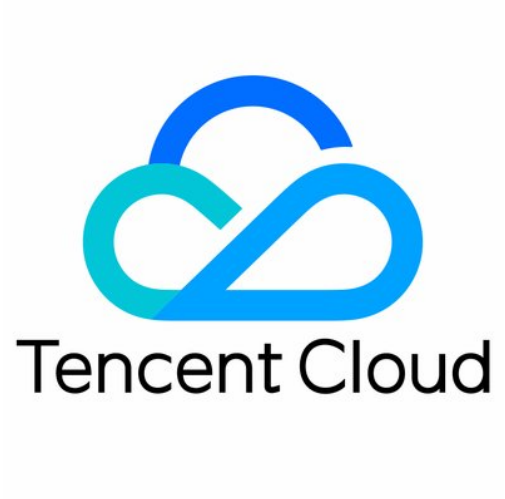
Cloud File Storage

Best Practices

Product Documentation



Contents

Best Practices
Using CFS on TKE

Last updated：2022-01-18 14:38:06

Overview

Tencent Kubernetes Engine (TKE) is a container management service with high scalability and performance. It provides a variety of application release methods and continuous delivery capabilities and supports microservice architectures. It can solve environment-related issues in the process of development, testing, and OPS and help you reduce costs and improve efficiency.

Businesses using containers usually involve shared access by multiple containers to a high number of configuration files, model files, log data, and attachments in such scenarios as business application deployment, DevOps, machine learning, and auto-scaling. Especially, in scenarios like machine learning, intelligent recommendation, and log data processing, in addition to basic data sharing, shared storage is also required to provide services that feature high throughput, high IOPS, and low latency and can sustain high concurrent requests. CFS can provide such services after simple configuration and mount to containers, making it particularly suitable for use in conjunction with TKE. This document describes how to use CFS on TKE.

Prerequisites

You have already created a TKE cluster. If you haven't done so, please create one first as instructed in Deploying TKE.

Applying for CFS Resources and Getting Mount Point IP

- If you do not have a file system, please create one as instructed in Creating File Systems and Mount Points. Please make sure that the VPC you select when creating the file system is the same as that of your TKE master to ensure network interconnectivity.
- If you already have a file system in the same VPC as your TKE instance, you can enter the "File System Details" page to get the mount point IP.

Configuring CFS File System Mounting

Step 1. Start the NFS client on the node

Before mounting, please make sure that `nfs-utils` or `nfs-common` has already been installed in the system. The installation method is as follows:

- CentOS

```
sudo yum install nfs-utils
```

- Ubuntu

```
sudo apt-get install nfs-common
```

Step 2. Create a PV

Run the following command to create a PesistentVolume (PV) of CFS type.

```
apiVersion: v1
kind: PersistentVolume
metadata:
name: cfs
spec:
capacity:
storage: 10Gi
volumeMode: Filesystem
accessModes:
- ReadWriteMany
persistentVolumeReclaimPolicy: Retain
mountOptions:
- hard
- nfsvers=4
nfs:
path: /
server: 10.0.1.41
```

Note：

- nfs.server: mount point IP of the CFS file system obtained above. The file system IP is 10.0.1.41 in this example.
- nfs.path: root directory or subdirectory of the CFS file system. The root directory is used in this example.

Step 3. Create a PVC

Then, create a PersistentVolumeClaim (PVC) and bind it to the PV.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
name: cfsclaim
spec:
accessModes:
- ReadWriteMany
volumeMode: Filesystem
storageClassName: ""
resources:
requests:
storage: 10Gi
```

Step 4. Create a pod

Create a pod to declare that this volume is used for mounting.

