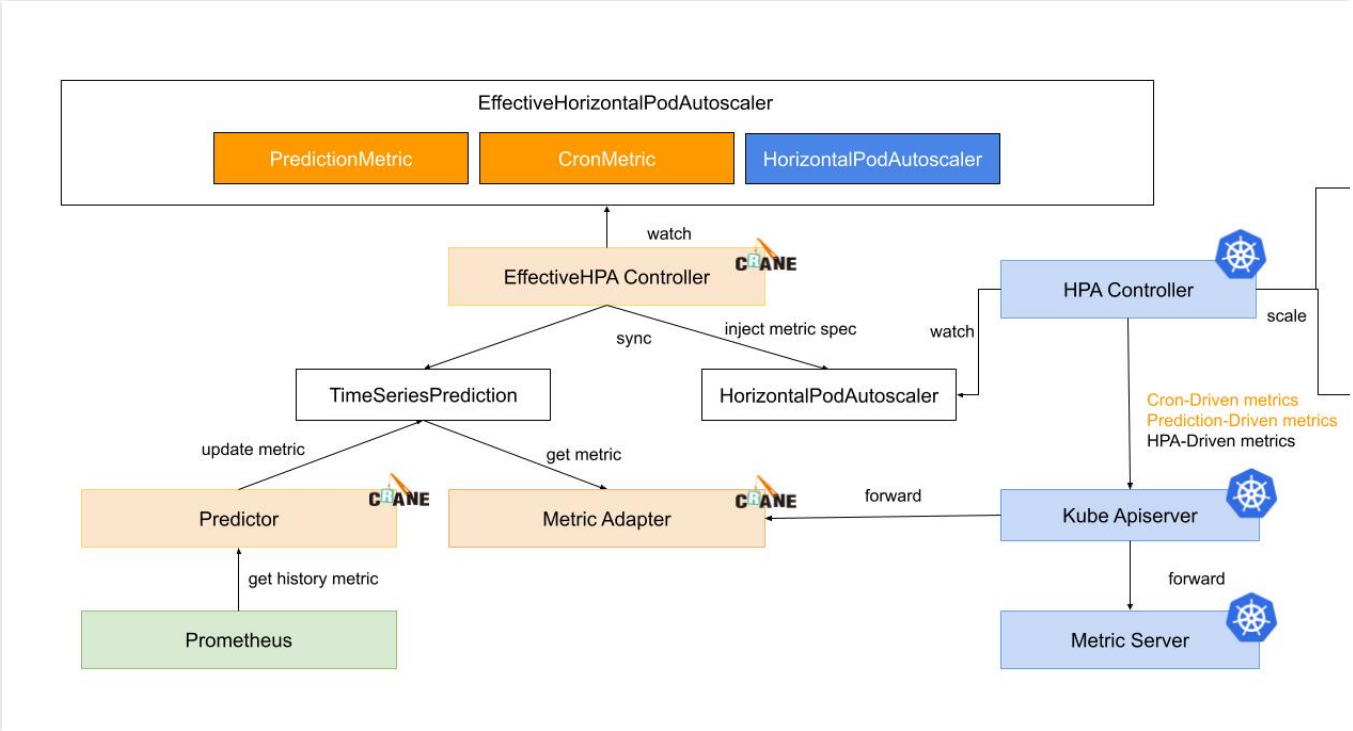
When multiple elastic metric thresholds are configured, HPA will calculate the number of replicas for each metric separately and select the highest number of replicas as the final recommended result for elasticity.

Horizontal elasticity execution process

1. EffectiveHPAController creates HorizontalPodAutoscaler and TimeSeriesPrediction objects.
2. PredictionCore retrieves historical metrics from Prometheus, calculates them using the predictive algorithm, and records the results to TimeSeriesPrediction.
3. HPAController reads metric data from KubeApiServer through the metric client.
4. KubeApiServer routes the request to MetricAdapter of Crane.
5. HPAController calculates the results returned by all metrics to determine the final elasticity replica recommendation.
6. HPAController uses the scale API to scale up/down the target application.

The overall flowchart is as shown below:

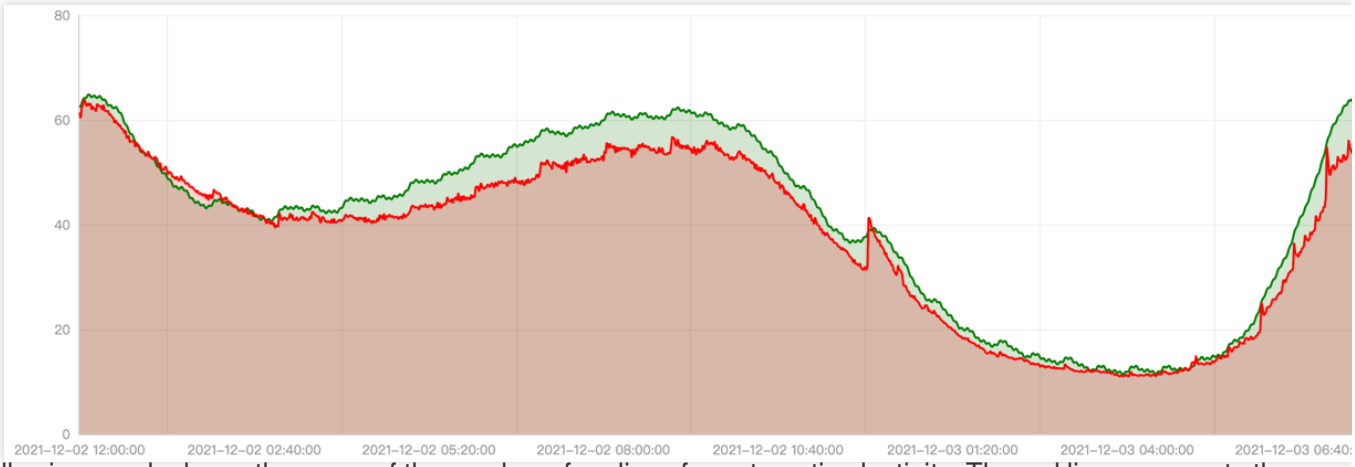
©2013-2022 Tencent Cloud. All rights reserved.



User cases

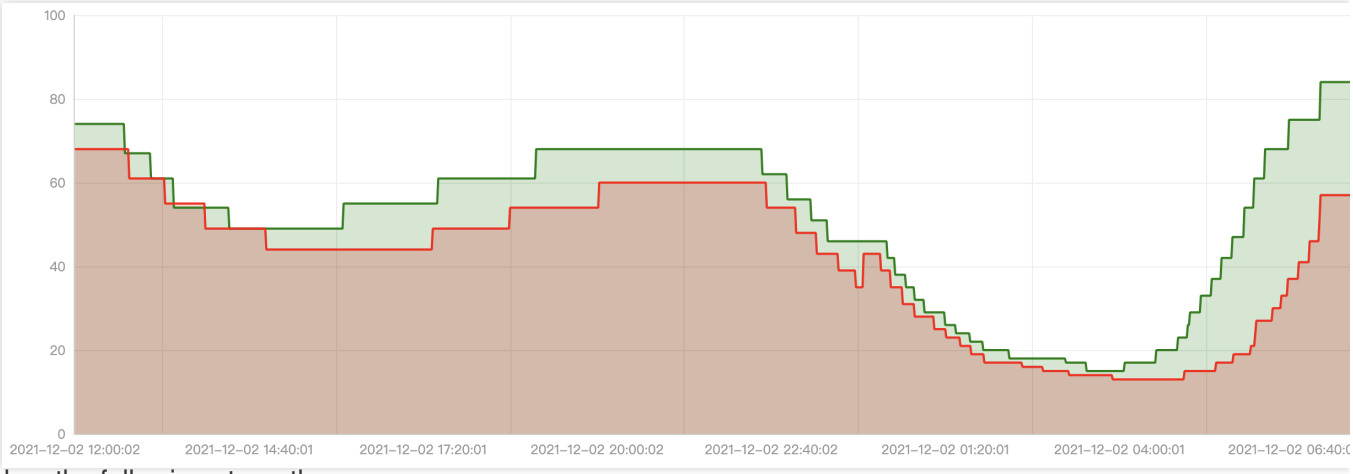
The actual effect of EHPA is demonstrated through a customer case in the production environment. In this case, the production data is replayed in the pre-release environment and the elasticity effects of using EHPA and the community HPA are compared.

The red line in the following figure represents the actual CPU utilization curve of the application within one day. You can see usage peaks at 8 am, 12 pm, and 8 pm. The green line represents the CPU utilization predicted by EHPA.



The following graph shows the curve of the number of replicas for automatic elasticity. The red line represents the number of replicas curve for the community HPA, and the green line represents the number of replicas curve for EHPA.

©2013-2022 Tencent Cloud. All rights reserved.



EHPA has the following strengths:

It scales before the arrival of traffic peaks.

It reduces ineffective scale-downs when the traffic first drops then immediately rises.

Compared to HPA, EHPA has fewer elasticity operations but is more efficient.

Scale policies

EHPA offers two types of scale policies: Auto and Preview. You can switch between them at any time and it will take effect immediately.

Auto

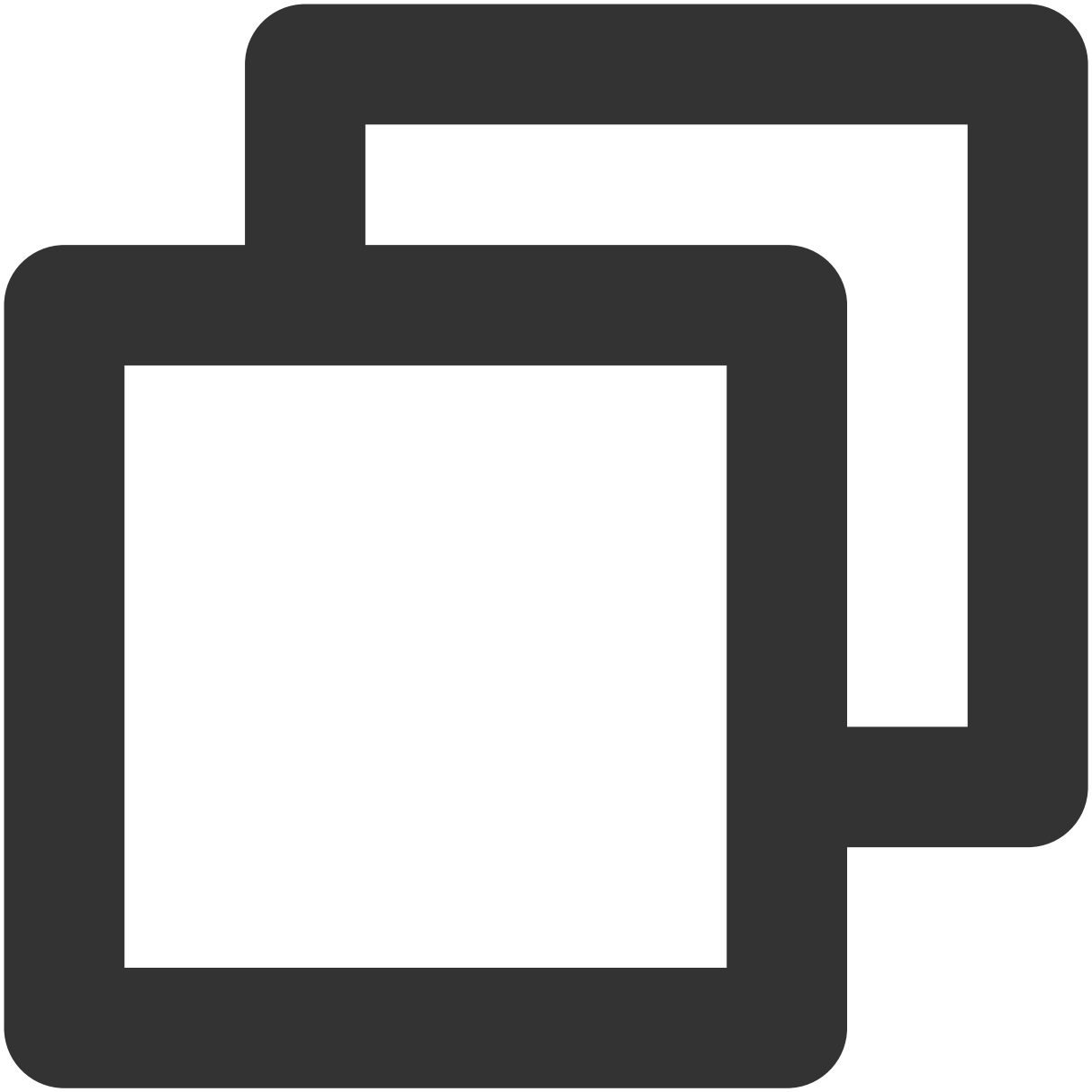
Under the Auto policy, EHPA automatically performs scaling. By default, EHPA's policy is set to Auto. In this mode, EHPA creates a community HPA object and automatically takes over its lifecycle. We recommend that you not modify or control this underlying HPA object, as it will be deleted when EHPA is removed.

Preview

The Preview policy provides the ability for EHPA not to automatically execute scaling. Therefore, you can observe the number of replicas calculated by EHPA through the desiredReplicas field. You can switch between the two modes at any time. When you switch to the Preview mode, you can adjust the number of application replicas through spec.specificReplicas. If spec.specificReplicas is empty, scaling is not performed for the application, but the number of replicas is still calculated.

Below is an example of an EHPA template configured in the Preview mode:

©2013-2022 Tencent Cloud. All rights reserved.



```
apiVersion: autoscaling.crane.io/v1alpha1
kind: EffectiveHorizontalPodAutoscaler
spec:
    scaleStrategy: Preview   # ScaleStrategy indicates the policy to scaling target
    pecificReplicas: 5      # SpecificReplicas specify the target replicas.
status:
    expectReplicas: 4        # expectReplicas is the calculated replicas that based
    currentReplicas: 4       # currentReplicas is actual replicas from target
```

©2013-2022 Tencent Cloud. All rights reserved.

©2013-2022 Tencent Cloud. All rights reserved.

Implementing Horizontal Scaling based on CLB monitoring metrics using KEDA in TKE

Last updated：2024-05-31 09:43:18

Business scenario

Business traffic on TKE usually enters through CLB (Tencent Cloud Load Balancer). Sometimes, you may want the workload to scale based on CLB's monitoring metrics, for example:
1. Long connection scenarios (such as game rooms and online meetings): Each user corresponds to one connection, and the maximum number of connections handled by each pod in the workload is relatively fixed. In this case, scaling can be performed based on the CLB connection count metric.
2. Online services using the HTTP protocol: The metric Queries Per Second (QPS) that a single pod in the workload can support is relatively fixed. In this case, scaling can be performed based on the CLB QPS metric.

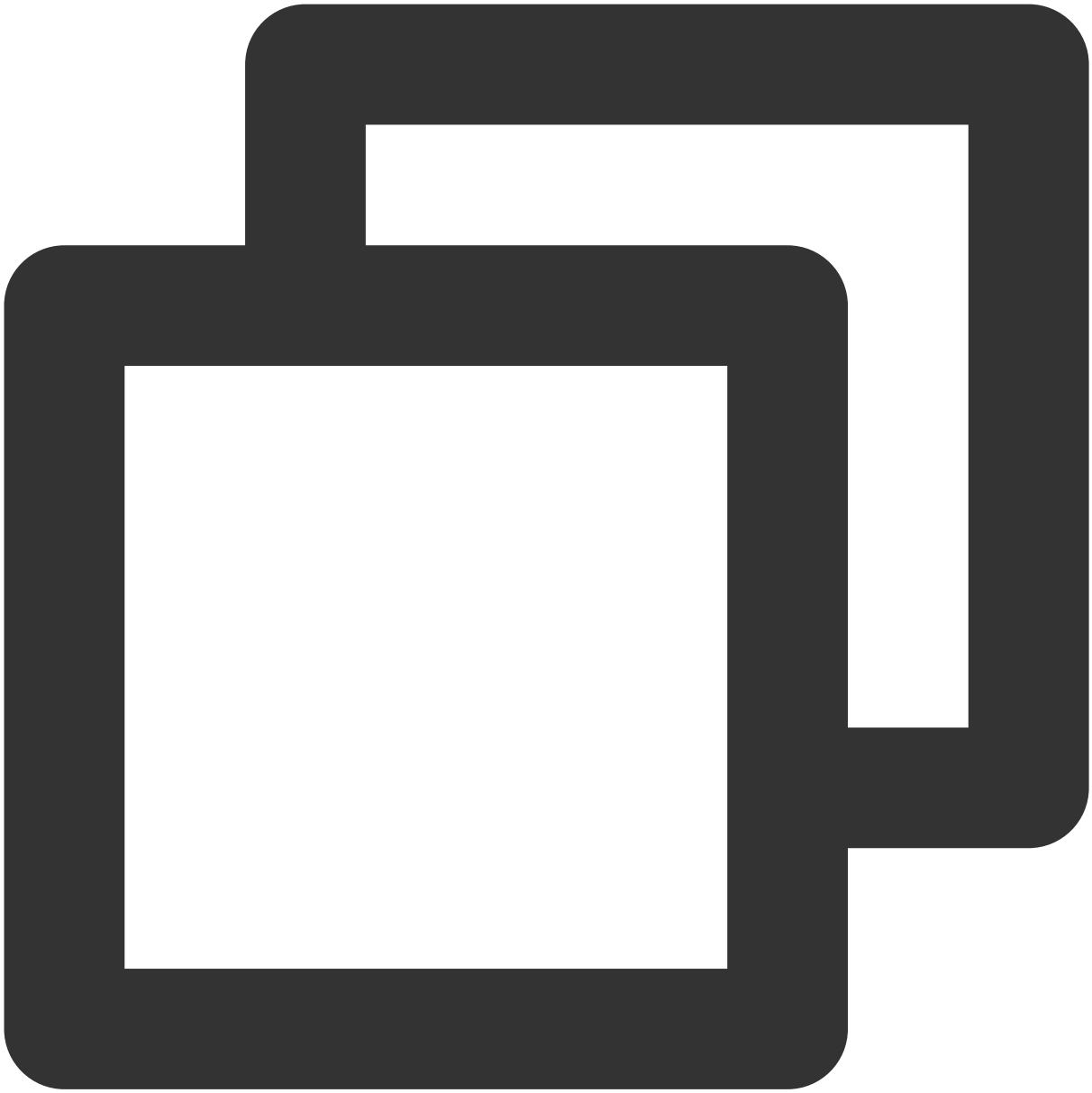
Introduction to keda-tencentcloud-clb-scaler

KEDA has many built-in triggers, but not for Tencent Cloud CLB. However, KEDA supports external triggers for expansion. The keda-tencentcloud-clb-scaler is a KEDA External Scaler based on Tencent Cloud CLB monitoring metrics, capable of auto scaling based on metrics such as CLB connection count, QPS, and bandwidth.

Directions

Installing keda-tencentcloud-clb-scaler

©2013-2022 Tencent Cloud. All rights reserved.



```
helm repo add clb-scaler https://imroc.github.io/keda-tencentcloud-clb-scaler
helm upgrade --install clb-scaler clb-scaler/clb-scaler -n keda \\
  --set region="ap-chengdu" \\
  --set credentials.secretId="xxx" \\
  --set credentials.secretKey="xxx"
```

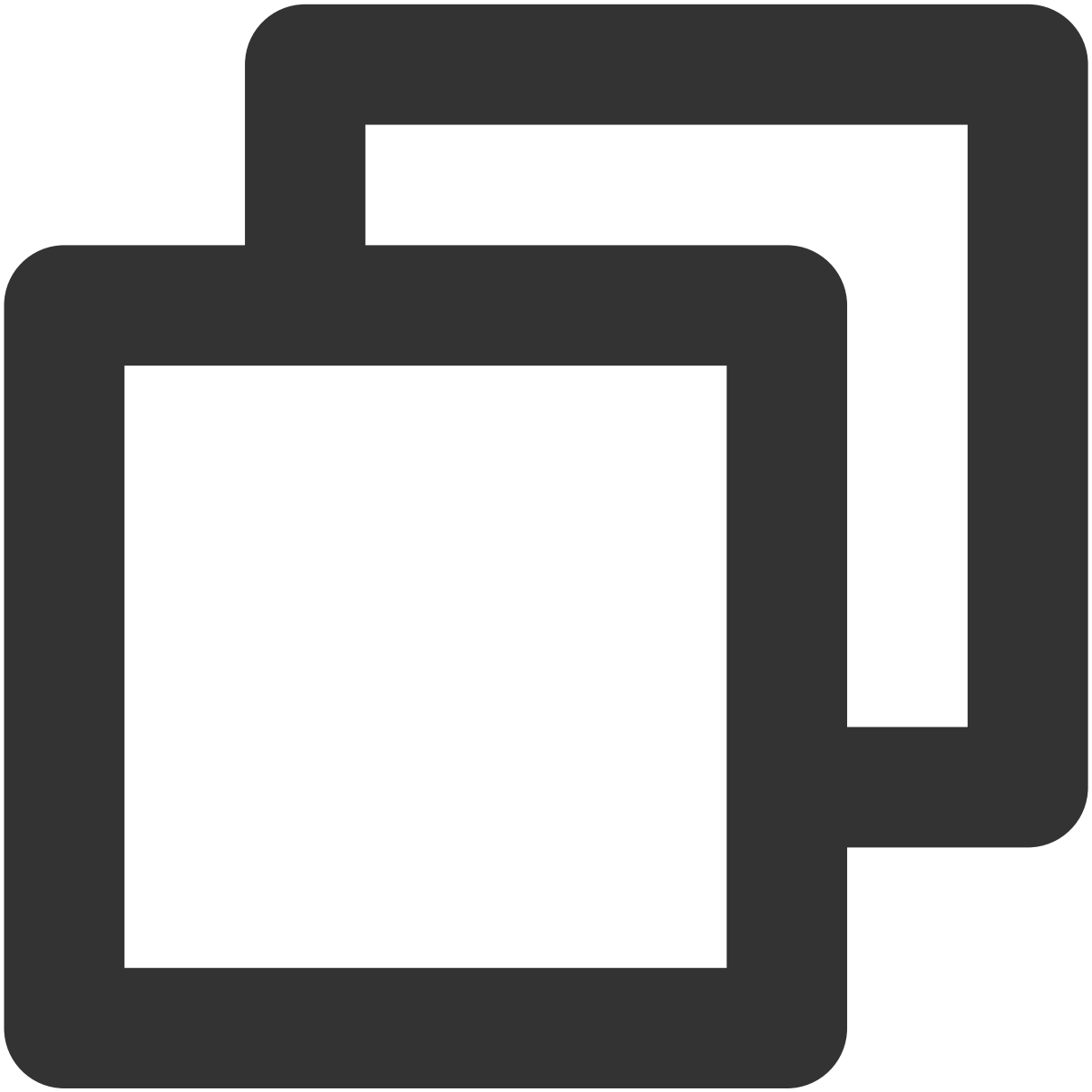
Replace the region with the one where your CLB instance is located (usually the same as the cluster region). For details on the region list, refer to Regions and availability zones.

The credentials.secretId and credentials.secretKey are the key pair of your Tencent Cloud account, used to access and retrieve CLB monitoring data. Replace them with your own key pair.

©2013-2022 Tencent Cloud. All rights reserved.

Deploying workload

You can use the following workload YAML sample for testing:



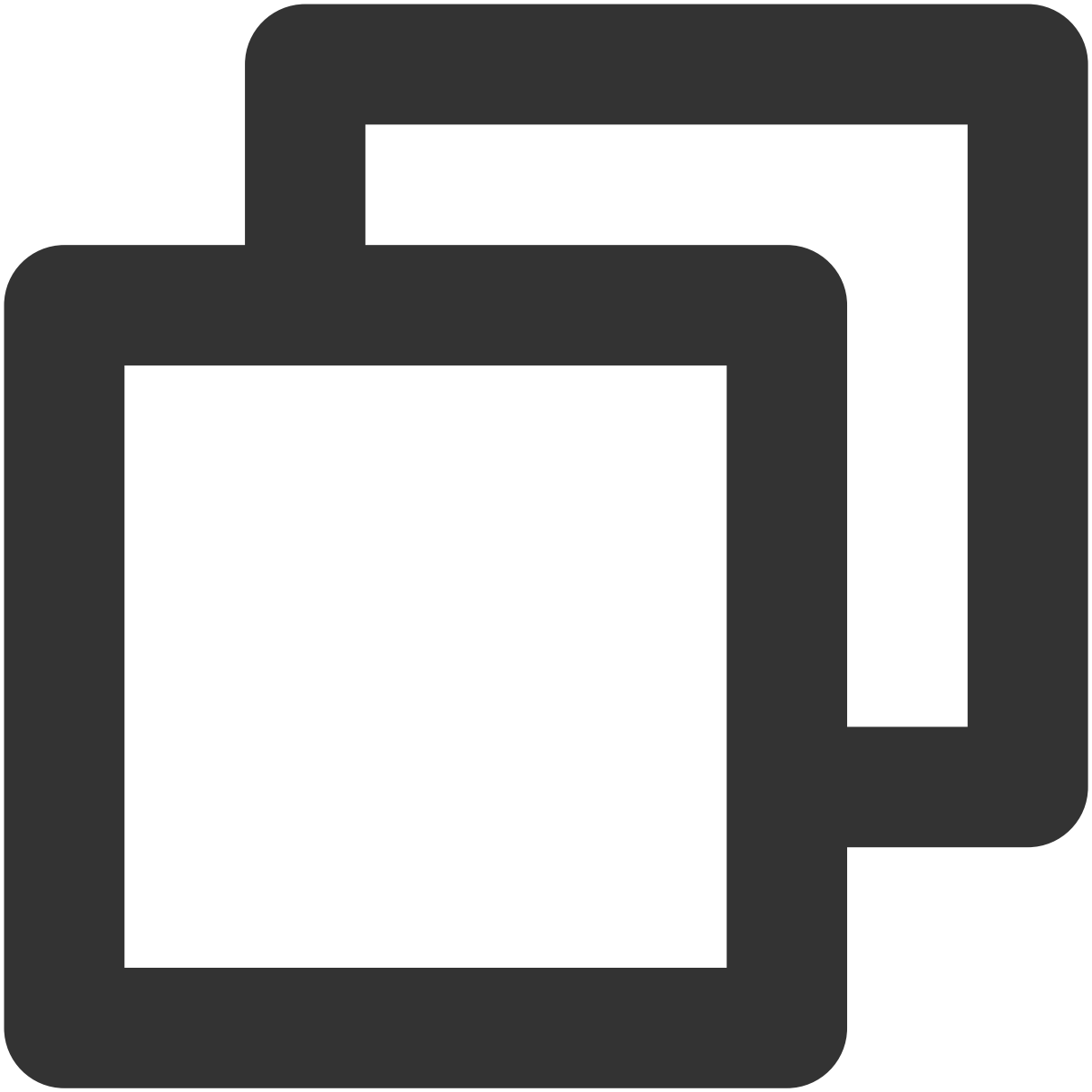
```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: httpbin
  name: httpbin
spec:
  ports:
```

©2013-2022 Tencent Cloud. All rights reserved.

```
    - port: 8080
      protocol: TCP
      targetPort: 80
  selector:
    app: httpbin
  type: LoadBalancer

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: httpbin
spec:
  replicas: 1
  selector:
    matchLabels:
      app: httpbin
  template:
    metadata:
      labels:
        app: httpbin
    spec:
      containers:
        - image: kennethreitz/httpbin:latest
          name: httpbin
```

After the workload is deployed, a corresponding public network CLB instance will be automatically created to manage traffic. You can run the following command to obtain the CLB ID:

©2013-2022 Tencent Cloud. All rights reserved.



```
$ kubectl svc httpbin -o jsonpath='{.metadata.annotations.service\\.kubernetes\\.io
lb-********
```

Record the obtained CLB ID, which will be used in subsequent KEDA configurations.

Using ScaledObject to configure auto scaling based on CLB monitoring metrics

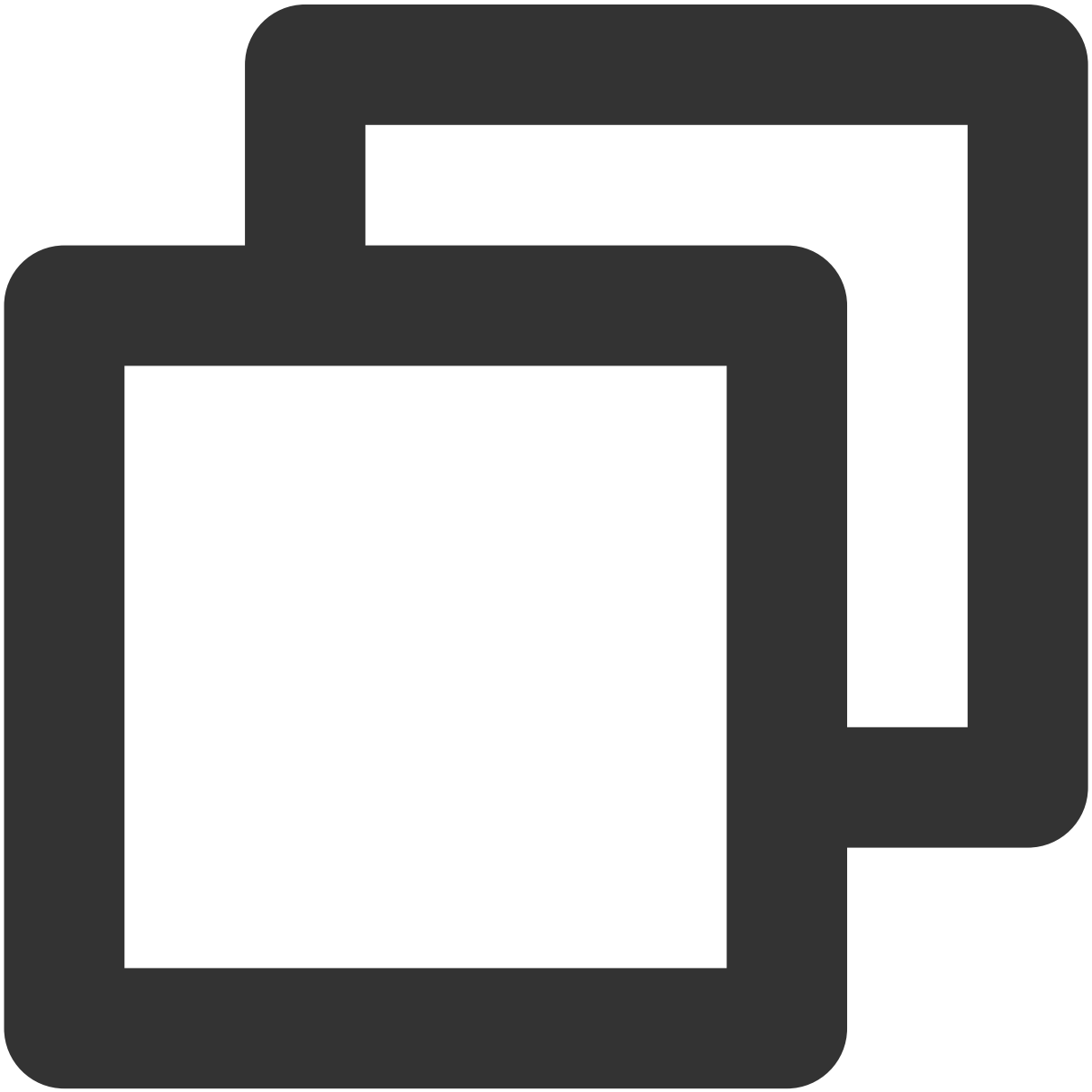
Configuration method

CLB-based monitoring metrics are commonly used for online services. Configure auto scaling using KEDA's ScaledObject, set the trigger type to external, and input the required metadata, mainly including the following fields:

©2013-2022 Tencent Cloud. All rights reserved.

The scalerAddress is the address used by keda-operator to call keda-tencentcloud-clb-scaler.
The loadBalancerId is the ID of the CLB instance.
The metricName is the name of the CLB monitoring metric. Most metrics are the same for public and private networks.
The threshold is the metric threshold for scaling up or down, meaning that scaling-up or scaling-down is performed by comparing the value of metricValue/number of pods to the threshold.
The listener is the unique optional configuration, specifying the CLB listener for monitoring metrics, in the format of protocol/port.

