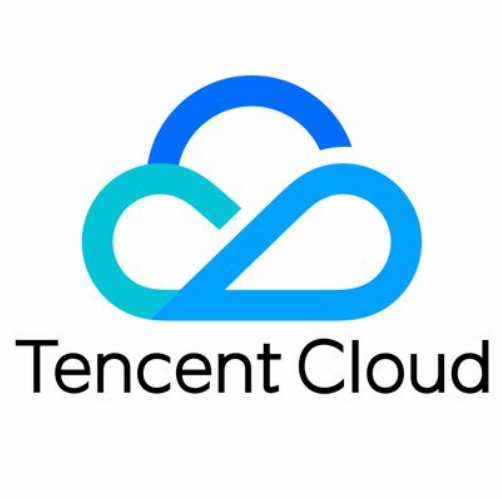
文件存储

实践教程

产品文档



文档目录

实践教程
NFS 客户端内核选择

最近更新时间：2024-07-05 17:49:12

NFS 客户端是基于内核态的客户端，因部分内核版本的 BUG，会导致 NFS 服务无法正常使用，为了保证您更好的使用体验，请使用我们推荐的内核版本。

已知的客户端问题

内核网络栈缺陷导致文件系统无响应（优先级：高）

当系统的内核版本为2.6.32-696~2.6.32-696.10.1（包括2.6.32-696，但不包括2.6.32-696.10.1）时，NFS 服务端繁忙，内核请求重传，有概率触发内核网络栈缺陷，造成操作无响应。
当操作无响应时，请重启 CVM 实例。更多信息，请参见 RHEL6.9:NFSv4 TCP transport stuck in FIN_WAIT_2 forever 。

内核缺陷导致文件系统无响应（优先级：高）

- 当系统的内核版本为以下几个版本时，NFS 服务端故障转移，可能造成 NFS 客户端的打开、读、写操作出现死锁情况，从而导致文件系统持续无响应。
  - Redhat 6、CentOS 6 2.6.32-696.3.1.el6。
  - Redhat 7、CentOS 7 3.10.0-229.11.1.el7之前的所有内核版本。
- Ubuntu 15.10 Linux 4.2.0-18-generic。
  当操作无响应时，请重启 CVM 实例。更多信息，请参见 RHEL7:NFSv4 client loops with WRITE/NFS4ERR_STALE_STATEID – if NFS server restarts multiple times within the grace period 。
- 当系统的内核版本为以下几个版本时，网络发生分区或抖动，造成连接重连，NFS 客户端可能由于没有正确处理错误码而持续无响应。现象是文件系统无响应且系统 message 中反复打印 bad sequence-id error。
  - Redhat 6、CentOS 6 2.6.32-696.16.1.el6之前的所有内核版本。
  - Redhat 7、CentOS 7 3.10.0-693.el7之前的所有内核版本。
    当操作无响应时，请重启 CVM 实例。更多信息，请参见 RHEL6/RHEL7:NFS4 client receiving NFS4ERR_BAD_SEQID drops nfs4 stateowner resulting in infinite loop of READ/WRITE+NFS4ERR_BAD_STATEID 。
- 当操作系统内核版本为 CentOS 和 RedHat 5.11.x 所有内核时，执行 ls 命令、包含通配符 * 或 ? 的命令以及其他需要对目录进行遍历的操作，均会由于内核缺陷导致卡顿或无响应。 请您升级内核版本，避免此问题。

不支持 chown 命令和系统调用（优先级：低）

系统的内核版本为2.6.32时，不支持 NFS 客户端执行 chown 命令和系统调用。

ls 操作无法终止（优先级：低）

- 当系统的内核版本为2.6.32-696.1.1.el6及之前版本时，在系统中执行 ls 操作的同时还在进行添加、删除文件、子目录操作，将导致 ls 操作永远无法终止。 请升级内核版本，避免此问题。
- 当系统的内核版本为4.18.0-305.12.1时，目录遍历操作如ls等，可能无法终止，请升级内核至4.18.0-305.19.1或更高版本修复此问题。由于CentOS 官方计划停止维护 CentOS Linux 项目，支持长期维护的内核方案选择可参考 Linux 内核获取 。

NFS文件系统推荐镜像

Linux系统镜像

操作系统类型	操作系统版本
CentOS	• CentOS 6.9 64位：2.6.32-696.16.1.el6.x86_64及以上 • CentOS 6.10 64位：2.6.32-754.17.1.el6.x86_64及以上 • CentOS 7.2 64位：3.10.0-514.26.2.el7.x86_64及以上 • CentOS 7.3 64位：3.10.0-514.26.2.el7.x86_64及以上 • CentOS 7.4 64位：3.10.0-693.2.2.el7.x86_64及以上 • CentOS 7.5 64位：3.10.0-862.14.4.el7.x86_64及以上 • CentOS 7.6 64位：3.10.0-957.21.3.el7.x86_64及以上 • CentOS 7.7 64位：3.10.0-1062.18.1.el7.x86_64及以上 • CentOS 8.x 64位：4.18.0-147.5.1.el8_1.x86_64及以上
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 64位：4.9.0-8-amd64及以上 • Debian 9.8 64位：4.9.0-8-amd64及以上 • Debian 9.10 64位：4.9.0-9-amd64及以上
Ubuntu	• Ubuntu 14.04 64位：4.4.0-93-generic及以上 • Ubuntu 16.04 64位：4.4.0-151-generic及以上

	• Ubuntu 18.04 64位：4.15.0-52-generic及以上 • Ubuntu 20.04 64位：5.4.0-31-generic及以上
OpenSuse	• OpenSuse 42.3 64位：4.4.90-28-default及以上
Suse	• Enterprise Server 12 SP2 64位：4.4.74-92.35-default及以上 • Enterprise Server 12 SP4 64位：4.12.14-95.16-default及以上
CoreOS	• CoreOS 1745.7.0 64位：4.19.56-coreos-r1及以上 • CoreOS 2023.4.0 64位：4.19.56-coreos-r1及以上

Windows系统镜像

操作系统类型	操作系统版本
Windows Server 2012	• Windows Server 2012 R2 数据中心版 64位中文版 • Windows Server 2012 R2 数据中心版 64位英文版
Windows Server 2016	• Windows Server 2016 数据中心版 64位中文版 • Windows Server 2016 数据中心版 64位英文版
Windows Server 2019	• Windows Server 2019 数据中心版 64位中文版 • Windows Server 2019 数据中心版 64位英文版

