Cloud File Storage

Best Practices

Product Documentation

Contents

Best Practices
Selecting Kernels for NFS Clients

Last updated：2024-01-22 22:15:48

Since the NFS client is kernel-based, the bugs in some kernel versions will make the NFS service unavailable. To enjoy a better experience, please use recommended kernel versions.

Known client issues

No response of the file system due to kernel network stack defects (priority: high)

For a system with a kernel version in the range of v2.6.32-696 to v2.6.32-696.10.1 (including v2.6.32-696, excluding v2.6.32-696.10.1), when the kernel requests to retry due to a busy NFS server, a kernel network stack defect may be triggered, resulting in no response to the operation.
If there is no response to the operation, restart the CVM instance. For more information, see RHEL6.9:NFSv4 TCP transport stuck in FIN_WAIT_2 forever.

No response of the file system due to kernel defects (priority: high)

For a system with one of the following kernel versions, the NFS server failover mechanism may cause file opening, read, and write deadlocks in the NFS client, resulting in no response of the file system.
Redhat 6, CentOS 6 2.6.32-696.3.1.el6
All kernel versions earlier than Redhat 7 and CentOS 7 3.10.0-229.11.1.el7
Ubuntu 15.10 Linux 4.2.0-18-generic
If there is no response to the operation, restart the CVM instance. For more information, see RHEL7:NFSv4 client loops with WRITE/NFS4ERR_STALE_STATEID - if NFS server restarts multiple times within the grace period.
For a system with one of the following kernel versions, if network reconnection occurs due to network partitioning or jitter, the NFS client may continue to have no response due to failure to handle error codes properly. The manifestation is as follows: The file system gives no response, and `bad sequence-id error` is repeatedly printed in the system message.
All kernel versions earlier than Redhat 6 and CentOS 6 2.6.32-696.16.1.el6
All kernel versions earlier than Redhat 7 and CentOS 7 3.10.0-693.el7
If there is no response to the operation, restart the CVM instance. For more information, see RHEL6/RHEL7:NFS4 client receiving NFS4ERR_BAD_SEQID drops nfs4 stateowner resulting in infinite loop of READ/WRITE+NFS4ERR_BAD_STATEID.
For an operating system with a kernel version of CentOS/RedHat 5.11.x, executing operations that require traversing a directory, such as the ls command and a command containing the wildcard character * or ?, will cause jams or no

response due to kernel defects. In this case, please upgrade the kernel to avoid this issue in the future.

Failure to support the chown command and system call (priority: low)

A system with a v2.6.32 kernel does not support executing the chown command and system call in the NFS client.

Failure to terminate ls operations (priority: low)

For a system with a kernel version of v2.6.32-696.1.1.el6 or earlier, adding and deleting files or subdirectories are also performed while ls operations are in progress, making ls operations unable to end.
In this case, please upgrade the kernel to avoid this issue in the future.
For a system with a kernel version of v4.18.0-305.12.1, ls and other directory traverse operations cannot end. Please upgrade the kernel to v4.18.0-305.12.1 to fix the issue. For more information, see kernel-4.18.0-305.19.1.el8_4.x86_64.

Recommended NFS images

Linux images

OS Type	OS Version
CentOS	CentOS-6.9-x86_64: v2.6.32-696.16.1.el6.x86_64 or later CentOS-6.10-x86_64: v2.6.32-754.17.1.el6.x86_64 or later CentOS-7.2-x86_64: v3.10.0-514.26.2.el7.x86_64 or later CentOS-7.3-x86_64: v3.10.0-514.26.2.el7.x86_64 or later CentOS-7.4-x86_64: v3.10.0-693.2.2.el7.x86_64 or later CentOS-7.5-x86_64: v3.10.0-862.14.4.el7.x86_64 or later CentOS-7.6-x86_64: v3.10.0-957.21.3.el7.x86_64 or later CentOS-7.7-x86_64: v3.10.0-1062.18.1.el7.x86_64 or later CentOS-8.-x86_64: 4.18.0-147.5.1.el8_1.x86_64 or later
Tencent OS Linux	TencentOS Server 2.2 (Tkernel 3) TencentOS Server 2.4 (Tkernel 4) TencentOS Server 2.6 (Final) TencentOS Server 3.1 (Tkernel 4)
Debian	Debian-9.6-amd64: v4.9.0-8-amd64 or later Debian-9.8-amd64: v4.9.0-8-amd64 or later Debian-9.10-amd64: 4.9.0-9-amd64 or later
Ubuntu	Ubuntu-14.04-amd64: v4.4.0-93-generic or later Ubuntu-16.04-amd64: v4.4.0-151-generic or later Ubuntu-18.04-amd64: v4.15.0-52-generic or later Ubuntu-20.04-amd64: v5.4.0-31-generic or later

OpenSuse	OpenSuse-42.3-x86_64: v4.4.90-28-default or later
SUSE	Enterprise Server 12 SP2 x86_64: v4.4.74-92.35-default or later Enterprise Server 12 SP4 x86_64: v4.12.14-95.16-default or later
CoreOS	CoreOS-1745.7.0_64: v4.19.56-coreos-r1 or later CoreOS-2023.4.0_64: v4.19.56-coreos-r1 or later

Windows images

OS Type	OS Version
Windows Server 2012	Windows Server 2012 R2 Datacenter 64-bit Chinese Windows Server 2012 R2 Datacenter 64-bit English
Windows Server 2016	Windows Server 2016 DataCenter 64-bit Chinese Windows Server 2016 Datacenter 64-bit English
Windows Server 2019	Windows Server 2019 Datacenter 64-bit Chinese Windows Server 2019 IDC 64-bit English

Managing Turbo CFS Directories

Last updated：2024-01-22 22:15:48

Overview

This practice aims to achieve higher performance and lower latency of a Turbo file system by better allocating directories and files.

Background

When clients access a file system, operations are performed based on the virtual file system (VFS) layer. The VFS layer serially grants and recalls locks when processing multiple processes reading and writing the same file. If a client's I/O is under an extremely large directory at a certain level, the serial lock operations of VFS may cause increased I/O latency.
As a strongly consistent file system, a Turbo file system maintains consistency for any client to read and write at any time. However, this also means that all clients' I/O operations will involve the granting or recall of backend distributed locks. The Turbo backend can handle lock requests concurrently based on distributed metadata services. However, to ensure good performance in terms of latency, we recommend you manage the number of directories and files according to the suggestions in this document.

Suggestions

If your business mainly involves reading with no frequent delete/update/create operations, we recommend that a single directory contain 10 to1 million subdirectories and files.
If your business involves frequent delete/update/create operations, we recommend that a single directory contain 10 to 10,000 subdirectories and files.

Common directory structures

Sort by hash code: A 64-character hash code is used, where the first two characters form a first-level subdirectory, the next two characters form a second-level subdirectory, and files are stored under the second-level subdirectory. When the statistics feature of hash functions works properly, files can be evenly distributed in these 65,536 directories. When the file system has 600 million files, each directory has an average of 10,000 files. When the file system has 1.2 billion files, each directory has an average of 20,000 files.

Sort by time: The year, month, and day form the first-level subdirectory, the hour forms the second-level subdirectory, and the minute forms the third-level subdirectory (which can be omitted).

Terminating Compute Instances

Last updated：2024-01-22 22:15:47

Overview

This practice provides a better method to terminate CVM instances when you need to dynamically adjust the compute instances for a Turbo file system. This method avoids lock conflict caused by improper operations.
When users need to terminate CVM instances, they usually call TerminateInstances. However, this API will force the instances to shut down (similar to cutting off the power) and return them. This method achieves fast termination, but for CFS Turbo, a strongly consistent distributed file system, the force client disconnection will make the server lock recall abnormal, which will lead to IO hangs. To avoid affecting business, we recommend you terminate instances using the method provided in this document.

Directions

Shutting down CVM instances

Call StopInstances to shut down instances, with `StopType` set to `SOFT_FIRST`. A soft shutdown will be performed. If the soft shutdown fails within 5 minutes, a forced shutdown will be performed. This method takes the timeliness of the shutdown into account, without affecting the normal use of the Turbo file system.
Note:
1. Do not set `StopType` to `HARD` (forced shutdown).
2. When you perform a shutdown using the console, do not select forced shutdown. Select `SOFT_FIRST`.
3. A soft shutdown terminates processes in an orderly manner and performs a sync operation, minimizing the impact of a forced shutdown on the distributed lock in a Turbo file system.

Terminating instances

Call DescribeInstancesStatus to query the status of the instances. When the status is `STOPPED`, call TerminateInstances to terminate the instances.

Using CFS on TKE

Last updated：2024-01-22 22:15:48

Overview

CFS is suitable for two use cases in a container environment:

Use case 1. Persistent storage of Pod/container data (dynamic mounting of CFS recommended)

CFS provides a space for persistent storage where the data is still stored when a Pod or container is terminated. In this way, the original space can be quickly mounted through PVCs to implement data reads/writes quickly when another Pod or container is started.
Compared with other schemes, a single-FS instance allows you to store the data from multiple Pods or containers and assign different subdirectories in the CFS instance to different Pods. Standard and High-Performance CFS instances are pay-as-you-go with no requirement for the minimum purchase capacity. This helps reduce the costs of persistent data storage for large-scale containers.

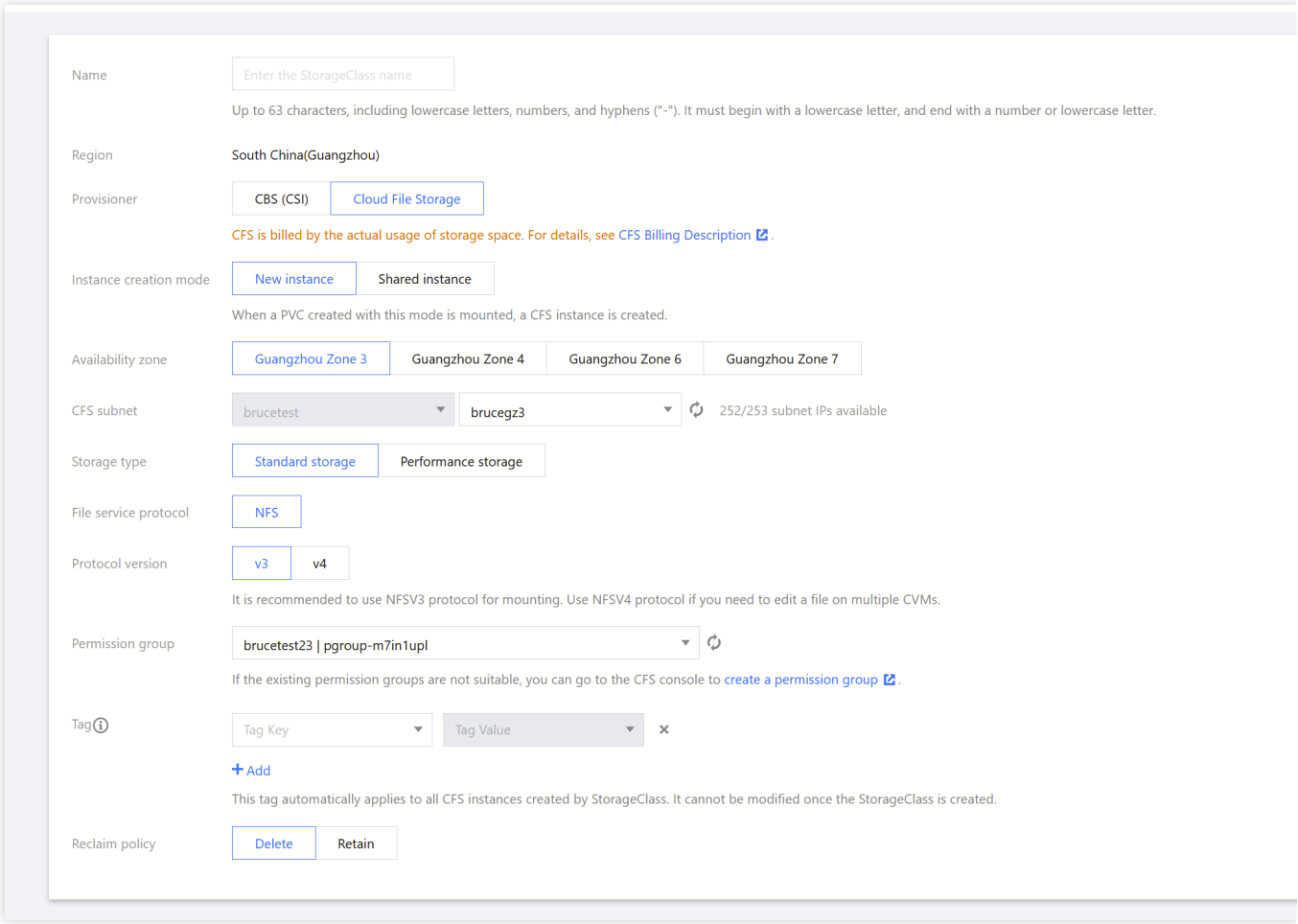
Use case 2. Multi-Pod/container data sharing (static mounting of CFS recommended)

CFS provides shared access to a directory space from multiple Pods or containers over NFS or private protocols for efficient data sharing. Compared with other schemes, this provides higher bandwidth and IOPS capabilities.
This section describes how to deploy a container workload in the Tencent Cloud console. For more information on how to write a YAML file, refer to the YAML file automatically generated after a StorageClass is created in the console.
Note:
Make sure that the CSI component in the TKE cluster is on v1.0.4 or later; otherwise, update it in the TKE console. The update operation does not affect the normal use of the container.

Directions

Dynamically mounting CFS

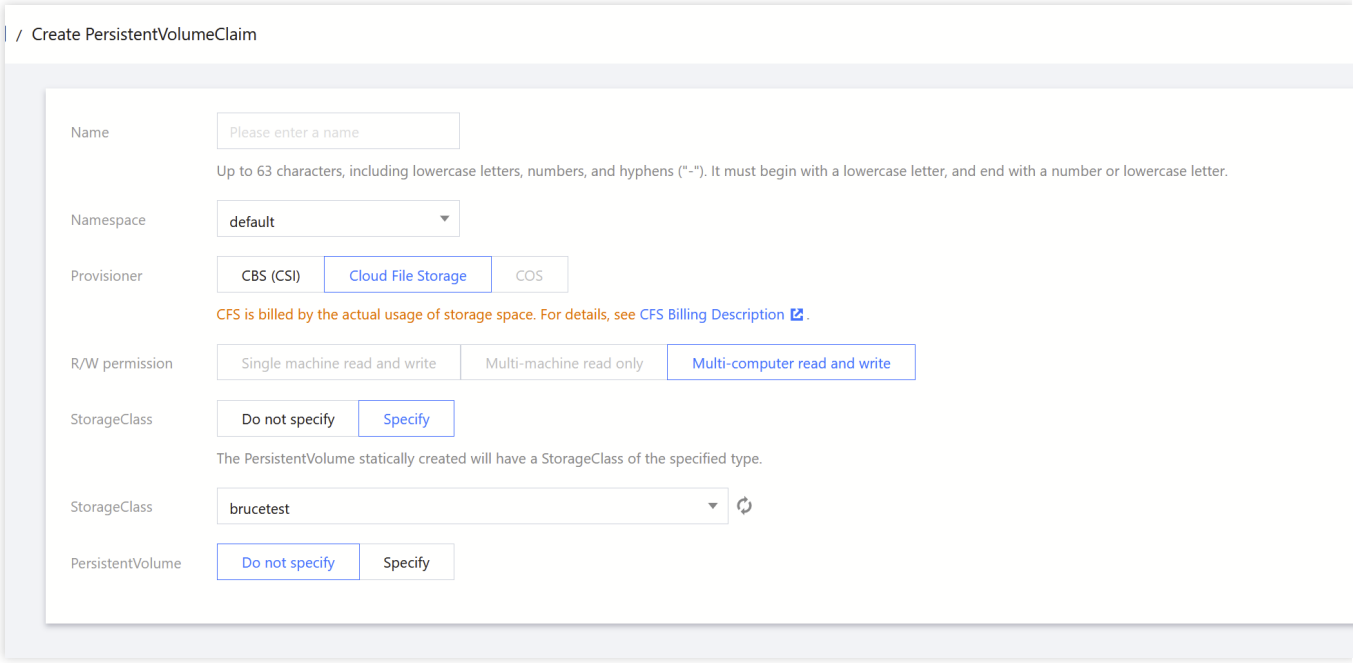
Note:
This mounting option is recommended for persistent storage of Pod/container data.
1. Create a StorageClass as instructed in Managing CFS Templates by Using a StorageClass.



Key configuration items are as described below:

Configuration Item	Description
Instance creation mode	Select **Shared instance**.
Availability zone	We recommend you select the same AZ as that of the container host.
Storage type	Select **Standard storage** or **Performance storage** as needed.
Protocol version	Unless in scenarios involving concurrent modifications, we recommend you use the NFS v3 protocol for better performance.
Reclaim policy	Select **Delete** or **Retain** as needed. To avoid unexpected data deletion, we recommend you select **Retain**.

