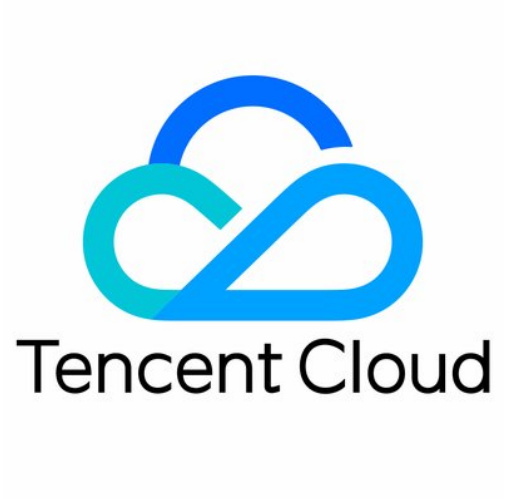
Cloud Object Storage

GooseFS

Product Documentation



Contents

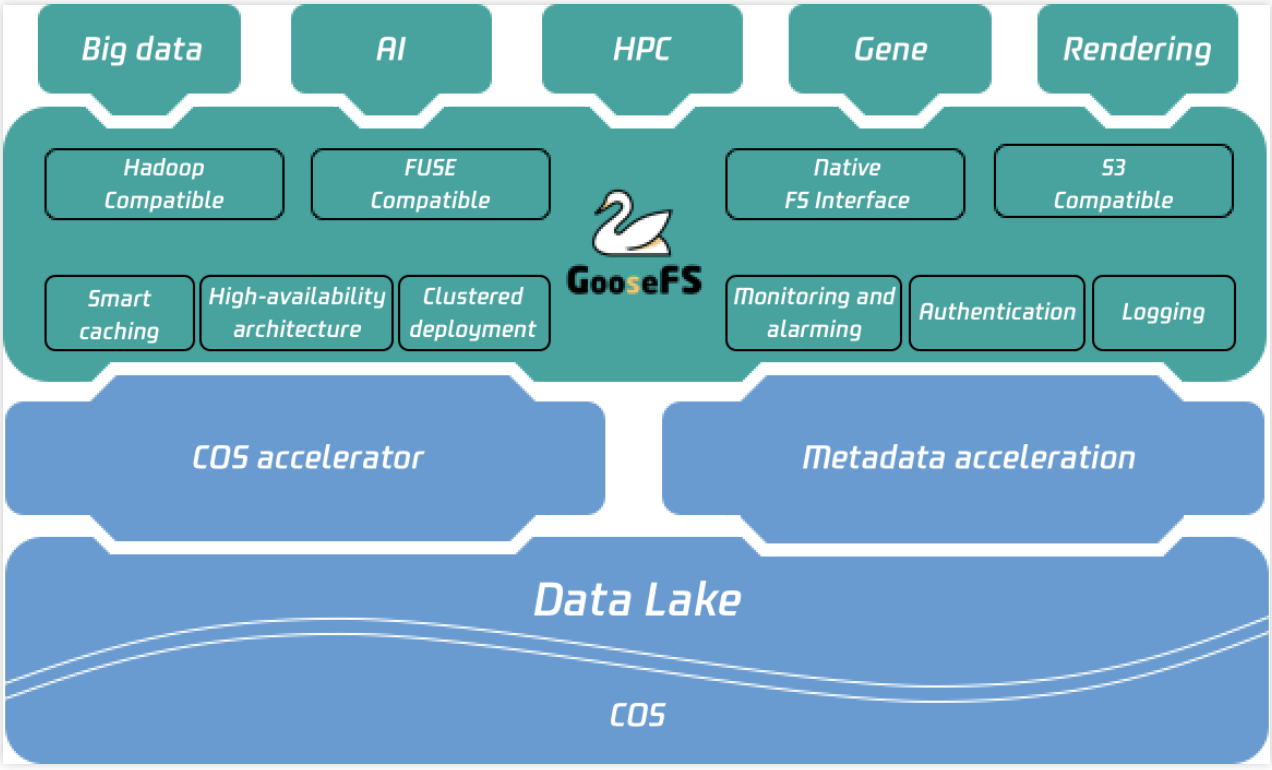
GooseFS Overview

Last updated：2024-03-25 16:04:01

Data Lake Accelerator Goose FileSystem (GooseFS) is a highly available, reliable, and elastic data lake acceleration service provided by Tencent Cloud. By using COS as the data lake storage base, it reduces the storage costs and offers a unified entry for computing applications in the data lake ecosystem, so as to accelerate the access of businesses such as big data analysis, machine learning, and AI. It uses a robust distributed cluster architecture to furnish a unified namespace and access protocol for upper-layer computing applications, making it easier for you to manage and transfer data across different storage systems.

Product Features

GooseFS aims to offer a one-stop cache solution and has native strengths in leveraging data locality, high-speed cache, and unified storage access syntax. It plays a core role in the data lake ecosystem as a connector between computing and storage as shown below:



GooseFS has the following features:

1. Cache acceleration and data locality: GooseFS can be deployed together with compute nodes to improve the data locality. Its high-speed cache feature can improve the storage performance and increase the bandwidth of data write to COS.
2. Integrated storage syntax: GooseFS provides unified file system (UFS) syntax to support the storage syntax of multiple storage services, such as COS, Hadoop, S3, K8s CSI, and FUSE, making it suitable for various ecosystems and use cases.
3. Tencent Cloud ecosystem services: Log, authentication, and monitoring services are available to enable the same operations as in COS.
4. Namespace management capabilities: GooseFS comes with different cache read/write and TTL management policies for different businesses and under file systems.
5. Metadata table sensing: GooseFS Catalog can sense metadata table in big data scenarios to prefetch cache at the table level.

Benefits

GooseFS has the following strengths in data lake scenarios:

Data I/O performance

GooseFS enables a distributed shared cache near compute nodes, where upper-layer computing applications can transparently and efficiently cache the hot data that needs to be accessed frequently to accelerate the data I/O. It also has the metadata cache feature to make metadata operations in big data scenarios faster, such as file data query and file list display. Together with big data buckets, it can further expedite file renaming. In addition, it allows you to choose different storage media, including MEM, SSD, NVMe, and HDD, based on your business needs to balance the business costs and data access performance.

Integrated storage

GooseFS provides a unified namespace that supports the storage syntax of not only COS but also many other storage services such as HDFS, K8s CSI, and FUSE. This furnishes a unified integrated storage solution for upper-layer businesses and simplifies the business Ops configuration. Integrated storage breaks the barriers between different data bases and makes it easier for upper-layer applications to manage and transfer data, thus improving the data usage efficiency.

Ecosystem affinity

GooseFS is fully compatible with the Tencent Cloud big data platform framework and can also be deployed on premises in a custom manner, showing an excellent ecosystem affinity. For example, you can use GooseFS in EMR to accelerate big data businesses and conveniently deploy it in CVM or self-built IDC. In addition, it supports transparent acceleration. If you have already activated COSN and CHDFS, you can simply modify the configurations to

automatically accelerate COSN and CHDFS business access via GooseFS with no need to modify business code and access paths.

Change Records

Last updated：2024-03-25 16:04:01

The update history for GooseFS is described as follows. If you have any queries or suggestions, please contact us.

Version	Update Date	Description	Download URL
1.4.1	March 1, 2023	Optimizations: Supported clearing and viewing the list of incomplete files. Optimized the locking granularity for the recursive metadata loading operation (loadmetadata -R). Bug fixes: Fixed the issue where the status of the authentication flow and data flow was inconsistent during Flume data write. Fixed the issue where deadlock occurred after writing large files exhausted client resources.	GooseFS: Click to download GooseFS (Kona JDK Edition): Click to download GooseFS client: Click to download GooseFS client (Kona JDK Edition): Click to download GooseFS-FUSE client: Click to download GooseFS-FUSE client (Kona JDK Edition): Click to download
1.4.0	November 11, 2022	New features: Provided the file decompression feature, which is in beta test and available in Shanghai Auto-Driving Cloud Zone only. Supported the temporary key management feature. Supported hierarchical traversal for `distributedLoad`. Supported downgraded read in GooseFS-FUSE. Optimizations: Supported hierarchical traversal capabilities for distributedLoad in GooseFS to recursively pull the metadata in the specified directory. Improved the sequential read performance of GooseFS-FUSE for large files. Added the percentile duration monitoring for master query and RocksDB update to improve the monitoring sensitivity of the metadata service. Reduced the cluster recovery time of GooseFS in HA mode to improve the cluster availability. Updated the COSN dependency version. You can access buckets with metadata acceleration enabled	GooseFS: Click to download GooseFS (Kona JDK Edition): Click to download GooseFS client: Click to download GooseFS client (Kona JDK Edition): Click to download GooseFS-FUSE client: Click to download GooseFS-FUSE client (Kona JDK Edition): Click to download

		over the native HDFS protocol, which improves the file operation performance in big data scenarios. Simplified and optimized GooseFS configuration by removing unnecessary configuration items. Simplified and optimized `listInfo` to improve the file listing performance. Bug fixes: Fixed the issue where workers received a high number of invalid async block requests. Optimized the processing logic of orphan blocks during worker reporting.	
1.3.0	July 25, 2022	New features: Supported Kerberos authentication. Supported access to buckets with metadata acceleration enabled by using native POSIX semantics. Supported the LRU algorithm for the master node cache elimination policy to avoid frequent cache replacement. Supported expired metadata cleanup for the master node. Optimizations: Optimized the lock bottleneck problem for the GooseFS master node to greatly improve the master QPS and increase the performance by around 35%. Supported concurrent formatting to improve the performance of GooseFS worker nodes. Supported overwrite operations for the GooseFS-FUSE client. Optimized the memory usage of the ls command in the GooseFS-FUSE client. Optimized the performance of ListNamespace in the GooseFS HDFS client. Bug fixes: Fixed RocksDB leaks and Core issues to avoid memory leaks. Fixed the issue where ZooKeeper Curator mistakenly printed logs. Fixed the issue of inaccurate UFS bandwidth reading. Fixed the LostWorker issue caused by excessive log printing during DistributedLoad.	GooseFS: Click to download GooseFS (Kona JDK Edition): Click to download GooseFS client: Click to download GooseFS client (Kona JDK Edition): Click to download GooseFS-FUSE client: Click to download GooseFS-FUSE client (Kona JDK Edition): Click to download
1.2.0	January 25, 2022	New features: Supported using GooseFS to upload logs to CLS.	GooseFS: Click to download

