# 文件存储

# 实践教程

腾讯云

# 文档目录

# 实践教程
# NFS 客户端挂载参数说明

最近更新时间：2025-06-12 11:33:11

本文主要说明常见的 NFS 客户端挂载选项，旨在帮助您更好的了解不同 NFS 挂载选项参数和适用场景。本文档着重于说明参数的含义，以及可能额外调整的参数，以满足特定场景的访问需求。

> **说明：**
> 在文件系统的挂载参数页面和客户端助手界面，已提供腾讯云建议的挂载参数，您可根据推荐配置进行挂载操作。

## 常见挂载参数说明

| 参数 | 参数说明 |
| --- | --- |
| vers | 表示使用 NFS 协议的版本。<br>• 值为3：表示使用 NFS V3挂载。<br>• 值为4.0：表示使用 NFS V4挂载。<br>**说明：**<br>建议传入 vers=3，使用 NFS V3协议，以获取更好的元数据性能。如有多客户端编辑同一文件的需求，可传入 vers=4。 |
| hard/soft | • hard：当文件存储 CFS 不可用时，客户端会自动进行重试。<br>• soft：当文件存储 CFS 不可用时，客户端直接返回 EIO 的错误。<br>**说明：**<br>• 此参数默认为 hard。<br>• 建议采用默认配置，当 CFS 后端主备切换时，客户端能自动重连，保证服务可用性。 |

| | |
|---|---|
| noresvport | 在网络重连时使用新的 TCP 端口，保障在网络发生故障恢复时不会中断连接。<br>**说明：**<br>建议配置，当 CFS 后端进行主备切换时，会更换内部端口。为保证主备切换后，业务能正常恢复，需添加此参数。 |
| ac/noac | • ac：缓存文件的元数据信息。<br>• noac：不缓存文件的元数据信息。<br>**说明：**<br>• 此参数默认为 ac。<br>• 建议采用默认配置，以获得更好的性能。<br>• 若对多客户端读取数据的一致性水平要求高，常见于各种工作流的场景，可配置为 noac。 |
| sync | 配置后，数据写入将转为同步写入。<br>**说明：**<br>此参数建议不添加，以保证更好的性能。如果对一致性水平要求较高，常见于各种工作流的场景，可配置此参数。 |
| lock/nolock | • nolock：将文件锁语义转为本地锁，不报错。<br>• lock：开启文件锁语义，若服务端不支持文件锁，则锁请求会报错。<br>**说明：**<br>默认为 lock。建议在 NFS V3协议挂载的时候配置 nolock，保证当业务调用文件锁时能正常运行。 |

**说明：**
如需了解更完整的 NFS 客户端挂载参数，可参见官方链接 MAN NFS 。

# 通用型 CFS 实践指南

最近更新时间：2025-06-24 14:07:41

## 概述

本文主要基于 通用型文件系统的 NFS 协议的特性，介绍在使用时重点需要关注的使用方法和姿势，以便提供更适合用户业务场景的服务。

## 场景一：客户端内核选择

### 背景说明

NFS 客户端是基于内核态的客户端，因部分内核版本的缺陷，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。

### 常见问题及修复建议

#### 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。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 。

| 操作系统类型 | 版本号 |
| --- | --- |
| Red Hat Enterprise Linux 6、CentOS 6 | 2.6.32-696.3.1.el6 |
| Red Hat Enterprise Linux 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。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 。

| 操作系统类型 | 版本号 |
| --- | --- |
| Red Hat Enterprise Linux 6、CentOS 6 | 2.6.32-696.16.1.el6之前的所有内核版本 |
| Red Hat Enterprise Linux 7、CentOS 7 | 3.10.0-693.el7之前的所有内核版本 |

- 当操作系统内核版本为 CentOS 和 Red Hat Enterprise Linux 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，以避免此问题。

### 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或以上版本修复此问题。

## NFS 推荐使用的内核镜像

### Linux 系统镜像

| 操作系统类型 | 操作系统版本 |
| --- | --- |
| CentOS | • CentOS 7.5 64位：3.10.0-862.14.4.el7.x86_64及以上<br>• CentOS 7.6 64位：3.10.0-957.21.3.el7.x86_64及以上<br>• CentOS 7.7 64位：3.10.0-1062.18.1.el7.x86_64及以上<br>• 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)<br>• TencentOS Server 2.4 (Tkernel 4)<br>• TencentOS Server 2.6(Final)<br>• TencentOS Server 3.1(Tkernel 4) |
| Debian | • Debian 9.6 64位：4.9.0-8-amd64及以上<br>• Debian 9.8 64位：4.9.0-8-amd64及以上<br>• 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<br>• Ubuntu 16.04 64位：4.4.0-151-generic 及以上<br>• 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<br>• 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<br>• 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<br>• 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<br>• 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<br>• Windows Server 2019数据中心版64位英文版 |

## 场景二：多进程或客户端并发写

### 背景说明

NFS 文件协议不提供强一致性的文件锁定机制，文件扩展和写入均非原子操作，因此，在多进程或多客户端并发写同一个文件的场景中，各进程或客户端维护各自的文件指针，默认缓存文件数据，最终导致不同客户端或进程查看到的文件状态可能不一致。常见问题包括写覆盖（后写入覆盖前写入）、内容交叉（写入操作交叉执行）、串行异常（类似串行写入，但无法保证顺序正确）等。

### 使用建议

- **（推荐）**优化工作流，安排不同进程或客户端写入各自独立的文件，通过合并程序定期将这些文件合并为最终输出，完全避免并发写入冲突，提供最佳性能和数据一致性。
- **（推荐）**换用自动加锁/解锁、并发读写一致性更高的 CFS Turbo文件系统。
- 当不同客户端必须写入同一文件时，请采用以下多重保障措施：
  - 挂载配置优化：使用 NFSv4代替 NFSv3协议挂载，添加 noac 挂载选项，确保文件状态实时同步。挂载命令可参考：

```
mount -t nfs -o vers=4.0,proto=tcp,noac,noresvport 10.XX.XX.XX:/
/localfolder
```

- 程序层手动加锁：在写入代码中使用 flock 系统调用实现互斥锁，并使用追加写入 O_APPEND 模式打开文件，确保严格遵循“获取锁→写入→释放锁”的工作流程。在程序中手动加锁可参考：

```
import os
import fcntl

O_WRONLY = os.O_WRONLY  # 以只写方式打开文件
O_APPEND = os.O_APPEND  # 以追加方式打开文件
O_DIRECT = getattr(os, 'O_DIRECT', 0o40000)  # 尝试获取系统的
O_DIRECT 常量，如果不存在则设置为默认值 0o40000
SEEK_END = os.SEEK_END  # 以文件尾为参考点

 # 假定并发写入的文件为 file1
filename = '/root/cfs/file1'

try:
    fd = os.open(filename, O_WRONLY | O_APPEND | O_DIRECT)

    # 尝试获取文件锁
    try:
        fcntl.flock(fd, fcntl.LOCK_EX | fcntl.LOCK_NB)
        print("Successfully acquired lock")

        # 定位到文件尾
        os.lseek(fd, 0, SEEK_END)

        # 写入数据
        data = b"hello world"
        os.write(fd, data)

    except BlockingIOError:
        print("File is locked by another process")
        # 如果文件被锁，则退出
        exit(1)
    finally:
        # 释放文件锁
        fcntl.flock(fd, fcntl.LOCK_UN)
        print("Lock released")
```

```
finally:
    if 'fd' in locals():
        os.close(fd)
```

> **注意：**
> 高并发场景下，锁等待会导致明显的延迟增加，可能会显著影响文件系统读写性能，请根据业务实际需求酌情设置。

# CFS Turbo 实践指南

最近更新时间：2025-06-24 11:29:42

## 概述

本文主要基于Turbo文件系统的特性，介绍在使用时重点需要关注的使用方法和姿势，以便用户能更合适的使用。

## 场景一：目录结构规划实践

### 背景说明

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。

### 使用建议

- 当您的业务不涉及频繁删除/修改/新增的情况下，例如以读为主，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。
- 建议单个文件系统实例的目录总数控制在1500万以内，避免创建过多的目录。