Configuration example 1: Auto scaling based on the metric CLB connection count

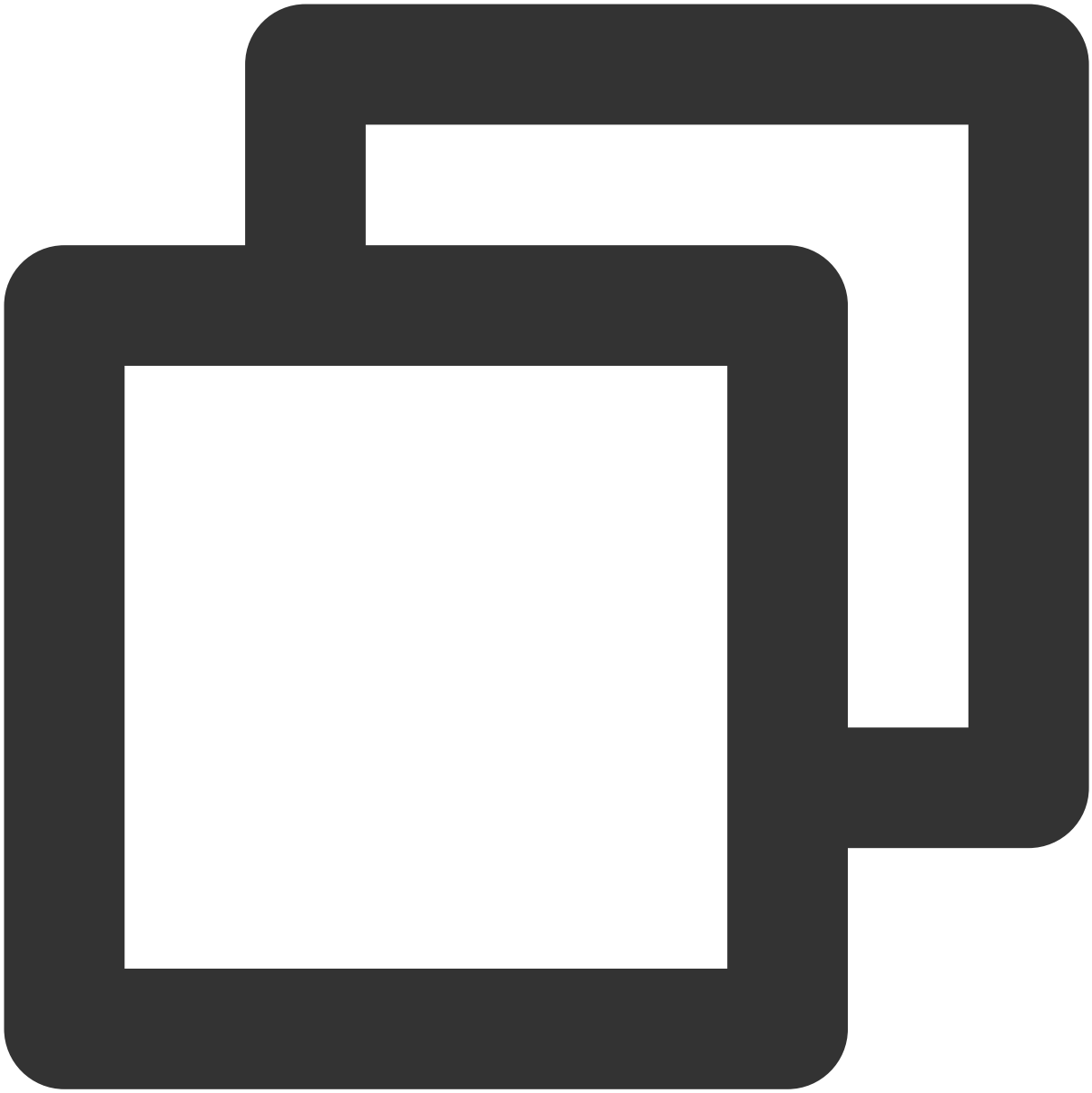


©2013-2022 Tencent Cloud. All rights reserved.

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: httpbin
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: httpbin
  pollingInterval: 15
  minReplicaCount: 1
  maxReplicaCount: 100
  triggers:
    - type: external
      metadata:
        # highlight-start
        scalerAddress: clb-scaler.keda.svc.cluster.local:9000
        loadBalancerId: lb-xxxxxxxx
        metricName: ClientConnnum # Connection count
        threshold: "100" # 100 connections per pod
        listener: "TCP/8080" # Optional. Specifies the listener in the format of pr
        # highlight-end
```

Configuration example 2: Auto scaling based on the metric CLB QPS

©2013-2022 Tencent Cloud. All rights reserved.



```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: httpbin
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: httpbin
  pollingInterval: 15
  minReplicaCount: 1
```

©2013-2022 Tencent Cloud. All rights reserved.

```
  maxReplicaCount: 100
  triggers:
    - type: external
      metadata:
        # highlight-start
        scalerAddress: clb-scaler.keda.svc.cluster.local:9000
        loadBalancerId: lb-xxxxxxxx
        metricName: TotalReq # Requests per second
        threshold: "500" # Average 500 QPS per pod
        listener: "TCP/8080" # Optional. Specifies the listener in the format of pr
        # highlight-end
```

©2013-2022 Tencent Cloud. All rights reserved.

Storage
Backing up and Restoring PVC via CBS-CSI Add-on

Last updated：2022-04-21 12:09:40

Operation Scenarios

If you need to create a snapshot of the PVC data disk to backup data, or to restore the backup snapshot data to a new PVC, you can use the CBS-CSI add-on. This document describes how to use the CBS-CSI add-on to implement data backup and restoration of PVC.

Prerequisites

- You have created a TKE cluster or created a Kubernetes cluster in Tencent Cloud CVM. The cluster version is v1.18 or later.
- You have installed CBS-CSI add-on.
- You have granted related permissions of CBS snapshot for `TKE_QCSRole` on the Access Management page of the console. For details, see CBS-CSI.

Operation Directions

Restoring PVC

Creating a VolumeSnapshotClass

1. Use the following YAML to create a VolumeSnapshotClass object, as shown below:

```
apiVersion: snapshot.storage.k8s.io/v1beta1
kind: VolumeSnapshotClass
metadata:
name: cbs-snapclass
driver: com.tencent.cloud.csi.cbs
deletionPolicy: Delete
```

©2013-2022 Tencent Cloud. All rights reserved.

2. Run the following command to check whether the VolumeSnapshotClass has been created successfully, as shown below:

```
$ kubectl get volumesnapshotclass
NAME DRIVER DELETIONPOLICY AGE
cbs-snapclass com.tencent.cloud.csi.cbs Delete 17m
```

Create a PVC snapshot object VolumeSnapshot

1. This document takes `new-snapshot-demo` as the snapshot name to create a VolumeSnapshot. Use the following YAML to create a VolumeSnapshot object, as shown below:

```
apiVersion: snapshot.storage.k8s.io/v1beta1
kind: VolumeSnapshot
metadata:
name: new-snapshot-demo
spec:
volumeSnapshotClassName: cbs-snapclass # Use the VolumeSnapshotClass created in
the above steps
source:
persistentVolumeClaimName: ssd-pvc # Replace it with the PVC name that needs to
be backed up
```

2. Run the following command to check whether the Volumesnapshot and Volumesnapshotcontent objects have been created successfully. If `READYTOUSE` is true, the creation is successful, as shown below:

```
$ kubectl get volumesnapshot
NAME READYTOUSE SOURCEPVC SOURCESNAPSHOTCONTENT RESTORESIZE SNAPSHOTCLASS SNAPS
HOTCONTENT CREATIONTIME AGE
new-snapshot-demo true ssd-pvc 20Gi cbs-snapclass snapcontent-170b2161-f158-4c9
e-a090-a38fdfd84a3e 2m36s 2m50s
$ kubectl get volumesnapshotcontent
NAME READYTOUSE RESTORESIZE DELETIONPOLICY DRIVER VOLUMESNAPSHOTCLASS VOLUMESNA
PSHOT AGE
snapcontent-170b2161-f158-4c9e-a090-a38fdfd84a3e true 21474836480 Delete com.te
ncent.cloud.csi.cbs cbs-snapclass new-snapshot-demo 3m3s
```

3. Run the following command to obtain the snapshot ID of the Volumesnapshotcontent object. The field is `status.snapshotHandle` (here takes snap-rsk8v75j as an example). You can log in to the TKE console .com/tke2) and use the snapshot ID to check whether the snapshot exists, as shown below:

©2013-2022 Tencent Cloud. All rights reserved.

```
$ kubectl get volumesnapshotcontent -o yaml snapcontent-170b2161-f158-4c9e-a090
-a38fdfd84a3e
...
status:
creationTime: 1607331318000000000
readyToUse: true
restoreSize: 21474836480
snapshotHandle: snap-rsk8v75j
```

Restoring data from the snapshot to a new PVC

1. This document takes the VolumeSnapshot object `new-snapshot-demo` created in the above step as an example. Use the following YAML to restore data from the snapshot to a new PVC, as shown below:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
name: restore-test
spec:
storageClassName: ssd-csi # Customize the storage class as needed
dataSource:
name: new-snapshot-demo # Use the VolumeSnapshot created in the above step
kind: VolumeSnapshot
apiGroup: snapshot.storage.k8s.io
accessModes:
- ReadWriteOnce # CBS is block storage, which only supports single machine read
and write
resources:
requests:
storage: 50Gi # The recommended storage capacity is the same as the capacity of
the restored PVC
```

2. Run the following command. You can check that the PVC has been created and bound to the PV, and you can find the corresponding diskid (here takes disk-ju0hw7no as an example) in the PV, as shown below:

```
$ kubectl get pvc restore-test
NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE
restore-test Bound pvc-940edf09-d622-4126-992b-0a209f048c7d 60Gi RWO ssd-topolo
gy 6m8s
$ kubectl get pv pvc-940edf09-d622-4126-992b-0a209f048c7d -o yaml
...
spec:
```

©2013-2022 Tencent Cloud. All rights reserved.

```
...
volumeHandle: disk-ju0hw7no
...
```

Note：
If StorageClass uses topology awareness (schedule the Pod first and then create the PV), that is, to specify `volumeBindingMode: WaitForFirstConsumer`, you need to deploy the Pod (to mount the PVC) to trigger the creation of the PV (create a new CBS from the snapshot and bind it to the PV).

©2013-2022 Tencent Cloud. All rights reserved.

Static Mounting of CFS-Turbo File System
Mounting of CFS-Turbo for TKE Clusters

Last updated：2023-08-03 16:23:11

Overview

You can mount a CFS Turbo storage for a TKE cluster by installing a `kubernetes-csi-tencentloud` add-on. This add-on is used to mount the Tencent Cloud CFS Turbo file system to a workload based on a private protocol. Currently, only static configuration is supported. For more information about CFS storage types, see Storage Types and Performance.

Prerequisites

You have created a TKE cluster or created a Kubernetes cluster on Tencent Cloud, and the cluster version is 1.14 or later.

Directions

Creating a file system

Create a CFS Turbo file system. For details, see Creating File Systems and Mount Targets.

> Note：
> After the file system is created, you need to associate the cluster network (vpc-xx) with the CCN instance of the file system. You can check it in the information about the file system mount target.

Deploying a RBAC policy

If you want to mount a CFS Turbo volume, you need to run the `kubectl apply -f csi-node-rbac.yaml` command to deploy a RBAC policy in the cluster. The following csi-node-rbac.yaml code is for your reference:

```
apiVersion: v1
kind: ServiceAccount
metadata:
name: cfsturbo-csi-node-sa
```

©2013-2022 Tencent Cloud. All rights reserved.

```
namespace: kube-system
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
name: cfsturbo-csi-node-role
rules:
- apiGroups: [""]
resources: ["persistentvolumes", "endpoints", "configmaps"]
verbs: ["get", "list", "watch", "create", "delete", "update"]
- apiGroups: [""]
resources: ["persistentvolumeclaims", "nodes"]
verbs: ["get", "list", "watch", "update"]
- apiGroups: [""]
resources: ["events"]
verbs: ["get", "list", "watch", "create", "update", "patch"]
- apiGroups: [""]
resources: ["secrets", "namespaces"]
verbs: ["get", "list"]
- apiGroups: [""]
resources: ["nodes", "pods"]
verbs: ["get", "list", "watch", "update"]
- apiGroups: ["storage.k8s.io"]
resources: ["volumeattachments", "volumeattachments"]
verbs: ["get", "list", "watch", "update", "patch"]
- apiGroups: ["storage.k8s.io"]
resources: ["storageclasses"]
verbs: ["get", "list", "watch"]
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
name: cfsturbo-csi-node-rolebinding
subjects:
- kind: ServiceAccount
name: cfsturbo-csi-node-sa
namespace: kube-system
roleRef:
kind: ClusterRole
name: cfsturbo-csi-node-role
apiGroup: rbac.authorization.k8s.io
```

Deploying a Node Plugin

1. Run the `kubectl apply -f csidriver.yaml` command. The following csidriver.yaml code is for your reference:

©2013-2022 Tencent Cloud. All rights reserved.

```
apiVersion: storage.k8s.io/v1beta1
kind: CSIDriver
metadata:
name: com.tencent.cloud.csi.cfsturbo
spec:
attachRequired: false
podInfoOnMount: false
```

2. Run the `kubectl apply -f csi-node.yaml` commad. The following csi-node.yaml code is for your reference:

```
# This YAML file contains driver-registrar & csi driver nodeplugin API objects
# that are necessary to run CSI nodeplugin for cfsturbo
kind: DaemonSet
apiVersion: apps/v1
metadata:
name: cfsturbo-csi-node
namespace: kube-system
spec:
selector:
matchLabels:
app: cfsturbo-csi-node
template:
metadata:
labels:
app: cfsturbo-csi-node
spec:
serviceAccount: cfsturbo-csi-node-sa
hostNetwork: true
containers:
- name: driver-registrar
image: ccr.ccs.tencentyun.com/tkeimages/csi-node-driver-registrar:v1.2.0
lifecycle:
preStop:
exec:
command: ["/bin/sh", "-c", "rm -rf /registration/com.tencent.cloud.csi.cfsturbo
/registration/com.tencent.cloud.csi.cfsturbo-reg.sock"]
args:
- "--v=5"
- "--csi-address=/plugin/csi.sock"
- "--kubelet-registration-path=/var/lib/kubelet/plugins/com.tencent.cloud.csi.c
fsturbo/csi.sock"
env:
- name: KUBE_NODE_NAME
valueFrom:
```

©2013-2022 Tencent Cloud. All rights reserved.

```
fieldRef:
fieldPath: spec.nodeName
volumeMounts:
- name: plugin-dir
mountPath: /plugin
- name: registration-dir
mountPath: /registration
- name: cfsturbo
securityContext:
privileged: true
capabilities:
add: ["SYS_ADMIN"]
allowPrivilegeEscalation: true
image: ccr.ccs.tencentyun.com/tkeimages/csi-tencentcloud-cfsturbo:v1.2.2
args :
- "--nodeID=$(NODE_ID)"
- "--endpoint=$(CSI_ENDPOINT)"
env:
- name: NODE_ID
valueFrom:
fieldRef:
fieldPath: spec.nodeName
- name: CSI_ENDPOINT
value: unix://plugin/csi.sock
imagePullPolicy: "IfNotPresent"
volumeMounts:
- name: plugin-dir
mountPath: /plugin
- name: pods-mount-dir
mountPath: /var/lib/kubelet/pods
mountPropagation: "Bidirectional"
- name: global-mount-dir
mountPath: /etc/cfsturbo/global
mountPropagation: "Bidirectional"
volumes:
- name: plugin-dir
hostPath:
path: /var/lib/kubelet/plugins/com.tencent.cloud.csi.cfsturbo
type: DirectoryOrCreate
- name: pods-mount-dir
hostPath:
path: /var/lib/kubelet/pods
type: Directory
- name: registration-dir
hostPath:
path: /var/lib/kubelet/plugins_registry
type: Directory
```

©2013-2022 Tencent Cloud. All rights reserved.

```
- name: global-mount-dir
hostPath:
path: /etc/cfsturbo/global
type: DirectoryOrCreate
```

Using a CFS Turbo volume

1. Create a CFS Turbo file system. For more information, see Creating a File System.
2. Use the following template to create a PV of CFS Turbo type.

```
apiVersion: v1
kind: PersistentVolume
metadata:
name: pv-cfsturbo
spec:
accessModes:
- ReadWriteMany
capacity:
storage: 10Gi
csi:
driver: com.tencent.cloud.csi.cfsturbo
# volumeHandle in PV must be unique, use pv name is better
volumeHandle: pv-cfsturbo
volumeAttributes:
# cfs turbo server ip
host: 10.0.0.116
# cfs turbo fsid (not cfs id)
fsid: xxxxxxxx
# cfs turbo rootdir
rootdir: /cfs
# cfs turbo subPath
path: /
proto: lustre
storageClassName: ""
```

Parameter description:

- **metadata.name**: The name of the created PV.
- **spec.csi.volumeHandle**: It must be consistent with the PV name.
- **spec.csi.volumeAttributes.host**: The IP address of the file system. You can check it in the information about file system mount target.
- **spec.csi.volumeAttributes.fsid**: The fsid of the file system (not the file system ID). You can check it in the file system mount target information. It is the string after "tcp0:/" and before "/cfs" in the mounting command, as shown in the following figure.

©2013-2022 Tencent Cloud. All rights reserved.

- **spec.csi.volumeAttributes.rootdir**: The root directory of the file system. “/cfs” is entered if it is left empty (the general mounting performance is enhanced if mounting to “/cfs”). If you want to specify a root directory for mounting, you must ensure that the root directory exists in the file system.
- **spec.csi.volumeAttributes.path**: The subdirectory of the file system. “/” is entered if it is left empty. If you want to specify a subdirectory for mounting, you must ensure that the subdirectory exists in rootdir of the file system. The directory accessed by the container is the rootdir+path directory of the file system (defaults to “/cfs/” directory).
- **spec.csi.volumeAttributes.proto**: The default protocol for mounting the file system.

> Note：
> You need to install a Client in the cluster node according to the version of operating system kernel before using `lustre` protocol to mount a CFS Turbo volume. For details, see Using CFS Turbo on Linux Clients.

3. Use the following template to create a PVC that binds a PV.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
name: pvc-cfsturbo
spec:
storageClassName: ""
volumeName: pv-cfsturbo
accessModes:
- ReadWriteMany
resources:
requests:
storage: 10Gi
```

Parameter description:

- **metadata.name**: The name of the created PVC.
- **spec.volumeName**: This need to be consistent with the name of PV created in the previous step.

©2013-2022 Tencent Cloud. All rights reserved.

4. Use the following template to create a Pod that mounts a PVC.

```
apiVersion: v1
kind: Pod
metadata:
name: nginx
spec:
containers:
- image: ccr.ccs.tencentyun.com/qcloud/nginx:1.9
imagePullPolicy: Always
name: nginx
ports:
- containerPort: 80
protocol: TCP
volumeMounts:
- mountPath: /var/www
name: data
volumes:
- name: data
persistentVolumeClaim:
claimName: pvc-cfsturbo
```

©2013-2022 Tencent Cloud. All rights reserved.

Static Mounting of CFS-Turbo for TKE Serverless Clusters

Last updated：2023-03-14 18:26:30

Use Cases

You can mount a CFS Turbo storage for a TKE Serverless. The add-on is used to mount the Tencent Cloud CFS Turbo file system to a workload based on a private protocol. Currently, only static configuration is supported. For more information about CFS storage types, see Storage Types and Performance.

Prerequisites

A TKE Serverless of v1.14 or later has been created.

Directions

Creating a file system

Create a CFS Turbo file system. For details, see Creating a File System and Mount Point.

Note

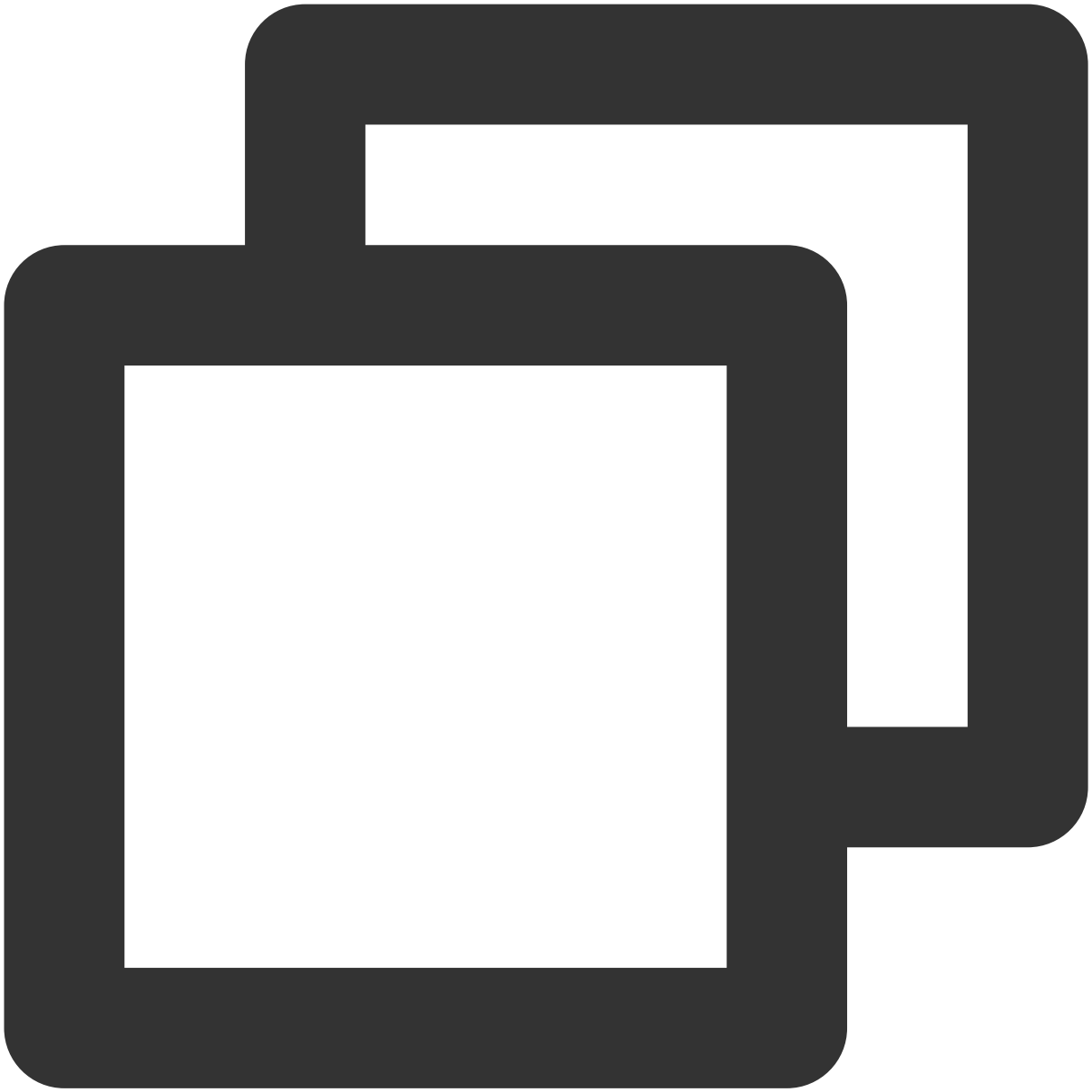
After the file system is created, you need to associate the cluster network (vpc-xx) with the CCN instance of the file system. You can check it in the information about file system mount target.

Deploying a Node Plugin

Step 1. Create a csidriver.yaml file

Here is an example of a csidriver.yaml file:

©2013-2022 Tencent Cloud. All rights reserved.

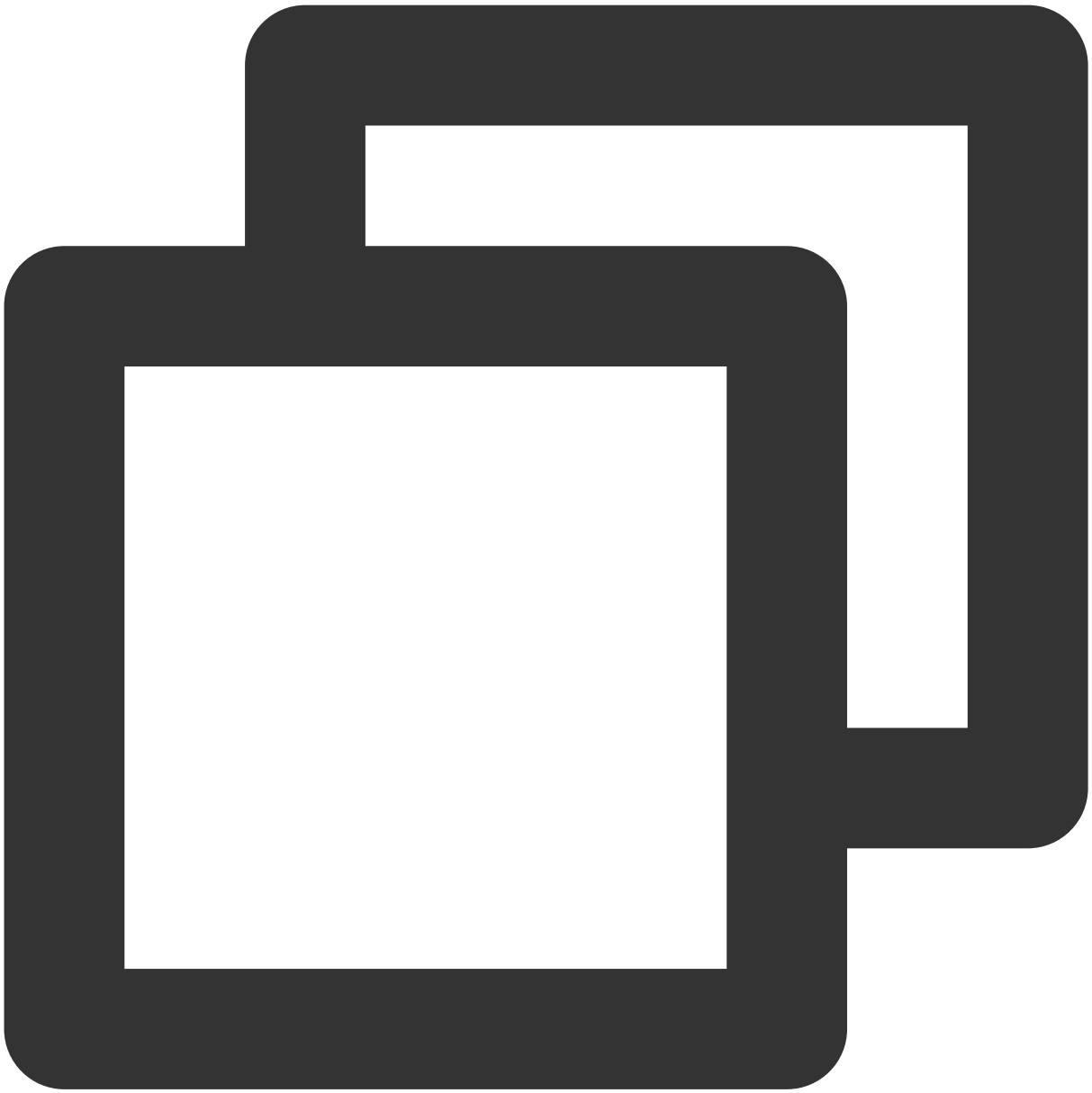


```
apiVersion: storage.k8s.io/v1
kind: CSIDriver
metadata:
  name: com.tencent.cloud.csi.cfsturbo
spec:
  attachRequired: false
  podInfoOnMount: false
```

Step 2. Create a CSI driver

©2013-2022 Tencent Cloud. All rights reserved.

Run the following command to create a CSI driver:

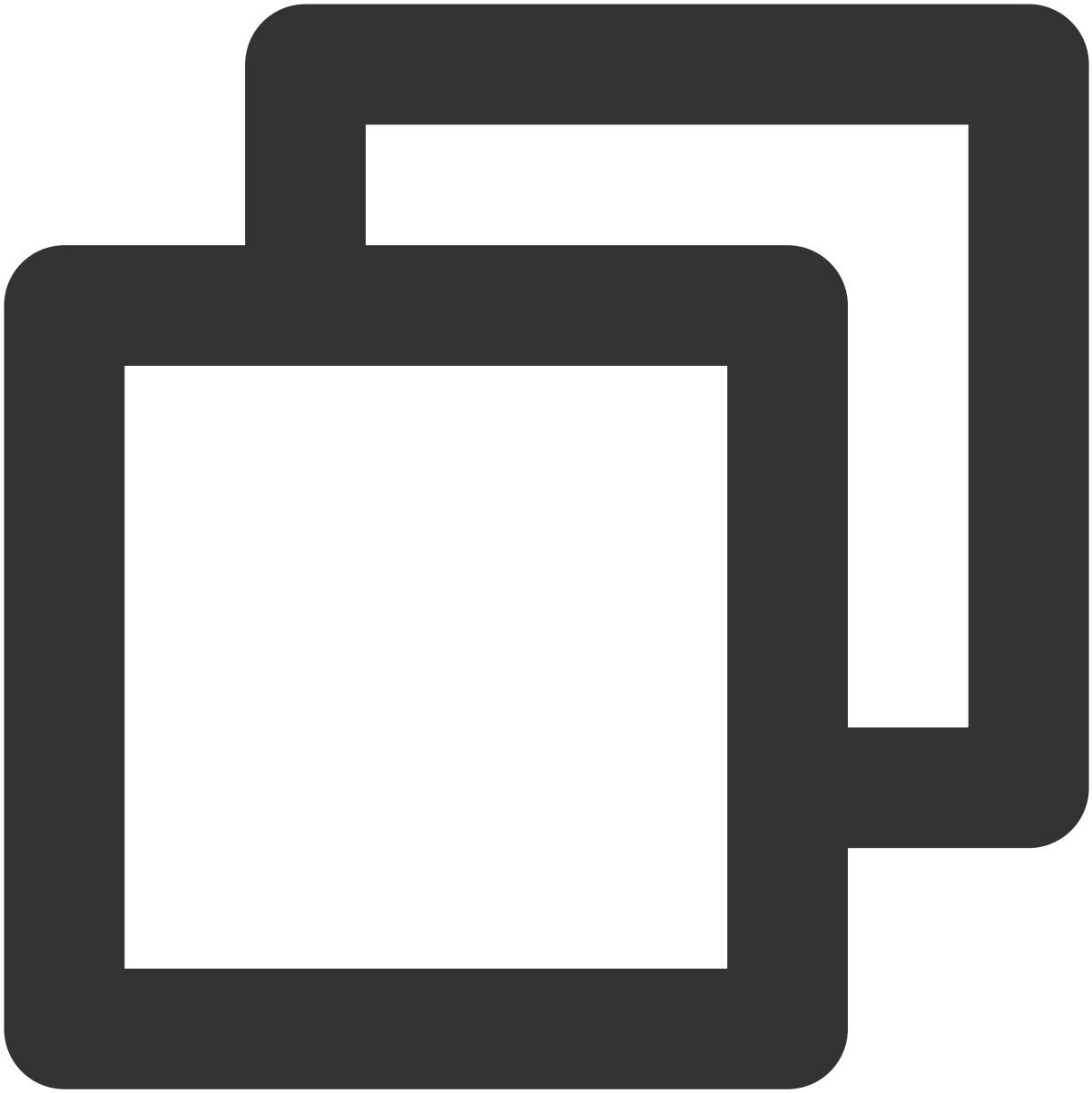


```
kubectl apply -f csidriver.yaml
```

Creating a CFS Turbo volume

Step 1. Create a CFS Turbo type PV based on the following template

©2013-2022 Tencent Cloud. All rights reserved.



```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-cfsturbo
spec:
  accessModes:
  - ReadWriteMany
  capacity:
    storage: 10Gi
  csi:
    driver: com.tencent.cloud.csi.cfsturbo
```

©2013-2022 Tencent Cloud. All rights reserved.