		Supported separated configuration of RocksDB by using Inode and Block metadata. Added the hot switch capability in the GooseFS client to improve disaster recovery measures. Added the infrastructure for certain full-linkage logs. Optimizations: Optimized the high Raft failover latency. Optimized the memory usage of the GooseFS HCFS client. Bug fixes: Fixed the issue of journal disorder. Fixed the gRPC issue caused by Ratis deadlock. Fixed the incorrect HDFSUnderFileSystemFactory loading position. Fixed the Log4j2 vulnerabilities. Fixed ufsPath prefix check errors.	GooseFS (Kona JDK Edition): Click to download GooseFS-FUSE client: Click to download
1.1.0	September 1, 2021	New features: Supported Ranger authentication . Supported concurrent listing. Supported DistributedLoad directory atomicity. Supported the namespace cache allowlist. Supported imperceptible OFS scheme acceleration. Optimizations: Added help commands for each subcommand in CLI. Optimized the logic to check for namespace existence during table mounting. Improved the monitoring of job workers and worker metrics. Bug fixes: Fixed the issue of DistributedLoad read bloat.	This version is no longer maintained. Download the latest version.
1.0.0	June 1, 2021	Namespace-based read/write policy and TTL management. Prefetch at Hive table and table partition levels. Compatibility with paths of existing COSN users to achieve imperceptible acceleration. Fluid integration with GooseFS. CHDFS support integration.	This version is no longer maintained. Download the latest version.

Getting Started

Last updated：2024-03-25 16:04:01

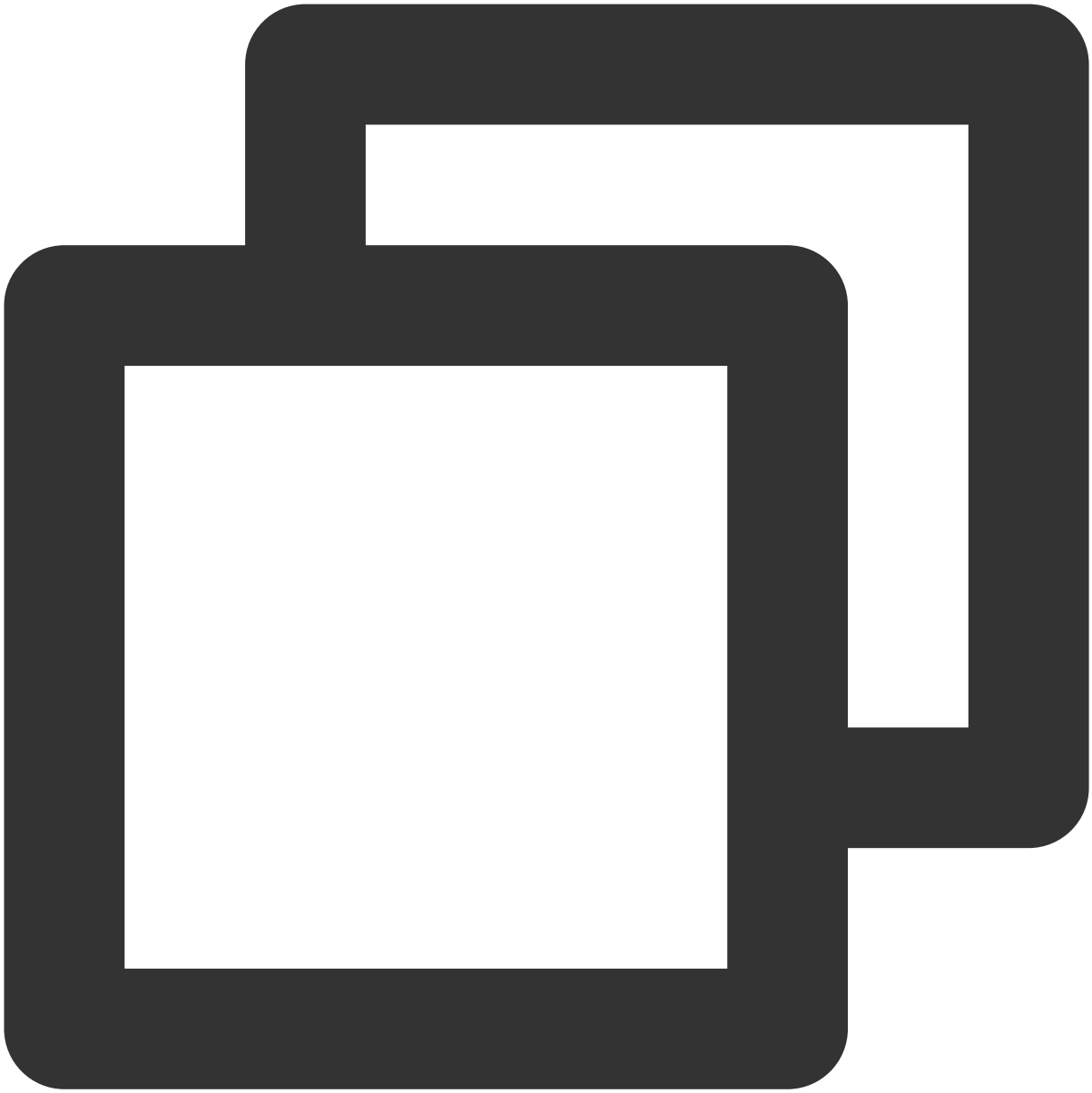
This document describes how to quickly deploy GooseFS on a local device, perform debugging, and use COS as remote storage.

Prerequisites

Before using GooseFS, you need to:
1. Create a bucket in COS as remote storage. For detailed directions, please see Getting Started With the Console.
2. Install Java 8 or a later version.
3. Install SSH, ensure that you can connect to the LocalHost using SSH, and log in remotely.
4. Purchase a CVM instance as instructed in Getting Started and make sure that the disk has been mounted to the instance.

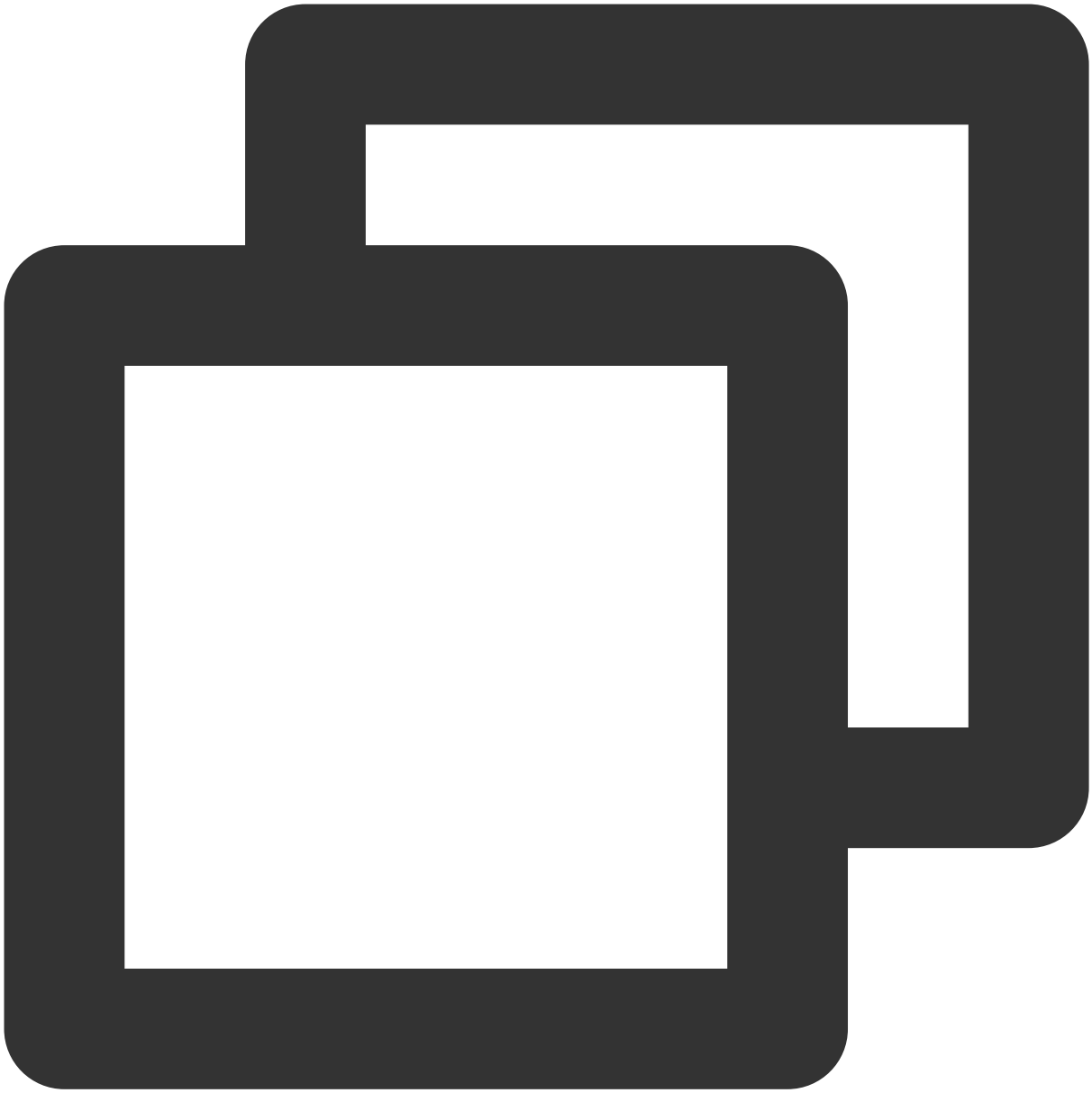
Downloading and Configuring GooseFS

1. Create and enter a local directory (you can also choose another directory as needed), and then download goosefs-1.4.2-bin.tar.gz。



