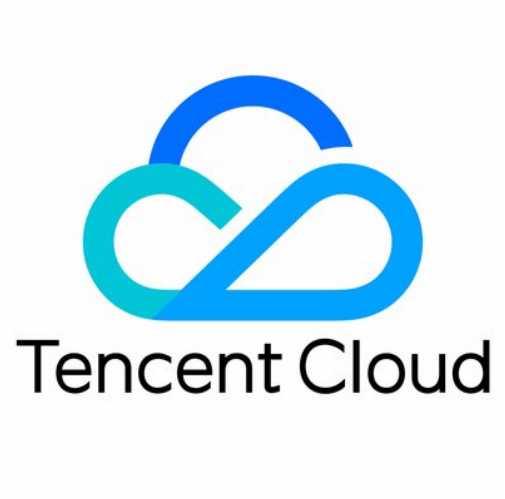
Tencent Kubernetes Engine

Cloud Native AI Guide

Product Documentation



Contents

Cloud Native AI Guide
- Ops Console Guide
  - Managing AI Environment
  - AI Add-On Management
  - AI Component List
    - Fluid
    - TF Operator
    - MPI Operator
    - Elastic Jupyter Operator
- Model Training
  - Running TF Training Job
  - Running PyTorch Training Job

Cloud Native AI Guide
Ops Console Guide
Managing AI Environment

Last updated：2022-10-12 11:56:32

This document describes AI environments and how to manage them, such as how to create, view, and delete them.

AI Environment Overview

AI environment is an important abstract concept of Cloud Native AI. An AI environment runs in a TKE/EKS container cluster, and the Ops team can manage its lifecycle as needed. For example, for different upper-layer AI businesses, the AI Ops team can combine applicable add-ons to set up diverse AI environments based on various business needs.

Directions

Creating an AI environment

1. Log in to the TKE console and click **Cloud Native AI** on the left sidebar.
2. On the **AI Environment** list page, click **Create** to enter the **Create AI Environment** page and set the parameters.

- **Environment Name**: Custom environment name. You can name the environment based on information such as business needs to facilitate subsequent resource management.
- **Region**: The region of the cluster where the AI environment is to be deployed.
- **Cluster Type**: The type of cluster where the AI environment is to be deployed.
- **Cluster**: The cluster where the AI environment is to be deployed.
- **Deploy Add-On**: Add-ons to be deployed in the AI environment. You can also install and delete add-ons in Add-On Management after environment creation.

5. Click **Create**.

Viewing an AI environment

After creating an AI environment, you can view it on the **AI Environment** list page.

Deleting an AI environment

1. Log in to the TKE console and click **Cloud Native AI** on the left sidebar.
2. On the **AI Environment** list page, click **Delete** on the right of the target environment.
3. In the **Delete AI Environment** pop-up window, read the notes on deletion and click **Confirm**.

AI Add-On Management

Last updated：2022-10-12 11:37:14

Overview

After creating an AI environment, you can combine AI add-ons as needed to set up an AI platform. This document describes how to add, delete, modify, and query AI add-ons.

Note：
The underlying layer of AI add-ons is implemented based on Helm Chart. After you create an AI environment, we recommend you not manage AI add-ons on the relevant page in the application marketplace. Instead, directly manage them **in the AI environment**, so as to avoid data inconsistency.

List of AI Add-Ons

Add-On	Use Case	Description
TF Operator	Model training	After installing it, you can run TF standalone/distributed training jobs.
MPI Operator	Elastic training	You can run elastic training jobs to fully utilize the computing resources.
Fluid	Cache acceleration	Fluid provides data prefetch and acceleration for cloud applications by using a distributed cache engine (GooseFS/Alluxio) with data observability, portability, and horizontal scalability.
Elastic Jupyter Operator	Algorithm debugging	Elastic Jupyter Operator provides an on-demand elastic Jupyter Notebook service to assign computing resources as needed.

AI Add-On Lifecycle Management

Creating an AI add-on

1. Log in to the TKE console and click **Cloud Native AI** on the left sidebar.

2. On the **AI Environment** list page, click the ID of the target AI environment to enter its **Basic Information** page.
3. On the left sidebar, click **Add-On Management**.
4. Click **Create** to enter the **Create AI Add-On** page and set the parameters.