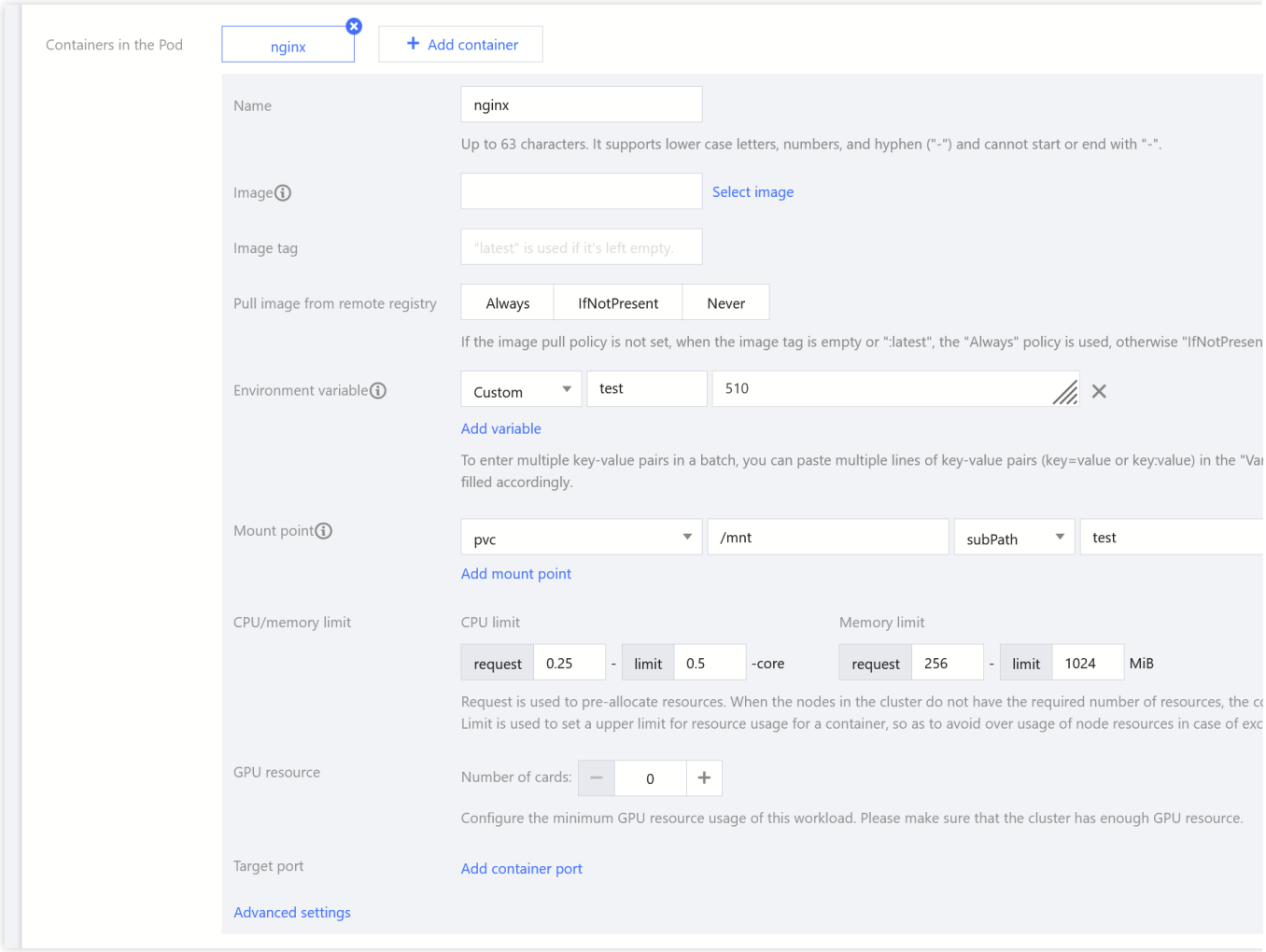
2. Create a PVC as instructed in Managing CFS by Using PVs and PVCs.



Key configuration items are as described below:

Configuration Item	Description
Namespace	Select a namespace as needed.
StorageClass	Select the StorageClass you just created.
PersistentVolume	You don't need to specify a PV for dynamic creation. Note: For a StorageClass based on a shared CFS instance, if you don't specify a PV when creating a PVC, the CSI plugin will automatically create a pay-as-you-go CFS instance when creating a PVC. This instance will be deleted when the PVC is deleted. Therefore, process PVCs created in this way with caution.

3. Create a Deployment as instructed in Deployment Management.



Key configuration items are as described below:

Configuration Item	Description
Volume	Name the volume as needed and select the PVC you just created.
Mount point	Select the volume and specify the path for mounting to the container. When you dynamically create a shared CFS instance, you need to specify an environment variable, and the CSI plugin will create a directory in the CFS instance corresponding to the selected PVC based on the value of the configured environment variable for the container to mount.

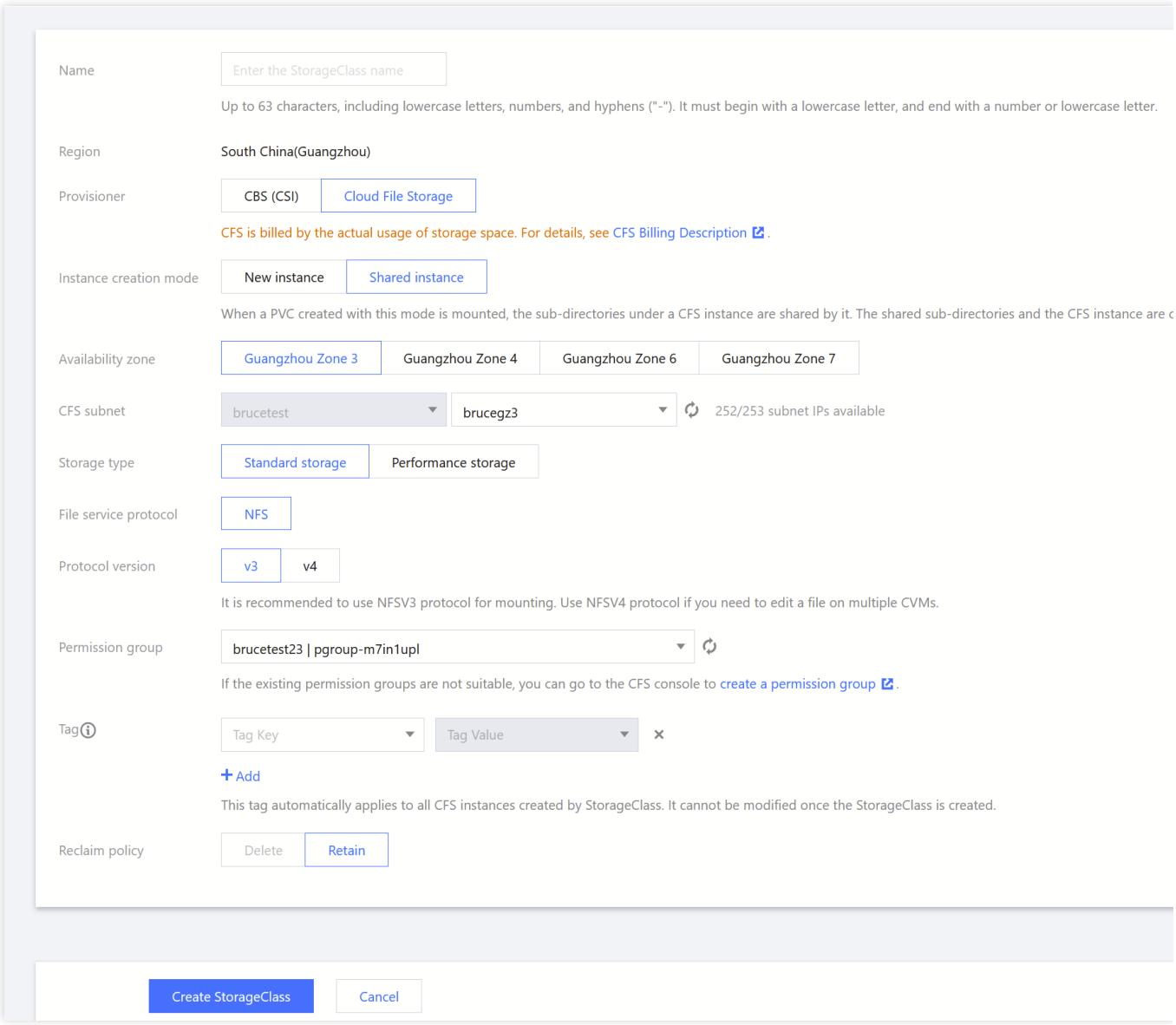
After completing the configuration, click **Create Workload**, and the system will create a container based on this configuration and mount CFS.

Statically mounting CFS

Note:

This mounting option is recommended for multi-Pod/container data sharing.

1. Create a StorageClass as instructed in Managing CFS Templates by Using a StorageClass.

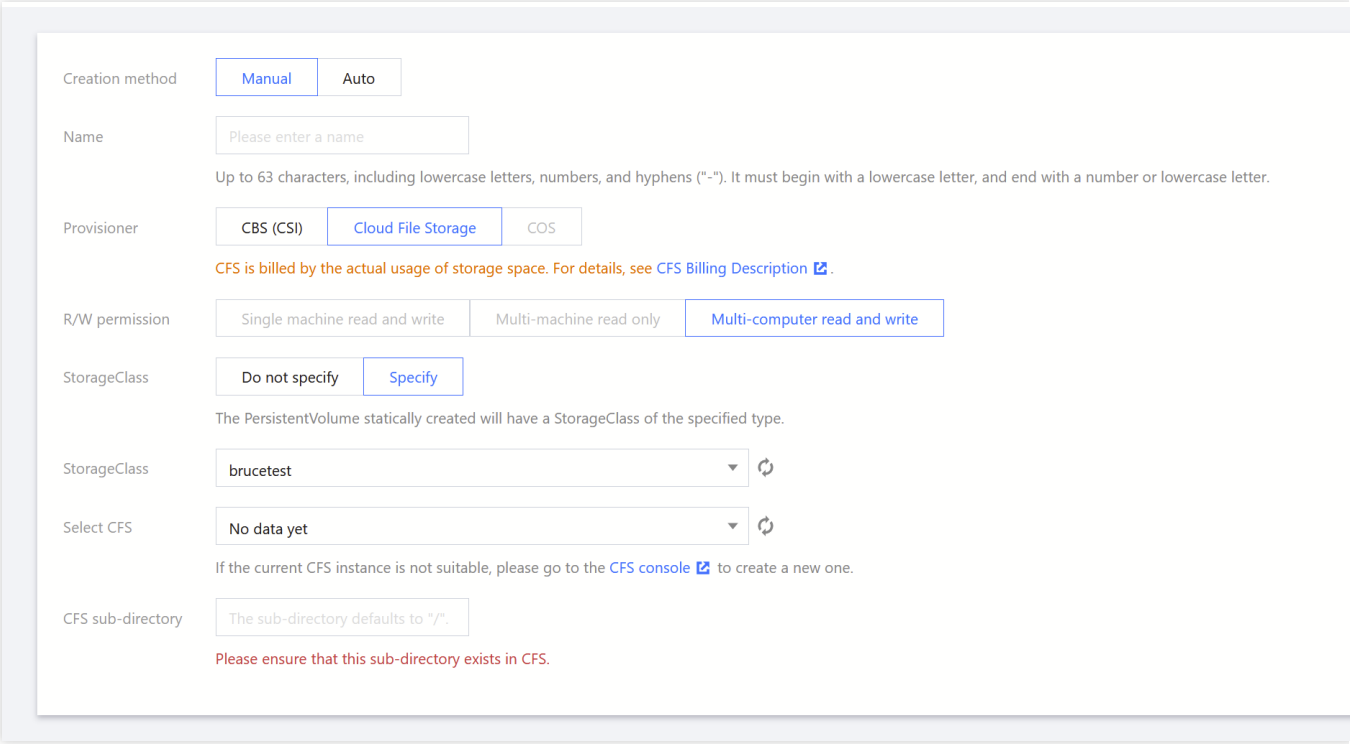


Key configuration items are as described below:

Configuration Item	Description
Instance creation mode	Select **Shared instance**.
Availability zone	We recommend you select the same AZ as that of the container host.
Storage type	Select **Standard storage** or **Performance storage** as needed.
Protocol version	Unless in scenarios involving concurrent modifications, we recommend you use the NFS v3 protocol for better performance.

Reclaim policy	Select **Delete** or **Retain** as needed. To avoid unexpected data deletion, we recommend you select **Retain**.

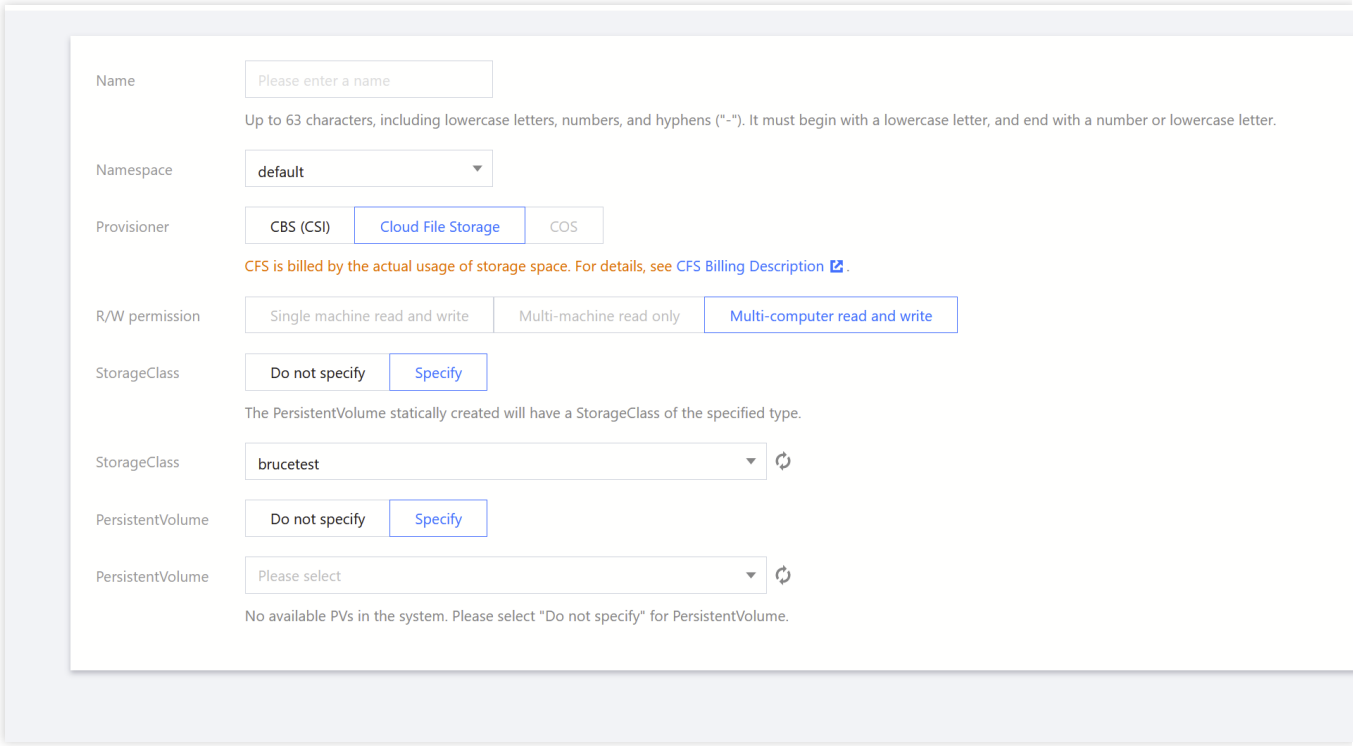
2. Create a PV as instructed in Managing CFS by Using PVs and PVCs.



Key configuration items are as described below:

Configuration Item	Description
Creation method	Select **Manual**; that is, specify a CFS instance for PV configuration.
StorgeClass	Select the StorageClass you just created.
Select CFS	Select a specified CFS instance. Note: During static creation, make sure that you already have a CFS instance in the same VPC as the container.
CFS subdirectory	CFS allows you to mount subdirectories. You can select different subdirectories and bind them to one or multiple PVs as needed to implement different degrees of data sharing.

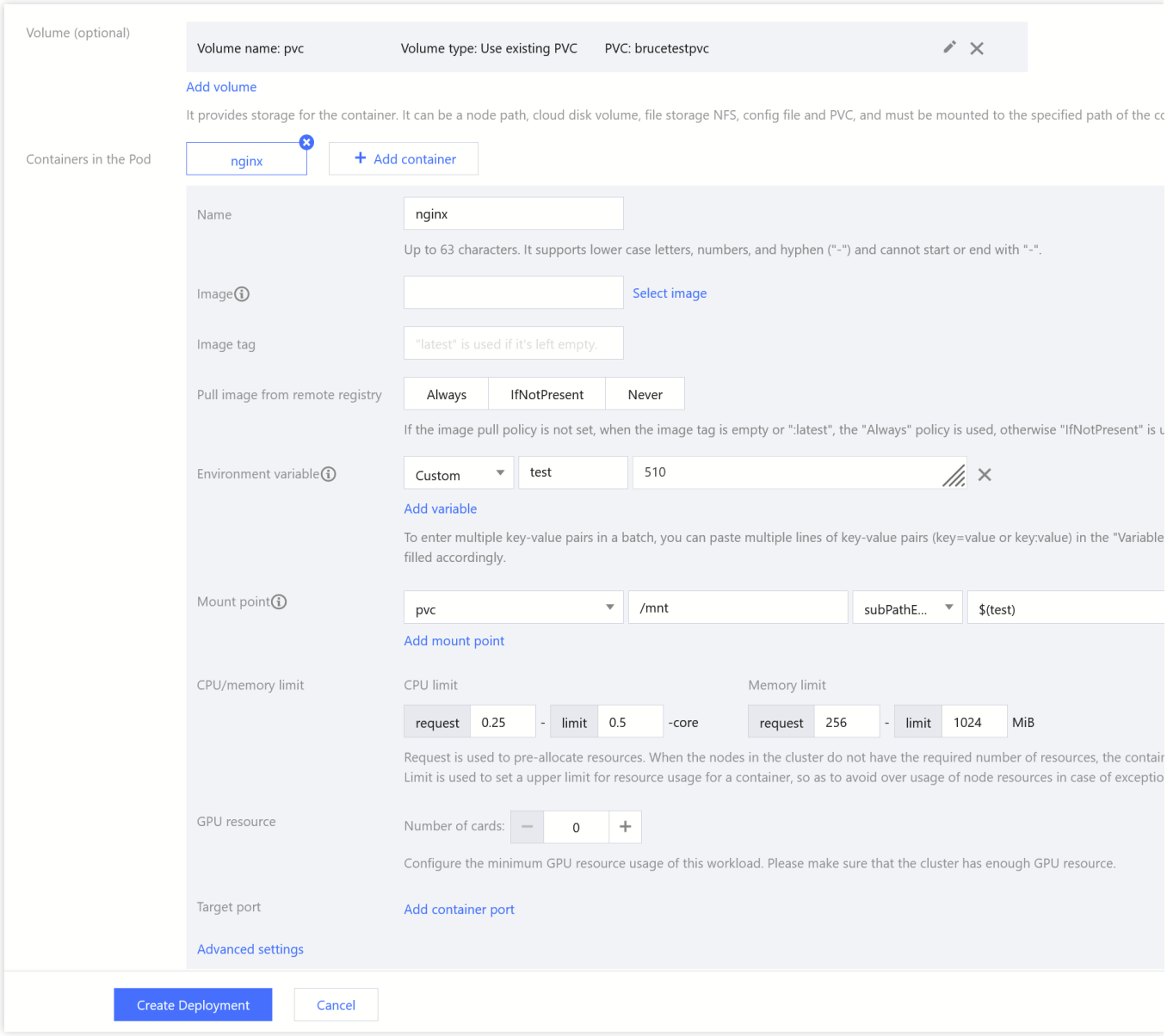
3. Create a PVC as instructed in Managing CFS by Using PVs and PVCs.