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

2. Run the following command to decompress the installation package and enter the extracted directory:

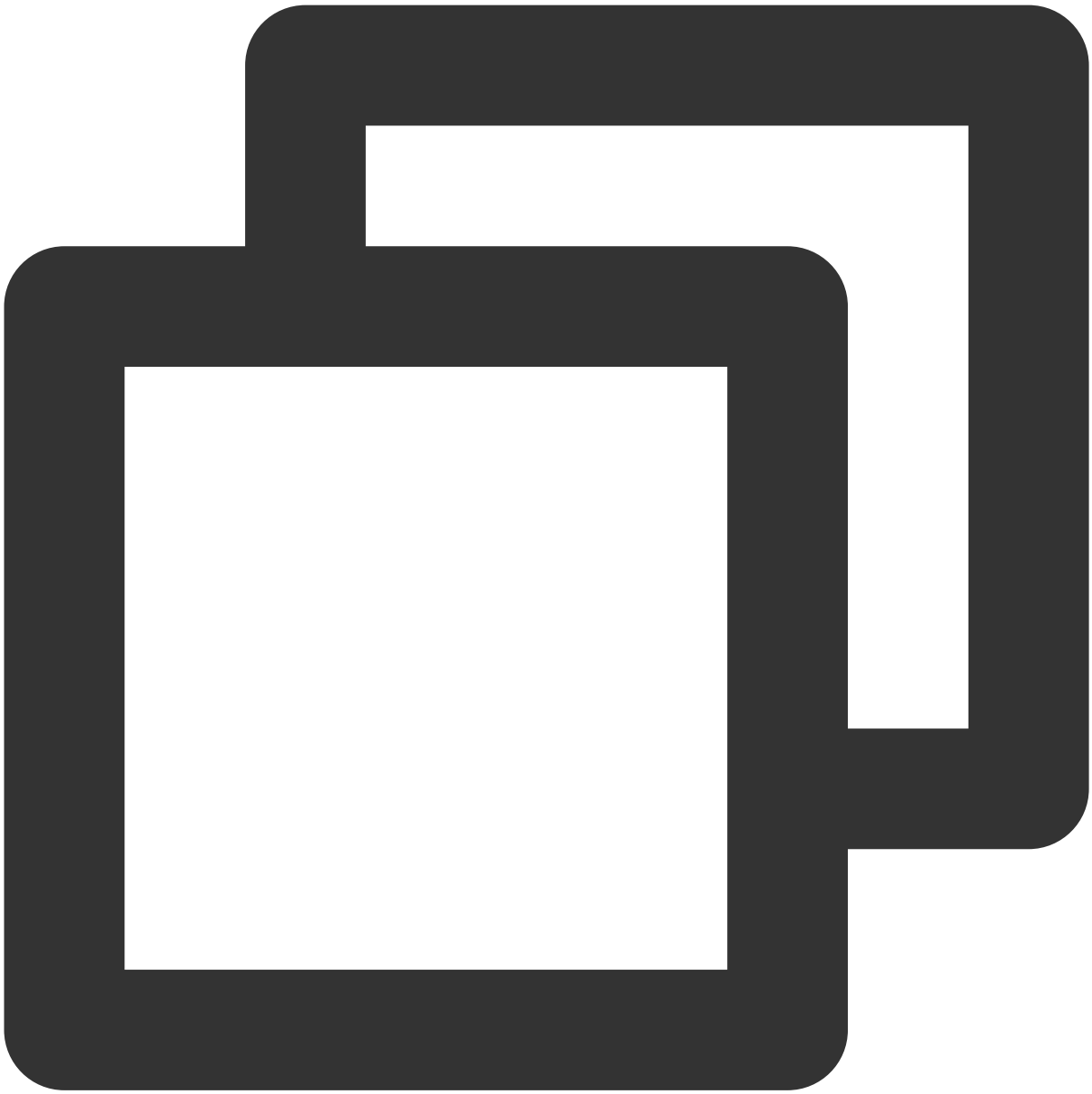


```
$ tar -zxvf goosefs-1.4.2-bin.tar.gz
$ cd goosefs-1.4.2
```

After the decompression, the home directory of GooseFS `goosefs-1.4.2` will be generated. This document uses `${GOOSEFS_HOME}` as the absolute path of this home directory.

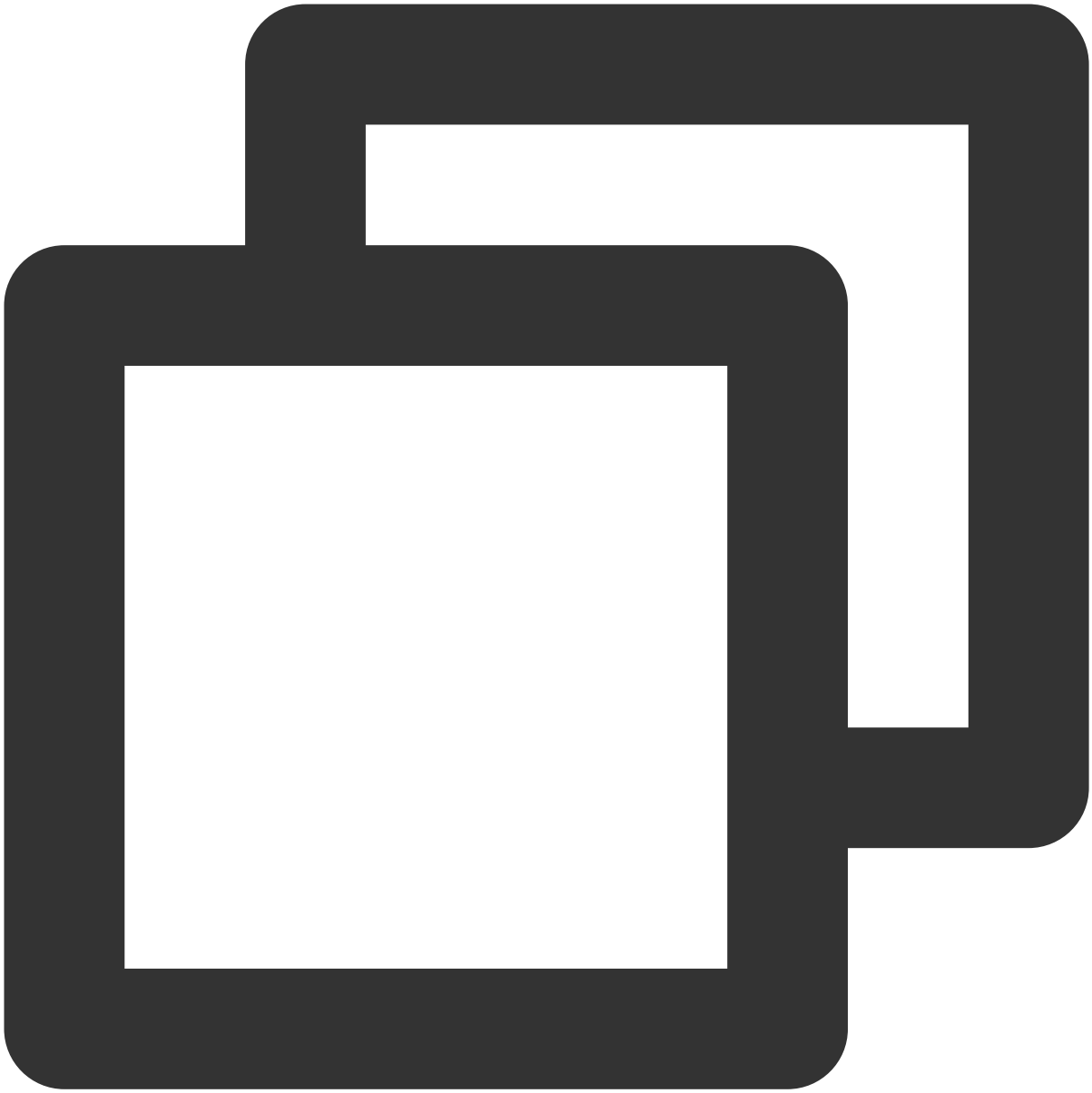
3. Create the `conf/goosefs-site.properties` configuration file in `${GOOSEFS_HOME}/conf`. GooseFS provides configuration templates for AI and big data scenarios, and you can choose an appropriate one as needed. Then, enter the editing mode to modify the configuration:

(1) Use the AI template. For more information, see GooseFS Configuration Practice for a Production Environment in the AI Scenario.