   The main parameters are described as follows:

   - Add-On Name: Custom add-on name.
   - Namespace: The namespace for installing the add-on.
   - Chart: The installation package of the add-on. Only one add-on can be installed at a time.
   - Parameter: Add-on configuration parameters. After the add-on is created, you can still update its parameters as instructed in "Updating an AI add-on".
5. Click **Done**.

Viewing an AI add-on

After creating an AI environment, you can view the list of installed AI add-ons in the AI environment.

Deleting an AI add-on

1. Select the ID of an AI environment to enter its **Basic Information** page.
2. On the left sidebar, click **Add-On Management**.
3. Select **Delete** on the right of the target add-on.
4. In the **Delete Add-On** pop-up window, read the notes on deletion and click **Confirm**.

Updating an AI add-on

1. Select the ID of an AI environment to enter its **Basic Information** page.
2. On the left sidebar, click **Add-On Management**.
3. Select **Update configuration** on the right of the target add-on.
4. On the **Update Add-On** pop-up page, configure add-on parameters as needed and click **Done**.

AI Component List
Fluid

Last updated：2022-06-21 11:14:06

Overview

Fluid is an open-source Kubernetes-native distributed dataset orchestrator and accelerator for data-intensive applications, such as big data and AI. It is hosted by the Cloud Native Computing Foundation (CNCF) as a sandbox project. By defining the abstraction of dataset resources, it features:

- **Native support for dataset abstraction**: Implements the basic capabilities required for data-intensive applications to achieve efficient data access and reduce the cost of multidimensional management.
- **Cloud data prefetch and acceleration**: Fluid provides data prefetch and acceleration for cloud applications by using a distributed cache engine (GooseFS/Alluxio) with data observability, portability, and horizontal scalability.
- **Co-orchestration for data and applications**: During application and data scheduling on the cloud, it takes their characteristics and location into consideration to improve the performance.
- **Multi-namespace management support**: Allows you to create and manage datasets in different namespaces.
- **Heterogeneous data source management**: Unifies the access to underlying data from different sources (COS, HDFS, and Ceph), applicable to hybrid cloud use cases.

Key Concepts

Dataset: A dataset is a set of logically related data that can be used by computing engines, such as Spark for big data and TensorFlow for AI. Smart data applications create core industry values. Managing datasets may require features in different dimensions, such as security, version management, and data acceleration.

Runtime: The execution engine that enforces dataset security and provides version management and data acceleration capabilities. It defines a set of APIs for dataset management and acceleration throughout the lifecycle.

GooseFS Runtime: It is a Java-based implementation of the execution engine developed by Tencent Cloud's COS team, supporting dataset management, caching, and COS. GooseFS is a Tencent Cloud product with dedicated product-level support, but its code is not open-source. Fluid enables dataset visualization, elastic scaling, and data migration by managing and scheduling GooseFS Runtime.

Alluxio Runtime: Based on open-source Alluxio, it is an implementation of the execution engine for dataset management and caching, supporting PVC, Ceph, and CPFS computing, thereby effectively supporting hybrid cloud use cases. Alluxio is an open-source scheme. In spite of the joint efforts of Tencent Cloud and the community to

promote the stability and performance of its data caching, there will be a delay in timeliness and response. Fluid enables dataset visualization, elastic scaling, and data migration by managing and scheduling Alluxio Runtime.

-	Alluxio	GooseFS
Underlying storage types	PVC, Ceph, HDFS, CPFS, NFS	OSS, EMR, PVC, Ceph, HDFS, CPFS, NFS
Support	Open-source community	Tencent Cloud products

Add-on Installation

Prerequisite dependencies

- Kubernetes cluster (v1.14 or later)
- CSI support in the cluster

Parameter configuration

During Helm deployment, all configuration items are included in `values.yaml` .
Some fields may need to be customized, as listed below:

Parameter	Description	Default Value
`workdir`	Backup address of the metadata in the cache engine	`/tmp`
`dataset.controller.image.repository`	Repository where the dataset controller image resides	`ccr.ccs.tencentyun.com/controller`
`dataset.controller.image.tag`	Dataset controller image version	`"v0.6.0-0bfc552"`