```
kind: Pod
apiVersion: v1
metadata:
name: mypod
spec:
containers:
- name: myfrontend
image: nginx
volumeMounts:
- mountPath: "/var/www/html"
name: mypd
volumes:
- name: mypd
persistentVolumeClaim:
claimName: cfsclaim
```

After the above steps are completed, you can use the file system in the newly created pod.

Using CFS on SCF

Last updated：2022-01-18 14:44:36

Overview

Tencent Cloud's Serverless Cloud Function (SCF) is a serverless execution environment that enables you to build and run applications without having to purchase and manage servers. Simply code in a supported language and set the execution conditions, and your code can be run on the Tencent Cloud infrastructure elastically and securely. SCF is an ideal computing platform for use cases such as real-time file processing and data processing.

SCF is a service-level computing resource that features fast iteration and super-fast deployment. As a result, it requires the separation of storage and computing, which makes CFS, a high-performance shared storage service, the best storage solution for SCF. With only a few easy steps, your function can easily access files stored in CFS. The benefits of using CFS on SCF are as follows:

- The execution space of functions is unlimited.
- Multiple functions can share the same file system to share files.

Directions

Associating SCF with a CFS access policy

> Note：
> To use the CFS service, SCF needs permission to operate on your CFS resources.

Follow the steps below to grant the permission to your account:

1. Associate the `SCF_QcsRole` role with the `QcloudCFSReadOnlyAccess` policy as instructed in Modifying a Role.
   If you don't perform this operation for your currently used account, problems such as the failure to save functions and the unavailability of CFS features may occur.
2. If the currently used account is a sub-account, please request the root account to associate the sub-account with the `QcloudCFSReadOnlyAccess` policy as instructed in Setting Sub-user Permissions.
   If you don't perform this operation for your currently used sub-account, problems such as the unavailability of CFS features may occur.

Creating a VPC

Build a VPC as instructed in Building Up an IPv4 VPC.

Creating CFS resources

Create a CFS file system as instructed in Creating File Systems and Mount Targets.

> Note：
> Currently, SCF allows only CFS file systems whose network type is VPC to be added as mount targets. When creating a CFS file system, please select the same VPC as that of the target function to enable communication.

Mounting and using the CFS file system on SCF

1. Log in to the SCF console and click **Function Service** in the left sidebar.
2. On the **Function Service** page, select the name of the function to be configured.
3. On the **Function Configuration** tab of the **Function Management** page, click **Edit** in the top-right corner.
4. Check **Enable** for **VPC**, and select the VPC where your CFS file system resides.
5. Check **Enable** for **File System**, and enter the following information to mount the file system, as shown below:
   - **User ID** and **User Group ID**: IDs of the user and user group in the CFS file system. SCF uses “10000” for both the user ID and user group ID by default to manipulate your CFS file system. Please set the file owner and corresponding group permission as needed and ensure that your CFS file system has the required permission. For more information, please see Managing Permissions.
   - **Remote Directory**: the remote directory in the CFS file system to be accessed by the function, which consists of a file system and a remote directory.
   - **Local Directory**: mount target of the local file system. You can use a subdirectory in the `/mnt/` directory to mount the CFS file system.
   - **File System ID**: select the file system to be mounted in the drop-down list.
   - **Mount Point ID**: select the ID of the mount target corresponding to the file system in the drop-down list.
6. Click **Save** at the bottom of the page to complete the configuration.
   You can edit the function code to start using the CFS file system, as shown below:

```
'use strict';
var fs = require('fs');
exports.main_handler = async (event, context) => {
await fs.promises.writeFile('/mnt/file.txt', JSON.stringify(event));
return event
}
```

Performance test for using the CFS file system on SCF

You can use this Demo to test how well CFS performs on SCF.

Using CFS Turbo to Integrate with TKE

Last updated：2022-01-18 14:50:24

Overview

This document describes how to integrate CFS Turbo with a Tencent Cloud Kubernetes Engine (TKE) cluster.

Prerequisites

- The operating system of the TKE host node is compatible with the Turbo series.
- You have installed a Turbo-based client on all TKE nodes. You are advised to use pshell to operate in batches. For the compatible operating systems and how to install the clients, please see Using CFS Turbo on Linux Clients.

Directions

Download and configure kubectl

1. Download kubectl by referring to kubectl Documentation.
2. Log in to the TKE console and click Cluster in the left sidebar.
3. On the **Cluster Management** page, click the ID of the target cluster to go to the cluster details page.
4. On the cluster details page, click **Basic Information**.
5. In **Cluster APIServer Information**, click **Download** to download the `kubeconfig` file to the default environment variable address and name it `config`, that is, `/usr/local/bin/config`.
6. In **Cluster APIServer Information**, set **Private Network Access** to  and click  to copy the command for host configuration.
7. Switch to the access machine to run the command copied from Step 6.

8. Run the following commands to configure the environment variable:

```
vi /etc/profile
KUBECONFIG=/usr/local/bin/config
PATH=$PATH
export KUBECONFIG
export PATH
```

9. Run the following command to verify whether kubectl has been installed successfully:

```
kubectl get node
```

If the following message is displayed, the installation is completed:

```
[root@VM-0-5-centos ~]# kubectl get node
NAME          STATUS   ROLES    AGE   VERSION
[redacted]    Ready    <none>   15d   v1.18.4-tke.9
```

Mounting the Turbo pod with a script

1. See TKE Turbo Plugin Guide and download the script.
2. Go to the `kubernetes-csi-tencentcloud/deploy/cfsturbo/kubernetes/` directory and upload `csi-node-rbac.yaml`, `csi-node.yaml`, and `csidriver.yaml` to the CVM management node that can access the TKE cluster.
3. Go to the `kubernetes-csi-tencentcloud/deploy/cfsturbo/examples/` directory to download the `static-allinone.yaml` sample file.
4. Modify the `static-allinone.yaml` file according to the actual PV, PVC, and pod attributes (e.g., name, image address). NGINX is used as an example here.

```
sudo mount.lustre -o sync,user_xattr 10.0.1.16@tcp0:/d3dcc487/cfs /path/to/moun
t
```

Where, the host is `10.0.1.16` and the fsid is `d3dcc487`.

The modified script sample is as follows:

```
apiVersion: v1
kind: PersistentVolume
metadata:
name: pv-cfsturbo
spec:
accessModes:
```

```
- ReadWriteMany
capacity:
storage: 10Gi
csi:
driver: com.tencent.cloud.csi.cfsturbo
volumeHandle: pv-cfsturbo
volumeAttributes:
# cfs turbo server ip
host: 10.0.0.116
# cfs turbo fsid(not cfs id)
fsid: xxxxxxxx
proto: lustre
storageClassName: ""
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
name: pvc-cfsturbo
spec:
storageClassName: ""
volumeName: pv-cfsturbo
accessModes:
- ReadWriteMany
resources:
requests:
storage: 10Gi
---
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
```

5. Run the following commands in sequence in the directory where the script is uploaded:
   - Configure RBAC:

     ```
     kubectl apply -f csi-node-rbac.yaml
     ```

   - Configure the node CSI plugin:

     ```
     kubectl apply -f csidriver.yaml
     ```

     ```
     kubectl apply -f csi-node.yaml
     ```

   - Create PV, PVC, and pod:

     ```
     kubectl create -f static-allinone.yaml
     ```

6. Run the following command to view the pod status:

   ```
   kubectl get pod -n default -o wide
   ```

   If the message below is displayed, the pod is created successfully:

   ```
   NAME     READY   STATUS    RESTARTS   AGE   IP          NODE           NOMINATED NODE   READINESS GATES
   nginx2   1/1     Running   0          36s   10.0.0.8    192.168.0.2    <none>           <none>
   ```

   If the value of `STATUS` is `ContainerCreating`, the creation failed. You can find the events in the TKE console to troubleshoot.

Using CFS Turbo on EKS

Last updated：2022-02-16 12:36:13

Overview

You can mount a CFS Turbo storage for an EKS cluster. The add-on is used to mount a Tencent Cloud CFS Turbo file system to a workload based on a proprietary protocol. Currently, only static configuration is supported. For more information about CFS storage types, see Storage Types and Performance.

Prerequisites

An EKS cluster of v1.14 or later has been created.

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 Node Plugin

Step 1. Create a csidriver.yaml file

Here is an example of a csidriver.yaml file:

```
apiVersion: storage.k8s.io/v1beta1
kind: CSIDriver
metadata:
name: com.tencent.cloud.csi.cfsturbo
spec:
attachRequired: false
podInfoOnMount: false
```

Step 2. Create a CSI driver

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

```
apiVersion: v1
kind: PersistentVolume
metadata:
name: pv-cfsturbo
spec:
accessModes:
- ReadWriteMany
capacity:
storage: 10Gi
csi:
driver: com.tencent.cloud.csi.cfsturbo
volumeHandle: pv-cfsturbo
volumeAttributes:
host: *.*.*.*
fsid: ********
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 the file system mount target.
- **spec.csi.volumeAttributes.fsid**: fsid of the file system (not the file system ID). You can check it in the information about the file system mount target. It is the string after "tcp0:/" and before "/cfs" in the mount command, as shown below.

Step 2. Create a PVC that is bound to the PV based on the following template

```
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**: 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
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
```