对象存储

数据湖存储

产品文档

腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2024 腾讯云版权所有

本文档著作权归腾讯云单独所有，未经腾讯云事先书面许可，任何主体不得以任何形式复制、修改、抄袭、传播全部或部分本文档内容。

【商标声明】

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。

【服务声明】

本文档意在向客户介绍腾讯云全部或部分产品、服务的当时的整体概况，部分产品、服务的内容可能有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

文档目录

- 数据湖存储
  - 云原生数据湖
    - 快速入门
  - 元数据加速
    - 元数据加速功能概述
    - 将 HDFS 数据迁移到已开启元数据加速功能的存储桶
    - 使用 HDFS 协议访问已开启元数据加速的存储桶
    - 在计算集群中挂载 COS 存储桶
    - 在 CDH 集群上通过 HDFS 协议访问 COS
    - 使用 Hadoop Filesystem API 代码访问 COS 元数据加速存储桶
    - 使用 DataX 在已开启元数据加速功能的存储桶间同步数据
    - 大数据安全
      - Ranger 介绍说明
  - 数据加速器 GooseFS
    - 产品概述
    - 更新日志
    - 快速入门
    - 核心特性
      - 缓存能力
      - 透明加速能力
      - 统一命名空间能力
      - Table 管理能力
      - GooseFS-FUSE 能力
    - 部署指南
      - 使用自建集群部署
      - 使用腾讯云 EMR 部署
      - 使用腾讯云 TKE 部署
      - 使用 Docker 部署
    - 运维指南
      - 日志指引
        - GooseFS 日志介绍
      - 监控指南
        - 获取 GooseFS 监控指标
        - 基于 Prometheus 搭建 GooseFS 监控体系
      - 集群配置实践

版权所有：腾讯云计算（北京）有限责任公司

数据安全
使用 Apache Ranger 控制 GooseFS 的访问权限
GooseFS 接入 Kerberos 认证

数据湖存储
云原生数据湖
快速入门

最近更新时间：2024-01-06 11:17:58

简介

云原生数据湖存储服务可以帮助您快速在容器服务（Tencent Kubernetes Engine，TKE）上部署一套基于对象存储（Cloud Object Storage，COS）的数据湖存储服务。您可以通过云原生数据湖存储服务，快速在一个 TKE 或者 EKS 集群上部署各类业务所需的大数据服务应用和 AI 服务应用，同时通过数据加速器 GooseFS 对接海量分布式存储服务。

概念和术语

使用云原生数据湖存储服务，您可以阅读以下说明初步了解相关的**概念和术语**：

环境：用于维护云原生数据湖存储中计算集群和存储服务之间的映射关系，推荐您统一从这入口管理计算集群和存储服务。

注意

如果您需要删除计算集群，建议您先清除云原生数据湖存储环境，然后在容器服务控制台上删除计算集群。

计算集群：用于运行各类计算业务的容器集群，您可以创建 TKE 集群或者 EKS 集群。

存储服务：特指对象存储服务，用于存储各类用于计算业务的数据。

应用市场：用于运行各类计算业务的应用组件，例如 Flink、Spark 等，您可以在创建环境时按需选择所需的应用。

注意

当您的容器集群被销毁时，您部署的应用也会被销毁。请谨慎进行删除操作。

数据加速器 GooseFS：可用于纳管不同的底层存储桶，并将热点数据缓存在计算集群中，加速您的计算业务。

您还可以通过如下文档预先了解一些基础信息：

对象存储 COS 服务：您可以通过 快速入门 了解如何创建存储桶并在存储桶中上传和下载文件。

容器服务：您可以通过 快速入门了解如何创建一个 TKE 集群或者 EKS 集群。

应用市场：您可以通过容器服务提供的 应用市场 了解如何在集群中创建并部署应用。

数据加速器 GooseFS：该服务可用于统一纳管不同的底层存储桶，并加速您的业务访问。

前提条件

当前云原生数据湖存储服务属于白名单能力，如果您需要使用，请 联系我们 开白使用。

云原生数据湖存储服务依赖容器服务和对象存储服务，您在使用过程中需要确保拥**有权限操作计算服务和存储服务**。如果您使用子账号登录，请确保该子账号至少拥有如下权限：

对象存储服务的存储桶和文件操作权限：

存储桶操作权限：如果需要管理存储桶配置，请联系主账号获取对应存储桶配置的操作权限；一般情况下该存储桶配置权限不影响数据读写，无需额外配置，最多授权读配置操作权限即可（例如 QcloudCOSBucketConfigRead 策略集）。

文件操作权限：一般情况下计算作业需要读写存储桶中的文件，可以联系主账号授权文件全读写权限（例如 QcloudCOSDataFullControl）或者由主账号按照 最小权限原则 授权。

容器集群的管理权限：

集群操作权限：一般情况需要授权集群的创建和操作权限，您可以参考 使用 TKE 预设策略授权 完成配置。

集群管理权限：TKE 提供了对接 Kubernetes RBAC 的授权模式，便于对子账号进行细粒度的访问权限控制，子账号操作时还需要参考TKE Kubernetes 对象级权限控制。

应用市场操作权限：应用市场依赖了镜像仓库操作，您可以参考 TKE 镜像仓库资源级权限设置 为子账号完成授权。

操作步骤

完整的操作步骤大致分为：创建环境、关联集群、部署计算应用、关联存储服务、管理环境等关键步骤，具体操作指引如下。

1. 登录 对象存储控制台。
2. 在左侧边导航栏中，单击**云原生数据湖存储**，进入云原生数据湖存储服务界面。
3. 在云原生数据湖存储服务界面中，页面视图会展示能力介绍、部署指引两部分内容：

我们会默认为您显示部署指引，您可以单击右上方的**收起指引**关闭指引导航。

云原生数据湖存储环境列表展示页面支持搜索。对于已存在环境，您可以进行如下操作：

单击**环境名称**，进入环境详情页面管理环境。

单击**关联集群**，打开 TKE 控制台进入对应集群详情页面。

单击关联存储桶，进入存储桶页面查看桶里的文件信息。

4. 单击**创建环境**，进入环境创建流程。

创建环境需要先选择对应的容器计算集群，其需要配置如下参数：

环境名称：用于标记环境信息，最长支持63个字符，全局唯一。

地域：选择容器集群的地域信息。

集群类型：可选 TKE 集群和 EKS 集群，如果当前地域下无集群，您可以单击**创建容器集群**，前往容器服务控制台新建集群。

集群：在**指定地域**和**指定集群类型**的条件下，用于部署计算应用服务、运行计算作业的集群名称。

计算应用：运行计算作业所需的应用服务，当前默认支持了 Flink、big-data-suite、colocation、airflow、pytorch 和 spark-operator 等应用，您可以按需选择；如果您需要部署自定义应用，可以前往容器服务控制台上自行部署。应用支持多选。

版权所有：腾讯云计算（北京）有限责任公司

5. 单击**下一步**，进入到**存储桶配置**页面视图。

您可以在该页面下为计算集群配置不同的存储桶，我们默认提供了数据加速器 GooseFS 服务，用于纳管不同存储桶并将数据缓存到计算集群的本地节点，用于加速计算作业。其需要配置如下参数：

地域信息：无法编辑，默认跟随计算集群所选地域。该地域下如果没有可选存储桶，您可以单击**创建存储桶**，新建一个存储桶用于计算任务使用。

存储桶：支持选择指定地域下的多个存储桶。支持只挂载存储桶中的某个文件目录。

注意

如果是挂载整个存储桶，那么无需输入第二个输入框；如果需要指定目录，可以通过输入目录名称来实现，格式形如 `prefix/*` 。

启用 GooseFS：GooseFS 服务用于加速计算作业性能，默认启用，无法更改。不会产生额外的费用消耗。

6. 单击**下一步**，进入到 **GooseFS 应用配置**页面视图。

由于在数据湖环境下，所有的计算任务均需要通过 GooseFS 服务来访问 COS，因此需要为 GooseFS 配置有权限访问指定存储桶的 secretId 和 secretKey。

7. 单击**下一步**，确认信息。

8. 如果您需要修改配置项，可单击**修改**对配置信息进行更改。确认无误后，单击**创建环境**，即可完成创建环境操作。**返回环境列表并刷新**，即可看到新建的云原生数据湖存储环境。

如果您需要**删除环境**，可以在环境列表单击**删除**，并在弹窗中确认本次删除操作即可。

9. 单击列表环境名称，可以进入**基本信息**页面。

我们使用了三个卡片视图分别描述环境信息、计算集群信息、存储桶信息。

数据湖环境信息：用于展示环境的名称、地域、关联的计算集群、存储服务和创建时间等信息。

计算集群信息：用于展示计算集群的名称、节点数量、CPU、内存、GPU 用量等基础信息。如果需要了解计算集群详情，您可以单击查看详情，跳转到容器服务控制台查看。

存储桶信息：用于展示计算集群绑定的存储桶名称、文件路径和 GooseFS 使用状态。如果您需要查看存储服务详情，可以单击查看详情查看。

以上步骤全部完成后，您即可完成一个数据湖环境创建流程。

版权所有：腾讯云计算（北京）有限责任公司

元数据加速
元数据加速功能概述

最近更新时间：2024-01-06 11:17:58

元数据加速功能是由腾讯云对象存储（Cloud Object Storage，COS）服务提供的高性能文件系统功能。元数据加速功能底层采用了云 HDFS 卓越的元数据管理功能，支持用户通过文件系统语义访问对象存储服务，系统设计指标可以达到百 GB 级别带宽、十万级 QPS 以及毫秒级延迟。存储桶在开启元数据加速功能后，可以广泛应用于大数据、高性能计算、机器学习、AI 等场景。

未开启元数据加速功能的存储桶，默认没有元数据访问加速的效果，不支持通过 POSIX 文件语义访问，在文件 LIST 性能上存在瓶颈，且不支持文件 RENAME 操作。开启元数据加速功能后，既可以通过原生的对象存储 API 以 POSIX 文件语义访问存储桶中的文件，也可以通过部署在客户端的 Hadoop 工具、控制台或者 SDK 等方式以 POSIX 文件语义访问存储桶中的文件。

注意

元数据加速功能只能在创建存储桶时开启，开启后不支持关闭，请结合您的业务情况**慎重考虑**是否开启。

当前元数据加速功能为公测功能，请 联系我们 申请公测资格。

使用限制

如下表格展示了目前存储桶开启元数据加速功能后，相关产品功能的支持情况：

指标项	开启元数据加速	未开启元数据加速
存储桶创建/查询/删除操作	支持	支持
对象上传/下载/删除操作	支持	支持
存储桶权限	支持	支持
对象权限	默认继承存储桶权限	支持
同城容灾	支持	支持
存储桶加密	不适用	支持
对象加密	不适用	支持
跨域访问 CORS	不适用	支持
防盗链	不适用	支持
版本控制	不适用	支持

版权所有：腾讯云计算（北京）有限责任公司

存储桶复制	不适用	支持
静态网站	不适用	支持
回源设置	不适用	支持
生命周期	支持	支持
清单功能	不适用	支持
存储桶标签	不适用	支持
COS Select	不适用	支持
COS Batch	不适用	支持
数据万象功能	不适用	支持

计费说明

目前元数据加速功能为公测功能，暂不进行商业化收费，后续如有进一步收费说明将以站内信、邮件、短信等方式通知到您。您可以关注您的 站内信 或者 计费概述，了解最新的计费动态。

将 HDFS 数据迁移到已开启元数据加速功能的存储桶

最近更新时间：2024-01-06 11:17:58

简介

元数据加速器是由腾讯云对象存储（Cloud Object Storage，COS）服务提供的高性能文件系统功能。元数据加速器底层采用了云 HDFS 卓越的元数据管理功能，支持用户通过文件系统语义访问对象存储服务，系统设计指标可以达到百 GB 级别带宽、10万级 QPS 以及 ms 级延迟。存储桶在开启元数据加速器后，可以广泛应用于大数据、高性能计算、机器学习、AI 等场景。有关元数据加速器的详细介绍，请参见 元数据加速器。
通过元数据加速服务，COS 提供了 Hadoop 文件系统的语义，因此利用 Hadoop Distcp 工具 可以方便地在对象存储 COS 与其他 Hadoop 文件系统之间进行双向的数据迁移，本文重点介绍如何通过 Hadoop Distcp 工具将本地 HDFS 中的文件搬迁到 COS 元数据加速存储桶中。

迁移环境准备

迁移工具准备

1. 下载下表中的 jar 包工具，并且放置到迁移集群提交机的本地目录下，例如 /data01/jars。

腾讯云 EMR 环境

安装说明

jar 包文件名	说明	下载地址
cos-distcp-1.12-3.1.0.jar	COSDistCp 相关包，拷贝数据到 COSN	可参见 COSDistCp 工具
chdfs_hadoop_plugin_network-2.8.jar	OFS 插件	点击下载

自建 Hadoop/CDH 等环境

软件依赖

Hadoop-2.6.0及以上版本、Hadoop-COS 插件8.1.5及以上版本，同时 cos_api-bundle 插件版本与 Hadoop-COS 版本对应，详情请参见 COSN github releases 确认。

安装说明

在 Hadoop 环境下，安装以下插件：

jar 包文件名	说明	下载地址

cos-distcp-1.12-3.1.0.jar	COSDistCp 相关包，拷贝数据到 COSN	可参见 COSDistCp 工具
chdfs_hadoop_plugin_network-2.8.jar	OFS 插件	点击下载
Hadoop-COS	Version >= 8.1.5	可参见 Hadoop-COS 工具
cos_api-bundle	版本需与 Hadoop-COS 对应	点击下载

注意

Hadoop-cos 自8.1.5版本开始支持 `cosn://bucketname-appid/` 方式访问元数据加速桶；
元数据加速功能只能在创建存储桶时开启，开启后不支持关闭，请结合您的业务情况慎重考虑是否开启，同时注意旧版本的 Hadoop-cos 包不能正常访问已开启元数据加速功能的存储桶。

2. 创建元数据加速存储桶，并配置元数据加速桶 HDFS 协议。详细步骤可参见 使用 HDFS 协议访问已开启元数据加速器的存储桶 中的“创建存储桶并配置 HDFS 协议”章节。

3. 修改迁移集群 `core-site.xml` ，修改完成后下发配置到所有的节点上。如果只是迁移数据，则不用重启大数据组件。

key	value	配置文件
fs.cosn.trsf.fs.ofs.impl	com.qcloud.chdfs.fs.CHDFSHadoopFileSystemAdapter	core-site.xml
fs.cosn.trsf.fs.AbstractFileSystem.ofs.impl	com.qcloud.chdfs.fs.CHDFSDelegateFSAdapter	core-site.xml
fs.cosn.trsf.fs.ofs.tmp.cache.dir	格式形如 /data/emr/hdfs/tmp/	core-site.xml
fs.cosn.trsf.fs.ofs.user.appid	客户 COS bucket 对应得 appid	core-site.xml
fs.cosn.trsf.fs.ofs.ranger.enable.flag	false	core-

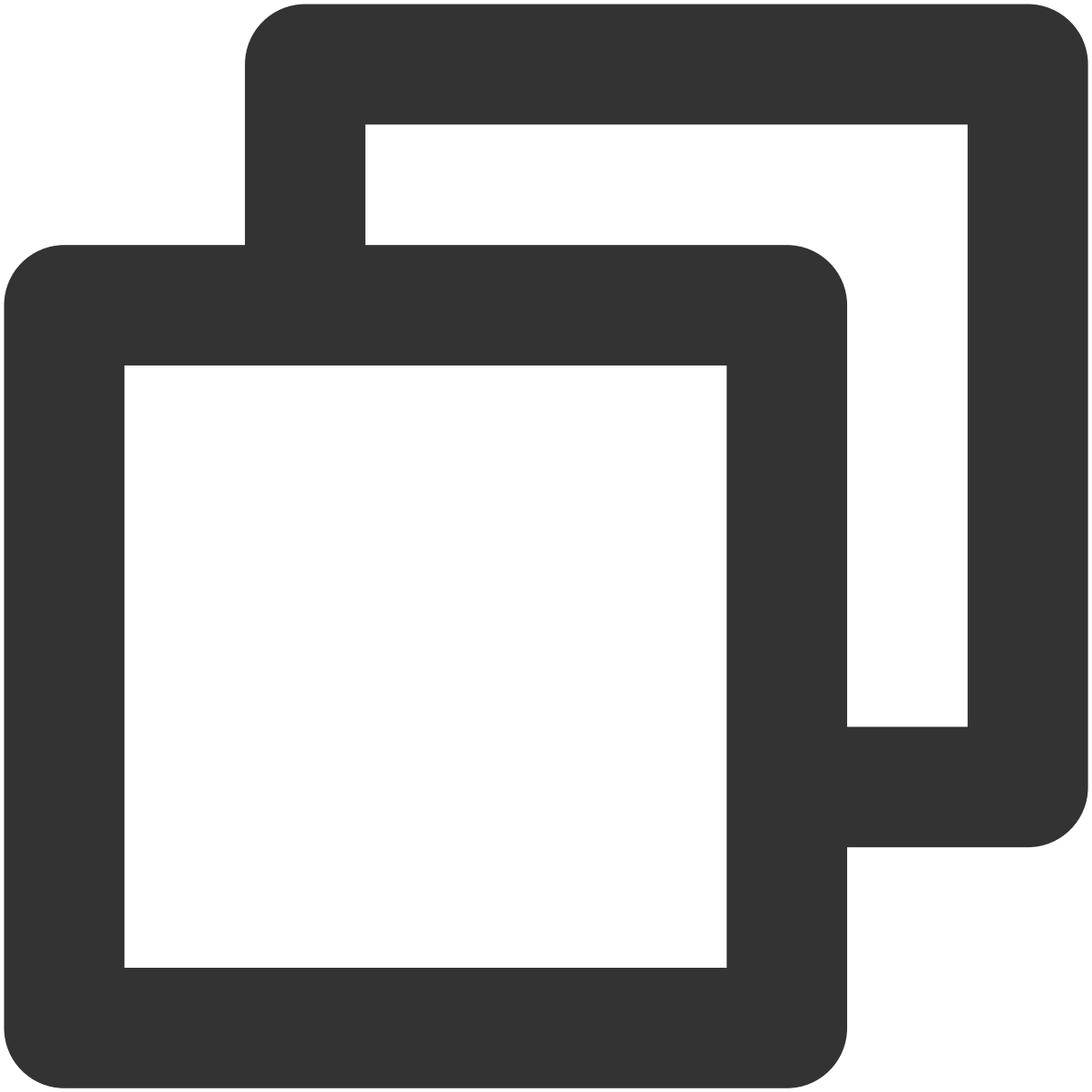
		site.xml
fs.cosn.trsf.fs.ofs.bucket.region	bucket 对应 region	core-site.xml

4. 验证迁移集群可以通过内网访问元数据加速存储桶，详细步骤可参见 使用 HDFS 协议访问已开启元数据加速器的存储桶 中的“配置计算集群访问 COS”章节。通过迁移集群提交验证是否能成功访问 COS。

存量迁移

1. 确定迁移目录

一般情况下，迁移数据会先从 HDFS 存储数据开始迁移，会选定源 HDFS 集群待迁移的目录，目标段保持和源路径相同，如下所示：
假设需要将 HDFS 路径 `hdfs:///data/user/target` 迁移到 `cosn://{bucketname-appid}/data/user/target` 。
为了保证迁移过程中，源端目录的文件不发生改变，会采用 HDFS 的快照功能，先给待迁移的目录打上快照，以当前日期作为快照文件名。



```
hdfs dfsadmin -disallowSnapshot hdfs:///data/user/
hdfs dfsadmin -allowSnapshot hdfs:///data/user/target
hdfs dfs -deleteSnapshot hdfs:///data/user/target {当前日期}
hdfs dfs -createSnapshot hdfs:///data/user/target {当前日期}
```

参考成功示例：

```
[hadoop@172 ~/alantong]$ hdfs dfsadmin -disallowSnapshot hdfs:///data/user/
Disallowing snaphot on hdfs:///data/user/ succeeded
[hadoop@172 ~/alantong]$ hdfs dfsadmin -allowSnapshot hdfs:///data/user/target
Allowing snaphot on hdfs:///data/user/target succeeded
[hadoop@172 ~/alantong]$ hdfs dfs -deleteSnapshot hdfs:///data/user/target/ 20220527
deleteSnapshot: Cannot delete snapshot 20220527 from path /data/user/target: the snapshot does not exist.
[hadoop@172 ~/alantong]$ hdfs dfs -createSnapshot hdfs:///data/user/target 20220527
Created snapshot /data/user/target/.snapshot/20220527
```

如果不想做快照，可直接以源端 target 文件进行迁移。

2. 使用 COSDistCp 迁移

启动 COSDistCp 任务，将文件从源 HDFS 复制到目标 COS 桶上。

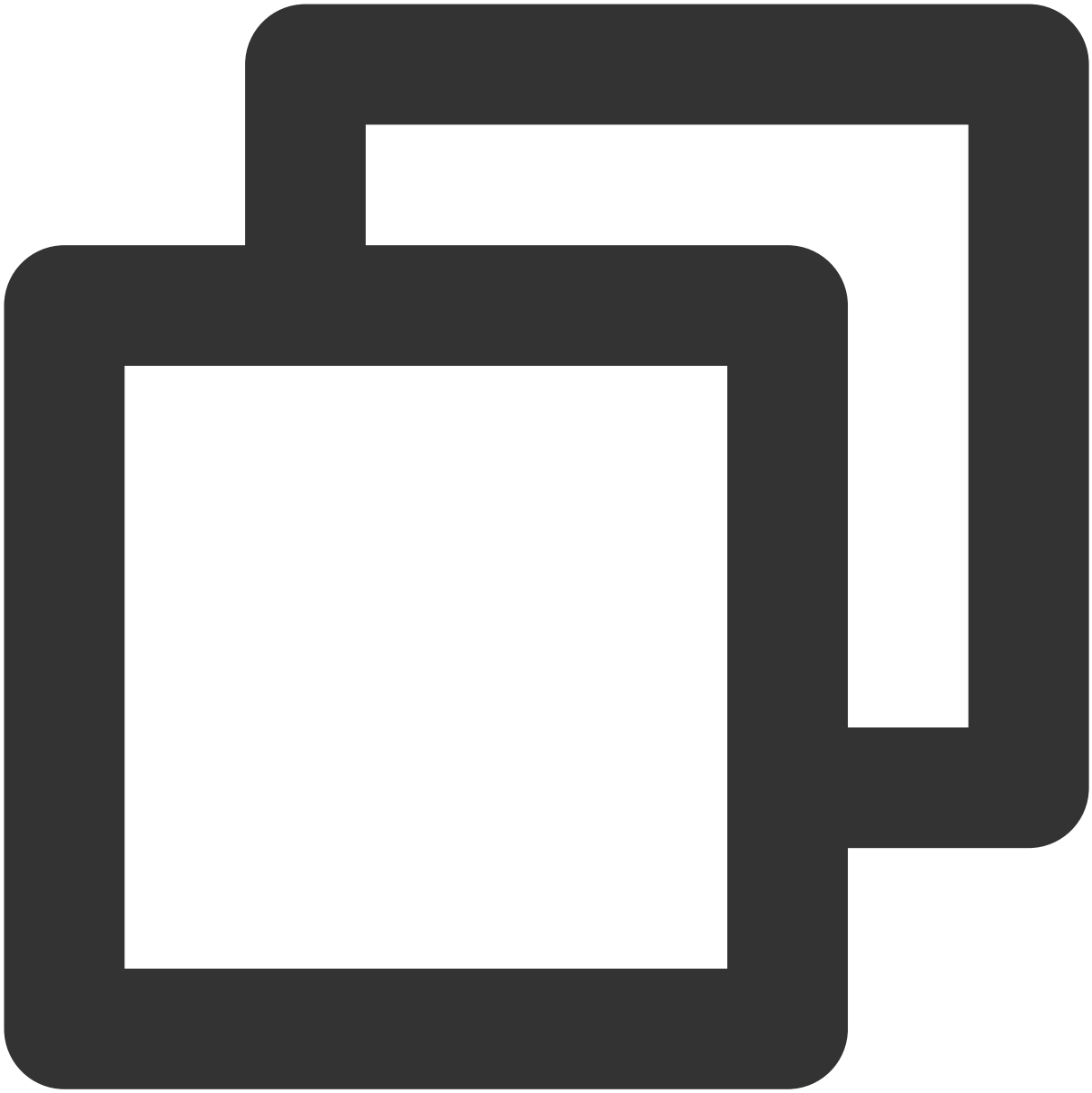
COSDistCp 为 MapReduce 任务，MapReduce 任务打印日志中会提示 MR 任务执行成功与否。如果失败可查看 YARN 页面，将日志或异常信息提供到 COS 进行排查。通过 COSDistCp 工具执行迁移任务分为如下几个步骤：

（1）创建临时目录

（2）运行 COSDistCp 任务

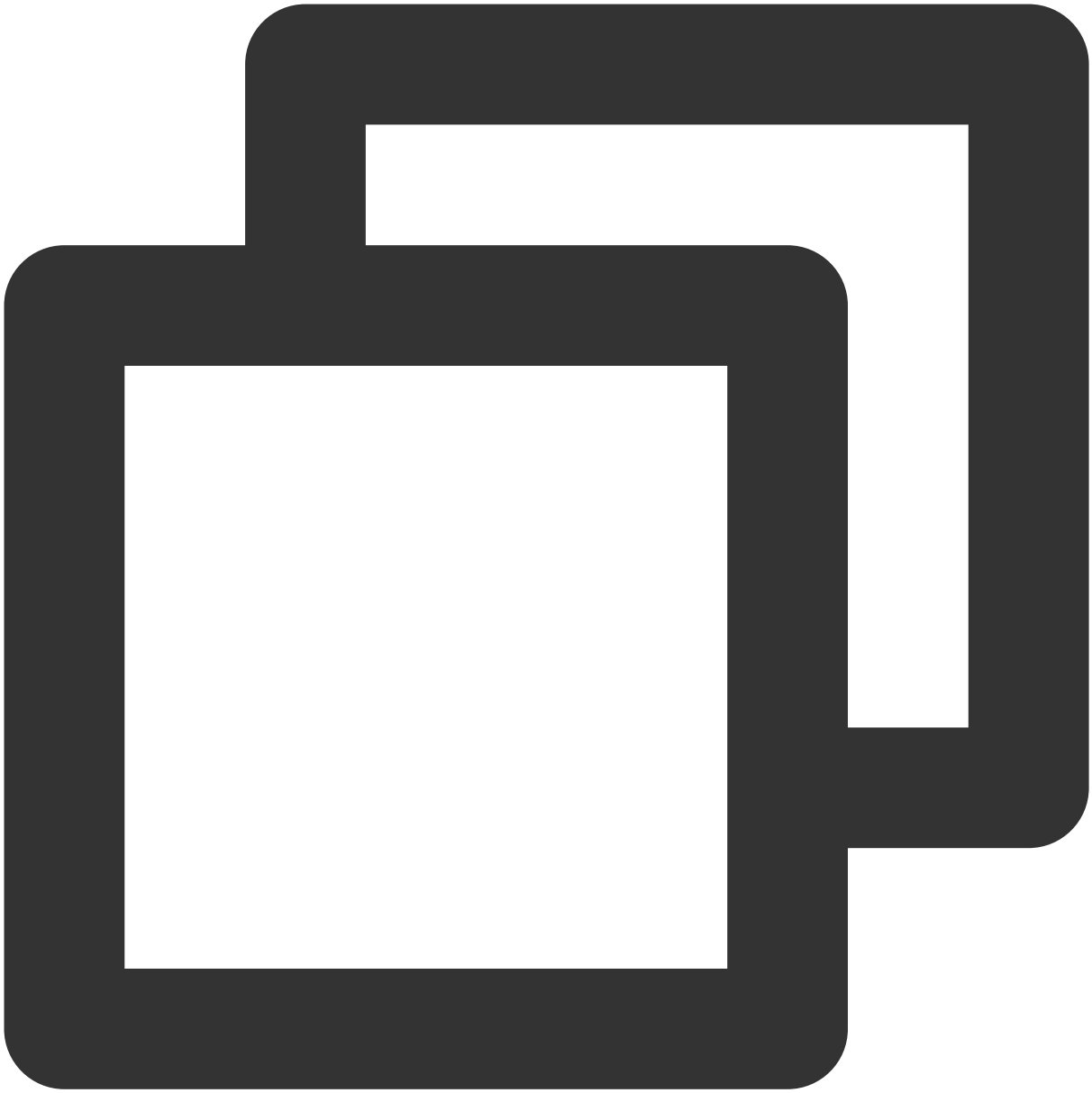
（3）失败文件重迁移

（1）创建临时目录



```
hadoop fs -libjars /data01/jars/chdfs_hadoop_plugin_network-2.8.jar -mkdir cosn://b
```

（2）运行 COSDistCp 任务



```
nohup hadoop jar /data01/jars/cos-distcp-1.10-2.8.5.jar -libjars  /data01/jars/chdf
```

参数说明如下，您可根据实际情况进行调整：

--taskNumber=VALUE 指定拷贝进程数，示例：--taskNumber=10。

--workerNumber=VALUE 指定拷贝线程数，COSDistCp 在每个拷贝进程中创建该参数大小的拷贝线程池。示例：--workerNumber=4。

--bandWidth 限制读取每个迁移文件的带宽，单位为：MB/s，默认-1，不限制读取带宽。示例：--bandWidth=10。

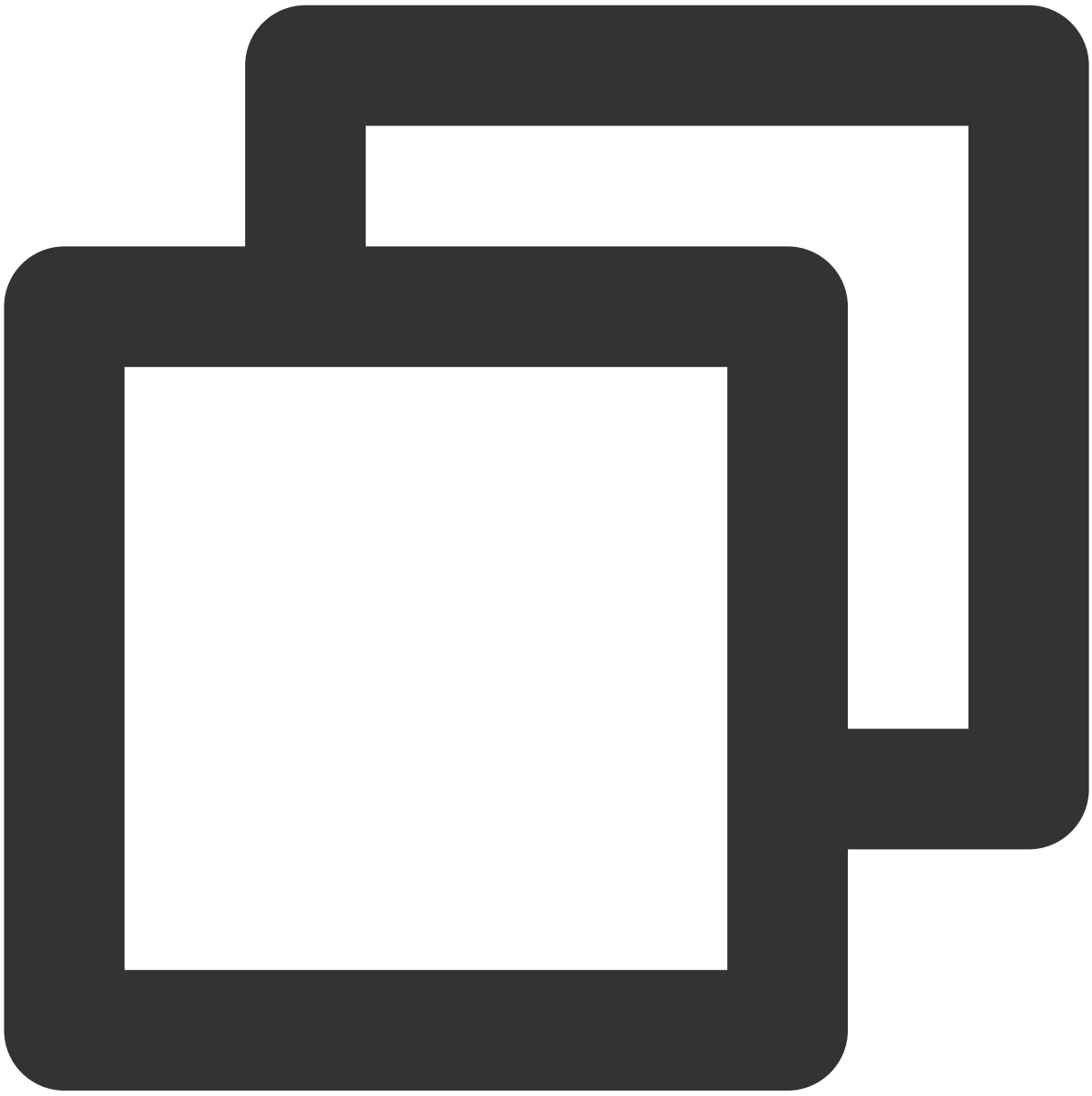
--cosChecksumType=CRC32C, 默认采用 CRC32C，但是需要 HDFS 集群支持 COMPOSITE_CRC32 校验，依赖 Hadoop 版本3.1.1+，如果 HDFS 版本低于3.1.1，此参数需要改为--cosChecksumType=CRC64。

注意

COSDistCp 迁移的总带宽限制计算公式为：taskNumber x workerNumber x bandWidth，您可以将 workerNumber 设置为 1，通过参数 taskNumber 控制迁移并发数，以及参数 bandWidth 控制单个并发的带宽。

在拷贝任务结束时，任务日志会输出文件拷贝的统计信息，相关计数器如下：

其中 FILES_FAILED 代表失败的个数，若无 FILES_FAILED 统计项则说明全部迁移成功。



```
CosDistCp Counters
        BYTES_EXPECTED=10198247
        BYTES_SKIPPED=10196880
        FILES_COPIED=1
```

版权所有：腾讯云计算（北京）有限责任公司

```
        FILES_EXPECTED=7
        FILES_FAILED=1
        FILES_SKIPPED=5
```

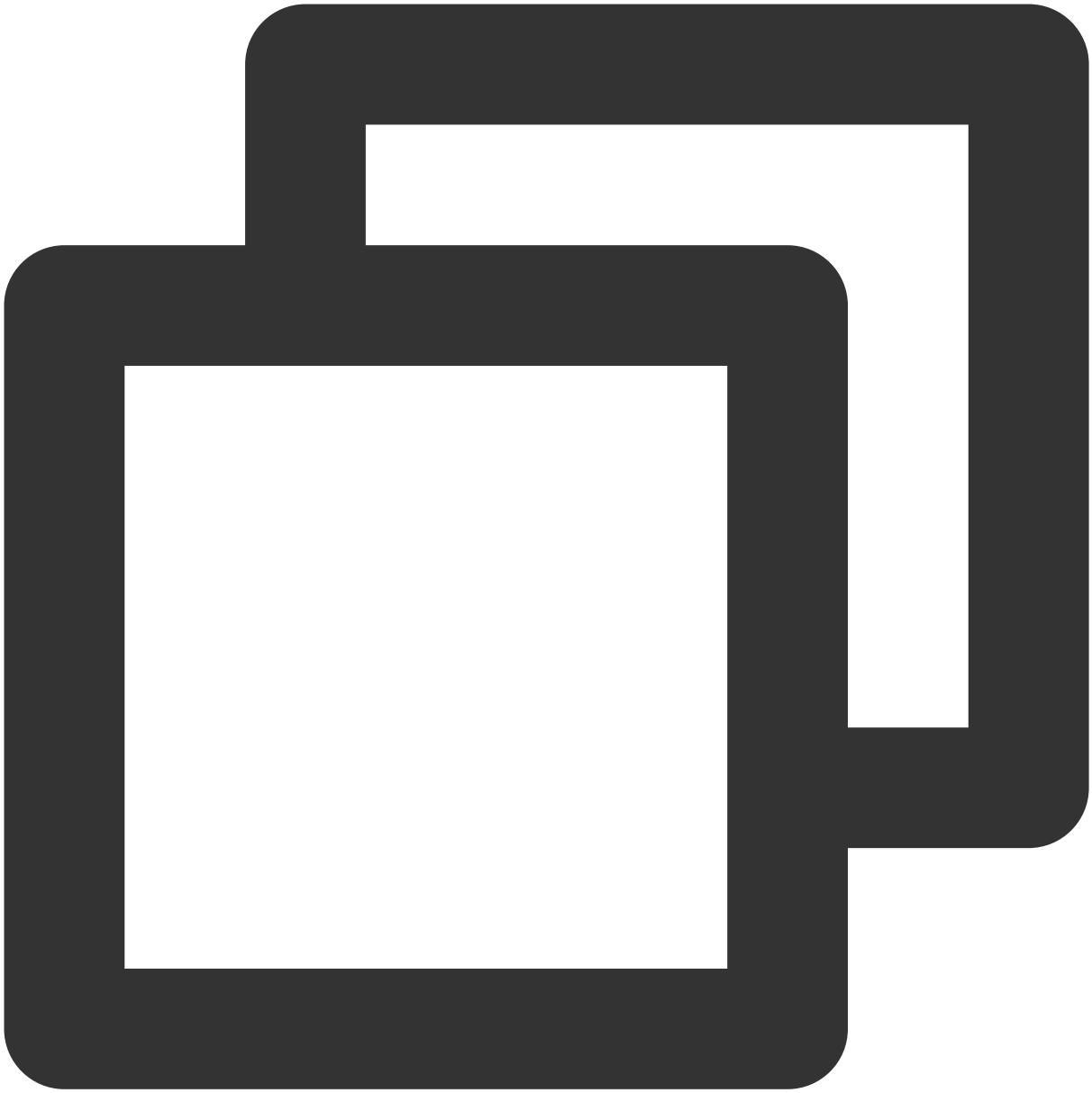
具体输出结果统计项说明如下：

统计项	说明
BYTES_EXPECTED	根据源目录统计的需拷贝的文件总大小，单位：字节
FILES_EXPECTED	根据源目录统计的需拷贝文件数，包含目录文件
BYTES_SKIPPED	长度或校验和值相等，不拷贝的文件总大小，单位：字节
FILES_SKIPPED	长度或校验和值相等，不拷贝的源文件数
FILES_COPIED	拷贝成功的源文件数
FILES_FAILED	拷贝失败的源文件数
FOLDERS_COPIED	拷贝成功的目录数
FOLDERS_SKIPPED	跳过的目录数

（3）失败文件重迁移

COSDistCp 工具不但可以解决大部分文件的迁移效率问题，同时也可以采用 `--delete` 参数支持 HDFS 和 COS 数据的完全一致。

使用 `--delete` 参数时，需要携带 `--deleteOutput=/xxx(自定义)` 参数，但不可以携带 `--diffMode` 参数。



```
nohup hadoop jar /data01/jars/cos-distcp-1.10-2.8.5.jar -libjars /data01/jars/chdfs
```

运行完成后，会将HDFS和COS的差异数据移动到 `trash` 目录下，并且在 `/xxx/failed` 目录下生成移动文件清单。删除 `trash` 目录下的数据可以采用 `hadoop fs -rm URL` 或者 `hadoop fs -rmr URL` 。

增量迁移

如果每一轮迁移过后，还存在更新的增量数据需要迁移，只需要重复执行全量迁移中的步骤即可，直到数据均已完成迁移。

使用 HDFS 协议访问已开启元数据加速的存储桶

最近更新时间：2024-06-21 16:54:35

元数据加速器概述

元数据加速器是由腾讯云对象存储（Cloud Object Storage，COS）服务提供的高性能文件系统功能。元数据加速器底层采用了云 HDFS 卓越的元数据管理功能，支持用户通过文件系统语义访问对象存储服务，系统设计指标可以达到百GB带宽、10万级 QPS 以及 ms 级延迟。存储桶在开启元数据加速器后，可以广泛应用于大数据、高性能计算、机器学习、AI 等场景。有关元数据加速器的详细介绍，请参见 元数据加速器。

使用 HDFS 协议访问的优势

以往基于 COS 的大数据访问主要使用 Hadoop-COS 工具来访问。Hadoop-COS 工具内部将 HCFS 接口适配为 COS 的 Restful 接口，从而对 COS 上的数据进行访问。由于 COS 和文件系统在元数据组织方式上的差异，导致元数据操作性能上存在性能差异，从而影响了大数据分析性能。开启元数据加速器的 Bucket，完全兼容 HCFS 协议，可以采用原生的 HDFS 接口直接访问，除了省去了 HDFS 协议到对象协议的转换开销外，更能提供原生 HDFS 的一些功能，例如目录原子高效 Rename、文件 Atime、Mtime 更新、高效目录 DU 统计、Posix ACL 权限支持等原生特性。

创建存储桶并配置 HDFS 协议

1. 创建 COS 存储桶，并且开启元数据加速器。
当存储桶创建完成后，进入存储桶的**文件列表**页面，您可在该页面进行文件上传和下载操作。
2. 在左侧菜单栏中，单击**性能配置 > 元数据加速能力**，可以看到元数据加速能力已开启。
如果是第一次创建**需开启元数据加速**的存储桶，需要按照提示进行相应的**授权**操作，单击授权完成后，将自动开启 HDFS 协议，并且看到默认的存储桶挂载点信息。
说明
如果提示未找到对应的 HDFS 文件系统，请单击 提交工单 联系我们获取帮助。
3. 在 HDFS 权限配置栏中，单击**新增权限配置**。
4. 在 VPC 网络名称列选择计算集群所在的 VPC 网络地址，在节点 IP 地址列填写 VPC 网段下需要放通的 IP 地址或者 IP 段，访问类型选择读写或者只读，配置完成后，单击**保存**即可。
说明
HDFS 权限配置与原生 COS 权限体系存在差异。当您使用 HDFS 协议访问时，推荐通过配置 HDFS 权限授权指定 VPC 内机器访问 COS 存储桶，以便获取和原生 HDFS 一致的权限体验。

版权所有：腾讯云计算（北京）有限责任公司

配置计算集群访问 COS

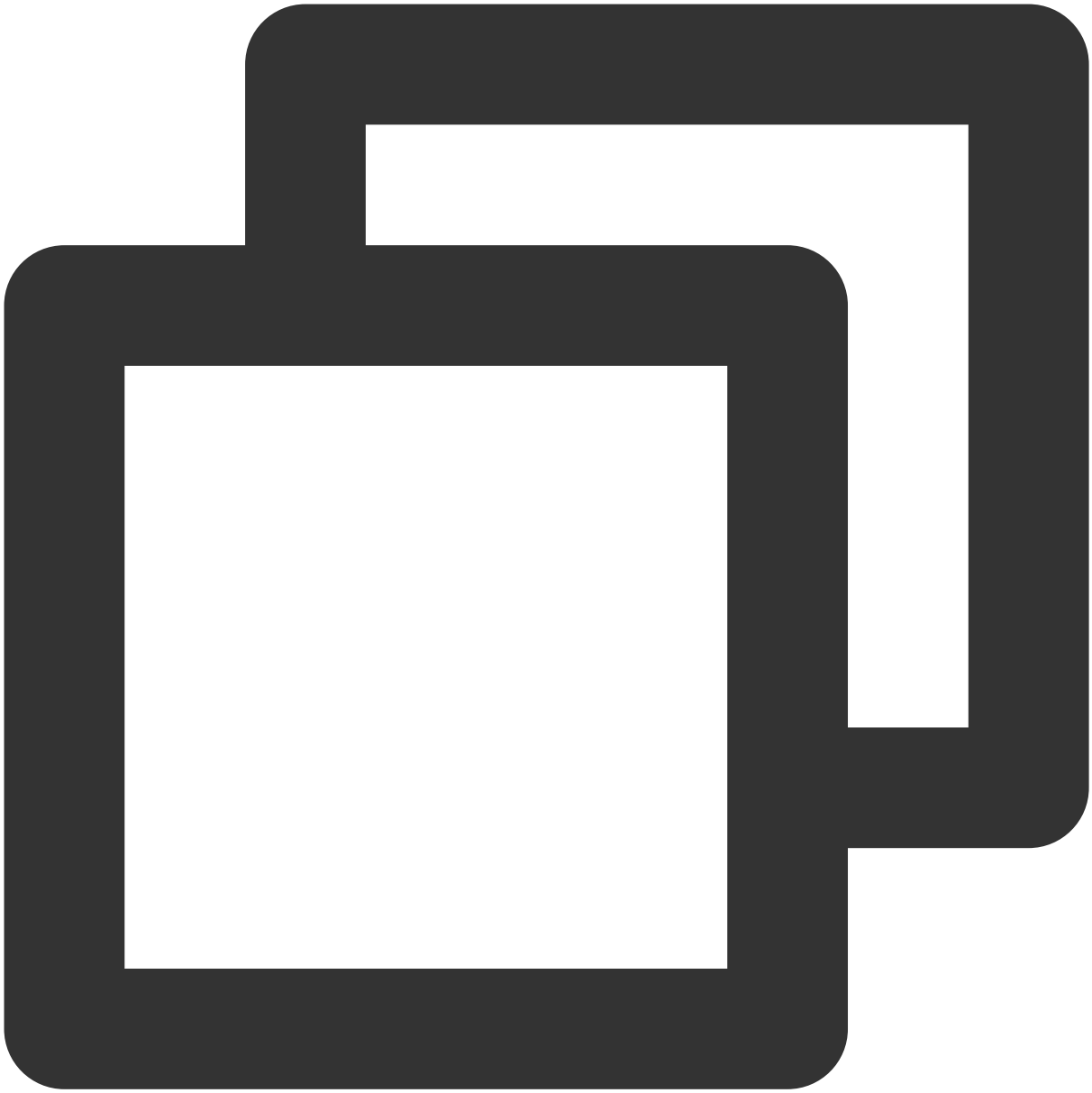
环境依赖

依赖项	chdfs-hadoop-plugin	COSN（hadoop-cos）	cos_api-bundle
需要版本	≥ 2.7版本	≥ 8.1.5版本	和 COSN 版本相对应，查看 COSN github releases 确认
开源下载地址	Github 地址	Github 地址	Github 地址

腾讯云 EMR 环境

腾讯云 EMR 环境 已经无缝集成 COS，您只需要完成以下几个步骤：

1. 找到一台 EMR 机器，并在该机器上执行以下命令，检查 EMR 环境下所需要的服务的文件夹下包的版本是否符合环境依赖要求。



```
find / -name "chdfs*"
find / -name "temrfs_hadoop*"
```

```
[root@172 ~]# find /usr/local/service/ -name 'chdfs*'
/usr/local/service/cosranger/sbin/chdfs-ranger-service-define.sh
/usr/local/service/cosranger/sbin/chdfs-ranger.json
/usr/local/service/spark/jars/chdfs_hadoop_plugin_network-2.8.jar
/usr/local/service/hive/spark/jars/chdfs_hadoop_plugin_network-2.8.jar
/usr/local/service/hadoop/share/hadoop/common/lib/chdfs_hadoop_plugin_network-2.8
/usr/local/service/apps/kylin-4.0.1/spark/jars/chdfs_hadoop_plugin_network-2.8.ja
/usr/local/service/tez/chdfs_hadoop_plugin_network-2.8.jar
/usr/local/service/ranger/ews/webapp/WEB-INF/classes/ranger-plugins/chdfs
/usr/local/service/ranger/ews/webapp/WEB-INF/classes/ranger-plugins/chdfs/chdfs-r
/usr/local/service/trino/plugin/hive/chdfs_hadoop_plugin_network-2.8.jar
```

查看搜索结果里确保两个 jar 包的版本符合上述环境依赖的要求。

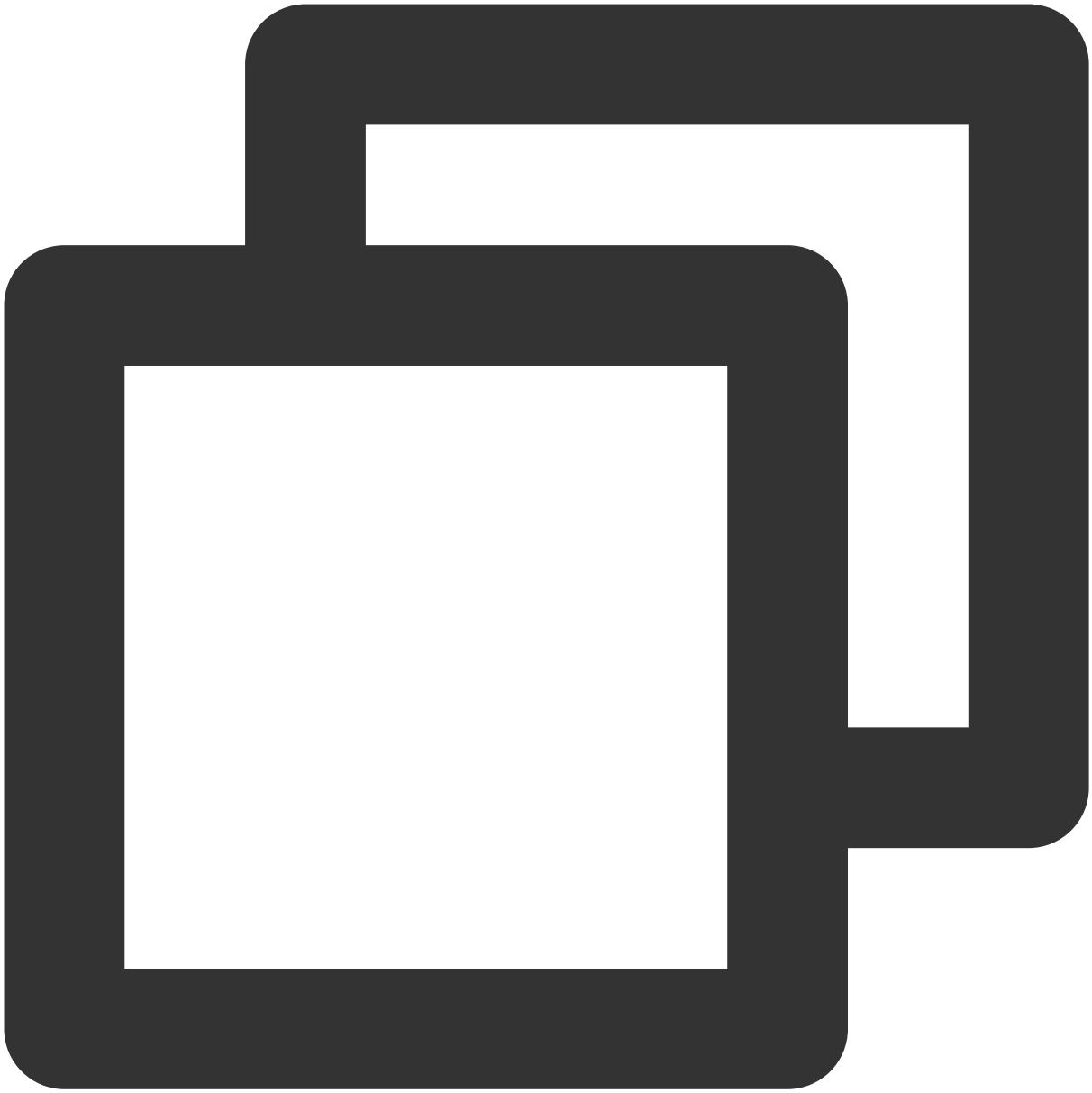
2. 如果 chdfs-hadoop-plugin 版本的包需要更新，则执行以下步骤进行更新：

下载更新 jar 包的脚本文件，下载地址如下：

update_cos_jar.sh

update_cos_jar_common.sh

把这两个脚本放到服务器 /root 目录下，为 update_cos_jar.sh 添加执行权限，执行以下命令：



```
sh update_cos_jar.sh  https://hadoop-jar-beijing-1259378398.cos.ap-beijing.myqcloud
```

参数替换为对应地域的存储桶，例如广州地域，则替换为 `https://hadoop-jar-guangzhou-1259378398.cos.ap-guangzhou.myqcloud.com/hadoop_plugin_network/2.7` 。

在每一台 EMR 节点上执行以上步骤，直到机器上的 jar 包都替换完成。

3. 如果 hadoop-cos 包或者 cos_api-bundle 版本的包需要更新，则执行以下步骤进行更新：

将 /usr/local/service/hadoop/share/hadoop/common/lib/hadoop-temrfs-1.0.5.jar 替换为 temrfs_hadoop_plugin_network-1.1.jar。

在 core-site.xml，新增配置项：

emr.temrfs.download.md5=822c4378e0366a5cc26c23c88b604b11

emr.temrfs.download.version=2.7.5-8.1.5-1.0.6 （2.7.5替换为您的 hadoop 版本，8.1.5替换为您需要的 hadoop-cos 包的版本，但确保版本不低于8.1.5, cos_api-bundle 版本会自动适配）

emr.temrfs.download.region=sh

emr.temrfs.tmp.cache.dir=/data/emr/hdfs/tmp/temrfs

core-site.xml 中修改配置 fs.cosn.impl=com.qcloud.emr.fs.TemrfsHadoopFileSystemAdapter。

4. 在 EMR 控制台 配置 core-site.xml，新增配置项 `fs.cosn.bucket.region` ，`fs.cosn.trsf.fs.ofs.bucket.region` 该参数用于指定存储桶所在的 COS 地域，例如 `ap-shanghai` 。

注意

`fs.cosn.bucket.region` 和 `fs.cosn.trsf.fs.ofs.bucket.region` 必须配置，该参数用于指定存储桶所在的 COS 地域，例如 `ap-shanghai` 。

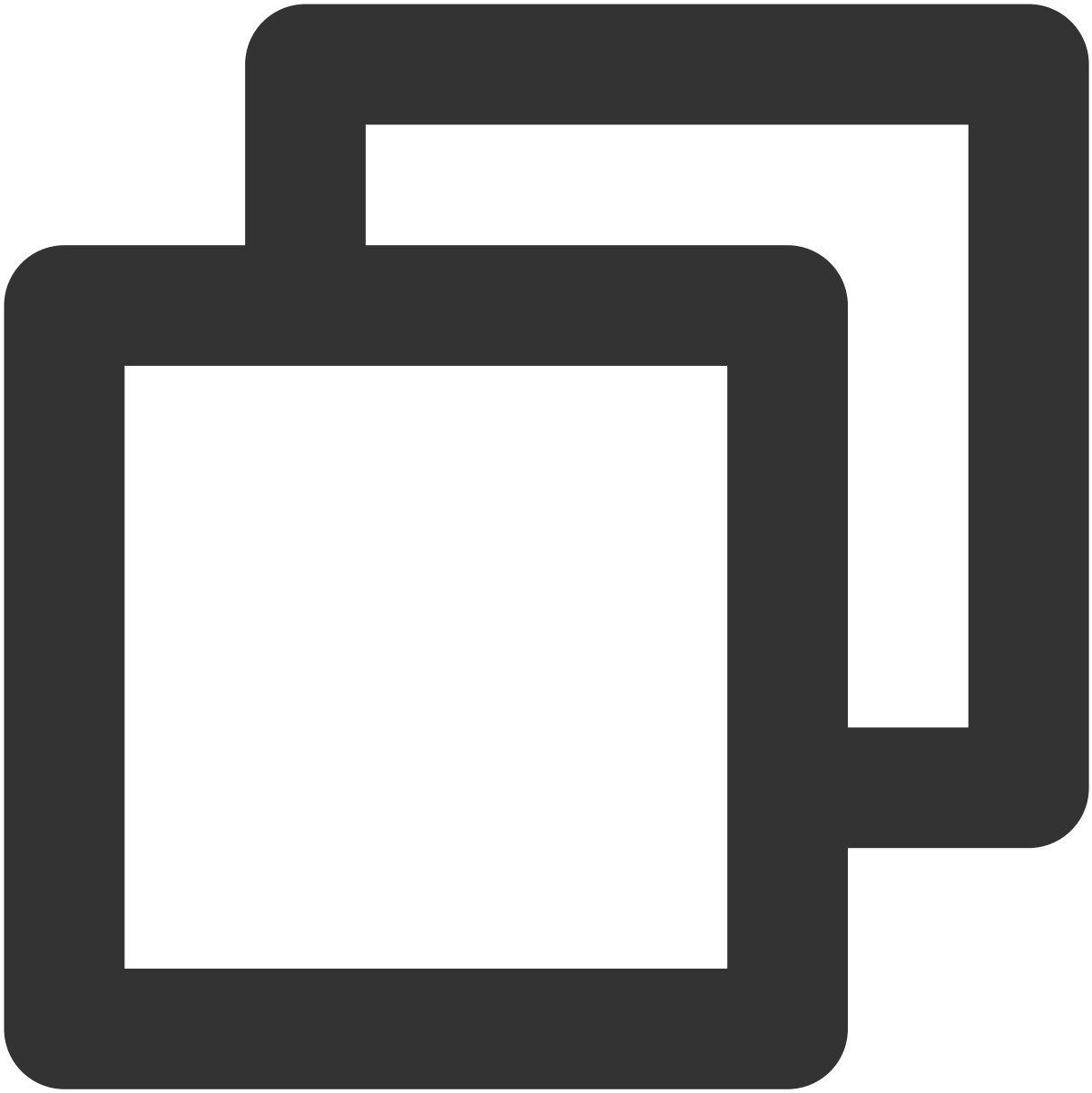
5. 重启 Yarn、Hive、Presto、Impala 等一些常驻服务。

自建 Hadoop/CDH 等环境

1. 自建环境 需要下载环境依赖中的符合版本要求的三个 jar 包。

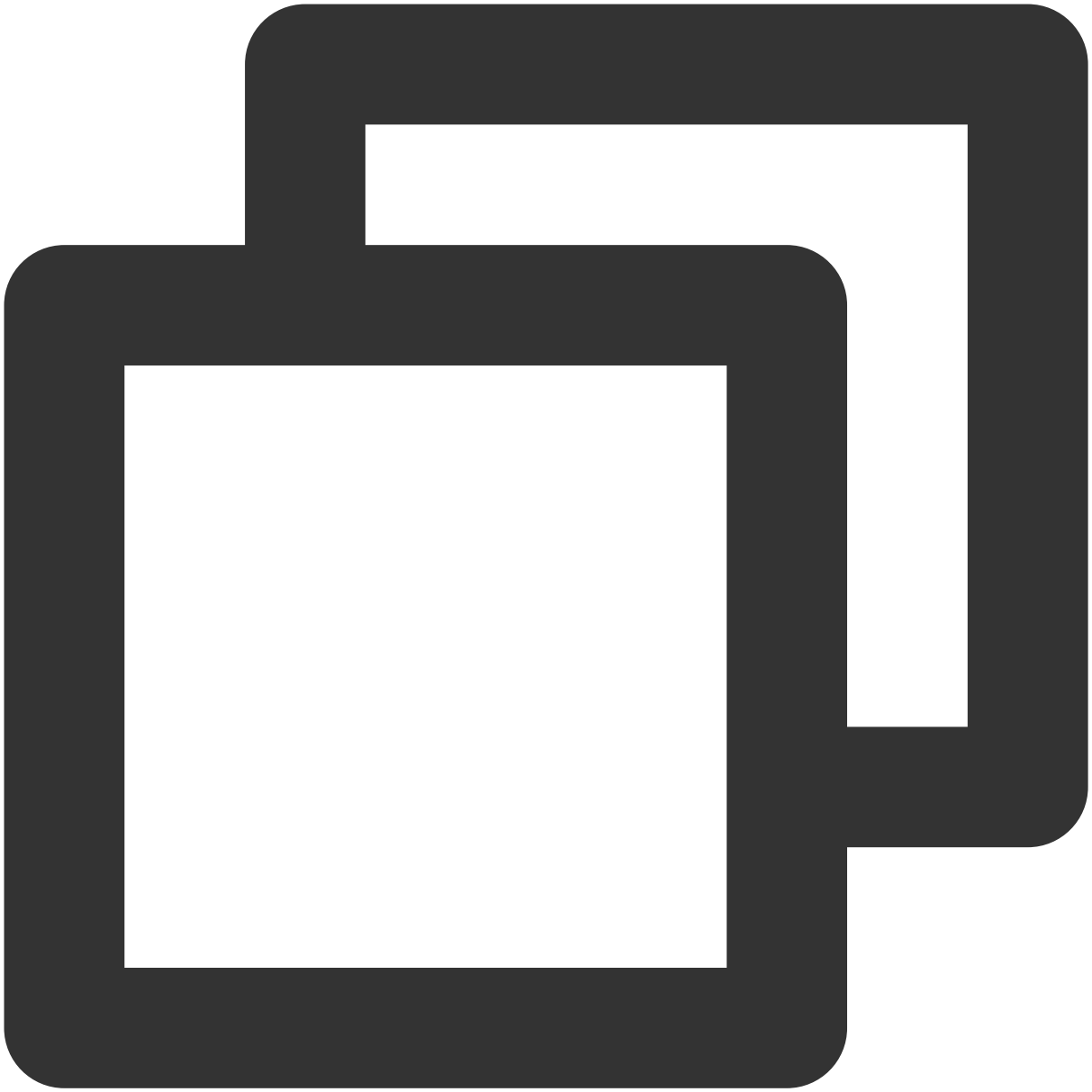
2. 下载后，将上述三个安装包正确放置到 Hadoop 集群中每台服务器的 `classpath` 路径下，例如 `/usr/local/service/hadoop/share/hadoop/common/lib/` （根据实际情况放置，不同组件可能放置的位置也不一样）。

3. 修改 hadoop-env.sh 文件。进入 `$HADOOP_HOME/etc/hadoop` 目录，编辑 hadoop-env.sh 文件，增加以下内容，将 cosn 相关 jar 包加入 Hadoop 环境变量：



```
for f in $HADOOP_HOME/share/hadoop/tools/lib/*.jar; do
  if [ "$HADOOP_CLASSPATH" ]; then
    export HADOOP_CLASSPATH=$HADOOP_CLASSPATH:$f
  else
    export HADOOP_CLASSPATH=$f
  fi
done
```

4. 在计算集群配置 `core-site.xml`，新增以下配置：



```
<!--cosn 的实现类-->
<property>
        <name>fs.cosn.impl</name>
        <value>org.apache.hadoop.fs.CosFileSystem</value>
</property>

<!--用户存储桶的地域信息，格式形如 ap-guangzhou-->
<property>
        <name>fs.cosn.bucket.region</name>
        <value>ap-guangzhou</value>
</property>
```

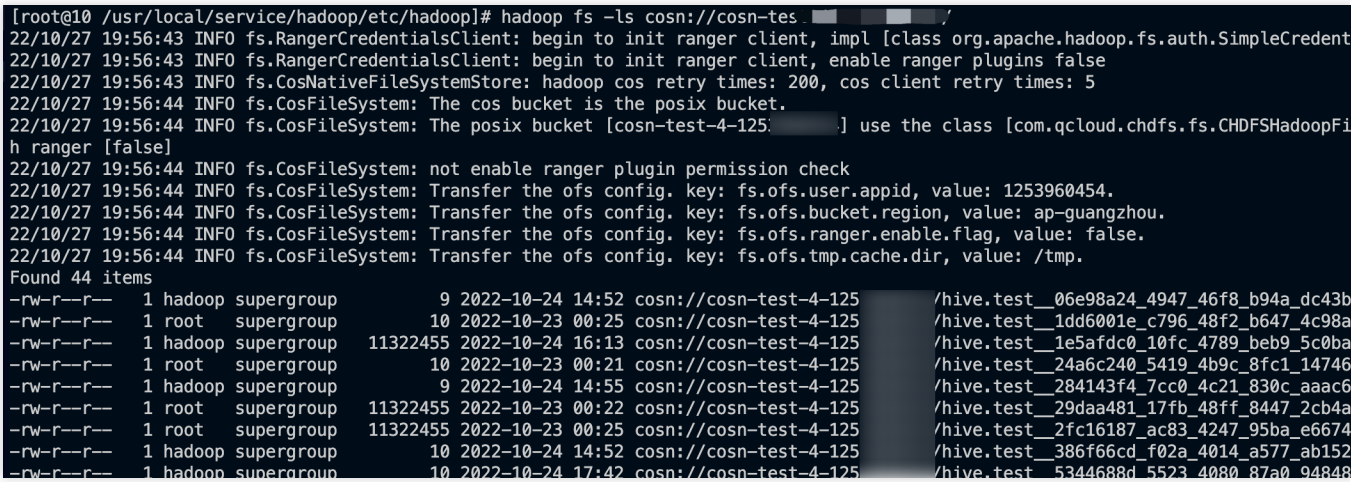
```
<!--用户存储桶的地域信息，格式形如 ap-guangzhou-->
<property>
        <name>fs.cosn.trsf.fs.ofs.bucket.region</name>
        <value>ap-guangzhou</value>
</property>

<!--配置 SecretId 和 SecretKey 的获取方式-->
<property>
        <name>fs.cosn.credentials.provider</name>
        <value>org.apache.hadoop.fs.auth.SimpleCredentialProvider</value>
</property>

<!--账户的 API 密钥信息。可登录 [访问管理控制台](https://console.tencentcloud.com/capi) 查
<property>
        <name>fs.cosn.userinfo.secretId</name>
        <value>XXXXXXXXXXXXXXXXXXXXXXXX</value>
</property>

<!--账户的 API 密钥信息。可登录 [访问管理控制台](https://console.tencentcloud.com/capi) 查
<property>
        <name>fs.cosn.userinfo.secretKey</name>
        <value>XXXXXXXXXXXXXXXXXXXXXXXX</value>
</property>

<!--配置账户的 appid-->
<property>
        <name>fs.cosn.trsf.fs.ofs.user.appid</name>
        <value>125XXXXXX</value>
</property>

<!--本地临时目录，用于存放运行过程中产生的临时文件-->
<property>
        <name>fs.cosn.trsf.fs.ofs.tmp.cache.dir</name>
        <value>/tmp</value>
</property>
```

5. 重启 Yarn、Hive、Presto、Impala 等一些常驻服务。

验证环境

所有环境配置完成后，您可以通过如下操作进行验证。

在客户端使用 Hadoop 命令行查看是否挂载成功。如下图所示：



登录 COS 控制台，查看存储桶文件列表，确认文件和目录是否一致。

Ranger 权限配置

HDFS 协议默认采用原生 POSIX ACL 方式进行鉴权，如果需要使用 Ranger 鉴权，可参考如下流程配置。

EMR 环境

1. EMR 环境集成了 COSRanger 服务，在 EMR 集群购买时勾选 COSRanger 服务。
2. 在 HDFS 协议的 **HDFS 鉴权模式**下，选择 **Ranger 鉴权模式**，配置上 Ranger 相应的地址信息即可。
在 core-site.xml 中新增配置项：fs.cosn.credentials.provider，设置为
org.apache.hadoop.fs.auth.RangerCredentialsProvider。
如果遇到 Ranger 相关问题，可参见 Ranger 介绍说明。

自建 Hadoop/CDH 等环境

1. 配置 Ranger 服务，通过 Ranger 服务以 HDFS 协议访问 COS，详情请参见 COS Ranger 权限体系解决方案 文档。
2. 在 HDFS 协议的 **HDFS 鉴权模式**下，选择 **Ranger 鉴权模式**，配置上 Ranger 相应的地址信息即可。
在 core-site.xml 中新增配置项：fs.cosn.credentials.provider，设置为：
org.apache.hadoop.fs.auth.RangerCredentialsProvider。

其他

大数据场景下，您可参考以下步骤以 HDFS 协议访问开启元数据加速能力的存储桶：
1. 在 `core-stie.xml` 中配置 HDFS 协议相关挂载点信息，如 创建存储桶并配置 HDFS 协议 所示。
2. 通过 Hive、MR、Spark 等组件访问存储桶，请参见 在计算集群中挂载 COS 存储桶。

3. 默认情况下，采用原生 `POSIX ACL` 方式进行鉴权，如果需要使用 `Ranger 鉴权`，可参考 COS Ranger 权限体系解决方案。

在计算集群中挂载 COS 存储桶

最近更新时间：2023-03-02 16:08:17

简介

对象存储（Cloud Object Storage，COS）可以通过开启元数据加速能力，拥有 HDFS 协议访问的能力。开启元数据加速能力后，COS 会为存储桶生成一个挂载点，您可以通过下载 HDFS 客户端，在客户端中输入该挂载点挂载 COS。本文将详细介绍如何在计算集群中挂载开启元数据加速的存储桶。

> 注意：
>
> - Hadoop-cos 自8.1.5版本开始支持 `cosn://bucketname-appid/` 方式访问元数据加速桶。
> - 元数据加速功能只能在创建存储桶时开启，开启后不支持关闭，请结合您的业务情况**慎重考虑**是否开启，同时注意旧版本的 Hadoop-cos 包不能正常访问已开启元数据加速功能的存储桶。

前提条件

- 确保计算集群中需要挂载的机器或者容器内已安装 Java 1.8。
- 确保计算集群中需要挂载的机器或者容器已授权访问，您需要在 HDFS 权限配置里指定可访问的 VPC 网络和 IP 地址。
- 依赖 JAR 包说明：

1. chdfs_hadoop_plugin_network-2.8.jar verison ≥ 2.7。
2. cos_api-bundle.jar version ≥ 5.6.69。
3. Hadoop-cos version ≥ 8.1.5。
4. ofs-java-sdk.jar (version ≥ 1.0.4) 自动拉取无需安装，运行 hadoop fs ls 成功后可以在 fs.cosn.trsf.fs.ofs.tmp.cache.dir 配置的目录下查看对应版本是否符合预期。

操作步骤

1. 下载 Hadoop 客户端工具安装包。
2. 下载 POSIX Hadoop 客户端工具安装包。
3. 下载 cos java sdk 安装包。

版权所有：腾讯云计算（北京）有限责任公司

4. 将安装包放到各节点 classpath 下保证任务启动能正常加载，例如 `$HADOOP_HOME/share/hadoop/common/lib/` 下。

> 注意：
> EMR 环境下自带依赖 jar 包，无需安装，可直接通过 POSIX 语义访问元数据加速桶。如需使用 s3 协议访问，则更改 fs.cosn.posix_bucket.fs.impl 配置项，详情请参见下文。

5. 编辑 `core-site.xml` 文件，新增以下基本配置：

> 注意：
> - 建议用户尽量避免在配置中使用永久密钥，采取配置子账号密钥或者临时密钥的方式有助于提升业务安全性。为子账号授权时请遵循 最小权限指引原则，避免发生预期外的数据泄露。
> - 如果您一定要使用永久密钥，建议对永久密钥的权限范围进行限制，可参考 最小权限指引原则 通过限制永久密钥的可执行操作、资源范围和条件（访问 IP 等），提升使用安全性。

```
<!--账户的 API 密钥信息。可登录 [访问管理控制台](https://console.tencentcloud.com/capi) 查看云 API 密钥。-->
<!--建议使用子账号密钥或者临时密钥的方式完成配置，提升配置安全性。为子账号授权时请遵循[最小权限指引原则](https://www.tencentcloud.com/document/product/436/32972)。-->
<property>
<name>fs.cosn.userinfo.secretId/secretKey</name>
<value>AKIDxxxxxxxxxxxxxxxxxxxxx</value>
</property>

<!--cosn 的实现类-->
<property>
<name>fs.AbstractFileSystem.cosn.impl</name>
<value>org.apache.hadoop.fs.CosN</value>
</property>

<!--cosn 的实现类-->
<property>
<name>fs.cosn.impl</name>
<value>org.apache.hadoop.fs.CosFileSystem</value>
</property>

<!--用户存储桶的地域信息，格式形如 ap-guangzhou-->
<property>
<name>fs.cosn.bucket.region</name>
```

```
<value>ap-guangzhou</value>
</property>

<!--本地临时目录，用于存放运行过程中产生的临时文件-->
<property>
<name>fs.cosn.tmp.dir</name>
<value>/tmp/hadoop_cos</value>
</property>
```

6. 将 `core-site.xml` 同步到所有 `hadoop` 节点上。

> 说明：
> 对于 EMR 集群，以上步骤3、4可在 EMR 控制台的组件管理中，修改 HDFS 配置即可。

7. 使用 `hadoop fs` 命令行工具，运行 `hadoop fs -ls cosn://${bucketname-appid}/` 命令，这里 `bucketname-appid` 为挂载地址，即存储桶名称。如果正常列出文件列表，则说明已经成功挂载 COS 存储桶。
8. 用户也可使用 `hadoop` 其他配置项，或者 `mr` 任务在开启了元数据加速能力的 COS 存储桶上运行数据任务。对于 `mr` 任务，可以通过 `-Dfs.defaultFS=ofs://${bucketname-appid}/` 将本次任务的默认输入输出 `FS` 改为对应的存储桶。

配置项说明

> 说明：
> 这里可以通过 POSIX 语义访问、S3协议访问两种方式访问元数据加速存储桶，我们这里建议使用 POSIX 语义访问方式，可以获得更好的性能。

1. 通用必填配置项

> 注意：
> 无论以哪种方式访问元数据加速存储桶，如下通用配置项必须设置。

配置项	配置项内容	说明

配置项	配置项内容	说明
fs.cosn.userinfo.secretId/secretKey	格式形如 AKIDxxxxxxxxxxxxxxxxxxx	填写您账户的 API 密钥信息。可登访问管理控制台 查看云 API 密钥。
fs.cosn.impl	org.apache.hadoop.fs.CosFileSystem	cosn 对 FileSystem 的实现类，固 org.apache.hadoop.fs.CosFileSys
fs.AbstractFileSystem.cosn.impl	org.apache.hadoop.fs.CosN	cosn 对 AbstractFileSystem 的实现类，固定为 org.apache.hadoop.fs.CosN。
fs.cosn.bucket.region	格式形如 ap-beijing	请填写待访问存储桶的地域信息，值请参见 地域和访问域名 中的地称，例如：ap-beijing、ap-guang等。兼容原有配置：fs.cosn.userinfo.region。
fs.cosn.tmp.dir	默认/tmp/hadoop_cos	请设置一个实际存在的本地目录，过程中产生的临时文件会暂时放于处。同时建议配置各节点该目录后空间和权限。

2. POSIX 访问方式必填配置项（推荐方式）

说明：

- POSIX 访问方式下除通用配置项外，还需要添加以下配置内容。POSIX 访问方式的 其他可选配置项 添加"fs.cosn.trsf." 前缀即可用于访问元数据加速桶。
- 需要注意的是原有 Hadoop cos 相关的配置项不再适用。

配置项	配置项内容	说明
fs.cosn.trsf.fs.AbstractFileSystem.ofs.impl	com.qcloud.chdfs.fs.CHDFSDelegateFSAdapter	元数据桶
fs.cosn.trsf.fs.ofs.impl	com.qcloud.chdfs.fs.CHDFSHadoopFileSystemAdapter	元数据桶
fs.cosn.trsf.fs.ofs.tmp.cache.dir	格式形如 /data/emr/hdfs/tmp/posix-cosn/	请设置一过程中产处。同时空间和权如 "/da cosn/"

配置项	配置项内容	说明
fs.cosn.trsf.fs.ofs.user.appid	格式形如 12500000000	必填。用
fs.cosn.trsf.fs.ofs.bucket.region	格式形如 ap-beijing	必填。用

3. S3 协议访问方式必填配置项

S3协议访问方式需要设置如下配置，其他可选项请参见 Hadoop-cos 配置项。

配置项	配置项内容	说明
fs.cosn.posix_bucket.fs.impl	org.apache.hadoop.fs.CosNFileSystem	POSIX 方式访问配置为 `com.qcloud.chdfs.fs.CHDFSHa` S3 协议方式访问配置为 `org.apache.hadoop.fs.CosNFi` 方式访问。

5. 注意事项

1. 不能使用旧的 hadoop cos jar 包访问开启元数据加速的存储桶。
2. 使用 Hadoop cos ≤ 8.1.5 版本 posix 方式访问已开启元数据加速的存储桶需要在控制台关闭 ranger 校验，8.1.5以上版本可支持在控制台打开 ranger 校验。

在 CDH 集群上通过 HDFS 协议访问 COS

最近更新时间：2024-01-06 11:17:59

简介

CDH（Cloudera's Distribution, including Apache Hadoop）是业界流行的 Hadoop 发行版本。本文指导如何在 CDH 环境下通过 HDFS 协议访问对象存储（Cloud Object Storage，COS）存储桶，以实现大数据计算与存储分离，提供灵活及低成本的大数据解决方案。

注意

通过 HDFS 协议访问 COS 存储桶，需要先开启元数据加速能力。

当前 COS 对大数据组件支持情况如下：

组件名称	CHDFS 大数据组件支持情况	服务组件是否需要重启
Yarn	支持	重启 NodeManager
Yarn	支持	重启 NodeManager
Hive	支持	重启 HiveServer 及 HiveMetastore
Spark	支持	重启 NodeManager
Sqoop	支持	重启 NodeManager
Presto	支持	重启 HiveServer 及 HiveMetastore 和 Presto
Flink	支持	否
Impala	支持	否
EMR	支持	否
自建组件	后续支持	无
HBase	不推荐	无

版本依赖

本文依赖的组件版本如下：

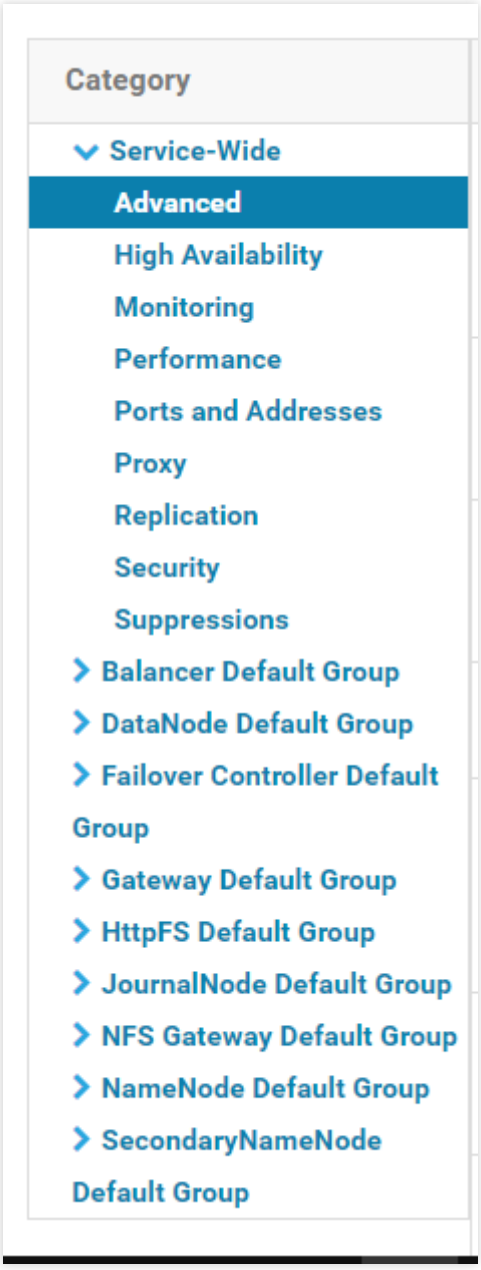
CDH 5.16.1

Hadoop 2.6.0

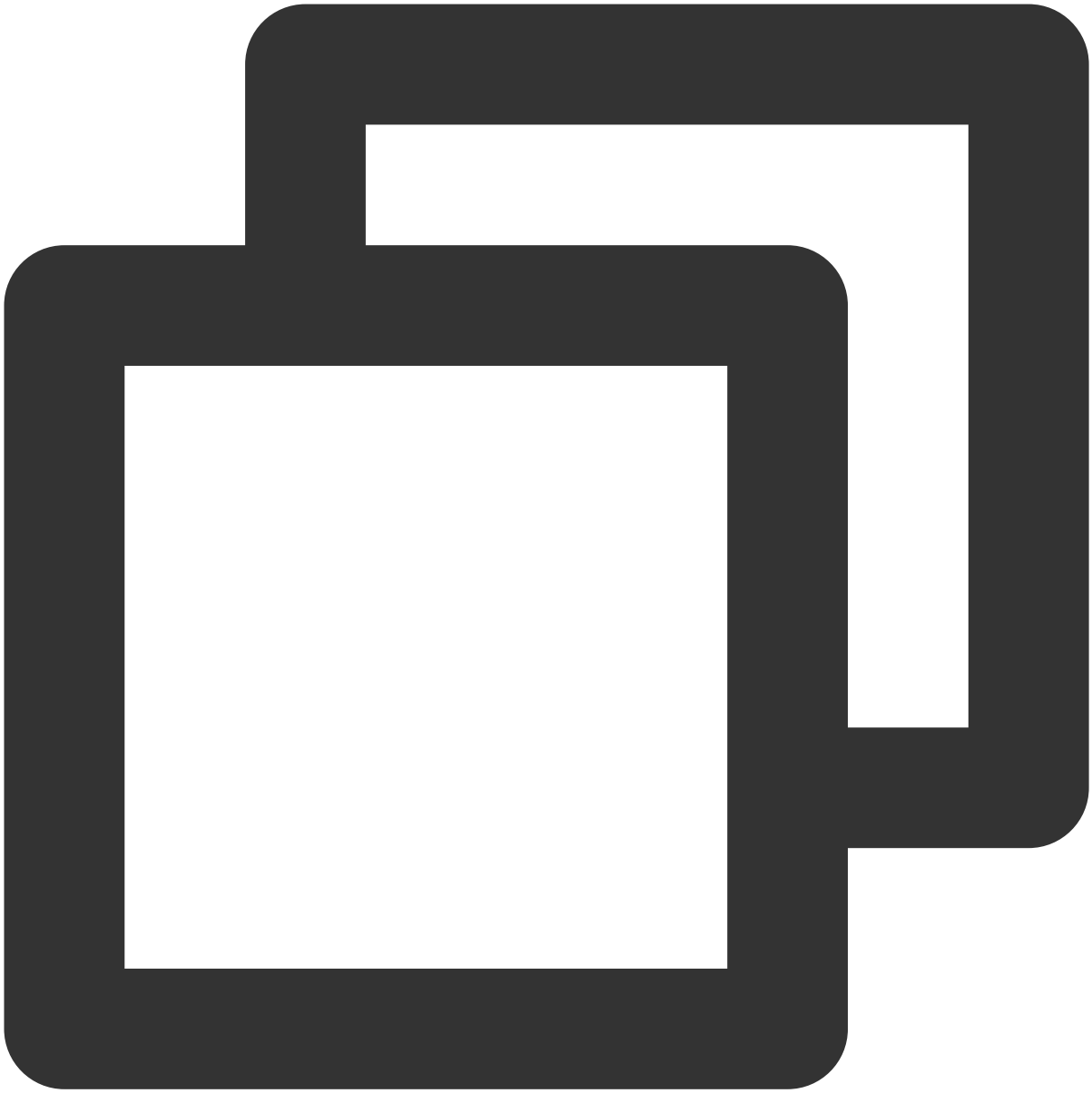
使用方法

存储环境配置

1. 登录 CDH 管理页面。
2. 在系统主页，选择**配置** > **服务范围** > **高级**，进入高级配置代码段页面，如下图所示：



3. 在 `Cluster-wide Advanced Configuration Snippet(Safety Valve) for core-site.xml` 的代码框中，填入 COS 大数据服务配置。



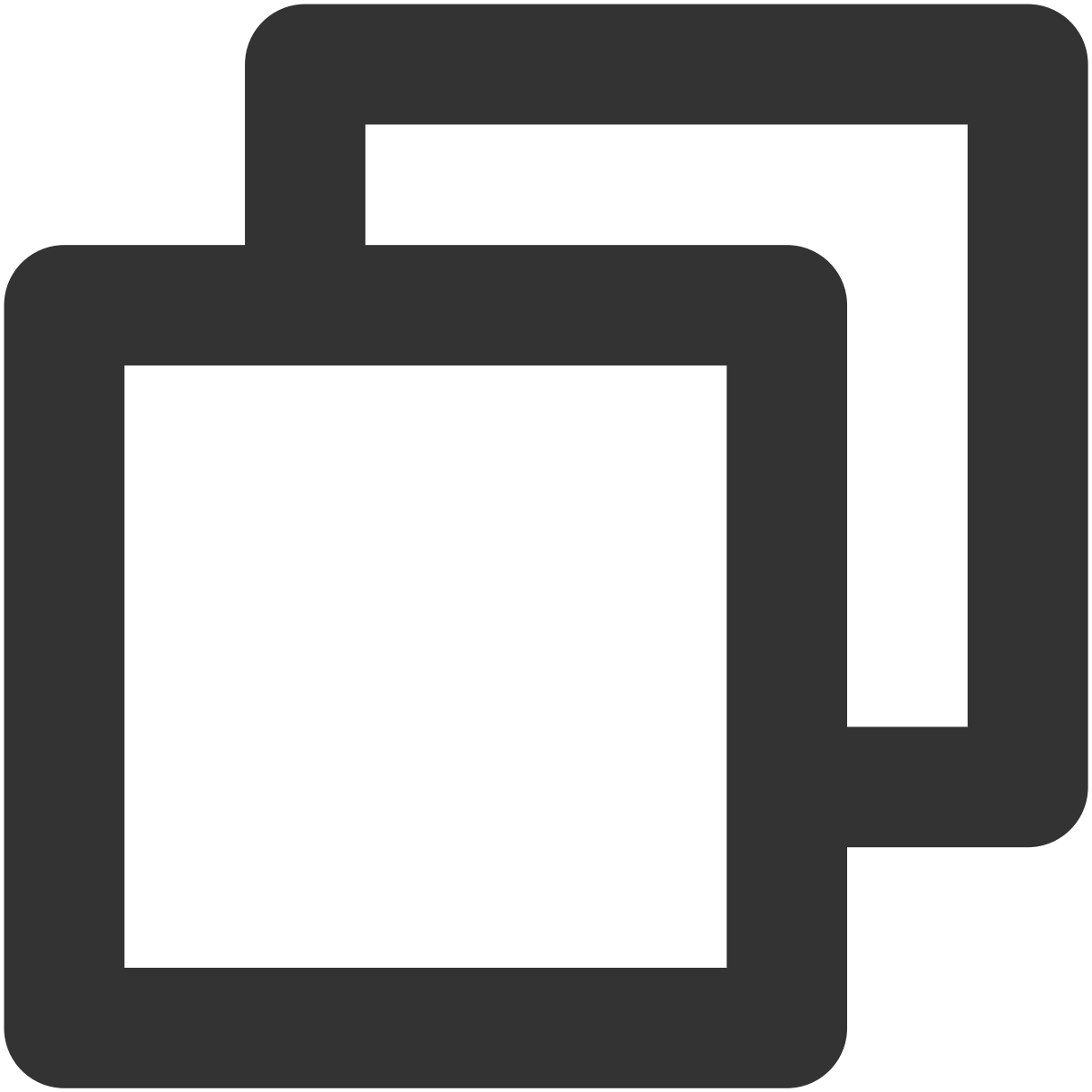
```
<property>
<name>fs.AbstractFileSystem.ofs.impl</name>
<value>com.qcloud.chdfs.fs.CHDFSDelegateFSAdapter</value>
</property>
<property>
<name>fs.ofs.impl</name>
<value>com.qcloud.chdfs.fs.CHDFSHadoopFileSystemAdapter</value>
</property>
<!--本地 cache 的临时目录，对于读写数据，当内存 cache 不足时会写入本地硬盘，这个路径若不存在会自
<property>
<name>fs.ofs.tmp.cache.dir</name>
```

```
<value>/data/emr/hdfs/tmp/chdfs/</value>
</property>
<!--appId-->
<property>
<name>fs.ofs.user.appid</name>
<value>1250000000</value>
</property>
```

以下为必选的配置项（需添加到 core-site.xml 中），其他配置可参见 在计算集群中挂载 COS 存储桶。

配置项	值	含义
fs.ofs.user.appid	1250000000	用户 appid
fs.ofs.tmp.cache.dir	/data/emr/hdfs/tmp/chdfs/	本地 cache 的临时目
fs.ofs.impl	com.qcloud.chdfs.fs.CHDFSHadoopFileSystemAdapter	chdfs 对 FileSystem com.qcloud.chdfs.fs.
fs.AbstractFileSystem.ofs.impl	com.qcloud.chdfs.fs.CHDFSDelegateFSAdapter	chdfs 对 AbstractFile com.qcloud.chdfs.fs.

4. 对 HDFS 服务进行操作，单击部署客户端配置，此时以上 core-site.xml 配置会更新到集群里的机器上。
5. 将最新的 客户端安装包 放置到 CDH HDFS 服务的 jar 包路径下，请根据实际值进行替换，示例如下：



```
cp chdfs_hadoop_plugin_network-2.0.jar /opt/cloudera/parcels/CDH-5.16.1-1.cdh5.16.1
```

注意

在集群中的每台机器都需要在相同的位置放置 SDK 包。

数据迁移

使用 Hadoop Distcp 工具将 CDH HDFS 数据迁移到 COS 存储桶中，详情请参见 Hadoop 文件系统与 COS 之间的数据迁移。

大数据套件使用 CHDFS

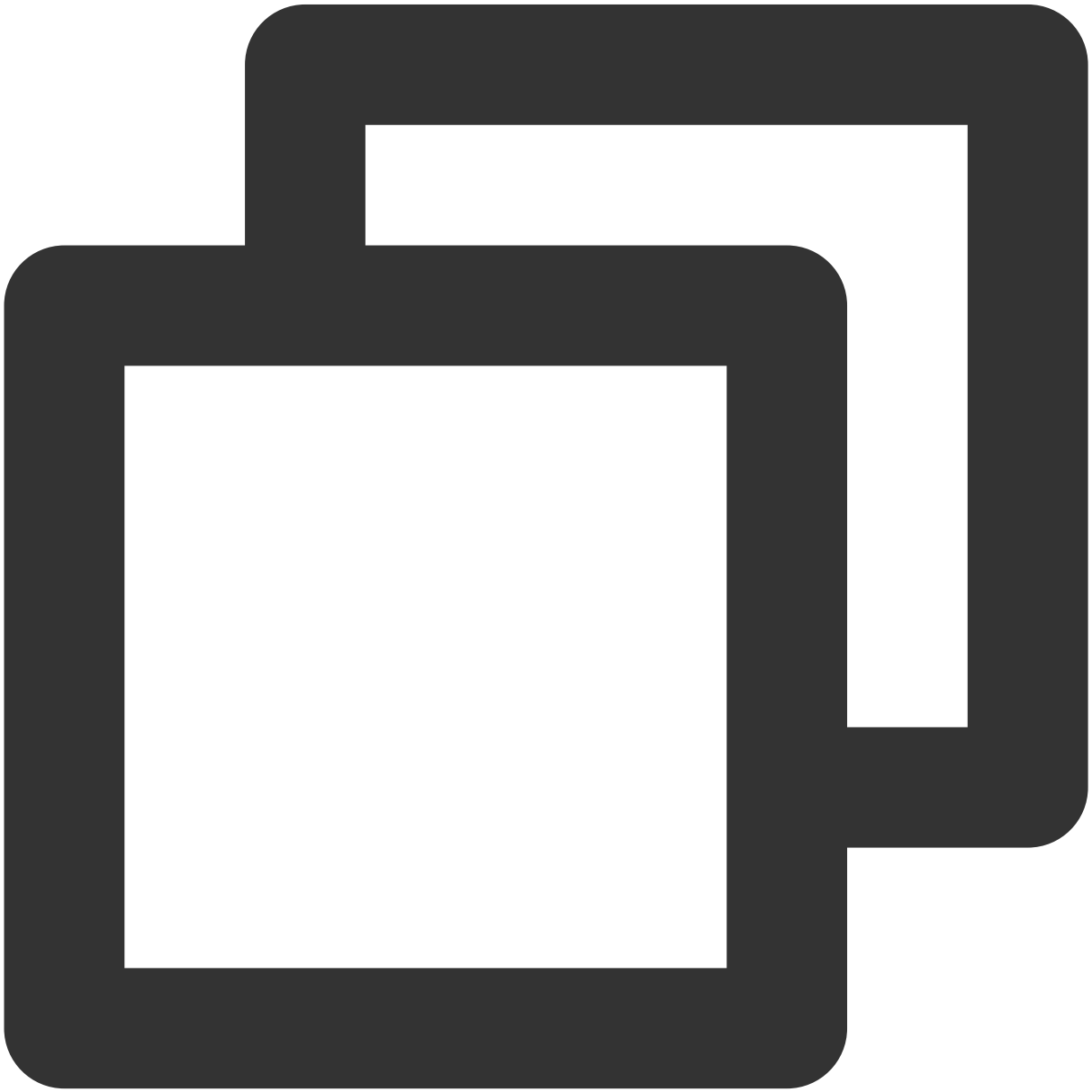
MapReduce

操作步骤

1. 按照 数据迁移 章节，配置好 HDFS 的相关配置，并将 COS 的 客户端安装包，放置到 HDFS 相应的目录。
2. 在 CDH 系统主页，找到 YARN，重启 NodeManager 服务（TeraGen 命令可以不用重启，但是 TeraSort 由于业务内部逻辑，需要重启 NodeManger，建议都统一重启 NodeManager 服务）。

示例

下面以 Hadoop 标准测试中的 TeraGen 和 TeraSort 为例：



```
hadoop jar ./hadoop-mapreduce-examples-2.7.3.jar teragen -Dmapred.map.tasks=4 1099

hadoop jar ./hadoop-mapreduce-examples-2.7.3.jar terasort  -Dmapred.map.tasks=4 ofs
```

说明

`ofs:// schema` 后面请替换为用户 CHDFS 的挂载点路径。

Hive

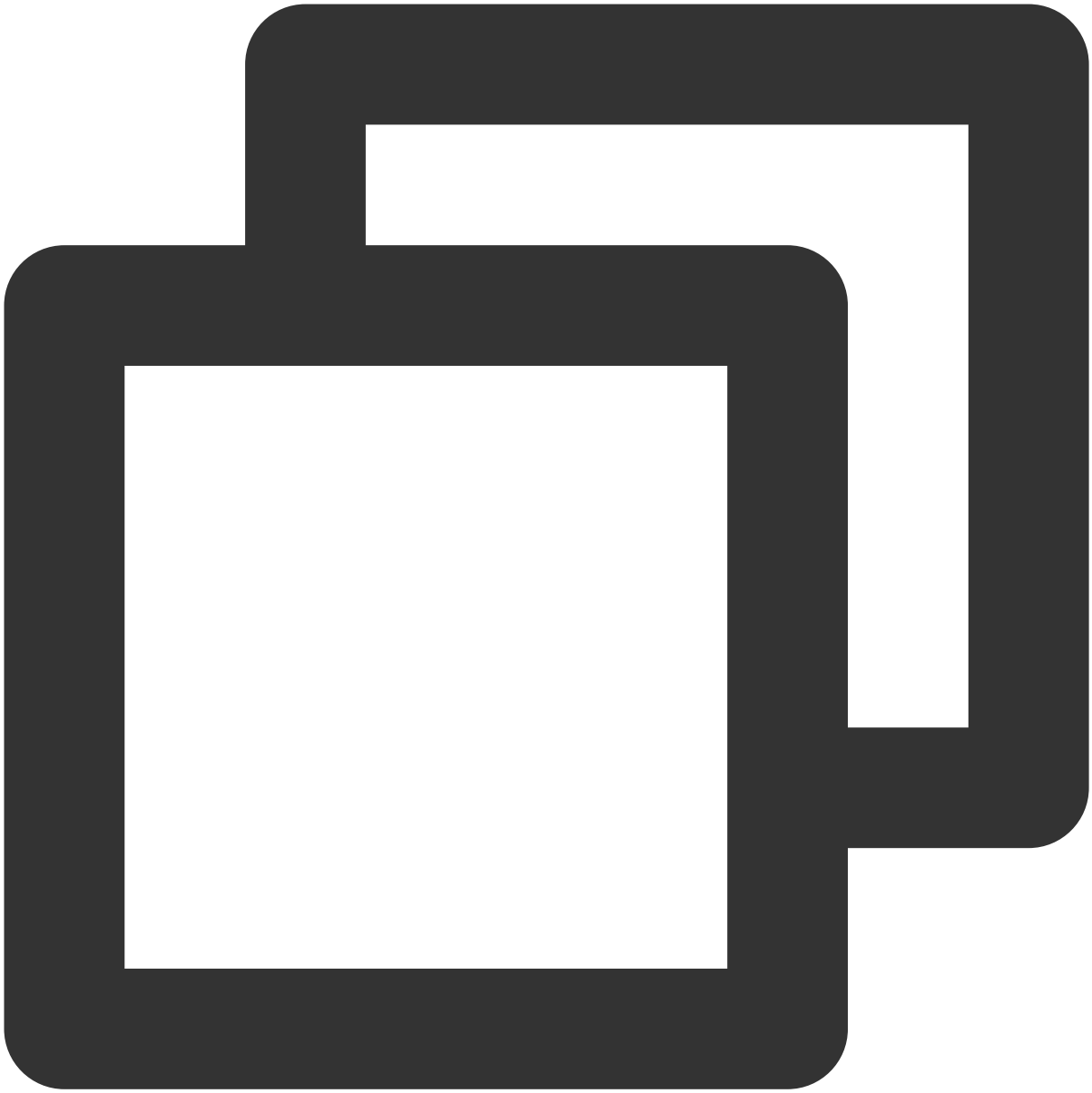
MR 引擎

操作步骤

1. 按照 数据迁移 章节，配置好 HDFS 的相关配置，并且将COS 的 客户端安装包，放置到 HDFS 相应的目录。
2. 在 CDH 主页面，找到 HIVE 服务， 重启 Hiveserver2 及 HiverMetastore 角色。

示例

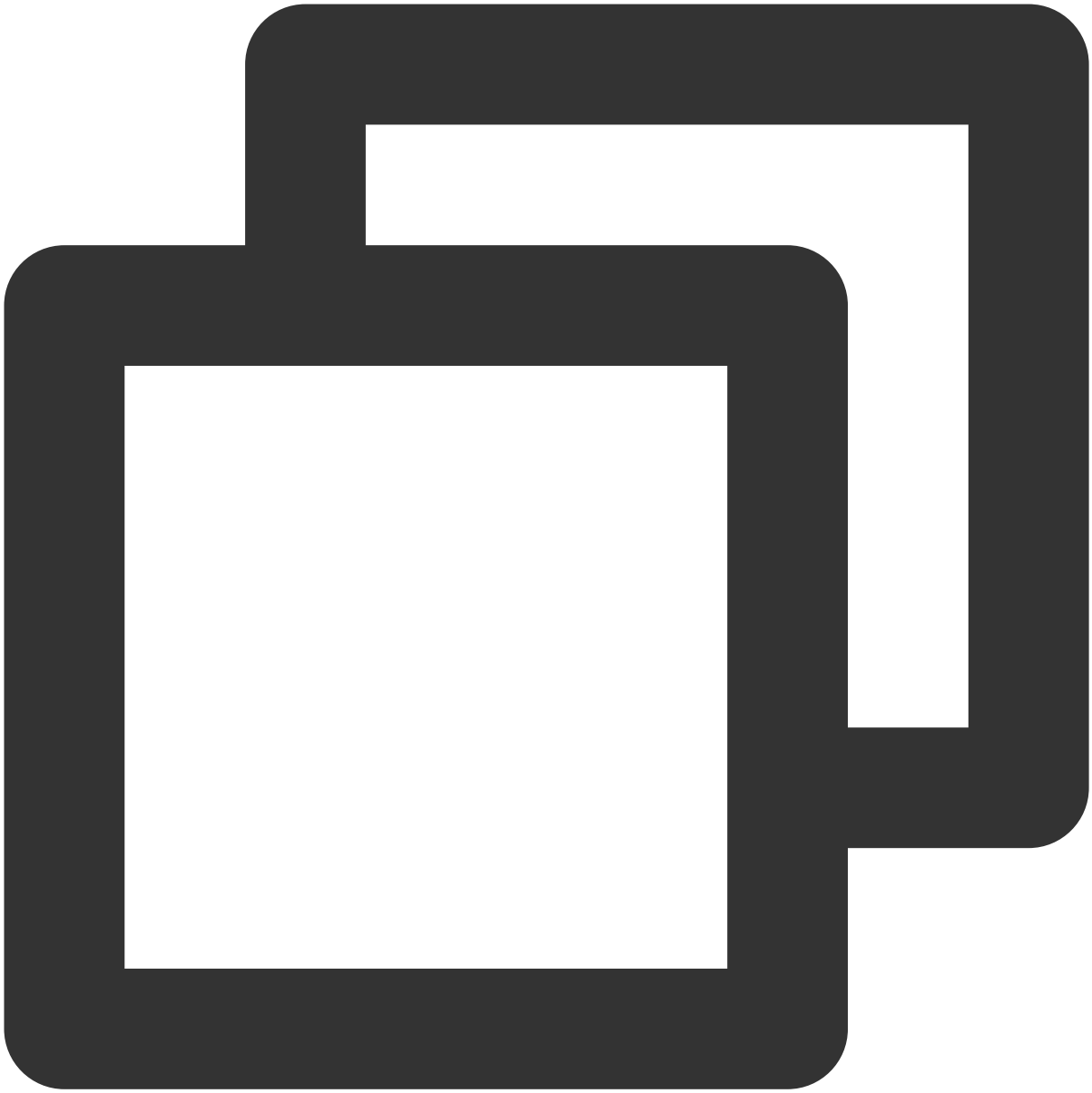
某用户的真实业务查询，例如执行 Hive 命令行，创建一个 Location，作为在 CHDFS 上的分区表：



```
CREATE TABLE `report.report_o2o_pid_credit_detail_grant_daily`(
  `cal_dt` string,
  `change_time` string,
  `merchant_id` bigint,
  `store_id` bigint,
  `store_name` string,
  `wid` string,
  `member_id` bigint,
  `meber_card` string,
  `nickname` string,
  `name` string,
```

```
  `gender` string,
  `birthday` string,
  `city` string,
  `mobile` string,
  `credit_grant` bigint,
  `change_reason` string,
  `available_point` bigint,
  `date_time` string,
  `channel_type` bigint,
  `point_flow_id` bigint)
PARTITIONED BY (
  `topicdate` string)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.orc.OrcSerde'
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcInputFormat'
    OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat'
LOCATION
  'ofs://examplebucket-1250000000/user/hive/warehouse/report.db/report_o2o_pid_cred
TBLPROPERTIES (
  'last_modified_by'='work',
  'last_modified_time'='1589310646',
  'transient_lastDdlTime'='1589310646')
```

执行 SQL 查询：



```
select count(1) from report.report_o2o_pid_credit_detail_grant_daily;
```

观察结果如下：

```
hive> select count(1) from report.report_o2o_pid_credit_detail_grant_daily;
Query ID = root_20200713121818_3a911a0c-2496-4e7e-b59d-b2a26266a6ab
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1594351711155_0014, Tracking URL = http://hadoop01:8088/prox
y/application_1594351711155_0014/
Kill Command = /opt/cloudera/parcels/CDH-5.16.1-1.cdh5.16.1.p0.3/lib/hadoop/bin
/hadoop job  -kill job_1594351711155_0014
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2020-07-13 12:18:19,189 Stage-1 map = 0%,  reduce = 0%
2020-07-13 12:18:25,391 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 8.89 s
ec
2020-07-13 12:18:30,544 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 10.8
 sec
MapReduce Total cumulative CPU time: 10 seconds 800 msec
Ended Job = job_1594351711155_0014
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 10.8 sec   HDFS Read: 17383
HDFS Write: 6 SUCCESS
Total MapReduce CPU Time Spent: 10 seconds 800 msec
OK
27677
Time taken: 19.128 seconds, Fetched: 1 row(s)
```

Tez 引擎

Tez 引擎需要将 COS 的 客户端安装包导入到 Tez 的压缩包内，下面以 apache-tez.0.8.5 为例进行说明：

操作步骤

1. 找到 CDH 集群安装的 tez 包，然后解压，例如/usr/local/service/tez/tez-0.8.5.tar.gz。
2. 将 COS 的 客户端安装包放置到解压后的目录下，然后重新压缩输出一个压缩包。
3. 将新的压缩包上传到 tez.lib.uris 指定的路径下（如果之前存在路径则直接替换即可）。
4. 在 CDH 主页面，找到 HIVE，重启 hiveserver 和 hivemetastore。

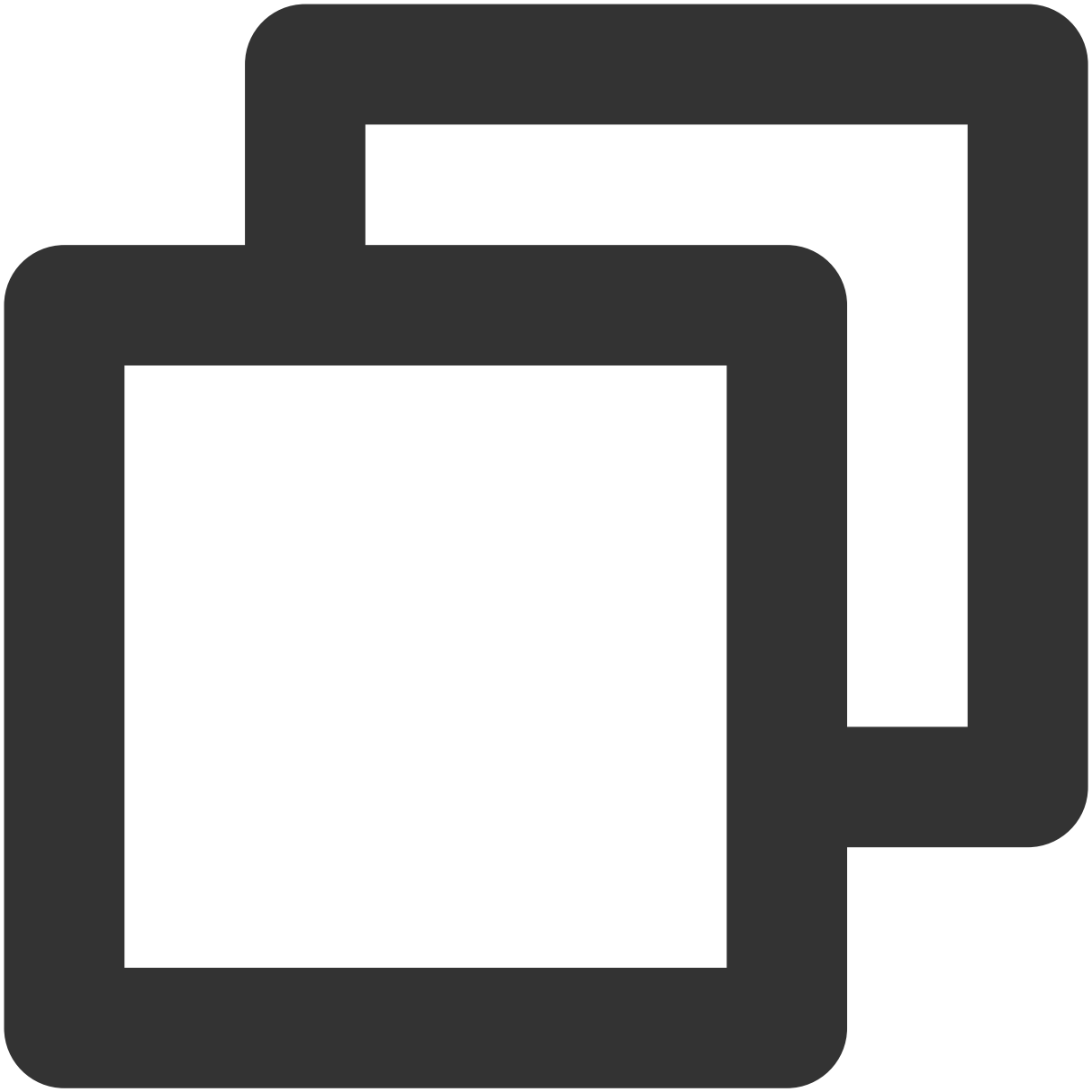
Spark

操作步骤

1. 按照 数据迁移 章节，配置好 HDFS 的相关配置，并且将 COS 的 客户端安装包，放置到 HDFS 相应的目录。
2. 重启 NodeManager 服务。

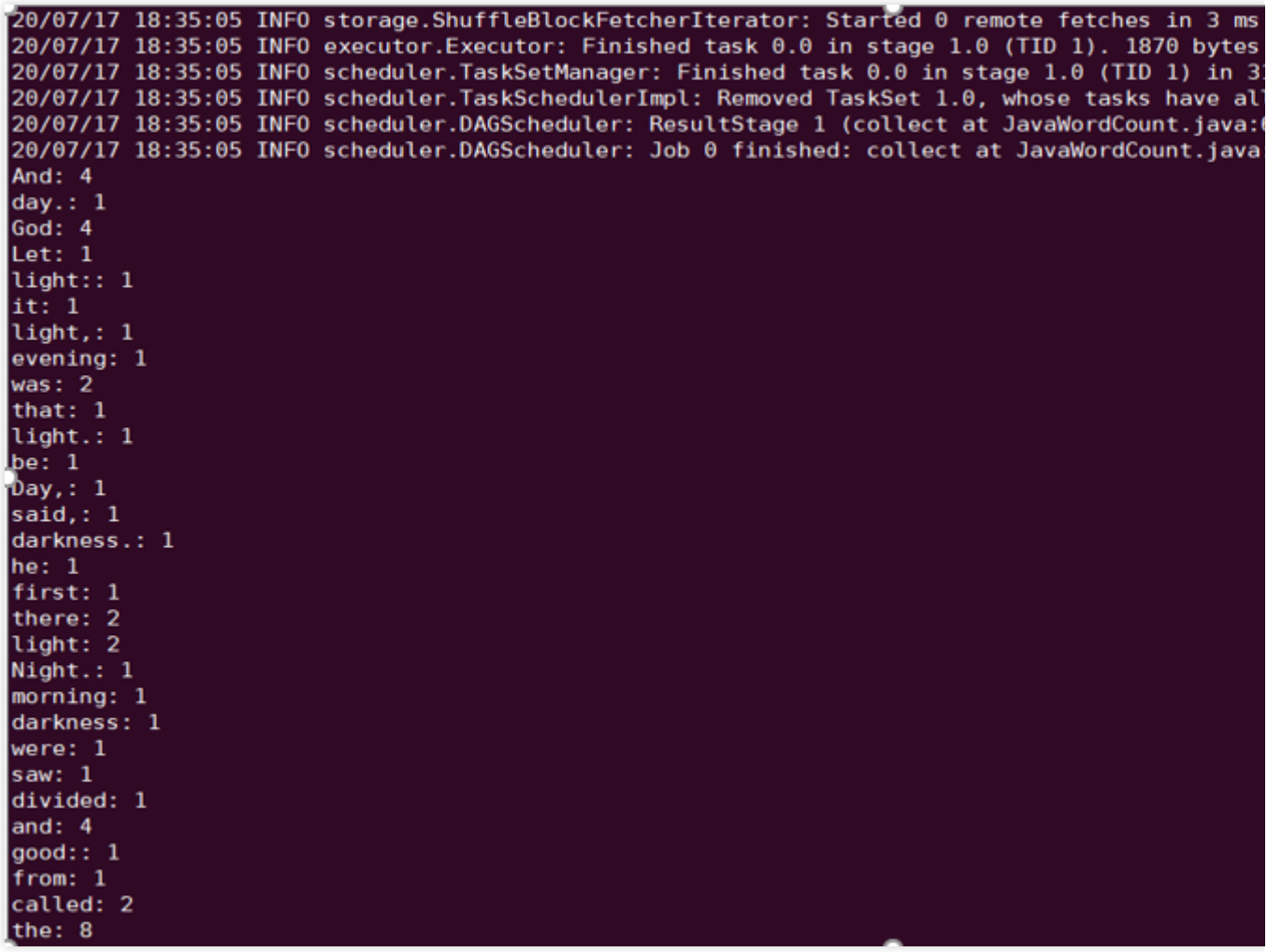
示例

以进行 Spark example word count 测试为例。



```
spark-submit  --class org.apache.spark.examples.JavaWordCount --executor-memory 4g
```

执行结果如下：



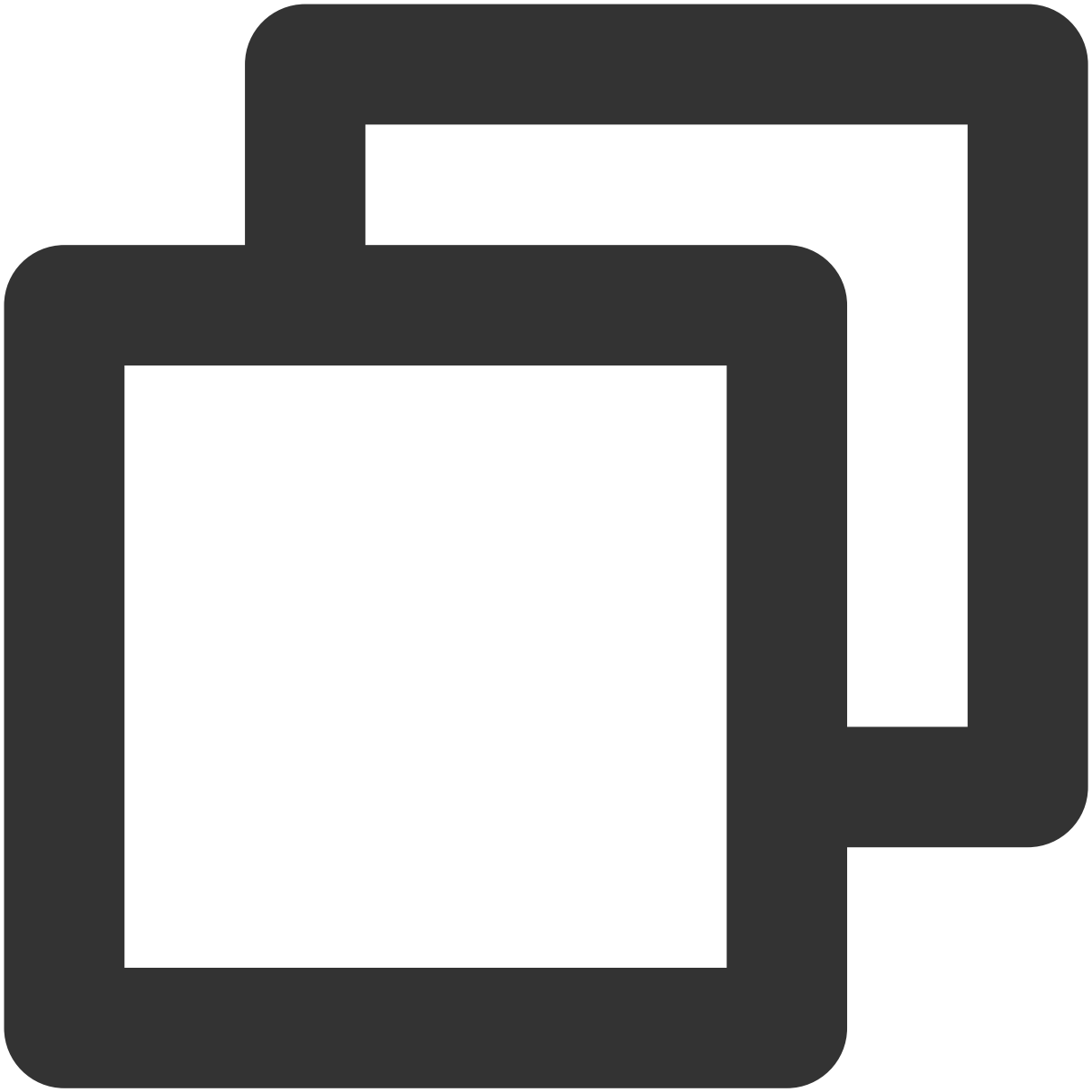
Sqoop

操作步骤

1. 按照 数据迁移 章节，配置好 HDFS 的相关配置，并且将 COS 的 客户端安装包，放置到 HDFS 相应的目录。
2. COS 的 客户端安装包还需要放到 sqoop 目录下（例如/opt/cloudera/parcels/CDH-5.16.1-1.cdh5.16.1.p0.3/lib/sqoop/）。
3. 重启 NodeManager 服务。

示例

以导出 MYSQL 表到 COS 为例，可参考 关系型数据库和 HDFS 的导入导出 文档进行测试。



```
sqoop import --connect "jdbc:mysql://IP:PORT/mysql" --table sqoop_test --username r
```

执行结果如下：

```
20/07/17 18:48:33 INFO mapreduce.Job:  map 100% reduce 0%
20/07/17 18:48:33 INFO mapreduce.Job: Job job_1594976906551_0011 completed successful
20/07/17 18:48:33 INFO mapreduce.Job: Counters: 35
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=526689
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=295
                HDFS: Number of bytes written=0
                HDFS: Number of read operations=3
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=0
                OFS: Number of bytes read=0
                OFS: Number of bytes written=104
                OFS: Number of read operations=0
                OFS: Number of large read operations=0
                OFS: Number of write operations=3
        Job Counters
                Launched map tasks=3
                Other local map tasks=3
                Total time spent by all maps in occupied slots (ms)=36308
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms)=9077
                Total vcore-milliseconds taken by all map tasks=9077
                Total megabyte-milliseconds taken by all map tasks=37179392
        Map-Reduce Framework
                Map input records=3
                Map output records=3
                Input split bytes=295
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=277
                CPU time spent (ms)=6430
                Physical memory (bytes) snapshot=1371717632
                Virtual memory (bytes) snapshot=18832379904
                Total committed heap usage (bytes)=6655311872
        File Input Format Counters
                Bytes Read=0
        File Output Format Counters
                Bytes Written=104
20/07/17 18:48:33 INFO mapreduce.ImportJobBase: Transferred 0 bytes in 13.0961 second
20/07/17 18:48:33 INFO mapreduce.ImportJobBase: Retrieved 3 records.
20/07/17 18:48:33 INFO fs.CHDFSHadoopFileSystemAdapter: actual-file-system-close used
20/07/17 18:48:33 INFO fs.CHDFSHadoopFileSystemAdapter: end-close time: 1594982913402
```

Presto

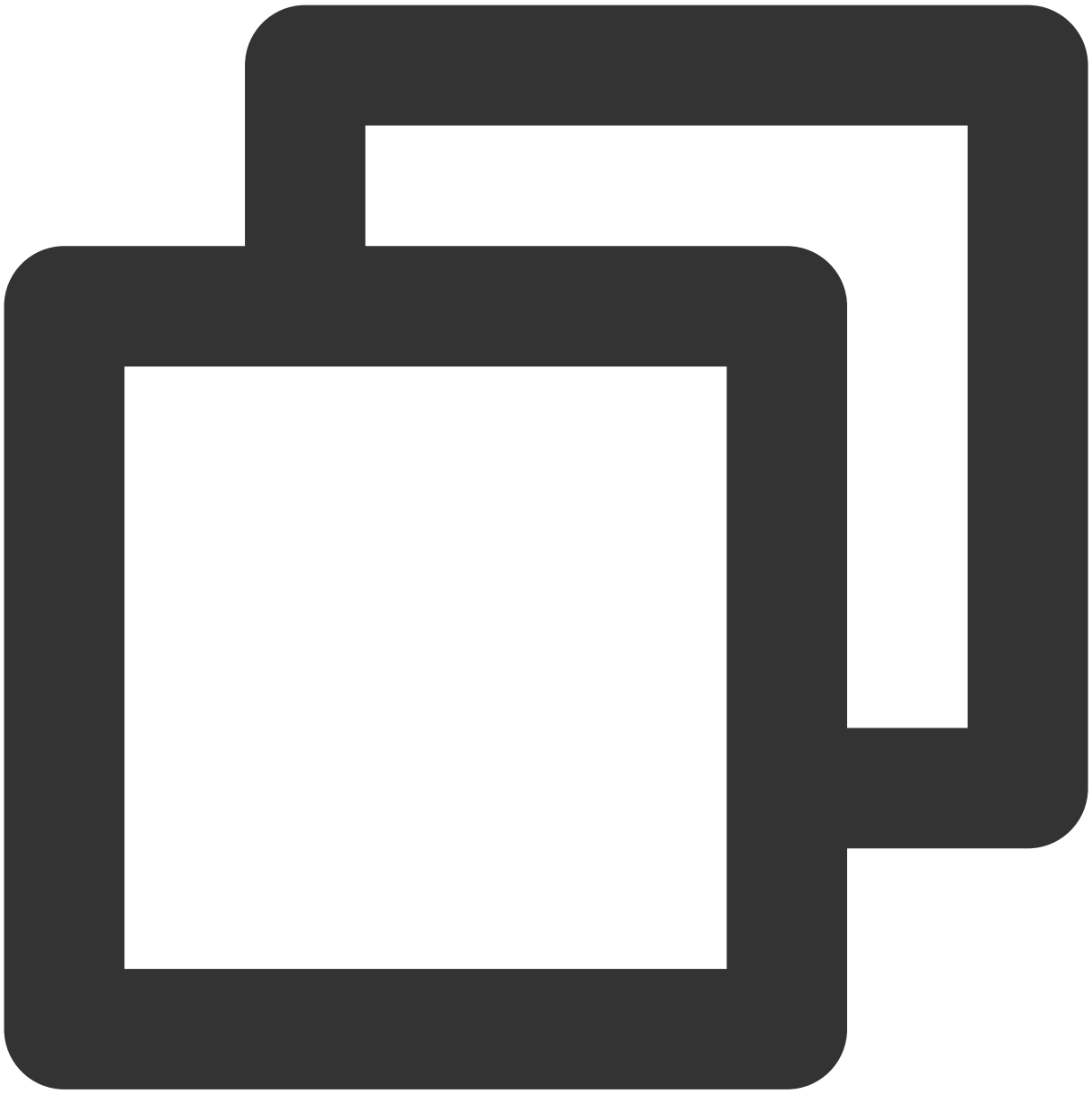
操作步骤

1. 按照 数据迁移 章节，配置好 HDFS 的相关配置，并且将 COS 的 客户端安装包，放置到 HDFS 相应的目录。
2. COS 的 客户端安装包还需要放到 presto 目录下（例如/usr/local/services/cos_presto/plugin/hive-hadoop2）。
3. 由于 presto 不会加载 hadoop common 下的 gson-2...jar，需将 gson-2...jar 也放到 presto 目录下（例如 /usr/local/services/cos_presto/plugin/hive-hadoop2，仅 COS 依赖 gson）。
4. 重启 HiveServer、HiveMetaStore 和 Presto 服务。

版权所有：腾讯云计算（北京）有限责任公司

示例

以 HIVE 创建 Location 为 COS 的表查询为例：



```
select * from chdfs_test_table where bucket is not null limit 1;
```

说明

chdfs_test_table 为 location 是 ofs scheme 的表。

查询结果如下：

```
presto:inventory_search> select * from chdfs_test_table_0720 where bucket
   appid    |       bucket        |                          path
------------+---------------------+-------------------------------------
 125        | ssuupv80105841qq206 | 444600684/20190316/3/01a9ee49bd4045179
(1 row)

Query 20200720 112833 00061 thb89, FINISHED, 7 nodes
```

使用 Hadoop Filesystem API 代码访问 COS 元数据加速存储桶

最近更新时间：2023-03-02 16:10:46

操作场景

如果对象存储（Cloud Object Storage，COS）存储桶开启了元数据加速，除了可以使用 Hadoop 命令行、大数据组件等方式操作外，还可以通过 Hadoop Filesystem API，使用 Java 代码来访问元数据加速桶。本文指导您如何通过 Java 代码访问元数据加速桶。

前提条件

- 确保已经开通元数据加速，并且进行了正确的环境部署和 HDFS 协议配置。具体部署和配置，详情请参见 使用 HDFS 协议访问已开启元数据加速器的存储桶。
- 如果有 Hadoop 环境，可以验证下 Hadoop 命令行是否能正确访问。

操作步骤

1. 新建 maven 工程，并在 maven 的 pom.xml 中添加以下依赖项（请根据自己实际 Hadoop 版本及环境设置 hadoop-common 包、hadoop-cos 包和 cos_api-bundle 包的版本）。

```
<dependencies>
<dependency>
<groupId>org.apache.hadoop</groupId>
<artifactId>hadoop-common</artifactId>
<version>2.8.5</version>
<scope>provided</scope>
</dependency>
<dependency>
<groupId>com.qcloud.cos</groupId>
<artifactId>hadoop-cos</artifactId>
<version>xxx</version>
</dependency>
<dependency>
<groupId>com.qcloud</groupId>
<artifactId>cos_api-bundle</artifactId>
<version>xxx</version>
```

```
</dependency>

</dependencies>
```

2. 参考如下 hadoop 的代码进行修改。其中的配置项可参见 配置项说明 文档进行修改。**以及重点关注其中数据持久化和可见性相关的说明。**
以下只列出了部分常见的文件系统的操作，其他的接口可参见 Hadoop FileSystem 接口文档。

```
package com.qcloud.cos.demo;

import org.apache.commons.io.IOUtils;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FileChecksum;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;

import java.io.IOException;
import java.net.URI;
import java.nio.ByteBuffer;

public class Demo {
private static FileSystem initFS() throws IOException {
Configuration conf = new Configuration();
// 配置项可参见 https://cloud.tencent.com/document/product/436/6884#.E4.B8.8B.E8.BD.BD.E4.B8.8E.E5.AE.89.E8.A3.85
// 以下配置是必填项

conf.set("fs.cosn.impl", "org.apache.hadoop.fs.CosFileSystem");
conf.set("fs.AbstractFileSystem.cosn.impl", "org.apache.hadoop.fs.CosN");
conf.set("fs.cosn.userinfo.secretId", "xxxxxx");
conf.set("fs.cosn.userinfo.secretKey", "xxxxxx");
conf.set("fs.cosn.bucket.region", "xxxxxx");
conf.set("fs.cosn.tmp.dir", "/data/chdfs_tmp_cache");

// 配置项可参考 https://cloud.tencent.com/document/product/436/71550
// 通过 POSIX 访问方式必填配置项(推荐方式)
conf.set("fs.cosn.trsf.fs.AbstractFileSystem.ofs.impl", "com.qcloud.chdfs.fs.CHDFSDelegateFSAdapter");
conf.set("fs.cosn.trsf.fs.ofs.impl", "com.qcloud.chdfs.fs.CHDFSHadoopFileSystemAdapter");
conf.set("fs.cosn.trsf.fs.ofs.tmp.cache.dir", "com.qcloud.chdfs.fs.CHDFSHadoopFil
```

```
eSystemAdapter");
conf.set("fs.cosn.trsf.fs.ofs.impl", "com.qcloud.chdfs.fs.CHDFSHadoopFileSystemAd
apter");
conf.set("fs.cosn.trsf.fs.ofs.tmp.cache.dir", "/data/chdfs_tmp_cache");

// appid 根据实际 appid 进行替换
conf.set("fs.cosn.trsf.fs.ofs.user.appid", "1250000000");
// region 根据实际地域进行替换
conf.set("fs.cosn.trsf.fs.ofs.bucket.region", "ap-beijing");
// 其他可选配置参考官网文档 https://cloud.tencent.com/document/product/436/6884#.E4.B
8.8B.E8.BD.BD.E4.B8.8E.E5.AE.89.E8.A3.85
// 是否开启 CRC64 校验，默认不开启，此时无法使用 hadoop fs -checksum 命令获取文件的 CRC64
校验值
conf.set("fs.cosn.crc64.checksum.enabled", "true");
String cosHadoopFSUrl = "cosn://examplebucket-12500000000/";
return FileSystem.get(URI.create(cosHadoopFSUrl), conf);
}

private static void mkdir(FileSystem fs, Path filePath) throws IOException {
fs.mkdirs(filePath);
}

private static void createFile(FileSystem fs, Path filePath) throws IOException {
// 创建一个文件（如果存在则将其覆盖）
// if the parent dir does not exist, fs will create it!
FSDataOutputStream out = fs.create(filePath, true);
try {
// 写入一个文件
String content = "test write file";
out.write(content.getBytes());
} finally {
// close 返回成功，表示数据写入成功，若抛出异常，表示数据写入失败
out.close();
}
}

private static void readFile(FileSystem fs, Path filePath) throws IOException {
FSDataInputStream in = fs.open(filePath);
try {
byte[] buf = new byte[4096];
int readLen = -1;
do {
readLen = in.read(buf);
} while (readLen >= 0);
} finally {
IOUtils.closeQuietly(in);
}
```

```
}


private static void queryFileOrDirStatus(FileSystem fs, Path path) throws IOExcep
tion {
FileStatus fileStatus = fs.getFileStatus(path);
if (fileStatus.isDirectory()) {
System.out.printf("path %s is dir\n", path);
return;
}

long fileLen = fileStatus.getLen();
long accessTime = fileStatus.getAccessTime();
long modifyTime = fileStatus.getModificationTime();
String owner = fileStatus.getOwner();
String group = fileStatus.getGroup();


System.out.printf("path %s is file, fileLen: %d, accessTime: %d, modifyTime: %d,
owner: %s, group: %s\n",
path, fileLen, accessTime, modifyTime, owner, group);
}


// 默认的校验类型为 COMPOSITE-CRC32C
private static void getFileCheckSum(FileSystem fs, Path path) throws IOException
{
FileChecksum checksum = fs.getFileChecksum(path);
System.out.printf("path %s, checkSumType: %s, checkSumCrcVal: %d\n",
path, checksum.getAlgorithmName(), ByteBuffer.wrap(checksum.getBytes()).getInt
());
}


private static void copyFileFromLocal(FileSystem fs, Path chdfsPath, Path localPa
th) throws IOException {
fs.copyFromLocalFile(localPath, chdfsPath);
}


private static void copyFileToLocal(FileSystem fs, Path chdfsPath, Path localPat
h) throws IOException {
fs.copyToLocalFile(chdfsPath, localPath);
}


private static void renamePath(FileSystem fs, Path oldPath, Path newPath) throws
```

```
IOException {
fs.rename(oldPath, newPath);
}


private static void listDirPath(FileSystem fs, Path dirPath) throws IOException {
FileStatus[] dirMemberArray = fs.listStatus(dirPath);


for (FileStatus dirMember : dirMemberArray) {
System.out.printf("dirMember path %s, fileLen: %d\n", dirMember.getPath(), dirMem
ber.getLen());
}
}


// 递归删除标志用于删除目录
// 如果递归为 false 并且 dir 不为空，则操作将失败
private static void deleteFileOrDir(FileSystem fs, Path path, boolean recursive)
throws IOException {
fs.delete(path, recursive);
}


private static void closeFileSystem(FileSystem fs) throws IOException {
fs.close();
}


public static void main(String[] args) throws IOException {
// 初始化文件系统
FileSystem fs = initFS();


// 创建文件
Path chdfsFilePath = new Path("/folder/exampleobject.txt");
createFile(fs, chdfsFilePath);


// 读取文件
readFile(fs, chdfsFilePath);


// 查询文件或目录
queryFileOrDirStatus(fs, chdfsFilePath);
```

```
// 获取文件校验和
getFileCheckSum(fs, chdfsFilePath);


// 从本地复制文件
Path localFilePath = new Path("file:///home/hadoop/cosn_demo/data/exampleobject.t
xt");
copyFileFromLocal(fs, chdfsFilePath, localFilePath);


// 获取文件到本地
Path localDownFilePath = new Path("file:///home/hadoop/cosn_demo/data/exampleobje
ct.txt");
copyFileToLocal(fs, chdfsFilePath, localDownFilePath);


// 重命名
Path newPath = new Path("/doc/example.txt");
renamePath(fs, chdfsFilePath, newPath);


// 删除文件
deleteFileOrDir(fs, newPath, false);


// 创建目录
Path dirPath = new Path("/folder");
mkdir(fs, dirPath);


// 在目录中创建文件
Path subFilePath = new Path("/folder/exampleobject.txt");
createFile(fs, subFilePath);


// 列出目录
listDirPath(fs, dirPath);


// 删除目录
deleteFileOrDir(fs, dirPath, true);


// 关闭文件系统
closeFileSystem(fs);
}
}
```