Turbo 目录管理

最近更新时间：2024-01-22 22:15:48

使用场景

该实践主要应用在使用 Turbo 文件系统下，如何更好的进行目录结构和文件数的分配，以实现更高的性能和更低的延时。

背景介绍

- 客户端访问文件系统时，都会基于 VFS layer 进行操作，VFS 层在处理多进程同时读写相同文件时，会串行的进行锁的授予和召回。那么当某个客户端的 IO 在某一层超大目录下时，因 VFS 串行的锁操作行为，会导致 IO 延时变大。
- Turbo 系统为强一致的文件系统，在实现任意客户端读写任意时刻一致的强大功能下，也意味着所有客户端，在执行 IO 操作，都会涉及到后端分布式锁的授予或召回。Turbo 后端基于分布式的元数据服务能并发地处理锁的请求，但为保证良好的延时表现，我们建议您按照本文推荐的方式进行目录和文件数的管理。

最佳实践建议

- 当您的业务不涉及频繁删除/修改/新增的情况下，如以读为主，建议单目录下子目录和文件数控制在10个–100万个。
- 当您的业务涉及频繁删除/修改/新增的情况下，建议单目录下子目录和文件数控制在10个–1万个。

常见的目录分级方案

- 哈希码串区段分类：一个64个字符的哈希码串，头2个字符形成一级子目录，次2个字符形成二级子目录，二级子目录下放文件。哈希函数统计特性良好时，文件可以比较均匀地分布到这65536个目录。文件系统文件规模为6亿时，每一个目录平均1万个文件；文件系统文件规模为12亿时，每一个目录平均2万个文件。
- 时间分类：年月日构成第一级子目录，小时构成第二级子目录，分钟构成第三级子目录（可以没有）。

计算实例销毁

最近更新时间：2024-01-22 22:15:48

使用场景

该实践主要应用在使用 Turbo 文件系统下，当需要动态的调整计算实例时，为避免不当的操作方式引发的锁冲突，提供的一种更佳的实例销毁方式。
通常用户在销毁 CVM 机器时，会直接调用 TerminateInstances 接口进行销毁，而这个接口实际会对服务器进行强制关机（类似切断电源）并退还。此方案的优点是销毁快，但对于 CFS Turbo 这种强一致的分布式文件系统，客户端强行失联会使服务端锁召回异常，进而导致 IO Hang。为避免影响客户业务的正常使用，推荐使用本文建议的方式进行销毁，可达到对业务无影响的效果。

操作步骤

对云服务器执行关机操作

调用 StopInstances 接口进行关机，Stoptype 选择 SOFT_FIRST，此方案会优先进行正常关机，当5分钟内未正常关机后，再执行强制关机。此方案能在保证 Turbo 正常使用的情况下，兼顾关机的时效性。

> **说明：**
> 1. 关机时请勿将 stoptype 选择 hard 模式，此模式为强制关机。
> 2. 控制台上关机，请勿选择强制关机。选择关机即可，此选项为 SOFT_FIRST。
> 3. 正常的关机启动，会有序终止进程，并执行 sync 操作，可有效降低拉闸式关机对 Turbo 分布式锁的影响。

销毁实例

调用 DescribeInstancesStatus 查询实例状态。当状态为 STOPPED 时，再执行 TerminateInstances 接口进行销毁。

在容器 TKE 上使用 CFS

最近更新时间：2024-06-25 15:22:07

操作场景

文件存储（Cloud File Storage，CFS）在容器环境如下主要适用于两类场景：

场景一：POD/容器数据的持久化存储，推荐使用动态挂载 CFS

CFS 可提供一个持久化存储的空间，当 POD/容器销毁时，数据仍然保存；当 POD/容器再次启动时，可通过 PVC 快速挂载原空间，实现数据的读写操作。
相比于其他方案，单 FS 实例可同时支持多个 POD/容器的数据存储，并支持根据 CFS 实例中的不同子目录，分配给不同的 POD。CFS 通用标准型/性能型按照实际使用容量进行计费，且无最小购买容量要求，可降低用户大规模容器持久化数据存储的成本。

场景二：多 POD/容器的数据共享，推荐使用静态挂载 CFS

CFS 通过 NFS/私有协议提供了一个共享访问的目录空间给多个 POD/容器，实现数据资源的高效共享，相比于其他方案，能提供更高的带宽和 IOPS 能力。
本文将重点介绍基于腾讯云控制台，部署容器 workload 的方法。具体的 YAML 写法，可参考通过控制台创建 StorageClass 后自动生成 YAML 文件。

> **注意：**
> 使用前，需确保容器服务 （Tencent Kubernetes Engine，TKE）集群的 CSI 组件在1.0.4版本以上。若版本不对，可在 TKE 控制台上进行更新，CSI 组件的更新不影响容器的正常使用。

操作步骤

动态挂载 CFS

> **说明：**
> POD/容器数据的持久化存储场景，推荐此方式进行挂载。

1. 创建 StorageClass，具体操作请参见 通过控制台创建 StorageClass 文档。

相关的关键配置项如下：

配置项	配置项.说明
实例创建模式	此场景选择共享实例。
可用区	建议选择与容器宿主机相同的可用区，以便获得更好的性能。
存储类型	根据实际性能需求可选择通用标准型存储和通用性能型存储。
协议版本	在非多个同时修改/编辑的场景下，建议使用 NFS V3协议，以便得到更好的性能。

回收策略	可根据实际需要，选择删除和保留，为避免数据误删优先建议选择保留。

2. 创建 PVC，具体操作请参见 创建 PVC 文档。

相关的关键配置项如下：

配置项	配置项说明
命名空间	根据实际需要选择不同的命名空间。
Storageclass	选择刚才创建的 StorgeClass。
是否指定 Persistent Volume	动态创建可不指定 PV。说明：基于 CFS 共享型实例的 StorgeClass，在创建 PVC 时，若不指定 PV，CSI 插件会在创建 PVC 的同时，自动创建一个按量付费的 CFS 实例，此实例会随着 PVC 的删除而删除，故需谨慎处理基于此方式创建的 PVC。

3. 创建 Deployment，具体操作请参见 创建 Deployment 文档。

相关的关键配置项如下：

配置项	配置项说明
数据卷	根据实际需求，对数据卷进行命名，并选中刚创建的 PVC。
挂载点	选择对应的数据卷，指定在容器本地的挂载路径。在共享 CFS 实例的动态创建方式下，需要指定具体的环境变量，CSI 插件会基于配置的环境变量的变量值，在所选 PVC 对应的 CFS 实例里创建目录供容器挂载。