### 具体方案

1. 可根据哈希或时间，对文件进行分层、分类存放，避免在一个子目录下存放过多文件。
    - 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。
2. 通过文件名的前缀来区分，避免嵌套很多层目录之后，仅存放一个文件，造成目录数远大于文件数的情况。例如：
    - 原存储方式：/cfs/1/2/3/4/5/6/file1
    - 建议存储方式：/cfs/1_2_3_4_5_6_file1

### 底层原理

- 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。
- CFS Turbo 为强一致的文件系统，其后端通过一套分布式锁的服务，来实现任意时刻数据均为一致状态。因此客户端在执行 IO 操作，会涉及到后端分布式锁的授予或召回。当大量文件集中在同一层目录下时，尤其是涉及

到频繁写操作时（写锁为互斥锁），当此目录下发生读请求后，会额外涉及到对写锁的召回和授予，进而对读取速度产生负面影响。

- CFS Turbo 底层的元数据是分布式的架构，每一个目录下的文件的元数据都是打散在内部多个对象上。因此，创建目录的资源开销会大于文件的开销，建议用户把目录数量控制在合理的范围内（单实例1500万目录以内），避免达到系统规格上限，影响使用。

# 场景二：文件系统根目录（或浅层目录）操作实践

## 背景说明

该实践主要应用在使用 Turbo 文件系统下，如何对根目录（或浅层目录）的操作进行规划。

## 使用建议

- 尽量避免对根目录或浅层目录进行大量的写操作。
- 需要避免业务侧的探活程序直接对 CFS 的根目录做写探测。

## 具体方案

1. 业务侧的程序写入分目录执行，不要在根目录或浅层目录执行高负载的写入。
2. 关闭对 CFS Turbo 的写探测，新建一个单独的子目录做写探测，或者使用腾讯云提供的探活指标进行告警监控，通过对 告警指标 中的客户端检测健康成功率来判断该实例是否正常运行。

## 底层原理

Turbo 文件系统为强一致文件系统，对根目录或浅层目录的写操作，会对根目录或浅层目录加上独占的写锁。读请求访问发生的时候，会依次查询路径上的目录项，需要拿到每一层级目录的读锁。当有大量对根目录的写探测，同时再叠加上高压力的读取请求，会产生高频且不必要的锁授予和召回的操作，会拖慢整个系统的访问速度，进而影响业务访问效率。

# 场景三：客户端销毁/关机实践

## 背景说明

该实践主要应用在需要对访问 Turbo 的客户端进行销毁或者停止时，如何进行合适的操作，避免因强制关机等问题，导致业务访问异常。

## 使用建议

- 当需要销毁客户端机器时，不要对客户端节点进行强制关机，应使用软关机的方式进行。
- 当需要卸载 Turbo 文件系统时，使用 umount 方式进行卸载，不要使用 `umount -f` 强制卸载。

## 具体方案

### 步骤1：实例关机

**方式1：调用 API 接口关机**

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。

**方式2：通过控制台操作关机**

1. 登录 云服务器控制台，选中需要关机的云服务器，单击**关机**。
2. 选择**关机方式**为**关机**。

> **说明：**
> - 调用 API 关机时，请勿将 StopType 选择 hard 模式，此模式为强制关机。
> - 在控制台上关机时，请勿选择强制关机，选择关机即可，此选项默认为 SOFT_FIRST。
> - 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。

### 步骤2：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。

## 底层原理

CFS Turbo 为强一致文件系统，通过分布式锁的机制，保证客户端数据的一致性。如果客户端未通过正常途径就断开和服务端的连接，会导致此客户端持有的锁无法正常释放，进而影响其他客户端访问对应的文件。

> **说明：**
> 若异常断开后，客户端的锁会在一个超时时间范围后释放（默认为2分钟），多其他客户端最多会造成2分钟的 io hang。为了避免不必要的 io hang，建议通过上文提到的 SOFT_FIRST 方式进行关机。

# 在容器 TKE 上使用 CFS

最近更新时间：2025-05-23 15:18:01

## 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。

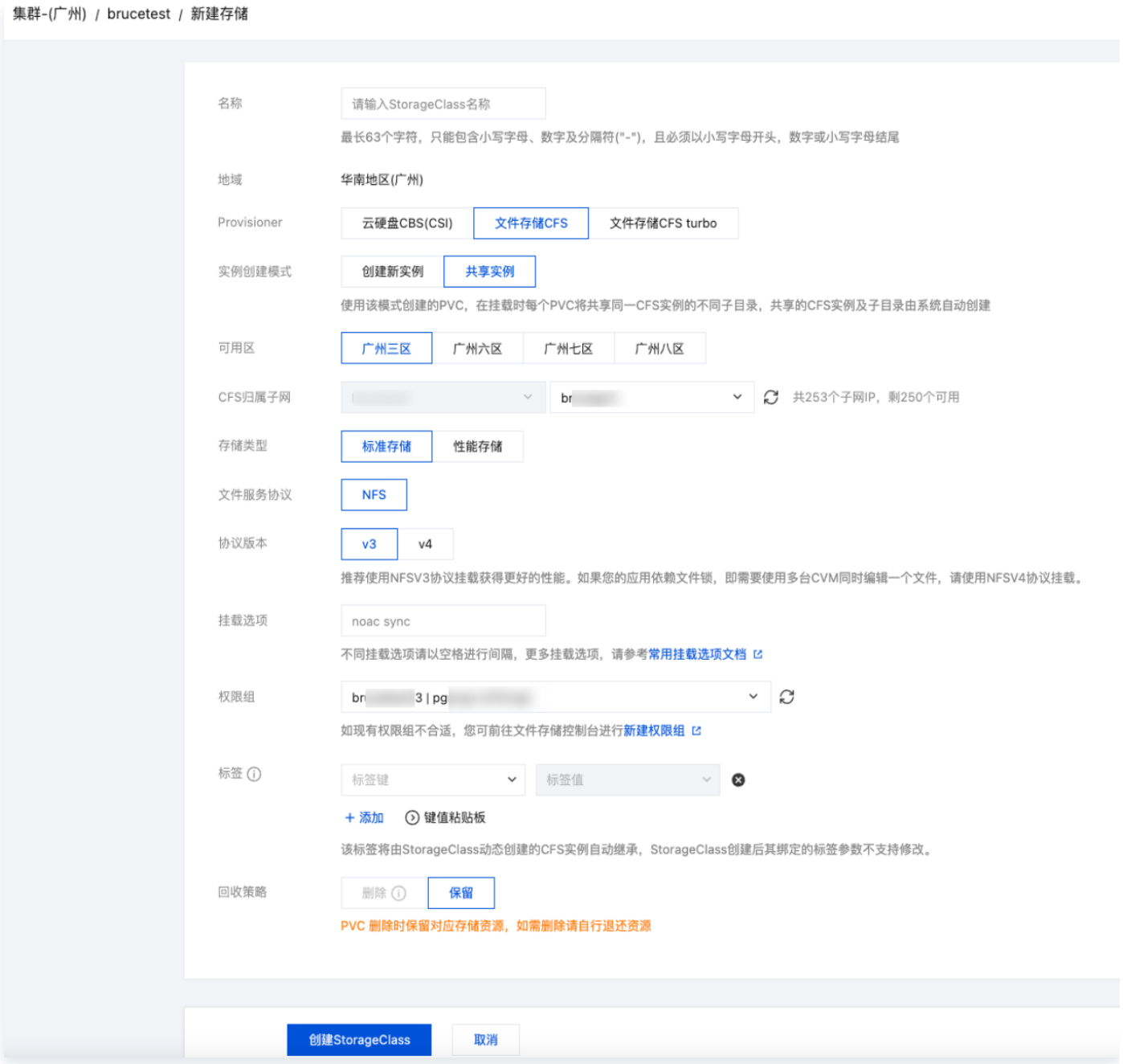
> **注意：**
> 使用前，需确保容器服务（Tencent Kubernetes Engine，TKE）集群的 CSI 组件在1.0.4版本以上。若版本不对，可在 TKE 控制台上进行更新，CSI 组件的更新不影响容器的正常使用。