3. 编译和运行。

说明：

- 运行前，请确保已正确设置 classpath。classpath 需包含 Hadoop common 包以及元数据加速桶依赖的 Jar 包的路径。
- 对于 EMR 环境，如果您按照 使用 HDFS 协议访问已开启元数据加速器的存储桶 逐步操作，那么 Hadoop common 包通常在 `/usr/local/service/hadoop/share/hadoop/common/` 目录下，元数据加速桶依赖的 Jar 包通常在 `/usr/local/service/hadoop/share/hadoop/common/lib/` 目录下。

使用 DataX 在已开启元数据加速功能的存储桶间同步数据

最近更新时间：2022-12-15 16:25:33

简介

DataX 是开源异构数据源离线同步工具，实现了包括 MySQL、SQL Server、Oracle、PostgreSQL、HDFS、Hive、HBase、OTS、ODPS 等各种异构数据源之间高效的数据同步功能。

已开启元数据加速功能的 COS 存储桶可以为业务提供 HCFS（Hadoop Compatible File System）语义访问，充当 Hadoop 系统中的 HDFS 服务的作用。

本文将介绍如何使用 DataX 在两个开启了元数据加速功能的存储桶之间同步数据。

环境依赖

- HADOOP-COS 与对应版本的 cos_api-bundle。
- DataX 版本：DataX-3.0。

下载与安装

获取 HADOOP-COS

- 在官方 Github 上下载 HADOOP-COS 与对应版本的 cos_api-bundle。
- 在官方 Github 上下载 chdfs-hadoop-plugin。

获取 DataX 软件包

在官方 Github 上下载 DataX。

安装 HADOOP-COS

下载 HADOOP-COS 后，将 `hadoop-cos-2.x.x-${version}.jar` 、 `cos_api-bundle-${version}.jar` 和 `chdfs_hadoop_plugin_network-${version}.jar` 这三个 jar 包分别拷贝到 Datax 解压路径 `plugin/reader/hdfsreader/libs/` 和 `plugin/writer/hdfswriter/libs/` 下。

使用方法

存储桶配置

进入已开启元数据加速功能的存储桶，在 **HDFS 权限配置**中，配置运行 DataX 机器的 VPC 网络。

> 说明：
> 数据源存储桶至少需要允许该 VPC 内的读请求，数据写入目标存储桶至少需要允许该 VPC 内的写请求。

DataX 配置

1. 修改 datax.py 脚本

打开 DataX 解压目录下的 bin/datax.py 脚本，修改脚本中的 CLASS_PATH 变量为如下：

```
CLASS_PATH = ("%s/lib/*:%s/plugin/reader/hdfsreader/libs/*:%s/plugin/writer/hdfsw
riter/libs/*:.") % (DATAX_HOME, DATAX_HOME, DATAX_HOME)
```

2. 在配置 JSON 文件里配置 hdfsreader 和 hdfswriter

示例 JSON 如下：

```
{
"job": {
"setting": {
"speed": {
"byte": 10485760
},
"errorLimit": {
"record": 0,
"percentage": 0.02
}
},
"content": [{
"reader": {
"name": "hdfsreader",
"parameter": {
"path": "/test/",
"defaultFS": "cosn://examplebucket1-1250000000/",
"column": ["*"],
"fileType": "text",
"encoding": "UTF-8",
"hadoopConfig": {
```

```
"fs.cosn.impl": "org.apache.hadoop.fs.CosFileSystem",
"fs.cosn.trsf.fs.ofs.bucket.region": "ap-guangzhou",
"fs.cosn.bucket.region": "ap-guangzhou",
"fs.cosn.tmp.dir": "/tmp/hadoop_cos",
"fs.cosn.trsf.fs.ofs.tmp.cache.dir": "/tmp/",
"fs.cosn.userinfo.secretId": "COS_SECRETID",
"fs.cosn.userinfo.secretKey": "COS_SECRETKEY",
"fs.cosn.trsf.fs.ofs.user.appid": "1250000000"
},
"fieldDelimiter": ","
}
},
"writer": {
"name": "hdfswriter",
"parameter": {
"path": "/",
"fileName": "hive.test",
"defaultFS": "cosn://examplebucket2-1250000000/",
"column": [
{"name":"col1","type":"int"},
{"name":"col2","type":"string"}
],
"fileType": "text",
"encoding": "UTF-8",
"hadoopConfig": {
"fs.cosn.impl": "org.apache.hadoop.fs.CosFileSystem",
"fs.cosn.trsf.fs.ofs.bucket.region": "ap-guangzhou",
"fs.cosn.bucket.region": "ap-guangzhou",
"fs.cosn.tmp.dir": "/tmp/hadoop_cos",
"fs.cosn.trsf.fs.ofs.tmp.cache.dir": "/tmp/",
"fs.cosn.userinfo.secretId": "COS_SECRETID",
"fs.cosn.userinfo.secretKey": "COS_SECRETKEY",
"fs.cosn.trsf.fs.ofs.user.appid": "1250000000"
},
"fieldDelimiter": ",",
"writeMode": "append"
}
}
}]
}
}
```

配置说明如下：

- hadoopConfig 配置为 cosn 所需要的配置。
- defaultFS 填写为 cosn 的路径，例如 `cosn://examplebucket-1250000000/` 。

- fs.cosn.userinfo.region 和 fs.cosn.trsf.fs.ofs.bucket.region 修改为存储桶所在的地域，例如 `ap-guangzhou`，详情请参见 地域和访问域名。
- COS_SECRETID 和 COS_SECRETKEY 修改为 COS 密钥。
- fs.ofs.user.appid 和 fs.cosn.trsf.fs.ofs.user.appid 修改为用户 appid。

> 注意：
>
> fs.cosn.trsf.fs.ofs.bucket.region 和 fs.cosn.trsf.fs.ofs.user.appid 在 hadoop-cos 8.1.7及以上版本中已去除，使用时请注意版本差异。其他配置请参考 HDFS reader、writer 配置项。

执行数据迁移

将配置文件保存为 hdfs_job.json 文件，存放到 job 目录下，并执行以下命令行：

```
[root@172 /usr/local/service/datax]# python bin/datax.py job/hdfs_job.json
```

观察屏幕正常输出如下：

```
2022-10-23 00:25:24.954 [job-0] INFO JobContainer -
[total cpu info] =>
averageCpu | maxDeltaCpu | minDeltaCpu
-1.00% | -1.00% | -1.00%

[total gc info] =>
NAME | totalGCCount | maxDeltaGCCount | minDeltaGCCount | totalGCTime | maxDeltaG
CTime | minDeltaGCTime
PS MarkSweep | 1 | 1 | 1 | 0.034s | 0.034s | 0.034s
PS Scavenge | 14 | 14 | 14 | 0.059s | 0.059s | 0.059s
2022-10-23 00:25:24.954 [job-0] INFO JobContainer - PerfTrace not enable!
2022-10-23 00:25:24.954 [job-0] INFO StandAloneJobContainerCommunicator - Total 1
000003 records, 9322478 bytes | Speed 910.40KB/s, 100000 records/s | Error 0 reco
rds, 0 bytes | All Task WaitWriterTime 1.000s | All Task WaitReaderTime 6.259s |
Percentage 100.00%
2022-10-23 00:25:24.955 [job-0] INFO JobContainer -
任务启动时刻 : 2022-10-23 00:25:12
任务结束时刻 : 2022-10-23 00:25:24
任务总计耗时 : 12s
任务平均流量 : 910.40KB/s
记录写入速度 : 100000rec/s
读出记录总数 : 1000003
读写失败总数 : 0
```

使用 Ranger 和 Kerberos 场景下

在 Hadoop 权限体系中，认证由 Kerberos 提供，授权鉴权由 Ranger 负责。开启了 Ranger 和 Kerberos 后，使用 DataX 对接已开启元数据加速功能的存储桶的步骤大致和上述步骤一样，但有一些需要额外进行操作和配置。

1. 已开启元数据加速功能的存储桶支持腾讯云大数据权限管控方案 COS Ranger Service（在 EMR 控制台购买 Ranger 和 cosranger 组件时会自动安装；如果自行安装，可参考 CHDFS Ranger 权限体系解决方案）。

2. 将 `cosn-ranger-interface-1.x.x-${version}.jar` 和 `hadoop-ranger-client-for-hadoop-${version}.jar` 这两个 jar 包拷贝到 Datax 解压路径 `plugin/reader/hdfsreader/libs/` 和 `plugin/writer/hdfswriter/libs/` 下。单击前往 Github 下载

3. 进入已开启元数据加速功能的存储桶，在 **HDFS 鉴权模式**中，选择 **Ranger 鉴权**，配置 Ranger 地址（非 COS Ranger 地址）

4. 在 JSON 配置文件里配置 hdfsreader 和 hdfswriter。

```
{
"job": {
"setting": {
"speed": {
"byte": 10485760
},
"errorLimit": {
"record": 0,
"percentage": 0.02
}
},
"content": [{
"reader": {
"name": "hdfsreader",
"parameter": {
"path": "/test/",
"defaultFS": "cosn://examplebucket1-1250000000/",
"column": ["*"],
"fileType": "text",
"encoding": "UTF-8",
"hadoopConfig": {
"fs.cosn.impl": "org.apache.hadoop.fs.CosFileSystem",
"fs.cosn.trsf.fs.ofs.bucket.region": "ap-guangzhou",
"fs.cosn.bucket.region": "ap-guangzhou",
"fs.cosn.tmp.dir": "/tmp/hadoop_cos",
"fs.cosn.trsf.fs.ofs.tmp.cache.dir": "/tmp/",
```

```
"fs.cosn.trsf.fs.ofs.user.appid": "1250000000",
"fs.cosn.credentials.provider": "org.apache.hadoop.fs.auth.RangerCredentialsPro
vider",
"qcloud.object.storage.zk.address": "172.16.0.30:2181",
"qcloud.object.storage.ranger.service.address": "172.16.0.30:9999",
"qcloud.object.storage.kerberos.principal": "hadoop/172.16.0.30@EMR-5IUR9VWW"
},
"haveKerberos": "true",
"kerberosKeytabFilePath": "/var/krb5kdc/emr.keytab",
"kerberosPrincipal": "hadoop/172.16.0.30@EMR-5IUR9VWW",
"fieldDelimiter": ","
}
},
"writer": {
"name": "hdfswriter",
"parameter": {
"path": "/",
"fileName": "hive.test",
"defaultFS": "cosn://examplebucket2-1250000000/",
"column": [
{"name":"col1","type":"int"},
{"name":"col2","type":"string"}
],
"fileType": "text",
"encoding": "UTF-8",
"hadoopConfig": {
"fs.cosn.impl": "org.apache.hadoop.fs.CosFileSystem",
"fs.cosn.trsf.fs.ofs.bucket.region": "ap-guangzhou",
"fs.cosn.bucket.region": "ap-guangzhou",
"fs.cosn.tmp.dir": "/tmp/hadoop_cos",
"fs.cosn.trsf.fs.ofs.tmp.cache.dir": "/tmp/",
"fs.cosn.trsf.fs.ofs.user.appid": "1250000000",
"fs.cosn.credentials.provider": "org.apache.hadoop.fs.auth.RangerCredentialsPro
vider",
"qcloud.object.storage.zk.address": "172.16.0.30:2181",
"qcloud.object.storage.ranger.service.address": "172.16.0.30:9999",
"qcloud.object.storage.kerberos.principal": "hadoop/172.16.0.30@EMR-5IUR9VWW"
},
"haveKerberos": "true",
"kerberosKeytabFilePath": "/var/krb5kdc/emr.keytab",
"kerberosPrincipal": "hadoop/172.16.0.30@EMR-5IUR9VWW",
"fieldDelimiter": ",",
"writeMode": "append"
}
}
}]
```

版权所有：腾讯云计算（北京）有限责任公司

```
}
}
```

新增配置说明如下：

- fs.cosn.credentials.provider 配置为 org.apache.hadoop.fs.auth.RangerCredentialsProvider ， 使用 Ranger 鉴权。
- qcloud.object.storage.zk.address 配置 ZOOKEEPER 地址。
- qcloud.object.storage.ranger.service.address 配置 COS Ranger 地址。
- haveKerberos 配置为 true。
- qcloud.object.storage.kerberos.principal 和 kerberosPrincipal 配置为 Kerberos 认证 Principal 名（在开启了 kerboros 的 emr 环境中可从 core-site.xml 读取）。
- kerberosKeytabFilePath 配置为 keytab 认证文件的绝对路径（在开启了 kerboros 的 emr 环境中可从 ranger-admin-site.xml 读取）。

常见问题

报错 java.io.IOException: Permission denied: no access groups bound to this mountPoint examplebucket2-1250000000, access denied 或者 java.io.IOException: Permission denied: No access rules matched，该如何处理？

检查存储桶的 **HDFS 权限配置**中的 VPC 网络配置，配置好运行机器的 ip 地址或网段，例如 emr 需要配置所有节点的 ip 地址。

报错 java. lang. RuntimeException: java. lang.ClassNotFoundException: Class org.apache.hadoop.fs.con.ranger.client.RangerQcloudObjectStorageClientImpl not found，该如何处理？

检查是否将 `cosn-ranger-interface-1.x.x-${version}.jar` 和 `hadoop-ranger-client-for-hadoop-${version}.jar` 拷贝到 Datax 解压路径 `plugin/reader/hdfsreader/libs/` 以及 `plugin/writer/hdfswriter/libs/` 下。单击前往 Github 下载

报错 java.io.IOException: Login failure for hadoop/_HOST@EMR-5IUR9VWW from keytab /var/krb5kdc/emr.keytab: javax.security.auth.login.LoginException: Unable to obtain password from user，该如何处理？

检查 kerberosPrincipal 和 qcloud.object.storage.kerberos.principal 项，是否将 hadoop/172.16.0.30@EMR-5IUR9VWW 错误地设置为 `hadoop/_HOST@EMR-5IUR9VWW` 。因为 datax 并不能解析 `_HOST` 域名，所以需要将 `_HOST` 换成 ip。可以使用 `klist -ket /var/krb5kdc/emr.keytab` 命令来查找合适的 Principal。

报错 java.io.IOException: init fs.cosn.ranger.plugin.client.impl failed，该如何处理？

检查 json 文件中的 hadoopConfig 项是否没有配置 qcloud.object.storage.kerberos.principal。如是，则需要配置。

大数据安全
Ranger 介绍说明

最近更新时间：2024-01-06 11:17:58

背景

COS Ranger Service 是腾讯云存算分离推出的大数据权限管控方案，具有细粒度、兼容 Hadoop Ranger 以及可插拔的优势，便于用户统一管理大数据组件和云端托管存储权限，具体架构方案及说明可查看 COS Ranger 权限体系解决方案。
COS Ranger Service 一经推出后便得到广泛使用，然而由于客户业务众多，背景复杂，产生一些问题，现整理相关 Ranger 介绍说明、版本细节以及注意事项。

版本介绍

相关组件

相关组件主要有 Ranger-Plugin、COS Ranger Server、COS Ranger Client（也就是 Hadoop-Ranger-Client）、COSN Ranger Interface。

Ranger-Plugin

根据 Ranger 协议（具体可参见 Apache 官方文档）提供 Ranger 服务端的服务定义插件。它们提供了 Ranger 侧的 COS 服务描述，部署了该插件后，用户即可在 Ranger 的控制页面上，填写 COS 的权限策略，例如设置 path、bucket、user、group 等访问策略。

COS Ranger Server

该服务集成了 Ranger 的客户端，周期性从 Ranger 服务端同步权限策略，在收到客户的鉴权请求后，在本地进行权限校验。 同时它提供了 Hadoop 中 DelegationToken 相关的生成、续租等接口。
EMR 环境中默认安装目录在 /usr/local/service/cosranger/lib 下。例如包名 cos-ranger-service-5.1.2-jar-with-dependencies.jar，5.1.2 即为 cos ragner server 的版本号。

COS Ranger Client

Hadoop sdk 插件通过 core-site.xml 文件中配置对其进行动态加载，把权限校验的请求转发给 COS Ranger Server。
EMR 环境中默认安装目录在 /usr/local/service/hadoop/share/hadoop/common/lib 下。例如包名 hadoop-ranger-client-for-hadoop-2.8.5-5.0.jar，2.8.5 是 hadoop 版本号，5.0是该包的版本号。

COSN Ranger Interface

该插件由 COS Ranger Server 和 COS Ranger Client 公共数据定义以及接口定义。
EMR 环境中默认安装目录在 /usr/local/service/hadoop/share/hadoop/common/lib 下。例如包名 cosn-ranger-interface-1.0.4.jar，1.0.4即为 COSN Ranger Interface 版本号。
以上 jar 包均可前往 Github 获取。其他组件例如 impala、presto 等对应的 COS Ranger Client 包联系 EMR 团队。
在 EMR 控制台购买 Ranger 和 cosranger 组件时会自动安装以上组件；如果自行安装，可参考 CHDFS Ranger 权限体系解决方案。

版本说明

根据核心架构区分，版本总体上分为两大类：**依赖 zookeeper 服务与发现** 和 **不依赖 zookeeper 服务与发现** 版本。COS Ranger Server 提供服务，供 COS Ranger Client 调用。
如果 COS Ranger Client 通过 zookeeper 去发现 COS Ranger Server 的服务地址，就需要配置 zookeeper 地址，这就是**依赖 zookeeper 版本**的特性。
如果 COS Ranger Client 不依赖 zookeeper 去发现 COS Ranger Server 服务，则需要配置 qcloud.object.storage.ranger.service.address 直接指定 COS Ranger Server 服务地址，不必去依赖 zookeeper 去发现 COS Ranger Server 服务；这就是**不依赖 zookeeper 版本**的特性。

版本对应关系

组件	依赖 zookeeper 服务与发现	不依赖 zookeeper 服务与发现
COS Ranger Server	v5.0.9 及早期版本	v5.1.1 及以上版本
COS Ranger Client	v3.9 及早期版本	v4.1 及以上版本
COSN Ranger Interface	v1.0.3 版本	v1.0.4 版本及以上

注意

如使用**不依赖 zookeeper 服务与发现**的版本，COS Ranger Server 须 v5.1.1以上（推荐 **v5.1.2**），COS Ranger Client 须 v4.1 以上（推荐 **v5.0**），且 COSN Ranger Interface 须 v1.0.4 版本及以上。

版本兼容关系

版本对应关系可依照上述两大类去匹配；但由于众多客户，背景复杂，不一定按照上述两大类来区分的，故下表列出各组件间的兼容关系，每一行表示可兼容。

COS Ranger Client 版本	COS Ranger Server 版本	COSN Ranger Interface 版本	是否依赖 zookeeper 服务与发现
version ≤ v3.9	version ≤ v5.0.9	v1.0.3	是
version ≤ v3.9	version ≥ v5.1.1	v1.0.4	是
version ≥ v4.1	version ≥ v5.1.1	v1.0.4	否

version ≥ v5.0	version ≤ v5.0.9	v1.0.4	是
version ≥ v5.0	version ≥ v5.1.1	v1.0.4	否

注意

COS Ranger Client v5.0 可兼容所有版本 COS Ranger Server 。

COS Ranger Client v4.1 只能兼容 v 5.1.1及以上版本的 COS Ranger Server。

COS Ranger Client v3.x 虽然也可以兼容所有版本的 COS Ranger Server，但只能依赖 zookeeper 去发现COS Ranger Server 服务（后文会说明依赖 zookeeper 版本的弊病）。

COS Ranger Client 和 COSN Ranger Interface 包需要放在同一目录下，供 hadoop sdk 插件动态加载。

使用说明

元数据加速桶和 CHDFS 文件系统需要在官网控制台打开 Ranger 校验。

使用**依赖 zookeeper 服务与发现**版本，core-site.xml 需要配置 qcloud.object.storage.zk.address，value 为 zookeeper 地址（用逗号分隔）。

如果 COS Ranger Server 使用 v5.1.1 及 v5.1.2 版本，而 COS Ranger Client 使用 v3.x 版本，此时仍然是依赖 zookeeer 注册与发现，core-site.xml 需要配置 qcloud.object.storage.zk.address，value 为 zookeeper 地址（用逗号分隔，例如10.0.0.8:2181,10.0.0.9:2181,10.0.0.10:2181）。

使用**不依赖 zookeeper 服务与发现**版本，core-site.xml 需要配置 qcloud.object.storage.ranger.service.address，value 为 COS Ranger Server 服务地址（用逗号分隔，例如127.0.0.1:9999,128.0.0.1:9999）。

使用 ofs 协议访问，core-site.xml 需要配置 fs.ofs.ranger.enable.flag 为 true。

使用 cosn 协议访问，core-site.xml 需要配置 fs.cosn.credentials.provider ，设置为：org.apache.hadoop.fs.auth.RangerCredentialsProvider。

配置项说明

配置项	说明	示例
qcloud.object.storage.zk.address	COS Ranger Server 注册的 zk 地址	10.0.0.8:2181,10.0.0.9:2181,10.0.0.10:2181
qcloud.object.storage.ranger.service.address	COS Ranger Server RPC 服务地址	127.0.0.1:9999,128.0.0.1:9999

fs.ofs.ranger.enable.flag	使用 ofs 协议时的 Ranger 开关	true
fs.cosn.credentials.provider	使用 cosn 协议时的 Ranger 认证类路径	org.apache.hadoop.fs.auth.RangerCredentialsProvide
fs.cosn.posix.bucket.use_ofs_ranger.enabled	是否走 chdfs ranger 鉴权配置	默认为 false，即 COSN Ranger 鉴权。 配置为 true，则为 chdfs ranger 鉴权 说明：该项为 hadoop-cos v8.1.7 及以上版本的新增配置项

配置项表

组件版本	配置项
cos ranger verser ≤ v5.0.9 cos ranger client ≤ v3.9 OR = v5.0 ofs 协议访问	qcloud.object.storage.zk.address fs.ofs.ranger.enable.flag
cos ranger verser ≤ v5.0.9 cos ranger client ≤ v3.9 OR = v5.0 cosn 协议访问	qcloud.object.storage.zk.address fs.cosn.credentials.provider
cos ranger verser = v5.1.1 OR = v5.1.2 cos ranger client ≥ v4.1 ofs 协议访问	qcloud.object.storage.ranger.service.address fs.ofs.ranger.enable.flag
cos ranger verser = v5.1.1 OR = v5.1.2 cos ranger client ≥ v4.1 cosn 协议访问	qcloud.object.storage.ranger.service.address fs.cosn.credentials.provider

注意

如果使用 cosn 协议访问元数据加速桶，且希望走 chdfs ranger 鉴权，请设置 fs.cosn.posix.bucket.use_ofs_ranger.enabled 为 true；且 hadoop-cos 版本要大于等于 v8.1.7。

版权所有：腾讯云计算（北京）有限责任公司

新增或调整上述配置，大数据组件如 YARN 中 ResourceManager/NodeManager、Hive 中 HiveMetaStore/HiveServer2、Impala 及 Presto 下应用等都需要重启。

推荐版本

组件	版本号
cos-ranger-server	>= v5.1.2
cos-ranger-client	>= v5.0
cosn-ranger-interface	>= v1.0.4

推荐说明

之所以推荐上述版本原因如下：

zookeeper 只用来选主，不进行服务注册与发现，极大减少大数据作业时 zookeeper 压力；因为每一个大数据作业时，会有大量 task 去访问 zookeeper 来进行 COS Ranger Server 服务发现，对 zookeeper 压力比较大，从而影响其他大数据组件稳定。

V5.0 版本的 hadoop-ranger-client 包可兼容旧版本 COS Ranger Server 包，可方便老用户升级 COS Ranger Server。

COS Ranger Server 5.1.2之前的版本，可能会存在获取到的 leader IP 和 leader latch 中的 IP 不一致的情况；而且，后期会简化 COS Ranger Server 注册到 zk 的信息，有利于后续拓展或升级。

修复若干 bug。

认证和鉴权常见问题

报错 IOException: init fs.ofs.ranger.client.impl failed，该如何处理？

若 Caused by: java.io.IOException: invalid zk address null，则 core-site.xml 需要配置 qcloud.object.storage.zk.address，value 为 zookeeper 地址（用逗号分隔，例如 10.0.0.8:2181,10.0.0.9:2181,10.0.0.10:2181）。

若 Caused by: java.io.IOException: ranger client is null, maybe ranger server for qcloud object storage is not deployed! 则参考下一个问题。

报错 ranger client is null, maybe ranger server for qcloud object storage is not deployed!，该如何处理？

这种报错主要原因主要有以下几种：

如果 hadoop-ranger-client 包是 v3.8 及以下版本，可能是 zookeeper watch 丢失导致的，建议升级到 v5.0。

检查配置项 qcloud.object.storage.zk.address 或 qcloud.object.storage.ranger.service.address。

检查下 COS Ranger Server 服务和进程是否正常。

版权所有：腾讯云计算（北京）有限责任公司

报错 Expect ranger service addresses: [127.0.0.1:6080,128.0.0.1:6080], but actual ranger service address，该如何处理？

这种报错原因是元数据加速桶或 CHDFS 官网控制台打开 Ranger 校验，然而客户端并没有打开 Ranger 校验。

使用 ofs 协议访问，core-site.xml 需要配置 fs.ofs.ranger.enable.flag 为 true。

使用 cosn 协议访问，core-site.xml 需要配置 fs.cosn.credentials.provider，设置为：org.apache.hadoop.fs.auth.RangerCredentialsProvider。

RangerQcloudObjectStorageClient 类未找到，该如何处理？

```
Exception in thread "main" java.lang.NoClassDefFoundError: org/apache/hadoop/fs/cosn/ranger/client/Ranger
   at java.lang.ClassLoader.defineClass1(Native Method)
   at java.lang.ClassLoader.defineClass(ClassLoader.java:756)
   at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:142)
   at java.net.URLClassLoader.defineClass(URLClassLoader.java:468)
   at java.net.URLClassLoader.access$100(URLClassLoader.java:74)
   at java.net.URLClassLoader$1.run(URLClassLoader.java:369)
   at java.net.URLClassLoader$1.run(URLClassLoader.java:363)
   at java.security.AccessController.doPrivileged(Native Method)
   at java.net.URLClassLoader.findClass(URLClassLoader.java:362)
   at java.lang.ClassLoader.loadClass(ClassLoader.java:418)
   at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:352)
   at java.lang.ClassLoader.loadClass(ClassLoader.java:405)
   at java.lang.ClassLoader.loadClass(ClassLoader.java:351)
   at java.lang.Class.forName0(Native Method)
   at java.lang.Class.forName(Class.java:348)
   at org.apache.hadoop.conf.Configuration.getClassByNameOrNull(Configuration.java:2134)
   at org.apache.hadoop.conf.Configuration.getClassByName(Configuration.java:2099)
   at chdfs.1.0.6.com.qcloud.chdfs.fs.CHDFSHadoopFileSystem.initRangerClientImpl(CHDFSHadoopFileSystem.jav
   at chdfs.1.0.6.com.qcloud.chdfs.fs.CHDFSHadoopFileSystem.doInitialize(CHDFSHadoopFileSystem.java:313)
   at chdfs.1.0.6.com.qcloud.chdfs.fs.CHDFSHadoopFileSystem.initialize(CHDFSHadoopFileSystem.java:245)
   at com.qcloud.chdfs.fs.CHDFSHadoopFileSystemAdapter.initialize(CHDFSHadoopFileSystemAdapter.java:106)
   at org.apache.hadoop.fs.FileSystem.createFileSystem(FileSystem.java:2669)
   at org.apache.hadoop.fs.FileSystem.access$200(FileSystem.java:94)
   at org.apache.hadoop.fs.FileSystem$Cache.getInternal(FileSystem.java:2703)
   at org.apache.hadoop.fs.FileSystem$Cache.get(FileSystem.java:2685)
   at org.apache.hadoop.fs.FileSystem.get(FileSystem.java:373)
   at org.apache.hadoop.fs.Path.getFileSystem(Path.java:295)
   at org.apache.hadoop.mapred.FileInputFormat.singleThreadedListStatus(FileInputFormat.java:258)
   at org.apache.hadoop.mapred.FileInputFormat.listStatus(FileInputFormat.java:229)
   at org.apache.hadoop.mapred.FileInputFormat.getSplits(FileInputFormat.java:315)
```

缺少 cosn-ranger-interface 包，可前往 Github 的 cosn-ranger-interface 目录下获取。

其他相关类未找到，可确认下 cosn-ranger-interface 包和 hadoop-ranger-client 包是否存在、版本是否匹配、以及是否放在正确路径下。

其他相关类未找到，还有一种情况是包 shade 路径问题，这个需要联系我们协助排查和处理。

报 NoSuchMethodError，该如何处理？

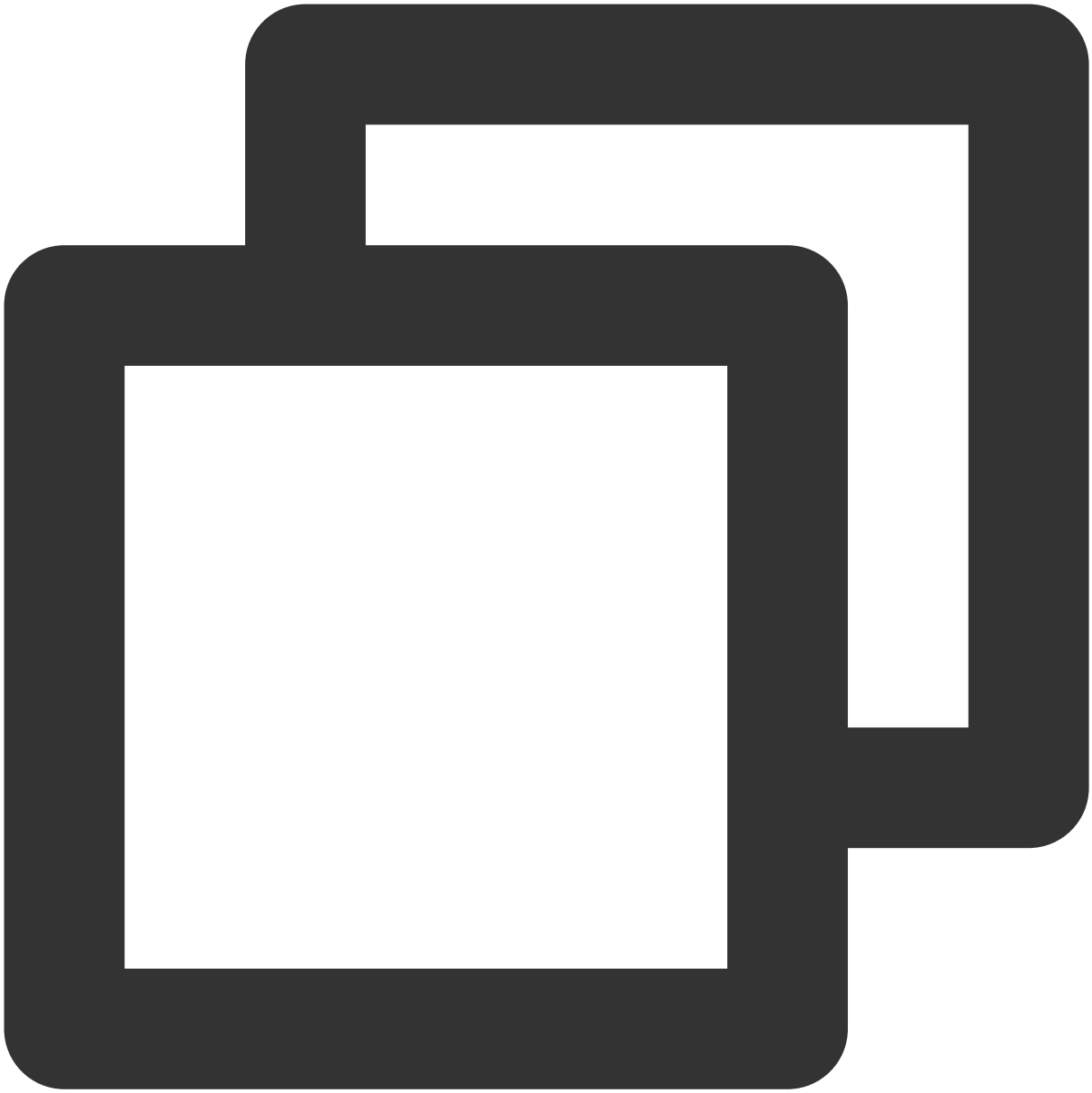
```
java.lang.NoSuchMethodError: org.apache.hadoop.fs.cosn.ranger.client.RangerQcloudObjectStorageCl
        at org.apache.hadoop.fs.CosFileSystem.checkPermission(CosFileSystem.java:1146)
        at org.apache.hadoop.fs.CosFileSystem.mkdirs(CosFileSystem.java:575)
        at org.apache.hadoop.fs.FilterFileSystem.mkdirs(FilterFileSystem.java:332)
        at io.prestosql.plugin.hive.util.HiveWriteUtils.createDirectory(HiveWriteUtils.java:572)
        at io.prestosql.plugin.hive.util.HiveWriteUtils.createTemporaryPath(HiveWriteUtils.java:
        at io.prestosql.plugin.hive.HiveLocationService.forExistingTable(HiveLocationService.jav
        at io.prestosql.plugin.hive.HiveMetadata.beginInsert(HiveMetadata.java:1587)
        at io.prestosql.plugin.hive.HiveMetadata.beginInsert(HiveMetadata.java:277)
        at io.prestosql.spi.connector.ConnectorMetadata.beginInsert(ConnectorMetadata.java:443)
        at io.prestosql.plugin.base.classloader.ClassLoaderSafeConnectorMetadata.beginInsert(Cla
        at io.prestosql.metadata.MetadataManager.beginInsert(MetadataManager.java:840)
        at io.prestosql.sql.planner.optimizations.BeginTableWrite$Rewriter.createWriterTarget(Be
        at io.prestosql.sql.planner.optimizations.BeginTableWrite$Rewriter.visitTableFinish(Begi
        at io.prestosql.sql.planner.optimizations.BeginTableWrite$Rewriter.visitTableFinish(Begi
        at io.prestosql.sql.planner.plan.TableFinishNode.accept(TableFinishNode.java:106)
        at io.prestosql.sql.planner.plan.SimplePlanRewriter$RewriteContext.rewrite(SimplePlanRew
        at io.prestosql.sql.planner.plan.SimplePlanRewriter$RewriteContext.lambda$defaultRewrite
        at java.base/java.util.stream.ReferencePipeline$3$1.accept(ReferencePipeline.java:195)
        at java.base/java.util.Collections$2.tryAdvance(Collections.java:4747)
        at java.base/java.util.Collections$2.forEachRemaining(Collections.java:4755)
        at java.base/java.util.stream.AbstractPipeline.copyInto(AbstractPipeline.java:484)
        at java.base/java.util.stream.AbstractPipeline.wrapAndCopyInto(AbstractPipeline.java:474
        at java.base/java.util.stream.ReduceOps$ReduceOp.evaluateSequential(ReduceOps.java:913)
        at java.base/java.util.stream.AbstractPipeline.evaluate(AbstractPipeline.java:234)
        at java.base/java.util.stream.ReferencePipeline.collect(ReferencePipeline.java:578)
        at io.prestosql.sql.planner.plan.SimplePlanRewriter$RewriteContext.defaultRewrite(Simple
        at io.prestosql.sql.planner.plan.SimplePlanRewriter.visitPlan(SimplePlanRewriter.java:38
        at io.prestosql.sql.planner.plan.SimplePlanRewriter.visitPlan(SimplePlanRewriter.java:22
        at io.prestosql.sql.planner.plan.PlanVisitor.visitOutput(PlanVisitor.java:49)
        at io.prestosql.sql.planner.plan.OutputNode.accept(OutputNode.java:83)
        at io.prestosql.sql.planner.plan.SimplePlanRewriter.rewriteWith(SimplePlanRewriter.java:
        at io.prestosql.sql.planner.optimizations.BeginTableWrite.optimize(BeginTableWrite.java:
        at io.prestosql.sql.planner.LogicalPlanner.plan(LogicalPlanner.java:206)
        at io.prestosql.sql.planner.LogicalPlanner.plan(LogicalPlanner.java:195)
        at io.prestosql.sql.planner.LogicalPlanner.plan(LogicalPlanner.java:190)
        at io.prestosql.execution.SqlQueryExecution.doPlanQuery(SqlQueryExecution.java:450)
        at io.prestosql.execution.SqlQueryExecution.planQuery(SqlQueryExecution.java:430)
        at io.prestosql.execution.SqlQueryExecution.start(SqlQueryExecution.java:382)
        at io.prestosql.execution.SqlQueryManager.createQuery(SqlQueryManager.java:237)
        at io.prestosql.dispatcher.LocalDispatchQuery.lambda$startExecution$7(LocalDispatchQuery
        at io.prestosql.$gen.Presto_350____20210903_063152_2.run(Unknown Source)
        at java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1
        at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:
        at java.base/java.lang.Thread.run(Thread.java:834)
```

这种报错原因是原先有该方法，但是加载到的类中却没有。出现这种情况主要有两种：

由于版本迭代，新版本包新增了该方法，而旧版本包中没有。

也有可能是加载其他包中的同名类。

在/usr/local/service下执行命令：



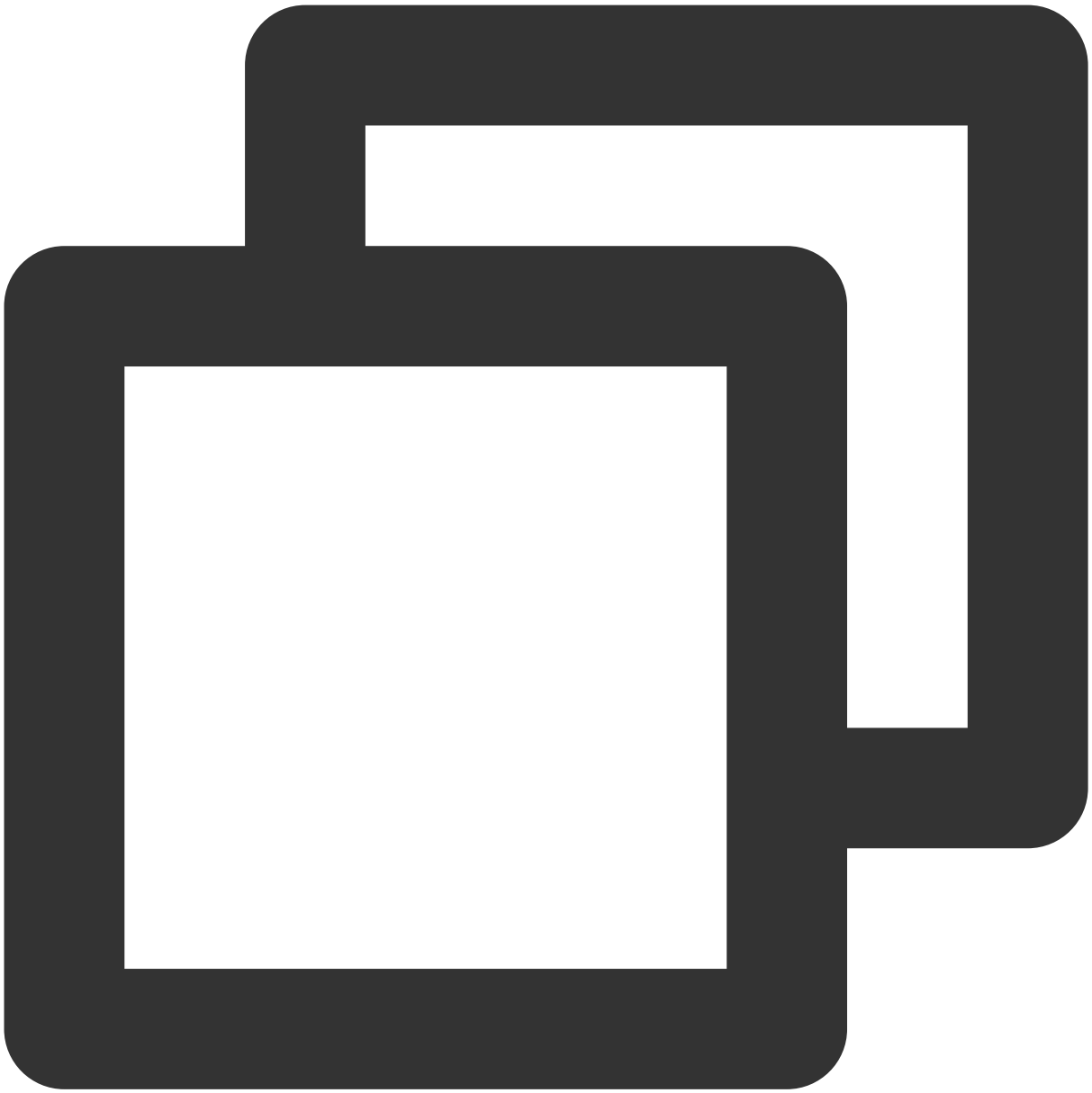
```
find . -name "*.jar" -exec grep -Hls "org/apache/hadoop/fs/cosn/ranger/client/Range
```

找到相关包删除即可。如果是其他类，修改上述命令的类路径即可。

报错 java.lang.ClassCastException org.apache.hadoop.fs.cosn.ranger.protocol.ClientQCloudObjectStorageProtocolProtos$GetSTSRequest cannot be cast to com.google.protobuf.Message，该如何处理？

类似这种错误，一般是包污染问题，机器上存在旧版本包，因 protobuf 协议不一致导致的，和下面 alluxio 包污染问题基本一样。

在/usr/local/service下执行命令：



```
find . -name "*.jar" -exec grep -Hls "org/apache/hadoop/fs/cosn/ranger/protocol/Cli
```

找到相关包删除即可。如果是其他类，修改上述命令的类路径即可。

修改 Ranger policy 未生效，该如何处理？

如果是 chdfs，修改 COS Ranger Server 配置文件 ranger-chdfs-security.xml 中的配置项：ranger.plugin.chdfs.policy.pollIntervalMs 调小（单位毫秒）。

如果是 cosn，修改 COS Ranger Server 配置文件 ranger-cos-security.xml 中的配置项：ranger.plugin.cos.policy.pollIntervalMs 调小（单位毫秒）。

Ranger policy 策略配置 group 未生效，该如何处理？

如果配置 user 后生效了，需要找 EMR 团队确认 group 同步问题；其他情况请联系我们。

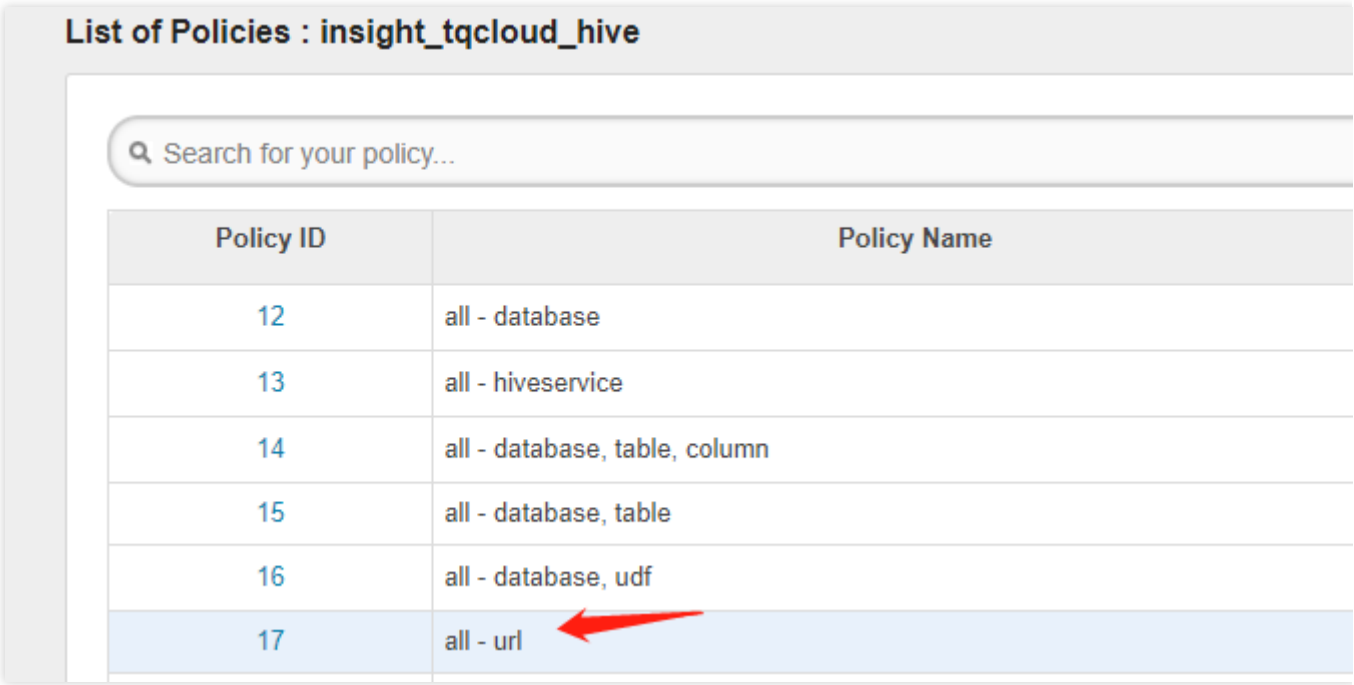
Ranger policy 配置存储路径策略规则，该如何处理？

Ranger 对 Path 校验规则其实很简单，主要就是字符串匹配。如果有文件 /a/b/c，配置 policy 的 path 规则为 **/a/**，访问 /**a** 或 /**a**/ 都是无法访问的；因为 sdk 会把访问路径末尾 / 给去掉，最后到 Ranger 那边路径就变成了 /**a**，无法匹配上 /**a**/ ；如果访问 /**a**/**b** 或者 /**a**/**b**/**c**，这两个 path 的前缀部分是刚好可以匹配上 policy 中的 path 规则 **/a/**。

Hive 指定 COSN 或 OFS 路径建表报 HiveAccessControlException，该如何处理？

```
0: jdbc:hive2://t-qcbj5-hadoop-hive-001.test-> CREATE external TABLE IF NOT EXISTS test ( eid int, name String)
. . . . . . . . . . . . . . . . . . . . . . .> ROW FORMAT DELIMITED
. . . . . . . . . . . . . . . . . . . . . . .> FIELDS TERMINATED BY ','
. . . . . . . . . . . . . . . . . . . . . . .> LOCATION 'cosn://com-nio-insight-test-1253431691/tianbo/test';
Error: Error while compiling statement: FAILED: HiveAccessControlException Permission denied: user [hadoop] does not have [READ] privilege on [cosn://co
```

需要 hive 放开对 URL 的校验，需要在 ranger 控制台 hive 里配置允许 url 权限：



注意

注意看一下报错日志的格式，这种报错多半是 Ranger 服务报出来的，大多数情况下是 ranger admin 配置权限有误。

kerberos 下 spark 提交任务报 HiveAccessControlException，该如何处理？

应用程序需要与其他安全 Hadoop 文件系统交互，则需要在启动时将其 URI 显式提供给 Spark，配置参数 spark.kerberos.access.hadoopFileSystems=cosn://bucket-appid,ofs://f4mxxxxxxxx-Xxxx.chdfs.ap-guangzhou.myqcloud.com 可参考 Spark 官网文档。

SPARK 删表不会进回收站，该如何处理？

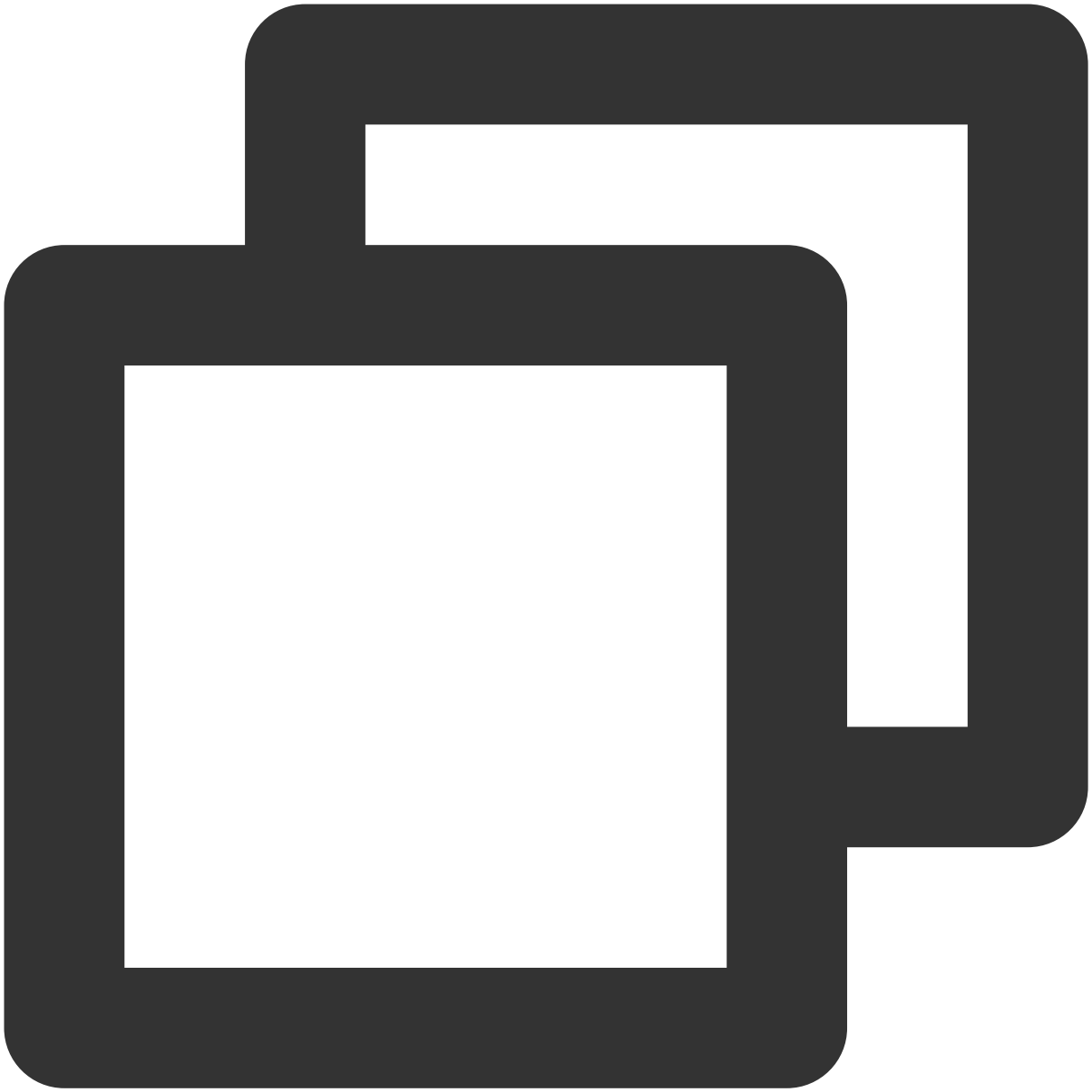
Spark 中 create table 指定 Location 等价于创建外部表，删除外部表无法删除数据，可参考 Spark 官网文档说明。

说明

hive on mr 情况下 drop table 可删除。

Hive 执行 INSERT 语句报 AccessControlException，该如何处理？

Hive 默认引擎是 MapReduce，yarn-site.xml 文件新增配置项：

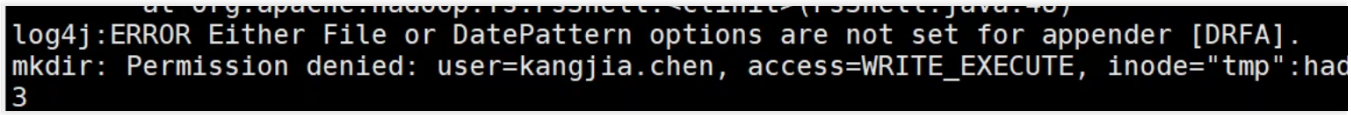


```
<property>
    <name>mapreduce.job.hdfs-servers</name>
    <value>
        ofs://f4mxxxxxxx-XXXX,cosn://bucketname-appid,${fs.defaultFS}
    </value>
</property>
```

如果 hive 引擎是 tez，则在 tez-site.xml 文件中新增配置项 tez.job.fs-servers，value 值同上。

如果是 beeline 连 hive，需要重启 hiveserver2 加载新的 yarn-site 配置。

访问 OFS 报错，该如何处理？



报错是 ofs 后端返回的，启用 ranger 后，需要关闭 posix。操作如下：

CHDFS 控制台

COS bucket 配置

YARN 命令行提交任务 报 renew token failed，该如何处理？

yarn 命令行执行时，需要 -Dmapreduce.job.send-token-conf 参数。

如何自建 cosranger？

可参见 COS Ranger 权限体系解决方案 、CHDFS Ranger 权限体系解决方案。

腾讯云 EMR 中如何启用 Ranger？

在 emr 控制台购买 Ranger 和 cosranger 组件，省去自己部署麻烦。

如果是 chdfs，在 core-site.xml 中新增配置项：fs.ofs.ranger.enable.flag，设置为：true。

如果是 cosn，在 core-site.xml 中新增配置项：fs.cosn.credentials.provider，设置为：

org.apache.hadoop.fs.auth.RangerCredentialsProvider。

NodeCache 空指针异常，该如何处理？

确认 hadoop-ranger-client 版本，如果是 v3.8，建议升级到 v5.0；其他情况请联系我们。

出现这种报错原因是大数据作业时并发程度比较高，zookeeper 压力比较高，zookeeper watch 有丢失导致的。

启用 cosranger 后 hadoop fs 命令报 java.lang.IllegalArgumentException: Failed to specify server's Kerberos principal name，该如何处理？

core-site.xml 新增配置项：qcloud.object.storage.kerberos.principal。

如果是 hdfs 集群报该错，core-site.xml 新增配置项：dfs.namenode.kerberos.principal。

版权所有：腾讯云计算（北京）有限责任公司

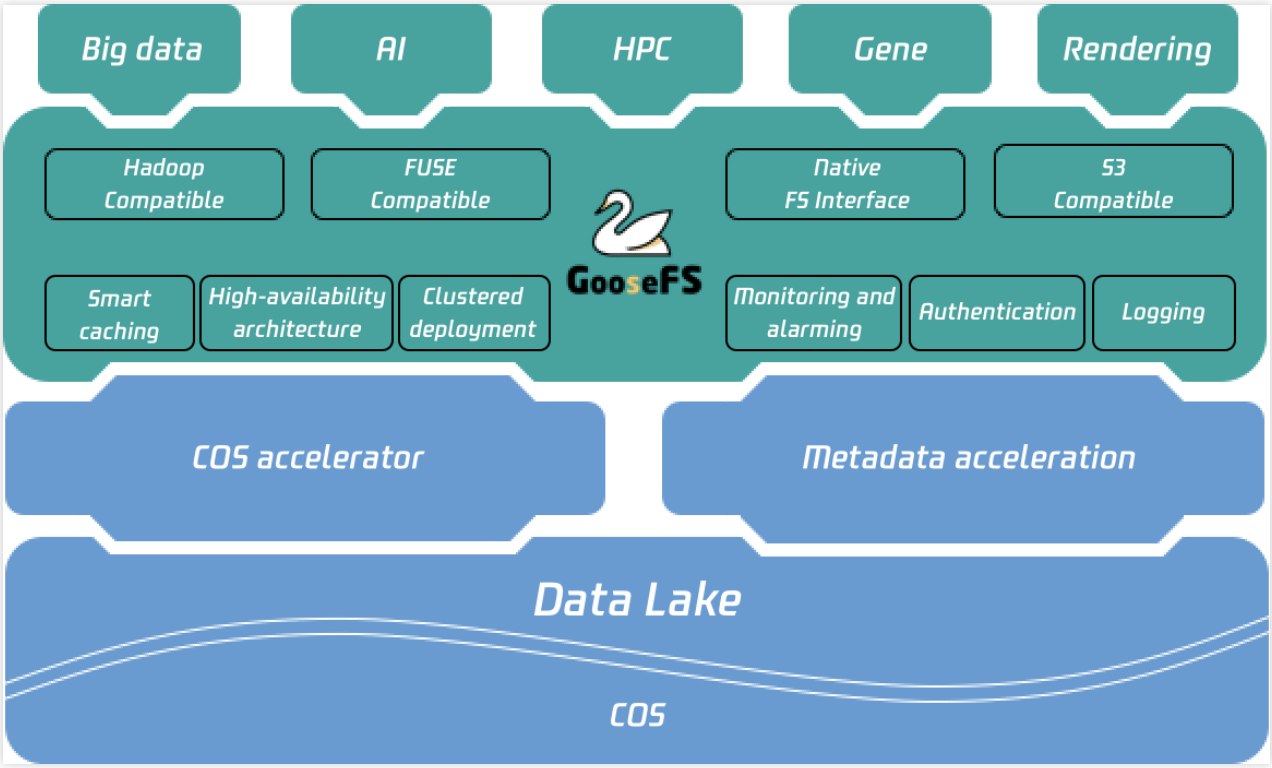
数据加速器 GooseFS
产品概述

最近更新时间：2024-01-06 11:17:59

数据湖加速器（Data Lake Accelerator Goose FileSystem，GooseFS），是由腾讯云推出的高可靠、高可用、弹性的数据湖加速服务。依靠对象存储（Cloud Object Storage，COS）作为数据湖存储底座的成本优势，为数据湖生态中的计算应用提供统一的数据湖入口，加速海量数据分析、机器学习、人工智能等业务访问存储的性能；采用了分布式集群架构，具备弹性、高可靠、高可用等特性，为上层计算应用提供统一的命名空间和访问协议，方便用户在不同的存储系统管理和流转数据。

产品功能

GooseFS 旨在提供一站式的缓存解决方案，在利用数据本地性和高速缓存，统一存储访问语义等方面具有天然的优势；GooseFS 在腾讯云数据湖生态中扮演着“上承计算，下启存储”的核心角色，如下图所示。



GooseFS 提供了以下功能：

1. 缓存加速和数据本地化（Locality）：GooseFS 可以与计算节点混合部署提高数据本地性，利用高速缓存功能解决存储性能问题，提高写入 COS 的带宽。

2. 融合存储语义：GooseFS 提供 UFS（Unified FileSystem）的语义，可以支持 COS、Hadoop、S3、K8S CSI、FUSE 等多个存储语义，使用于多种生态和应用场景。
3. 统一的腾讯云相关生态服务：包括日志、鉴权、监控，实现了与 COS 操作统一。
4. 提供 Namespace 管理能力，针对不同业务、不同的Under File System，提供不同的读写缓存策略以及生命周期（TTL）管理。
5. 感知 Table 元数据功能：对于大数据场景下数据 Table，提供 GooseFS Catalog 用于感知元数据 Table ，提供 Table 级别的 Cache 预热。

产品优势

GooseFS 在数据湖场景中具有如下几点明显的优势：

数据 I/O 性能

GooseFS 部署提供近计算端的分布式共享缓存，上层计算应用可以透明地、高效地从远端存储将需要频繁访问的热数据缓存到近计算端，加速数据 I/O 性能。GooseFS 提供了元数据缓存功能，可以加速大数据场景下查询文件数据以及列出文件列表等元数据操作的性能。配合大数据存储桶使用，还可进一步加速重命名文件的操作性能。此外，业务可以按需选择 MEM、SSD、NVME 以及 HDD 盘等不同的存储介质，平衡业务成本和数据访问性能。

存储一体化

GooseFS 提供了统一的命名空间，不仅支持了对象存储 COS 存储语义，也支持 HDFS、K8S CSI 以及 FUSE 等语义，为上层业务提供了一体化的融合存储方案，简化业务侧运维配置。存储一体化能够打通不同数据底座的壁垒，方便上层应用管理和流转数据，提升数据利用的效率。

生态亲和性

GooseFS 全兼容腾讯云大数据平台框架，也支持客户侧自定义的本地部署，具备优秀的生态亲和性。业务侧不仅可以在腾讯云弹性 MapReduce 产品中使用 GooseFS 加速大数据业务，也可以便捷地将 GooseFS 本地化部署在公有云 CVM 或者自建 IDC 内。此外，GooseFS 支持透明加速能力，对于已经使用腾讯云 COSN 和 CHDFS 的用户，只需做简单的配置修改，即可实现不修改任何业务代码和访问路径的前提下，自动使用GooseFS 加速 COSN 和 CHDFS 的业务访问。

版权所有：腾讯云计算（北京）有限责任公司

更新日志

最近更新时间：2024-06-13 09:43:34

GooseFS 版本更新列表如下，如您有任何疑问或建议，欢迎 联系我们。

版本号	更新日期	更新说明	下载链接
1.4.1	2023年3月1日	功能优化 支持清理和查看未完成文件列表 优化递归加载元数据（loadmetadata -R）操作加锁粒度 问题修复 修复 Flume 写入场景认证流和数据流状态统一问题 修复大文件写入耗尽客户端资源后死锁的问题	GooseFS ：点击下载 GooseFS（KonaJDK 版本）：点击下载 GooseFS 客户端：点击下载 GooseFS 客户端（KonaJDK 版本）：点击下载 GooseFS FUSE 客户端：点击下载 GooseFS FUSE 客户端（KonaJDK 版本）：点击下载
1.4.0	2022年11月11日	新增功能 提供文件解压缩功能（公测，仅上海自动驾驶专区可用） 支持临时密钥托管服务 distributedLoad 支持层级遍历 GooseFS-FUSE 支持降级读 功能优化 GooseFS distributedLoad 能力支持层级遍历能力，支持递归拉取指定目录下的元数据信息 FUSE 读性能优化，提升大文件顺序读性能 增加 Master 查询/更新 RocksDB 的分位耗时监控，提升元数据服务的监控灵敏度 优化了 GooseFS HA 模式下的集群恢复时间，提升了集群可用性 CosN 依赖版本升级，支持通过原生 HDFS 协议访问已开启元数据加速的存储桶，提升大数据场景下的文件操作性能 GooseFS 配置精简优化，减少了不必要的配置项，提升了配置易用性 listInfo 精简优化，提升文件 list 性能 问题修复 修复了 Worker 接收大量无效 async block 请求的问题	GooseFS ：点击下载 GooseFS（KonaJDK 版本）：点击下载 GooseFS 客户端：点击下载 GooseFS 客户端（KonaJDK 版本）：点击下载 GooseFS FUSE 客户端：点击下载 GooseFS FUSE 客户端（KonaJDK 版本）：点击下载

版权所有：腾讯云计算（北京）有限责任公司

		优化了 Worker 上报时对孤立 block 的处理逻辑	
1.3.0	2022年7月25日	新增功能 支持 Kerberos 认证。 支持访问开启 元数据加速能力 的存储桶，以原生 POSIX 语义访问对象存储服务。 Master 节点缓存淘汰策略支持 LRU 算法，避免缓存频繁置换。 Master 节点支持过期元数据清理能力。 功能优化 GooseFS Master 节点优化了锁瓶颈问题，显著提升 Master QPS，带来 35% 左右的性能优化效果。 GooseFS Worker 节点支持并发 Format，提升操作性能。 GooseFS Fuse 客户端支持覆盖写操作。 GooseFS Fuse 客户端优化了 ls 命令的内存占用问题。 GooseFS HDFS 客户端优化 ListNamespace 的性能。 Bug 修复 修复了 RocksDB 泄漏和 Core 的问题，避免内存泄露。 修复了 Zookeeper Curator 误打日志的问题。 修复了 UFS 读带宽不准的问题。 修复了 DistributedLoad 的时候由于日志打印过多导致的 LostWorker问题。	GooseFS ：点击下载 GooseFS（KonaJDK 版本）：点击下载 GooseFS 客户端：点击下载 GooseFS 客户端（KonaJDK 版本）：点击下载 GooseFS FUSE 客户端：点击下载 GooseFS FUSE 客户端（KonaJDK 版本）：点击下载
1.2.0	2022年1月25日	新增功能 GooseFS 支持将日志上报云日志系统。 Inode 和 Block 元信息支持单独配置 RocksDB。 GooseFS Client 支持热开关能力，提升故障容灾措施。 添加了部分全链路日志的基础设施。 功能优化 优化 Raft 切主延迟高的问题。 优化了 GooseFS HCFS Client 的内存占用。 Bug 修复 修复 Journal 乱序的问题。 Ratis 死锁导致的 GRPC 问题。 修复了 HDFSUnderFileSystemFactory 加载位置不正确的问题。 修复了 log4j2 的安全漏洞问题。 修复了 ufsPath 前缀检查错误的问题。	GooseFS：点击下载 GooseFS（KonaJDK 版本）：点击下载 GooseFS FUSE 客户端：点击下载
1.1.0	2021年9月	新增功能：	当前版本不再维护，请下载

	1日	支持 ranger 鉴权。 支持并发 list。 支持 distributedLoad 目录原子性。 支持 namespace 的缓存白名单。 支持 ofs scheme 透明加速。 功能优化： cli 为各个 subcmd 增加 help 命令。 优化 table 挂载检测 namespace 是否存在。 完善 job worker/worker metrics 监控。 Bug 修复： 修复 distributedLoad 读膨胀问题。	最新版本
1.0.0	2021年6月1日	基于 namespace 的读写策略与 TTL 管理。 基于 Hive Table 维度以及 Table Partition 维度的预热。 兼容存量 COSN 用户的路径兼容，实现透明加速。 Fluid 集成 GooseFS。 集成了 CHDFS 支持。	当前版本不再维护，请下载最新版本

快速入门

最近更新时间：2024-01-06 11:17:58

本文档主要提供 GooseFS 快速部署、调试的相关指引，提供在本地机器上部署 GooseFS，并将对象存储（Cloud Object Storage，COS）作为远端存储的步骤指引，具体步骤如下：

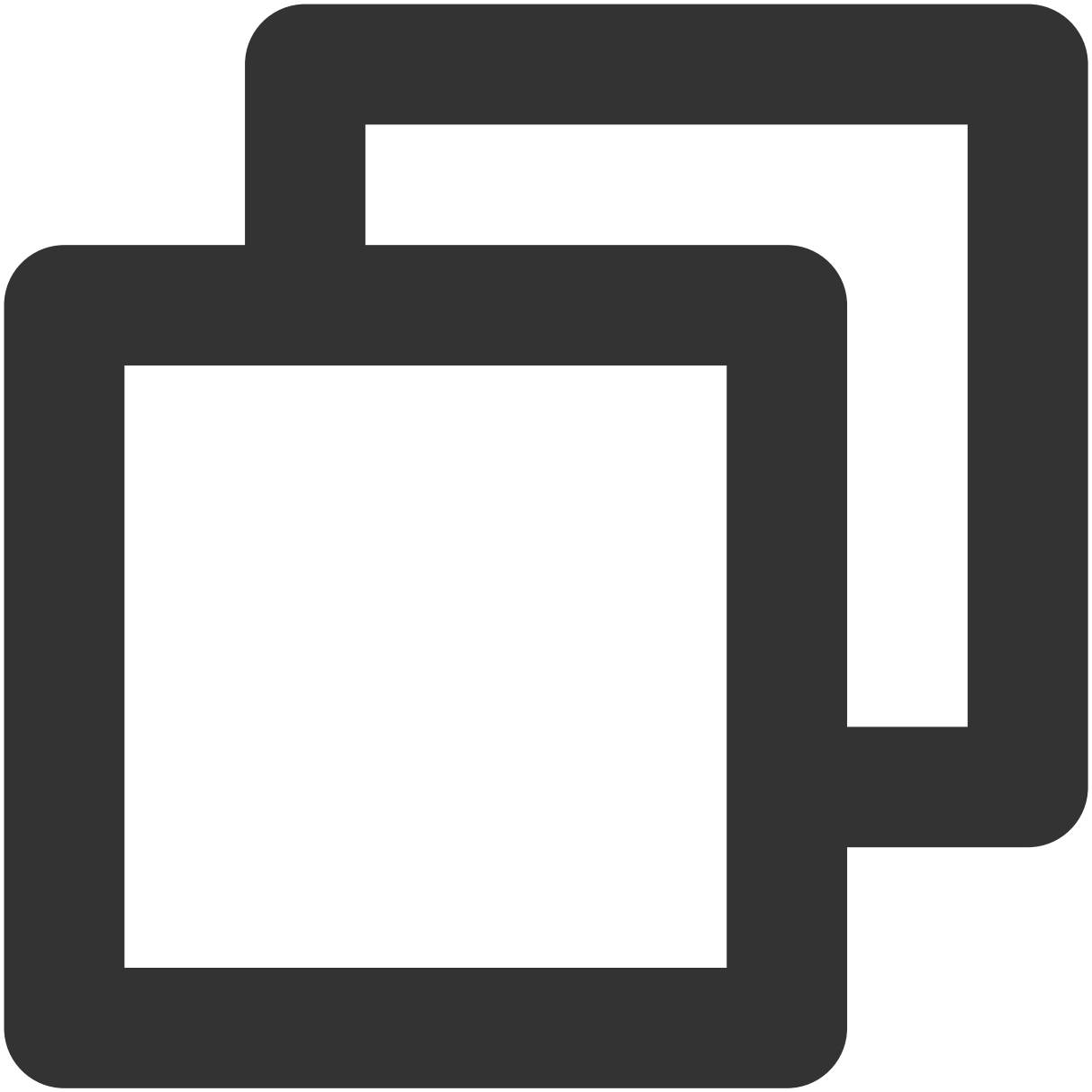
前提条件

在使用 GooseFS 之前，您还需要准备以下工作：
1. 在 COS 服务上创建一个存储桶以作为远端存储，操作指引请参见 控制台快速入门。
2. 安装 Java 8 或者更高的版本。
3. 安装 SSH，确保能通过 SSH 连接到 LocalHost，并远程登录。
4. 在 CVM 服务上购买一台实例，操作指引详见 购买云服务器，并确保磁盘已经挂载到实例上。

下载并配置 GooseFS

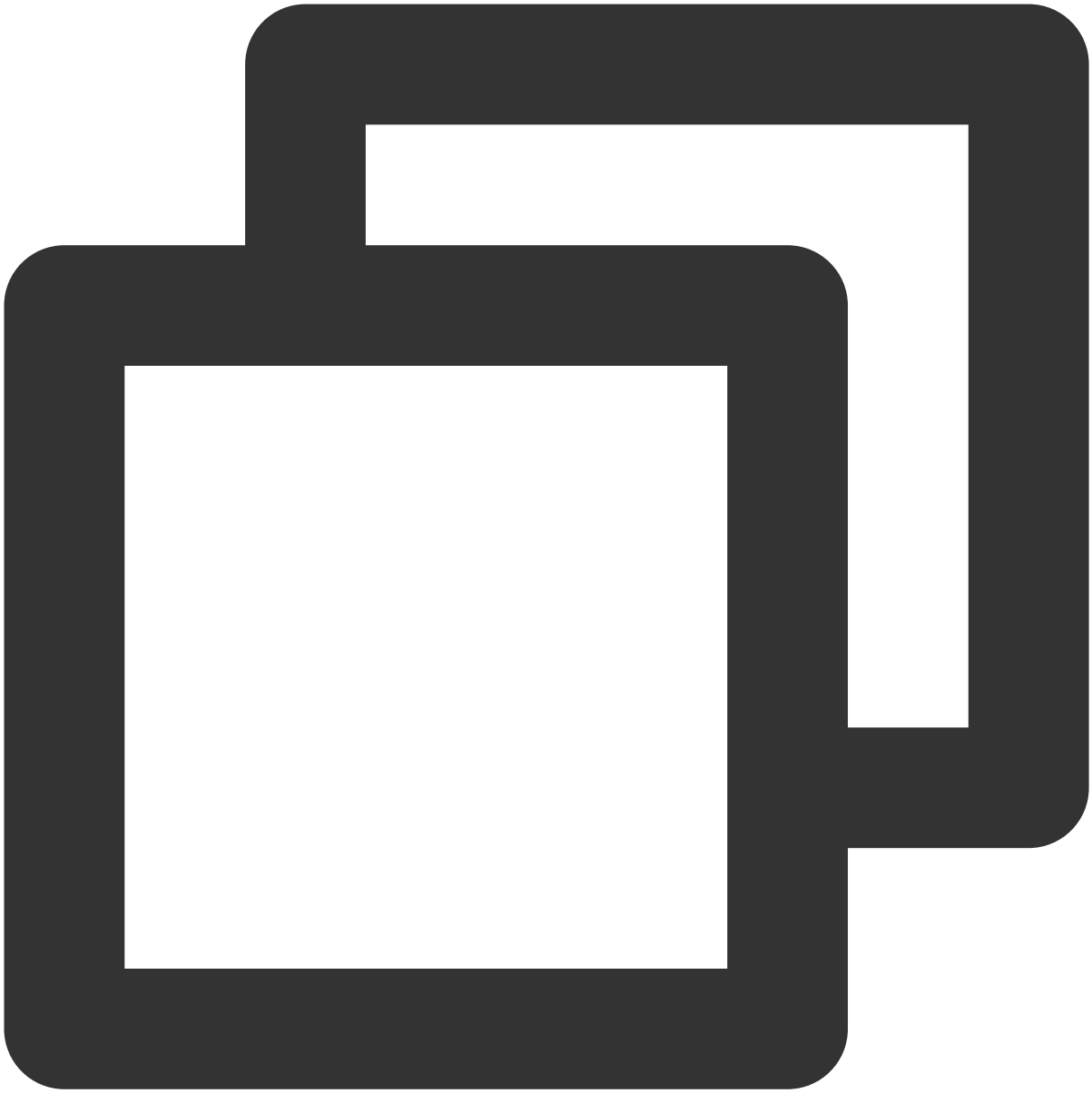
1. 新建本地目录并进入该目录下(您也可以按需选择其他目录)，从官方仓库下载 GooseFS 安装包到本地。安装包 Github 地址：goosefs-1.4.2-bin.tar.gz。

版权所有：腾讯云计算（北京）有限责任公司



```
$ cd /usr/local
$ mkdir /service
$ cd /service
$ wget https://downloads.tencentgoosefs.cn/goosefs/1.4.2/release/goosefs-1.4.2-bin.
```

2. 执行如下命令，对安装包进行解压，解压后进入安装包目录下。



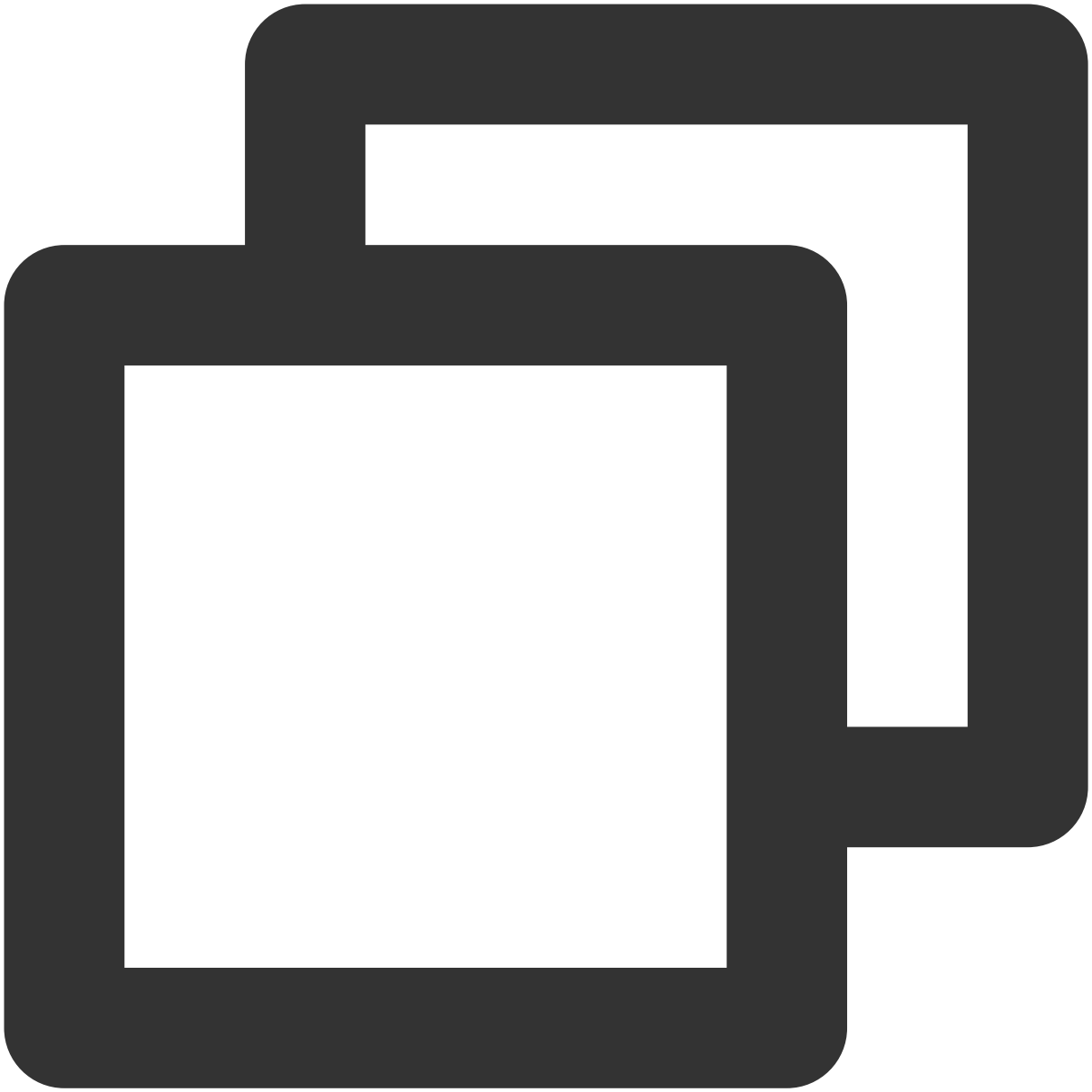
```
$ tar -zxvf goosefs-1.4.2-bin.tar.gz
$ cd goosefs-1.4.2
```

解压后，得到 goosefs-1.4.2，即 GooseFS 的主目录。下文将以 `${GOOSEFS_HOME}` 代指该目录的绝对路径。

3. 在 `${GOOSEFS_HOME}/conf` 的目录下，创建 `conf/goosefs-site.properties` 的配置文件。

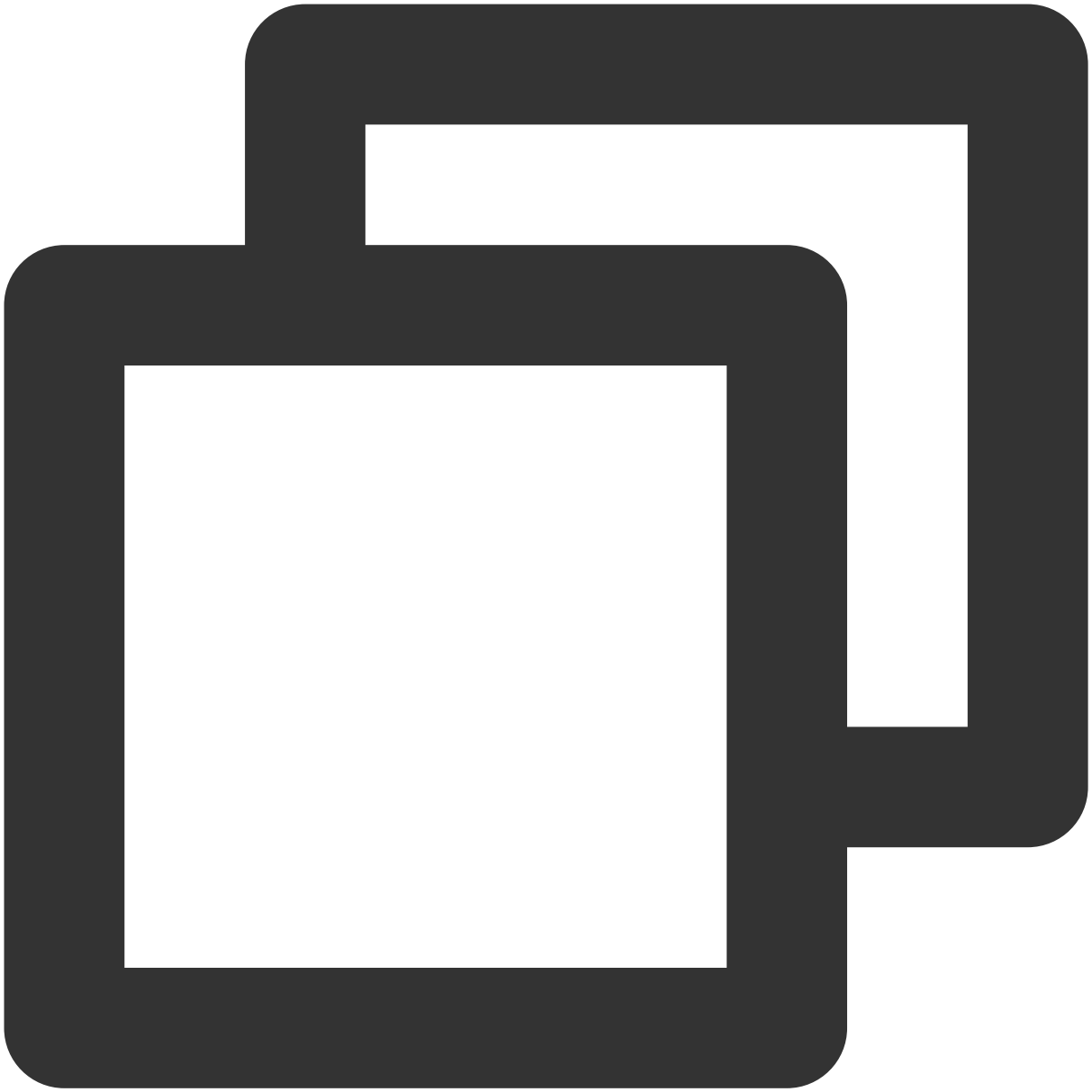
GooseFS 提供 AI 和大数据两个场景的配置模板，可以使用任意上述内置模板，然后进入编辑模式修改配置：

（1）使用 AI 场景模板，更多信息可参考GooseFS AI 场景生产环境配置实践。



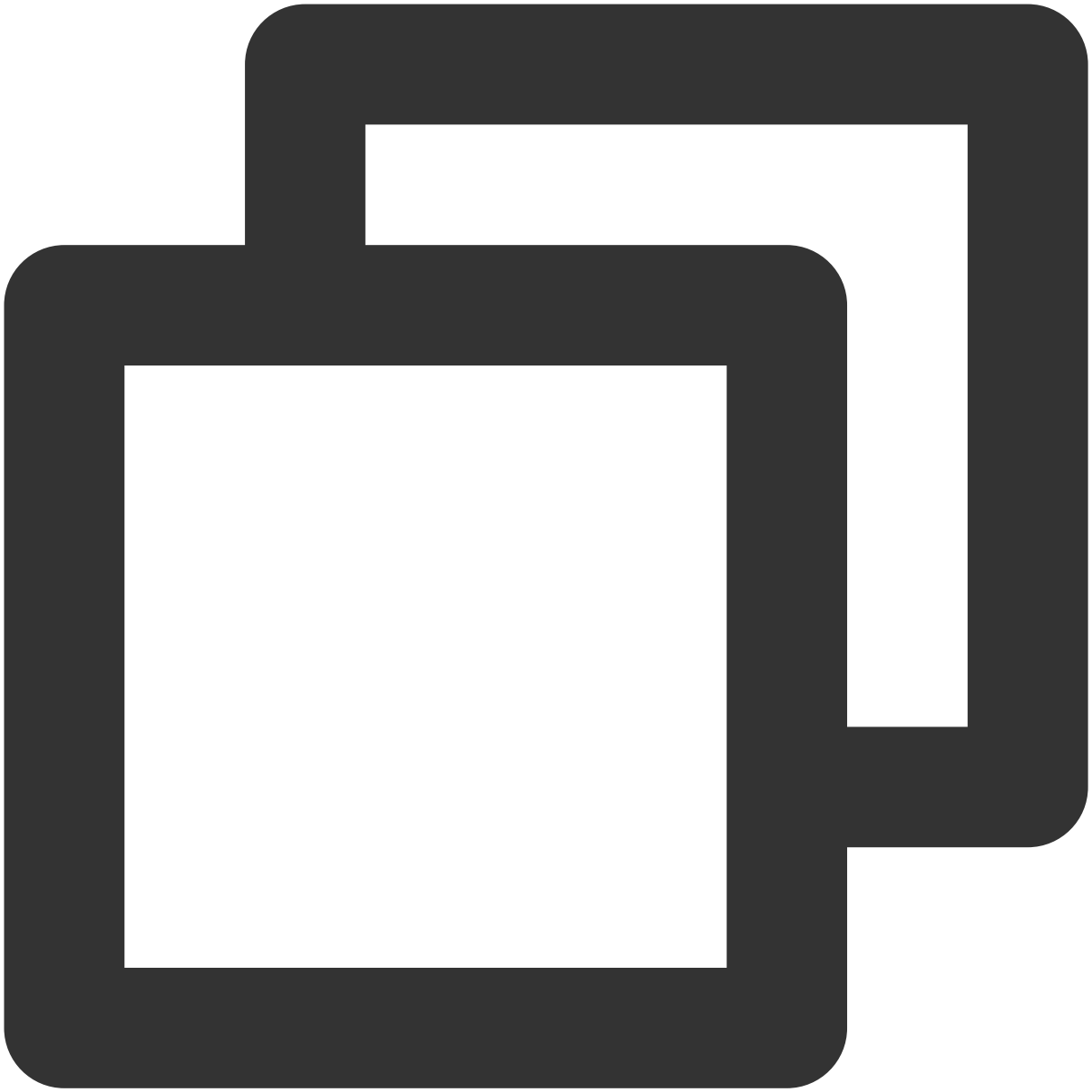
```
$ cp conf/goosefs-site.properties.ai_template conf/goosefs-site.properties
$ vim conf/goosefs-site.properties
```

（2）使用大数据场景模板，更多信息可参考GooseFS 大数据场景生产环境配置实践。



```
$ cp conf/goosefs-site.properties.bigdata_template conf/goosefs-site.properties
$ vim conf/goosefs-site.properties
```

4. 在配置文件 `conf/goosefs-site.properties` 中，调整如下配置项：



```
# Common properties
# 调整Master节点host信息
goosefs.master.hostname=localhost
goosefs.master.mount.table.root.ufs=${goosefs.work.dir}/underFSStorage

# Security properties
# 调整权限配置
goosefs.security.authorization.permission.enabled=true
goosefs.security.authentication.type=SIMPLE

# Worker properties
```

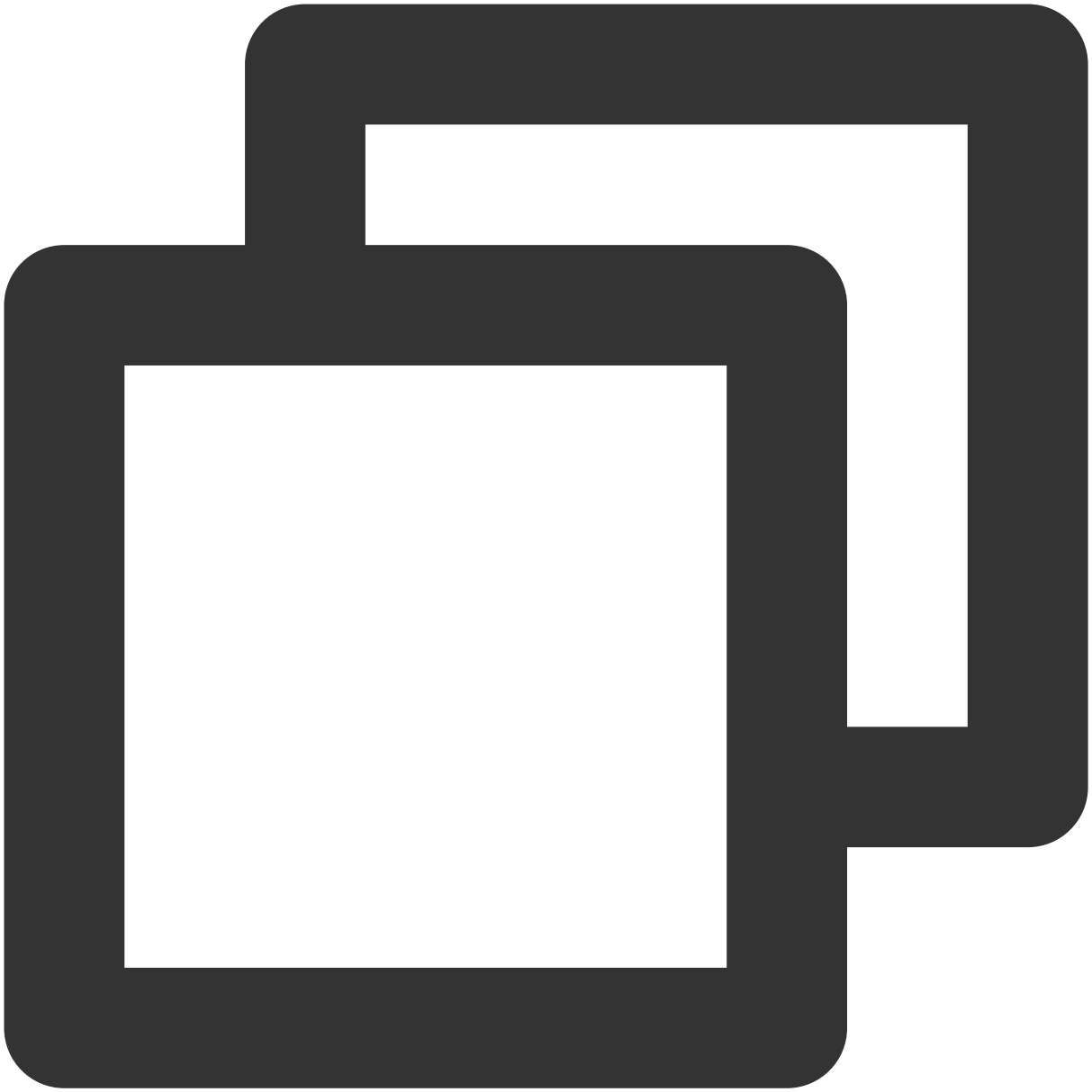
```
# 调整worker节点配置，指定本地缓存介质、缓存路径和缓存容量大小
goosefs.worker.ramdisk.size=1GB
goosefs.worker.tieredstore.levels=1
goosefs.worker.tieredstore.level0.alias=SSD
goosefs.worker.tieredstore.level0.dirs.path=/data
goosefs.worker.tieredstore.level0.dirs.quota=80G


# User properties
# 指定文件读写缓存策略
goosefs.user.file.readtype.default=CACHE
goosefs.user.file.writetype.default=MUST_CACHE
```

注意

配置 `goosefs.worker.tieredstore.level0.dirs.path` 该路径参数前，需要先新建这一路径。

启用 GooseFS

1. 启用 GooseFS 前，需要进入到 GooseFS 目录下执行启动指令：



```
$ cd /usr/local/service/goosefs-1.4.2
$ ./bin/goosefs-start.sh all
```

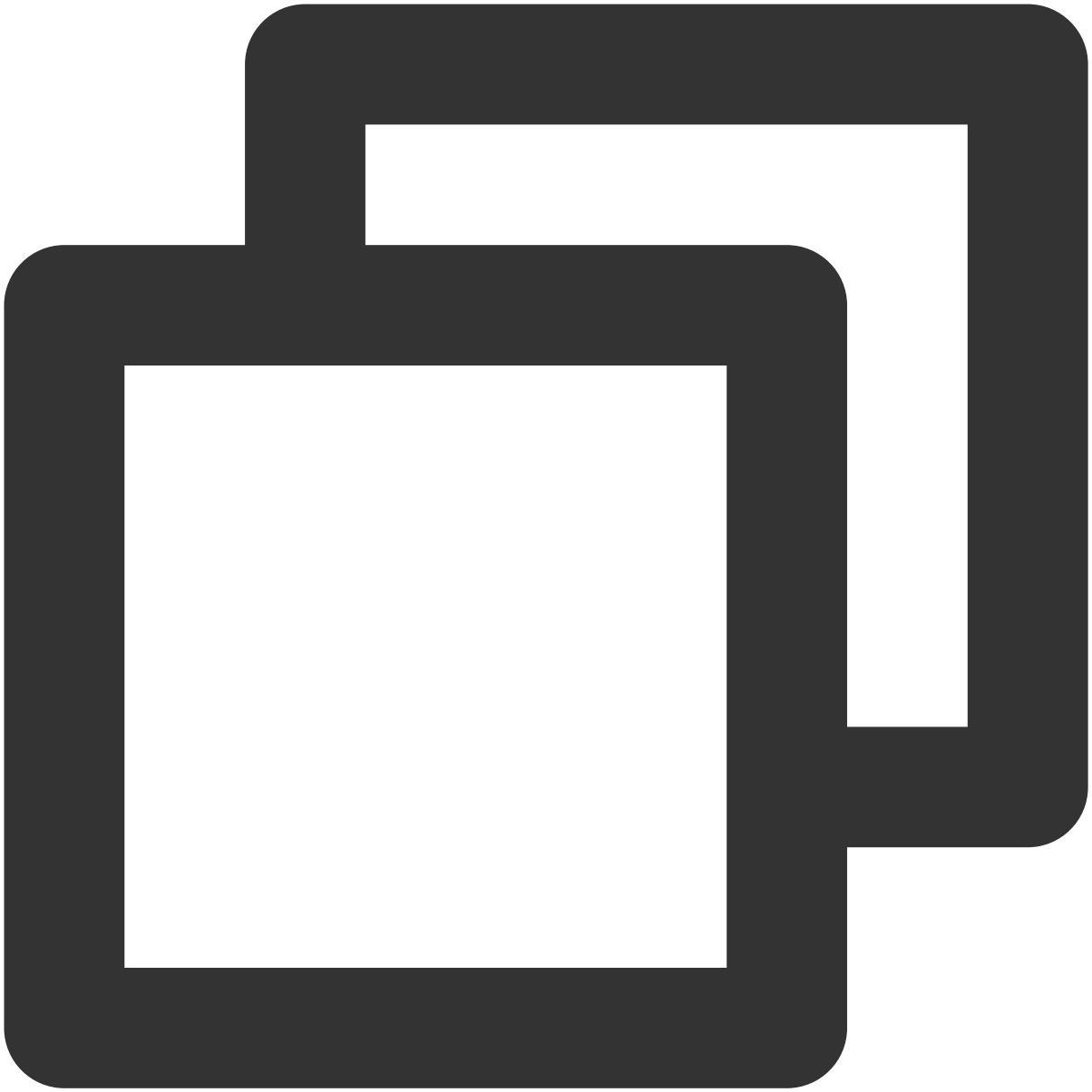
执行该指令后，可以看到如下页面：

```
Executing the following command on all worker nodes and logging to /usr/local/service/goosefs
rt.sh -a proxy
Waiting for tasks to finish...
All tasks finished
----------------------------------------
Starting to monitor all remote services.
----------------------------------------
--- [ OK ] The master service @ VM-0-5-centos is in a healthy state.
--- [ OK ] The job_master service @ VM-0-5-centos is in a healthy state.
--- [ OK ] The worker service @ VM-0-5-centos is in a healthy state.
--- [ OK ] The job_worker service @ VM-0-5-centos is in a healthy state.
--- [ OK ] The proxy service @ VM-0-5-centos is in a healthy state.
[root@VM-0-5-centos goosefs-1.3.0]#
```

该命令执行完毕后，可以访问 `http://localhost:9201` 和 `http://localhost:9204`，分别查看 Master 和 Worker 的运行状态。

使用 GooseFS 挂载 COS（COSN） 或腾讯云 HDFS（CHDFS）

如果 GooseFS 需要挂载 COS（COSN）或腾讯云 HDFS（CHDFS）到 GooseFS 的根路径上，则需要先在 `conf/core-site.xml` 配置中指定 COSN 或 CHDFS 的必需配置项，其中包括但不限于：`fs.cosn.impl`、`fs.AbstractFileSystem.cosn.impl` 以及 `fs.cosn.userinfo.secretId` 和 `fs.cosn.userinfo.secretKey` 等，如下所示：



```
<!-- COSN related configurations -->
<property>
  <name>fs.cosn.impl</name>
  <value>org.apache.hadoop.fs.CosFileSystem</value>
</property>

<property>
   <name>fs.AbstractFileSystem.cosn.impl</name>
   <value>com.qcloud.cos.goosefs.CosN</value>
```

```
</property>


<property>
    <name>fs.cosn.userinfo.secretId</name>
    <value></value>
</property>


<property>
    <name>fs.cosn.userinfo.secretKey</name>
    <value></value>
</property>


<property>
    <name>fs.cosn.bucket.region</name>
    <value></value>
</property>


<!-- CHDFS related configurations -->
<property>
   <name>fs.AbstractFileSystem.ofs.impl</name>
   <value>com.qcloud.chdfs.fs.CHDFSDelegateFSAdapter</value>
</property>


<property>
   <name>fs.ofs.impl</name>
   <value>com.qcloud.chdfs.fs.CHDFSHadoopFileSystemAdapter</value>
</property>


<property>
   <name>fs.ofs.tmp.cache.dir</name>
   <value>/data/chdfs_tmp_cache</value>
</property>


<!--appId-->
<property>
   <name>fs.ofs.user.appid</name>
   <value>1250000000</value>
</property>
```

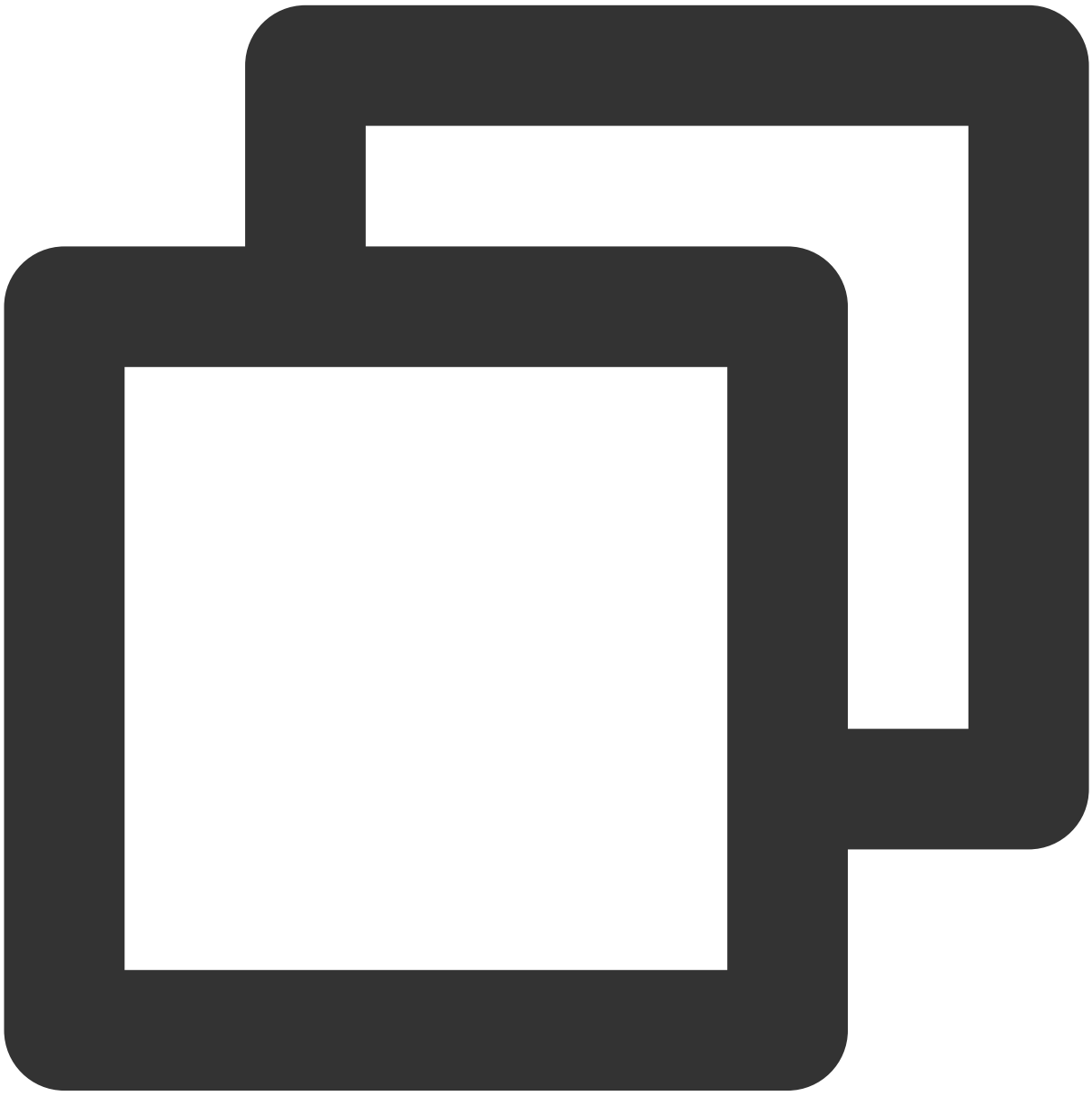
说明

COSN 的完整配置可参考：Hadoop 工具。

CHDFS 的完整配置可参考：挂载 CHDFS。

下面将介绍一下如何通过创建 Namespace 来挂载 COS 或 CHDFS 的方法和步骤。

1. 创建一个命名空间 namespace 并挂载 COS：



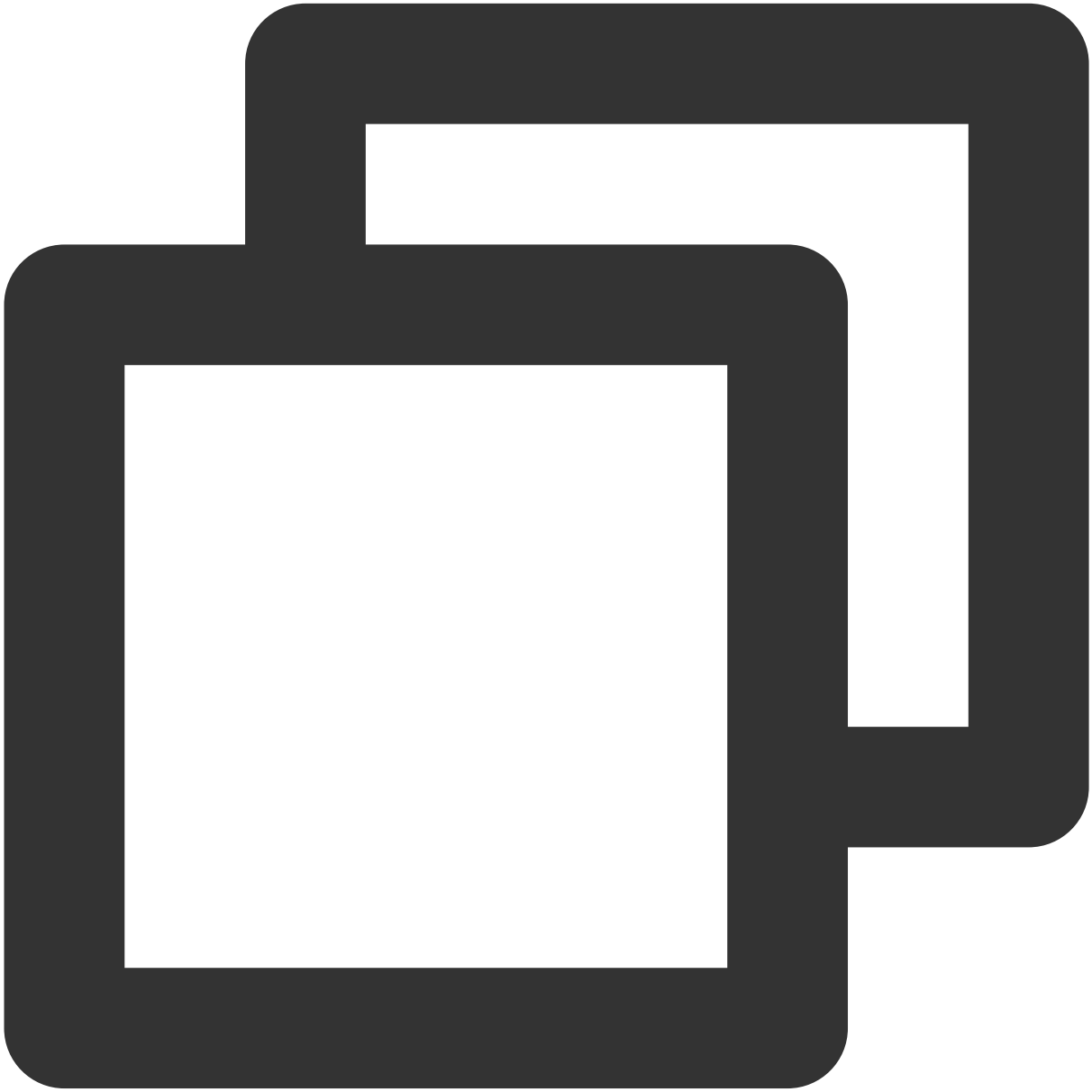
```
$ goosefs ns create myNamespace cosn://bucketName-1250000000/ \\
--secret fs.cosn.userinfo.secretId=AKXXXXXXXXXXX \\
--secret fs.cosn.userinfo.secretKey=XXXXXXXXXXXX \\
--attribute fs.cosn.bucket.region=ap-xxx \\
```

注意

创建挂载 COSN 的 namespace 时，必须使用 `--secret` 参数指定访问密钥，并且使用 `--attribute` 指定 Hadoop-COS（COSN）所有必选参数，具体的必选参数可参考 Hadoop 工具。

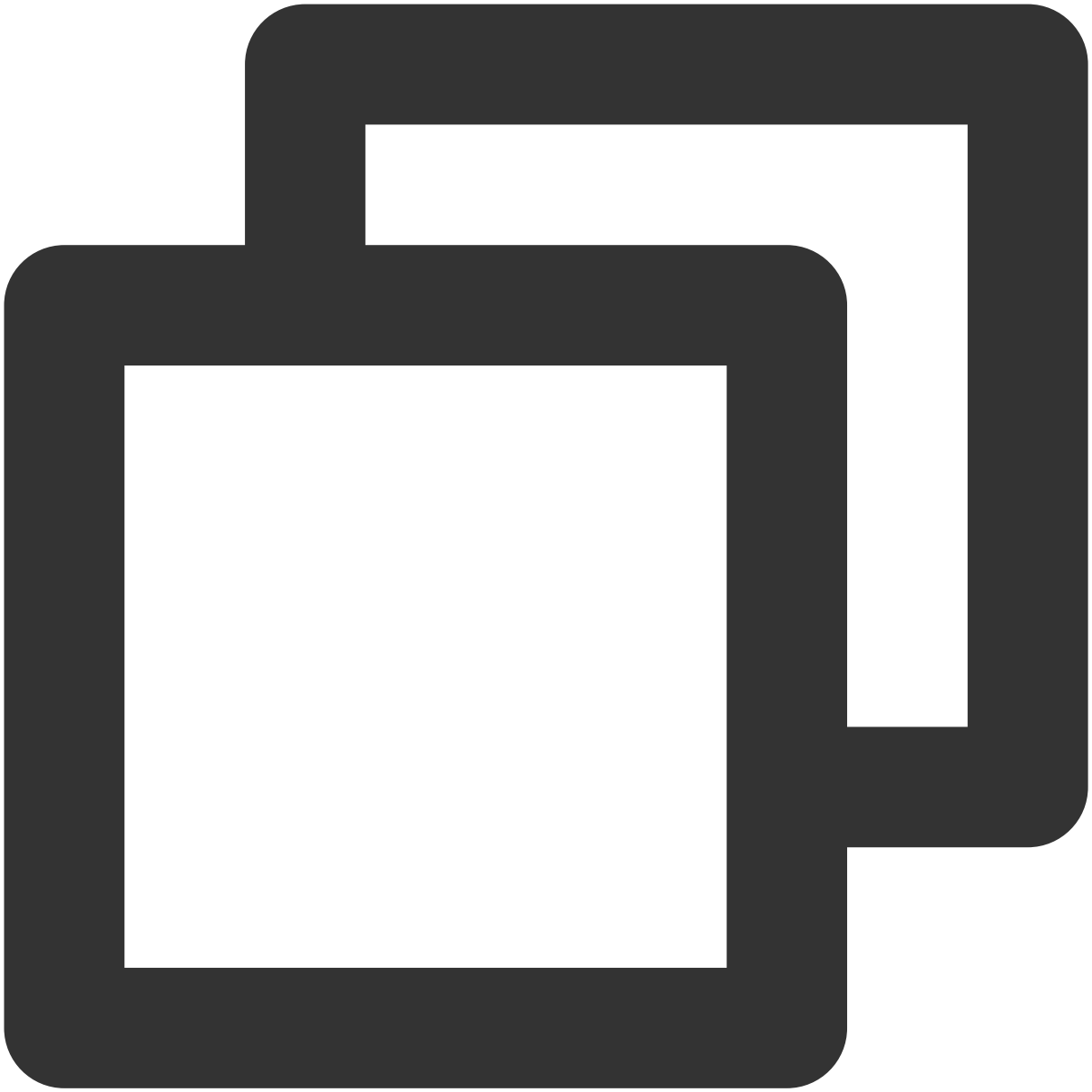
创建 Namespace 时，如果没有指定读写策略（rPolicy/wPolicy），默认会使用配置文件中指定的 read/write type，或使用默认值（CACHE/CACHE_THROUGH）。

同理，也可以创建一个命名空间 namespace 用于挂载腾讯云 HDFS：



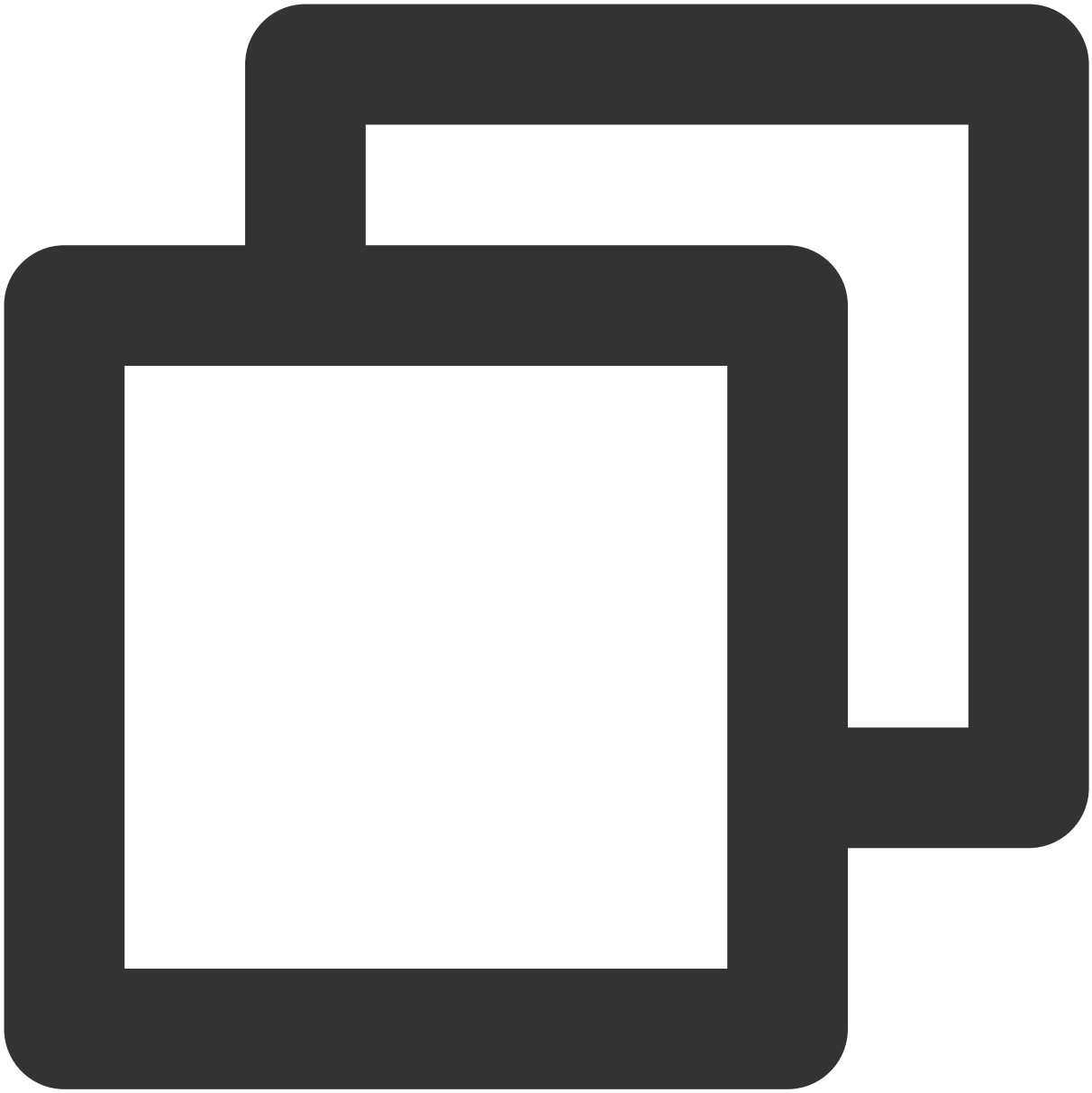
```
goosefs ns create MyNamespaceCHDFS ofs://xxxxx-xxxx.chdfs.ap-guangzhou.myqcloud.com
--attribute fs.ofs.user.appid=1250000000
--attribute fs.ofs.tmp.cache.dir=/tmp/chdfs
```

2. 创建成功后，可以通过 `ls` 命令列出集群中创建的所有 namespace：



```
$ goosefs ns ls
namespace              mountPoint               ufsPath                              cr
myNamespace    /myNamespace   cosn://bucketName-125xxxxxx/3TB  03-11-2021 11:43:06:
myNamespaceCHDFS /myNamespaceCHDFS ofs://xxxxx-xxxx.chdfs.ap-guangzhou.myqcloud.com
```

3. 执行如下命令，指定 namespace 的详细信息。



```
$ goosefs ns stat myNamespace

NamespaceStatus{name=myNamespace, path=/myNamespace, ttlTime=-1, ttlAction=DELETE,
```

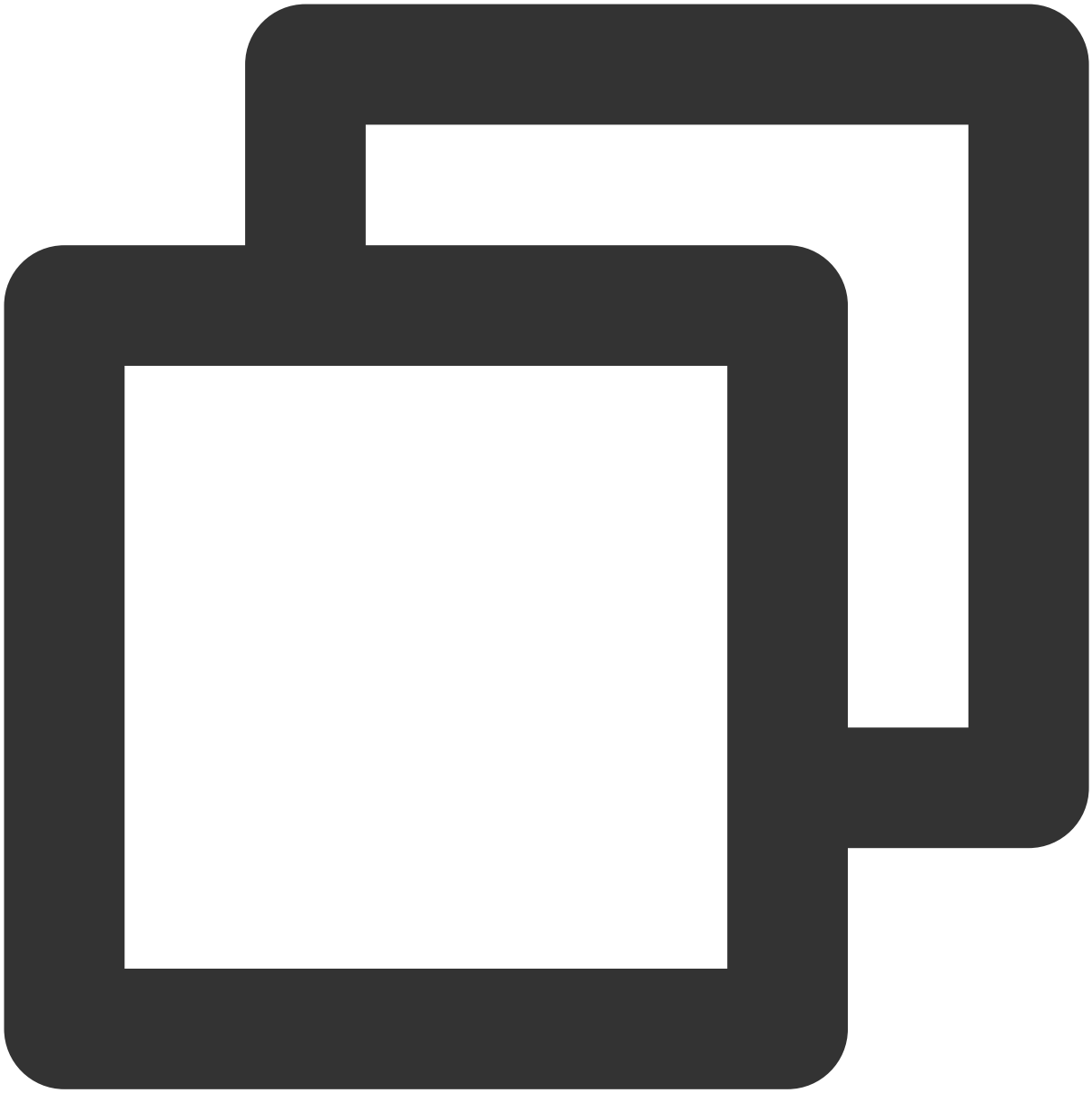
元数据中记录的信息包括如下内容：

序号	参数	描述
1	name	namespace 的名字

2	path	namespace 在 GooseFS 中的路径
3	ttlTime	namespace 下目录和文件的 ttl 周期
4	ttlAction	namespace 下目录和文件的 ttl 处理动作，有两种处理动作：FREE 和 DELETE，默认是 FREE
5	ufsPath	namespace 在 ufs 上的挂载路径
6	creationTimeMs	namespace 的创建时间，单位是毫秒
7	lastModificationTimeMs	namespace 下目录和文件的最后修改时间，单位是毫秒
8	persistenceState	namespace 的持久化状态
9	mountPoint	namespace 是否是一个挂载点，始终为 true
10	mountId	namespace 挂载点 ID
11	acl	namespace 的访问控制列表
12	defaultAcl	namespace 的默认访问控制列表
13	owner	namespace 的 owner
14	group	namespace 的 owner 所属的 group
15	mode	namespace 的 POSIX 权限
16	writePolicy	namespace 的 写策略
17	readPolicy	namespace 的 读策略

使用 GooseFS 预热 Table 中的数据

1. GooseFS 支持将 Hive Table 中的数据预热到 GooseFS 中，在预热之前需要先将相关的 DB 关联到 GooseFS 上，相关命令如下：

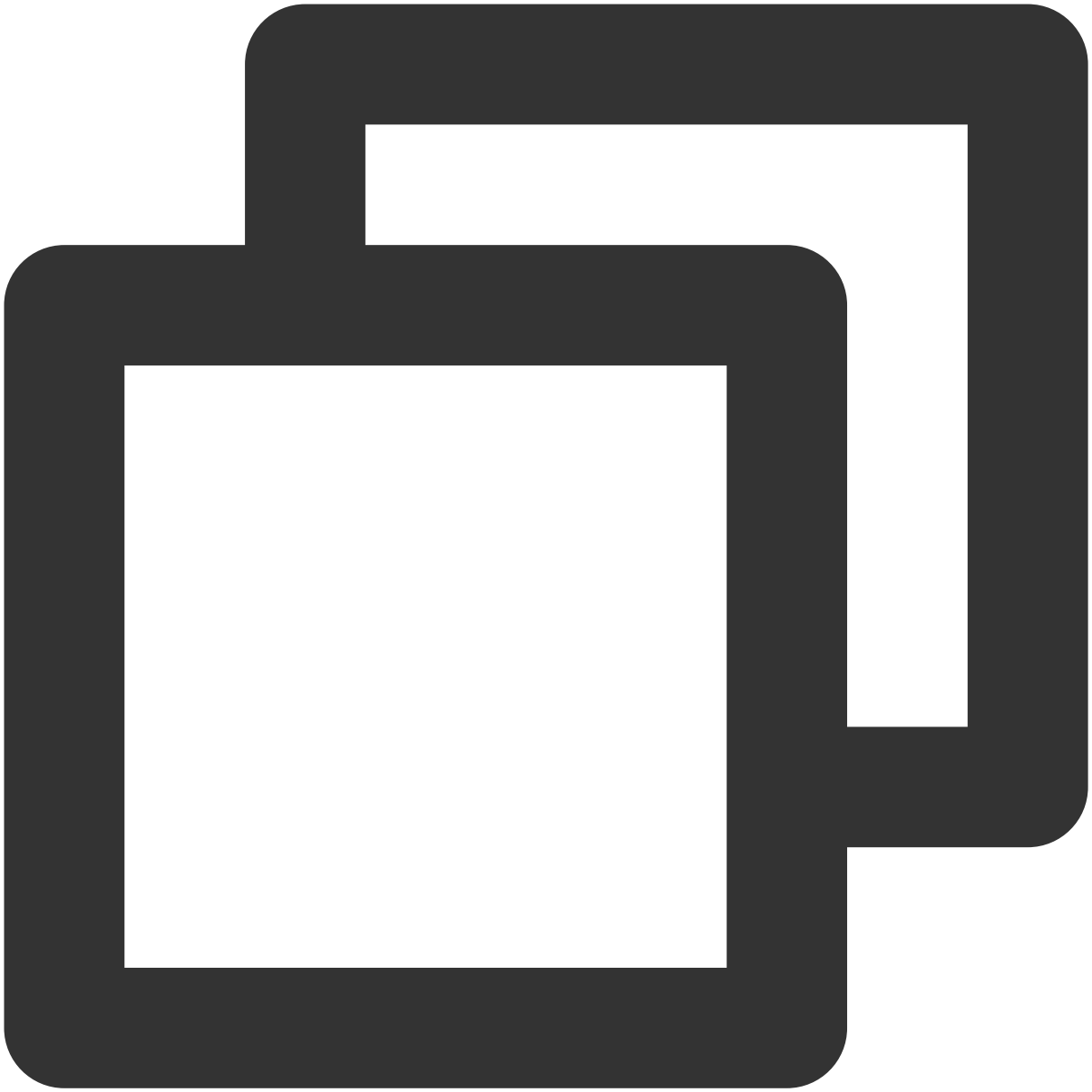


```
$ goosefs table attachdb --db test_db hive thrift://
172.16.16.22:7004 test_for_demo
```

注意

命令中的 thrift 需要填写实际的 Hive Metastore 的地址。

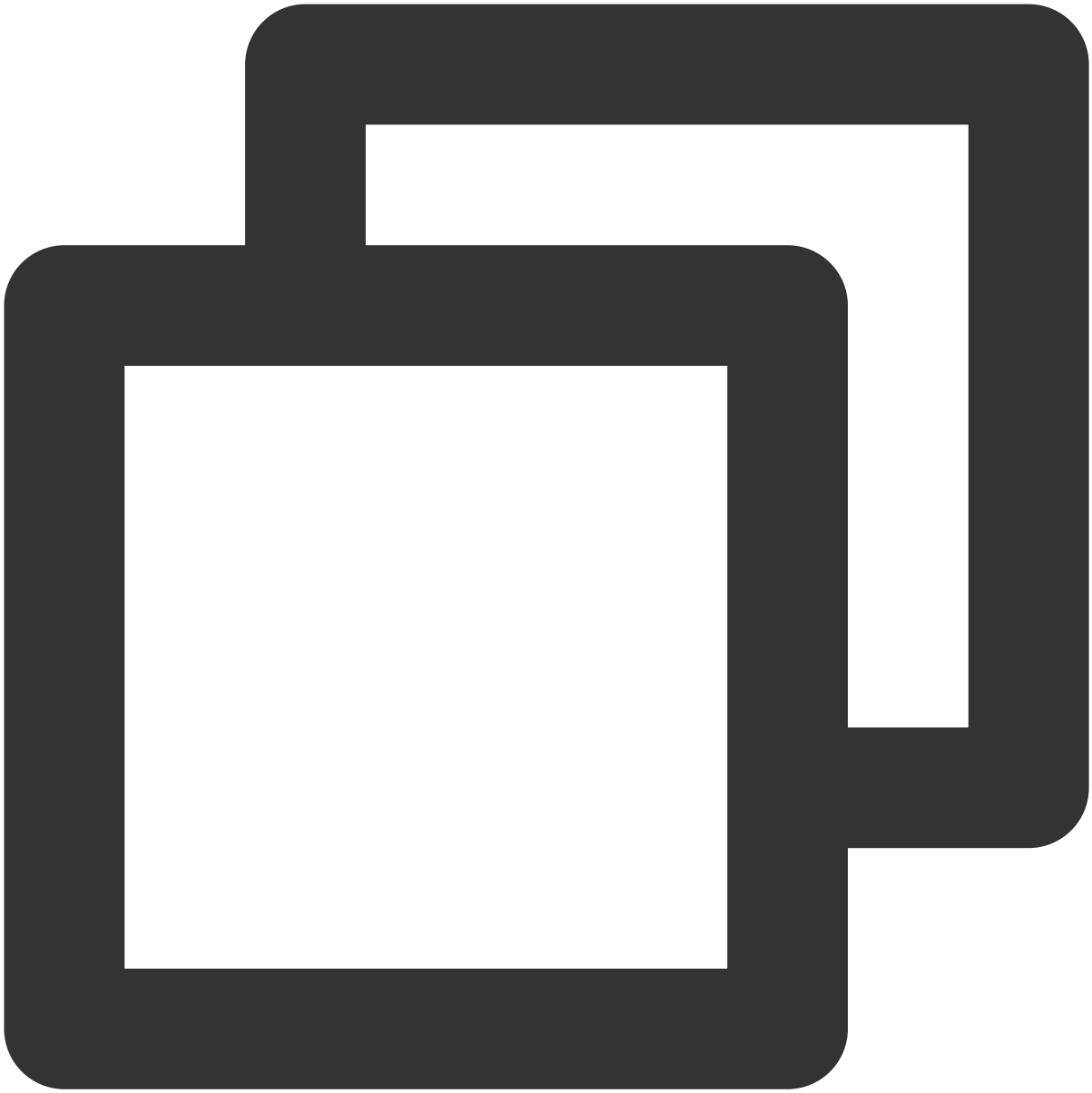
2. 添加完 DB 后，可以通过 ls 命令查看当前关联的 DB 和 Table 的信息：



```
$ goosefs table ls test_db web_page

OWNER: hadoop
DBNAME.TABLENAME: testdb.web_page (
   wp_web_page_sk bigint,
   wp_web_page_id string,
   wp_rec_start_date string,
   wp_rec_end_date string,
   wp_creation_date_sk bigint,
   wp_access_date_sk bigint,
   wp_autogen_flag string,
```

```
    wp_customer_sk bigint,
    wp_url string,
    wp_type string,
    wp_char_count int,
    wp_link_count int,
    wp_image_count int,
    wp_max_ad_count int,
)
PARTITIONED BY (
)
LOCATION (
    gfs://172.16.16.22:9200/myNamespace/3000/web_page
)
PARTITION LIST (
    {
    partitionName: web_page
    location: gfs://172.16.16.22:9200/myNamespace/3000/web_page
    }
)
```