Key configuration items are as described below:

Configuration Item	Description
Namespace	Select a namespace as needed.
StorageClass	Select the StorageClass you just created.
PersistentVolume	Select **Specify** and select the PV you just created.

4. Create a Deployment as instructed in Deployment Management.



Key configuration items are as described below:

Configuration Item	Description
Volume	Name the volume as needed and select the PVC you just created.
Mount point	Select the volume and specify the path for mounting to the container. To achieve automatic subdirectory creation, you can use two methods. Method 1: Select subPath as the mount point and enter the name of the path you want to mount, such as test. Then, the CSI plugin will automatically create the test directory under the root path of the file system and automatically mount it to the specified directory of the container. Method 2: Add an environment variable and assign a value to it. Select subPathExpr as the mount point and select the environment variable. The CSI plugin will use the value of the environment variable as the directory name, automatically create the directory under the root path of the file system, and automatically mount it to the specified directory of the container.

After completing the configuration, click **Create Workload**, and the system will create a container based on this configuration and mount CFS.

Using CFS on SCF

Last updated：2024-01-22 22:15:48

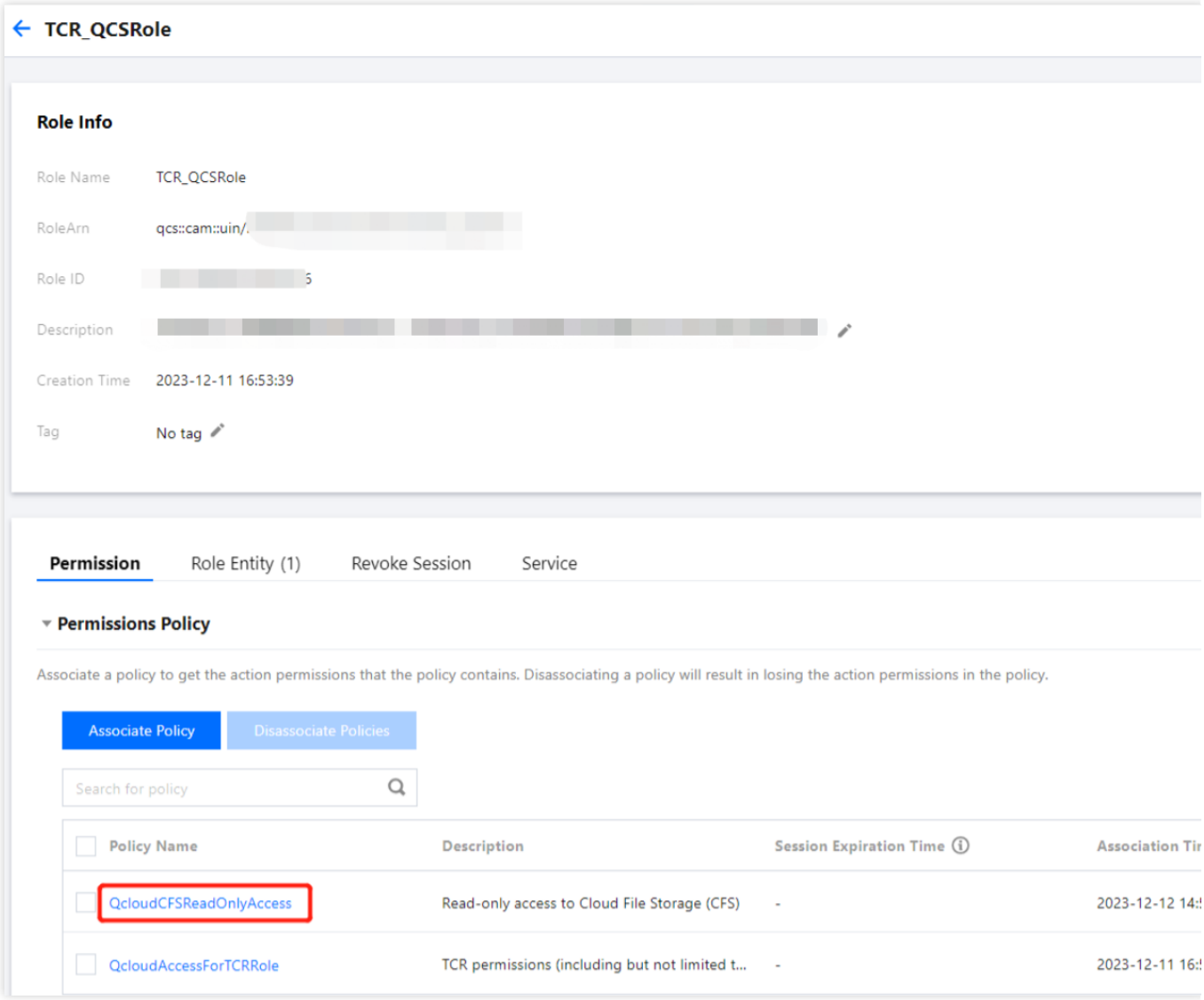
Overview

Tencent Cloud 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

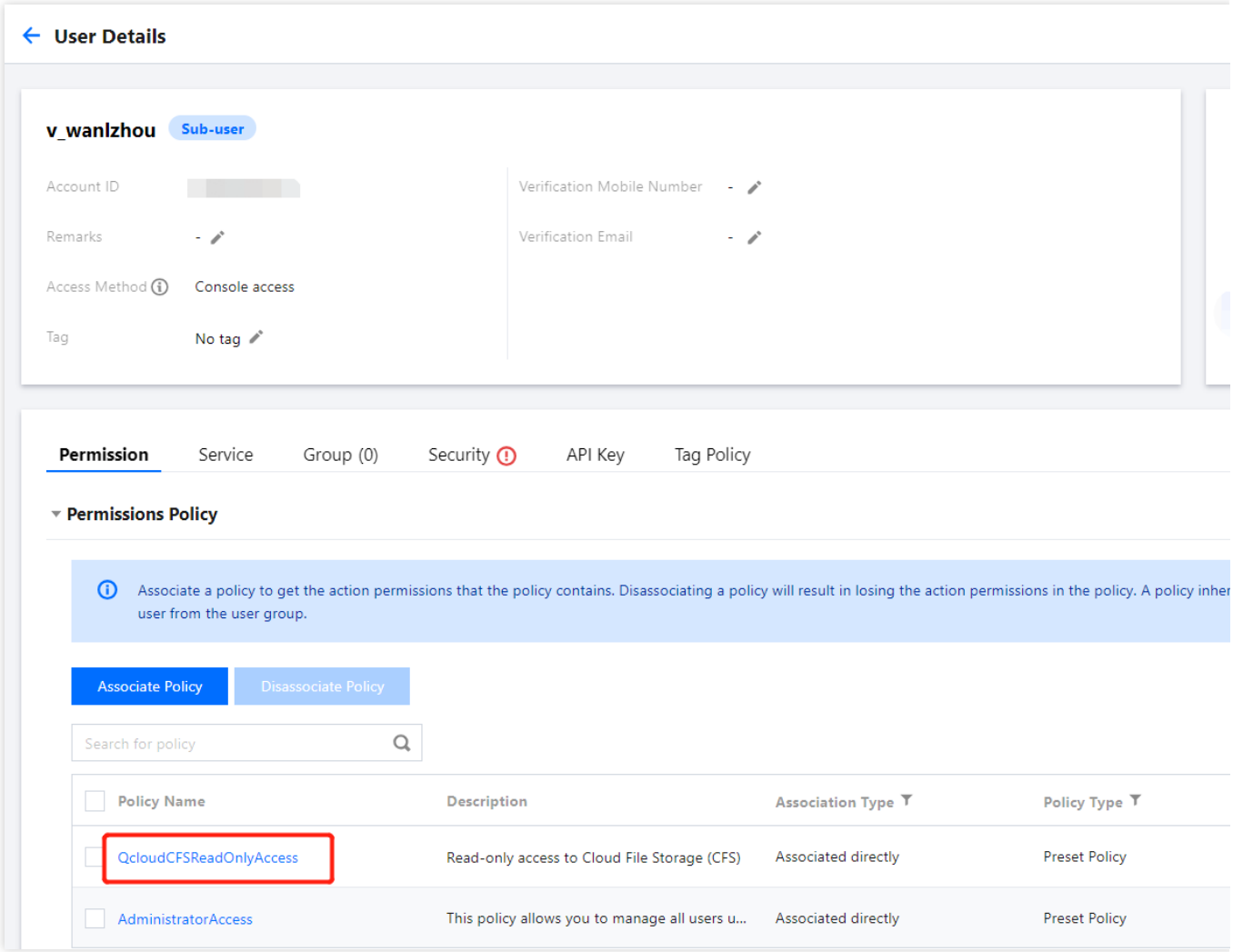
Associating an authorization policy

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 Role. The result of a successful association is shown below:
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, request the root account to associate the sub-account with the `QcloudCFSReadOnlyAccess` policy as instructed in Setting Sub-user Permissions. The result of a successful association is shown below:

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 with network type being VPC to be added as mount targets. When creating a CFS file system, select the same VPC as that of the target function to enable communication.

Mounting and using a CFS file system

1. Log in to the SCF console and select **Functions** on 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.

User ID and **User group ID**: IDs of the user and user group in CFS file system. SCF uses "10000" for both the user ID and user group ID by default to manipulate your CFS file system. Set the file owner and corresponding group permission as needed and ensure that your CFS file system has the required permission. For more information, 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.

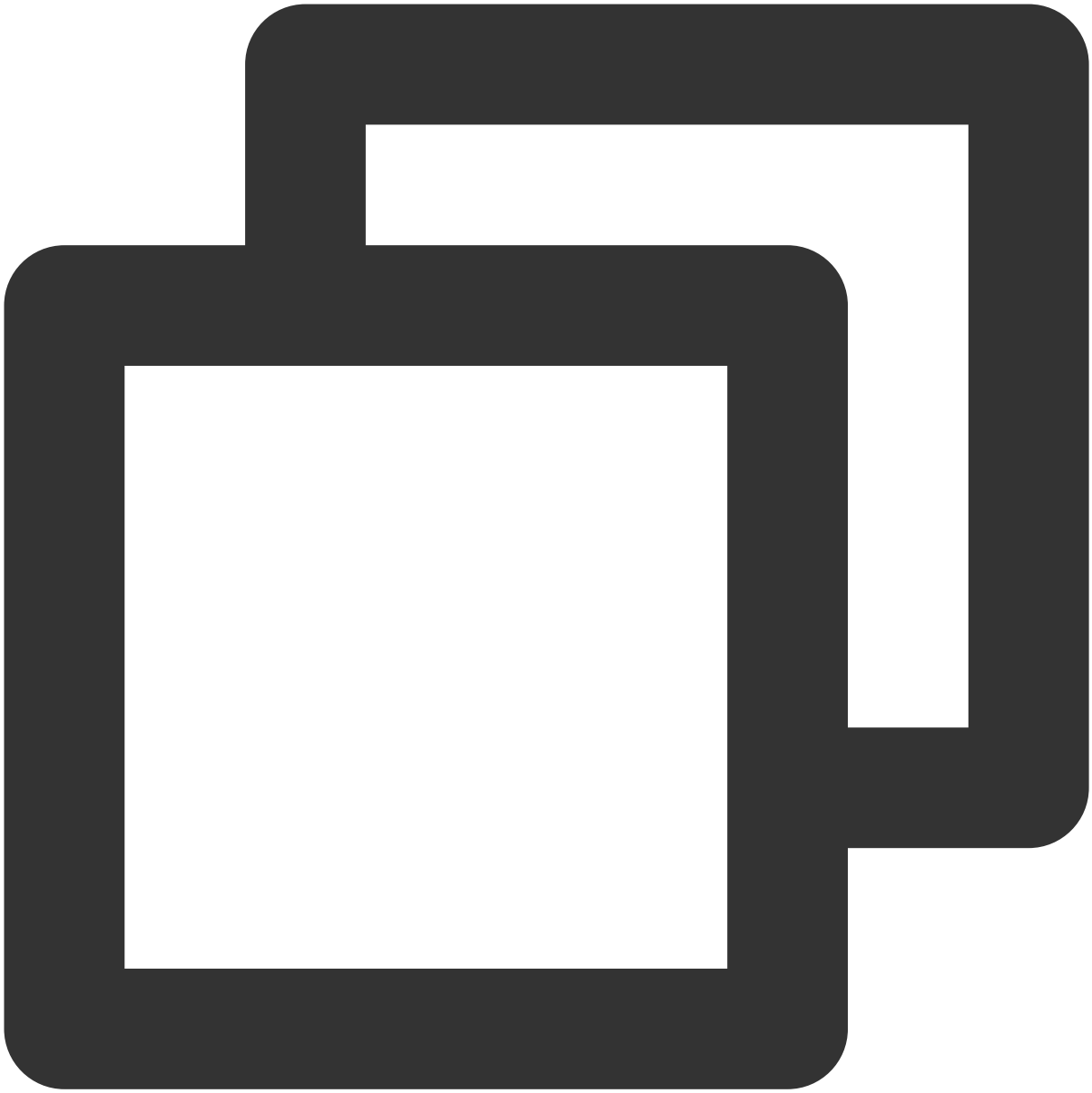
Local directory: Mount target of the local file system. You can use a subdirectory in the `/mnt/` directory to mount the CFS file system.

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.

You can run the following function code to start using the CFS file system.



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

Performance test for using a CFS file system on SCF

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

Using CFS Turbo on TKE

Last updated：2024-01-22 22:15:48

Overview

This document describes how to integrate CFS Turbo with a Tencent 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 client, see Using CFS Turbo on Linux Clients.

Directions

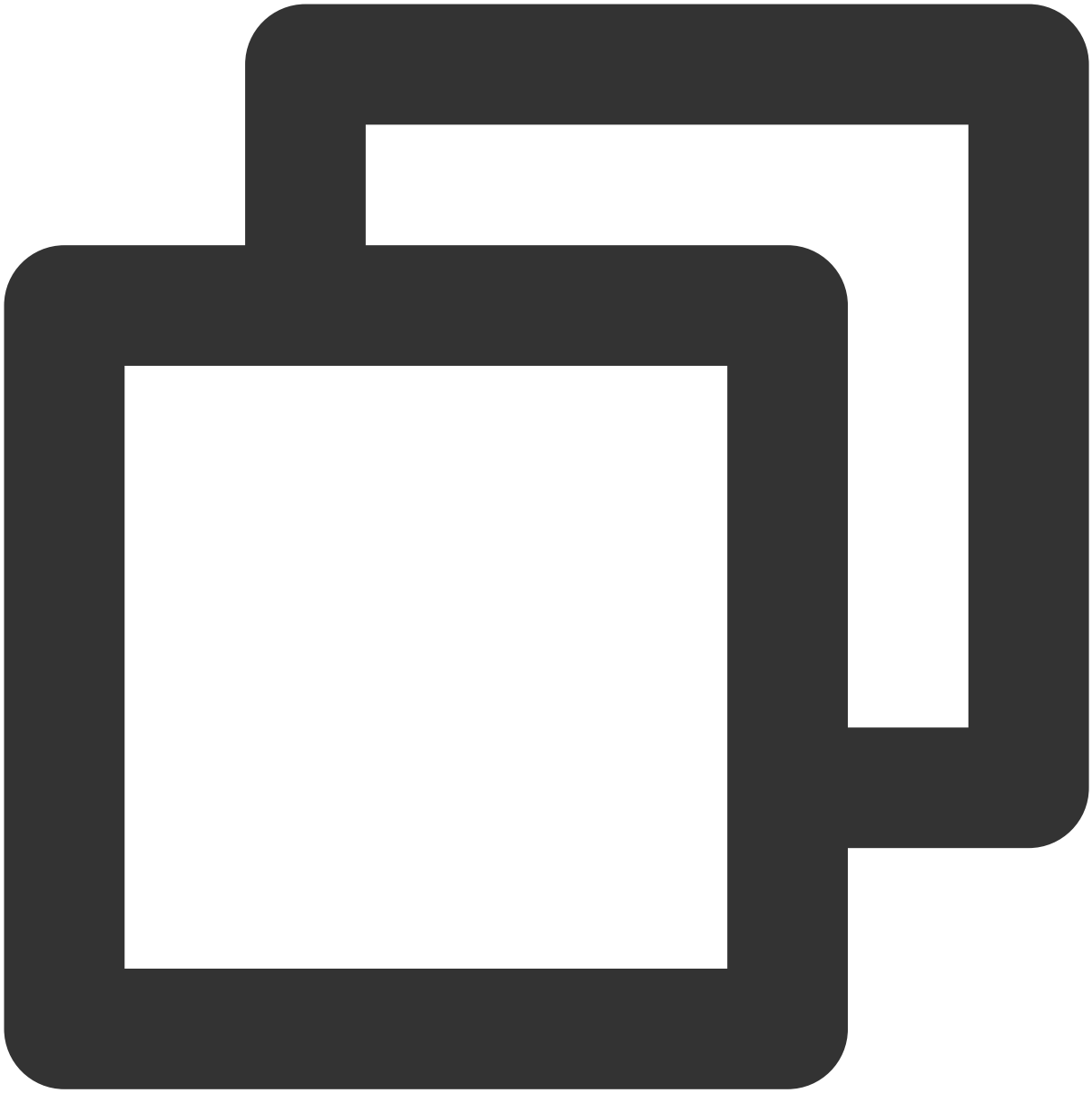
Using kubectl to connect to a cluster

Use kubectl to connect to a TKE cluster. For more information, see Connecting to a Cluster.