## 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。



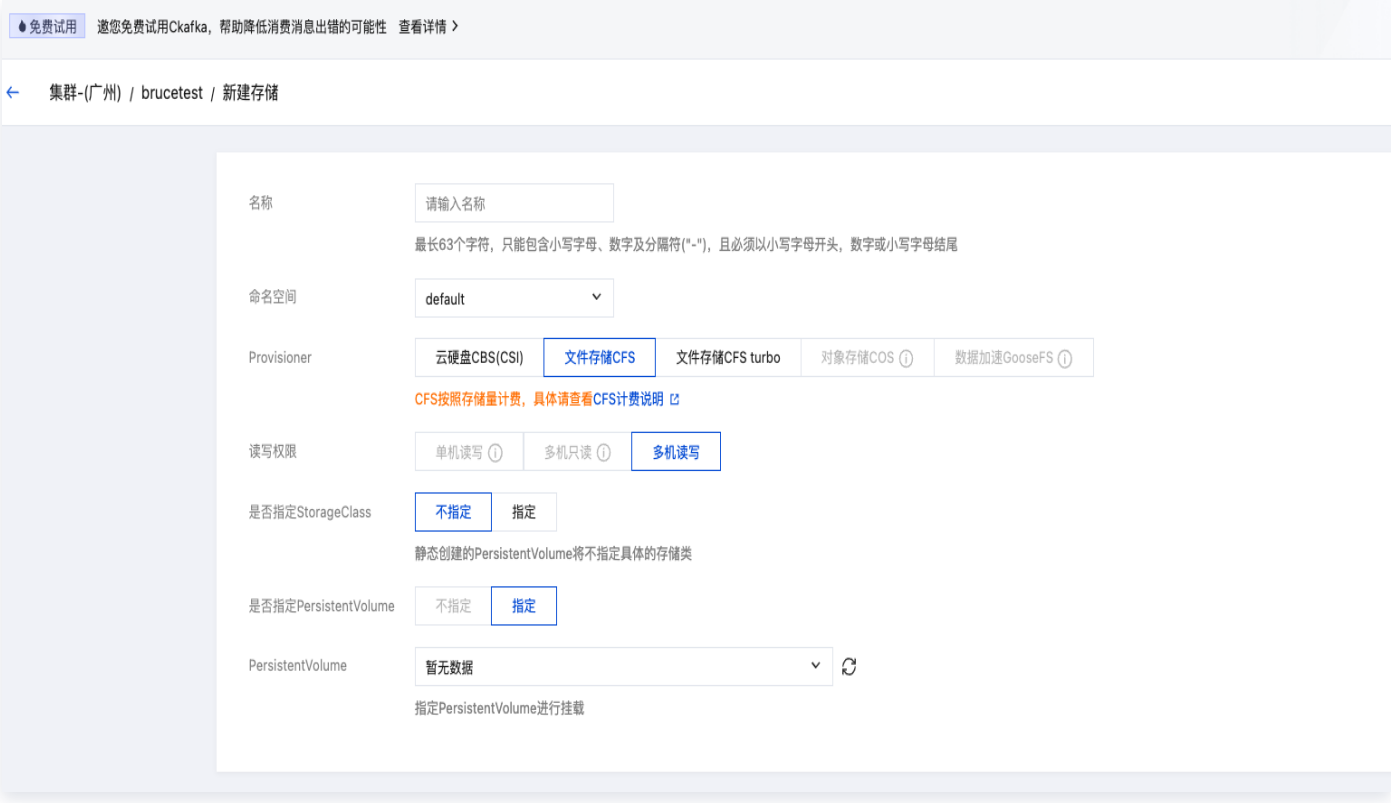

**相关的关键配置项如下：**

| 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。 |
| 挂载选项 | 根据实际情况，选择合适的挂载参数，默认可保持为空。 |
| 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。 |

> **说明：**
> 详细参数请参见 StorageClass 管理文件存储模板

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 StorageClass。 |
| 是否指定 PersistentVolume | 动态创建可不指定 PV。<br>说明：基于 CFS 共享型实例的 StorageClass，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：

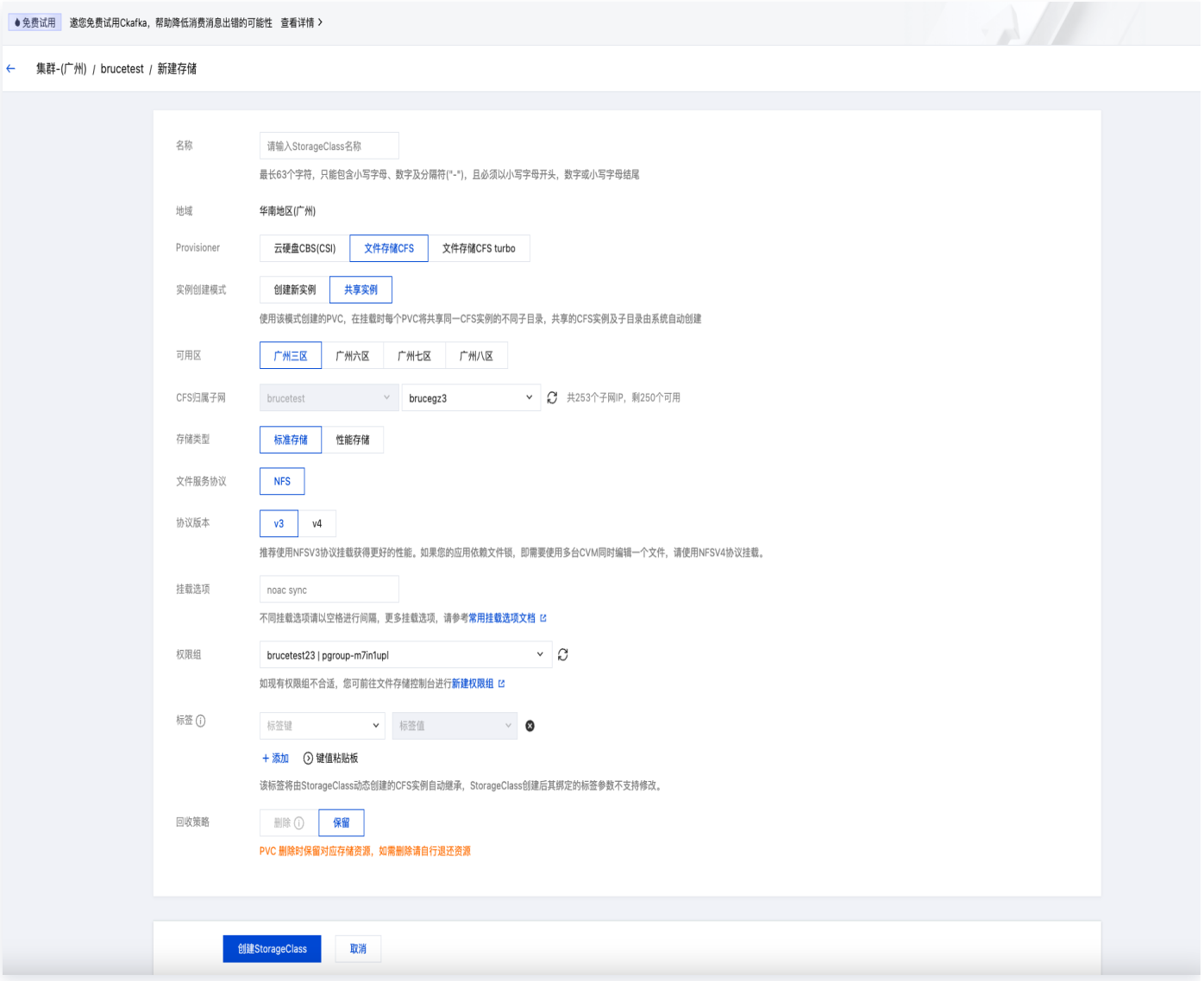
| 配置项 | 配置项说明 |
| --- | --- |
| 数据卷 | 根据实际需求，对数据卷进行命名，并选中刚创建的 PVC。 |
| 挂载点 | 选择对应的数据卷，指定在容器本地的挂载路径。在共享 CFS 实例的动态创建方式下，需要指定具体的环境变量，CSI 插件会基于配置的环境变量的变量值，在所选 PVC 对应的 CFS 实例里创建目录供容器挂载。 |

4. 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。 |

| | |
|---|---|
| 挂载选项 | 根据实际情况，选择合适的挂载参数，默认可保持为空。 |
| 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。 |