3. 通过 load 命令预热 Table 中的数据：

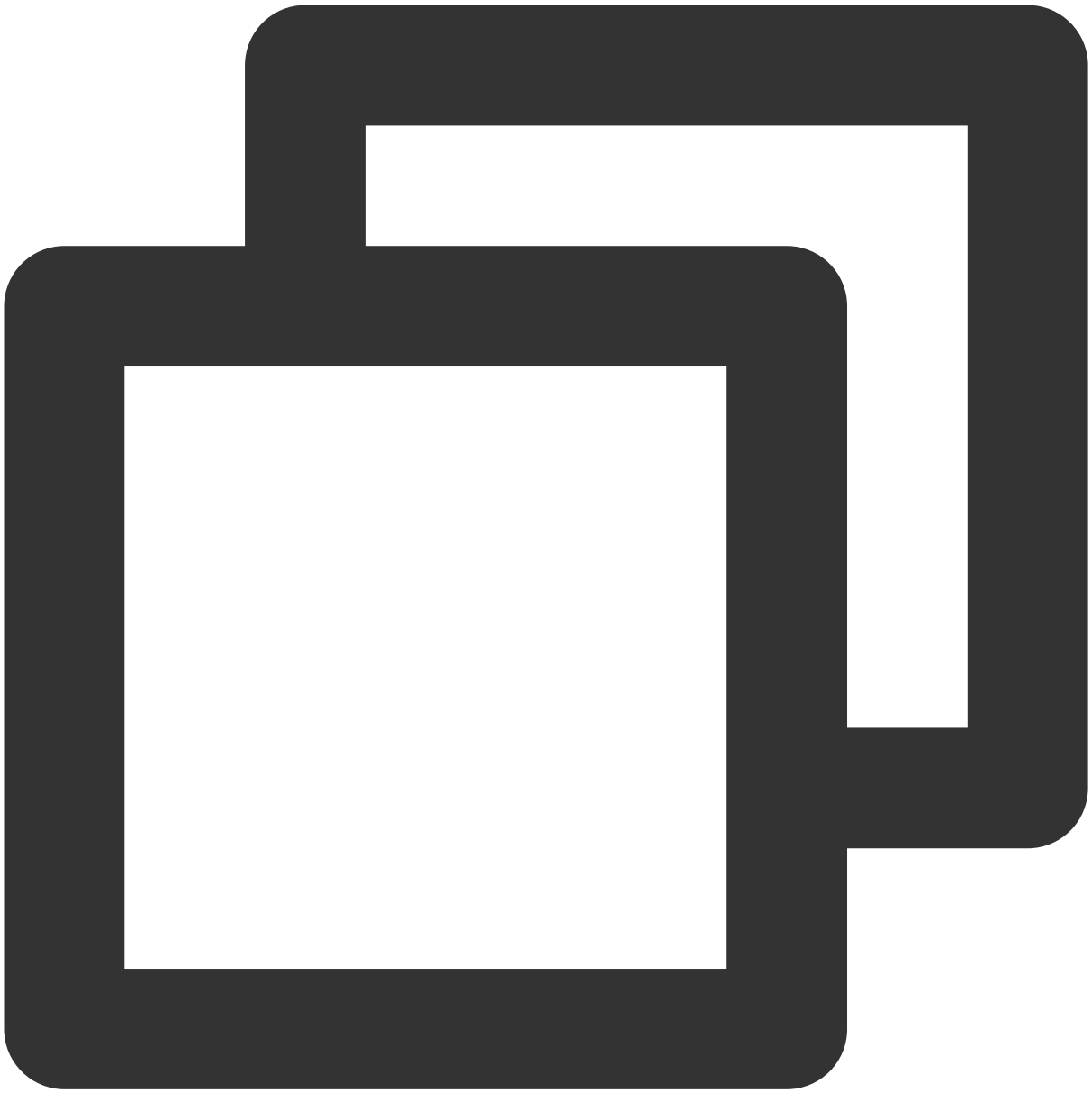


```
$ goosefs table load test_db web_page
Asynchronous job submitted successfully, jobId: 1615966078836
```

预热 Table 中的数据是一个异步任务，因此会返回一个任务 ID。可以通过 goosefs job stat <Job Id> 命令查看预热作业的执行进度。当状态为 "COMPLETED" 后，则整个预热过程完成。

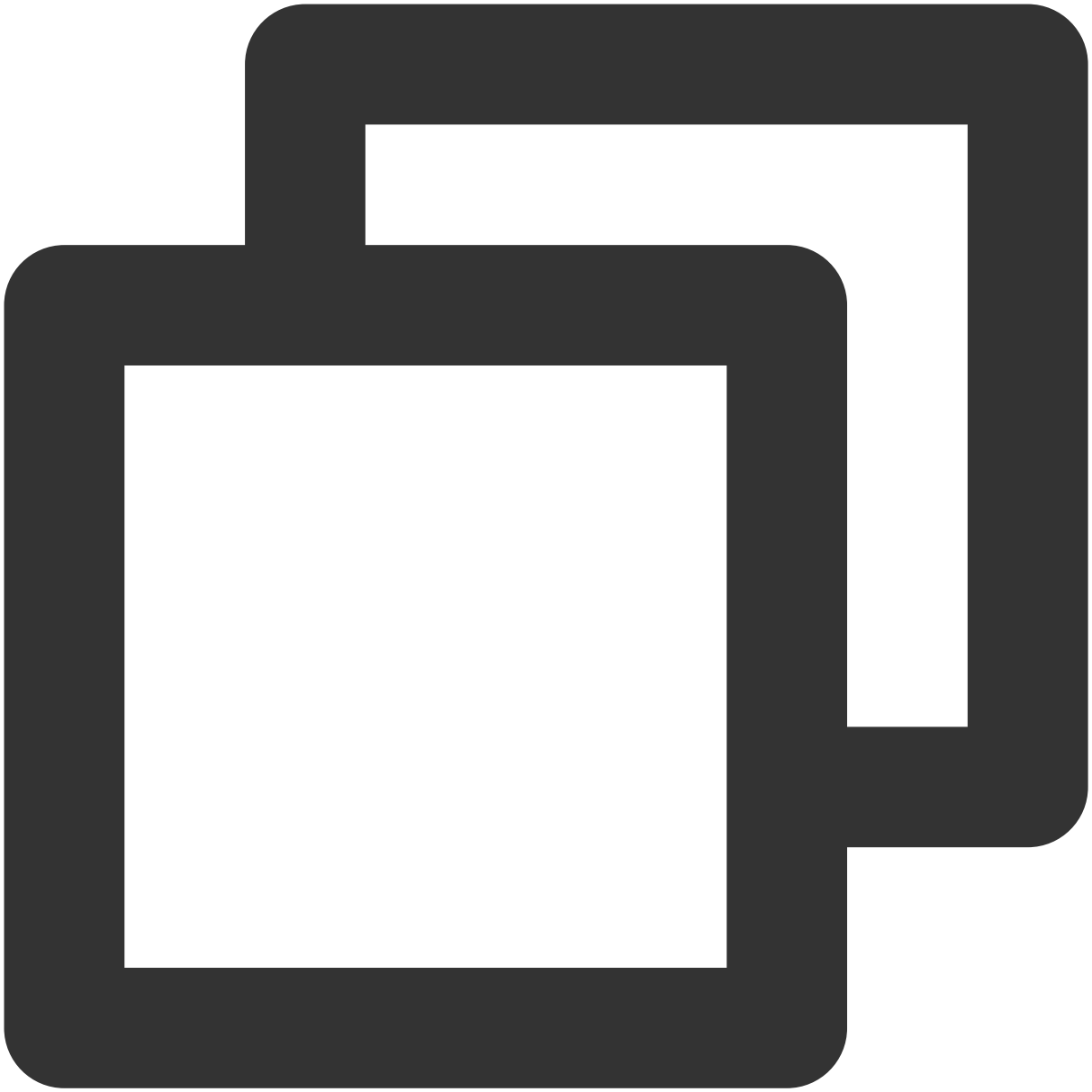
使用 GooseFS 进行文件上传和下载操作

1. GooseFS 支持绝大部分文件系统操作命令，可以通过以下命令来查询当前支持的命令列表：



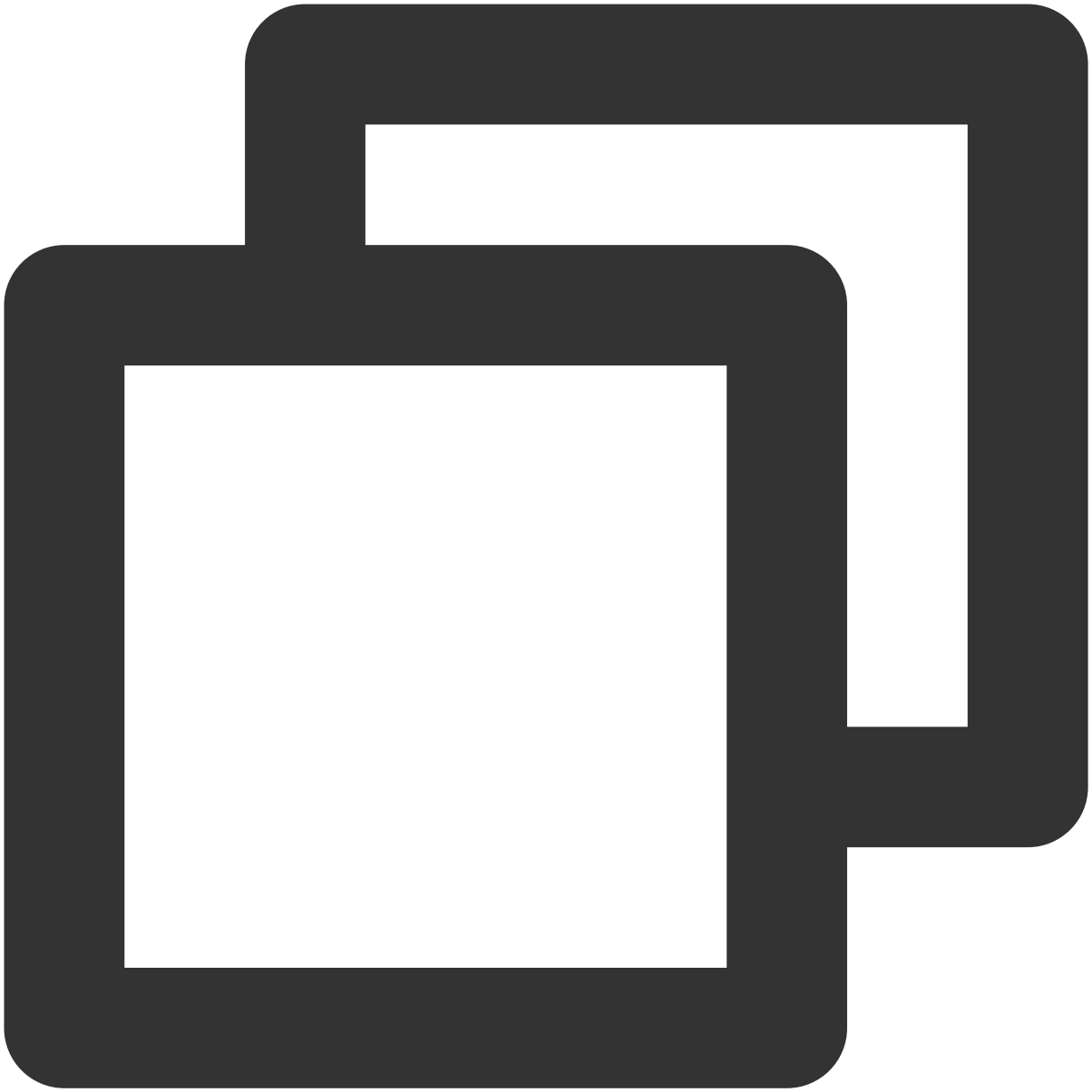
```
$ goosefs fs
```

2. 可以通过 `ls` 命令列出 GooseFS 中的文件，以下示例展示如何列出根目录下的所有文件：



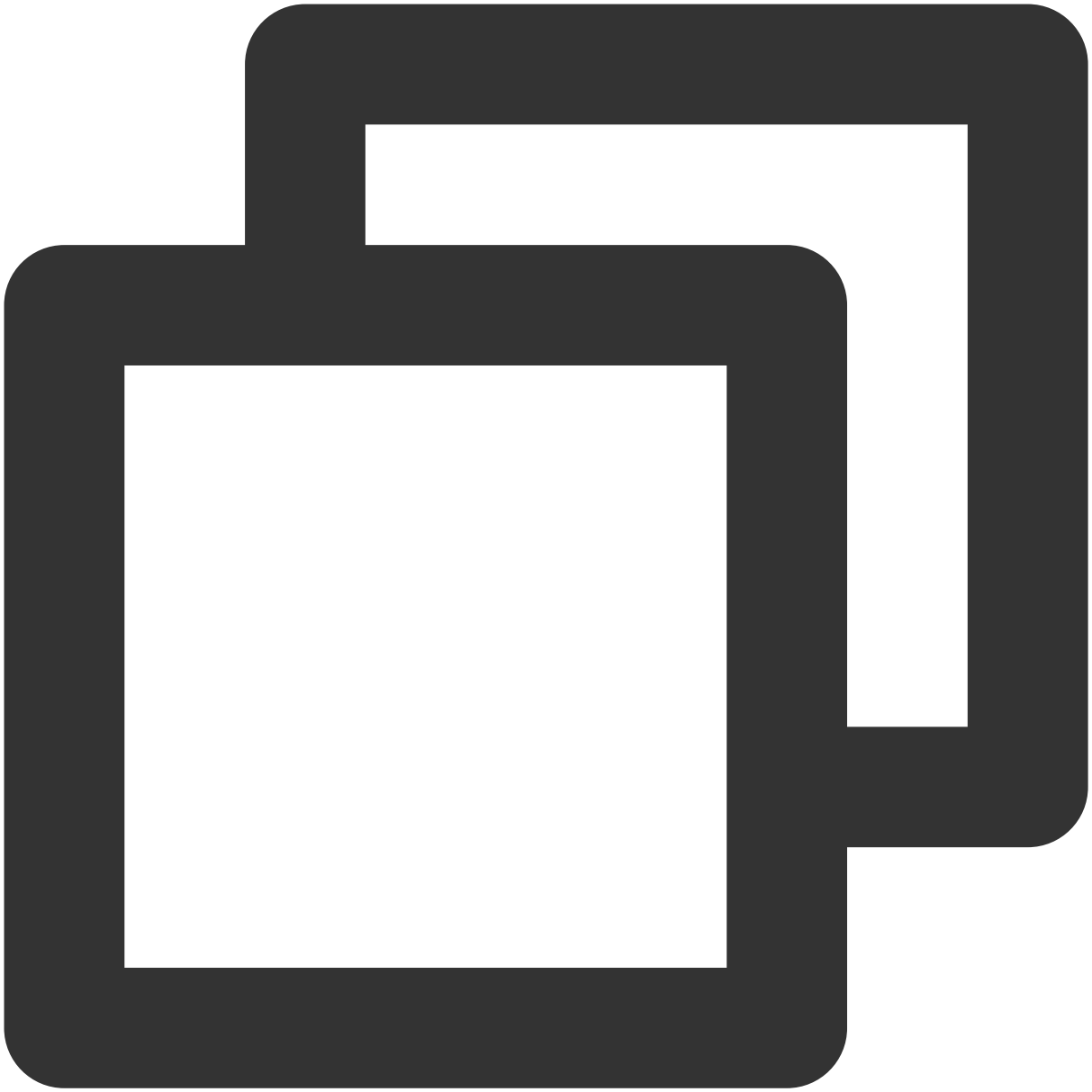
```
$ goosefs fs ls /
```

3. 可以通过 `copyFromLocal` 命令将数据从本地拷贝到 GooseFS 中：



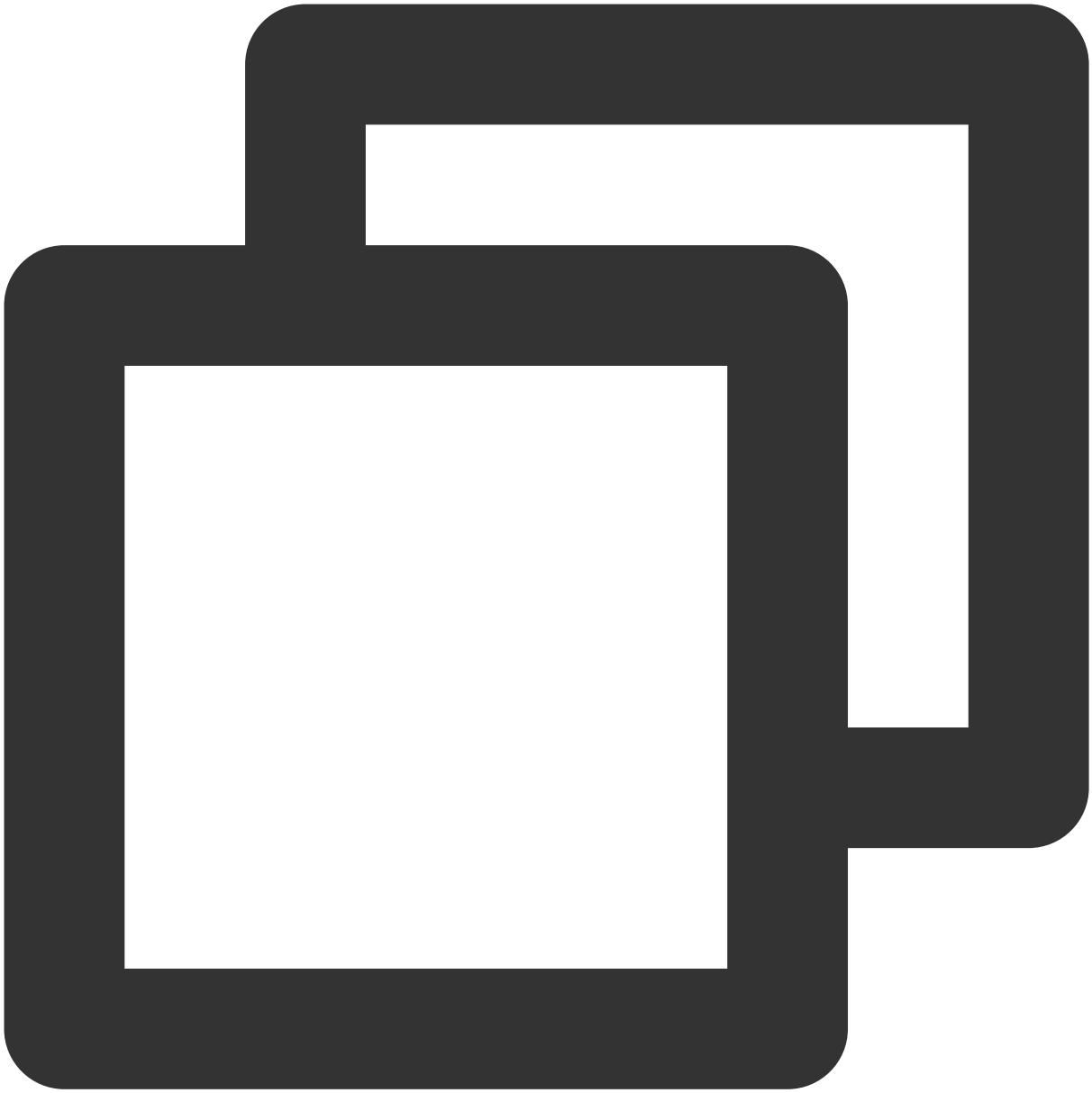
```
$ goosefs fs copyFromLocal LICENSE /LICENSE
Copied LICENSE to /LICENSE
$ goosefs fs ls /LICENSE
-rw-r--r--  hadoop         supergroup                20798       NOT_PERSISTED 03-26
```

4. 可以通过 `cat` 命令查看文件内容：



```
$ goosefs fs cat /LICENSE
Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/
TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION
...
```

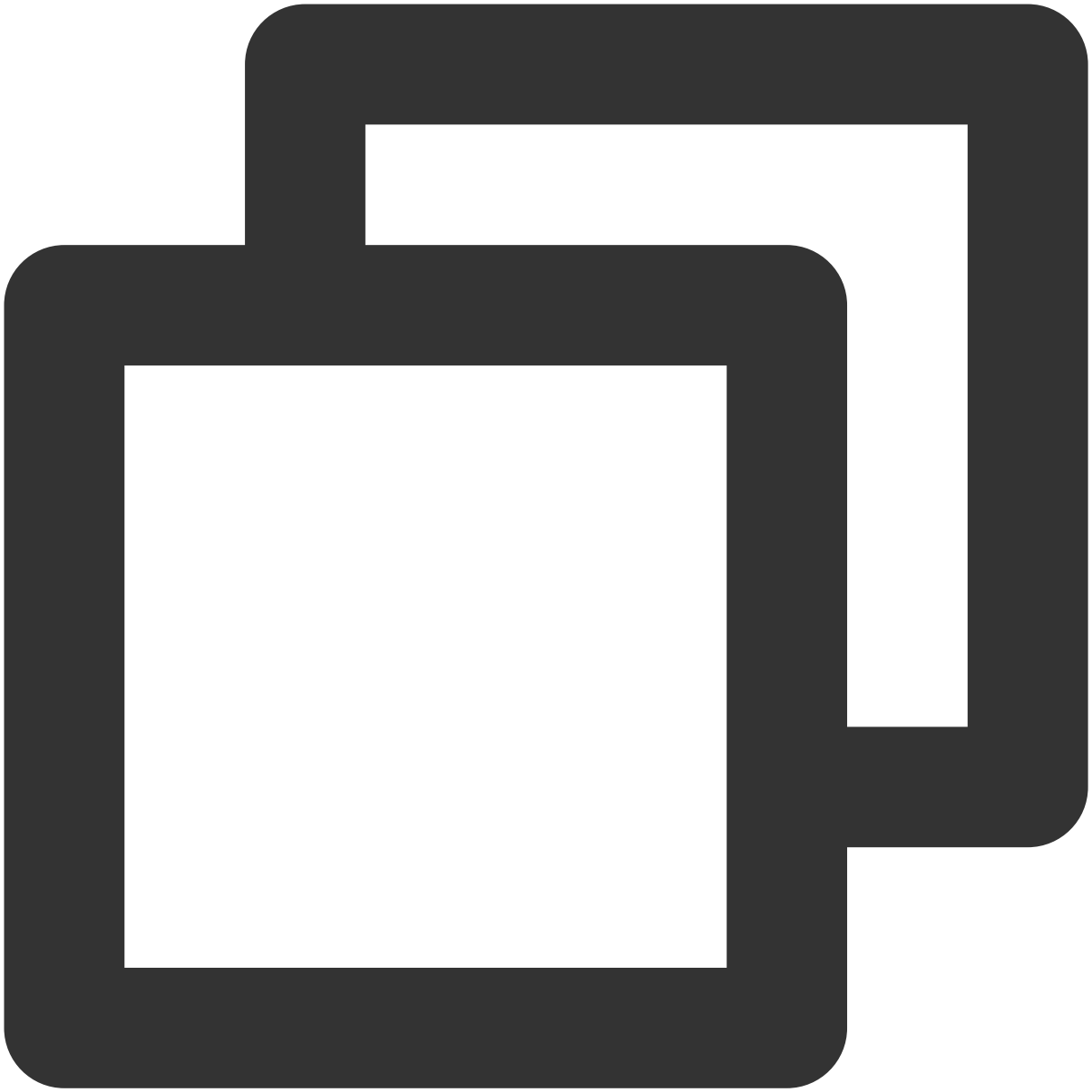
5. GooseFS 默认使用本地磁盘作为底层文件系统，默认文件系统路径为 `./underFSStorage`，可以通过 `persist` 命令将文件持久化存储到本地文件系统中：



```
$ goosefs fs persist /LICENSE
persisted file /LICENSE with size 26847
```

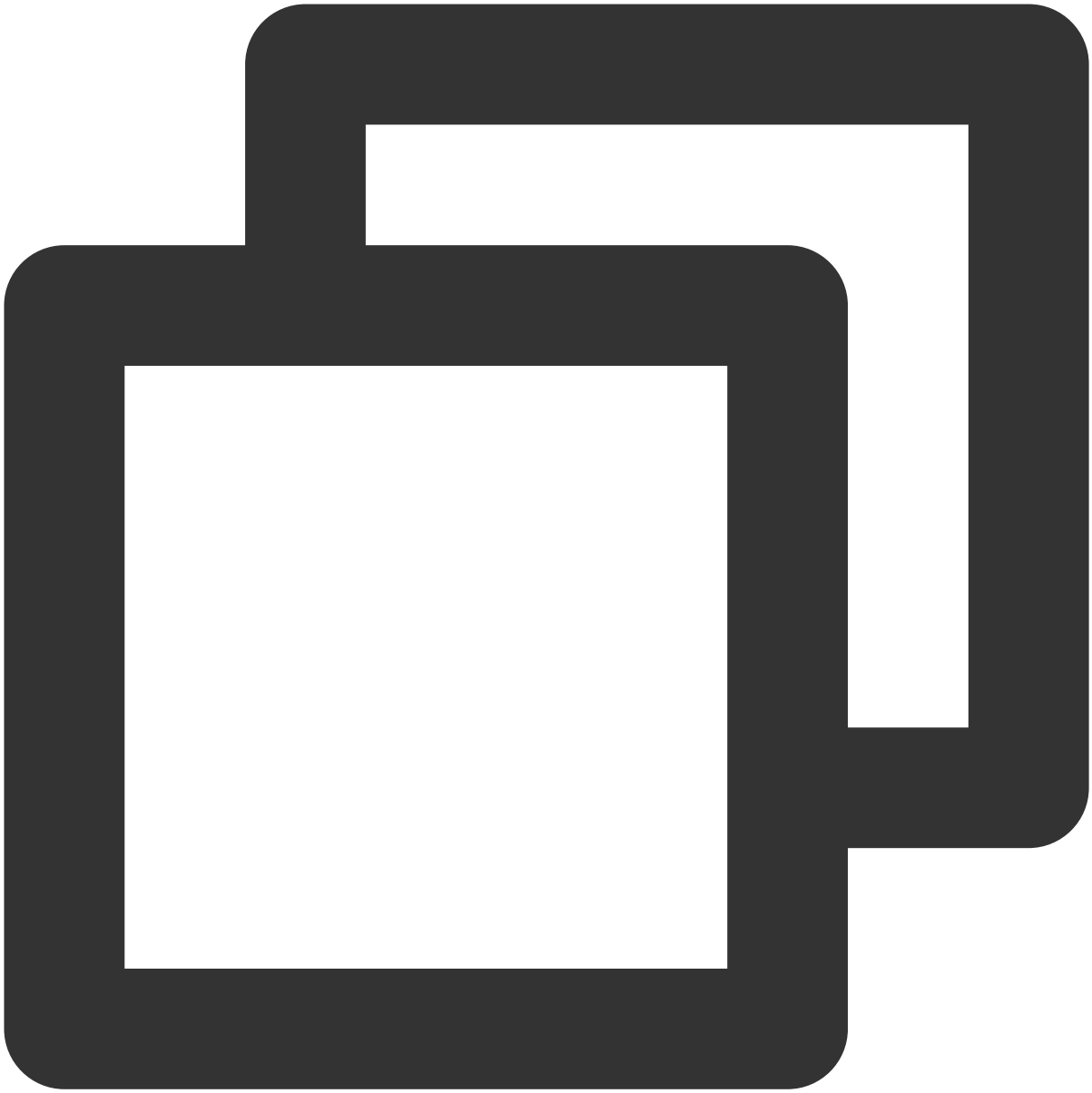
使用 GooseFS 加速文件上传和下载操作

1. 检查文件存储状态，确认文件是否已被缓存。文件状态 `PERSISTED` 代表文件已在内存中，文件状态 `NOT_PERSISTED` 则代表文件不在内存中：



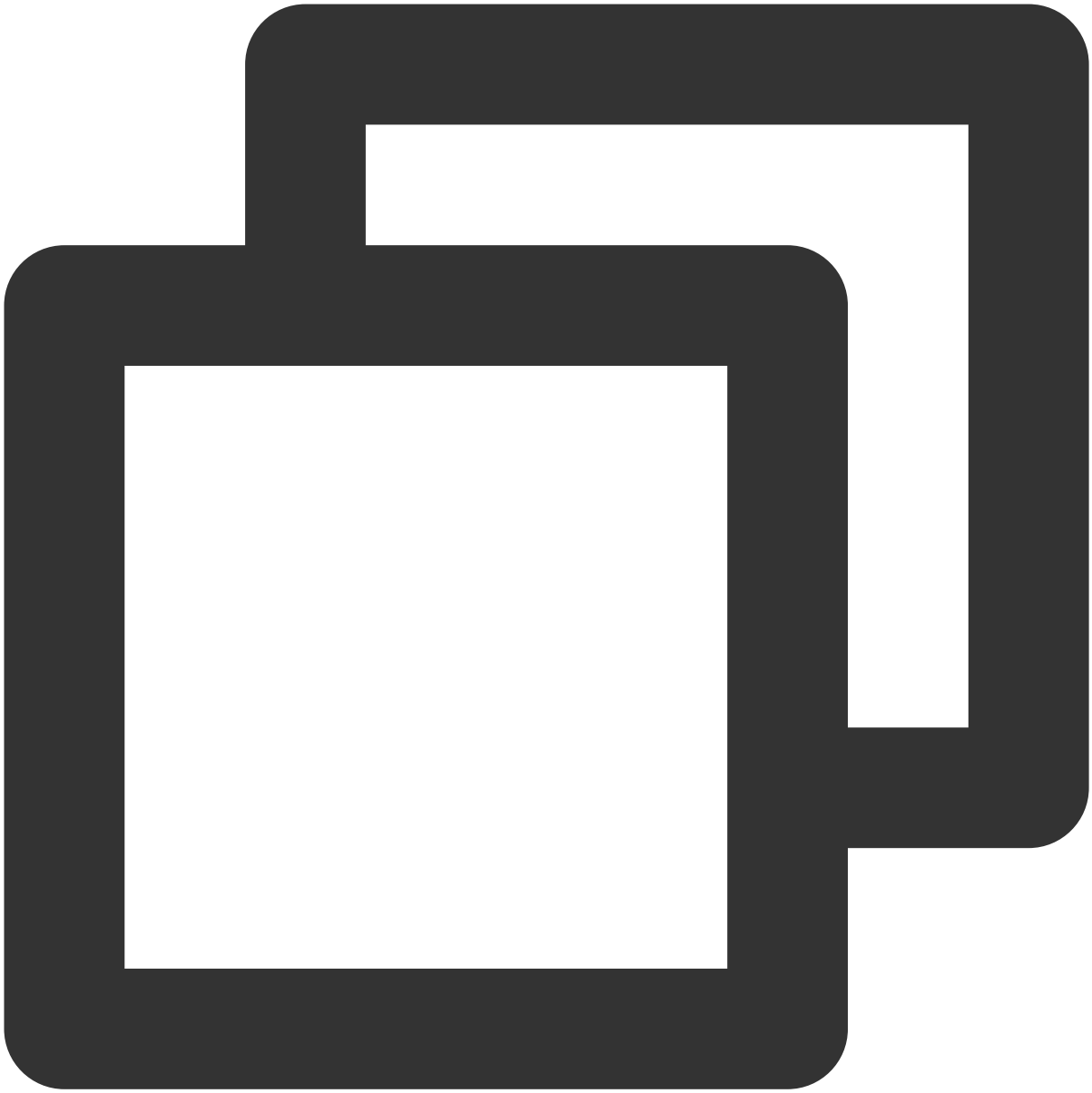
```
$ goosefs fs ls /data/cos/sample_tweets_150m.csv
-r-x------ staff  staff 157046046 NOT_PERSISTED 01-09-2018 16:35:01:002   0% /data/
```

2. 统计文件中有多少单词 “tencent”，并计算操作耗时：



```
$ time goosefs fs cat /data/s3/sample_tweets_150m.csv | grep-c tencent
889
real    0m22.857s
user    0m7.557s
sys     0m1.181s
```

3. 将该数据缓存到内存中可以有效提升查询速度，详细示例如下：

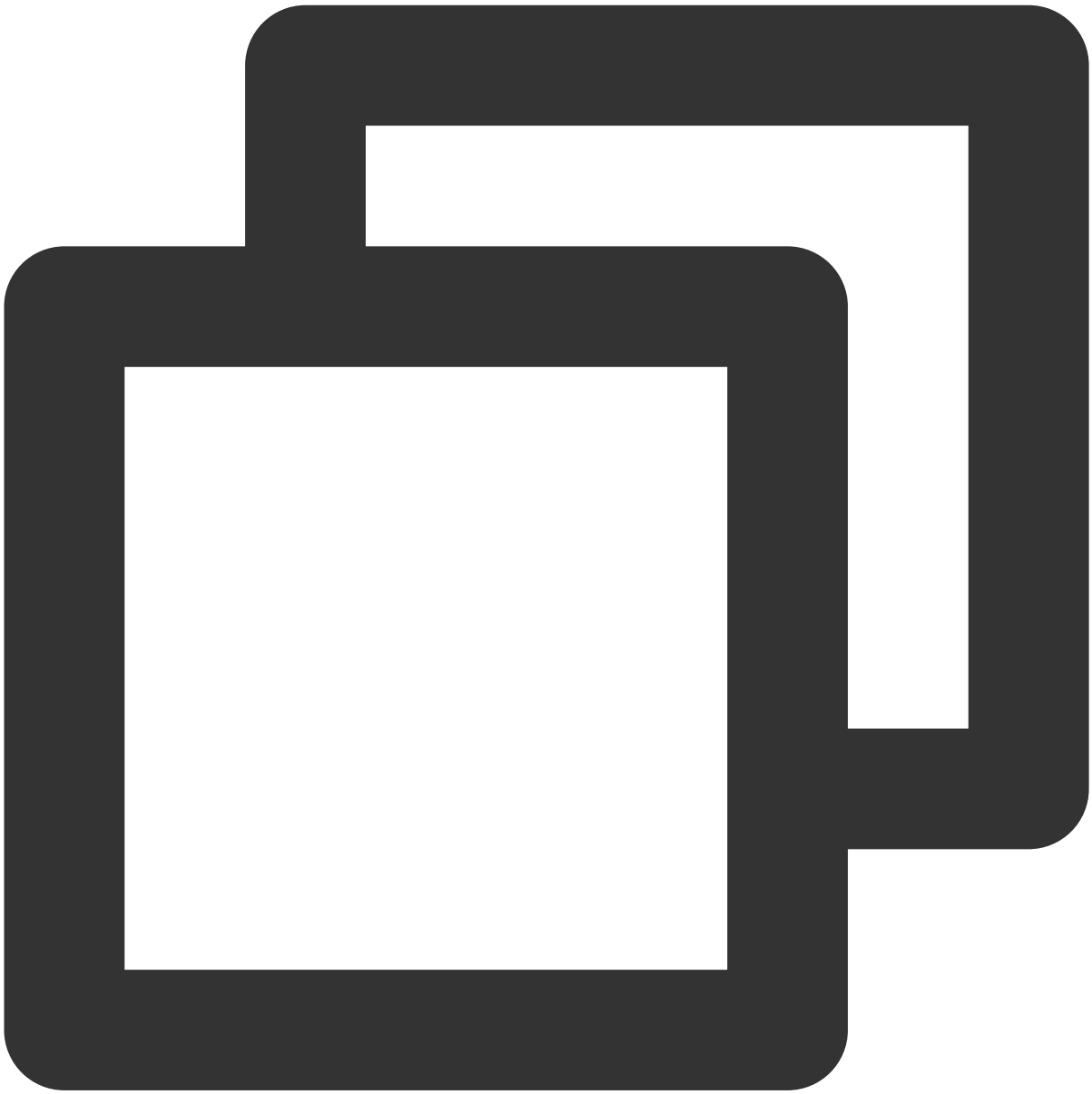


```
$ goosefs fs ls /data/cos/sample_tweets_150m.csv
-r-x------ staff  staff 157046046
ED 01-09-2018 16:35:01:002   0% /data/cos/sample_tweets_150m.csv
$ time goosefs fs cat /data/s3/sample_tweets_150m.csv | grep-c tencent
889
real    0m1.917s
user    0m2.306s
sys      0m0.243s
```

可见，系统处理延迟从1.181s减少到了0.243s，得到了10倍的提升。

关闭 GooseFS

通过如下命令可以关闭 GooseFS：



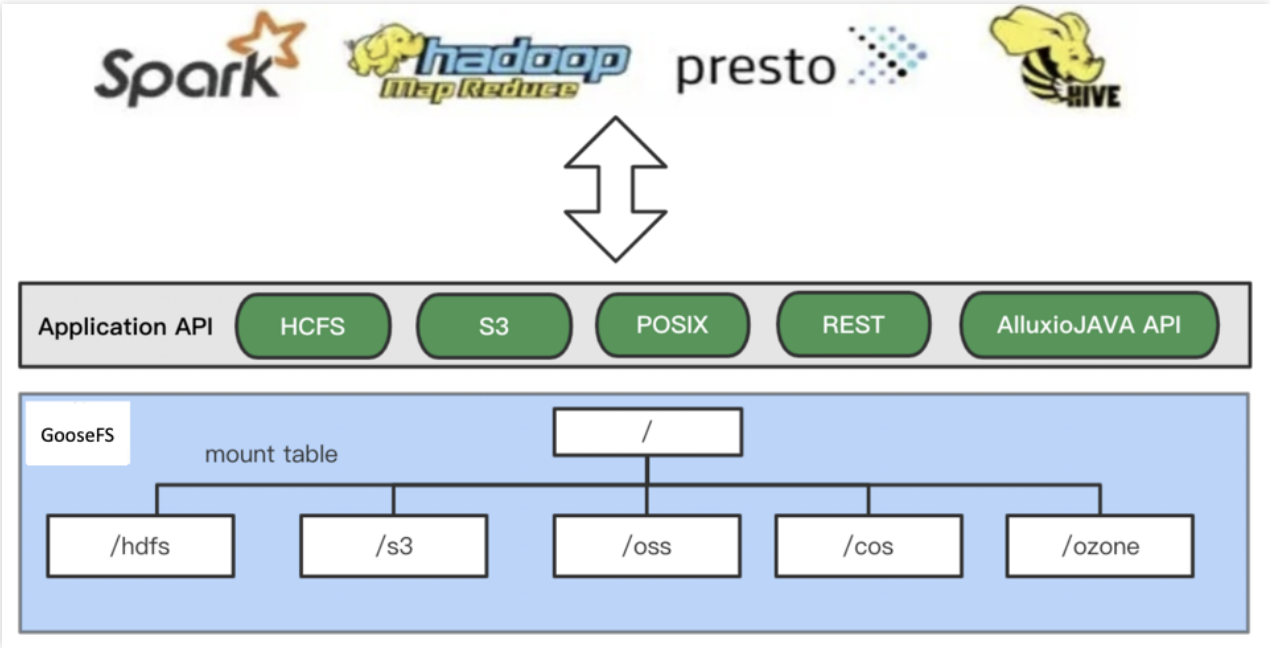
```
$ ./bin/goosefs-stop.sh local
```

核心特性
缓存能力

最近更新时间：2024-01-06 11:17:59

缓存能力概述

GooseFS 基于开源 Alluxio 的架构，提供强大的分布式 Cache 能力，能够帮助用户统一跨各种平台用户数据的同时，还有助于为用户提升总体 I/O 吞吐量。
主要通过以下两套方式：
GooseFS NameSpace 远端存储：NS 存储空间底座来自于外部存储，可以跨越 COSN 和 HDFS，连接一个或者多个 NS 到同一个底层存储，可以持久化海量数据。选用 COSN 的存算分离方案，GooseFS NameSpace 借助 COSN 可以给大数据存储提供弹性扩展和高可用能力。
GooseFS 本地 Cache 存储：GooseFS 通过 master 和 worker 提供的分布式 Cache 能力，将应用层产生的数据持久化在应用节点本地的内存和磁盘上，主要存储热数据和暂态数据。每个本地存储节点都可由用户配置，NameSpace 对接的远端持久化层也会经过同一个 client 服务用户的读取操作。
用户可以操作 Cache Policy，决定读写模式中使用本地 Cache 存储和 NameSpace 远端存储。



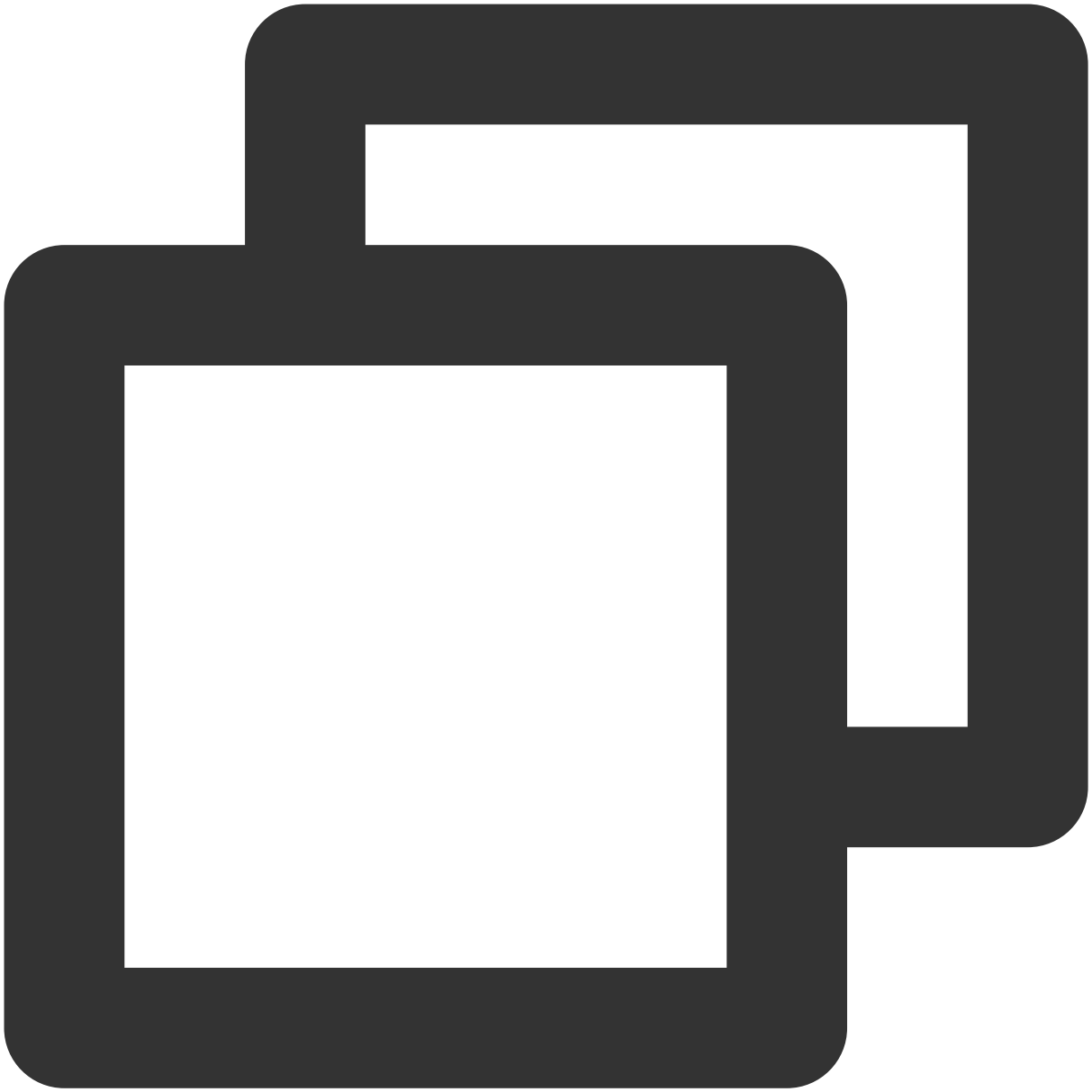
GooseFS 提供相同的本地 Cache 和远端 NameSpace 整合的能力，帮助用户全面实现存储分离的统一方案，同时建立应用节点的数据亲缘性。

GooseFS 缓存配置

用户可以进入 goosefs-site.properties 文件中查看 GooseFS 缓存配置情况。目前GooseFS的缓存设置可以从缓存层级、缓存淘汰策略等方面了解，缓存层级方面主要有单层存储和多层存储两种，缓存淘汰策略主要区分为 LRU 和 LRFU 两种。以下进行详细介绍。

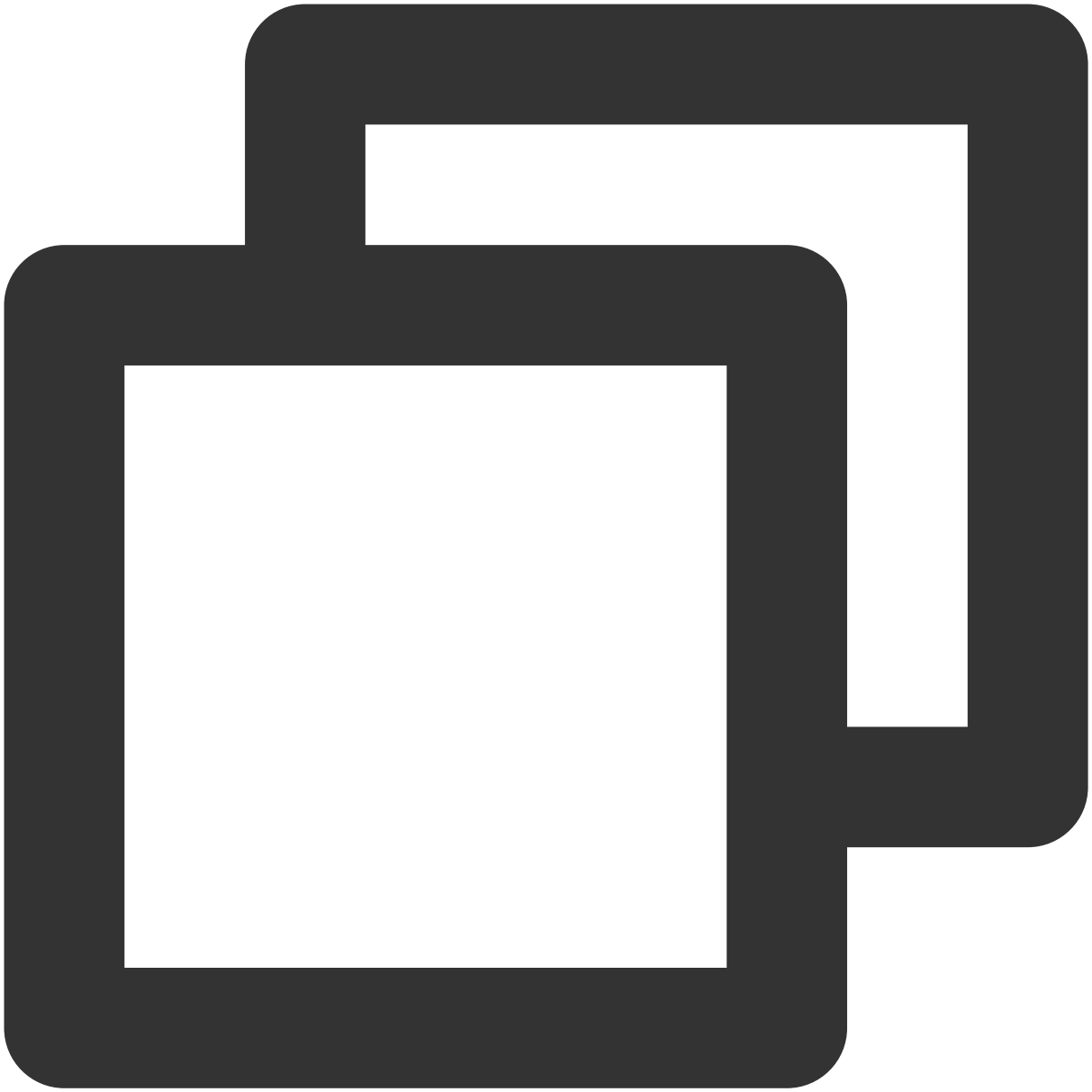
1. 缓存层级

GooseFS 提供单层存储和多层存储两种缓存层级。单层存储只使用一种存储介质，多层存储使用多种存储，可以根据业务负载情况来决定使用何种存储介质，以提供匹配的 I/O 性能。GooseFS 默认配置为单层存储，在多层存储的场景下，缓存淘汰会带来较多的性能开销，因此推荐使用**单层存储**。根据 I/O 性能的高低，一般可以选用 MEM（内存）、SSD（固态存储）和 HDD（硬盘存储）作为存储介质。
通过修改 goosefs-site.properties 配置文件中的 levels 参数可以修改缓存层级，入参为 1 时代表只使用单层存储，入参为 3 代表使用三层存储：



```
goosefs.worker.tieredstore.levels=1
```

通过修改 goosefs-site.properties 配置文件中的 alias 参数可以修改缓存层对应的存储介质：



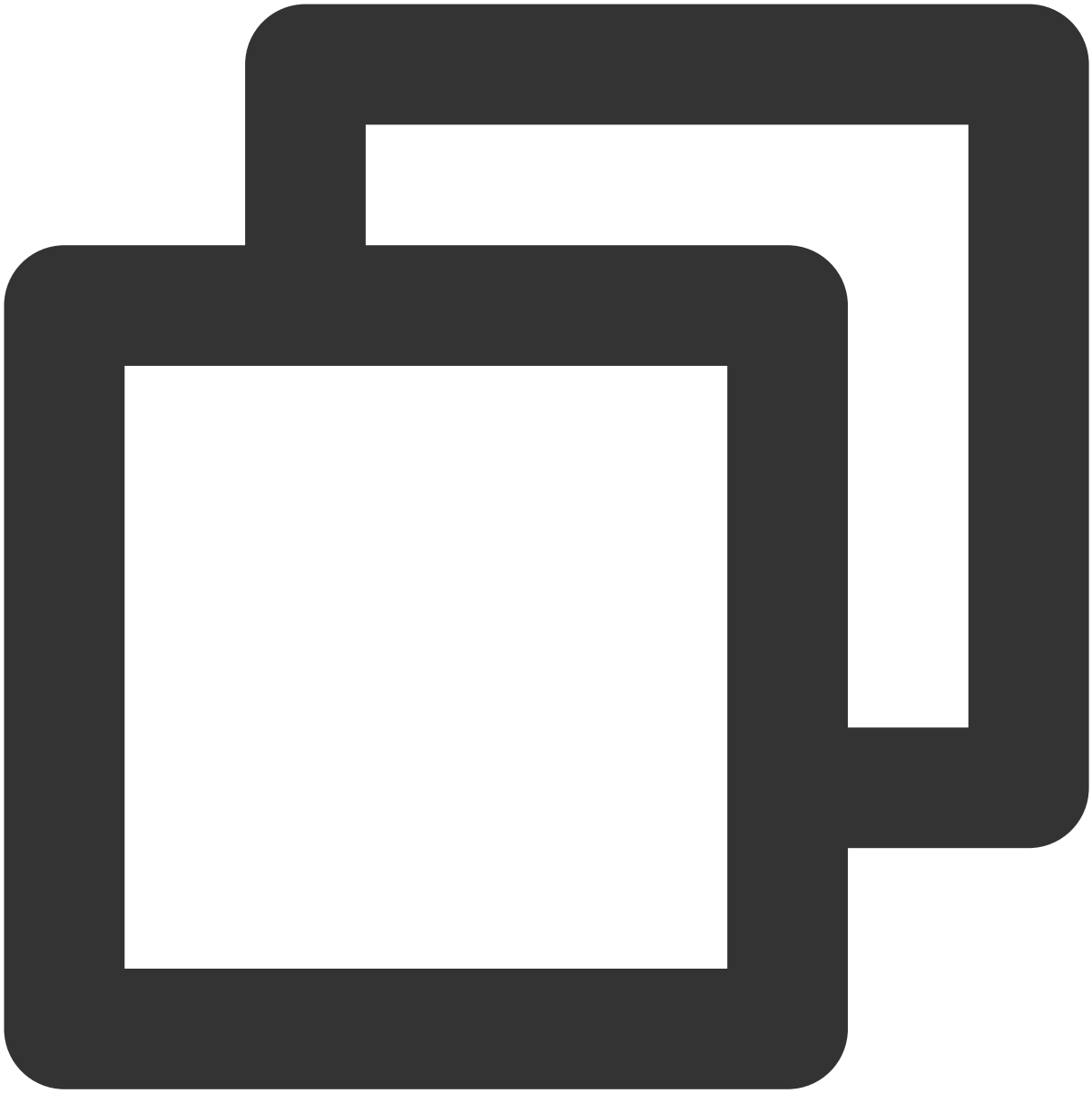
```
goosefs.worker.tieredstore.level{x}.alias=MEM
```

注意

level{x} 代表第几层存储，例如 level0 代表单层存储。

1.1 单层存储

单层存储模式下，常用的系统配置项如下：



```
goosefs.worker.tieredstore.levels=1
goosefs.worker.tieredstore.level0.alias=HDD
goosefs.worker.ramdisk.size=16GB
goosefs.worker.memory.size=100GB
goosefs.worker.tieredstore.level0.dirs.path=/data/GooseFSWorker,/data1/GooseFSWorke
goosefs.worker.tieredstore.level0.dirs.mediumtype=MEM,MEM,MEM,SSD,SSD
goosefs.worker.tieredstore.level0.dirs.quota=16GB,16GB,16GB,100GB,100GB
```

以下逐项介绍配置项内容。

ramdisk.size：该配置项用于指定 worker 节点占用内存的大小。ramdisk 的容量必须比 memory 小。设置后 GooseFS 会为每个 worker 节点分配指定大小的内存空间，并消耗系统总内存。

memory.size：该配置项用于指定 GooseFS 系统总内存，设置后 GooseFS 将会自动占用指定大小的存储介质，实际物理存储空间必须比 memory 数值大。

dirs.path：该配置项用于指定目录，由 GooseFS 为指定目录分配存储介质，需要结合 dirs.mediumtype 使用，如上示例展示将 /data/GooseFSWorker 挂载到 MEM 存储介质上，将/data4/GooseFSWorker 挂载到 SSD 存储介质上。需要注意，每个目录的顺序需要和存储介质的顺序一一匹配。

dirs.mediumtype：该配置项用于指定存储介质，由 GooseFS 为指定目录分配存储介质，需要结合 dirs.path 使用。默认可选的存储介质包括 MEM，SSD，如果挂载了其他存储介质，如 HDD，可按需修改配置。

dirs.quota：改配置项用于指定每个目录的预置空间，配置后由 GooseFS 为指定目录分配好空间，需要和目录的顺序一一对应。如上示例展示了需要为 /data/GooseFSWorker,/data1/GooseFSWorker,/data2/GooseFSWorker 等目录分配 16GB MEM 空间，为 /data3/GooseFSWorker,/data4/GooseFSWorker 目录分配 100GB SSD 空间。

1.2 分层存储

分层存储模式下，数据块的读写模式与单层存储的读写模式存在差异。单层存储模式下数据会直接从一种存储介质中读写，多层存储模式下数据会默认优先写入最顶层存储介质，读取的时候则需要将数据先挪到最顶层存储介质。具体地：

数据写入：新的数据块会被默认写入最顶层的存储介质；如果最顶层存储介质的存储空间已经满了，GooseFS 会把数据顺位存储到下一层的存储介质中；如果所有存储介质都已经存满数据，那么 GooseFS 会按照用户指定的缓存淘汰策略淘汰过期数据；如果过期数据无法淘汰，并且所有可用存储空间已满，则无法写入数据。

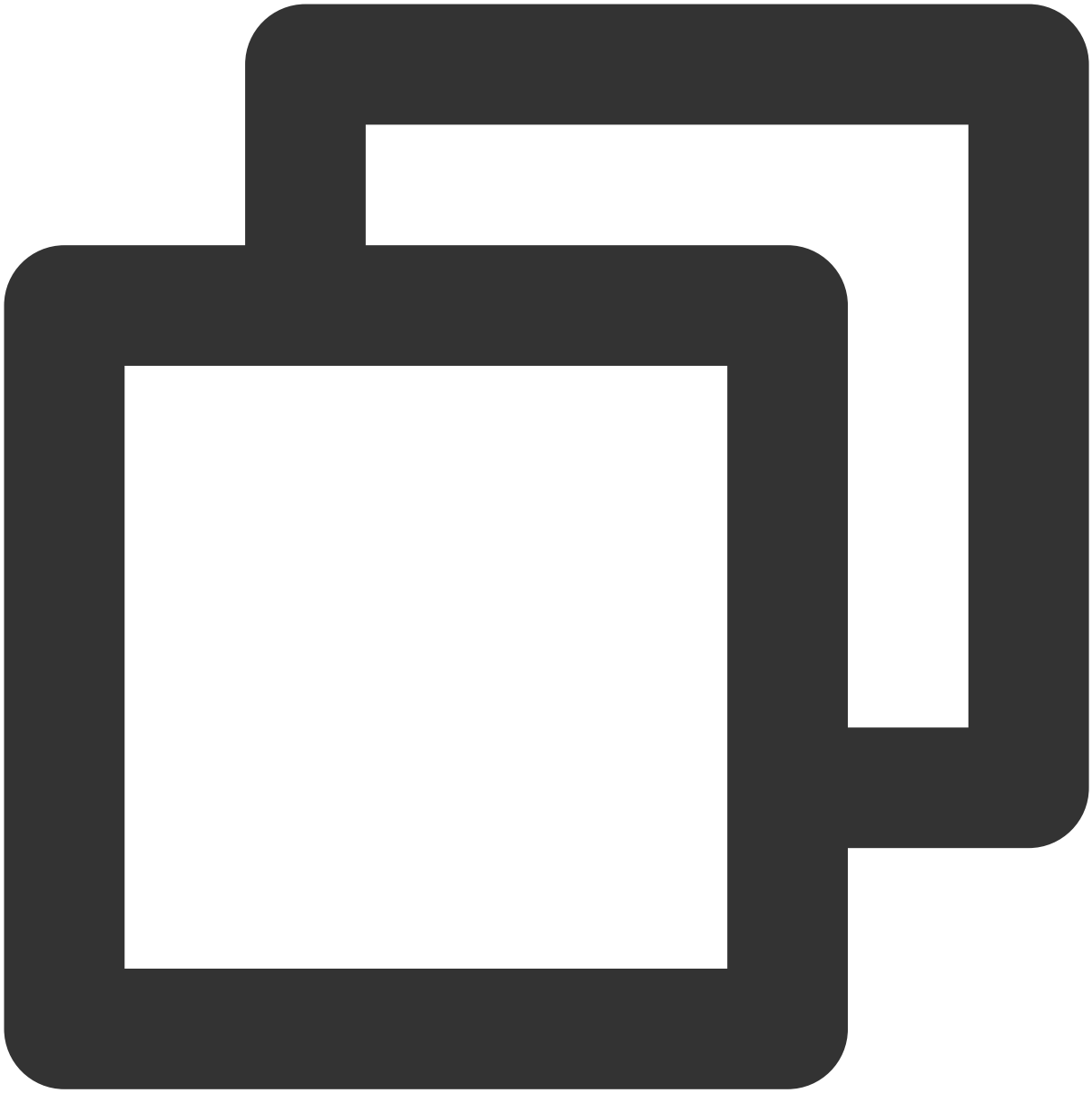
数据读取：多层存储模式下，会将冷数据透明地转储到更低层次的存储介质中，读取数据会将数据加热并放置到顶层存储中。

注意

在既定的淘汰策略下，GooseFS 会按照指定的释放空间来清理一定量的数据，该参数可以通过 goosefs.worker.tieredstore.free.ahead.bytes 指定，默认值为0。

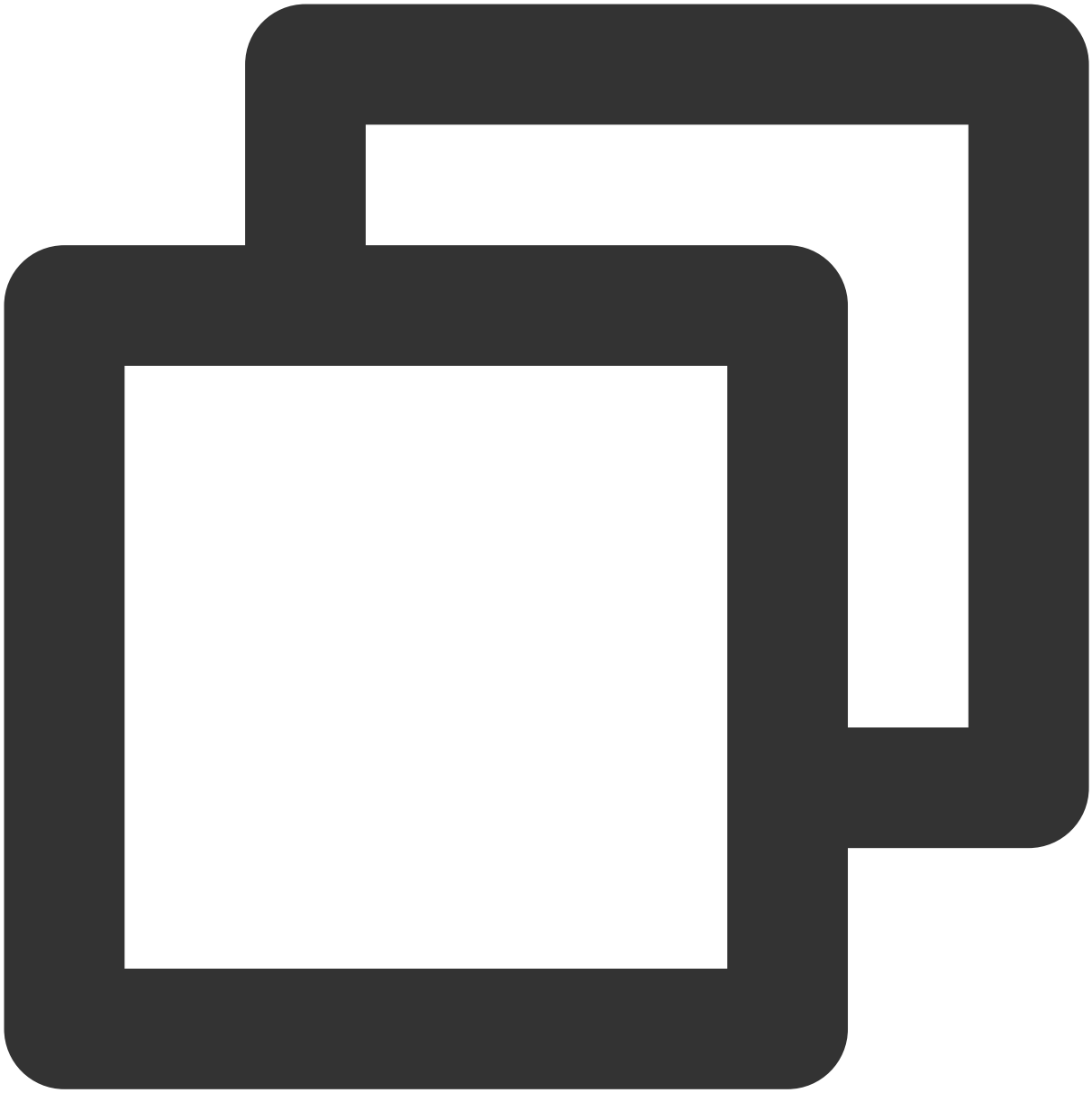
分层存储模式下，读取数据时可能会频繁将数据从低层存储介质转储到顶层存储介质中，进而导致频繁的缓存淘汰情况，对性能带来一定损耗，一般情况下，**推荐使用单层存储**。

分层存储模式下，常用的系统配置项如下：



```
goosefs.worker.tieredstore.levels
goosefs.worker.tieredstore.level{x}.alias
goosefs.worker.tieredstore.level{x}.dirs.path
goosefs.worker.tieredstore.level{x}.dirs.mediumtype
goosefs.worker.tieredstore.level{x}.dirs.quota
```

其中，x 代表缓存层级，一般而言可以设置 3 层存储，分别对应 MEM，SSD，HDD等存储介质。以 2 层存储，存储介质分别为 MEM，SSD，每一层分配100GB存储为例，对应的配置信息如下：



```
goosefs.worker.tieredstore.levels=2
goosefs.worker.tieredstore.level0.alias=MEM
goosefs.worker.tieredstore.level0.dirs.path=/data/GooseFSWorker
goosefs.worker.tieredstore.level0.dirs.mediumtype=MEM
goosefs.worker.tieredstore.level0.dirs.quota=100GB
goosefs.worker.tieredstore.level1.alias=SSD
goosefs.worker.tieredstore.level1.dirs.path=/data1/GooseFSWorker
goosefs.worker.tieredstore.level1.dirs.mediumtype=SSD
goosefs.worker.tieredstore.level1.dirs.quota=100GB
```

GooseFS 支持配置多层存储，层数没有限制，但是每一层的命名（alias）均需要保证唯一性。一般来说，提供 3 层缓存，每一层分别对应 MEM，SSD，HDD就起到较好的分层效果。

2. 缓存淘汰策略

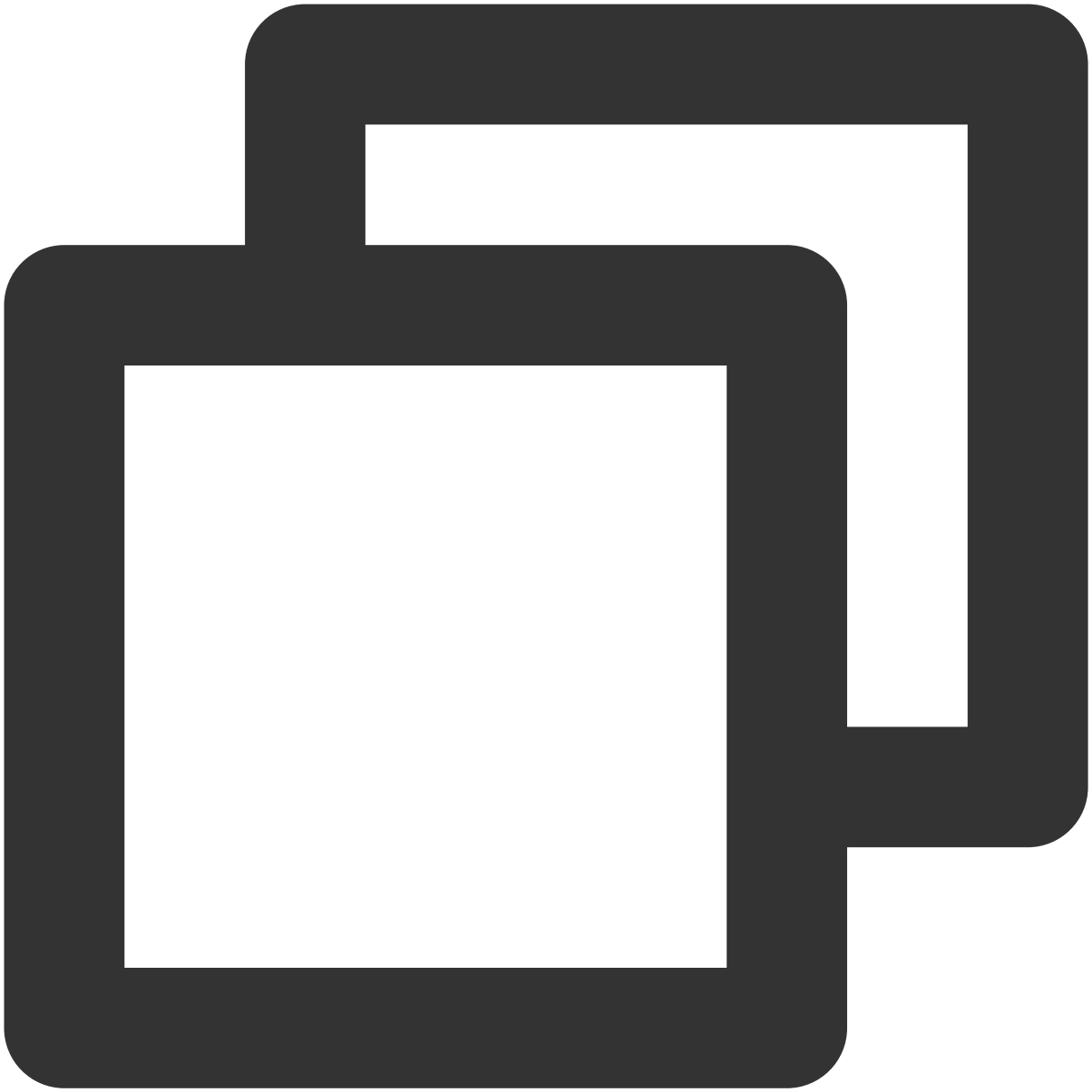
GooseFS 提供了两种缓存淘汰策略：

LRUAnnotator：按照近期使用次数最少（least-recently-used）的顺序淘汰缓存，是 GooseFS 的默认策略。

LRFUAnnotator：按照近期使用次数最少（least-recently-used）和近期使用频次最少（least-frequently-used）的顺序淘汰缓存，通过权重配置 goosefs.worker.block.annotator.lrfu.step.factor 和 goosefs.worker.block.annotator.lrfu.attenuation.factor 来调整两种策略在淘汰过程中起的作用。

如果将 least-recently-used 的权重调整至最大，那么该策略的表现与 LRUAnnotator一致。

可以在 goosefs.worker.block.annotator.class 这一配置项中配置缓存淘汰策略，配置时需要正确指定策略名称：



```
goosefs.worker.block.annotator.LRUAnnotator
goosefs.worker.block.annotator.LRFUAnnotator
```

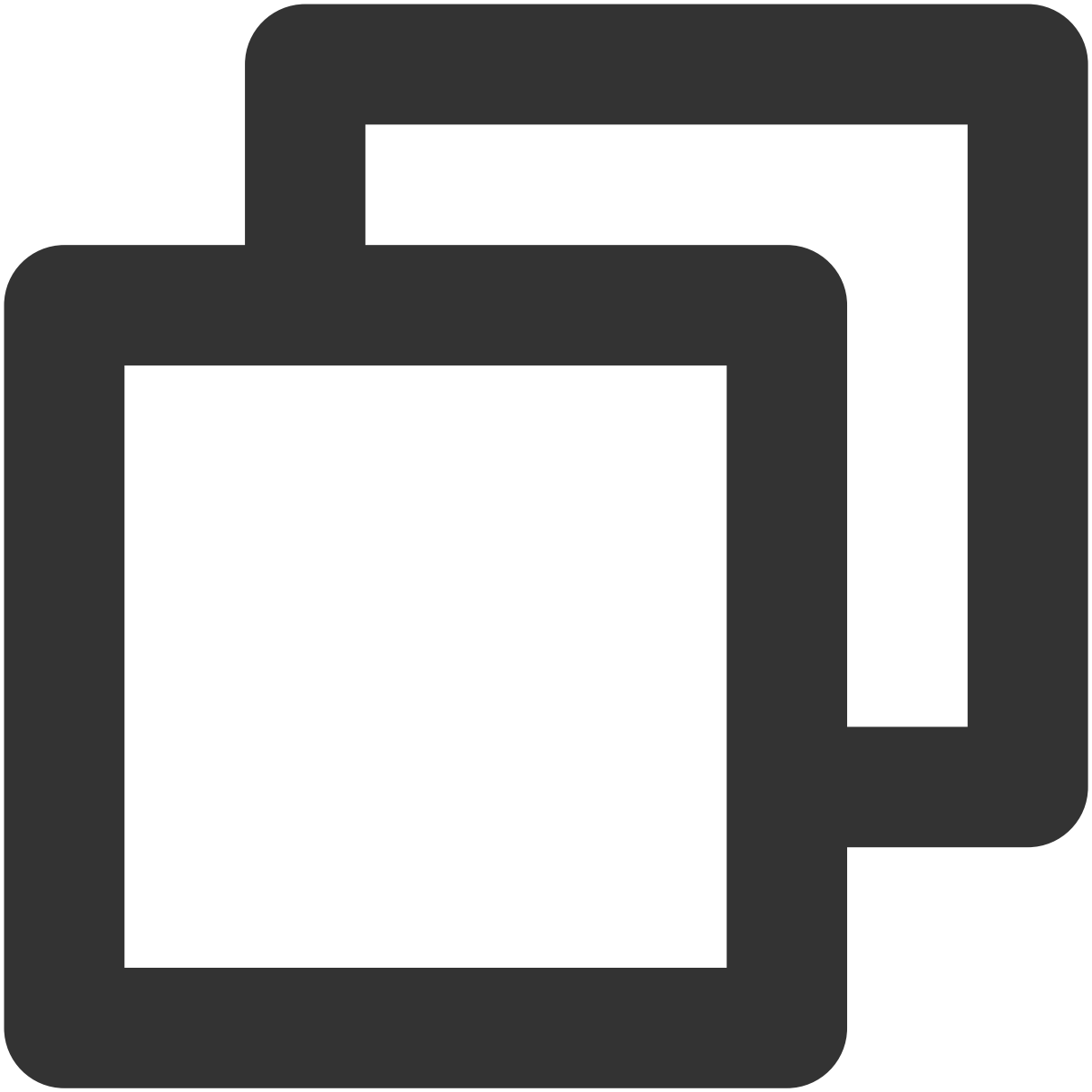
数据生命周期管理

这里的生命周期是指在 GooseFS 中的数据生命周期，而不是远端存储系统 UFS 的，生命周期管理主要有如下四种操作：

free：此操作用于从 GooseFS 中删除对应目录或者文件的数据（不会删除 UFS 中的对应数据），此类操作主要是用来释放 GooseFS 的缓存空间，供其他更热的数据使用。
load：此操作用于将 UFS 对应目录或者文件的数据加载到 GooseFS 中，此类操作主要用于冷数据转热，从而提高 I/O 性能。
persist：此操作用于将 GooseFS 内的数据写入到 UFS 存储中，此类操作主要用于持久化数据，从而避免数据丢失。
TTL(Time to Live)：TTL 属性主要用于设置目录或者文件在 GooseFS 中的生命周期，在超过其生命周期后，会从 GooseFS 和 UFS 中删除。通过配置，也可支持仅删除 GooseFs 数据或仅释放磁盘空间。

1. 从 GooseFS 中释放数据

通过 free 指令可以从 GooseFS 中释放数据，如下示例展示了释放数据后，GooseFS 中数据的状态变成 0%：



```
$ goosefs fs free /data/test.txt
/data/test.txt was successfully freed from GooseFS space.
$ goosefs fs ls /data/test.txt
-rw-rw-rw-  hadoop         hadoop                       14       PERSISTED 03-11-202
```

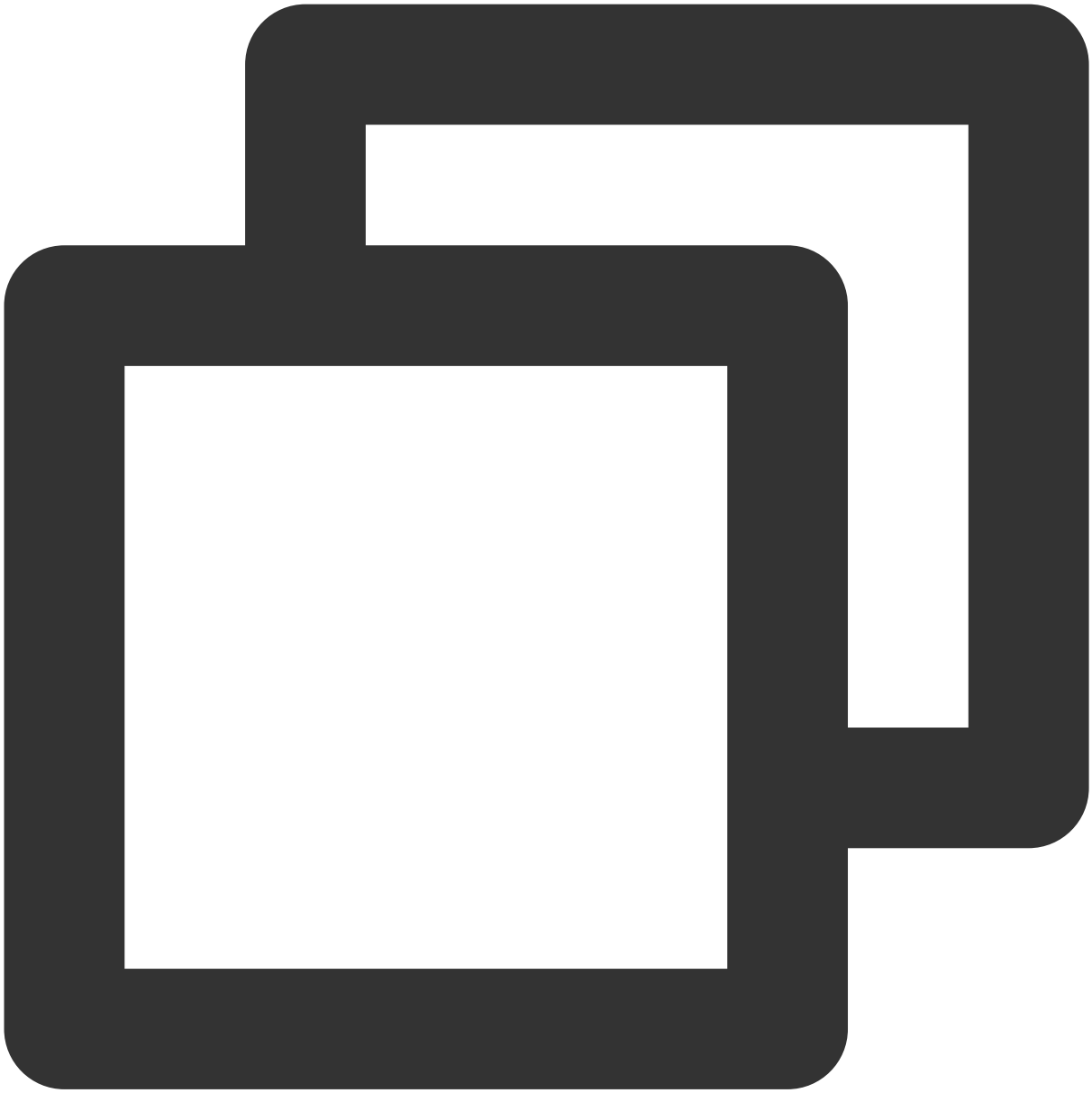
示例中的 /data/test.txt 可以是任何合法的 GooseFS 文件路径，如果数据存储于远端存储 UFS 中，则可以通过 load 指令重新加载。

注意

一般而言，推荐通过配置缓存策略自动清理 GooseFS 中的历史数据。

2. 往 GooseFS 中加载数据

通过 load 指令可以从 GooseFS 中加载数据，如下示例展示了释放数据后，GooseFS 中数据的状态变成 100% ：



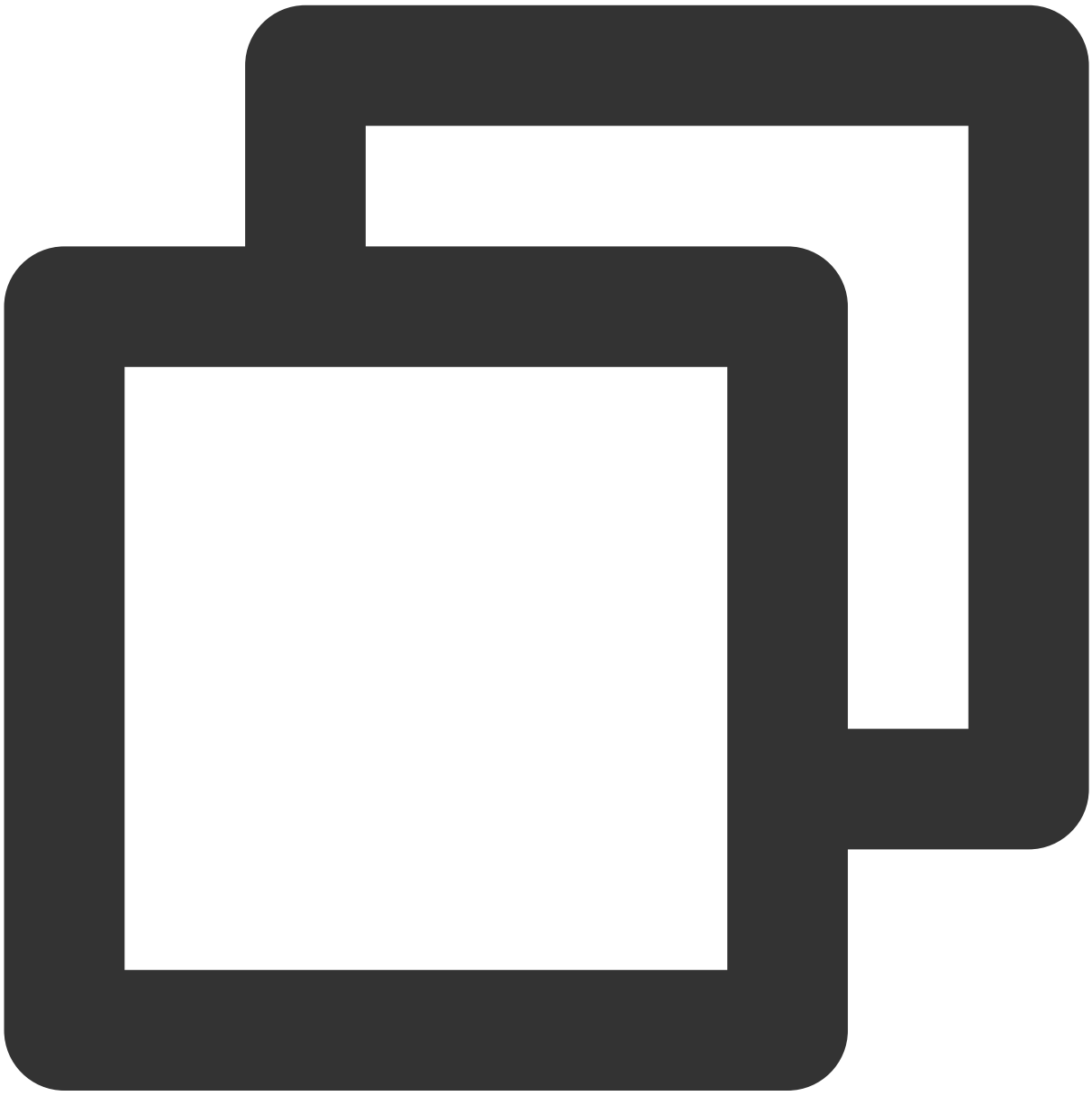
```
$ goosefs fs load /data/test.txt
/data/test.txt loaded
$ goosefs fs ls /data/test.txt
-rw-rw-rw-  hadoop         hadoop                        14       PERSISTED 03-11-202
```

示例中的 /data/test.txt 可以是任何合法的 GooseFS 文件路径。如果是从本地文件系统中加载数据，则需要使用 copyFromLocal 指令，需要注意的是，从本地文件系统中加载数据的操作并不会将数据持久化到远端存储 UFS 中。

在默认情况下， GooseFS 能在首次访问文件时自动加载数据到 GooseFS 到缓存中，因此一般无需使用该指令加载数据；但如果业务需要提前加载数据到缓存中，可以通过该指令加载。

3. 在 GooseFS 中持久化数据

通过 persist 指令可以将数据持久化到远端存储系统 UFS 中：

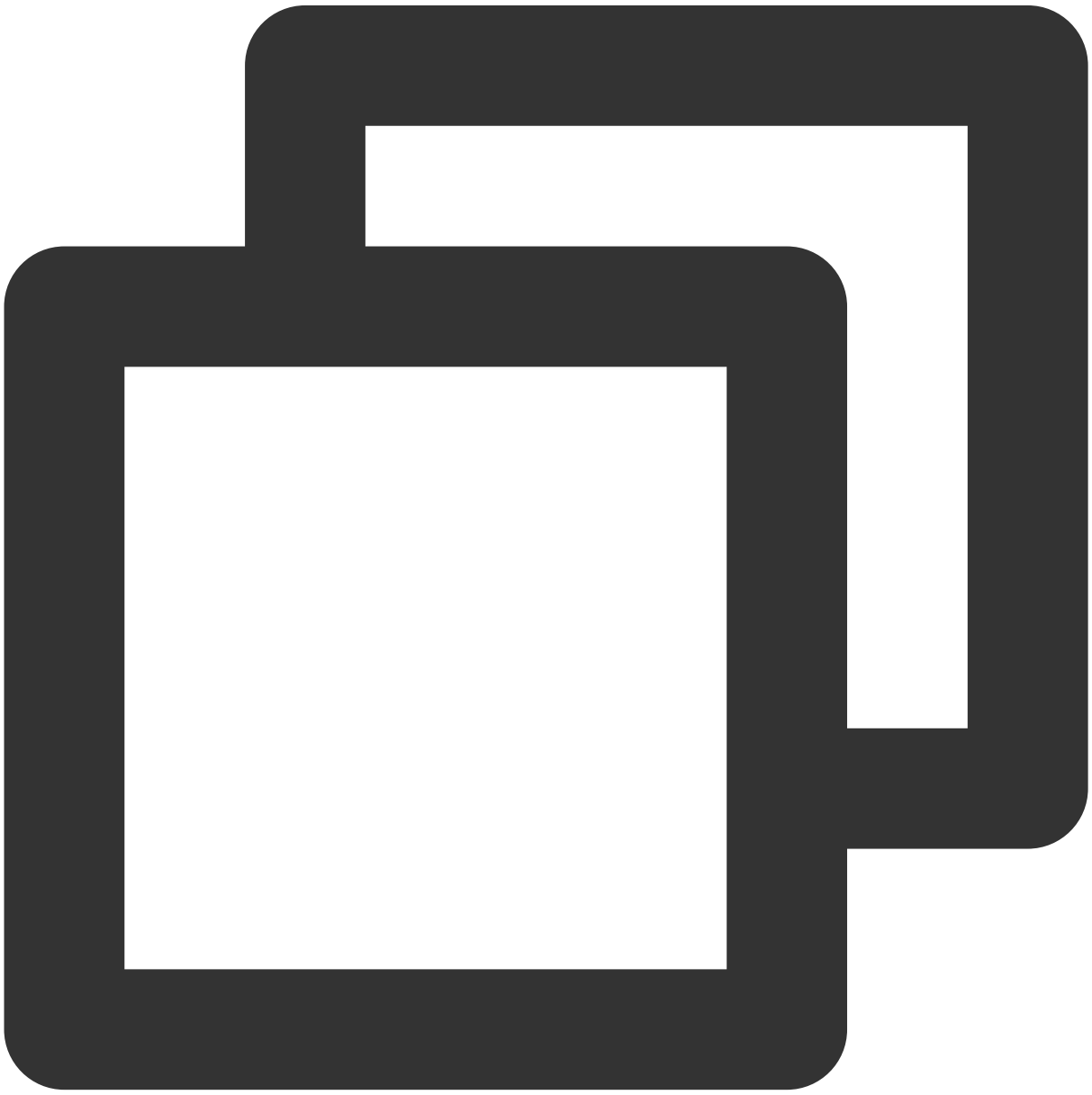


```
$ goosefs fs persist /data/test.txt
$ goosefs fs ls /data/test.txt
-rw-rw-rw-  hadoop         hadoop                        14       PERSISTED 03-11-202
```

示例中的 /data/test.txt 可以是任何合法的 GooseFS 文件路径。推荐通过缓存策略配置来自动执行持久化的操作。

4. 设置数据 TTL 属性

GooseFS 支持为命名空间中的任意文件或者目录设置 TTL 属性。该属性可以保证业务数据可以按照指定的时间周期定期删除，为新文件释放空间，保证本地磁盘空间的有效利用。 GooseFS 文件和目录的 TTL 属性可以作为文件元数据的一部分，可以保证在集群重启后 TTL 属性依然生效，后端线程的定期巡检程序会检查这些 TTL 属性并自动清理过期文件。
定期巡检程序的巡检周期和巡检类型可以在 goosefs-site.preperties 中设置，以下示例将巡检周期设置为 10 分钟，巡检类型设置为 FREE ：



```
goosefs.master.ttl.checker.interval=10m
goosefs.user.file.create.ttl.action=FREE
```

设置完毕后，GooseFS 后台线程会每隔 10 分钟巡检一遍，当次巡检周期内发现的过期文件会在下一次巡检周期过完后释放缓存资源。如 00:00:00 进行了第一次巡检发现一批过期文件，这批文件缓存会在 00:10:00 清除。

注意

默认情况下该入参单位为毫秒（ms），可以在入参时指定好检测时间周期的单位，如秒（s）,分钟（m），小时（h）等，更多说明可以查看配置说明的介绍。

数据复制管理

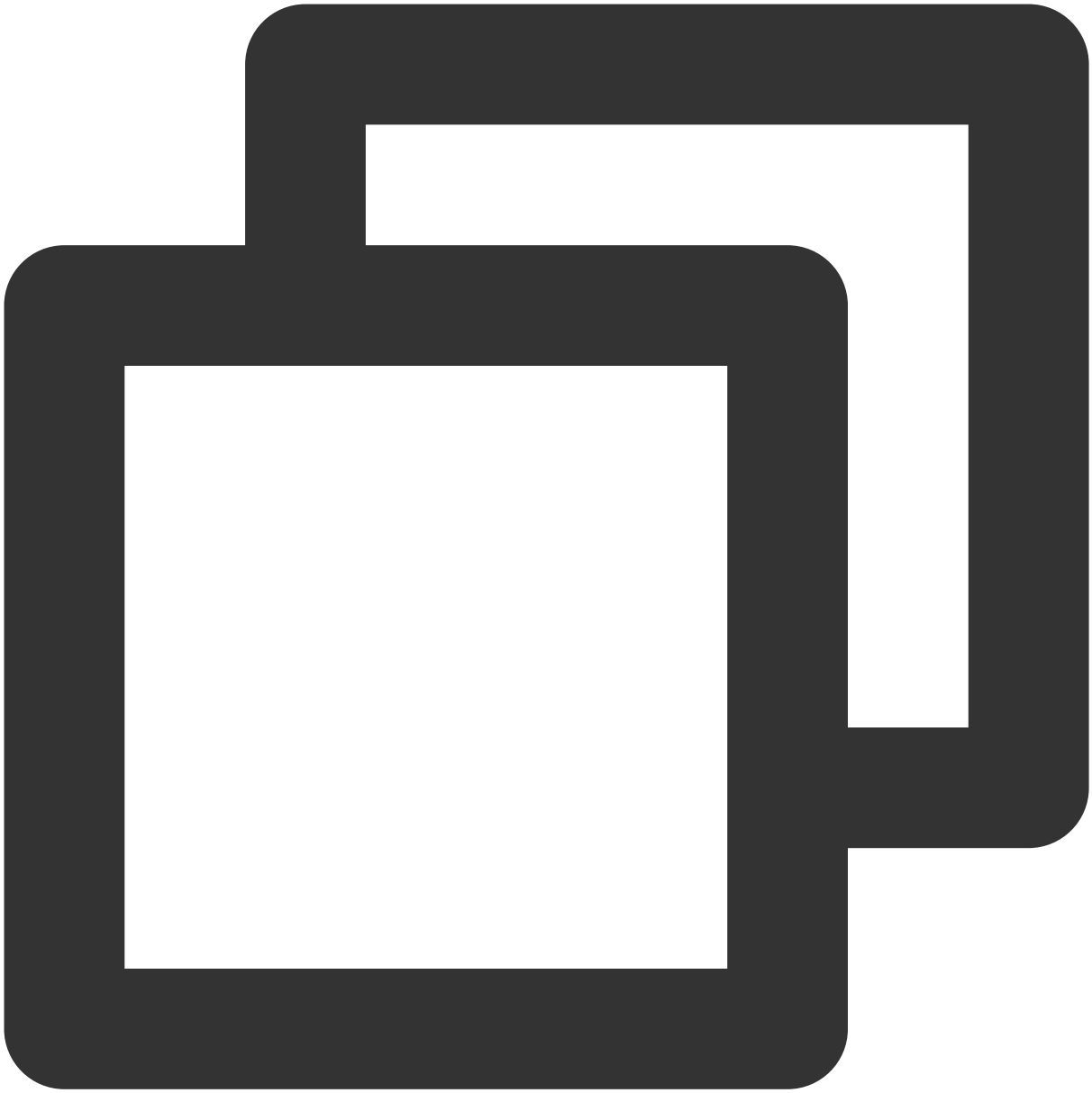
数据复制主要分为被动复制和主动复制两种形式。

1. 被动复制

GooseFS中的每个文件都包含一个或多个分布在集群中存储的存储块，默认情况下 GooseFS 可以根据负载和容量情况自动调整数据分块的复制级别。被动复制会发生在以下场景：
多个客户端同时读取相同的块，会导致多个 block 同时在不同的 worker 上存在。
当开启优先本地读时，数据不在本地，则会发起 remote read，并保存在本地 worker 上。
当被动复制产生的副本数大于设定的副本数目时，其会异步的去删除多余的副本。以上过程对用户完全透明。

2. 主动复制

主动复制是通过设置文件的副本数目实现的，对于不满足设定副本数的 block，其会异步的补足，对于超过设定副本数的 block，其会删除多余的副本。具体的，可以通过如下指令调整主动复制的级别：



```
$ goosefs fs setReplication  [-R] [--max | --min ] <path>
```

相关参数说明如下：

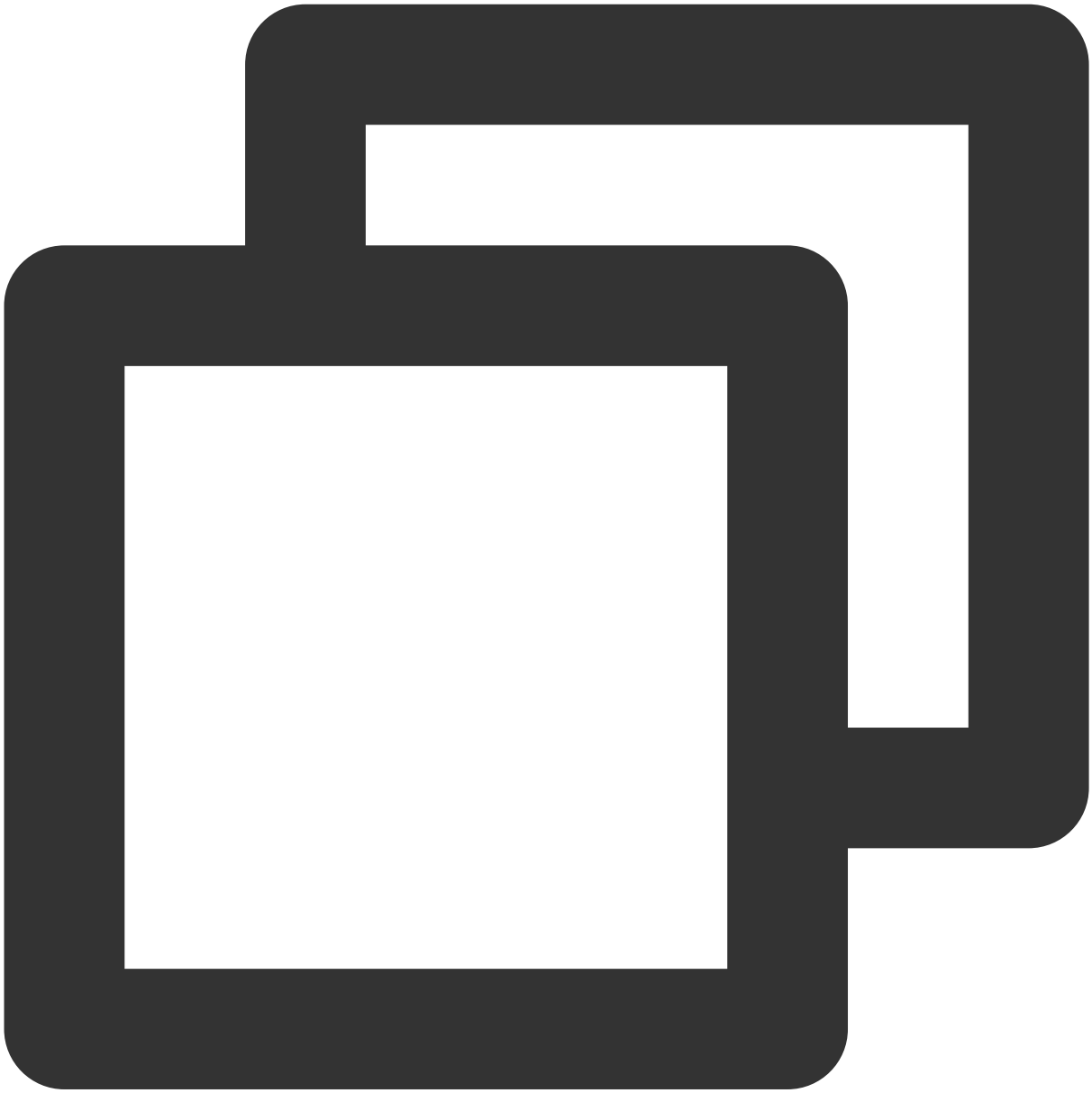
max：指定文件的最大副本数，默认值为 -1，表示不设置上限；如果设置为 0 代表不在 GooseFS 中存储指定文件的冷数据。一般设置为正整数，设置后 GooseFS 会检查文件副本数量并删除多余的副本。

min：指定文件的最小副本数，默认值为0，代表 GooseFS 会在数据变冷后删除该数据不留存任何副本。一般设置为正整数，并且**需要小于 max 值**，设置后 GooseFS 会检查文件副本数量，如果副本数量比最小值小会自动补齐副本数。

path：指定的文件名，可以为目录或者具体的文件路径。

R：如果 path 中入参为目录，则指定 -R 可以按照指定的最小副本数和最大副本数，重复递归复制指定目录下的所有文件和子目录。

如果需要查看指定文件的复制情况，可以通过 stat 指令查看，如下示例展示 /data/test.txt 这个文件的最大副本数为 -1，意味着该文件变冷后会被过期删除：

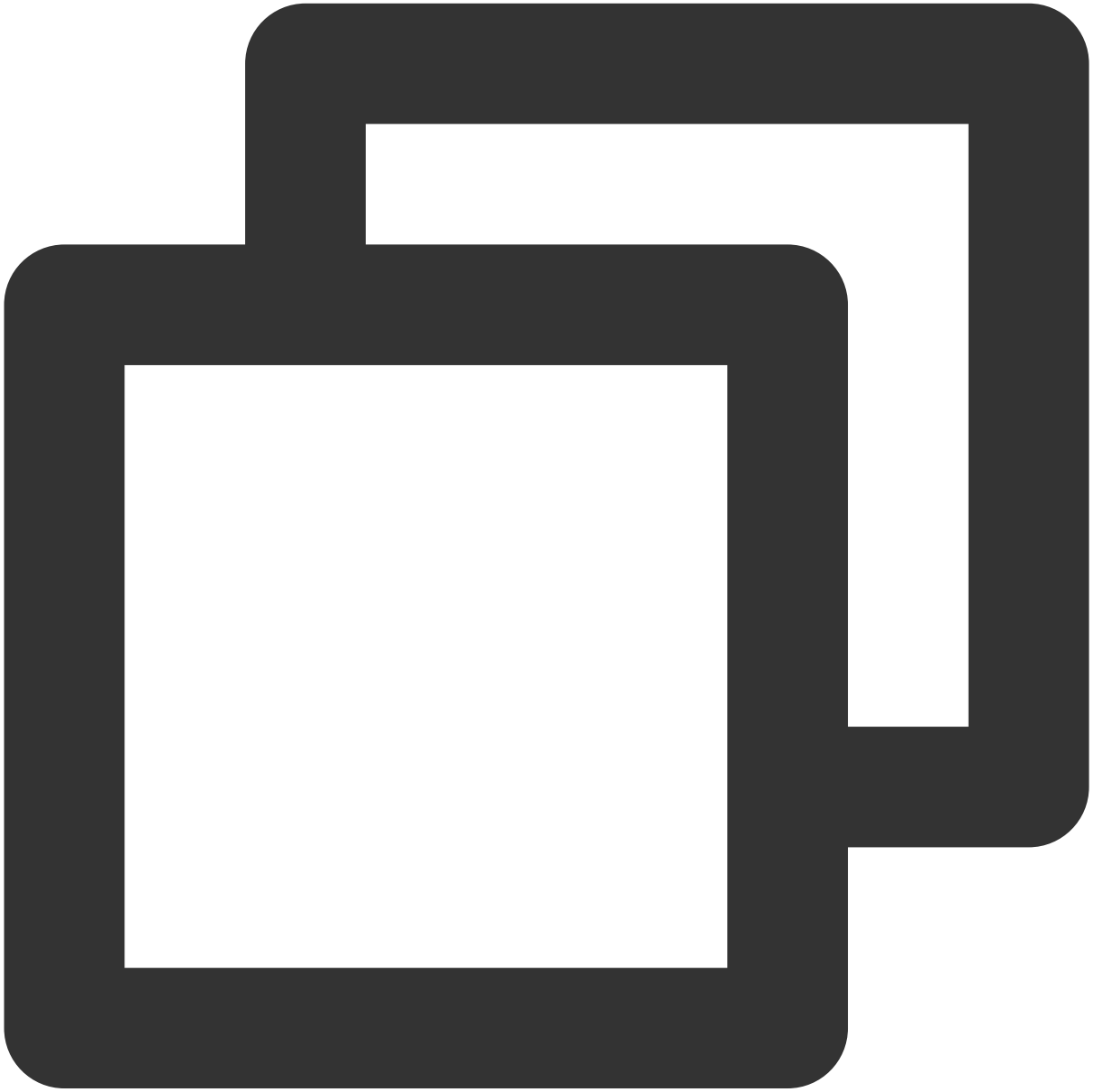


```
$ goosefs fs stat /data/test.txt
/data/test.txt is a file path.
FileInfo{fileId=50331647, fileIdentifier=null, name=test.txt, path=/data/test.txt,
This file does not contain any blocks.
```

查看缓存用量

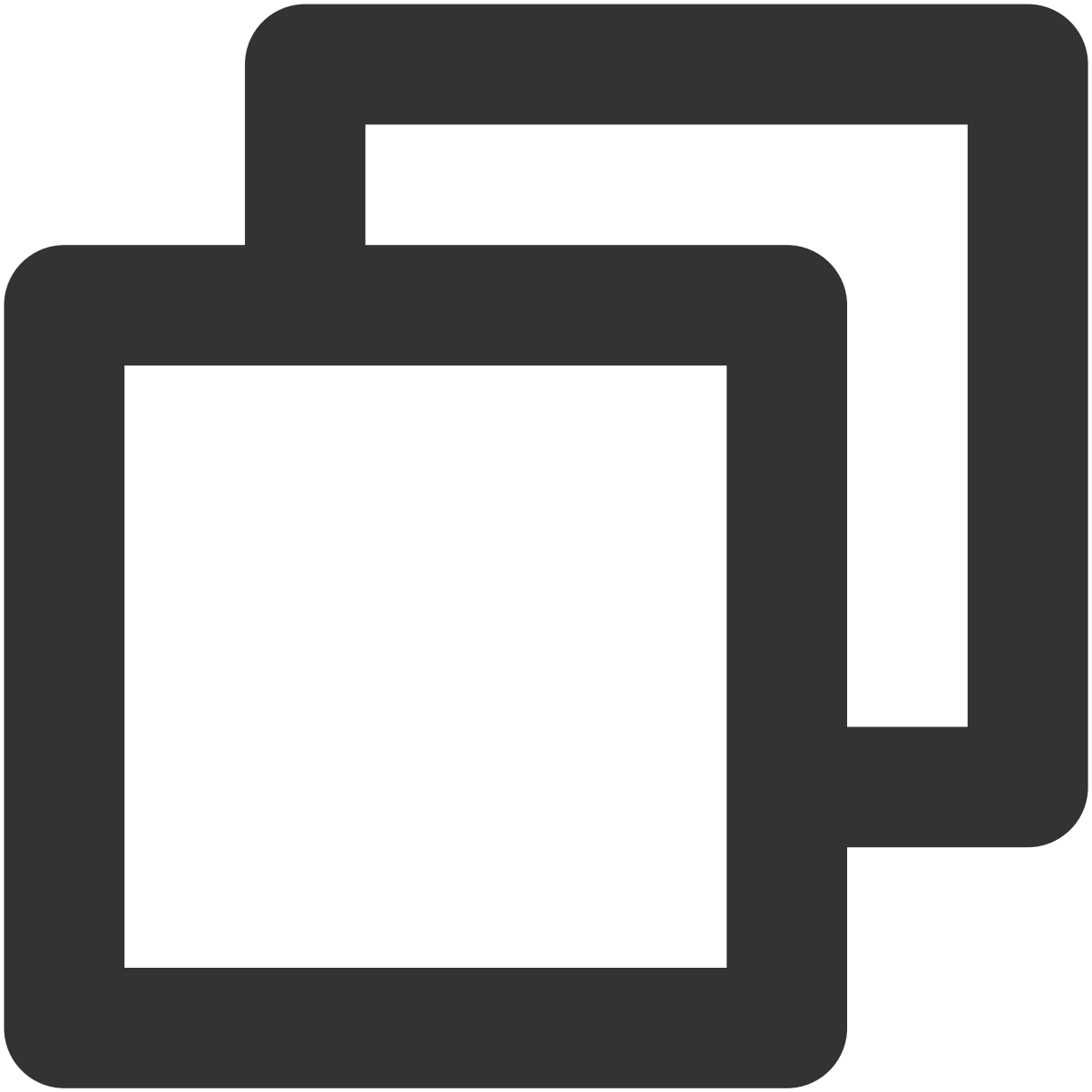
GooseFS 会记录本地缓存的容量和使用情况，可以通过如下指令查看 GooseFS 的运行情况，针对性地管理和维护本地缓存。

查看 GooseFS 正在使用的缓存，单位为字节：



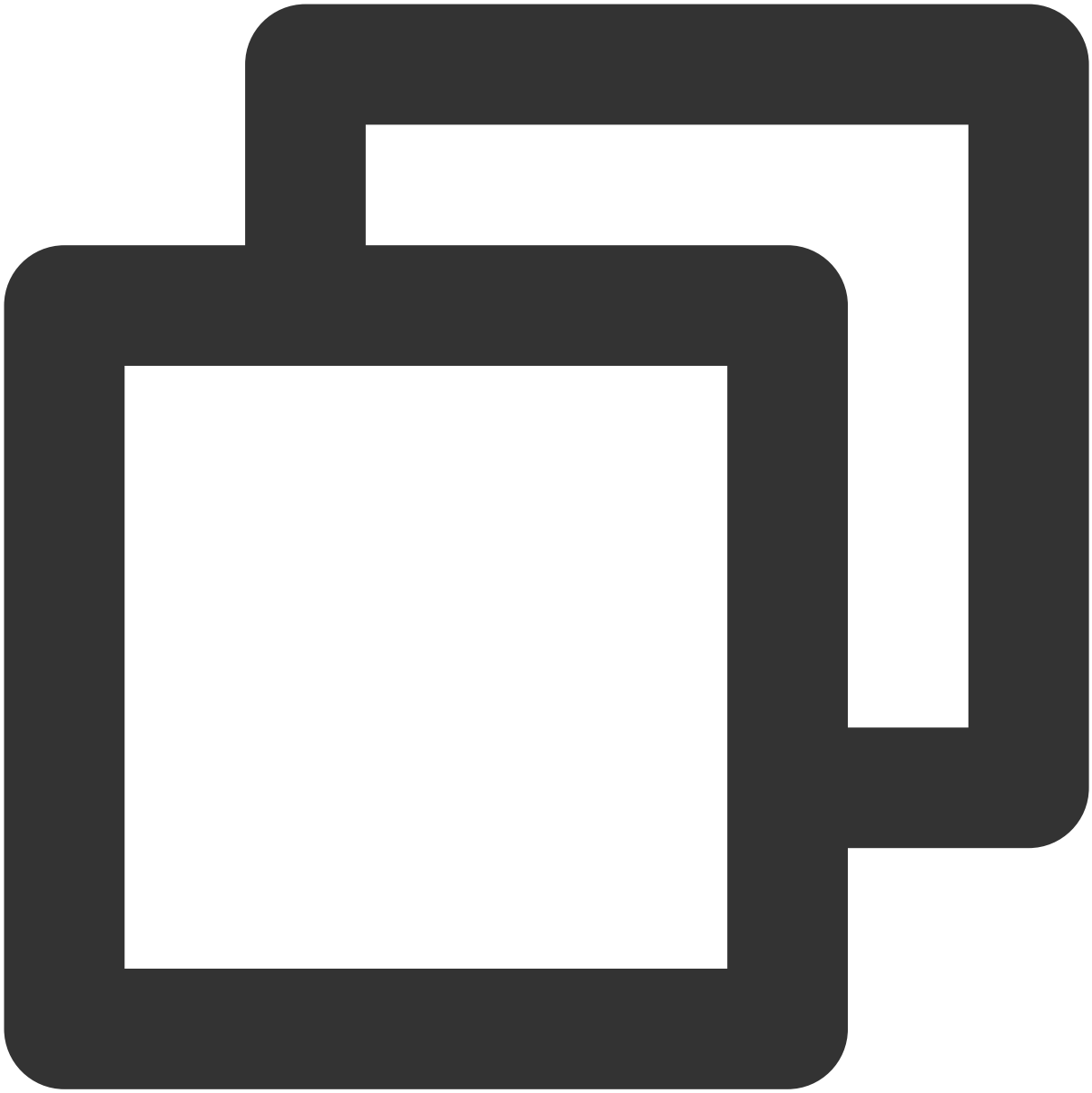
```
$ goosefs fs getUsedBytes
Used Bytes: 0
```

查看 GooseFS 总缓存容量，单位为字节：



```
$ goosefs fs getCapacityBytes
Capacity Bytes: 1610612736000
```

查看 GooseFS 的缓存使用报告：



```
$ goosefs fsadmin report
GooseFS cluster summary:
    Master Address: 172.16.16.16:19998
    Web Port: 19999
    Rpc Port: 19998
    Started: 04-12-2021 10:52:05:255
    Uptime: 0 day(s), 1 hour(s), 28 minute(s), and 57 second(s)
    Version: 2.5.0-SNAPSHOT
    Safe Mode: false
    Zookeeper Enabled: false
    Live Workers: 3
```

```
    Lost Workers: 0
    Total Capacity: 1500.00GB
        Tier: HDD  Size: 1500.00GB
    Used Capacity: 0B
        Tier: HDD  Size: 0B
    Free Capacity: 1500.00GB
```

透明加速能力

最近更新时间：2024-01-06 11:17:58

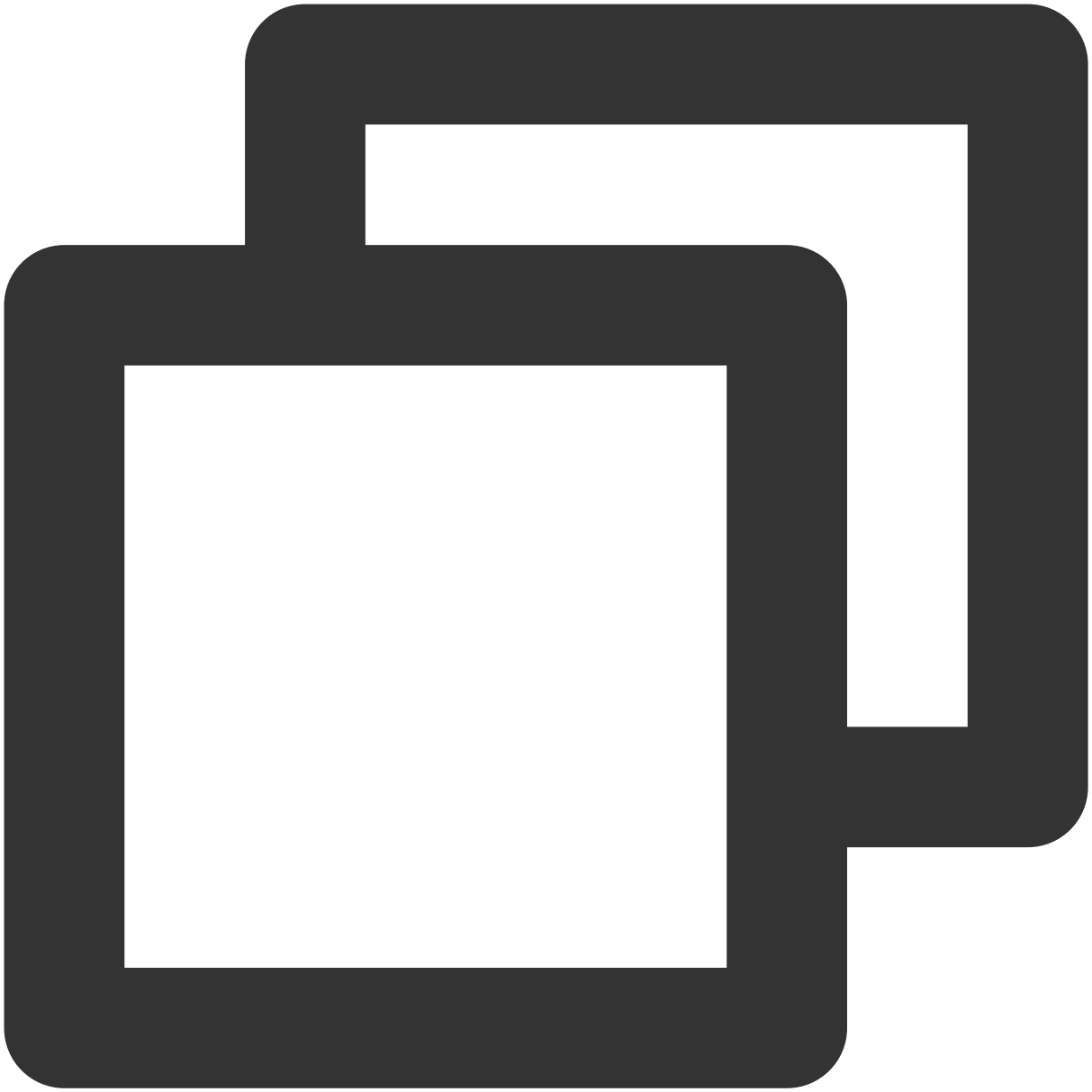
概述

透明加速能力用于加速 CosN 访问 COS 的性能。CosN 工具 是基于腾讯云对象存储（Cloud Object Storage, COS）提供的标准的 Hadoop 文件系统实现，可以为 Hadoop、Spark 以及 Tez 等大数据计算框架集成 COS 提供支持。用户可使用实现了 Hadoop 文件系统接口的 CosN 插件，读写存储在 COS 上的数据。对于已经使用 CosN 工具访问 COS 的用户，GooseFS 提供了一种客户端路径映射方式，让用户可以在不修改当前 Hive table 定义的前提下，仍然能够使用 CosN scheme 访问 GooseFS，该特性方便用户在不修改已有表定义的前提下，对 GooseFS 的功能和性能进行对比测试。对于云 HDFS 用户（CHDFS），也可以通过修改配置，实现使用 OFS Scheme 访问 GooseFS 的目的。

CosN Schema 和 GooseFS Schema 的映射说明如下：

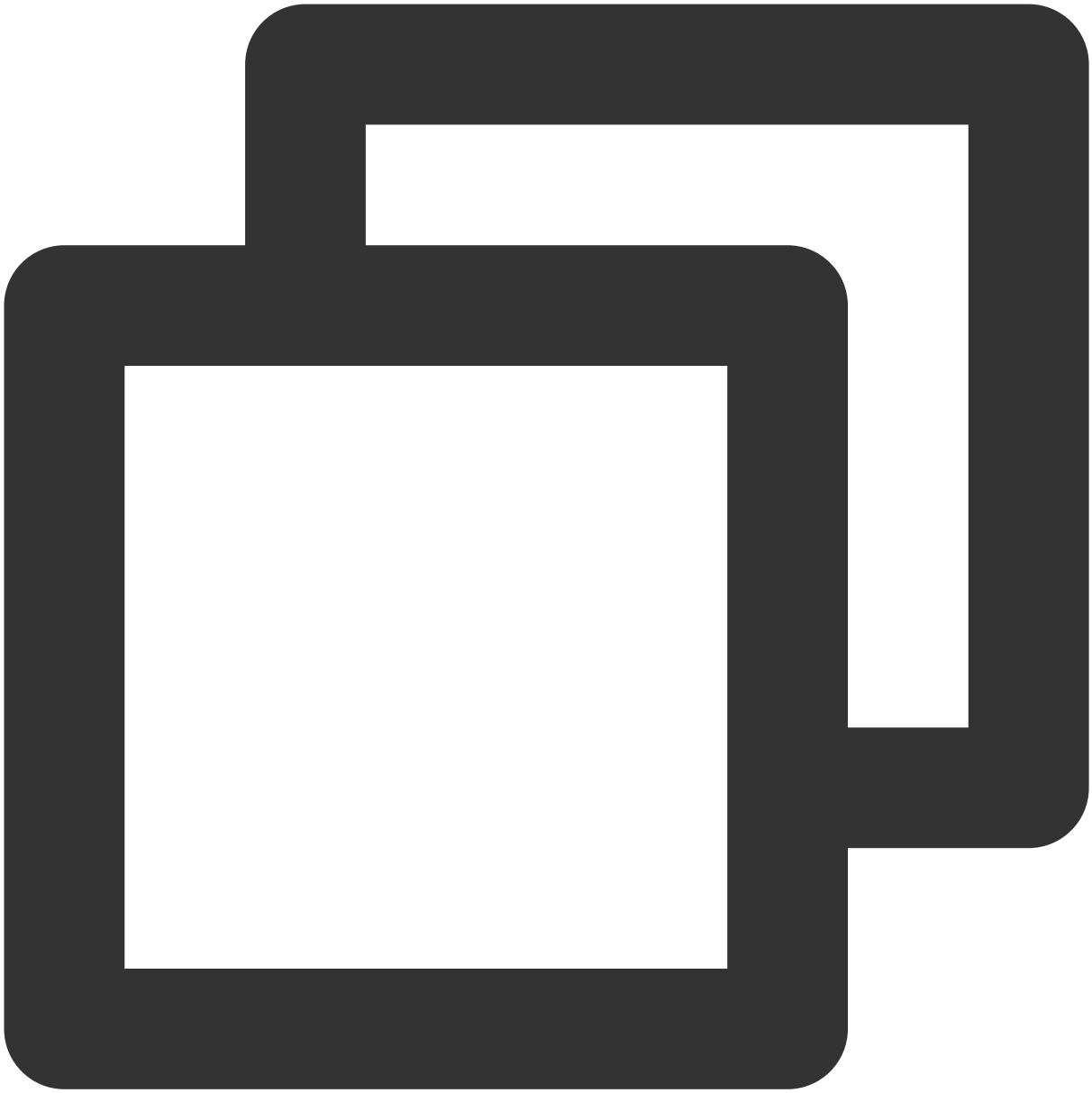
假设 Namespace warehouse 对应的 UFS 路径为 `cosn://examplebucket-1250000000/data/warehouse/`，则 CosN 到 GooseFS 的路径映射关系如下：

版权所有：腾讯云计算（北京）有限责任公司



```
cosn://examplebucket-1250000000/data/warehouse -> /warehouse/
cosn://examplebucket-1250000000/data/warehouse/folder/test.txt ->/warehouse/folder/
```

GooseFS 到 CosN 的路径映射关系如下：



```
/warehouse ->cosn://examplebucket-1250000000/data/warehouse/
/warehouse/ -> cosn://examplebucket-1250000000/data/warehouse/
/warehouse/folder/test.txt -> cosn://examplebucket-1250000000/data/warehouse/folder
```

CosN Scheme 访问 GooseFS 特性，通过在客户端维持 GooseFS 路径和底层文件系统 CosN 路径之间的映射关系，并将 CosN 路径的请求转换为 GooseFS 的请求。映射关系周期性刷新，您可以通过修改 GooseFS 配置文件 goosefs-site.properties 中的配置项 goosefs.user.client.namespace.refresh.interval 调整刷新间隔，默认值为60s。

注意

如果访问的 CosN 路径无法转换为 GooseFS 路径，对应的 Hadoop API 调用会抛出异常。

操作示例

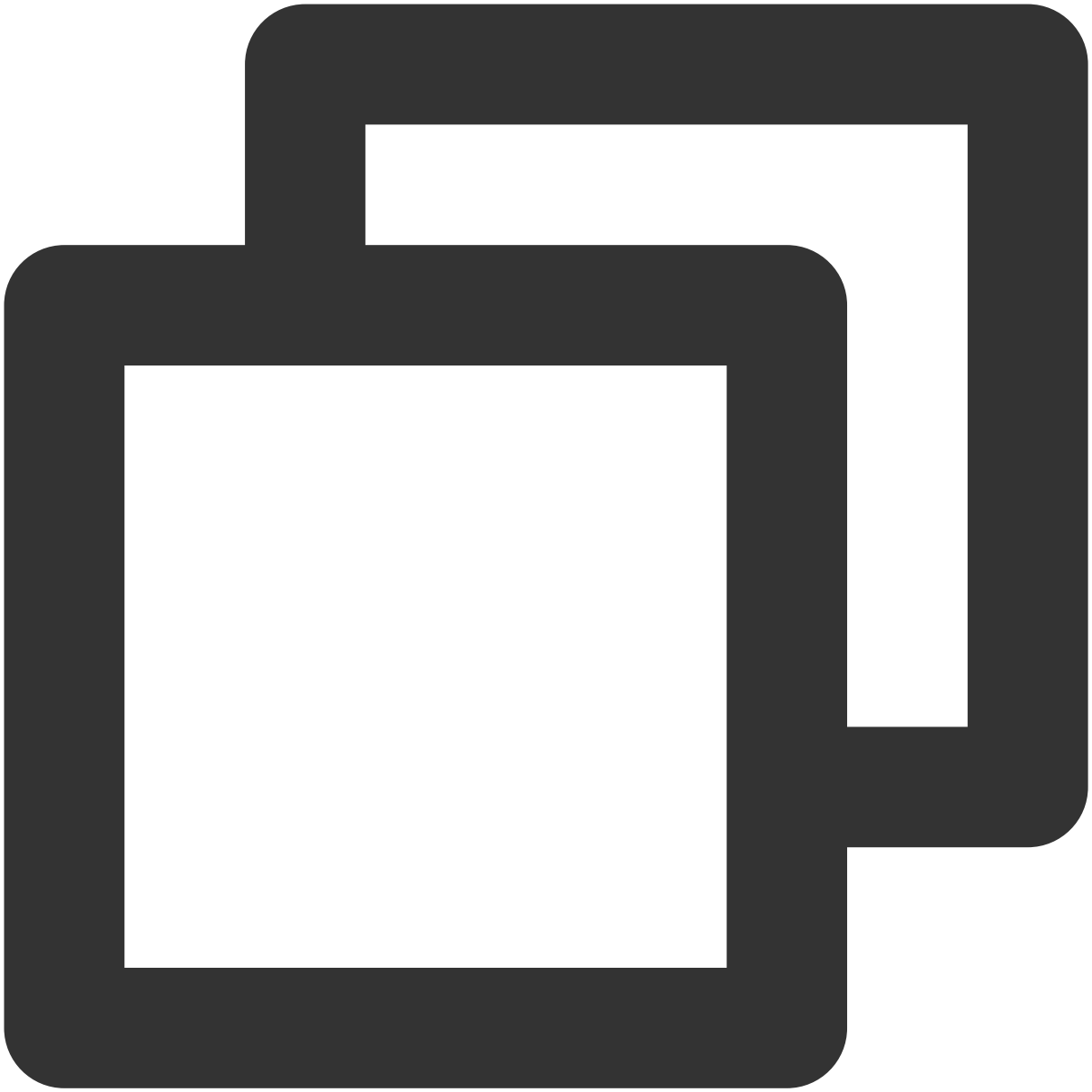
该示例演示了 Hadoop 命令行以及 Hive 中，如何使用 gfs://、 cosn://、ofs:// 三种 Schema 访问 GooseFS。操作流程如下：

1. 准备数据和计算集群

参考 创建存储桶 文档，创建一个测试用途的存储桶。
参考 创建文件夹 文档，在存储桶根路径下创建一个名为 ml-100k 的文件夹。
从 Grouplens 下载 ml-100k 数据集，并将文件 u.user 上传到 `<存储桶根路径>/ml-100k` 下。
参考 EMR 指引文档，购买一个 EMR 集群并配置 HIVE 组件。

2. 环境配置

i. 将 GooseFS 的客户端 jar 包（goosefs-1.0.0-client.jar）放入 share/hadoop/common/lib/ 目录下：

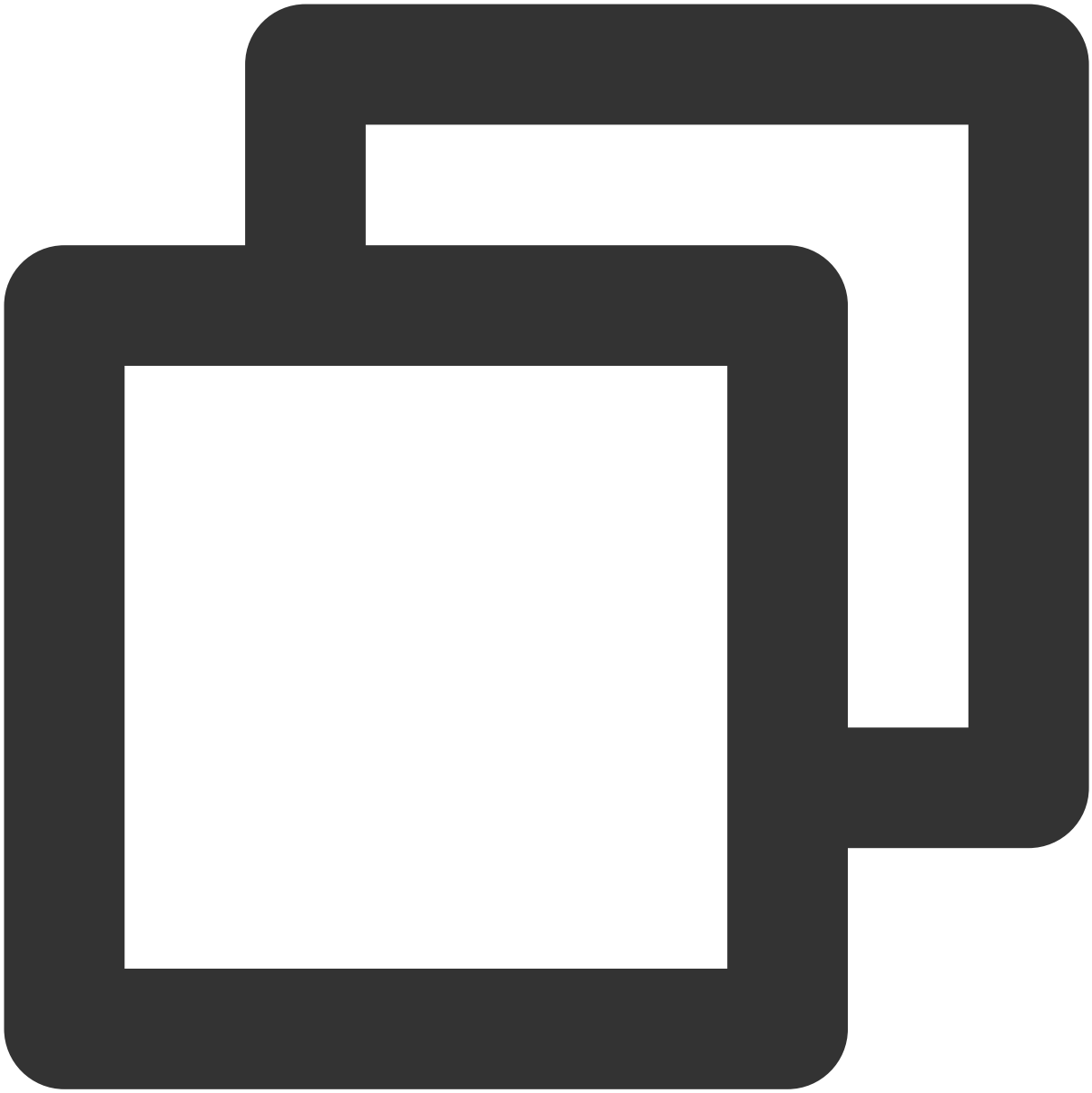


```
cp goosefs-1.0.0-client.jar  hadoop/share/hadoop/common/lib/
```