Parameter	Description	Default Value
`csi.registrar.image.repository`	Repository where the CSI registrar image resides	`"ccr.ccs.tencentyun.co driver-registrar"`
`csi.registrar.image.tag`	CSI registrar image version	`"v1.2.0"`
`csi.plugins.image.repository`	Repository where the CSI plugins image resides	`"ccr.ccs.tencentyun.co`
`csi.plugins.image.tag`	CSI plugins image version	`"v0.6.0-def5316"`
`csi.kubelet.rootDir`	`kubelet root` folder	`"/var/lib/kubelet"`
`runtime.mountRoot`	Root address of the FUSE mount in the cache engine	`"/var/lib/kubelet"`
`runtime.goosefs.enable`	Enable GooseFS cache engine	`"true"`

Parameter	Description	Default Value
`runtime.goosefs.init.image.repository`	Repository where the initialized image of the GooseFS cache engine resides	`"ccr.ccs.tencentyun.co`
`runtime.goosefs.init.image.tag`	Version of the initialized image of the GooseFS cache engine	`"v0.6.0-0cd802e"`
`runtime.goosefs.controller.image.repository`	Repository where the controller image of the GooseFS cache engine resides	`"ccr.ccs.tencentyun.co` `controller"`
`runtime.goosefs.controller.image.tag`	Version of the controller image of the GooseFS cache engine	`"v0.6.0-bbf4ea0"`

Parameter	Description	Default Value
`runtime.goosefs.runtime.image.repository`	Repository where the GooseFS cache engine image resides	`"ccr.ccs.tencentyun.co`
`runtime.goosefs.runtime.image.tag`	Version of the GooseFS cache engine image	`"v1.1.10"`
`runtime.goosefs.fuse.image.repository`	Repository where the FUSE add-on image of the GooseFS cache engine resides	`"ccr.ccs.tencentyun.co`
`runtime.goosefs.fuse.image.tag`	Version of the FUSE add-on image of the GooseFS cache engine	`"v1.1.10"`
`runtime.alluxio.runtimeWorkers`	Maximum number of the concurrent workers of the Alluxio cache engine controller	`"3"`

Parameter	Description	Default Value
`runtime.alluxio.portRange`	Alluxio cache engine add-on port range	`"20000-26000"`
`runtime.alluxio.enable`	Enable Alluxio cache engine	`"true"`
`runtime.alluxio.init.image.repository`	Repository where the initialization image of the Alluxio cache engine resides	`"ccr.ccs.tencentyun.co`
`runtime.alluxio.init.image.tag`	Version of the initialization image of the Alluxio cache engine	`"v0.6.0-def5316"`
`runtime.alluxio.controller.image.repository`	Repository where the controller image of the Alluxio cache engine resides	`"ccr.ccs.tencentyun.co` `controller"`
`runtime.alluxio.controller.image.tag`	Version of the controller image of the Alluxio cache engine	`"v0.6.0-0cd802e"`

Parameter	Description	Default Value
`runtime.alluxio.runtime.image.repository`	Repository where the Alluxio cache engine image resides	`"ccr.ccs.tencentyun.co`
`runtime.alluxio.runtime.image.tag`	Version of the Alluxio cache engine image	`"release-2.5.0-2-SNAPS`
`runtime.alluxio.fuse.image.repository`	Repository where the FUSE add-on image of the Alluxio cache engine resides	`"ccr.ccs.tencentyun.co` `fuse"`
`runtime.alluxio.fuse.image.tag`	Version of the FUSE add-on image of the Alluxio cache engine	`"release-2.5.0-2-SNAPS`

Best Practices

For more information, see the Fluid documentation.

TF Operator

Last updated：2022-10-12 11:37:14

Overview

Developed by the Kubeflow community, TF-Operator is an add-on used to help deploy and execute TensorFlow distributed training jobs in a Kubernetes cluster.

After deployment, you can create, view, and delete TF jobs.

Prerequisite dependencies

Kubernetes cluster (v1.16 or later)

Deployment

During Helm deployment, all configuration items are included in `values.yaml`.