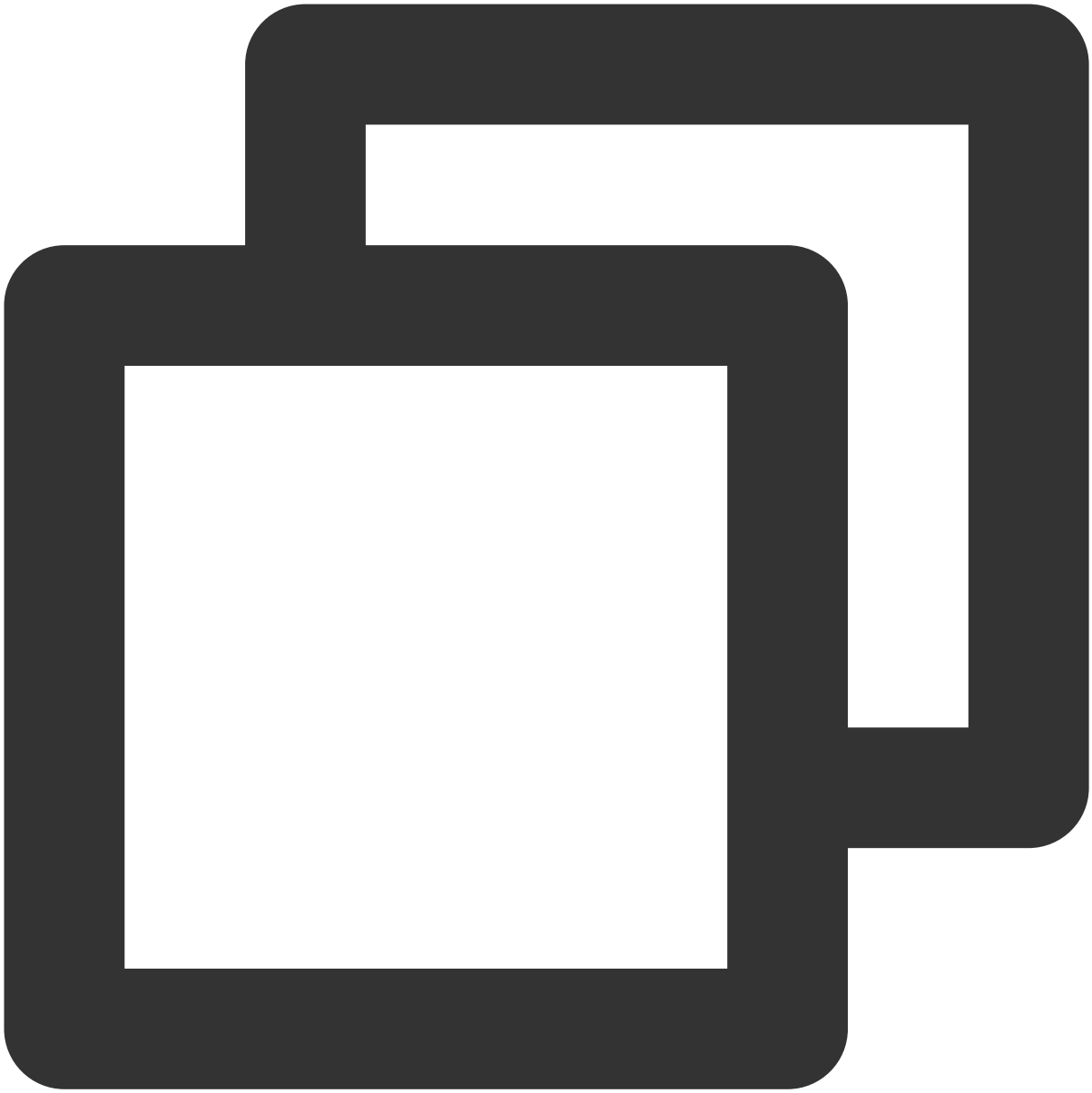
```
$ cp conf/goosefs-site.properties.ai_template conf/goosefs-site.properties
$ vim conf/goosefs-site.properties
```

(2) Use the big data template. For more information, see GooseFS Configuration Practice for a Production Environment in the Big Data Scenario.



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

4. Modify the following configuration items in the configuration file `conf/goosefs-site.properties` :



```
# Common properties
# Modify the master node's host information
goosefs.master.hostname=localhost
goosefs.master.mount.table.root.ufs=${goosefs.work.dir}/underFSStorage

# Security properties
# Modify the permission configuration
goosefs.security.authorization.permission.enabled=true
goosefs.security.authentication.type=SIMPLE

# Worker properties
```

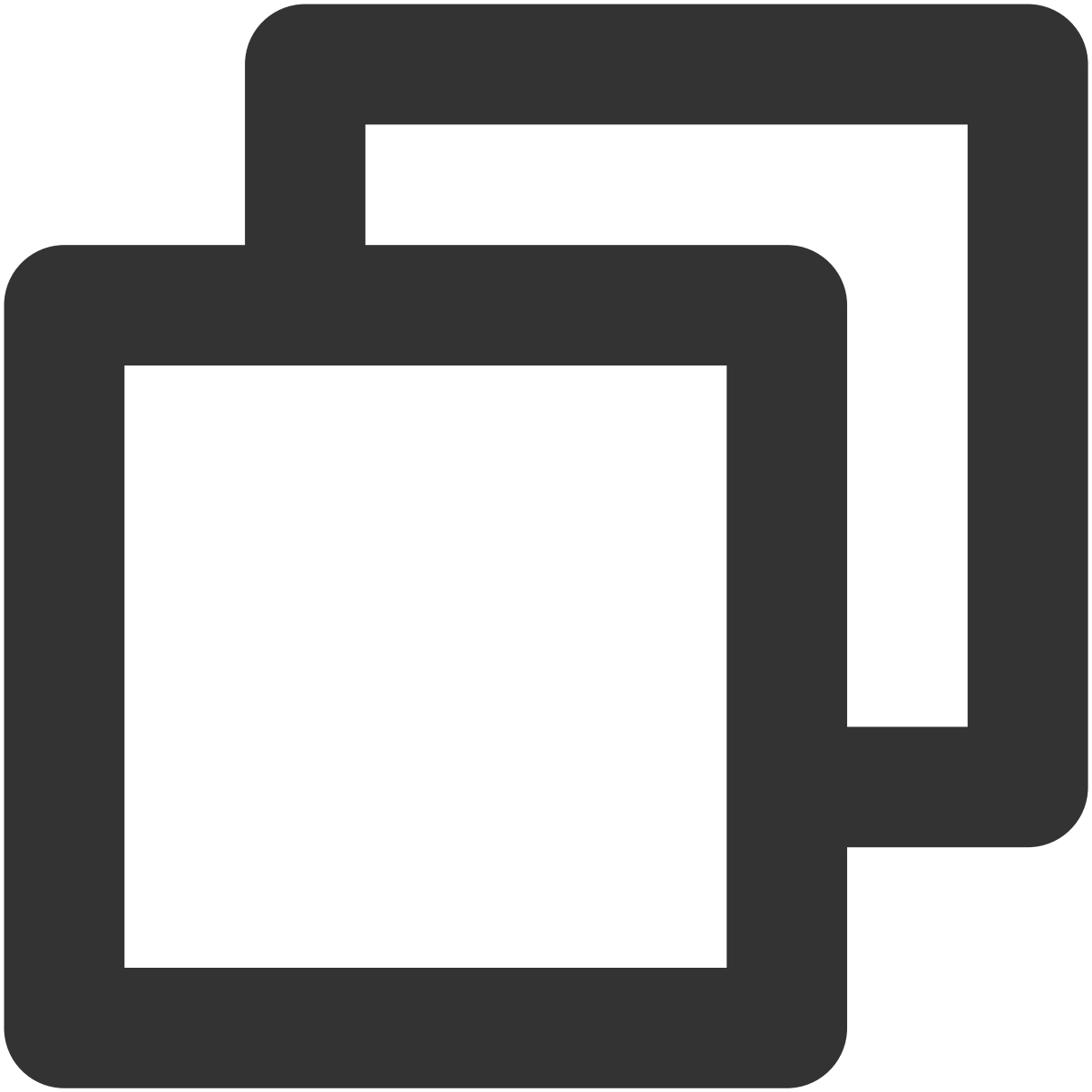
```
# Modify the worker node configuration to specify the local cache medium, cache pat
goosefs.worker.ramdisk.size=1GB
goosefs.worker.tieredstore.levels=1
goosefs.worker.tieredstore.level0.alias=SSD
goosefs.worker.tieredstore.level0.dirs.path=/data
goosefs.worker.tieredstore.level0.dirs.quota=80G


# User properties
# Specify the cache policies for file reads and writes
goosefs.user.file.readtype.default=CACHE
goosefs.user.file.writetype.default=MUST_CACHE
```

Note:

Before configuring the path parameter `goosefs.worker.tieredstore.level0.dirs.path` , you need to create the path first.

Running GooseFS

1. Before starting GooseFS, you need to enter the GooseFS directory and run the startup command:



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

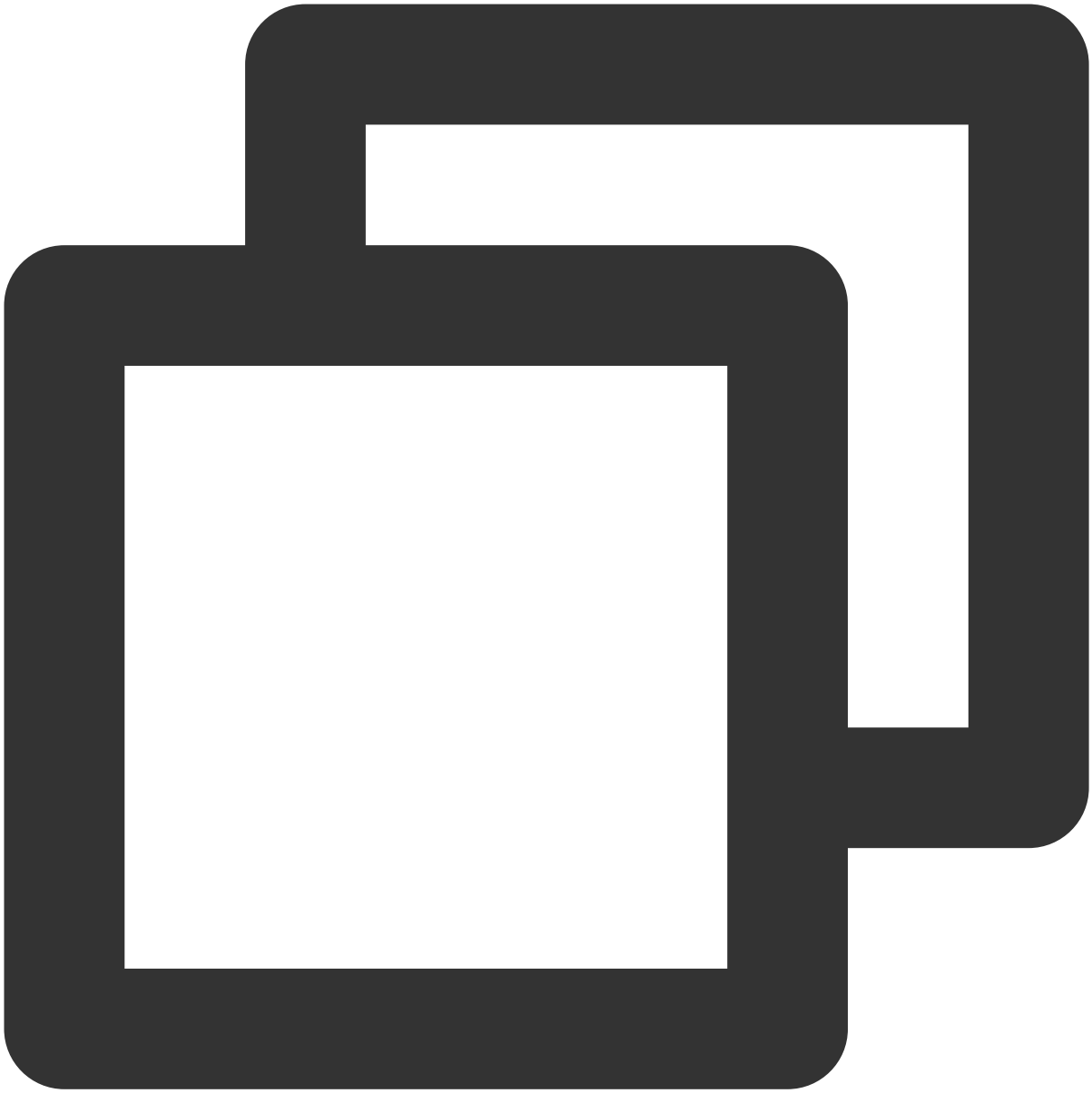
After running this command, you can see the following page:

```
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]#
```