注意

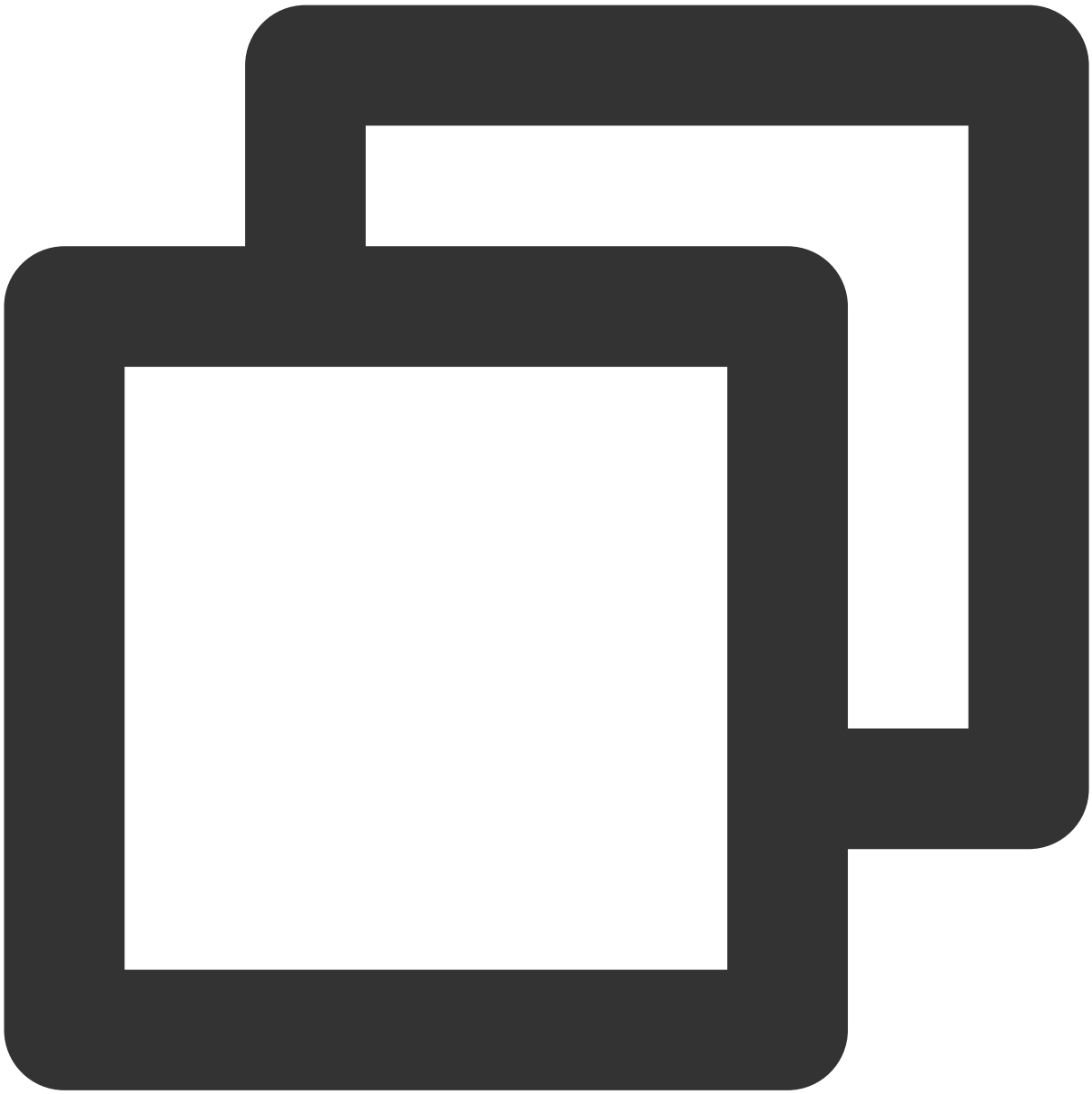
配置变更和添加 jar 包，需同步到集群上所有节点。

ii. 修改 Hadoop 配置文件 etc/hadoop/core-site.xml，指定 GooseFS 的实现类：



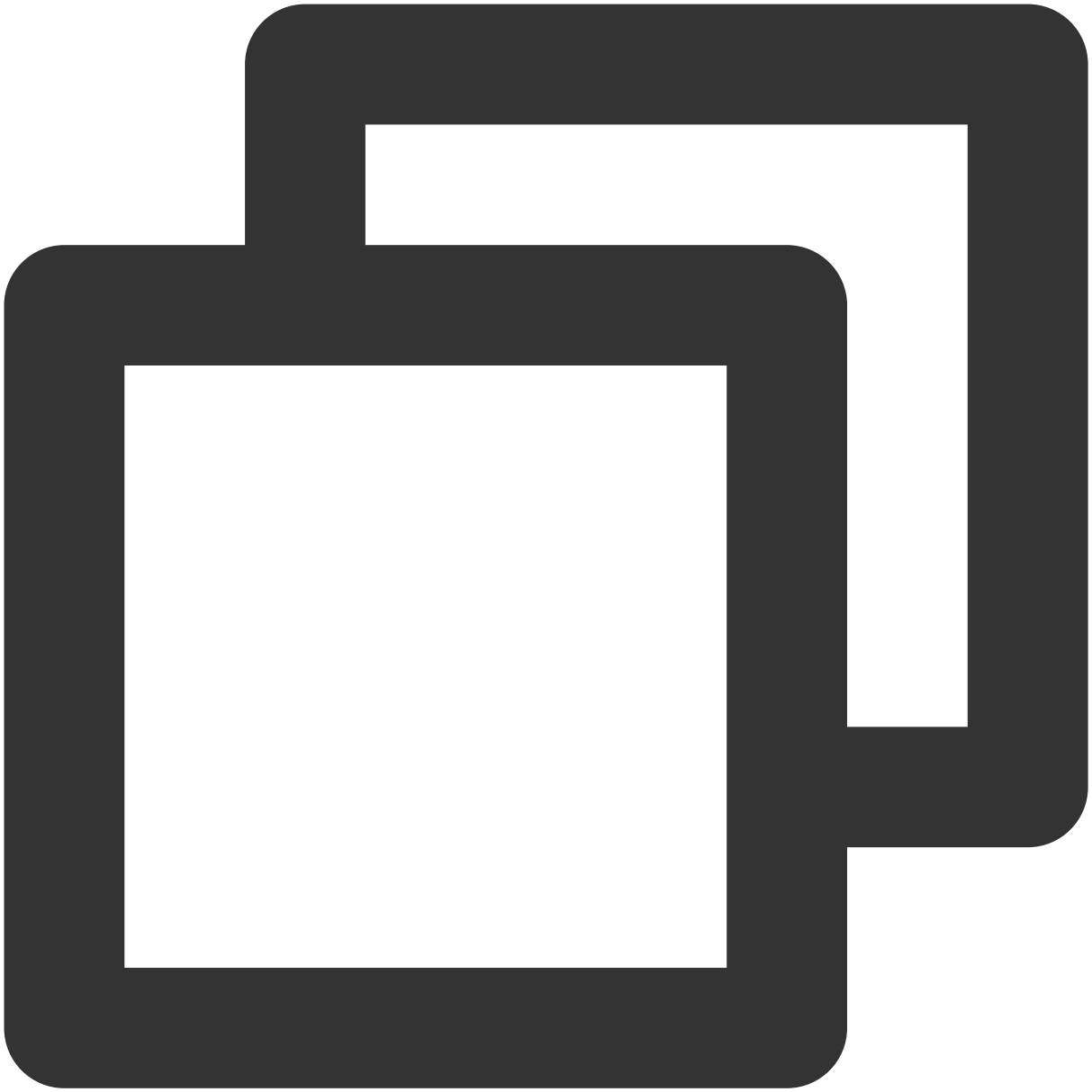
```
<property>
  <name>fs.AbstractFileSystem.gfs.impl</name>
  <value>com.qcloud.cos.goosefs.hadoop.GooseFileSystem</value>
</property>
<property>
  <name>fs.gfs.impl</name>
  <value>com.qcloud.cos.goosefs.hadoop.FileSystem</value>
</property>
```

iii. 执行如下 Hadoop 命令，检查是否能够通过 gfs:// Scheme 访问 GooseFS，其中 <MASTER_IP> 为 Master 节点的 IP：



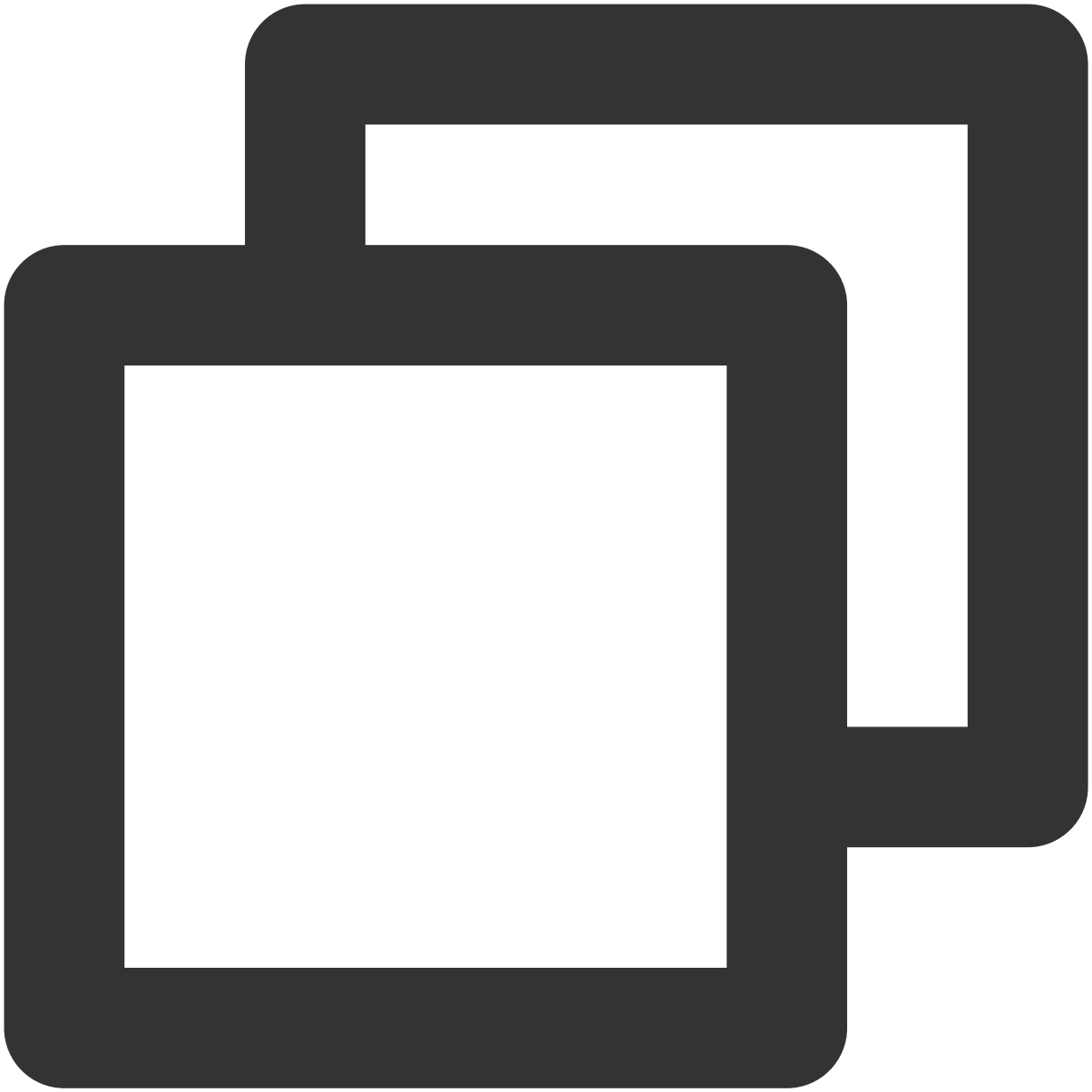
```
hadoop fs -ls gfs://<MASTER_IP>:9200/
```

iv. 将 GooseFS 的客户端 jar 包放到 Hive 的 auxlib 目录下，使得 Hive 能加载到 GooseFS Client 包：



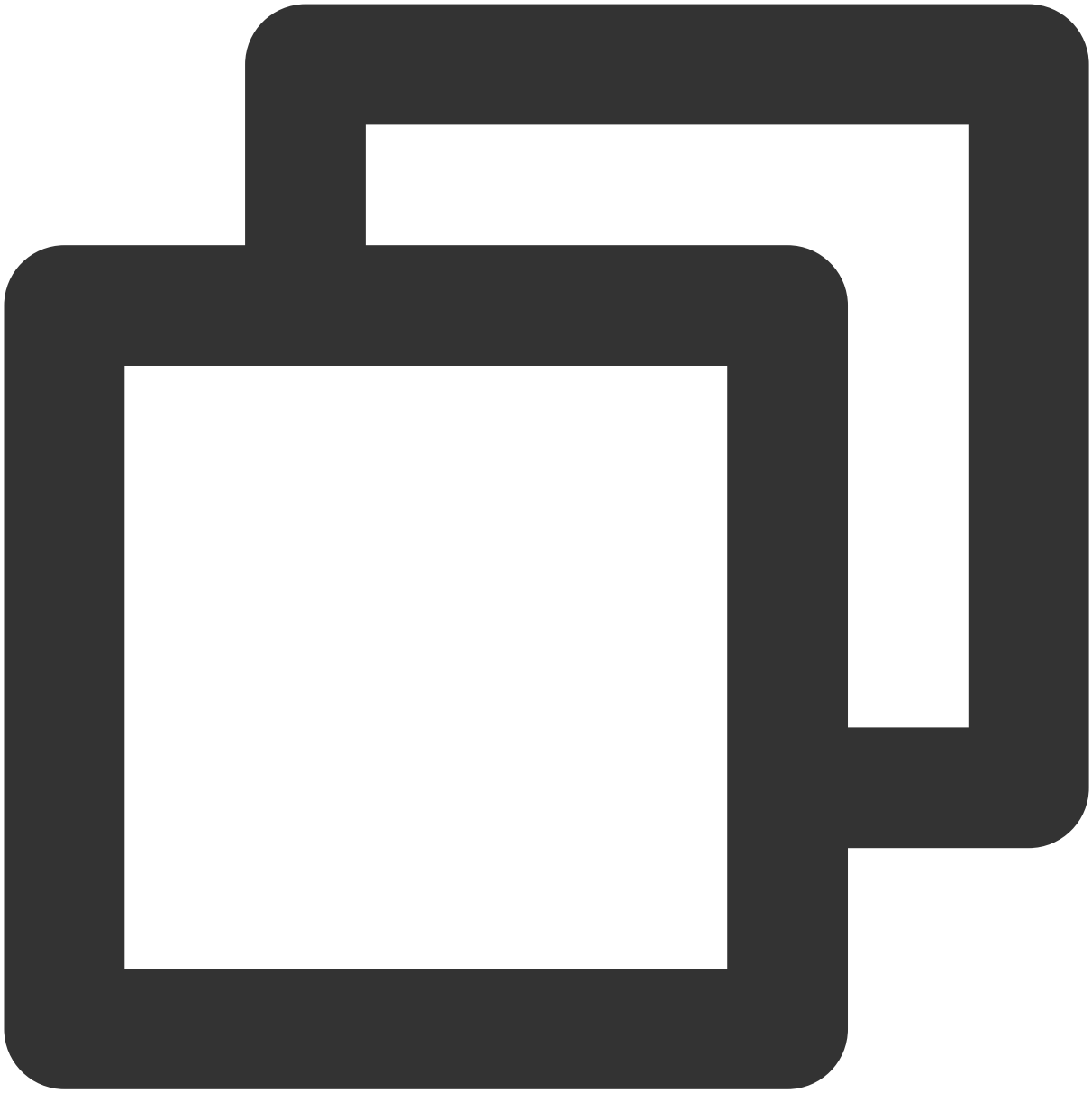
```
cp goosefs-1.0.0-client.jar  hive/auxlib/
```

v. 执行如下命令，创建 UFS Scheme 为 CosN 的 Namespace，并列出 Namespace。您可将该命令中的 examplebucket-1250000000 替换为你的 COS 存储桶，SecretId 和 SecretKey 替换为您的密钥信息：



```
goosefs ns create ml-100k cosn://examplebucket-1250000000/ml-100k  --secret fs.cosn
goosefs ns ls
```

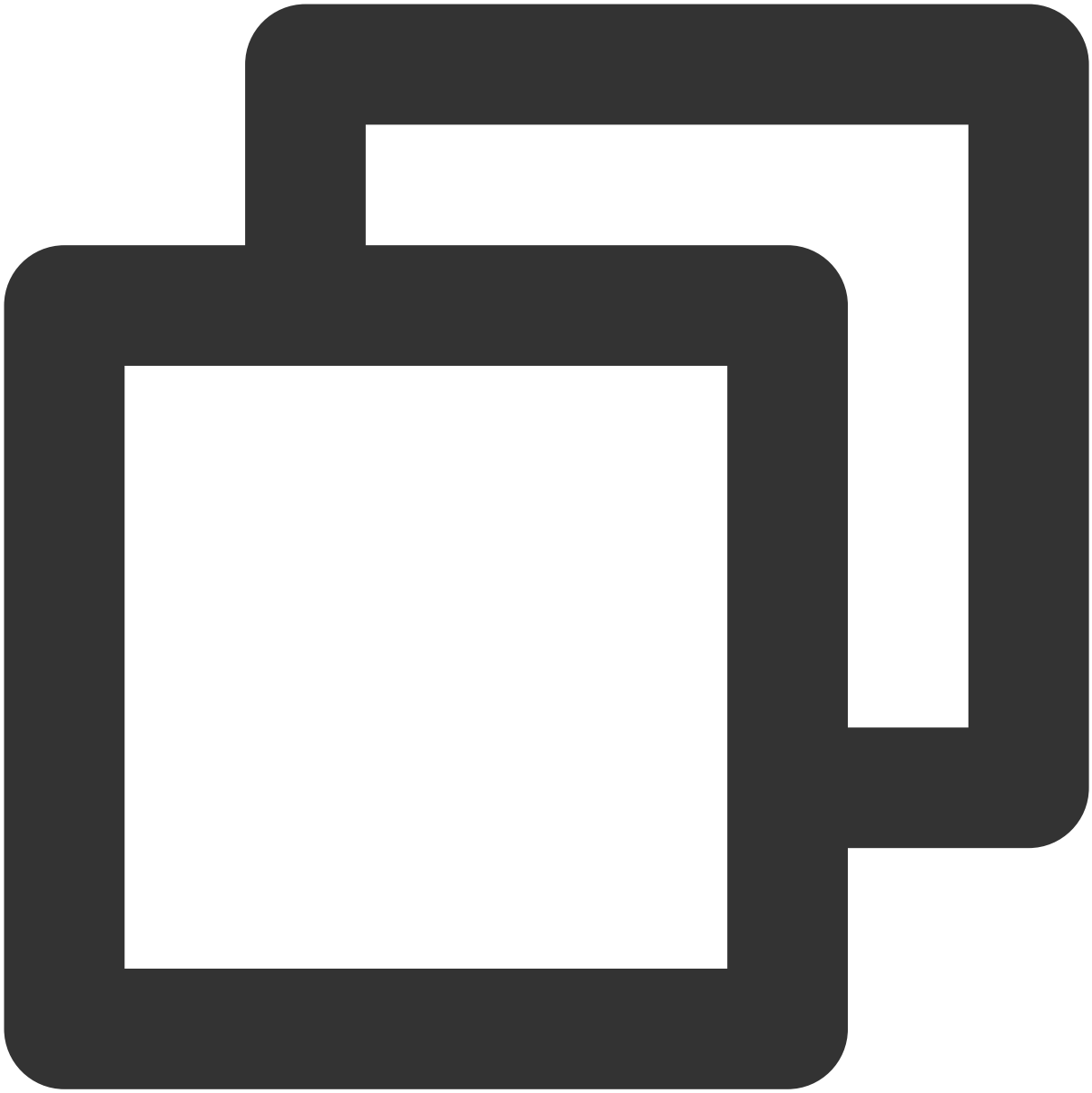
vi. 执行命令，创建 UFS Scheme 为 OFS 的 Namespace，并列出 Namespace。您可将该命令中的 instance-id 替换为您的 CHDFS 实例，1250000000 替换为您的 APPID：



```
goosefs ns create ofs-test  ofs://instance-id.chdfs.ap-guangzhou.myqcloud.com/ofs-t
goosefs ns ls
```

3. 创建 GooseFS Schema 表和查询数据

通过如下指令执行：

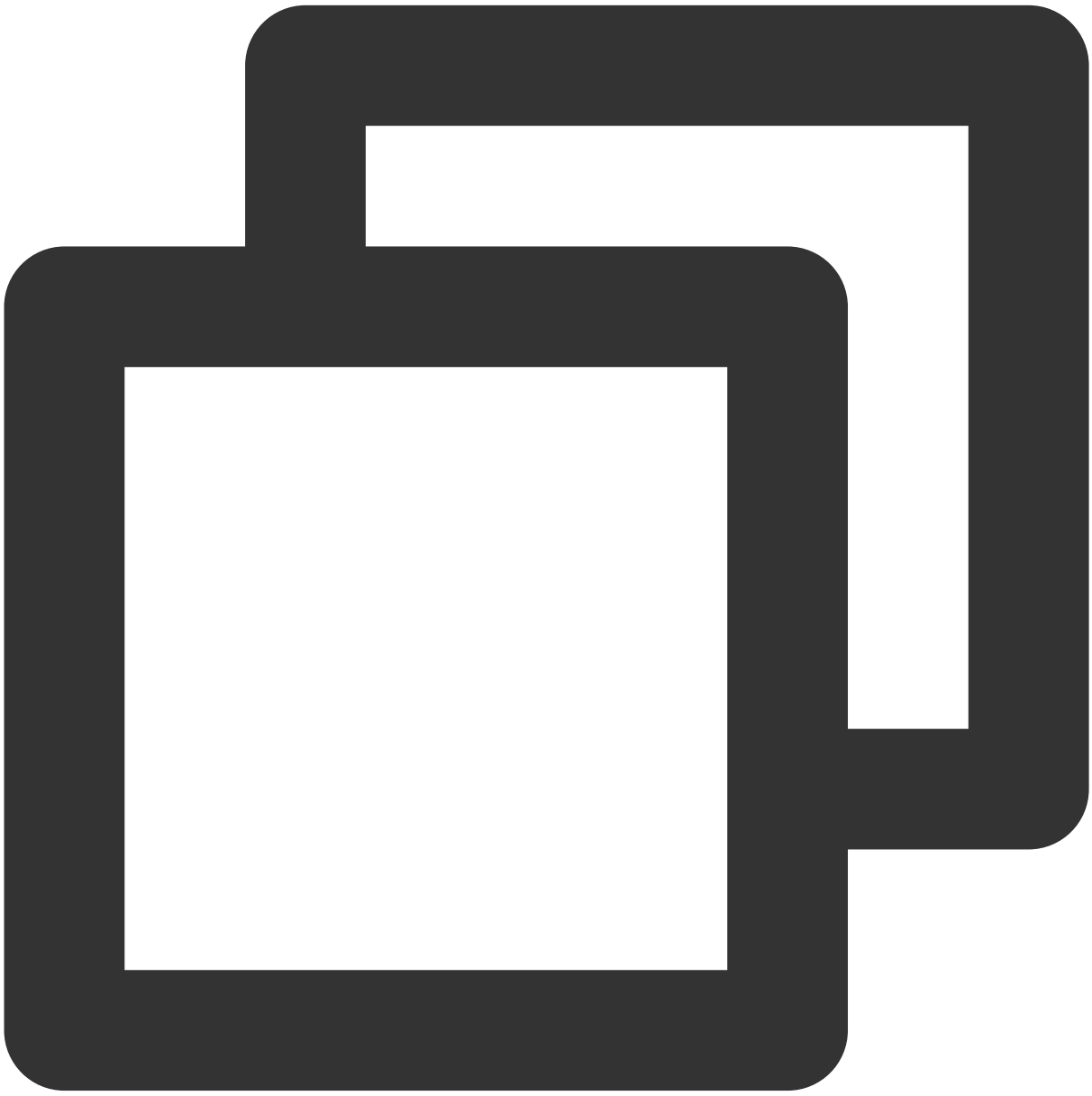


```
create database goosefs_test;

use goosefs_test;
CREATE TABLE u_user_gfs (
userid INT,
age INT,
gender CHAR(1),
occupation STRING,
zipcode STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
```

```
STORED AS TEXTFILE
LOCATION 'gfs://<MASTER_IP>:<MASTER_PORT>/ml-100k';

select sum(age) from u_user_gfs;
```

4. 创建 CosN Schema 表和查询数据

通过如下指令执行：

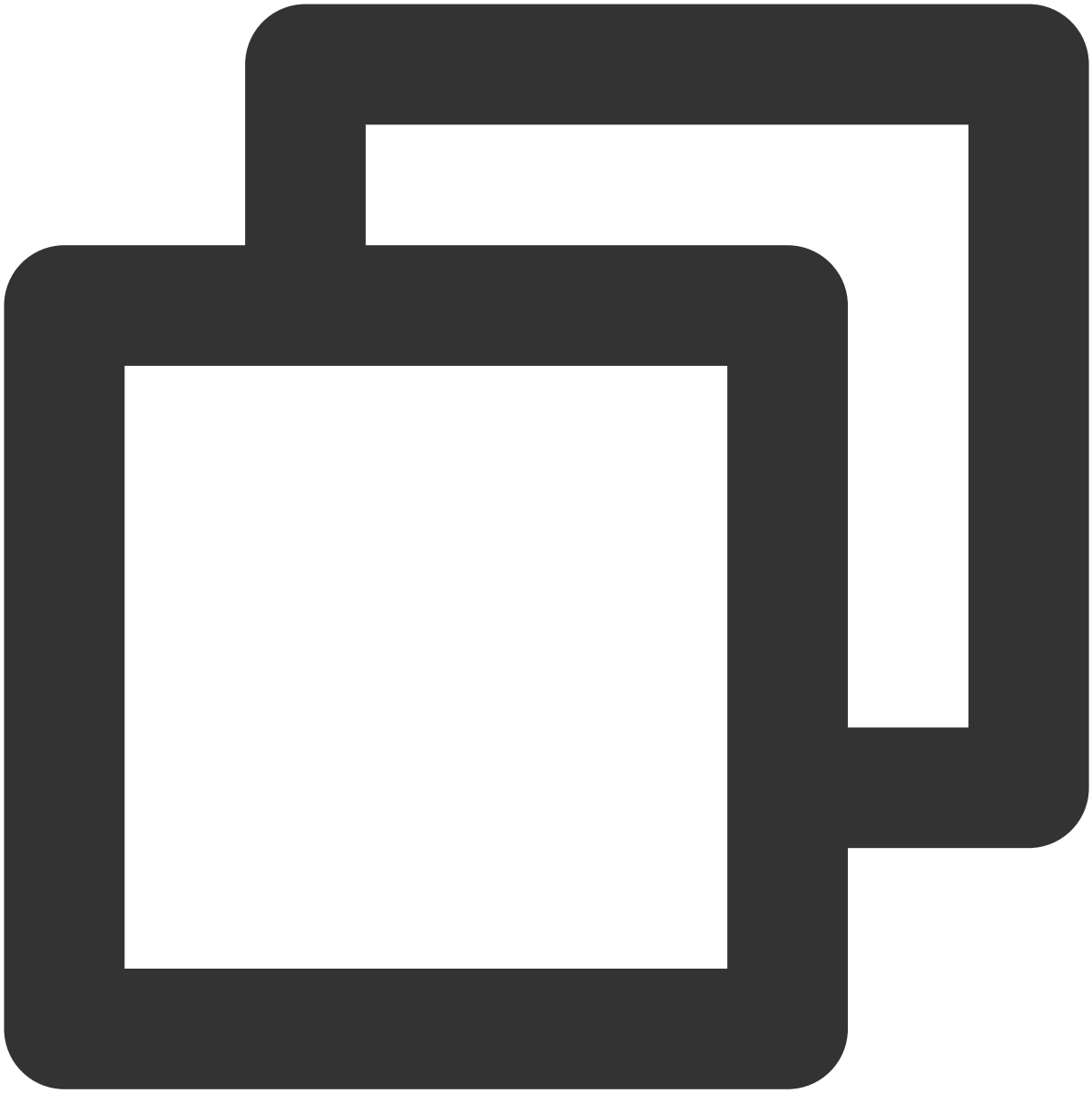


```
CREATE TABLE u_user_cosn (
userid INT,
```

```
age INT,
gender CHAR(1),
occupation STRING,
zipcode STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
STORED AS TEXTFILE
LOCATION 'cosn://examplebucket-1250000000/ml-100k';

select sum(age) from u_user_cosn;
```

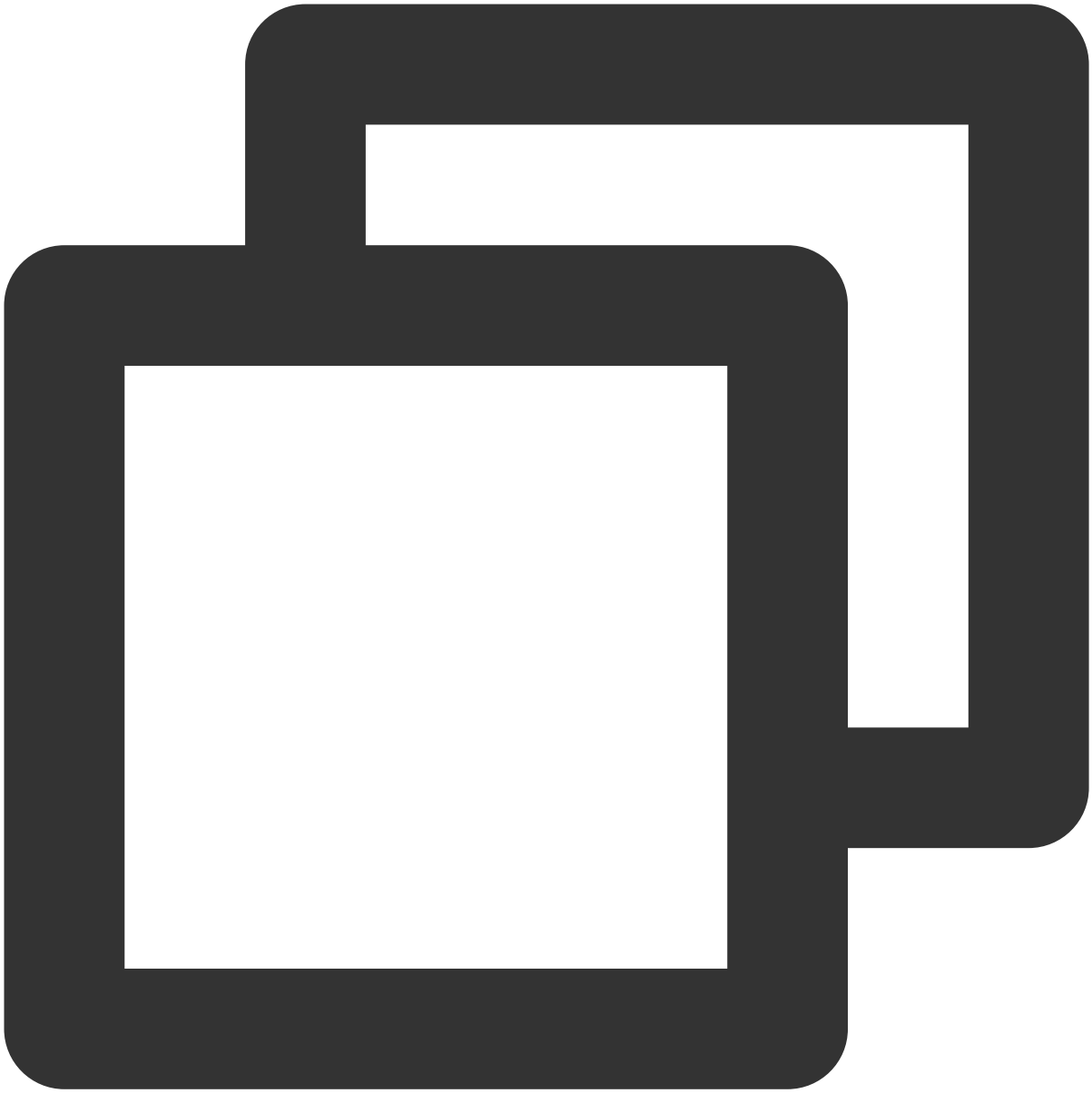
5. 修改 CosN 的实现为 GooseFS 的兼容实现

修改 hadoop/etc/hadoop/core-site.xml：



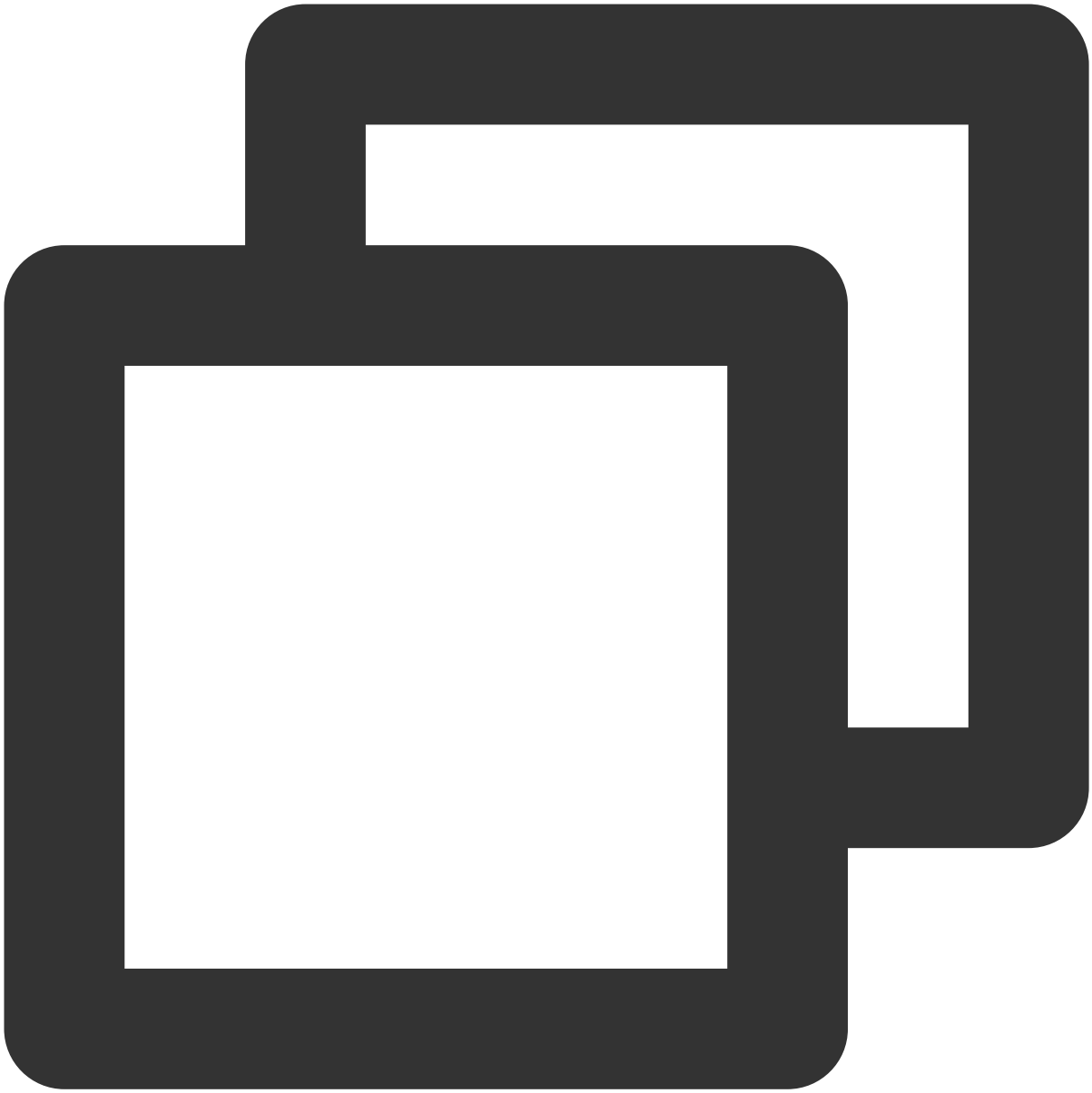
```
 <property>
     <name>fs.AbstractFileSystem.cosn.impl</name>
     <value>com.qcloud.cos.goosefs.hadoop.CosN</value>
</property>
<property>
     <name>fs.cosn.impl</name>
     <value>com.qcloud.cos.goosefs.hadoop.CosNFileSystem</value>
</property>
```

执行 Hadoop 命令，如果路径无法转换为 GooseFS 中的路径，命令的输出中会包含报错信息：



```
hadoop fs -ls  cosn://examplebucket-1250000000/ml-100k/

Found 1 items
-rw-rw-rw-   0 hadoop hadoop      22628 2021-07-02 15:27 cosn://examplebucket-12500

hadoop fs -ls cosn://examplebucket-1250000000/unknow-path
ls: Failed to convert ufs path cosn://examplebucket-1250000000/unknow-path to Goose
```

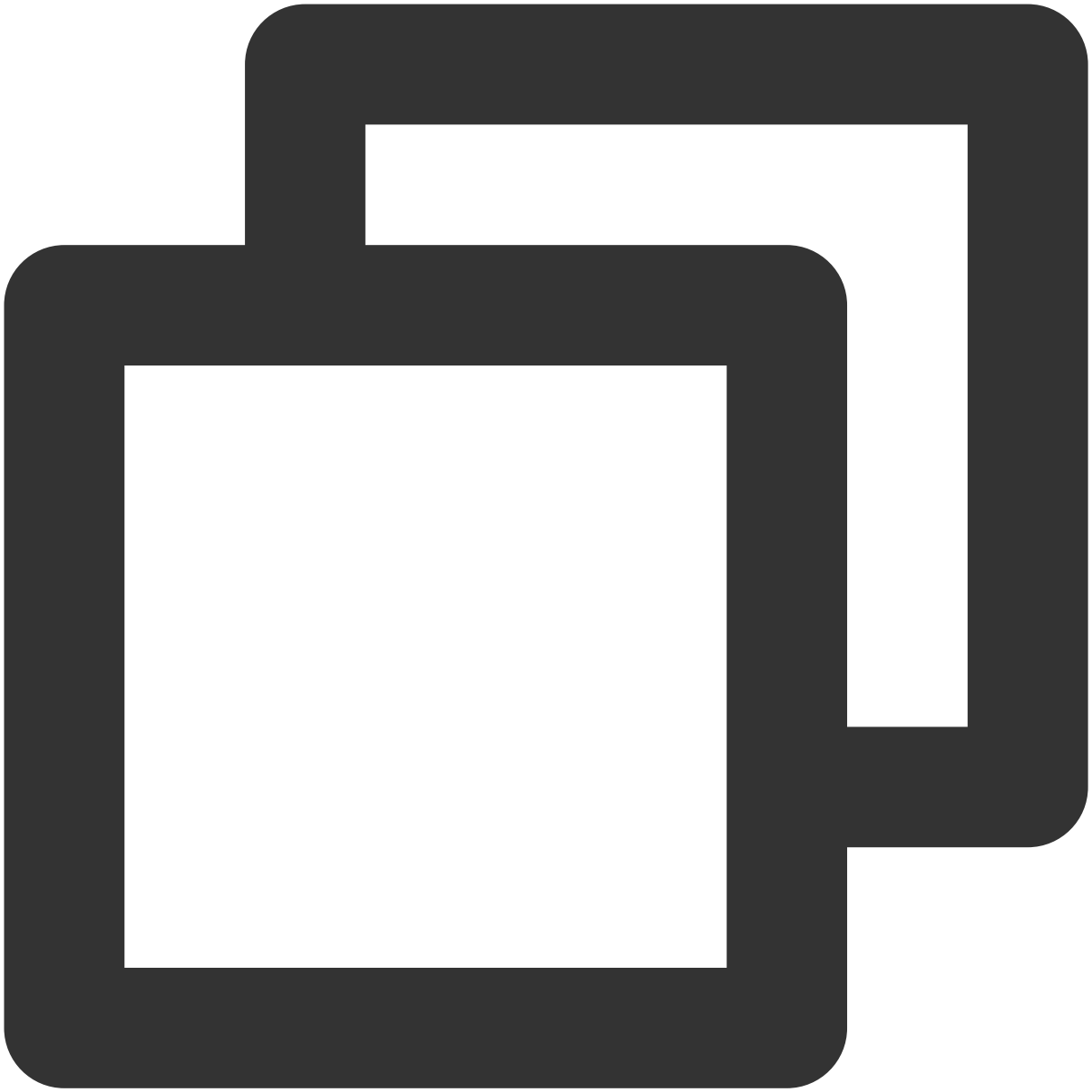
重新执行 Hive 查询语句：



```
select sum(age) from u_user_cosn;
```

6. 创建 OFS Schema 表和查询数据

通过如下命令执行：



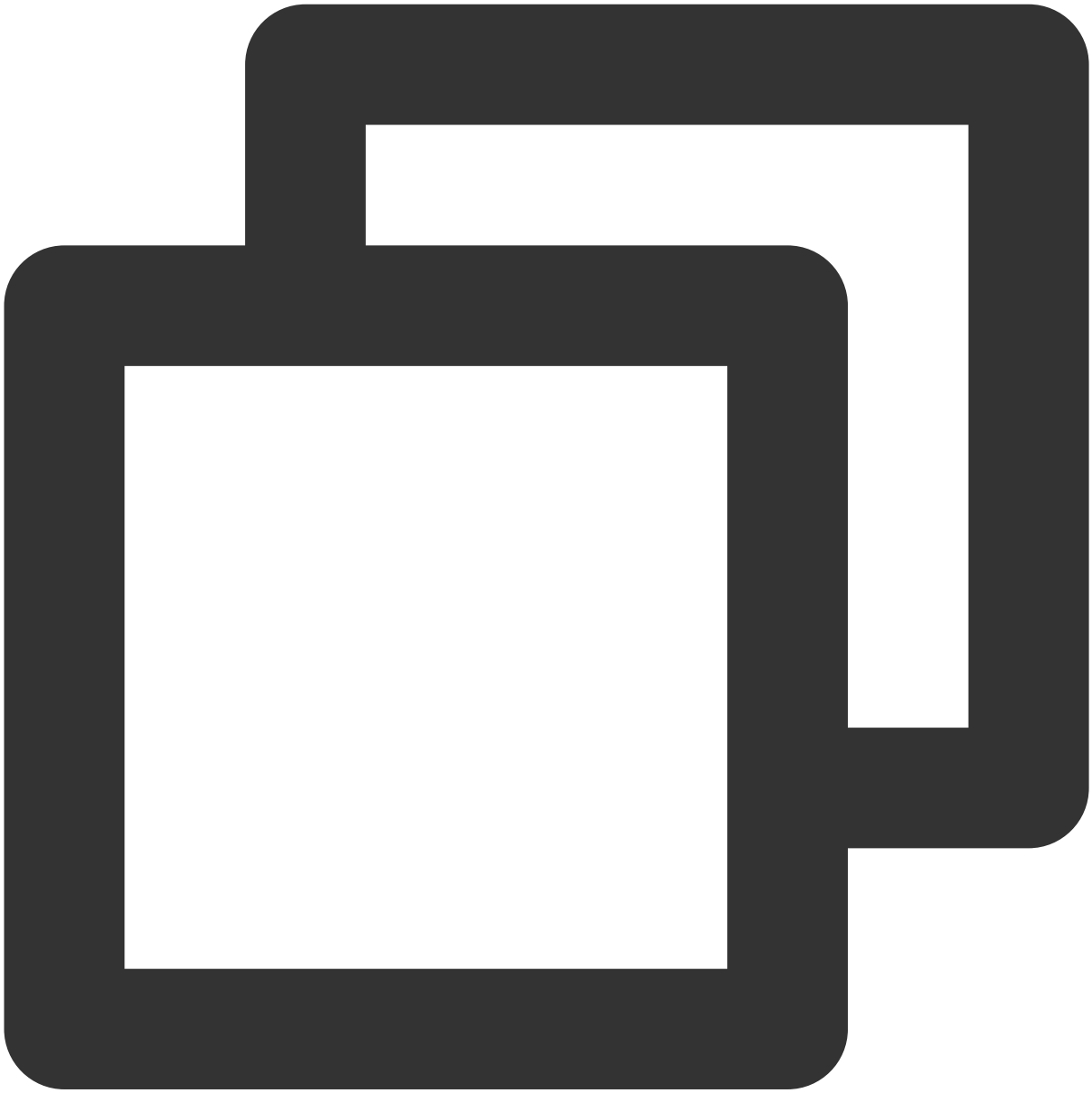
```
CREATE TABLE u_user_ofs (
userid INT,
age INT,
gender CHAR(1),
occupation STRING,
zipcode STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
STORED AS TEXTFILE
LOCATION 'ofs://instance-id.chdfs.ap-guangzhou.myqcloud.com/ofs-test/';
```

版权所有：腾讯云计算（北京）有限责任公司

```
select sum(age) from u_user_ofs;
```

7. 修改 OFS 的实现为 GooseFS 的兼容实现

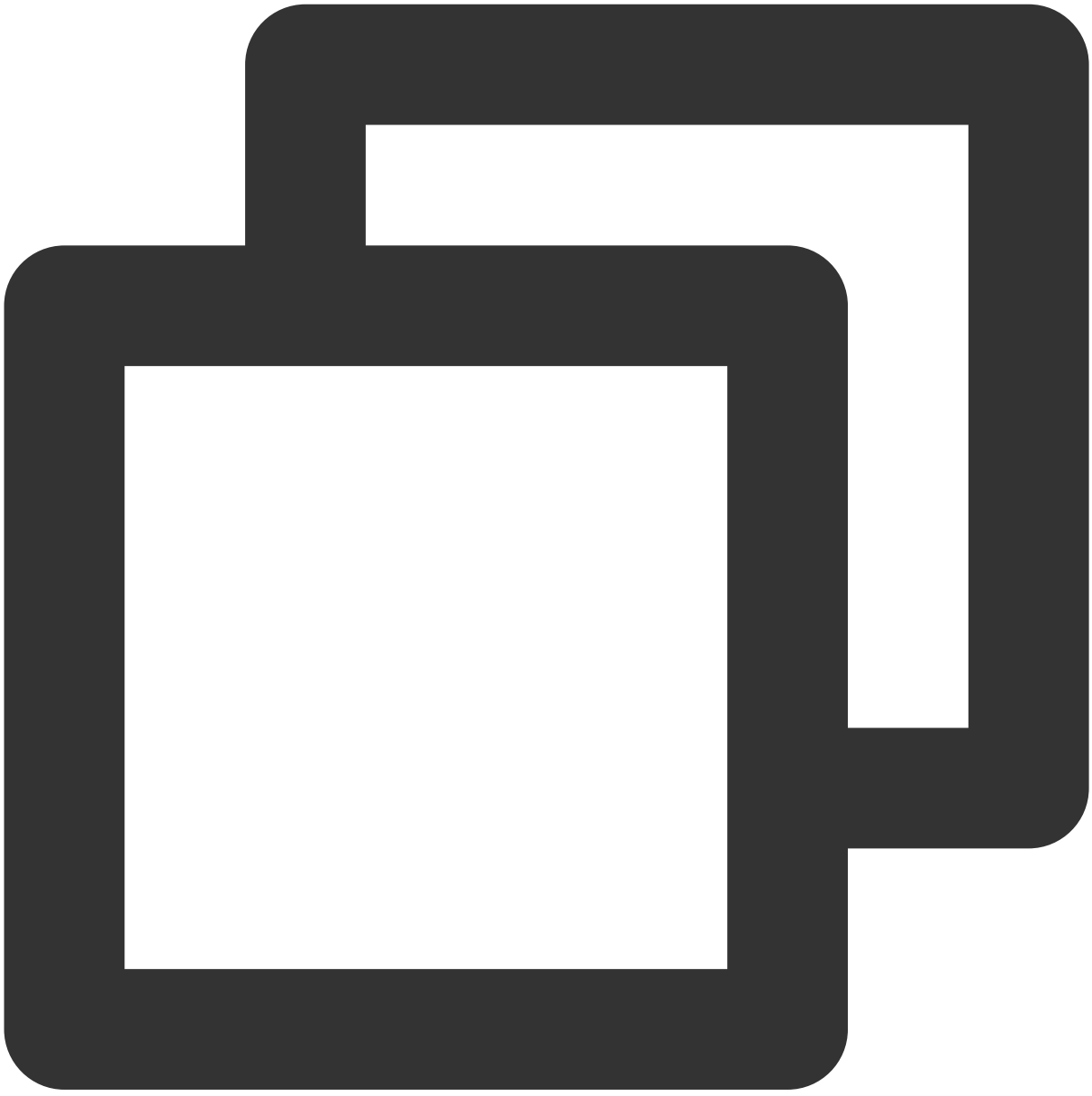
修改 hadoop/etc/hadoop/core-site.xml：



```
<property>
     <name>fs.AbstractFileSystem.ofs.impl</name>
     <value>com.qcloud.cos.goosefs.hadoop.CHDFSDelegateFS</value>
</property>
<property>
```

```
    <name>fs.ofs.impl</name>
    <value>com.qcloud.cos.goosefs.hadoop.CHDFSHadoopFileSystem</value>
</property>
```

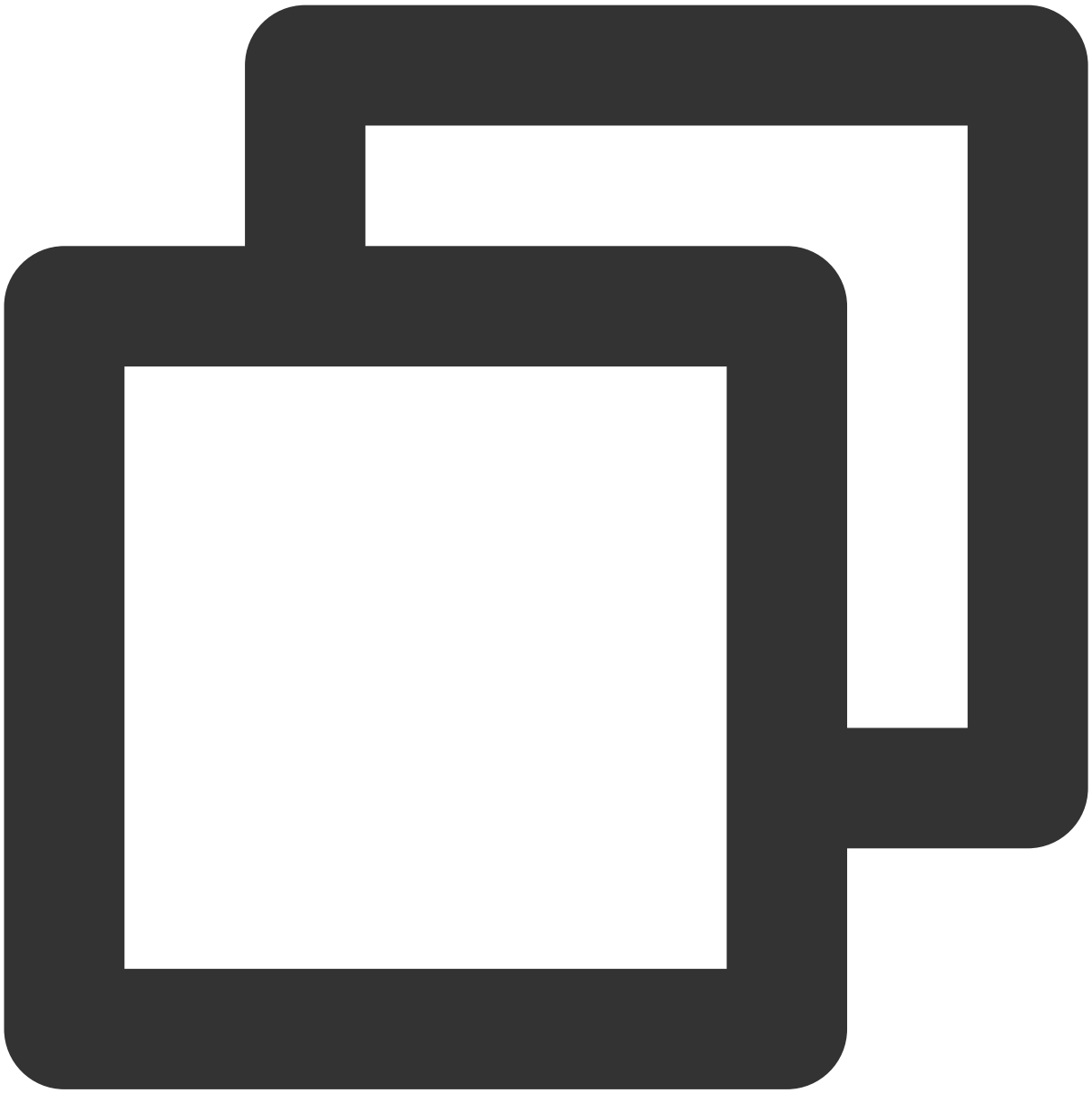
执行 Hadoop 命令，如果路径无法转换为 GooseFS 中的路径，则输出结果中会包含报错信息：



```
hadoop fs -ls  ofs://instance-id.chdfs.ap-guangzhou.myqcloud.com/ofs-test/

Found 1 items
-rw-r--r--   0 hadoop hadoop      22628 2021-07-15 15:56 ofs://instance-id.chdfs.ap

hadoop fs -ls  ofs://instance-id.chdfs.ap-guangzhou.myqcloud.com/unknown-path
```

```
ls: Failed to convert ufs path ofs://instance-id.chdfs.ap-guangzhou.myqcloud.com/un
```

重新执行 Hive 查询语句：



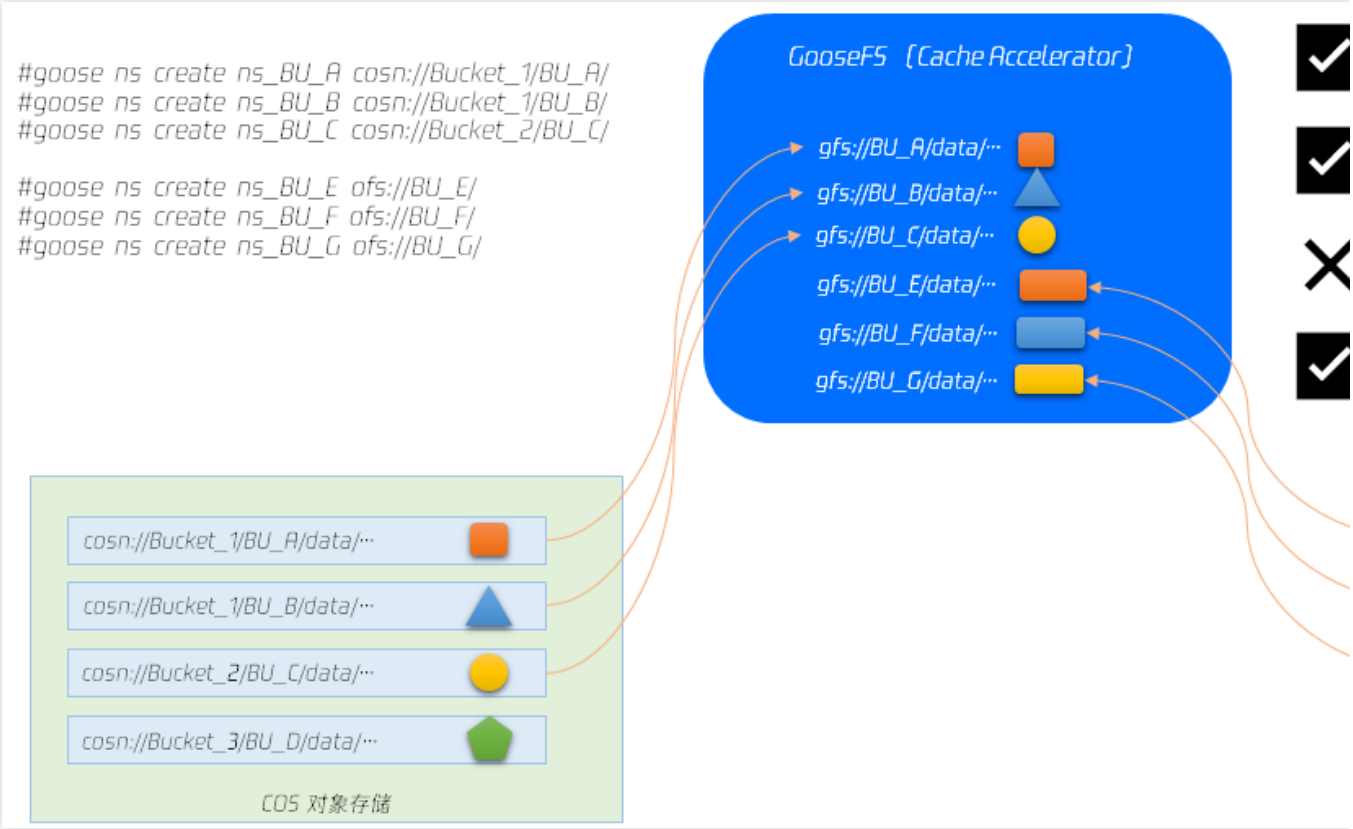
```
select sum(age) from u_user_ofs;
```

统一命名空间能力

最近更新时间：2024-01-06 11:17:59

统一命名空间能力概述

GooseFS 统一命名空间（NameSpace）能力通过透明的命名机制，可以融合多种不同的底层存储系统访问语义，为用户提供了一个统一的数据管理交互视图。
GooseFS 通过统一命名空间能力对接不同的底层存储系统，如本地文件系统、腾讯云对象存储（Cloud Object Storage，COS）、腾讯云云 HDFS（Cloud HDFS，CHDFS）等，与这些底层存储系统进行通信，并为上层业务提供统一的访问接口和文件协议。业务侧只需使用 GooseFS 的访问接口，即可访问存储在不同底层存储系统中的数据。



上图显示了统一命名空间的工作原理。您可以通过 GooseFS 创建命名空间的指令 create ns ，将 COS 和 云 HDFS 的指定文件目录挂载到 GooseFS 中，然后通过 gfs:// 的这一统一的 schema 访问数据。详细说明如下所示：
COS 总共有3个存储桶，分别是 bucket-1、bucket-2、bucket-3，其中 bucket-1 有 BU_A 和 BU_B 两个目录，bucket-1 和 bucket-2 均挂载在 GooseFS 中。
云 HDFS 中有 BU_E、BU_F、BU_G 和 BU_H 共 4 个目录，其中除了 BU_H 其余目录均挂载到 GooseFS 上。

版权所有：腾讯云计算（北京）有限责任公司

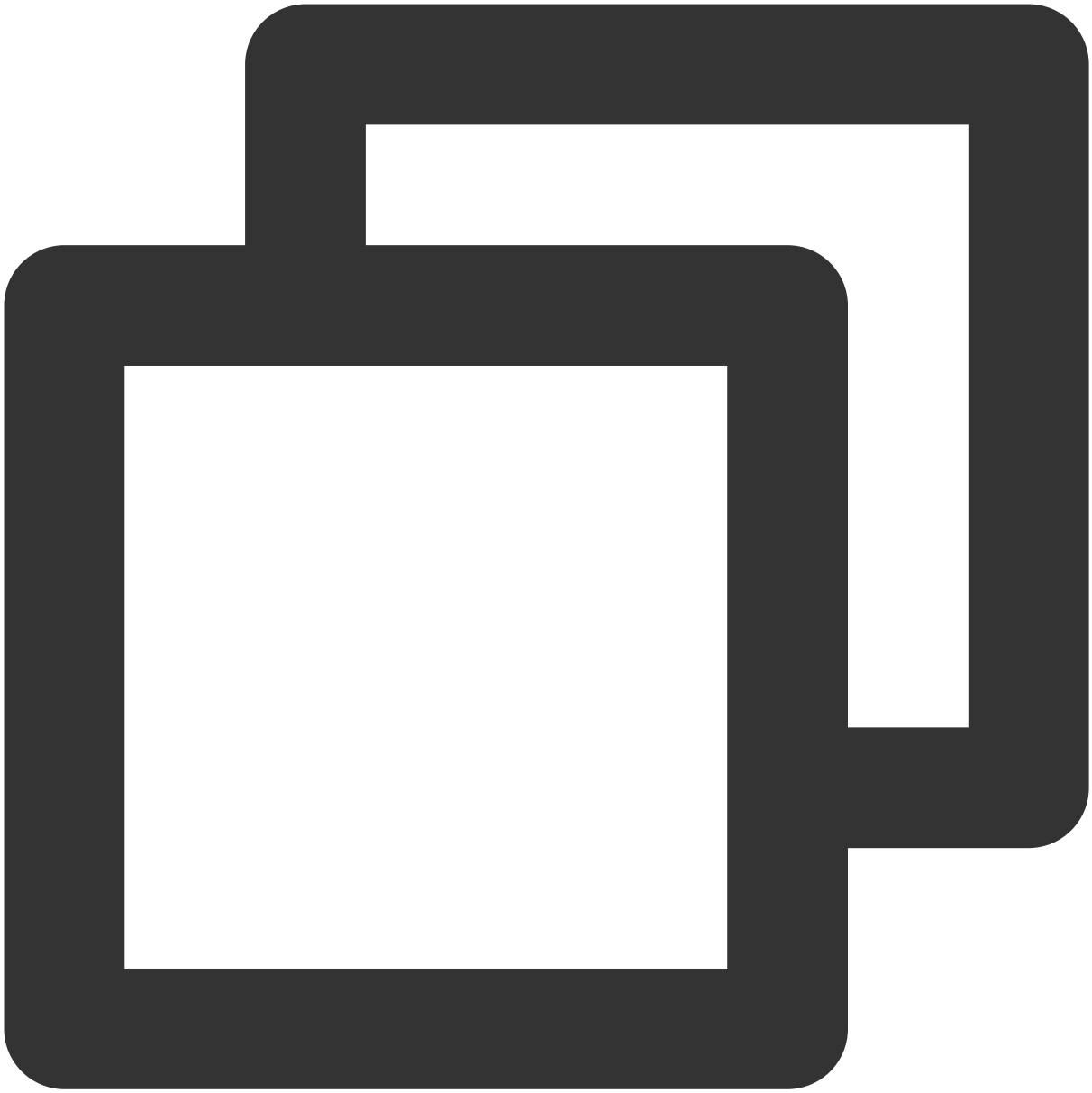
在 GooseFS 的文件操作中，如果以 gfs:// 这一统一的 schema 访问 BU_A 和 BU_E 这两个目录，均可正常访问，且文件缓存在 GooseFS 的本地文件系统中。
BU_A 和 BU_E 这两个存储在底层文件系统（COS、云 HDFS）中的目录已经挂载到 GooseFS 中，如果文件已经缓存在 GooseFS 的上，可以通过 gfs:// 这一统一的 schema 访问（例如 hadoop fs ls gfs://BU_A ）；也可以通过各个远端文件系统的命名空间访问（例如 hadoop fs ls cosn://bucket-1/BU_A ）。
如果文件未被缓存在 GooseFS 上，此时通过 gfs:// 这一形式访问则会失败，因为文件未被缓存在本地文件系统中，但仍然可以通过底层存储系统的命名空间访问。

使用统一命名空间能力

您可以通过 ns 操作在 GooseFS 中创建命名空间，并将底层存储系统映射到 GooseFS 上，目前支持的底层存储系统包括 COS、云 HDFS、本地 HDFS 等。创建命名空间的操作与 Linux 文件系统中挂载文件卷盘类似。创建命名空间后，GooseFS 可以为客户端提供一个具有统一访问语义的文件系统。目前 GooseFS 命名空间的操作指令集如下：

注意

建议用户尽量避免在配置中使用永久密钥，采取配置子账号密钥或者临时密钥的方式有助于提升业务安全性。为子账号授权时建议按需授权子账号可执行的操作和资源，避免发生预期外的数据泄露。
如果您一定要使用永久密钥，建议对永久密钥的权限范围进行限制，可通过限制永久密钥的可执行操作、资源范围和条件（访问 IP 等），提升使用安全性。



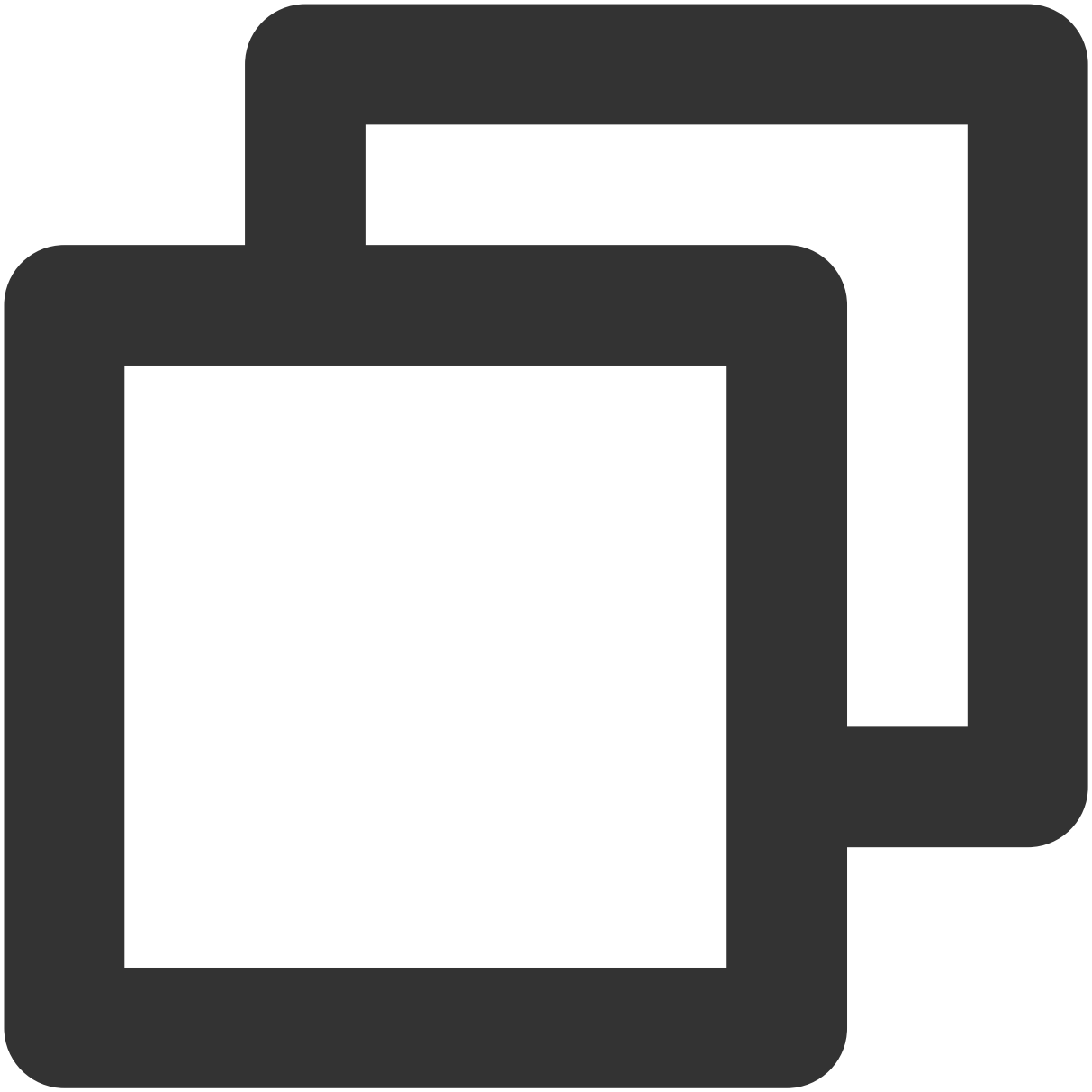
```
$ goosefs ns
Usage: goosefs ns [generic options]
         [create <namespace> <CosN/Chdfs path> <--wPolicy <1-6>> <--rPolicy <1-5>>
         [delete <namespace>]
         [help [<command>]]
         [ls [-r|--sort=option|--timestamp=option]]
         [setPolicy [--wPolicy <1-6>] [--rPolicy <1-5>] <namespace>]
         [setTtl [--action delete|free] <namespace> <time to live>]
         [stat <namespace>]
         [unsetPolicy <namespace>]
         [unsetTtl <namespace>]
```

上述指令集中各项指令的能力简述如下：

指令	说明
create	用于创建命名空间，将一个远端存储系统（UFS）映射到命名空间中；支持在创建命名空间时设置读写缓存策略；需要传入有权限的密钥信息（secretId、secretKey）。
delete	用于删除指定命名空间。
ls	用于列出指定命名空间的详细信息，例如挂载点、UFS 路径、创建时间、缓存策略、TTL 信息等。
setPolicy	用于设置指定命名空间的缓存策略。
setTtl	用于设置指定命名空间的 TTL。
stat	用于提供指定命名空间的描述性信息，例如挂载点、UFS 路径、创建时间、缓存策略、TTL 信息、持久化状态、用户组、ACL、最后一次访问时间、修改时间等。
unsetPolicy	用于重置指定命名空间的缓存策略。
unsetTtl	用于重置指定命名空间的 TTL。

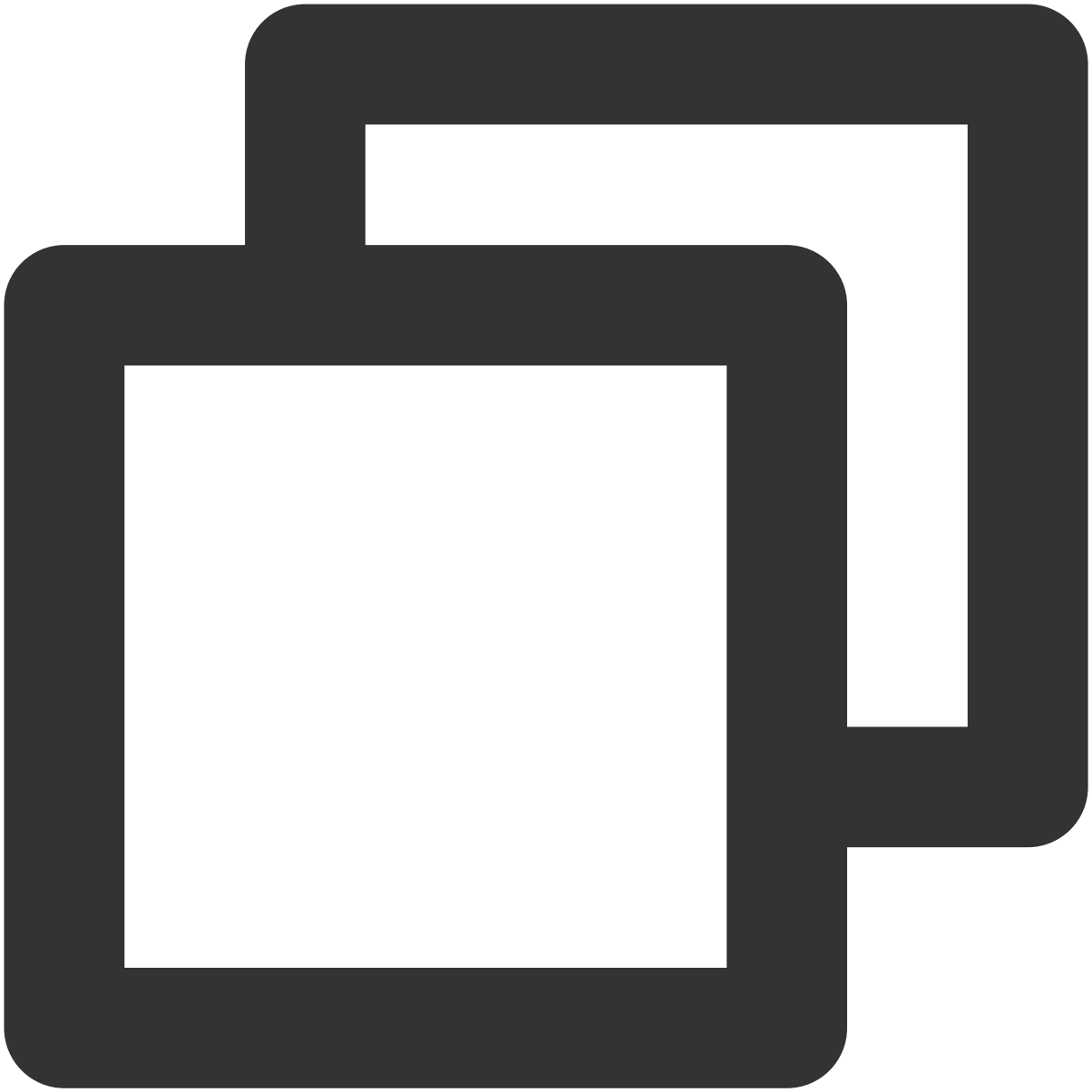
创建和删除命名空间

通过 GooseFS 创建命名空间操作，可以将频繁访问的热数据从远端存储系统缓存到本地高性能存储节点中，为本地计算业务提供高性能的数据访问。如下指令展示了将 COS 中的存储桶 example-bucket、存储桶中的 example-prefix 目录以及云 HDFS 文件系统分别映射到 test_cos、test_cos_prefix 和 test_chdfs 等命名空间下。



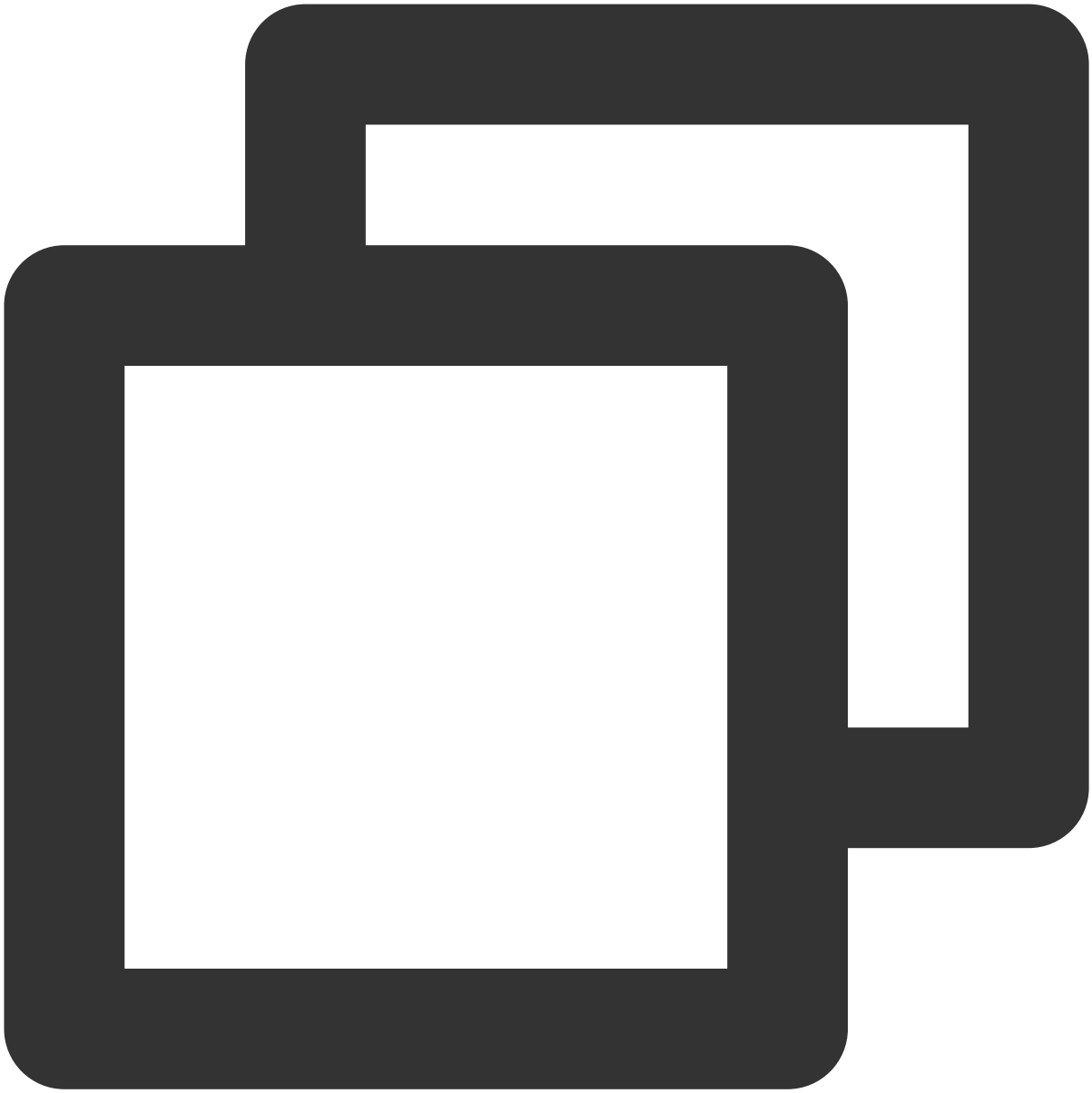
```
# 将 COS 存储桶 example-bucket 映射到 test_cos 命名空间中
$ goosefs ns create test_cos cosn://example-bucket-1250000000/ --wPolicy 1 --rPolic

# 将 COS 存储桶 example-bucket 的 example-prefix 目录映射到 test_cos_prefix 命名空间中
$ goosefs ns create test_cos_prefix cosn://example-bucket-1250000000/example-prefix

# 将 云HDFS 文件系统 f4ma0l3qabc-Xy3 映射到 test_chdfs 命名空间中
$ goosefs ns create test_chdfs ofs://f4ma0l3qabc-Xy3/ --wPolicy 1 --rPolicy 1 --att
```

创建成功后，可以通过 goosefs fs ls 指令查看目录详情：



```
$ goosefs fs ls /test_cos
```

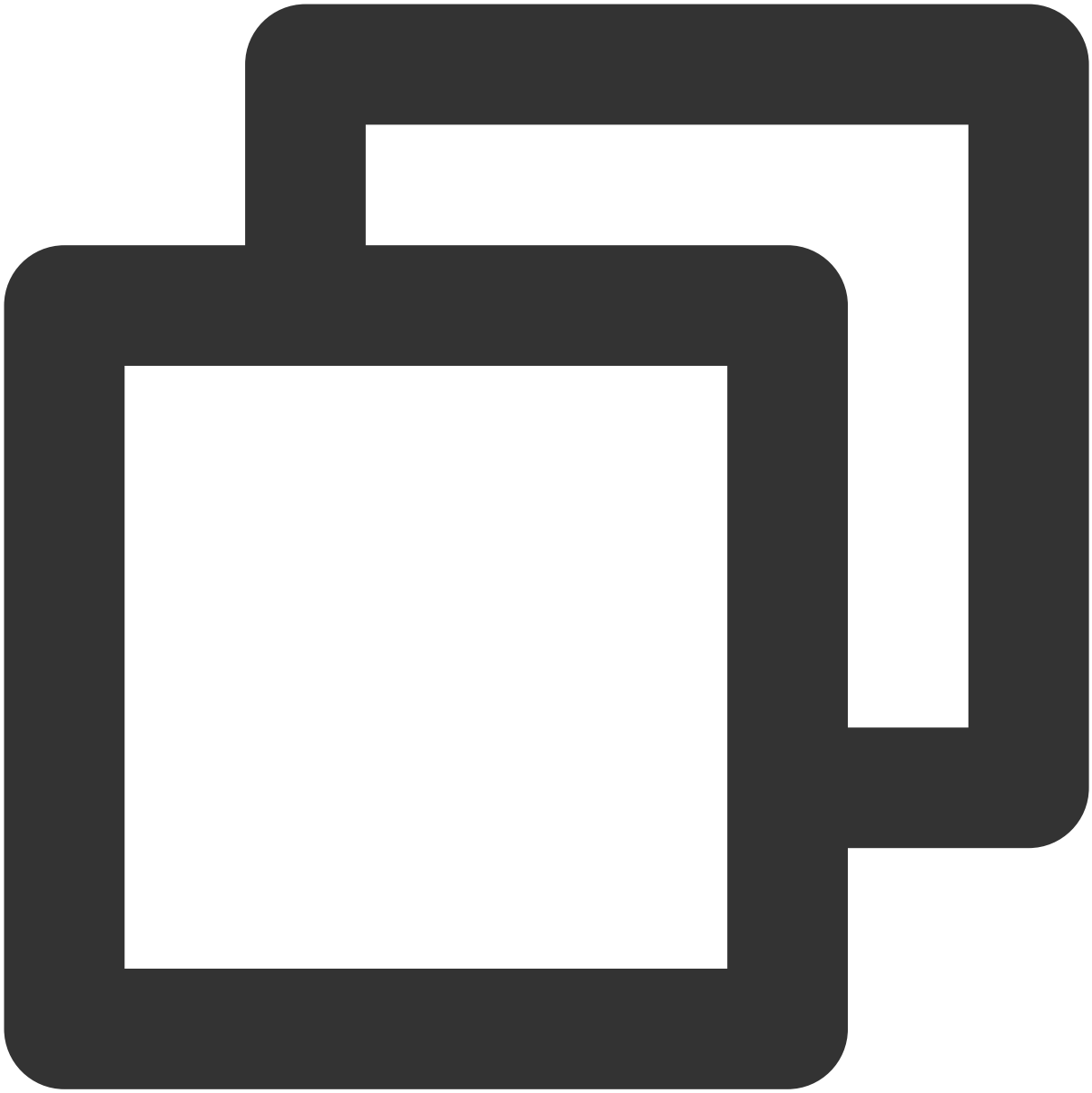
对于不需要使用的命名空间，可以通过 delete 指令来删除：



```
$ goosefs ns delete test_cos
Delete the namespace: test_cos
```

设置缓存策略

用户可通过 setPolicy 和 unsetPolicy 设置指定命名空间的缓存策略。设置缓存策略的指令集如下：



```
$goosefs ns setPolicy [--wPolicy <1-6>] [--rPolicy <1-5>] <namespace>
```

其中各项参数的含义如下：

wPolicy：写缓存策略，支持6种写缓存策略。

rPolicy：读缓存策略，支持5种读缓存策略。

namespace：指定的命名空间。

目前 GooseFS 支持的读写缓存策略分别如下：

写缓存策略

策略名字	行为	对应 Write_Type	数据安	写效率

			全性	
MUST_CACHE（1）	数据仅存储在 GooseFS，不会写入远端存储系统中。	MUST_CACHE	不可靠	高
TRY_CACHE（2）	缓存有空间时就写入 GooseFS 中，缓存如果没空间则直接写入到底层存储中。	TRY_CACHE	不可靠	中
CACHE_THROUGH（3）	尽量缓存数据，同时同步写入远端存储系统。	CACHE_THROUGH	可靠	低
THROUGH（4）	数据不存储在 GooseFS，直接写远端存储系统。	THROUGH	可靠	中
ASYNC_THROUGH（5）	数据写入 GooseFS 中，并异步刷新到远端存储系统。	ASYNC_THROUGH	弱可靠	高

说明

Write_Type：指用户调用 SDK 或者 API 向 GooseFS 中写入数据时指定的文件缓存策略，对单个文件生效。
写缓存策略设置后如果需要变更，需要审慎评估缓存数据的重要性，如果数据重要，建议先保证缓存数据已经持久化，否则缓存数据可能出现丢失情况。例如将写缓存从 `MUST_CACHE` 变更到 `CACHE_THROUGH` 后，如果未调用 `persist` 指令持久化数据，那么即将淘汰的数据无法写入到底层，会出现丢失的情况。

读缓存策略

策略名字	行为	元数据同步	对应 Read_Type
NO_CACHE（1）	不缓存数据，直接从远端存储系统中读数据。	NO	NO_CACHE
CACHE（2）	元数据访问行为：如果命中缓存时，元数据以 Master 中的为准，不会主动从底层同步元数据。 数据流访问行为：数据流的 ReadType 采用 CACHE 策略。	Once	CACHE
CACHE_PROMOTE（3）	元数据访问行为：与 CACHE 模式相同。	Once	CACHE_PROMOTE

	数据流访问行为：数据流的 ReadType 采用 CACHE_PROMOTE 策略。		
CACHE_CONSISTENT_PROMOTE（4）	元数据行为：每次读取操作前均先同步远端存储系统 UFS 上的元数据，如果 UFS 中不存在，则抛出异常 Not Exists。 数据流访问行为：数据流的 ReadType 采用 CACHE_PROMOTE 策略，命中以后，缓存到最热的缓存介质中。	Always	CACHE
CACHE_CONSISTENT（5）	元数据行为：与 CACHE_CONSISTENT_PROMOTE 相同。 数据流访问行为：数据流的 ReadType 采用 CACHE 策略，即 CACHE 命中，不会在不同的介质层中移动数据。	Always	CACHE_PROMOTE

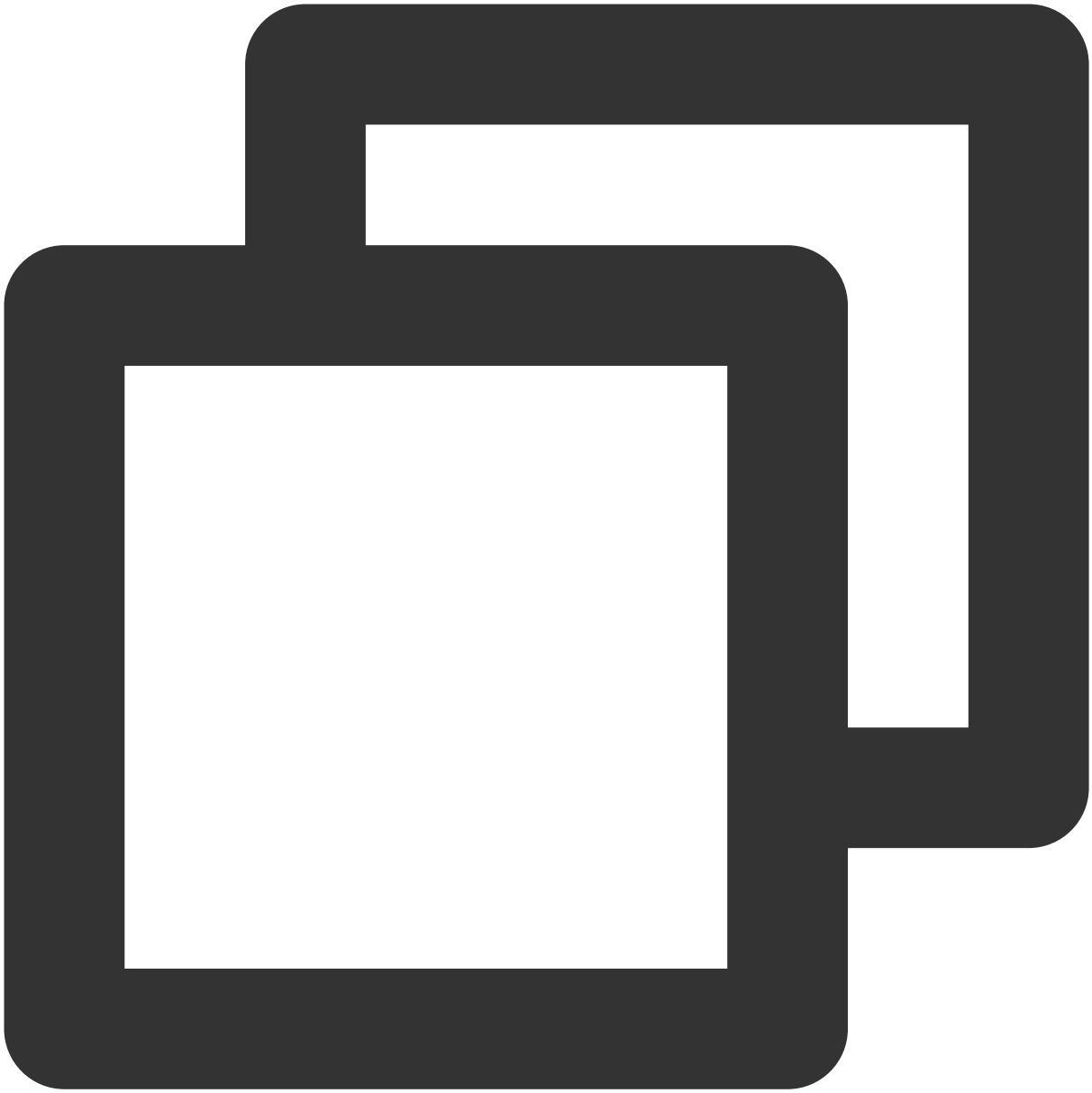
说明

Read_Type：指用户调用 SDK 或者 API 从 GooseFS 中读取数据时指定的文件缓存策略，对单个文件生效。

结合目前大数据的业务实践，我们推荐的读写缓存策略组合主要如下：

写缓存策略	读缓存策略	策略组合表现
CACHE_THROUGH（3）	CACHE_CONSISTENT（5）	缓存和远端存储系统数据强一致。
CACHE_THROUGH（3）	CACHE（2）	写强一致性，读最终一致性。
ASYNC_THROUGH（5）	CACHE_CONSISTENT（5）	写最终一致性，读强一致性。
ASYNC_THROUGH（5）	CACHE（2）	读写最终一致性。
MUST_CACHE（1）	CACHE（2）	只从缓存中读数据。

如下示例展示了将指定命名空间 test_cos 的读写缓存策略分别设置为 CACHE_THROUGH 和 CACHE_CONSISTENT：



```
$ goosefs ns setPolicy --wPolicy 3 --rPolicy 5 test_cos
```

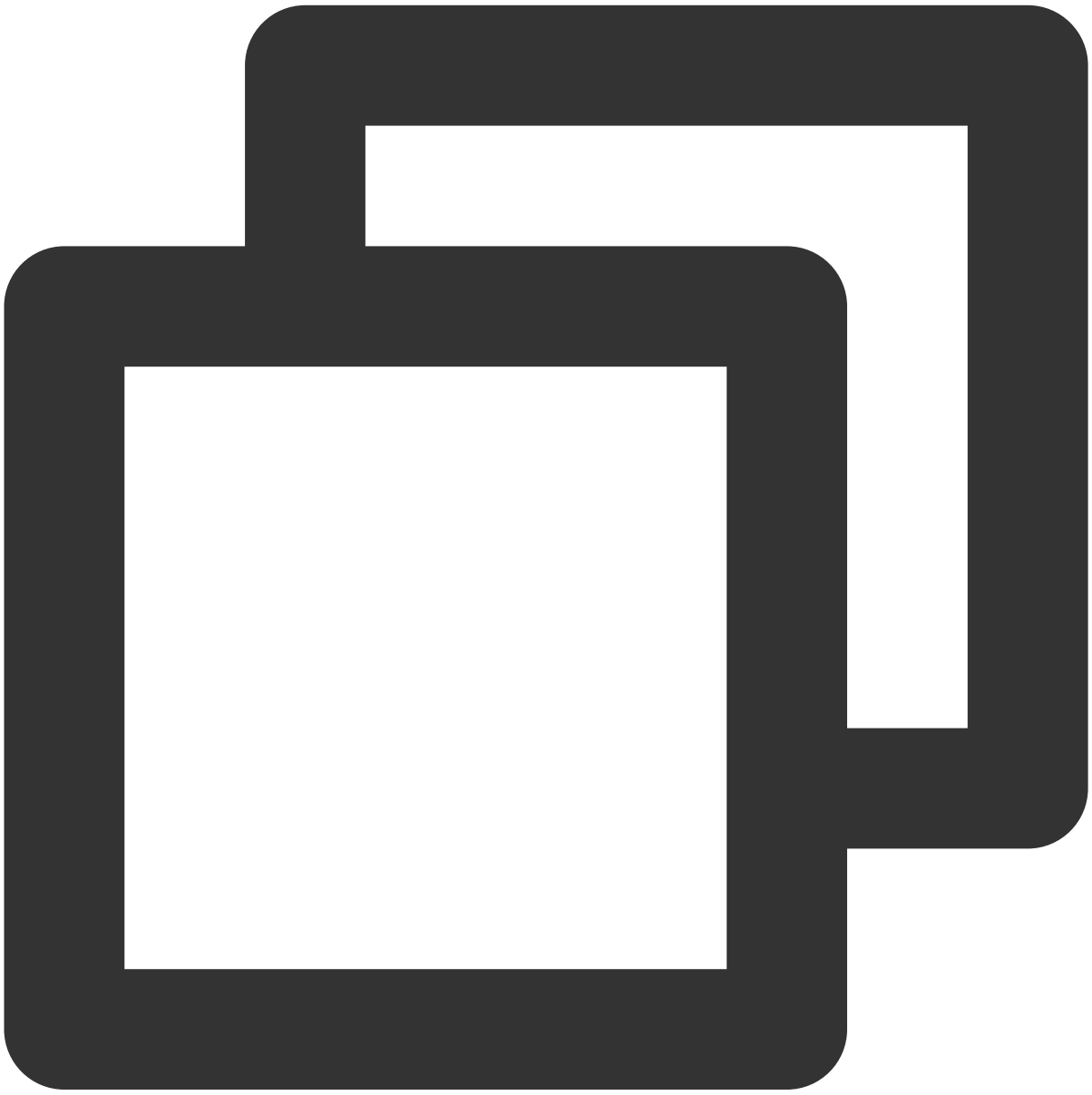
注意

除了在创建命名空间时指定缓存策略，用户还可以通过在读写文件时，针对指定文件设置 ReadType 或者 Write_Type，或者通过 properties 配置文件配置全局缓存策略。多个策略同时存在的时候，优先级为用户自定义优先级 > Namespace 读写策略 > 配置文件的全局缓存策略配置。其中，针对读策略会采用用户自定义 ReadType 和 Namespace 的 DirReadPolicy 的组合生效，即数据流读策略采用用户自定义 ReadType，元数据采用 Namespace 的策略。

版权所有：腾讯云计算（北京）有限责任公司

例如，GooseFS 中存在一个 COSN 命名空间，读策略为 CACHE_CONSISTENT；假设在该命名空间中存在一个 test.txt 的文件。客户端读取 test.txt 时，ReadType 指定了 CACHE_PROMOTE。那么整个读取行为就是同步元数据并且 CACHE_PROMOTE。

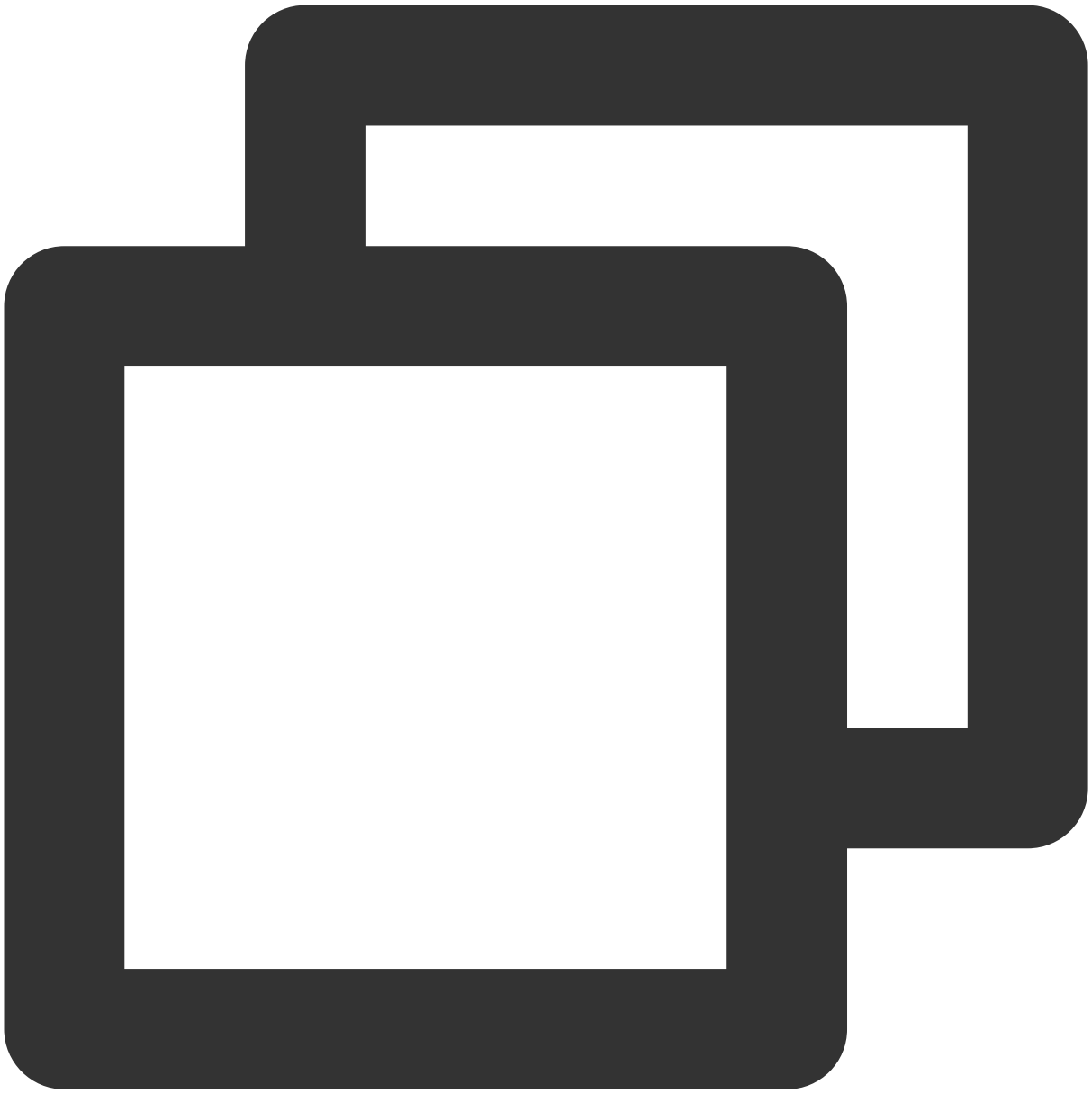
如果需要重置读写缓存策略，可以通过 unsetPolicy 指令实现，如下策略展示了重置 test_cos 命名空间的读写缓存策略：



```
$ goosefs ns unsetPolicy test_cos
```

设置 TTL

TTL 用于管理缓存在 GooseFS 本地节点的数据，配置 TTL 参数可以让缓存数据在指定的时间后执行指定的操作，例如 delete 或者 free 操作。目前设置 TTL 的操作指令如下：



```
$ goosefs ns setTtl [--action delete|free] <namespace> <time to live>
```

其中各项参数的含义如下：

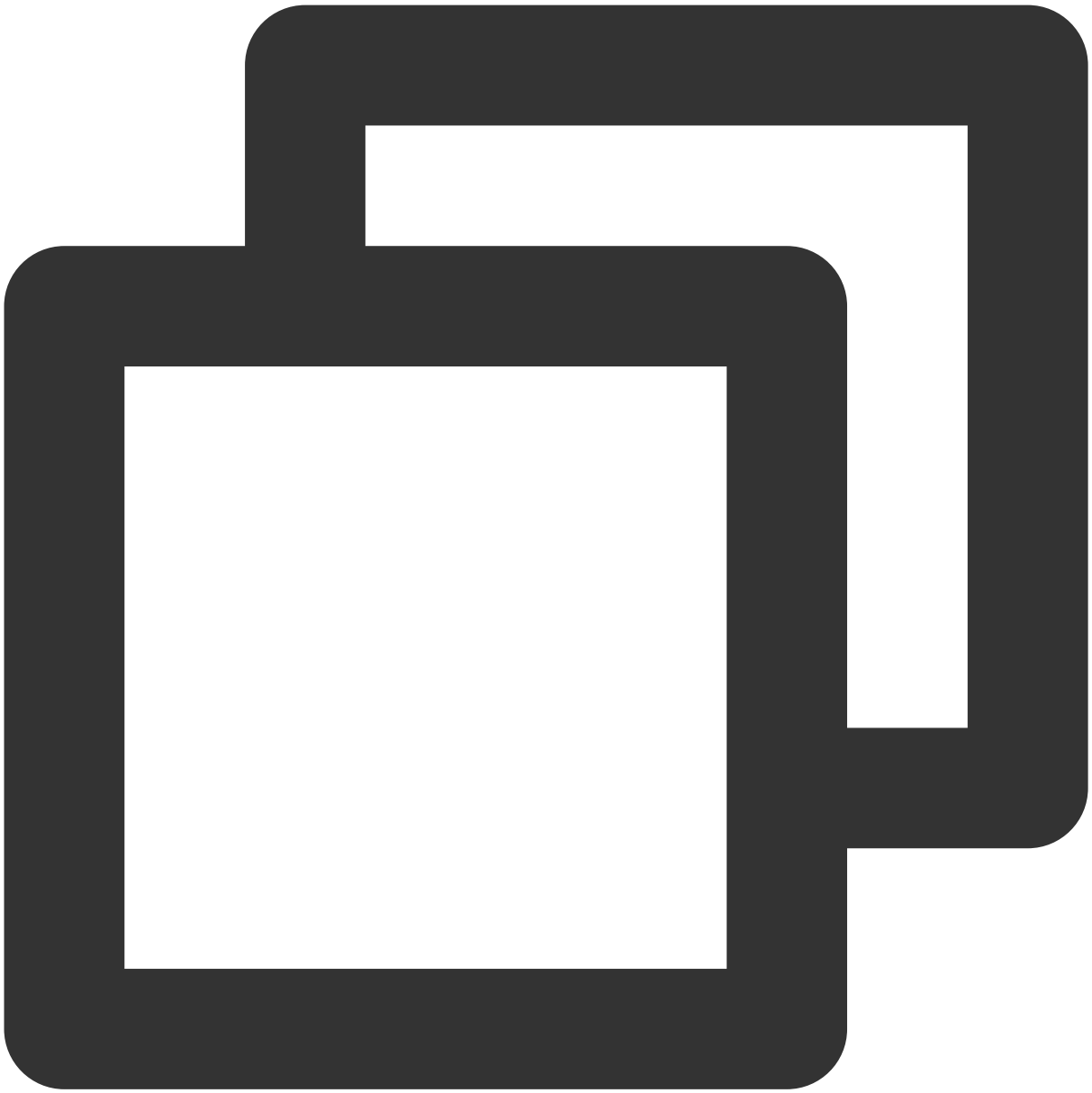
action：缓存时间到期后执行的操作，目前支持 delete 和 free 两种操作。delete 操作会删除缓存和 UFS 上的数据，free 操作只会删除缓存上的数据。

namespace：指定的命名空间。

time to live：数据缓存时间，单位毫秒。

版权所有：腾讯云计算（北京）有限责任公司

如下示例展示了将指定命名空间 test_cos 的策略设置为60秒到期后删除：



```
$ goosefs ns setTtl --action free test_cos 60000
```

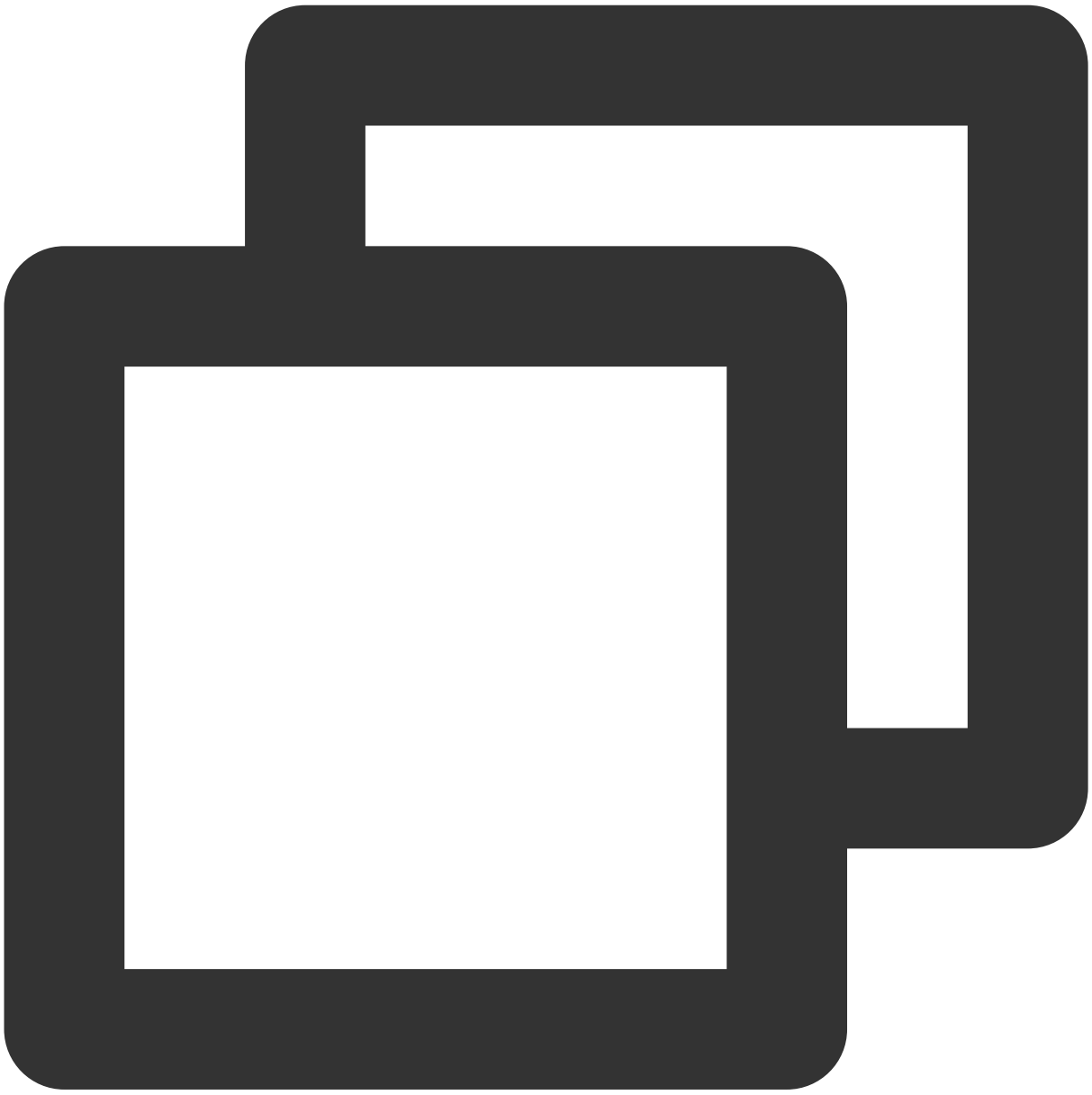
元数据信息管理

本小节介绍 GooseFS 如何管理元数据，包括元数据的同步、更新等内容。GooseFS 为用户提供了统一命名空间能力，用户可以通过统一的 gfs:// 的路径来访问不同底层存储系统上的文件，只需要指定好底层存储系统的路径即可。

我们推荐您使用 GooseFS 作为统一的数据接入层，统一从 GooseFS 进行数据读写，保障元数据信息的一致性。

元数据同步概述

您可以通过在 conf/goosefs-site.properties 配置文件中修改元数据同步周期来管理元数据同步周期，配置参数如下：



```
goosefs.user.file.metadata.sync.interval=<INTERVAL>
```

同步周期支持如下3种入参：

入参数值为-1：代表元数据在首次加载到 GooseFS 中后就不再进行更新。

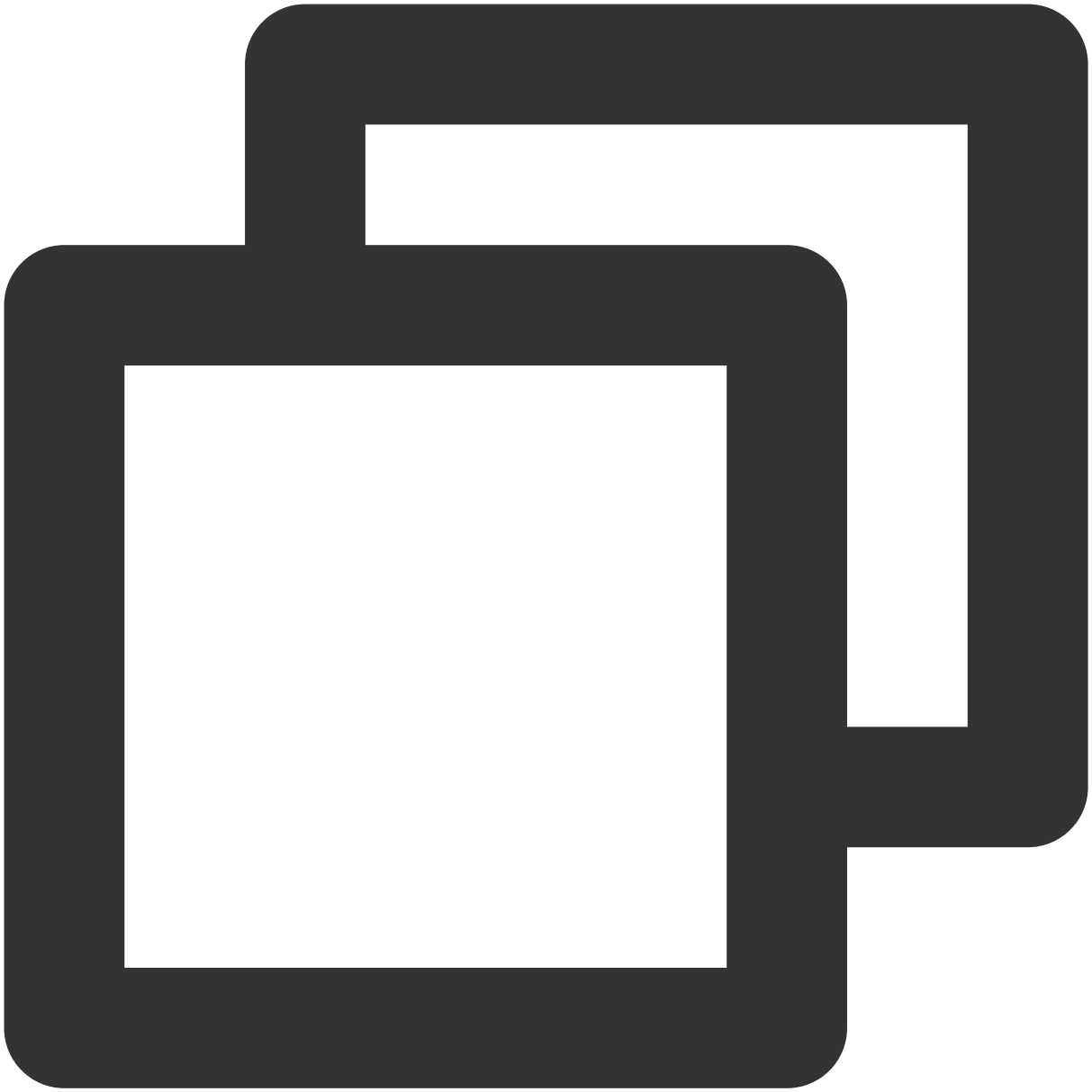
入参数值为0：代表 GooseFS 会在每次读写操作后都更新元数据。

入参数值为正整数：代表 GooseFS 会按照指定的时间间隔，周期性地更新元数据。
您可以综合考虑您的节点数量、GooseFS 集群和底层存储的 I/O 距离、底层存储类型等因素，选择合适的同步周期。通常：
GooseFS 集群节点数量越多，元数据同步延迟越大。
GooseFS 集群所处的机房和底层存储的物理距离越远，元数据同步延迟越大。
底层存储系统对元数据同步延迟的影响主要视系统请求 QPS 负载情况而定，QPS 负载越高则同步延迟相对更低。

元数据同步管理方式

配置方式

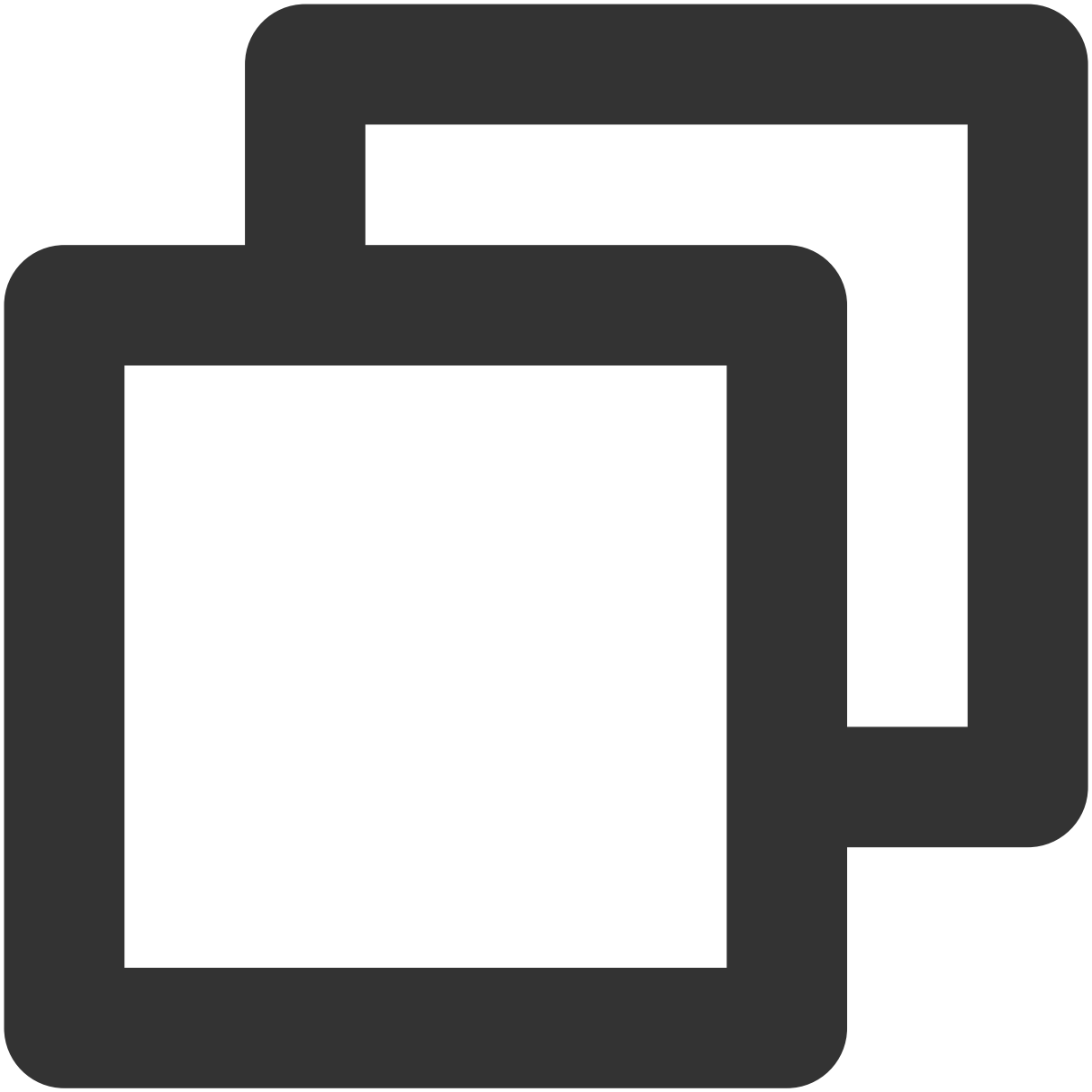
1. 通过命令行配置
您可以通过命令行的方式设置元数据信息同步周期：



```
goosefs fs ls -R -Dgoosefs.user.file.metadata.sync.interval=0 <path to sync>
```

2. 通过配置文件配置

对于大规模的 Goosefs 集群而言，您可以通过 goosefs-site.properties 配置文件批量配置集群中 Master 节点的元数据信息同步周期，其他节点的同步周期会默认按照该周期值执行。



```
goosefs.user.file.metadata.sync.interval=1m
```

注意

很多业务会选择按照目录来区分数据的用途，不同目录的数据访问频次并不全部相同。元数据同步周期可以按照不同目录设置不同的周期值，对于一些经常变化的目录，可以将该目录的元数据同步周期设置为较短的时间（如5分钟），对于极少变化或者不变化的目录，可以将同步周期设置为-1，这样 GooseFS 不会自动同步该目录的元数据。

推荐配置

根据业务访问模式的差异，您可以设置不同的元数据同步周期：

访问模式		元数据同步周期	说明
所有文件请求都经过 GooseFS		-1	-
绝大部分文件请求都经过 GooseFS	使用 HDFS 作为 UFS	推荐使用热更新或者按路径更新	如果 HDFS 的更新特别频繁，推荐将更新周期设置为-1，禁止更新
	使用 COS 作为 UFS	推荐按照路径配置更新周期	按照不同目录配置不同的更新周期，可以缓解元数据同步的压力
上传文件请求一般不经过 GooseFS	使用 HDFS 作为 UFS	推荐按照路径配置更新周期	
	使用 COS 作为 UFS	推荐按照路径配置更新周期	

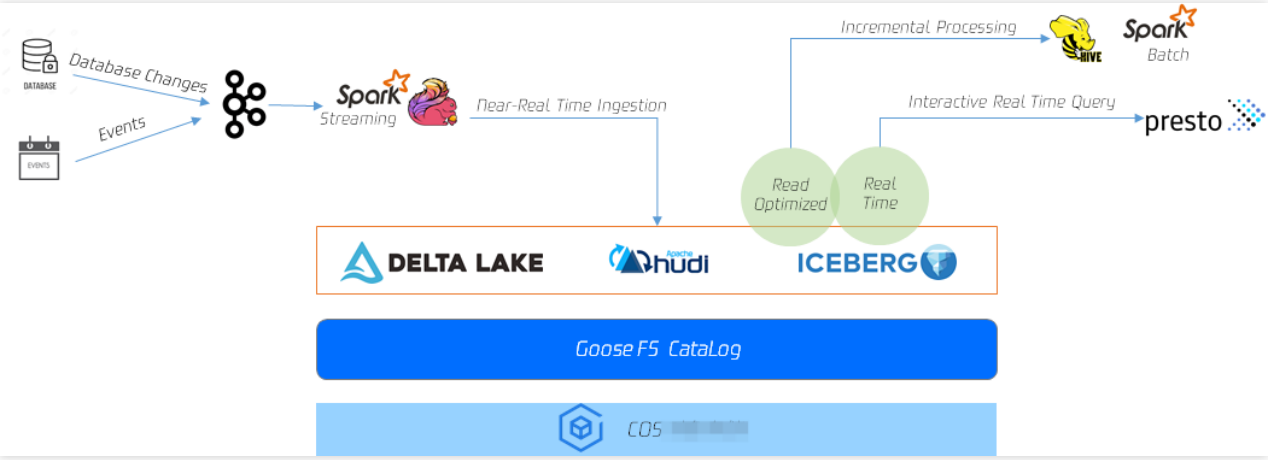
版权所有：腾讯云计算（北京）有限责任公司

Table 管理能力

最近更新时间：2024-01-06 11:17:59

Table 管理能力概述

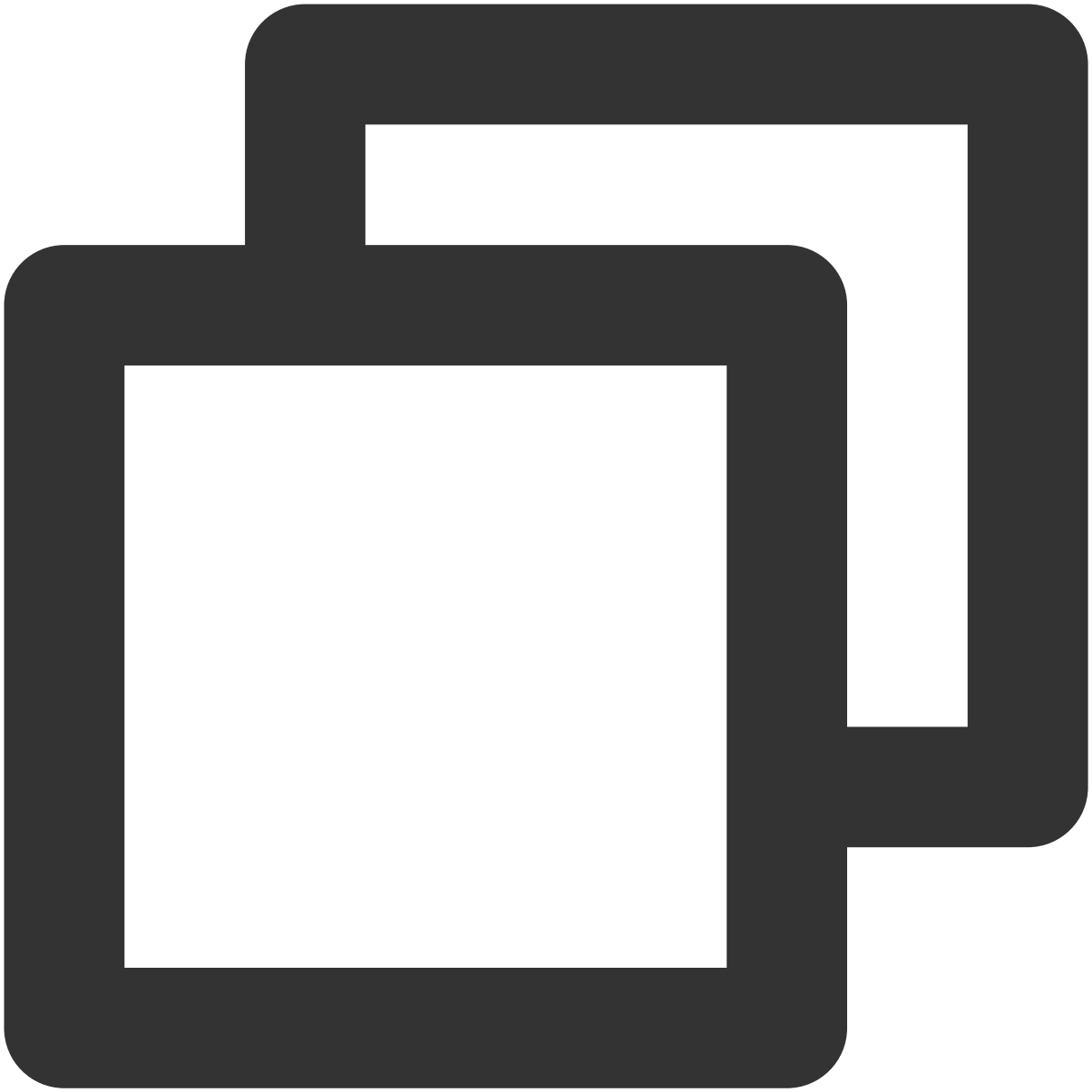
GooseFS Table 管理能力用于管理结构化数据，为 SparkSQL、Hive、Presto 等上层计算应用提供数据库表管理能力，目前底层支持对接 Hive MetaStore。Table 管理能力能够帮助各类 SQL 引擎读取指定的数据内容，能够有效提升大数据场景下对数据的访问效率。



GooseFS Table 管理能力目前主要支持了以下特性：
元数据层面的描述能力。GooseFS Catalog 提供源自远程元数据服务（Hive MetaStore）的元数据缓存服务，针对 SparkSQL，Hive，SQL Presto 等 SQL 引擎做查询时，可以根据 GooseFS Catalog 中的元数据缓存服务来确定读取数据大小、目标数据位置以及数据结构，具备与 Hive MetaStore 相同的能力表现。
表级数据预缓存能力。GooseFS Catalog 能够感知数据表和数据存储路径的对应关系，进而可以提供 Table 级别以及 Table Partition 级别的缓存预热能力，帮助用户提前按照表结构缓存数据，极大提高访问性能。
跨存储服务的统一元数据服务。通过 GooseFS Catalog 运行上层计算应用，可以同时对不同的底层存储系统提供访问加速能力。同时 GooseFS Catalog 可以提供跨越存储服务的统一元数据查询能力，只需要一个 GooseFS 客户端开启 Catalog 功能，即可查询不同存储系统，例如 HDFS、COS、CHDFS 中的数据。

使用 GooseFS Table 管理能力

GooseFS Table 管理能力通过 goosefs table 指令集实现，提供了 DB 的绑定和解绑、查询 DB 信息、查询表信息、数据加载、数据淘汰等能力。GooseFS Table 管理指令集如下所示：



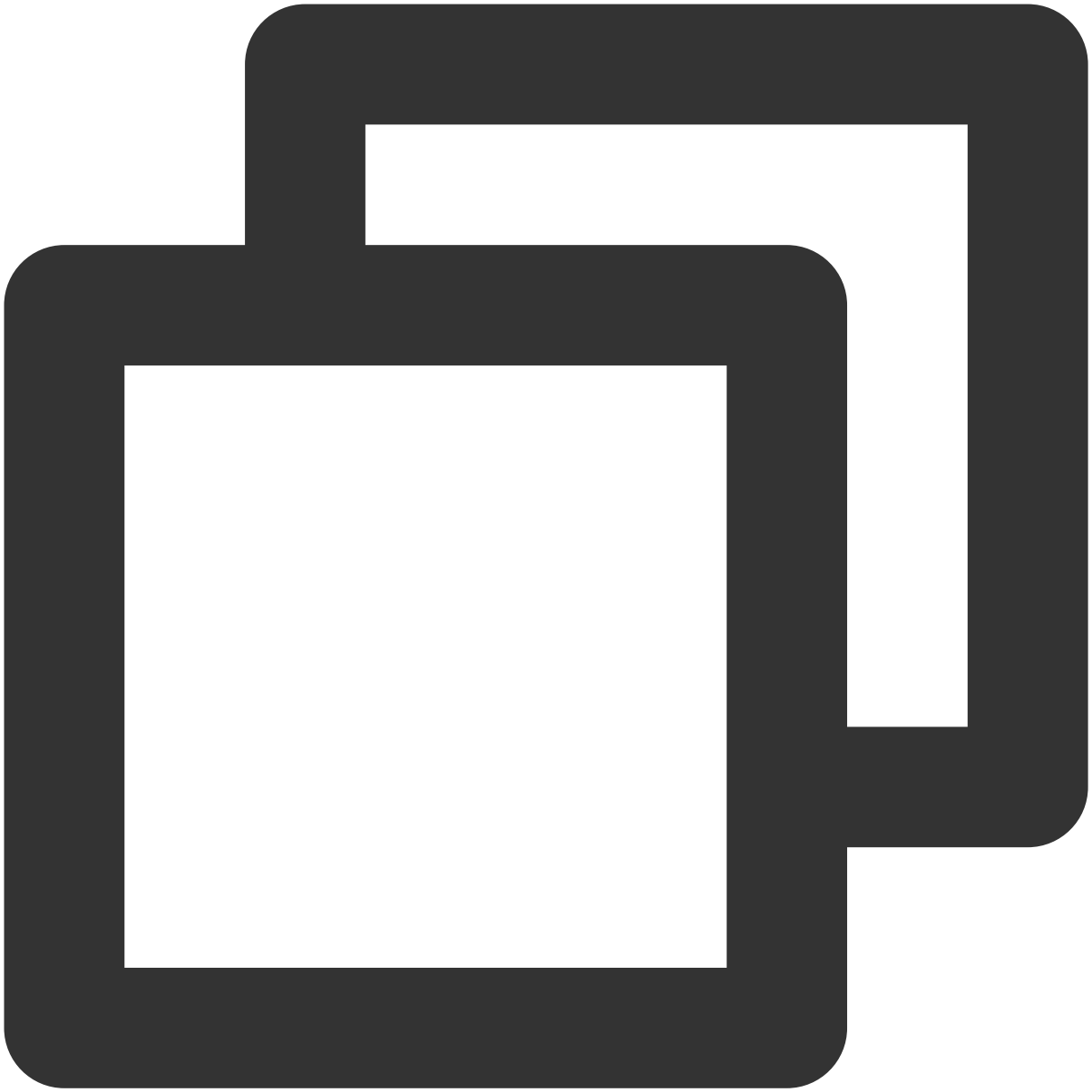
```
$ goosefs table
Usage: goosefs table [generic options]
     [attachdb [-o|--option <key=value>] [--db <goosefs db name>] [--ignore-sync-er
     [detachdb <db name>]
     [free <dbName> <tableName> [-p|--partition <partitionSpec>]]
     [load <dbName> <tableName> [-g|--greedy] [--replication <num>] [-p|--partition
     [ls [<db name> [<table name>]]]
     [stat <dbName> <tableName>]
     [sync <db name>]
```

上述指令集中各项指令的能力简述如下：

attachdb：挂载数据库，将一个远端数据库绑定到 GooseFS 上，目前仅支持 Hive MetaStore。
detachdb：卸载数据库，将 GooseFS 上绑定的数据库解绑。
free：清除指定 DB.Table 的数据缓存，可支持 Partition 粒度。
load：缓存指定 DB.Table 的数据，可支持 partition 粒度，支持通过 replication 设置缓存的副本数。
ls：列出指定 DB 或 DB.Table 的元数据信息。
stat：查询指定 DB.Table 的文件数目、总大小、以及缓存百分比。
sync：同步指定 DB 的内容。
transform：将指定 DB 关联的 Table 转换为新的 Table。
transformStatus：Table 转换的进度情况。

1. 挂载 DB

预热指定 Table 数据到 GooseFS 之前，需要将对应的 DB 挂载到 GooseFS 上。如下指令展示了将指定地址 metastore_host:port 中的数据库 goosefs_db_demo 挂载到 GooseFS 中，并将该 DB 在 GooseFS 中命名为 test_db：



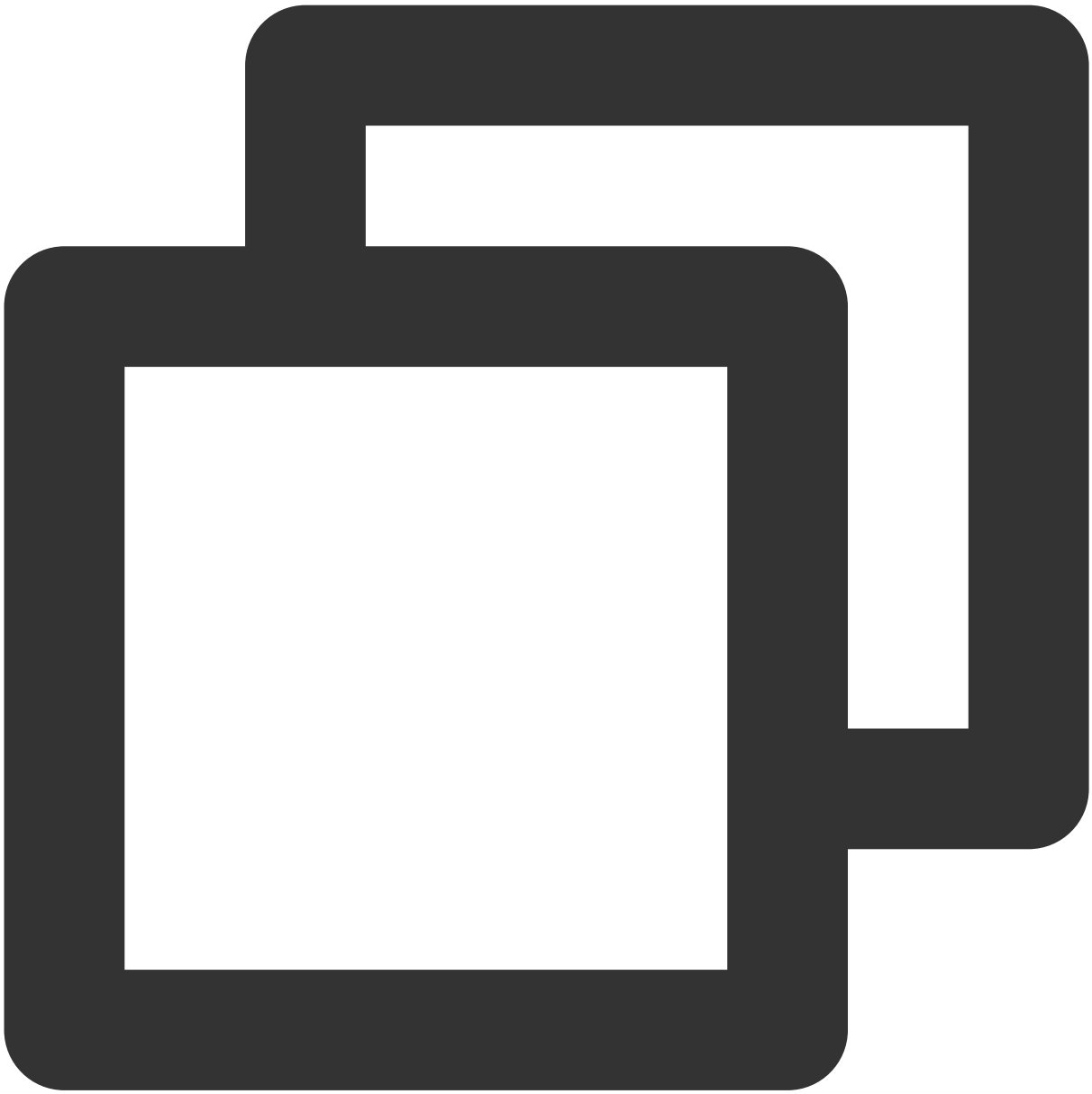
```
$ goosefs table attachdb --db test_db hive thrift://metastore_host:port goosefs_db_

response of attachdb
```