Creating a Pod for mounting Turbo using YAML files

1. Read the Turbo plugin description.
2. Go to the `kubernetes-csi-tencentcloud/deploy/cfsturbo/kubernetes/` directory and upload `csi-node-rbac.yaml`, `csi-node.yaml`, and `csidriver-new.yaml` files to the kubectl management node.
3. Go to the `kubernetes-csi-tencentcloud/deploy/cfsturbo/examples/` directory and download the `pv.yaml`, `pvc.yaml`, and `pod.yaml` sample files.
4. Modify the `pv.yaml`, `pvc.yaml`, and `pod.yaml` files based on the PV, PVC, and Pod attributes, such as name and image address.

Below are the sample YAML files:

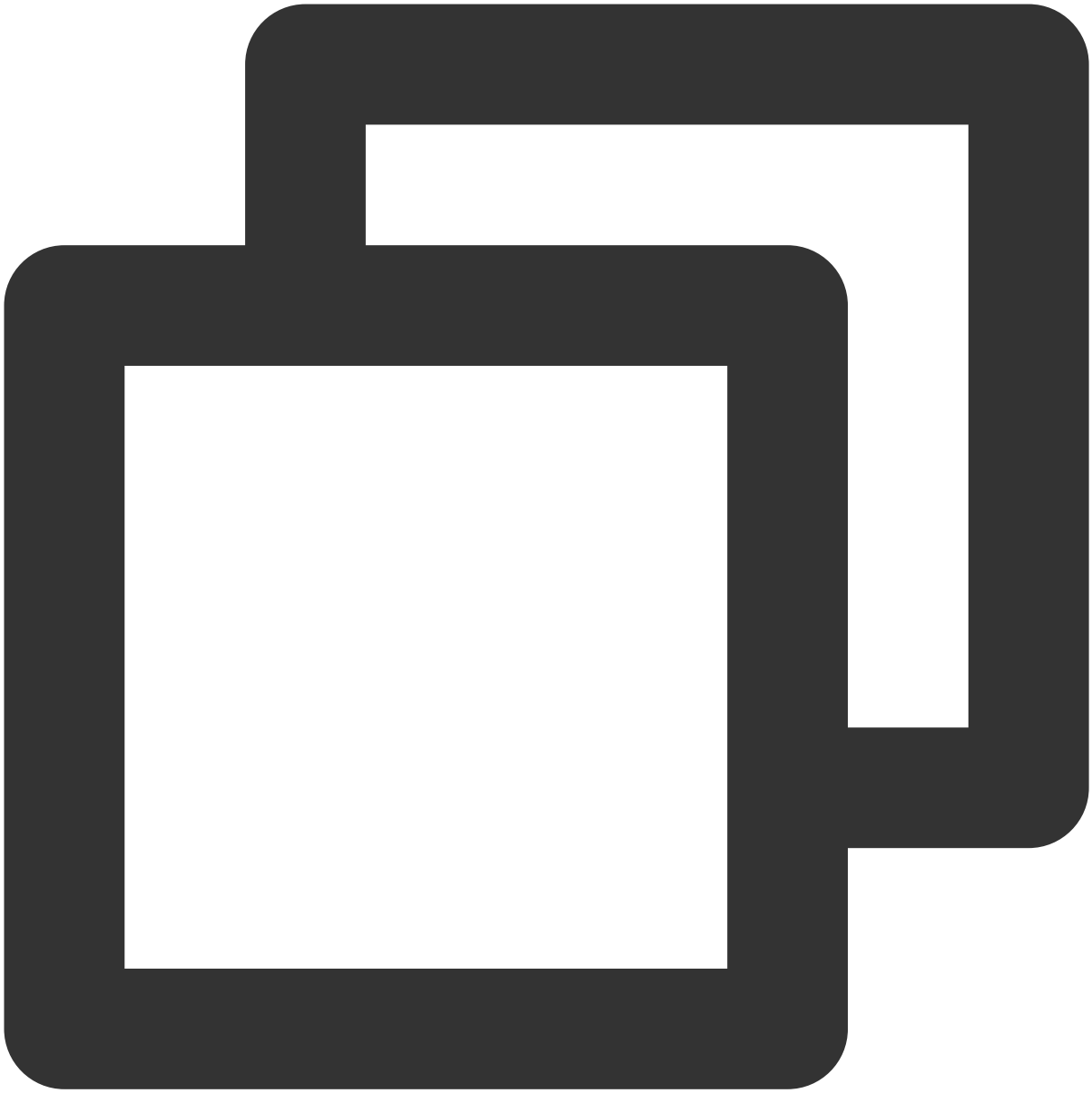


```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: csi-cfsturbo-pv
spec:
  accessModes:
  - ReadWriteMany
  capacity:
    storage: 10Gi
  csi:
    driver: com.tencent.cloud.csi.cfsturbo
```

```
    # volumeHandle in PV must be unique, use pv name is better
    volumeHandle: csi-cfsturbo-pv
    volumeAttributes:
      # cfs turbo proto
      proto: lustre
      # cfs turbo rootdir
      rootdir: /cfs
      # cfs turbo fsid (not cfs id)
      fsid: d3dcc487
      # cfs turbo server ip
      host: 10.0.1.16
      # cfs turbo subPath
      path: /
  storageClassName: ""
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: csi-cfsturbo-pvc
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
  # You can specify the pv name manually or just let Kubernetes bind the pv and pvc
  volumeName: csi-cfsturbo-pv
  # cfsturbo only supports static provisioning, the StorageClass name should be emp
  storageClassName: ""
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: csi-cfsturbo-pod
  name: csi-cfsturbo-pod
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: csi-cfsturbo-pod
  template:
    metadata:
      labels:
        k8s-app: csi-cfsturbo-pod
    spec:
      containers:
```

```
      - image: nginx
        name: csi-cfsturbo-pod
        volumeMounts:
          - mountPath: /csi-cfsturbo
            name: csi-cfsturbo
    volumes:
      - name: csi-cfsturbo
        persistentVolumeClaim:
          # Replaced by your pvc name.
          claimName: csi-cfsturbo-pvc
```

Below is the sample command for mounting:



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

Below are key parameters:

proto:lustre: Keep this parameter unchanged.

roodir:/cfs: Keep this parameter unchanged.

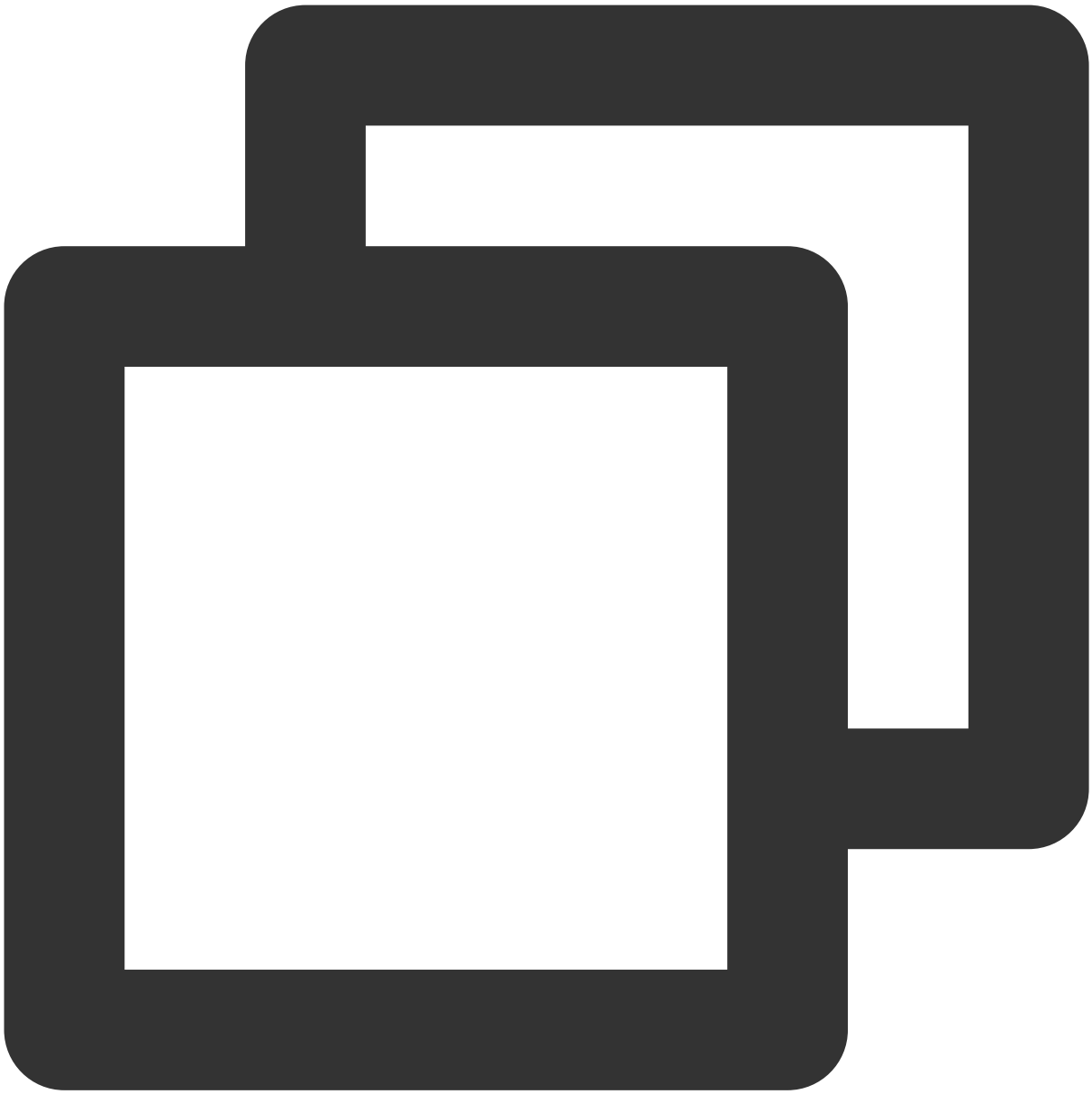
fsid:d3dcc487: Here, `fsid` is not the `CFSID` , and you need to enter the information in the mount directory.

host:10.0.1.16: IP of the mount point.

path: You can adjust it based on the target subdirectory. To directly mount the root directory, enter "/".

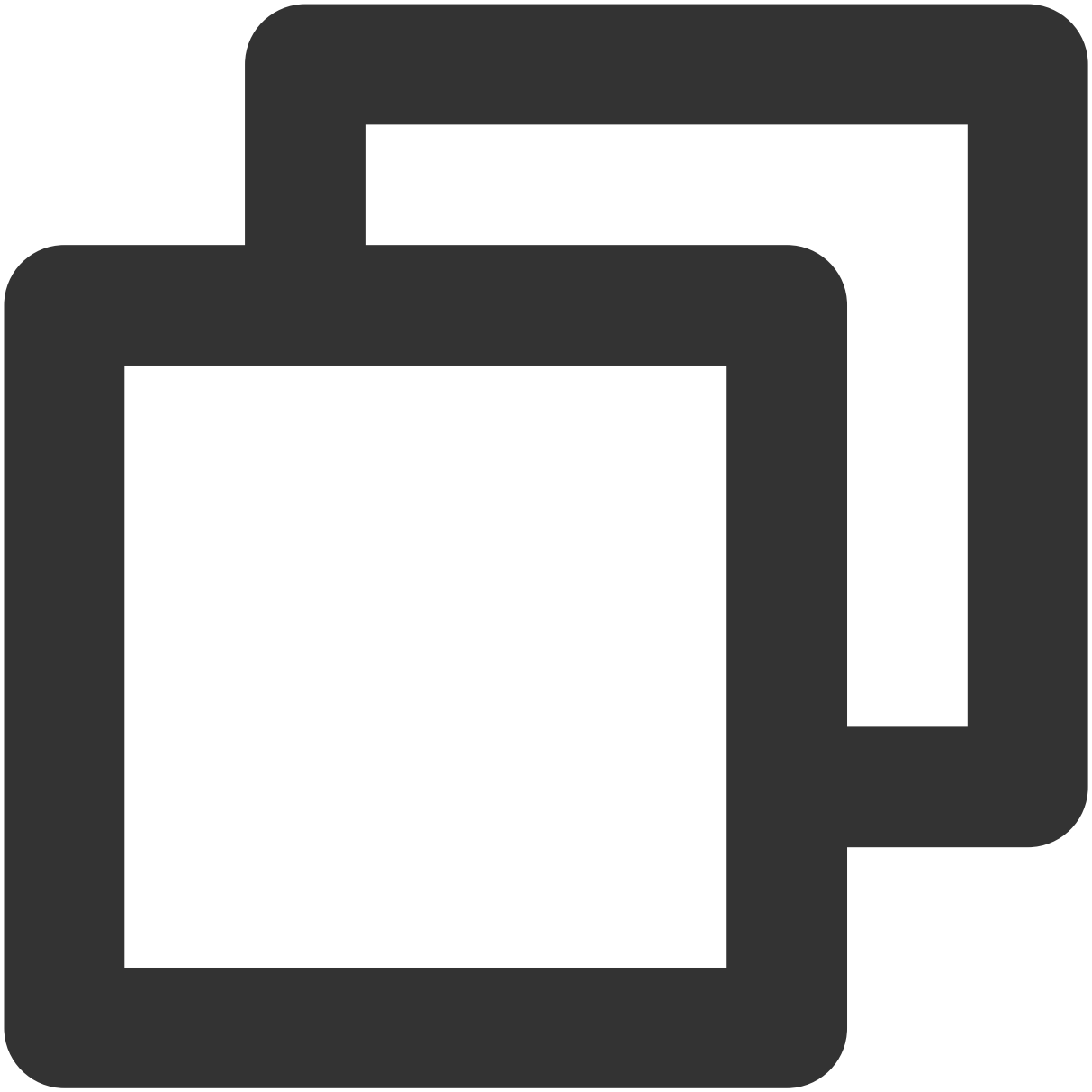
5. Run the following commands in sequence in the directory where the script is uploaded:

Configure RBAC and CSI plugins.



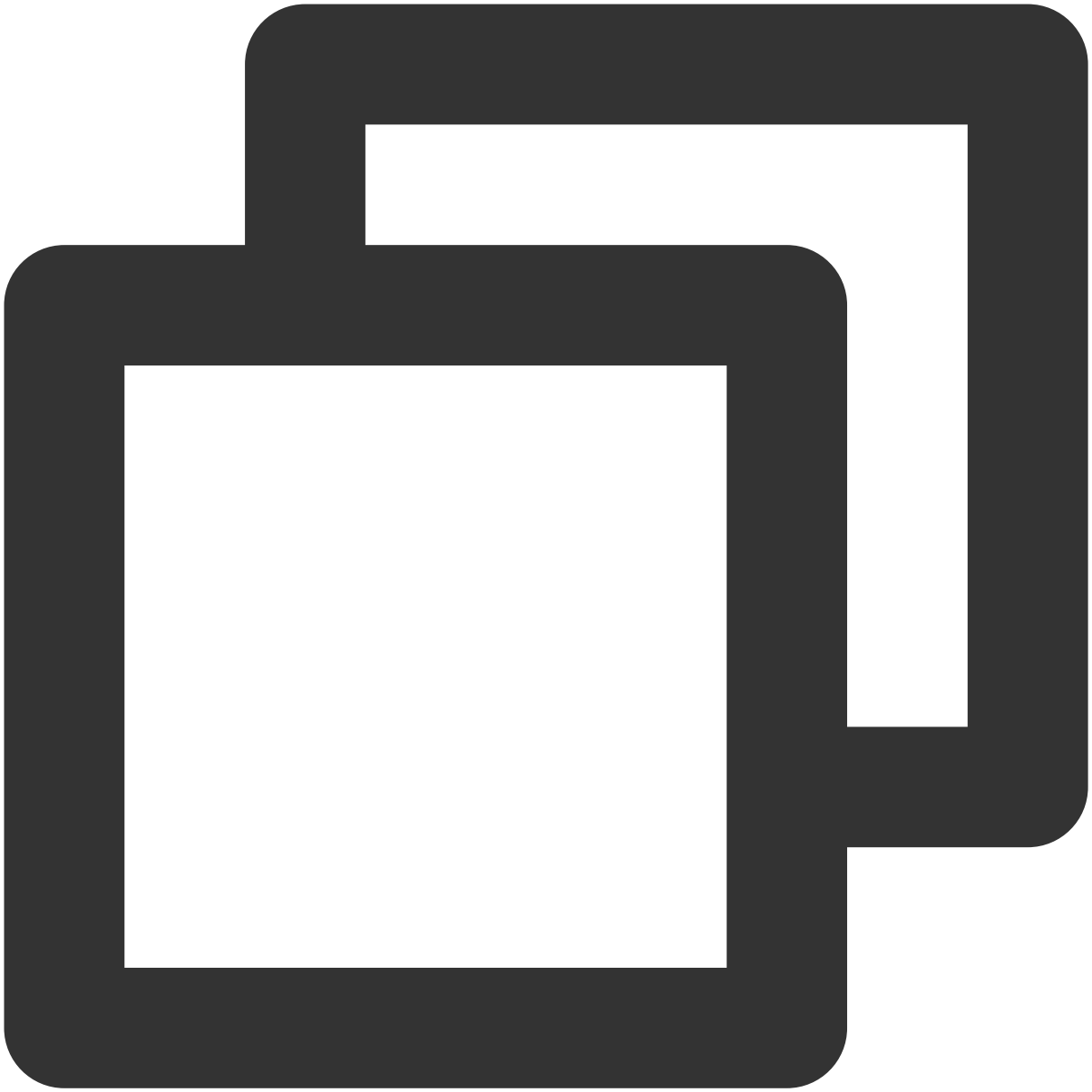
```
kubectl apply -f csi-node-rbac.yaml && kubectl apply -f csidriver-new.yaml && kubec
```