配置完成后，单击**创建 Workload**，系统将基于此配置创建容器，并挂载 CFS。

静态挂载 CFS

> **说明：**

> 多 POD/容器的数据共享场景，推荐此方式进行挂载。

1. 创建 StorageClass，具体操作请参见 通过控制台创建 StorageClass 文档。

相关的关键配置项如下：

配置项	配置项.说明
实例创建模式	此场景选择共享实例。
可用区	建议选择与容器宿主机相同的可用区，以便获得更好的性能。

存储类型	根据实际性能需求可选择通用标准型存储和通用性能型存储。
协议版本	在非多个同时修改/编辑的场景下，建议使用 NFS V3 协议，以便得到更好的性能。
回收策略	可根据实际需要，选择删除和保留，为避免数据误删优先建议选择保留。

2. 创建 PV，具体操作请参见 静态创建 PV 文档。

相关的关键配置项如下：

配置项	配置项说明
来源设置	选择静态创建，即指定某个 CFS 实例配置 PV。
Storge Class	选择刚才创建的 StorgeClass。
选择 CFS	选择一个指定的 CFS。说明：静态创建时需要保证已经有一个 CFS 实例，同时该 CFS 实例需与容器在同一个 VPC 网络环境。
CFS 子	CFS 可以允许挂载子目录，用户可根据实际需要选择不同的子目录绑定至一个或多个 PV