注意

metastore_host:port 可以替换为任意合法可连接的 Hive MetaStore 地址。

2. 查看 Table 信息

绑定完数据库后，可以通过 ls 指令查看已挂载的 DB 和 Table 信息，如下指令展示了如何查询 test_db 中的 web_page 表信息：

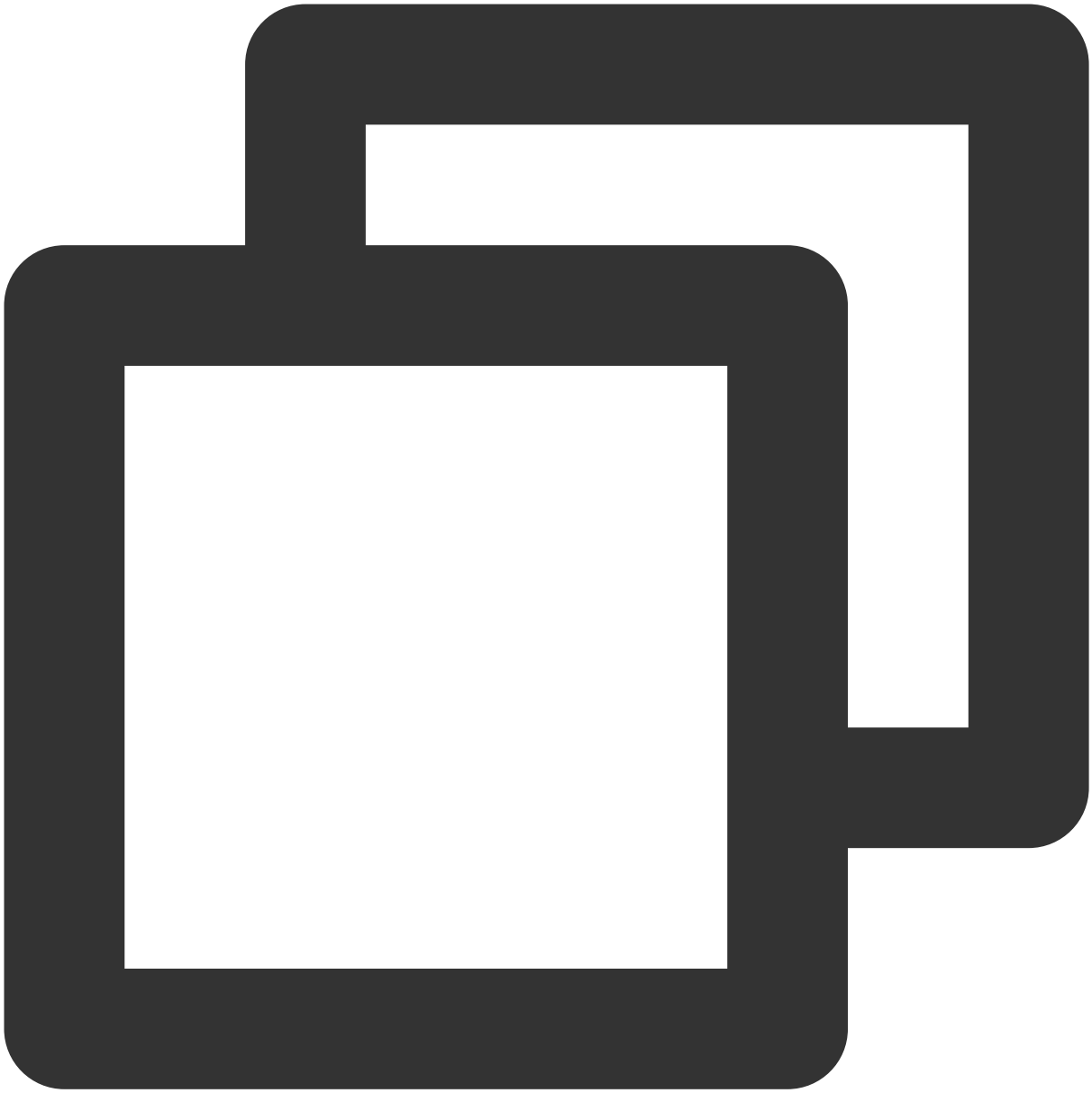


```
$ goosefs table ls test_db web_page

OWNER: hadoop
DBNAME.TABLENAME: testdb.web_page (
    wp_web_page_sk bigint,
    wp_web_page_id string,
    wp_rec_start_date string,
    wp_rec_end_date string,
```

```
    wp_creation_date_sk bigint,
    wp_access_date_sk bigint,
    wp_autogen_flag string,
    wp_customer_sk bigint,
    wp_url string,
    wp_type string,
    wp_char_count int,
    wp_link_count int,
    wp_image_count int,
    wp_max_ad_count int,
)
PARTITIONED BY (
)
LOCATION (
    gfs://metastore_host:port/myiNamespace/3000/web_page
)
PARTITION LIST (
    {
    partitionName: web_page
    location: gfs://metastore_host:port/myNamespace/3000/web_page
    }
)
```

3. 预热 Table 中的数据

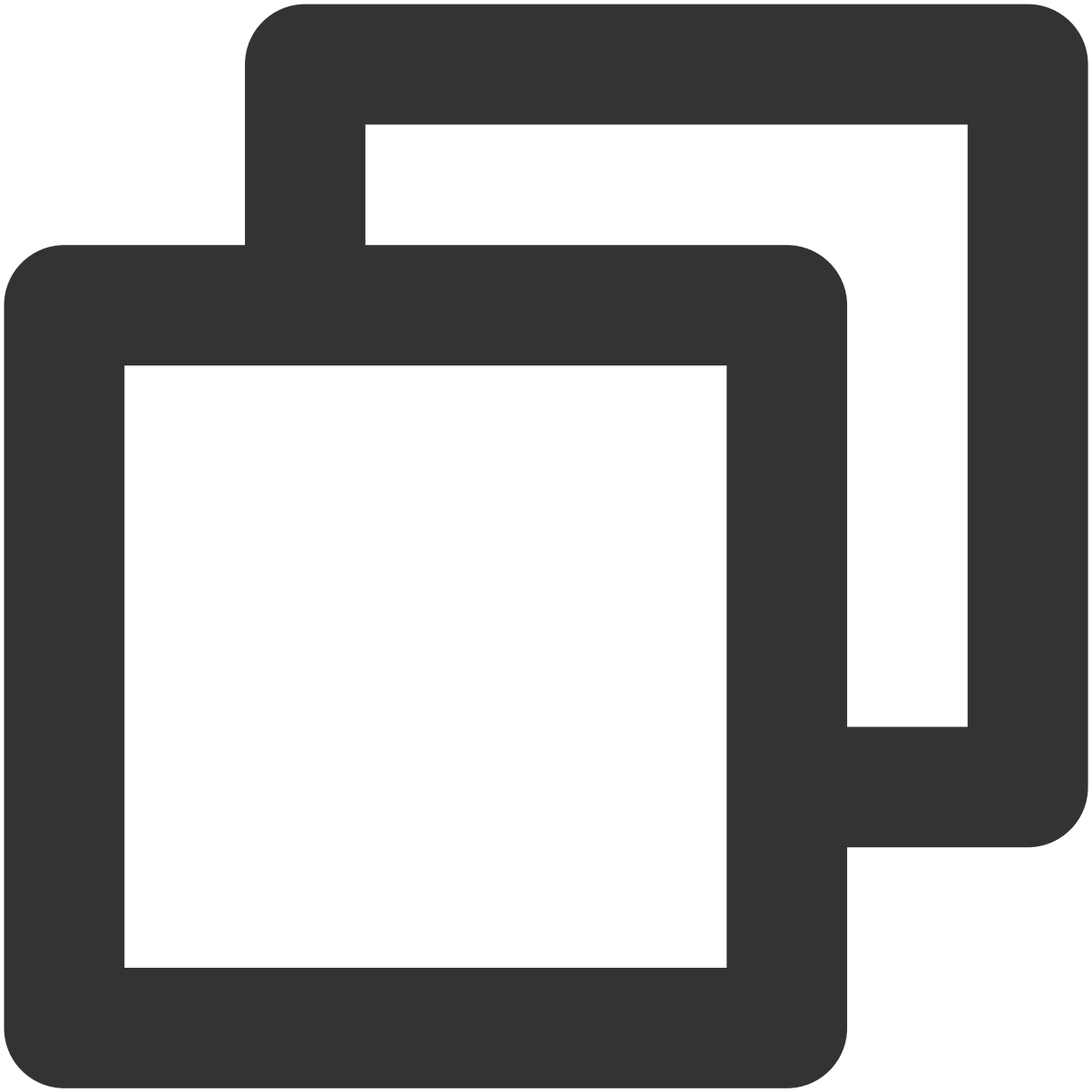
预热 Table 的指令下发后会在后台发起一个异步作业，GooseFS 会在启动作业后返回一个作业 ID，可以通过 `job stat <ID>` 指令查询任务的运行状态，同时可以通过 `table stat` 指令查看预热百分比。预热指令如下：



```
$ goosefs table load test_db web_page
Asynchronous job submitted successfully, jobId: 1615966078836
```

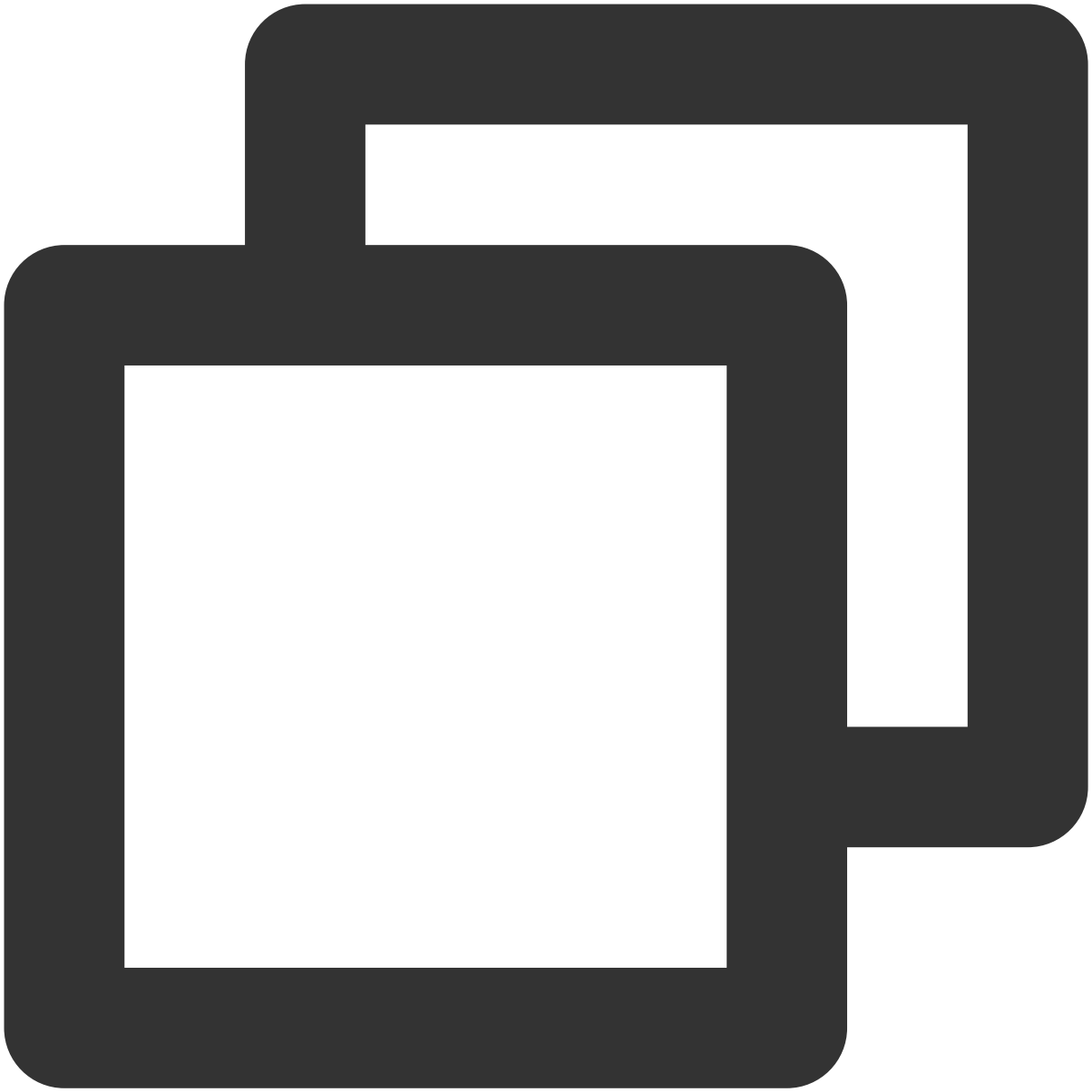
4. 查看 Table 预热情况

通过 `job stat` 指令可以查看预热 Table 作业的执行进度。当状态为 COMPLETED 时，整个预热过程完成，如果状态为 FAILED，可以在 master.log 文件中查看日志记录，排查预热错误的原因：



```
$ goosefs job stat 1615966078836
COMPLETED
```

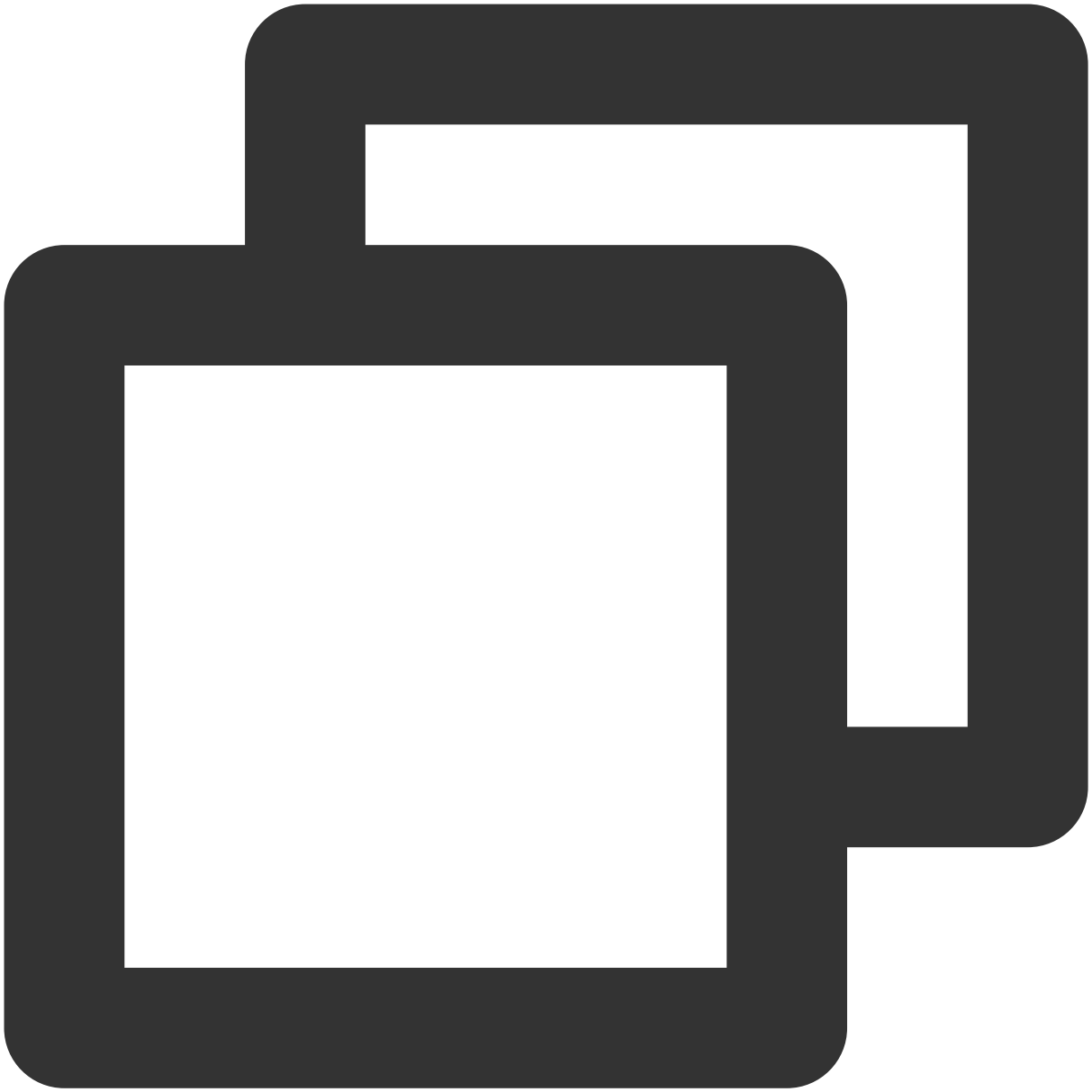
当 Table 完成预热后，可以通过 stat 指令查看指定 Table 的概况。



```
$ goosefs table stat test_db web_page
detail
```

5. 释放 Table

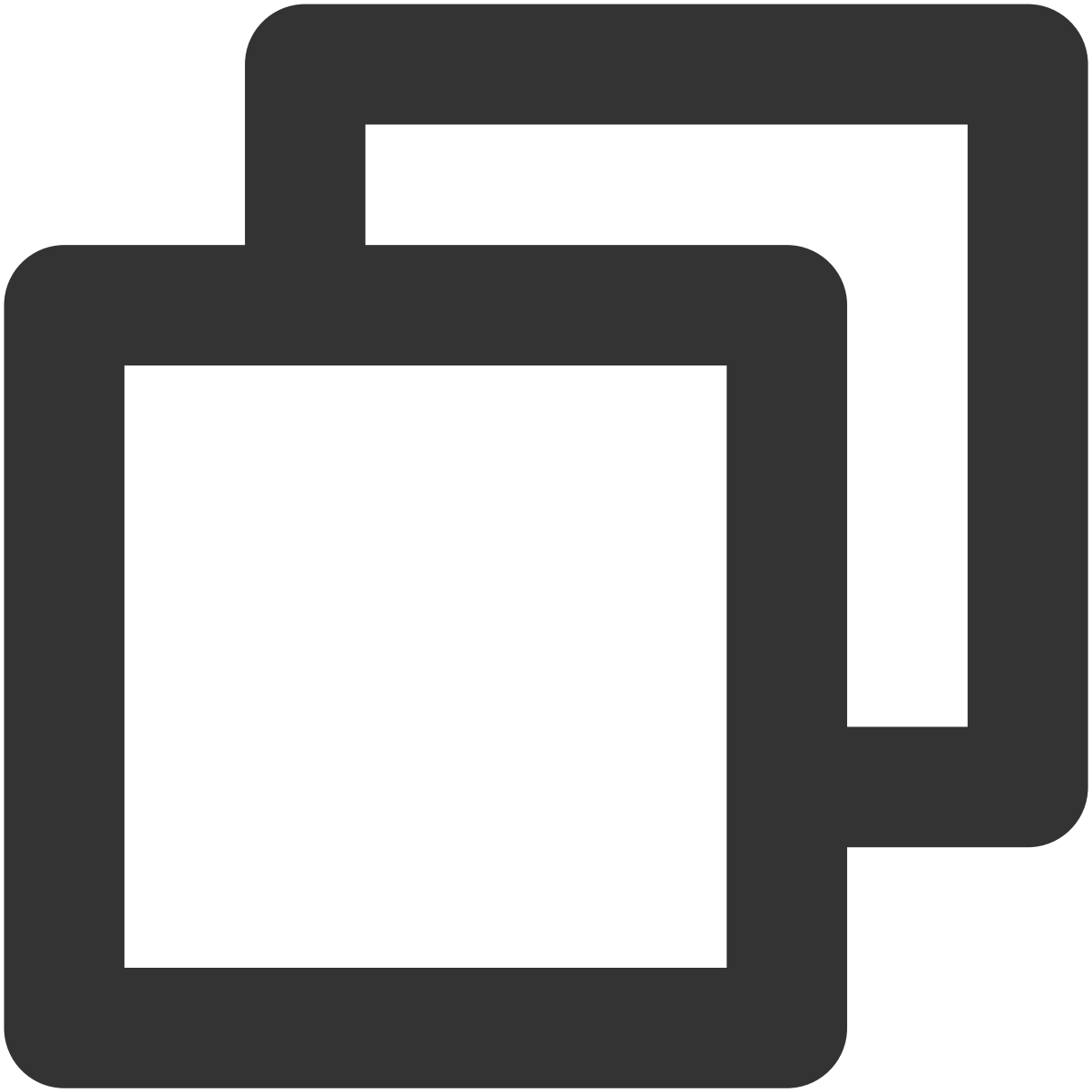
通过以下指令可以从 GooseFS 中释放指定 Table 数据缓存：



```
$ goosefs table free test_db web_page
detail
```

6. 卸载 DB

通过以下指令可以从 GooseFS 中卸载指定 DB:



```
$ goosefs table detachdb test_db
detail
```

GooseFS-FUSE 能力

最近更新时间：2024-01-06 11:17:58

GooseFS-FUSE 可以在一台 Unix 机器上的本地文件系统中挂载一个 GooseFS 分布式文件系统。通过使用该特性，一些标准的命令行工具（例如 ls、cat 以及 echo）可以直接访问 GooseFS 分布式文件系统中的数据。此外更重要的是使用不同语言实现的应用程序，例如 C、C++、Python、Ruby、Perl、Java 都可以通过标准的 POSIX 接口（例如 open、write、read）来读写 GooseFS，而不需要任何 GooseFS 的客户端整合与设置。
GooseFS-FUSE 是基于 FUSE 这个项目，并且都支持大多数的文件系统操作。但是由于 GooseFS 固有的属性，例如它的一次性、不可改变的文件数据模型，该挂载的文件系统与 POSIX 标准不完全一致，尚有一定的局限性。因此，请先阅读 局限性，从而了解该特性的作用以及局限。

安装要求

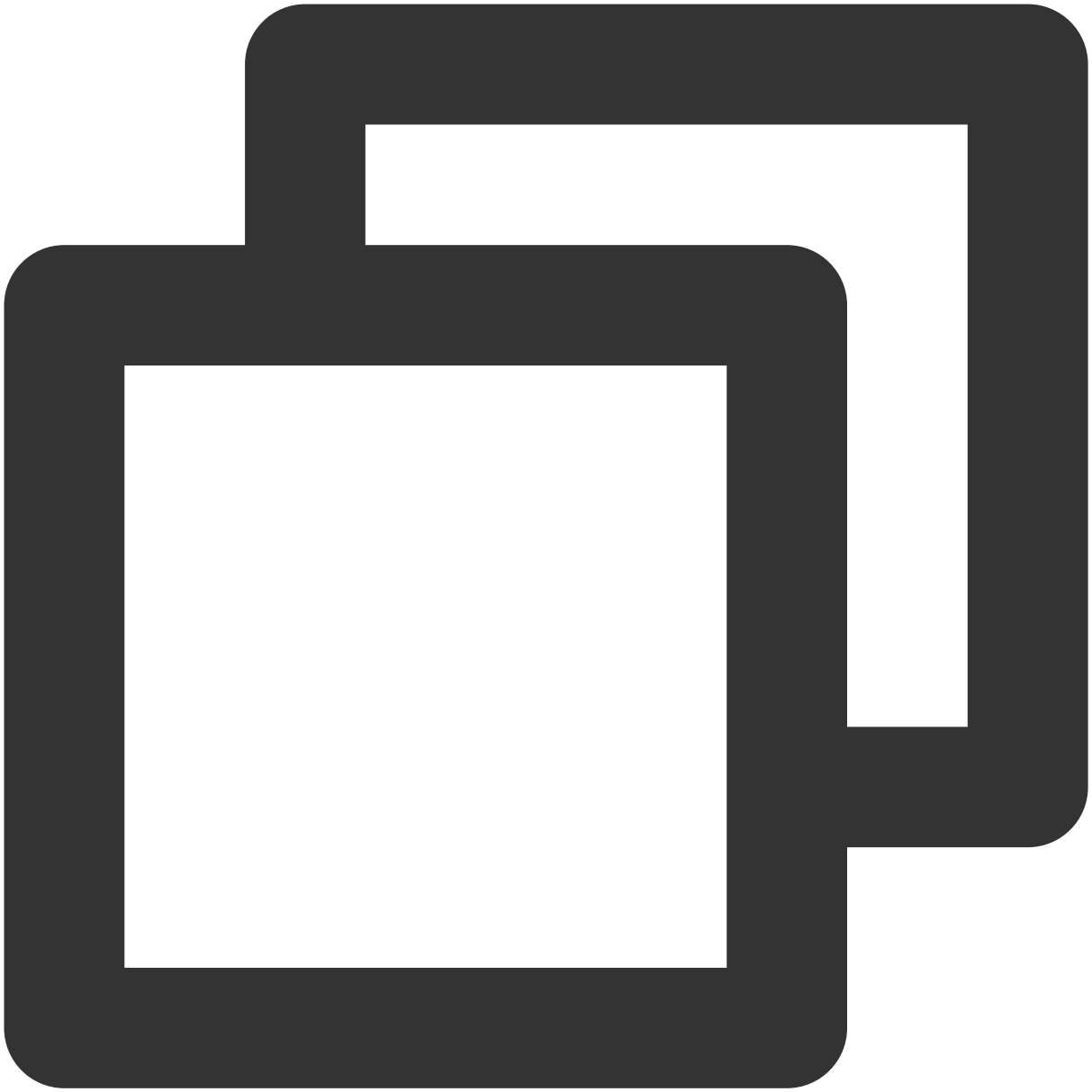
JDK 1.8及以上
Linux 系统：libfuse 2.9.3及以上（可以使用 2.8.3版本，但会提示一些警告）
MAC 系统：osxfuse 3.7.1及以上

用法

挂载 GooseFS-FUSE

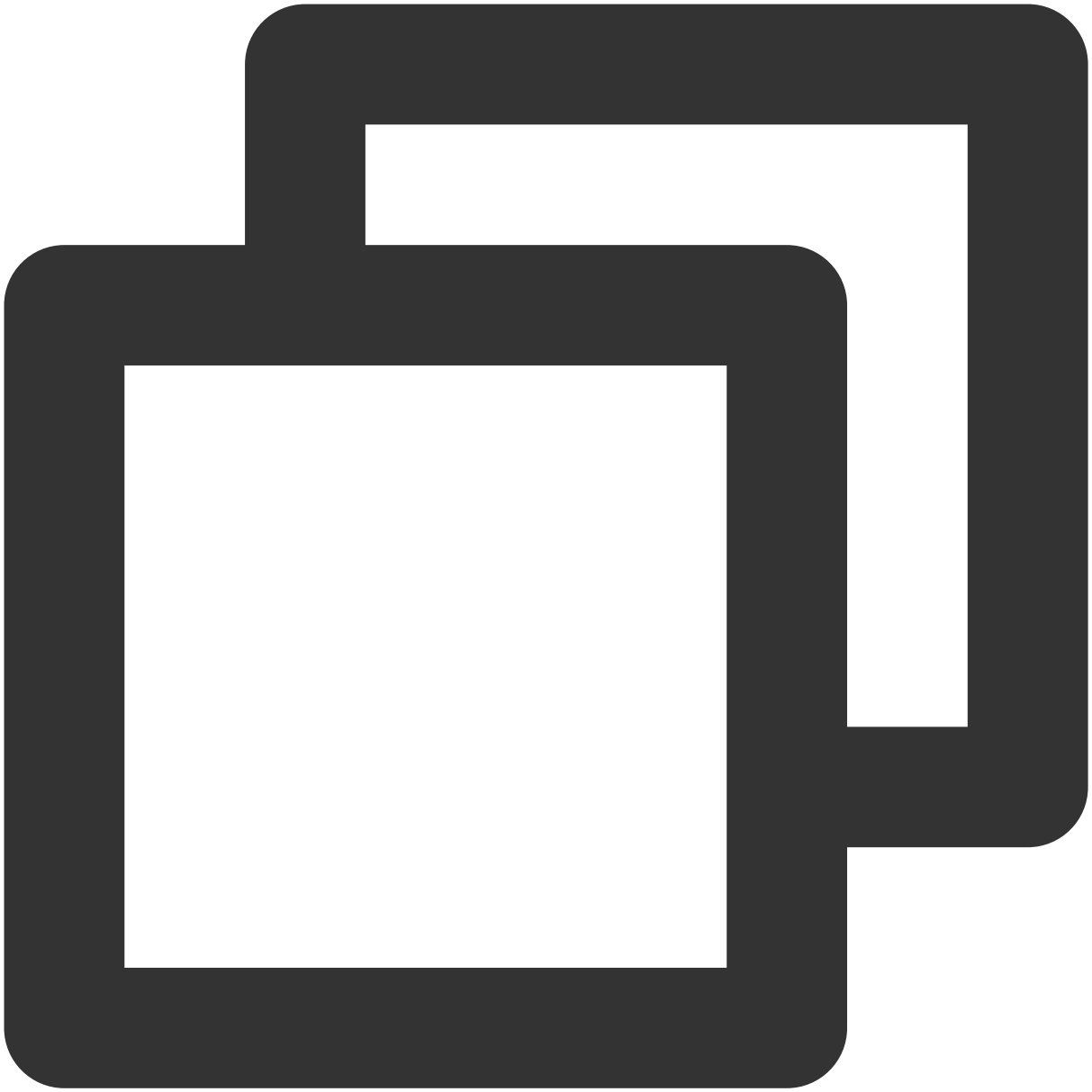
完成配置以及启动 GooseFS 集群后，在需要挂载 GooseFS 的节点上启动 Shell，并进入 $GOOSEFS_HOME 目录执行以下命令：

版权所有：腾讯云计算（北京）有限责任公司



```
$ integration/fuse/bin/goosefs-fuse mount mount_point [GooseFS_path]
```

该命令会启动一个后台 Java 进程，用于将对应的 GooseFS 路径挂载到 `<mount_point>` 指定的路径。例如以下命令，将 GooseFS 路径 /people 挂载到本地文件系统的 /mnt/people 目录下。



```
$ integration/fuse/bin/goosefs-fuse mount /mnt/people /people
Starting goosefs-fuse on local host.
goosefs-fuse mounted at /mnt/people. See /lib/GooseFS/logs/fuse.log for logs
```

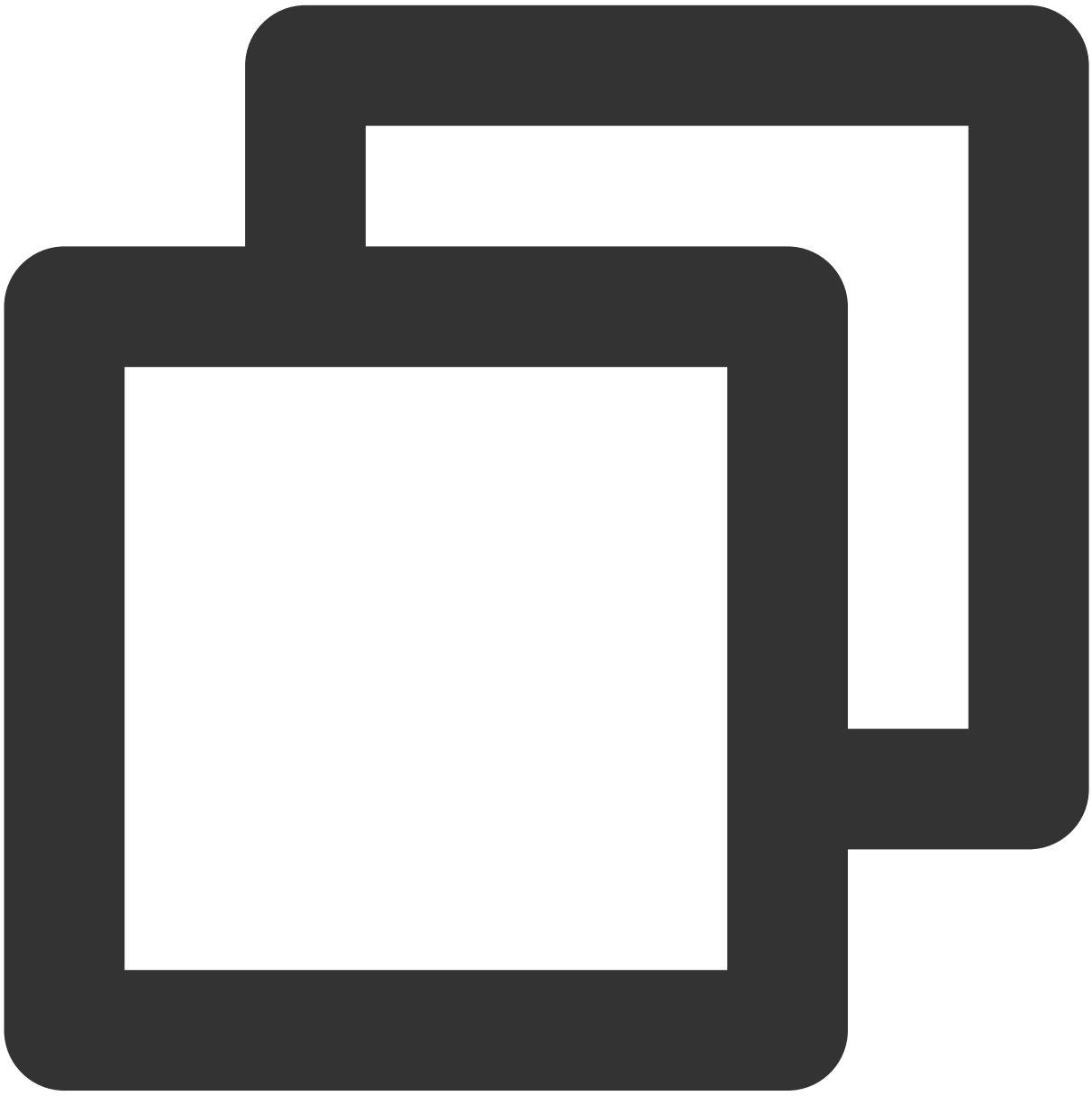
当 GooseFS_path 没有给定时，GooseFS-FUSE 会默认挂载到 GooseFS 根目录下(/)。您可以多次调用该命令，将 GooseFS 挂载到不同的本地目录下。所有的 GooseFS-FUSE 会共享 $GOOSEFS_HOME\\logs\\fuse.log 日志文件。该日志文件对于错误排查很有帮助。

注意

`<mount_point>` 必须是本地文件系统中的一个空文件夹，并且启动 GooseFS-FUSE 进程的用户拥有该挂载点及对其的读写权限。

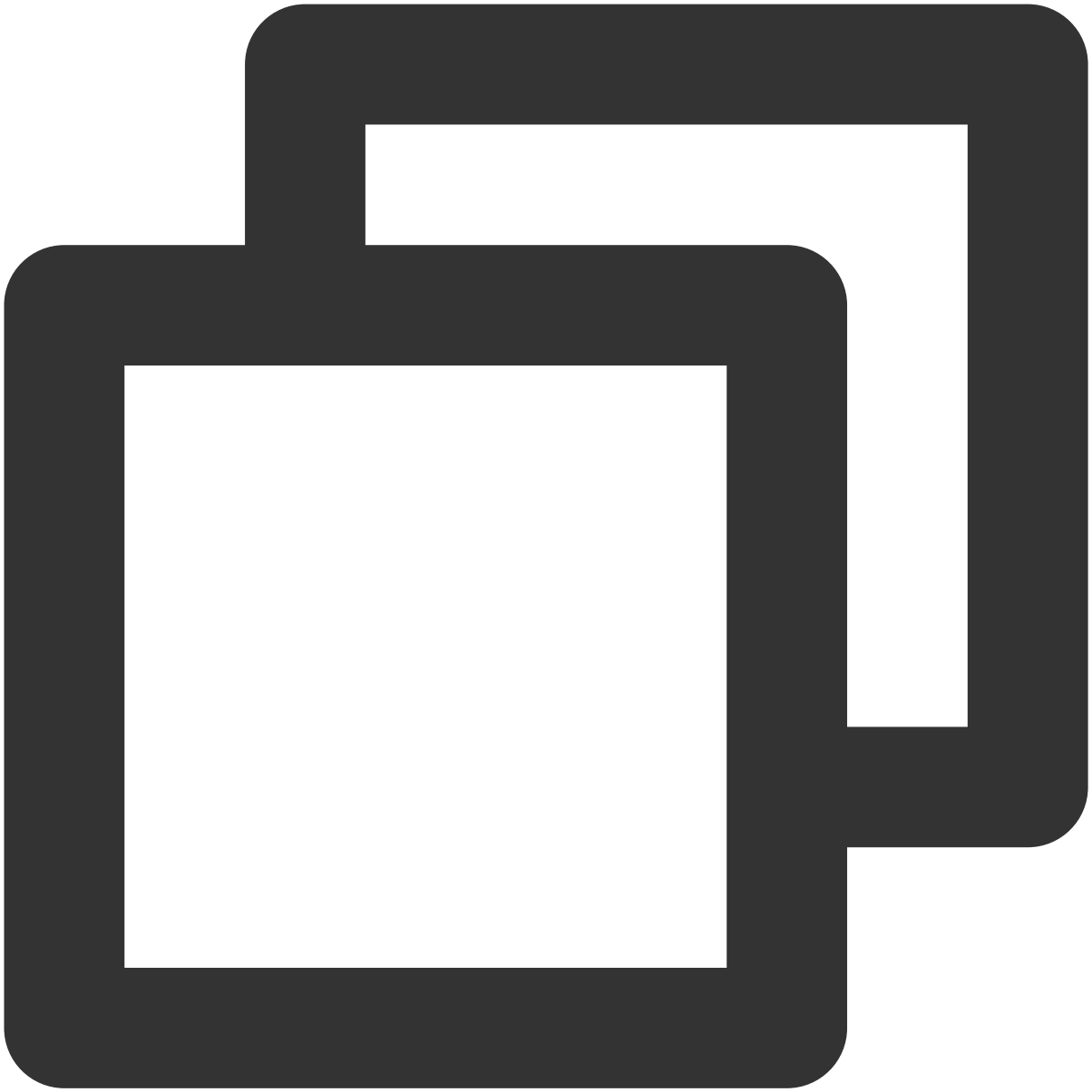
卸载 GooseFS-FUSE

卸载 GooseFS-FUSE 时，需在该节点上启动 Shell，并进入$GOOSEFS_HOME 目录执行以下命令：



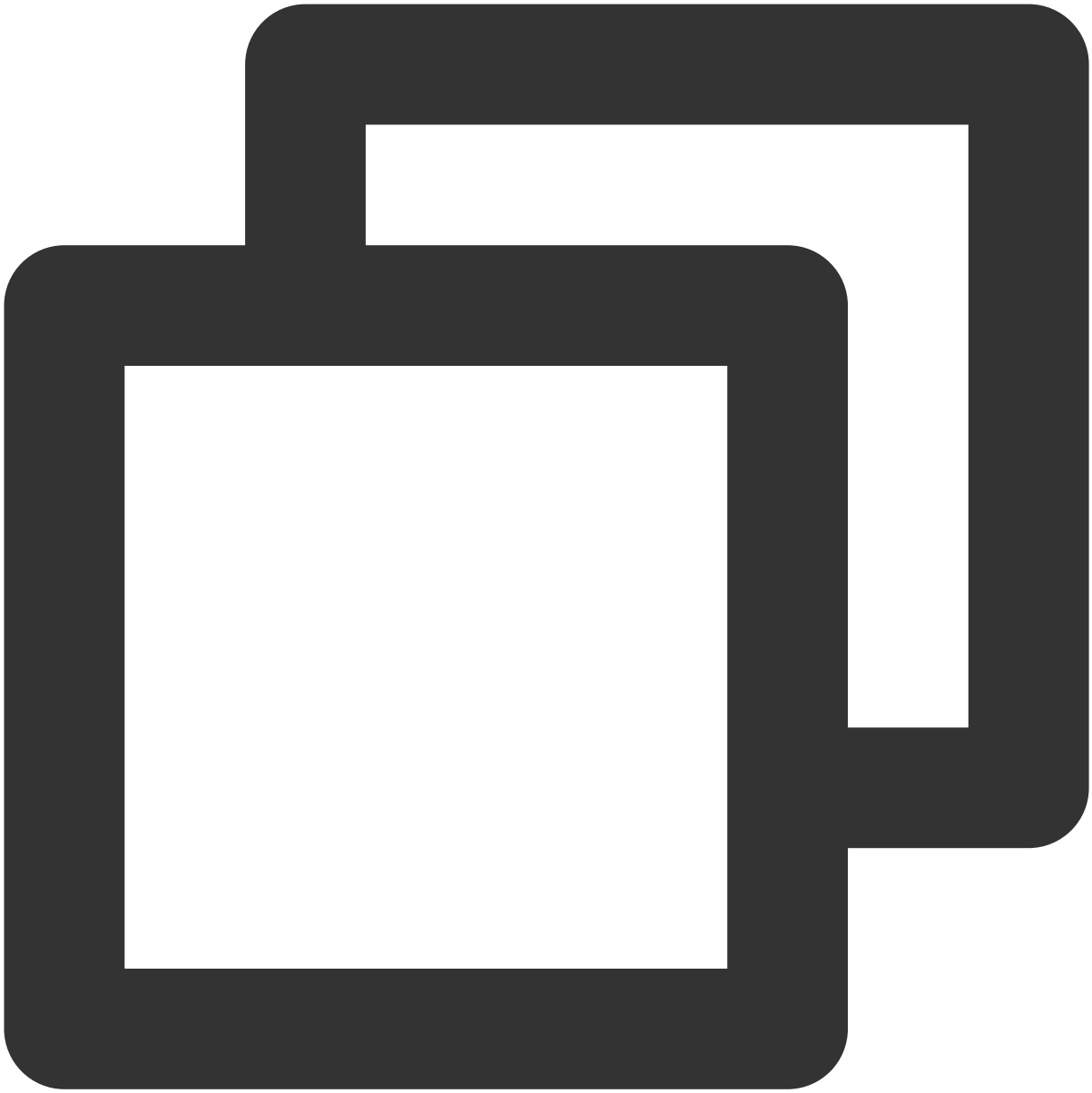
```
$ integration/fuse/bin/goosefs-fuse umount mount_point
```

该命令将终止 goosefs-fuse java 后台进程，并卸载该文件系统。例如：



```
$ integration/fuse/bin/goosefs-fuse umount /mnt/people
Unmount fuse at /mnt/people (PID: 97626).
```

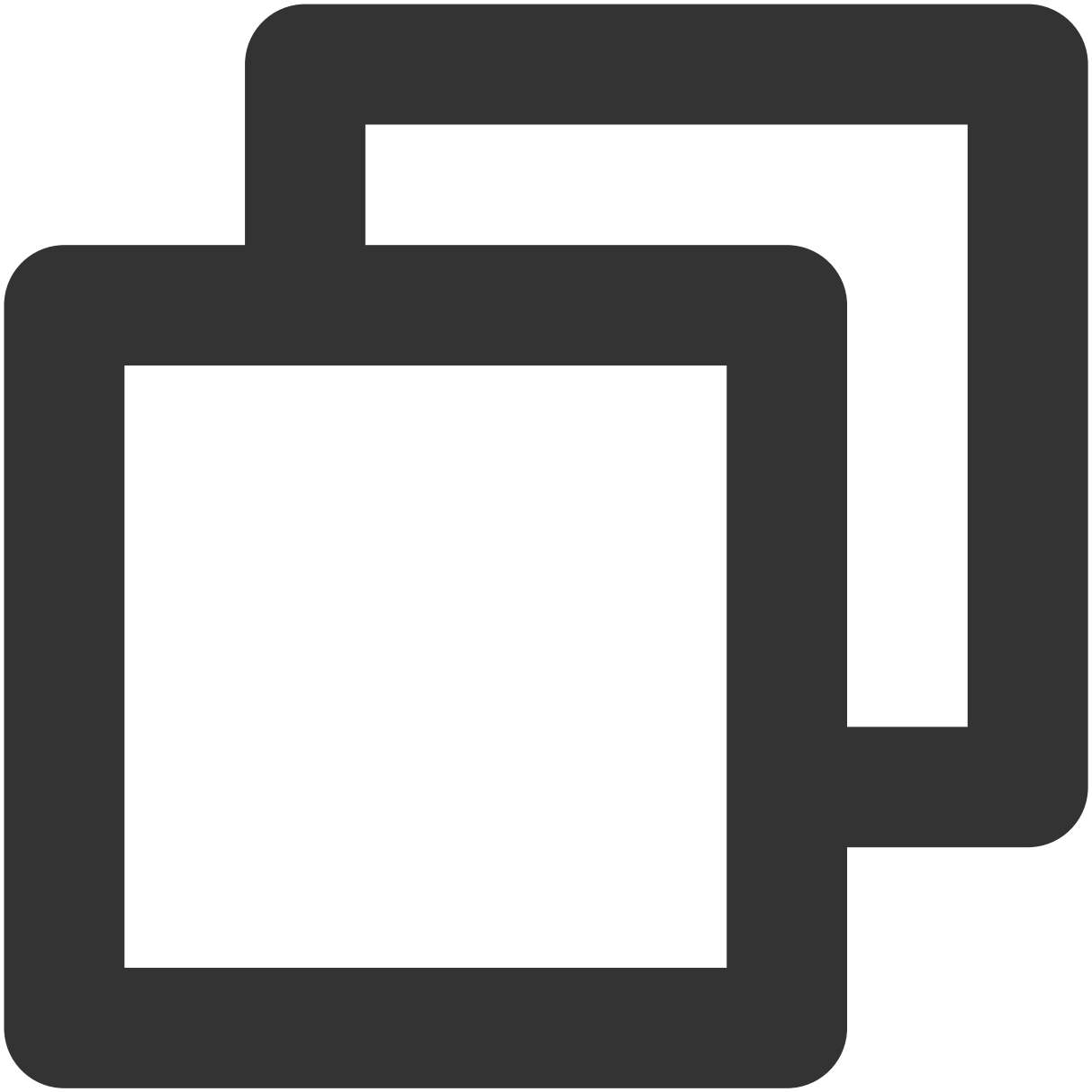
默认情况下，如果有任何读写操作未完成，unmount 操作会等待最多120s。如果120s后读写操作仍未完成，那么 Fuse 进程会被强行结束，这会导致正在读写的文件失败，您可以添加 -s 参数来避免 Fuse 进程被强行结束。例如：



```
$ ${GOOSEFS_HOME}/integration/fuse/bin/goosefs-fuse unmount -s /mnt/people
```

检查 GooseFS-FUSE 是否在运行

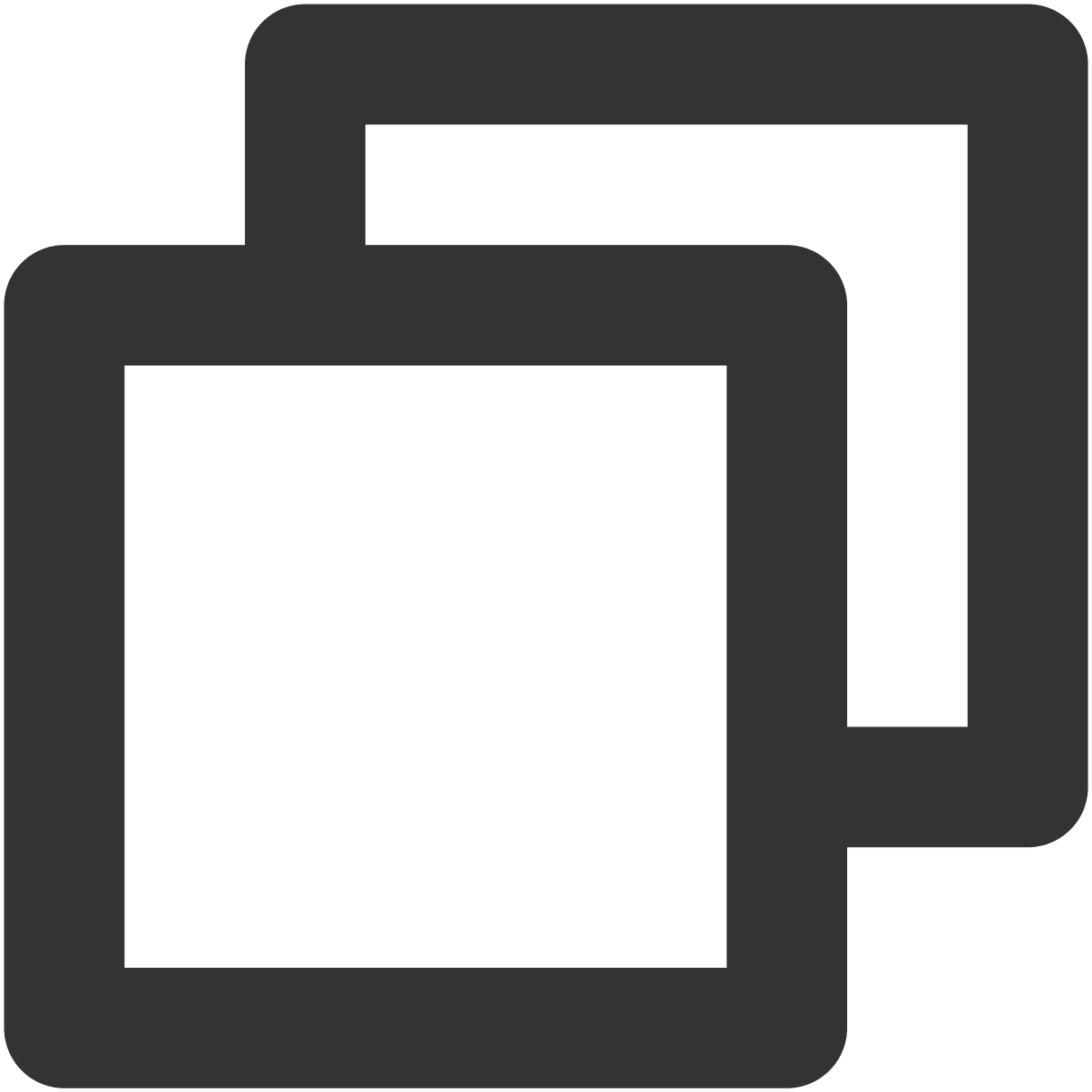
罗列所有的挂载点，需在该节点上启动 Shell，并进入 $GOOSEFS_HOME 目录执行以下命令：



```
$ integration/fuse/bin/goosefs-fuse stat
```

该命令会输出包括 pid、mount_point、GooseFS_path 在内的信息。

例如输出可以是以下格式：



```
$ pid    mount_point     GooseFS_path
80846  /mnt/people     /people
80847  /mnt/sales       /sales
```

Goosefs-FUSE 目录结构

conf 目录下：

masters：master 服务器的 IP 配置文件

workers：worker 服务器的 IP 配置文件

goosefs-site.properties：goosefs 配置文件

libexec：goosefs-fuse 运行依赖的 lib 库文件

goosefs-fuse-1.4.0：goosefs-fuse 后台运行的 jar 包

log：日志目录

可选配置

GooseFS-FUSE 基于标准的 GooseFS-core-client-fs 进行操作。如果您希望它像使用其他应用的 client 一样，自定义该 GooseFS-core-client-fs 的行为。

可通过编辑 $GOOSEFS_HOME/conf/goosefs-site.properties 配置文件来更改客户端选项。

注意

所有的更改应该在 GooseFS-FUSE 启动之前完成。

局限性

目前，GooseFS-FUSE 支持大多数基本文件系统的操作。然而，由于 GooseFS 某些内在的特性，您需要注意以下几点：

不支持对文件进行随机写入和追加写入操作。

文件只能顺序地写入一次，并且无法修改。这意味着如果要修改一个文件，您需要先删除该文件，或者 open 操作时，携带 O_TRUNC 标志符，将文件长度置为0。

不支持读取挂载点中正在写入的文件。
不支持对文件长度进行 truncate 操作。
不支持 soft/hard link；GooseFS 没有 hard-link 和 soft-link 的概念，所以不支持与之相关的命令，例如 ln。此外关于 hard-link 的信息也不在 ll 的输出中显示。
以对象存储作为底层存储时，Rename 操作非原子。
只有当 GooseFS 的 GooseFS.security.group.mapping.class 选项设置为ShellBasedUnixGroupsMapping 的值时，文件的用户与分组信息才与 Unix 系统的用户分组对应。否则 chown 与 chgrp 的操作不生效，而 ll 返回的用户与分组为启动 GooseFS-FUSE 进程的用户与分组信息。

性能考虑

由于 FUSE 和 JNR 的配合使用，与直接使用原生文件系统 API 相比，使用挂载文件系统的性能会相对较差。
大多数性能问题的原因在于，每次进行 read 或 write 操作时，内存中都存在若干个副本，并且 FUSE 将写操作的最大粒度设置为128KB。其性能可以利用 kernel 3.15 引入的 FUSE 回写（write-backs）缓存策略从而得到大幅提高（但 libfuse 2.x 用户空间库目前尚不支持该特性）。

GooseFS-FUSE 配置参数

以下是 GooseFS-FUSE 相关的配置参数：

参数	默认值	描述
goosefs.fuse.cached.paths.max	500	定义内部 GooseFS-FUSE 缓存的大小，该缓存维护本地文件系统路径和 Alluxio 文件 URI 之间最常用的转换。
goosefs.fuse.debug.enabled	false	允许 FUSE 调试输出，该输出会被重定向到 `goosefs.logs.dir` 指定目录中的 `fuse.out` 日志文件。
goosefs.fuse.fs.name	goosefs-fuse	FUSE 挂载文件系统使用的描述性名称。
goosefs.fuse.jnifuse.enabled	true	使用 JNI-Fuse 库以获得更好的性能。 如果禁用，将使用 JNR-Fuse。
goosefs.fuse.shared.caching.reader.enabled	false	（实验性）使用共享 grpc 数据读取器，通过 GooseFS JNI Fuse 在多进程文件读取中获得更好的性能。块数据将缓存在客户端，因此 Fuse 进程需要更多内存。

goosefs.fuse.logging.threshold	10s	当花费的时间超过阈值时，记录 FUSE API 调用。
goosefs.fuse.maxwrite.bytes	131072	FUSE 写操作的粒度（bytes），注意目前 128KB 是 Linux 内核限制的上界。
goosefs.fuse.user.group.translation.enabled	false	是否在 FUSE API 中将 GooseFS 的用户与组转化为对应的 Unix 用户与组。当设为 false 时，所有 FUSE 文件的用户与组将会显示为挂载 goosefs-fuse 线程的用户与组。

常见问题

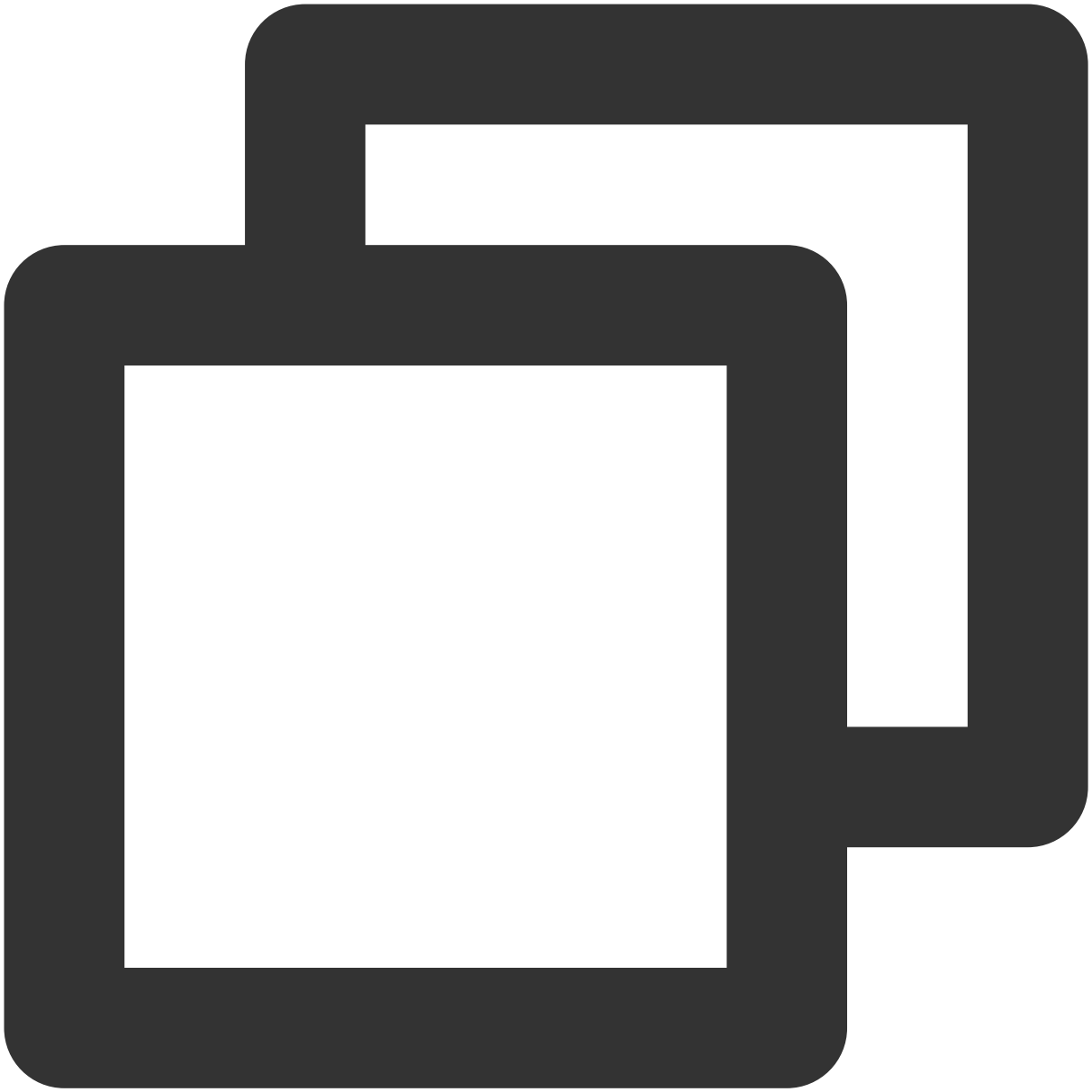
缺少 libfuse 库文件

在您执行 GooseFS-Fuse 挂载之前，需要安装 libfuse。

```
2021-10-13 20:04:42,970 INFO  TieredIdentityFactory - Initialized tiered identity TieredIdentity(node=10.91.2
2021-10-13 20:04:43,560 ERROR GooseFSFuse - launch fuse failed:
java.io.IOException: Failed to mount GooseFS file system
        at com.qcloud.cos.goosefs.fuse.GooseFSFuse.launchFuse(GooseFSFuse.java:192)
        at com.qcloud.cos.goosefs.fuse.GooseFSFuse.main(GooseFSFuse.java:126)
Caused by: java.lang.UnsatisfiedLinkError: /private/var/folders/5_/sp1cwpdd1xn9w96x5xkw7qnc0000gn/T/libjnifuse
5_/sp1cwpdd1xn9w96x5xkw7qnc0000gn/T/libjnifuse957879065968834264.jnilib, 1): Library not loaded: /usr/local/l
  Referenced from: /private/var/folders/5_/sp1cwpdd1xn9w96x5xkw7qnc0000gn/T/libjnifuse957879065968834264.jnil
  Reason: image not found
        at java.lang.ClassLoader$NativeLibrary.load(Native Method)
        at java.lang.ClassLoader.loadLibrary0(ClassLoader.java:1934)
        at java.lang.ClassLoader.loadLibrary(ClassLoader.java:1817)
        at java.lang.Runtime.load0(Runtime.java:810)
        at java.lang.System.load(System.java:1086)
        at com.qcloud.cos.goosefs.jnifuse.utils.NativeLibraryLoader.loadLibraryFromJar(NativeLibraryLoader.jav
        at com.qcloud.cos.goosefs.jnifuse.utils.NativeLibraryLoader.loadLibrary(NativeLibraryLoader.java:83)
        at com.qcloud.cos.goosefs.jnifuse.LibFuse.loadLibrary(LibFuse.java:48)
        at com.qcloud.cos.goosefs.jnifuse.LibFuse.<clinit>(LibFuse.java:32)
        at com.qcloud.cos.goosefs.jnifuse.AbstractFuseFileSystem.<clinit>(AbstractFuseFileSystem.java:41)
        at com.qcloud.cos.goosefs.fuse.GooseFSFuse.launchFuse(GooseFSFuse.java:154)
        ... 1 more
```

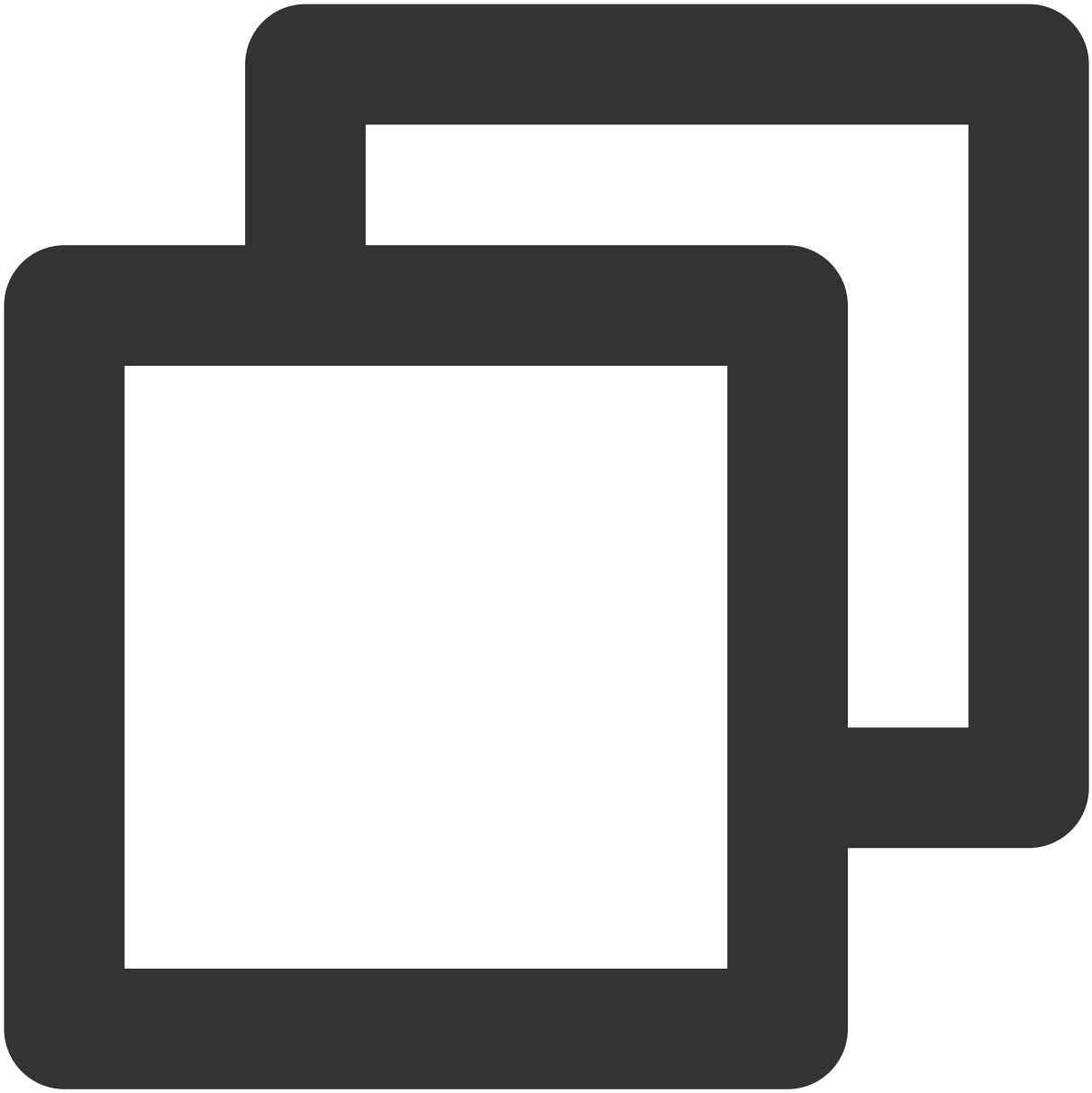
方式一

安装命令：



```
yum install fuse-devel
```

查看是否安装成功：



```
find / -name libfuse.so*
```

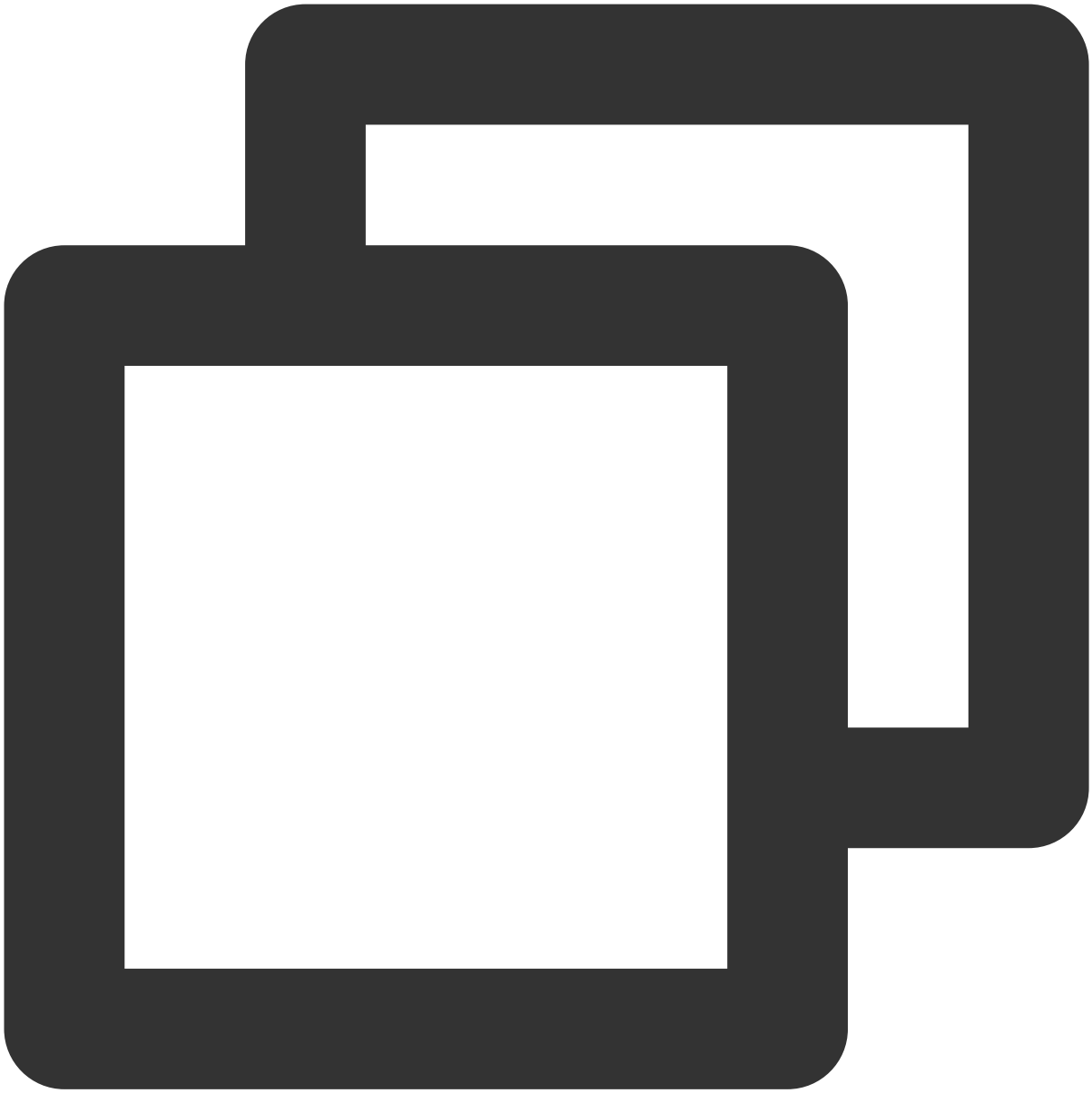
方式二

更新旧版本 libfuse.so.2.9.2，安装步骤如下：

说明

在 CentOS 7安装 libfuse，CentOS 7默认安装的是 libfuse.so.2.9.2。

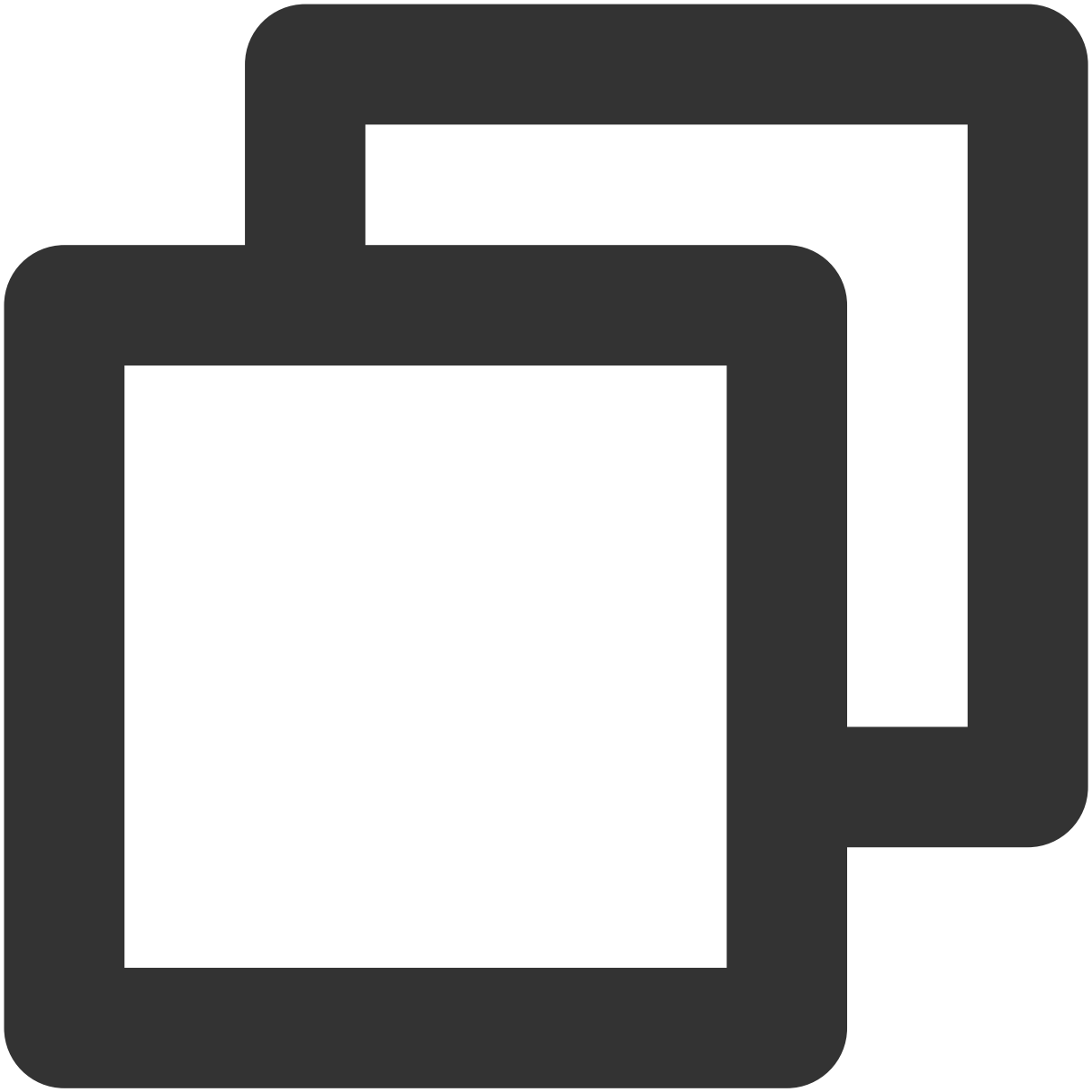
首先，下载 libfuse 源码，并编译生成 libfuse.so.2.9.7。



```
tar -zxvf fuse-2.9.7.tar.gz
cd fuse-2.9.7/ && ./configure && make && make install
echo -e '\\n/usr/local/lib' >> /etc/ld.so.conf
ldconfig
```

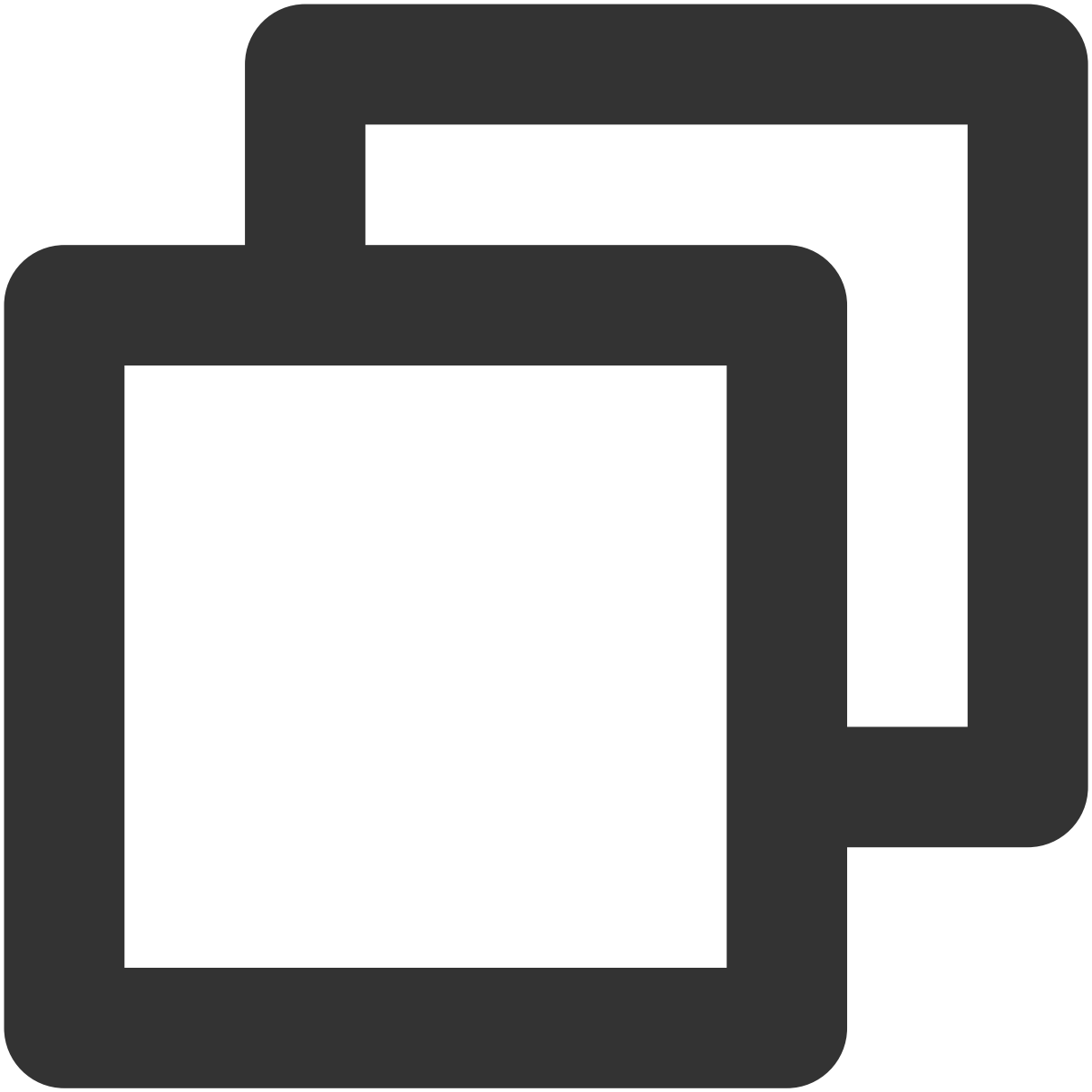
其次，下载、编译及生成 libfuse.so.2.9.7 后，然后按照以下步骤进行安装替换。

1.1 执行以下命令，查找旧版本 libfuse.so.2.9.2 库链接。



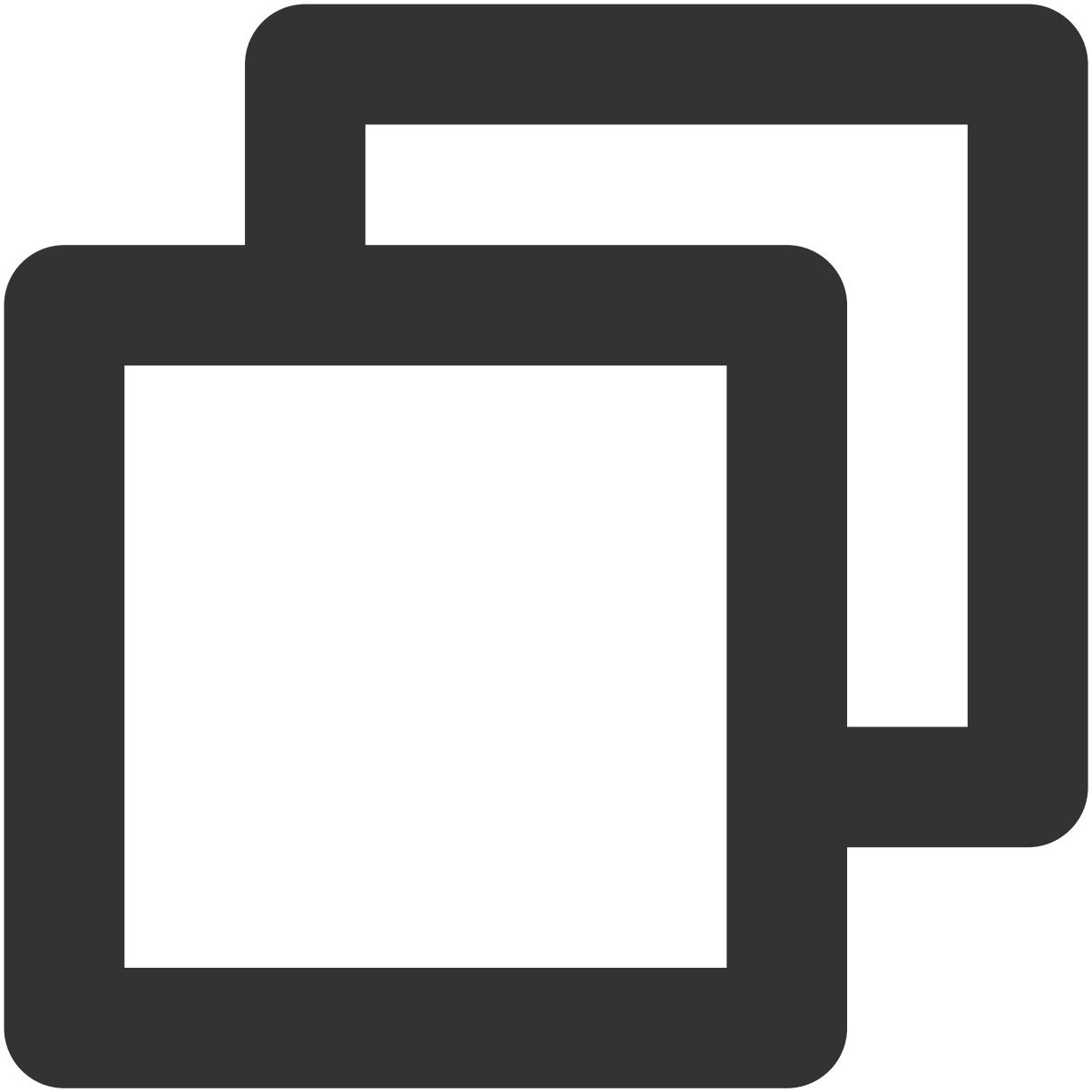
```
find / -name libfuse.so*
```

1.2 执行以下命令，将 libfuse.so.2.9.7拷贝至旧版本库 libfuse.so.2.9.2 所在位置。



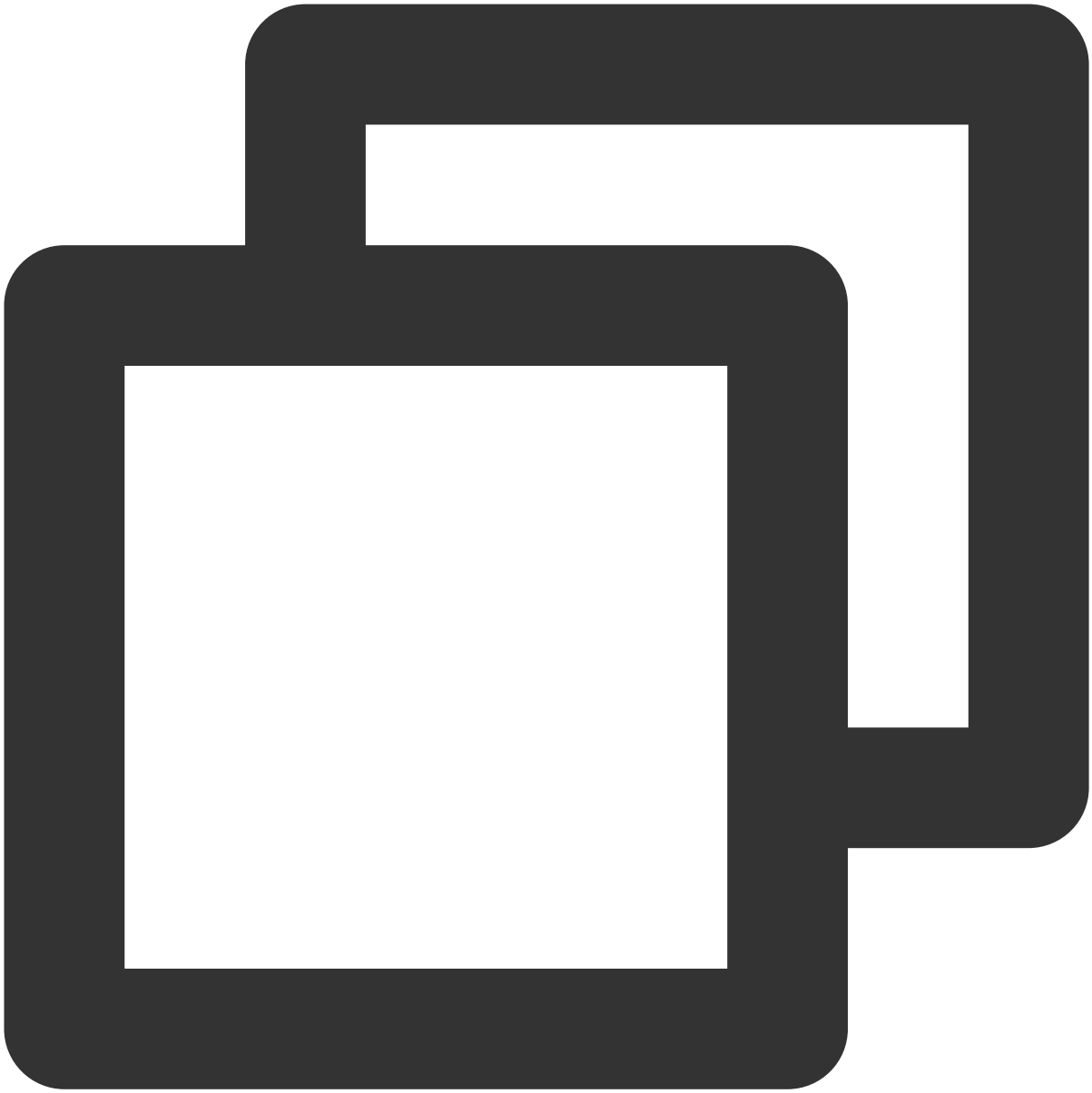
```
cp /usr/local/lib/libfuse.so.2.9.7 /usr/lib64/
```

1.3 执行以下命令， 删除旧版本 libfuse.so 库的所有链接。



```
rm -f /usr/lib64/libfuse.so
rm -f /usr/lib64/libfuse.so.2
```

1.4 执行以下命令，建立与被删除旧版本链接类似的 libfuse.so.2.9.7 库链接。



```
ln -s /usr/lib64/libfuse.so.2.9.7 /usr/lib64/libfuse.so
ln -s /usr/lib64/libfuse.so.2.9.7 /usr/lib64/libfuse.so.2
```

VIM 编辑挂载点中的文件报错 Write error in swap file？

您可以通过更改 VIM 的配置，使用 VIM 7.4 及之前的版本，对 GooseFS-Fuse 挂载点中的文件进行编辑。VIM 的 swap 文件用于暂存用户对文件的修改内容，方便 VIM 在机器宕掉重启后，可以根据 swap 文件，对未保存的修改内容进行恢复。上面的报错是由于 VIM 对 swap 文件，涉及随机写入操作，而 GooseFS 不支持对文件的随机写入。解

决方法：您可以通过在 VIM 编辑窗口中，执行如下的 VIM 命令关闭 swap 文件生成，`:set noswapfile`，或者在配置文件 ~/.vimrc 中添加配置行 `set noswapfile`。

版权所有：腾讯云计算（北京）有限责任公司

部署指南
使用自建集群部署

最近更新时间：2024-01-06 11:17:59

本文主要介绍如何规范化地在单机、集群以及腾讯云 EMR 集群（暂未集成支持 GooseFS 的版本）中部署、配置以及运行 GooseFS。

部署环境要求

硬件环境

目前 GooseFS 支持主流 x86/x64 架构的 Linux/MacOS 运行环境（**注意：MacOS M1处理器未验证支持**）。详细的运行节点配置如下：

Master 节点

CPU：工作主频1GHz 及以上，考虑到 Master 需要处理大量，建议生产环境使用多核架构处理器。
物理内存：1GB 及以上，生产环境建议使用8GB 以上的物理内存配置。
磁盘：20GB 及以上，建议生产环境配备高性能的 NVME SSD 盘作为元数据缓存盘（RocksDB 模式）。

Worker 节点

CPU：工作主频1GHz 及以上，考虑到 Worker 节点也需要处理大量的并发请求，同样建议生产环境使用多核架构处理器。
物理内存：2GB 及以上，生产环境建议根据性能需求选用合适的物理内存配置。例如，生产环境中需要将大量的数据块缓存到 Worker 节点的内存中或是采用 MEM + SSD/HDD 的混合存储时，需要根据实际可能会缓存到内存中的数据量来配备物理内存。无论使用何种缓存模式，生产环境都建议 Worker 节点配备16GB 及以上的物理内存。
磁盘：20GB 及以上 SSD/HDD 盘，建议根据实际生产环境中所需要预热缓存的数据量大小来估计每个 Worker 节点所需配置的磁盘空间。另外，为了更好的性能体验，建议配备采用 NVME 接口的 SSD 盘作为 Worker 节点的数据盘。

软件环境

Red Hat Enterprise Linux 5.5+、Ubuntu Linux 12.04 LTS+（不使用批量部署时可以支持）、CentOS 7.4+以及 TLinux 2.x（Tencent Linux 2.x），基于 Intel 架构 MacOS 10.8.3 以上版本。腾讯云 CVM 用户推荐使用 CentOS 7.4、Tencent（TLinux 2.x）或 TencentOS Server 2.4 版本的操作系统。
OpenJDK 1.8/Oracle JDK 1.8，需要注意的是，未验证支持 JDK 1.9 及以上版本。
支持 SSH 工具集（包括 SSH 和 SCP 工具）、Expect Shell（使用批量部署时需要）工具以及 Yum 包管理工具。

集群网络环境

版权所有：腾讯云计算（北京）有限责任公司

Master/Worker 节点之间外网或内网互通。使用批量部署时，要求 Master 能够正常访问外网软件源，或正确配置包管理系统的内网软件源。
集群能够通过外网或内网访问到 COS 存储桶或者 CHDFS 文件系统。

安全组与用户权限要求

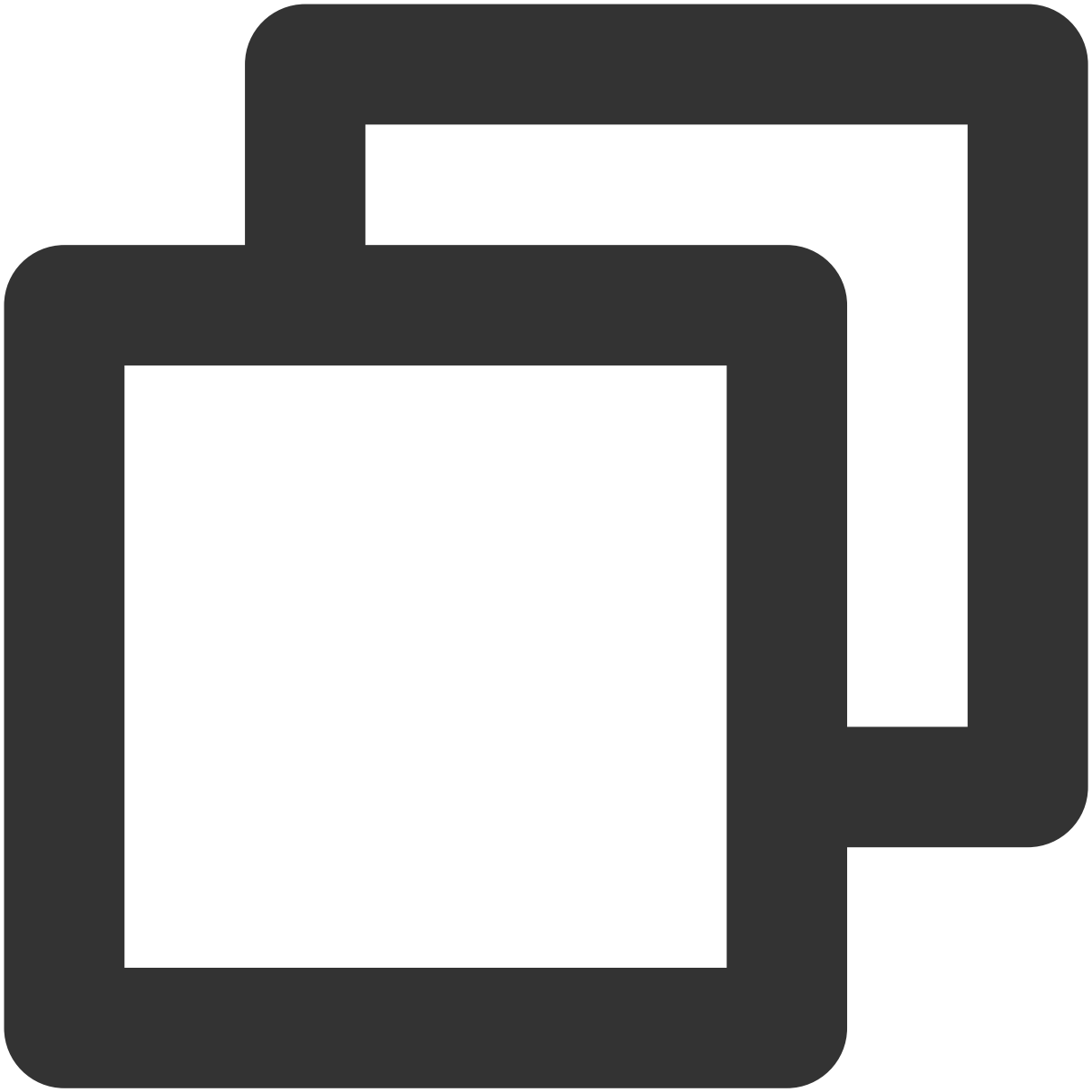
一般情况，GooseFS 建议用户使用专用的 Linux 账户部署运行 GooseFS， 例如在自建集群和 EMR 环境中，可以统一使用 hadoop 用户来部署运行 GooseFS。在批量部署中，则需要补充以下用户能力和权限：
具备切换到 root 或使用 sudo 权限的能力；
部署运行账户具备读写安装目录的权限；
Master 节点具备 SSH 登录到集群中所有 Worker 节点的权限；
集群中对应的节点角色需要放开对应的端口：Master（9200和9201）、Worker（9203和9204）、Job Master(9205、9206和9207)、Job Worker（9208、9209和9210）、Proxy（9211）以及 LogServer（9212）。

单机模式部署运行（伪分布式架构）

伪分布式架构部署主要适用于体验和调试 GooseFS，初级用户可以在自己的 Linux 或 Mac 系统主机上快速体验和调试 GooseFS。

部署

1. 首先 下载 GooseFS 的二进制分发包。
2. 分发包下载后解压，进入到 GooseFS 的目录中，执行如下操作：
通过拷贝 conf/goosefs-site.properties.template 创建 conf/goosefs-site.properties 配置文件：



```
$ cp conf/goosefs-site.properties.template conf/goosefs-site.properties
```

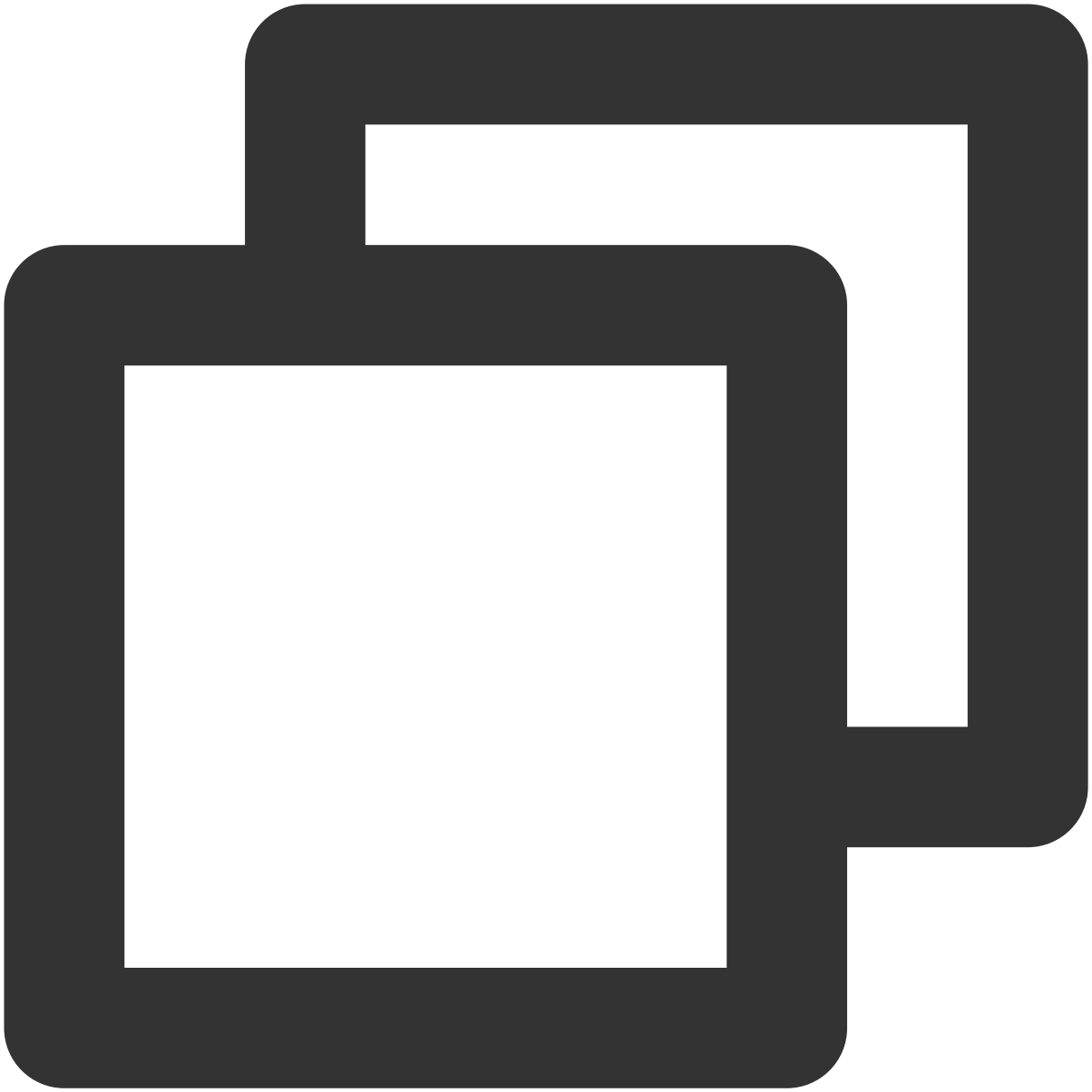
在 conf/goosefs-site.properties 配置文件中设置 `goosefs.master.hostname` 为 `localhost` 。

在 conf/goosefs-site.properties 配置文件中设置 `goosefs.master.mount.table.root.ufs` 为本地文件系统中的目录，例如：/tmp 或是 /data/goosefs 等。

建议配置 localhost 的 SSH 免密登录，否则格式化和启动等操作，需要输入 localhost 的登录密码。

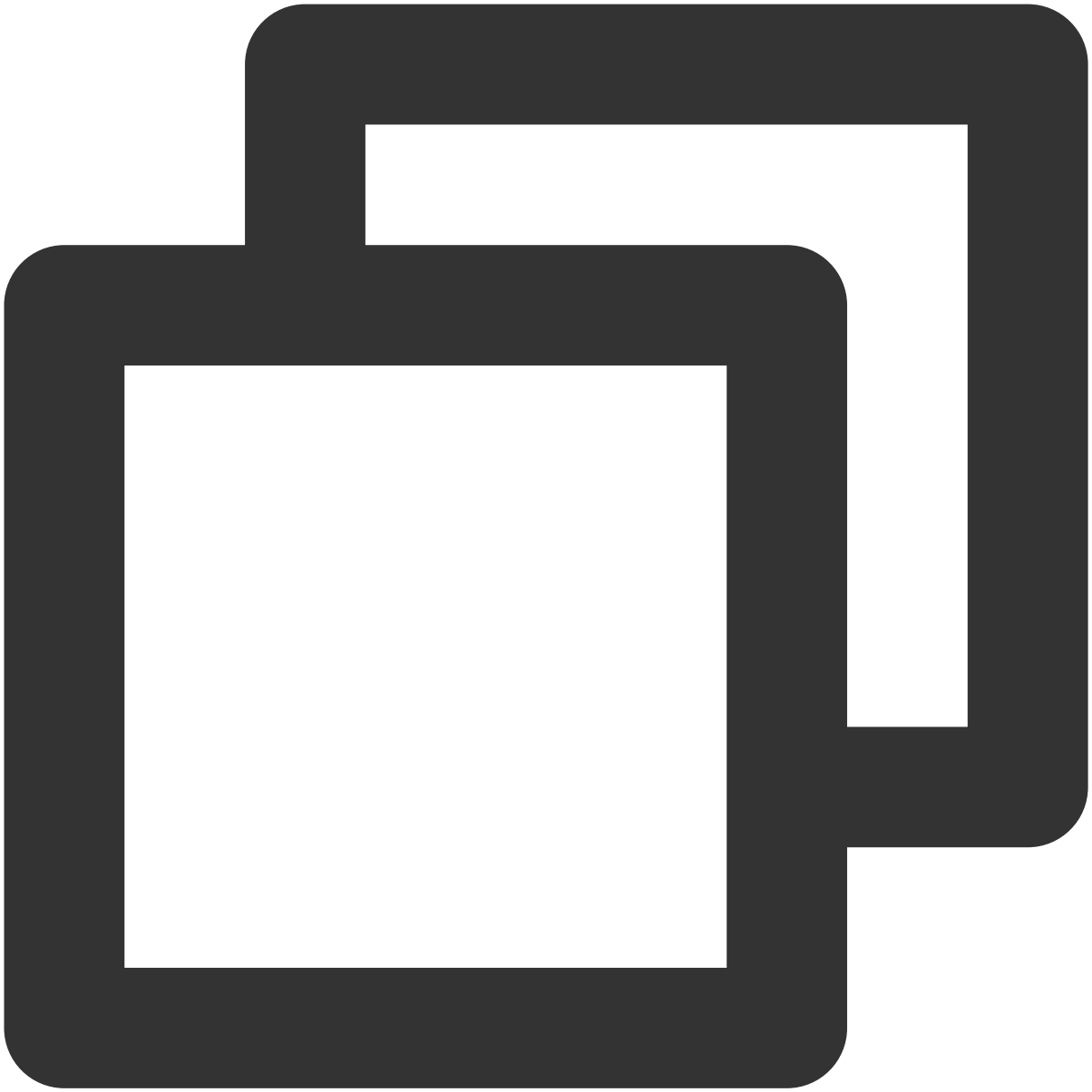
运行

执行如下命令，就可以完成挂载一个 RamFS 文件系统：



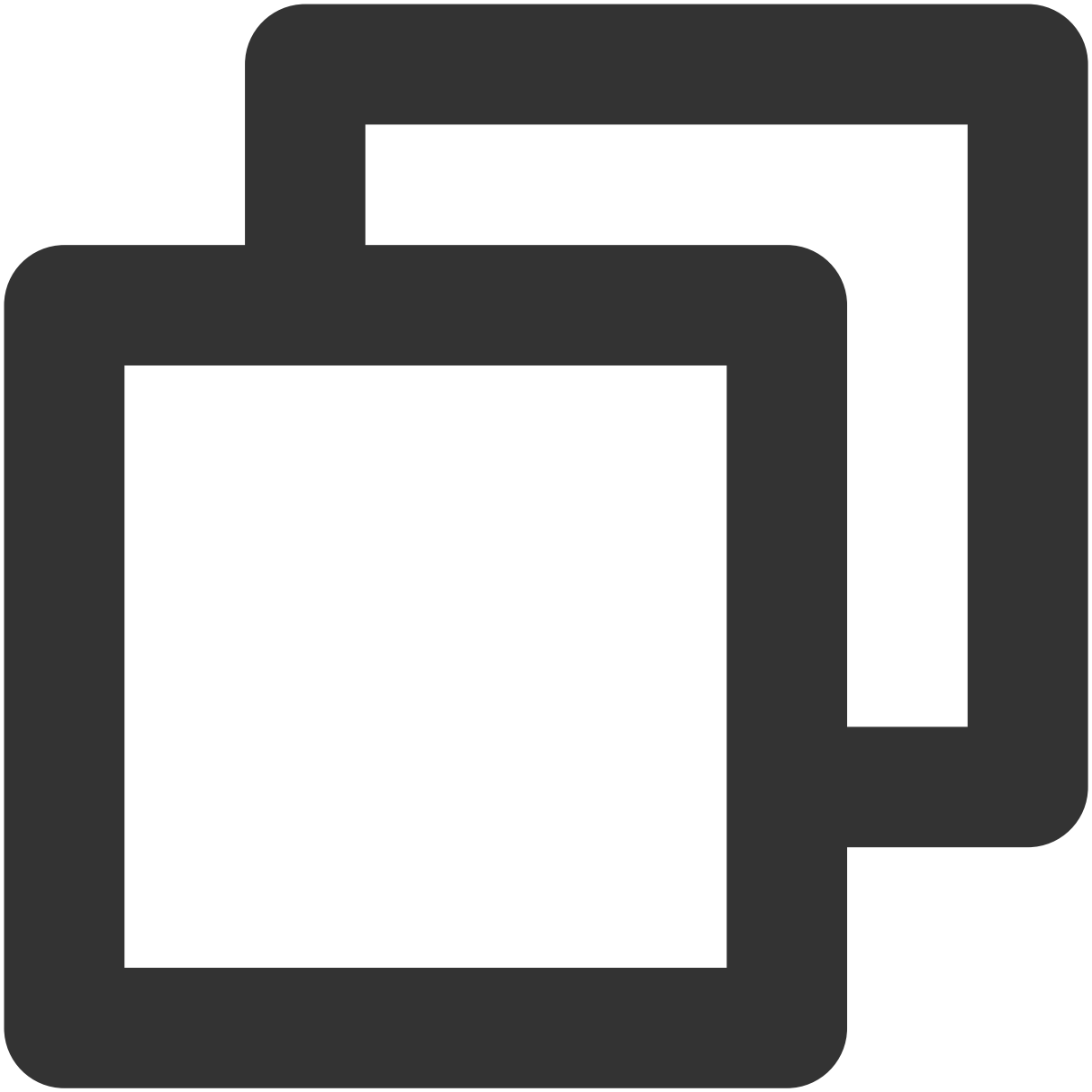
```
$ ./bin/goosefs-mount.sh SudoMount
```

同样，也可以不执行上面的命令，而直接在启动 GooseFS 集群时挂载：



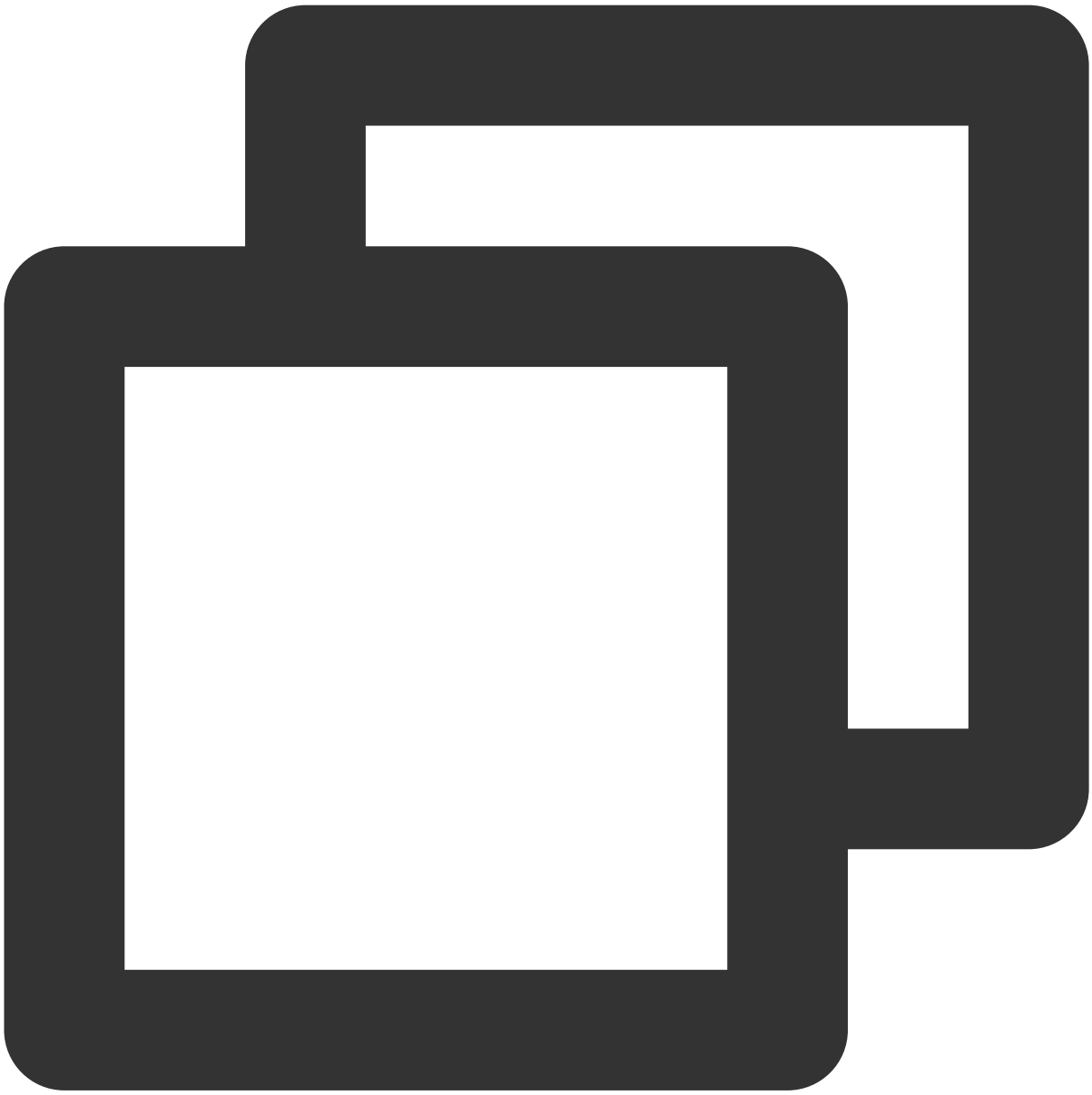
```
$ ./bin/goosefs-start.sh local SudoMount
```

当启动后，使用 `jps` 命令，能够看到伪分布式模式下，GooseFS 的所有进程：



```
$ jps
35990 Jps
35832 GooseFSSecondaryMaster
35736 GooseFSMaster
35881 GooseFSWorker
35834 GooseFSJobMaster
35883 GooseFSJobWorker
35885 GooseFSProxy
```

然后就可以使用 `goosefs` 命令行，执行 namespace、fileSystem、job 以及 table 的各项操作了。例如，上传一个本地文件到 GooseFS 中，并且列出当前 GooseFS 中根目录下的文件和目录：



```
$ goosefs fs copyFromLocal test.txt /
Copied file:///Users/goosefs/test.txt to /

$ goosefs fs ls /
-rw-r--r--  goosefs         staff                        0       PERSISTED 04-28-20
```

GooseFS 的命令行为用户提供了管理和访问 GooseFS 所有命令行接口，包括命名空间（namespace）、表（table）、作业（job）以及常用的文件系统（fs）操作等，具体可参考我们的官方文档或使用 `goosefs -h` 命令行参数获得详细帮助信息。

集群模式部署运行

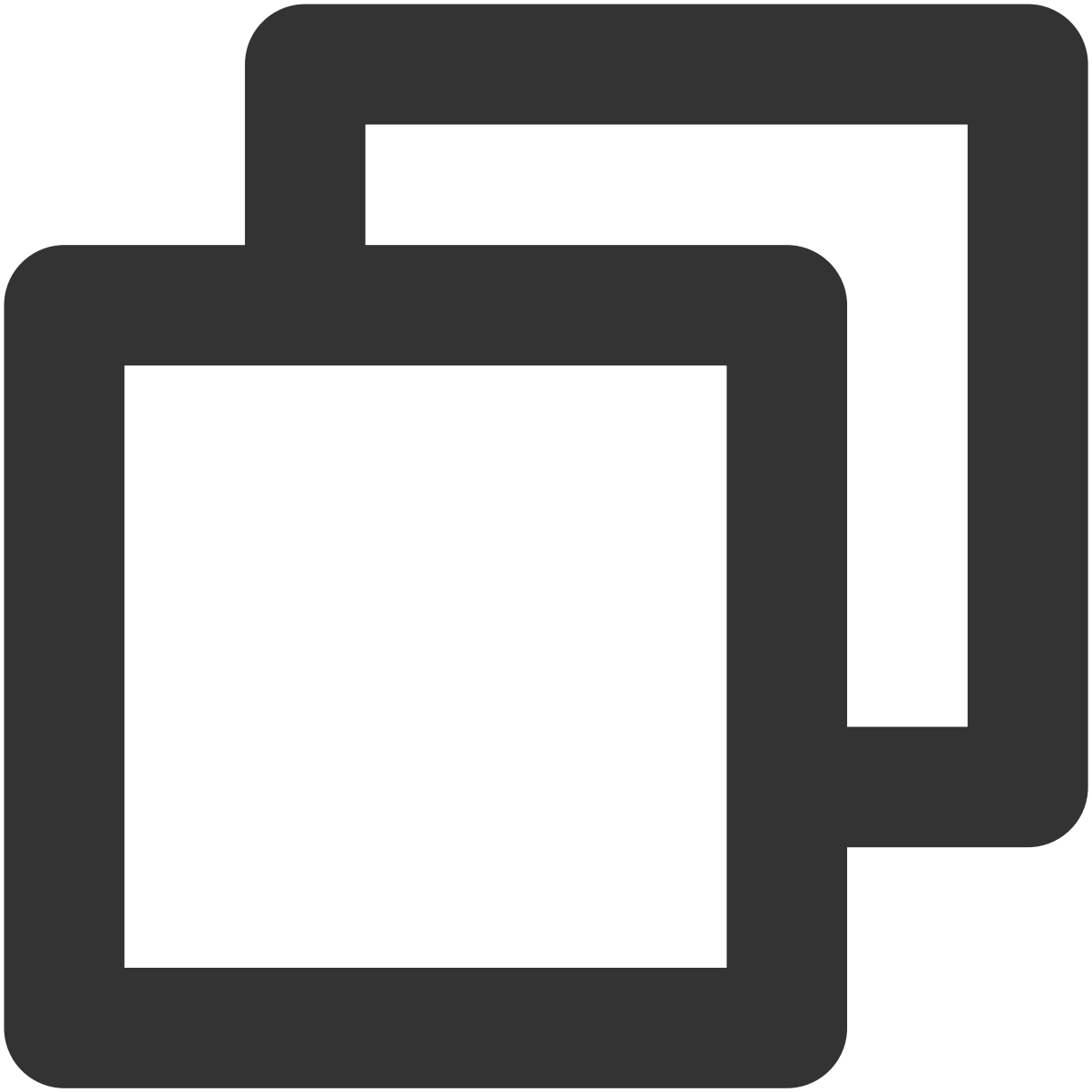
集群部署运行主要针对用户自建 IDC 集群的生产环境或尚未集成 GooseFS 的腾讯云 EMR 生产环境，这里有具体分为 Standalone 架构部署与高可用（HA）架构部署。
GooseFS 在 `scripts` 目录下提供了批量配置 SSH 免密登录以及批量安装部署 GooseFS 的脚本工具，可供用户方便快捷地完成 GooseFS 大规模集群的安装部署。用户可自检前一章节的部署环境要求中的批量部署条件，灵活地选择是否可以执行批量部署。

批量部署工具介绍

GooseFS 在 `scripts` 目录下提供了批量配置 SSH 免密登录以及批量部署安装 GooseFS 的工具化脚本，用户在满足执行条件的前提下，可按照如下步骤完成批量部署任务：
进入GooseFS解压目录， `cd ${GOOSEFS_HOME}` 。
配置 `conf/masters` 和 `conf/workers` 中的主机名或 IP 地址。
然后进入到 `scripts` 目录中，认真填写 `commons.sh` 这一脚本文件中的 `SCRIPT_EXEC_USER` 和 `SCRIPT_EXEC_PASSWORD` 。再切换到 `root` 账户或使用 `sudo` 身份执行 `config_ssh.sh` 完成整个集群的免密登录配置。
免密配置完成后，可以执行 `validate_env.sh` 工具，校验整个集群配置状态。
创建配置文件目录软连接， `ln -s conf ~/.goosefs` 。
最后执行 `goosefs copy .` 和 `goosefs copy ~/.goosefs` 将 goosefs 二进制包和配置文件分发到所有的节点。
在整个部署流程都成功完成以后，可以执行 `./bin/goosefs-start.sh all SudoMount` 来启动运行整个集群。默认配置下，所有的运行日志都会记录到 `${GOOSEFS_HOME}/logs` 下。

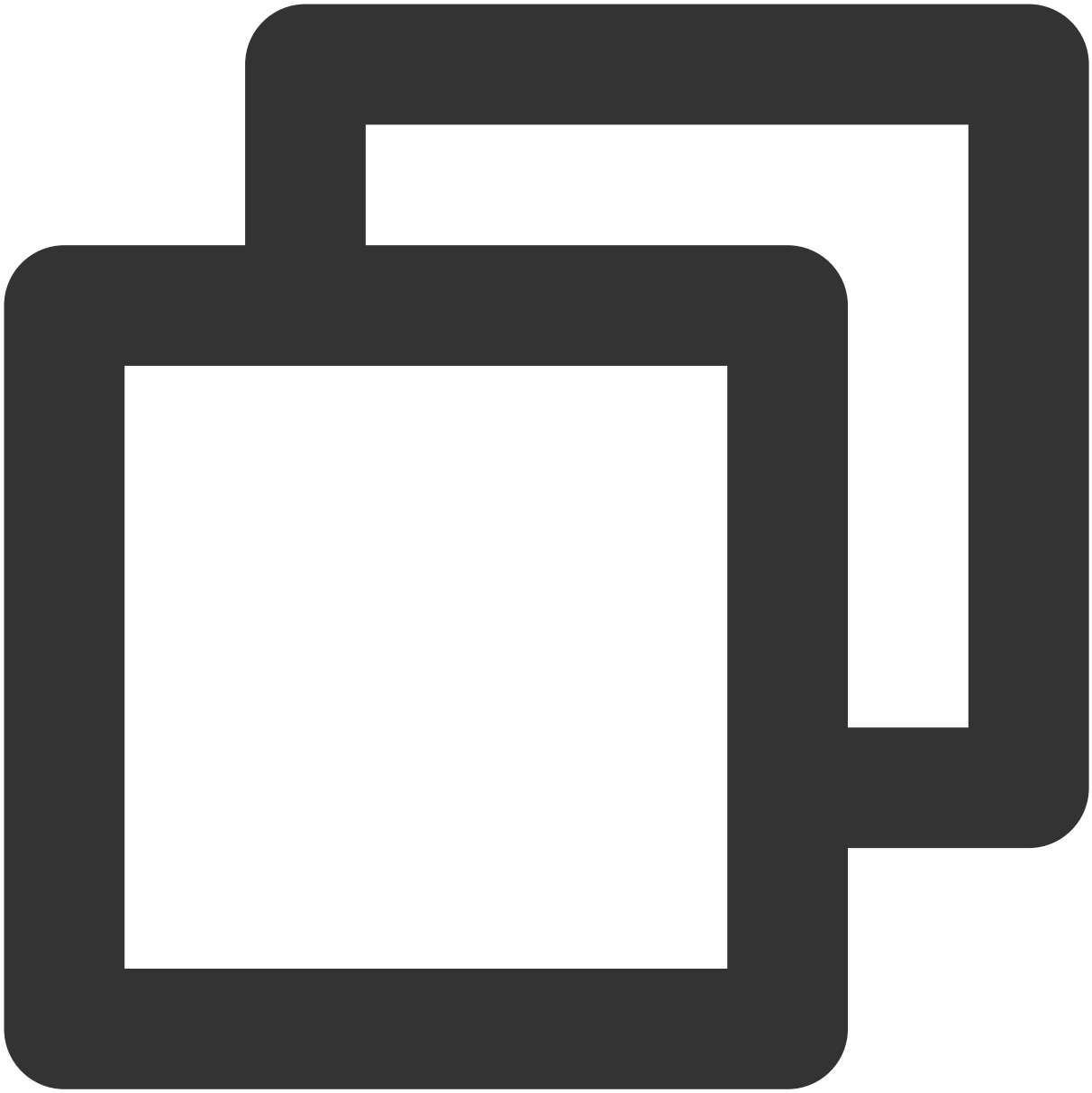
Standalone 架构部署

Standalone 架构采用的是单 Master 节点，多 Worker 节点的集群部署架构。具体可参考如下步骤部署运行：
1. 下载 GooseFS 的二进制分发包。
2. 使用 `tar zxvf goosefs-x.x.x-bin.tar.gz` 命令解压到安装路径后。可参见批量部署工具的介绍配置和执行集群的批量部署，也可以继续参考下文详细地手动部署流程。
（1）从 `conf` 目录下拷贝 `template` 文件创建配置文件：



```
$ cp conf/goosefs-site.properties.template conf/goosefs-site.properties
```

（2）在 `goosefs-site.properties` 配置文件中指定如下配置：



```
goosefs.master.hostname=<MASTER_HOSTNAME>
goosefs.master.mount.table.root.ufs=<STORAGE_URI>
```

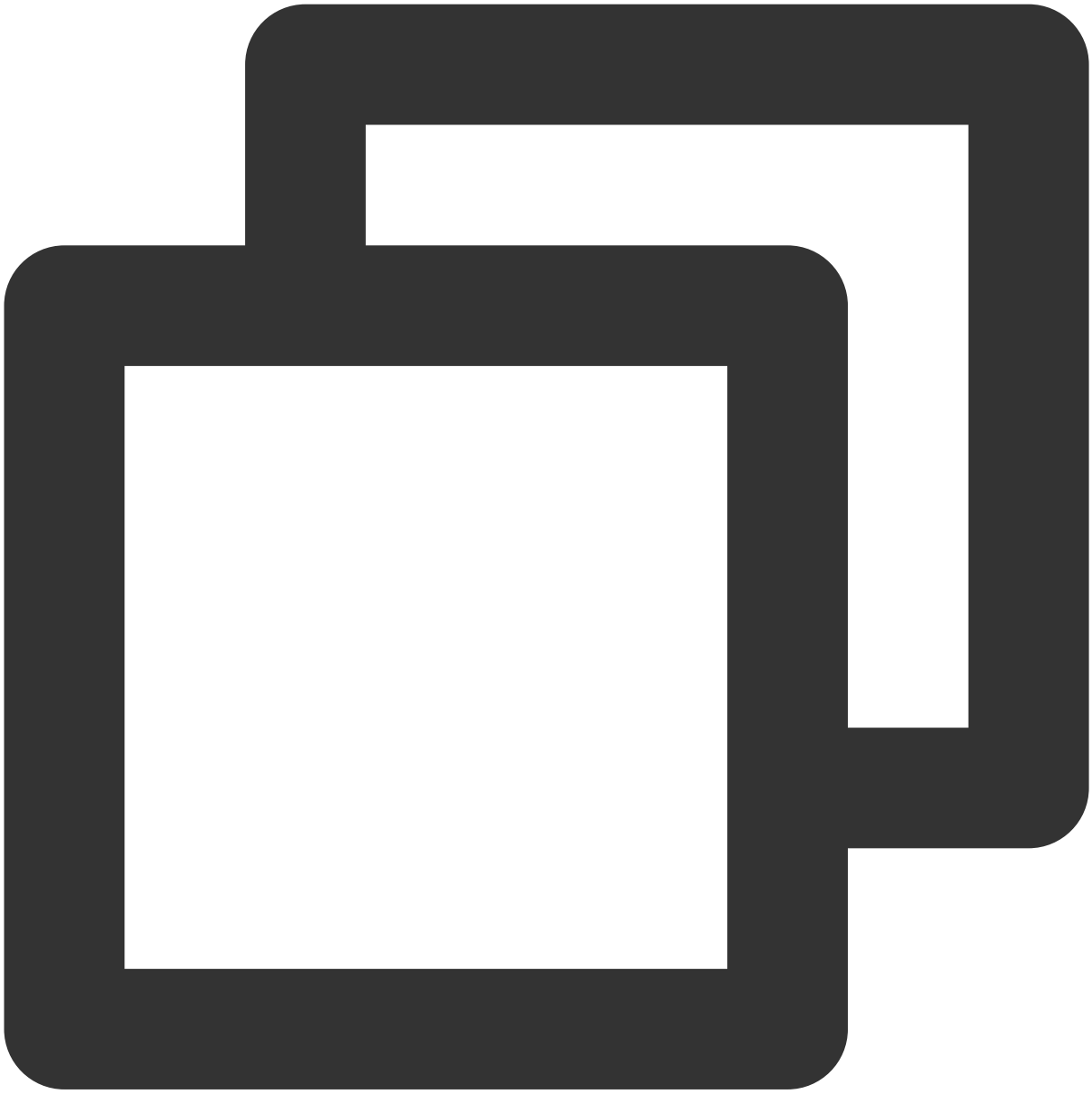
其中，`goosefs.master.hostname` 设置为单 master 节点的 hostname 或 ip。`goosefs.master.mount.table.root.ufs` 则指定 GooseFS 根目录所挂载的底层文件系统（UFS）路径 URI。

注意

这个 URI 必须保证 Master 和 Worker 节点都能访问到，因此不支持本地目录。

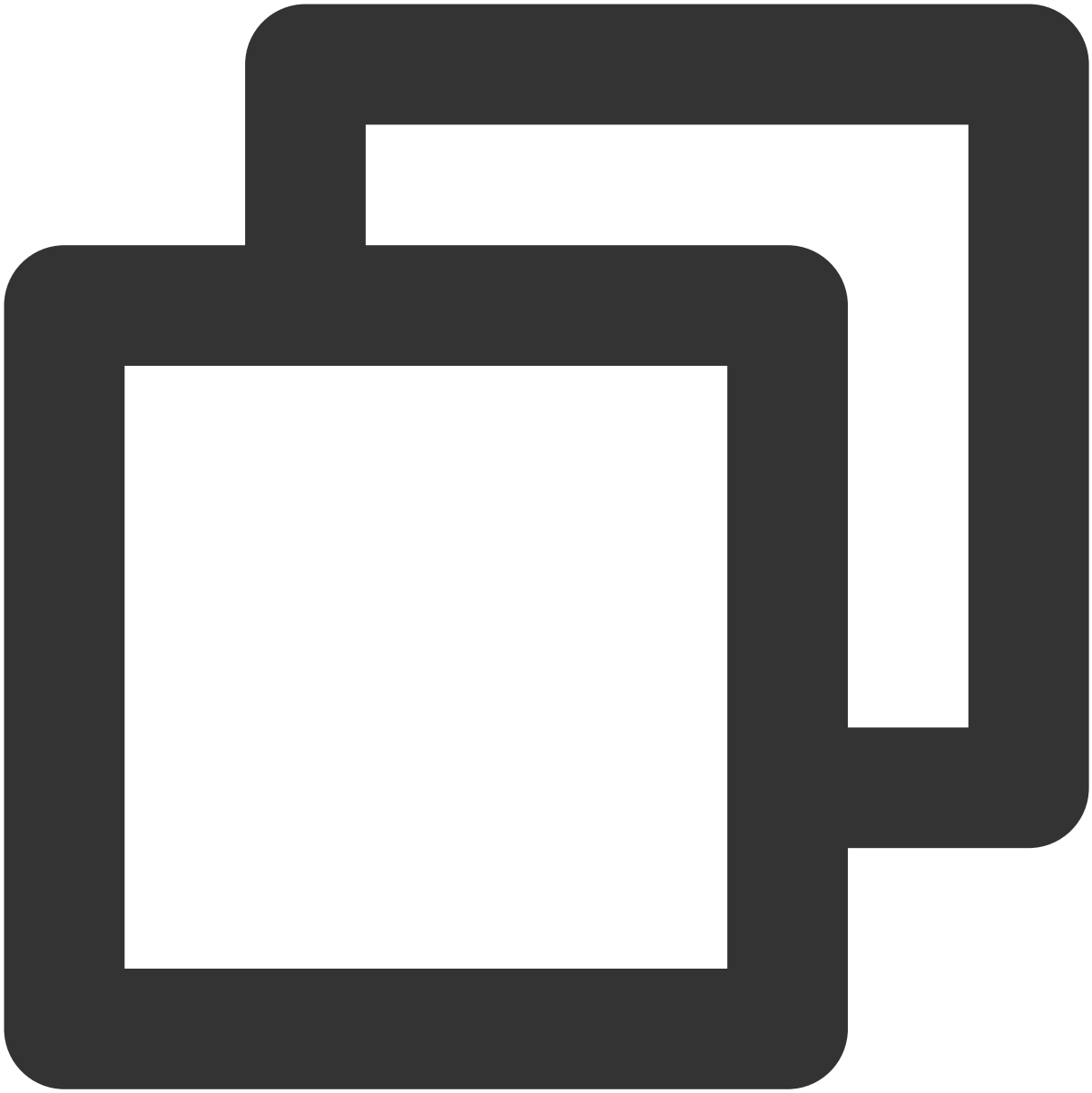
例如，您可以挂载一个 COS 路径为 GooseFS 根路径：`goosefs.master.mount.table.root.ufs=cosn://bucket-1250000000/goosefs/`。

在 `masters` 配置文件中指定单 Master 节点的 hostname 或 ip，例如：



```
# The multi-master Zookeeper HA mode requires that all the masters can access
# the same journal through a shared medium (e.g. HDFS or NFS).
# localhost
cvm1.compute-1.myqcloud.com
```

在 `workers` 配置文件中指定所有 Worker 节点的 host 或 ip，例如：



```
# An GooseFS Worker will be started on each of the machines listed below.
# localhost
cvm2.compute-2.myqcloud.com
cvm3.compute-3.myqcloud.com
```

待所有配置都设置完毕后，执行 `./bin/goosefs copyDir conf/` 就可以将配置同步到所有节点。

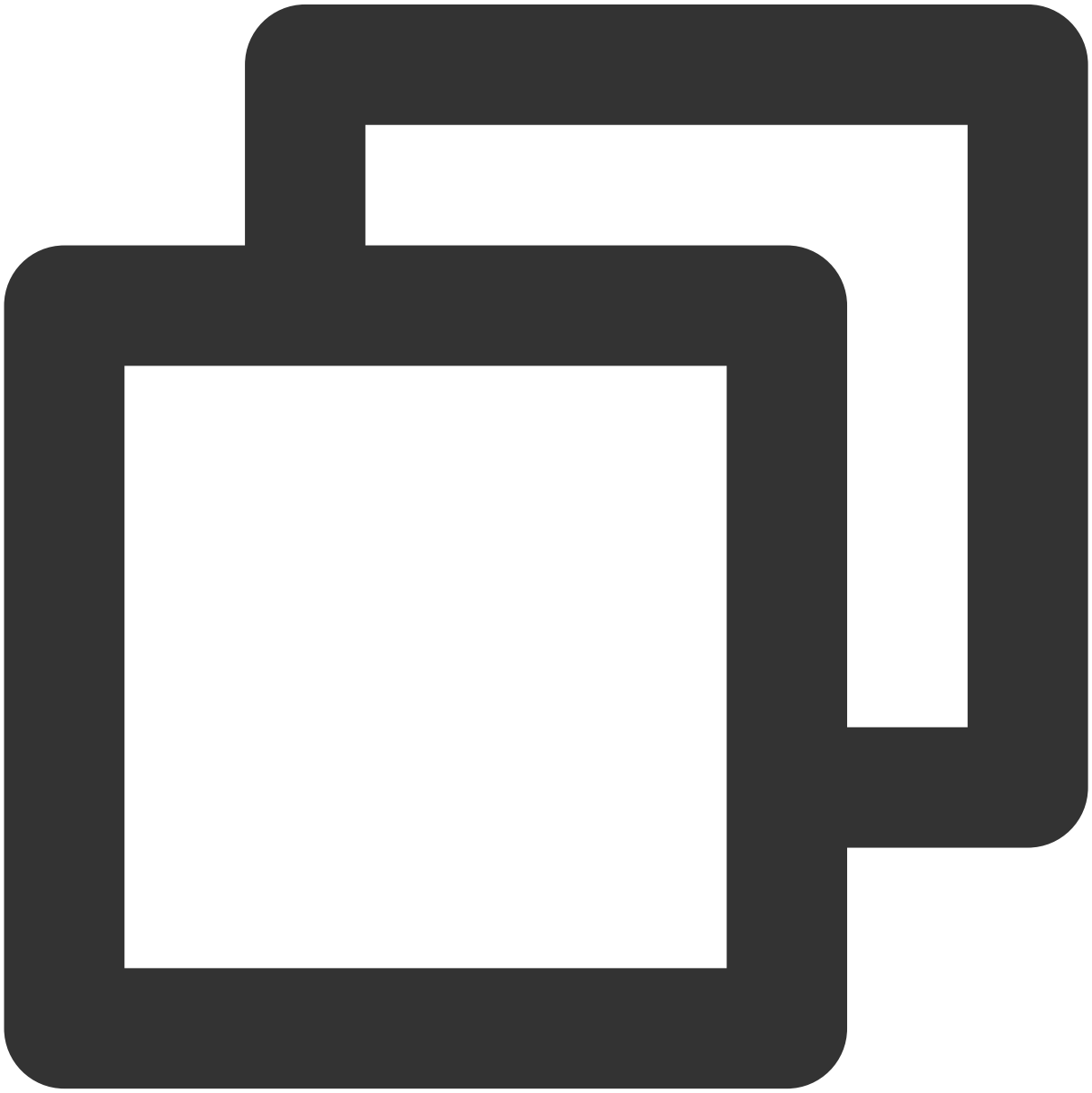
最后执行 `./bin/goosefs-start.sh all` 就可以启动 GooseFS 集群。