```
    volumeHandle: pv-cfsturbo
    volumeAttributes:
      host: *.*.*.*
      fsid: ********
      # cfs turbo subPath
      path: /
  storageClassName: ""
```

Parameters:

metadata.name: The name of the created PV.

spec.csi.volumeHandle: It must be consistent with the PV name.

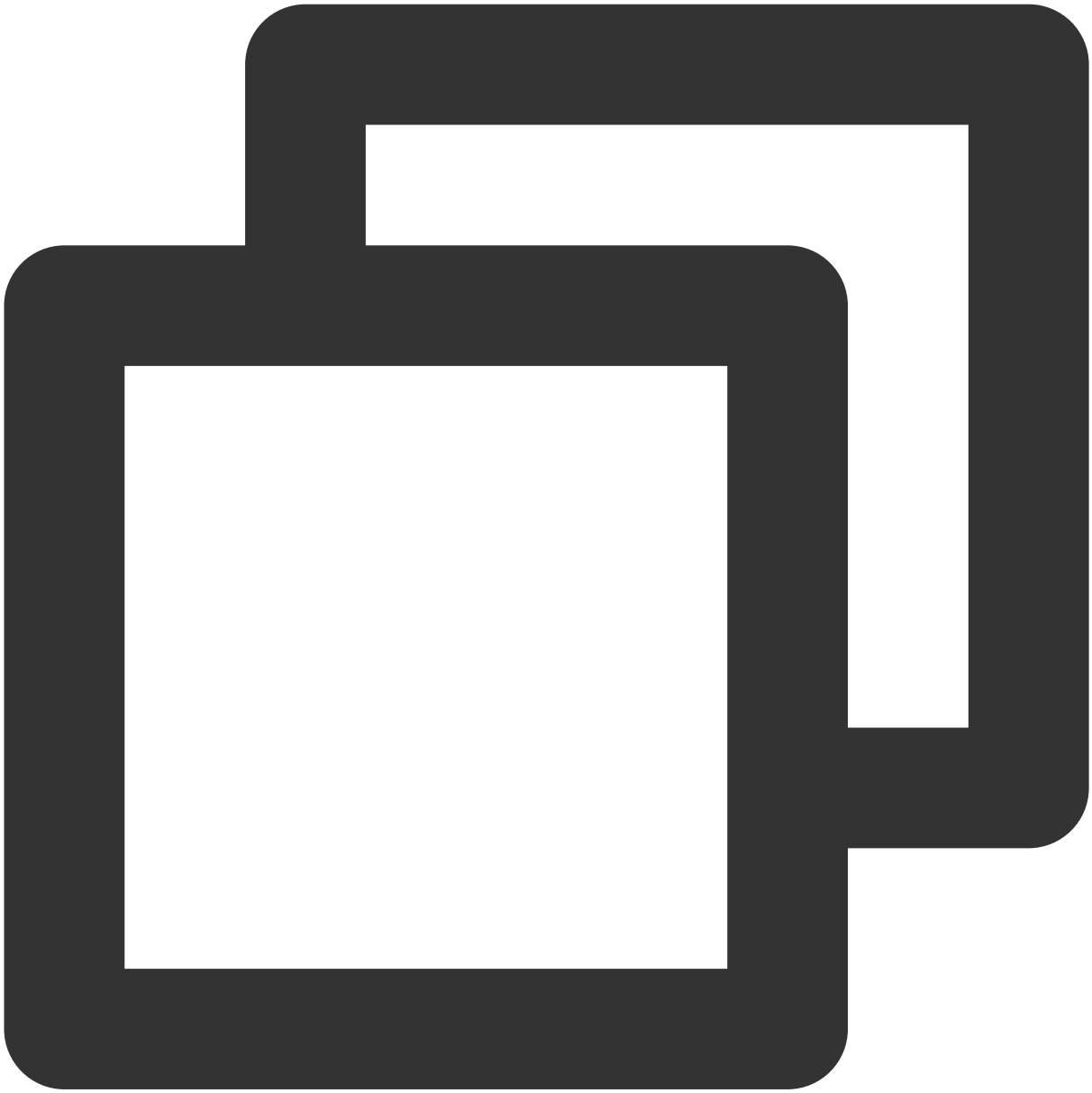
spec.csi.volumeAttributes.host: The IP address of the file system. You can check it in the file system mount target information.

spec.csi.volumeAttributes.fsid: The fsid of the file system (not the file system ID). You can check it in the file system mount target information. It is the string between `tcp0:/` and `/cfs` in the mount command, as shown in the following figure.

spec.csi.volumeAttributes.path: The subdirectory of the file system. `/` is entered if it is left empty (to improve the mounting performance, `/` is located under the `/cfs` directory). If you want to specify a subdirectory for mounting, you must ensure that the subdirectory exists in `/cfs` in the file system. The workload cannot access the parent directory of the subdirectory after mounting. For example, for `path: /test` , you must ensure that the `/cfs/test` exists in the file system.

Step 2. Create a PVC that is bound to the PV based on the following template

©2013-2022 Tencent Cloud. All rights reserved.



```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-cfsturbo
spec:
  storageClassName: ""
  volumeName: pv-cfsturbo
  accessModes:
  - ReadWriteMany
  resources:
    requests:
```

©2013-2022 Tencent Cloud. All rights reserved.

```
      storage: 10Gi
```

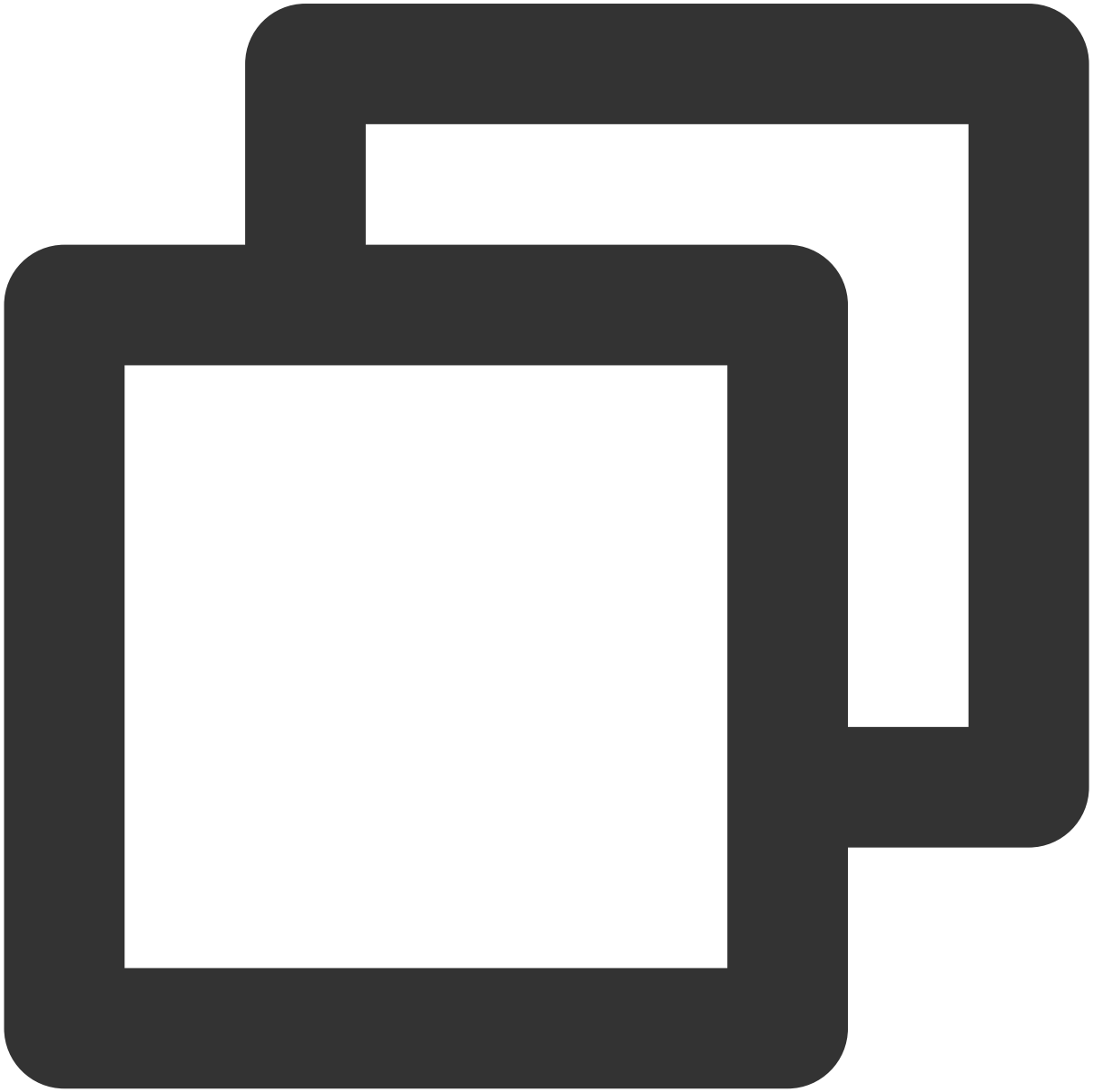
Parameters:

metadata.name: The name of the created PVC.

spec.volumeName: It must be consistent with the name of the PV created in Step 1.

Using a CFS Turbo volume

Create a Pod that mounts the PVC based on the following template.



```
apiVersion: v1
```

©2013-2022 Tencent Cloud. All rights reserved.

```
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: ccr.ccs.tencentyun.com/qcloud/nginx:1.9
    imagePullPolicy: Always
    name: nginx
    ports:
    - containerPort: 80
      protocol: TCP
    volumeMounts:
      - mountPath: /var/www
        name: data
  volumes:
  - name: data
    persistentVolumeClaim:
      claimName: pvc-cfsturbo
```

©2013-2022 Tencent Cloud. All rights reserved.

Containerization
Accelerated Pull of Images Outside the Chinese Mainland

Last updated：2022-04-20 19:01:30

Operation Scenario

Currently, the container images of most open-source apps (such as Kubernetes and TensorFlow) are hosted on image hosting platforms outside of the Chinese mainland (such as DockerHub and `quay.io`). As a result, pulling images in the Chinese mainland may be slow or even fail due to network issues. A common solution is to manually pull images to local storage and then push them to a self-built image repository for manual synchronization. This process is very complicated and does not cover all repositories or the latest image versions.
Tencent Container Registry (TCR) Enterprise Edition provides an acceleration service for mainstream image hosting platforms outside of the Chinese mainland to effectively resolve difficulties in image pulling, thereby facilitating the deployment of open-source apps. This document introduces how TKE clusters use the TCR acceleration service to accelerate image pulling outside of the Chinese mainland.

Limits

- Currently, the acceleration service is available only to TKE and TCR users.
- The acceleration service currently can be accessed only from Tencent Cloud VPCs. Access from the Internet is not yet allowed. The relevant domain name can be accessed but cannot actually provide the acceleration feature.

Directions

For TKE clusters, acceleration has been configured for the public images of the DockerHub platform by default. If you need acceleration for image repositories on other platforms, such as `quay.io`, you need to modify the configuration. The configuration method for clusters with a Docker runtime environment is different from that for clusters with a Containerd runtime environment.

- Configuration for Docker clusters
- Configuration for Containerd clusters

©2013-2022 Tencent Cloud. All rights reserved.

For nodes with a Docker runtime environment, because Docker itself does not support acceleration configuration except for `docker.io`, when you use container images other than `docker.io` from outside the Chinese mainland, you need to run the following command to change the image address domain name from `quay.io` to `quay.tencentcloudcr.com`. See the example below:

```
docker pull quay.tencentcloudcr.com/k8scsi/csi-resizer:v0.5.0
```

©2013-2022 Tencent Cloud. All rights reserved.

Image Layering Best Practices

Last updated：2023-05-19 16:05:27

Overview

This document describes how to build and manage business images in layers and provides best practices for managing container images of all types using TCR.

Advantages of container image layering

- Resources are shared to improve the utilization.
- Image management is standardized to facilitate DevOps implementation.
- TCR's Ops-free image acceleration easily makes large-scale image distribution faster by 5-10 times.
- TCR Enterprise has been accessed to Tencent CloudAudit. You can check the logs of read and write operations of instances, namespaces, and image repositories in "Event History".

Prerequisites

Before using a private image managed in TCR for application deployment, you need to complete the following preparations:

- You have created a TCR Enterprise instance in the TCR console. If you haven't done so, create one first. For more information, see Creating an Enterprise Edition Instance.
- If you are using a sub-account, you must have granted the sub-account operation permissions for the corresponding instance. For more information, see Example of Authorization Solution of TCR Enterprise.

```
Note: This also applies to the existing TCR instances. You only need to modify th
e image repository address.
```

1. F3S Docker Files Overview

The project consists of the following parts:

```
$ tree -L 3 ./f3s-docker-files
./f3s-docker-files
├── README.md ------ README file
```

©2013-2022 Tencent Cloud. All rights reserved.

```
├── DockerBuildImages.sh ------ Image build script
├── 0.base ------ 0. Build various types of system images at the base layer
│ ├── alpine ------ Build the alpine system image at the base layer
│ │ └── Dockerfile
│ └── centos-7.8 ------ Build the CentOS 7.8 system image at the base layer
│ ├── Dockerfile
│ └── centos-7.8.2003-x86_64-docker.tar.xz
├─ 1.ops ------ 1. Build various types of images at the Ops layer
│ └── Dockerfile-alpine ------ Build the alpine image at the Ops layer
├── 2.lang ------ 2. Build various types of images at the language layer
│ └── Dockerfile-alpine-kona ------ alpine-kona image at the language layer
└── 3.app ------ 3. Build various types of images at the application layer
├── jmeter
│ ├── Dockerfile-jmeter-base ------ Build the jmeter-base image
│ ├── Dockerfile-jmeter-grafana-reporter ------ Build the jmeter-grafana-reporter
image
│ ├── Dockerfile-jmeter-master ------ Build the jmeter-master image
│ └── Dockerfile-jmeter-slave ------ Build the jmeter-slave image
├── nginx
│ ├── Dockerfile-alpine-nginx ------ Build the alpine-nginx image
│ ├── default.conf
│ └── nginx.conf
└── skywalking
└── Dockerfile-alpine-kona-skywalking ------ Build the alpine-kona-skywalking ima
ge
```

- **alpine/Dockerfile**: Build with the official Alpine 3.13 Docker image to support common Ops tools.
- **centos-7.8/Dockerfile**: Build with the official CentOS 7.8 Docker image to support common Ops tools.
- **Dockerfile-alpine-kona**: Build with the Dockerfile-alpine and TencentKona 8.0.5 binary package. The Kona is partially trimmed to control the image size.
- **Dockerfile-jmeter-base**: Build based on the official JMeter 5.4.1 binary package.
- **Dockerfile-jmeter-grafana-reporter**: Build based on Grafana-Reporter to generate JMeter PDF reports from Grafana dashboards.
- **Dockerfile-jmeter-master**: Build based on Jmeter-base to implement distributed master stress test with JMeter.
- **Dockerfile-jmeter-slave**: Build based on Jmeter-base to implement distributed slave stress test with JMeter.
- **Dockerfile-alpine-nginx**: Build with Dockerfile-alpine to add NGINX configuration initialization and logging specifications.
- **Dockerfile-alpine-kona-skywalking**: Build with the official Dockerfile-alpine-kona and SkyWalking 8.5 binary package.

2. Project Resource Description

©2013-2022 Tencent Cloud. All rights reserved.

2.0 Dockerfile-alpine

==========ALPINE DOCKER FILE==========

```
# build
FROM alpine:3.13

ENV FROM alpine:3.13

# The Alpine image does not contain `tzdata`, so you cannot set the time zone dir
ectly via the environment variable `TZ`. To this end, you need to install `tzdata
`:
ENV TZ=Asia/Shanghai

RUN echo 'http://mirrors.tencent.com/alpine/v3.13/main/' > /etc/apk/repositories
\
&& echo 'http://mirrors.tencent.com/alpine/v3.13/community/' >> /etc/apk/reposito
ries \
&& apk --no-cache add apache2-utils \
bind-tools \
bridge-utils \
busybox-extras \
curl \
ebtables \
ethtool \
fio \
fping \
iperf3 \
iproute2 \
iptables \
iputils \
ipvsadm \
jq \
lftp \
lsof \
mtr \
netcat-openbsd \
net-tools \
nmap \
procps \
psmisc \
rsync \
smartmontools \
strace \
sysstat \
tcpdump \
tree \
```

©2013-2022 Tencent Cloud. All rights reserved.

```
tzdata \
unzip \
util-linux \
wget \
zip \
&& echo "${TZ}" > /etc/timezone \
&& ln -sf /usr/share/zoneinfo/${TZ} /etc/localtime \
&& rm -rf /var/cache/apk/*

ENV BUILD f3s-docker-file.tencentcloudcr.com/f3s-tcr/alpine:v3.13


# docker build -t f3s-docker-file.tencentcloudcr.com/f3s-tcr/alpine:v3.13 .
```

==========Build, tag and push the base image==========

```
cd $pwd/0.base/alpine
docker build -t f3s-docker-file.tencentcloudcr.com/f3s-tcr/alpine:v3.13 -f Docker
file .
docker push f3s-docker-file.tencentcloudcr.com/f3s-tcr/alpine:v3.13
```

2.1 Dcokerfile-CentOS-7.8

==========Centos-7.8 DOCKER FILE==========

```
# build
# CentOS 7.8 official Dockerfile: https://github.com/CentOS/sig-cloud-instance-im
ages/blob/CentOS-7.8.2003-x86_64/docker/Dockerfile
# CentOS 7.8 official package: wget https://raw.githubusercontent.com/CentOS/sig-
cloud-instance-images/CentOS-7.8.2003-x86_64/docker/centos-7.8.2003-x86_64-docke
r.tar.xz

FROM scratch

ADD centos-7.8.2003-x86_64-docker.tar.xz /

LABEL name="CentOS Base Image" \
vendor="CentOS" \
license="GPLv2" \
build-date="20200504"


# Add some widgets and change the time zone
RUN set -ex \
&& yum install -y wget \
&& rm -rf /etc/yum.repos.d/CentOS-* \
```

©2013-2022 Tencent Cloud. All rights reserved.

```
# Add the `Tencent yum` source
&& wget -O /etc/yum.repos.d/CentOS-Base.repo http://mirrors.cloud.tencent.com/rep
o/centos7_base.repo \
&& yum fs filter documentation \
&& yum install -y atop \
bind-utils \
curl \
dstat \
ebtables \
ethtool \
fping \
htop \
iftop \
iproute \
jq \
less \
lsof \
mtr \
nc \
net-tools \
nmap-ncat \
perf \
psmisc \
strace \
sysstat \
tcpdump \
telnet \
tree \
unzip \
wget \
which \
zip \
ca-certificates \
&& rm -rf /etc/localtime \
&& ln -s /usr/share/zoneinfo/Asia/Shanghai /etc/localtime \
# Install dumb-init
&& wget -O /usr/local/bin/dumb-init https://github.com/Yelp/dumb-init/releases/do
wnload/v1.2.5/dumb-init_1.2.5_x86_64 \
&& chmod +x /usr/local/bin/dumb-init \
# Install gosu grab gosu for easy step-down from root
# https://github.com/tianon/gosu/releases
&& wget -O /usr/local/bin/gosu "https://github.com/tianon/gosu/releases/download/
1.13/gosu-amd64" \
&& chmod +x /usr/local/bin/gosu \
&& gosu nobody true \
# Install the Chinese language package to solve the VI garbled text issue
&& yum -y install kde-l10n-Chinese glibc-common \
```

©2013-2022 Tencent Cloud. All rights reserved.

```
&& localedef -c -f UTF-8 -i zh_CN zh_CN.utf8 \
&& export LC_ALL=zh_CN.utf8 \
&& yum clean all \
&& rm -rf /tmp/* \
&& rm -rf /var/lib/yum/* \
&& rm -rf /var/cache/yum

# Solve the LESS garbled text issue
ENV LESSCHARSET utf-8

# Set language environment variables
ENV LANG=en_US.UTF-8

# If this line is not added, `stdin: true` and `tty: true` in Kubernetes will not
take effect
CMD ["/bin/bash"]


ENV BUILD f3s-docker-file.tencentcloudcr.com/f3s-tcr/centos:v7.8

# docker build -t f3s-docker-file.tencentcloudcr.com/f3s-tcr/centos:v7.8 .
```

==========Build, tag and push the base image==========

```
cd $pwd/0.base/centos-7.8
docker build --no-cache -t f3s-docker-file.tencentcloudcr.com/f3s-tcr/centos:v7.8
-f Dockerfile .
docker push f3s-docker-file.tencentcloudcr.com/f3s-tcr/centos:v7.8
# To test run: docker run --name test -it --rm f3s-docker-file.tencentcloudcr.co
m/f3s-tcr/centos:v7.8 uname -a
# docker export <container-id> | docker import f3s-docker-file.tencentcloudcr.co
m/f3s-tcr/centos:v7.8
# quick interative termnal: docker run -it --entrypoint=sh f3s-docker-file.tencen
tcloudcr.com/f3s-tcr/centos:v7.8 sh
```

2.2 Dcokerfile-Ops

==========Ops DOCKER FILE==========

```
# build
FROM f3s-docker-file.tencentcloudcr.com/f3s-tcr/alpine:v3.13
MAINTAINER westzhao

ENV LANG=C.UTF-8

# Download the Ops tool
```

©2013-2022 Tencent Cloud. All rights reserved.

```
RUN apk --no-progress --purge --no-cache add --upgrade wget \
curl \
mysql-client \
busybox \
busybox-extras \
bash \
bash-doc \
bash-completion \
tzdata \
vim \
unzip && \
# Download the glibc to support JDK and solve the Chinese character issue && \
wget -q -O /etc/apk/keys/sgerrand.rsa.pub https://alpine-pkgs.sgerrand.com/sgerra
nd.rsa.pub && \
wget https://github.com/sgerrand/alpine-pkg-glibc/releases/download/2.33-r0/glibc
-2.33-r0.apk && \
wget https://github.com/sgerrand/alpine-pkg-glibc/releases/download/2.33-r0/glibc
-bin-2.33-r0.apk && \
wget https://github.com/sgerrand/alpine-pkg-glibc/releases/download/2.33-r0/glibc
-i18n-2.33-r0.apk && \
apk add glibc-2.33-r0.apk glibc-bin-2.33-r0.apk glibc-i18n-2.33-r0.apk && \
rm glibc-2.33-r0.apk glibc-bin-2.33-r0.apk glibc-i18n-2.33-r0.apk && \
/usr/glibc-compat/bin/localedef -i en_US -f UTF-8 C.UTF-8 && \
echo "export LANG=$LANG" > /etc/profile.d/locale.sh && \
# Change the time zone
mkdir -p /share/zoneinfo/Asia/ && \
mkdir -p /etc/zoneinfo/Asia/ && \
cp /usr/share/zoneinfo/Asia/Shanghai /etc/localtime && \
cp /usr/share/zoneinfo/Asia/Shanghai /share/zoneinfo/Asia/Shanghai && \
cp /usr/share/zoneinfo/Asia/Shanghai /etc/zoneinfo/Asia/Shanghai && \
echo "Asia/Shanghai" > /etc/timezone && \
apk del tzdata && \
# Delete the APK cache && \
rm -rf /var/cache/apk/*
```

==========Build, tag and push the base image==========

```
docker build --no-cache -t f3s-docker-file.tencentcloudcr.com/f3s-tcr/alpine:late
st -f ./1.ops/Dockerfile-alpine .
docker push f3s-docker-file.tencentcloudcr.com/f3s-tcr/alpine:latest
# To test run: docker run --name test -it --rm f3s-docker-file.tencentcloudcr.co
m/f3s-tcr/alpine:latest sh $(java -version)
# docker export <container-id> | docker import f3s-docker-file.tencentcloudcr.co
m/f3s-tcr/alpine:latest
# quick interative termnal: docker run -it --entrypoint=sh f3s-docker-file.tencen
tcloudcr.com/f3s-tcr/alpine:latest sh
```

©2013-2022 Tencent Cloud. All rights reserved.

2.3 Dockerfile-alpine-kona

==========Alpine Kona DOCKER FILE==========

```
# build
FROM f3s-docker-file.tencentcloudcr.com/f3s-tcr/alpine:latest
MAINTAINER westzhao

ENV LANG=C.UTF-8


# Download the Kona installation package via wget && \
RUN cd /opt && \
wget https://github.com/Tencent/TencentKona-8/releases/download/8.0.5-GA/TencentK
ona8.0.5.b12_jdk_linux-x86_64_8u282.tar.gz && \
tar -xvf TencentKona8.0.5.b12_jdk_linux-x86_64_8u282.tar.gz && \
rm TencentKona8.0.5.b12_jdk_linux-x86_64_8u282.tar.gz && \
ln -nfs /opt/TencentKona-8.0.5-282 /opt/jdk && \
# Trim the unused resources of the JDK && \
rm /opt/jdk/release && \
rm /opt/jdk/THIRD_PARTY_README && \
rm /opt/jdk/LICENSE && \
rm /opt/jdk/ASSEMBLY_EXCEPTION && \
rm -rf /opt/jdk/sample/ && \
rm -rf /opt/jdk/demo/ && \
rm -rf /opt/jdk/src.zip && \
rm -rf /opt/jdk/man/ && \
rm -rf /opt/jdk/lib/missioncontrol && \
rm -rf /opt/jdk/lib/visualvm && \
rm -rf /opt/jdk/lib/ant-javafx.jar && \
rm -rf /opt/jdk/lib/javafx-mx.jar && \
rm -rf /opt/jdk/lib/jconsole.jar && \
rm -rf /opt/jdk/jre/lib/amd64/libawt_xawt.so && \
rm -rf /opt/jdk/jre/lib/amd64/libjavafx_font_freetype.so && \
rm -rf /opt/jdk/jre/lib/amd64/libjavafx_font_pango.so && \
rm -rf /opt/jdk/jre/lib/amd64/libjavafx_font.so && \
rm -rf /opt/jdk/jre/lib/amd64/libjavafx_font_t2k.so && \
rm -rf /opt/jdk/jre/lib/amd64/libjavafx_iio.so && \
rm -rf /opt/jdk/jre/lib/amd64/libjfxwebkit.so && \
rm -rf /opt/jdk/jre/lib/desktop && \
rm -rf /opt/jdk/jre/lib/ext/jfxrt.jar && \
rm -rf /opt/jdk/jre/lib/fonts && \
rm -rf /opt/jdk/jre/lib/locale/de && \
rm -rf /opt/jdk/jre/lib/locale/fr && \
rm -rf /opt/jdk/jre/lib/locale/it && \
rm -rf /opt/jdk/jre/lib/locale/ja && \
rm -rf /opt/jdk/jre/lib/locale/ko && \
```

©2013-2022 Tencent Cloud. All rights reserved.

```
rm -rf /opt/jdk/jre/lib/locale/ko.UTF-8 && \
rm -rf /opt/jdk/jre/lib/locale/pt_BR && \
rm -rf /opt/jdk/jre/lib/locale/sv && \
rm -rf /opt/jdk/jre/lib/locale/zh_HK.BIG5HK && \
rm -rf /opt/jdk/jre/lib/locale/zh_TW && \
rm -rf /opt/jdk/jre/lib/locale/zh_TW.BIG5 && \
rm -rf /opt/jdk/jre/lib/oblique-fonts && \
rm -rf /opt/jdk/jre/lib/deploy.jar && \
rm -rf /opt/jdk/jre/lib/locale/

# JAVA_HOME
ENV JAVA_HOME=/opt/jdk
ENV CLASSPATH=.:$JAVA_HOME/lib/
ENV PATH=$JAVA_HOME/bin:$PATH
```

==========Build, tag and push the base image==========

```
docker build --no-cache -t f3s-docker-file.tencentcloudcr.com/f3s-tcr/alpine-kon
a:latest -f ./2.lang/Dockerfile-alpine-kona .
docker push f3s-docker-file.tencentcloudcr.com/f3s-tcr/alpine-kona:latest
# To test run: docker run --name test -it --rm f3s-docker-file.tencentcloudcr.co
m/f3s-tcr/alpine-kona:latest sh $(java -version)
# docker export <container-id> | docker import f3s-docker-file.tencentcloudcr.co
m/f3s-tcr/alpine-kona:latest
# quick interative termnal: docker run -it --entrypoint=sh f3s-docker-file.tencen
tcloudcr.com/f3s-tcr/alpine-kona:latest sh
```

2.4 Dockerfile-alpine-kona-skywalking

==========Alpine Kona SkyWalking DOCKER FILE==========

```
# build
FROM f3s-docker-file.tencentcloudcr.com/f3s-tcr/alpine-kona:latest
MAINTAINER westzhao

ENV LANG=C.UTF-8

# Download the Ops tool
RUN mkdir /3.app && \
wget -q -O /3.app/apache-skywalking-apm-8.5.0.tar.gz https://archive.apache.org/d
ist/skywalking/8.5.0/apache-skywalking-apm-8.5.0.tar.gz && \
tar zxf /3.app/apache-skywalking-apm-8.5.0.tar.gz -C /3.app && \
mv /3.app/apache-skywalking-apm-bin/agent /3.app/skywalking && \
rm -rf /3.app/apache-skywalking-apm-8.5.0.tar.gz && \
rm -rf /3.app/apache-skywalking-apm-bin/
```

©2013-2022 Tencent Cloud. All rights reserved.

```
# JAVA_HOME
ENV JAVA_HOME=/opt/jdk
ENV CLASSPATH=.:$JAVA_HOME/lib/
ENV PATH=$JAVA_HOME/bin:$PATH
```

==========Build, tag and push the base image==========

```
docker build --no-cache -t f3s-docker-file.tencentcloudcr.com/f3s-tcr/alpine-kona
-skywalking:latest -f ./3.app/skywalking/Dockerfile-alpine-kona-skywalking .
docker push f3s-docker-file.tencentcloudcr.com/f3s-tcr/alpine-kona-skywalking:lat
est
# To test run: docker run --name test -it --rm f3s-docker-file.tencentcloudcr.co
m/f3s-tcr/alpine-kona-skywalking:latest sh $(java -version)
# docker export <container-id> | docker import f3s-docker-file.tencentcloudcr.co
m/f3s-tcr/alpine-kona-skywalking:latest
# quick interative termnal: docker run -it --entrypoint=sh f3s-docker-file.tencen
tcloudcr.com/f3s-tcr/alpine-kona-skywalking:latest sh
```

2.5 Dockerfile-jmeter-base

==========JMETER BASE DOCKER FILE==========

```
# build
FROM f3s-docker-file.tencentcloudcr.com/f3s-tcr/alpine-kona:latest
MAINTAINER westzhao

ARG JMETER_VERSION=5.4.1

# Download JMeter
RUN mkdir /jmeter && \
cd /jmeter && \
wget https://archive.apache.org/dist/jmeter/binaries/apache-jmeter-$JMETER_VERSIO
N.tgz && \
tar -xzf apache-jmeter-$JMETER_VERSION.tgz && \
rm apache-jmeter-$JMETER_VERSION.tgz && \
# Download JMeterPlugins-Standard && \
cd /jmeter/apache-jmeter-$JMETER_VERSION/ && \
wget -q -O /tmp/JMeterPlugins-Standard-1.4.0.zip https://jmeter-plugins.org/downl
oads/file/JMeterPlugins-Standard-1.4.0.zip && \
unzip -n /tmp/JMeterPlugins-Standard-1.4.0.zip && \
rm /tmp/JMeterPlugins-Standard-1.4.0.zip && \
# Download pepper-box && \
wget -q -O /jmeter/apache-jmeter-$JMETER_VERSION/lib/ext/pepper-box-1.0.jar http
s://github.com/raladev/load/blob/master/JARs/pepper-box-1.0.jar?raw=true && \
# Download bzm-parallel && \
cd /jmeter/apache-jmeter-$JMETER_VERSION/ && \
```

©2013-2022 Tencent Cloud. All rights reserved.