Create PV, PVC, and Pod:



```
kubectl create -f pv.yaml && kubectl create -f pvc.yaml && kubectl create -f pod.ya
```

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
nginx2   1/1     Running   0          36s   10.0.0.8    192.168.0.2   <none>
```

If the value of `STATUS` is `ContainerCreating`, the creation failed. You can view the failure reason based on the events in the TKE console.

Using CFS Turbo on TKE Serverless Cluster

Last updated：2024-01-22 22:15:47

Overview

You can mount a Cloud File Storage (CFS) Turbo storage for TKE Serverless Cluster. 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 classes, see Storage Classes and Performance.

Prerequisites

A TKE Serverless Cluster of v1.14 or later has been created.

Directions

Creating a file system

Create a CFS Turbo file system as instructed in Creating a File System and Mount Target.

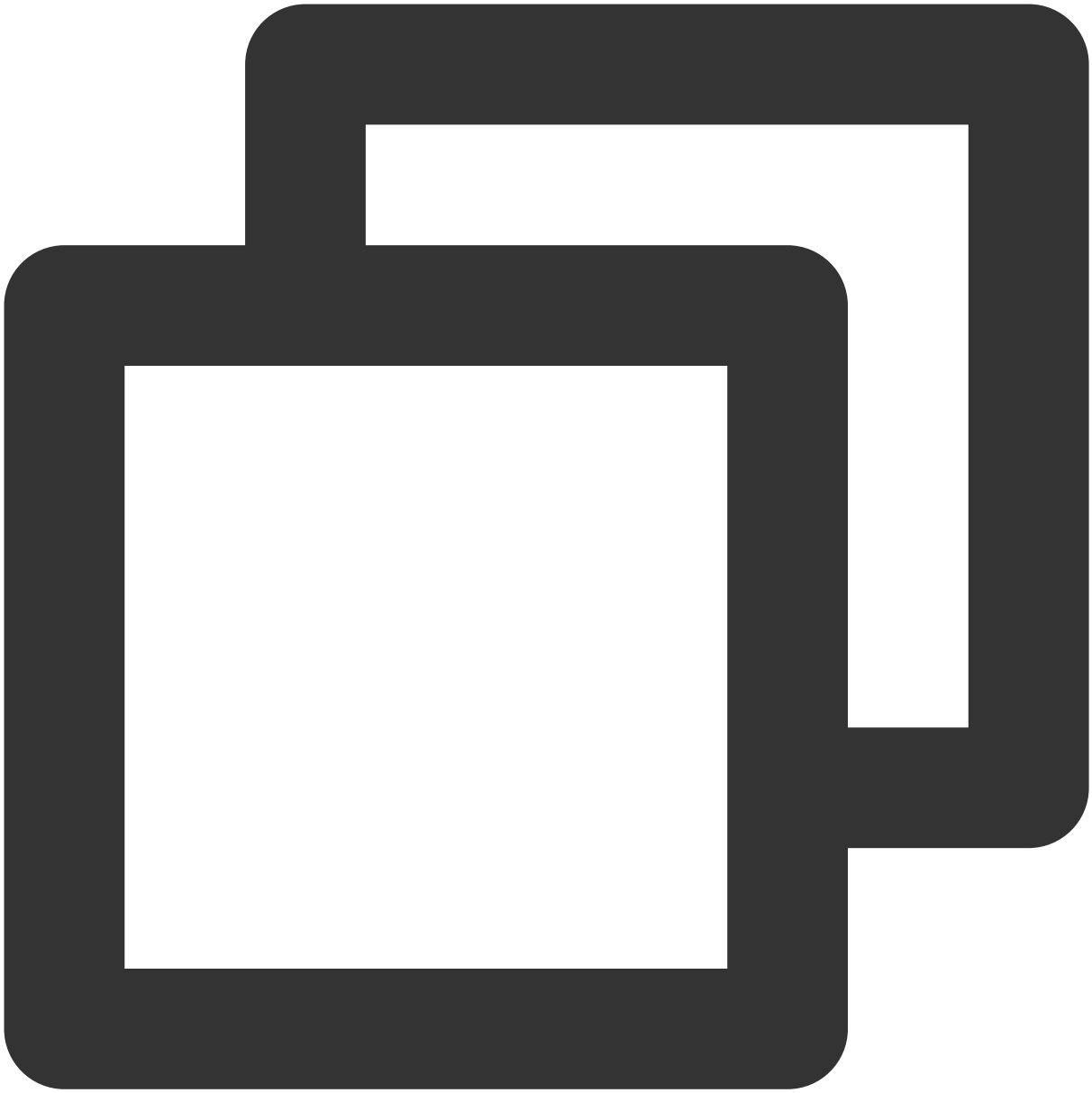
Note:

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 the associated instance in the file system mount target information.

Deploying a Node Plugin

Step 1. Create a csidriver.yaml file

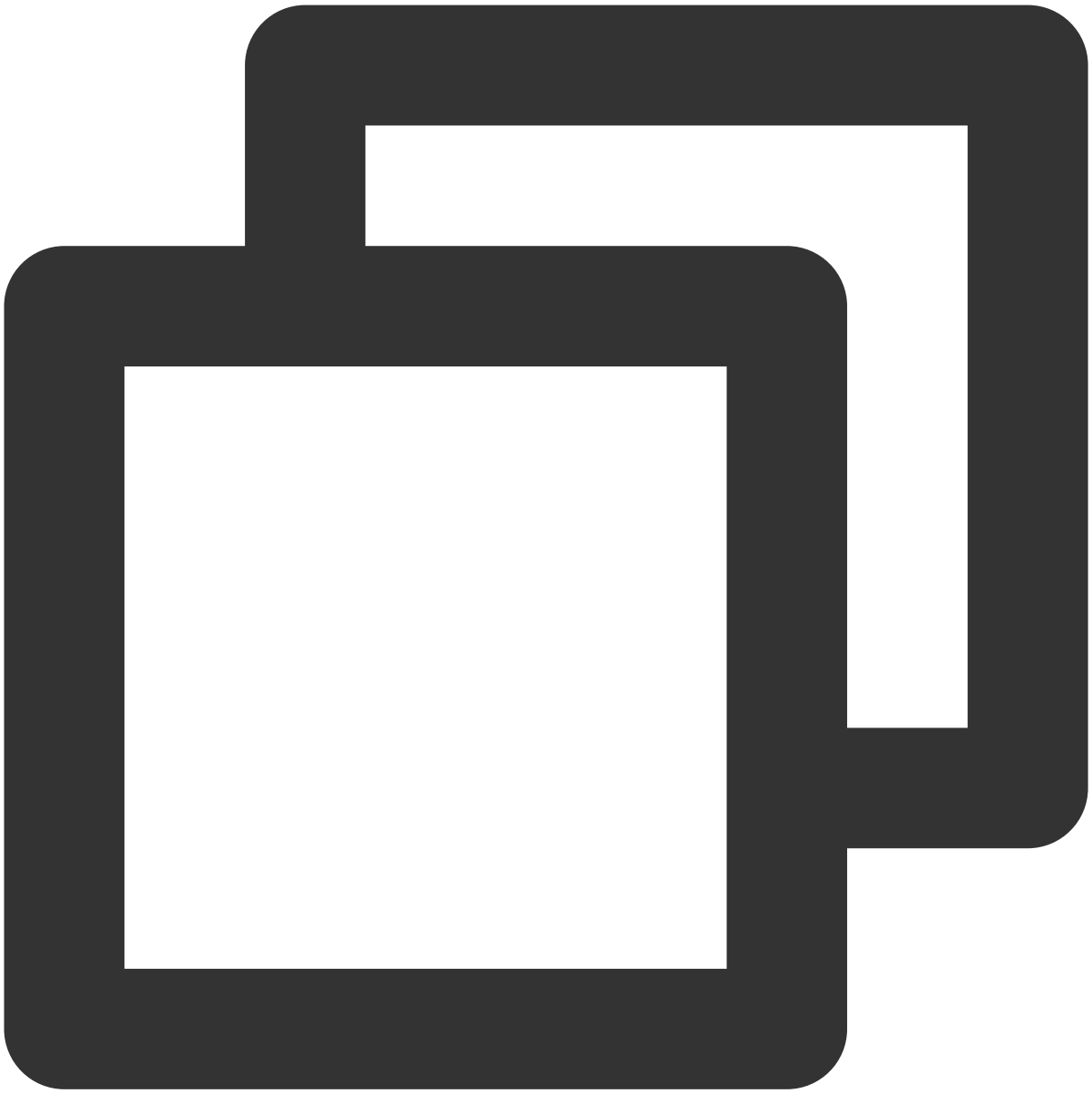
Here is an example of a csidriver.yaml file:



```
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

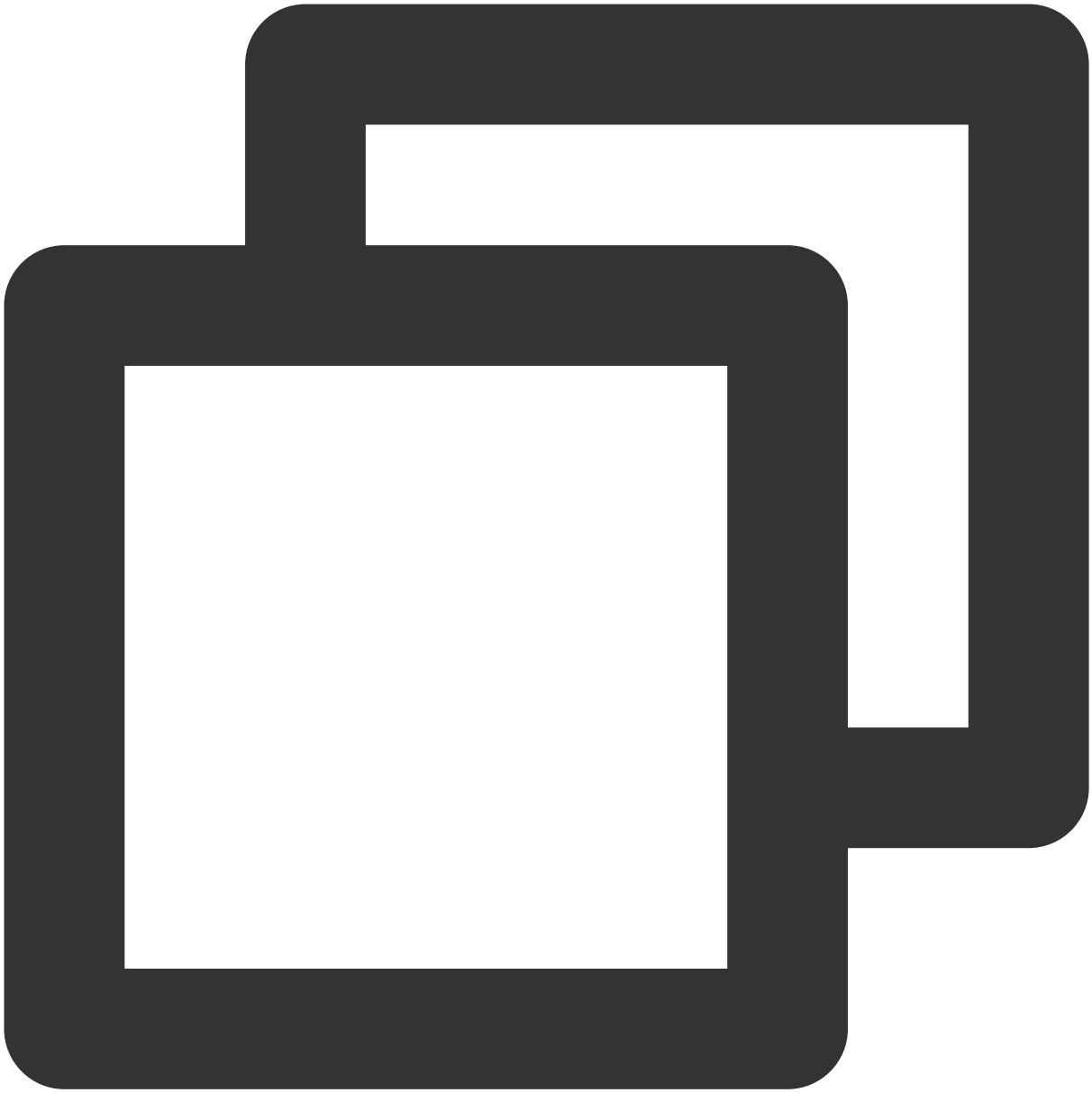
Run the following command to create a CSI driver:



```
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: 10.0.0.1
      # Set host to the actual mount target IP
      fsid: d3816815
      # Set fsid to the actual FSID
      path: /
      # CFS Turbo subdirectory
  storageClassName: ""
```

Parameters:

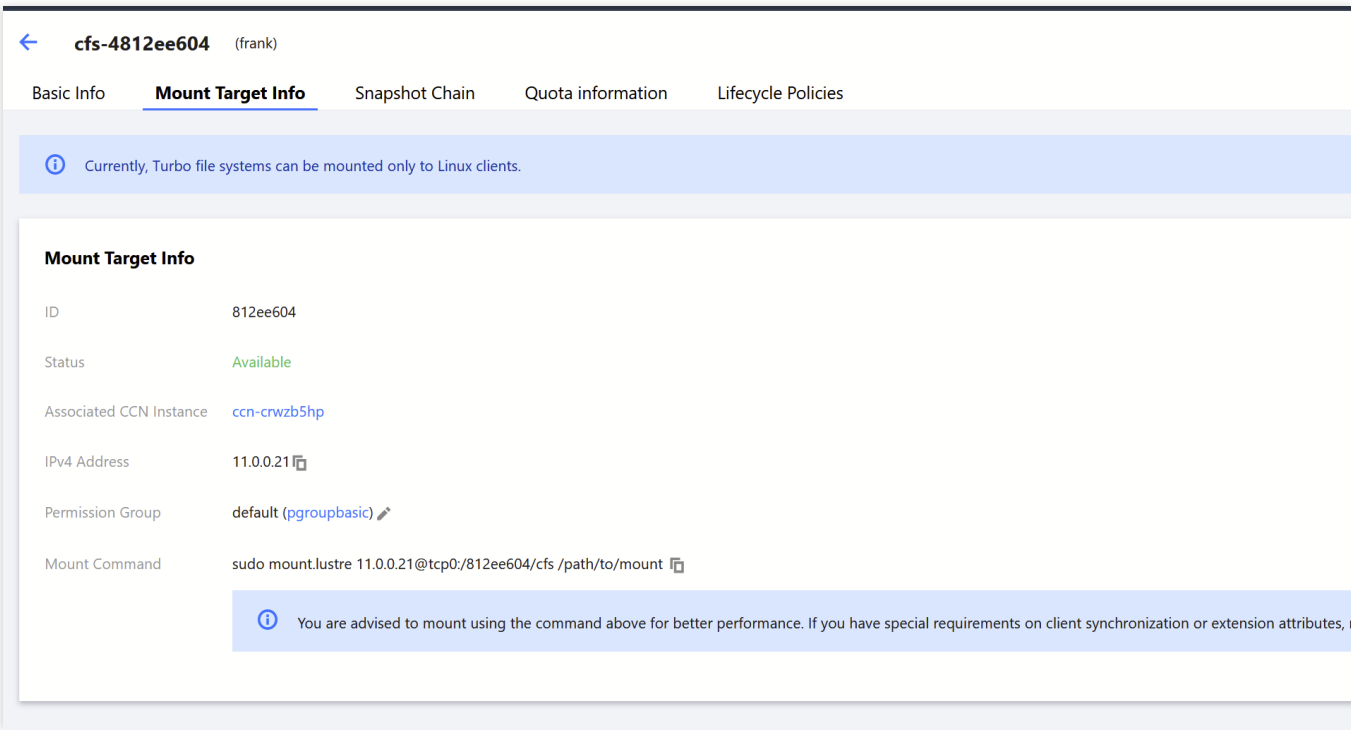
metadata.name: The name of the created PV.

spec.csi.volumeHandle: It must be consistent with the PV name.