After the command above is executed, you can access `http://localhost:9201` and `http://localhost:9204` to view the running status of the master and the worker, respectively.

Mounting COS or Tencent Cloud HDFS to GooseFS

To mount COS or Tencent Cloud HDFS to the root directory of GooseFS, configure the required parameters of COSN/CHDFS (including but not limited to `fs.cosn.impl`, `fs.AbstractFileSystem.cosn.impl`, `fs.cosn.userinfo.secretId`, and `fs.cosn.userinfo.secretKey`) in `conf/core-site.xml`, as shown below:



```
<!-- 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>
```

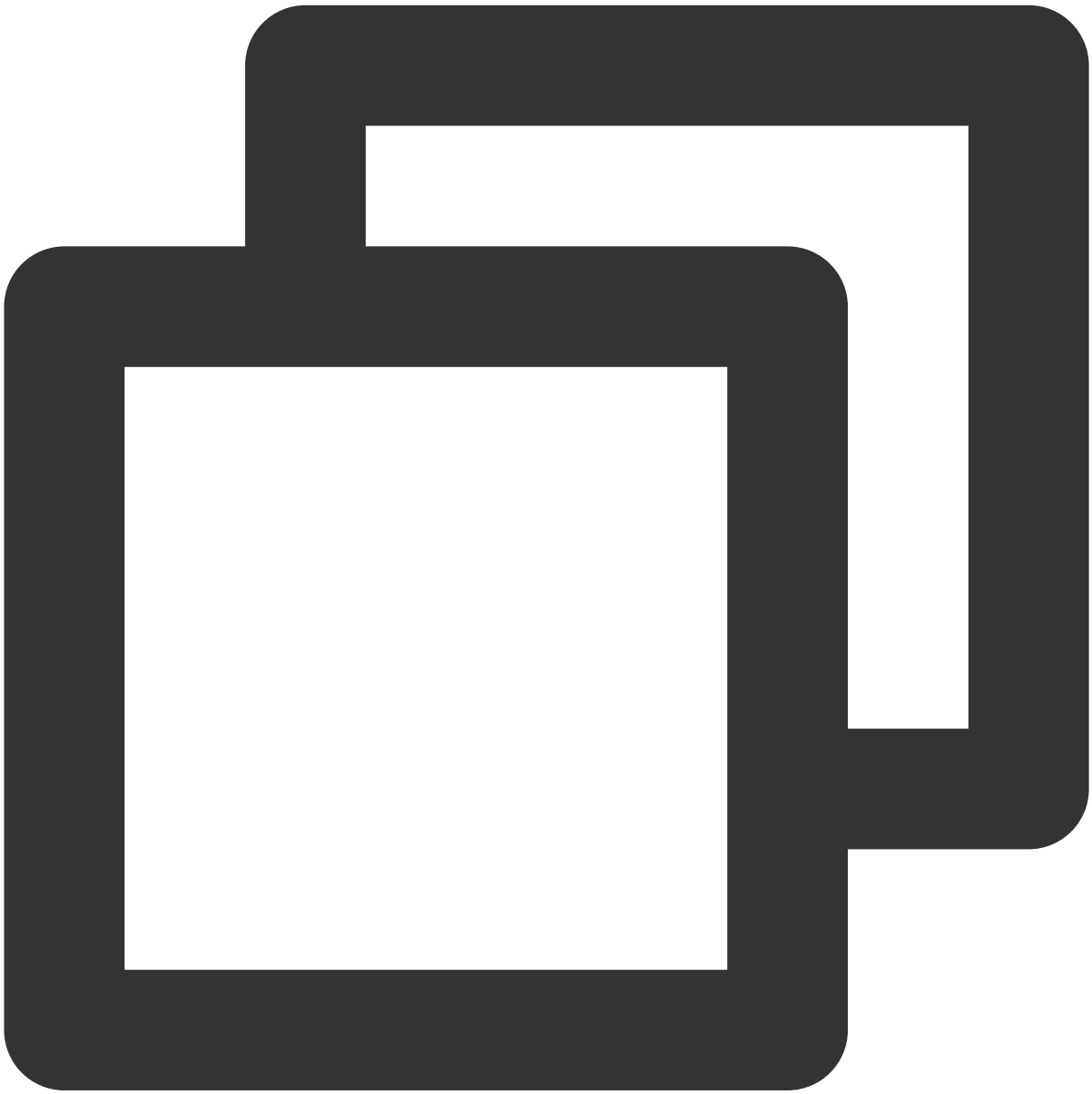
Note:

For the complete configuration of COSN, please see Hadoop.

For the complete configuration of CHDFS, see Mounting CHDFS Instance.

The following describes how to create a namespace to mount COS or CHDFS.

1. Create a namespace and mount COS:



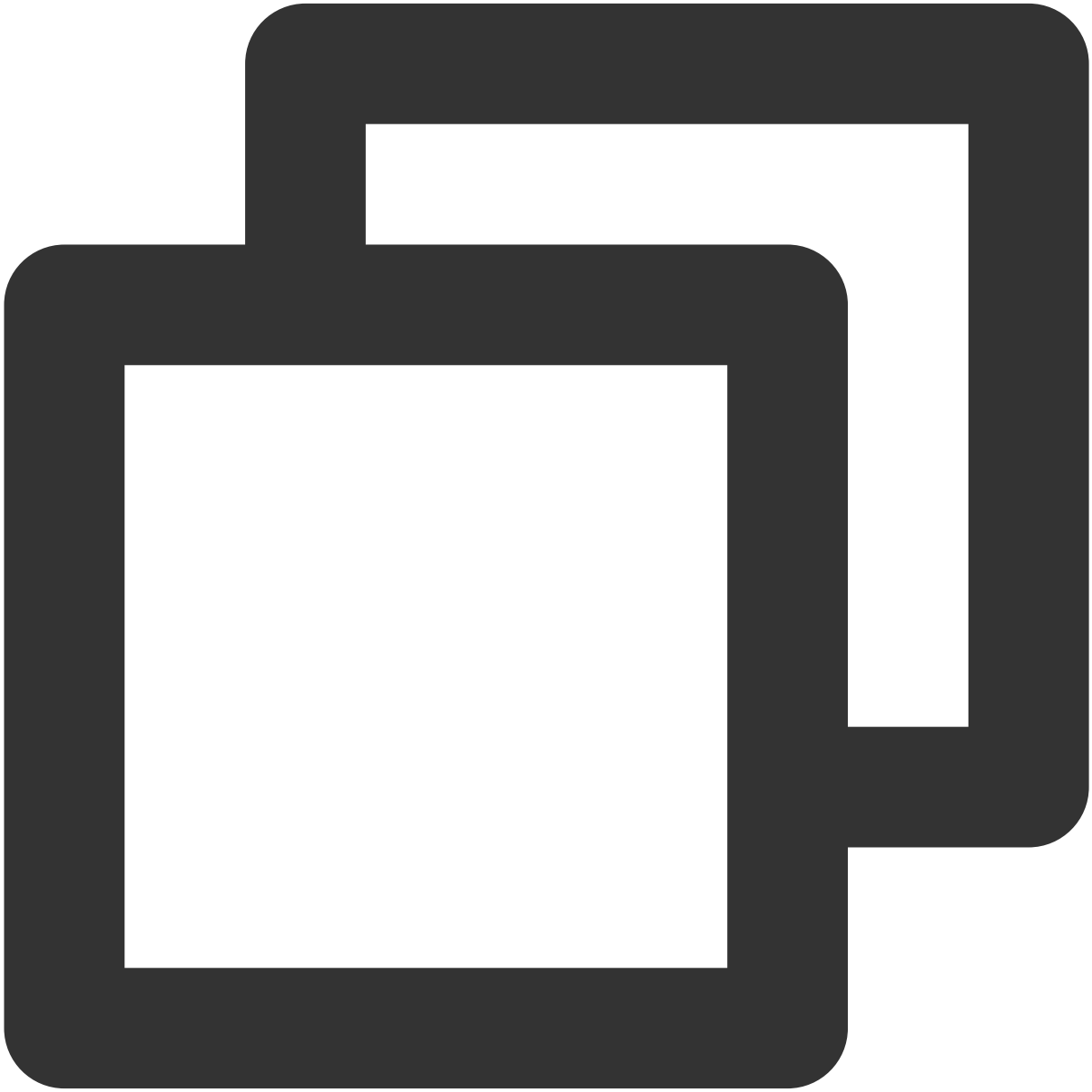
```
$ goosefs ns create myNamespace cosn://bucketName-1250000000/ \\
--secret fs.cosn.userinfo.secretId=AKXXXXXXXXXXX \\
--secret fs.cosn.userinfo.secretKey=XXXXXXXXXXXX \\
--attribute fs.cosn.bucket.region=ap-xxx \\
```

Note:

When creating the namespace that mounts COSN, you must use the `--secret` parameter to specify the key, and use `--attribute` to specify all required parameters of Hadoop-COS (COSN). For the required parameters, please see Hadoop.