```
wget -q -O /tmp/bzm-parallel-0.7.zip https://jmeter-plugins.org/files/packages/bz
m-parallel-0.7.zip && \
unzip -n /tmp/bzm-parallel-0.7.zip && \
rm /tmp/bzm-parallel-0.7.zip

ENV JMETER_HOME /jmeter/apache-jmeter-$JMETER_VERSION/

ENV PATH $JMETER_HOME/bin:$PATH
```

==========Build, tag and push the base image==========

```
docker build --no-cache -t f3s-docker-file.tencentcloudcr.com/f3s-tcr/jmeter-bas
e:latest -f ./3.app/jmeter/Dockerfile-jmeter-base .
docker push f3s-docker-file.tencentcloudcr.com/f3s-tcr/jmeter-base:latest
```

2.6 Dockerfile-jmeter-master

==========JMETER-MASTER DOCKER FILE==========

```
# build
FROM f3s-docker-file.tencentcloudcr.com/f3s-tcr/jmeter-base:latest
MAINTAINER westzhao

EXPOSE 60000
```

==========Build, tag and push the base image==========

```
docker build --no-cache -t f3s-docker-file.tencentcloudcr.com/f3s-tcr/jmeter-mast
er:latest -f ./3.app/jmeter/Dockerfile-jmeter-master .
docker push f3s-docker-file.tencentcloudcr.com/f3s-tcr/jmeter-master:latest
```

2.7 Dockerfile-jmeter-slave

==========JMETER-SLAVES DOCKER FILE==========

```
# build
FROM f3s-docker-file.tencentcloudcr.com/f3s-tcr/jmeter-base:latest
MAINTAINER westzhao

EXPOSE 1099 50000

ENTRYPOINT $JMETER_HOME/bin/jmeter-server \
-Dserver.rmi.localport=50000 \
-Dserver_port=1099 \
-Jserver.rmi.ssl.disable=true
```

©2013-2022 Tencent Cloud. All rights reserved.

==========Build, tag and push the base image==========

```
docker build --no-cache -t f3s-docker-file.tencentcloudcr.com/f3s-tcr/jmeter-slav
e:latest -f ./3.app/jmeter/Dockerfile-jmeter-slave .
docker push f3s-docker-file.tencentcloudcr.com/f3s-tcr/jmeter-slave:latest
```

2.8 Dockerfile-jmeter-grafana-reporter

==========JMETER-GRAFANA-REPORTER DOCKER FILE==========

```
# build
# Multi-stage builds
FROM golang:1.14.7-alpine3.12 AS build
MAINTAINER westzhao
# Download the Ops and compilation tools
WORKDIR /go/src/${owner:-github.com/8710925}/reporter
# ADD . .
# RUN go install -v github.com/8710925/reporter/cmd/grafana-reporter

RUN apk --no-progress --purge --no-cache add --upgrade git && \
# Compile grafana-reporter
git clone https://${owner:-github.com/8710925}/reporter . \
&& go install -v github.com/8710925/reporter/cmd/grafana-reporter


# create grafana reporter image
FROM alpine:3.12
COPY --from=build /go/src/${owner:-github.com/8710925}/reporter/util/texlive.prof
ile /
COPY --from=build /go/src/${owner:-github.com/8710925}/reporter/util/SIMKAI.ttf /
usr/share/fonts/west/

RUN apk --no-progress --purge --no-cache add --upgrade wget \
curl \
fontconfig \
unzip \
tzdata \
perl-switch && \
wget -qO- \
"https://github.com/yihui/tinytex/raw/master/tools/install-unx.sh" | \
sh -s - --admin --no-path \
&& mv ~/.TinyTeX /opt/TinyTeX \
&& /opt/TinyTeX/bin/*/tlmgr path add \
&& tlmgr path add \
&& chown -R root:adm /opt/TinyTeX \
```

©2013-2022 Tencent Cloud. All rights reserved.

```
&& chmod -R g+w /opt/TinyTeX \
&& chmod -R g+wx /opt/TinyTeX/bin \
&& tlmgr update --self --repository http://mirrors.tuna.tsinghua.edu.cn/CTAN/syst
ems/texlive/tlnet \
&& tlmgr install epstopdf-pkg ctex everyshi everysel euenc \
# Change the time zone
&& cp /usr/share/zoneinfo/Asia/Shanghai /etc/localtime \
&& echo "Asia/Shanghai" > /etc/timezone \
&& apk del tzdata \
# Cleanup
&& fmtutil-sys --all \
&& texhash \
&& mktexlsr \
&& apk del --purge -qq \
&& rm -rf /var/lib/apt/lists/*

COPY --from=build /go/bin/grafana-reporter /usr/local/bin


ENTRYPOINT [ "/usr/local/bin/grafana-reporter","-ip","jmeter-grafana:3000" ]
```

==========Build, tag and push the base image==========

```
docker build --no-cache -t f3s-docker-file.tencentcloudcr.com/f3s-tcr/jmeter-graf
ana-reporter:latest -f ./3.app/jmeter/Dockerfile-jmeter-grafana-reporter .
docker push f3s-docker-file.tencentcloudcr.com/f3s-tcr/jmeter-grafana-reporter:la
test
```

2.9 Dockerfile-alpine-nginx

==========Alpine Nginx DOCKER FILE==========

```
# build
FROM f3s-docker-file.tencentcloudcr.com/f3s-tcr/alpine-kona:latest
MAINTAINER westzhao

ENV LANG=C.UTF-8

# Download the Ops tool
RUN apk --no-progress --purge --no-cache add --upgrade nginx && \
# Delete the APK cache && \
rm -rf /var/cache/apk/*

COPY ./3.app/nginx/default.conf /etc/nginx/http.d/default.conf
COPY ./3.app/nginx/nginx.conf /etc/nginx/nginx.conf
```

©2013-2022 Tencent Cloud. All rights reserved.

```
EXPOSE 80 443

CMD ["/usr/sbin/nginx", "-g", "daemon off;", "-c", "/etc/nginx/nginx.conf"]
```

==========Build, tag and push the base image==========

```
docker build --no-cache -t f3s-docker-file.tencentcloudcr.com/f3s-tcr/alpine-ngin
x:latest -f ./3.app/nginx/Dockerfile-alpine-nginx .
docker push f3s-docker-file.tencentcloudcr.com/f3s-tcr/alpine-nginx:latest
# To test run: docker run --name test -it --rm -p 8888:80 f3s-docker-file.tencent
cloudcr.com/f3s-tcr/alpine-nginx:latest nginx -v
# docker export <container-id> | docker import f3s-docker-file.tencentcloudcr.co
m/f3s-tcr/alpine-nginx:latest
# quick interative termnal: docker run -it --entrypoint=sh f3s-docker-file.tencen
tcloudcr.com/f3s-tcr/alpine-nginx:latest sh
```

©2013-2022 Tencent Cloud. All rights reserved.

Microservice
Hosting Dubbo to TKE

Last updated：2023-02-15 10:53:54

Overview

This document describes how to host a Dubbo application to TKE.

Strengths of hosting Dubbo applications to TKE

- Improve the resource utilization.
- Kubernetes is a natural fit for microservice architectures.
- Improve the Ops efficiency and facilitate DevOps implementation.
- Highly scalable Kubernetes makes it easy to dynamically scale applications.
- TKE provides Kubernetes master management to ease Kubernetes cluster Ops and management.
- TKE is integrated with other cloud-native products of Tencent Cloud to help you better use Tencent Cloud products.

Best Practices

The following describes how to host a Dubbo application to TKE by using the Q Cloud Book Mall (QCBM) project as an example.

QCBM overview

QCBM is an online bookstore demo project developed by using the microservice architecture and the Dubbo 2.7.8 framework. It is deployed and hosted on CODING. For more information, see here. QCBM contains the following microservices:

Microservice	Description
QCBM-Front	Frontend project developed through React, built and deployed based on the Nginx 1.19.8 Docker image.
QCBM-Gateway	API gateway that accepts HTTP requests from the frontend and converts them into Dubbo requests at the backend.
User-Service	Dubbo-based microservice, providing user registration, login, and authentication features.

©2013-2022 Tencent Cloud. All rights reserved.

Microservice	Description
Favorites-Service	Dubbo-based microservice, providing book favorites.
Order-Service	Dubbo-based microservice, providing order generation and query features.
Store-Service	Dubbo-based microservice, providing the book information storage feature.

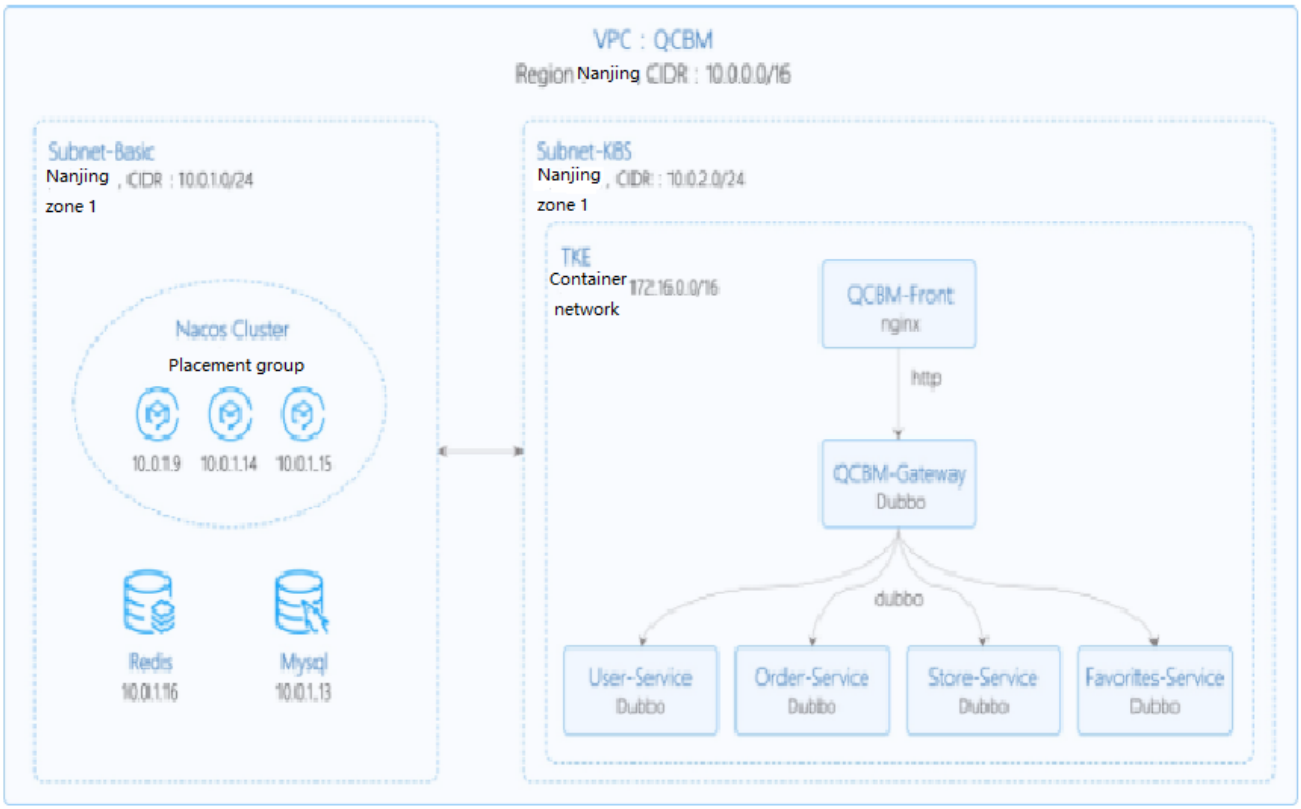
QCBM architecture and add-ons

In the following best practice, applications deployed in CVM are containerized and hosted to TKE. In this use case, one VPC is used and divided into two subnets:

- **Subnet-Basic**: Deployed with stateful basic services, including Dubbo's service registry Nacos, MySQL, and Redis.
- **Subnet-K8S**: Deployed with QCBM application services, all of which are containerized and run in TKE.

©2013-2022 Tencent Cloud. All rights reserved.

The VPC is divided as shown below:



The network planning for the QCBM instance is as shown below:

Network Planning	Description
Region/AZ	Nanjing/Nanjing Zone 1
VPC	CIDR: 10.0.0.0/16
Subnet-Basic	Nanjing Zone 1, CIDR block: 10.0.1.0/24
Subnet-K8S	Nanjing Zone 1, CIDR block: 10.0.2.0/24
Nacos cluster	Nacos cluster built with three 1-core 2 GB MEM Standard SA2 CVM instances, with IP addresses of 10.0.1.9, 10.0.1.14, and 10.0.1.15

©2013-2022 Tencent Cloud. All rights reserved.

The add-ons used in the QCBM instance are as shown below:

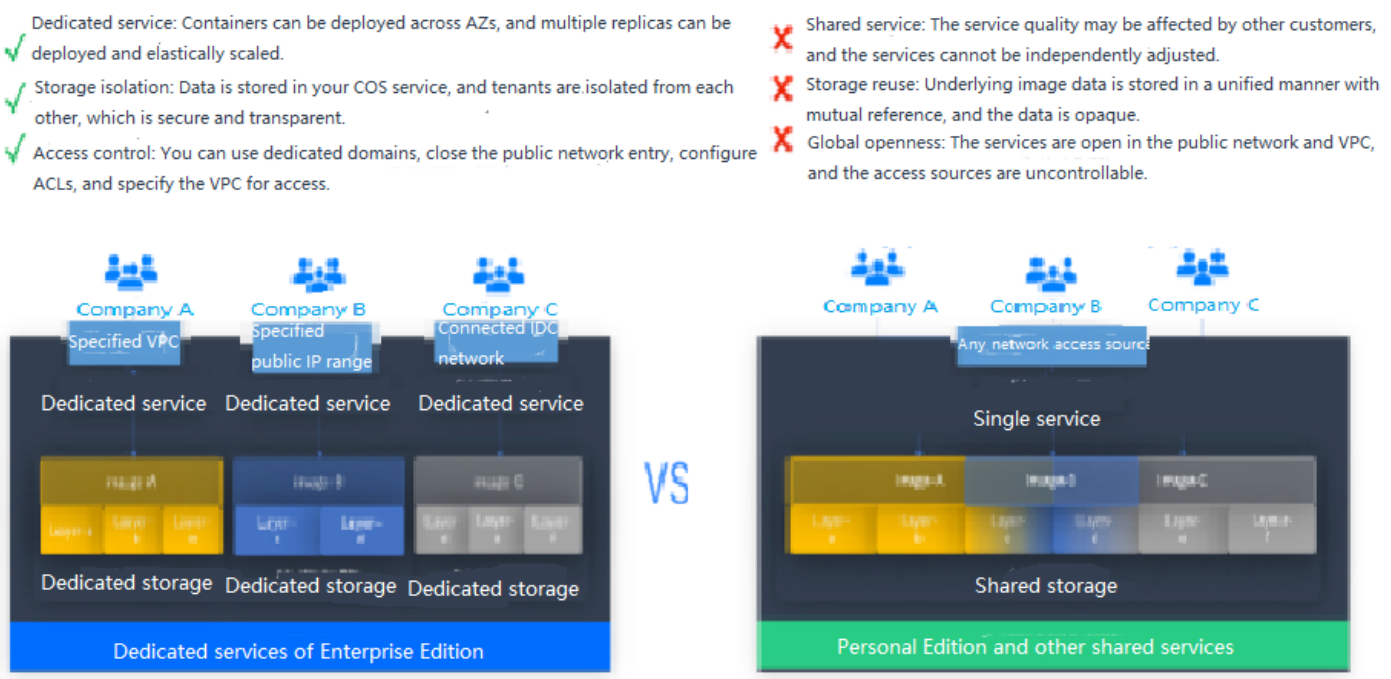
Add-on	Version	Source	Remarks
k8s	1.8.4	Tencent Cloud	TKE management mode
MySQL	5.7	Tencent Cloud	TencentDB for MySQL with two nodes
Redis	5.0	Tencent Cloud	TencentDB for Redis Standard Edition
CLS	N/A	Tencent Cloud	Log service
TSW	N/A	Tencent Cloud	Accessed with SkyWalking 8.4.0 Agent, which can be downloaded here
Java	1.8	Open-source community	Docker image of Java 8 JRE
Nacos	2.0.0	Open-source community	Download here
Dubbo	2.7.8	Open-source community	GitHub address

Overview

TCR

Tencent Cloud Tencent Container Registry (TCR) are available in Personal Edition and Enterprise Edition as differentiated below:

©2013-2022 Tencent Cloud. All rights reserved.



QCBM is a Dubbo containerized demo project, so TCR Personal Edition is perfectly suited to its needs. However, for enterprise users, TCR Enterprise Edition is recommended. To use an image repository, see Basic Image Repository Operations.

TSW

Tencent Service Watcher (TSW) provides cloud-native service observability solutions that can trace upstream and downstream dependencies in distributed architectures, draw topologies, and provide multidimensional call

©2013-2022 Tencent Cloud. All rights reserved.

observation by service, API, instance, and middleware. It is further described as shown below:

TSW is architecturally divided into four modules:

Show All

Data collection (client)

展开&收起

You can use an open-source probe or SDK to collect data. If you are migrating to the cloud, you can change the reporting address and authentication information only and keep most of the configurations on the client.

Data processing (server)

展开&收起

Data is reported to the server via the Pulsar message queue, converted by the adapter into an OpenTracing-compatible format, and assigned to real-time and offline computing as needed.

- Real-time computing provides real-time monitoring, statistical data display, and fast response to the connected alarming platform.
- Offline computing aggregates the statistical data in large amounts over long periods of time and leverages big data analytics to provide business value.

©2013-2022 Tencent Cloud. All rights reserved.

Storage

展开&收起

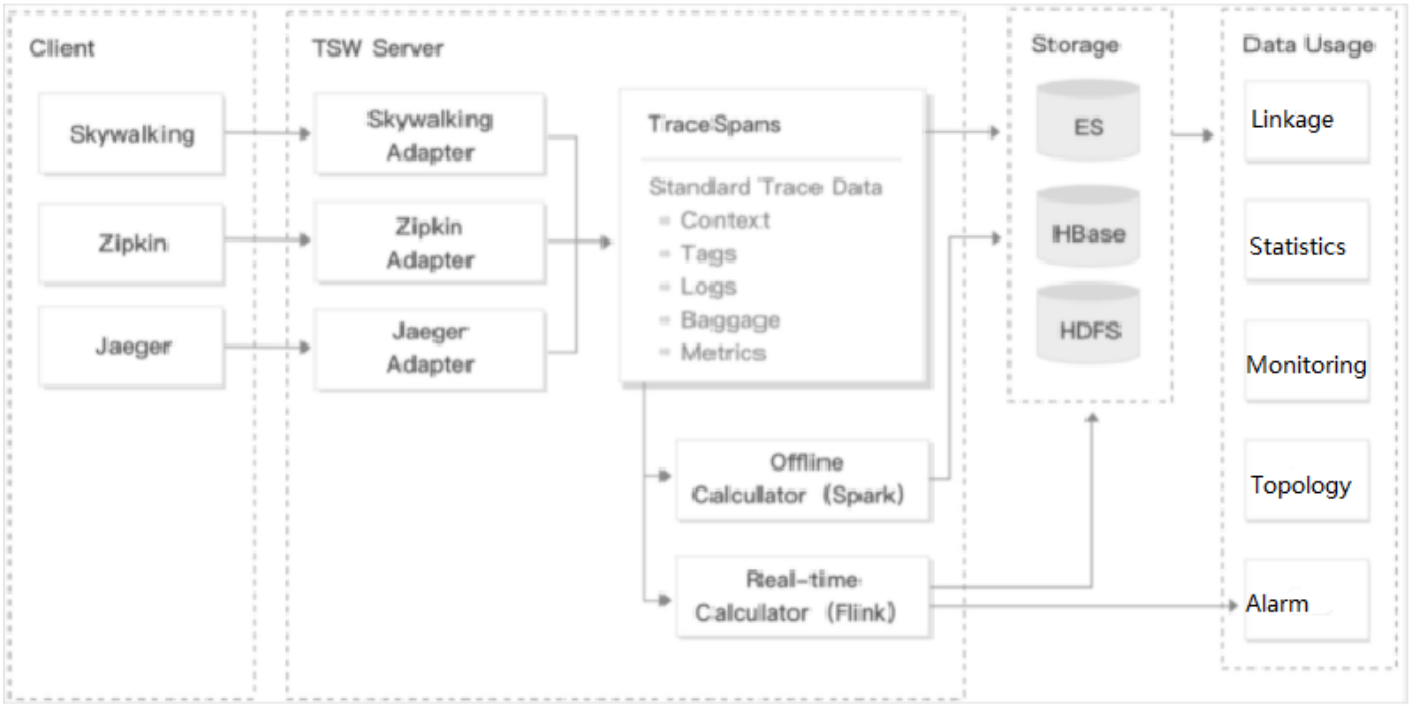
The storage layer can adapt to use cases with different data types, writing at the server layer, and query and reading requests at the data usage layer.

Data usage

展开&收起

The data usage layer provides underlying support for console operations, data display, and alarming.

The architecture is as shown below:



Directions

Building basic service cluster

©2013-2022 Tencent Cloud. All rights reserved.

- In the TencentDB for MySQL console, create an instance and use qcbm-ddl.sql to initialize it. For more information, see Creating MySQL Instance.
- In the TencentDB for Redis console, create an instance and initialize it. For more information, see Creating TencentDB for Redis Instance.
- In the CLB console, create a **private network** CLB instance for `Subnet-K8S` (the ID of this CLB instance will be used later). For more information, see Creating CLB Instances.
- Apply for the TSW beta test. TSW is currently in beta test and supports both Java and Go.
- Deploy the Nacos cluster:
  i. In the CVM console, purchase three 1-core 2 GB MEM Standard SA2 CVM instances. For more information, see Creating Instances via CVM Purchase Page.
  ii. Log in to the instance and run the following command to install Java.

```
yum install java-1.8.0-openjdk.x86_64
```

  Run the following command. If Java version information is output, Java is successfully installed.

```
java - version
```

  iii. Deploy the Nacos cluster as instructed in Cluster deployment instructions.

Building Docker image

Writing Dockerfile

The following uses `user-service` as an example to describe how to write a Dockerfile. The project directory structure of `user-service` is displayed, **Dockerfile** is in the root directory of the project, and **user-service-1.0.0.zip** is the packaged file that needs to be added to the image.

```
➜ user-service tree
├── Dockerfile
├── assembly
│ ....
├── bin
│ ....
├── pom.xml
├── src
│ ....
├── target
│ .....
│ └── user-service-1.0.0.zip
└── user-service.iml
```

The Dockerfile of `user-service` is as shown below:

©2013-2022 Tencent Cloud. All rights reserved.

```
FROM java:8-jre

ARG APP_NAME=user-service
ARG APP_VERSION=1.0.0
ARG FULL_APP_NAME=${APP_NAME}-${APP_VERSION}

# The working directory of the container is `/app`.
WORKDIR /app

# Add the locally packaged application to the image.
COPY ./target/${FULL_APP_NAME}.zip .

# Create the `logs` directory. Decompress and delete the original files and direc
tory after the decompression.
RUN mkdir logs \
&& unzip ${FULL_APP_NAME}.zip \
&& mv ${FULL_APP_NAME}/** . \
&& rm -rf ${FULL_APP_NAME}*

# Start script and parameters of `user-service`
ENTRYPOINT ["/app/bin/user-service.sh"] CMD ["start", "-t"]

# Dubbo port number
EXPOSE 20880
```

Note：

- Java applications in the production environment have a lot of configuration parameters, making the start script complex. It's a heavy workload to write all the content of the start script to the Dockerfile, which is far less flexible than shell scripts and can't implement fast troubleshooting. We recommend you not enable the start script.
- In general, **nohup** is used at the end of the start script to start the Java application, but the deamon process that comes along will cause the container to exit directly after execution. Therefore, you need to change `nohup java ${OPTIONS} -jar user-service.jar > ${LOG_PATH} 2>&1 &` to `java ${OPTIONS} -jar user-service.jar > ${LOG_PATH} 2>&1` .
- As each Run command in the Dockerfile will generate an image layer, we recommend you combine these commands into one.

Building image

©2013-2022 Tencent Cloud. All rights reserved.

TCR provides both automatic and manual methods to build an image. To demonstrate the build process, the manual method is used.

The image name needs to be in line with the convention of `ccr.ccs.tencentyun.com/[namespace]/[ImageName]:[image tag]` :

- Here, `namespace` can be the project name to facilitate image management and use. In this document, `QCBM` represents all the images under the QCBM project.
- `ImageName` can contain the `subpath` , generally used for multi-project use cases of enterprise users. In addition, if a local image is already built, you can run the `docker tag` command to rename the image in line with the naming convention.

1. Run the following command to build an image as shown below:

```
# Recommended build method, which eliminates the need for secondary tagging operations
sudo docker build -t ccr.ccs.tencentyun.com/[namespace]/[ImageName]:[image tag]
# Build a local `user-service` image. The last `.` indicates that the Dockerfile is stored in the current directory (`user-service`).
➜ user-service docker build -t ccr.ccs.tencentyun.com/qcbm/user-service:1.0.0 .
# Rename existing images in line with the naming convention
sudo docker tag [ImageId] ccr.ccs.tencentyun.com/[namespace]/[ImageName]:[image tag]
```

2. After the build is complete, you can run the following command to view all the images in your local repository.

```
docker images
```

A sample is as shown below:

```
➜  qcbm docker images
REPOSITORY                                      TAG                 IMAGE ID            CREATED             SIZE
ccr.ccs.tencentyun.com/qcbm/qcbm-gateway        1.0.0               b9516e1a0717        About an hour ago   558MB
ccr.ccs.tencentyun.com/qcbm/favorites-service   1.0.0               157465cc30f2        About an hour ago   512MB
ccr.ccs.tencentyun.com/qcbm/order-service       1.0.0               aad52ddfc3d7        About an hour ago   512MB
ccr.ccs.tencentyun.com/qcbm/store-service       1.0.0               a7fcc435820f        About an hour ago   509MB
ccr.ccs.tencentyun.com/qcbm/user-service        1.0.0               cdc6910691ef        About an hour ago   512MB
java                                            8-jre               e44d62cf8862        4 years ago         311MB
➜  qcbm
```

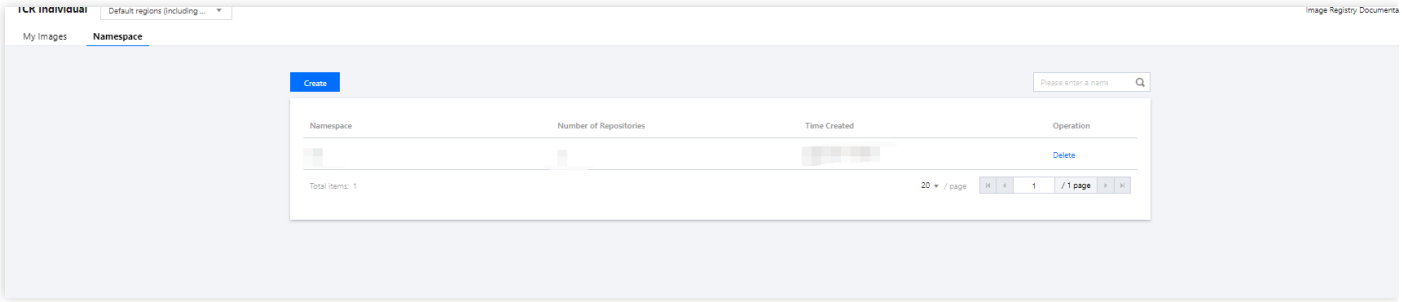
Uploading image to TCR

©2013-2022 Tencent Cloud. All rights reserved.

Creating namespace

The QCBM project uses TCR Personal Edition (TCR Enterprise Edition is recommended for enterprise users).

1. Log in to the TKE console.
2. Click **TCR** > **Personal** > **Namespace** to enter the **Namespace** page.
3. Click **Create** and create the `qcbm` namespace in the pop-up window. All the images of the QCBM project are stored under this namespace as shown below:



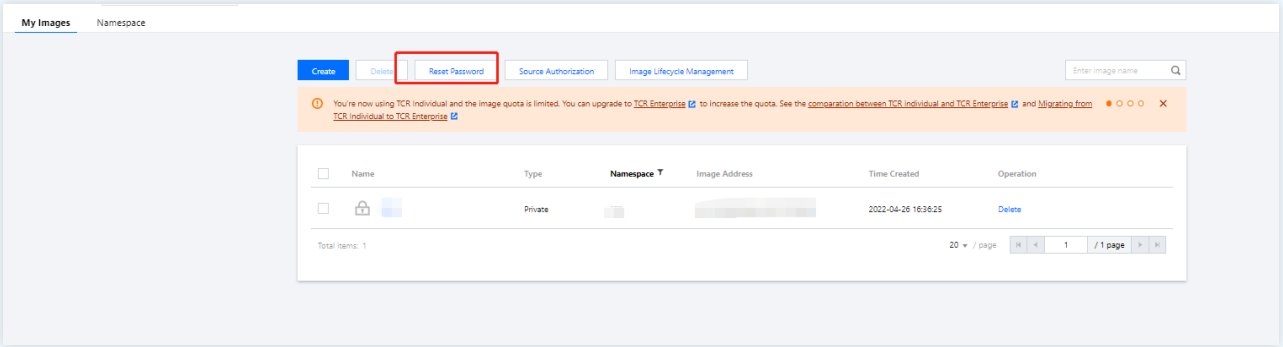
Uploading image

Log in to TCR and upload an image.

1. Run the following command to log in to TCR.

```
docker login --username=[Tencent Cloud account ID] ccr.ccs.tencentyun.com
```

- You can get your Tencent Cloud account ID on the Account Info page.
- If you forget your **TCR login password**, you can reset it in My Images of TCR Personal Edition.

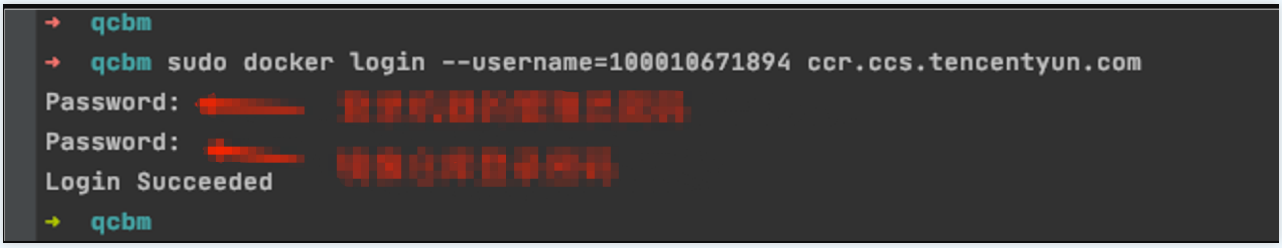


- If you are prompted that you have no permission to run the command, add `sudo` before the command and run it as shown below. In this case, you need to enter two passwords, the server admin password required for `sudo` and the **TCR login password**.

©2013-2022 Tencent Cloud. All rights reserved.

```
sudo docker login --username=[Tencent Cloud account ID] ccr.ccs.tencentyun.com
```

As shown below:



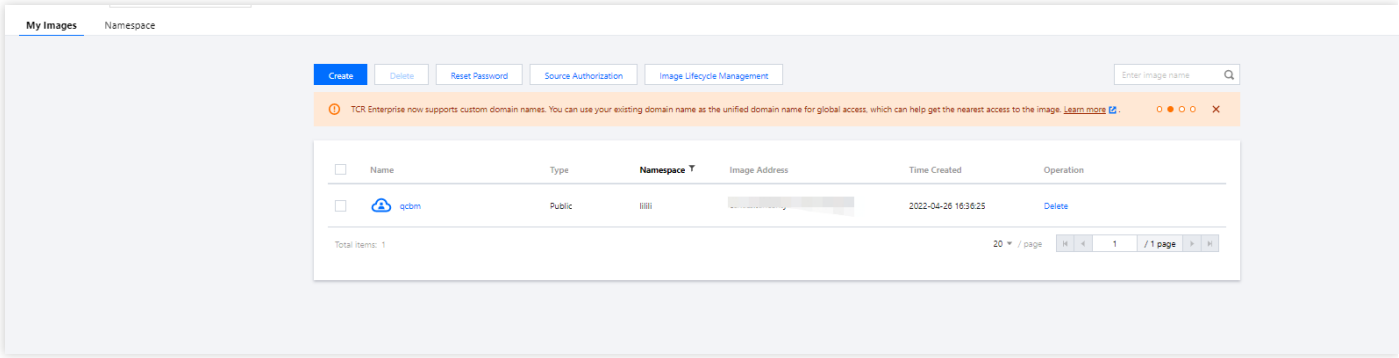
2. Run the following command to push the locally generated image to TCR.