高可用架构部署

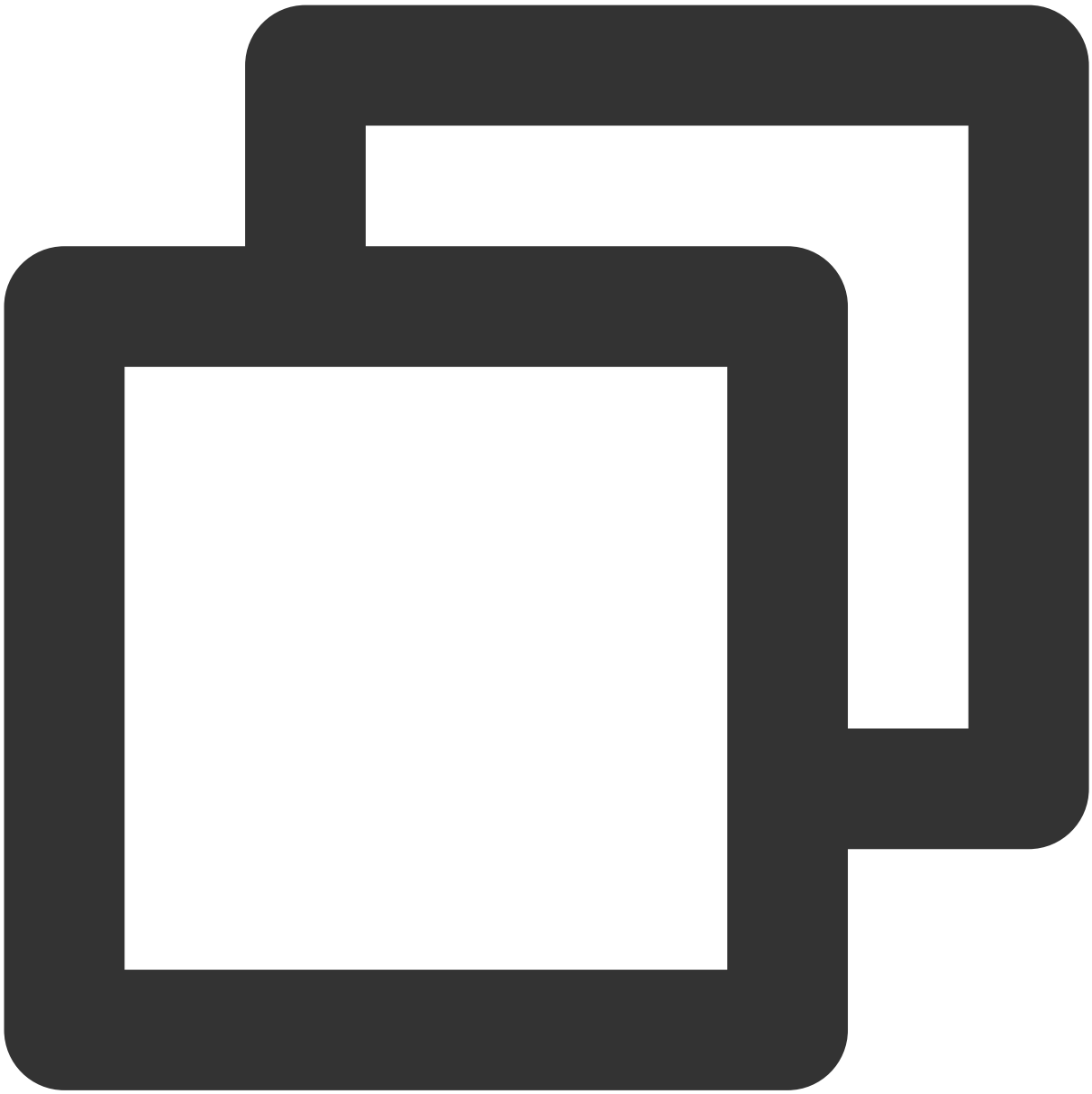
单 Master 节点的 Standalone 架构容易出现单点问题，因此，实际生产环境建议部署多 Master 节点来获得高可用的系统架构。多个 Master 节点中只有一个会作为主（Leader）节点对外提供服务，其他的备（Standby）节点通过同步共享日志（Journal）来保持与主节点相同的文件系统状态。当主节点故障宕机后，会从当前的备节点中自动选择一个接替主节点继续对外提供服务，这样就消除了系统的单点故障，实现了整体高可用架构。
目前 GooseFS 支持基于 Raft 日志和 Zookeeper 两种方式来实现主备节点状态的强一致性。下面将会针对这两种部署方式分别进行介绍。

基于 Raft 嵌入式日志的高可用部署配置

首先依据配置模板创建配置文件：



```
$ cp conf/goosefs-site.properties.template conf/goosefs-site.properties
```



```
goosefs.master.mount.table.root.ufs=<STORAGE_URI>
goosefs.master.embedded.journal.addresses=<EMBBEDDED_JOURNAL_ADDRESS>
```

说明

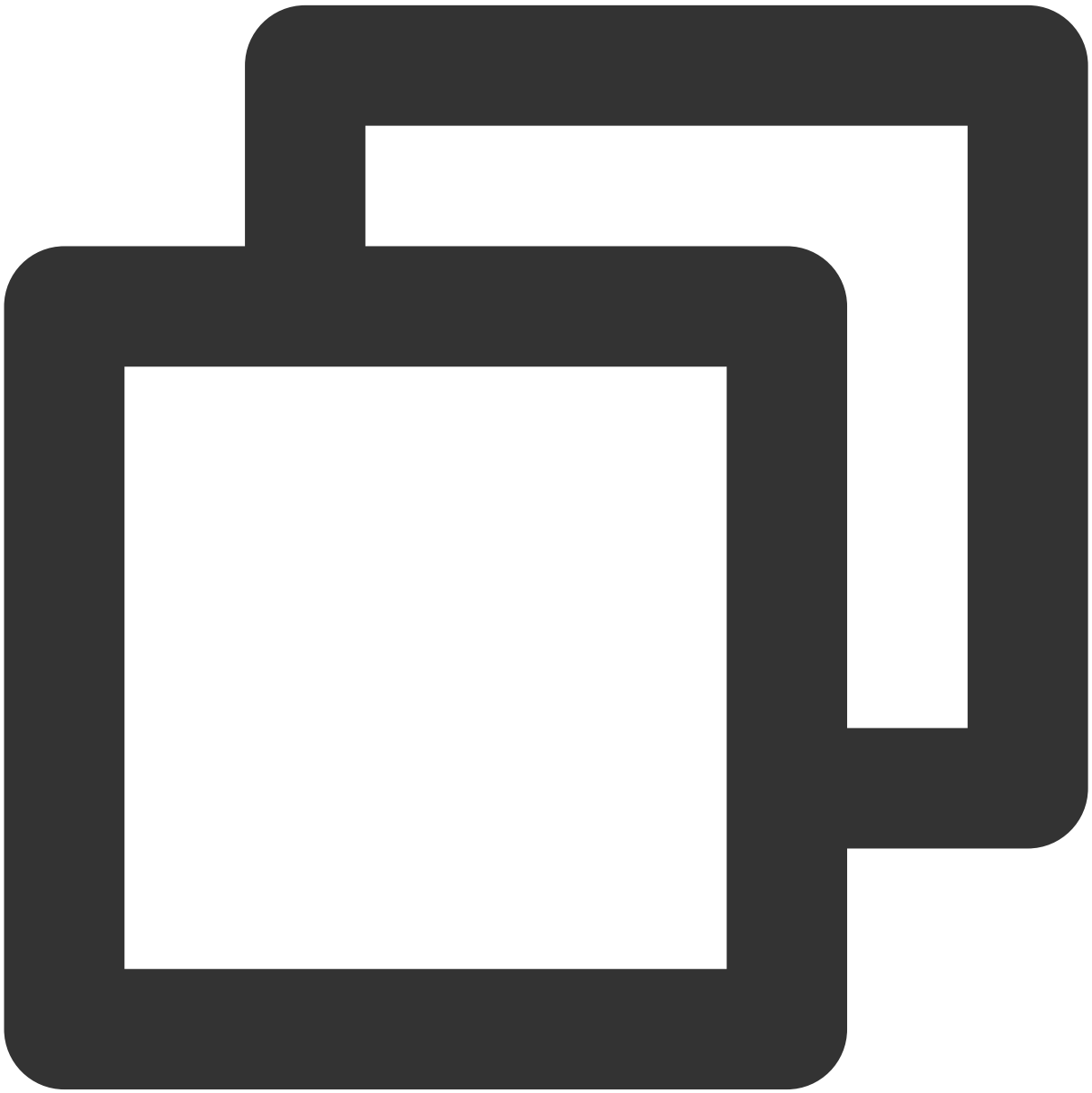
上述配置选项中，说明如下：

`goosefs.master.mount.table.root.ufs` 则设置为挂载到 GooseFS 根目录的底层存储 URI。

`goosefs.master.embedded.journal.addresses` 则设置为所有备节点的 `ip:embedded_journal_port` 或 `host:embedded_journal_port` 。其中，embedded_journal_port 默认为9202。例如：192.168.1.1:9202,192.168.1.2:9202,192.168.1.3:9202。

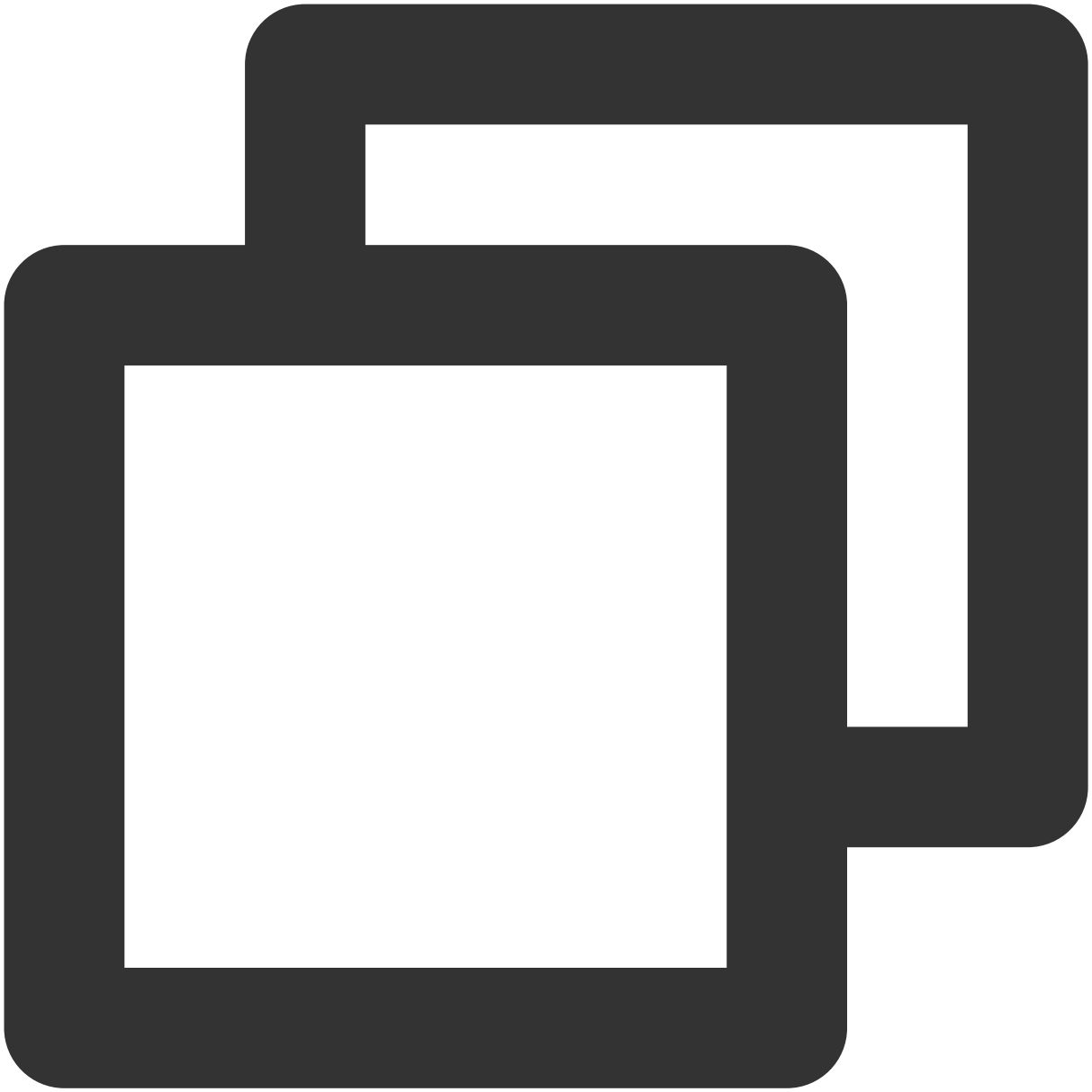
基于 Raft 嵌入式日志的部署方案依赖于 copycat 的 Leader 选举机制。因此，Raft 的 HA 部署架构不可与 Zookeeper 相互混用。

在完成所有配置以后，同样使用如下命令同步所有配置：

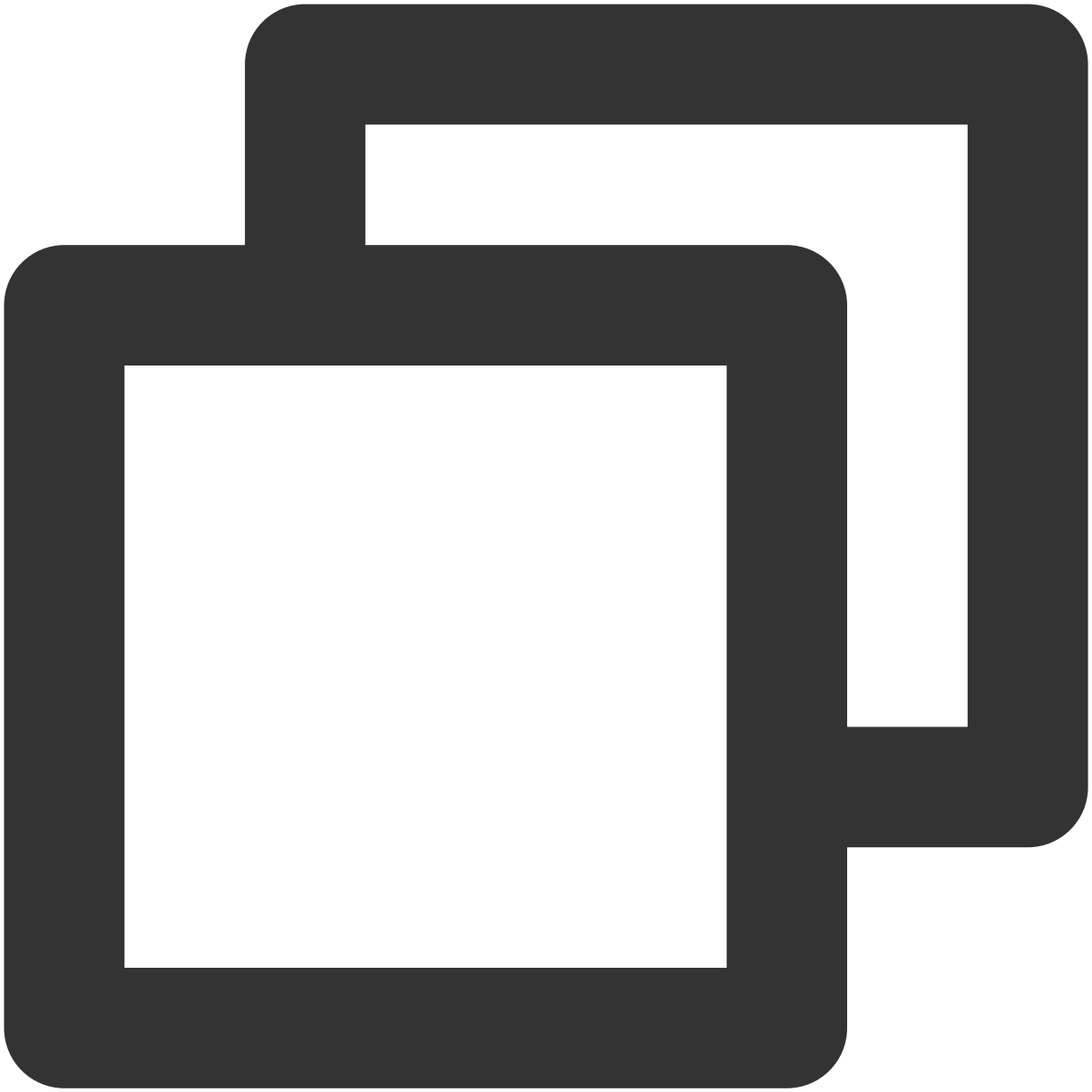


```
$ ./bin/goosefs copyDir conf/
```

最后，完成格式化以后，即可启动 GooseFS 集群：

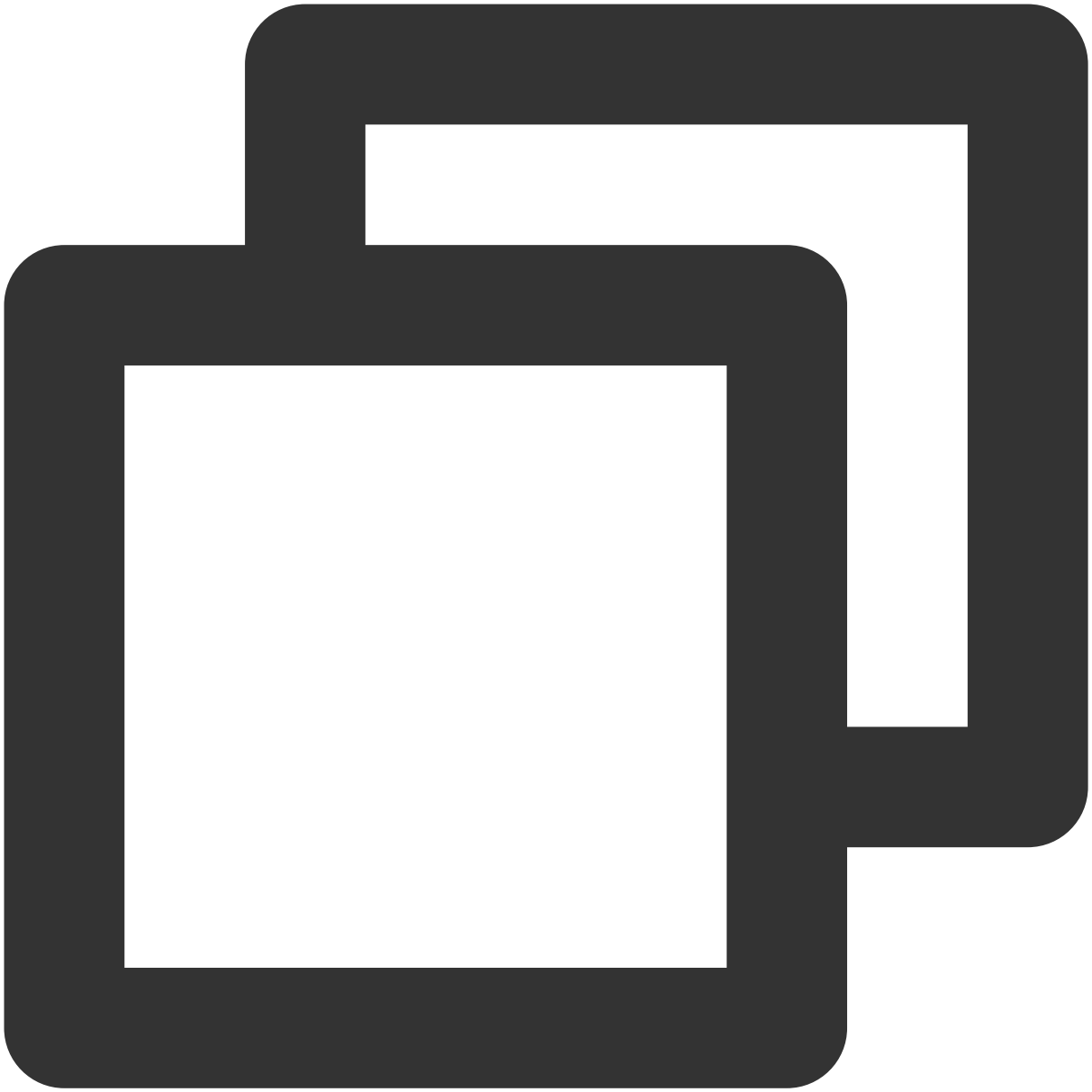


```
$ ./bin/goosefs format
```



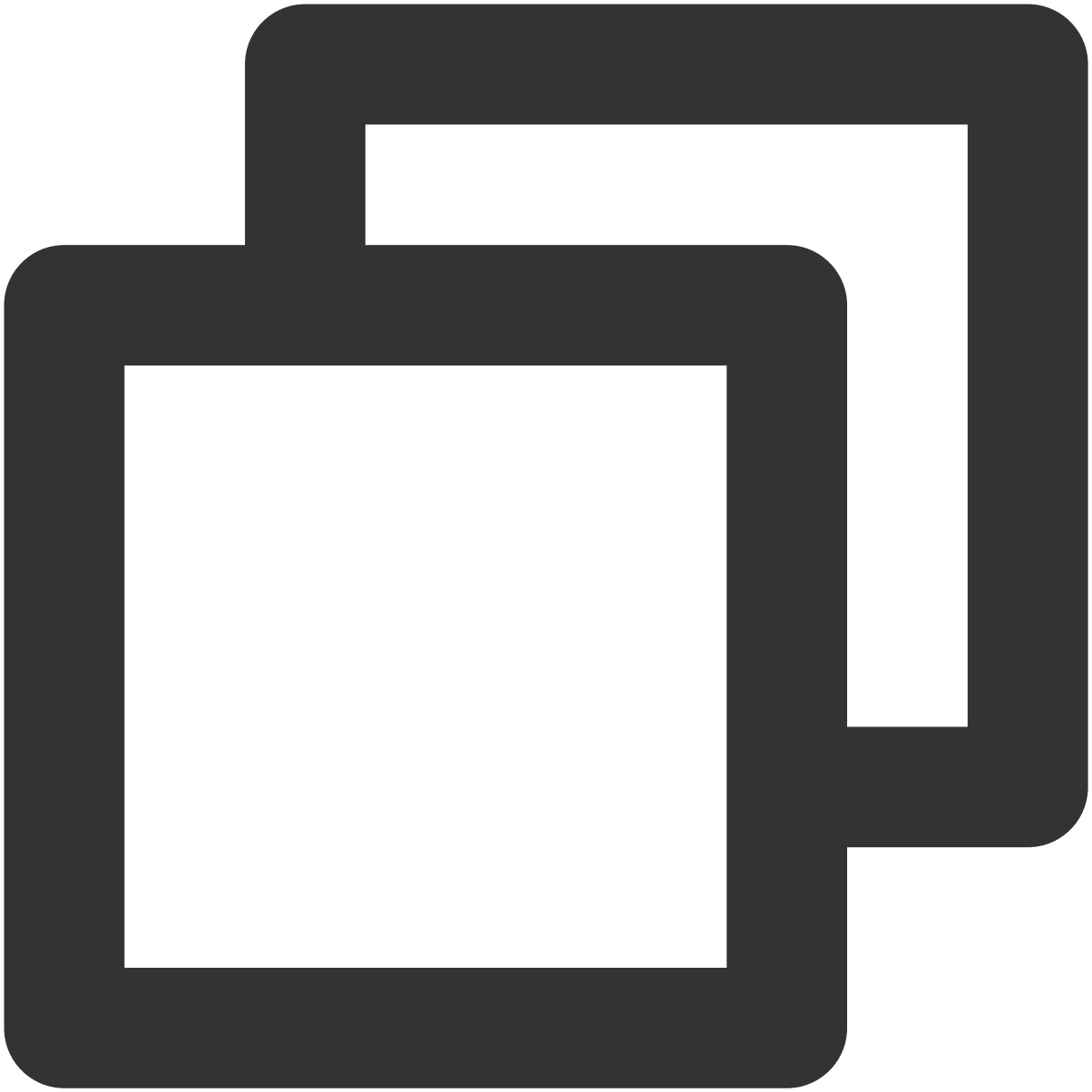
```
$ ./bin/goosefs-start.sh all
```

如果提示权限不足，则尝试如下方式重启：



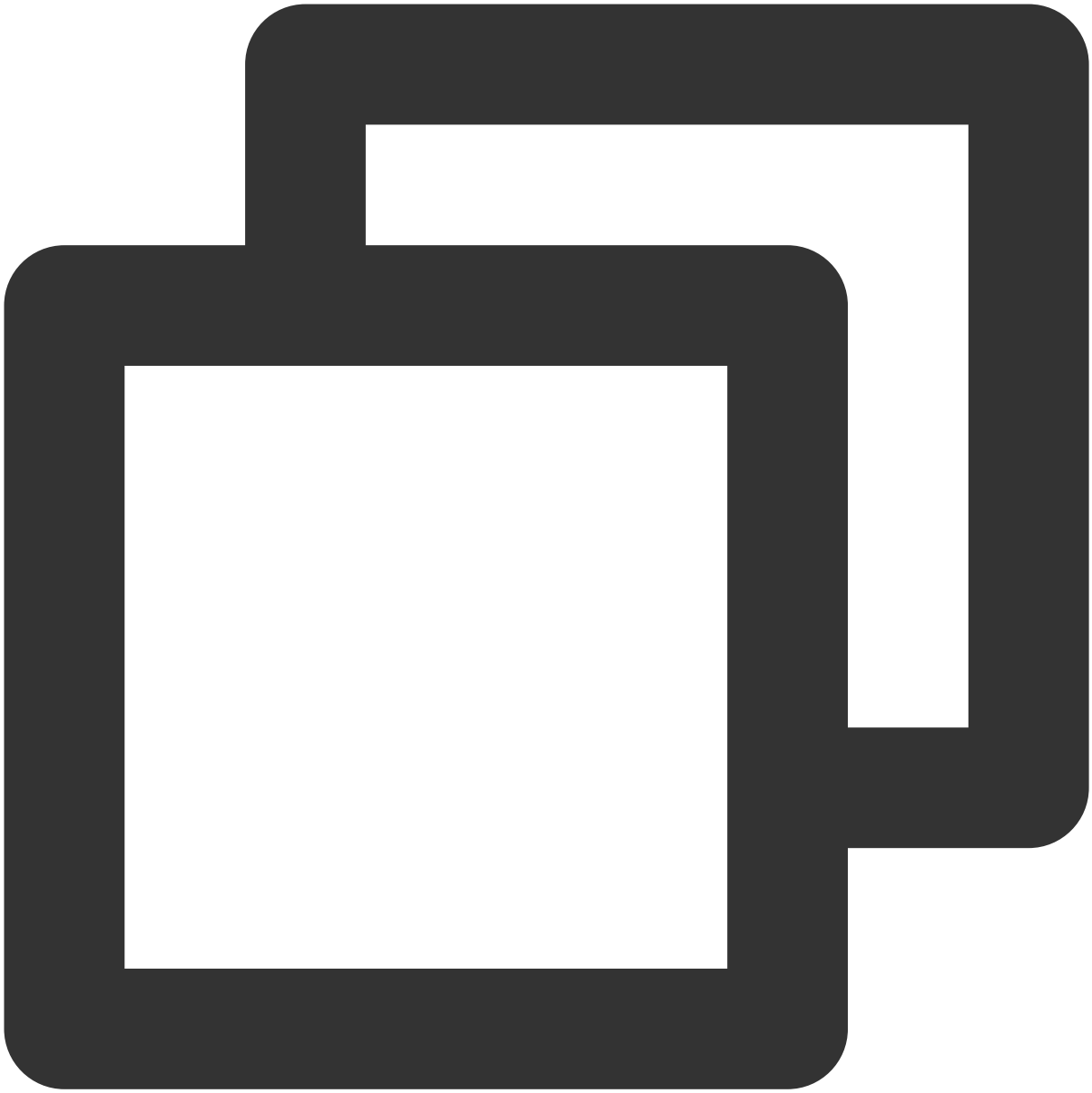
```
$ ./bin/goosefs-stop.sh all
$ ./bin/goosefs-start.sh all SudoMount
```

使用如下命令可查看当前的主（Leader）节点：



```
$ ./bin/goosefs fs leader
```

也可以使用如下命令查看集群状态：



```
$ ./bin/goosefs fsadmin report
```

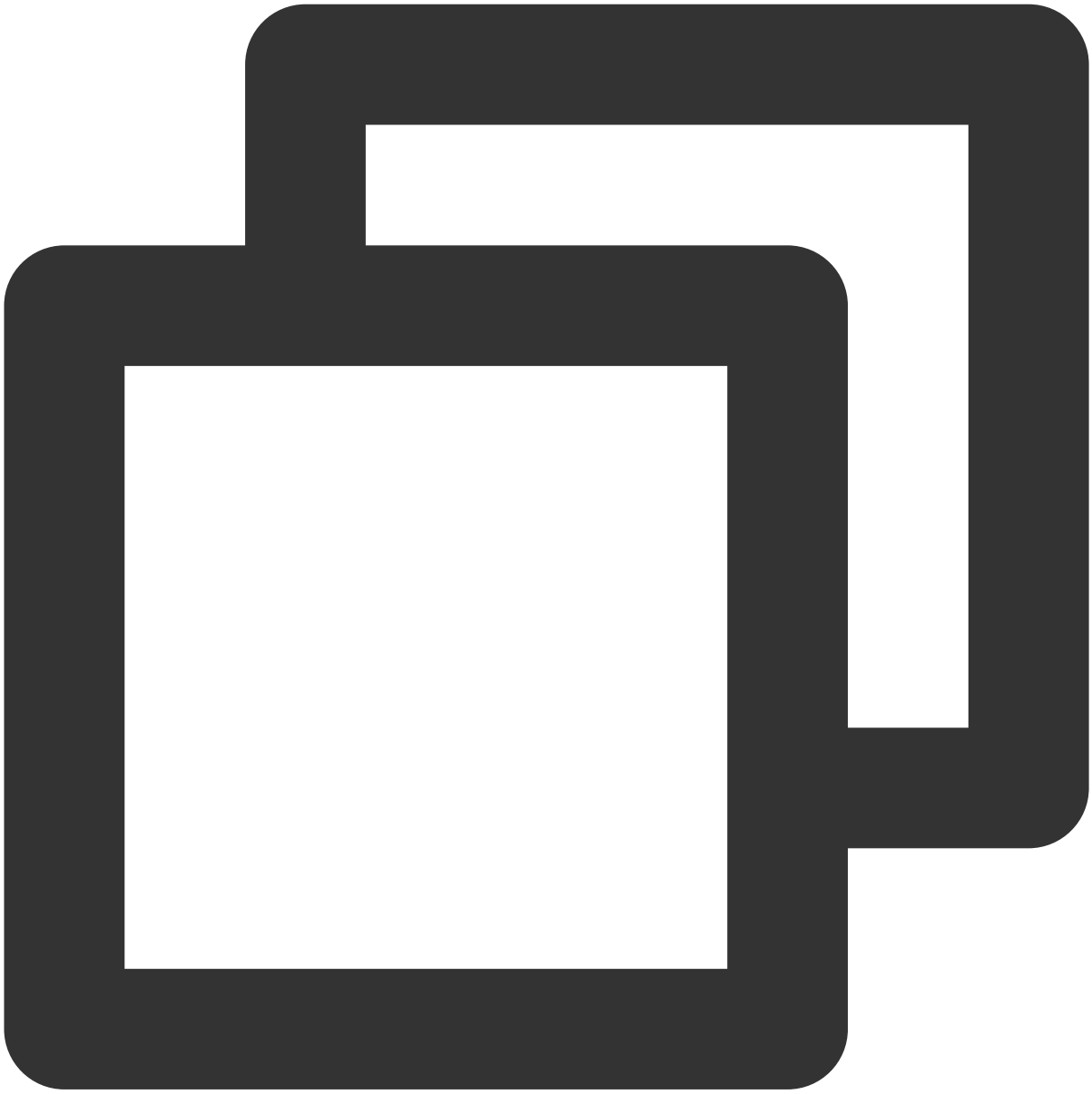
基于 Zookeeper 的共享日志部署配置

配置 Zookeeper 服务搭建 GooseFS 的高可用架构需要满足如下条件：

Zookeeper 集群，GooseFS Master 使用 Zookeeper 进行 Leader 选取，GooseFS 的客户端和 Worker 节点则通过 Zookeeper 查询主 Master 节点；

高可用强一致的共享存储系统，并且保证所有 GooseFS 的 Master 节点均可以访问到。主 Master 节点会将日志写入到该存储系统上，备（Standby）节点则会不断地从共享存储系统上读取日志，并且重放来保持与主节点的状态一致

性。一般情况，该共享存储系统，推荐设置为 HDFS 或 COS。例如，`hdfs://10.0.0.1:9000/goosefs/journal` 或 `cosn://bucket-1250000000/journal`。配置如下：



```
goosefs.zookeeper.enabled=true
goosefs.zookeeper.address=<ZOOKEEPER_ADDRESSS>
goosefs.master.journal.type=UFS
goosefs.master.journal.folder=<JOURNAL_URI>
```

需要注意的是，ZK 模式的 `JOURNAL_URI` 必须是共享存储系统的 URI。然后使用 `./bin/goosefs copyDir conf/` 将配置同步分发到集群中的所有节点。最后启动集群 `./bin/goosefs-start.sh all` 即可。

GooseFS 的进程列表

当执行 `goosefs-start.sh all` 脚本并启动 GooseFS 后，集群将包含如下进程：

进程	描述
GooseFSMaster	默认 RPC 端口为9200，Web 端口为9201
GooseFSWorker	默认 RPC 端口为9203，Web 端口为9204
GooseFSJobMaster	默认 RPC 端口为9205，Web 端口为9206
GooseFSProxy	默认 Web 端口为9211
GooseFSJobWorker	默认 RPC 端口为9208，Web 端口为9210

版权所有：腾讯云计算（北京）有限责任公司

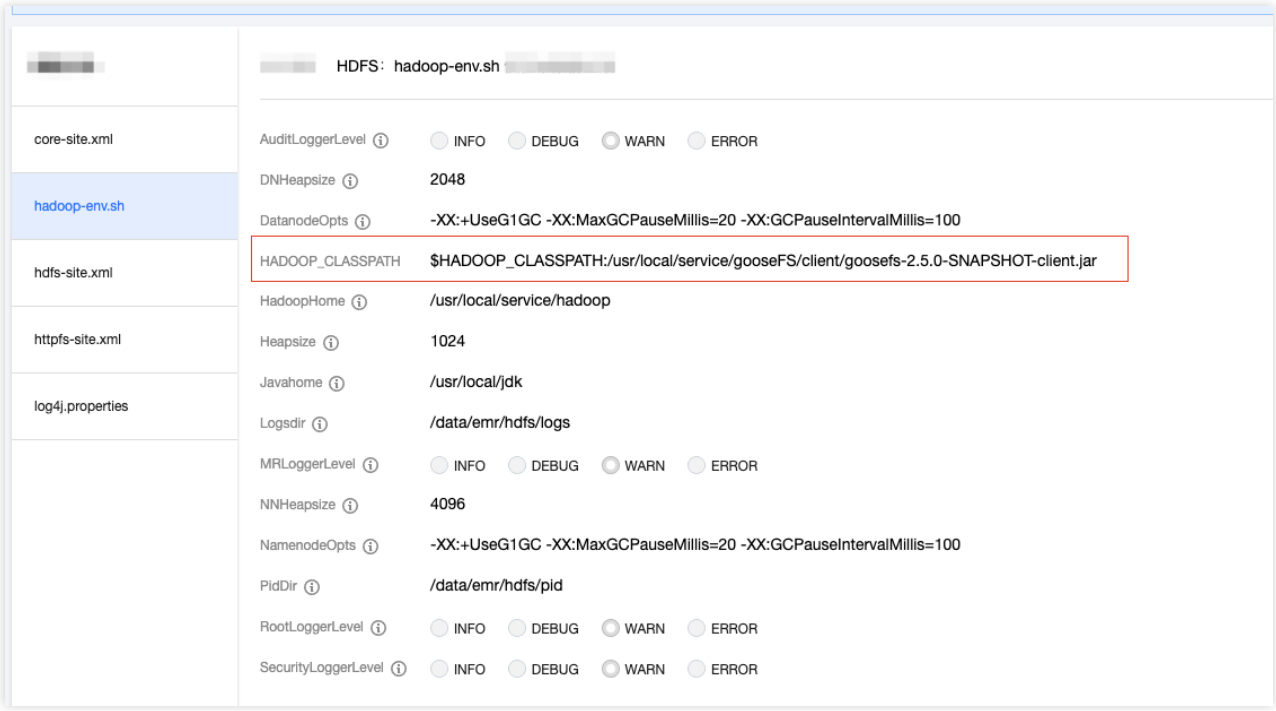
使用腾讯云 EMR 部署

最近更新时间：2024-01-06 11:17:59

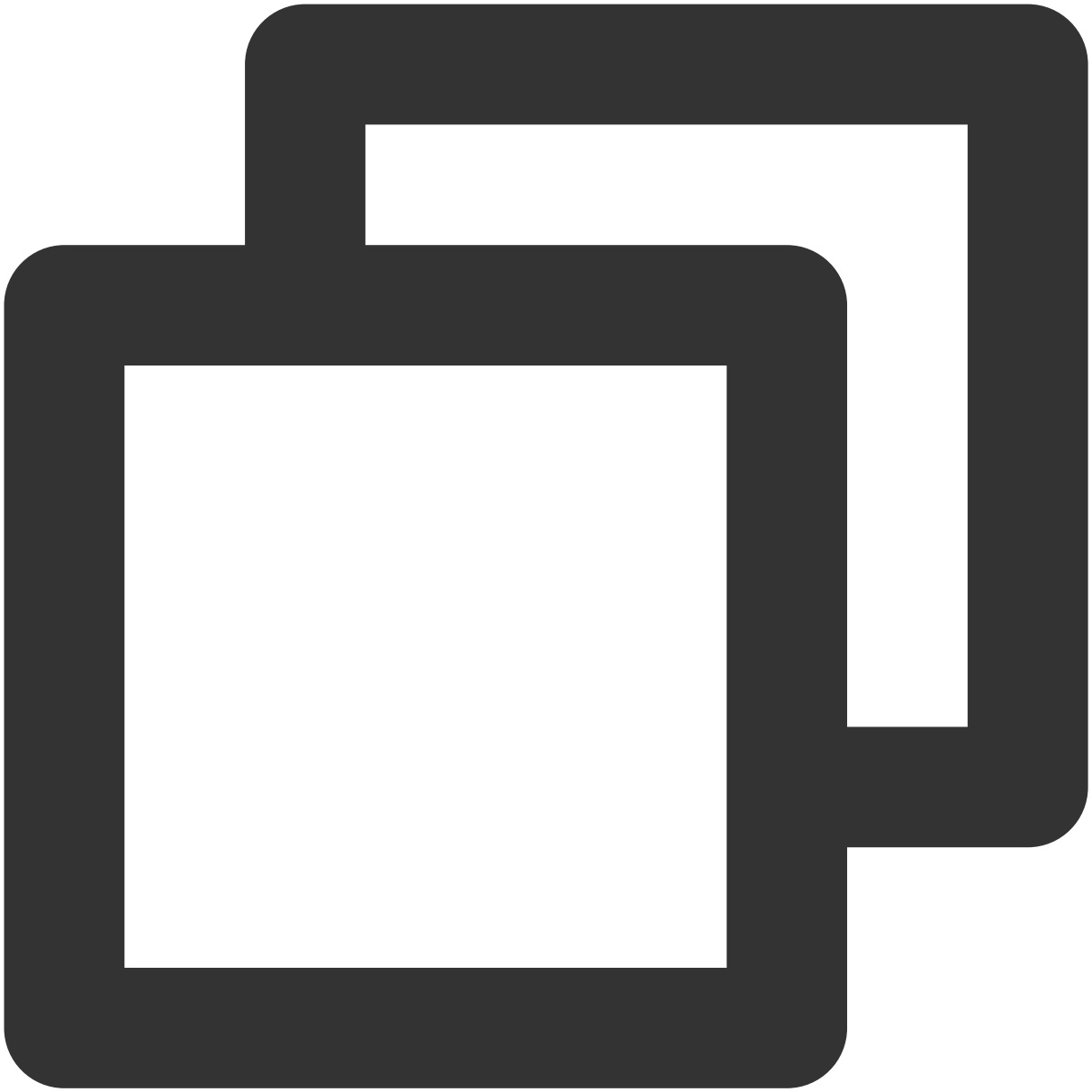
目前 GooseFS 已经集成到了腾讯云 EMR 环境中，将会在最新的 EMR 版本中发布。届时，用户无需针对腾讯云 EMR 环境单独部署，可以像使用其他 EMR 组件一样直接使用 GooseFS。
下文将针对未集成 GooseFS 的腾讯云 EMR 存量集群，介绍如何部署配置 GooseFS 的 EMR 环境。
首先，参照 集群模式部署运行 章节的内容，选择生产环境合适的部署架构，完成集群部署。
其次，针对 EMR 支持组件进行配置，本文以 Hadoop MapReduce、Spark 以及 Flink 对 GooseFS 的支持来讲解。

Hadoop MapReduce 支持

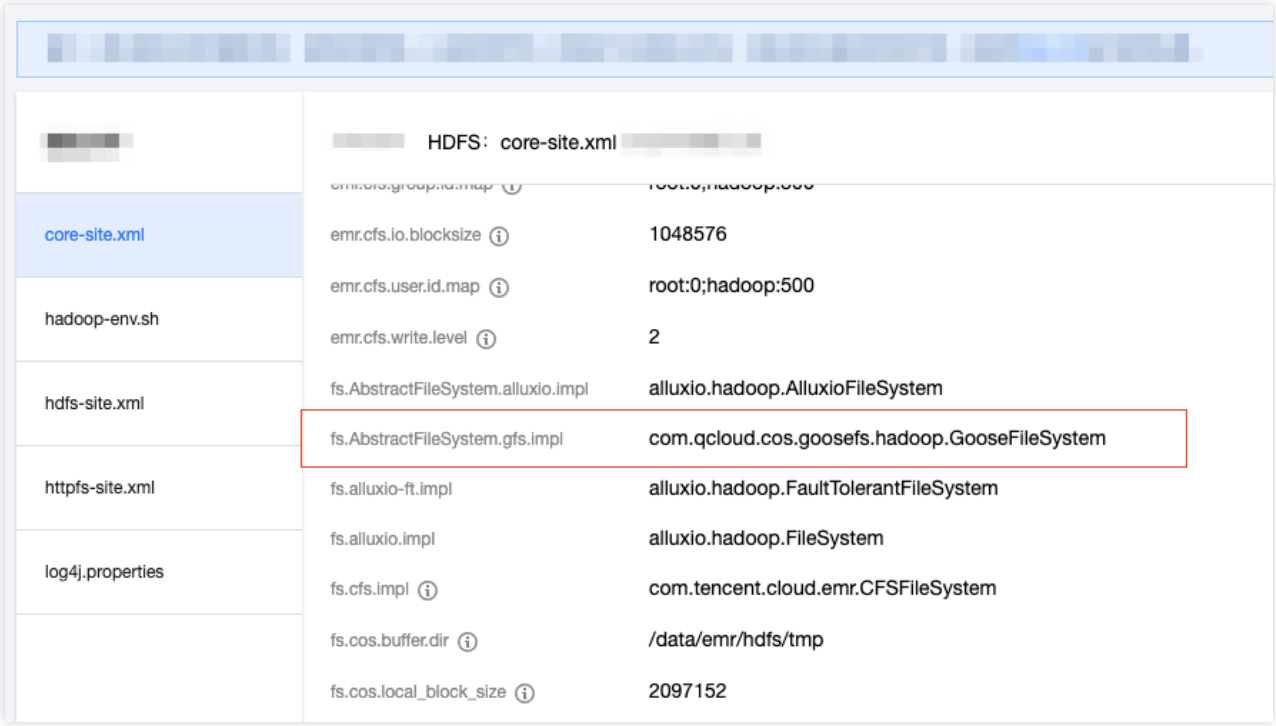
为了使得 Hadoop 的 MapReduce 作业能够读写 GooseFS 中的数据，需要在 hadoop-env.sh 中将 GooseFS Client 的依赖路径添加到 HADOOP_CLASSPATH，这个操作可以在 EMR 的控制台上完成，如下所示：



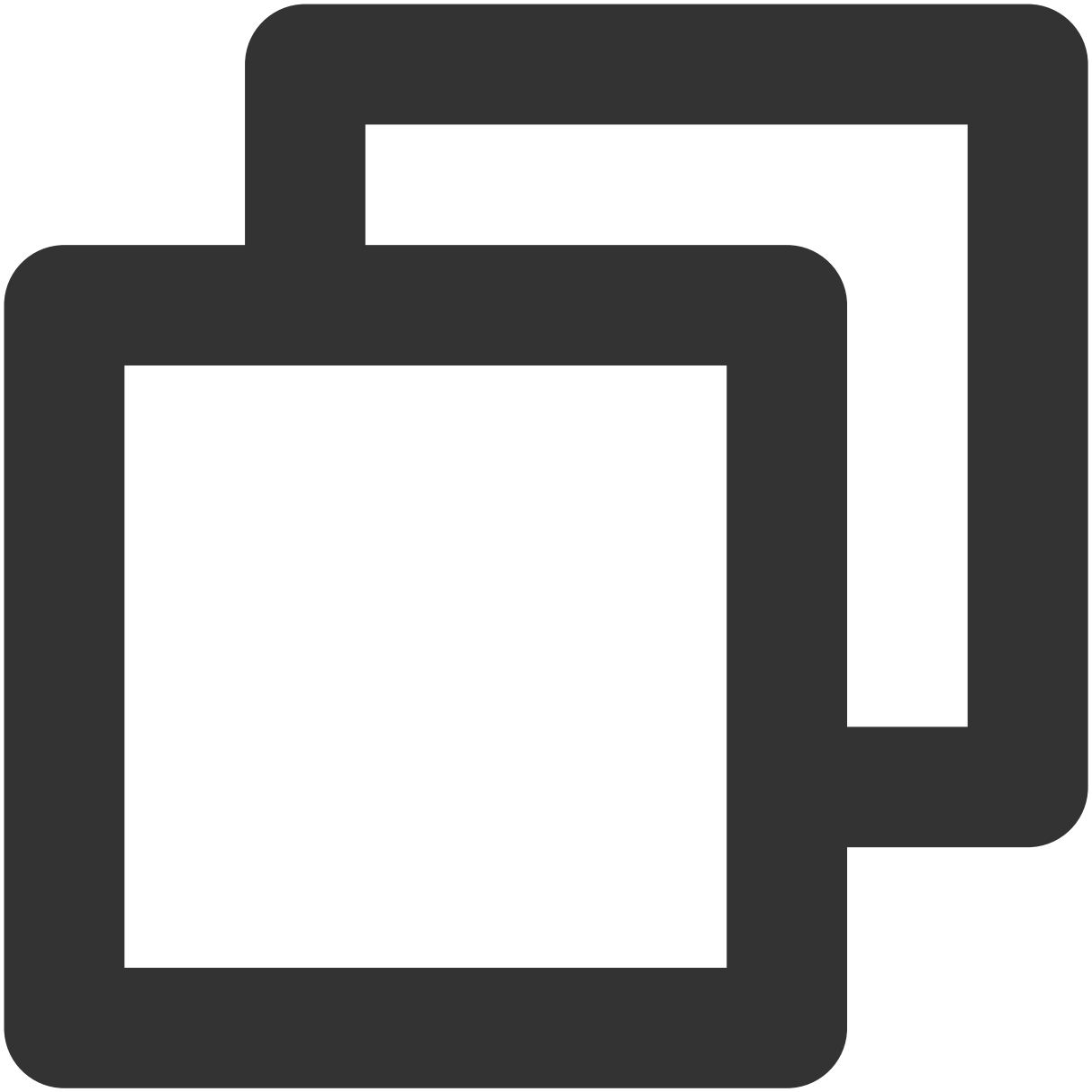
同时，还需要配置在 core-site.xml 中配置 GooseFS 的 HCFS 实现，同样这个操作也可以在 EMR 的控制台上完成：配置 fs.AbstractFileSystem.gfs.impl 为如下：



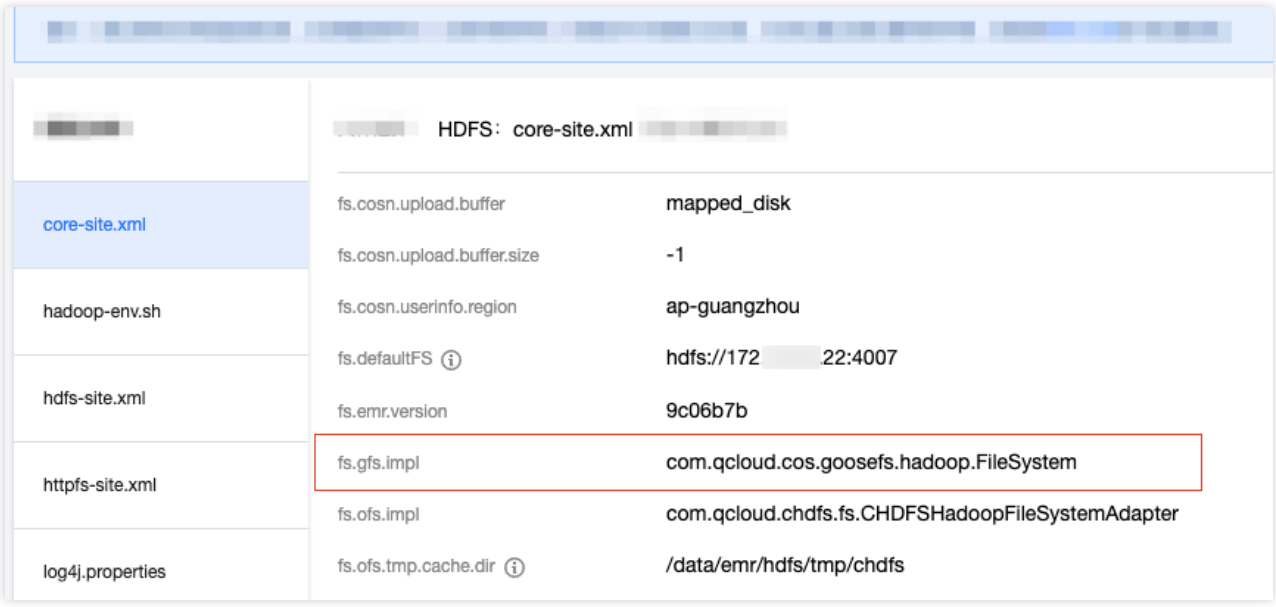
```
com.qcloud.cos.goosefs.hadoop.GooseFileSystem
```



配置 fs.gfs.impl 为如下：



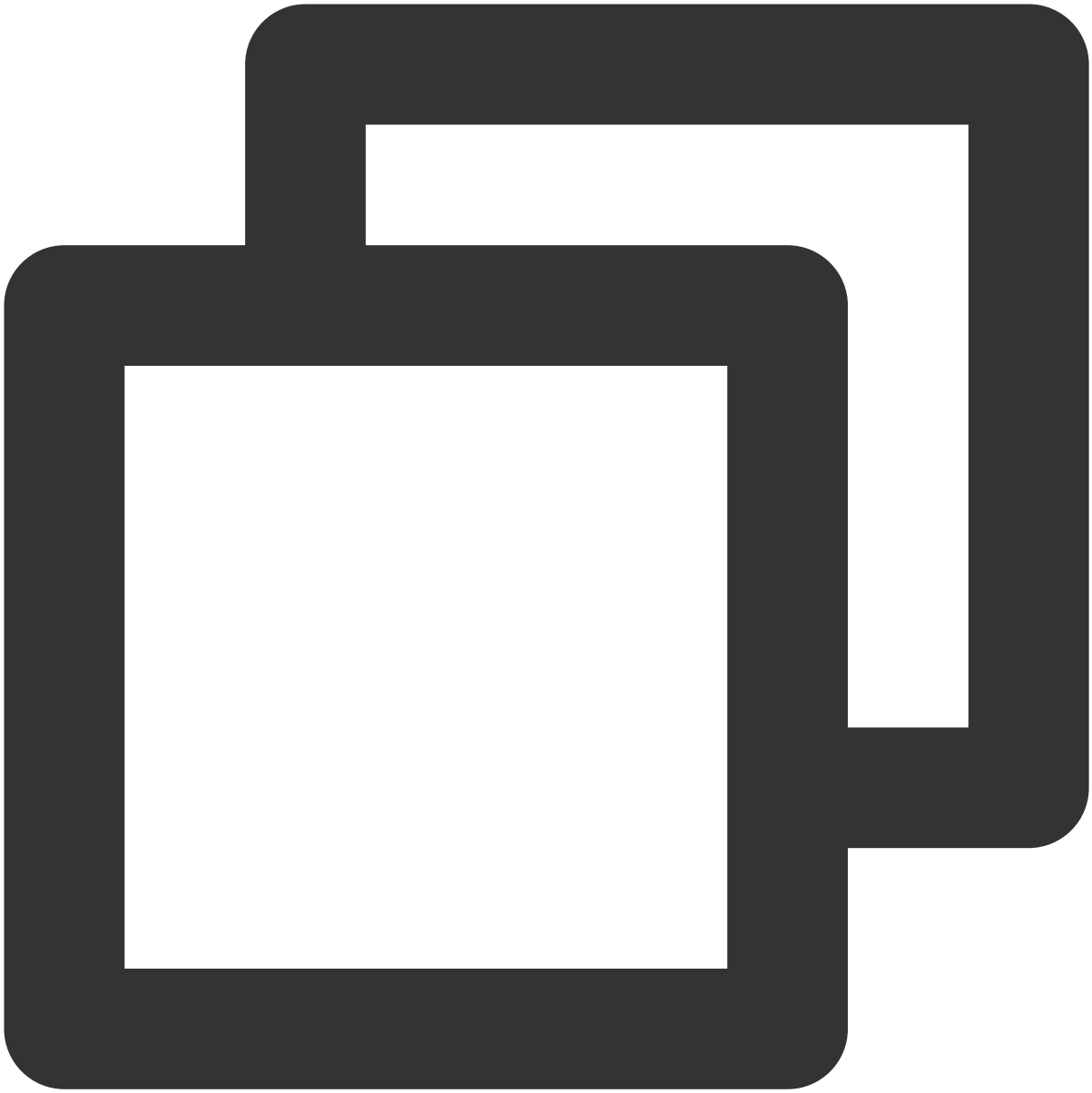
```
com.qcloud.cos.goosefs.hadoop.FileSystem
```



下发配置后，重启 YARN 相关组件即可生效。

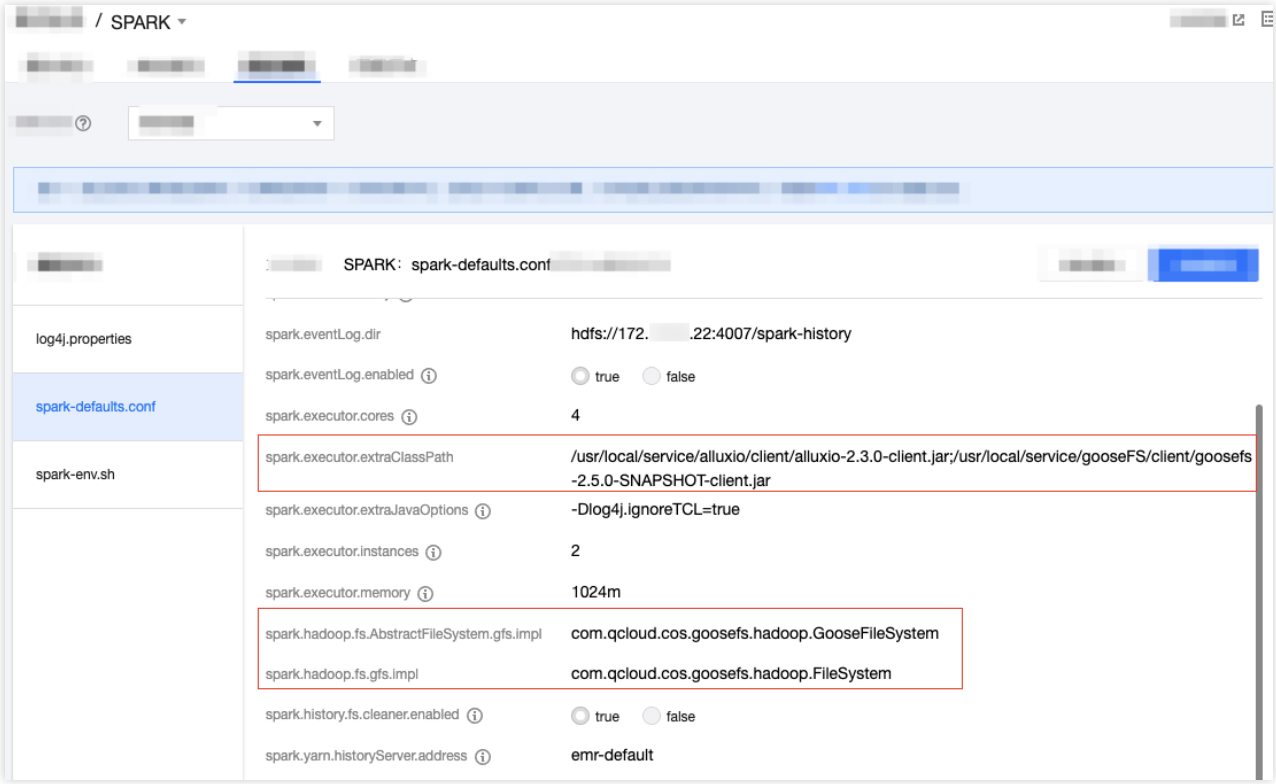
Spark 支持

为了使得 Spark 能够访问goosefs，同样需要配置 GooseFS 的 client 依赖包到 spark 的 executor classpath 中，同时在 spark-defaults.conf 中指定：



```
...
spark.driver.extraClassPath ${GOOSEFS_HOME}/client/goosefs-x.x.x-client.jar
spark.executor.extraClassPath ${GOOSEFS_HOME}/client/goosefs-x.x.x-client.jar
spark.hadoop.fs.gfs.impl com.qcloud.cos.goosefs.hadoop.FileSystem
spark.hadoop.fs.AbstractFileSystem.gfs.impl com.qcloud.cos.goosefs.hadoop.GooseFile
...
```

同样，该操作也可以在 EMR 控制台上 Spark 组件中配置和下发：



Flink 支持

腾讯云 EMR 的 Flink 采用的是 Flink on YARN 的部署模式，因此原则上只要确保 ${FLINK_HOME}/flink-conf.yaml 中正确设置 `fs.hdfs.hadoopconf` 到 hadoop 的配置路径下即可，腾讯云 EMR 集群中一般为 `/usr/local/service/hadoop/etc/hadoop` 。

无需设置其他配置项，直接使用 Flink on YARN 的方式提交 Flink 作业即可，作业中需要访问 GooseFS 的路径为 `gfs://master:port/<path>` 。

注意

Flink 访问 GooseFS 时，必须指定 master 和 port。

Hive、Impala、HBase、Sqoop 以及 Oozie 支持

当配置 Hadoop MapReduce 的环境支持以后，Hive、Impala、HBase 等组件无需单独配置支持，即可正常使用。

使用腾讯云 TKE 部署

最近更新时间：2024-01-11 14:22:28

使用 TKE 部署 GooseFS 需要借助 开源组件 Fluid 部署，TKE 应用市场已上架。部署包括两步：
1. 通过 Fluid helm chart 部署 controller。
2. 通过 kubectl 创建 Dataset 和 GooseFS runtime。

准备事项

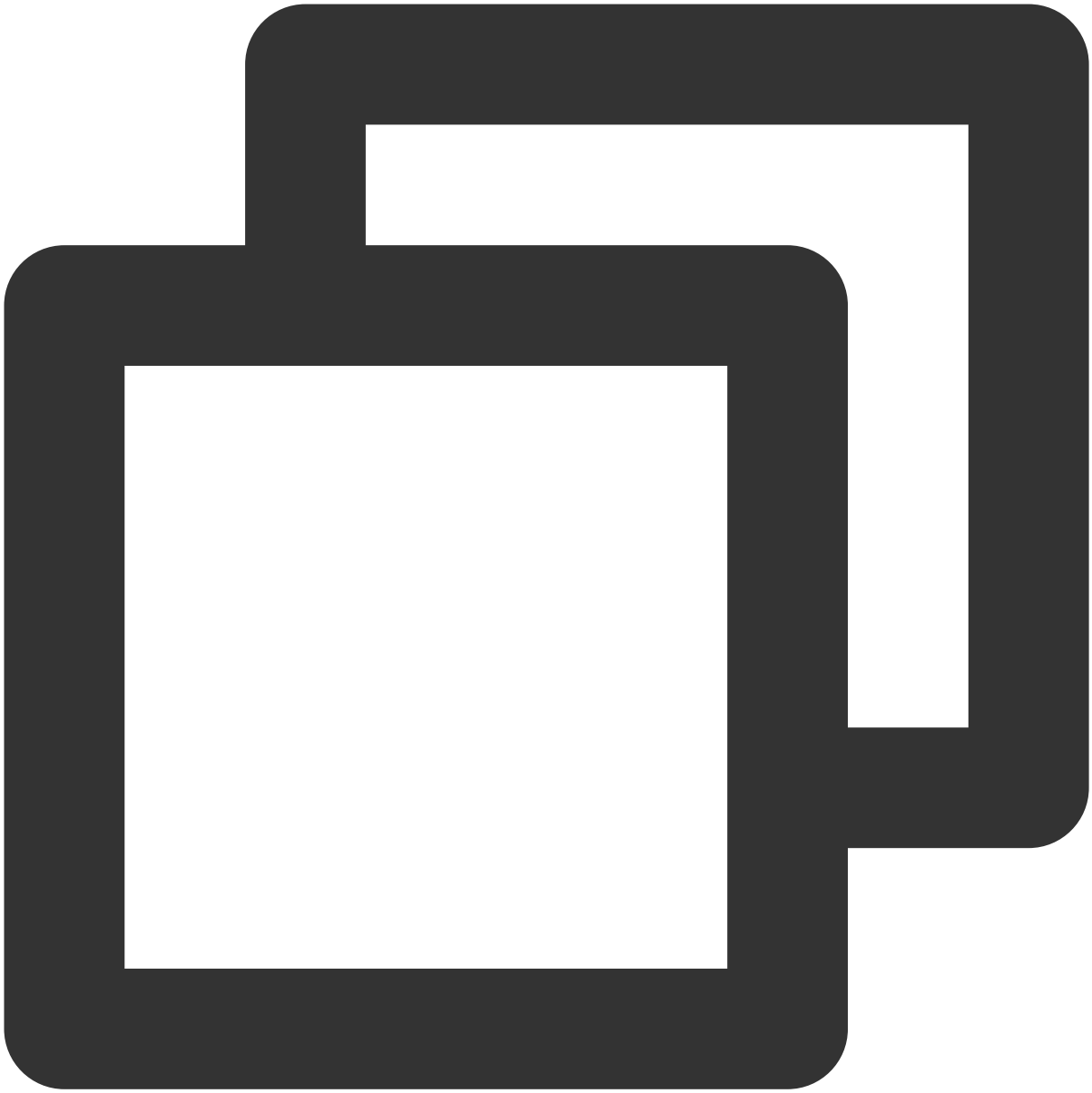
1. 拥有腾讯云 TKE 集群。
2. 已安装 kubectl，版本 v1.18及以上。

安装步骤

1. 在 TKE 应用市场 找到 fluid 应用。
2. 安装 Fluid Controller。
3. 检查 controller 组件。在左侧【集群】中找到对应集群，如果看到了两个 controller，则说明 fluid 组件安装成功。

操作演示

1. 集群访问授权



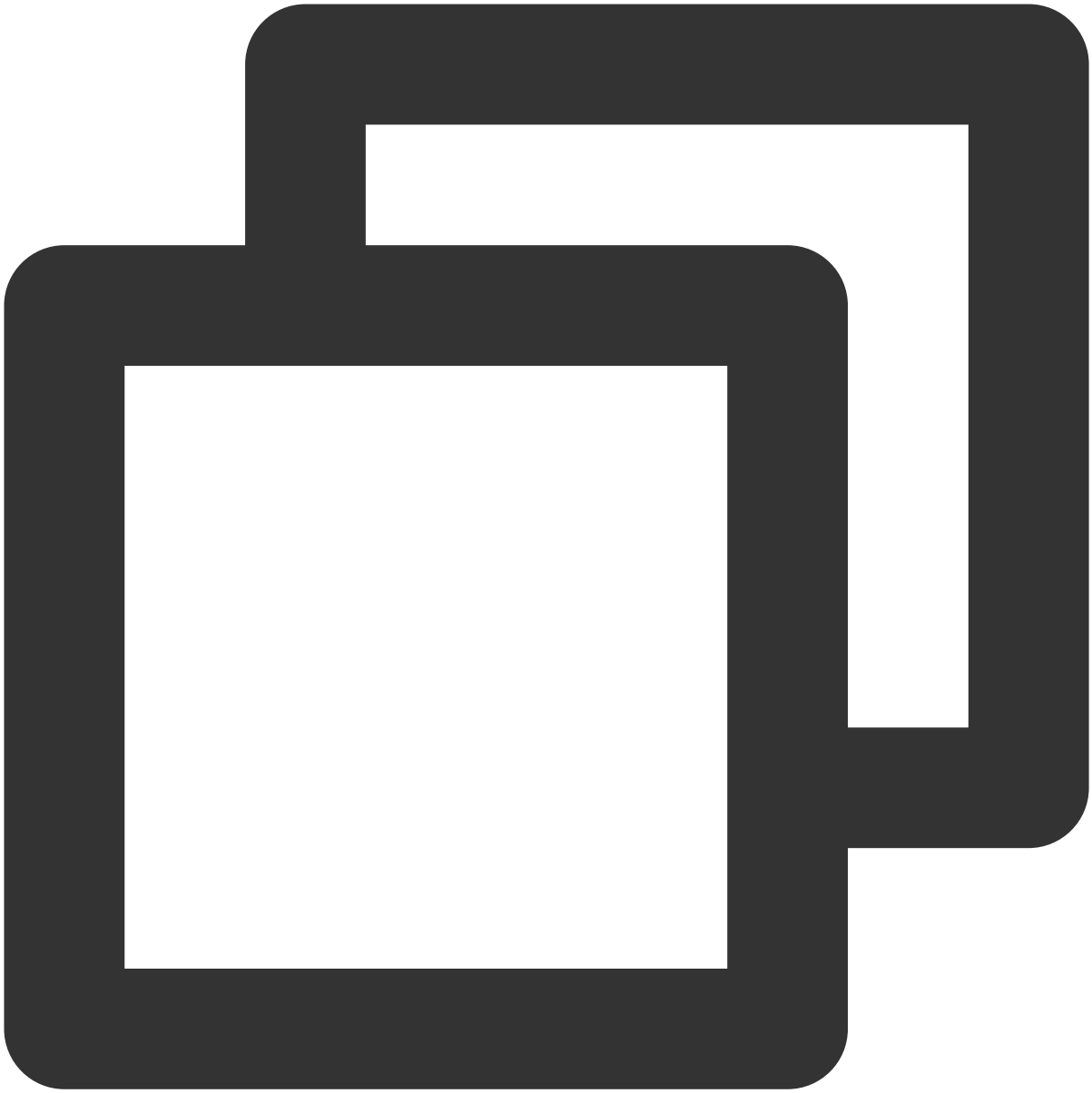
```
[root@master01 run]# export KUBECONFIG=xxx/cls-xxx-config （从tke控制台页面，下载集群凭证
```

注意

集群 API Server 需要开启外网访问权限。

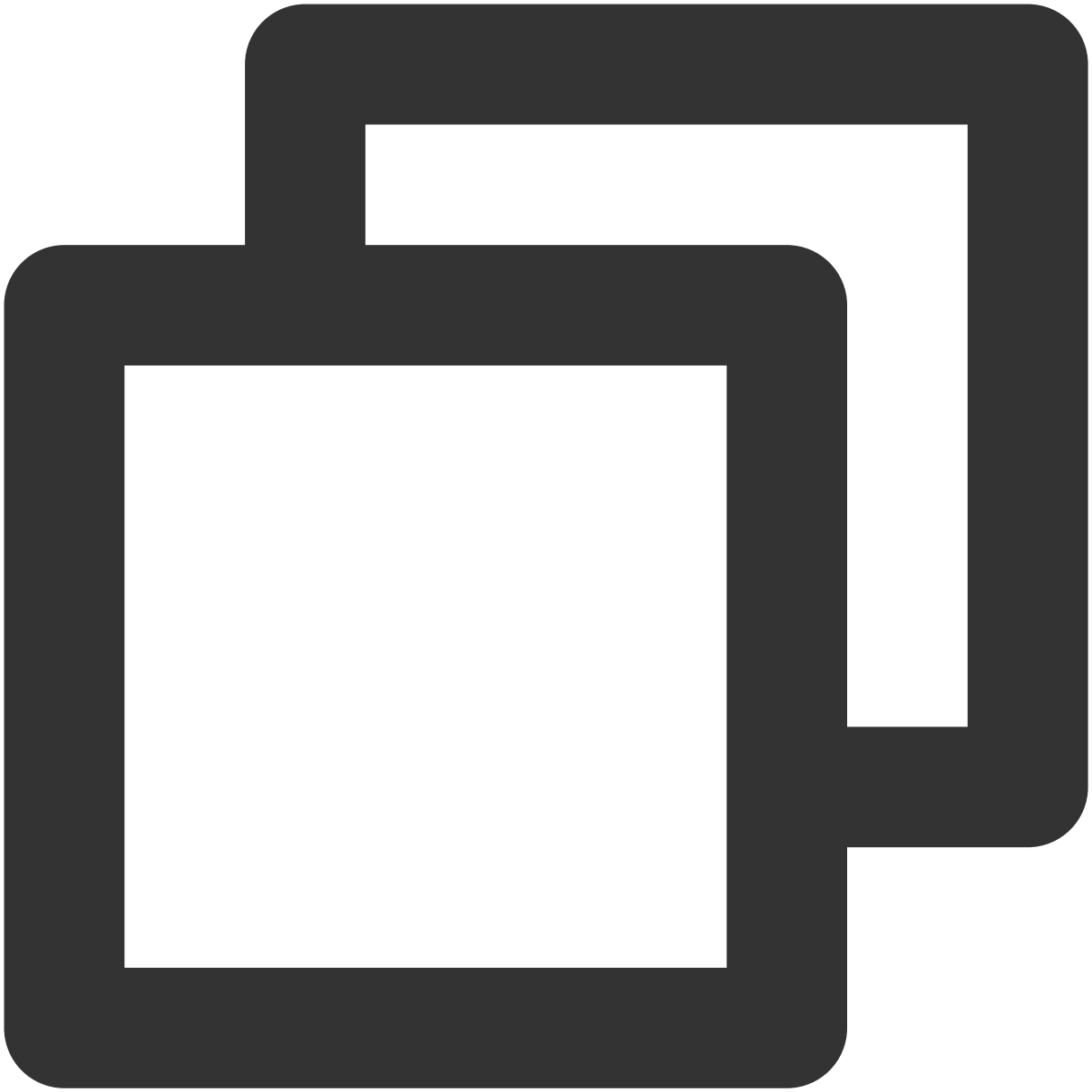
2. 创建 UFS 数据集 Dataset（COS 为例）

先创建 secret.yaml 用于加密， 模版如下：



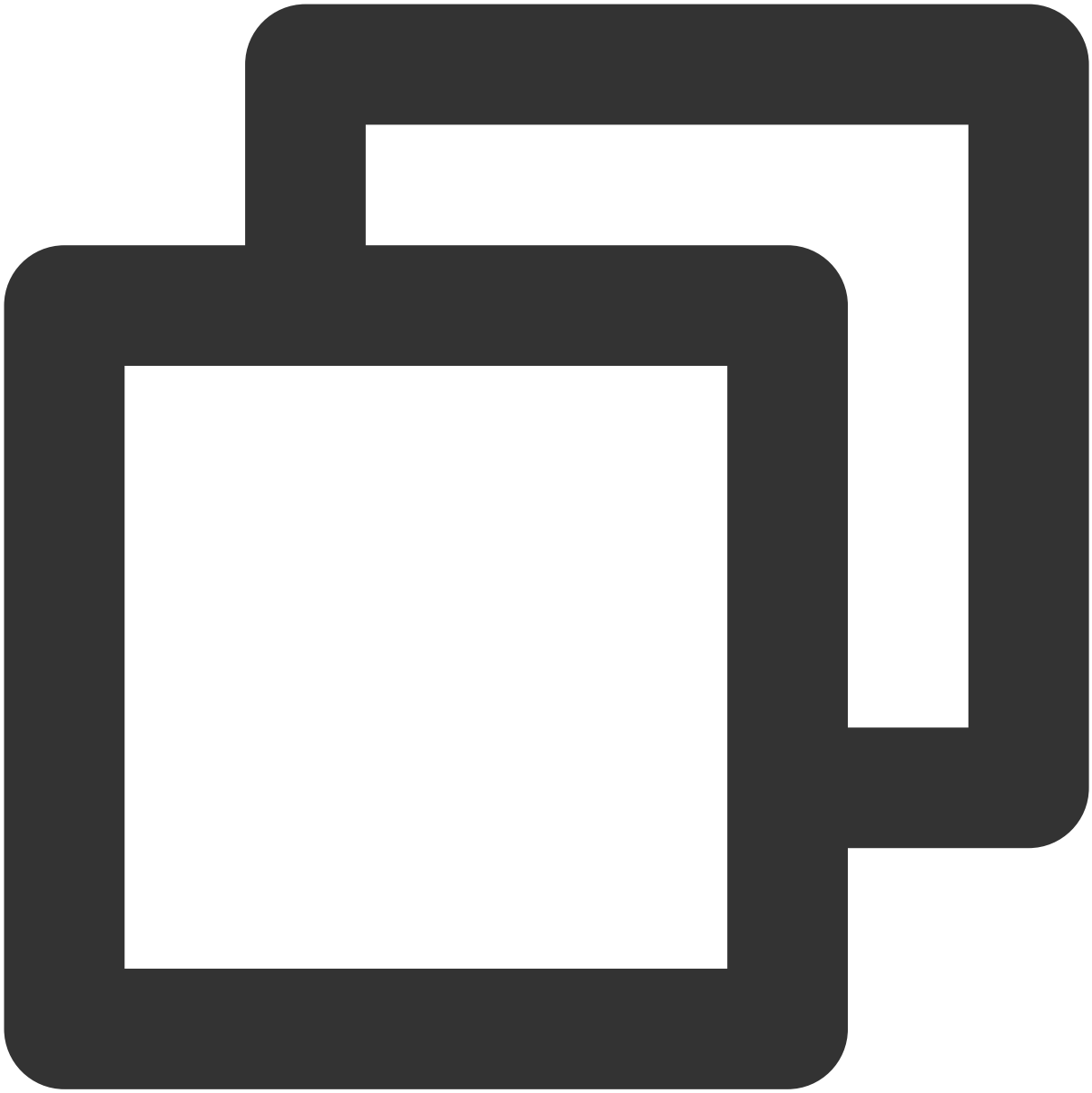
```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
stringData:
  fs.cosn.userinfo.secretKey: xxx
  fs.cosn.userinfo.secretId:xxx
```

创建secret：



```
[root@master01 ~]# kubectl apply  -f secret.yaml
secret/mysecret created
```

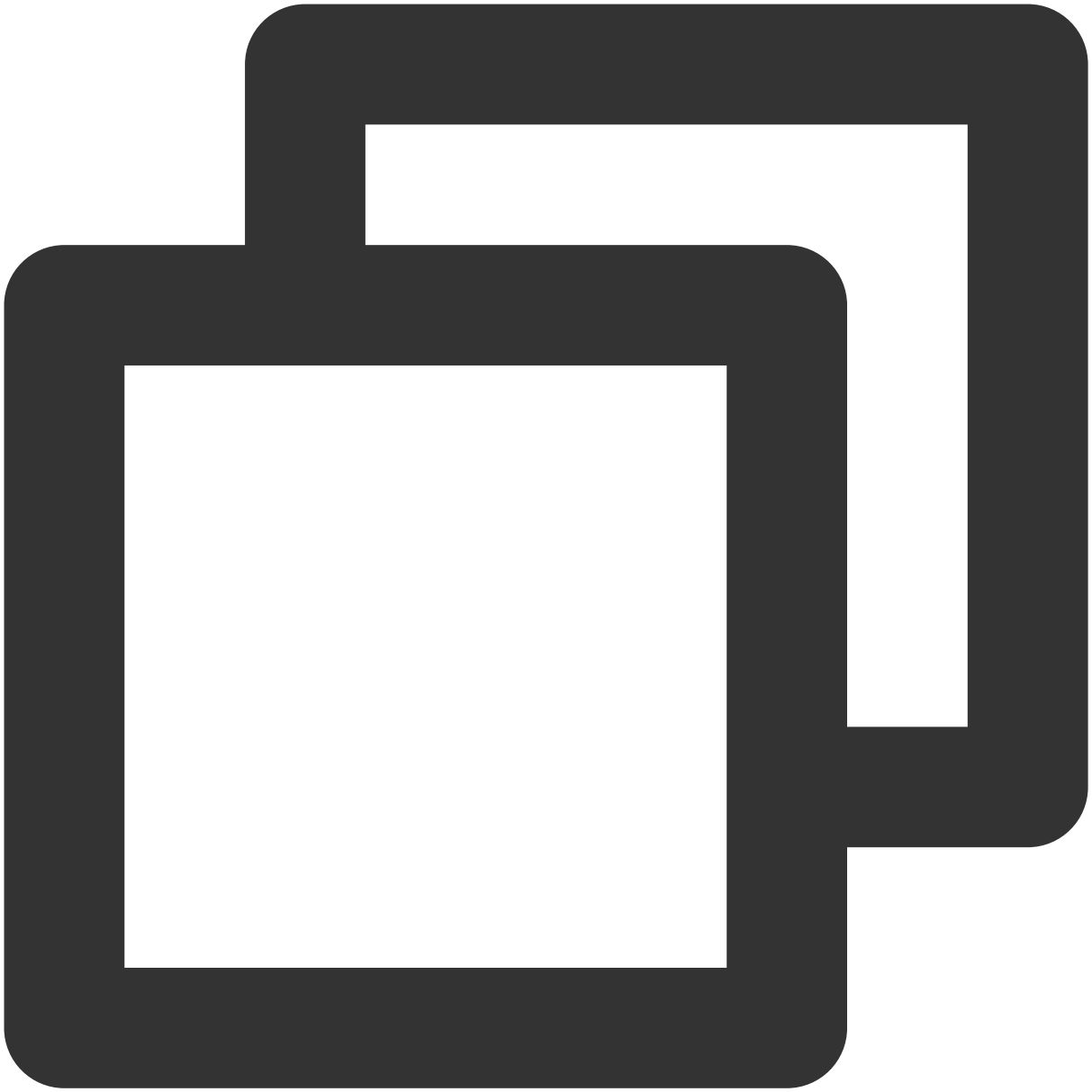
dataset.yaml 模版如下：



```
apiVersion: data.fluid.io/v1alpha1
kind: Dataset
metadata:
  name: slice1
spec:
  mounts:
  - mountPoint: cosn://{your bucket}
    name: slice1
    options:
      fs.cosn.bucket.region: ap-beijing
      fs.cosn.impl: org.apache.hadoop.fs.CosFileSystem
```

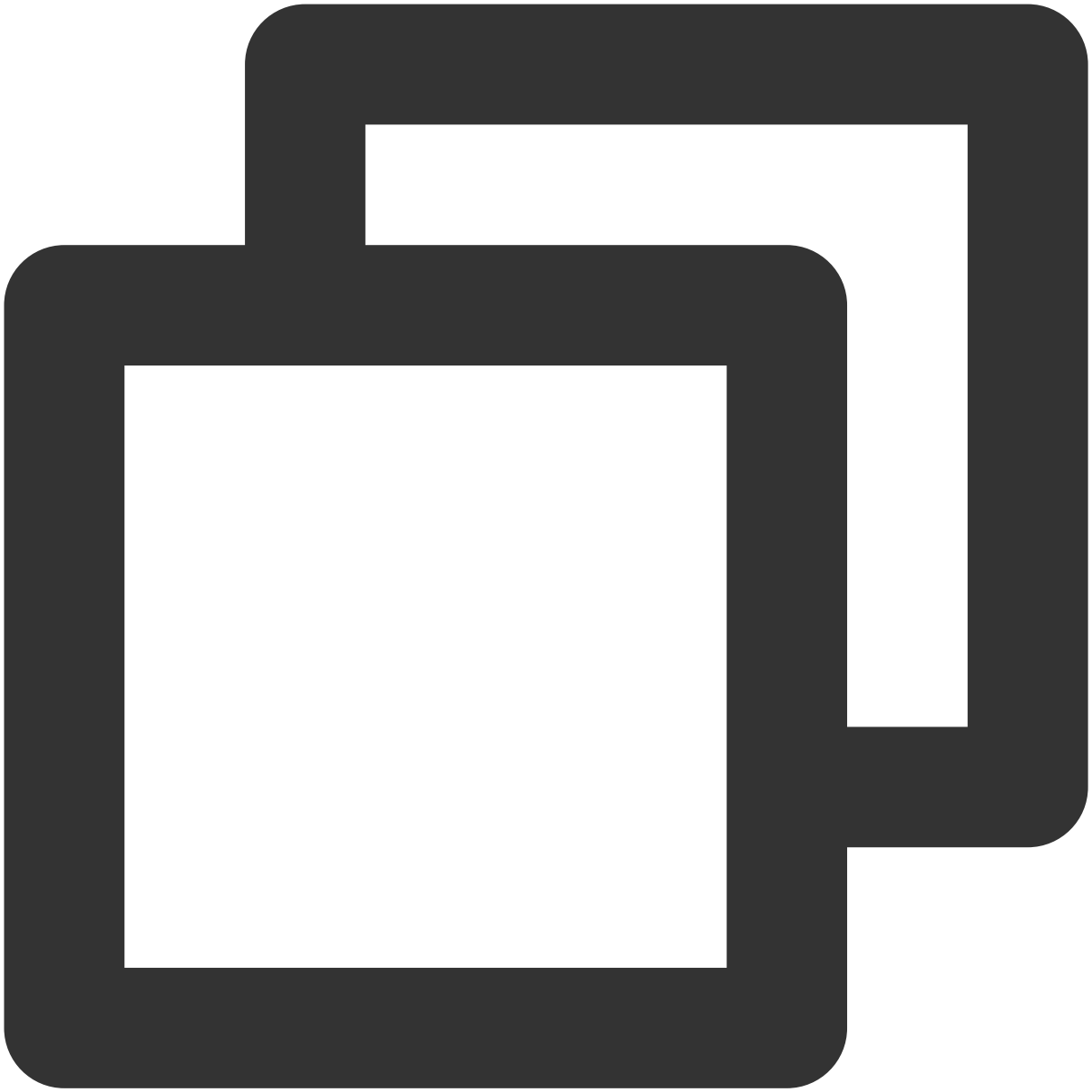
```
    fs.AbstractFileSystem.cosn.impl: org.apache.hadoop.fs.CosN
    fs.cosn.userinfo.appid: "${your appid}"
  encryptOptions:
    - name: fs.cosn.userinfo.secretKey
      valueFrom:
        secretKeyRef:
          name: mysecret
          key: fs.cosn.userinfo.secretKey
    - name: fs.cosn.userinfo.secretId
      valueFrom:
        secretKeyRef:
          name: mysecret
          key: fs.cosn.userinfo.secretId
```

创建 dataset



```
[root@master01 run]# kubectl apply -f dataset.yaml
dataset.data.fluid.io/slice1 created
```

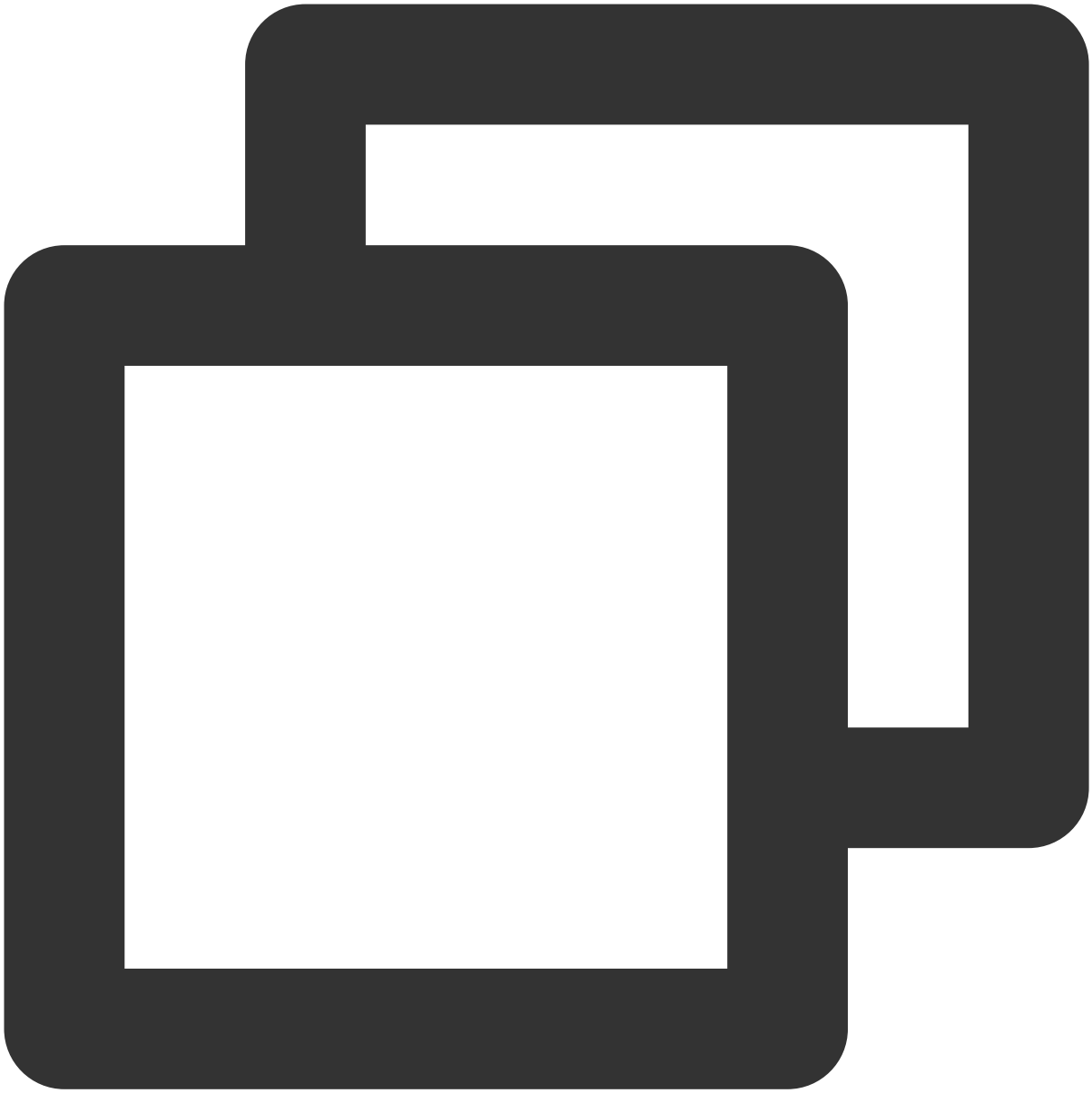
查询 Dataset 状态，处于 NotBond 状态：



```
[root@master01 run]# kubectl get dataset
NAME     UFS TOTAL SIZE   CACHED   CACHE CAPACITY   CACHED PERCENTAGE   PHASE
slice1
                                                                        NotBound   11s
```

3. 创建 runtime

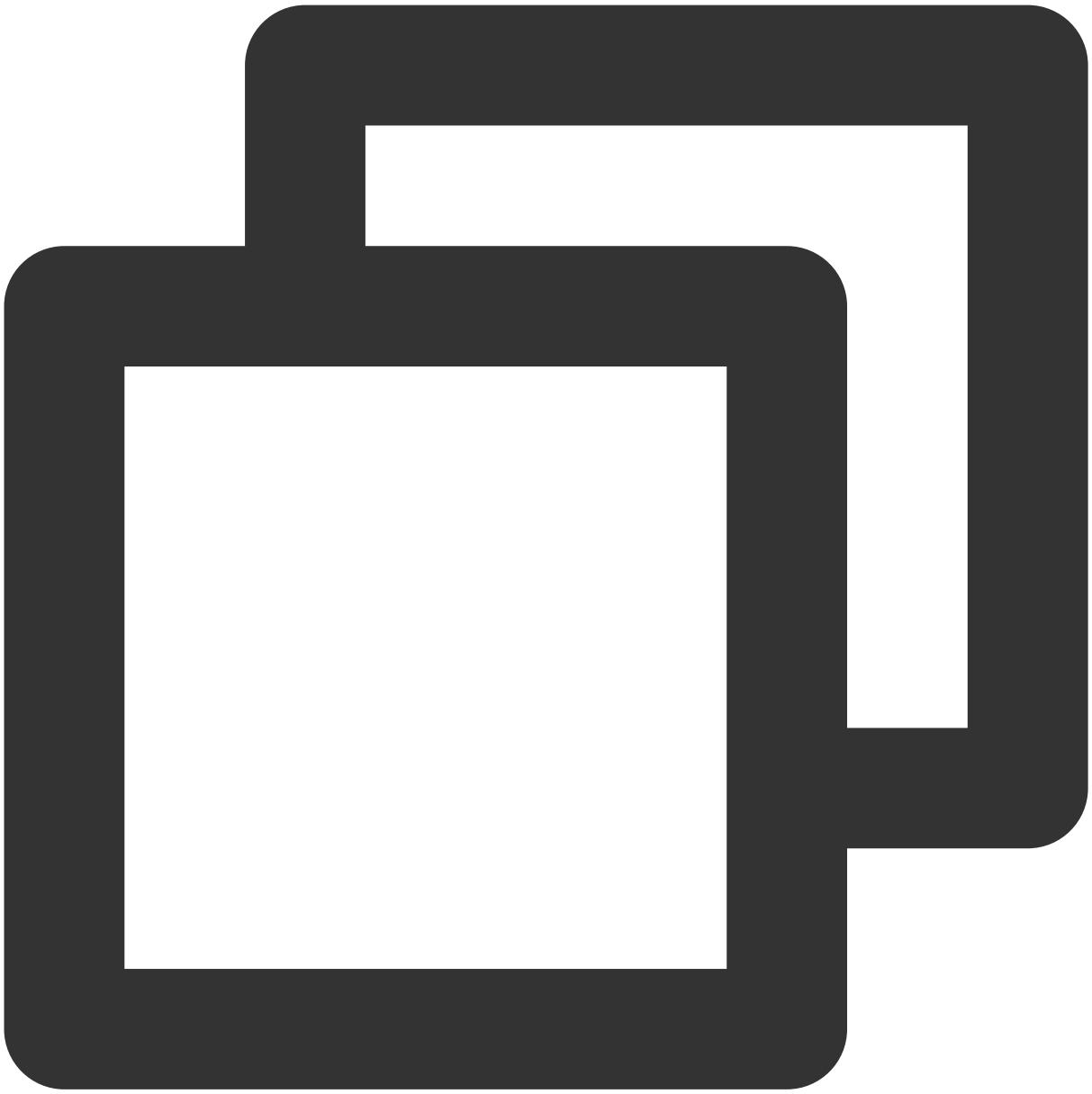
runtime.yaml 模板如下：



```
apiVersion: data.fluid.io/v1alpha1
kind: GooseFSRuntime
metadata:
  name: slice1
spec:
  replicas: 1
  data:
    replicas: 1
  goosefsVersion:
    imagePullPolicy: Always
    image: ${img_uri}
```

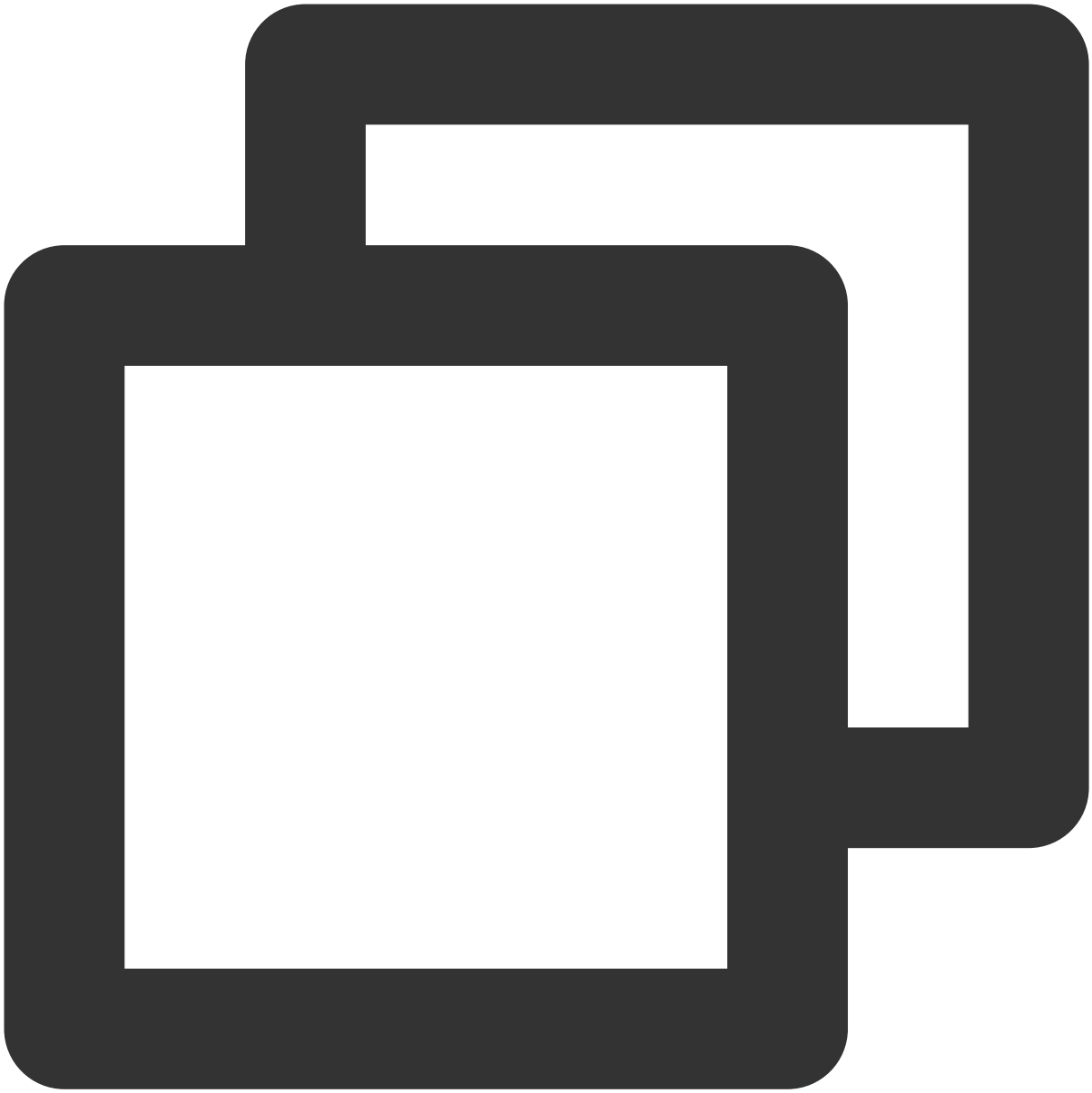
```
    imageTag: ${tag}
  tieredstore:
    levels:
      - mediumtype: MEM
        path: /dev/shm
        quota: 1Gi
        high: "0.95"
        low: "0.7"
  properties:
    # goosefs user
    goosefs.user.file.writetype.default: MUST_CACHE
  master:
    replicas: 1
    journal:
      volumeType: hostpath
    jvmOptions:
      - "-Xmx40G"
      - "-XX:+UnlockExperimentalVMOptions"
      - "-XX:ActiveProcessorCount=8"
  worker:
    jvmOptions:
      - "-Xmx12G"
      - "-XX:+UnlockExperimentalVMOptions"
      - "-XX:MaxDirectMemorySize=32g"
      - "-XX:ActiveProcessorCount=8"
    resources:
      limits:
        cpu: 8
  fuse:
    imagePullPolicy: Always
    image: ${fuse_uri}
    imageTag: ${tag_num}
    env:
      MAX_IDLE_THREADS: "32"
    jvmOptions:
      - "-Xmx16G"
      - "-Xms16G"
      - "-XX:+UseG1GC"
      - "-XX:MaxDirectMemorySize=32g"
      - "-XX:+UnlockExperimentalVMOptions"
      - "-XX:ActiveProcessorCount=24"
    resources:
      limits:
        cpu: 16
    args:
      - fuse
      - --fuse-opts=kernel_cache,ro,max_read=131072,attr_timeout=7200,entry_timeout
```

创建 runtime：



```
[root@master01 run]# kubectl apply -f runtime.yaml
goosefsruntime.data.fluid.io/slice1 created
```

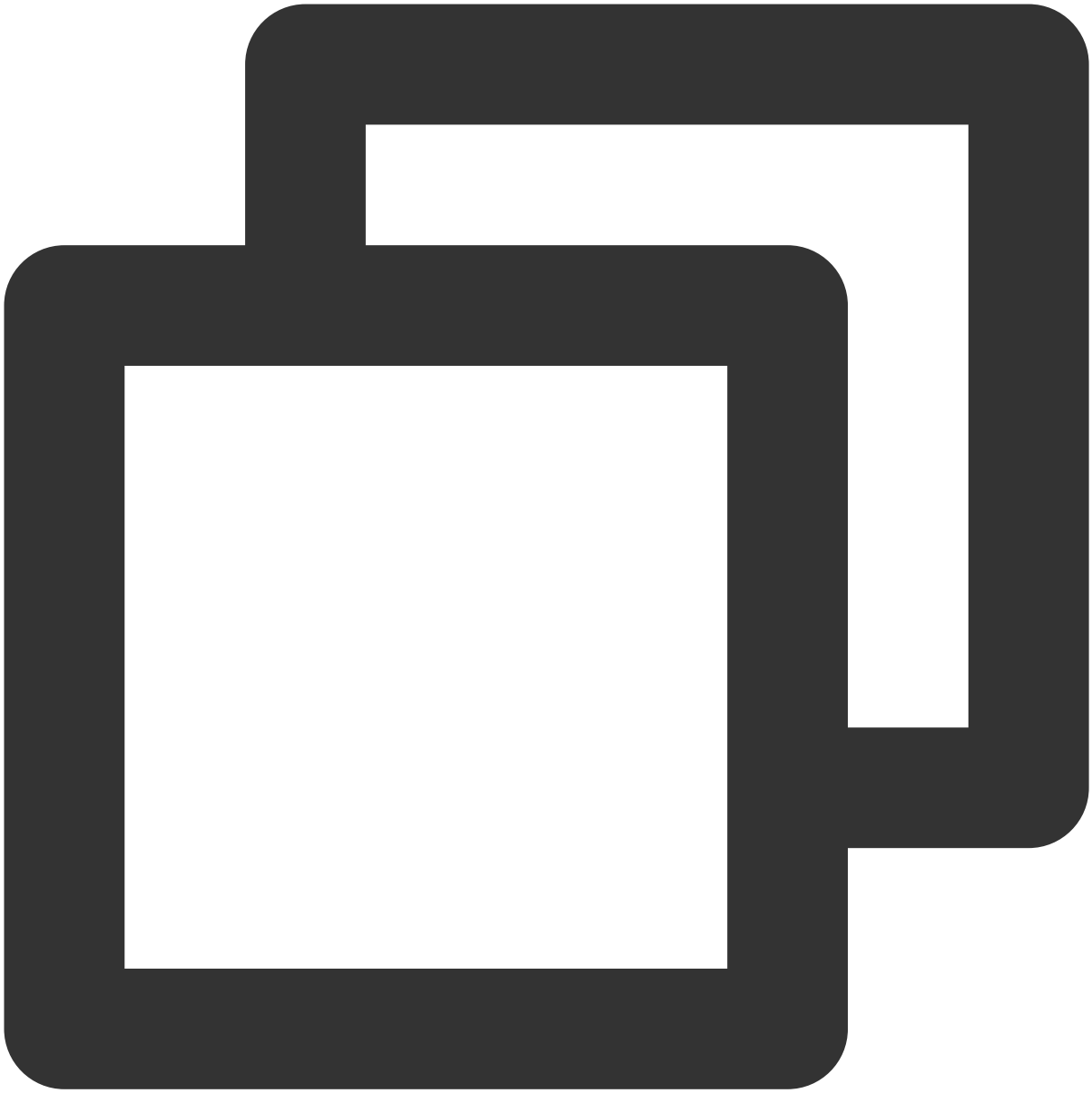
检查 goosefs 组件状态：



```
[root@master01 run]# kubectl get pods
NAME                   READY   STATUS    RESTARTS   AGE
slice1-fuse-xsvwj      1/1     Running   0          37s
slice1-master-0        2/2     Running   0          118s
slice1-worker-fzpdw    2/2     Running   0          37s
```

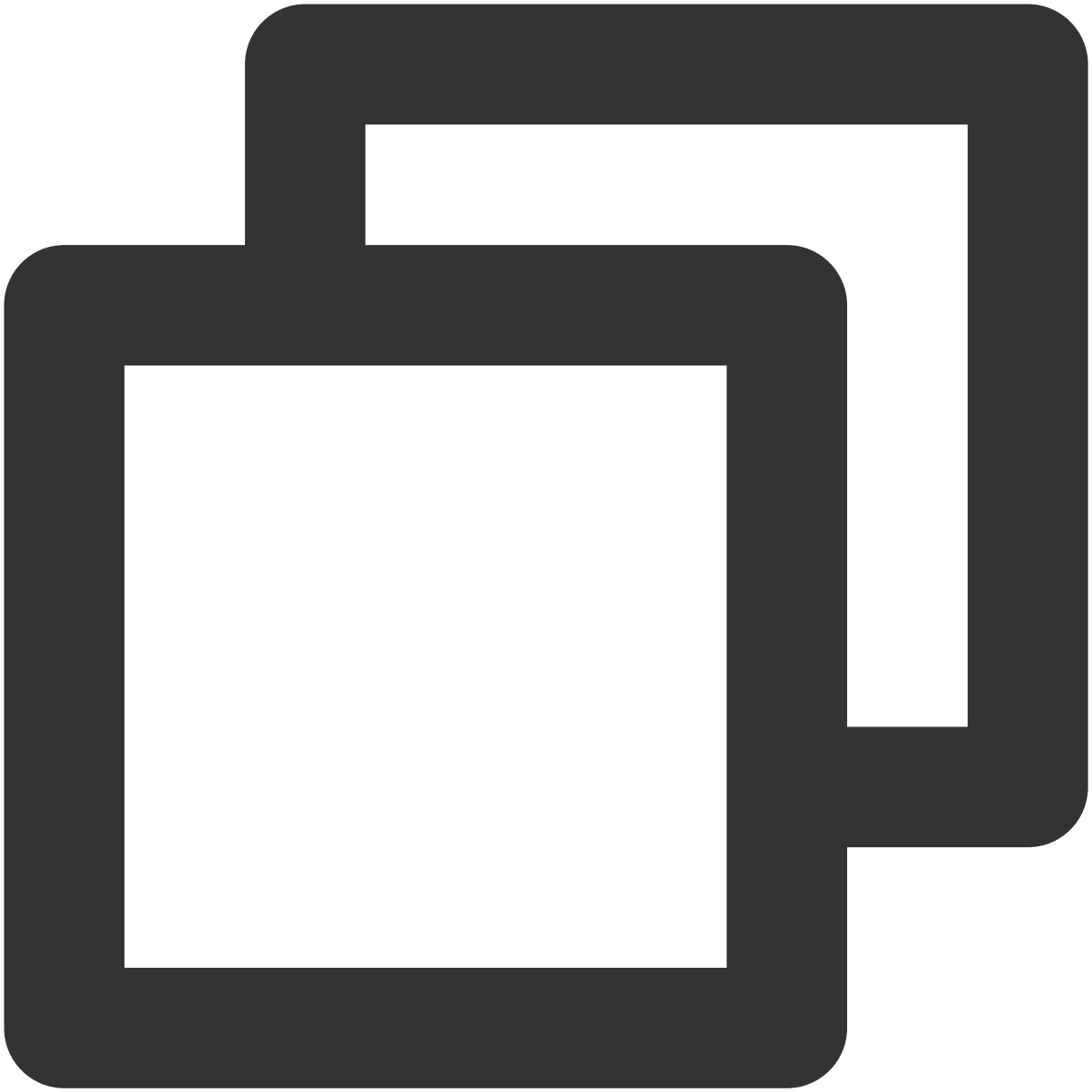
4. 预热数据

dataload.yaml 预热组件如下：



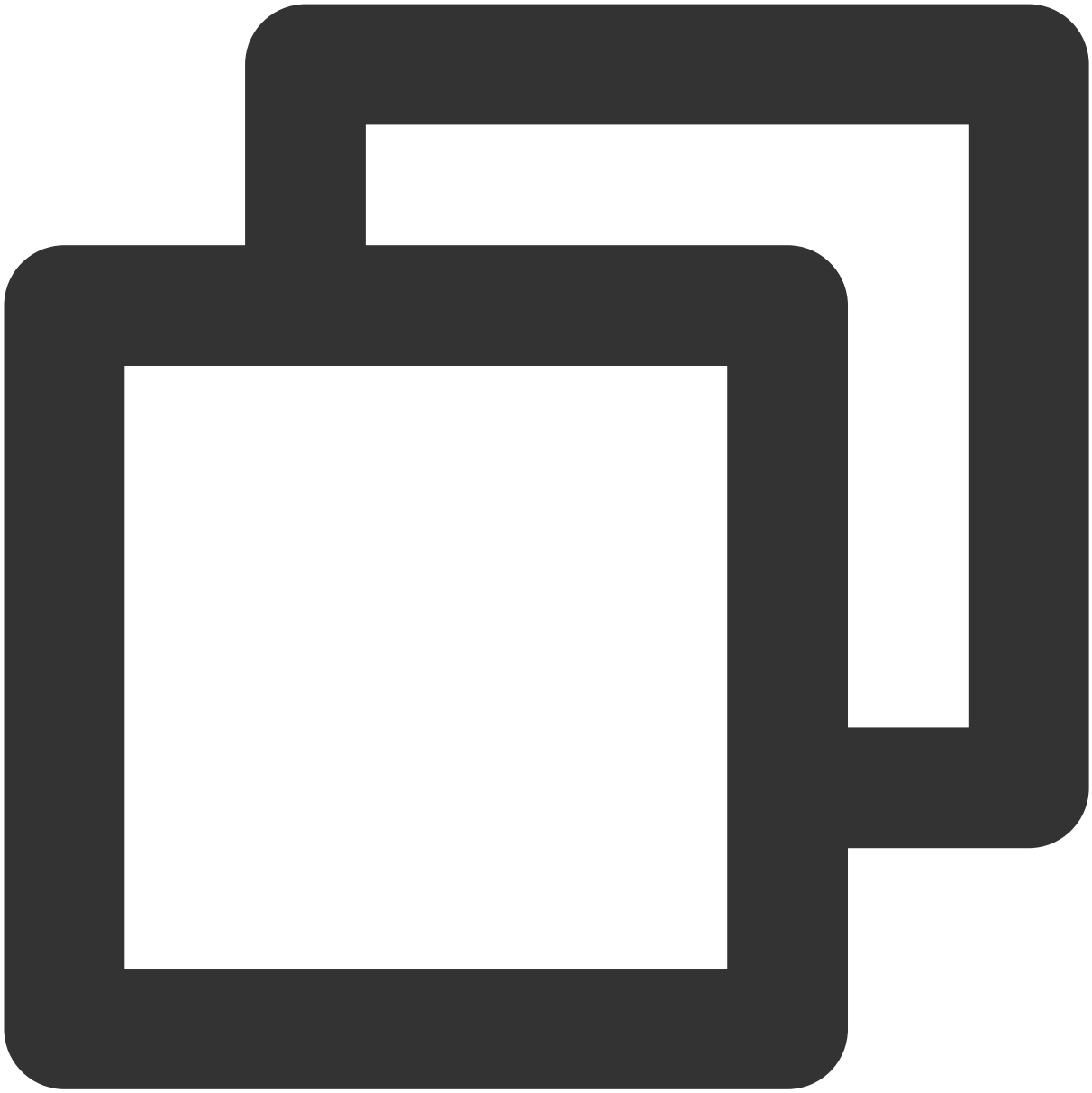
```
apiVersion: data.fluid.io/v1alpha1
kind: DataLoad
metadata:
  name: slice1-dataload
spec:
  dataset:
    name: slice1
    namespace: default
```

此时 Dataset 处于 Bond 状态，Cached 比例是0%。



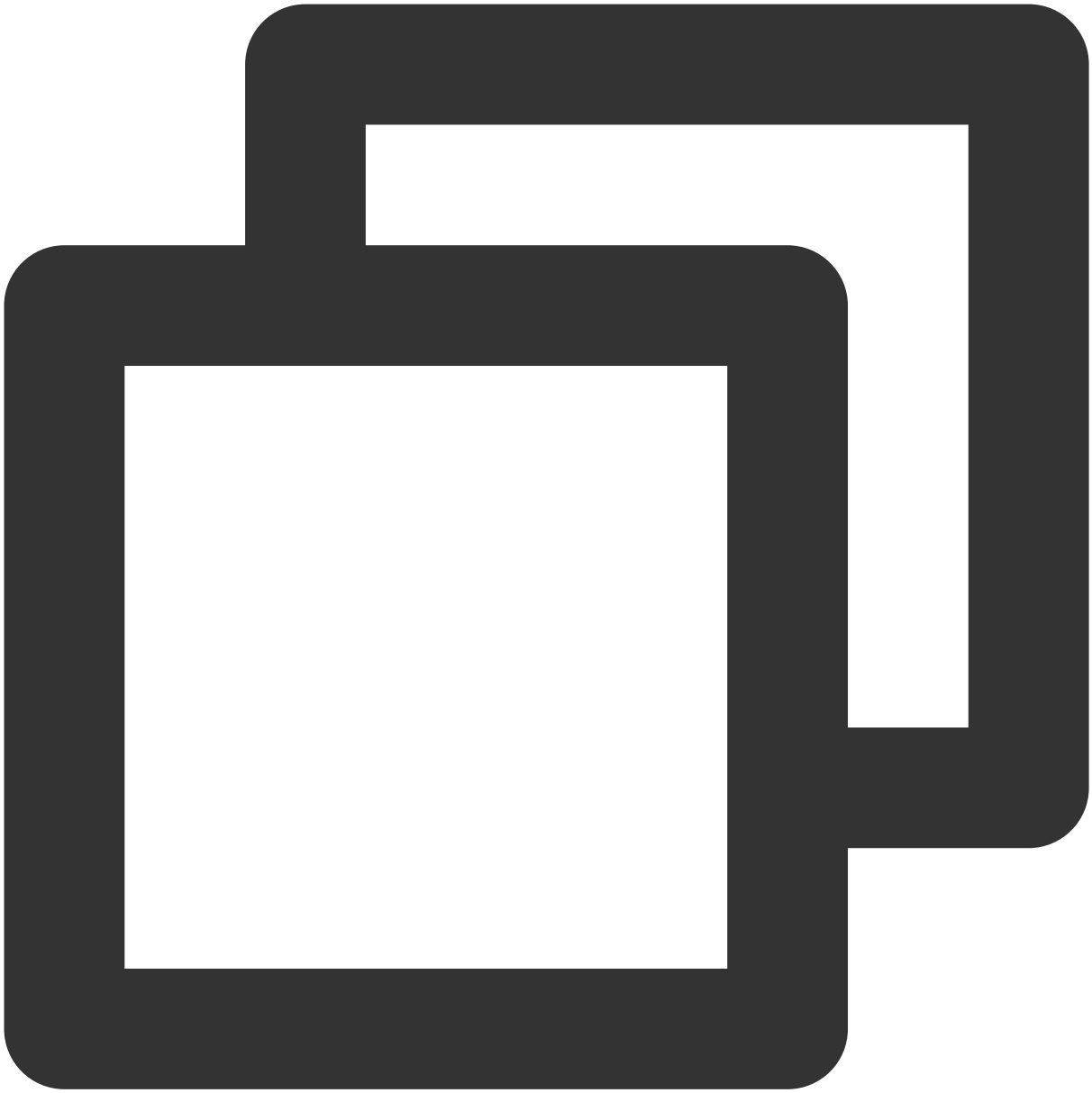
```
[root@master01 run]# kubectl get dataset
NAME     UFS TOTAL SIZE   CACHED   CACHE CAPACITY   CACHED PERCENTAGE   PHASE   AGE
slice1   97.67MiB         0.00B    4.00GiB          0.0%                Bound   31m
```

执行数据预热：



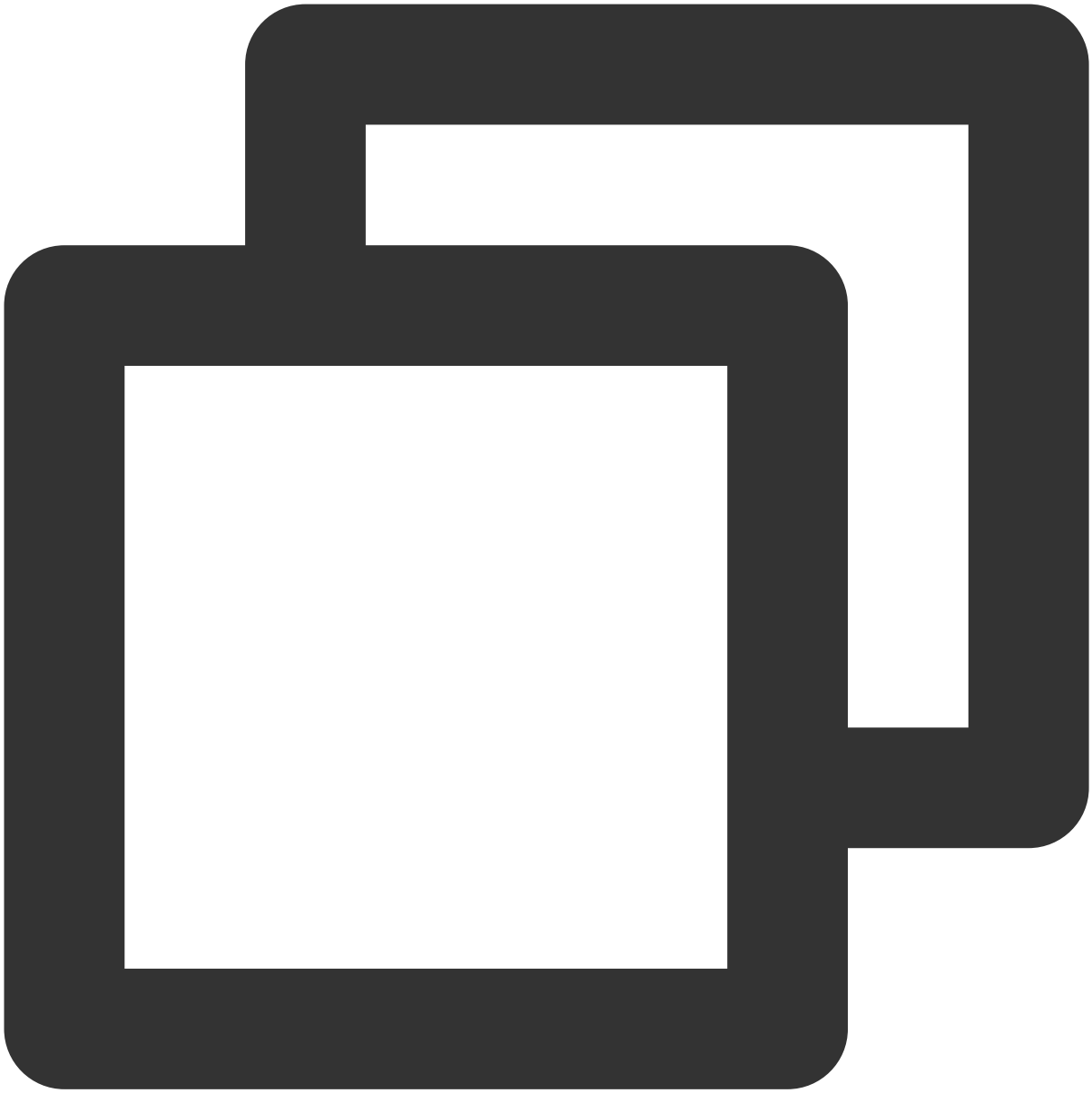
```
[root@master01 run]# kubectl apply -f dataload.yaml
dataload.data.fluid.io/slice1-dataload created
```

查看数据预热进度：



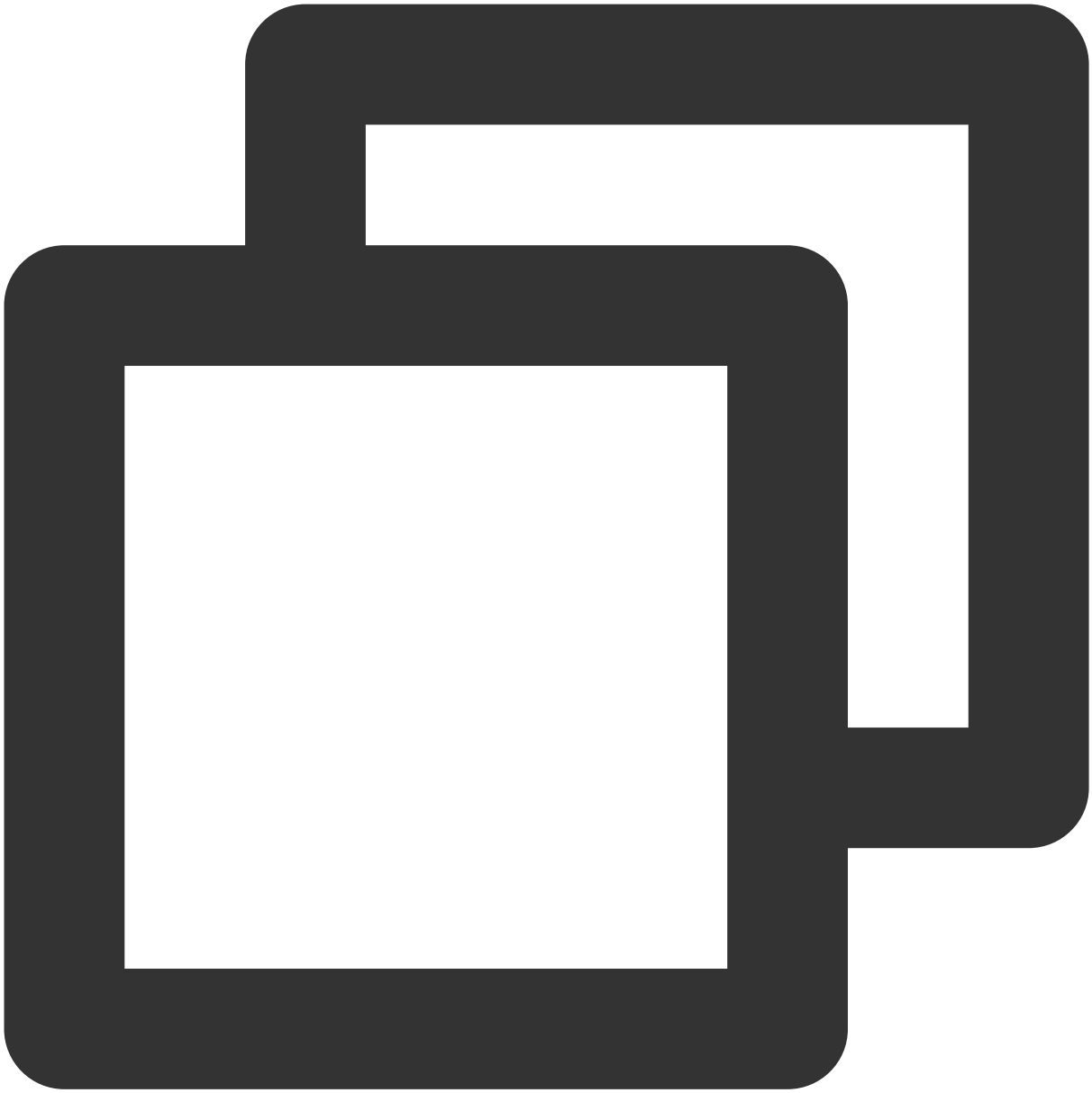
```
[root@master01 run]# kubectl get dataset --watch
NAME     UFS TOTAL SIZE   CACHED     CACHE CAPACITY   CACHED PERCENTAGE   PHASE   A
slice1   97.67MiB         52.86MiB   4.00GiB          54.1%               Bound   3
slice1   97.67MiB         53.36MiB   4.00GiB          54.6%               Bound   3
slice1   97.67MiB         53.36MiB   4.00GiB          54.6%               Bound   3
slice1   97.67MiB         53.87MiB   4.00GiB          55.2%               Bound   3
slice1   97.67MiB         53.87MiB   4.00GiB          55.2%               Bound   3
```

数据预热完成100%：



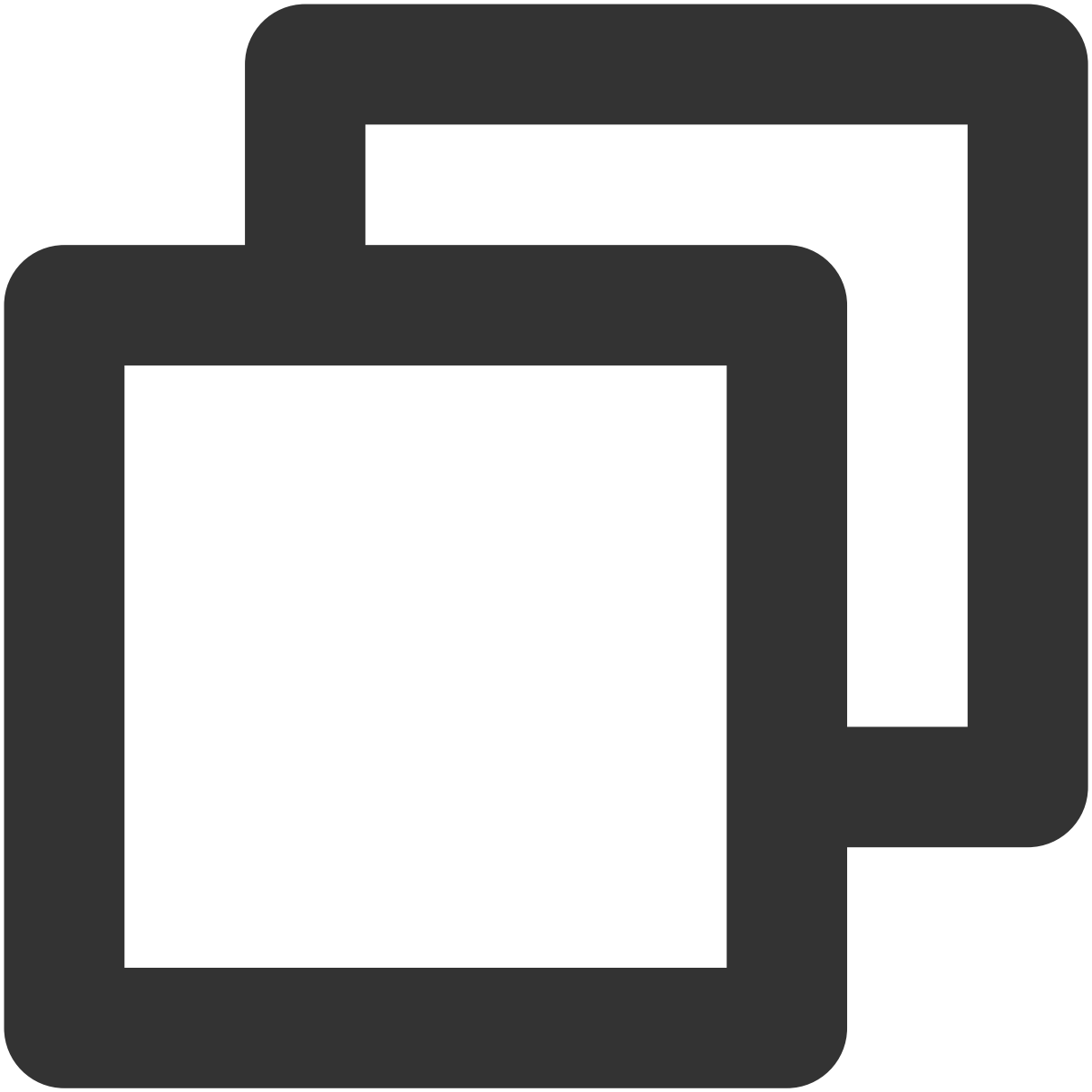
```
[root@master01 run]# kubectl get dataset --watch
NAME     UFS TOTAL SIZE   CACHED     CACHE CAPACITY   CACHED PERCENTAGE   PHASE   A
slice1   97.67MiB         97.67MiB   4.00GiB          100.0%              Bound   4
```

5. 检查数据



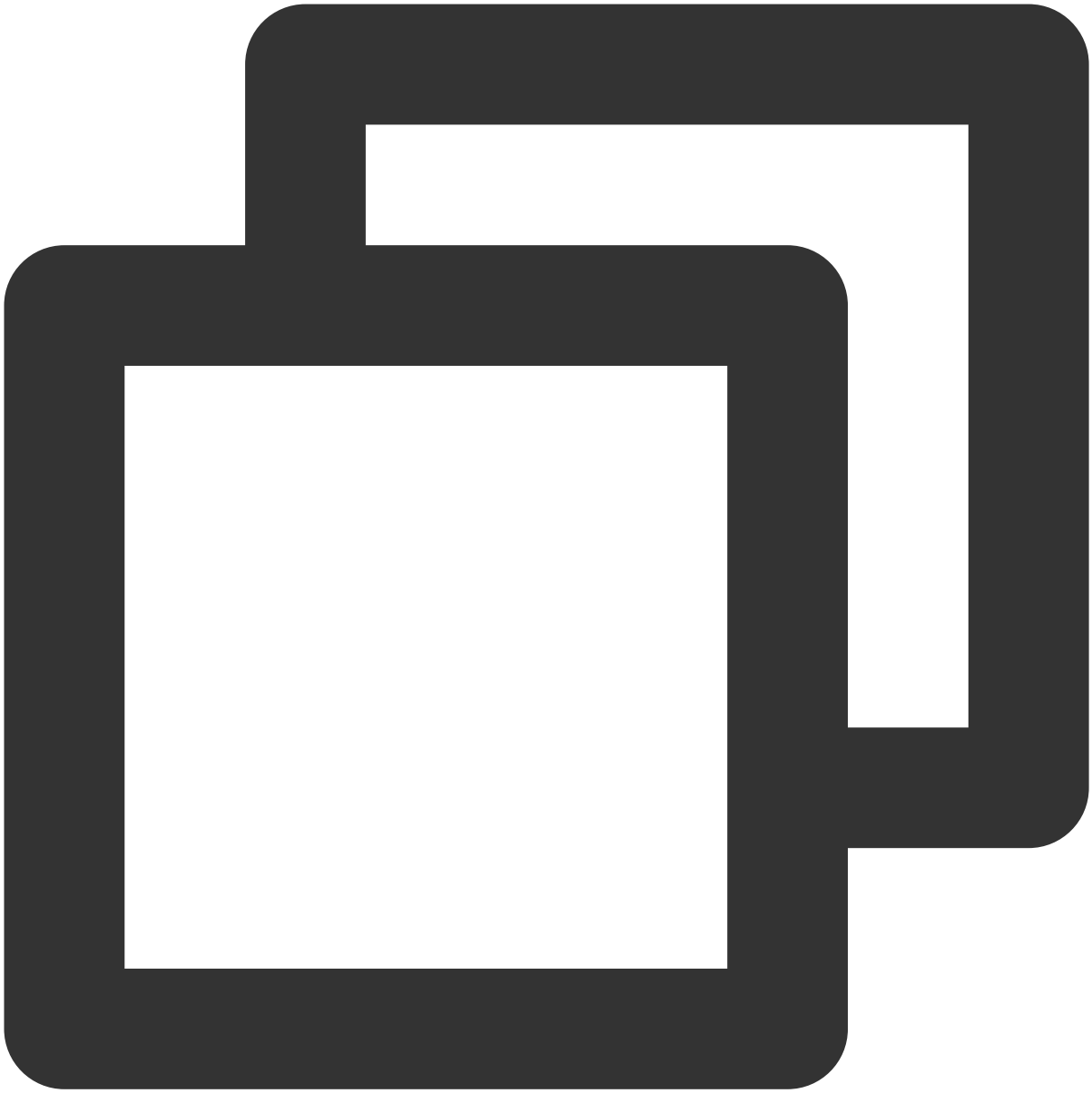
```
[root@master01 run]# kubectl get pods
NAME                                  READY   STATUS      RESTARTS   AGE
slice1-dataload-loader-job-km6mg      0/1     Completed   0          12m
slice1-fuse-xsvwj                     1/1     Running     0          17m
slice1-master-0                       2/2     Running     0          19m
slice1-worker-fzpdw                   2/2     Running     0          17m
```