> **说明：**
> 详细参数请参见 StorageClass 管理文件存储模板 。

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：

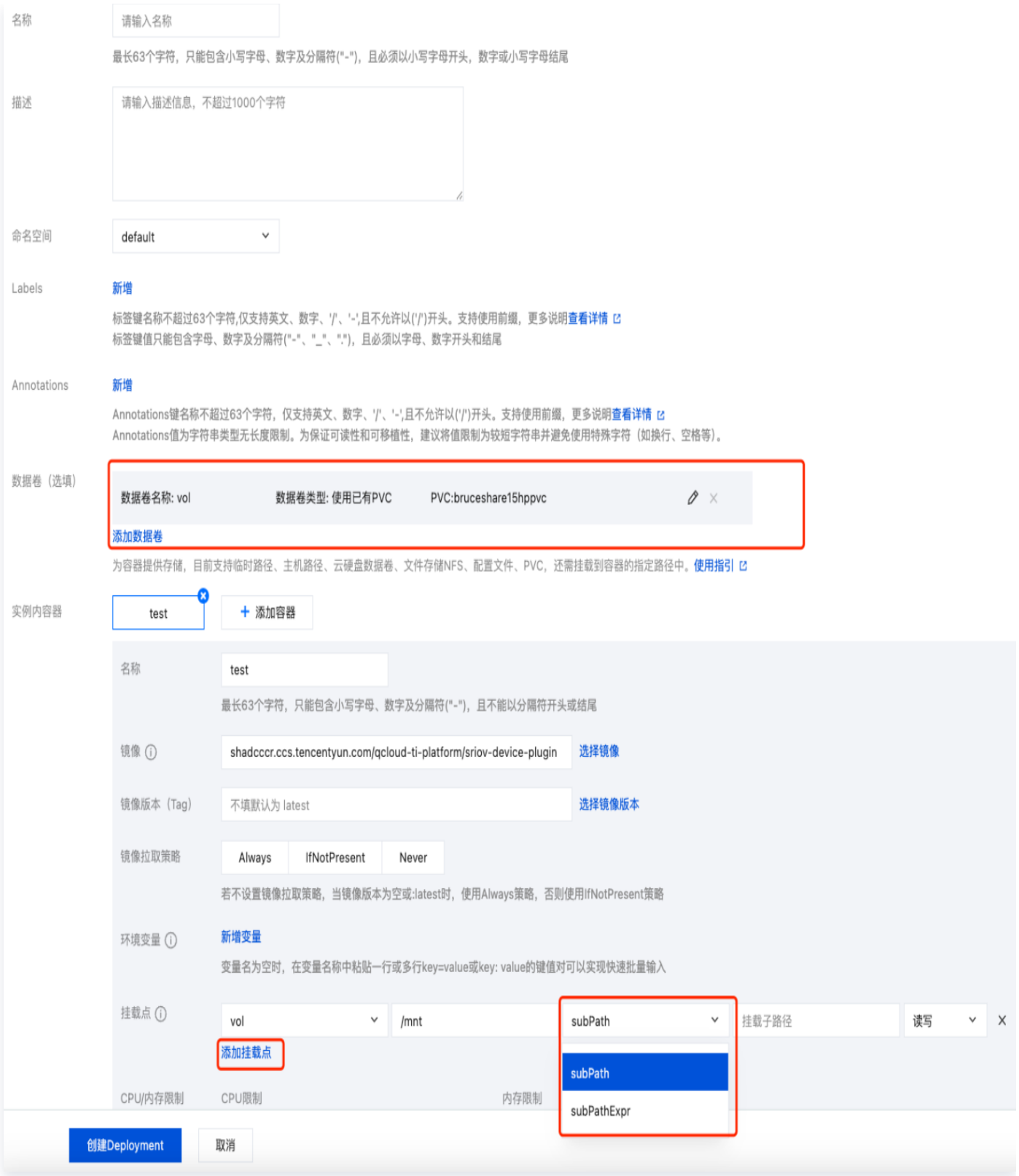
| 配置项 | 配置项说明 |
|---|---|
| 来源设置 | 选择静态创建，即指定某个 CFS 实例配置 PV。 |
| StorageClass | 选择刚才创建的 StorageClass。 |
| 选择 CFS | 选择一个指定的 CFS。说明：静态创建时需要保证已经有一个 CFS 实例，同时该 CFS 实例需与容器在同一个 VPC 网络环境。 |

| | |
|---|---|
| CFS 子目录 | 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 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。


| 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 StorageClass。 |
| 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 | 根据实际需求，对数据卷进行命名，并选中刚创建的 PVC。 |

| | |
|---|---|
| 卷 | |
| 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。<br>• 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。<br>• 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：2025-06-11 17:13:02

## 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。
云函数 SCF 使用文件存储 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的 CFS 资源的权限。如您使用的账号未赋予云函数权限，可能出现函数无法保存，CFS 相关功能无法使用等问题。

请参考以下步骤为账号进行授权操作：

- 如您的账号为主账号，请参见 修改角色 为 `SCF_QcsRole` 角色关联 `QcloudCFSReadOnlyAccess` 策略。
  关联成功则如下图所示：

- 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 `QcloudCFSReadOnlyAccess` 策略。关联成功则如下图所示：

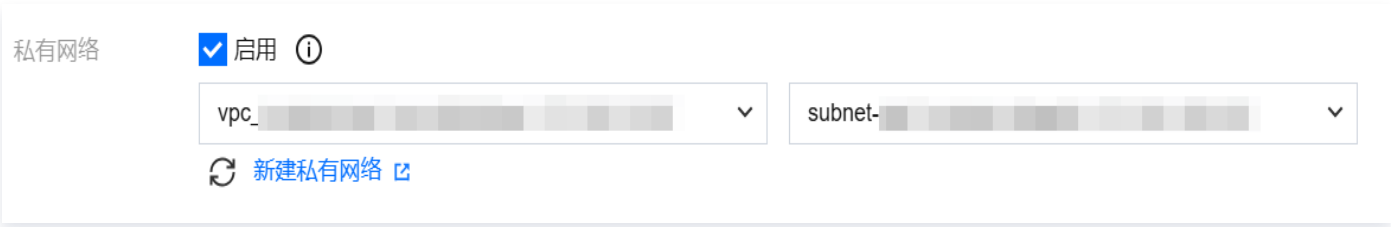

## 创建 CFS 文件系统

具体操作请参考 创建 CFS 文件系统 。

> **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，单击需要使用 CFS 存储的函数名，进入**函数管理**页面。
3. 选择**函数配置**页签，单击右上角的**编辑**。
4. 在**网络配置**中，勾选启用并选择 CFS 文件系统所在的 VPC。