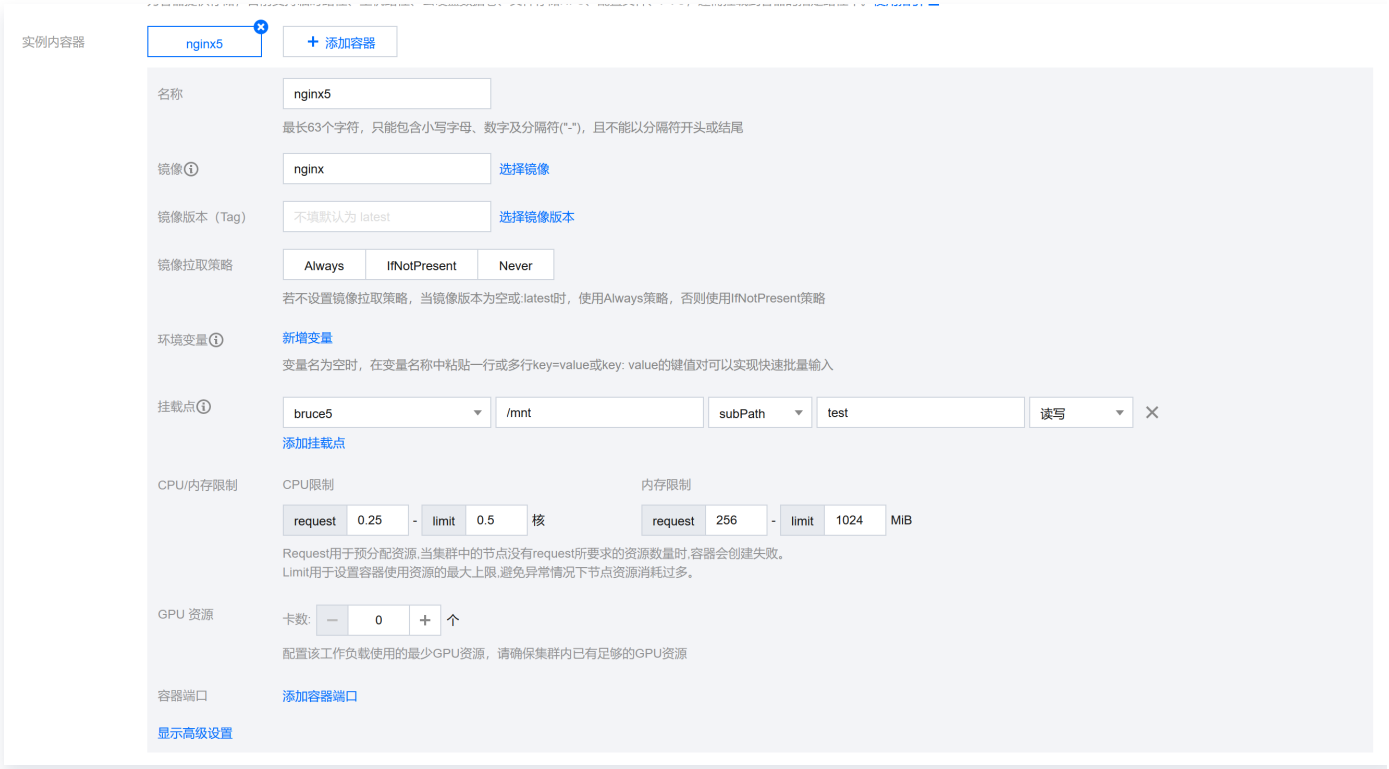
目录	上，实现不同程度的数据共享。

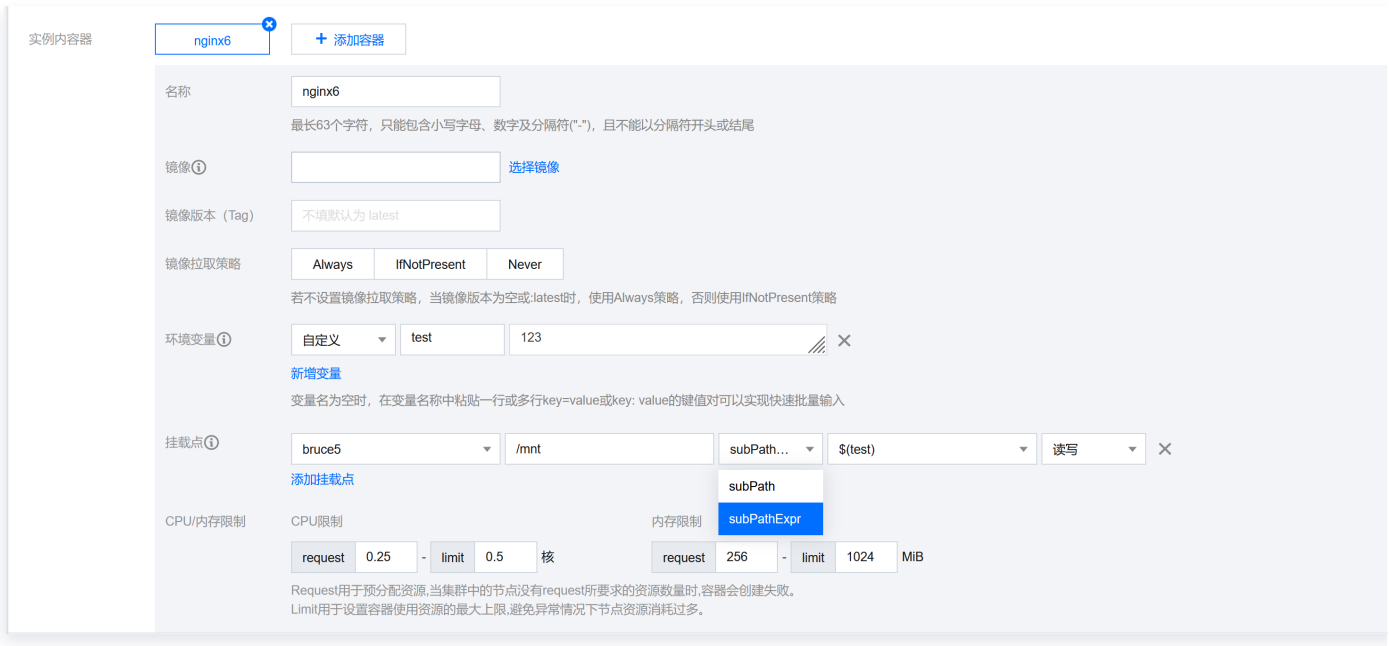
3. 创建 PVC，具体操作请参见 创建 PVC 文档。

相关的关键配置项如下：

配置项	配置项说明
命名空间	根据实际需要选择不同的命名空间。
Storageclass	选择刚才创建的 StorgeClass。
是否指定PersistentVolume	选择指定，并选择刚才创建的 PV。

4. 创建 Deployment，具体操作请参见 创建 Deployment 文档。





相关的关键配置项如下：

配置项	配置项说明

数据卷	根据实际需求，对数据卷进行命名，并选中刚创建的 PVC。
挂载点	选择对应的数据卷，指定在容器本地的挂载路径。对于自动创建子目录的场景，目前有两种方案可以使用。方案一，挂载点选择subPath，直接填写希望挂载的路径名称如 test，则 CSI 插件会自动在文件系统根路径下创建 test 目录，并自动挂载至容器的指定目录下。方案二，添加环境变量，并给环境变量赋值。挂载点选择 subPathExpr，并选择对应的环境变量。则 CSI 插件将会以该环境变量的变量值作为目录名，自动在文件系统的根路径下创建该目录，并自动挂载至容器的指定目录下。

5. 配置完成后，单击**创建 Workload**，系统将基于此配置创建容器，并挂载 CFS。

在云函数 SCF上使用 CFS

最近更新时间：2024-01-22 22:15:47

操作场景

云函数（Serverless Cloud Function，SCF） 是腾讯云为企业和开发者们提供的无服务器执行环境，帮助您在无需购买和管理服务器的情况下运行代码。您只需使用平台支持的语言编写核心代码并设置代码运行的条件，即可在腾讯云基础设施上弹性、安全地运行代码。SCF 是实时文件处理和数据处理等场景下理想的计算平台。
SCF 是服务级别的计算资源，它的快速迭代、极速部署的特性天然需要存储与计算分离，而文件存储（Cloud File Storage，CFS）提供的高性能共享存储服务为 SCF 最佳的存储方案 。只需几步简单配置，您的函数即可轻松访问存储在 CFS 文件系统中的文件。SCF 使用 CFS 的优势如下：