进入 goosefs master 容器：



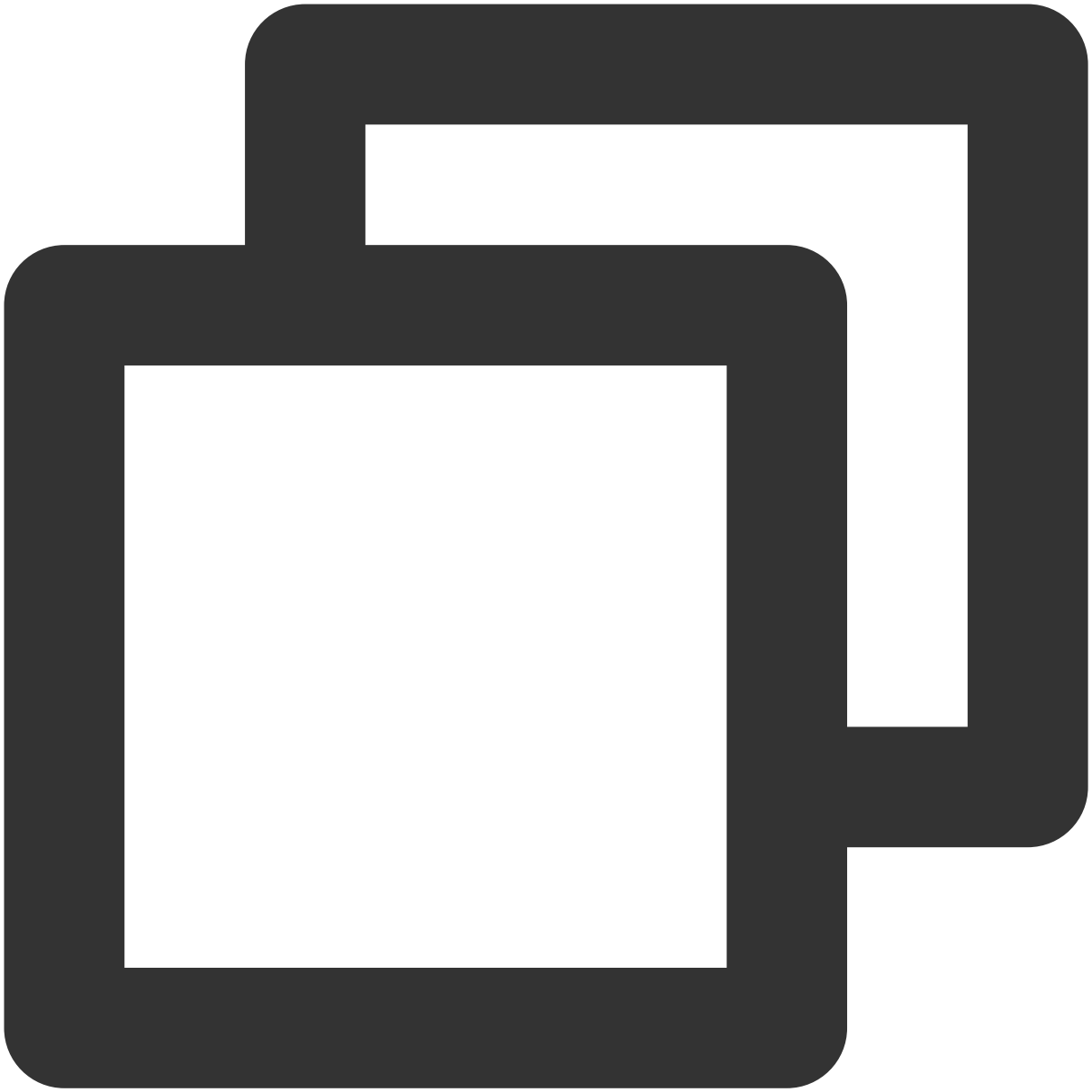
```
[root@master01 run]# kubectl exec -it slice1-master-0 -- /bin/bash
Defaulting container name to goosefs-master.
```

列出 goosefs 目录：



```
[root@VM-2-40-tlinux goosefs-1.0.0-SNAPSHOT-noUI-noHelm]# goosefs fs ls /slice1
          10240       PERSISTED 06-25-2021 16:45:11:809 100% /slice1/p1
              1       PERSISTED 05-24-2021 16:07:37:000  DIR /slice1/a
          10000       PERSISTED 05-26-2021 19:29:05:000  DIR /slice1/p2
```

查看某个具体文件：



```
[root@VM-2-40-tlinux goosefs-1.0.0-SNAPSHOT-noUI-noHelm]# goosefs fs ls /slice1/a/
            12       PERSISTED 06-25-2021 16:45:11:809 100% /slice1/a/1.xt
```

使用 Docker 部署

最近更新时间：2024-01-06 11:17:58

本文主要介绍如何使用 Docker 部署 GooseFS。

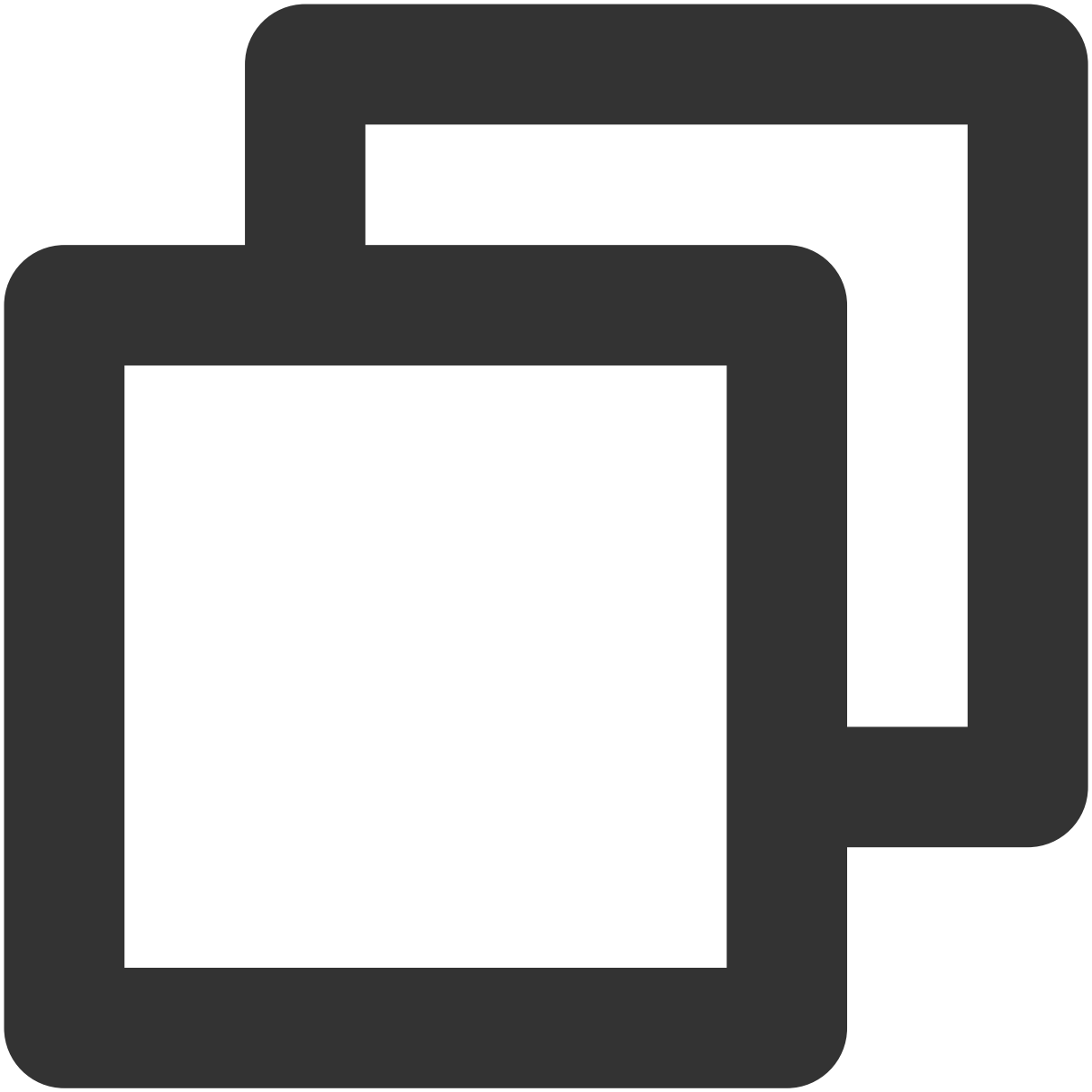
准备事项

1. 安装 Docker 19.03.14 及以上。
2. CentOS 7及以上。
3. 已获取 GooseFS docker 镜像，例如 goosefs:v1.0.0。

安装步骤

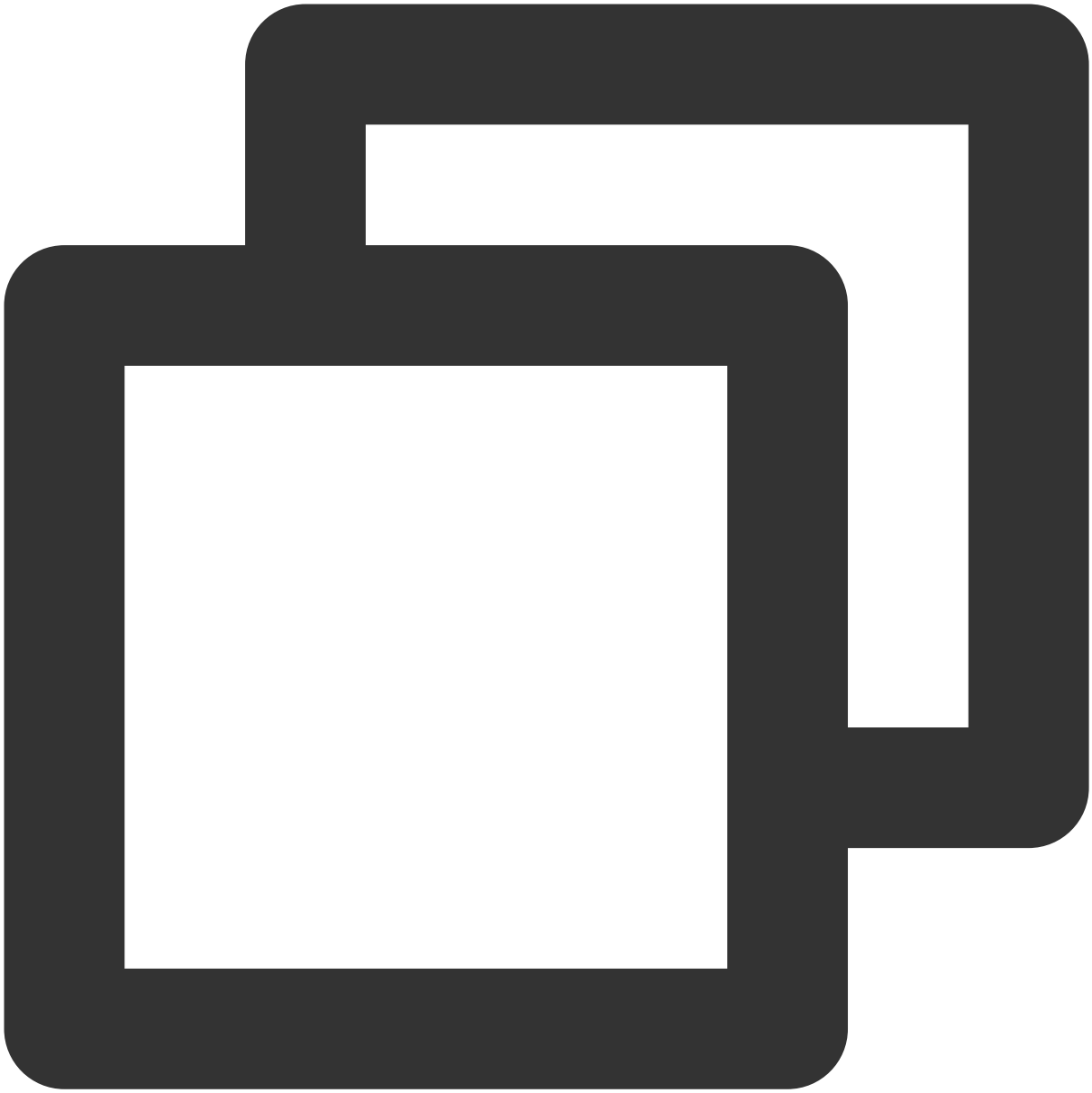
1. 创建 ufs 目录，将本机目录挂载到 GooseFS 根目录：

版权所有：腾讯云计算（北京）有限责任公司



```
mkdir /tmp/goosefs_ufs
```

2. 启动 master 进程：



```
docker run -d  --rm \\
--net=host \\
--name=goosefs-master \\
-v /tmp/goosefs_ufs:/opt/data \\
-e GOOSEFS_JAVA_OPTS=" \\
-Dgoosefs.master.hostname=localhost \\
-Dgoosefs.master.mount.table.root.ufs=/opt/data" \\
goosefs:v1.0.0 master
```

说明

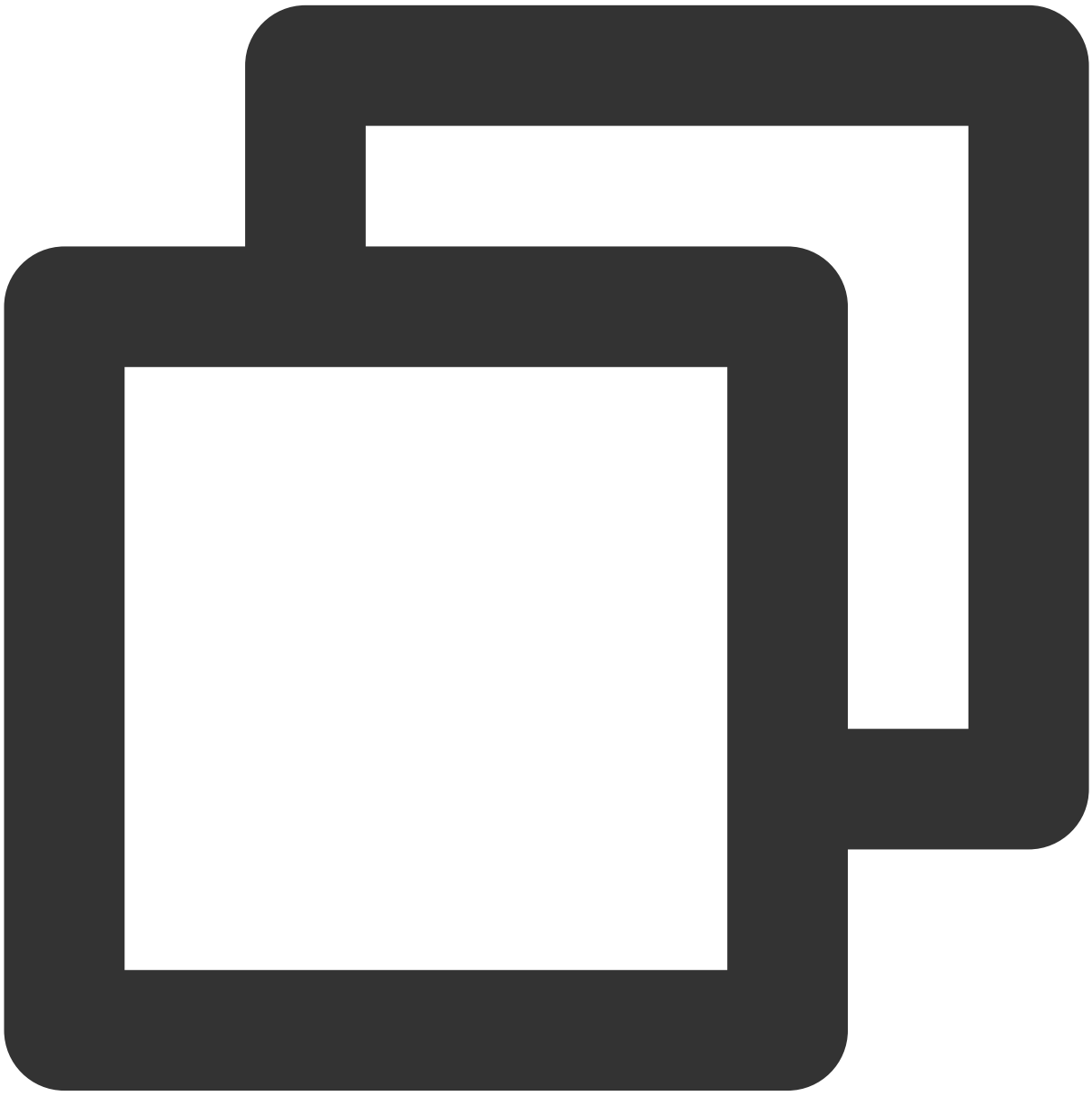
-Dgoosefs.master.hostname：设置 master 地址。

-Dgoosefs.master.mount.table.root.ufs：设置 GooseFS 根目录挂载点。

-v /tmp/goosefs_ufs:/opt/data：将本地目录映射到 docker 容器内。

--net=host：docker 采用 host 网络。

3. 启动 worker 进程：

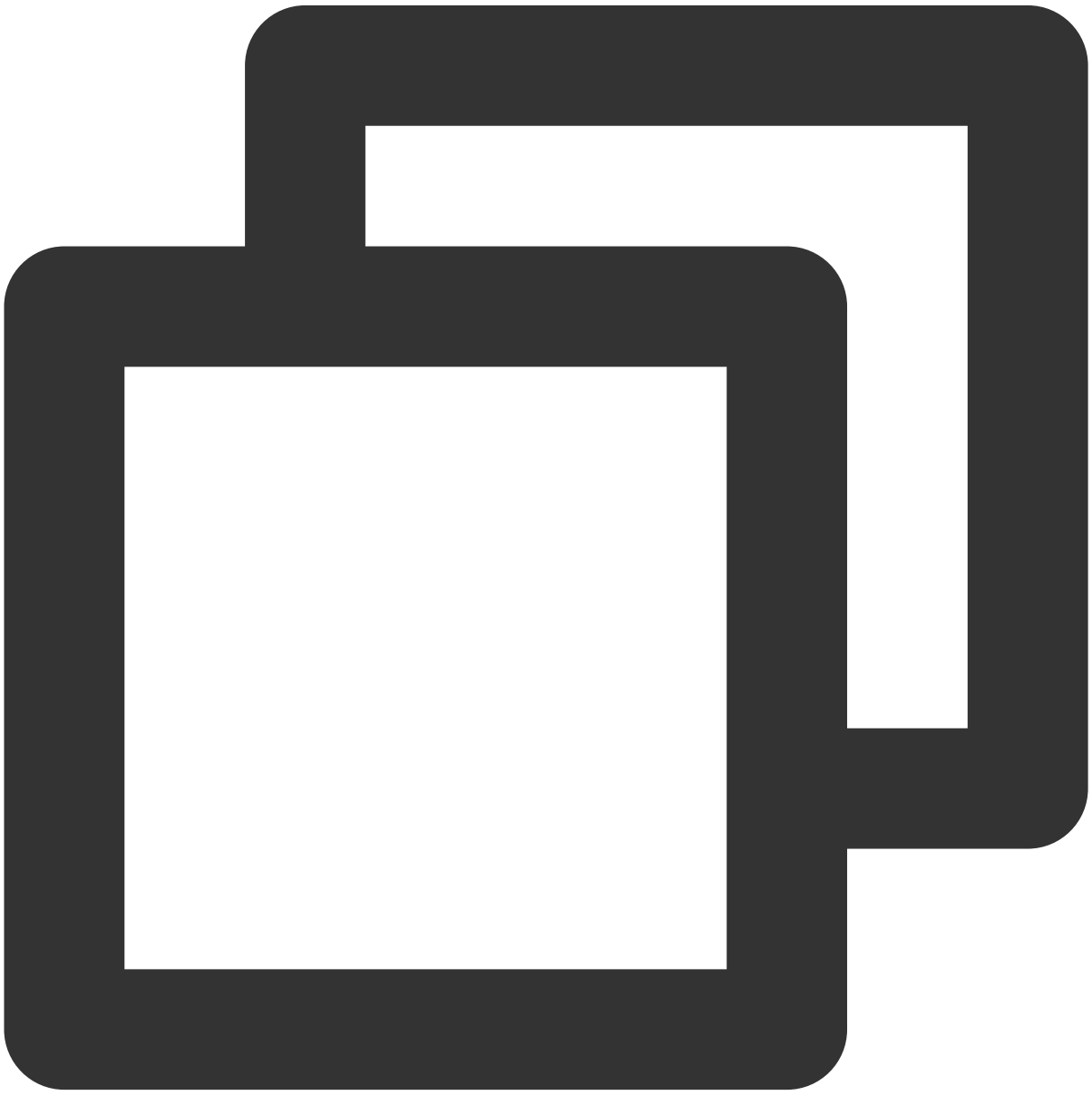


```
docker run -d --rm \\
--net=host \\
--name=goosefs-worker1 \\
--shm-size=1G \\
```

```
-e GOOSEFS_JAVA_OPTS=" \\
-Dgoosefs.worker.memory.size=1G \\
-Dgoosefs.master.hostname=localhost" \\
goosefs:v1.0.0 worker
```

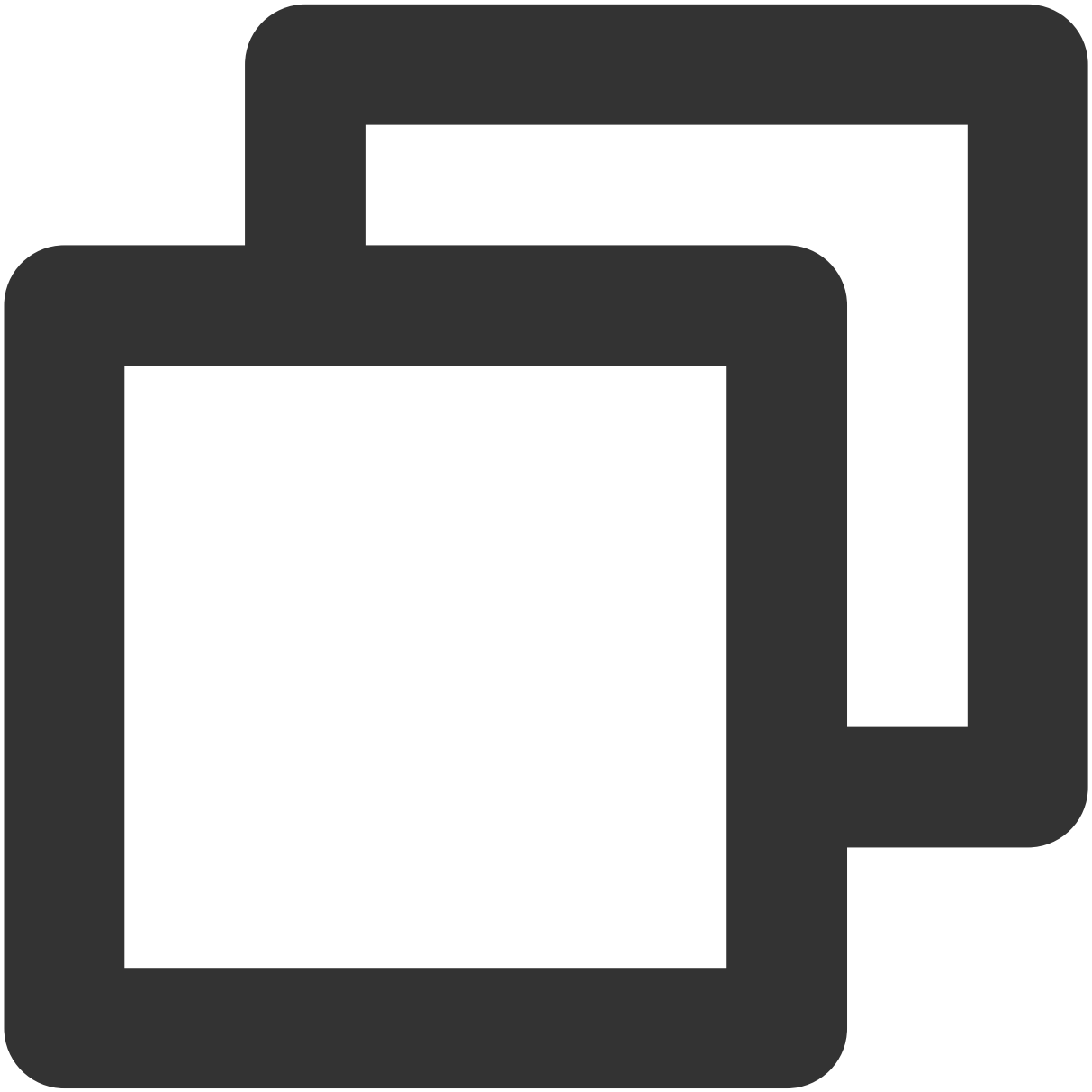
操作演示

1. 查看容器：



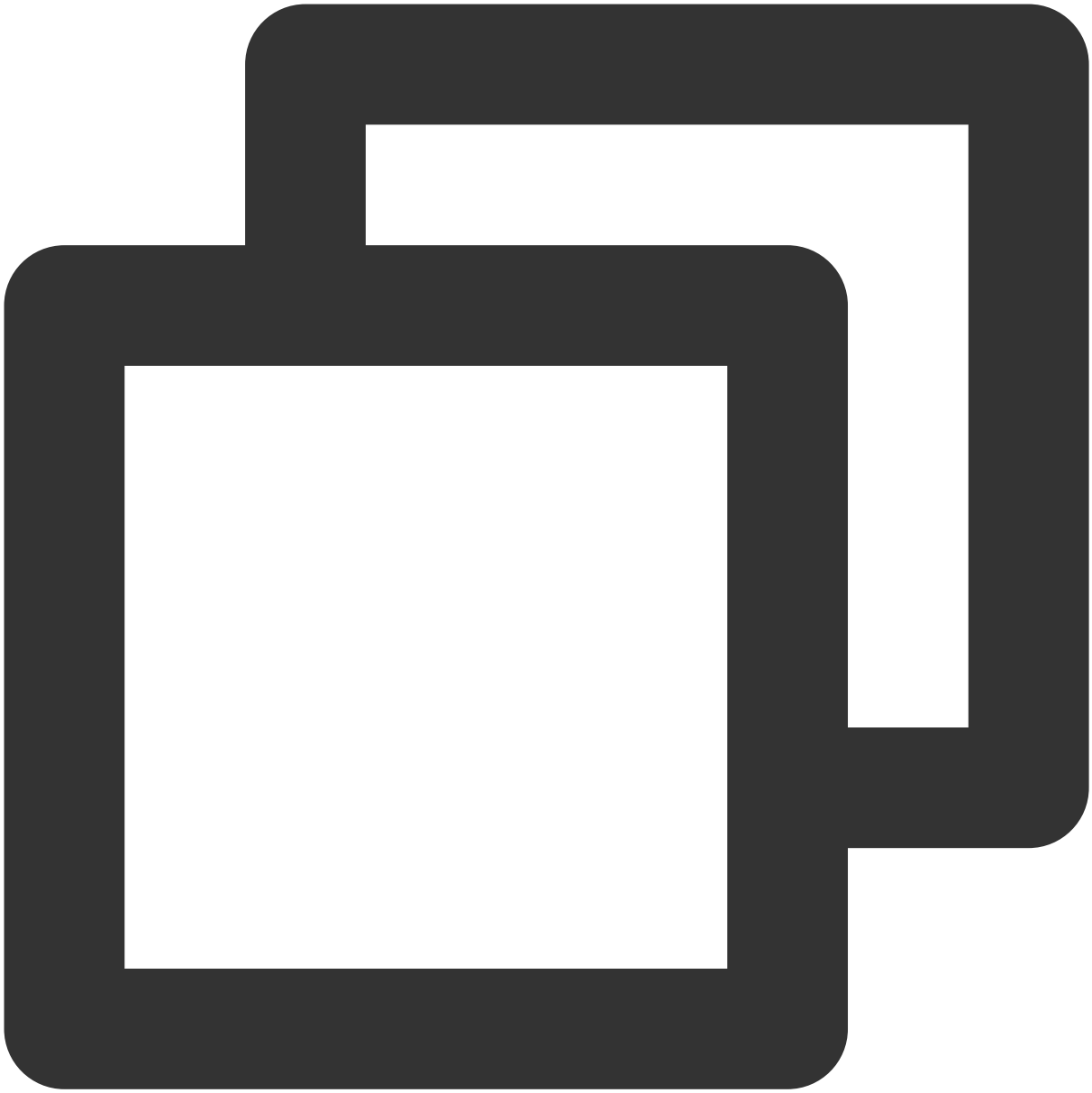
```
[root@VM-0-7-centos ~]# docker ps | grep goosefs
0bda1cac76f4        goosefs:v1.0.0      "/entrypoint.sh mast…"   32 minutes ago
b6260f9a0134        goosefs:v1.0.0      "/entrypoint.sh work…"   About an hour ago
```

2. 进入容器：



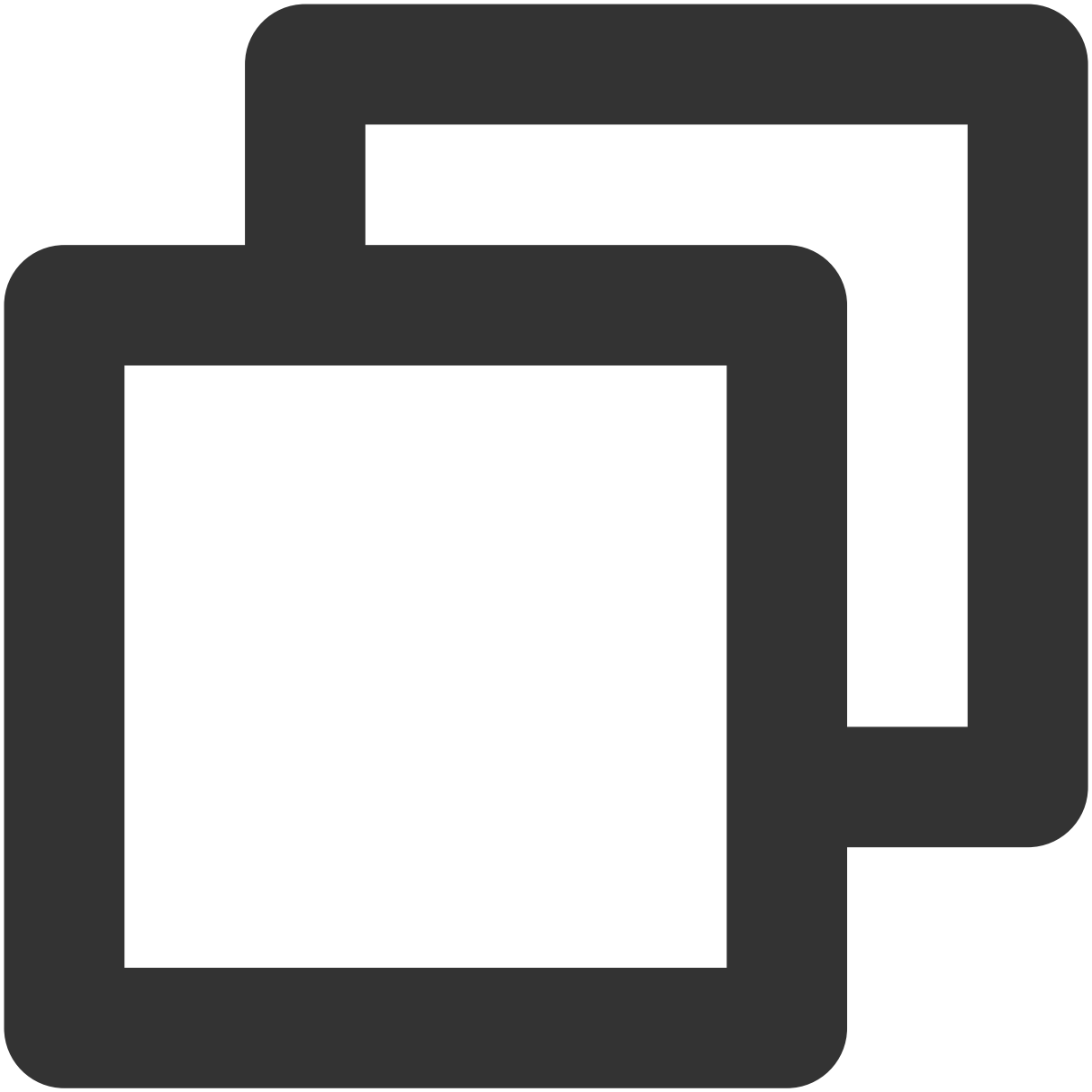
```
docker exec -it 0bda1cac76f4 /bin/bash
```

3. 挂载 COS 目录：



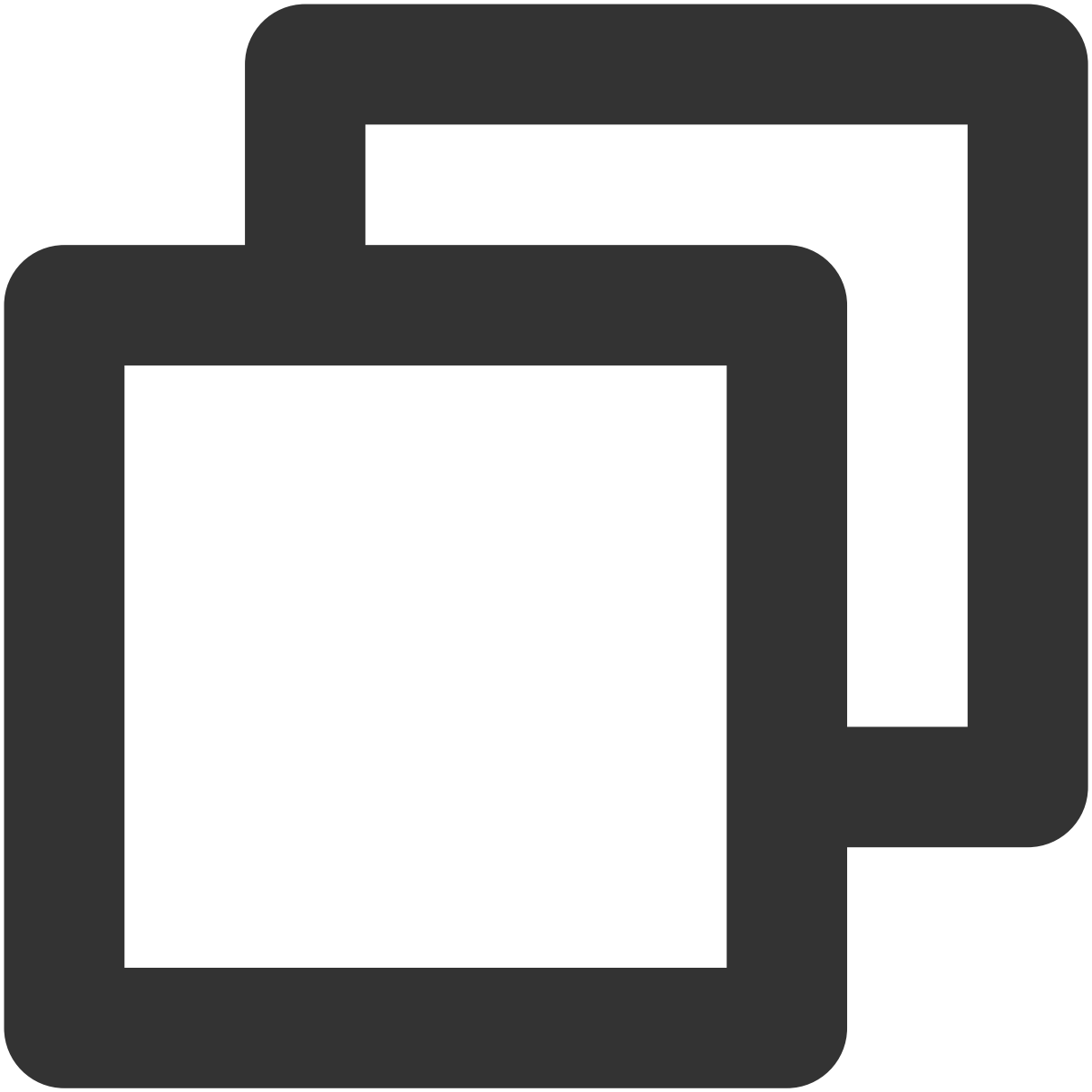
```
goosefs fs mount --option fs.cosn.userinfo.secretId={secretId} \\
    --option fs.cosn.userinfo.secretKey={secretKey} \\
    --option fs.cosn.bucket.region=ap-beijing \\
    --option fs.cosn.impl=org.apache.hadoop.fs.CosFileSystem \\
    --option fs.AbstractFileSystem.cosn.impl=org.apache.hadoop.fs.CosN \\
    /cosn {cos桶}
```

4. 查看目录：



```
[goosefs@VM-0-7-centos goosefs-1.0.0-SNAPSHOT-noUI-noHelm]$ goosefs fs ls /
drwxrwxrwx  goosefs        goosefs                       1       PERSISTED 01-01-197
drwxr-xr-x  root           root                          0       PERSISTED 06-25-202
```

5. 查看 worker 节点：



```
[goosefs@VM-0-7-centos goosefs-1.0.0-SNAPSHOT-noUI-noHelm]$ goosefs fsadmin report
Capacity information for all workers:
   Total Capacity: 1024.00MB
       Tier: MEM  Size: 1024.00MB
   Used Capacity: 0B
       Tier: MEM  Size: 0B
   Used Percentage: 0%
   Free Percentage: 100%

Worker Name      Last Heartbeat   Storage       MEM
```

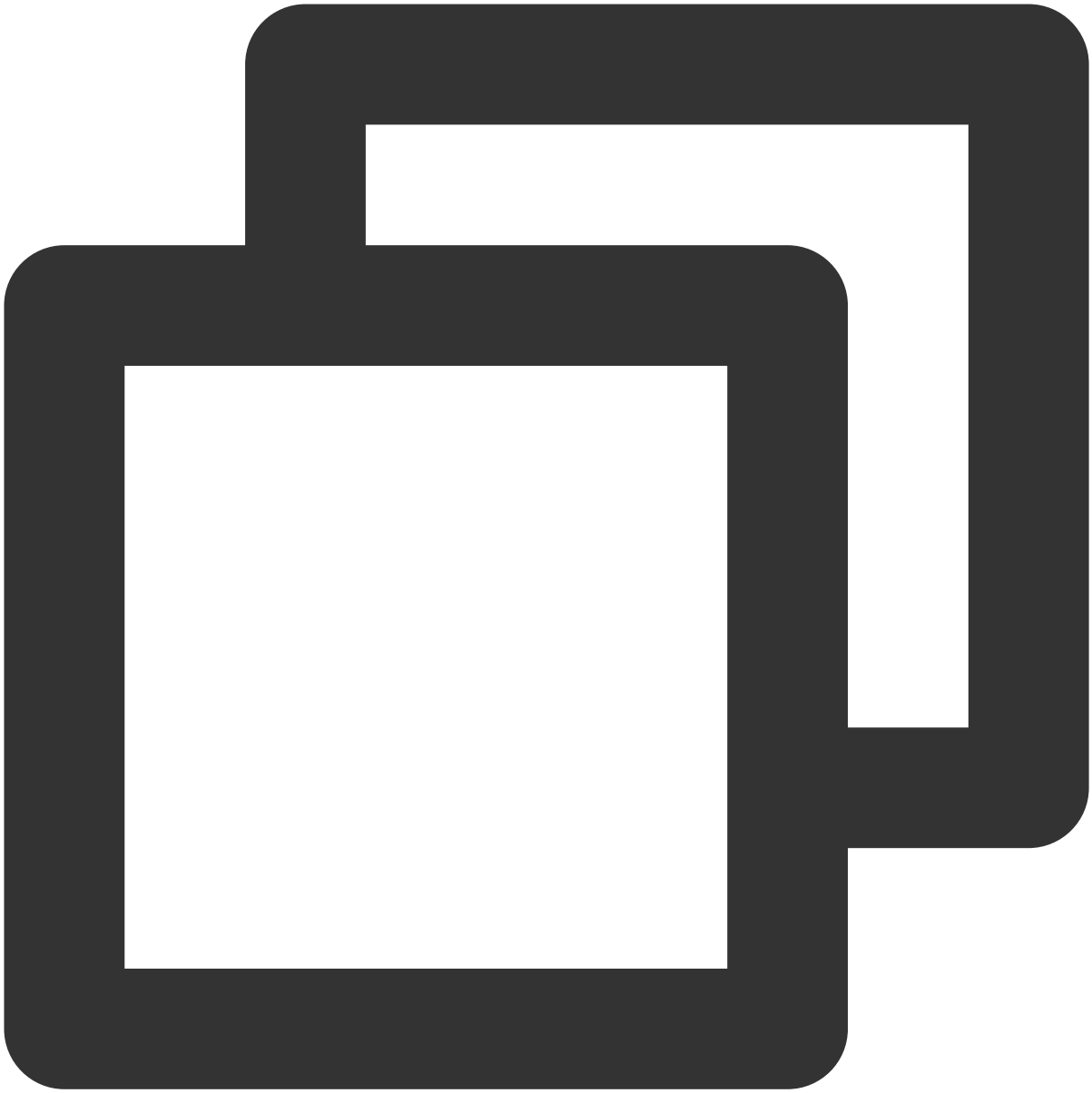
```
  172.31.0.7       0                capacity       1024.00MB
                                    used           0B (0%)
```

运维指南
日志指引
GooseFS 日志介绍

最近更新时间：2024-01-06 11:17:59

GooseFS 的 Master 和 Worker 节点，以及 Spark 等计算框架通过 GooseFS Client 请求 GooseFS 时，都会记录请求日志，用户可对输出日志进行分析，进行问题排查。GooseFS 日志输出基于 log4j 实现，可以通过修改 log4j.properties 配置文件来调整 GooseFS 的日志输出，例如日志存储路径，日志级别，是否记录 RPC 调用情况等。用户可以到 GooseFS 的配置文件目录下，打开并修改 log4j.properties 文件：

版权所有：腾讯云计算（北京）有限责任公司



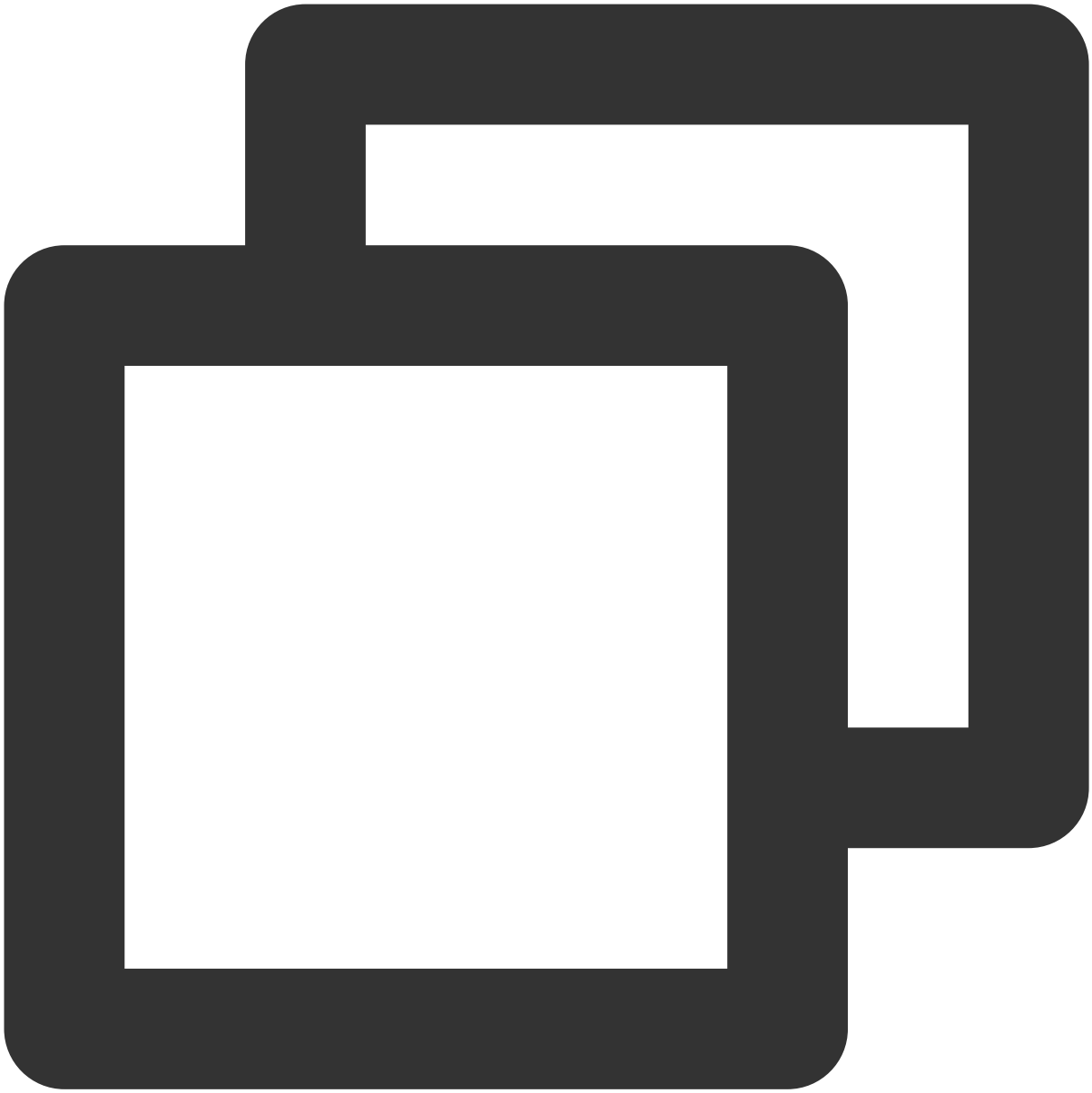
```
$ cd /usr/local/service/goosefs/conf
$ cat log4j.properties

# May get overridden by System Property

log4j.rootLogger=INFO, ${goosefs.logger.type}, ${goosefs.remote.logger.type}

log4j.category.goosefs.logserver=INFO, ${goosefs.logserver.logger.type}
log4j.additivity.goosefs.logserver=false

log4j.logger.AUDIT_LOG=INFO, ${goosefs.master.audit.logger.type}
```

```
log4j.additivity.AUDIT_LOG=false

...
```

下文将详细介绍 GooseFS 的日志配置：

日志存储位置

GooseFS 采集的日志默认存储在 ${GOOSEFS_HOME}/logs 目录下。其中，Master 采集的日志存储在 logs/master.log 中，Worker 采集的日志存储在 logs/worker.log 中。需要注意的是， 节点进程异常抛出的日志会记录在 master.out 或者 worker.out 中，正常情况下这两类文件均为空文件，系统有异常时会记录异常信息以便追溯。
以 Master 节点的日志存储配置为例，以下为常用的几项配置：



```
# Appender for Master
log4j.appender.MASTER_LOGGER=org.apache.log4j.RollingFileAppender
log4j.appender.MASTER_LOGGER.File=${goosefs.logs.dir}/master.log
log4j.appender.MASTER_LOGGER.MaxFileSize=10MB
log4j.appender.MASTER_LOGGER.MaxBackupIndex=100
log4j.appender.MASTER_LOGGER.layout=org.apache.log4j.PatternLayout
log4j.appender.MASTER_LOGGER.layout.ConversionPattern=%d{ISO8601} %-5p %c{1} - %m%n
```

配置参数介绍如下：

MASTER_LOGGER：指定配置 MASTER 的日志输出。

MASTER_LOGGER.File：指定日志存储路径，可以通过修改路径来自定义日志存储位置。
MASTER_LOGGER.MaxFileSize：指定单个日志文件大小的上限。
MASTER_LOGGER.MaxBackupIndex：指定日志文件数上限。
MASTER_LOGGER.layout：指定日志输出格式模板。
MASTER_LOGGER.layout.ConversionPattern：指定日志输出的具体格式。

注意

.log 文件是滚动存储的，您可以将其备份到 UFS，例如对象存储中；.out 文件不滚动存储，如果需要清理 .out 文件需要手动发起删除操作。

更多 log4j 的参数配置可以参考 log4j configuration 文档。

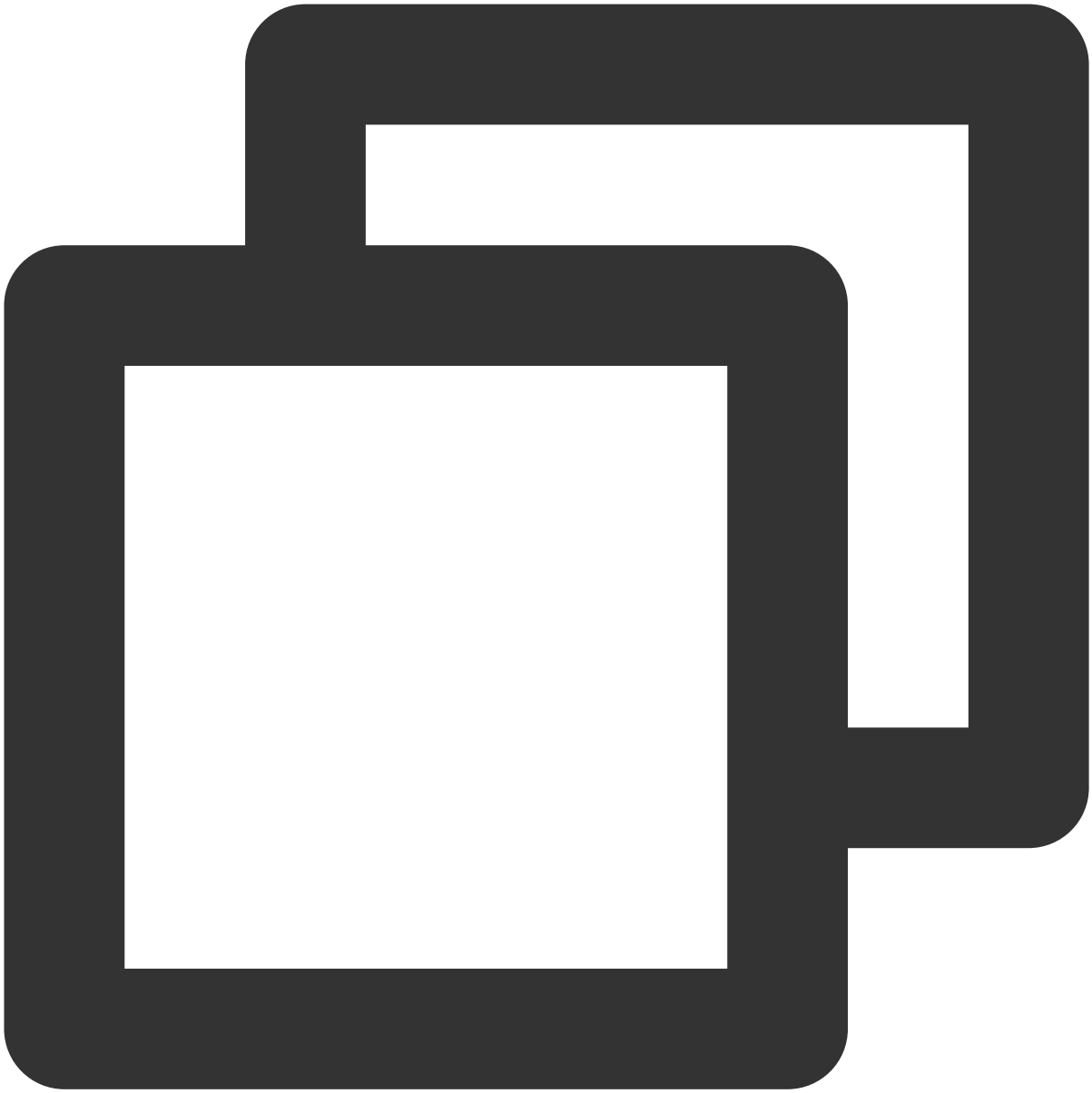
GooseFS 仅存储本身产生的日志，上层计算应用产生的日志可以根据计算应用的日志配置，查看日志存储位置。常见计算应用的日志配置信息可见：Apache Hadoop, Apache HBase, Apache Hive, Apache Spark。

日志级别

GooseFS 提供了以下5种级别的日志：
TRACE: 详细的调用日志，适用于调试方法和类的调用。
DEBUG: 较详细的调用日志，适用于 DEBUG 过程中排查问题。
INFO: 请求处理过程中的关键信息。
WARN: 警告类信息，任务可以继续执行，但需要注意可能存在问题。
ERROR: 系统报错信息，影响任务进行。

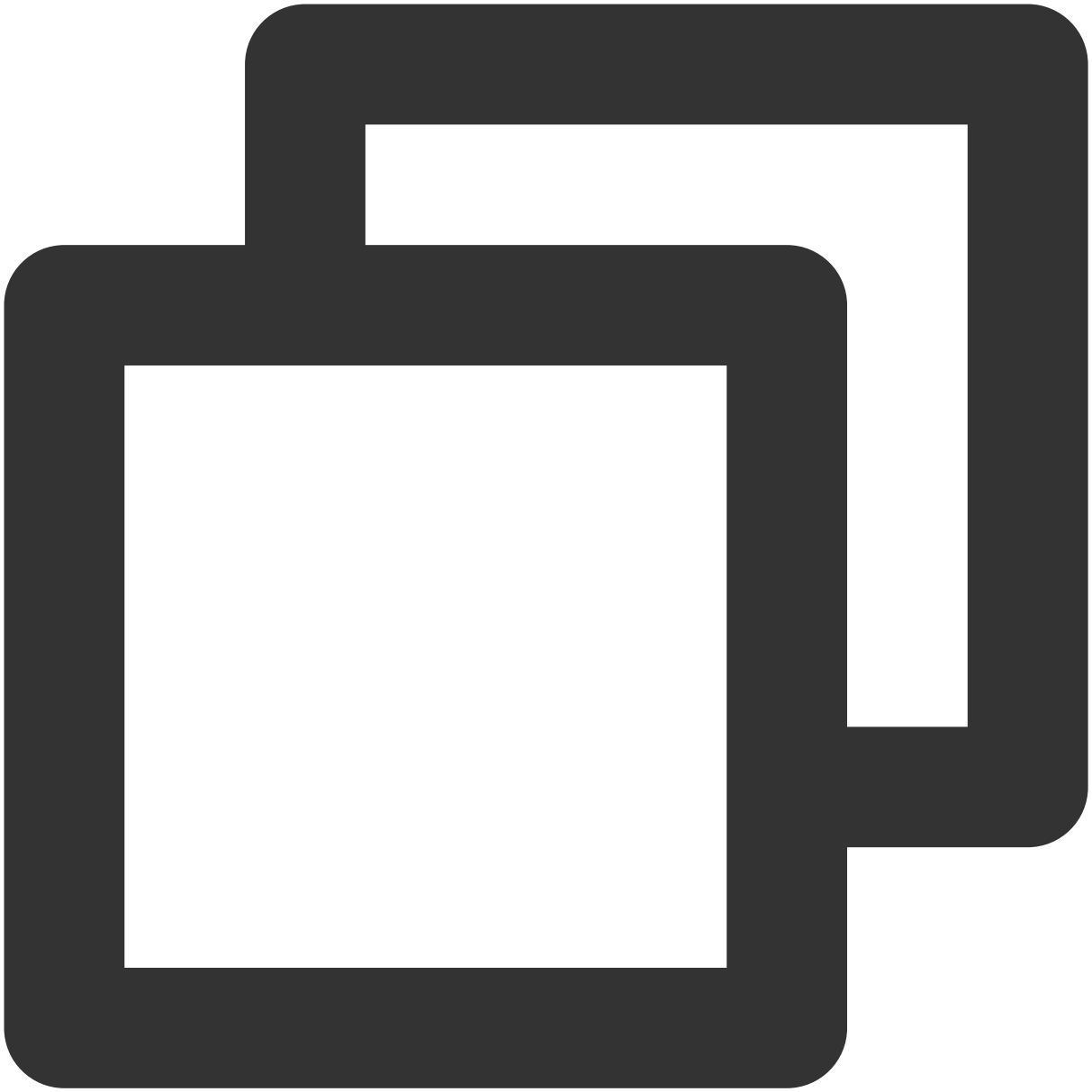
上述5种级别日志的详细程度从高到低，配置高等级的日志级别会一并记录低等级的日志信息。默认情况下 GooseFS 配置 INFO 级别的日志，记录 INFO、WARN、ERROR三个级别的日志。

可以到 GooseFS 的配置文件目录下打开并修改 log4j.properties 文件，如下示例展示了将所有 GooseFS 的日志级别修改为 DEBUG 级别：



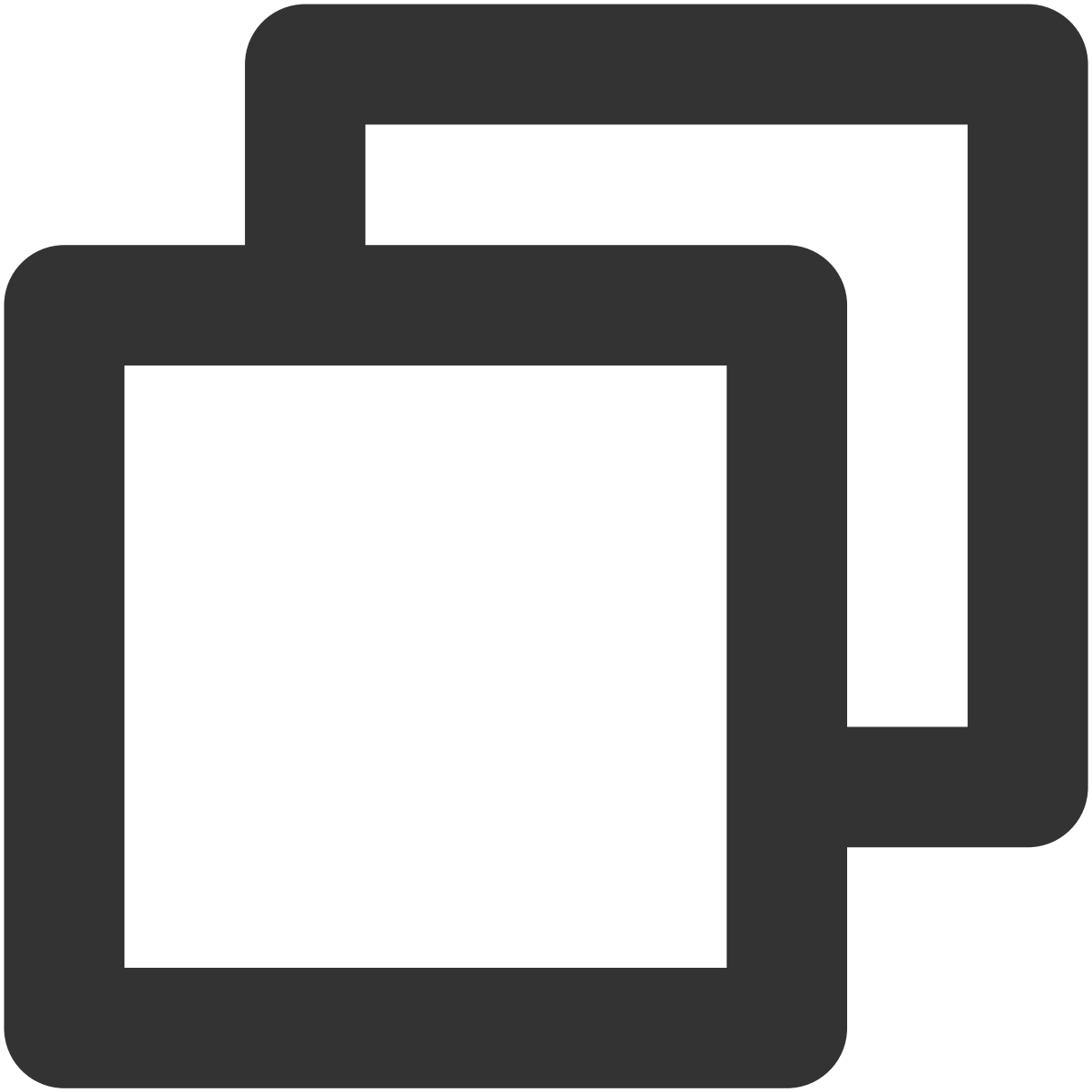
```
log4j.rootLogger=DEBUG, ${goosefs.logger.type}, ${goosefs.remote.logger.type}
```

如果需要修改指定类的日志级别，可以在配置文件中添加声明，如下示例展示指定 GooseFSFileInStream 这个类的日志级别为 DEBUG：



```
log4j.logger.com.qcloud.cos.goosefs.client.file.GooseFSFileInStream=DEBUG
```

一般而言，推荐在日志配置文件中修改日志级别。但在一些特定的场景下，用户可能需要在集群运行过程中修改日志参数，此时可以通过在命令行中输入 goosefs logLevel 指令进行调整。如下为 logLevel 支持的配置选项：



```
usage: logLevel [--level <arg>] --logName <arg> [--target <arg>]
    --level <arg>     The log level to be set.
    --logName <arg>   The logger's name(e.g. com.qcloud.cos.goosefs.master.file.Def
    --target <arg>    <master|workers|host:webPort>. A list of targets separated by
```

各项配置的详细说明如下：

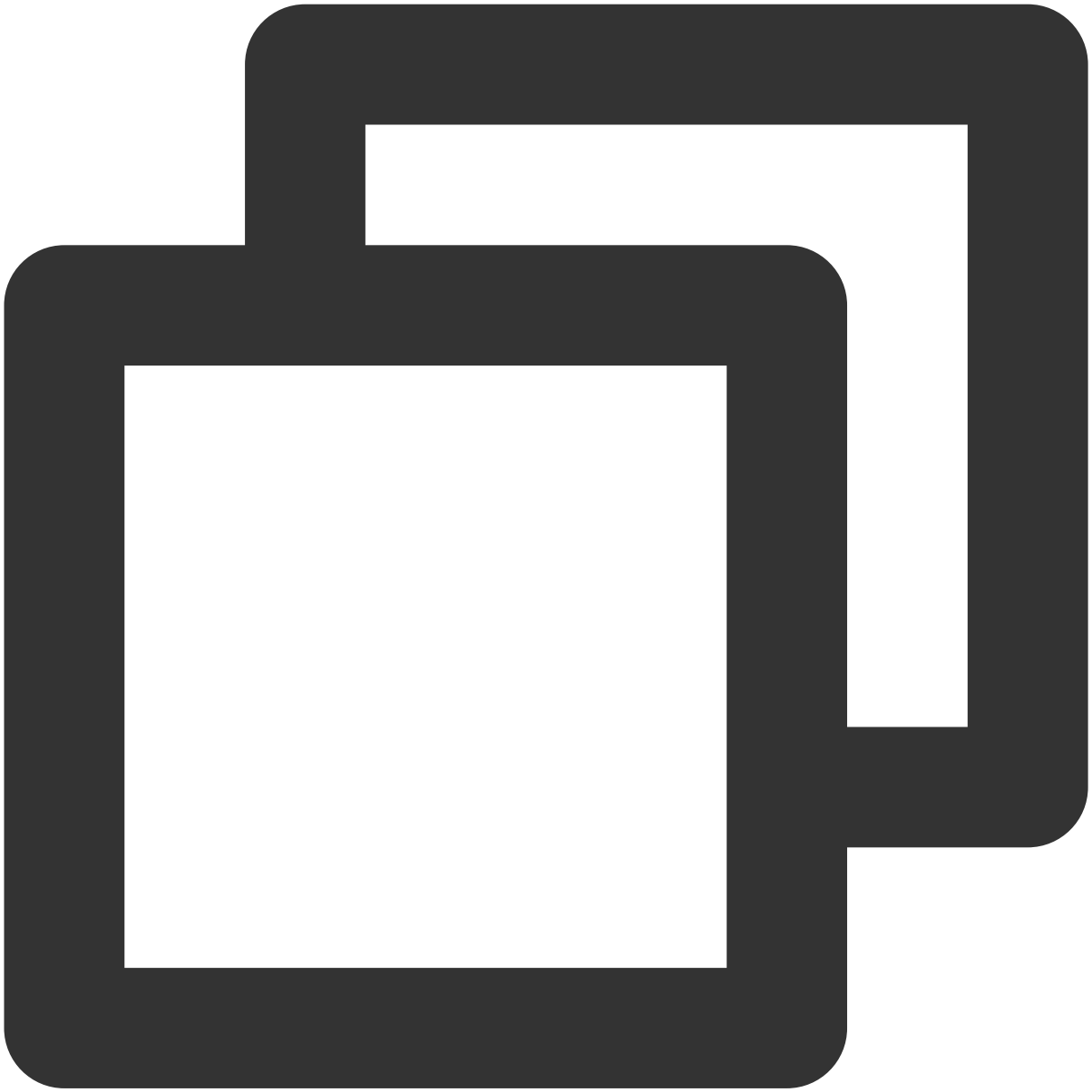
level：日志级别，支持 TRACE、DEBUG、INFO、WARN、ERROR 五种级别。

logName：日志输出 logger，如 com.qcloud.cos.goosefs.underfs.hdfs.HdfsUnderFileSystem 等。

target：修改范围，可以设置为指定的 Master 或者 Worker 节点（通过 IP：PORT 方式指定），默认为 Master 和所有 Worker 节点。

版权所有：腾讯云计算（北京）有限责任公司

用户可以按需在系统运行过程中调整日志级别，以便排查系统运行过程中的问题。如以下示例展示了在运行过程中将所有 Worker 节点上 com.qcloud.cos.goosefs.underfs.hdfs.HdfsUnderFileSystem 这个类的日志级别调整为 DEBUG 级别，并在调试完成后调整回 INFO 级别：



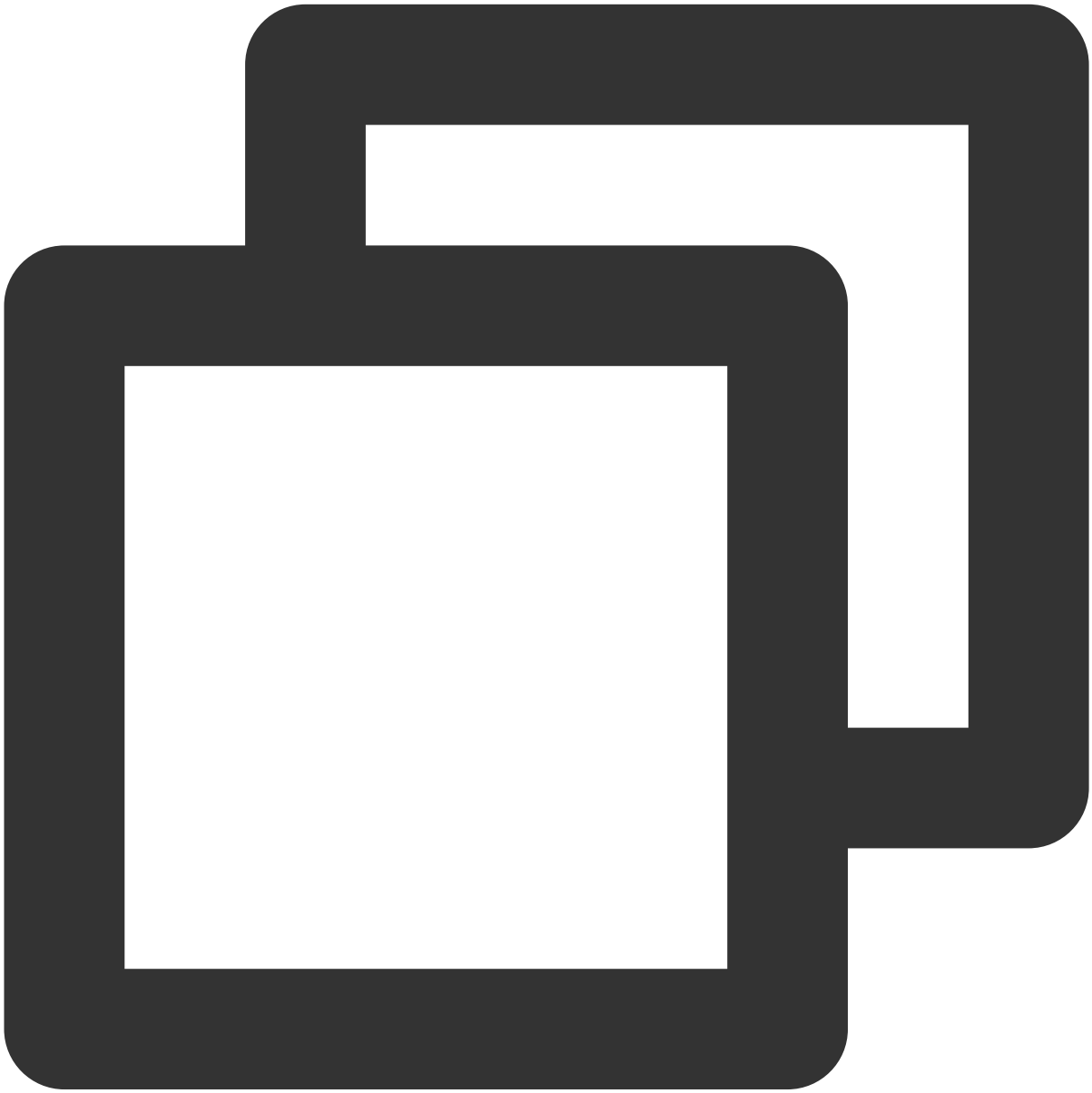
```
$  goosefs logLevel --logName=com.qcloud.cos.goosefs.underfs.hdfs.HdfsUnderFileSyst

$  goosefs logLevel --logName=com.qcloud.cos.goosefs.underfs.hdfs.HdfsUnderFileSyst
```

高级配置

GooseFS 支持配置 GC 事件日志，FUSE 接口日志，RPC 调用日志，UFS 操作日志，以及进行日志分割和日志筛选等操作。以下介绍部分常用高级配置的使用方式。

GC 事件日志

GooseFS 将 GC 事件日志记录在 .out 文件中， 可以通过在 conf/goosefs-env.sh 中添加如下配置：

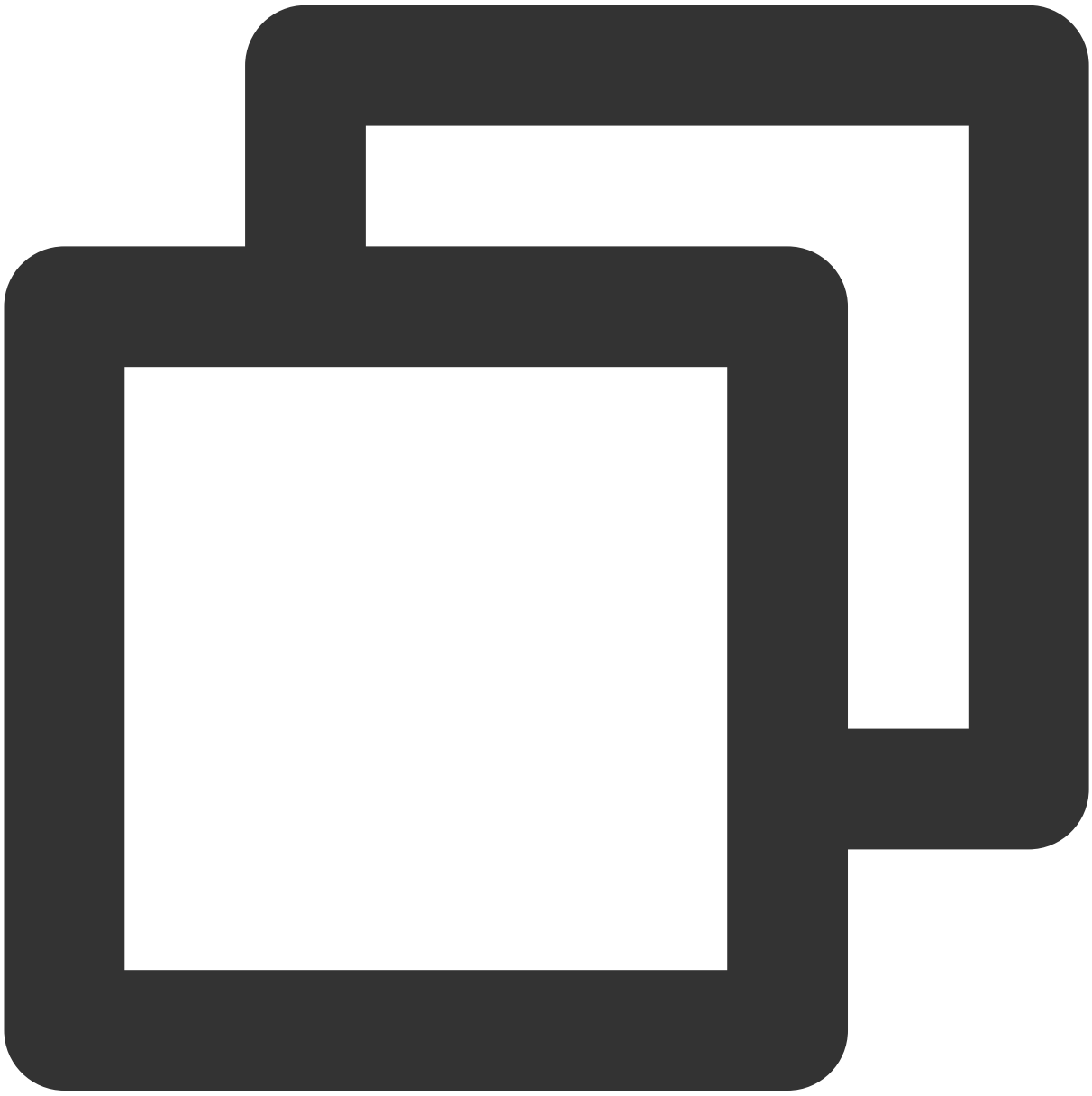


```
GOOSEFS_JAVA_OPTS+=" -XX:+PrintGCDetails -XX:+PrintTenuringDistribution -XX:+PrintG
```

GOOSEFS_JAVA_OPTS 为所有类型 GooseFS 节点的 Java 虚拟机参数，也可以通过指定 GOOSEFS_MASTER_JAVA_OPTS 和 GOOSEFS_WORKER_JAVA_OPTS 分别指定 Master 和 Worker上的虚拟机参数。

FUSE 接口日志

可以在 conf/log4j.properties 文件中配置记录 FUSE 日志级别：



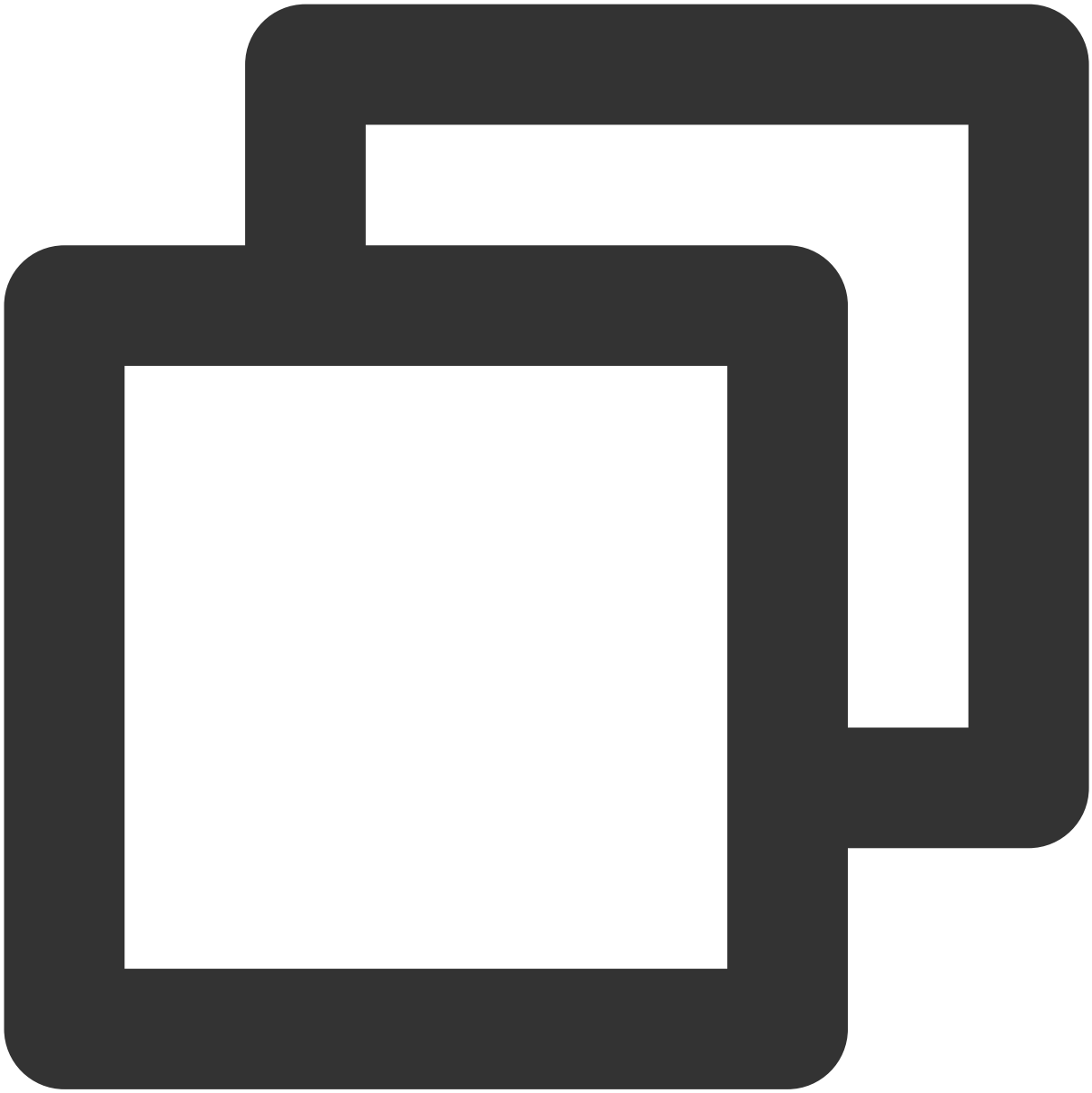
```
goosefs.logger.com.qcloud.cos.goosefs.fuse.GoosefsFuseFileSystem=DEBUG
```

启用后 FUSE 接口日志可以在 logs/fuse.log 查看。

启用 RPC 调用日志

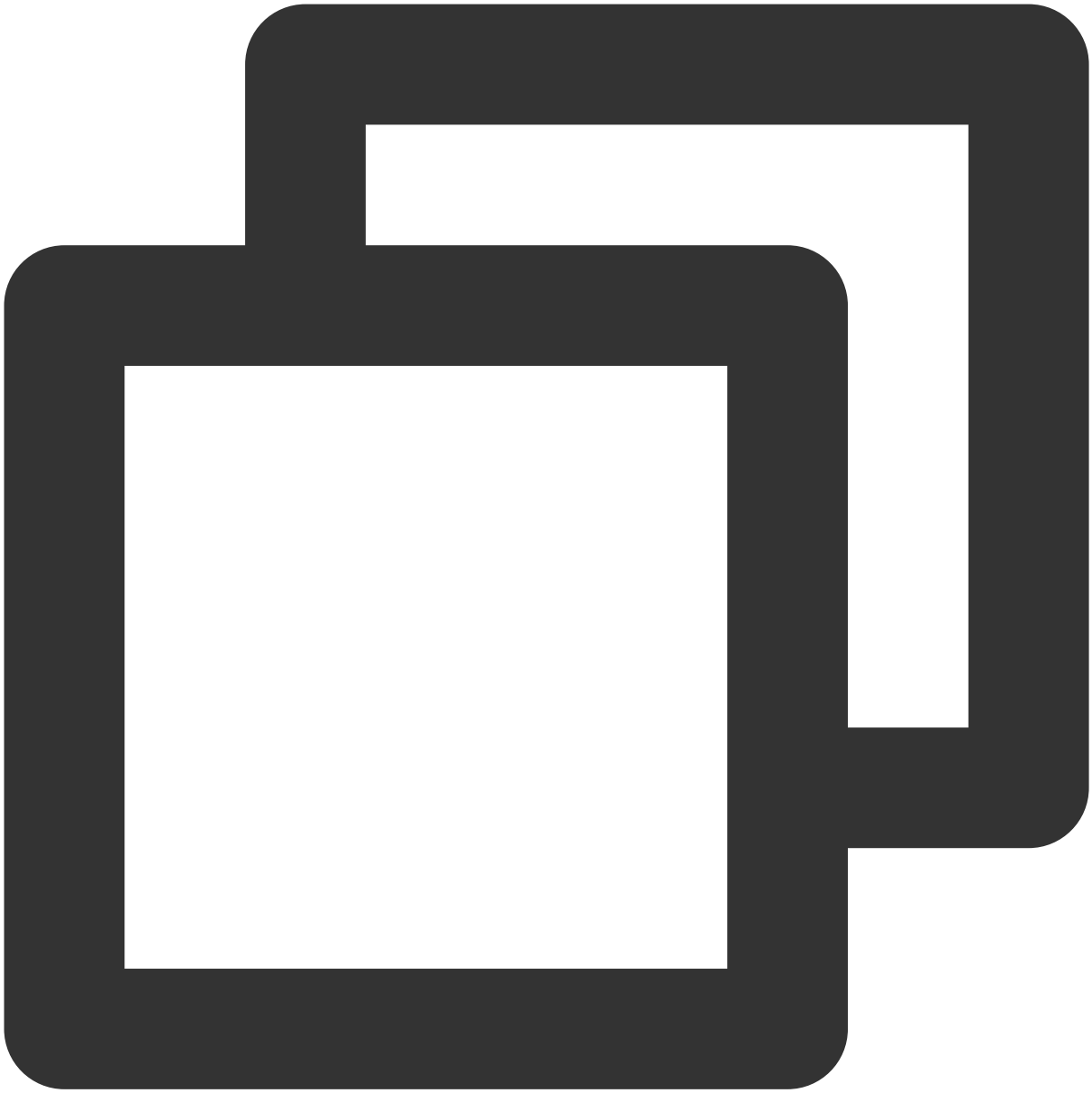
GooseFS 支持在 conf/log4j.properties 配置文件中，配置 Client 端或 Master 端的 RPC 调用日志。

可以通过在 log4j.properties 文件中，配置客户端输出 RPC 请求日志：



```
log4j.logger.com.qcloud.cos.goosefs.client.file.FileSystemMasterClient=DEBUG # Clie
log34j.logger.com.qcloud.cos.goosefs.client.block.BlockSystemMasterClient=DEBUG # C
```

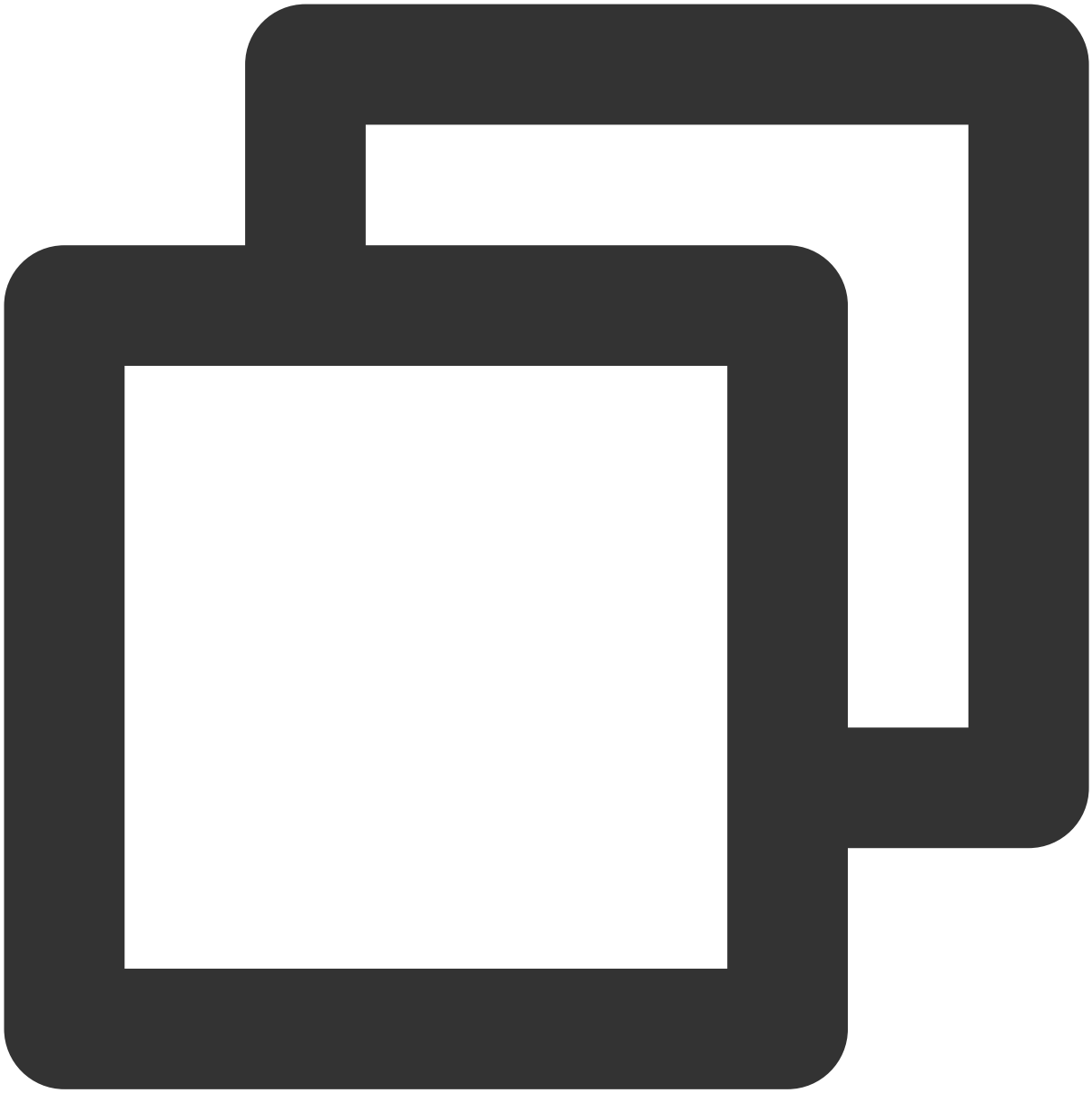
可以通过 logLevel 指令来，配置 Master 输出 RPC 请求日志：



```
$ goosefs logLevel \\--logName=com.qcloud.cos.goosefs.master.file.FileSystemMasterC
$ goosefs logLevel \\--logName=com.qcloud.cos.goosefs.master.block.BlockSystemMaste
```

UFS 操作日志

UFS 操作日志输出配置，可以通过在 log4j.properties 文件中添加配置项，或者通过 logLevel 指令进行操作。下面以 logLevel 指令为例：



```
$ goosefs logLevel \\--logName=com.qcloud.cos.goosefs.underfs.UnderFileSystemWithLo
$ goosefs logLevel \\--logName=com.qcloud.cos.goosefs.underfs.UnderFileSystemWithLo
```

分割日志

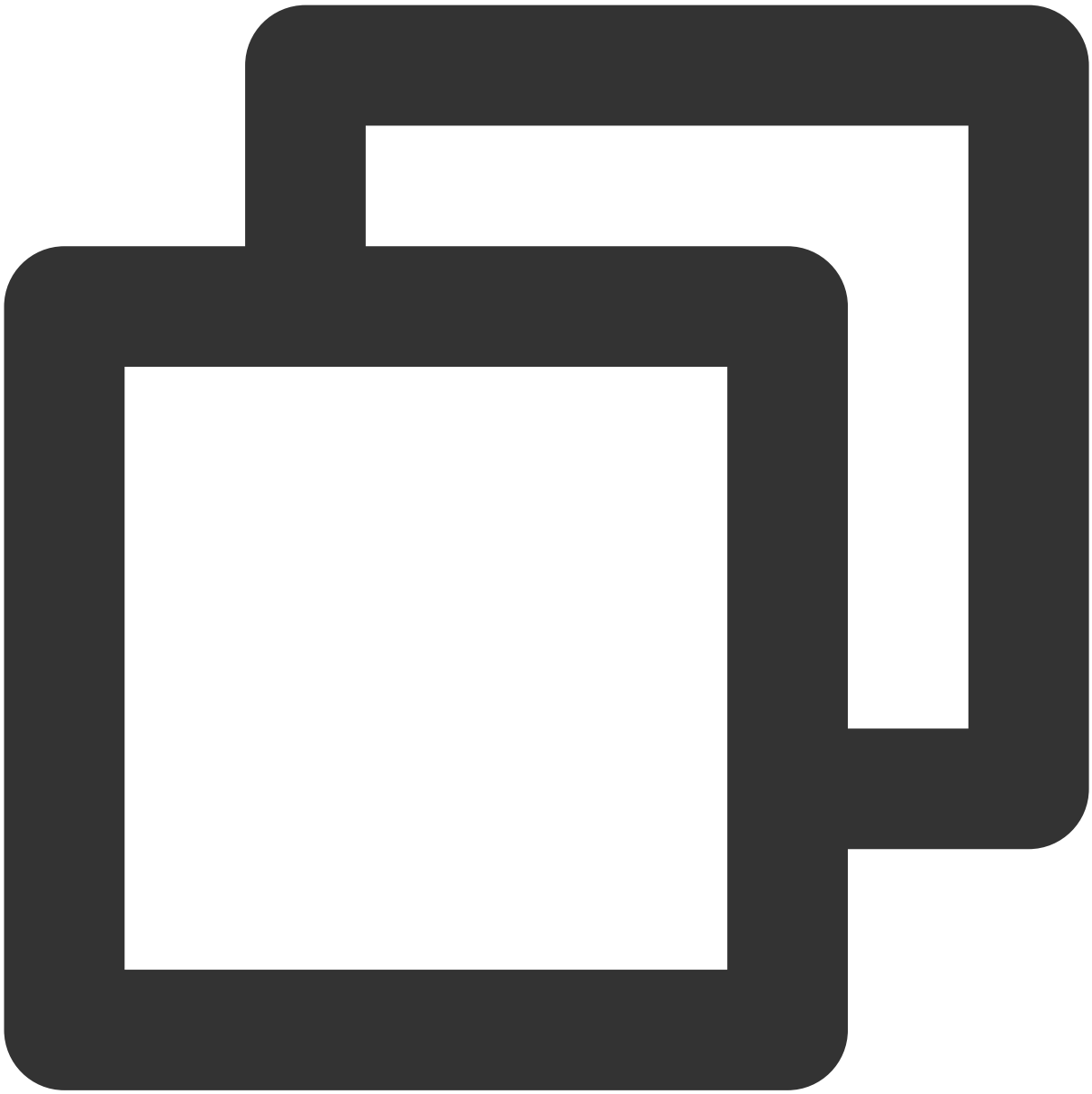
GooseFS 支持将指定类型日志存储在指定存储路径，如果将所有日志都记录在 .log 文件，可能产生以下问题：

当集群规模大，吞吐量较多时，master.log 或者 worker.log 文件可能变得异常庞大，或者滚动产生大量日志文件。

日志信息较多，不利于针对性分析的进行日志分析。

大量日志存储在本地节点，占用空间。

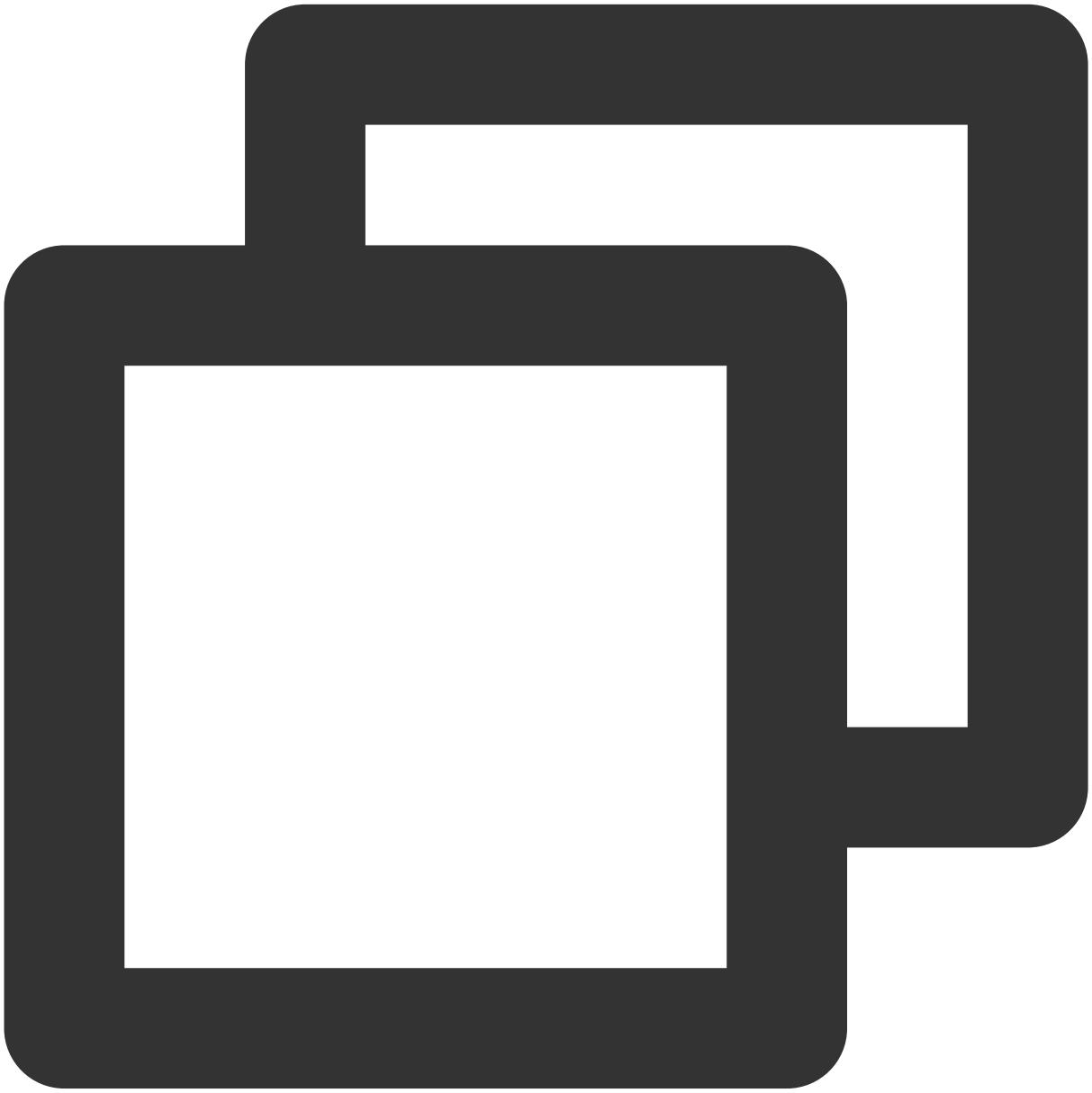
因此业务可以根据需要在 log4j.properties 文件中添加配置项，将指定类产生的日志存储至指定文件路径，如下示例展示了将 StateLockManager 类的日志存储在 statelock.log 中：



```
log4j.category.com.qcloud.cos.goosefs.master.StateLockManager=DEBUG, State_LOCK_LOG
log4j.additivity.com.qcloud.cos.goosefs.master.StateLockManager=false
log4j.appender.State_LOCK_LOGGER=org.apache.log4j.RollingFileAppender
log4j.appender.State_LOCK_LOGGER.File=<GOOSEFS_HOME>/logs/statelock.log
log4j.appender.State_LOCK_LOGGER.MaxFileSize=10MB
log4j.appender.State_LOCK_LOGGER.MaxBackupIndex=100
log4j.appender.State_LOCK_LOGGER.layout=org.apache.log4j.PatternLayout
log4j.appender.State_LOCK_LOGGER.layout.ConversionPattern=%d{ISO8601} %-5p %c{1} -
```

筛选日志

GooseFS 支持按照一定的条件筛选并记录日志，而不是全量记录所有日志。例如在进行性能测试时需要记录 RPC 调用日志，但业务侧并不需要记录所有 RPC 调用日志，只需要记录某些延迟较高的日志即可，此时可以通过在 log4j.properties 文件中添加配置项添加日志筛选条件即可，如下示例分别展示了筛选 RPC 调用延迟超过200ms的请求和 FUSE 调用超过1s的请求：



```
goosefs.user.logging.threshold=200ms
goosefs.fuse.logging.threshold=1s
```

监控指南
获取 GooseFS 监控指标

最近更新时间：2024-01-06 11:17:58

Goosefs 基于 Coda Hale Metrics Library 库记录监控数据，支持通过命令行、控制台、文件等多种途径获取指标，目前支持的指标获取方式包括：
MetricsServlet：将监控指标以 Json 格式提供给用户。
CsvSink：通过 CSV 文件方式展示监控指标，配置后会周期性地生成记录监控指标的 CSV 文件。
PrometheusMetricsServlet:将监控指标以 Prometheus 定义的格式提供给用户。
上述监控指标的配置可以通过配置文件来指定。GooseFS 监控指标的配置文件默认文件路径为 `$GooseFS_HOME/conf/metrics.properties`，支持通过 goosefs.metrics.conf.file 指定自定义监控配置文件。GooseFS 为用户提供了一个默认模板 metrics.properties.template，包含了所有可配置的属性。

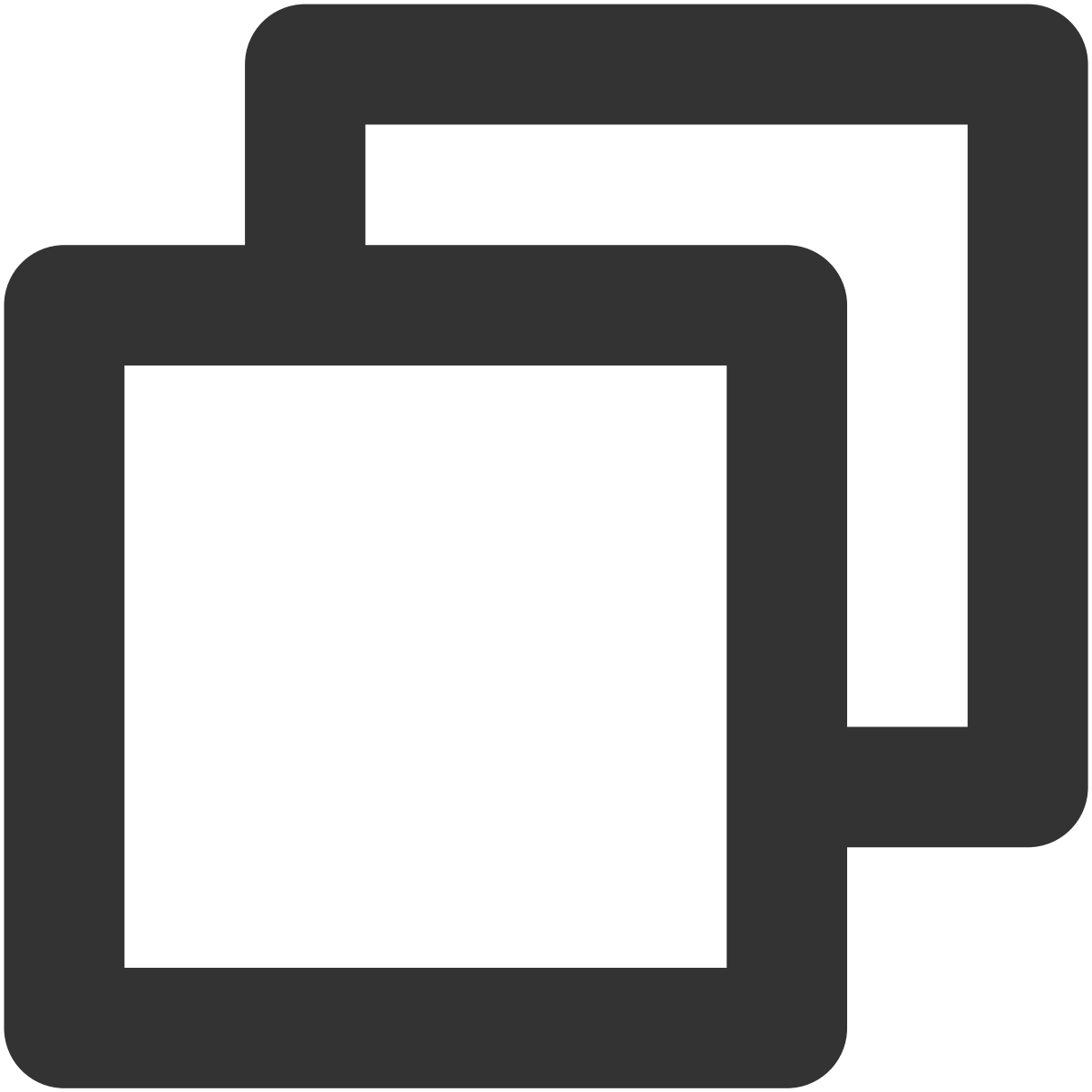
获取监控指标

以下介绍三种基础的获取监控指标的途径：

1. 通过 JSON 格式拉取监控指标

GooseFS 默认的获取监控指标的方式是通过 JSON 格式拉取，对应着 MetricsServlet 这一配置项。可以在命令行中向 GooseFS 的 Leading Master 节点发起一个 HTTP 请求，拉取所需的监控指标，其中 master 的 metrics 端口为 9201，worker 的 metrics 端口为 9204。请求指令格式如下：

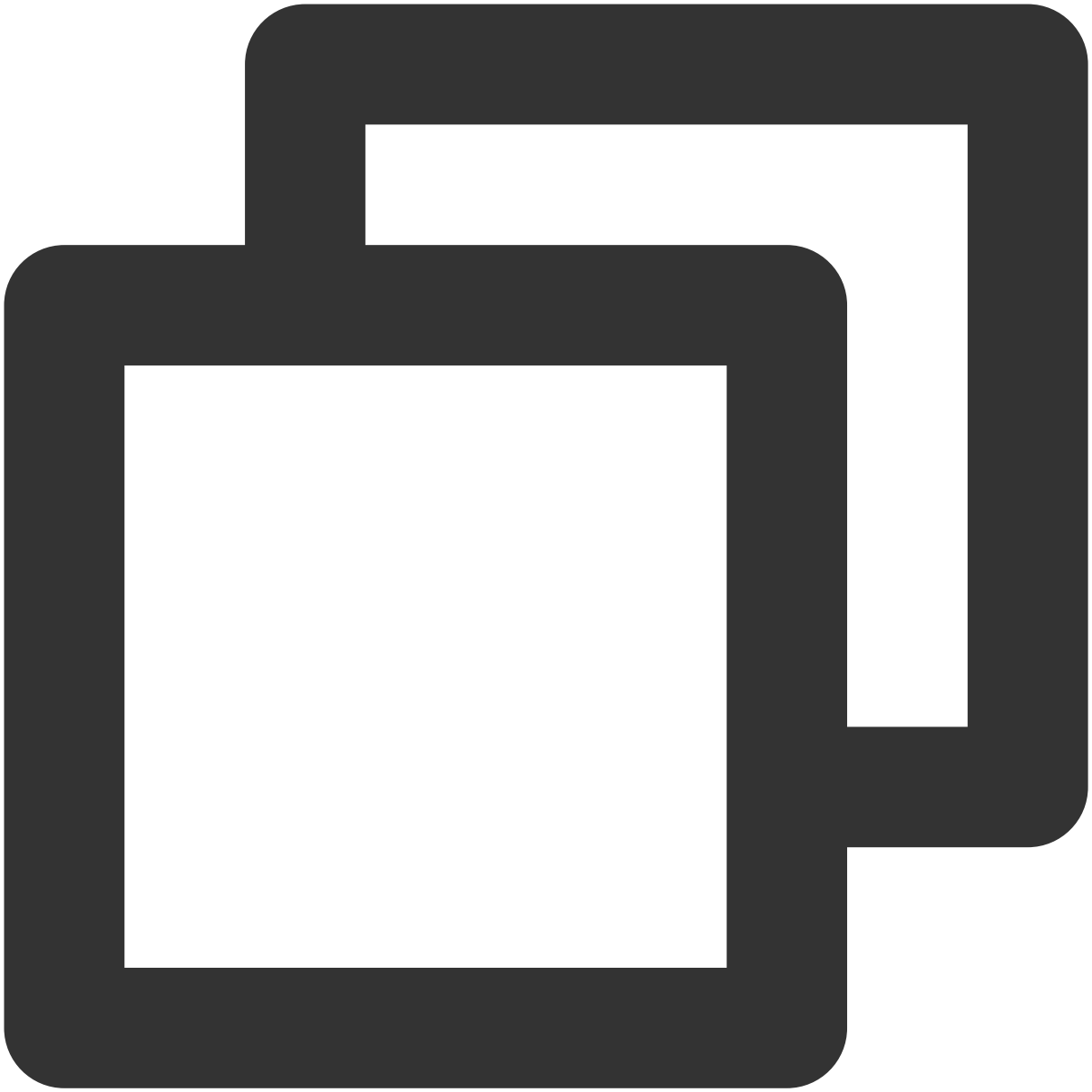
版权所有：腾讯云计算（北京）有限责任公司



```
$ curl <LEADING_MASTER_HOSTNAME>:<MASTER_WEB_PORT>/metrics/json
```

如上示例中，需要是合法的 MASTER 节点的IP， 需要为已经启用的端口。

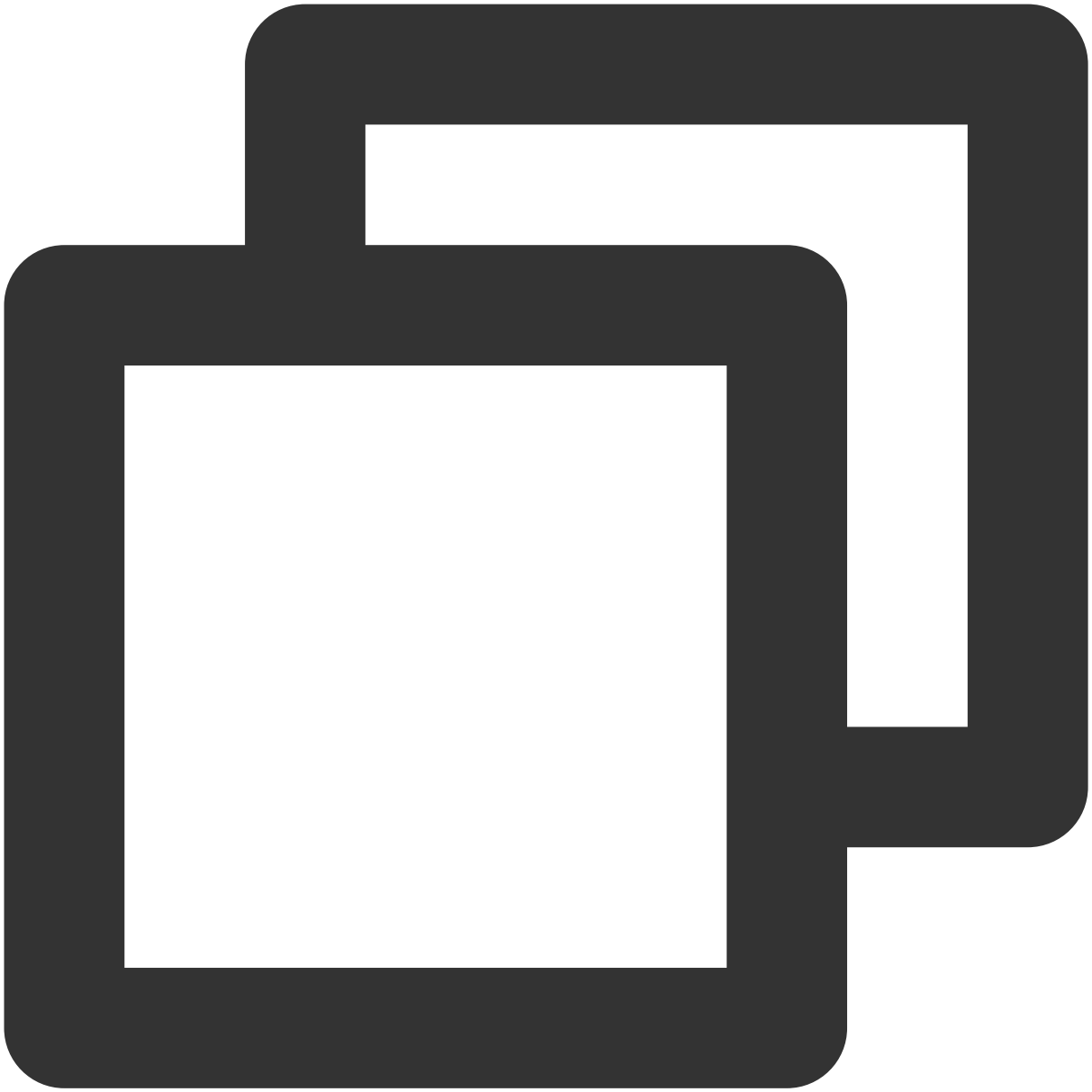
如果需要获取某个 WORKER 节点的监控指标，可以通过如下方式获取：



```
$ curl <WORKER_HOSTNAME>:<WORKER_WEB_PORT>/metrics/json
```

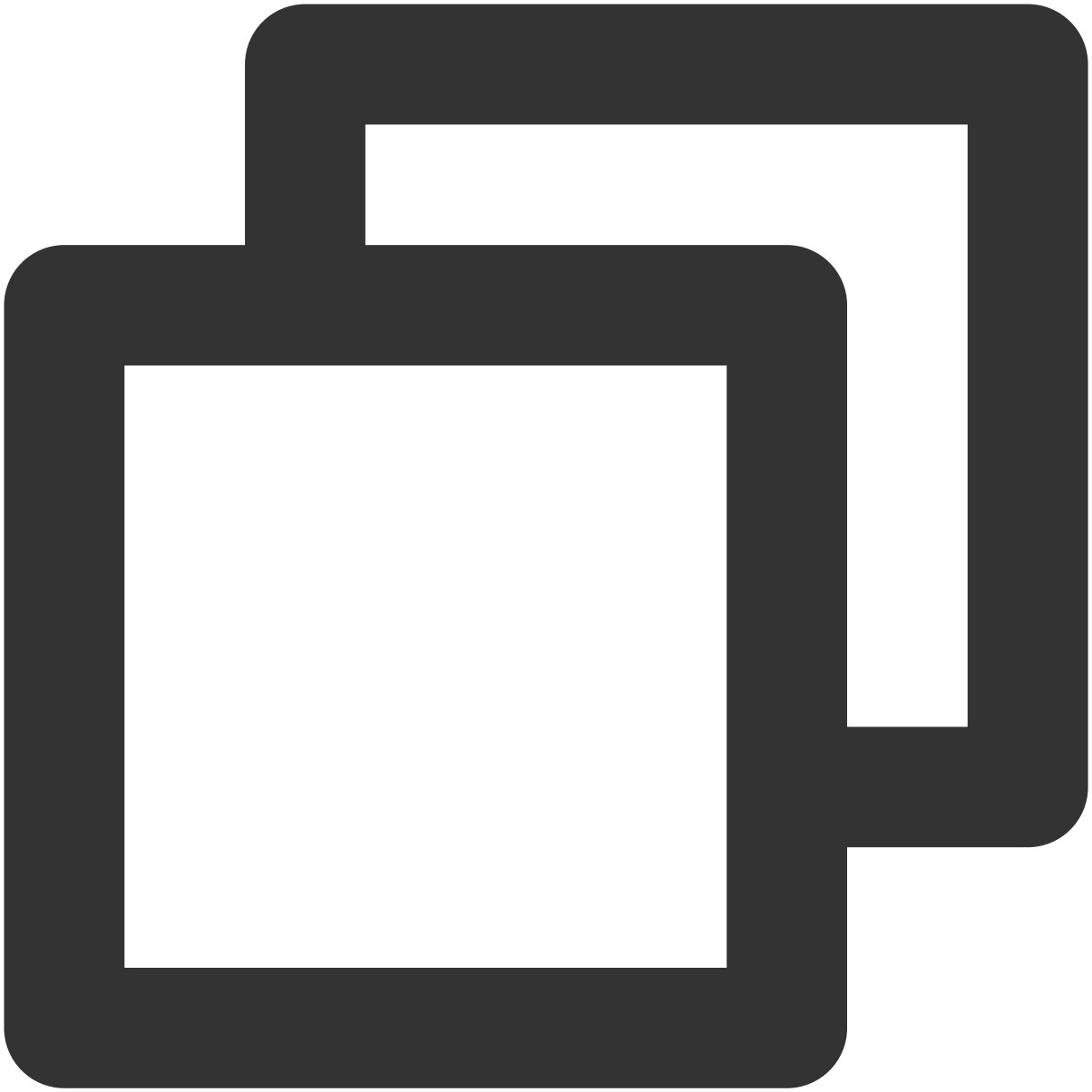
2. 通过 CSV 文件获取监控指标

GooseFS 支持将数据导出为 CSV 格式文件，通过该能力获取监控指标，首先需要准备一个存储监控指标的目录：



```
$ mkdir /tmp/goosefs-metrics
```

准备好存储路径后，修改配置文件 conf/metrics.properties ，启用 CsvSink 能力：



```
sink.csv.class=goosefs.metrics.sink.CsvSink # 启用CsvSink能力

sink.csv.period=1 # 设置监控指标导出周期
sink.csv.unit=senconds # 设置监控指标导出周期的单位

sink.csv.directory=/tmp/goosefs-metrics # 设置监控指标导出路径
```

配置好后需要重启节点以便配置生效。配置生效后，监控指标将周期性地导出成 CSV 格式并存储在指定路径下。

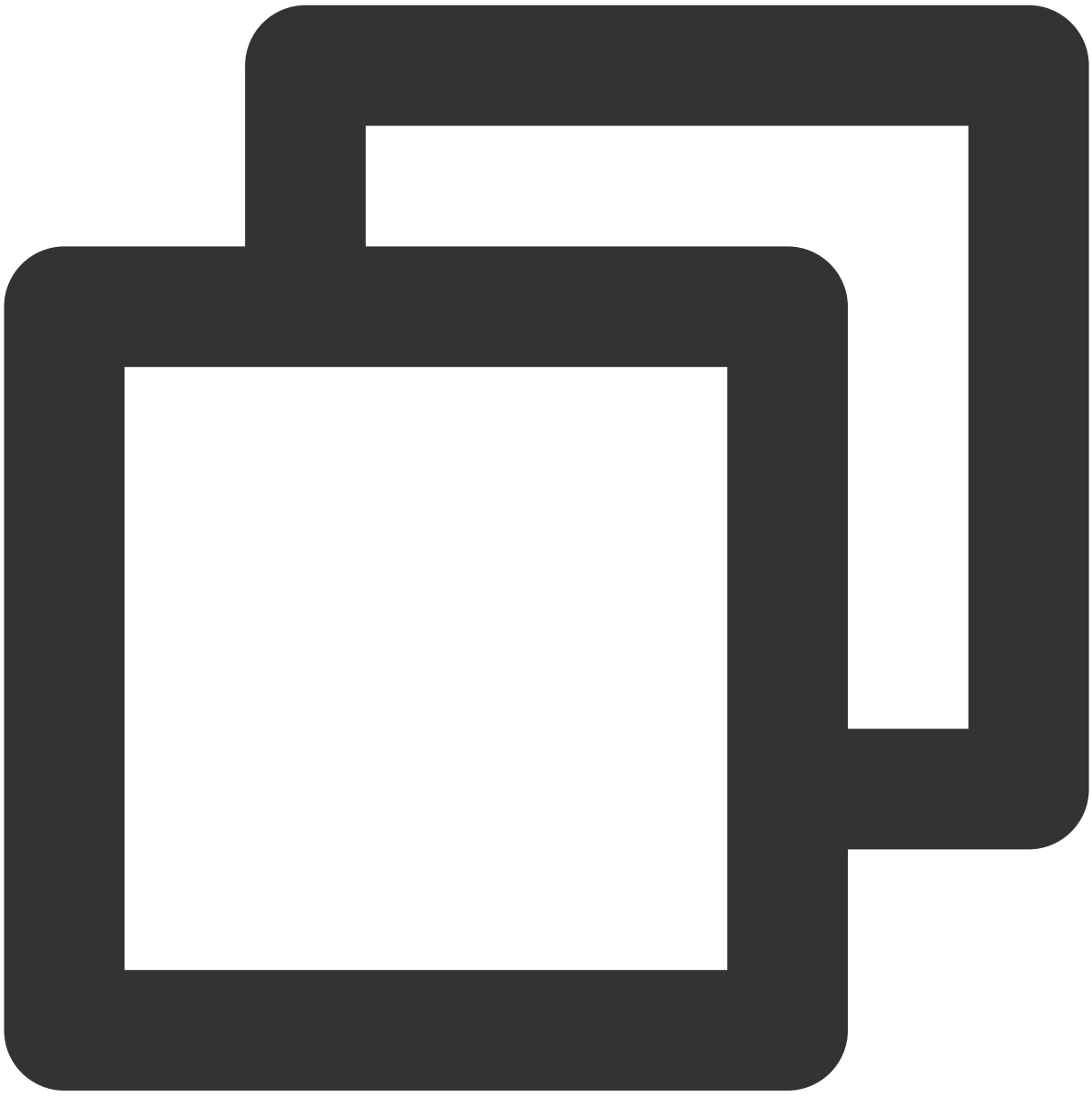
注意

GooseFS 准备了监控配置模板，可以参考 conf/metrics.properties.template文件；

如果 GooseFS 是集群化部署，需要保证指定的指标存储路径能被所有节点读取。

3. 拉取 Prometheus 监控指标

GooseFS master 和 worker 的 Prometheus 的监控指标可用如下的命令查看，其中 master 的 metrics 端口为9201，worker 的 metrics 端口为 9204：



```
curl <LEADING_MASTER_HOSTNAME>:<MASTER_WEB_PORT>/metrics/prometheus/
# HELP Master_CreateFileOps Generated from Dropwizard metric import (metric=Master.
...
```

```
curl <WORKER_IP>:<WOKER_PORT>/metrics/prometheus/
# HELP pools_Code_Cache_max Generated from Dropwizard metric import (metric=pools.C
...
```

基于 Prometheus 搭建 GooseFS 监控体系

最近更新时间：2024-01-06 11:17:58

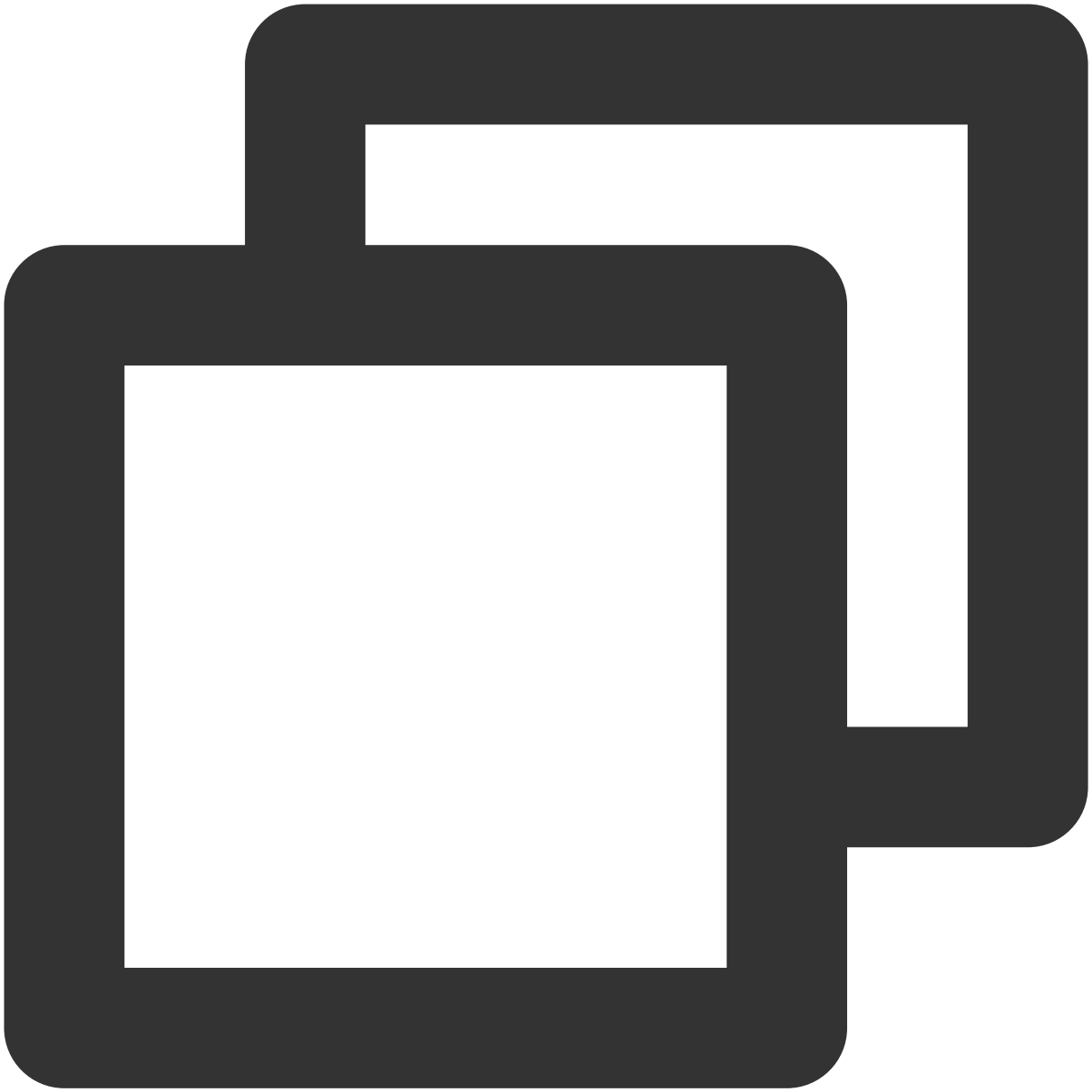
Goosefs 可以通过配置将指标数据输出到不同的监控系统中，Prometheus 是其中之一。Prometheus 是一个开源的监控框架，目前腾讯云可观测平台已集成了 Prometheus，下文将重点介绍 Goosefs 监控指标，以及将监控指标上报到自建的 Prometheus 和云上 Prometheus 的流程。

准备工作

通过 Prometheus 构建监控体系需要先做如下准备工作：
配置 GooseFS 集群
下载 Prometheus 官方安装包或腾讯云 Prometheus 安装包
下载和配置 Grafana

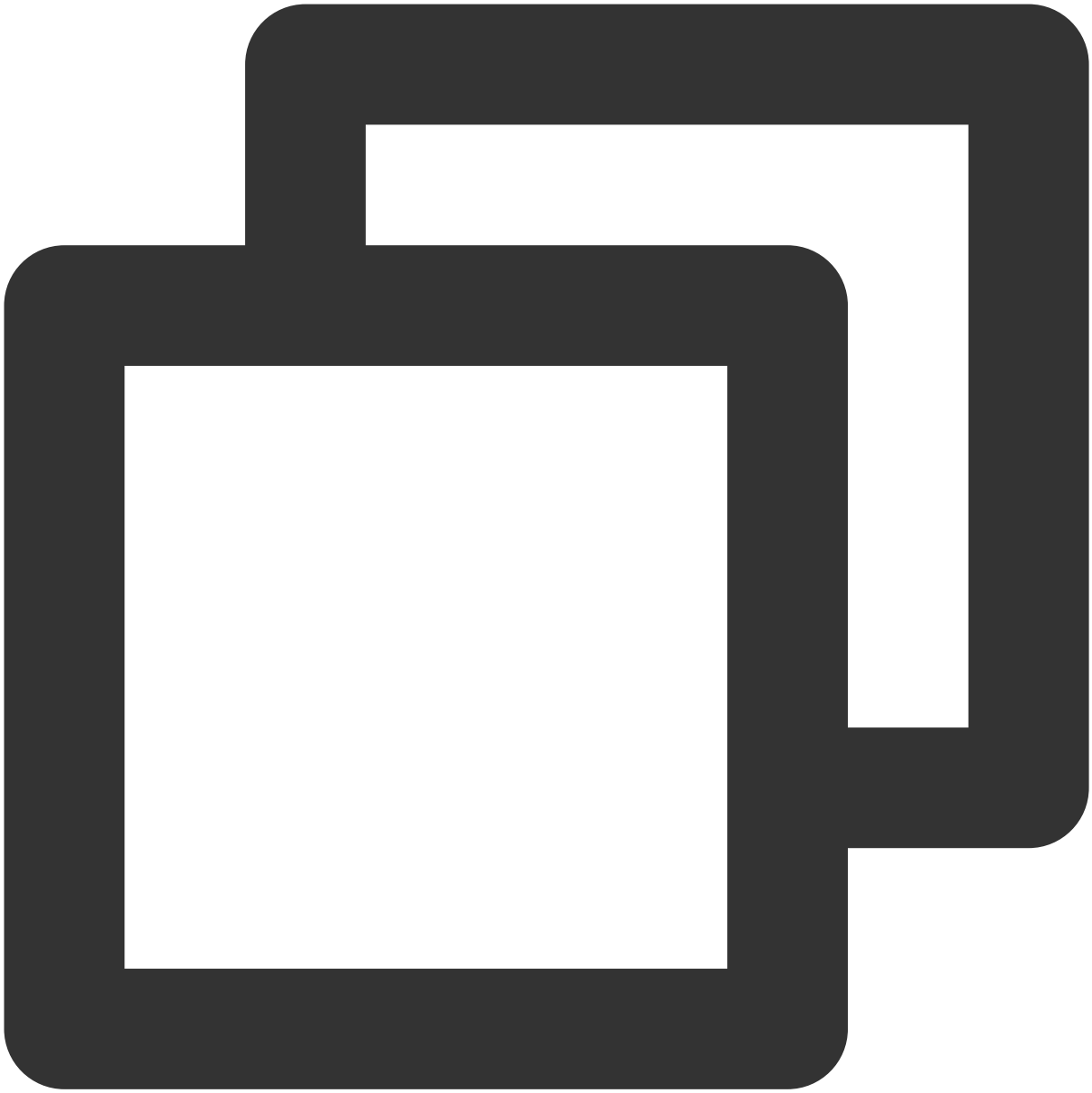
启用 GooseFS 监控指标上报配置

1. 编辑 GooseFs 配置 conf/goosefs-site.properties， 添加如下配置项，并使用 goosefs copyDir conf/ 拷贝到所有 worker节点，并重启集群 `./bin/goosefs-start.sh all` 。



```
goosefs.user.metrics.collection.enabled=true
goosefs.user.metrics.heartbeat.interval=10s
```

2. master 和 worker 的 Prometheus 的监控指标可用如下的命令查看，其中 master 的 metrics 端口为9201，worker 的 metrics 端口为 9204：

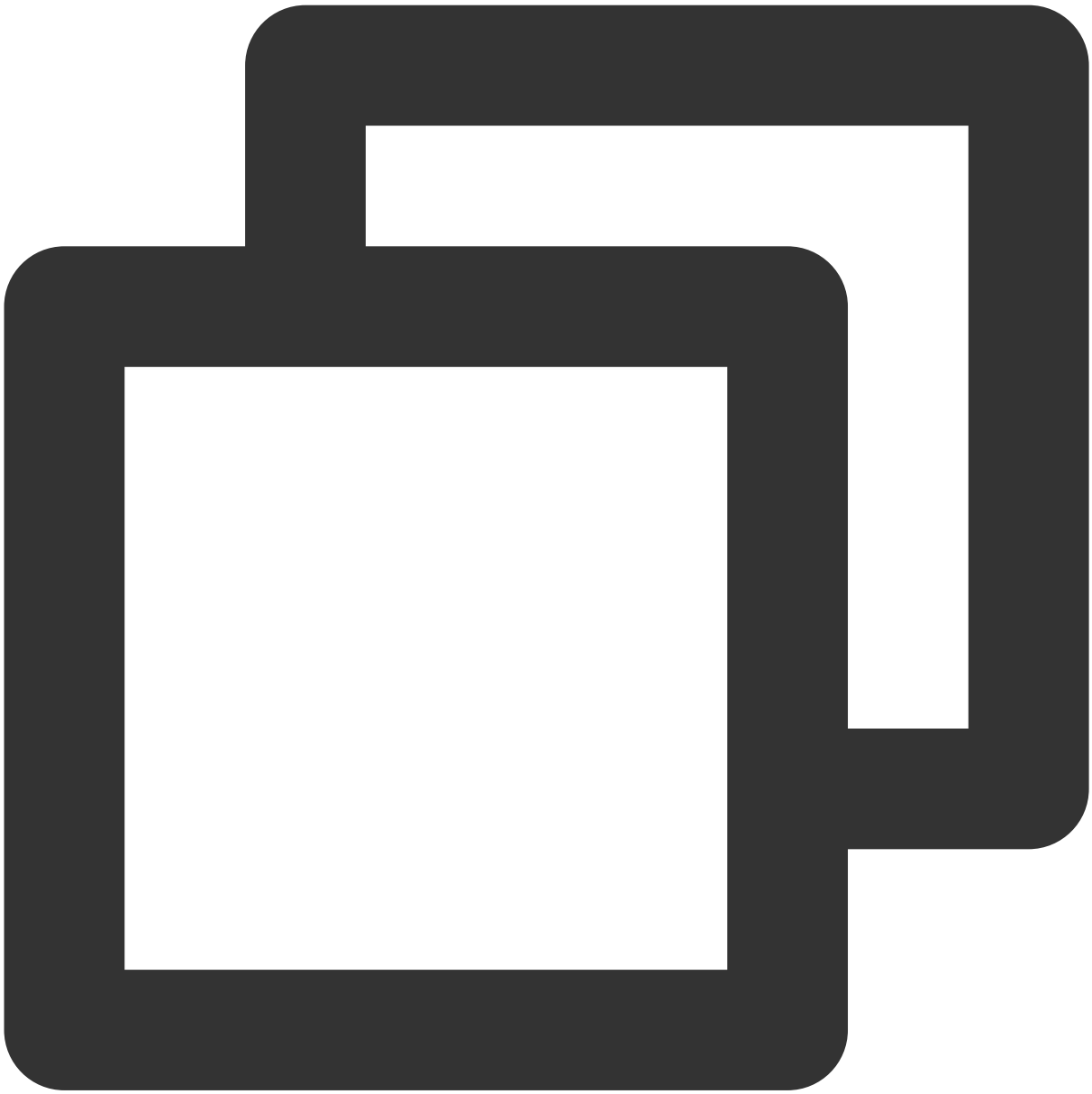


```
curl <LEADING_MASTER_HOSTNAME>:<MASTER_WEB_PORT>/metrics/prometheus/
# HELP Master_CreateFileOps Generated from Dropwizard metric import (metric=Master.
...

curl <WORKER_IP>:<WOKER_PORT>/metrics/prometheus/
# HELP pools_Code_Cache_max Generated from Dropwizard metric import (metric=pools.C
...
```