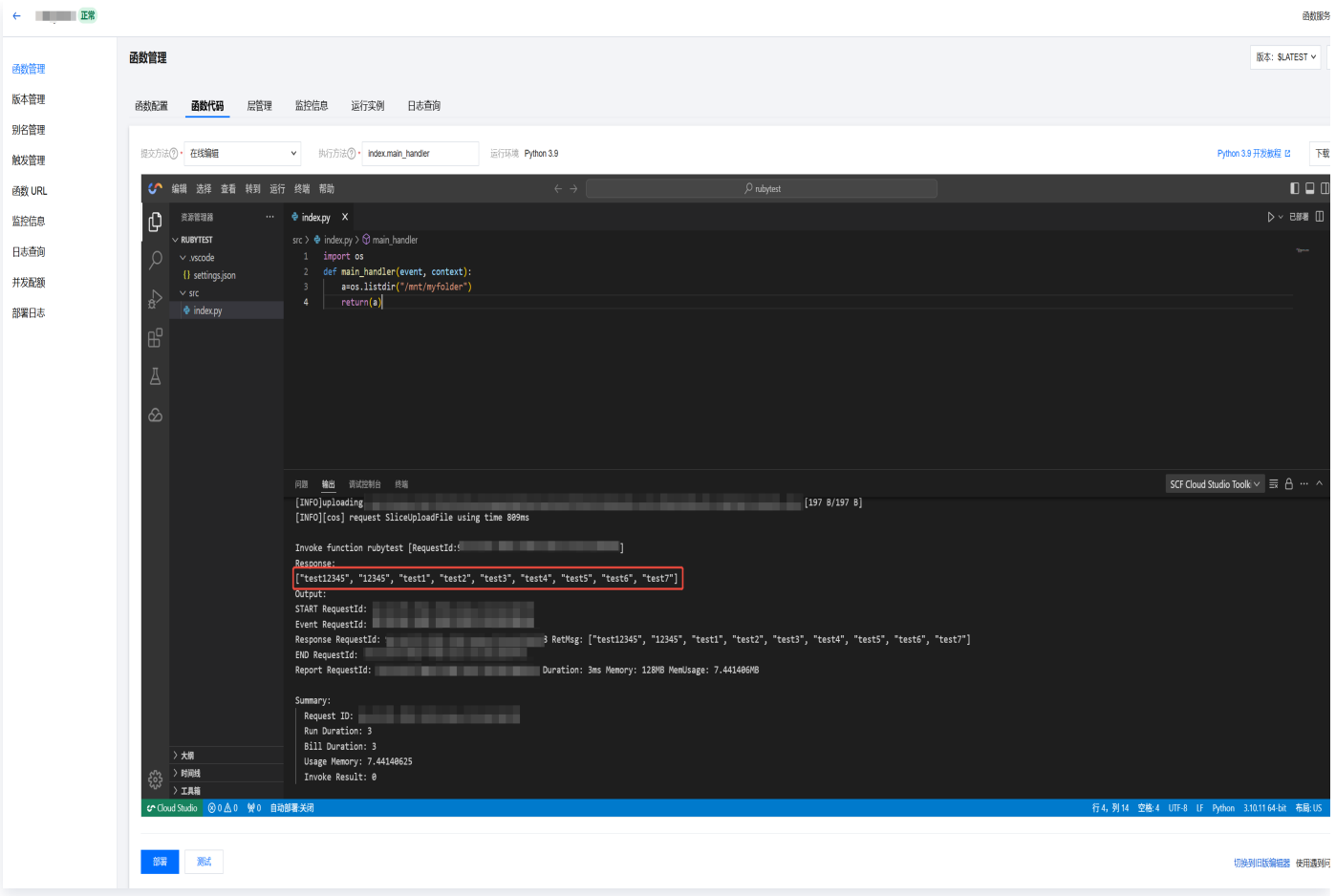


5. 在**文件系统**中单击**添加文件系统**，并按照以下信息进行挂载。如下图所示：


- **文件系统类型**：目前仅支持挂载使用 CFS 通用系列，CFS Turbo 系列正在灰度中。
- **文件系统 ID**：在下拉列表中选择需挂载的文件系统。如果您未提前创建，可以**新建文件系统**。
- **挂载点 ID**：在下拉列表中选择对应文件系统的挂载点 ID。
- **远程目录**：云函数需要访问 CFS 文件系统的远端目录，由文件系统和远端子目录两部分组成，远端子目录路径必须存在。
- **本地目录**：为本地文件系统的挂载路径，推荐挂载到 `/home/` 、`/mnt/` 或 `/data/` 目录的子目录挂载。

6. 单击页面下方的**保存**即可完成配置。您在 SCF 中对本地目录执行的读写操作，等同于在远端目录进行读写操作。
7. 为了验证是否挂载成功，您可以在**函数管理** > **函数代码**中部署并测试以下程序，观察远程目录的目录结构是否被成功返回。

```
import os
def main_handler(event, context):
    a=os.listdir("/mnt")
    return(a)
```




# SCF 使用 CFS 文件系统性能测试

您可以使用此 脚本 测试云函数 SCF 使用 CFS 时的读写性能。

# 在容器 TKE 上使用 CFS Turbo

最近更新时间：2025-06-24 14:09:01

## 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。

## 操作步骤

### 安装并设置 CFS 扩展组件

1. 进入 TKE 集群列表页，单击目标 TKE 集群 ID，进入集群详情页。
2. 在左侧选择**组件管理**，在组件页面单击**新建**。
3. 勾选**存储** > **CFSTurbo**，单击**完成**即可创建组件。

### 创建 PersistentVolume

1. 进入 TKE 集群列表页，单击使用 CFS 的 TKE 集群 ID，进入集群详情页。
2. 在左侧选择**存储**，在 **PersistentVolume** 页面单击**新建**，配置参数如下：
   - **来源设置**：选择**静态创建**。
   - **名称**：填写名称。
   - **Provisioner**：选择 **文件存储 CFS Turbo**。
   - **读写权限**：文件存储 CFS Turbo 仅支持**多机读写**。
   - **是否指定 StorageClass**：选择**不指定**，对应 YAML 文件中的 `storageClassName` 字段取值为空字符串。
   - 填写 CFS Turbo 根目录和子目录。


3. 单击**创建 PersistentVolume**，即可完成创建。

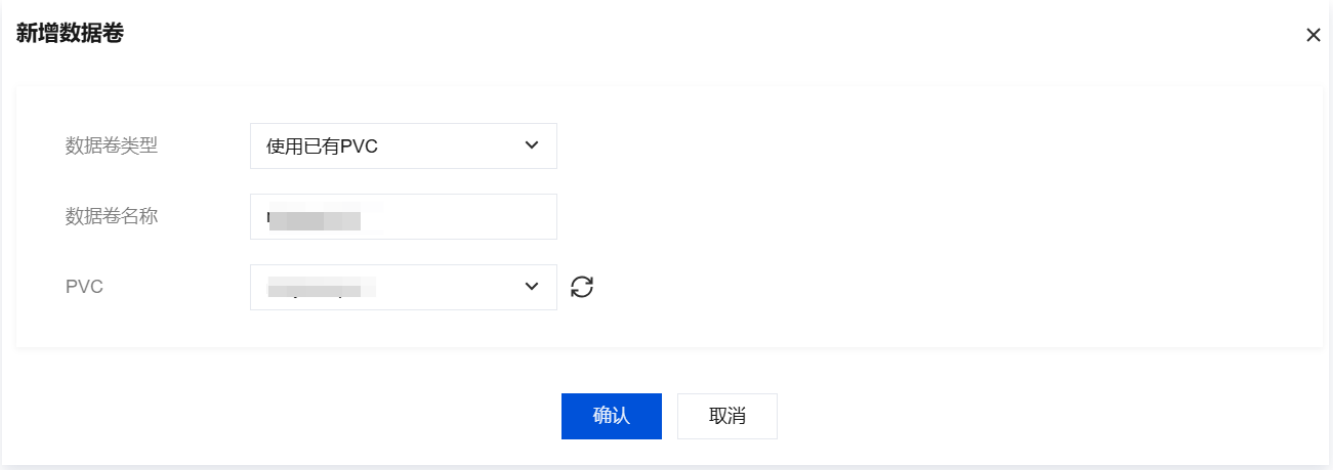
## 创建 PersistentVolumeClaim

1. 进入 TKE 集群列表页，单击使用 CFS 的 TKE 集群 ID，进入集群详情页。
2. 在左侧选择**存储**，在 **PersistentVolumeClaim** 页面单击**新建**，配置参数如下：
   - **Provisioner**：选择**文件存储 CFS Turbo**。
   - **读写权限**：文件存储 CFS Turbo 仅支持**多机读写**。
   - **是否指定 StorageClass**：如果当前集群内没有满足条件的 PV 可选，请选择**不指定** StorageClass。
   - **是否指定 PersistentVolume**：选择**指定**，并关联上一步创建的 **PersistentVolume**。
     PersistentVolume 需要与指定的 StorageClass 相同并且状态为 Available 和 Released。

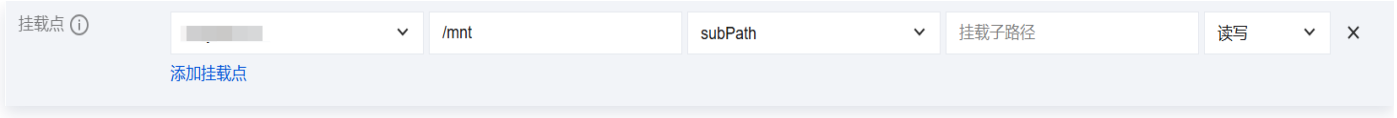

3. 单击**创建 PersistentVolumeClaim**，即可完成创建。

## 创建工作负载

1. 进入 TKE 集群列表页，单击使用 CFS 的 TKE 集群 ID，进入集群详情页。
2. 在左侧选择**工作负载**，在**Deployment**页面单击**新建**。
3. 在新建工作负载页面单击**添加数据卷**。
4. 在新增数据卷弹窗中，数据卷类型选择**使用已有 PVC**，并选择上述已创建的 PVC，填写数据卷名称，单击**确认**。