When you create the namespace, if there is no read/write policy (rPolicy/wPolicy) specified, the read/write type set in the configuration file, or the default value (CACHE/CACHE_THROUGH) will be used.

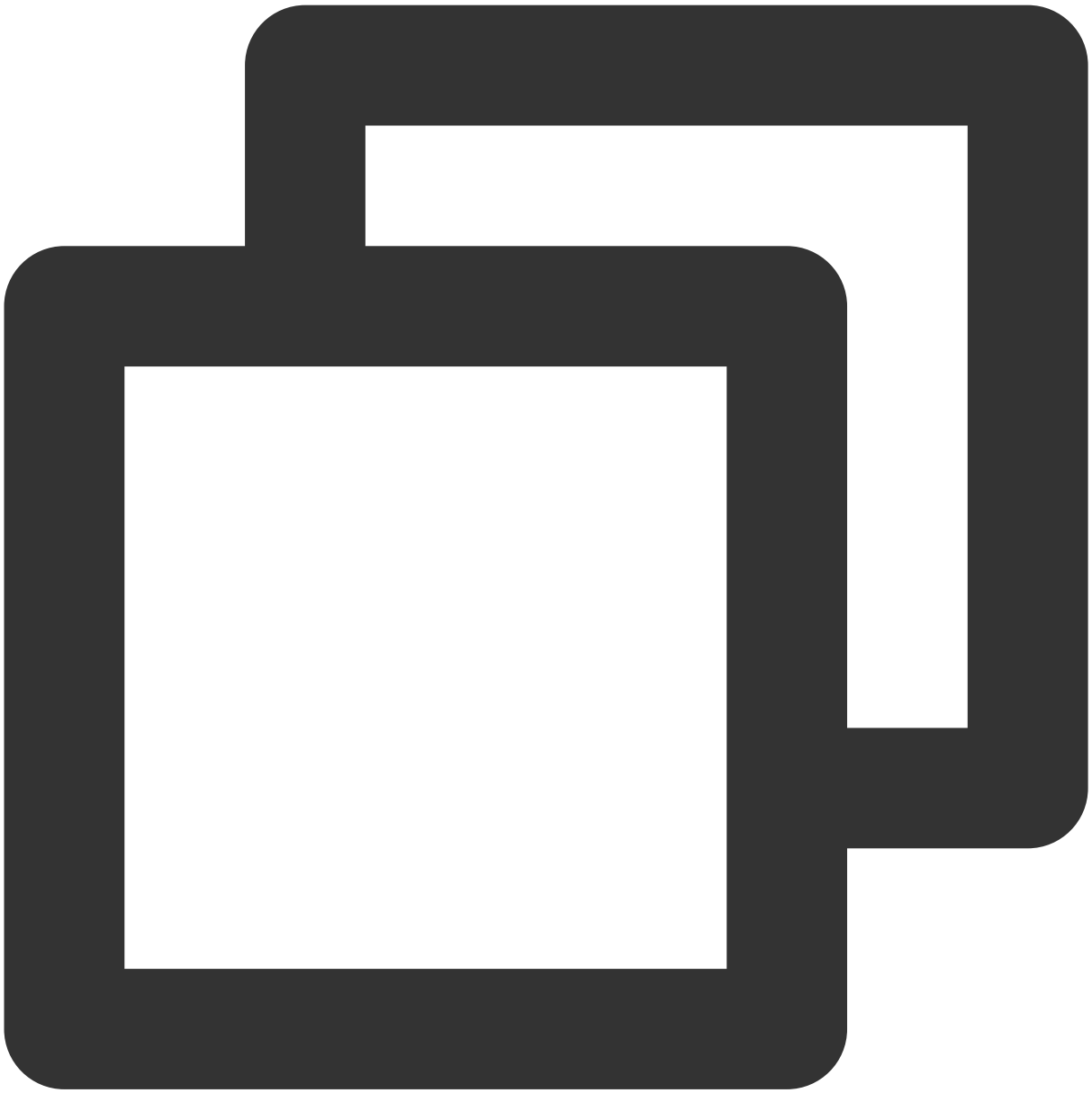
Likewise, create a namespace to mount Tencent Cloud HDFS:



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

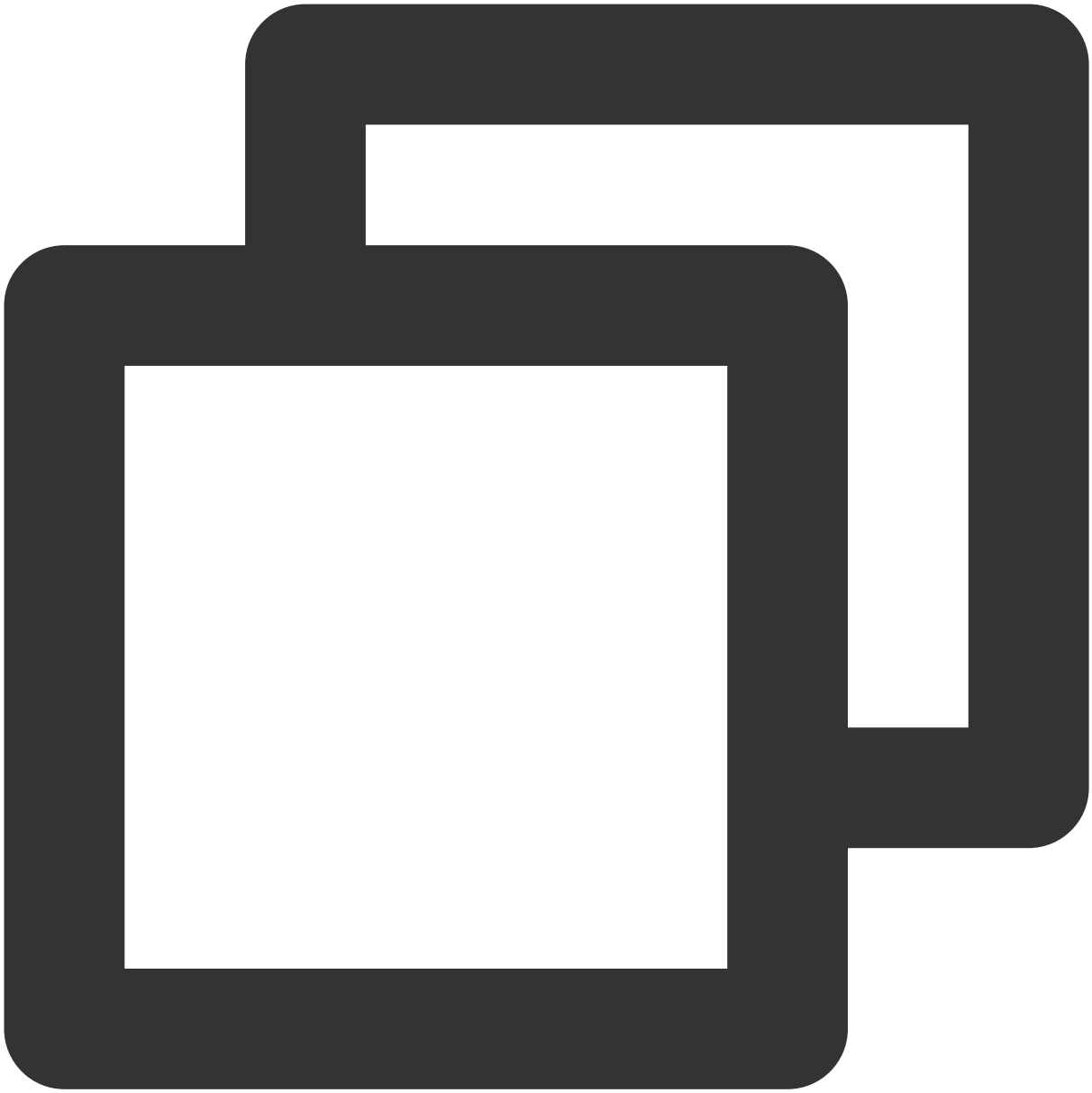
```
--attribute fs.ofs.tmp.cache.dir=/tmp/chdfs
```