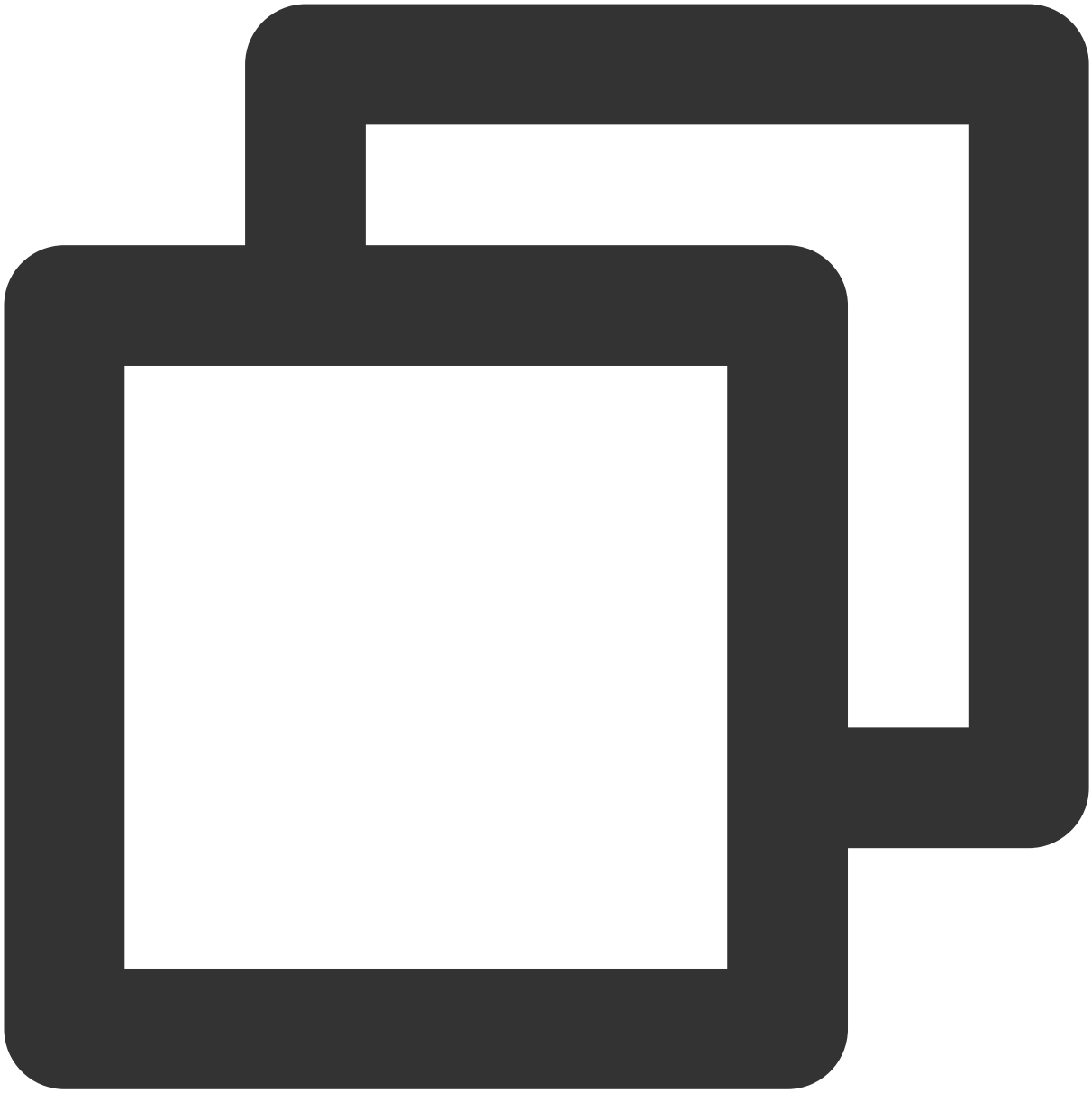
spec.csi.volumeAttributes.host: The IP address of the file system. You can check it in the file system mount target information.

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 after "tcp0:/" and before "/cfs" in the mount command.

spec.csi.volumeAttributes.path: File system subdirectory, which defaults to "/" if not specified. (To improve mounting performance, the plugin backend actually locates the "/cfs" directory for "/"). If you want to mount a subdirectory, ensure that the subdirectory exists in the file system "/cfs". After a subdirectory is mounted, the workload will not be able to access its upper-level directories. For example, if you want to set "path: /test", ensure that "/cfs/test" exists in the file system.



Step 2. Create a PVC based on the following template to bind the PV



```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-cfsturbo
spec:
  storageClassName: ""
  volumeName: pv-cfsturbo
  accessModes:
  - ReadWriteMany
  resources:
    requests:
```

```
      storage: 10Gi
```

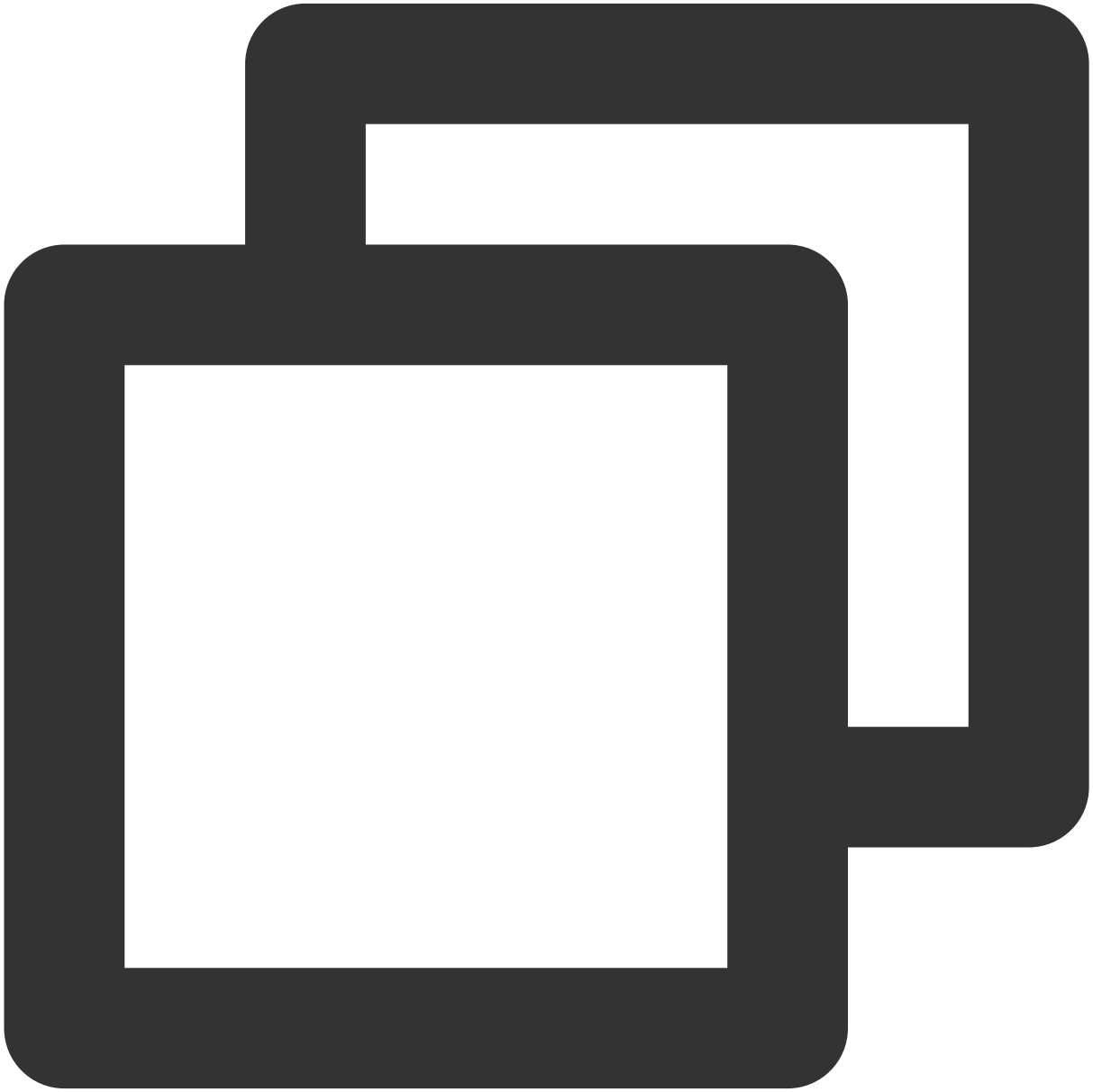
Parameters:

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 and mount 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
```

Selecting a Network for Turbo CFS

Last updated：2024-01-22 22:15:48

CFS Turbo is Tencent Cloud's high-performance parallel file system. Compared with traditional general CFS, it supports CCN and VPC networks (with respective pros and cons) on the client and server. This document describes the two network types to help you select a more suitable one for your business. If you use CCN, we recommend you create Turbo file systems based on CCN; otherwise, you can select VPC for easier use, if applicable.

CCN

Overview

Specified CIDR blocks are assigned to Turbo file systems, and CCN capabilities are leveraged to connect the VPC and storage server network, implementing the interaction between compute instances and storage.

Pro	Con
CIDR block planning is separately conducted for storage systems for more efficient and convenient management of security groups. Turbo CFS has its own CIDR block to reserve sufficient IPs for future scaling without limits. Turbo CFS can be accessed more easily across VPCs. IPs of the existing VPC are not occupied.	CCN is required for network connection. If CCN is not used, a new component needs to be introduced, which is quite complicated.

Best practices

You can select 10/11/30/172/192 CIDR blocks to create a CCN instance. It is recommended that Turbo CFS be assigned 11/30 blocks, which are server-side blocks and do not occupy business IPs.

VPC

Overview

IPs are mapped to the existing VPC on the storage server for mount and access.

Pro	Con
It is the easiest to use and is similar to general CFS. No new components need to be introduced in this simple solution.	During large scaling, subnet IPs may be insufficient. VPC IPs will be occupied, and the storage is bound to the VPC, which hinders network isolation.

Copying Data

Last updated：2024-01-22 22:15:47

Overview

During your use of Cloud File Storage (CFS), you may need to copy data in many scenarios. This document provides recommended solutions for fast data copying in Linux operating systems.

Scenario 1: Data synchronization between CFS file systems, between different directories in a file system, and between a file system and a cloud disk

Data copying is required in many cases. The above three cases in scenario 1 are all about data copying between different directories from the perspective of a Linux operating system. Their operation methods are similar. You can use the `cp` command for simple and fast copying. However, if there are numerous files, the copying will be very slow due to the single-threaded running mode of `cp`. In this case, we recommend you use the method provided in this document for copying to achieve concurrent acceleration.

Scenario 2: Data synchronization between a file system and COS

When the public network is required for storage access or data import, it is recommended to use COS as a transit. You can use various basic tools provided by COS for data uploading and downloading. The COSBrowser and COSCMD tools are recommended.

Note

The directions apply to scenario 1. For scenario 2, refer to COSBrowser and COSCMD.

Prerequisites

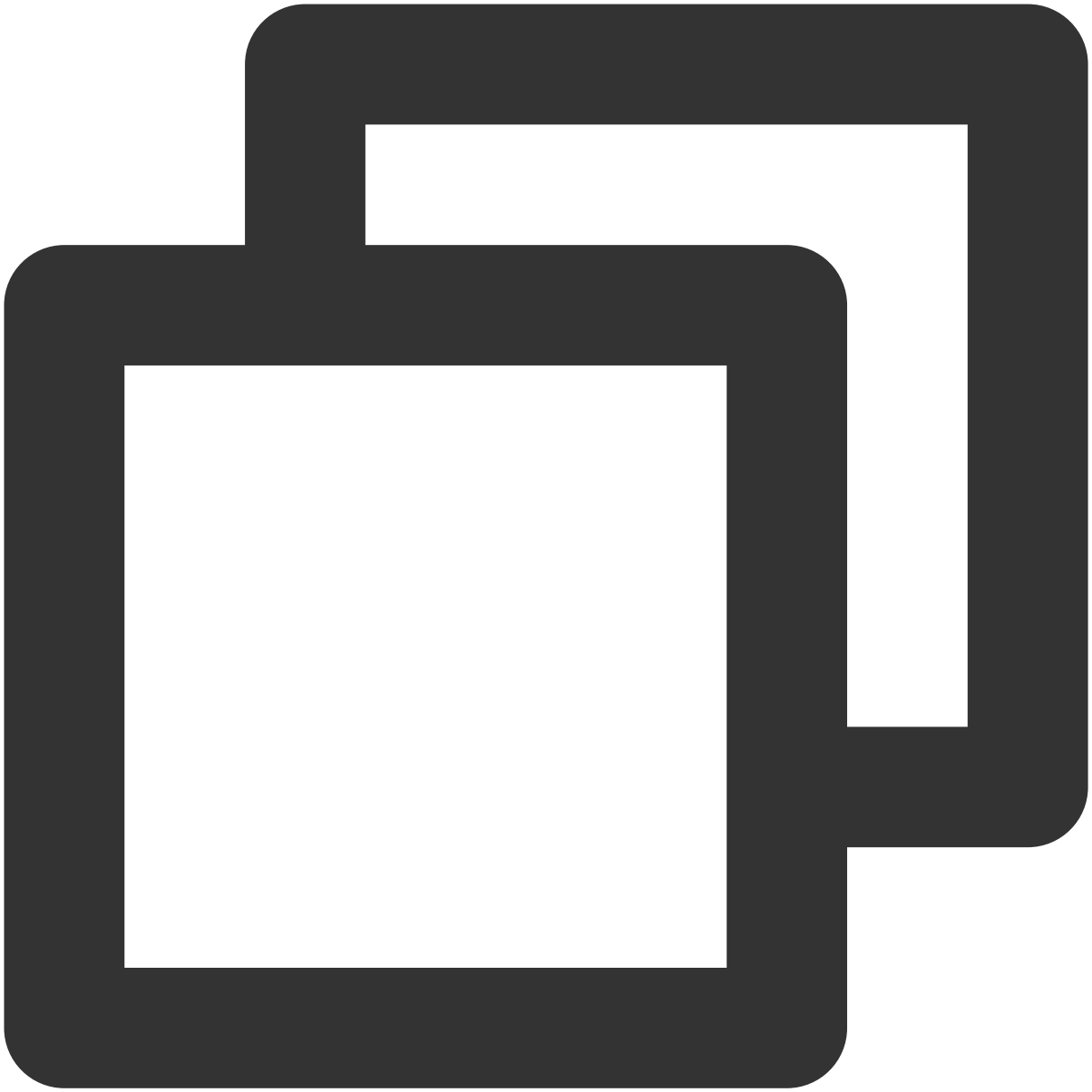
The source and destination directories for copying exist on the CVM.

Note:

CFS supports cross-VPC access. You can connect multiple VPCs through Cloud Connect Network (CCN) and then mount and access a file system.

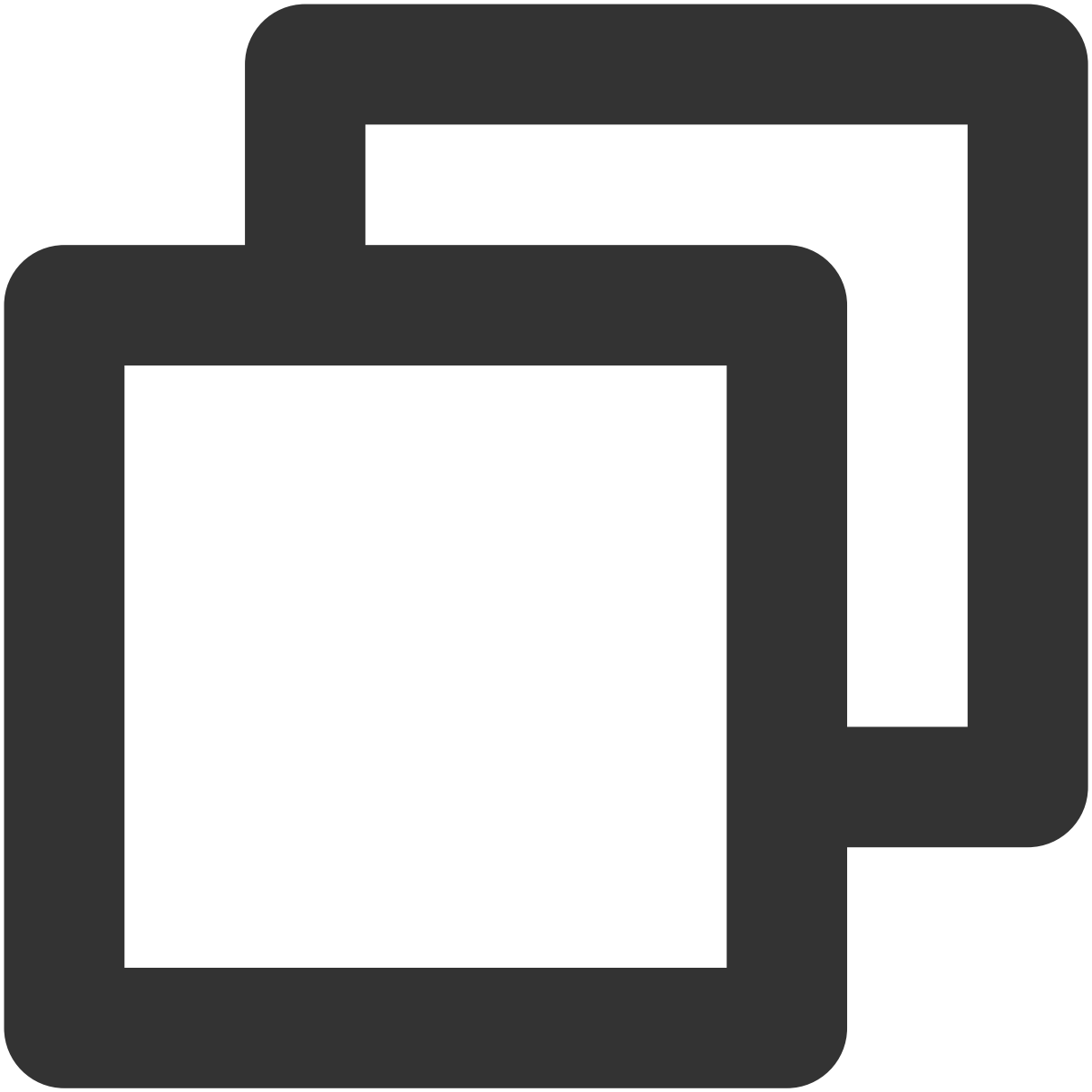
Directions

1. Download and install the Rclone tool.