- 函数执行空间不受限。
- 多个函数可共用一个文件系统，实现文件共享。

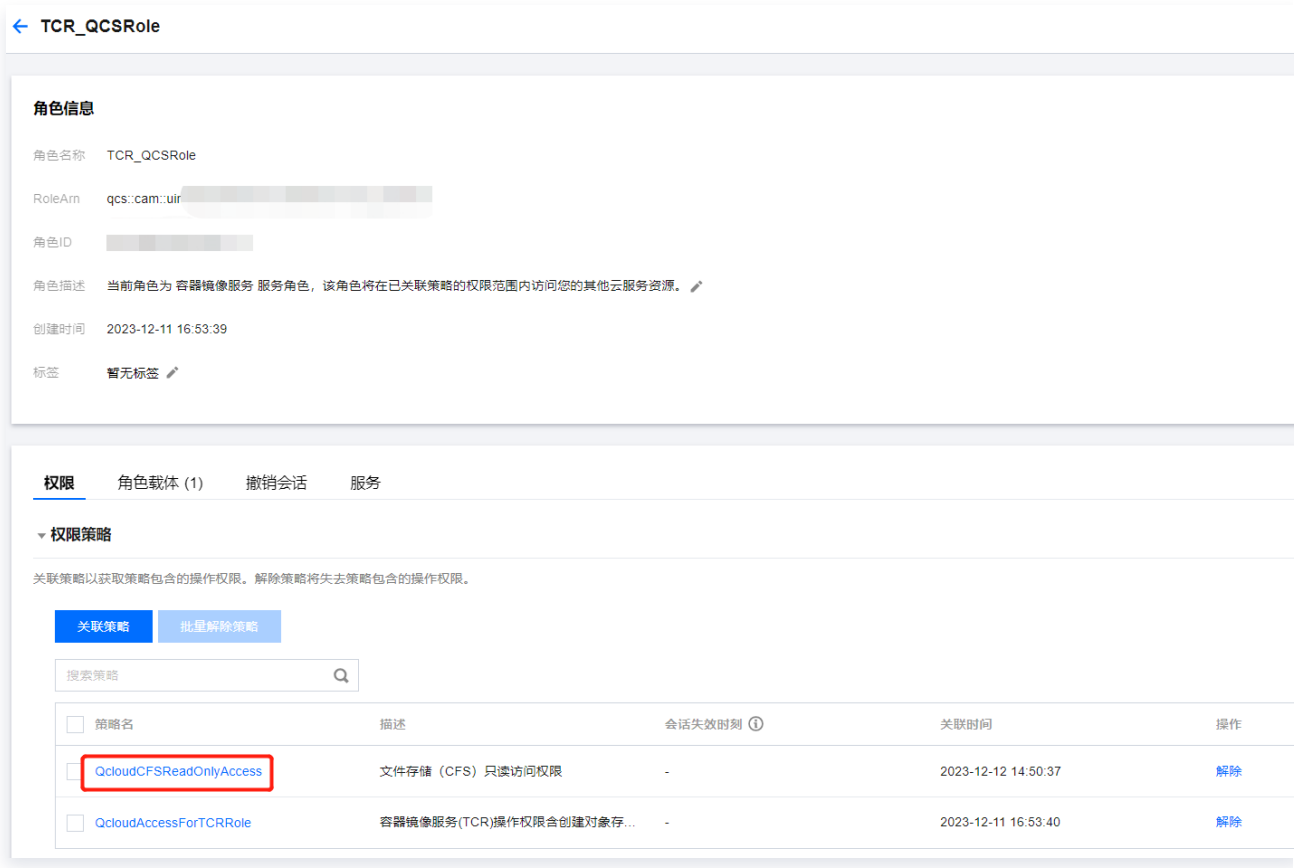
操作步骤

关联授权策略

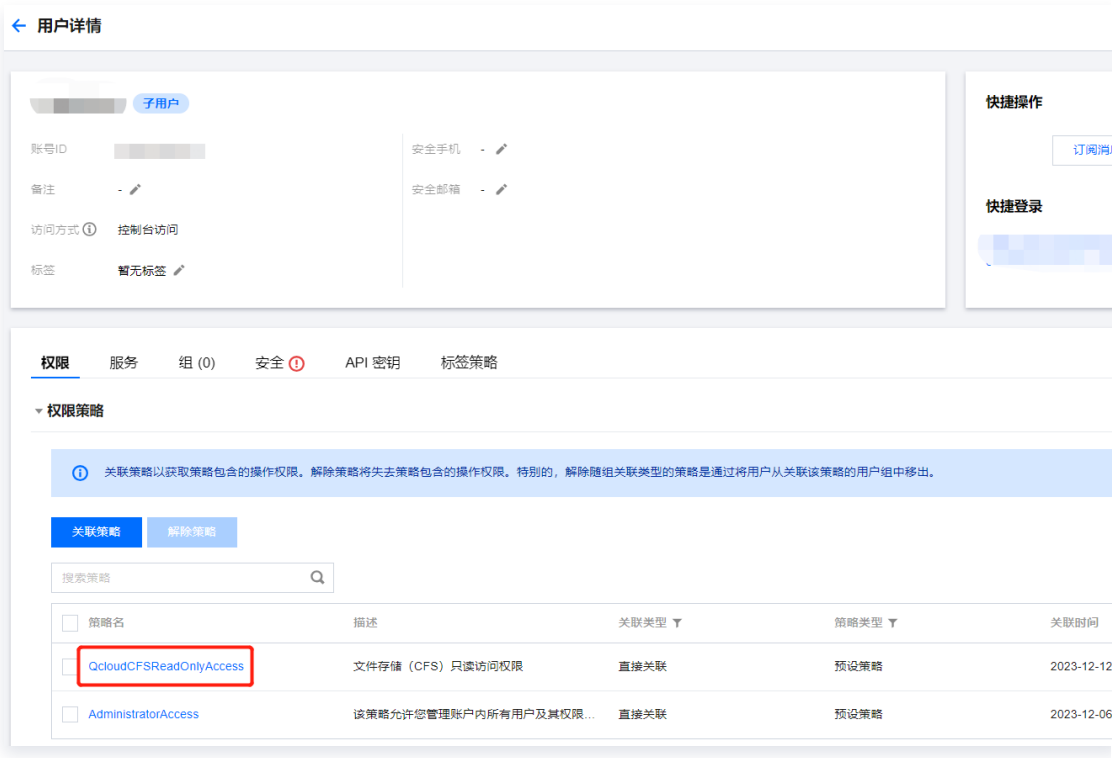
> **注意：**
> 如需使用 CFS 功能，云函数需要能够操作您 CFS 资源的权限。

请参考以下步骤为账号进行授权操作：

1. 请参见 修改角色，为 `SCF_QcsRole` 角色关联 `QcloudCFSReadOnlyAccess` 策略。关联成功则如下图
   如您使用的账号未进行该操作，则可能出现函数无法保存，CFS 相关功能无法使用等问题。



2. 如您使用账号为子账号，则请联系主账号并参见 子用户权限设置 为您的子账号关联 `QcloudCFSReadOnlyAc`
   如您使用的子账号未进行该操作，则可能出现无法使用 CFS 相关功能的问题。



创建私有网络 VPC

请参见 快速搭建 IPv4 私有网络 完成 VPC 创建。

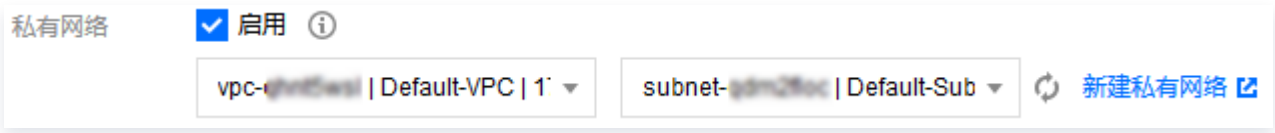
创建 CFS 资源

请参见 创建 CFS 文件系统 完成创建操作。

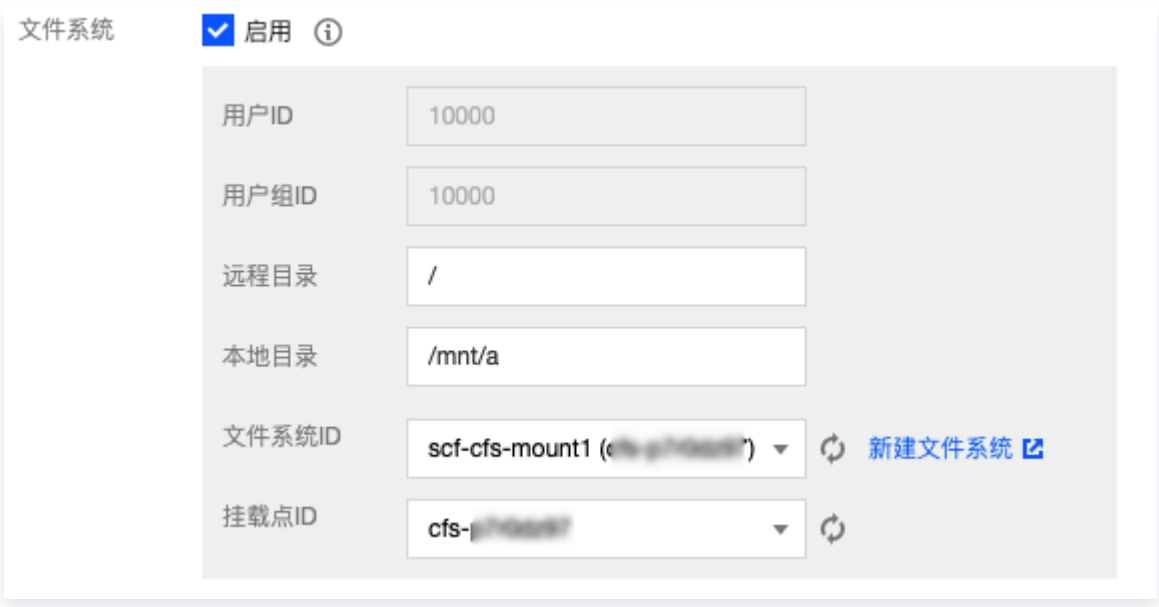
> **注意：**
> 目前云函数仅支持添加网络类型为 VPC 的 CFS 文件系统作为挂载点。请在创建的 CFS 文件系统时，选择与函数相同的 VPC，以确保网络能够互通。