版权所有：腾讯云计算（北京）有限责任公司

上报监控指标到自建 Prometheus

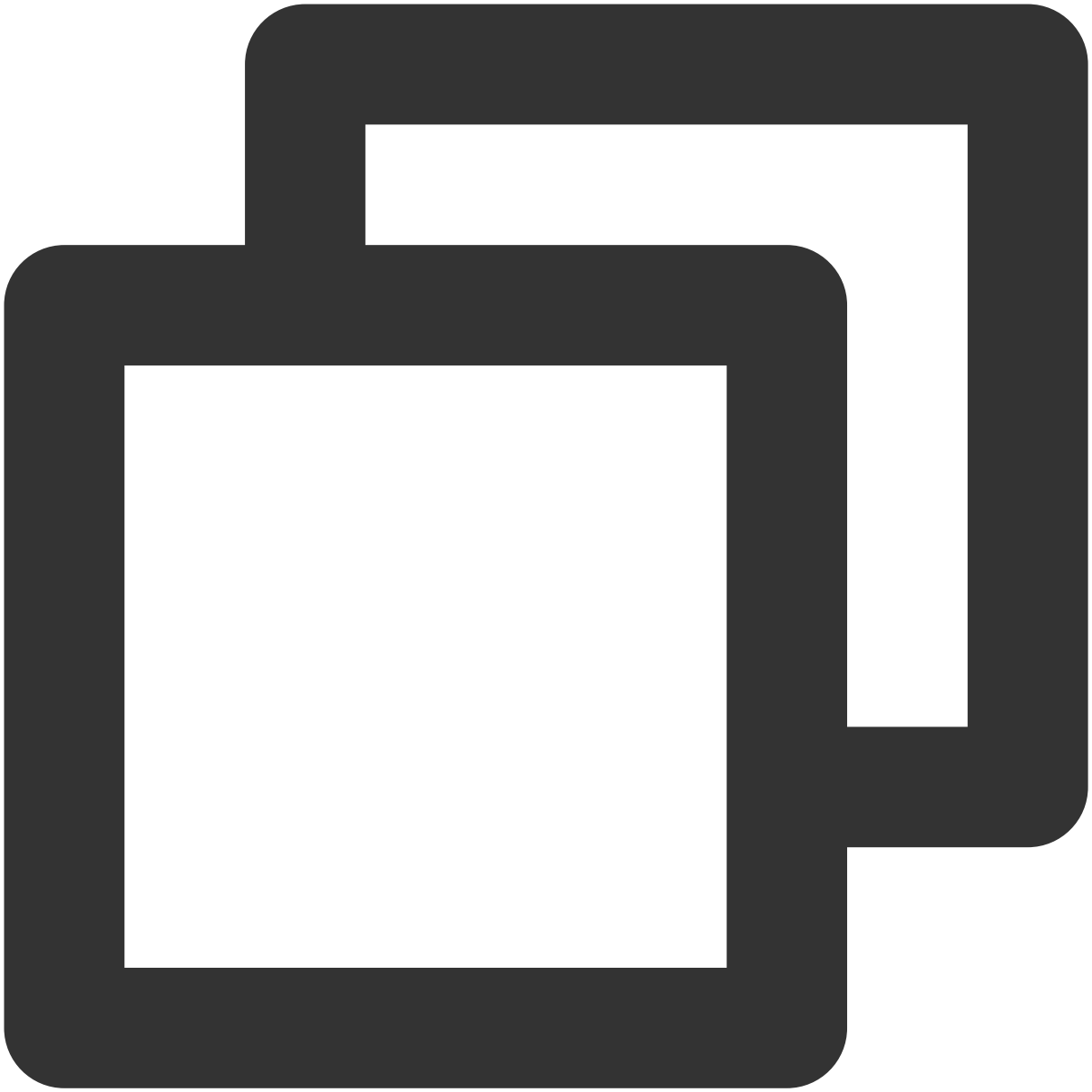
1. 下载 Prometheus 安装包并解压，修改 prometheus.yml：



```
# prometheus.yml
global:
 scrape_interval:     10s
 evaluation_interval: 10s
scrape_configs:
 - job_name: 'goosefs masters'
     metrics_path: /metrics/prometheus
```

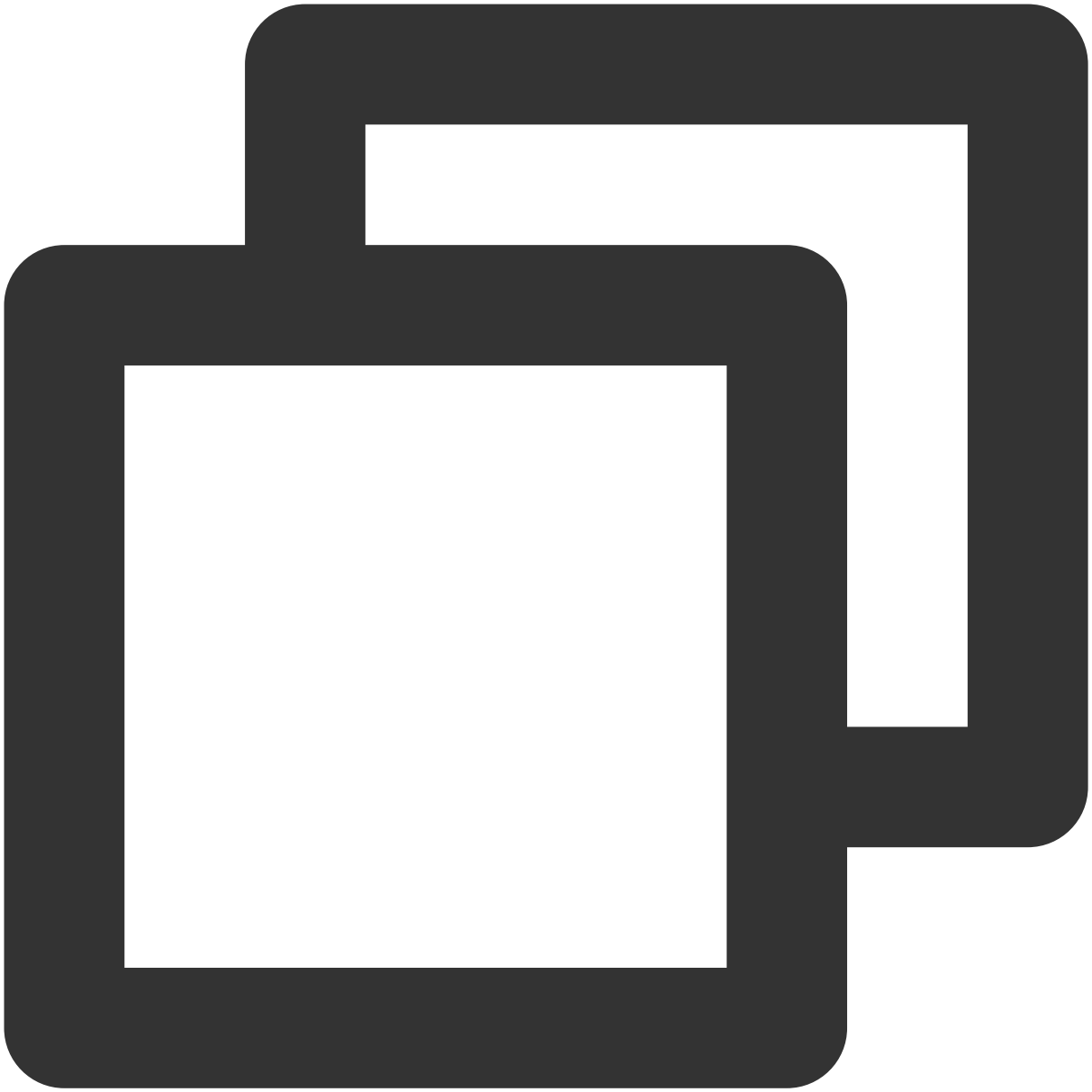
```
    file_sd_configs:
    - refresh_interval: 1m
    files:
    - "targets/cluster/masters/*.yml"
- job_name: 'goosefs workers'
    metrics_path: /metrics/prometheus
    file_sd_configs:
    - refresh_interval: 1m
    files:
    - "targets/cluster/workers/*.yml"
```

2. 创建 targets/cluster/masters/masters.yml，添加 master 的 IP 和 port：



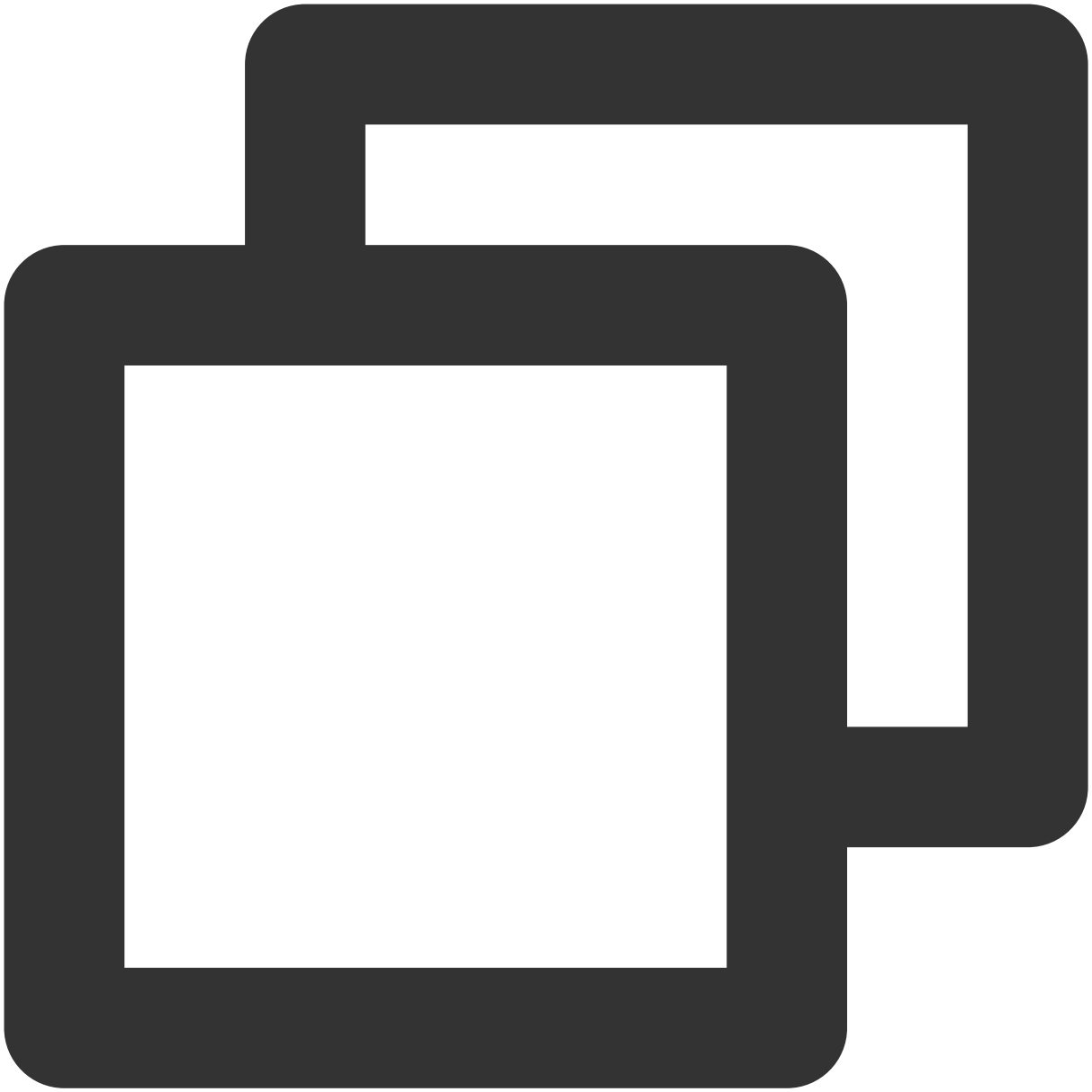
```
- targets:
  - "&lt;TARGERTS_MASTER_IP>:&lt;TARGERTS_MASTER_PORT>"
```

3. 创建 targets/cluster/workers/workers.yml，添加 worker 的 IP 和 port：



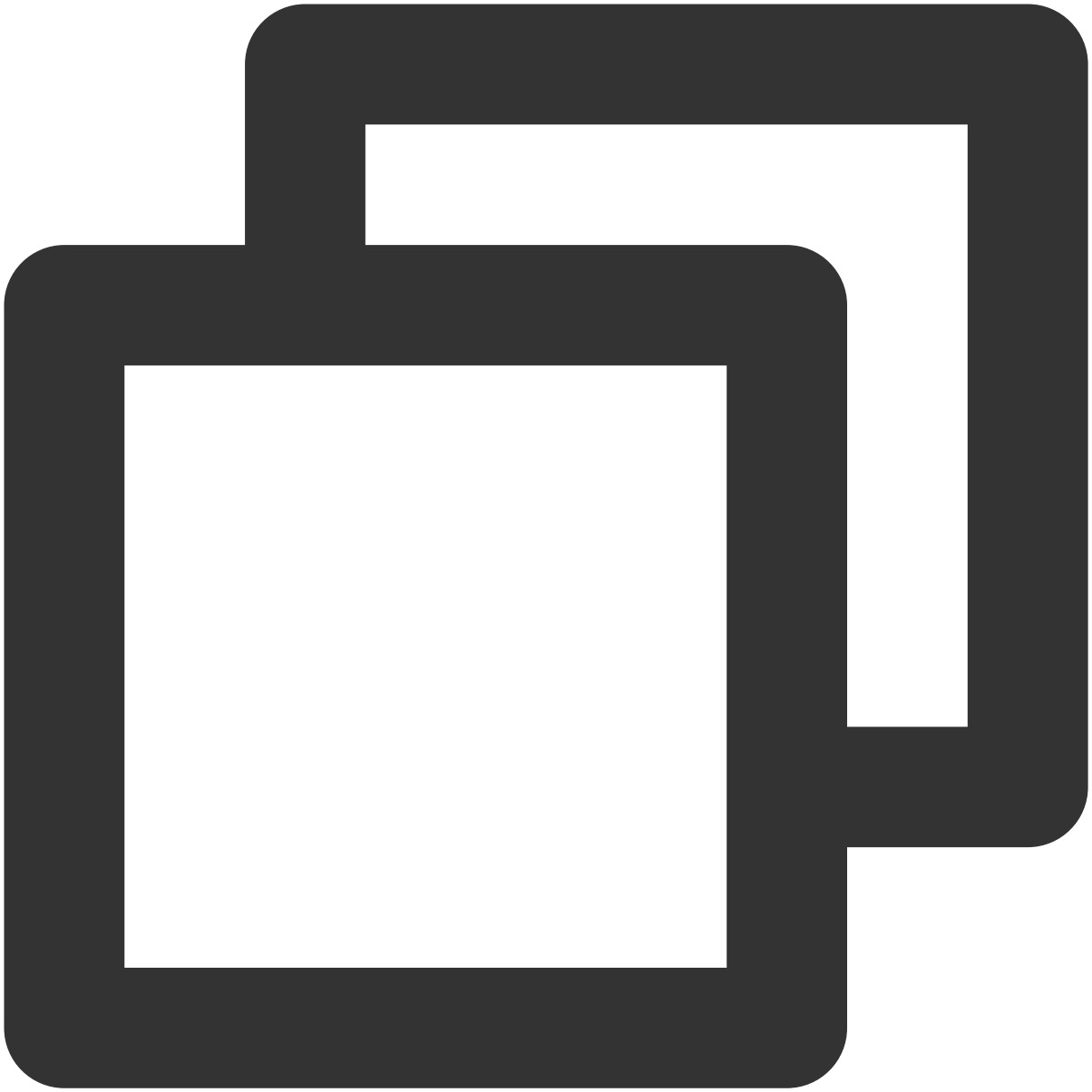
```
- targets:
  - "&lt;TARGERTS_WORKER_IP>:&lt;TARGERTS_WORKER_PORT>"
```

4. 启动 Prometheus，其中 --web.listen-address 指定 Prometheus 监听地址，默认端口号 9090：



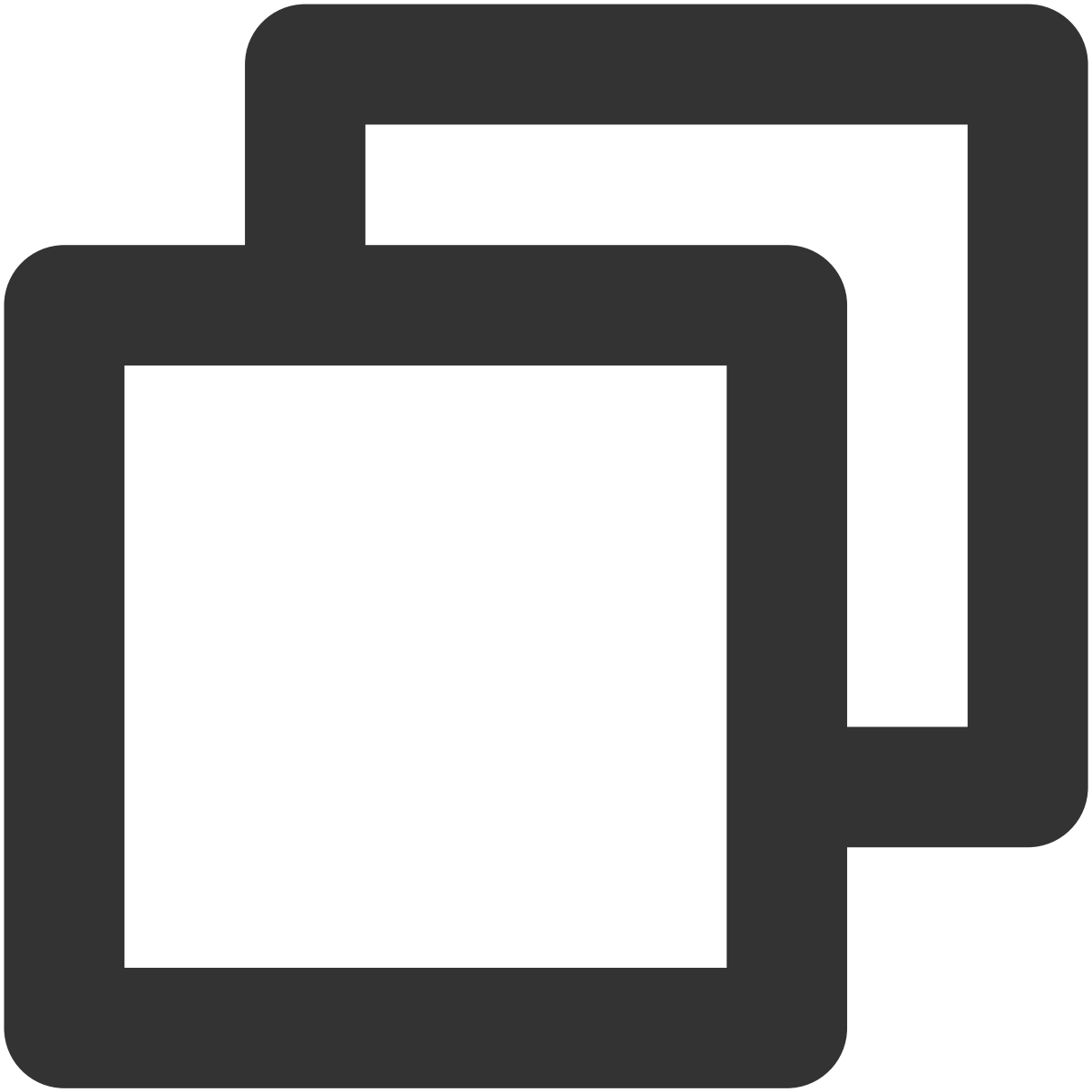
```
nohup ./prometheus --config.file=prometheus.yml --web.listen-address="<LISTEN_IP>:<
```

5. 查看可视化界面：



```
http://<PROMETHEUS_BI_IP>:<PROMETHEUS_BI_PORT>
```

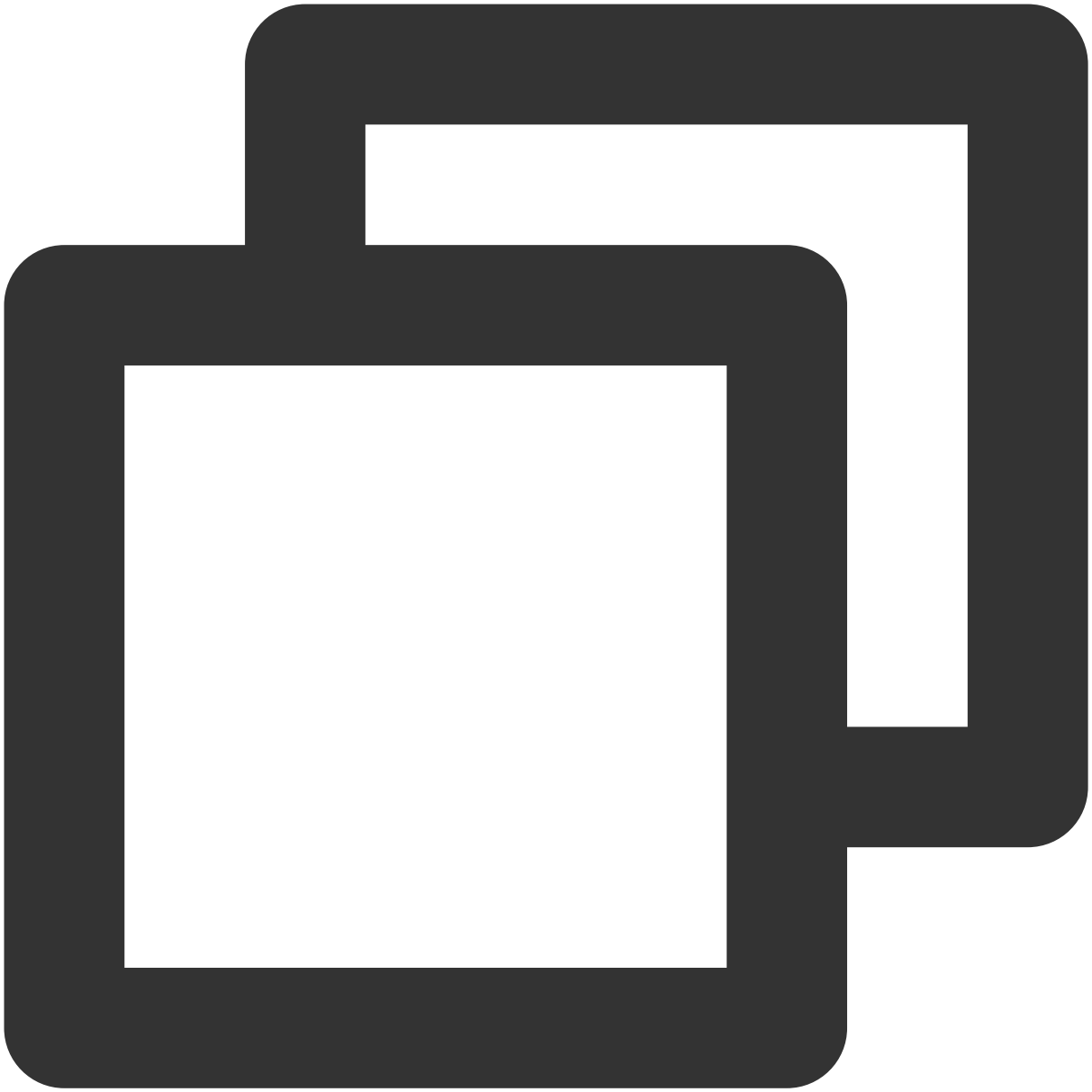
6. 查看机器实例：



```
http://<PROMETHEUS_BI_IP>:<PROMETHEUS_BI_PORT>/targets
```

上报监控指标到腾讯云 Prometheus

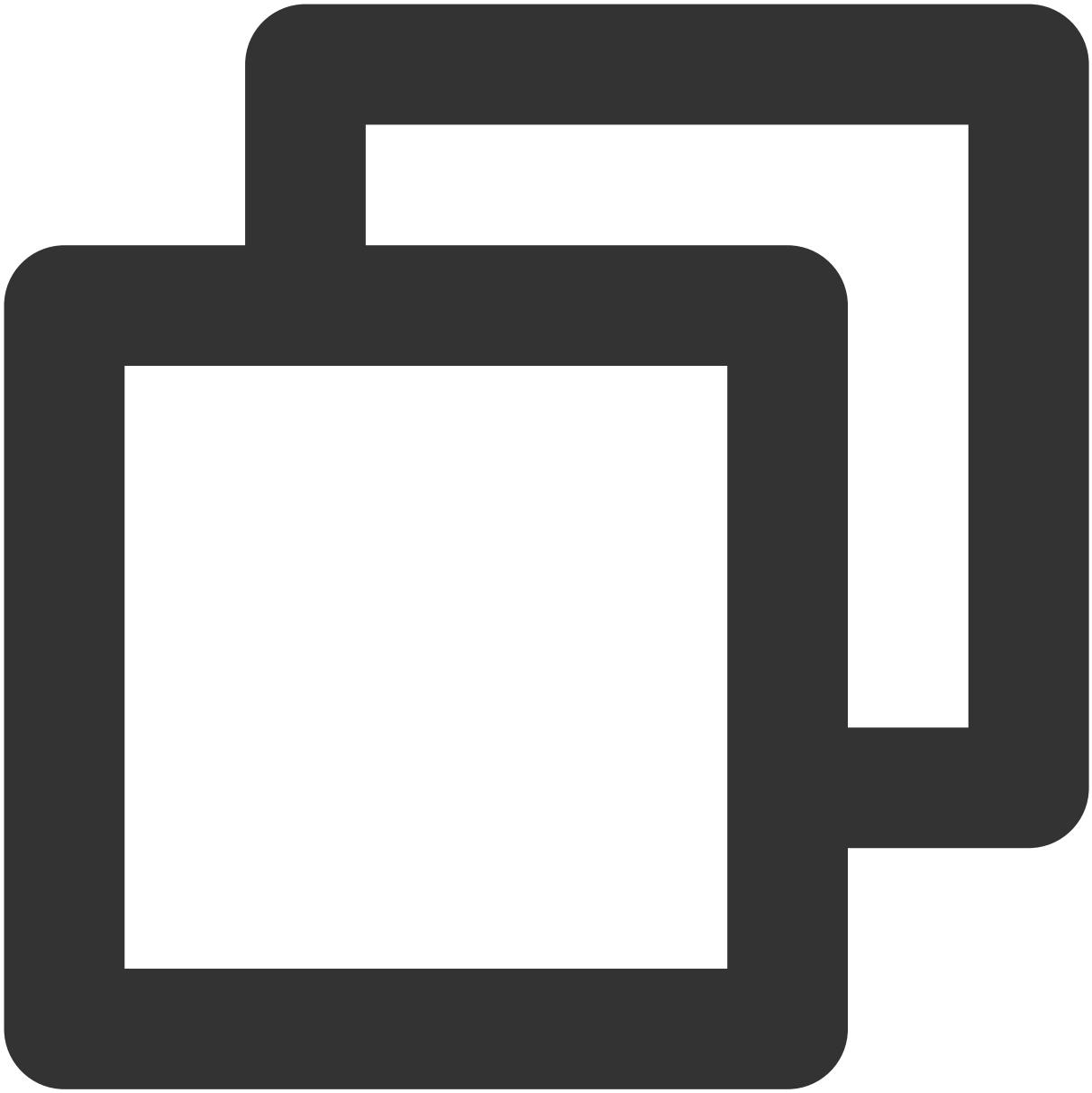
1. 按照安装指南中的指引，在 master 机器上安装 Promethus agent：



```
wget https://rig-1258344699.cos.ap-guangzhou.myqcloud.com/prometheus-agent/agent_in
```

2. 配置 master 和 worker 的抓取任务：

方式一：



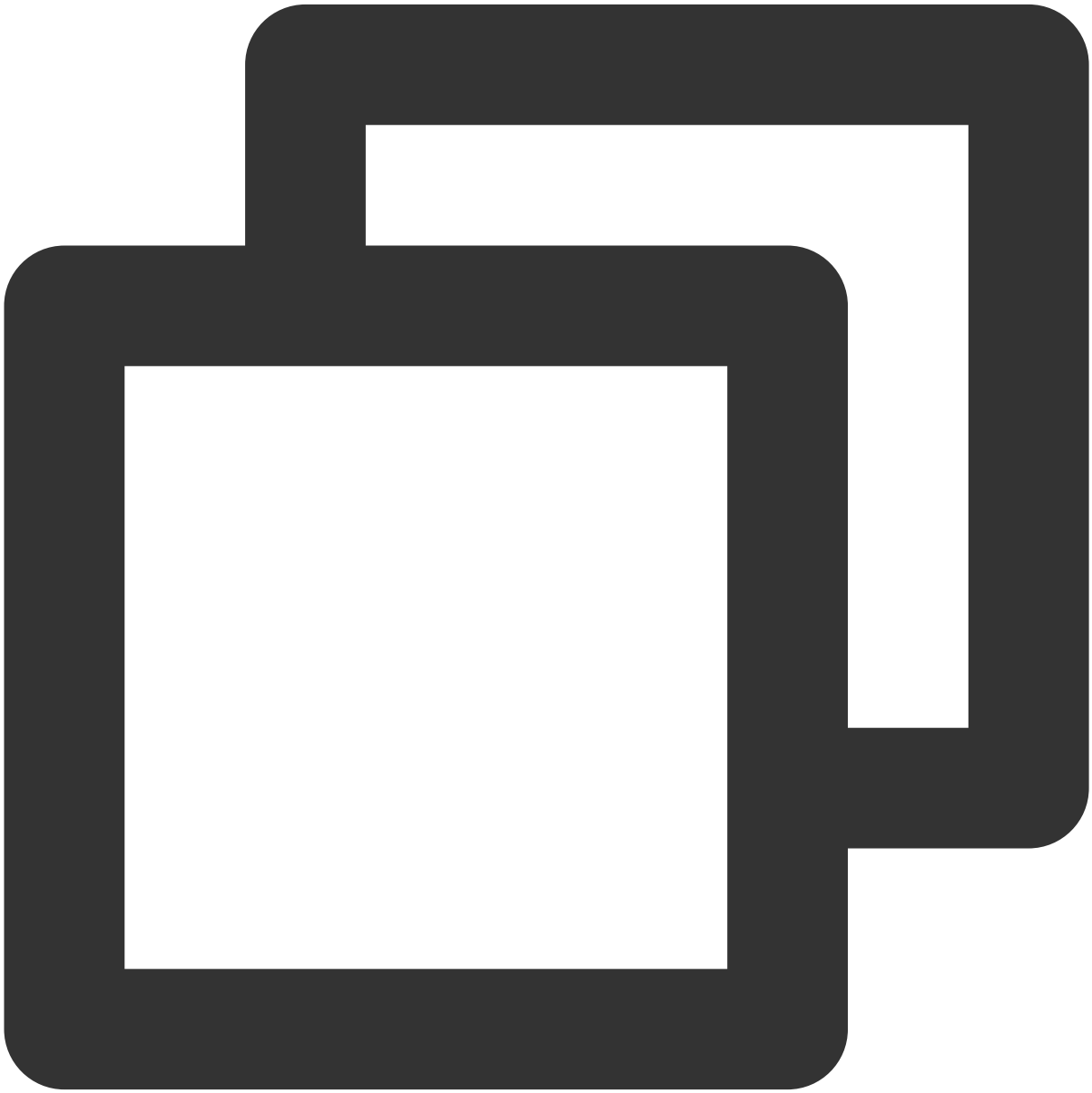
```
job_name: goosefs-masters
honor_timestamps: true
metrics_path: /metrics/prometheus
scheme: http
file_sd_configs:
- files:
  - /usr/local/services/prometheus/targets/cluster/masters/*.yml
  refresh_interval: 1m
job_name: goosefs-workers
honor_timestamps: true
metrics_path: /metrics/prometheus
```

```
scheme: http
file_sd_configs:
 - files:
  - /usr/local/services/prometheus/targets/cluster/workers/*.yml
  refresh_interval: 1m
```

注意

job_name 中没有空格，而单机的 Prometheus 的 job_name 中可以包含空格。

方式二：



```
job_name: goosefs masters
```

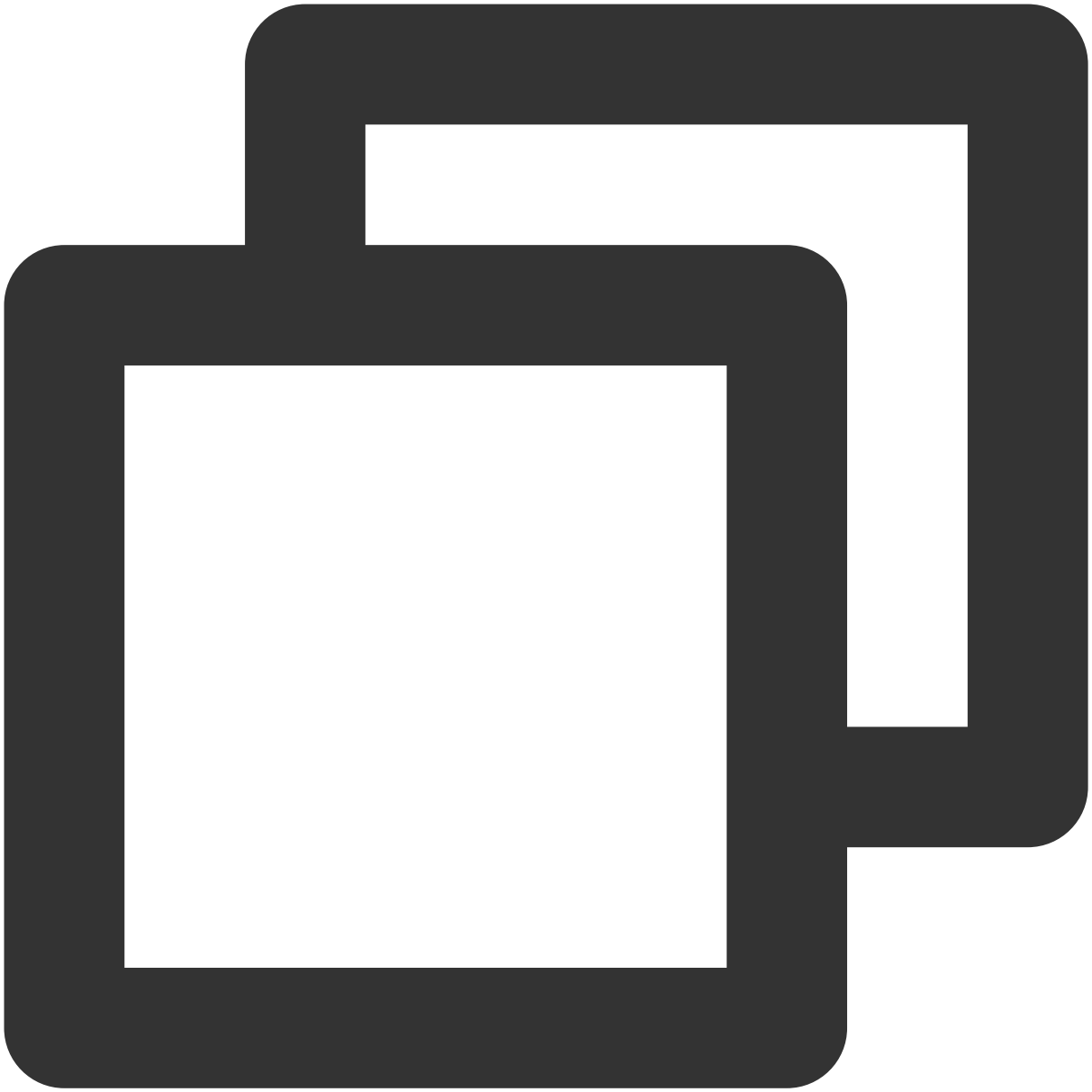
```
honor_timestamps: true
metrics_path: /metrics/prometheus
scheme: http
static_configs:
- targets:
 - "<TARGERTS_MASTER_IP>:<TARGERTS_MASTER_PORT>"
 refresh_interval: 1m

job_name: goosefs workers
honor_timestamps: true
metrics_path: /metrics/prometheus
scheme: http
static_configs:
- targets:
 - "<TARGERTS_WORKER_IP>:<TARGERTS_WORKER_PORT>"
 refresh_interval: 1m
```

注意

抓取任务按方式二配置，则无需在 targets/cluster/masters/ 路径下创建 masters.yml 和 workers.yml 文件。

使用 Grafana 查看监控指标

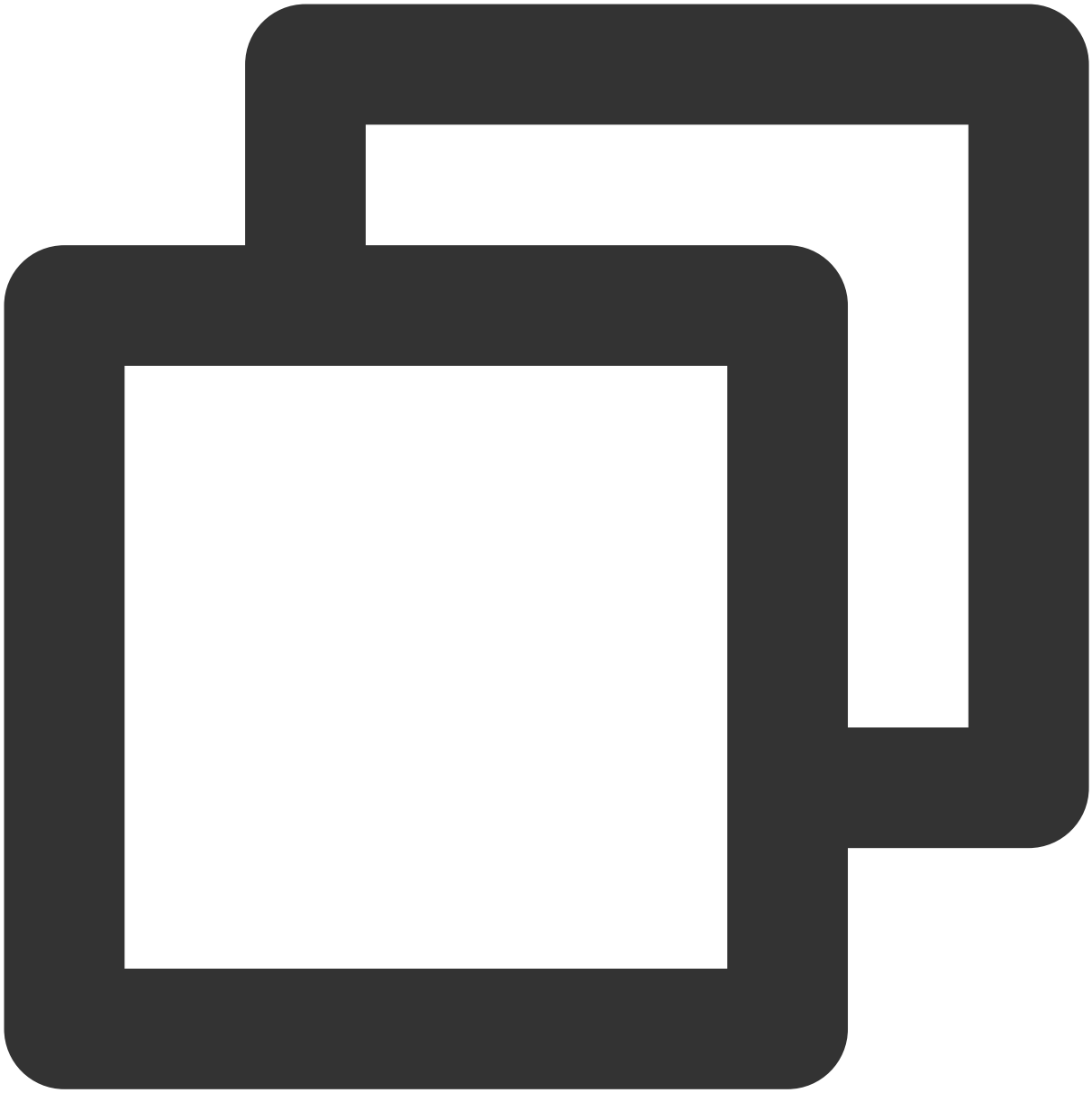
1. 启动 Grafana：



```
nohup ./bin/grafana-server web > grafana.log 2>&1 &
```

2. 打开登录页面 http://<GRAFANA_IP>:<GRAFANA_PORT>，Grafana 的默认端口为 3000，username 和 password 都是 admin，首次登录后可修改密码。

3. 进入页面后，添加 Prometheus 的 Datasource：



```
<PROMETHEUS_IP>:<PROMETHEUS_PORT>
```

4. 导入 Goosefs 的 Grafana 模板，选择 json 导入（点此下载 json），并选择上面创建的 Datasource。

注意

云上 Prometheus 购买时需设置密码，云上 Grafana 的可视化监控界面配置和上面类似，注意 job_name 需要配置成一致。

5. 修改 DashBoard 以后，可以将 DashBoard 导出来。

集群配置实践

最近更新时间：2024-01-06 11:17:58

数据加速器 GooseFS 提供了多种部署方式，本文档主要介绍在大数据场景中，通常使用的集群模式部署。

AI 场景生产环境配置实践

大数据场景生产环境配置实践

数据安全
使用 Apache Ranger 控制 GooseFS 的访问权限

最近更新时间：2024-01-06 11:17:59

概述

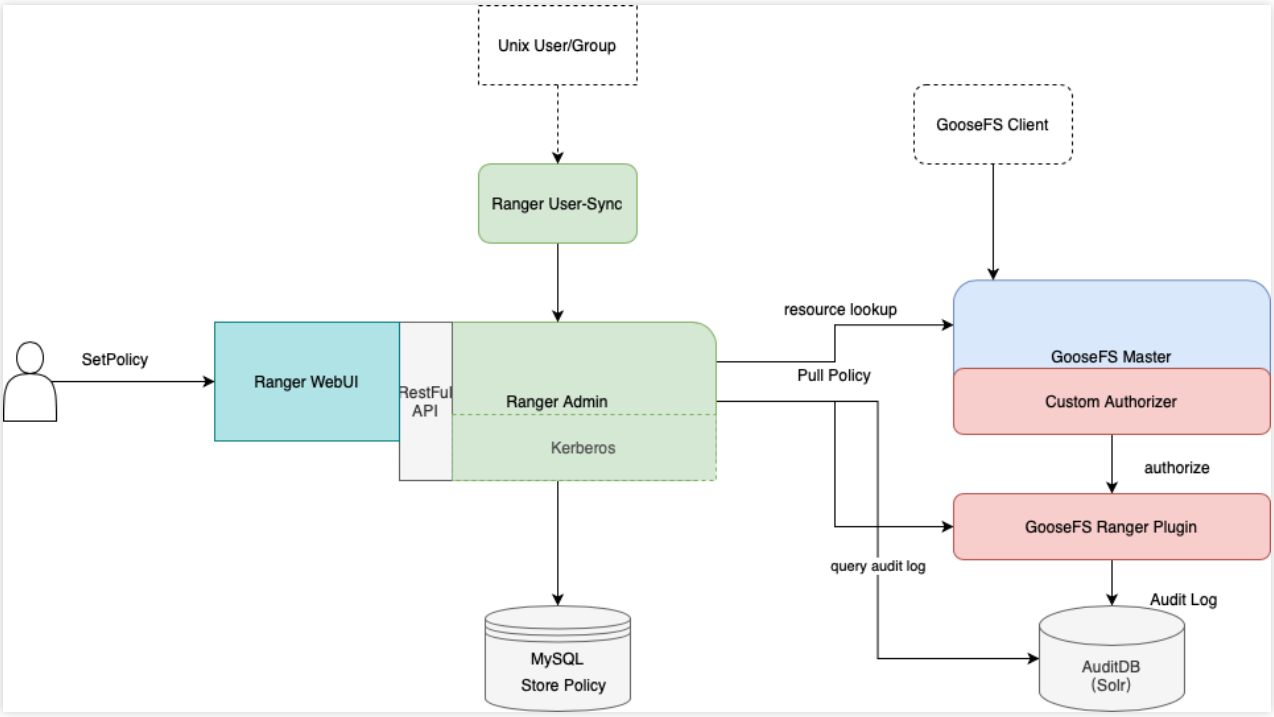
Apache Ranger 是大数据生态系统中用于控制访问权限的一个标准鉴权组件，GooseFS 作为大数据和数据湖场景下的加速存储系统，也已经支持接入 Apache Ranger 的统一鉴权平台中，本文将介绍使用 Apache Ranger 控制 GooseFS 的资源访问权限。

优势

GooseFS 作为一款云原生加速存储系统，在 Apache Ranger 支持上已经做到了与 HDFS 近乎一致的访问控制行为。因此，原先使用 HDFS 的大数据用户可以非常轻松地迁移到 GooseFS 上来，并且直接复用 HDFS 的 Ranger 权限策略，即可获得一致的使用体验。
GooseFS with Ranger 相比 HDFS with Ranger 的鉴权架构，还额外提供了 Ranger + 原生 ACL 的联合鉴权选项，在 Ranger 鉴权失效时，还可选择使用原生 ACL 鉴权，可解决一些 Ranger 鉴权策略配置不完善的问题。

GooseFS with Ranger 的鉴权架构

版权所有：腾讯云计算（北京）有限责任公司



为了支持将 GooseFS 集成到 Ranger 鉴权平台中，我们开发了 GooseFS Ranger Plugin，它同时部署在 GooseFS Master 节点和 Ranger Admin 侧。负责完成如下工作：

GooseFS Master 节点侧：

提供 `Authorizer` 接口，为 GooseFS Master 上的每一次元数据请求提供鉴权结果。

连接 Ranger Admin 获取用户配置的鉴权策略。

Ranger Admin 侧：

为 Ranger Admin 提供 GooseFS 的资源查找（resource lookup）的能力。

提供配置校验的能力。

开始部署

准备工作

在开始使用前，需确保环境中已部署并配置了 Ranger 相关的组件（即包括：Ranger Admin 和 Ranger UserSync），并且确保 Ranger 的 WebUI 可以正常打开和使用。

部署组件

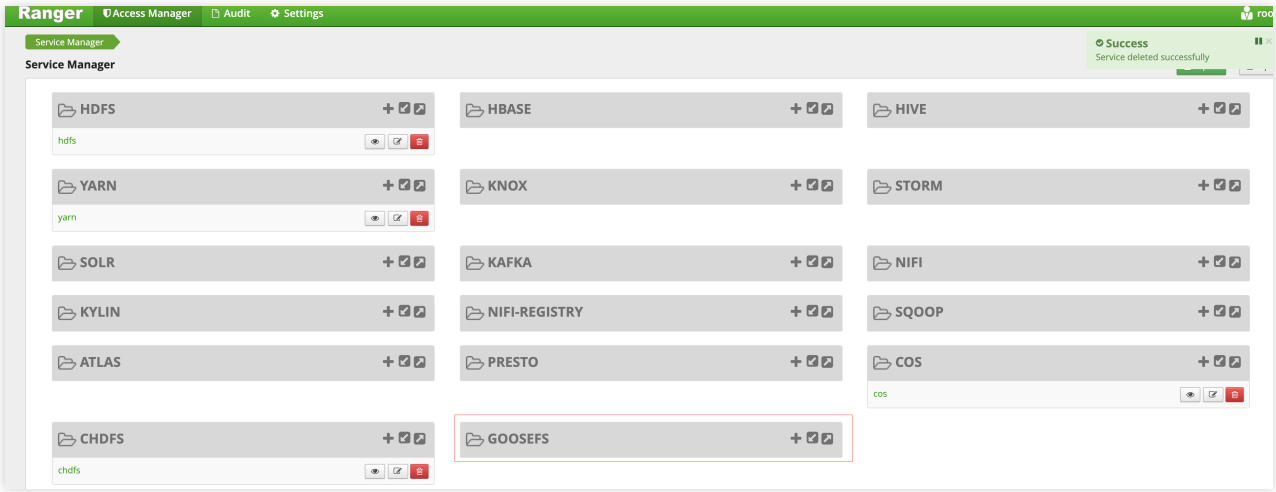
在 Ranger Admin 侧部署 GooseFS Ranger Plugin 并注册对应服务

说明

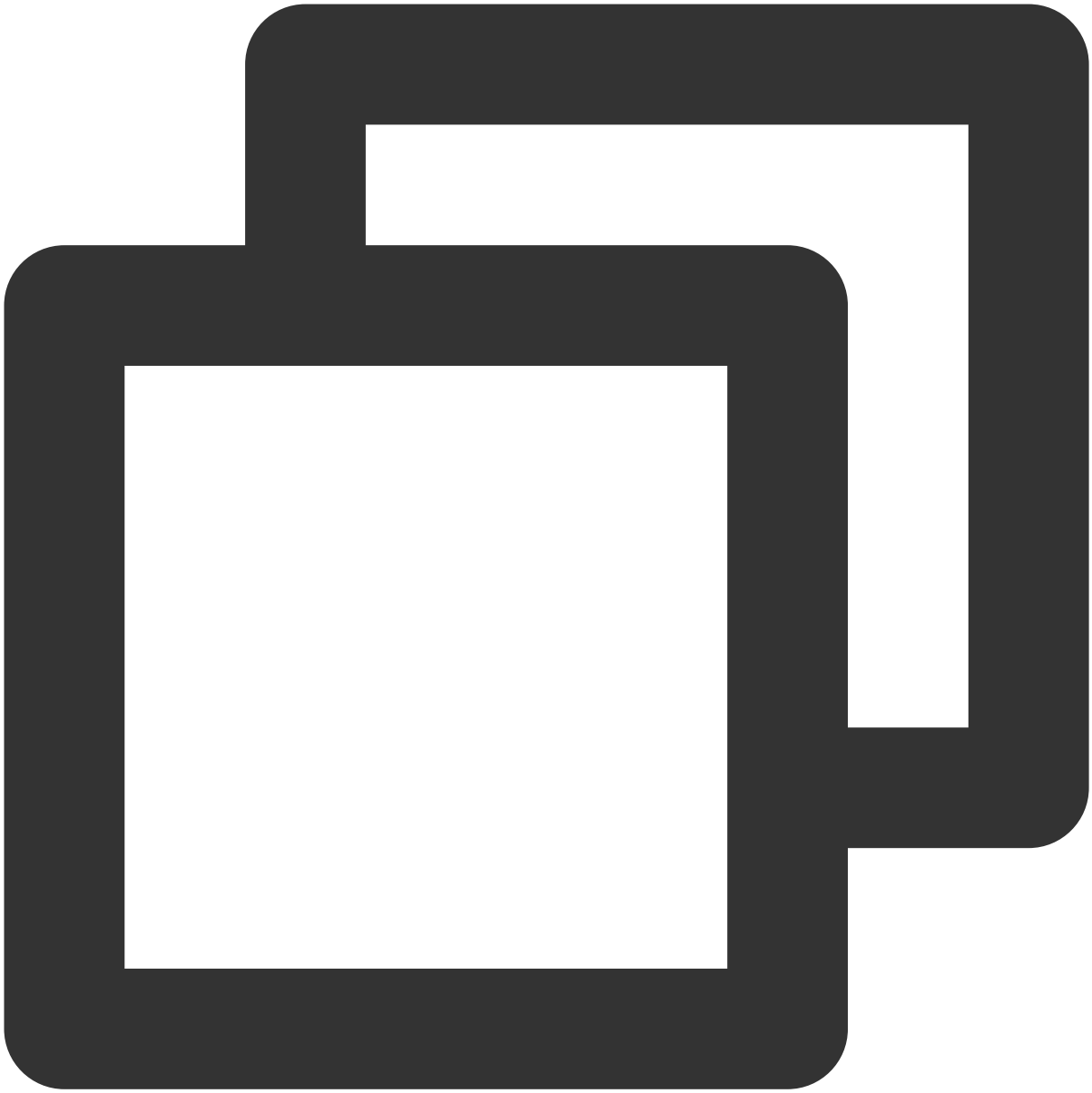
单击 此处 下载 GooseFS Ranger Plugin。

部署步骤如下：

1. 在 Ranger 服务定义目录下新建 GooseFS 的目录（注意，目录权限至少保证 x 与 r 的权限）。
1. 如果使用的是腾讯云 EMR 集群，则 Ranger 的服务定义目录
在：`/usr/local/service/ranger/ews/webapp/WEB-INF/classes/ranger-plugins`。
2. 如果是自建 Hadoop 集群，则可以通过在 ranger 目录下查找 hdfs 等已经接入到 ranger 服务的组件，查找目录位置。



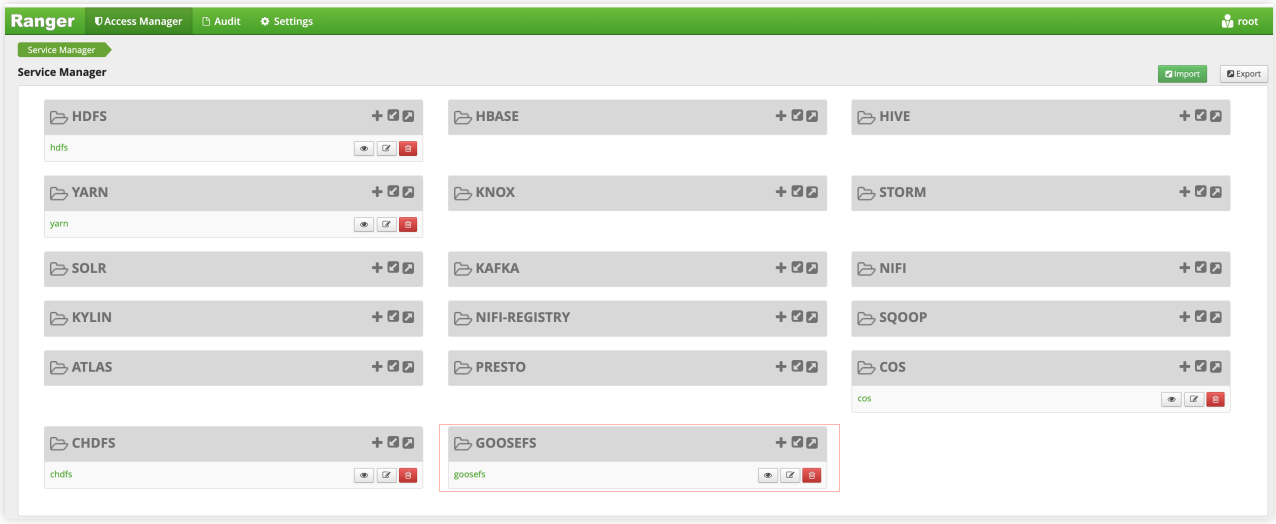
3. 在 GooseFS 的目录下，放入 goosefs-ranger-plugin-${version}.jar 和 ranger-servicedef-goosefs.json，并且具备读权限。
4. 重启 Ranger 服务。
5. 在 Ranger 上，按照如下命令，注册 GooseFS Service。



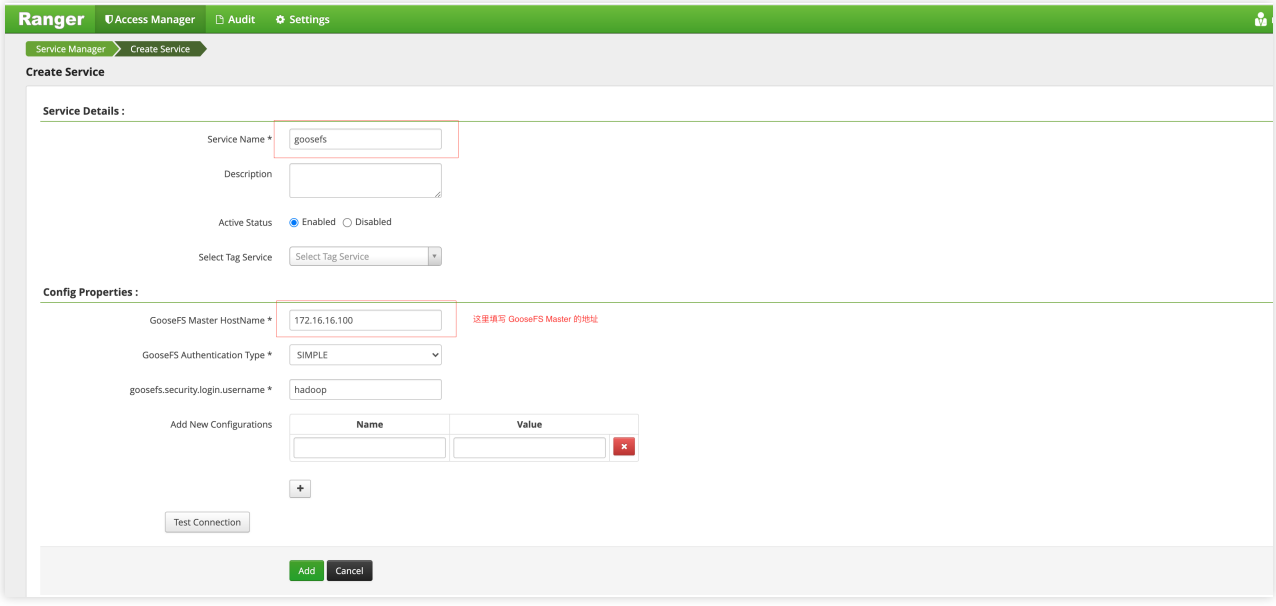
```
# 生成服务，需要传入 Ranger 管理员的账号和密码，以及 Ranger 的服务地址
# 对于腾讯云 EMR 集群，管理员用户是 root，密码是构建 EMR 集群时设置的 root 密码，ranger 服务的
adminUser=root
adminPasswd=xxxx

rangerServerAddr=10.0.0.1:6080

curl -v -u${adminUser}:${adminPasswd} -X POST -H "Accept:application/json" -H "Cont

# 服务注册成功后，会返回一个服务 ID，请务必记录下这个ID
# 如果要删除 GooseFS 的服务，则传入刚刚返回的服务 ID，执行如下命令即可：
```

```
serviceId=104
curl -v -u${adminUser}:${adminPasswd} -X DELETE -H "Accept:application/json" -H "Co
```

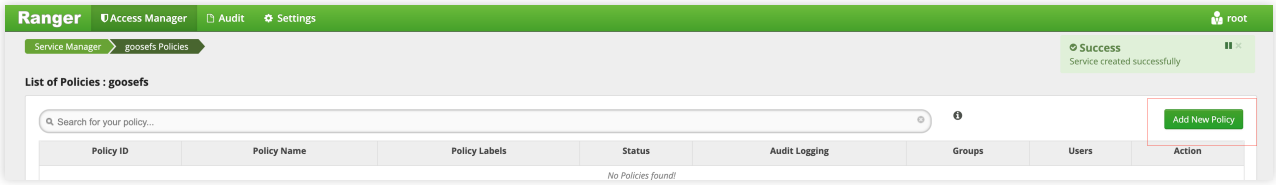
6. 创建成功后，在 Ranger 的 Web 控制台上即可看到 GooseFS 相关的服务：



7. 在 GooseFS 服务侧单击【+】，定义 goosefs 服务实例。



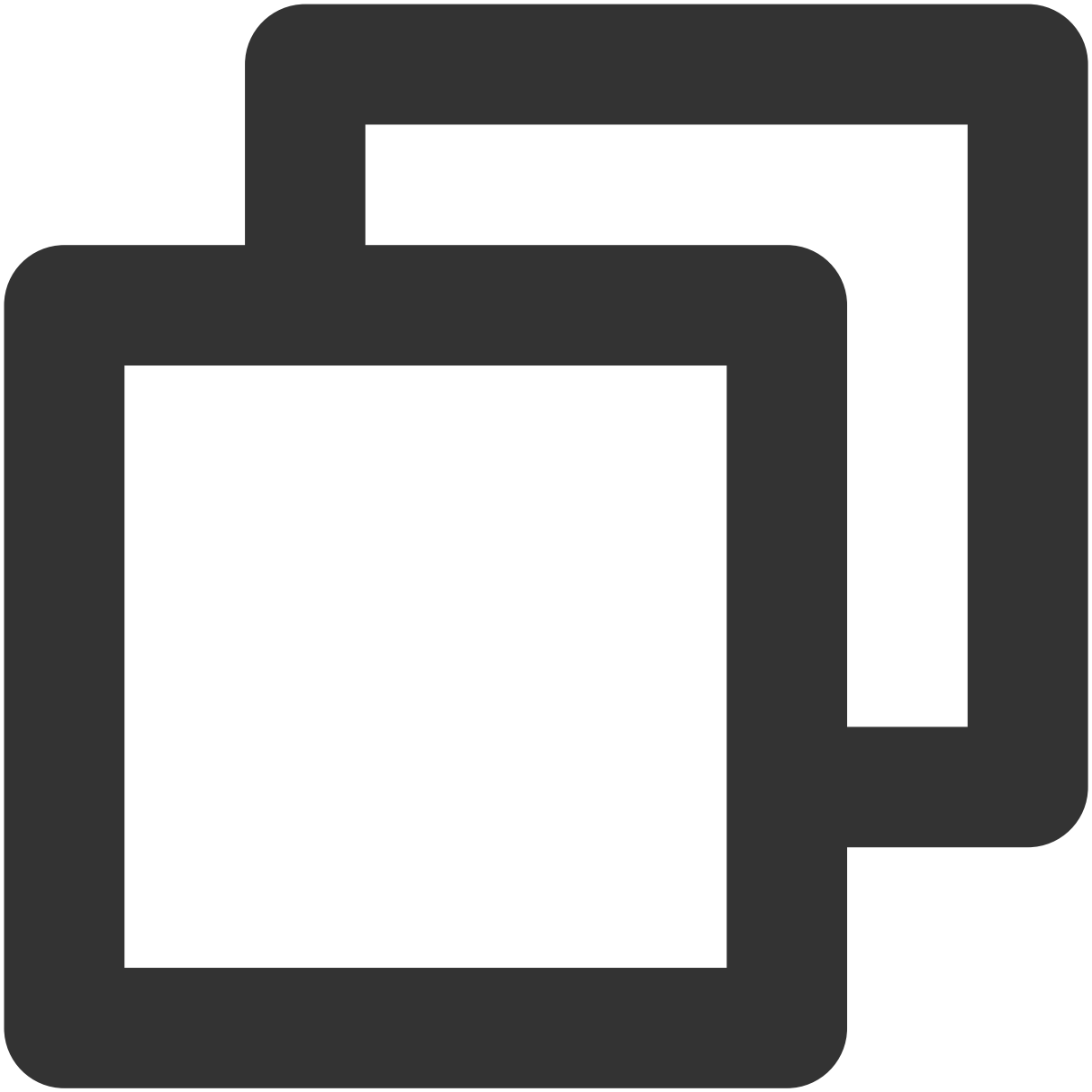
8. 单击新生成的 goosefs 服务实例，即可添加鉴权 policy。



在 GooseFS Master 侧部署 GooseFS Ranger Plugin 并配置启用 Ranger 鉴权

1. 将 goosefs-ranger-plugin-${version}.jar 放入 ${GOOSEFS_HOME}/lib 路径下，并且至少具备读权限。
2. 将 ranger-goosefs-audit.xml、ranger-goosefs-security.xml 以及 ranger-policymgr-ssl.xml 三个文件放入 `\\${GOOSEFS_HOME}/conf` 路径下，并分别填写其必要配置：

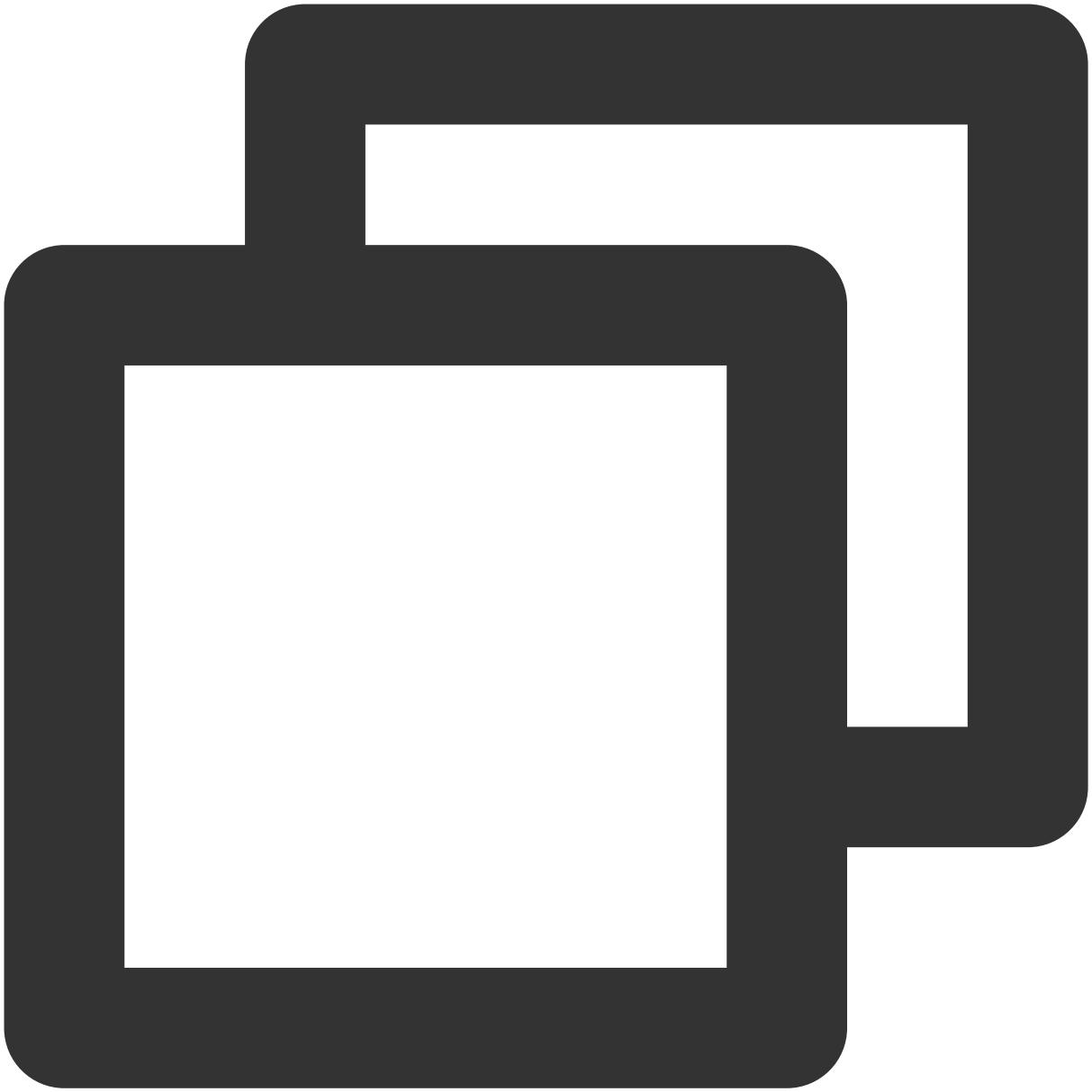
ranger-goosefs-security.xml：



```
<configuration xmlns:xi="http://www.w3.org/2001/XInclude">
  <property>
    <name>ranger.plugin.goosefs.service.name</name>
    <value>goosefs</value>
  </property>

  <property>
    <name>ranger.plugin.goosefs.policy.source.impl</name>
    <value>org.apache.ranger.admin.client.RangerAdminRESTClient</value>
  </property>
```

```
    <property>
      <name>ranger.plugin.goosefs.policy.rest.url</name>
      <value>http://10.0.0.1:6080</value>
    </property>

    <property>
      <name>ranger.plugin.goosefs.policy.pollIntervalMs</name>
      <value>30000</value>
    </property>

    <property>
      <name>ranger.plugin.goosefs.policy.rest.client.connection.timeoutMs</name>
      <value>1200</value>
    </property>

    <property>
      <name>ranger.plugin.goosefs.policy.rest.client.read.timeoutMs</name>
      <value>30000</value>
    </property>
  </configuration>
```

ranger-goosefs-audit.xml（不开启审计，可不配置）：



```
<configuration>
  <property>
 <name>xasecure.audit.is.enabled</name>
 <value>false</value>
  </property>

  <property>
 <name>xasecure.audit.db.is.async</name>
 <value>true</value>
  </property>
```

```
 <property>
<name>xasecure.audit.db.async.max.queue.size</name>
<value>10240</value>
 </property>

 <property>
<name>xasecure.audit.db.async.max.flush.interval.ms</name>
<value>30000</value>
 </property>

 <property>
<name>xasecure.audit.db.batch.size</name>
<value>100</value>
 </property>

 <property>
<name>xasecure.audit.jpa.javax.persistence.jdbc.url</name>
<value>jdbc:mysql://localhost:3306/ranger_audit</value>
 </property>

 <property>
<name>xasecure.audit.jpa.javax.persistence.jdbc.user</name>
<value>rangerLogger</value>
 </property>

 <property>
<name>xasecure.audit.jpa.javax.persistence.jdbc.password</name>
<value>none</value>
 </property>

 <property>
<name>xasecure.audit.jpa.javax.persistence.jdbc.driver</name>
<value>com.mysql.jdbc.Driver</value>
 </property>

 <property>
<name>xasecure.audit.credential.provider.file</name>
<value>jceks://file/etc/ranger/hadoopdev/auditcred.jceks</value>
 </property>

 <property>
<name>xasecure.audit.hdfs.is.enabled</name>
<value>true</value>
 </property>

 <property>
<name>xasecure.audit.hdfs.is.async</name>
```

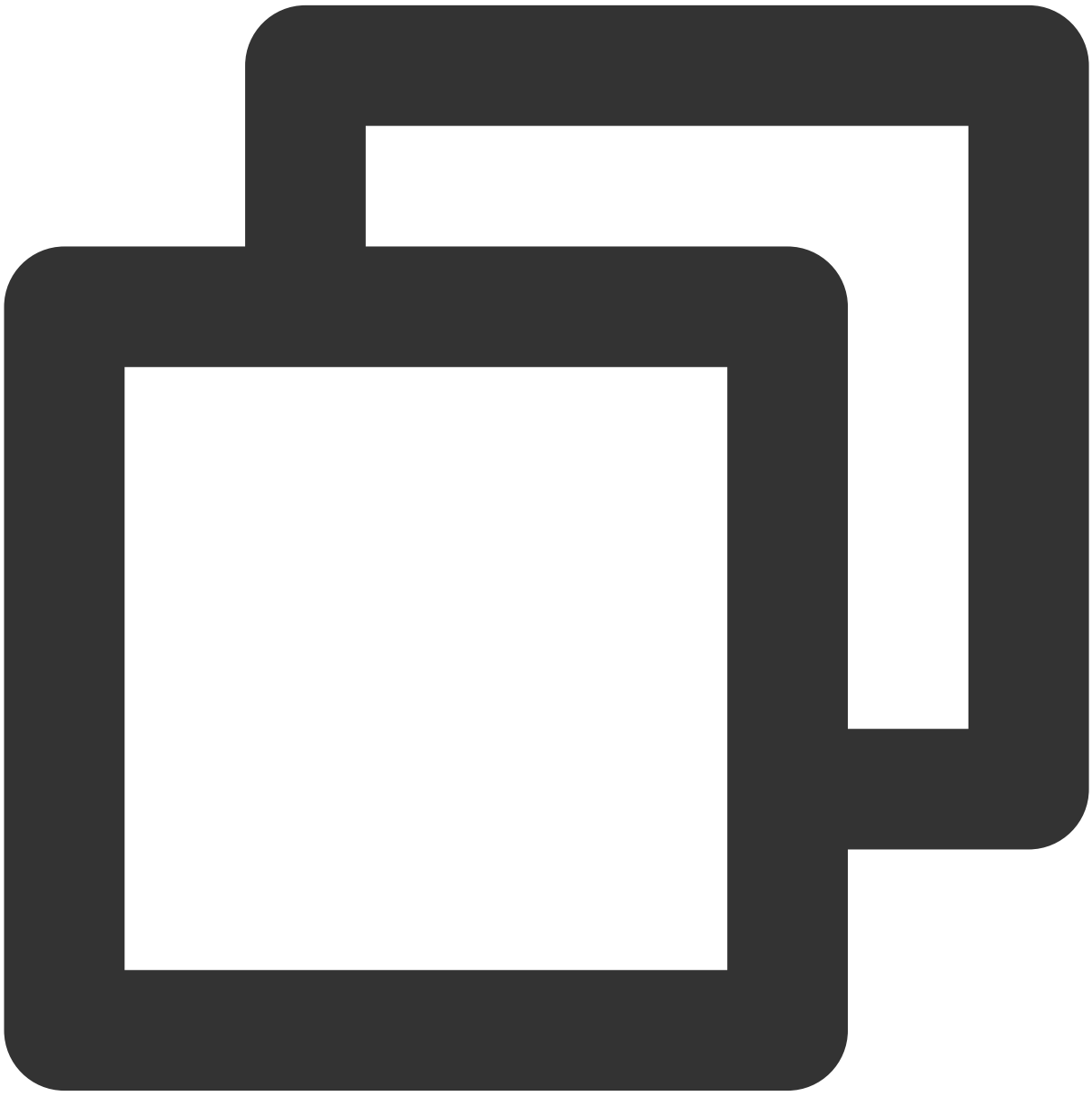
```
  <value>true</value>
   </property>

   <property>
  <name>xasecure.audit.hdfs.async.max.queue.size</name>
  <value>1048576</value>
   </property>

   <property>
  <name>xasecure.audit.hdfs.async.max.flush.interval.ms</name>
  <value>30000</value>
   </property>

   <property>
  <name>xasecure.audit.hdfs.config.encoding</name>
  <value></value>
   </property>

<!--  hdfs audit provider config-->
   <property>
  <name>xasecure.audit.hdfs.config.destination.directory</name>
  <value>hdfs://NAMENODE_HOST:8020/ranger/audit/</value>
   </property>

   <property>
  <name>xasecure.audit.hdfs.config.destination.file</name>
  <value>%hostname%-audit.log</value>
   </property>

   <proeprty>
  <name>xasecure.audit.hdfs.config.destination.flush.interval.seconds</name>
  <value>900</value>
   </proeprty>

   <property>
  <name>xasecure.audit.hdfs.config.destination.rollover.interval.seconds</name>
  <value>86400</value>
   </property>

   <property>
  <name>xasecure.audit.hdfs.config.destination.open.retry.interval.seconds</name>
  <value>60</value>
   </property>

   <property>
  <name>xasecure.audit.hdfs.config.local.buffer.directory</name>
  <value>/var/log/hadoop/%app-type%/audit</value>
```

```
  </property>

  <property>
 <name>xasecure.audit.hdfs.config.local.buffer.file</name>
 <value>%time:yyyyMMdd-HHmm.ss%.log</value>
  </property>

  <property>
 <name>xasecure.audit.hdfs.config.local.buffer.file.buffer.size.bytes</name>
 <value>8192</value>
  </property>

  <property>
 <name>xasecure.audit.hdfs.config.local.buffer.flush.interval.seconds</name>
 <value>60</value>
  </property>

  <property>
 <name>xasecure.audit.hdfs.config.local.buffer.rollover.interval.seconds</name>
 <value>600</value>
  </property>

  <property>
 <name>xasecure.audit.hdfs.config.local.archive.directory</name>
 <value>/var/log/hadoop/%app-type%/audit/archive</value>
  </property>

  <property>
 <name>xasecure.audit.hdfs.config.local.archive.max.file.count</name>
 <value>10</value>
  </property>

<!--  log4j audit provider config -->
  <property>
 <name>xasecure.audit.log4j.is.enabled</name>
 <value>false</value>
  </property>

  <property>
 <name>xasecure.audit.log4j.is.async</name>
 <value>false</value>
  </property>

  <property>
 <name>xasecure.audit.log4j.async.max.queue.size</name>
 <value>10240</value>
  </property>
```

```
  <property>
 <name>xasecure.audit.log4j.async.max.flush.interval.ms</name>
 <value>30000</value>
  </property>

<!--  kafka audit provider config -->
  <property>
 <name>xasecure.audit.kafka.is.enabled</name>
 <value>false</value>
  </property>

  <property>
 <name>xasecure.audit.kafka.async.max.queue.size</name>
 <value>1</value>
  </property>

  <property>
 <name>xasecure.audit.kafka.async.max.flush.interval.ms</name>
 <value>1000</value>
  </property>

  <property>
 <name>xasecure.audit.kafka.broker_list</name>
 <value>localhost:9092</value>
  </property>

  <property>
 <name>xasecure.audit.kafka.topic_name</name>
 <value>ranger_audits</value>
  </property>

<!-- ranger audit solr config -->
  <property>
 <name>xasecure.audit.solr.is.enabled</name>
 <value>false</value>
  </property>

  <property>
 <name>xasecure.audit.solr.async.max.queue.size</name>
 <value>1</value>
  </property>

  <property>
 <name>xasecure.audit.solr.async.max.flush.interval.ms</name>
 <value>1000</value>
  </property>
```

```
  <property>
 <name>xasecure.audit.solr.solr_url</name>
 <value>http://localhost:6083/solr/ranger_audits</value>
  </property>
</configuration>
```

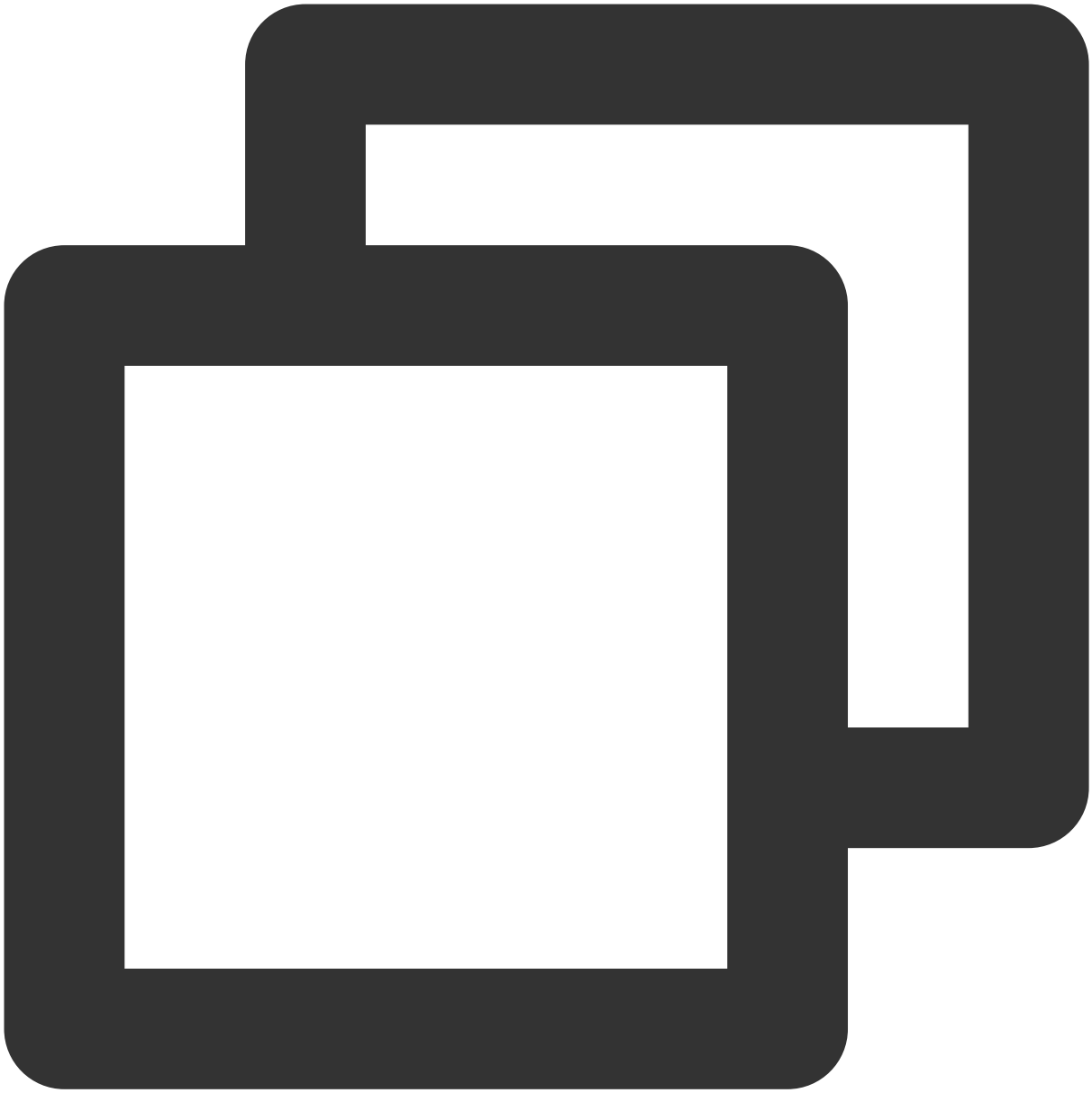
ranger-policymgr-ssl.xml



```
<configuration>
  <property>
    <name>xasecure.policymgr.clientssl.keystore</name>
```

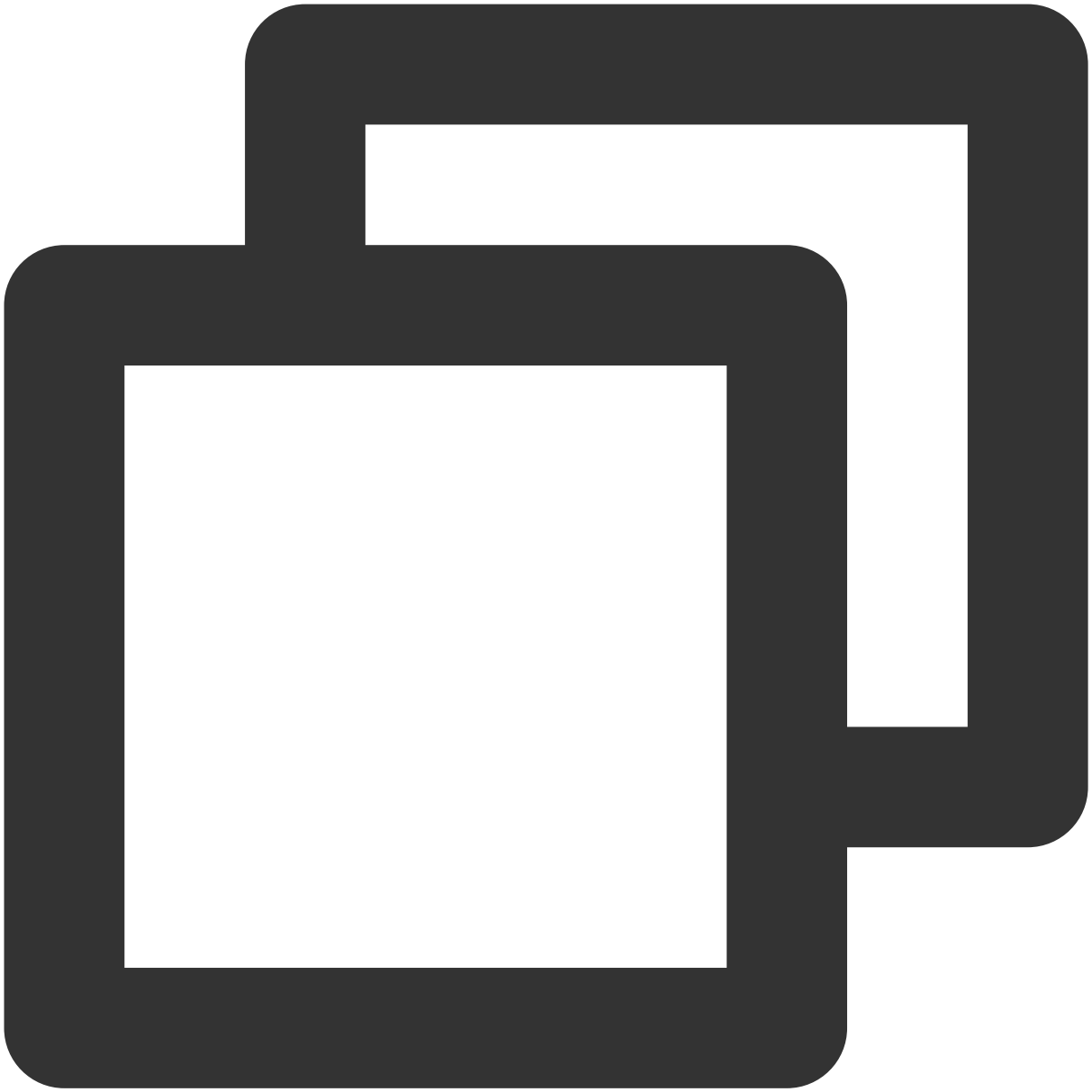
```
    <value>hadoopdev-clientcert.jks</value>
  </property>

  <property>
    <name>xasecure.policymgr.clientssl.truststore</name>
    <value>cacerts-xasecure.jks</value>
  </property>

  <property>
    <name>xasecure.policymgr.clientssl.keystore.credential.file</name>
    <value>jceks://file/tmp/keystore-hadoopdev-ssl.jceks</value>
  </property>

  <property>
    <name>xasecure.policymgr.clientssl.truststore.credential.file</name>
    <value>jceks://file/tmp/truststore-hadoopdev-ssl.jceks</value>
  </property>
</configuration>
```

3. 在 goosefs-site.properties 文件中，添加如下配置：



```
...
goosefs.security.authorization.permission.type=CUSTOM
goosefs.security.authorization.custom.provider.class=org.apache.ranger.authorizatio
...
```

4. 在 ${GOOSEFS_HOME}/libexec/goosefs-config.sh 中，将 goosefs-ranger-plugin-${version}.jar 添加到 GooseFS 的类路径中：



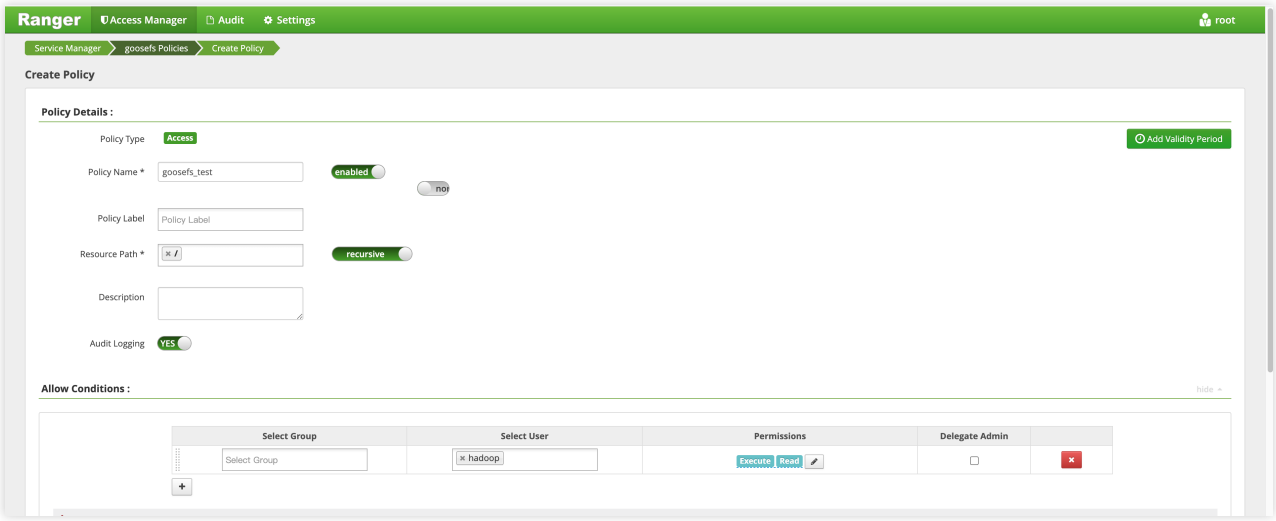
```
...
GOOSEFS_RANGER_CLASSPATH="${GOOSEFS_HOME}/lib/ranger-goosefs-plugin-${version}.jar"
GOOSEFS_SERVER_CLASSPATH=${GOOSEFS_SERVER_CLASSPATH}:${GOOSEFS_RANGER_CLASSPATH}
...
```

至此，所有配置完成。

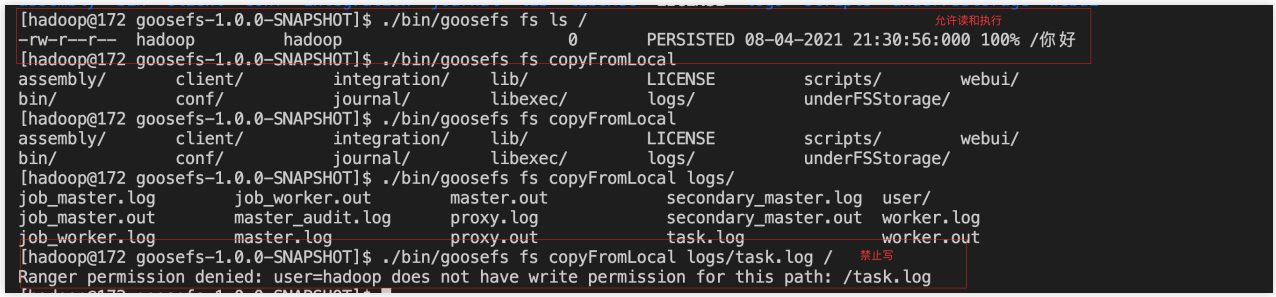
验证使用

例如，添加一条允许 hadoop 用户对 GooseFS 的根目录具备读取和执行权限，但是不允许写的策略，方法如下：

1. 添加策略，如下所示：



2. 策略添加成功后，对策略进行验证，即可看到策略已经生效，如下所示：



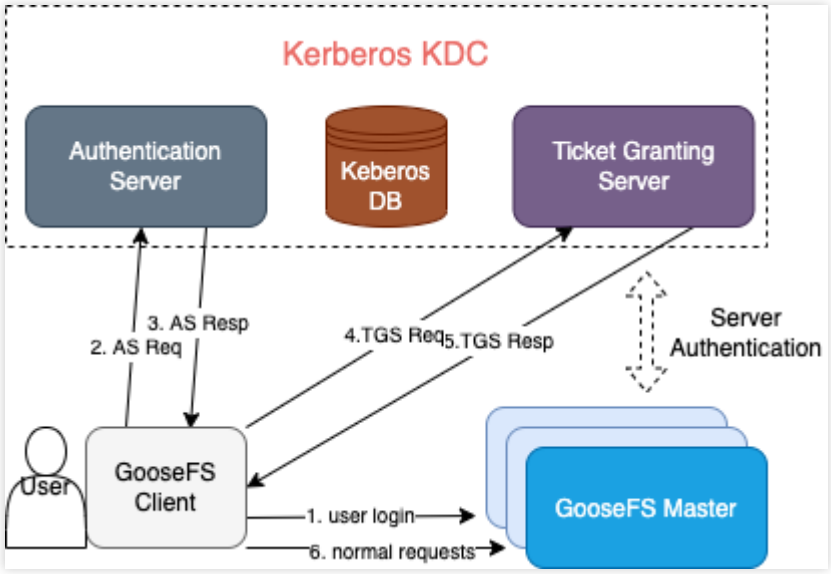
GooseFS 接入 Kerberos 认证

最近更新时间：2024-01-06 11:17:59

概述

Kerberos 是大数据生态系统中广泛应用的统一认证服务，GooseFS 作为大数据和数据湖场景下的加速存储服务，支持集群节点和用户访问接入 Kerberos 认证服务中。本文将详细介绍如何配置 GooseFS 接入 Kerberos 认证服务以及如何使用 Hadoop Delegation Token 认证。

GooseFS 接入 Kerberos 的认证架构



GooseFS Kerberos 认证优势

与 HDFS 接入 Kerberos 的认证架构和流程基本一致，在 HDFS 上启用了 Kerberos 认证流程的应用可以很容易地迁移到 GooseFS。
支持 Hadoop 的 Delegation Token 认证机制，因此可以很好地兼容 Hadoop 生态的应用作业。

配置 GooseFS 接入 Kerberos 认证

前提条件

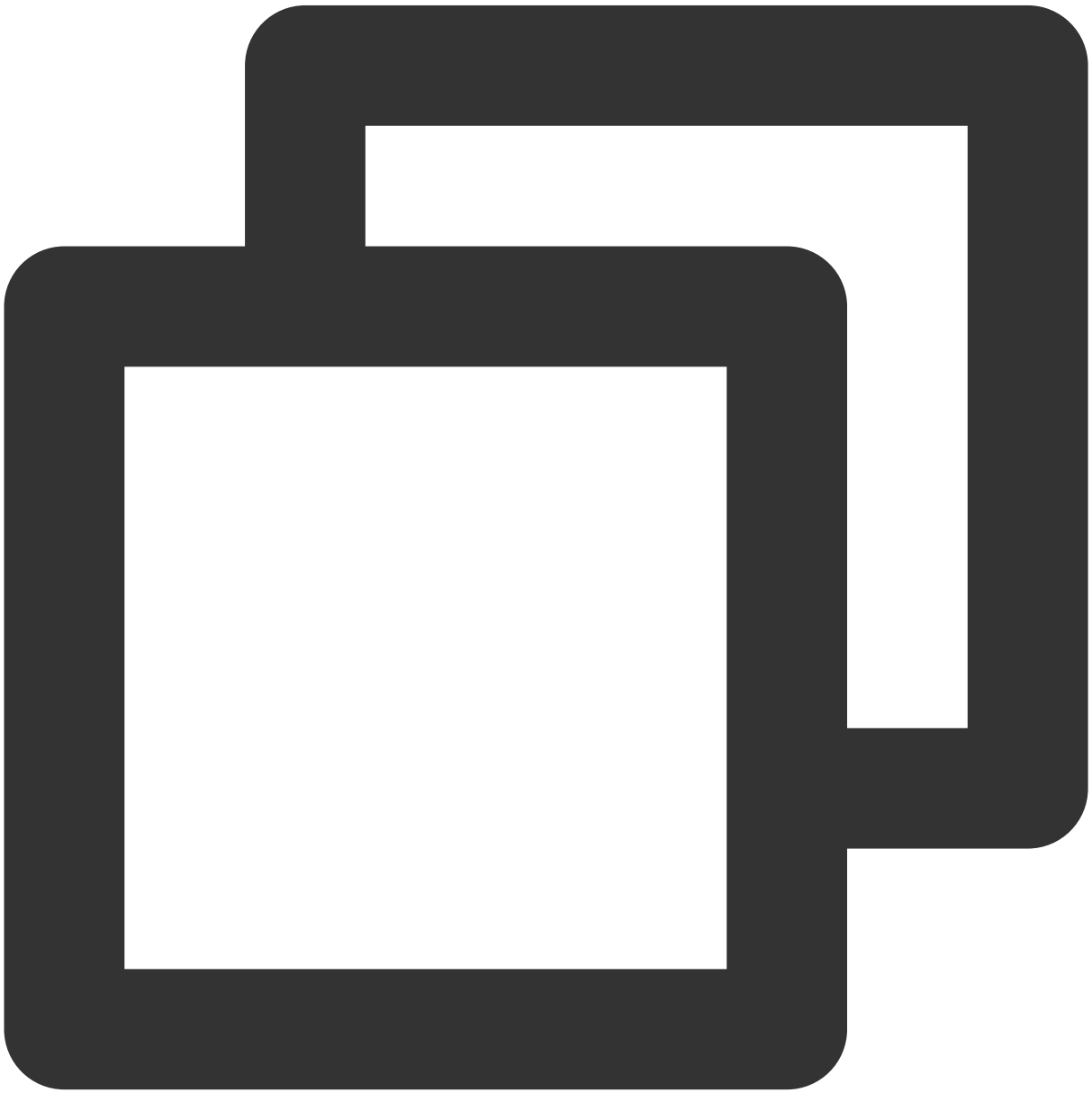
GooseFS 1.3.0 及以上版本。
确保环境中已存在 Kerberos KDC 服务，并且 GooseFS 以及应用客户端能够正常访问 Kerberos KDC 服务相关端口。

在 Kerberos KDC 中创建 GooseFS 相关的身份信息

首先，我们需要在 Kerberos KDC 中创建 GooseFS 集群 Server 与 Client 相关的 Kerberos 身份信息，然后才能继续后续的接入配置。这里我们在 **Kerberos KDC 服务器**上借助 `kadmin.local` 交互式工具来完成创建：

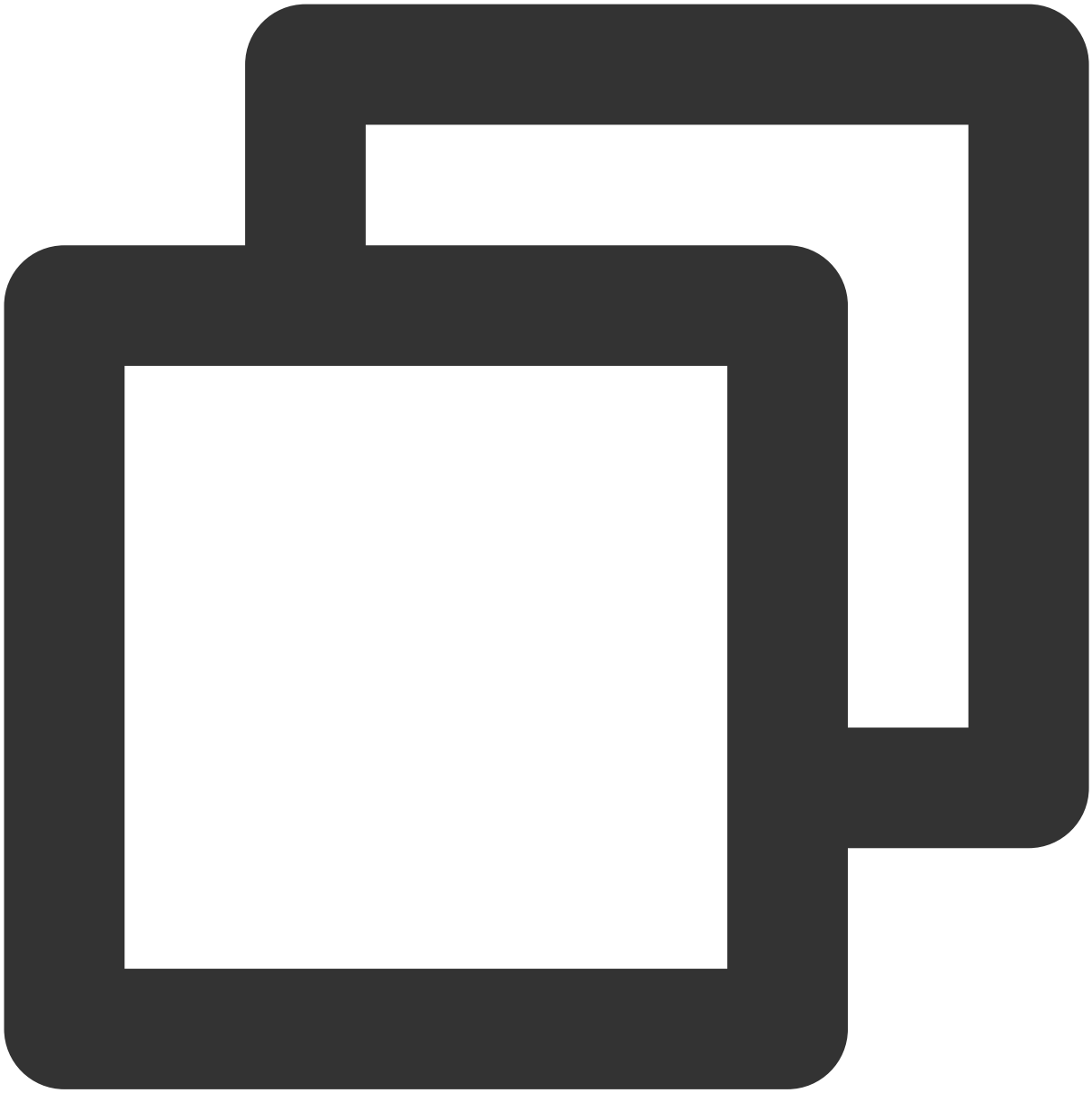
注意

`kadmin.local` 工具需要 root/sudo 权限执行。



```
$ sudo kadmin.local
```

如果执行成功，则进入了 kadmin 的交互式 shell 环境中：



```
Authenticating as principal root/admin@GOOSEFS.COM with password.
kadmin.local:
```

其中，`kadmin.local` 表示该交互式执行环境的命令提示符。

创建 GooseFS Server / Client 相关的身份信息

如下通过一个简单测试集群以及应用场景样例来介绍整个 Kerberos 的配置过程。

1. 集群环境说明

采用单 Master 双 Worker 的 Standalone 部署架构：

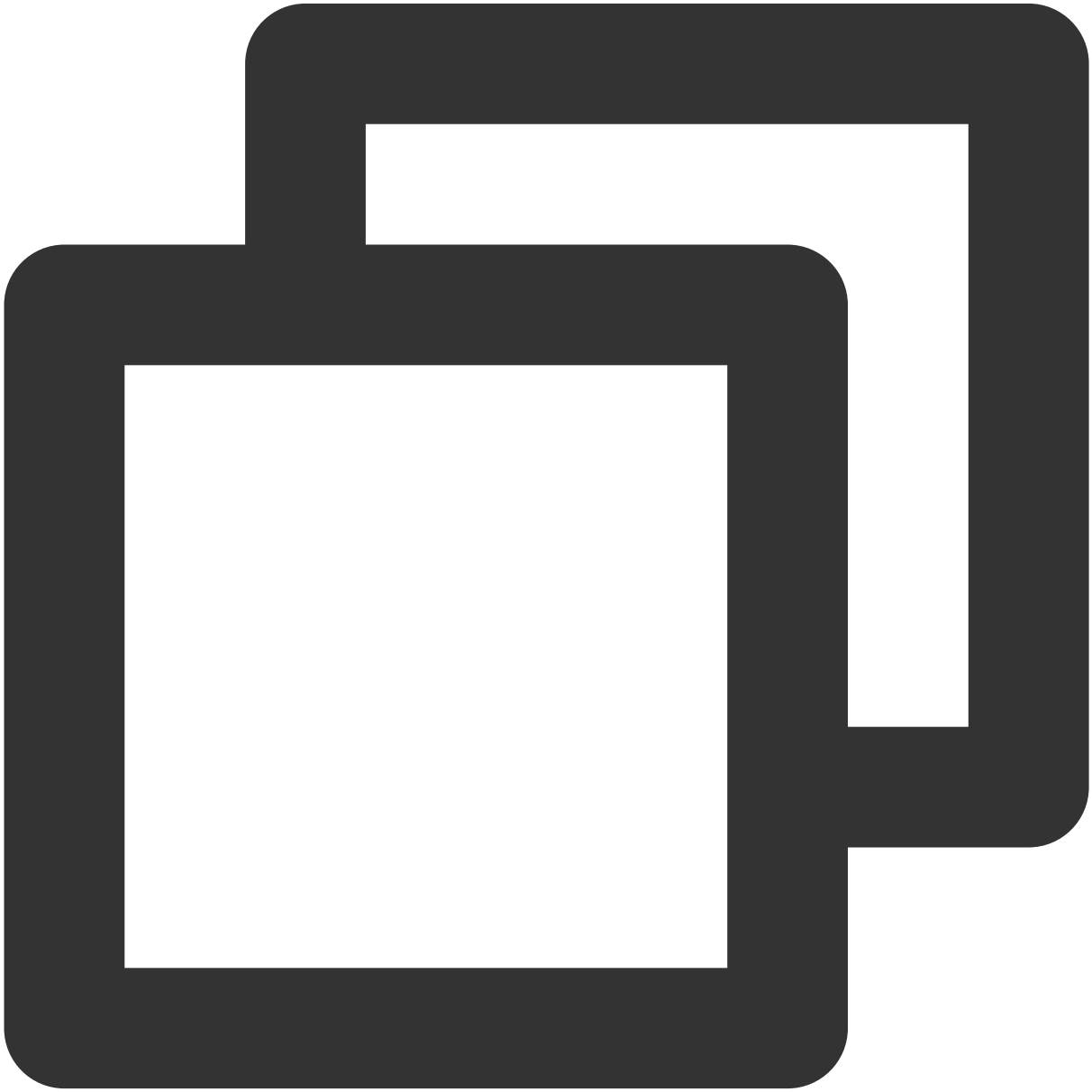
Master（JobMaster）：172.16.0.1

Worker1（JobWorker1）：172.16.0.2

Worker2（JobWorker2）：172.16.0.3

Client：172.16.0.4

2. 在 `kadmin.local` 中创建 Server 和 Client 身份认证相关信息：



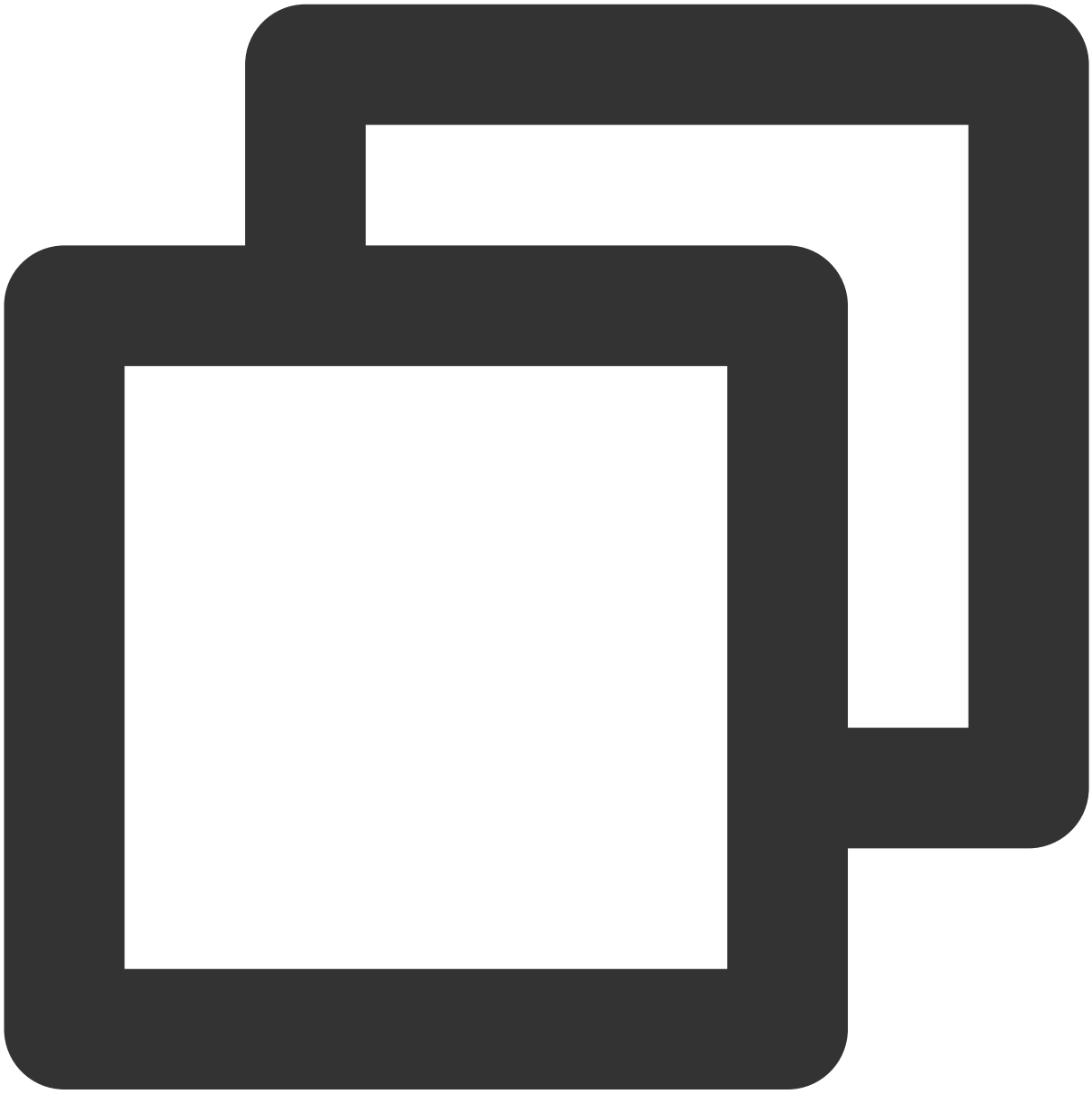
```
kadmin.local: addprinc -randkey goosefs/172.16.0.1@GOOSEFS.COM
kadmin.local: addprinc -randkey client/172.16.0.4@GOOSEFS.COM
```

版权所有：腾讯云计算（北京）有限责任公司

注意

此处使用 `-randkey` 选项的原因在于 GooseFS 无论 Server 还是 Client 登录均使用 `keytab` 文件认证，不使用明文密码。若身份信息需要用于 password 登录场景，则可以去掉该选项。

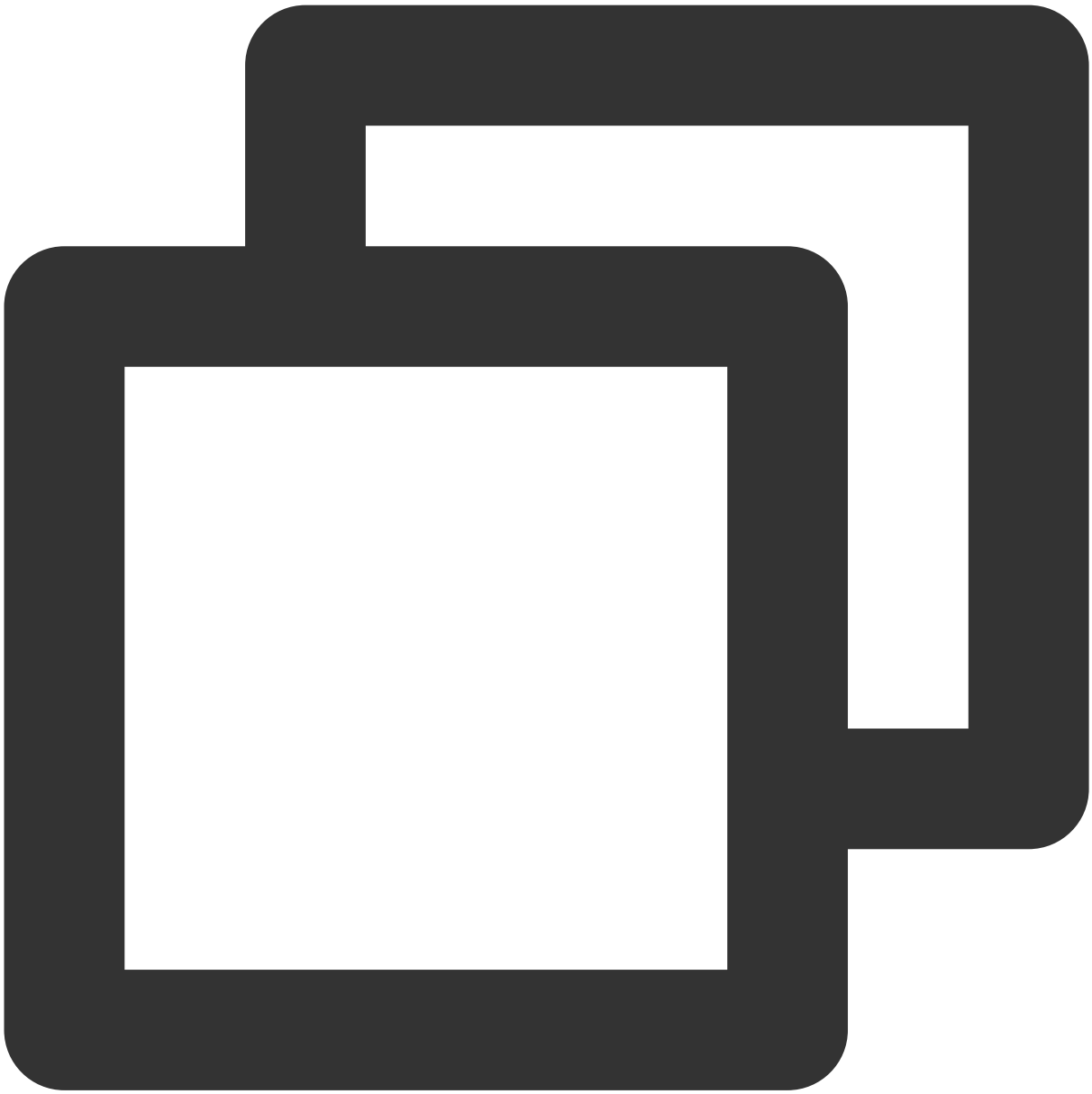
3. 生成导出每个身份对应的 `keytab` 文件：



```
kadmin.local: xst -k goosefs_172_16_0_1.keytab goosefs/172.16.0.1@GOOSEFS.COM
kadmin.local: xst -k client_172_16_0_4.keytab client/172.16.0.4@GOOSEFS.COM
```

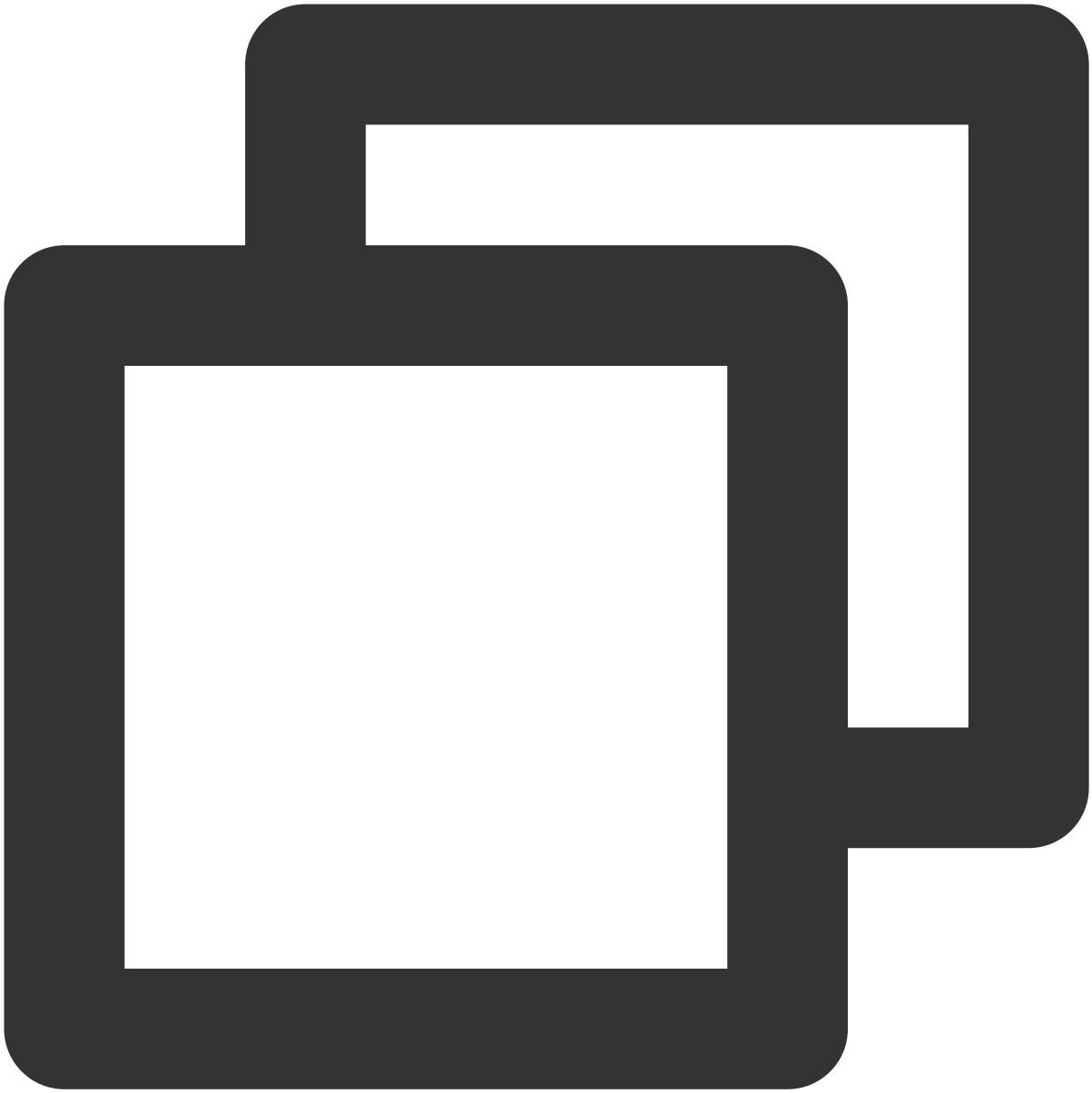
配置 GooseFS Server/Client 接入使用 Kerberos 认证

1. 将上述导出 `keytab` 文件分发到对应的机器上，此处建议的路径是 `${GOOSEFS_HOME}/conf/` ：



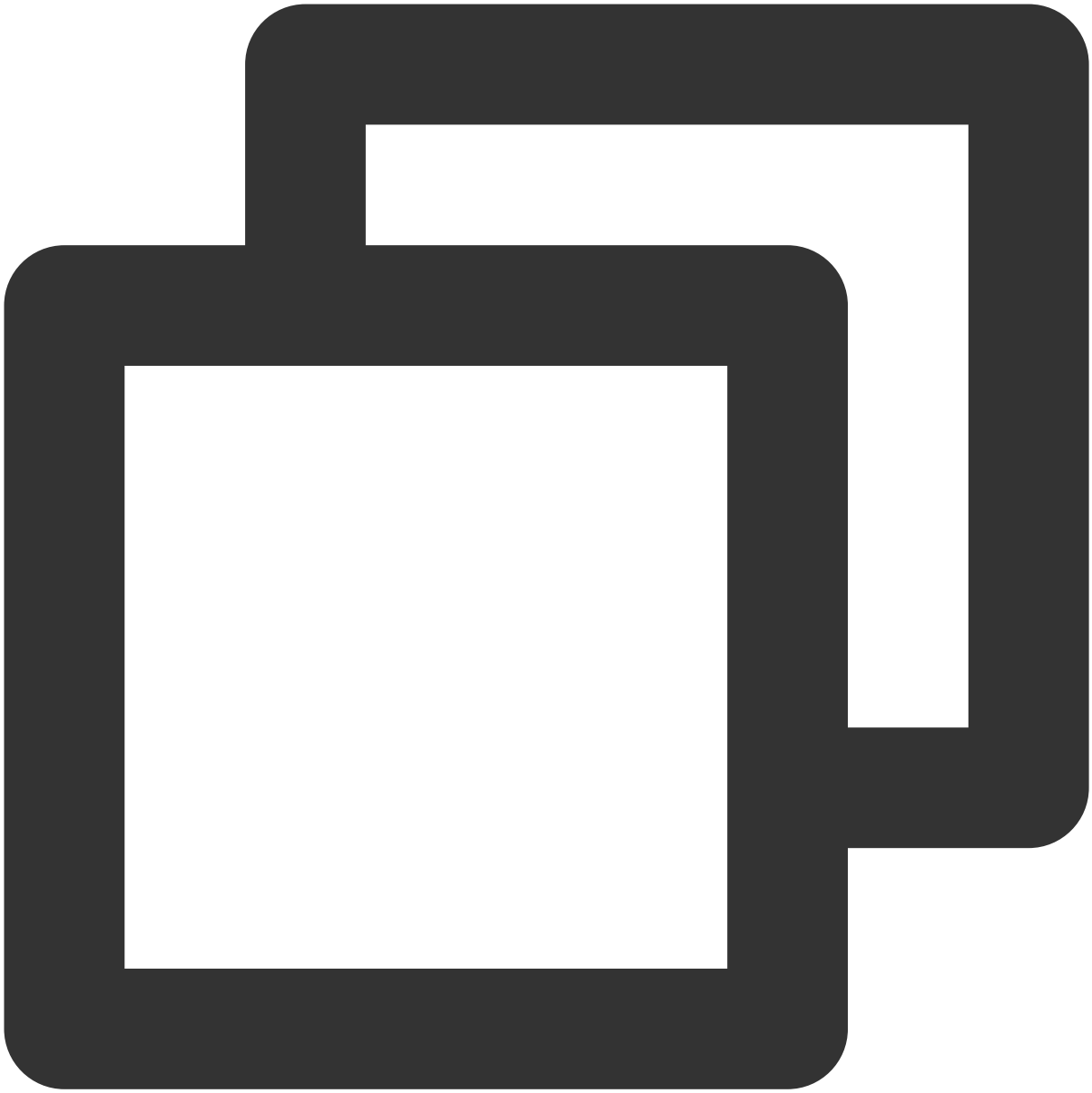
```
$ scp goosefs_172_16_0_1.keytab <username>@172.16.0.1:${GOOSEFS_HOME}/conf/
$ scp goosefs_172_16_0_1.keytab <username>@172.16.0.2:${GOOSEFS_HOME}/conf/
$ scp goosefs_172_16_0_1.keytab <username>@172.16.0.3:${GOOSEFS_HOME}/conf/
$ scp client_172_16_0_4.keytab <username>@172.16.0.4:${HOME}/.goosefs/
```

2. 在对应机器上，将 Server Principal KeyTab 文件的所属用户/用户组修改为 GooseFS Server 启动时所使用的用户及用户组（其目的是为了让 GooseFS 启动时有足够的权限读取）。



```
$ chown <GooseFS_USER>:<GOOSEFS_USERGROUP> goosefs_172_16_0_1.keytab
$ # 同时修改 Unix 访问权限位
$ chmod 0440 goosefs_172_16_0_1.keytab
```

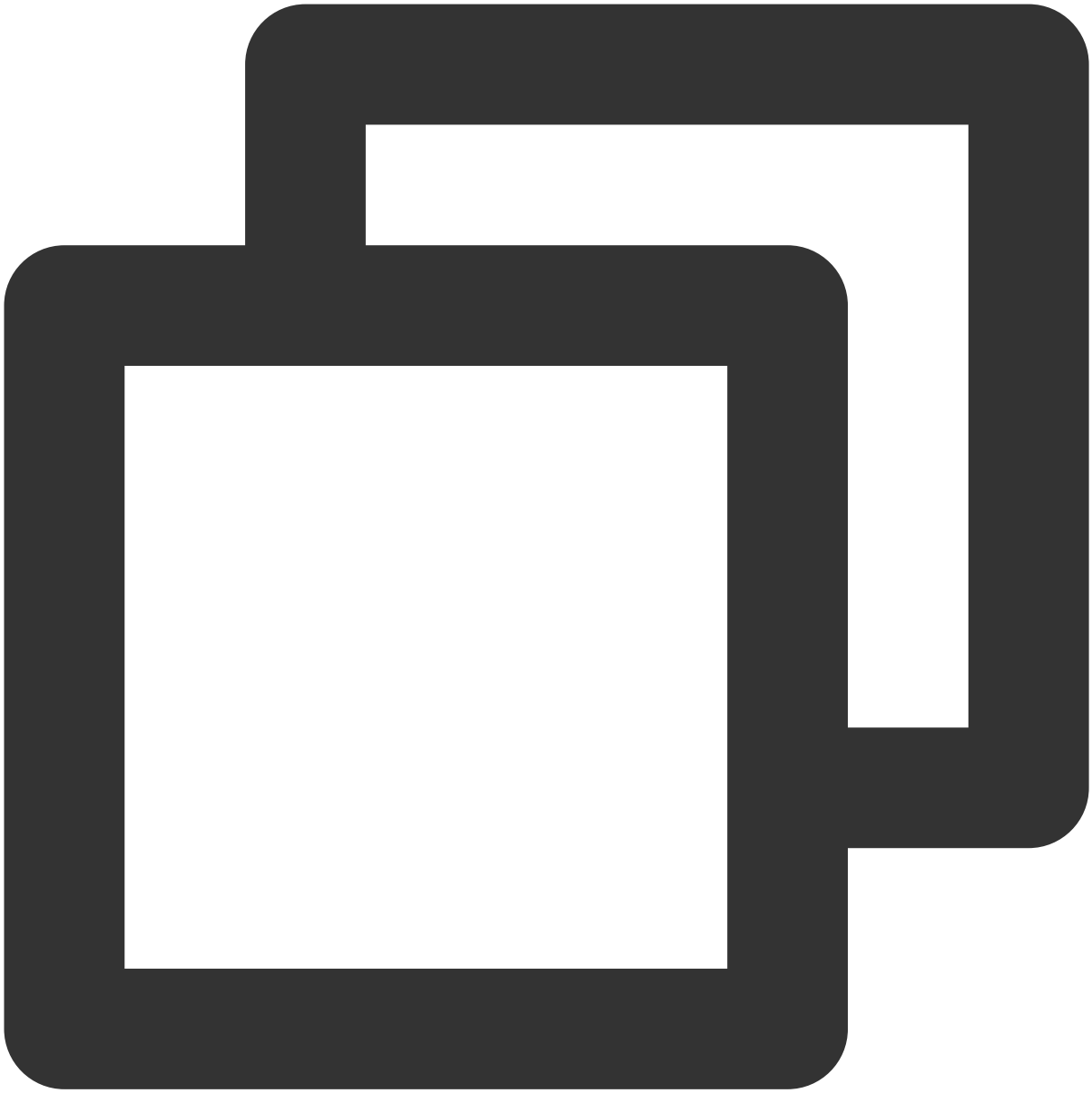
3. 将 Client 的 KeyTab 的所属用户/用户组修改为发起 GooseFS 请求的客户端账户。其目的同样是为了保证 Client 有足够的权限读取该文件。



```
$ chown <client_user>:<client_usergroup> client_172_16_0_4.keytab
$ # 同时修改 Unix 访问权限位
$ chmod 0440 client_172_16_0_4.keytab
```

Server 和 Client 端 GooseFS 配置

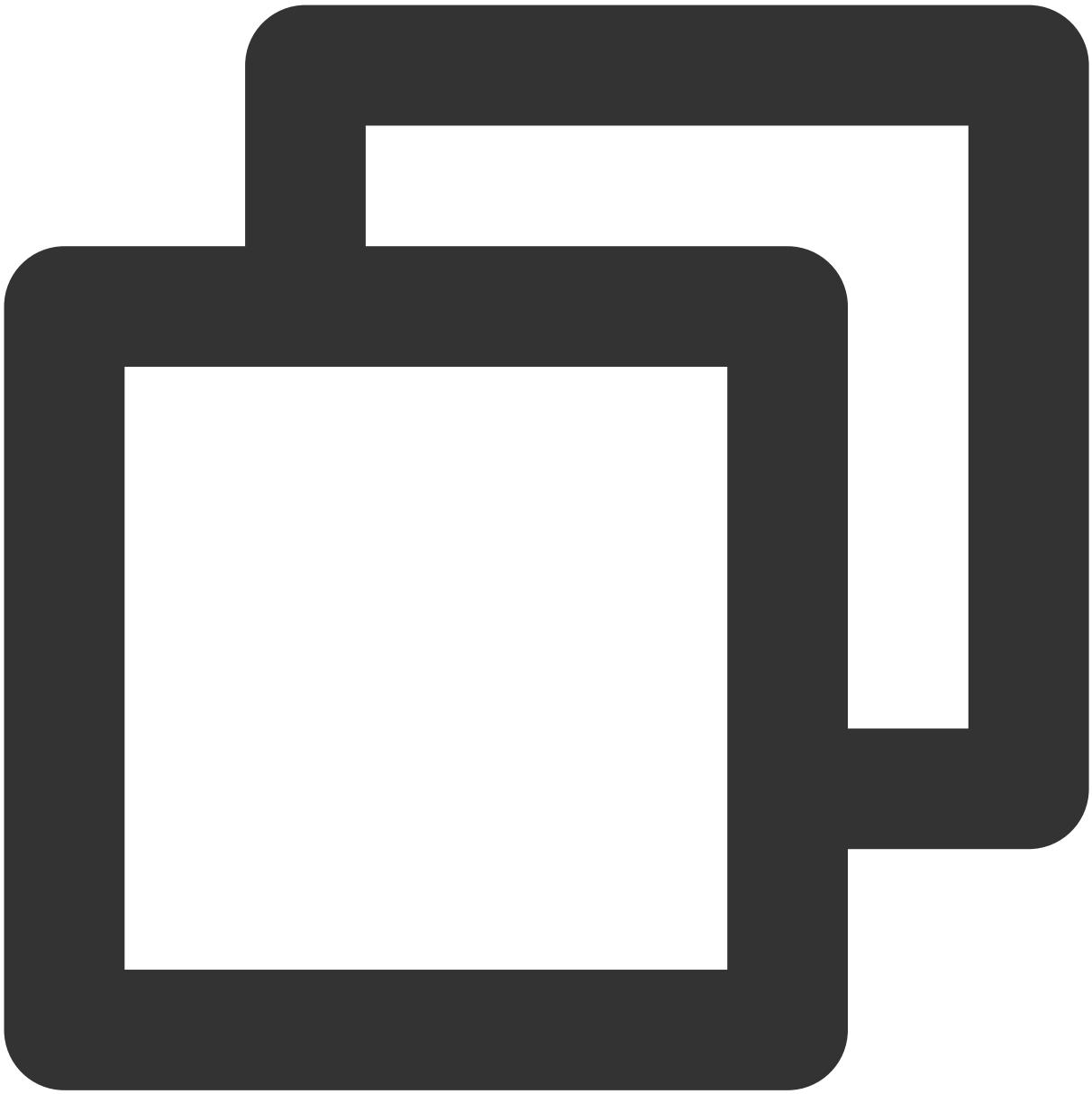
1. Master/Worker Server 的 goosefs-site.properties



```
# Security properties
# Kerberos properties
goosefs.security.authorization.permission.enabled=true
goosefs.security.authentication.type=KERBEROS
goosefs.security.kerberos.unified.instance.name=172.16.0.1
goosefs.security.kerberos.server.principal=goosefs/172.16.0.1@GOOSEFS.COM
goosefs.security.kerberos.server.keytab.file=${GOOSEFS_HOME}/conf/goosefs_172_16_0_
```

配置完成 GooseFS Server 端的认证配置后，重启整个集群以使得配置生效。

2. Client 的 goosefs-site.properties



```
# Security properties
# Kerberos properties
goosefs.security.authorization.permission.enabled=true
goosefs.security.authentication.type=KERBEROS
goosefs.security.kerberos.unified.instance.name=172.16.0.1
goosefs.security.kerberos.server.principal=goosefs/172.16.0.1@GOOSEFS.COM
goosefs.security.kerberos.client.principal=client/172.16.0.4@GOOSEFS.COM
goosefs.security.kerberos.client.keytab.file=${GOOSEFS_HOME}/conf/client_172_16_0_4
```

版权所有：腾讯云计算（北京）有限责任公司

注意

Client 端需要指定 Server 的 principal。其原因为在 Kerberos 认证体系中，KDC 需要知道当前 Client 所访问的 Service，而 GooseFS 是通过 Server 的 principal 区分 Client 当前请求的 Service。

至此，GooseFS 接入 Kerberos 的基础认证完毕，后续客户端发起的请求都将经过 Kerberos 进行身份认证。