```
wget https://downloads.rclone.org/v1.53.4/rclone-v1.53.4-linux-amd64.zip --no-check
unzip rclone-v1.53.4-linux-amd64.zip
chmod 0755 ./rclone-*/rclone
cp ./rclone-*/rclone /usr/bin/
rm -rf ./rclone-*
```

2. Run the following command to copy data:



```
rclone copy /mnt/src /mnt/dst -Pvv --transfers 32 --checkers 64 --copy-links --crea
```

This tool does not copy metadata information such as owner, ctime, and atime, by default. If you need to copy metadata information, use the `rsync` command.

Note:

The parameters in the command are described as follows (set `transfers` and `checkers` as needed based on the system specifications):

transfers: The number of concurrent file transfers (recommended up to twice the number of cores).

checkers: The number of concurrent scans of local files (recommended up to twice the number of cores).

P: Real-time display of the progress of data copying (the progress is refreshed every 500 ms. If `P` is not added to the command, the progress is refreshed every minute).
copy-links: Soft links for copying.
vv: Print copying logs.
create-empty-src-dirs: Copy empty directories.
3. After data copying is complete, you can view logs to check the results for different files.

CFS Storage Performance Testing

Last updated：2024-05-23 16:40:02

This document mainly introduces how to perform performance testing on the CFS File System in a reasonable manner.

Key Performance Metrics Instructions

Latency: The duration it takes to process read and write requests, measured in milliseconds. Typically, this is benchmarked using a 1 MB file with a single-stream (single-threaded) 4 K small I/O for testing.
IOPS: The number of data blocks read or written per second, measured in operations per second. In the industry, it is commonly benchmarked using a 100 MB file through concurrent (multi-host, multi-threaded) 4 K small I/O for testing.
Throughput: The amount of data read or written per second, measured in GiB/s or MiB/s. Typically, this is benchmarked using a 100 MB file with concurrent (multi-host, multi-threaded) 1 M large I/O for testing.

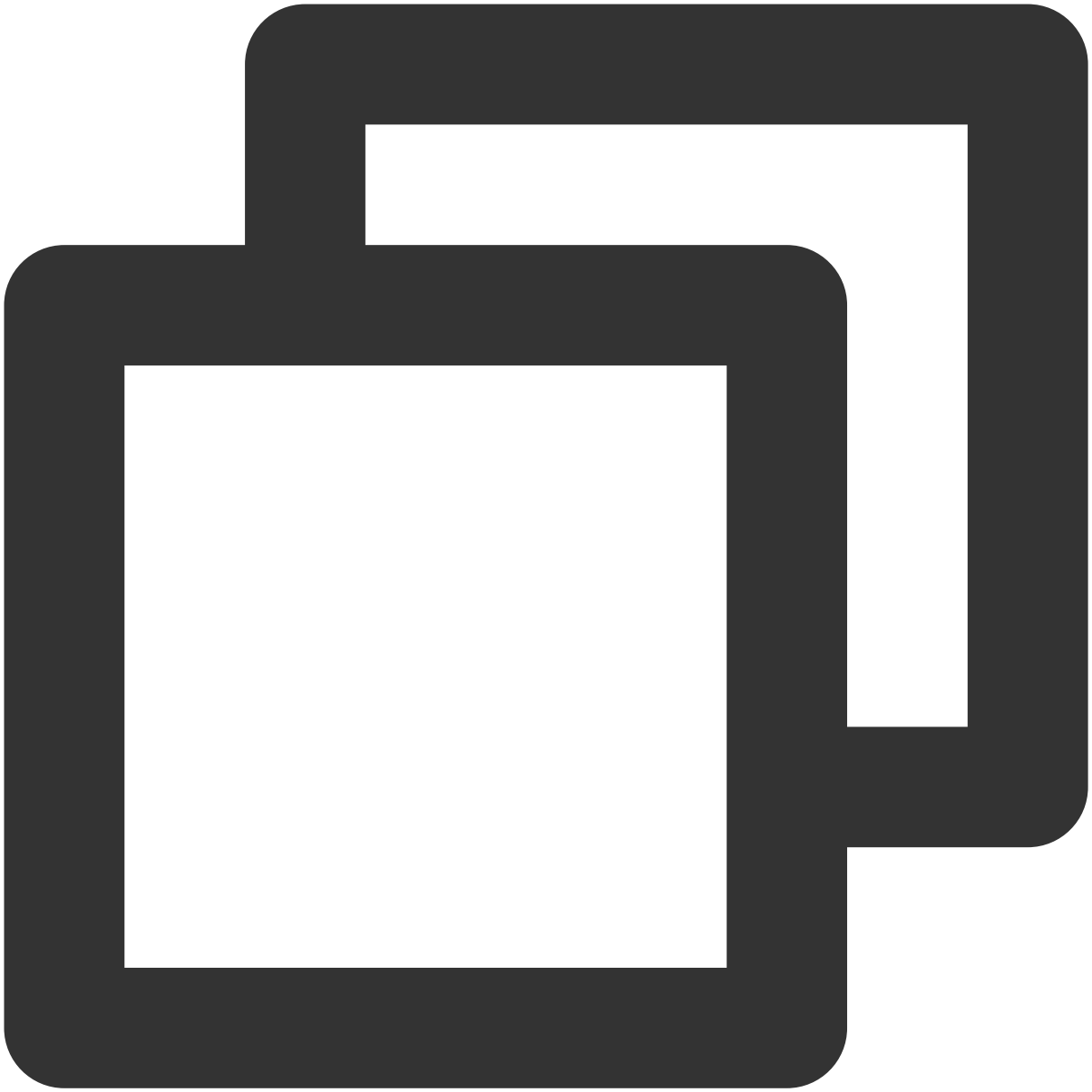
Notes

CFS's provided Performance Specifications , excluding latency parameters, all require a certain scale of machines with a sufficient number of cores to conduct concurrency stress testing to reach their maximum values. Typically, preparing 16 cloud servers with more than 32C as clients can meet the needs of most testing scenarios. For other stress testing requirements, cloud servers can be configured according to actual situations.
When stress testing CFS's General Standard and General Performance types, due to the server-side cache acceleration feature, read performance may exceed the standard values when cache hits occur. This is an expected phenomenon.
During performance stress testing, especially latency tests, it is necessary to ensure that the CVMs (clients) and CFS are located within the same availability zone. Performance results from cross-availability zone tests will significantly differ from standard values and should be avoided as much as possible.

Directions

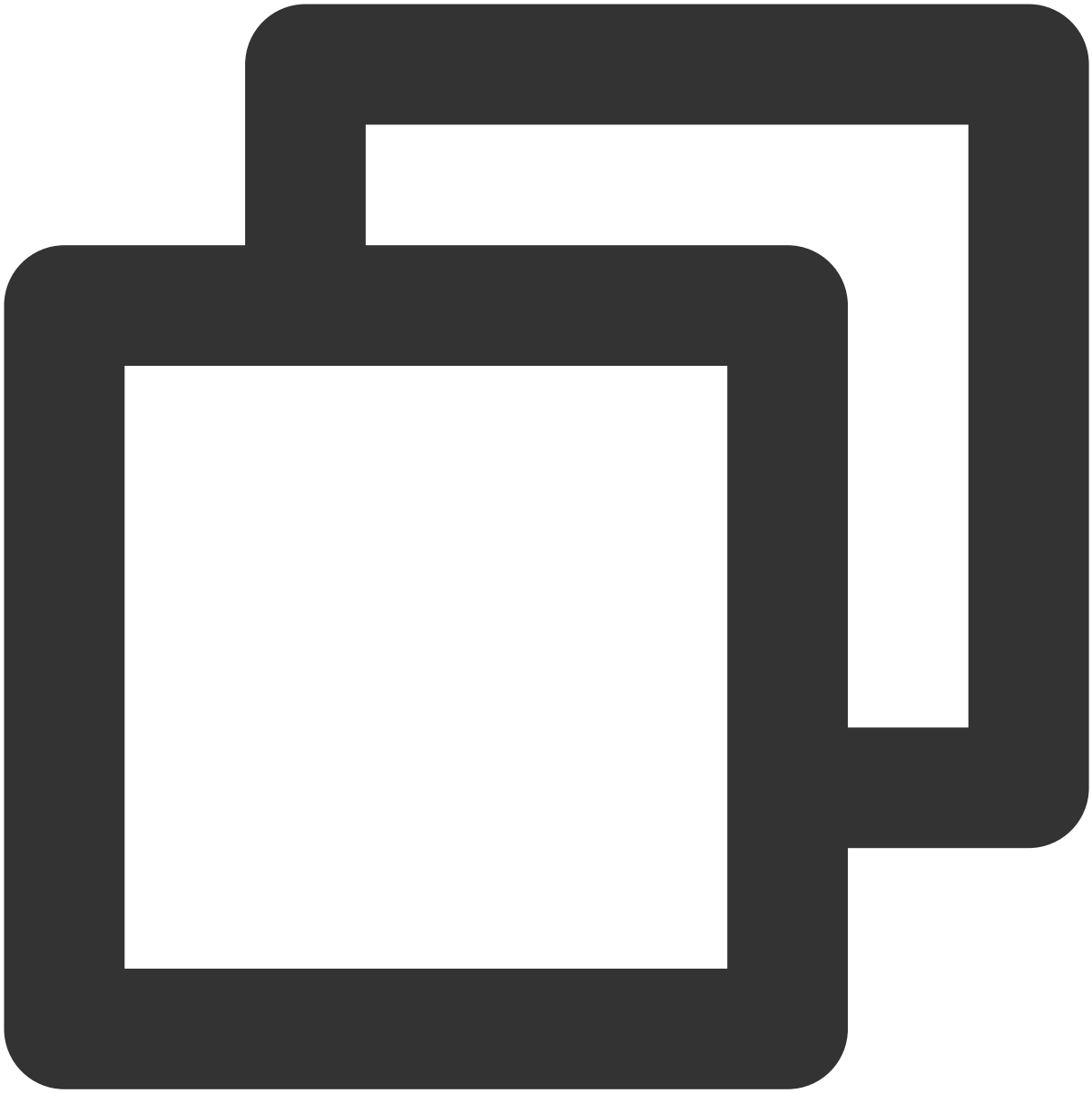
Installing Stress Testing Software

CentOs/Tlinux:



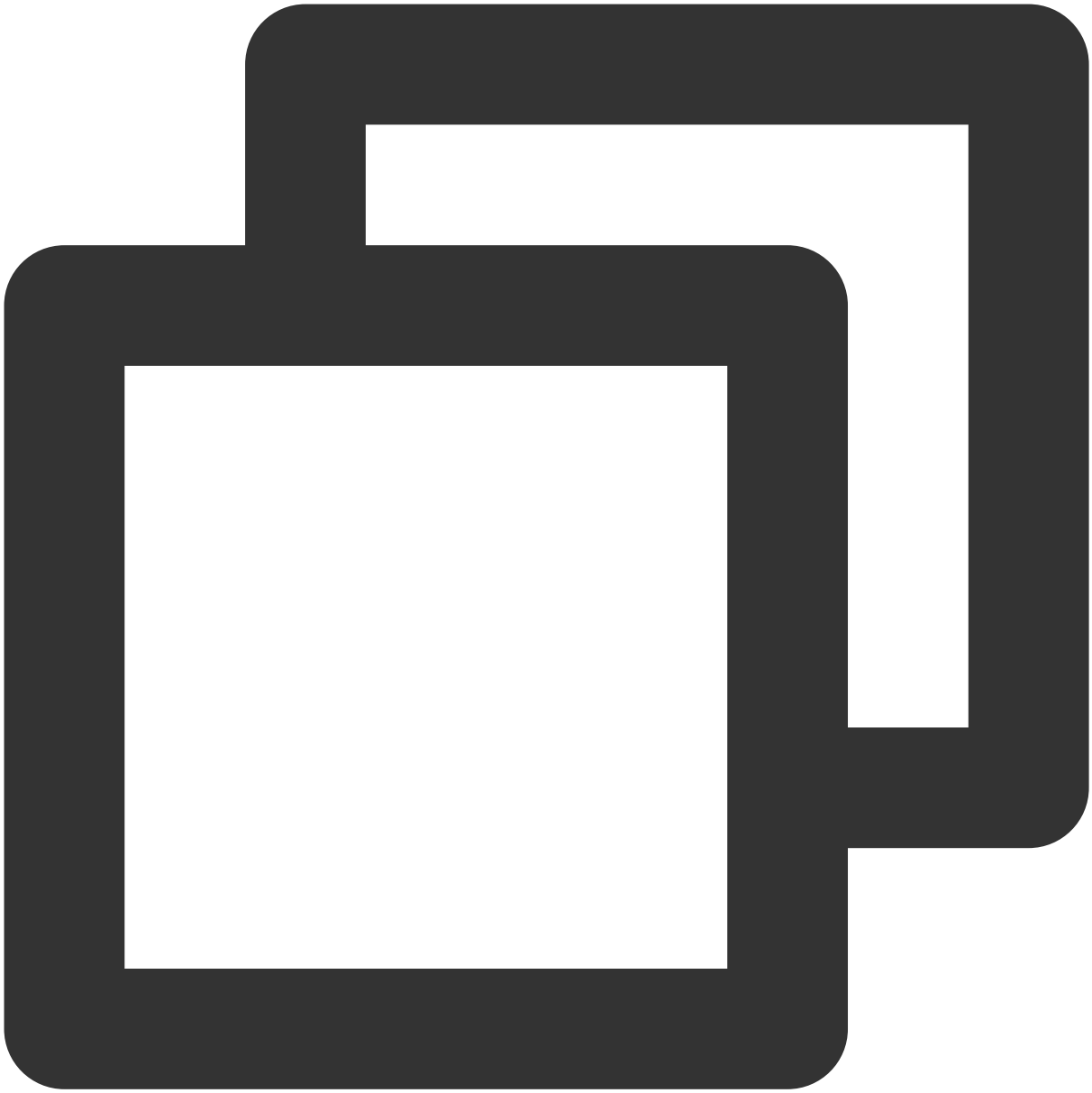
```
sudo yum install fio
```

Ubuntu/Debian



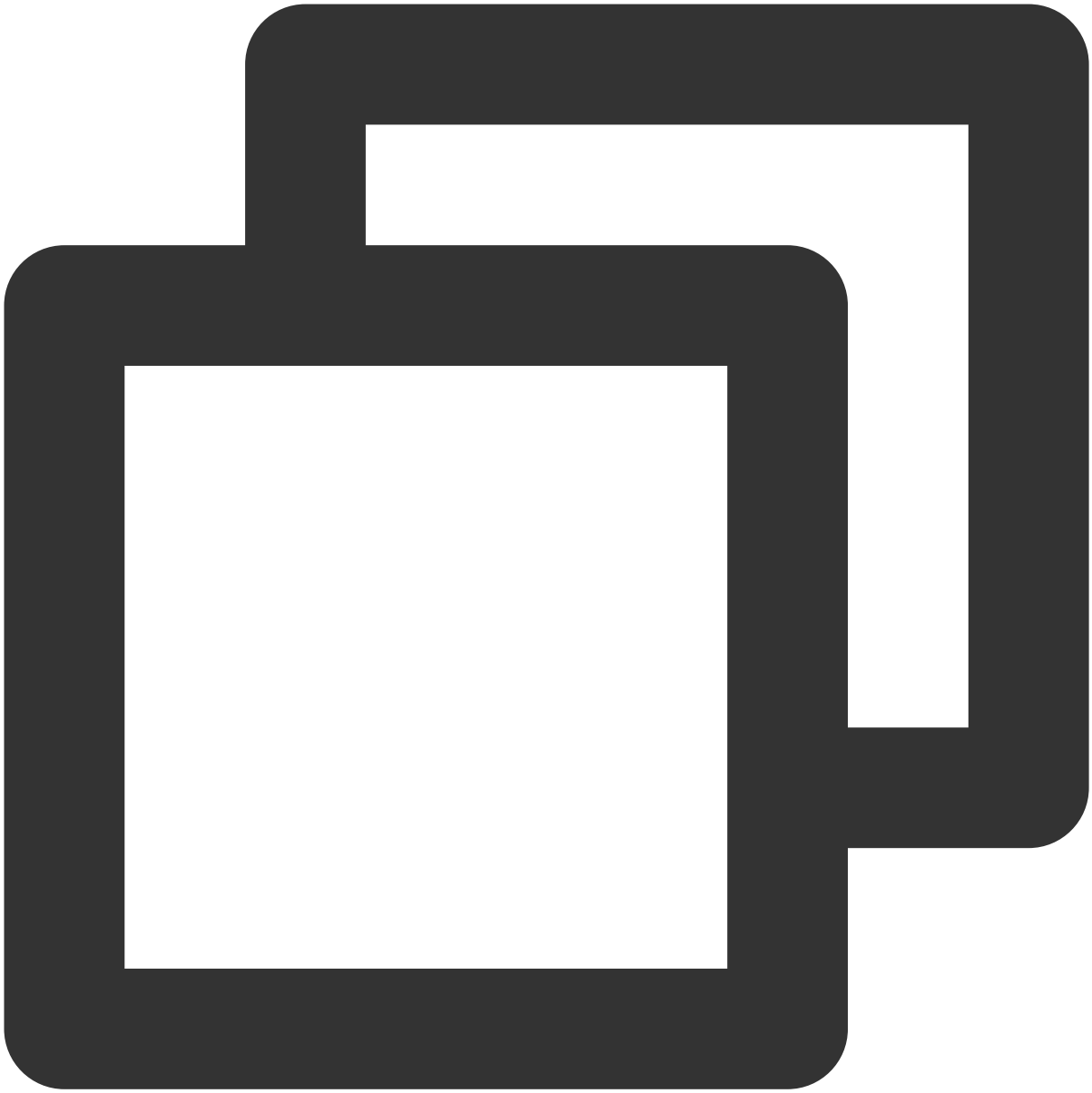
```
sudo apt-get install fio
```

Read Latency Testing



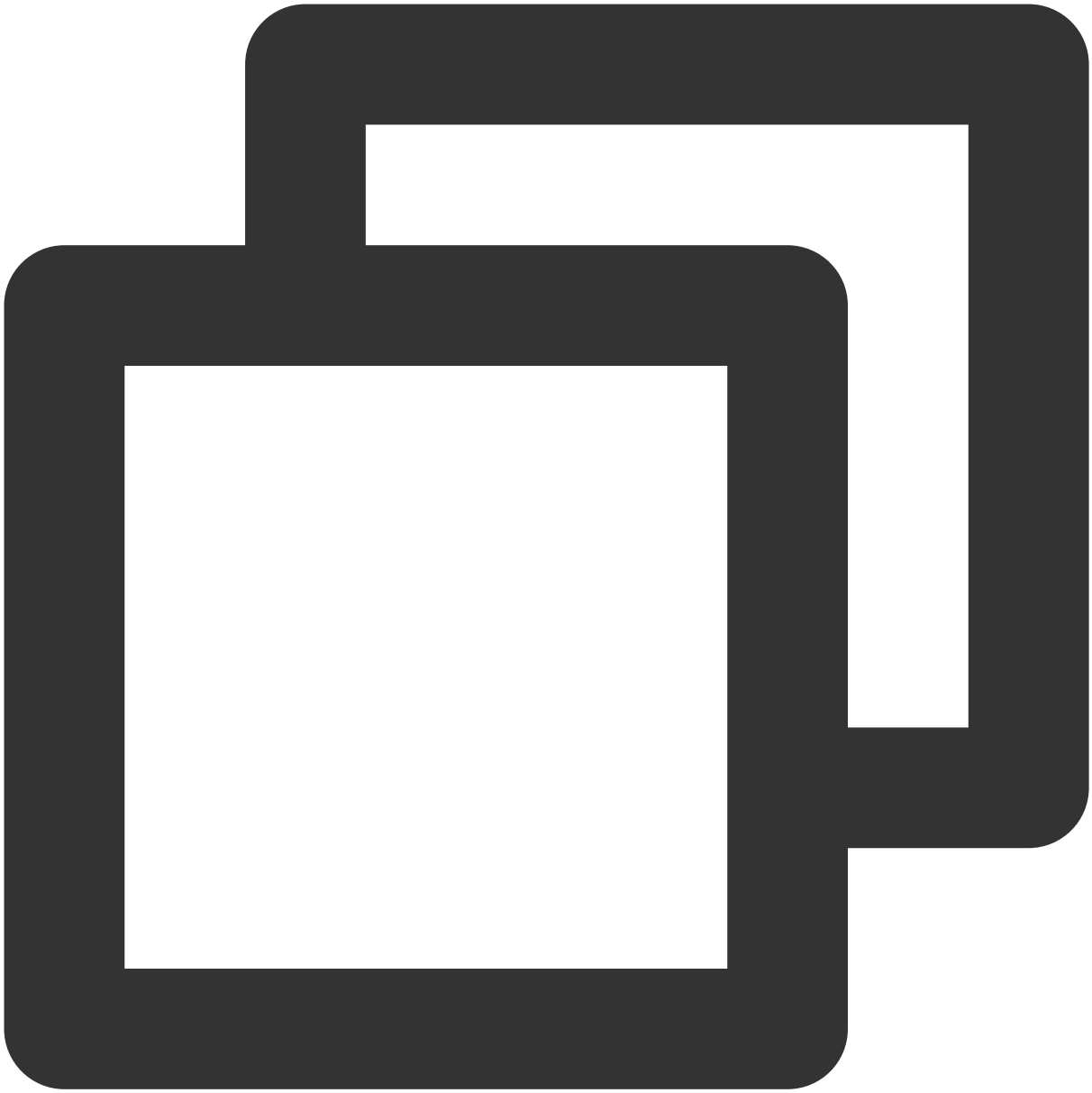
```
fio -directory=/path/to/cfs -iodepth=1 -time_based=1 -thread -direct=1 -ioengine=li
```

Write Latency Testing



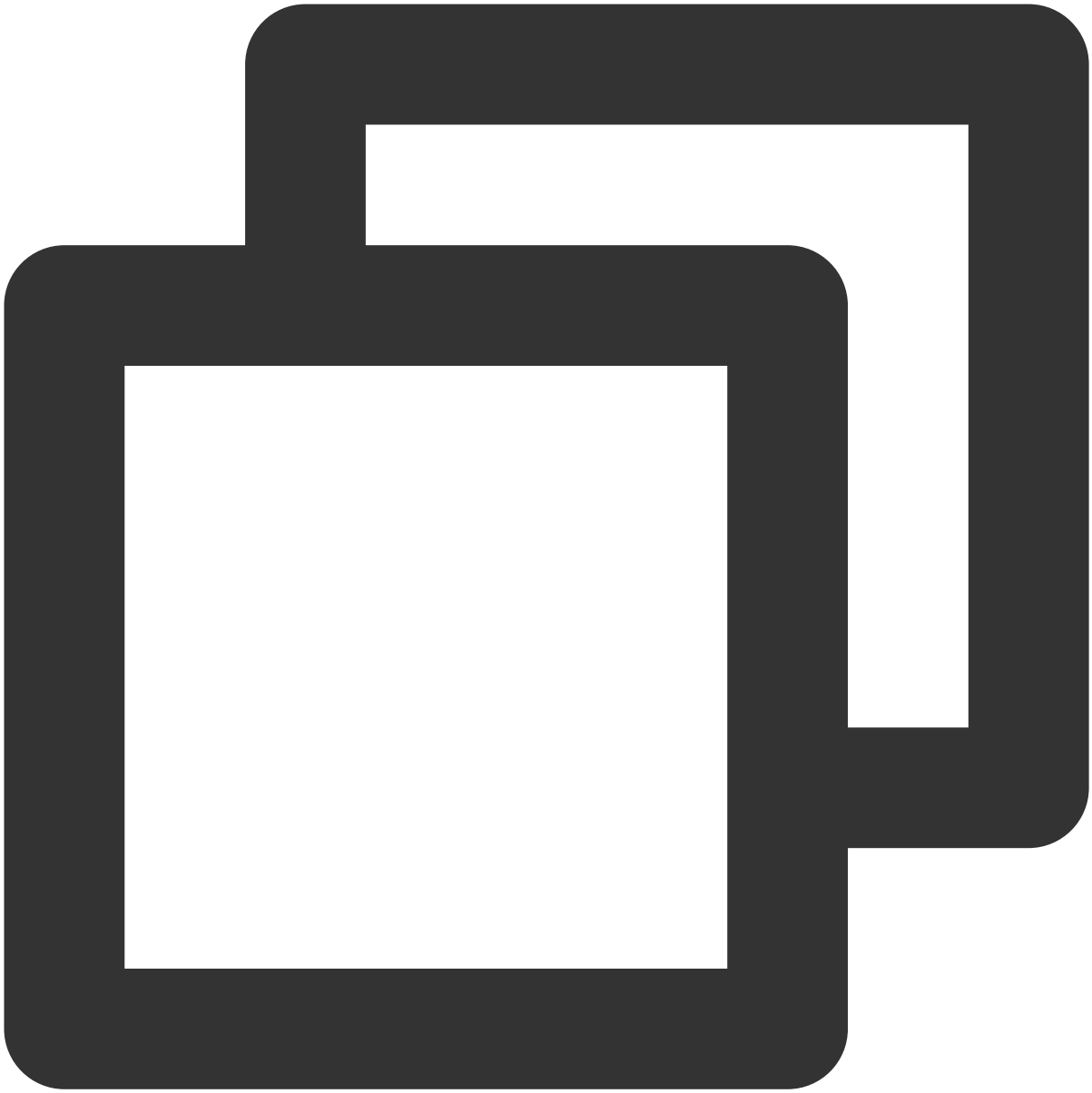
```
fio -directory=/path/to/cfs -iodepth=1 -time_based=1 -thread -direct=1 -ioengine=li
```

Read IOPS Testing



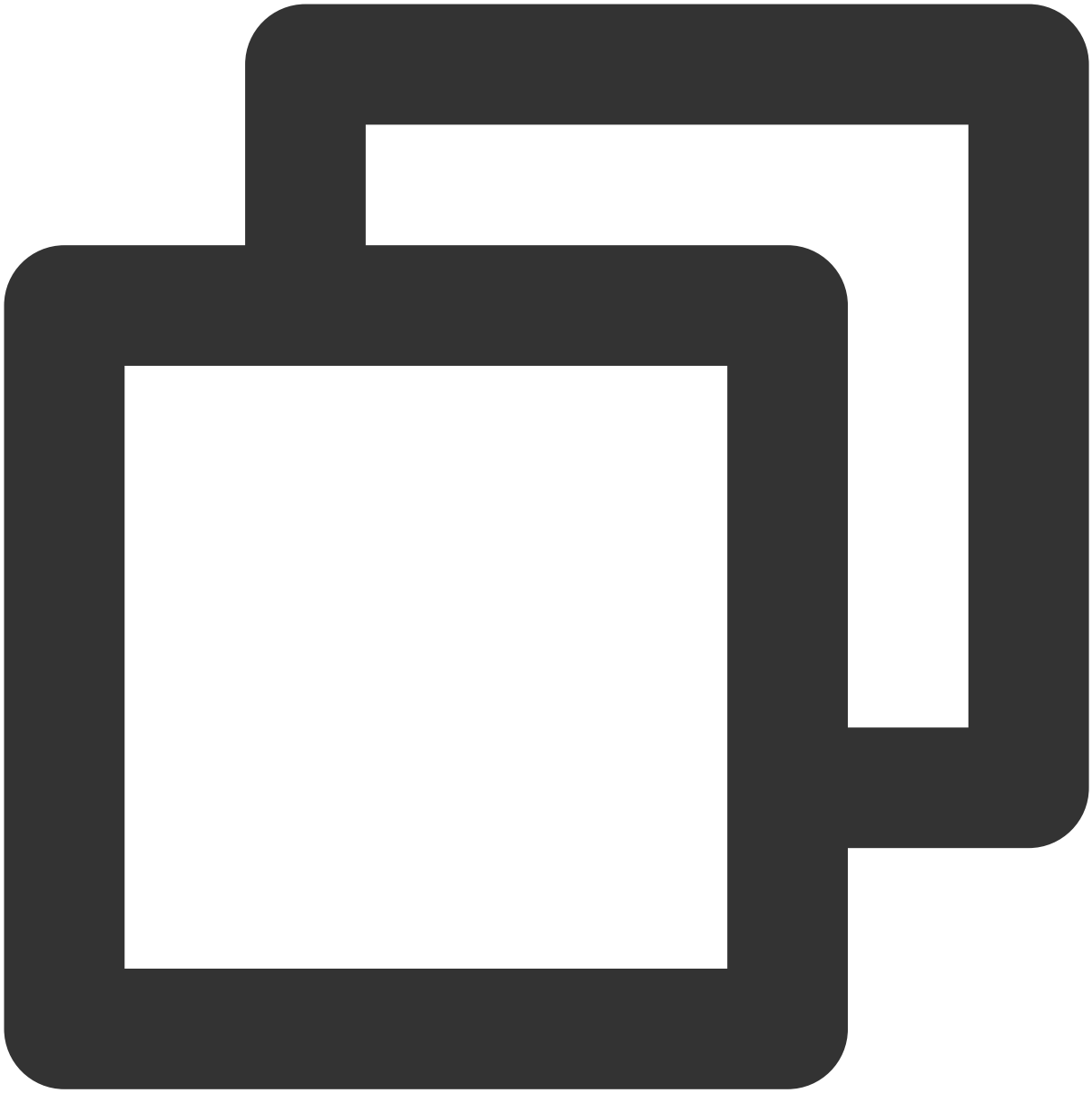
```
fio -directory=/path/to/cfs -iodepth=1 -time_based=1 -thread -direct=1 -ioengine=li
```

Write IOPS Testing



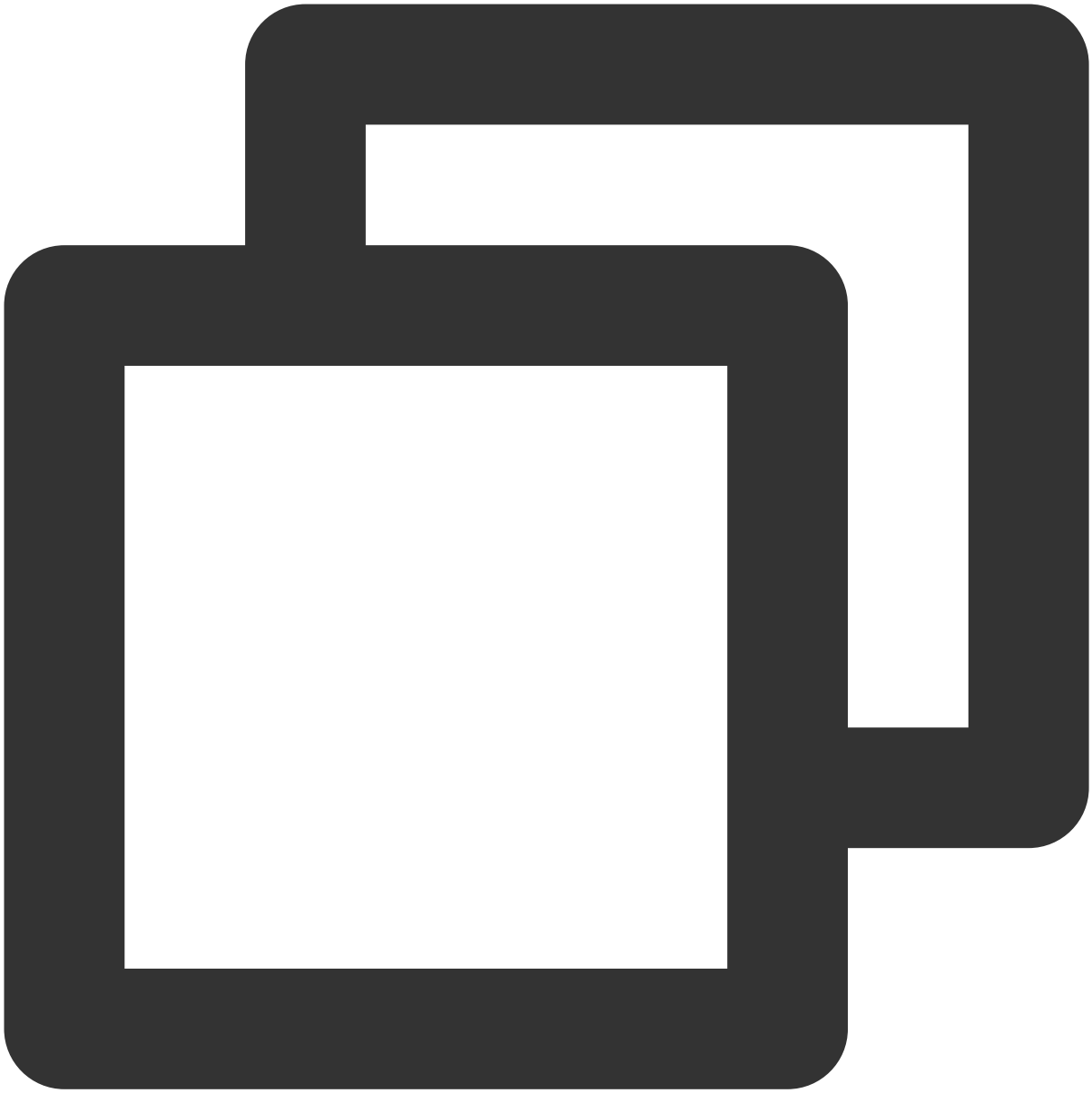
```
fio -directory=/path/to/cfs -iodepth=1 -time_based=1 -thread -direct=1 -ioengine=li
```

Read Throughput Testing