挂载并使用 CFS 文件系统

1. 登录云函数控制台，选择左侧导航栏中的 函数服务 。
2. 在“函数服务”页面，选择需配置的函数名。
3. 在“函数管理”页面的**函数配置**页签中，单击右上角的**编辑**。
4. 在“私有网络”中，勾选启用并选择 CFS 文件系统所在的 VPC。如下图所示：



5. 在“文件系统”中勾选启用，并按照以下信息进行挂载。如下图所示：



- **用户 ID 及用户组 ID**：这两个值等同于 CFS 文件系统中的用户及用户组。云函数默认用户及用户组值为10000，来操作您的 CFS 文件系统。请按需设置文件的拥有者及相应组的权限，并确保您的 CFS 文件系统已配置相应权限。详情请参见 权限设置 。
- **远程目录**：为云函数需访问 CFS 文件系统的远端目录，由文件系统和远端目录两部分组成。
- **本地目录**：为本地文件系统的挂载点。您可使用 `/mnt/` 目录的子目录挂载 CFS 文件系统。
- **文件系统 ID**：在下拉列表中选择需挂载的文件系统。
- **挂载点 ID**：在下拉列表中选择对应文件系统的挂载点 ID。

6. 单击页面下方的**保存**即可完成配置。
   您可执行以下函数代码，开始使用 CFS 文件系统。

```
'use strict';
var fs = require('fs');
exports.main_handler = async (event, context) => {
  await fs.promises.writeFile('/mnt/myfolder/file1.txt',
JSON.stringify(event));
  return event;
};
```

SCF 使用 CFS 文件系统性能测试

您可以使用此 脚本 测试 SCF 使用 CFS 时的性能。

在容器 TKE 上使用 CFS Turbo

最近更新时间：2024-01-22 22:15:48

简介

本文为您介绍如何使用 CFS Turbo 对接容器服务（Tencent Kubernetes Engine，TKE）集群。

前提条件

- TKE 的宿主机节点满足 Turbo 系列兼容的操作系统。
- 已在所有 TKE 节点安装 Turbo 的私有客户端，推荐使用 pshell 工具进行批量操作。

相关的操作系统兼容列表及私有客户端安装方式，可参见 在 Linux 客户端上使用 CFS Turbo 文件系统 文档。

操作步骤

配置kubectl连接集群

您可参见 TKE连接集群，配置 kubectl，以管理 TKE 容器集群。

通过 yaml 文件创建挂载 Turbo 的 POD

1. 阅读 TKE Turbo 插件的说明文档
2. 进入 `kubernetes-csi-tencentcloud/deploy/cfsturbo/kubernetes/` 目录，分别将 csi-node-rbac.yaml、csi-node.yaml 和 csidriver-new.yaml 文件上传至 kubectl 管理节点。
3. 进入 `kubernetes-csi-tencentcloud/deploy/cfsturbo/examples/` 目录，下载 pv.yaml、pvc.yaml、 pod.yaml 这三个示例文件。
4. 根据实际 PV、PVC、POD 的相关属性（如名称、镜像地址等），修改 pv.yaml、pvc.yaml、 pod.yaml 文件。yaml 示例如下：

```
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 to
bind the pv and pvc.
  volumeName: csi-cfsturbo-pv
  # cfsturbo only supports static provisioning, the StorageClass name
should be empty.
  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
```

以此挂载指令为例：

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

关键参数说明如下：

- proto:lustre, 请保持此参数不要进行修改。
- roodir:/cfs，请保持此参数不要进行修改。
- fsid:d3dcc487，此处的 fsid 不是 CFSID，需要填写挂载路径里的信息。
- host:10.0.1.16，挂载点 IP。
- path 可根据实际需要挂载的子目录进行调整，若直接挂载根目录则填写"/"。

5. 在上传脚本文件的目录下，依次执行如下命令。

- 配置 RBAC、CSI 插件。

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

- 创建 PV、PVC、POD。

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

6. 执行如下命令，查看 POD 状态。

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

返回如下结果，即表示创建成功。

```
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>
```

若状态（STATUS）持续为 ContainerCreating，即表示创建失败。您可在 TKE 控制台的事件中，查看失败原因。

在 Serverless 容器服务上使用 CFS Turbo

最近更新时间：2024-06-25 15:22:07

使用场景

为 Serverless 容器服务挂载文件存储（Cloud File Storage，CFS）Turbo 类型存储，该组件基于私有协议将腾讯云 CFS Turbo 文件系统挂载到工作负载，目前仅支持静态配置。CFS 存储类型详情见 文件存储类型及性能规格 。

前提条件

已创建 Serverless 容器服务且版本 >=1.14。

使用步骤

创建文件系统

创建 CFS Turbo 文件系统，具体操作请参见 创建文件系统 。

> **注意：**
> 文件系统创建后，需将集群网络（vpc-xx）关联到文件系统的 云联网 （可在文件系统挂载点信息中查看）。

部署 Node Plugin

步骤1：新建 csidriver.yaml 文件

csidriver.yaml 文件示例如下：

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

步骤2：创建 csidriver

执行以下命令创建 csidriver：

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

创建 CFS Turbo 存储卷

步骤1：使用以下模板创建 CFS Turbo 类型 PV

```
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
      #根据实际的挂载点IP填写
      fsid: d3816815
      #根据实际的FSID填写
      path: /
      #cfs turbo subPath
  storageClassName: ""
```