```
docker push ccr.ccs.tencentyun.com/[namespace]/[ImageName]:[image tag]
```

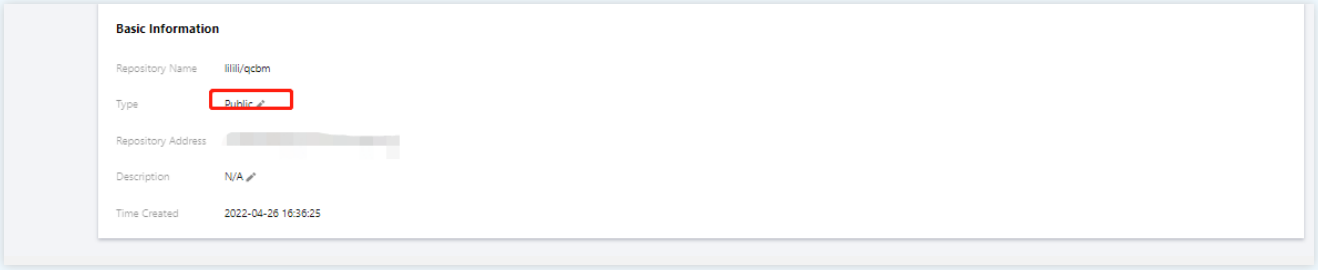
As shown below:

3. In My Images, you can view all the uploaded images. The following figure shows the five QCBM images uploaded to TCR.

©2013-2022 Tencent Cloud. All rights reserved.



The default image type is `Private`. If you want to let others use the image, you can set it to `Public` in **Image Info** as shown below:



Deploying service in TKE

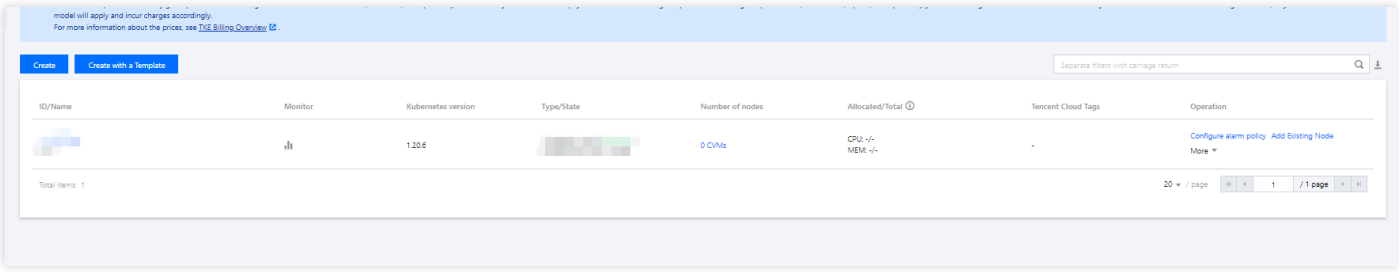
Creating K8s cluster of QCBM

1. Before the deployment, you need to create a K8s cluster as instructed in Quickly Creating a Standard Cluster.

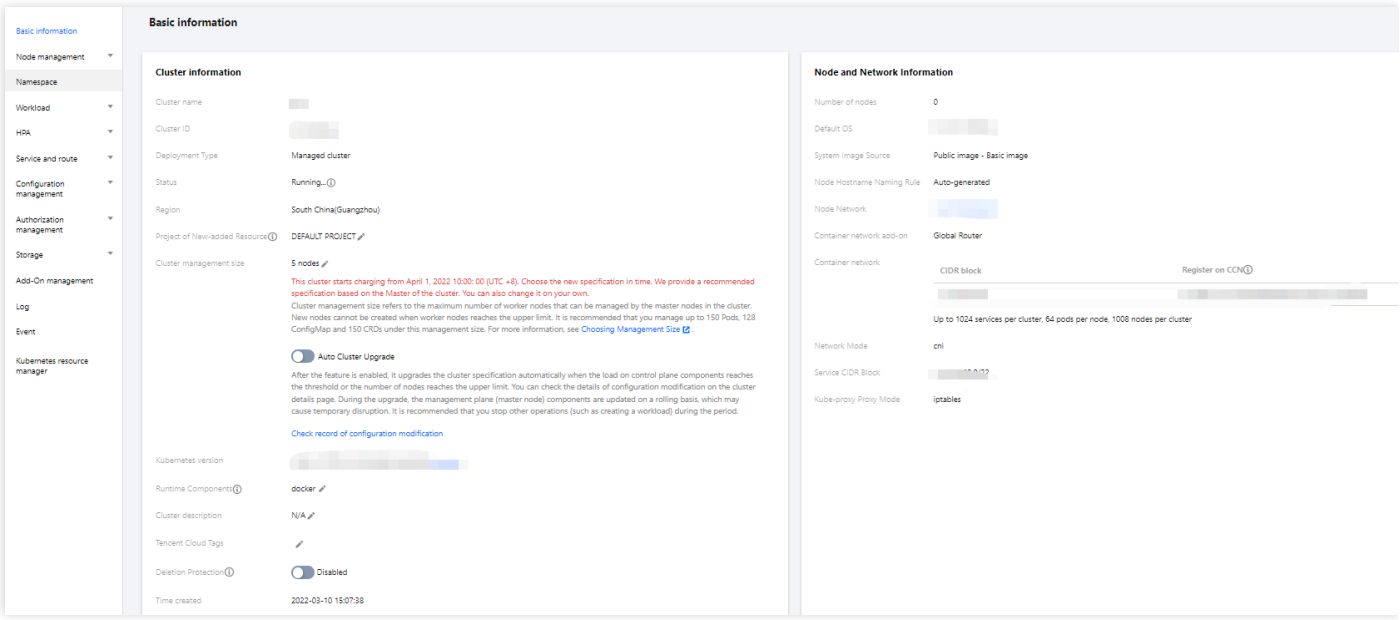
Note：
When a cluster is created, we recommend you enable **Placement Group** on the **Select Model** page. It helps distribute CVM instances across different hosts to increase the system reliability.

2. After the cluster is created, you can view its information on the cluster management page in the TKE console. Here, the new cluster is named `qcbm-k8s-demo` as shown below:

©2013-2022 Tencent Cloud. All rights reserved.



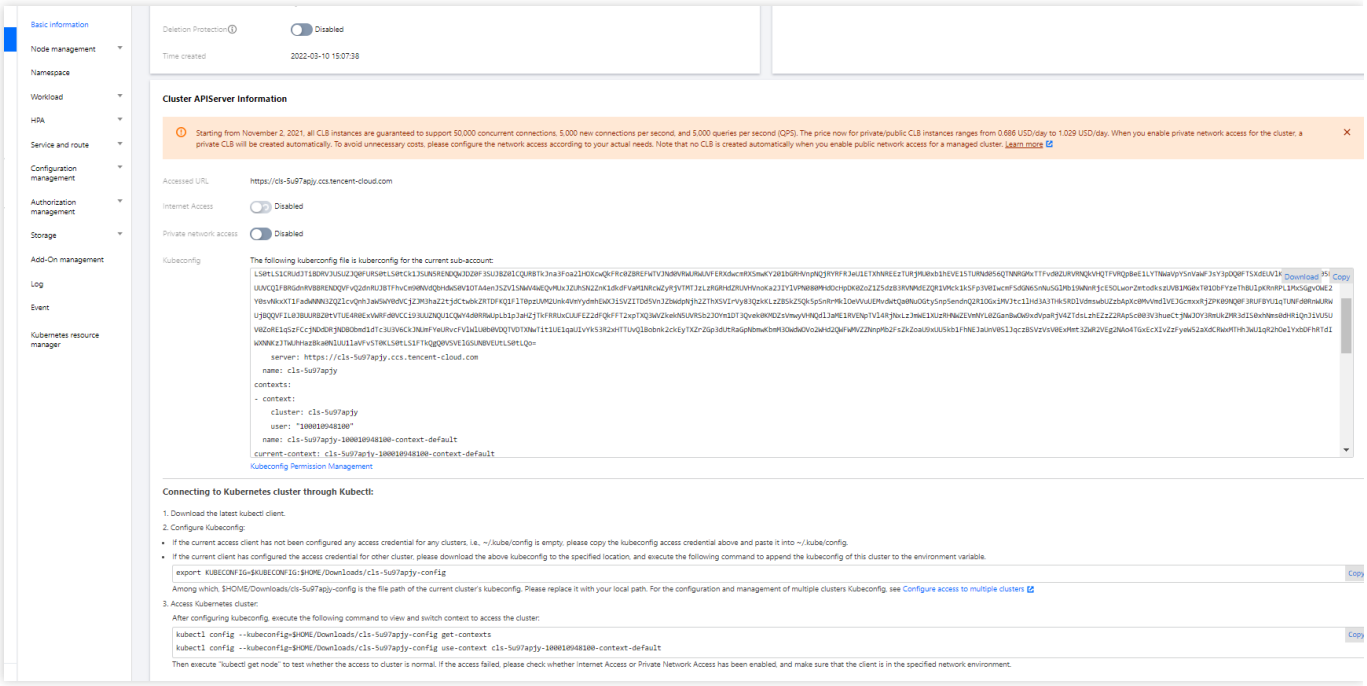
3. Click the **Cluster Name** to enter the **Basic Info** page to view the cluster configuration information as shown below:



4. (Optional) If you want to use K8s management tools such as kubectl and Lens, you need to follow two steps:
   i. Enable public network access.
   ii. Store the API authentication token in the local `config` file under `user home/.kube` (choose another if the `config` file has content) to ensure that the default cluster can be accessed each time. If you choose not to store the token in the `config` file under `.kube`, see the **Instructions on Connecting to Kubernetes**

©2013-2022 Tencent Cloud. All rights reserved.

Cluster via kubectl under **Cluster API Server Info** in the console as shown below:



Creating namespace

A namespace is a logical environment in a Kubernetes cluster that allows you to divide teams or projects. You can create a namespace in the following three methods, and method 1 is recommended.

- Method 1. Use the command line
- Method 2. Use the console
- Method 3. Use YAML

Run the following command to create a namespace:

```
kubectl create namespace qcbm
```

Using ConfigMap to store configuration information

ConfigMap allows you to decouple the configuration from the running image, making the application more portable. The QCBM backend service needs to get the Nacos, MySQL, and Redis host and port information from the environment variables and store them by using ConfigMap.
You can use ConfigMap to store configuration information in the following two methods:

- Method 1. Use YAML
- Method 2. Use the console

©2013-2022 Tencent Cloud. All rights reserved.

The following is the ConfigMap YAML for QCBM, where **values of pure digits require double quotation marks**, for example, `MYSQL_PORT` in the sample YAML below:

```
# Create a ConfigMap.

apiVersion: v1
kind: ConfigMap
metadata:
name: qcbm-env
namespace: qcbm
data:
NACOS_HOST: 10.0.1.9
MYSQL_HOST: 10.0.1.13
REDIS_HOST: 10.0.1.16
NACOS_PORT: "8848"
MYSQL_PORT: "3306"
REDIS_PORT: "6379"
SW_AGENT_COLLECTOR_BACKEND_SERVICES: xxx # TSW access address as described below
```

Using Secret to store sensitive information

A Secret can be used to store sensitive information such as passwords, tokens, and keys to reduce exposure risks. QCBM uses it to store account and password information.
You can use a Secret to store sensitive information in the following two methods:

- Method 1. Use YAML
- Method 2. Use the console

The following is the YAML for creating a Secret in QCBM, where the `value` of the Secret needs to be a Base64-encoded string.

```
# Create a Secret.
apiVersion: v1
kind: Secret
metadata:
name: qcbm-keys
namespace: qcbm
labels:
qcloud-app: qcbm-keys
data:
# xxx is the Base64-encoded string, which can be generated by using the echo -n
raw string | base64 shell command.
MYSQL_ACCOUNT: xxx
MYSQL_PASSWORD: xxx
```

©2013-2022 Tencent Cloud. All rights reserved.

```
REDIS_PASSWORD: xxx
SW_AGENT_AUTHENTICATION: xxx # TSW access token as described below
type: Opaque
```

Deploying Deployment

A Deployment declares the Pod template and controls the Pod running policy, which is suitable for deploying stateless applications. Both front and Dubbo services of QCBM are stateless applications and can use the Deployment.

YAML parameters for the `user-service` Deployment are as shown below:

Parameter	Description
replicas	Indicates the number of Pods to be created.
image	Image address
imagePullSecrets	The key to pull an image, which can be obtained from **Cluster** > **Configuration Management** > **Secret**. It is not required for public images.
env	• Defines Pod environment variables and values. • The `key-value` defined in the ConfigMap can be referenced by using `configMapKeyRef` . • The `key-value` defined in the Secret can be referenced by using `secretKeyRef` .
ports	Specifies the port number of the container. It is `20880` for Dubbo applications.

A complete sample YAML file for the `user-service` Deployment is as follows:

```
# user-service Deployment

apiVersion: apps/v1
kind: Deployment
metadata:
name: user-service
namespace: qcbm
labels:
app: user-service
version: v1
spec:
replicas: 1
selector:
matchLabels:
app: user-service
version: v1
```

©2013-2022 Tencent Cloud. All rights reserved.

```
template:
metadata:
labels:
app: user-service
version: v1
spec:
containers:
- name: user-service
image: ccr.ccs.tencentyun.com/qcbm/user-service:1.1.4
env:
- name: NACOS_HOST # IP address of the Dubbo service registry Nacos
valueFrom:
configMapKeyRef:
key: NACOS_HOST
name: qcbm-env
optional: false
- name: MYSQL_HOST # MySQL address
valueFrom:
configMapKeyRef:
key: MYSQL_HOST
name: qcbm-env
optional: false
- name: REDIS_HOST # Redis IP address
valueFrom:
configMapKeyRef:
key: REDIS_HOST
name: qcbm-env
optional: false
- name: MYSQL_ACCOUNT # MySQL account
valueFrom:
secretKeyRef:
key: MYSQL_ACCOUNT
name: qcbm-keys
optional: false
- name: MYSQL_PASSWORD # MySQL password
valueFrom:
secretKeyRef:
key: MYSQL_PASSWORD
name: qcbm-keys
optional: false
- name: REDIS_PASSWORD # Redis password
valueFrom:
secretKeyRef:
key: REDIS_PASSWORD
name: qcbm-keys
optional: false
- name: SW_AGENT_COLLECTOR_BACKEND_SERVICES # SkyWalking backend service address
```

©2013-2022 Tencent Cloud. All rights reserved.

```
valueFrom:
configMapKeyRef:
key: SW_AGENT_COLLECTOR_BACKEND_SERVICES
name: qcbm-env
optional: false
- name: SW_AGENT_AUTHENTICATION # Authentication token for SkyWalking Agent to c
onnect to the backend service
valueFrom:
secretKeyRef:
key: SW_AGENT_AUTHENTICATION
name: qcbm-keys
optional: false
ports:
- containerPort: 20880 # Dubbo port name
protocol: TCP
imagePullSecrets: # The key to pull the image. It is not required as the images
of all QCBM services are public.
- name: qcloudregistrykey
```

Deploying Service

You can specify the Service type with Kubernetes `ServiceType`, which defaults to `ClusterIP`. Valid values of `ServiceType` include the following:

- LoadBalancer: Provides public network, VPC, and private network access.
- NodePort: : Accesses services through the CVM IP and host port.
- ClusterIP: Accesses services through the service name and port.

For a production system, the gateway needs to be accessible within the VPC or private network, and the front needs to provide access to the private and public networks. Therefore, you need to set `ServiceType` to `LoadBalancer` for the QCBM gateway and front.
TKE enriches the `LoadBalancer` mode by configuring the Service through annotations.

If you use the `service.kubernetes.io/qcloud-loadbalancer-internal-subnetid` annotations, a private network CLB instance will be created when the Service is deployed. In general, we recommend you create the CLB instance in advance and use the `service.kubernetes.io/loadbalance-id` annotations in the deployment YAML to improve the efficiency.

The deployment YAML for the `qcbm-front` Service is as follows:

```
# Deploy the `qcbm-front` Service.
apiVersion: v1
kind: Service
metadata:
```

©2013-2022 Tencent Cloud. All rights reserved.

```
name: qcbm-front
namespace: qcbm
annotations:
# ID of the CLB instance of `Subnet-K8S`
service.kubernetes.io/loadbalance-id: lb-66pq34pk
spec:
externalTrafficPolicy: Cluster
ports:
- name: http
port: 80
targetPort: 80
protocol: TCP
selector: # Map the backend `qcbm-gateway` to the Service.
app: qcbm-front
version: v1
type: LoadBalancer
```

Deploying Ingress

An Ingress is a collection of rules that allow external access to the cluster Service, thereby eliminating the need to expose the Service. For QCBM projects, you need to create an Ingress for `qcbm-front`, which corresponds to the following YAML:

```
# Deploy the `qcbm-front` Ingress.

apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
name: front
namespace: qcbm
annotations:
ingress.cloud.tencent.com/direct-access: "false"
kubernetes.io/ingress.class: qcloud
kubernetes.io/ingress.extensiveParameters: '{"AddressIPVersion":"IPV4"}'
kubernetes.io/ingress.http-rules: '[{"host":"qcbm.com","path":"/","backend":{"se
rviceName":"qcbm-front","servicePort":"80"}}]'
spec:
rules:
- host: qcbm.com
http:
paths:
- path: /
backend: # Associate with backend services.
serviceName: qcbm-front
servicePort: 80
```

©2013-2022 Tencent Cloud. All rights reserved.

Viewing deployment result

So far, you have completed the deployment of QCBM in TKE and can view the deployment result in the following steps:

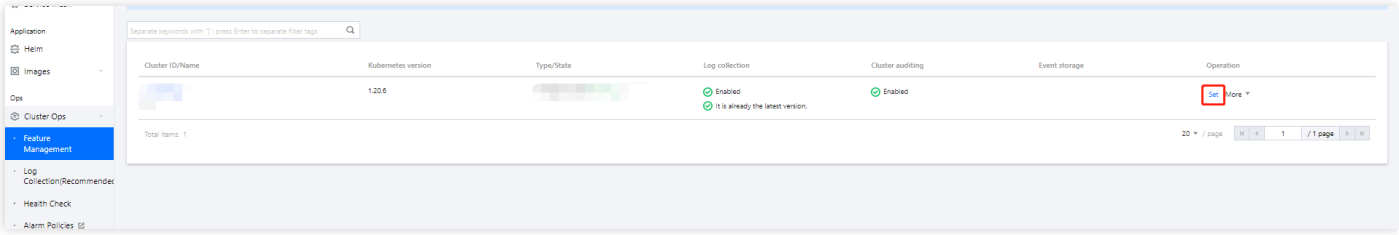
1. Log in to the TKE console and click the **Cluster ID/Name** to enter the cluster details page.
2. Click **Services and Routes** > **Ingress** to enter the **Ingress** page, where you can see the created Ingress. You can access the QCBM page through the Ingress VIP.

Integrating CLS

Enabling container log collection

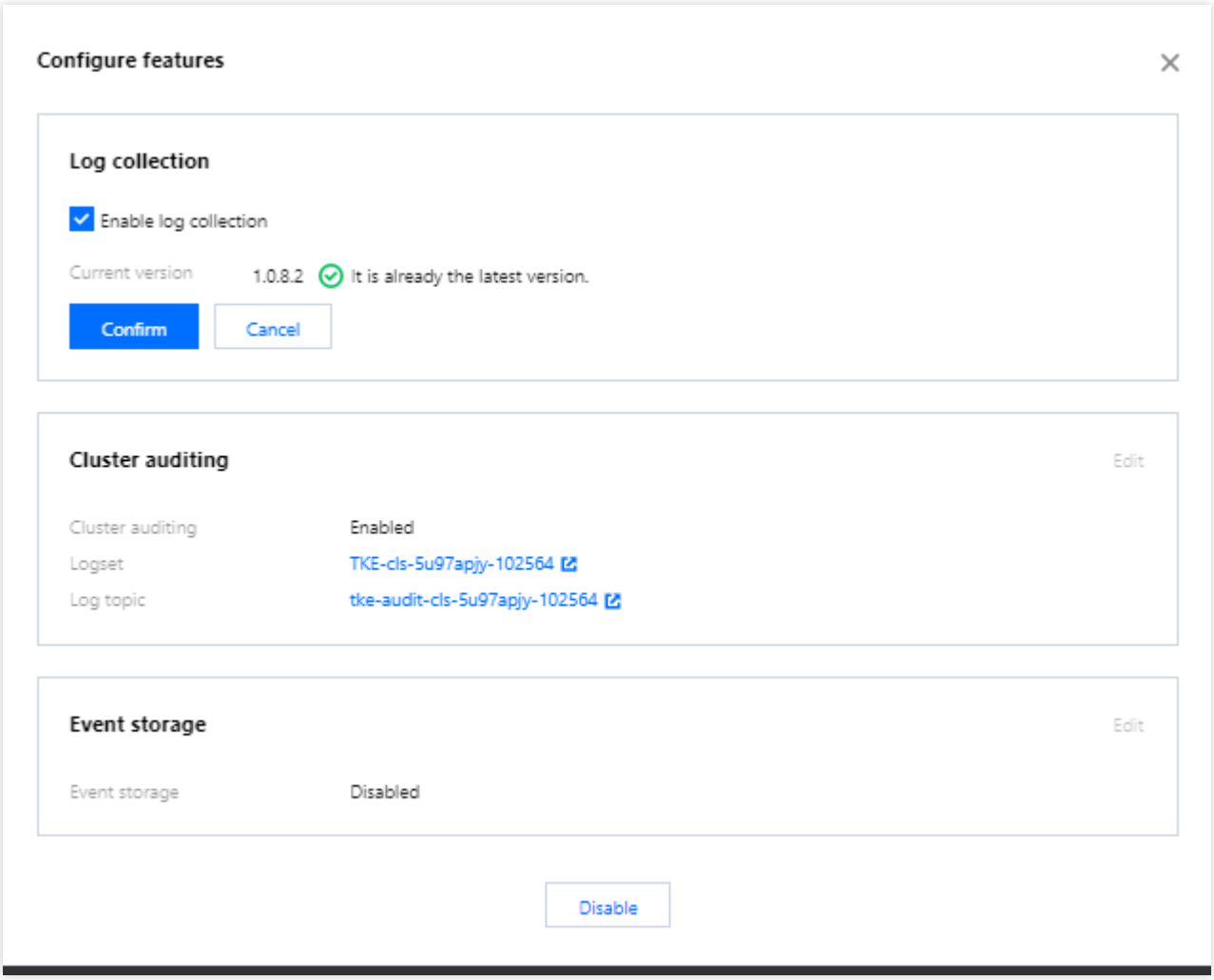
The container log collection feature is disabled by default and needs to be enabled as instructed below:

1. Log in to the TKE console and click **Cluster Ops** > **Feature Management** on the left sidebar.
2. At the top of the **Feature Management** page, select the region. On the right of the target cluster, click **Set**.



3. On the **Configure Features** page, click **Edit** for log collection and select **Enable Log Collection** as shown below:

©2013-2022 Tencent Cloud. All rights reserved.



4. Click **OK**.

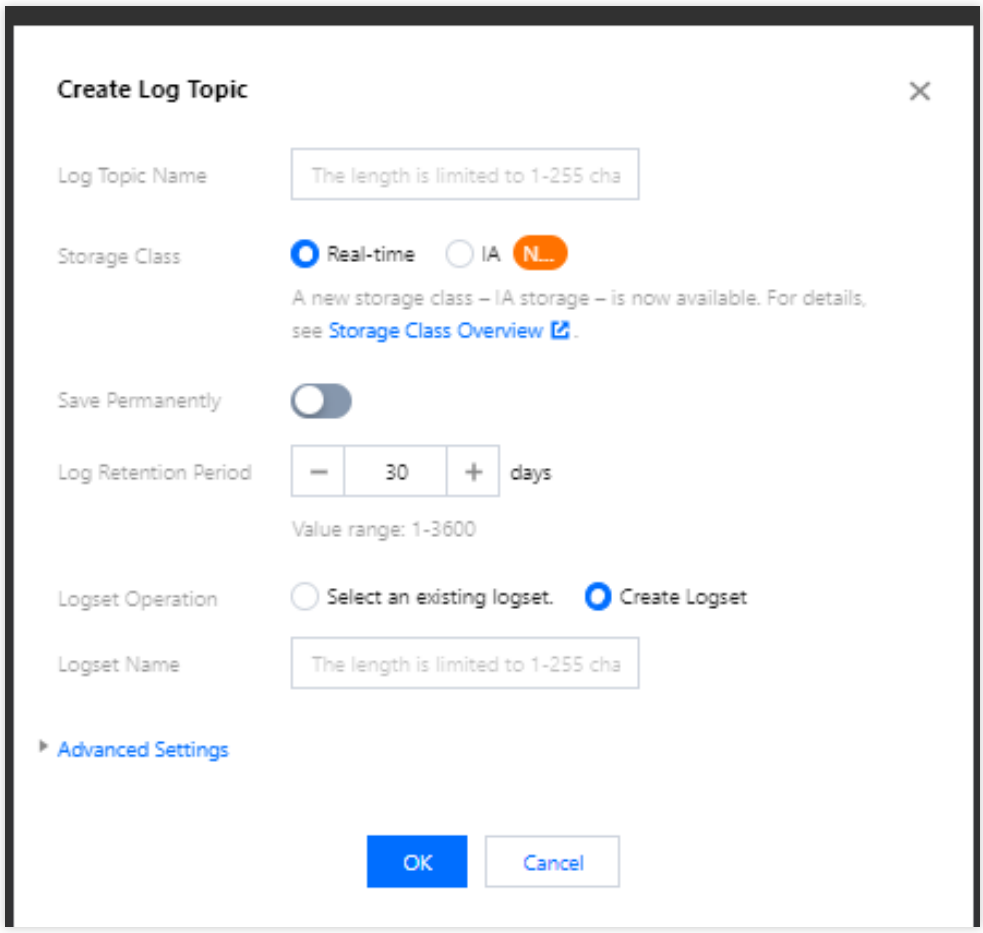
Creating log topic and logset

QCBM is deployed in Nanjing region, so you need to select Nanjing region when creating logsets:

1. Log in to the CLS console and select Nanjing region on the **Log Topic** page.

©2013-2022 Tencent Cloud. All rights reserved.

2. Click **Create Log Topic** and enter the relevant information in the pop-up window as prompted as shown below:



- **Log Topic Name**: Enter `qcbm` .
- **Logset Operation**: Select **Create Logset**.
- **Logset Name**: Enter `qcbm-logs` .

3. Click **OK**.

> Note：
> As QCBM has multiple backend microservices, you can create a log topic for each microservice to facilitate log categorization.
>
> - A log topic is created for each QCBM service.
> - You need the log topic ID when creating log rules for containers.

Configuring log collection rule

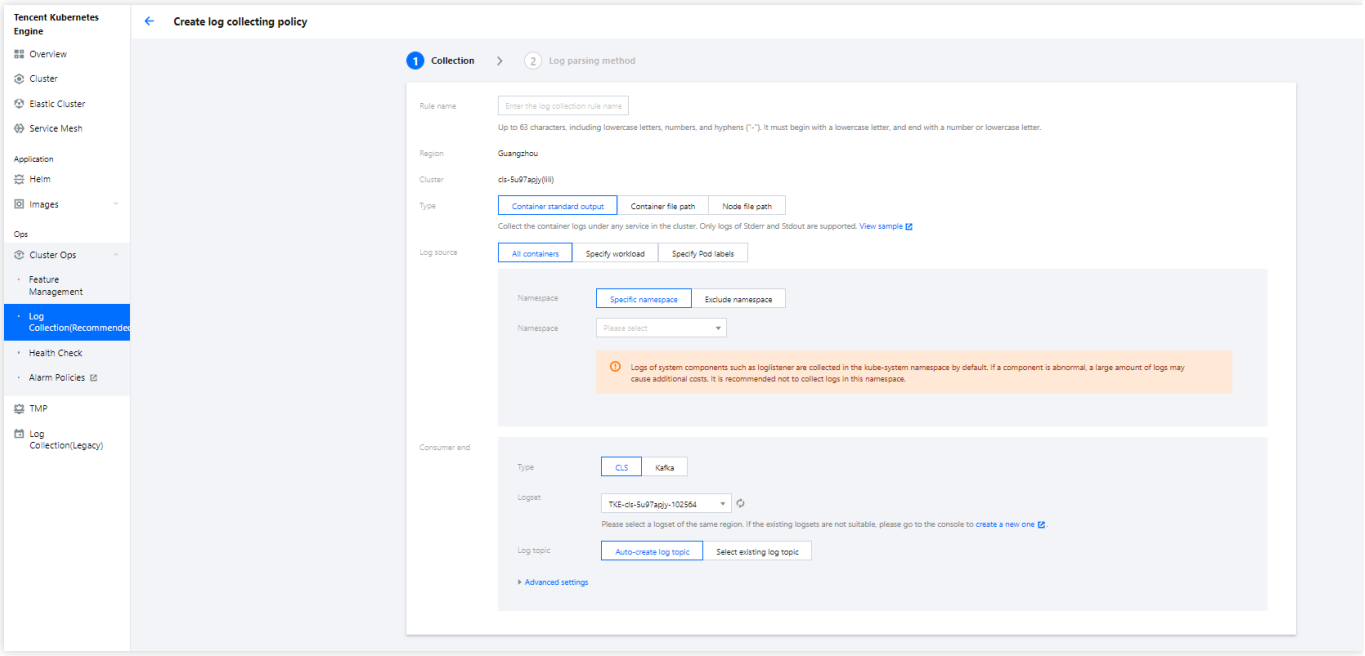
You can configure container log collection rules in the console or with CRD.

©2013-2022 Tencent Cloud. All rights reserved.

- Method 1. Use the console
- Method 2. Use CRD

Log rules specify the location of a log in a container:

1. Log in to the TKE console and click **Cluster Ops** > **Log Rules** on the left sidebar.
2. On the **Log Rules** page, click **Create** to create a rule.
   - **Log Source**: Specify the location of a log in a container. All the QCBM logs are output to the `/app/logs` directory, so you can use the container file path to specify the workload and log location.
   - **Consumer**: Select the previously created logset and topic.

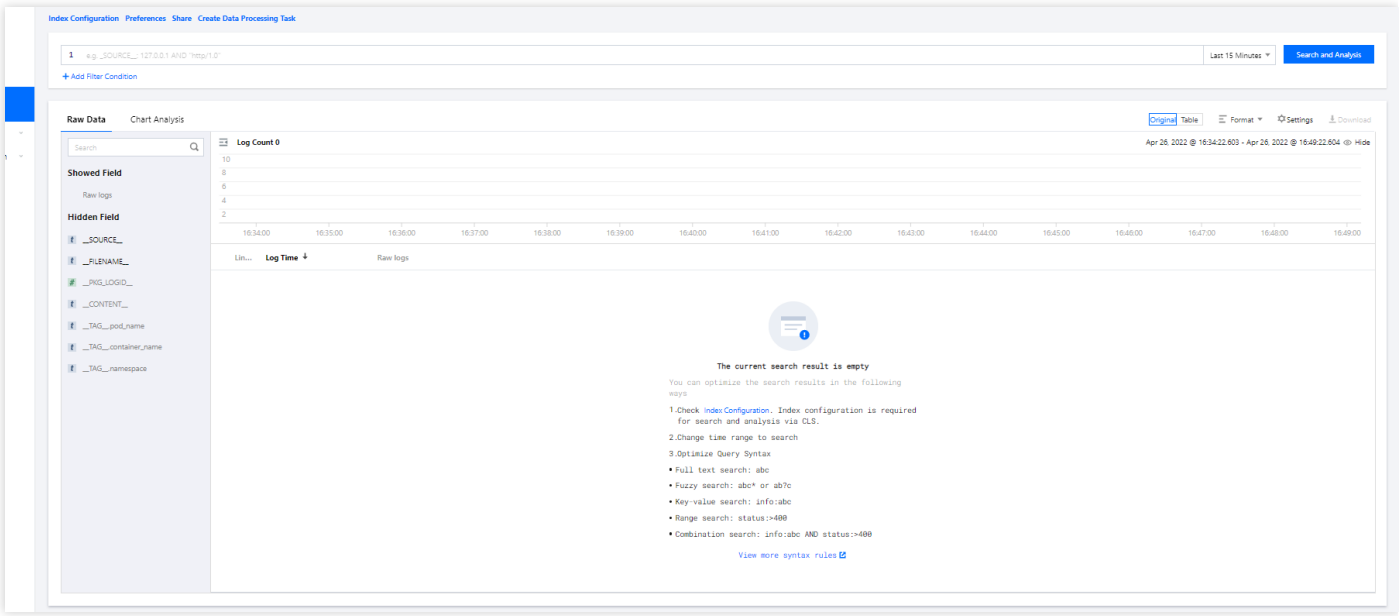


3. Click **Next** to enter the **Log Parsing Method**. Here, single-line text is used for QCBM. For more information on the log formats supported by CLS, see Full Text in a Single Line.

Viewing log

1. Log in to the CLS console and enter the **Search and Analysis** page.
2. On the **Search and Analysis** page, **Create Index** for the logs first and then click **Search and Analysis** to view the logs.

©2013-2022 Tencent Cloud. All rights reserved.



Note：

You can't find logs if no indexes are created.

Integrating TSW

TSW is currently in beta test and can be deployed in Guangzhou and Shanghai. Here, Shanghai is used as an example (QCBM is deployed in Nanjing).

Accessing TSW - getting access point information

1. Log in to the TSW console and click **Service Observation** > **Service List** on the left sidebar.
2. Click **Access Service** and select Java and the SkyWalking data collection method. The access method provides the **Access Point** and **Token** information.

Accessing TSW - application and container configuration

Enter the **Access Point** and **Token** of the TSW obtained in the previous step in `collector.backend_service` and `agent.authentication` respectively in the `agent.config` of SkyWalking. `agent.service_name` is the service name, and `agent.namespace` can be used to group

©2013-2022 Tencent Cloud. All rights reserved.

microservices under the same domain. `user-service` configuration is as shown below:

```
# The agent namespace
agent.namespace=${SW_AGENT_NAMESPACE:QCBM}

# The service name in UI
agent.service_name=${SW_AGENT_NAME:user-service}

# The number of sampled traces per 3 seconds
# Negative or zero means off, by default
agent.sample_n_per_3_secs=${SW_AGENT_SAMPLE:-1}

# Authentication active is based on backend setting, see application.yml for more details.
agent.authentication = ${SW_AGENT_AUTHENTICATION:[redacted]}

# Backend service addresses.
collector.backend_service=${SW_AGENT_COLLECTOR_BACKEND_SERVICES:ap-shanghai.tencentservicewatcher.com:11800}
```

You can also configure SkyWalking Agent by using environment variables. QCBM uses the ConfigMap and Secret to configure environment variables:

- Use the ConfigMap to configure `SW_AGENT_COLLECTOR_BACKEND_SERVICES` .
- Use the Secret to configure `SW_AGENT_AUTHENTICATION` .

As shown below:

```
---
# 创建 ConfigMap
apiVersion: v1
kind: ConfigMap
metadata: meta.v1.ObjectMeta
data:
  NACOS_HOST: 10.0.1.9
  MYSQL_HOST: 10.0.1.13
  REDIS_HOST: 10.0.1.16
  SW_AGENT_COLLECTOR_BACKEND_SERVICES: ap-shanghai.tencentservicewatcher.com:11800

---
# 创建 Secret
apiVersion: v1
kind: Secret
metadata: meta.v1.ObjectMeta
data:
  MYSQL_ACCOUNT: [redacted]
  MYSQL_PASSWORD: [redacted]
  REDIS_PASSWORD: [redacted]
  SW_AGENT_AUTHENTICATION: [redacted]
type: Opaque
```

©2013-2022 Tencent Cloud. All rights reserved.

At this point, you have completed TSW access. After starting the container service, you can view the call chain, service topology, and SQL analysis in the TSW console.

Using TSW

Viewing call exception through service API or call chain

1. Log in to the TSW console and click **Service Observation** > **API Observation** on the left sidebar.
2. On the **API Observation** page, you can view the call status of all APIs under a service, including the number of requests, success rate, error rate, response time, and other metrics.

Using TSW to analyze add-on (such as SQL and caching) call

1. Log in to the TSW console and click **Add-on Call Observation** > **SQL Call** on the left sidebar.
2. On the **SQL Call** page, you can view the call details of SQL, NoSQL, MQ, and other add-ons. For example, you can quickly locate frequent SQL requests and slow queries in your application with the number and durations of SQL requests.

Viewing service topology

1. Log in to the TSW console and click **Chain Tracing** > **Distributed Dependency Topology** on the left sidebar.
2. On the **Distributed Dependency Topology** page, you can view the completed service dependencies as well as information such as the number of calls and average latency.

©2013-2022 Tencent Cloud. All rights reserved.

Hosting SpringCloud to TKE

Last updated：2022-11-04 16:14:00

Overview

This document describes how to host a Spring Cloud application to TKE.

Hosting Spring Cloud applications to TKE has the following advantages:

- Improve the resource utilization.
- Kubernetes is a natural fit for microservice architectures.
- Improve the Ops efficiency and facilitate DevOps implementation.
- Highly scalable Kubernetes makes it easy to dynamically scale applications.
- TKE provides Kubernetes master management to ease Kubernetes cluster Ops and management.
- TKE is integrated with other cloud-native products of Tencent Cloud to help you better use Tencent Cloud products.

Best Practices

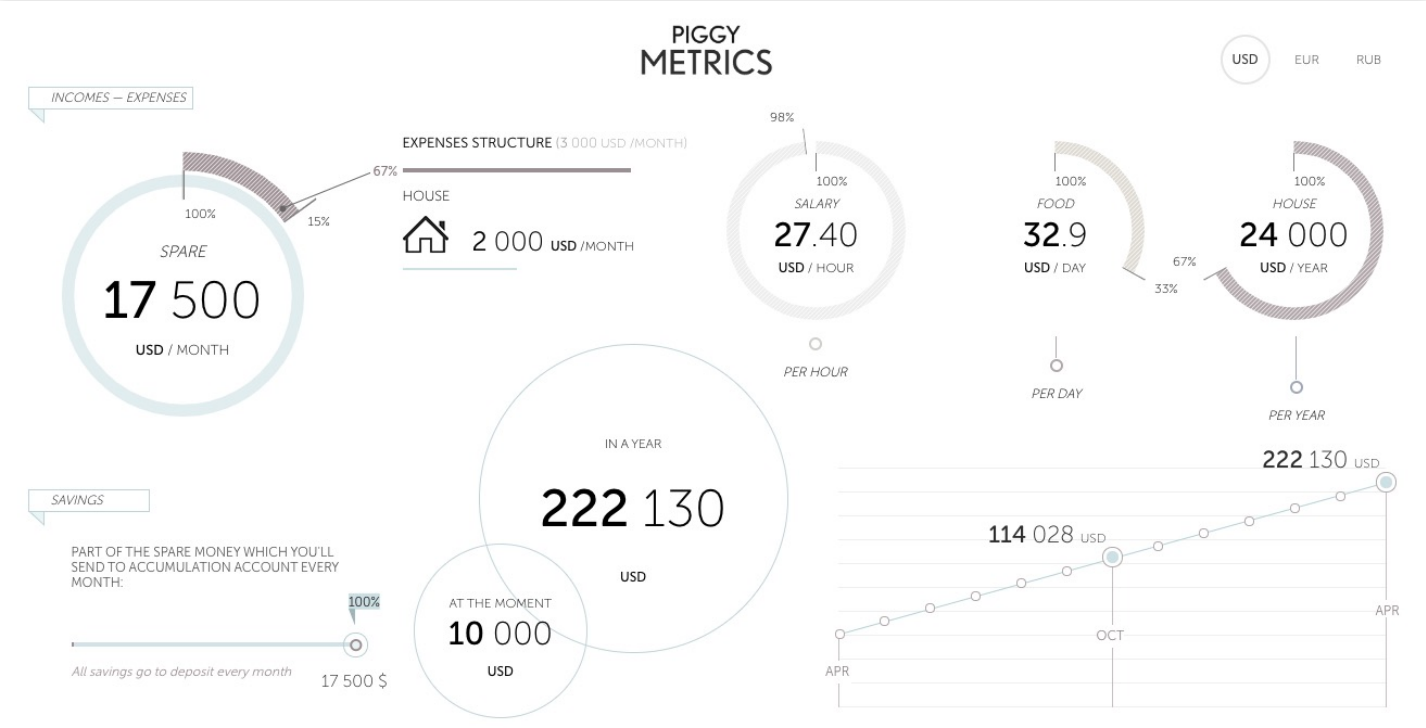
PiggyMetrics overview

This document describes how to host a Spring Cloud application to TKE by forking the open-source PiggyMetrics on GitHub and adapting it to Tencent Cloud products.

> Note：
> The modified PiggyMetrics deployment project is hosted on GitHub. After creating the basic service cluster, you can download the deployment project and deploy it in TKE.

©2013-2022 Tencent Cloud. All rights reserved.

The PiggyMetrics homepage is as shown below:



PiggyMetrics is a microservice-architecture application for personal finances developed by using the Spring Cloud framework.

PiggyMetrics consists of the following microservices:

Microservice	Description
API gateway	It's a Spring Cloud Zuul-based gateway and the aggregated portal for calling backend APIs, providing reverse routing and load balancing (Eureka + Ribbon) as well as rate limiting (Hystrix). Client single-page applications and the Zuul gateway are deployed together to simplify deployment.
Service registration and discovery	A Spring Cloud Eureka registry. Business services are registered through Eureka when they are enabled, and service discovery is performed through Eureka when services are called.
Authorization and authentication service	An authorization and authentication center based on Spring Security OAuth2. The client gets the access token through the Auth Service during logins, and so does service call. Each resource server verifies the token through the Auth Service.
Configuration service	A configuration center based on Spring Cloud Config to centrally manage configuration files for all Spring services.

©2013-2022 Tencent Cloud. All rights reserved.

Microservice	Description
Soft loading and rate limiting	Ribbon and Hystrix based on Spring Cloud. Zuul calls backend services through Ribbon for soft loading and Hystrix for rate limiting.
Metrics and dashboard	Hystrix Dashboard based on Spring Cloud Turbine, aggregating all the PiggyMetrics streams generated by Hystrix and displaying them on the Hystrix Dashboard.

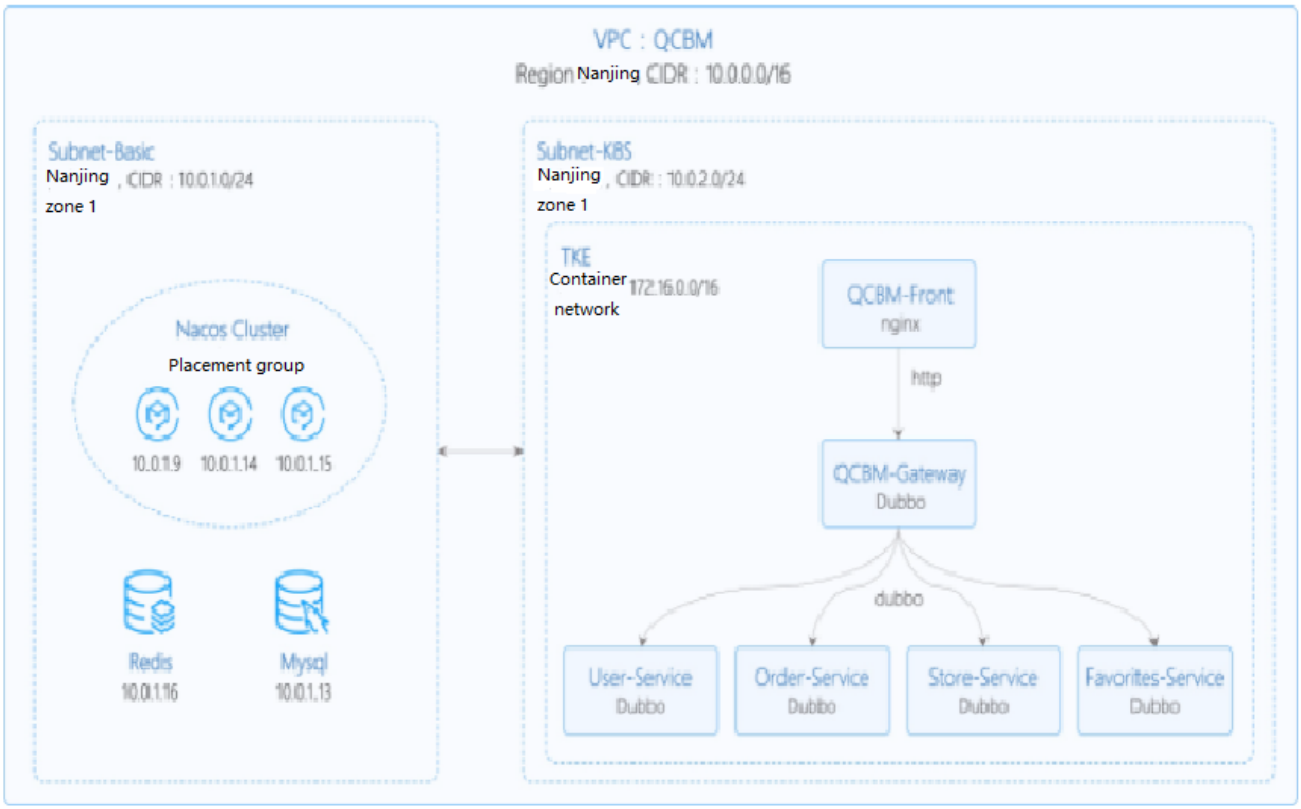
PiggyMetrics deployment architecture and add-ons

In the following best practice, applications deployed in CVM are containerized and hosted to TKE. In this use case, one VPC is used and divided into two subnets:

- **Subnet-Basic** is deployed with stateful basic services, including Dubbo's service registry Nacos, MySQL, and Redis.
- **Subnet-K8S** is deployed with PiggyMetrics application services, all of which are containerized and run in TKE.

©2013-2022 Tencent Cloud. All rights reserved.

The VPC is divided as shown below:



The network planning for the PiggyMetrics instance is as shown below:

Network Planning	Description
Region/AZ	Nanjing/Nanjing Zone 1
VPC	CIDR: 10.0.0.0/16
Subnet-Basic	Nanjing Zone 1, CIDR block: 10.0.1.0/24
Subnet-K8S	Nanjing Zone 1, CIDR block: 10.0.2.0/24
Nacos cluster	Nacos cluster built with three 1-core 2 GB MEM Standard SA2 CVM instances with IP addresses of 10.0.1.9, 10.0.1.14, and 10.0.1.15

©2013-2022 Tencent Cloud. All rights reserved.

The add-ons used in the PiggyMetrics instance are as shown below:

Add-on	Version	Source	Remarks
K8S	1.8.4	Tencent Cloud	TKE management mode
MongoDB	4.0	Tencent Cloud	TencentDB for MongoDB WiredTiger engine
CLS	N/A	Tencent Cloud	Log service
TSW	N/A	Tencent Cloud	Accessed with SkyWalking 8.4.0 Agent, which can be downloaded here
Java	1.8	Open-source community	Docker image of Java 8 JRE
Spring Cloud	Finchley.RELEASE	Open-source community	Spring Cloud website

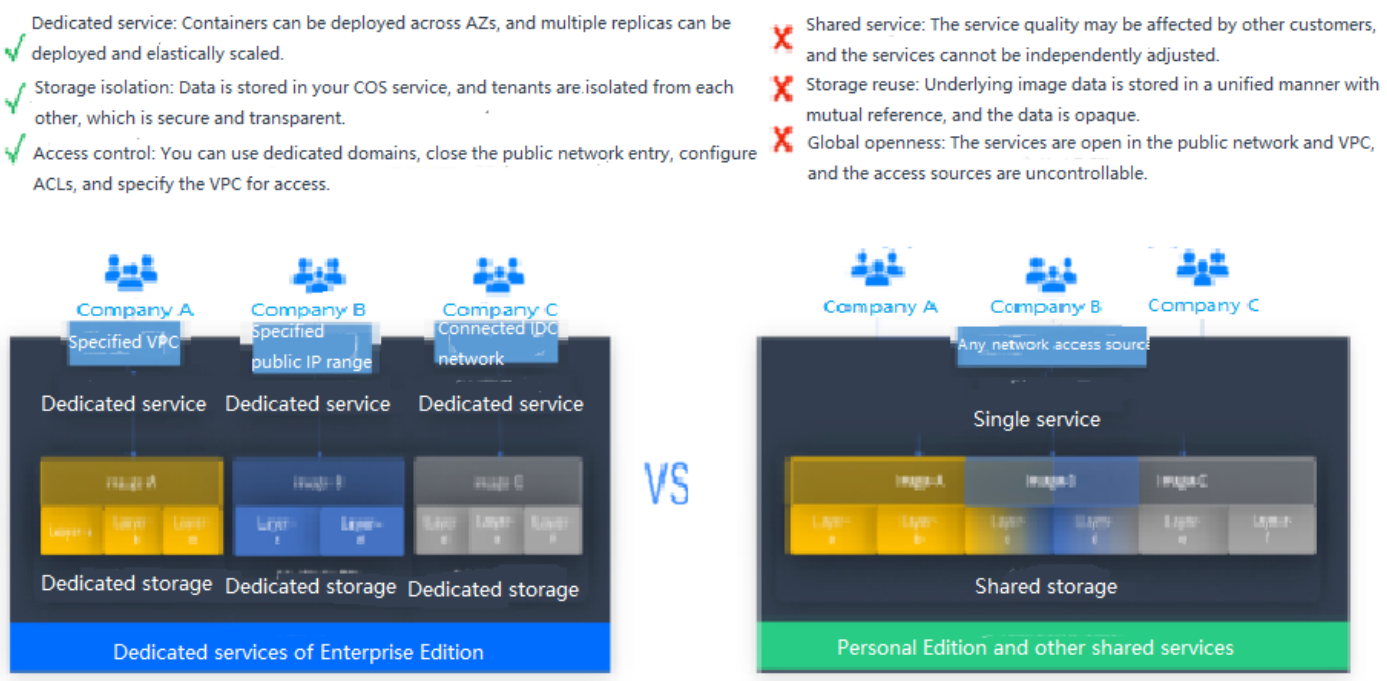
Overview

TCR

Tencent Cloud Tencent Container Registry (TCR) are available in Personal Edition and Enterprise Edition as differentiated below:

- TCR Personal Edition is only deployed in Guangzhou, while TCR Enterprise Edition is deployed in every region.
- TCR Personal Edition doesn't offer SLA guarantee.

©2013-2022 Tencent Cloud. All rights reserved.



PiggyMetrics is a Dubbo containerized demo project, so TCR Personal Edition perfectly meets its needs. However, for enterprise users, TCR Enterprise Edition is recommended. To use an image repository, see Basic Image Repository Operations.

TSW

Tencent Service Watcher (TSW) provides cloud-native service observability solutions that can trace upstream and downstream dependencies in distributed architectures, draw topologies, and provide multidimensional call

©2013-2022 Tencent Cloud. All rights reserved.

observation by service, API, instance, and middleware.

TSW is architecturally divided into four modules:

Show All

Data collection (client)

展开&收起

You can use an open-source probe or SDK to collect data. If you are migrating to the cloud, you can change the reporting address and authentication information only and keep most of the configurations on the client.

Data processing (server)

展开&收起

Data is reported to the server via the Pulsar message queue, converted by the adapter into an OpenTracing-compatible format, and assigned to real-time and offline computing as needed.

- Real-time computing provides real-time monitoring, statistical data display, and fast response to the connected alarming platform.
- Offline computing aggregates the statistical data in large amounts over long periods of time and leverages big data analytics to provide business value.

©2013-2022 Tencent Cloud. All rights reserved.

Storage

展开&收起

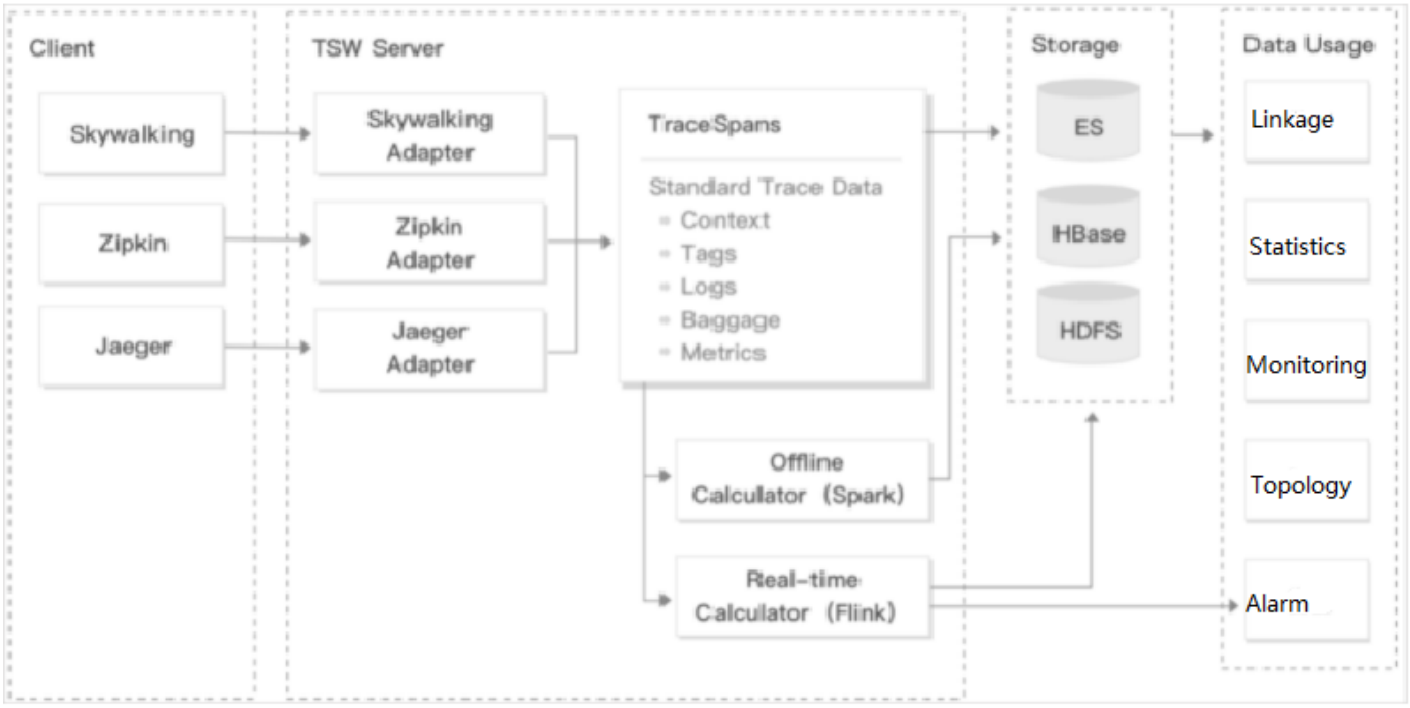
The storage layer can adapt to use cases with different data types, writing at the server layer, and query and reading requests at the data usage layer.

Data usage

展开&收起

The data usage layer provides underlying support for console operations, data display, and alarming.

The architecture is as shown below:



Directions

Creating basic service cluster

©2013-2022 Tencent Cloud. All rights reserved.

- In the TencentDB for MongoDB console, create an instance and run the following command to initialize it:

```
# Download the MongoDB client, decompress it, and enter the `bin` directory.

wget https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-3.6.18.tgz
tar -zxvf mongodb-linux-x86_64-3.6.18.tgz
cd mongodb-linux-x86_64-3.6.18/bin

# Run the following command to initialize MongoDB, where `mongouser` is the ad
min account created when the MongoDB instance is created.

./mongo -u mongouser -p --authenticationDatabase "admin" [mongodb IP]/piggymet
rics mongo-init.js
```

> A **guest** user of the `piggymetrics` library is created in the MongoDB initialization script **mongo-init.js** by default, which can be modified as needed.

- In the CLB console, create a private network CLB instance for `Subnet-K8S` (the ID of this CLB instance will be used later).

- TSW is currently in beta test and supports both Java and Go.

Building Docker image

Writing Dockerfile

The following uses `account-service` as an example to describe how to write a Dockerfile. The project directory structure of `account-service` is displayed, **Dockerfile** is in the root directory of the project, and **account-service.jar** is the packaged file that needs to be added to the image.

```
➜ account-service tree
├── Dockerfile
├── skywalking
│ ├── account.config
│ └── skywalking-agent.zip
├── pom.xml
├── src
│ ....
├── target
│ .....
```

©2013-2022 Tencent Cloud. All rights reserved.

```
| └── account-service.jar
└── account-service.iml
```

Note：
Here, SkyWalking Agent is used as the TSW access client that reports call chain information to the TSW backend. For more information on how to download SkyWalking Agent, see PiggyMetrics deployment architecture and add-ons.

The Dockerfile of `account-service` is as shown below:

```
FROM java:8-jre

# Working directory in the container

/appWORKDIR /app

# Add the locally packaged application to the image.

ADD ./target/account-service.jar

# Copy SkyWalking Agent to the image.

COPY ./skywalking/skywalking-agent.zip

# Decompress SkyWalking Agent and delete the original compressed file.

RUN unzip skywalking-agent.zip && rm -f skywalking-agent.zip

# Add the SkyWalking configuration file.

COPY ./skywalking/account.config ./skywalking-agent/config/agent.config

# Start the application.

CMD ["java", "-Xmx256m", "-javaagent:/app/skywalking-agent/skywalking-agent.jar"
, "-jar", "/app/account-service.jar"]

# Port description of the application

EXPOSE 6000
```

©2013-2022 Tencent Cloud. All rights reserved.

Note：
As each Run command in the Dockerfile will generate an image layer, we recommend you combine these commands into one.

Image build

TCR provides both automatic and manual methods to build an image. To demonstrate the build process, the manual method is used.

The image name needs to be in line with the convention of `ccr.ccs.tencentyun.com/[namespace]/[ImageName]:[image tag]`:

- Here, `namespace` can be the project name to facilitate image management and use. In this document, `piggymetrics` represents all the images under the PiggyMetrics project.
- `ImageName` can contain the `subpath`, generally used for multi-project use cases of enterprise users. In addition, if a local image is already built, you can run the `docker tag` command to rename the image in line with the naming convention.

1. Run the following command to build an image as shown below:

```
# Recommended build method, which eliminates the need for secondary tagging operations

sudo docker build -t ccr.ccs.tencentyun.com/[namespace]/[ImageName]:[image tag]

# Build a local `account-service` image. The last `.` indicates that the Dockerfile is stored in the current directory (`account-service`).

➜ account-service docker build -t ccr.ccs.tencentyun.com/piggymetrics/account-service:1.0.0 .

# Rename existing images in line with the naming convention

sudo docker tag [ImageId] ccr.ccs.tencentyun.com/[namespace]/[ImageName]:[image tag]
```

2. After the build is complete, you can run the following command to view all the images in your local repository.

```
docker images | grep piggymetrics
```

©2013-2022 Tencent Cloud. All rights reserved.

A sample is as shown below:

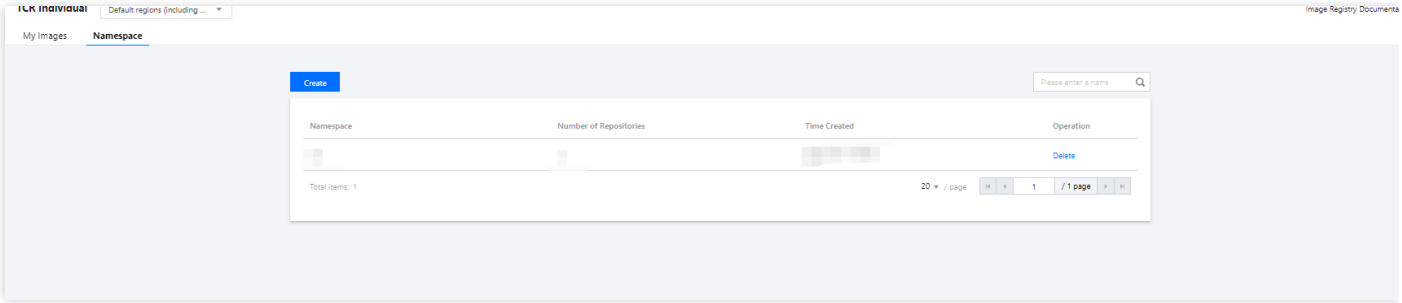
```
→  account-service docker images | grep "piggymetrics"
ccr.ccs.tencentyun.com/piggymetrics/account-service          1.0.2          134cf538f5f7        10 minutes ago      405MB
ccr.ccs.tencentyun.com/piggymetrics/turbine-stream-service   1.0.1          9faaae7517d4        24 hours ago        356MB
ccr.ccs.tencentyun.com/piggymetrics/gateway                  1.0.3          0351544fb1c9        3 days ago          393MB
ccr.ccs.tencentyun.com/piggymetrics/config-server            1.0.5          cbb1216e4d04        3 days ago          340MB
ccr.ccs.tencentyun.com/piggymetrics/notification-service     1.0.1          5f34870d1d7c        3 days ago          404MB
ccr.ccs.tencentyun.com/piggymetrics/statistics-service       1.0.1          034f5239967a        4 days ago          404MB
ccr.ccs.tencentyun.com/piggymetrics/auth-service             1.0.1          b3aadfa22c0d        4 days ago          398MB
ccr.ccs.tencentyun.com/piggymetrics/monitoring               1.0.0          2ed5e7c9e133        4 days ago          343MB
ccr.ccs.tencentyun.com/piggymetrics/registry                 1.0.0          e946d0ed8c34        4 days ago          356MB
```

Uploading image to TCR

Creating namespace

The PiggyMetrics project uses TCR Personal Edition (TCR Enterprise Edition is recommended for enterprise users).

1. Log in to the TKE console.
2. Click **TCR** > **Personal** > **Namespace** to enter the **Namespace** page.
3. Click **Create** and create the `piggymetrics` namespace in the pop-up window. All the images of the PiggyMetrics project are stored under this namespace as shown below:



Uploading image

Log in to TCR and upload an image.

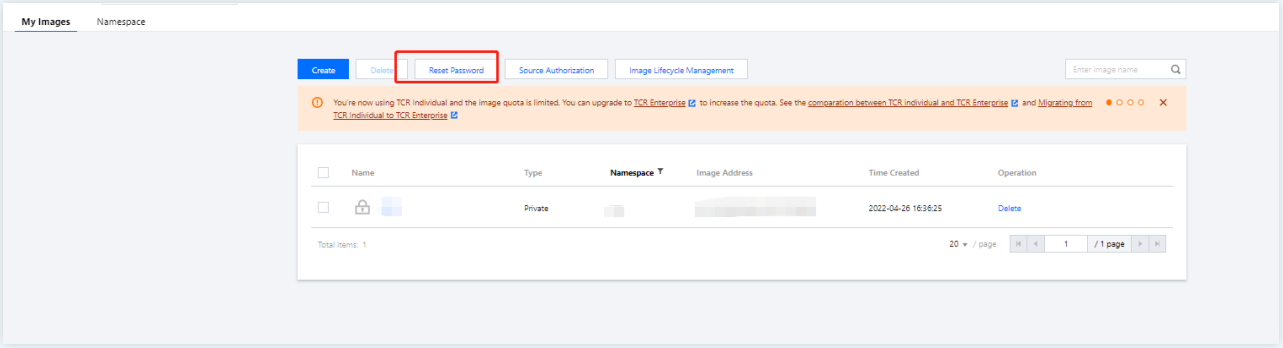
1. Run the following command to log in to TCR.

```
docker login --username=[Tencent Cloud account ID] ccr.ccs.tencentyun.com
```

- You can get your Tencent Cloud account ID on the Account Info page.

©2013-2022 Tencent Cloud. All rights reserved.

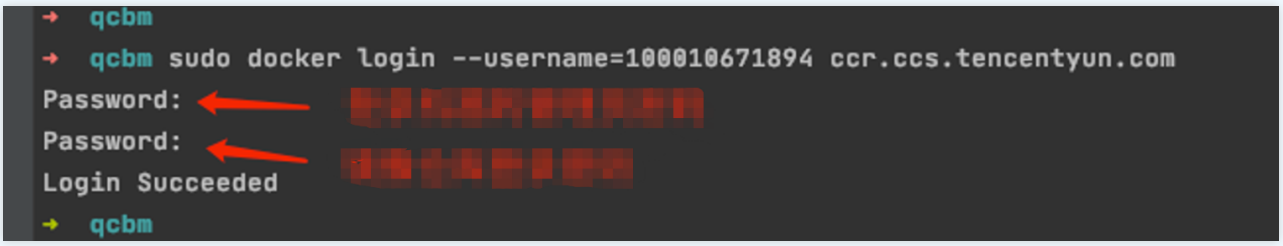
- If you forget your **TCR login password**, you can reset it in My Images of TCR Personal Edition.



- If you are prompted that you have no permission to run the command, add `sudo` before the command and run it as shown below. In this case, you need to enter two passwords, the server admin password required for `sudo` and the **TCR login password**.