```
fio -directory=/path/to/cfs -iodepth=1 -time_based=1 -thread -direct=1 -ioengine=li
```

Write Throughput Testing



```
fio -directory=/path/to/cfs -iodepth=1 -time_based=1 -thread -direct=1 -ioengine=li
```

FIO Parameter Description

Parameter	Parameter Description
direct	Indicates whether to use direct I/O. Default value is 1.

	Value of 1: Indicates to use direct I/O, and ignores the client's I/O cache, with data being written or read directly. Value of 0: Indicates not to use direct I/O. **Note:** This parameter cannot bypass server-side caching.
iodepth	Indicates the I/O queue depth during testing. For example, -iodepth=1 means the maximum number of I/Os in an FIO control request is 1.
rw	Indicates the read/write strategy during testing. You can set it to: randwrite: random write randread: random read read: sequential read write: sequential write randrw: mixed random read and write **Note:** Usually, during stress testing, random read/write are used. If sequential read/write performance metrics are required, parameters can be adjusted accordingly.
ioengine	Indicates which I/O engine FIO should use during testing. Usually libaio is chosen to ensure asynchronous issuance of data IO. **Note:** If libaio is not chosen during stress testing, the performance bottleneck is mainly on the ioengine, not on the storage side.
bs	Indicates the block size of the I/O unit.
size	Indicates the testing file size. FIO will read/write the specified file size in its entirety before stopping the test. Unless it is limited by other options (such as runtime). If this parameter is not specified, FIO will use the full size of the given file or device. The size can also be given as a percentage between 1 and 100. For example, if size=20% is specified, FIO will use 20% of the full size of the given file or device.
numjobs	Indicates the number of concurrent threads for the testing.
runtime	Indicates the testing duration. It is the FIO execution duration.
group_reporting	Indicates the testing results display mode. If this parameter is specified, the testing results will summarize the statistical information for each process, instead of for different tasks.
directory	Indicates the file system mount point to be tested. **Note:** When this parameter is chosen, FIO will by default create files at this path, the number of which equals numjobs, for stress testing. This parameter is mandatory for storage stress testing, as specifying a filename directly targets a single file testing.

name	Indicates the name of the testing task. It can be set according to actual needs.
thread	Conduct stress testing using multi-threading rather than multi-processing.
time_based	The value is set to 1: After the specified file size has been fully read and written, I/O operations are automatically repeated until the time specified by the runtime parameter expires. The value is set to 0: Testing stops immediately upon completion of reading and writing the specified file size. **Note:** Typically, the value is set to 1 during testing, which can ensure that the testing program runs continuously within the specified time period.

Note:

For more parameters about FIO stress testing, see the FIO documentation.

For testing across multiple machines, commands can be concurrently executed using pshell. Or see the FIO documentation to configure the cluster edition stress testing parameters.