参数说明：

- **metadata.name**：创建 PV 名称。
- **spec.csi.volumeHandle**：与 PV 名称保持一致。
- **spec.csi.volumeAttributes.host**：文件系统 IP 地址，可在文件系统挂载点信息中查看。
- **spec.csi.volumeAttributes.fsid**：文件系统 fsid（非文件系统 id），可在文件系统挂载点信息中查看（挂载命令中 "tcp0:/" 与 "/cfs" 之间的字符串，如下图）。
- **spec.csi.volumeAttributes.path**: 文件系统子目录，不填写默认为“/”（为提高挂载性能，插件后端将“/”目录实际定位到“/cfs目录下”）。如需指定子目录挂载，须确保该子目录在文件系统“/cfs”中存在，挂载后 workload 将无法访问到该子目录的上层目录。例如：path: /test，需在文件系统中保证/cfs/test目录存在。

步骤2：使用以下模板创建 PVC 绑定 PV

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-cfsturbo
spec:
  storageClassName: ""
  volumeName: pv-cfsturbo
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
```

参数说明：

- metadata.name：创建 PVC 名称。
- spec.volumeName：与 步骤1 中创建 PV 名称保持一致。

使用 CFS Turbo 存储卷

使用以下模板创建 Pod 挂载 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
```

Turbo 文件系统网络选择

最近更新时间：2024-01-22 22:15:48

CFS Turbo 为腾讯云高性能并行文件存储，相比于传统的通用型文件存储，其客户端和服务端目前支持两种网络类型，云联网网络和 VPC 网络，各有优劣。此文档将对这两种网络类型进行对比说明，用于帮助您选择更适合您使用的网络类型。建议有使用云联网的用户优先基于云联网网络创建 Turbo，未使用云联网的用户可根据实际需要酌情选择，若希望简单快捷的使用，可选择 VPC 网络。

云联网网络

简介

通过划分指定的网段给到 Turbo 文件系统，并基于云联网的能力，打通用户 VPC 网络和存储服务端网络的通信，实现计算实例与存储的双向交互。

优点	缺点
• 通过单独的存储网段规划，实现更高效便捷的安全组管理。 • Turbo 文件存储具有单独的网段，预留足够多的 IP，后续扩容无 IP 数量瓶颈。 • 可以更方便的跨 VPC 访问 Turbo 文件存储。 • 对用户现有 VPC 的 IP 地址无占用。	依赖云联网打通网络，对于未使用云联网的用户，需引入新组件，且具备一定复杂度。

最佳实践

可以选择10/11/30/172/192网段创建，建议给 Turbo 分配11/30网段，此网段为服务端网段，对业务 IP 无占用。

VPC 网络

简介

存储服务侧直接映射 IP 至用户现有的 VPC 网络，实现挂载访问。

优点	缺点
• 方便快捷，是最简单的使用方式，与通用型文件存储的使用方式类似。 • 无需引入新的组件，方案最简洁。	• 在大规模扩容使用时，可能存在子网IP不足的问题。 • 会占用 VPC 网络的 IP，且存储与 VPC 网络绑定，不易做网络隔离。

文件存储数据拷贝方案

最近更新时间：2025-07-02 11:40:47

使用场景

在日常使用文件存储时，通常会存在很多需要数据拷贝的场景，本文将介绍如何在 Linux 操作系统下，快速进行数据拷贝的推荐方案。

- **场景一：CFS 文件系统之间、文件系统内不同目录、文件系统和云硬盘间的数据同步。**

  背景说明：在日常使用中，频繁地需要进行数据拷贝。如上三类场景在 Linux 操作系统层面上，都是不同目录之间的数据拷贝，操作方式上是类似的。通常情况下，可执行 cp 指令进行简单快速的拷贝。但如果涉及到大量文件的场景，cp 单线程的运行模式会严重拖慢拷贝进度，此情况下，建议使用本文推荐的方式进行拷贝，可起到自并发加速的作用。

  - **方法一**：当需要详细的过程报告和可视化的操作界面时，推荐使用云迁移服务 CMG 的 文件存储批量迁移 。

    > **注意：**
    > 当前云迁移仅支持以AWS为源的文件存储（EFS）的批量迁移，当源端为 IDC 、本地文件存储、CFS 文件存储等的数据迁移能力即将上线，敬请期待。

  - **方法二**：当需要通过命令行工具时，推荐使用 rclone 工具进行操作，详情请参见下方操作说明。

- **场景二：文件系统与对象存储之间数据同步。**

  方法：可使用对象存储提供的各类基础工具进行数据上传、下载的操作。推荐使用的工具链接如下：COSBrowser 、COSCMD 。

操作说明

此操作步骤面向场景一的方法二使用，其余场景和方法可参考上述链接进行操作。

前置条件

在云服务器上已存在可以迁移的源端和目标端目录。

> **说明：**
> CFS 支持跨 VPC 访问，可通过 云联网 的方式打通多个 VPC，然后进行挂载访问。

操作步骤

1. 下载安装 rclone 工具。
   - 方法一：通过腾讯云镜像服务下载（推荐）：

> **注意：**
> 因涉及到请求腾讯云内网的镜像源，仅腾讯云的云服务器可使用，通过此方式的下载速度会快于从官网直接下载。

```
wget http://mirrors.tencentyun.com/install/cfsturbo-client//migrate_tools/rclone-v1.70.1-linux-amd64.zip && unzip rclone-v1.70.1-linux-amd64.zip && chmod 0755 ./rclone-*/rclone && cp ./rclone-*/rclone /usr/bin/ && rm -rf ./rclone-*
```

- 方法二：通过官网链接下载安装：

```
wget https://downloads.rclone.org/v1.70.1/rclone-v1.70.1-linux-amd64.zip --no-check-certificate && unzip rclone-v1.70.1-linux-amd64.zip && chmod 0755 ./rclone-*/rclone && cp ./rclone-*/rclone /usr/bin/ && rm -rf ./rclone-*
```

2. 执行命令，进行数据同步。

```
rclone copy /mnt/src /mnt/dst -Pvv --transfers 32 --metadata --checkers 64 --links --create-empty-src-dirs --retries=3 --modify-window=1s
```

若需要后台拷贝可参考如下指令：

```
nohup rclone copy /mnt/src /mnt/dst -vv --transfers 32 --checkers 64 --links --create-empty-src-dirs --retries 3 --modify-window=1s  >> /path/to/copy.log
```

> **说明：**
> 1. 参数说明如下，transfers 和 checkers 数目可以根据系统规格自行配置：
>   - transfers：传输文件的并发数目（建议不超过核心数的2倍）。
>   - checkers：扫描本地文件的并发数目（建议不超过核心数的2倍）。
>   - P：实时展示数据拷贝进度（500ms，若不加为1分钟刷新一次）。
>   - links：复制软链接。
>   - metadata：复制文件和目录的元数据信息
>   - vv：打印拷贝日志。