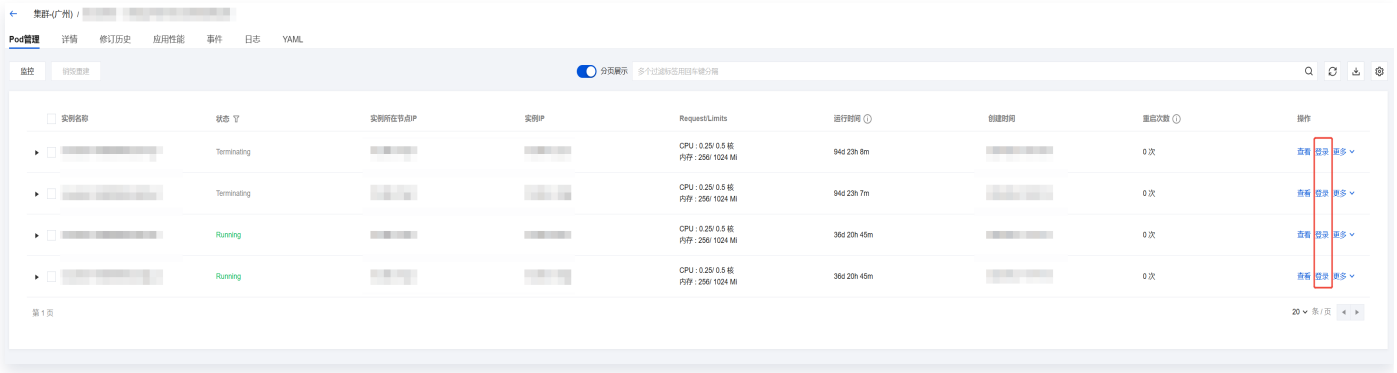



5. 在**实例内容器**的**挂载点**中，单击**添加挂载点**，选择上方创建的数据卷，设置数据卷挂载到容器中的路径。

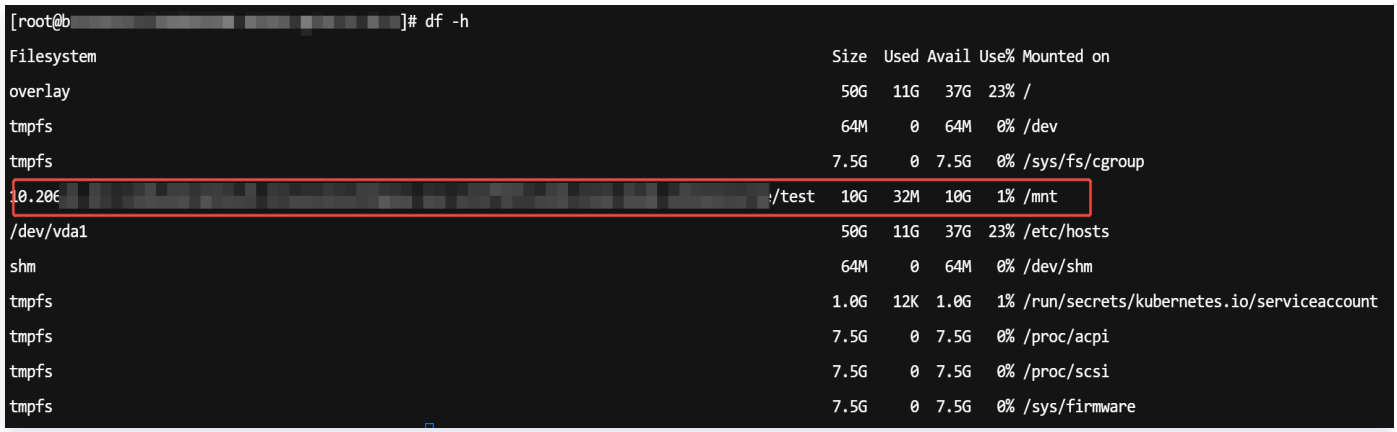



## 登录容器查看挂载情况

1. 进入 TKE 集群列表页，单击使用 CFS 的 TKE 集群 ID，进入集群详情页。
2. 在左侧选择**工作负载**，在 **Deployment** 页面单击新建的工作负载名称。
3. 进入 **Pod 管理**页面，在操作栏单击**登录**。



4. 在**容器登录**弹窗中，单击 **OrcaTerm 登录**。登录后执行 `df -h` 即可看到挂载的 CFS 文件系统。




# 补充：基于自建容器使用 CFS Turbo

## 前提条件

## 容器宿主机节点满足 Turbo 系列兼容的操作系统。

- 已在所有容器宿主机节点安装 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。

## 配置 kubectl 连接集群

您可参见 TKE 连接集群，配置 kubectl，以管理容器集群。

## 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。
5. 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: xxxxxxxx
      # 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.xx.xx@tcp0:/xxxxxxxx/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。
- rootdir：/cfs，请保持此参数不要进行修改。
- fsid：xxxxxxxx，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"/"。

6. 在上传脚本文件的目录下，依次执行以下命令。
   - 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
     ```

7. 执行以下命令，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   [redacted]  [redacted]    <none>           <none>
```

若状态（STATUS）持续为 ContainerCreating，即表示创建失败。您可进入 TKE 集群列表页，单击使用 CFS 的 TKE 集群 ID，进入集群详情页。在左侧菜单栏单击**事件**，查看失败原因。

# Turbo 文件系统网络选择

最近更新时间：2025-03-19 14:47:02

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。<br>• 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。<br>• 可以更方便的跨 VPC 访问 Turbo 文件存储。<br>• 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。

> ⚠ **注意：**
> Turbo 占用的 IP 数量较多，在每次扩容时都会涉及到占用新的 IP，而使用 VPC 网络的 Turbo，可能会在子网 IP 不足的情况下出现无法扩容的问题，因此不建议使用此方式创建。如确认需要使用此方式创建，可在谨慎评估后，提交工单 与我们联系。

| 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。<br>• 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。<br>• 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：2025-06-27 15:34:31

## 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 。
  - **方法二：**当需要通过命令行工具时，推荐使用本文提到的工具进行操作，详情请参见 操作说明 。
- **场景二：文件系统与对象存储之间数据同步。**
  - **方法一：**当涉及需要把文件存储 CFS 和对象存储进行数据拷贝，可使用 数据迁移服务 。

    > **说明：**
    > 此服务仅支持对象存储数据迁移至文件存储 CFS 中，如需文件传输到对象存储，可参考下述的方法二。

  - **方法二：**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：2025-06-12 11:33:11

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

| 指标 | 定义 | 业内常用测试方案 |
| --- | --- | --- |
| 时延 | 从发出一个 IO 请求到接收到该请求的响应所花费的时间，单位 ms。 | 设置1MiB大小的文件，通过单流（单线程）的方式，每次传输4KiB的小数据块，获取读写延时。 |
| IOPS | 每秒读写文件块（IO 操作）的数量，单位个/s。 | 设置100MiB大小的文件，通过并发（多机多线程）的方式，每次传输4KiB的小数据块，获取 IOPS。 |
| 吞吐 | 每秒读写数据量的大小，单位 GiB/s或MiB/s。 | 设置100MiB大小的文件，通过并发（多机多线程）的方式，每次传输1MiB或4MiB的大数据块，获取吞吐。 |

## 转换公式

- 核心公式：吞吐 = IOPS * 读写块大小
- IOPS 计算方式可参考：IOPS ≈ 队列深度 / 时延；IOPS = 读写总次数 / 任务整体延迟。后者仅适用于顺序无并发场景。
- 任务整体延迟受并发模型、设备队列、带宽瓶颈共同影响，当并发请求超过性能上限，请求排队将导致任务整体延迟上升。

**说明：**

- 吞吐、 IOPS 、整体时延相互关联，不同应用场景下的观测重点不同。例如：
  - 随机小IO场景（读写块大小为4KiB、8KiB）应重点关注 IOPS 和低时延；
  - 顺序大IO场景（读写块大小为128KiB、1MiB及以上）应重点关注吞吐和网络带宽；
  - 对响应速度敏感的场景应重点关注时延。
- 低时延是实现高 IOPS 和高吞吐的关键前提，高时延会限制实际能够达到的 IOPS 和吞吐，操作等待响应的时间越长，单位时间内能完成的操作或传输的数据量就越少。
- 您可以通过调整读写块大小以及并发数，实现目标吞吐，并避免触及 IOPS 上限，降低整体任务延迟。例如通用标准型 CFS 4KiB写时延约为7ms：
  - 当并发数为1，写入次数为250，则整体时延为7ms*250=1.75s，IOPS为 250/1.75s≈143个/s，吞吐为143个/s*4KiB=571KiB/s；

- 当并发数为128，写入次数为250，则整体时延为7ms*(250/128)≈13.7ms，IOPS为250/13.7ms≈18286个/s，吞吐为18285个/s*4KiB≈71MiB/s。

## 注意事项

- 压测时，因服务端缓存加速的特性，命中缓存时，读性能会超过标准值，此现象符合预期。
- 压测时，尤其是时延测试，需要保证 CVM（客户端）和 CFS 处于同可用区。受网络延迟影响，跨可用区性能结果会和标准值有较大差异，需尽可能避免。
- 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。以吞吐为例，最大不会超过 CVM 内网带宽，如果内网带宽太小，吞吐会被流量限制。例如，购买20T的 Turbo 标准型 CFS，初始带宽为2GiB/s，配套使用 CVM 实例规格 为 SA3.LARGE8，内网带宽（出+入）为2Gbps，则单机访问吞吐最大为0.25GiB/s，如果想要测到最高吞吐，则至少需要8台同规格 CVM 并发测试。
- FIO 工具压测时模拟正常文件的读写，不会对文件系统造成损坏，但测试时请不要使用生产环境进行压测，避免对您的业务造成不必要的影响。

## 操作步骤

1. 远程登录云服务器 CVM 实例，具体操作可参考 Linux 云服务器配置及登录 。
2. 安装压测软件
   - CentOS/TencentOS：