```
sudo docker login --username=[Tencent Cloud account ID] ccr.ccs.tencentyun.com
```

As shown below:



2. Run the following command to push the locally generated image to TCR.

```
docker push ccr.ccs.tencentyun.com/[namespace]/[ImageName]:[image tag]
```

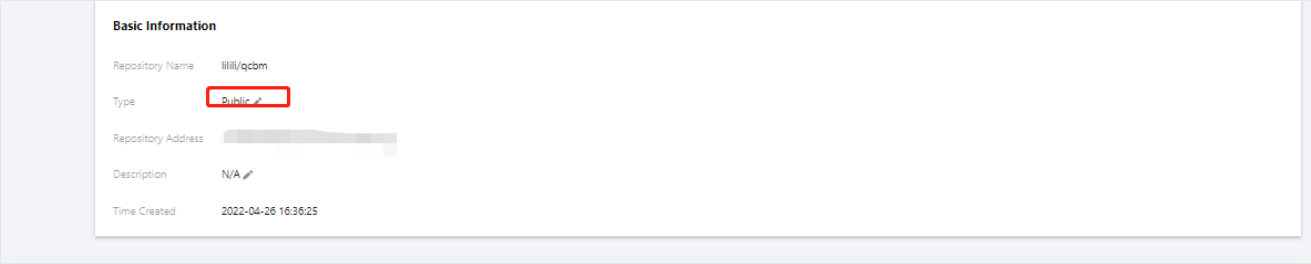
©2013-2022 Tencent Cloud. All rights reserved.

As shown below:

```
→  user-service docker push ccr.ccs.tencentyun.com/qcbm/user-service:1.0.1
The push refers to repository [ccr.ccs.tencentyun.com/qcbm/user-service]
bebcf5e72f77: Pushed
958f8e83f873: Pushed
a177e9d322e4: Pushed
73ad47d4bc12: Layer already exists
c22c27816361: Layer already exists
04dba64afa87: Layer already exists
500ca2ff7d52: Layer already exists
782d5215f910: Layer already exists
0eb22bfb707d: Layer already exists
a2ae92ffcd29: Layer already exists
1.0.1: digest: sha256:4af3e7ed8203a1bc92baf108ac8f65b8b00de750367e680dde4c1673bf90dd29 size: 2418
```

3. In My Images, you can view all the uploaded images.

The default image type is `Private`. If you want to let others use the image, you can set it to `Public` in **Image Info** as shown below:



Deploying service in TKE

Creating K8s cluster PiggyMetrics

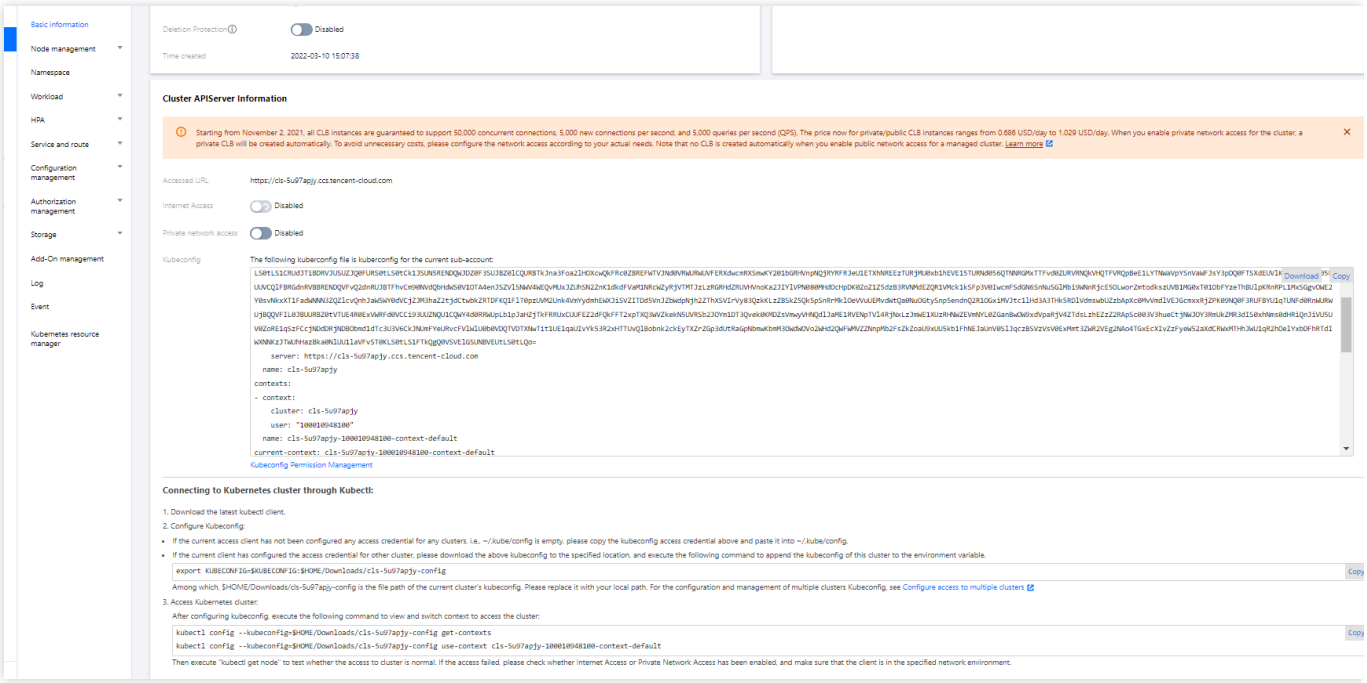
1. Before the deployment, you need to create a K8s cluster as instructed in Quickly Creating a Standard Cluster.

Note：
When a cluster is created, we recommend you enable **Placement Group** on the **Select Model** page. It helps distribute CVM instances across different hosts to increase the system reliability.

2. After the cluster is created, you can view its information on the Cluster Management page in the TKE console. Here, the new cluster is named `piggyMetrics`.

©2013-2022 Tencent Cloud. All rights reserved.

3. Click the `PiggyMetrics-k8s-demo` cluster to enter the **Basic Info** page to view the cluster configuration information.
4. (Optional) If you want to use K8s management tools such as kubectl and Lens, you need to follow two steps:
   i. Enable public network access.
   ii. Store the API authentication token in the local `config` file under `user home/.kube` (choose another if the `config` file has content) to ensure that the default cluster can be accessed each time. If you choose not to store the token in the `config` file under `.kube`, see the **Instructions on Connecting to Kubernetes Cluster via kubectl** under **Cluster API Server Info** in the console as shown below:



Creating namespace

A namespace is a logical environment in a Kubernetes cluster, allowing you to divide teams or projects. You can create a namespace in the following three methods, and method 1 is recommended.

- Method 1. Use the command line
- Method 2. Use the console
- Method 3. Use YAML

Run the following command to create a namespace:

```
kubectl create namespace piggymetrics
```

Using ConfigMap to store configuration information

©2013-2022 Tencent Cloud. All rights reserved.

ConfigMap allows you to decouple the configuration from the running image, making the application more portable. The PiggyMetrics backend service needs to get the MongoDB host and port information from the environment variables and store them by using the ConfigMap.
You can use ConfigMap to store configuration information in the following two methods:

- Method 1. Use YAML
- Method 2. Use the console

The following is the ConfigMap YAML for PiggyMetrics, where **values of pure digits require double quotation marks**.

```
# Create a ConfigMap.
apiVersion: v1
kind: ConfigMap
metadata:
name: piggymetrics-env
namespace: piggymetrics
data:
# MongoDB IP address
MONGODB_HOST: 10.0.1.13
# TSW access address as described below
SW_AGENT_COLLECTOR_BACKEND_SERVICES: ap-shanghai.tencentservicewatcher.com:11800
```

Using Secret to store sensitive information

A Secret can be used to store sensitive information such as passwords, tokens, and keys to reduce exposure risks. PiggyMetrics uses it to store account and password information.
You can use a Secret to store sensitive information in the following two methods:

- Method 1. Use YAML
- Method 2. Use the console

The following is the YAML for creating a Secret in PiggyMetrics, where the `value` of the Secret needs to be a Base64-encoded string.

```
# Create a Secret.
apiVersion: v1
kind: Secret
metadata:
name: piggymetrics-keys
namespace: piggymetrics
labels:
qcloud-app: piggymetrics-keys
```

©2013-2022 Tencent Cloud. All rights reserved.

```
data:
# Replace XXX below with the actual value.
MONGODB_USER: XXX
MONGODB_PASSWORD: XXX
SW_AGENT_AUTHENTICATION: XXX
type: Opaque
```

Deploying stateful service with StatefulSet

A StatefulSet is used to manage stateful applications. A Pod created accordingly has a persistent identifier in line with the specifications, which will be retained after the Pod is migrated, terminated, or restarted. When using persistent storage, you can map storage volumes to identifiers.
The basic add-ons and services under the PiggyMetrics project such as configuration services, registry, and RabbitMQ have their own data stored and are therefore suitable for deployment through StatefulSet.

Below is a sample deployment YAML for `config-server` :

```
---
kind: Service
apiVersion: v1
metadata:
name: config-server
namespace: piggymetrics
spec:
clusterIP: None
ports:
- name: http
port: 8888
targetPort: 8888
protocol: TCP
selector:
app: config
version: v1

---
apiVersion: apps/v1
kind: StatefulSet
metadata:
name: config
namespace: piggymetrics
labels:
app: config
version: v1
spec:
serviceName: "config-server"
```

©2013-2022 Tencent Cloud. All rights reserved.

```
replicas: 1
selector:
matchLabels:
app: config
version: v1
template:
metadata:
labels:
app: config
version: v1
spec:
terminationGracePeriodSeconds: 10
containers:
- name: config
image: ccr.ccs.tencentyun.com/piggymetrics/config-server:2.0.03
ports:
- containerPort: 8888
protocol: TCP
```

Deploying Deployment

A Deployment declares the Pod template and controls the Pod running policy, which is suitable for deploying stateless applications. PiggyMetrics backend services such as Account are stateless and can use the Deployment.

YAML parameters for the `account-service` Deployment are as follows:

Parameter	Description
replicas	Indicates the number of Pods to be created.
image	Image address
imagePullSecrets	The key to pull an image, which can be obtained from **Cluster** > **Configuration Management** > **Secret**. It is not required for public images.
env	• Defines Pod environment variables and values. • The `key-value` defined in the ConfigMap can be referenced by using `configMapKeyRef` . • The `key-value` defined in the Secret can be referenced by using `secretKeyRef` .
ports	Specifies the port number of the container. It is `6000` for `account-service` .

Below is a complete sample YAML file for the `account-service` Deployment:

©2013-2022 Tencent Cloud. All rights reserved.

```
# account-service Deployment
apiVersion: apps/v1
kind: Deployment
metadata:
name: account-service
namespace: piggymetrics
labels:
app: account-service
version: v1
spec:
replicas: 1
selector:
matchLabels:
app: account-service
version: v1
template:
metadata:
labels:
app: account-service
version: v1
spec:
containers:
- name: account-service
image: ccr.ccs.tencentyun.com/piggymetrics/account-service:1.0.1
env:
# MongoDB IP address
- name: MONGODB_HOST
valueFrom:
configMapKeyRef:
key: MONGODB_HOST
name: piggymetrics-env
optional: false
# MongoDB username
- name: MONGODB_USER
valueFrom:
secretKeyRef:
key: MONGODB_USER
name: piggymetrics-keys
optional: false
# MongoDB password
- name: MONGODB_PASSWORD
valueFrom:
secretKeyRef:
key: MONGODB_PASSWORD
name: piggymetrics-keys
optional: false
```

©2013-2022 Tencent Cloud. All rights reserved.

```
# TSW access point
- name: SW_AGENT_COLLECTOR_BACKEND_SERVICES
valueFrom:
configMapKeyRef:
key: SW_AGENT_COLLECTOR_BACKEND_SERVICES
name: piggymetrics-env
optional: false
# TSW access token
- name: SW_AGENT_AUTHENTICATION
valueFrom:
secretKeyRef:
key: SW_AGENT_AUTHENTICATION
name: piggymetrics-keys
optional: false
ports:
# Container port
- containerPort: 6000
protocol: TCP
imagePullSecrets: # Token to pull the image
- name: qcloudregistrykey
```

Deploying Service

You can specify the Service type with Kubernetes `ServiceType`, which defaults to `ClusterIP`. Valid values of `ServiceType` include the following:

- LoadBalancer: Provides public network, VPC, and private network access.
- NodePort: : Accesses services through the CVM IP and host port.
- ClusterIP: Accesses services through the service name and port.

The frontend pages and the gateway of PiggyMetrics are packaged together and need to provide services, so `ServiceType` is set to `LoadBalancer`. TKE enriches the `LoadBalancer` mode by configuring the Service through annotations.

If you use the `service.kubernetes.io/qcloud-loadbalancer-internal-subnetid` annotations, a private network CLB instance will be created when the Service is deployed. In general, we recommend you create the CLB instance in advance and use the `service.kubernetes.io/loadbalance-id` annotations in the deployment YAML to improve the efficiency.

Below is the deployment YAML for `gateway service`:

```
# Deploy `gateway service`.
apiVersion: v1
kind: Service
```

©2013-2022 Tencent Cloud. All rights reserved.