Some fields may need to be customized, as listed below:

Parameter	Description	Default Value
`image.repository`	The repository where the TF-Operator image resides	`ccr.ccs.tencentyun.com/kubeflow-oteam/tf-operator`
`image.tag`	TF-Operator image version	`"latest"`
`namespace.create`	Whether to create a separate namespace for TF-Operator	`true`
`namespace.name`	The namespace where TF-Operator is to be deployed	`"tf-operator"`

Best practices

See Running TF Training Job.

MPI Operator

Last updated：2022-10-12 11:37:14

Overview

Developed by the Kubeflow community, MPI-Operator is an add-on used to help deploy and execute data-parallel distributed training such as Horovod in a Kubernetes cluster.

After deployment, you can create, view, and delete MPI jobs.

Prerequisite dependencies

Kubernetes cluster (v1.16 or later)

Deployment

During Helm deployment, all configuration items are included in `values.yaml`.

Some fields may need to be customized, as listed below:

Parameter	Description	Default Value
`image.repository`	The repository where the MPI-Operator image resides	`ccr.ccs.tencentyun.com/kubeflow-oteam/mpi-operator`
`image.tag`	MPI-Operator image version	`"latest"`
`namespace.create`	Whether to create a separate namespace for MPI-Operator	`true`
`namespace.name`	The namespace where MPI-Operator is to be deployed	`"mpi-operator"`

Elastic Jupyter Operator

Last updated：2022-10-12 16:05:09

Overview

elastic-jupyter-operator is a native elastic Jupyter service in Kubernetes. It provides an elastic Jupyter Notebook service as needed with the following features:

- It automatically releases resources to the Kubernetes cluster when the GPU is idle.
- It supports delayed resource application, allowing you to apply for CPU, memory, and GPU resources as needed.
- Multiple Jupyter notebooks share a resource pool to increase the resource utilization.

Deployment

During Helm deployment, all configuration items are included in `values.yaml`.

Some fields may need to be customized, as listed below:

Parameter	Description	Default Value
`image.repository`	The repository where the image resides	`ccr.ccs.tencentyun.com/kubeflow-oteam/elastic-jupyter-operator`
`image.tag`	Image version	`"v0.1.1"`
`namespace.name`	Namespace	`"enterprise-gateway"`

How to use

Note：
For more information, see elastic-jupyter-operator.

1. Run the following command to create a Jupyter Gateway CR:

```
kubectl apply -f ./config/samples/kubeflow.tkestack.io_v1alpha1_jupytergateway.yaml
```

Below is the content of the YAML file:

```
apiVersion: kubeflow.tkestack.io/v1alpha1
kind: JupyterGateway
metadata:
name: jupytergateway-sample
spec:
cullIdleTimeout: 3600
```

Here, `cullIdleTimeout` is a configuration item. If a kernel is idle in the time in seconds specified by `cullIdleTimeout` , Gateway will repossess it to release resources.

2. Run the following command to create a Jupyter Notebook CR instance and specify the Gateway CR:

```
kubectl apply -f ./config/samples/kubeflow.tkestack.io_v1alpha1_jupyternotebook.y
aml
```

Below is the content of the YAML file:

```
apiVersion: kubeflow.tkestack.io/v1alpha1
kind: JupyterNotebook
metadata:
name: jupyternotebook-sample
spec:
gateway:
name: jupytergateway-sample
namespace: default
```

3. All resources in the cluster are as listed below:

```
NAME READY STATUS RESTARTS AGE
pod/jupytergateway-sample-6d5d97949c-p8bj6 1/1 Running 2 11d
pod/jupyternotebook-sample-5bf7d9d9fb-nq9b8 1/1 Running 2 11d
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
service/jupytergateway-sample ClusterIP 10.96.138.111 <none> 8888/TCP 11d
service/kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 31d
NAME READY UP-TO-DATE AVAILABLE AGE
deployment.apps/jupytergateway-sample 1/1 1 1 11d
deployment.apps/jupyternotebook-sample 1/1 1 1 11d
NAME DESIRED CURRENT READY AGE
replicaset.apps/jupytergateway-sample-6d5d97949c 1 1 1 11d
replicaset.apps/jupyternotebook-sample-5bf7d9d9fb 1 1 1 11d
```

4. Use a method such as NodePort, `kubectl port-forward` , or Ingress to expose Notebook CR to provide the Service. Here, `kubectl port-forward` is used as an example. Run the following command:

```
kubectl port-forward jupyternotebook-sample-5bf7d9d9fb-nq9b8 8888
```

API documentation

See API Reference.