```
sudo yum install fio
```

   - Ubuntu/Debian：

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

   - Windows：请下载msi格式的安装包并安装，FIO安装包下载地址请参见fio官网 。
3. 运行性能测试命令

## Linux 系统下的性能测试命令

本节以通用性能型 CFS 为例，在单台 Linux CVM 中执行以下命令测试性能。

**说明：**

- 实际测试时，需要将 -directory 中的 /path/to/cfs 替换成实际测试的文件系统路径。
- 读写测试中，测试软件FIO均会根据测试命令的不同，构造对应的测试数据，建议使用一个空目录进行测试。

- **读时延测试**
  - 测试命令

```
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
```

  - 结果样例

```
fio-3.28
Starting 1 thread
cfs_test: Laying out IO file (1 file / 1MiB)
Jobs: 1 (f=1): [r(1)][100.0%][r=22.5MiB/s][r=5755 IOPS][eta 00m:00s]
cfs_test: (groupid=0, jobs=1): err= 0: pid=3039221: Tue Jun  3 12:08:28 2025
  read: IOPS=5757, BW=22.5MiB/s (23.6MB/s)(1349MiB/60001msec)
    slat (nsec): min=899, max=82277, avg=2449.18, stdev=1192.70
    clat (usec): min=132, max=4437, avg=170.76, stdev=32.27
     lat (usec): min=152, max=4446, avg=173.29, stdev=32.30
    clat percentiles (usec):
     |  1.00th=[  159],  5.00th=[  161], 10.00th=[  161], 20.00th=[  163],
     | 30.00th=[  165], 40.00th=[  167], 50.00th=[  167], 60.00th=[  169],
     | 70.00th=[  172], 80.00th=[  174], 90.00th=[  180], 95.00th=[  188],
     | 99.00th=[  229], 99.50th=[  247], 99.90th=[  506], 99.95th=[  611],
     | 99.99th=[ 1631]
   bw (  KiB/s): min=19264, max=23680, per=100.00%, avg=23034.70, stdev=501.04, samples=119
   iops        : min= 4816, max= 5920, avg=5758.67, stdev=125.26, samples=119
  lat (usec)   : 250=99.57%, 500=0.32%, 750=0.07%, 1000=0.01%
  lat (msec)   : 2=0.02%, 4=0.01%, 10=0.01%
  cpu          : usr=0.56%, sys=2.33%, ctx=346479, majf=0, minf=2
  IO depths    : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued rwts: total=345456,0,0,0 short=0,0,0,0 dropped=0,0,0,0
     latency   : target=0, window=0, percentile=100.00%, depth=1

Run status group 0 (all jobs):
   READ: bw=22.5MiB/s (23.6MB/s), 22.5MiB/s-22.5MiB/s (23.6MB/s-23.6MB/s), io=1349MiB (1415MB), run=60001-60001msec
```

- **写时延测试**
  - 测试命令

```
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
```

- 结果样例

```
fio-3.28
Starting 1 thread
cfs_test: Laying out IO file (1 file / 1MiB)
Jobs: 1 (f=1): [w(1)][100.0%][w=6712KiB/s][w=1678 IOPS][eta 00m:00s]
cfs_test: (groupid=0, jobs=1): err= 0: pid=3040293: Tue Jun  3 12:11:18 2025
  write: IOPS=1657, BW=6629KiB/s (6788kB/s)(388MiB/60001msec); 0 zone resets
    slat (nsec): min=1059, max=69968, avg=2777.64, stdev=1395.83
    clat (usec): min=479, max=13293, avg=600.14, stdev=159.52
     lat (usec): min=482, max=13297, avg=603.00, stdev=159.54
    clat percentiles (usec):
     |  1.00th=[  506],  5.00th=[  519], 10.00th=[  529], 20.00th=[  553],
     | 30.00th=[  570], 40.00th=[  570], 50.00th=[  578], 60.00th=[  586],
     | 70.00th=[  603], 80.00th=[  611], 90.00th=[  652], 95.00th=[  701],
     | 99.00th=[ 1106], 99.50th=[ 1450], 99.90th=[ 2245], 99.95th=[ 2737],
     | 99.99th=[ 5342]
   bw (  KiB/s): min= 4976, max= 6912, per=99.99%, avg=6628.24, stdev=238.97, samples=119
   iops        : min= 1244, max= 1728, avg=1657.06, stdev=59.74, samples=119
  lat (usec)   : 500=0.27%, 750=96.38%, 1000=2.03%
  lat (msec)   : 2=1.16%, 4=0.12%, 10=0.03%, 20=0.01%
  cpu          : usr=0.17%, sys=0.64%, ctx=100026, majf=0, minf=1
  IO depths    : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued rwts: total=0,99434,0,0 short=0,0,0,0 dropped=0,0,0,0
     latency   : target=0, window=0, percentile=100.00%, depth=1

Run status group 0 (all jobs):
  WRITE: bw=6629KiB/s (6788kB/s), 6629KiB/s-6629KiB/s (6788kB/s-6788kB/s), io=388MiB (407MB), run=60001-60001msec
```

- **读 IOPS 测试**
  - 测试命令

```
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
```

- 结果样例

```
cfs_test: (g=0): rw=randread, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=1
...
fio-3.28
Starting 128 threads
Jobs: 128 (f=128): [r(128)][100.0%][r=14.0MiB/s][r=3585 IOPS][eta 00m:00s]
cfs_test: (groupid=0, jobs=128): err= 0: pid=3067161: Tue Jun  3 13:20:45 2025
  read: IOPS=3639, BW=14.2MiB/s (14.9MB/s)(854MiB/60036msec)
    slat (nsec): min=910, max=1240.0k, avg=3516.34, stdev=15894.79
    clat (usec): min=1160, max=47855, avg=35150.64, stdev=4246.28
     lat (usec): min=1219, max=47858, avg=35154.25, stdev=4244.15
    clat percentiles (usec):
     |  1.00th=[ 5669],  5.00th=[34341], 10.00th=[34866], 20.00th=[35390],
     | 30.00th=[35390], 40.00th=[35390], 50.00th=[35914], 60.00th=[35914],
     | 70.00th=[35914], 80.00th=[36439], 90.00th=[36439], 95.00th=[36963],
     | 99.00th=[36963], 99.50th=[36963], 99.90th=[38536], 99.95th=[40109],
     | 99.99th=[44827]
   bw (  KiB/s): min=13408, max=42927, per=100.00%, avg=14566.82, stdev=20.43, samples=15270
   iops        : min= 3352, max=10731, avg=3641.70, stdev= 5.11, samples=15270
  lat (msec)   : 2=0.02%, 4=0.04%, 10=1.90%, 20=0.02%, 50=98.03%
  cpu          : usr=0.00%, sys=0.01%, ctx=226463, majf=0, minf=251
  IO depths    : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued rwts: total=218521,0,0,0 short=0,0,0,0 dropped=0,0,0,0
     latency   : target=0, window=0, percentile=100.00%, depth=1