```
metadata:
name: gateway
namespace: piggymetrics
annotations:
# Replace it with the ID of the CLB instance of `Subnet-K8S`.
service.kubernetes.io/loadbalance-id: lb-hfyt76co
spec:
externalTrafficPolicy: Cluster
ports:
- name: http
port: 80
targetPort: 4000
protocol: TCP
selector: # Map the backend `gateway` to the Service.
app: gateway
version: v1
type: LoadBalancer
```

Viewing deployment result

At this point, you have completed the deployment of PiggyMetrics in TKE and can view the deployment result in the following steps:

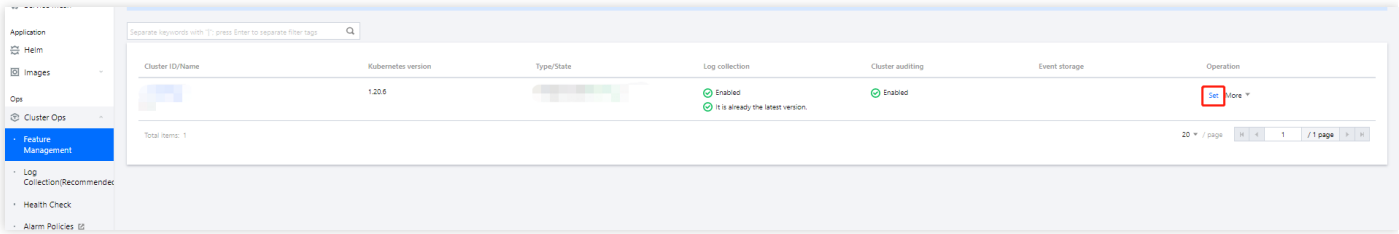
1. Log in to the TKE console and click the **Cluster ID/Name** to enter the cluster details page.
2. Click **Services and Routes** > **Service** to enter the **Service** page, where you can see the created Service. You can access the PiggyMetrics page through the `gateway service` VIP.

Integrating CLS

Enabling container log collection

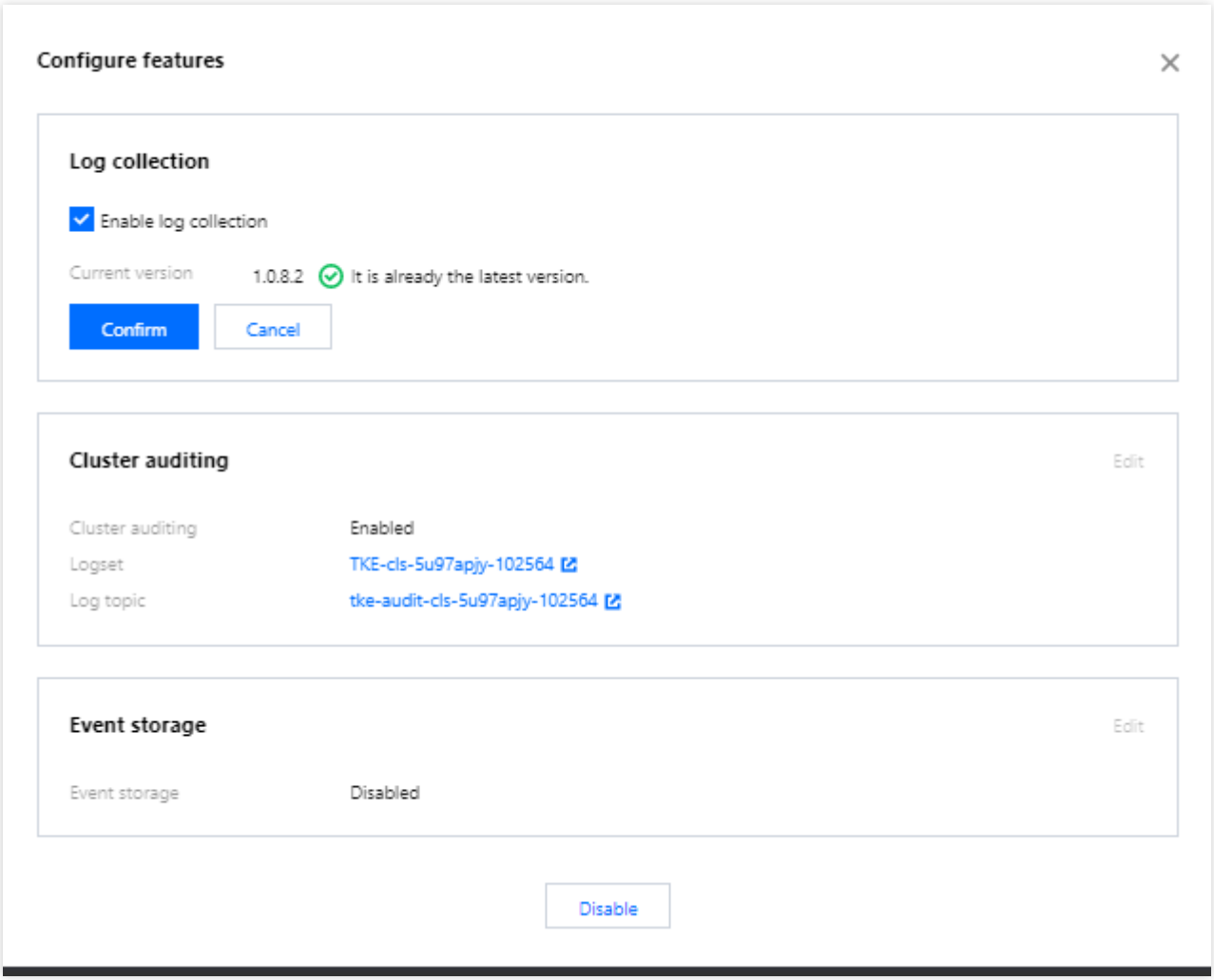
The container log collection feature is disabled by default and needs to be enabled as instructed below:

1. Log in to the TKE console and click **Cluster Ops** > **Feature Management** on the left sidebar.
2. At the top of the **Feature Management** page, select the region. On the right of the target cluster, click **Set**.



3. On the **Configure Features** page, click **Edit** for log collection, enable log collection, and confirm this operation as shown below:

©2013-2022 Tencent Cloud. All rights reserved.



Creating log topic and logset

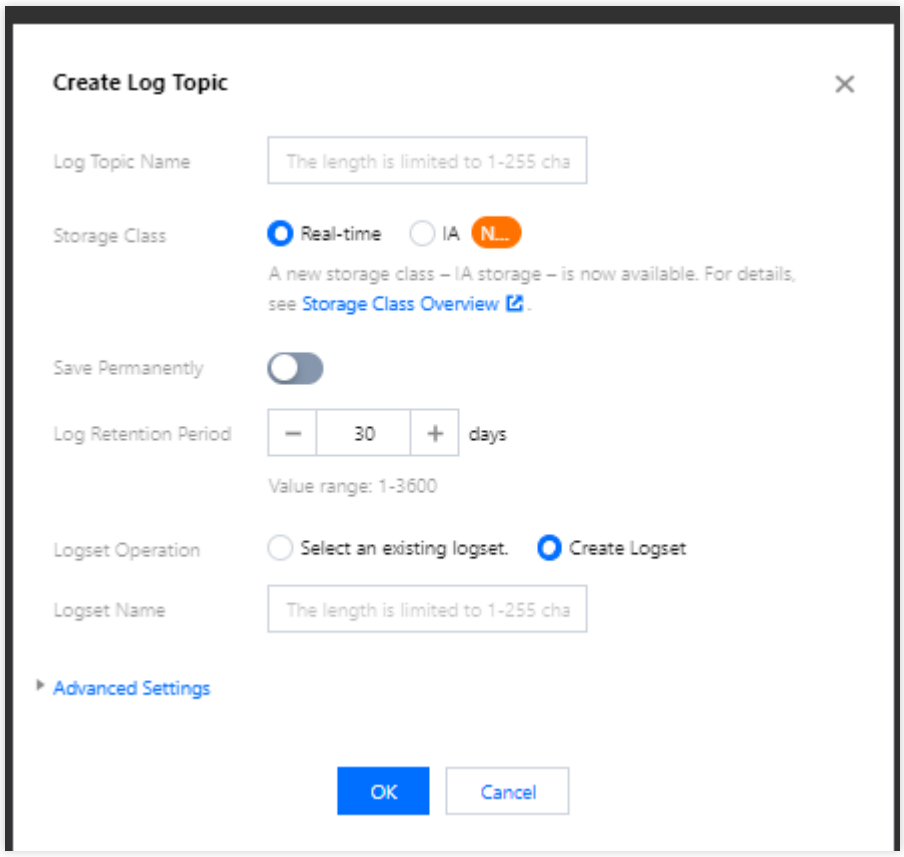
CLS is region-specific. To reduce the network latency, we recommend you select a region closest to your business when creating log resources, which are mainly logsets and log topics. A logset represents a project, a log topic represents a class of services, and a single logset can contain multiple log topics.

PiggyMetrics is deployed in Nanjing region, so you need to select Nanjing region on the **Log Topic** page when creating logsets:

1. Log in to the CLS console and select Nanjing region on the **Log Topic** page.

©2013-2022 Tencent Cloud. All rights reserved.

2. Click **Create Log Topic** and enter the relevant information in the pop-up window as prompted as shown below:



- **Log Topic Name**: Enter `piggymetrics`.
- **Logset Operation**: Select **Create Logset**.
- **Logset Name**: Enter `piggymetrics-logs`.

3. Click **OK**.

> Note：
> As PiggyMetrics has multiple backend microservices, you can create a log topic for each microservice to facilitate log categorization.
>
> - A log topic is created for each PiggyMetrics service.
> - You need the log topic ID when creating log rules for containers.

Configuring log collection rule

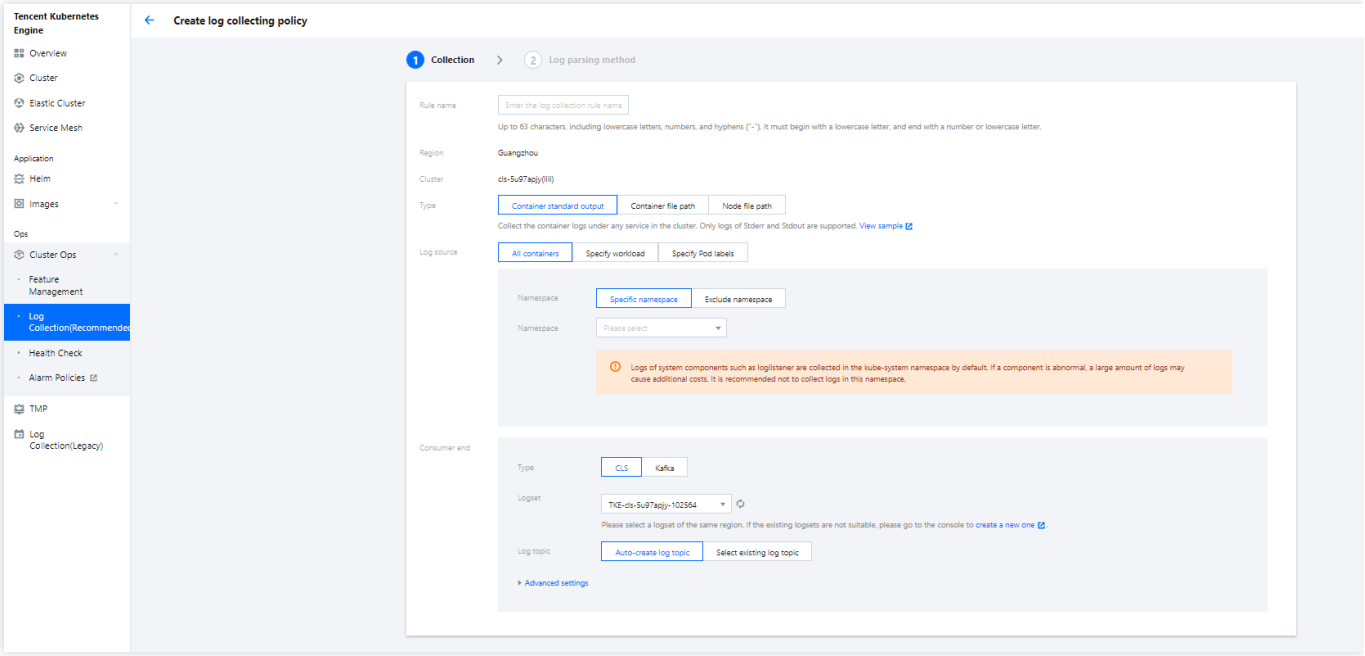
You can configure container log collection rules in the console or with CRD.

- Method 1. Use the console
- Method 2. Use CRD

©2013-2022 Tencent Cloud. All rights reserved.

Log rules specify the location of a log in a container:

1. Log in to the TKE console and click **Cluster Ops** > **Log Rules** on the left sidebar.
2. On the **Log Rules** page, click **Create** to create a rule.
   - **Log Source**: Specify the location of a log in a container. PiggyMetrics uses the default Spring Cloud configuration where all logs are printed to the standard output. Therefore, you can use the standard container output and specify a Pod Label.
   - **Consumer**: Select the previously created logset and topic.

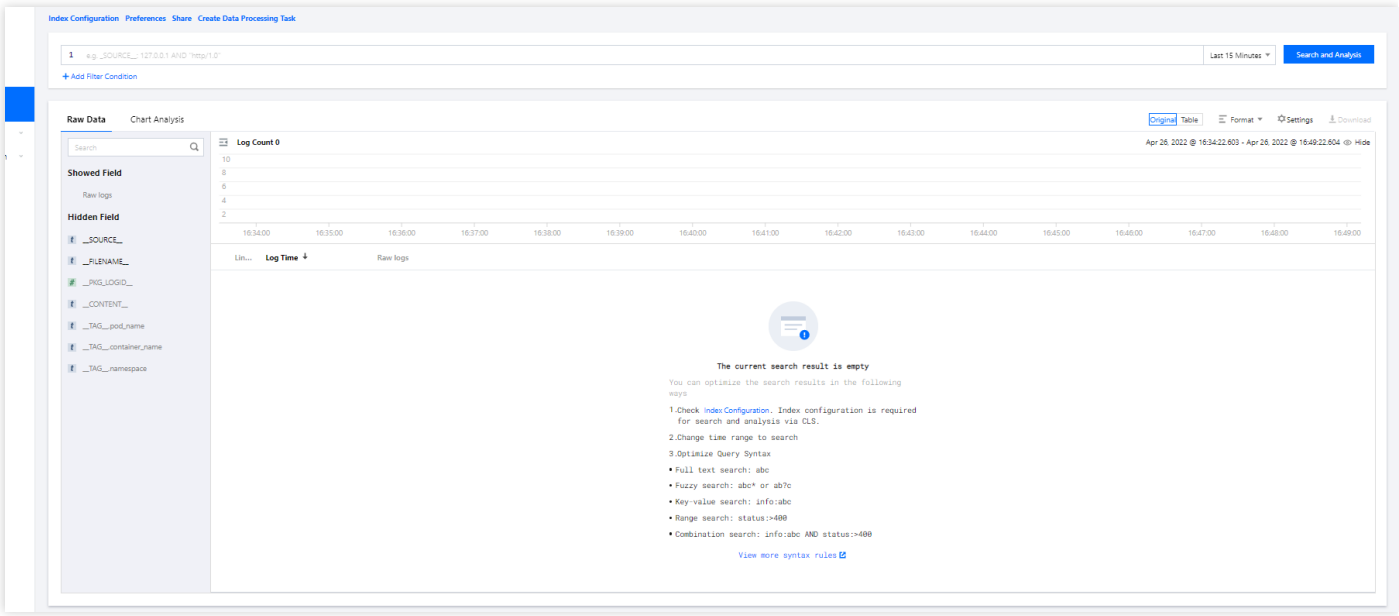


3. Click **Next** to enter the **Log Parsing Method**. Here, single-line text is used for PiggyMetrics. For more information on the log formats supported by CLS, see Full Text in a Single Line.

Viewing log

1. Log in to the CLS console and enter the **Search and Analysis** page.
2. On the **Search and Analysis** page, **Create Index** for the logs first and then click **Search and Analysis** to view the logs.

©2013-2022 Tencent Cloud. All rights reserved.



Note：

You can't find logs if no indexes are created.

Integrating TSW

TSW is currently in beta test and can be deployed in Guangzhou and Shanghai. Here, deployment in Shanghai is used as an example (PiggyMetrics is deployed in Nanjing).

Accessing TSW - getting access point information

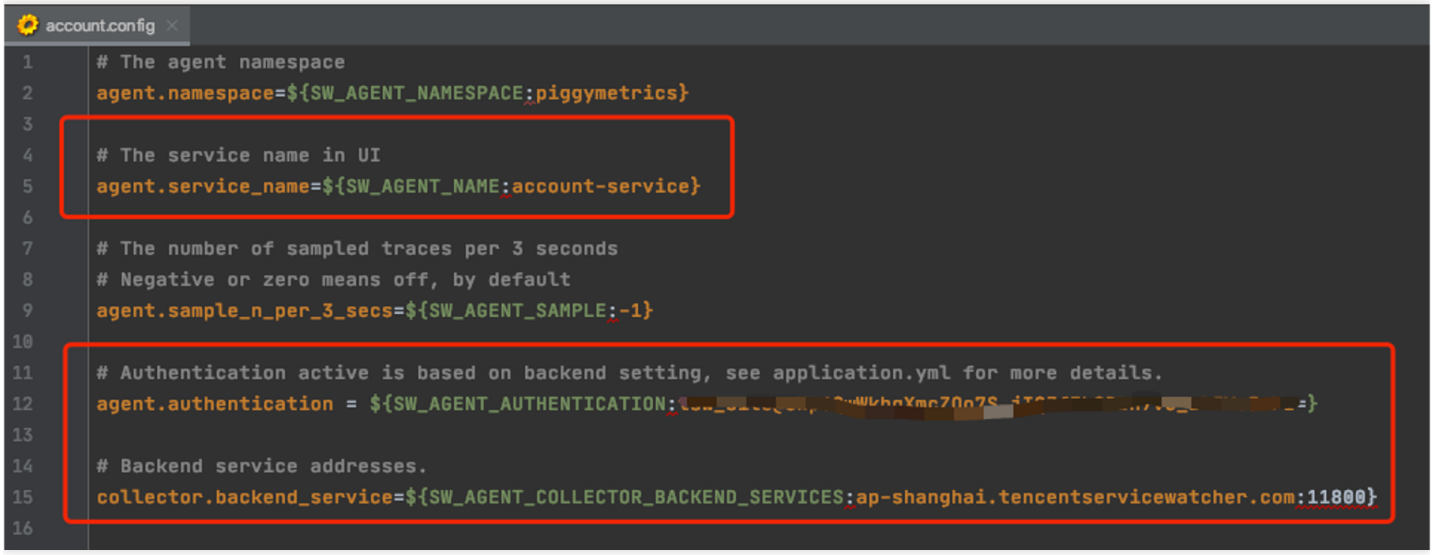
1. Log in to the TSW console and click **Service Observation** > **Service List** on the left sidebar.
2. Click **Access Service** and select Java and the SkyWalking data collection method. The access method provides the **Access Point** and **Token** information.

Accessing TSW - application and container configuration

Enter the **Access Point** and **Token** of the TSW obtained in the previous step in `collector.backend_service` and `agent.authentication` respectively in the `agent.config` of SkyWalking. `agent.service_name` is the service name, and `agent.namespace` can be used to group

©2013-2022 Tencent Cloud. All rights reserved.

microservices under the same domain. `account-service` configuration is as shown below:

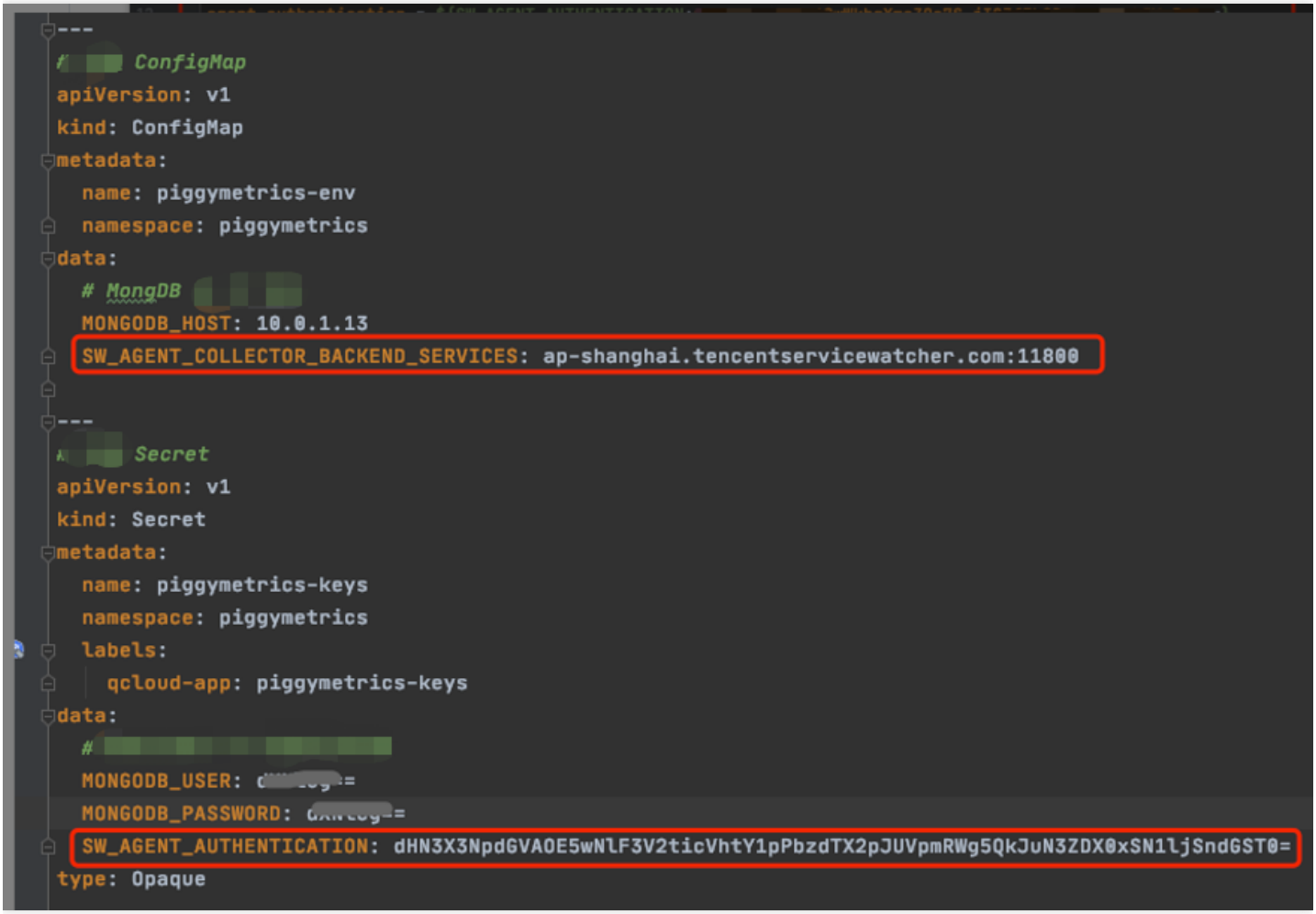


You can also configure SkyWalking Agent by using environment variables. PiggyMetrics uses the ConfigMap and Secret to configure environment variables:

- Use the ConfigMap to configure `SW_AGENT_COLLECTOR_BACKEND_SERVICES` .
- Use the Secret to configure `SW_AGENT_AUTHENTICATION` .

©2013-2022 Tencent Cloud. All rights reserved.

As shown below:



At this point, you have completed TSW access. After starting the container service, you can view the call chain, service topology, and SQL analysis in the TSW console.

Using TSW

Viewing call exception through service API or call chain

1. Log in to the TSW console and click **Service Observation** > **API Observation** on the left sidebar.
2. On the **API Observation** page, you can view the call status of all APIs under a service, including the number of requests, success rate, error rate, response time, and other metrics.
   The figure shows that the gateway and `account-service` responded too slowly and all `statistic-service` requests failed in the past hour.
3. Click the service name `statistics-service` to enter the information page. Click **API Observation**, and you can see that the API `{PUT}/{accountName}` throws a `NestedServletException` exception, which makes the API unavailable.
4. Click the **Trace ID** to view the call chain details.

©2013-2022 Tencent Cloud. All rights reserved.

Viewing service topology

1. Log in to the TSW console and click **Chain Tracing** > **Distributed Dependency Topology** on the left sidebar.
2. On the **Distributed Dependency Topology** page, you can view the completed service dependencies as well as information such as the number of calls and average latency.

©2013-2022 Tencent Cloud. All rights reserved.

Cost Management
Tools for Resource Utilization Improvement

Last updated：2023-02-09 10:29:13

Background

Public clouds are leased instead of purchased services with complete technical support and assurance, greatly contributing to business stability, scalability, and convenience. But more work needs to be done to reduce costs and improve efficiency, for example, adapting to application development, architecture design, management and Ops, and reasonable use in the cloud. According to the Kubernetes Standards for Cost Reduction and Efficiency Enhancement | Analysis of Containerized Computing Resource Utilization Rate, resource utilization is improved after IDC cloud migration, but not that much; the average utilization of containerized resources is only 13%, indicating a long and uphill way towards improvement.

This article details:

1. The reason for low **CPU and memory utilization** in Kubernetes clusters
2. TKE productized methods for easily improving resource utilization

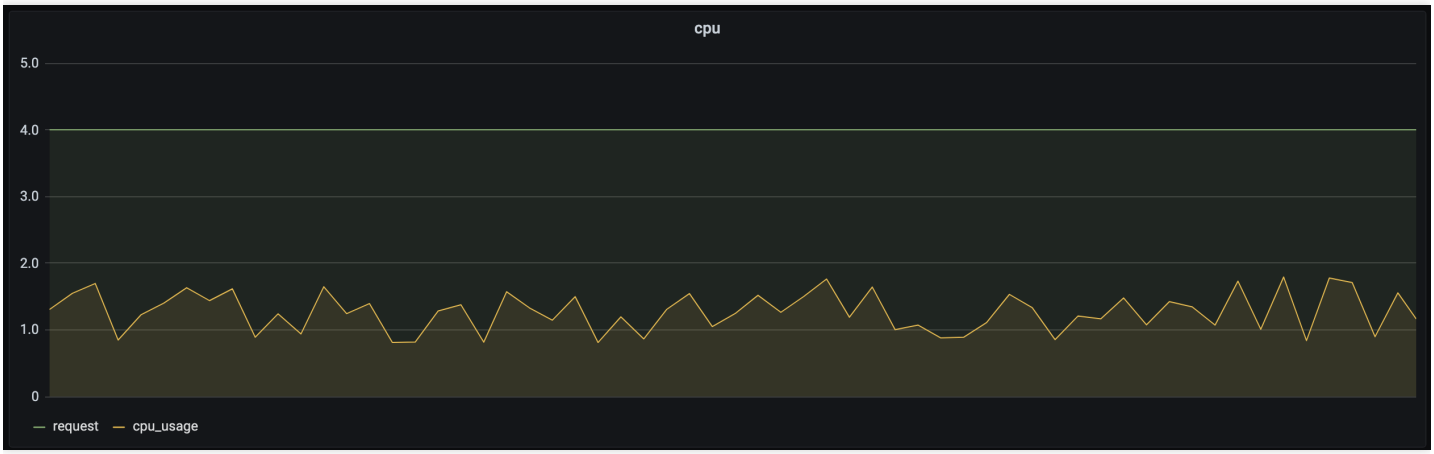
Resource Waste Scenarios

To figure out why utilization is low, let's look at a few cases of resource use:

Scenarios 1: Over 50% of reserved resources are wasted

The `Request` field in Kubernetes manages the CPU and memory reservation mechanism, which reserves certain resources in one container from being used by another. For more information, see Resource Management for Pods and Containers. If `Request` is set to a small value, resources may fail to accommodate the business, especially when the load becomes high. Therefore, users tend to set `Request` to a very high value to ensure the service reliability. However, the business load is not that high most of the time. Taking CPU as an example, the following figure shows the relationship between the resource reservation (request) and actual usage (cpu_usage) of a container in a real-world business scenario:

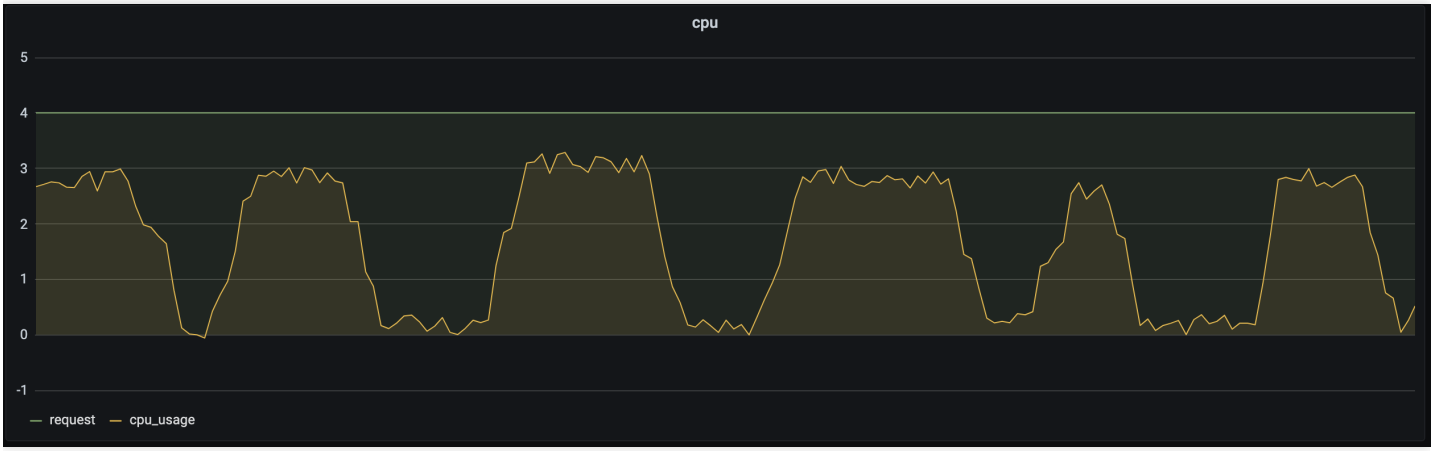
©2013-2022 Tencent Cloud. All rights reserved.



As you can see, resource reservation is way more than the actual usage, and the excessive part cannot be used by other loads. Obviously, setting `Request` to a very high value leads to great waste. In response, you need to set a proper value and limit infinite business requests as needed, so that resources will not be occupied overly by certain businesses. You can refer to `ResourceQuota` and `LimitRange` discussed later. In addition, TKE will launch a smart request recommendation product to help you narrow the gap between `Request` and `Usage`, effectively improving resource utilization while guaranteeing business stability.

Scenario 2: Business resource utilization sees an obvious change pattern, and resource waste is serious during off-peak hours, which usually last longer than peak hours

Most businesses see an obvious change pattern in resource utilization. For example, a bus system usually has a high load during the day and a low load at night, and a game often starts to experience a traffic surge on Friday night, which drops on Sunday night.

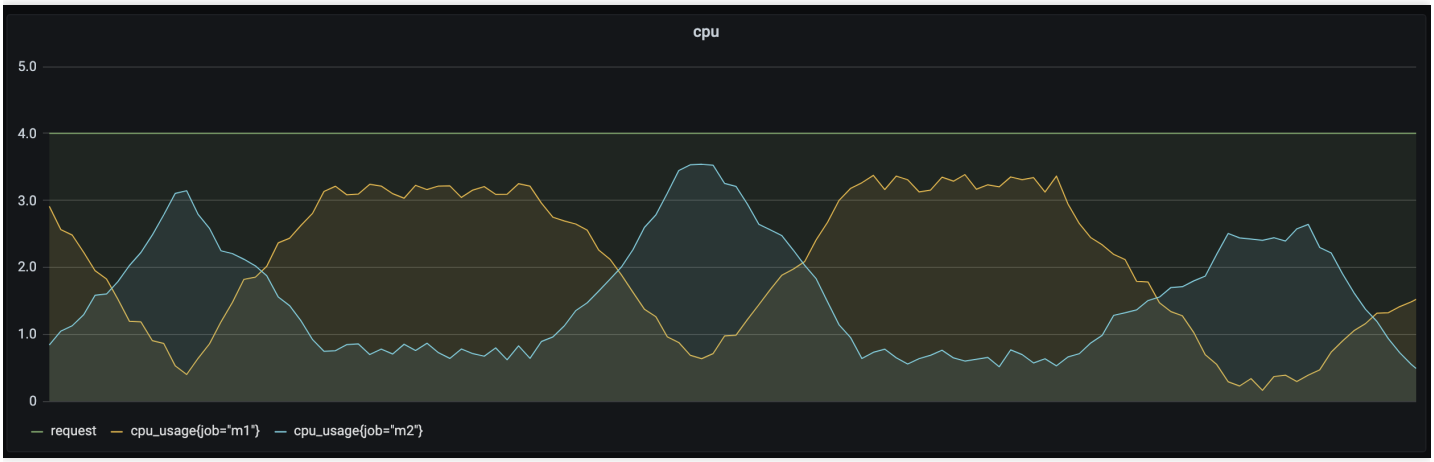


As you can see, the same business requests different amounts of resources during different time periods. If `Request` is set to a fixed value, utilization will be low when the load is low. The solution is to dynamically adjust the number of replicas to sustain different loads. For more information, see **HPA, HPC, and CA** discussed later.

Scenario 3: Resource utilization differs greatly by business type

©2013-2022 Tencent Cloud. All rights reserved.

Online businesses usually have a high load during the day and require a low latency, so they must be scheduled and run first. In contrast, offline businesses generally have low requirements for the operating time period and latency and can run during off-peak hours of online business loads. In addition, some businesses are computing-intensive and consume a lot of CPU resources, while others are memory-intensive and consume a lot of memory resources.



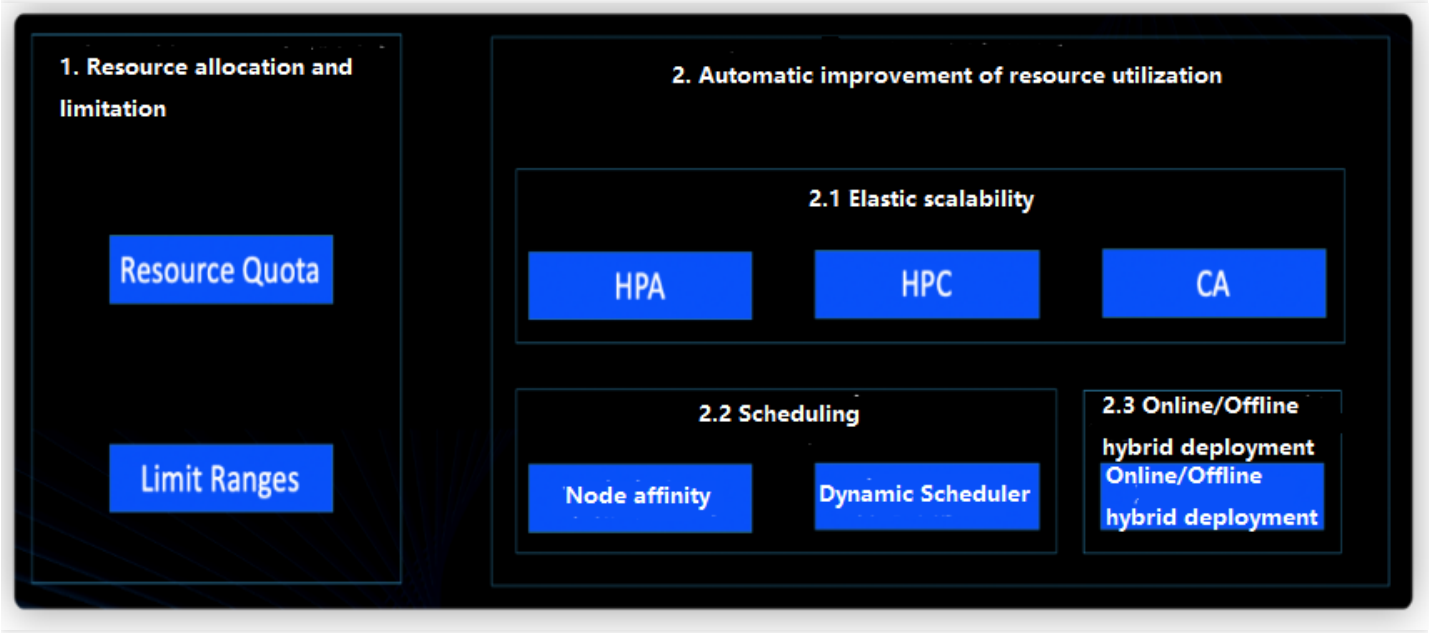
As shown above, online/offline hybrid deployment helps dynamically schedule offline and online businesses in different time periods to improve resource utilization. For computing-intensive and memory-intensive businesses, affinity scheduling can be used to find the right node. For detailed directions, see online/offline hybrid deployment and affinity scheduling discussed later.

Improving Resource Utilization in Kubernetes

TKE has productized a series of tools based on a large number of actual businesses, helping you easily and effectively improve resource utilization. There are two ways: 1. manual resource allocation and limitation based on Kubernetes

©2013-2022 Tencent Cloud. All rights reserved.

native capabilities; 2. automatic solution based on business characteristics.



1. Resource allocation and limitation

Imagine that you are a cluster administrator and your cluster is shared by four business departments. You need to ensure business stability while allowing for on-demand use of resources. In order to improve the overall utilization, it is necessary to limit the upper threshold of resources used by each business and prevent excess usage using some default values.

Ideally, `Request` and `Limit` values are set as needed. Here, `Request` is resource occupation, indicating the minimum amount of resources available for a container; `Limit` is resource limit, indicating the maximum amount of resources available for a container. This contributes to healthier container running and higher resource utilization, despite the fact that `Request` and `Limit` are often left unspecified. In the case of cluster sharing by teams/projects, `Request` and `Limit` tend to be set to high values to ensure stability. When you create a load in the TKE console, the following default values will be set for all containers, which are based on actual business analysis and estimation and may deviate from real-world requirements.

Resource	Request	Limit
CPU (core)	0.25	0.5
Memory (MiB)	256	1,024

To fine-tune your resource allocation and management, you can set namespace-level "resource quota" and "limit range" on TKE.

- `ResourceQuota` for resource allocation

©2013-2022 Tencent Cloud. All rights reserved.

- Use "LimitRange" for resource limitation

If your cluster has four businesses, you can use the namespace and `ResourceQuota` to isolate them and limit resources.

`ResourceQuota` is used to set a quota on resources in a namespace, which is an isolated partition in a Kubernetes cluster. A cluster usually contains multiple namespaces to house different businesses. You can set different `ResourceQuota` values for different namespaces to limit the cluster resource usage by a namespace, thus implementing preallocation and limitation. `ResourceQuota` applies to the following. For more information, see Resource Quotas.

1. Computing resources: Sum of "request" and "limit" of CPU and memory for all containers
2. Storage resources: Sum of storage requests of all PVCs.
3. Number of objects: Total number of resource objects such as PVC/Service/Configmap/Deployment

`ResourceQuota` use cases

- Assign different namespaces to different projects/teams/businesses and set resource quotas for each namespace for allocation.
- Set an upper limit on the amount of resources used by a namespace to improve cluster stability and prevent excessive preemption and consumption of resources by a single namespace.

`ResourceQuota` in TKE

TKE has productized `ResourceQuota`. You can directly use it in the console to limit the resource usage of a namespace. For detailed directions, see Namespace.

2. Automatic improvement of resource utilization

`ResourceQuota` and `LimitRange` for resource allocation and limitation respectively rely on experience and manual operations, mainly addressing unreasonable resource requests and allocation. This section describes how to improve resource utilization through automated dynamic adjustments from the perspectives of elastic scaling, scheduling, and online/offline hybrid deployment.

2.1 Elastic scaling

- HPA for elastic scaling by metric
- HPC for scheduled scaling
- CA for automatic adjustment of the number of nodes

In the second resource waste scenario, if your business goes through peak and off-peak hours, a fixed resource "request" value is bound to cause resource waste during off-peak hours. In this case, you can consider automatically

©2013-2022 Tencent Cloud. All rights reserved.

increasing and decreasing the number of replicas of the business load during peak and off-peak hours respectively to enhance the overall utilization.

Horizontal Pod Autoscaler (HPA) can automatically increase and decrease the number of Pod replicas in Deployment and StatefulSet based on metrics such as CPU and memory utilization to stabilize workloads and achieve truly on-demand usage.

HPA use cases

1. Traffic bursts: if traffic becomes heavy suddenly, the number of Pods is automatically increased in time at overload.
2. Automatic scaling in: if traffic is light, the number of Pods is automatically decreased to avoid waste at underload.

HPA in TKE

TKE supports many metrics for elastic scaling based on the custom metrics API, covering CPU, memory, disk, network, and GPU in most HPA scenarios. For more information on the list, see HPA Metrics. In complex scenarios such as automatic scaling based on the QPS per replica, the prometheus-adapter can be installed. For detailed directions, see Using Custom Metrics for Auto Scaling in TKE.

2.2 Scheduling

The Kubernetes scheduling mechanism is a native resource allocation mechanism which is efficient and graceful. Its core feature is to find the right node for each Pod. In TKE scenarios, the scheduling mechanism contributes to the transition from application-layer to resource-layer elastic scaling. A reasonable scheduling policy can be configured based on business characteristics by properly leveraging Kubernetes scheduling capabilities to effectively enhance resource utilization in clusters.

- Node affinity
- Dynamic Scheduler

If Kubernetes' scheduler accidentally scheduled a CPU-intensive business instance to a memory-intensive node, all the CPU of the node will be taken up, but its memory will be barely used, resulting in a serious waste of resources. In this case, you can flag such a node as a CPU-intensive node and flag a business load during creation to indicate that it is a CPU-intensive load. Kubernetes' scheduler will then schedule the load to a CPU-intensive node, a way of finding the most suitable node to effectively improve resource utilization.

When creating Pods, you can set node affinity to specify nodes to which Pods will be scheduled (these nodes are specified with Kubernetes labels).

Node affinity use cases

Node affinity is ideal for scenarios where there are workloads with different resource requirements running simultaneously in a cluster. For example, Tencent Cloud's CVM nodes can be CPU-intensive or memory-intensive. If certain businesses require much higher CPU usage than memory usage, using common CVMs will inevitably cause a

©2013-2022 Tencent Cloud. All rights reserved.

huge waste of memory. In this case, you can add a batch of CPU-intensive CVMs to the cluster and schedule CPU-consuming Pods to these CVMs, so as to improve the overall utilization. Similarly, you can manage heterogeneous nodes (such as GPUs) in the cluster, specify the amount of GPU resources needed in the workloads, and have the scheduling mechanism find the right nodes to run these workloads.

Node affinity in TKE

TKE provides an identical method to use node affinity as native Kubernetes. You can use this feature in the console or by configuring a YAML file. For detailed directions, see Proper Resource Allocation.

2.3 Online/Offline hybrid business deployment

If you have both online web businesses and offline computing businesses, you can use TKE's online/offline hybrid deployment technology to dynamically schedule and run businesses to improve resource utilization.

In the traditional architecture, big data and online businesses are independent and deployed in different resource clusters. Generally, big data businesses are for offline computing and experience peak hours during nights, during which online businesses are barely loaded. Leveraging complete isolation capabilities of containers (involving CPU, memory, disk I/O, and network I/O) and strong orchestration and scheduling capabilities of Kubernetes, cloud-native technologies implement the hybrid deployment of online and offline businesses to fully utilize resources during idle hours of online businesses.

Use cases of online/offline hybrid deployment

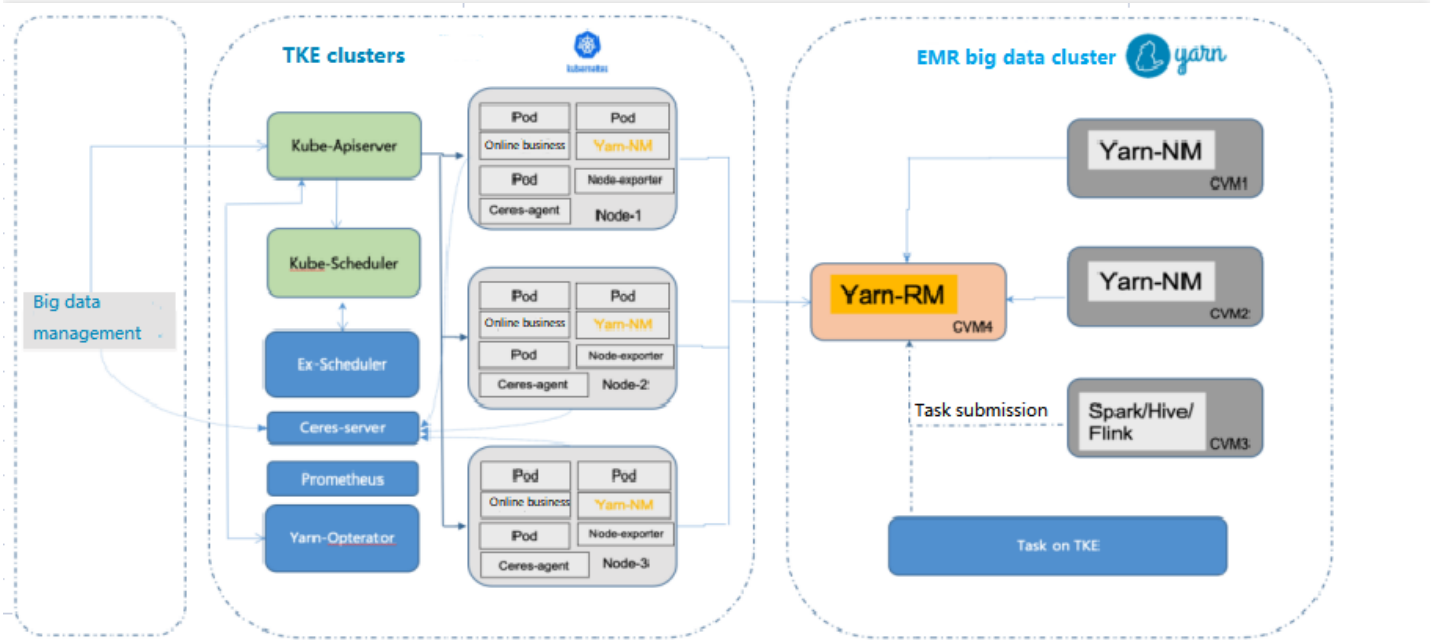
In the Hadoop architecture, offline and online jobs are in different clusters. Online and streaming jobs experience obvious load fluctuations, which means a lot of resources will be idle during off-peak hours, leading to great waste and higher costs. In clusters with online/offline hybrid deployment, offline tasks are dynamically scheduled to online clusters during off-peak hours, significantly improving resource utilization. Currently, Hadoop YARN can only statically allocate resources based on the static resource status reported by `NodeManager`, making it unable to well support hybrid deployment.

Online/Offline hybrid deployment in TKE

Online businesses experience obvious and regular load fluctuations, with a low resource utilization at night. In this case, the big data management platform delivers resource creation requests to Kubernetes clusters to increase the

©2013-2022 Tencent Cloud. All rights reserved.

computing power of the big data application. For more information, see here.



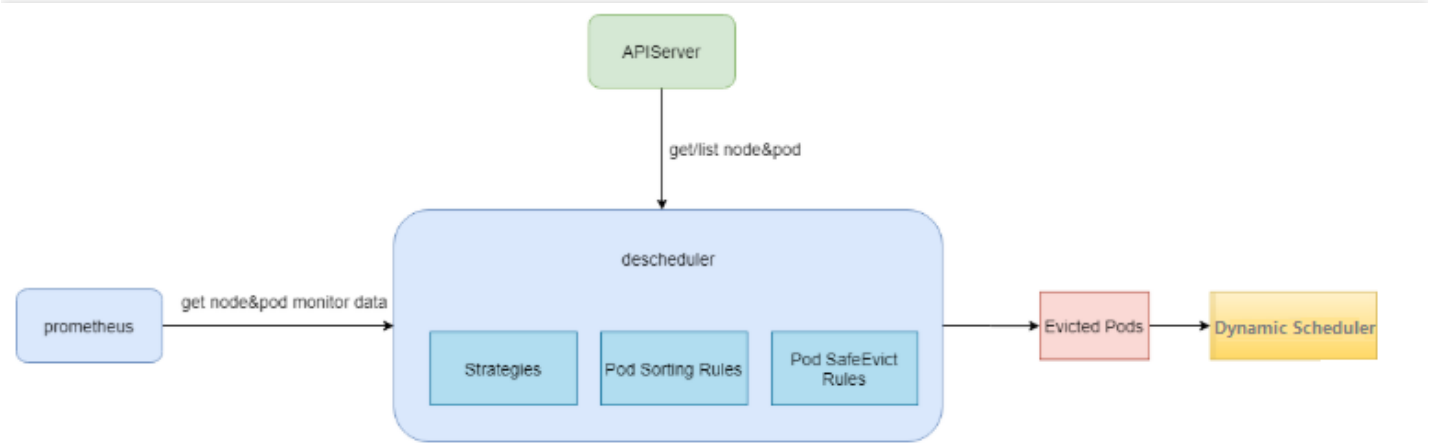
How to Balance Resource Utilization and Stability

Besides costs, system stability is another metric that weighs heavily in enterprise Ops. It's challenging to balance the two. On the one hand, the higher the resource utilization, the better for cost reduction; on the other hand, a too high resource utilization may cause overload and thereby OOM errors or CPU jitters.

To help enterprises get rid of the dilemma, TKE provides the **DeScheduler** to keep the cluster load under control. It is responsible for protecting nodes with risky loads and gracefully draining businesses from them. The relationship

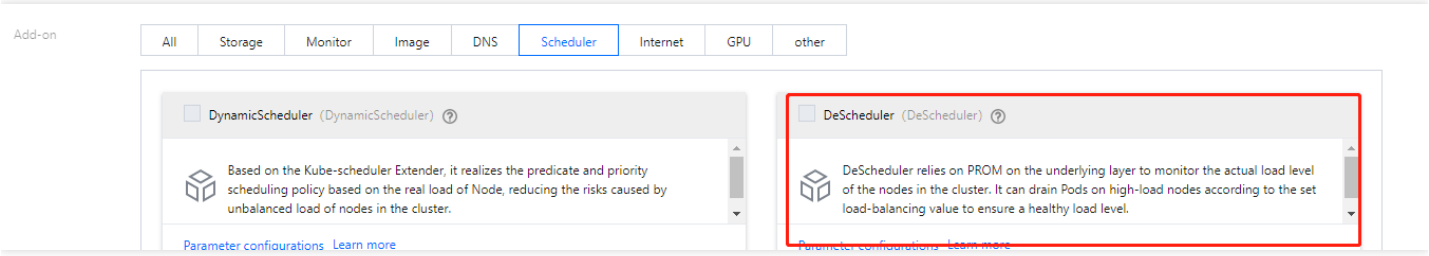
©2013-2022 Tencent Cloud. All rights reserved.

between the DeScheduler and the Dynamic Scheduler is as shown below:



DeScheduler in TKE

You can install and use the DeScheduler in an extended add-on. For detailed directions, see DeScheduler.



For more information on the DeScheduler guide, see here.

©2013-2022 Tencent Cloud. All rights reserved.