Model Training
Running TF Training Job

Last updated：2023-05-19 17:12:40

This document describes how to run a TF training job.

Prerequisites

- TF Operator has been installed in your AI environment.
- Your AI environment has GPU resources.

Directions

The following steps are based on the official distributed training examples in parameter server/worker mode of `TF-Operator`.

Preparing the training code

The code sample dist_mnist.py at the official website of Kubeflow is used.

Creating a training image

Image creation is easy. You only need to get an official image based on TensorFlow 1.5.0, copy the above code to the image, and configure `entrypoint`.

> Note：
> If `entrypoint` is not configured, you can also configure the container startup command when submitting a `TFJob`.

Submitting the job

1. Prepare a `TFJob` YAML file to define two parameter servers and four workers.

> Note

You need to replace the `<training image="">` placeholder with the address of the uploaded training image.

```
apiVersion: "kubeflow.org/v1"
kind: "TFJob"
metadata:
name: "dist-mnist-for-e2e-test"
spec:
tfReplicaSpecs:
PS:
replicas: 2
restartPolicy: Never
template:
spec:
containers:
- name: tensorflow
image: <training image>
Worker:
replicas: 4
restartPolicy: Never
template:
spec:
containers:
- name: tensorflow
image: <training image>
```

2. Run the following command to use `kubectl` to submit the `TFJob` :

```
kubectl create -f ./tf_job_mnist.yaml
```

3. Run the following command to view the job status:

```
kubectl get tfjob dist-mnist-for-e2e-test -o yaml
kubectl get pods -l pytorch_job_name=pytorch-tcp-dist-mnist
```

Running PyTorch Training Job

Last updated：2023-05-19 17:07:37

This document describes how to run a PyTorch training job.

Prerequisites

- PyTorch Operator has been installed in your AI environment.
- Your AI environment has GPU resources.

Directions

The following steps are based on the official distributed training examples of `PyTorch-Operator` .

Preparing the training code

The code sample mnist.py at the official website of Kubeflow is used.

Creating a training image

Training image creation is easy. You only need to get an official image based on PyTorch 1.0, copy the above code to the image, and configure `entrypoint` (if `entrypoint` is not configured, you can also configure the startup command when submitting a `PyTorchJob`).

> Note：
> The training code is written based on PyTorch 1.0. As APIs of different PyTorch versions may be incompatible, you may need to adjust the above training code in a PyTorch environment on other versions.

Submitting the job

1. Prepare a `PyTorchJob` YAML file to define one master worker and one worker.

> Note
> - You need to replace the `<training image="">` placeholder with the address of the uploaded training image.

- As GPU resources are configured in resource configuration, set `backend` for training to `"nccl"` in `args`; in jobs using no (Nvidia) GPU resources, use another backend such as `gloo`.

```
apiVersion: "kubeflow.org/v1"
kind: "PyTorchJob"
metadata:
name: "pytorch-dist-mnist-nccl"
spec:
pytorchReplicaSpecs:
Master:
replicas: 1
restartPolicy: OnFailure
template:
metadata:
annotations:
sidecar.istio.io/inject: "false"
spec:
containers:
- name: pytorch
image: <training image>
args: ["--backend", "nccl"]
resources:
limits:
nvidia.com/gpu: 1
Worker:
replicas: 1
restartPolicy: OnFailure
template:
metadata:
annotations:
sidecar.istio.io/inject: "false"
spec:
containers:
- name: pytorch
image: <training image>
args: ["--backend", "nccl"]
resources:
limits:
nvidia.com/gpu: 1
```

2. Run the following command to use `kubectl` to submit the `PyTorchJob`:

```
kubectl create -f ./pytorch_job_mnist_nccl.yaml
```

3. Run the following command to view the `PyTorchJob` :

```
kubectl get -o yaml pytorchjobs pytorch-dist-mnist-nccl
```

4. Run the following command to view Pods created by the PyTorch job:

```
kubectl get pods -l pytorch_job_name=pytorch-dist-mnist-nccl
```