Run status group 0 (all jobs):
   READ: bw=14.2MiB/s (14.9MB/s), 14.2MiB/s-14.2MiB/s (14.9MB/s-14.9MB/s), io=854MiB (895MB), run=60036-60036msec
```

- **写 IOPS 测试**
  - 测试命令

```
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
```

- 结果样例

```
Jobs: 128 (f=128): [w(128)][100.0%][w=11.2MiB/s][w=2859 IOPS][eta 00m:00s]
cfs_test: (groupid=0, jobs=128): err= 0: pid=3067991: Tue Jun  3 13:22:39 2025
  write: IOPS=2877, BW=11.2MiB/s (11.8MB/s)(675MiB/60043msec); 0 zone resets
    slat (nsec): min=909, max=306294, avg=3350.72, stdev=5571.21
    clat (usec): min=1101, max=268273, avg=44196.76, stdev=8965.59
     lat (usec): min=1158, max=268275, avg=44200.21, stdev=8965.47
    clat percentiles (msec):
     |  1.00th=[   10],  5.00th=[   41], 10.00th=[   42], 20.00th=[   43],
     | 30.00th=[   44], 40.00th=[   44], 50.00th=[   45], 60.00th=[   46],
     | 70.00th=[   47], 80.00th=[   47], 90.00th=[   48], 95.00th=[   50],
     | 99.00th=[   51], 99.50th=[   54], 99.90th=[  155], 99.95th=[  264],
     | 99.99th=[  268]
   bw (  KiB/s): min= 7616, max=32886, per=100.00%, avg=11555.03, stdev=21.36, samples=15191
   iops        : min= 1904, max= 8220, avg=2888.73, stdev= 5.34, samples=15191
  lat (msec)   : 2=0.01%, 4=0.12%, 10=1.02%, 20=1.10%, 50=96.12%
  lat (msec)   : 100=1.48%, 250=0.09%, 500=0.06%
  cpu          : usr=0.00%, sys=0.01%, ctx=187180, majf=0, minf=128
  IO depths    : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued rwts: total=0,172782,0,0 short=0,0,0,0 dropped=0,0,0,0
     latency   : target=0, window=0, percentile=100.00%, depth=1

Run status group 0 (all jobs):
  WRITE: bw=11.2MiB/s (11.8MB/s), 11.2MiB/s-11.2MiB/s (11.8MB/s-11.8MB/s), io=675MiB (708MB), run=60043-60043msec
```

- **读吞吐测试**
  - 测试命令

```
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
```

  - 结果样例

```
cfs_test: (groupid=0, jobs=128): err= 0: pid=3060587: Tue Jun  3 13:03:54 2025
  read: IOPS=918, BW=919MiB/s (963MB/s)(53.9GiB/60118msec)
    slat (usec): min=24, max=13207, avg=70.77, stdev=272.10
    clat (nsec): min=950, max=318885k, avg=138980775.12, stdev=43037481.52
     lat (msec): min=7, max=318, avg=139.05, stdev=43.01
    clat percentiles (msec):
     |  1.00th=[  110],  5.00th=[  112], 10.00th=[  113], 20.00th=[  115],
     | 30.00th=[  116], 40.00th=[  116], 50.00th=[  117], 60.00th=[  118],
     | 70.00th=[  122], 80.00th=[  180], 90.00th=[  205], 95.00th=[  234],
     | 99.00th=[  279], 99.50th=[  288], 99.90th=[  305], 99.95th=[  305],
     | 99.99th=[  317]
   bw (  KiB/s): min=407358, max=1343800, per=99.99%, avg=940685.07, stdev=2097.11, samples=15265
   iops        : min=  376, max= 1310, avg=914.30, stdev= 2.05, samples=15265
  lat (nsec)   : 1000=0.01%
  lat (msec)   : 10=0.01%, 20=0.02%, 50=0.10%, 100=0.32%, 250=95.97%
  lat (msec)   : 500=3.57%
  cpu          : usr=0.00%, sys=0.04%, ctx=116159, majf=0, minf=32640
  IO depths    : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued rwts: total=55234,0,0,0 short=0,0,0,0 dropped=0,0,0,0
     latency   : target=0, window=0, percentile=100.00%, depth=1

Run status group 0 (all jobs):
   READ: bw=919MiB/s (963MB/s), 919MiB/s-919MiB/s (963MB/s-963MB/s), io=53.9GiB (57.9GB), run=60118-60118msec
```

- **写吞吐测试**
  - **测试命令**

```
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
```

  - **结果样例**

```
Jobs: 128 (f=128): [w(128)][100.0%][w=208MiB/s][w=208 IOPS][eta 00m:00s]
cfs_test: (groupid=0, jobs=128): err= 0: pid=3066287: Tue Jun  3 13:18:48 2025
  write: IOPS=203, BW=204MiB/s (214MB/s)(12.1GiB/60571msec); 0 zone resets
    slat (usec): min=34, max=3237, avg=73.83, stdev=75.49
    clat (msec): min=9, max=686, avg=510.87, stdev=146.92
     lat (msec): min=9, max=687, avg=510.95, stdev=146.92
    clat percentiles (msec):
     |  1.00th=[   31],  5.00th=[  224], 10.00th=[  300], 20.00th=[  384],
     | 30.00th=[  460], 40.00th=[  523], 50.00th=[  575], 60.00th=[  600],
     | 70.00th=[  617], 80.00th=[  634], 90.00th=[  642], 95.00th=[  651],
     | 99.00th=[  667], 99.50th=[  676], 99.90th=[  684], 99.95th=[  684],
     | 99.99th=[  684]
   bw (  KiB/s): min=228975, max=676103, per=100.00%, avg=288402.16, stdev=851.38, samples=10874
   iops        : min=  147, max=  659, avg=279.47, stdev= 0.84, samples=10874
  lat (msec)   : 10=0.02%, 20=0.45%, 50=0.97%, 100=0.47%, 250=4.91%
  lat (msec)   : 500=29.18%, 750=64.00%
  cpu          : usr=0.00%, sys=0.01%, ctx=17523, majf=0, minf=128
  IO depths    : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued rwts: total=0,12353,0,0 short=0,0,0,0 dropped=0,0,0,0
     latency   : target=0, window=0, percentile=100.00%, depth=1

Run status group 0 (all jobs):
  WRITE: bw=204MiB/s (214MB/s), 204MiB/s-204MiB/s (214MB/s-214MB/s), io=12.1GiB (13.0GB), run=60571-60571msec
```

## 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。<br>• 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。<br>• 值为0：表示不使用 direct I/O。<br>**⚠ 注意：**<br>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。 |

| | |
|---|---|
| | **说明：**<br>您可以通过逐步增加队列深度，通过观察 IOPS 是否随并发数有所上升，判断性能是否达到瓶颈；通过观察平均时延是否陡增，观察并发能力极限。 |
| rw | 表示测试时的读写策略。您可以设置为：<br>• randwrite：随机写。<br>• randread：随机读。<br>• read：顺序读。<br>• write：顺序写。<br>• randrw：混合随机读写。<br>**说明：**<br>压测一般基于随机读写。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。<br>**注意：**<br>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。<br>**注意：**<br>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：**<br>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。<br>• 值为0：当指定的文件大小写完之后，立即停止。<br>**说明：**<br>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，或者参考 FIO 文档 配置集群版的压测参数。