- create-empty-src-dirs：针对空目录也执行拷贝。
- retries：对失败的拷贝进行自动重试的次数（可根据实际需要调整数值）。
- modify-window=1s：rclone 默认先根据文件大小、名称和 mtime 等元数据信息比对，若出现不一致再进行 MD5 值比对。此参数将 mtime 的变化容忍度设置为1s(若不设置默认为1ns)。因 Turbo 文件系统 mtime 的精度为秒级，如果涉及到拷贝到 Turbo 文件系统，为降低不必要的 MD5 比对，建议加上此参数。

2. 此工具重复运行可自动进行增量同步，识别增量的方式为全局扫描。

3. 等待数据完成同步后，可通过日志查看不同文件的迁移任务结果。

文件存储性能测试

最近更新时间：2024-05-23 16:40:15

本文主要介绍如何通过合理的方式对 CFS 云文件系统进行性能测试。

性能关键指标说明

- 时延：处理读写请求的耗时，单位ms。通常基于1MB的文件，采用单流（单线程）的4K 小 IO 进行时延的基准测试。
- IOPS：每秒读写数据块的数量，单位个/s。业内通常基于100MB 的文件，采用并发（多机多线程）的4K 小 IO 进行 IOPS 的基准测试。
- 吞吐：每秒读写数据量的大小，单位GiB/s或MiB/s。通常基于100MB 的文件，采用并发（多机多线程）的1M 大 IO 进行吞吐的基准测试。

注意事项

- CFS 云文件存储所提供的 性能规格，除时延参数外，均需要一定规模和核心数机器进行并发压测才能达到最大值。通常压测时准备16台32C 以上的云服务器作为客户端，可满足大部分场景的压测需求。其他压测需求，可根据实际情况配置对应的云服务器。
- 压测 CFS 通用标准型和通用性能型时，因服务端缓存加速的特性，在命中缓存时，读性能会超过标准值，此现象符合预期。
- 进行性能压测时，尤其是时延测试，需要保证云服务器（客户端）和 CFS 处于同可用区。跨可用区测试的性能结果会和标准值有较大差异，需尽可能避免。

操作步骤

安装压测软件

CentOs/Tlinux:

```
sudo yum install fio
```

Ubuntu/Debian

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

读时延测试

```
fio -directory=/path/to/cfs -iodepth=1 -time_based=1 -thread -direct=1 -
ioengine=libaio -rw=randread -bs=4K -size=1M -numjobs=1 -runtime=60 -
group_reporting -name=cfs_test
```

写时延测试

```
fio -directory=/path/to/cfs -iodepth=1 -time_based=1 -thread -direct=1 -
ioengine=libaio -rw=randwrite -bs=4K -size=1M -numjobs=1 -runtime=60 -
group_reporting -name=cfs_test
```

读 IOPS 测试

```
fio -directory=/path/to/cfs -iodepth=1 -time_based=1 -thread -direct=1 -
ioengine=libaio -rw=randread -bs=4K -size=100M -numjobs=128 -runtime=60
-group_reporting -name=cfs_test
```

写 IOPS 测试

```
fio -directory=/path/to/cfs -iodepth=1 -time_based=1 -thread -direct=1 -
ioengine=libaio -rw=randwrite -bs=4K -size=100M -numjobs=128 -runtime=60
-group_reporting -name=cfs_test
```

读吞吐测试

```
fio -directory=/path/to/cfs -iodepth=1 -time_based=1 -thread -direct=1 -
ioengine=libaio -rw=randread -bs=1M -size=100M -numjobs=128 -runtime=60
-group_reporting -name=cfs_test
```

写吞吐测试

```
fio -directory=/path/to/cfs -iodepth=1 -time_based=1 -thread -direct=1 -
ioengine=libaio -rw=randwrite -bs=1M -size=100M -numjobs=128 -runtime=60
-group_reporting -name=cfs_test
```

FIO 参数说明

参数	参数说明
direct	表示是否使用 direct I/O。默认值：1。 • 值为1：表示使用 direct I/O，忽略客户端 I/O 缓存，数据直写或直读。 • 值为0：表示不使用 direct I/O。 **注意：** 此参数无法穿透服务端缓存。
iodepth	表示测试时的 IO 队列深度。例如 –iodepth=1表示 FIO 控制请求中的 I/O 最大个数为1。
rw	表示测试时的读写策略。您可以设置为： • randwrite：随机写。 • randread：随机读。 • read：顺序读。 • write：顺序写。 • randrw：混合随机读写。 **说明：** 通常压力测试时，都是基于随机读写。如果需要顺序读写的性能值，可根据需要调整参数。
ioengine	表示测试时 FIO 选择哪种 I/O 引擎，通常选择 libaio，保证数据 IO 的异步下发。 **注意：** 压测时如不选择 libaio，其性能瓶颈主要在于 ioengine 上，非存储侧瓶颈。
bs	表示 I/O 单元的块大小（block size）。
size	表示测试文件大小。 FIO 会将指定的文件大小全部读/写完成，然后才停止测试，除非受到其他选项（例如运行时）的限制。 如果未指定该参数，FIO 将使用给定文件或设备的完整大小。也可以将大小作为1到100之间的百分比给出，例如指定 size=20%，FIO 将使用给定文件或设备完整大小的20%空间。
numjobs	表示测试的并发线程数。
runtime	表示测试时间，即 FIO 运行时长。
group_reporting	表示测试结果显示模式。 如果指定该参数，测试结果会汇总每个进程的统计信息，而不是以不同任务来统计信息。

directory	表示待测试的文件系统挂载路径。 **说明：** 选择此参数，FIO 将默认从此路径创建出 numjobs 数量的文件进行压测。存储压测时一定要选择此参数，如果直接指定 filename 是对单个文件进行压测。
name	表示测试任务名称，可以根据实际需要设定。
thread	使用多线程的方式进行压测，而非多进程。
time_based	• 值为1： 当指定的文件大小读写完之后，自动重复 IO，直到 runtime 参数指定的时间。 • 值为0：当指定的文件大小写完之后，立即停止。 **说明：** 通常测试时指定为1，能保证测试程序在指定的时间范围内持续运行。

说明：

- 更多的关于 FIO 压测的参数，可以参考 FIO 文档 。
- 如果需要多机测试，可通过 pshell 同时执行指令即可，或者参考 FIO 文档 配置集群版的压测参数。