2. After the namespaces are created, run the `ls` command to list all namespaces created in the cluster:



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

3. Run the following command to specify the namespace information:



```
$ goosefs ns stat myNamespace

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

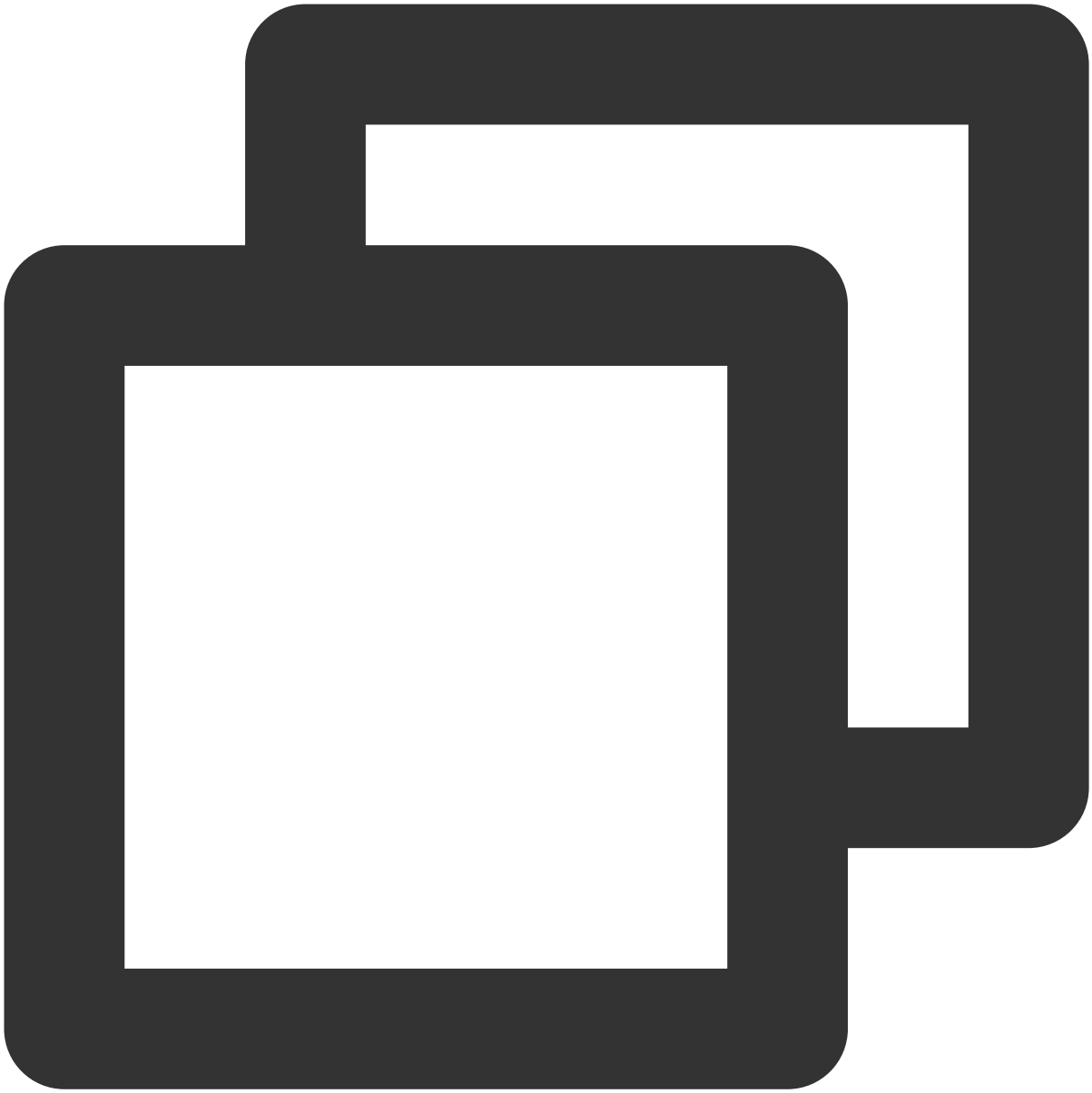
Information recorded in the metadata is as follows:

No.	Parameter	Description
1	name	Name of the namespace
2	path	Path of the namespace in GooseFS

3	ttlTime	TTL period of files and directories in the namespace
4	ttlAction	TTL action to handle files and directories in the namespace. Valid values: `FREE` (default), `DELETE`
5	ufsPath	Mount path of the namespace in the UFS
6	creationTimeMs	Time when the namespace is created, in milliseconds
7	lastModificationTimeMs	Time when files or directories in the namespace is last modified, in milliseconds
8	persistenceState	Persistence state of the namespace
9	mountPoint	Whether the namespace is a mount point. The value is fixed to `true` .
10	mountId	Mount point ID of the namespace
11	acl	ACL of the namespace
12	defaultAcl	Default ACL of the namespace
13	owner	Owner of the namespace
14	group	Group where the namespace owner belongs
15	mode	POSIX permission of the namespace
16	writePolicy	Write policy for the namespace
17	readPolicy	Read policy for the namespace

Loading Table Data to GooseFS

1. You can load Hive table data to GooseFS. Before the loading, attach the database to GooseFS using the following command:

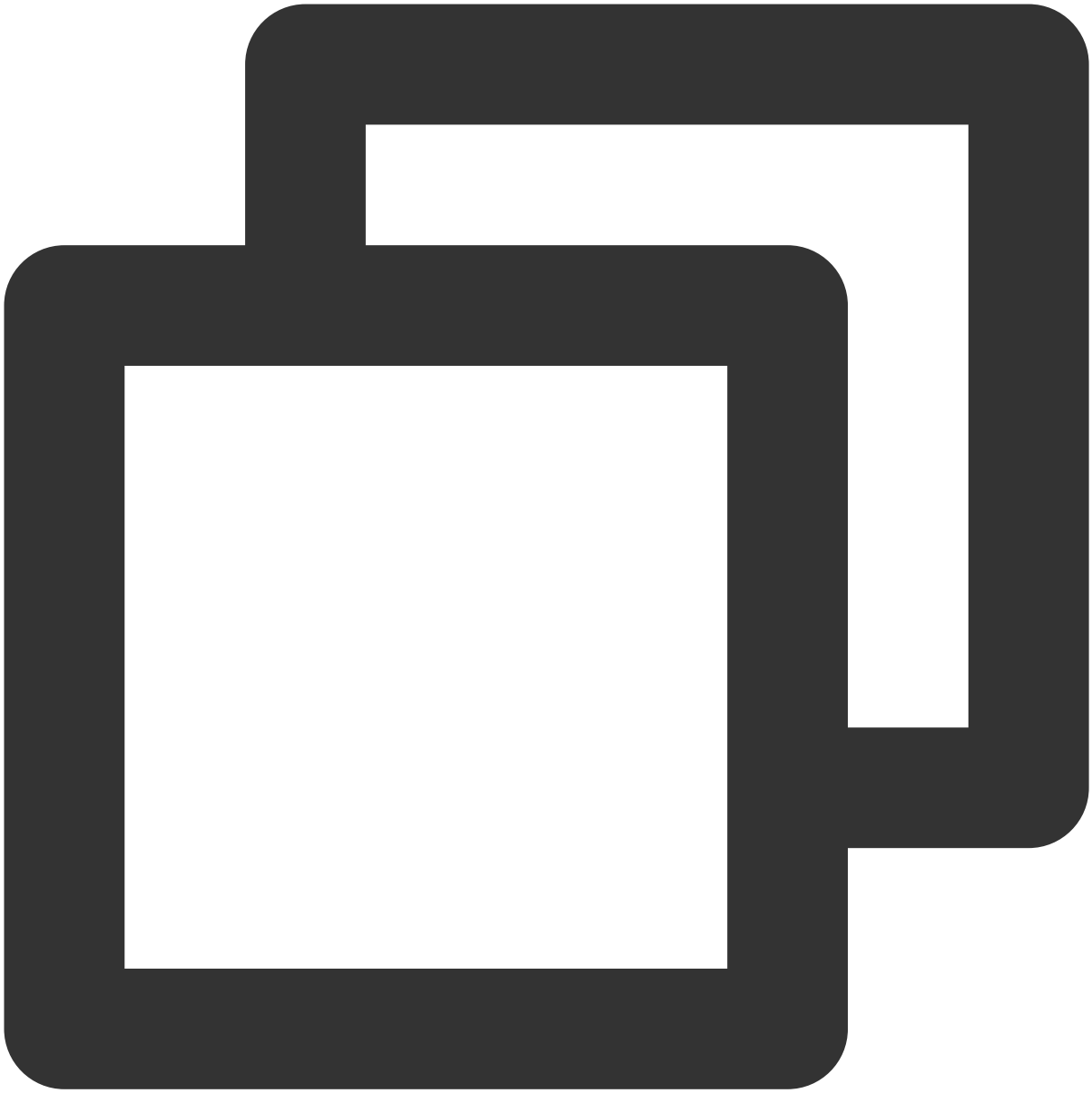


```
$ goosefs table attachdb --db test_db hive thrift://
172.16.16.22:7004 test_for_demo
```

Note:

Replace `thrift` in the command with the actual Hive Metastore address.

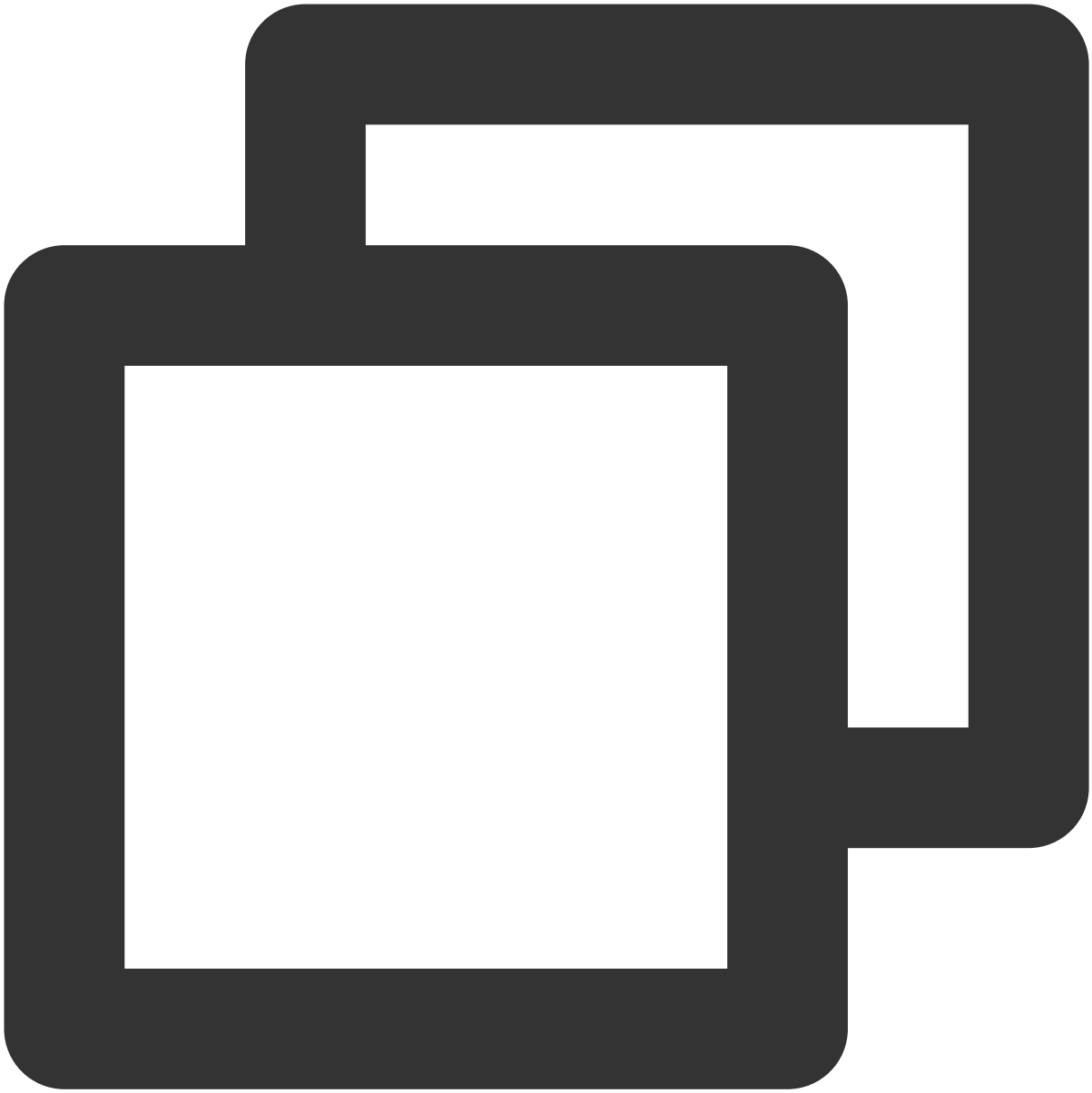
2. After the database is attached, run the `ls` command to view information about the attached database and table:



```
$ 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. Run the `load` command to load table data:

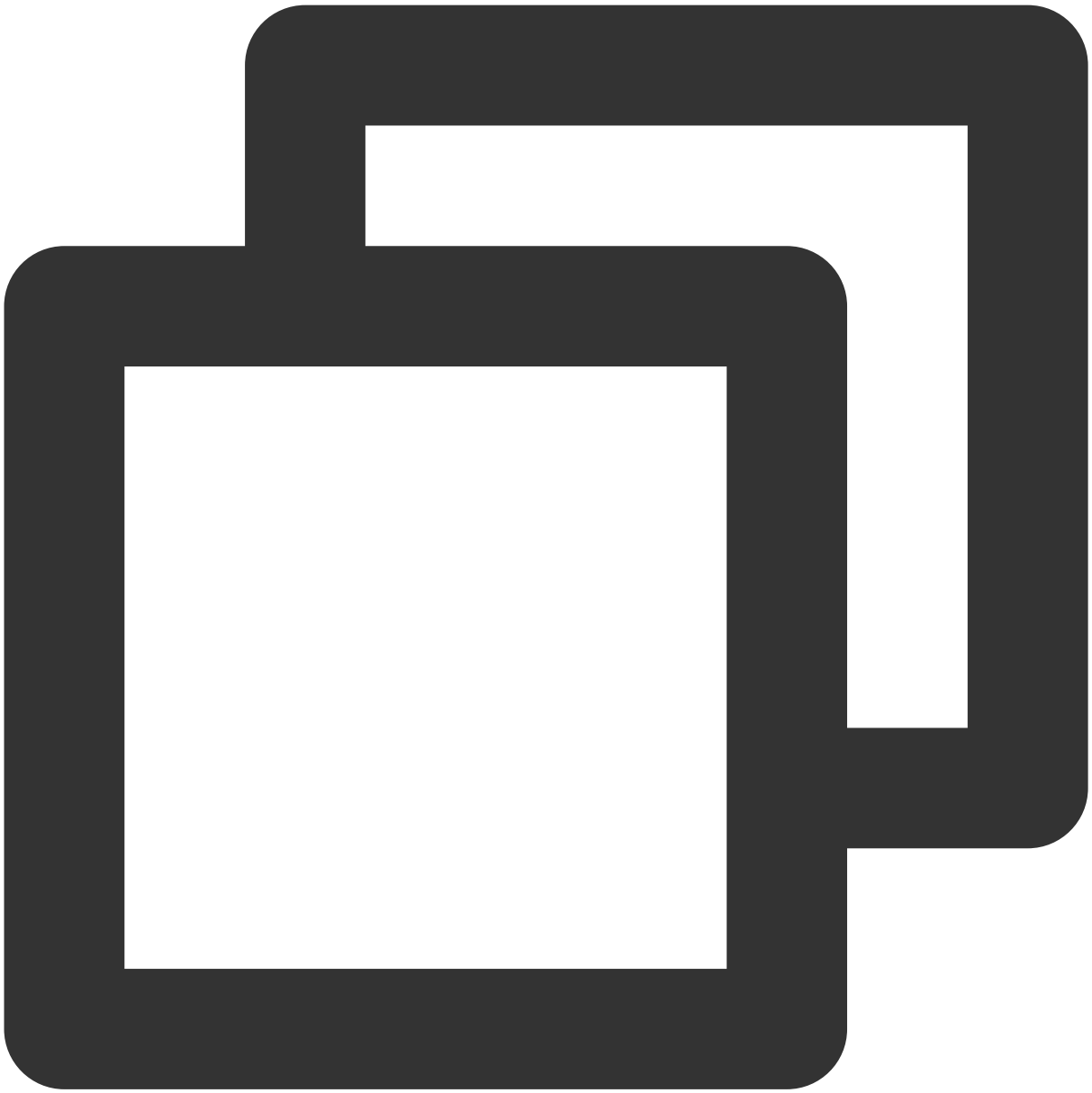


```
$ goosefs table load test_db web_page
Asynchronous job submitted successfully, jobId: 1615966078836
```

The loading of table data is asynchronous. Therefore, a job ID will be returned. You can run the `goosefs job stat <Job Id>` command to view the loading progress. When the status becomes "COMPLETED", the loading succeeds.

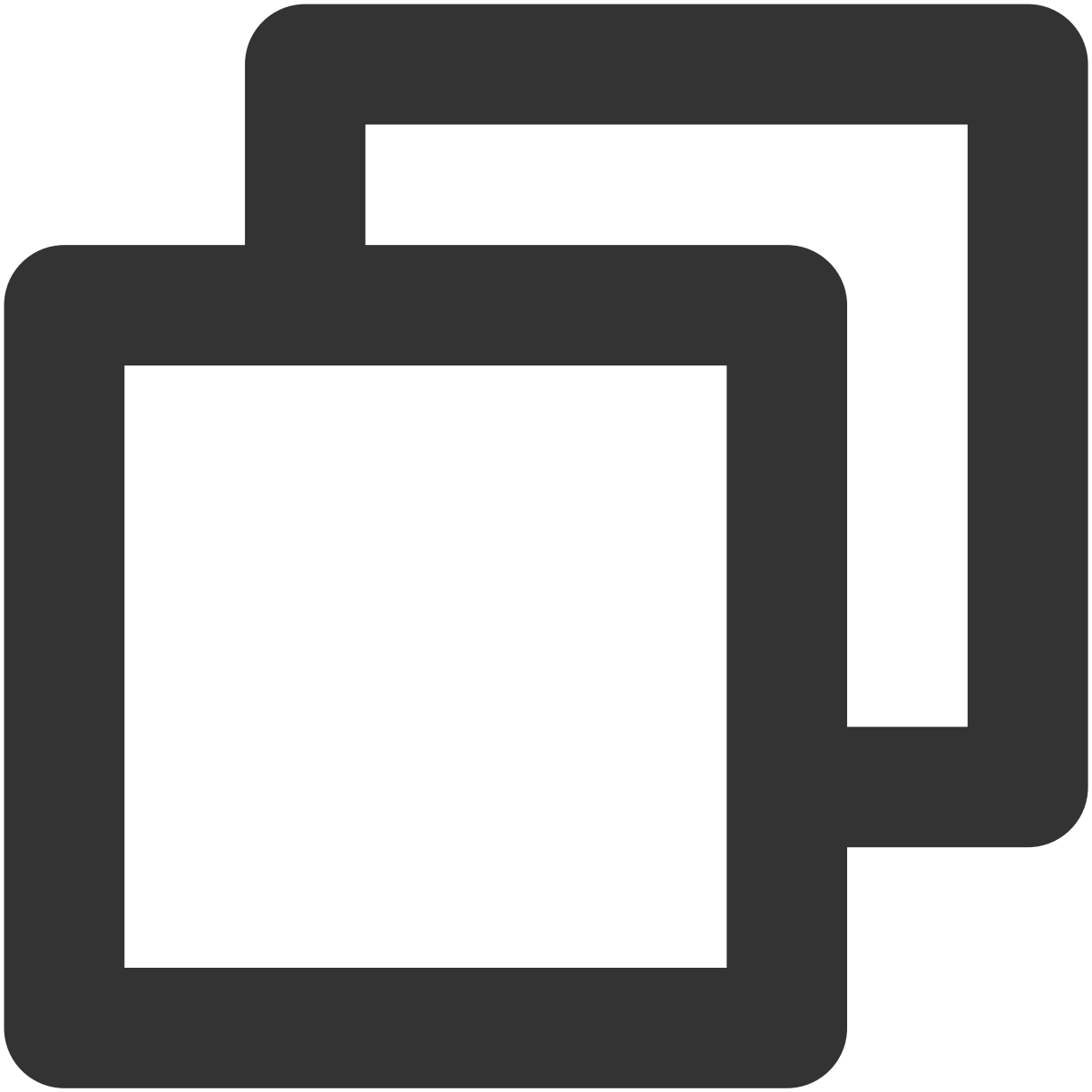
Using GooseFS for Uploads/Downloads

1. GooseFS supports most file system–related commands. You can run the following command to view the supported commands:



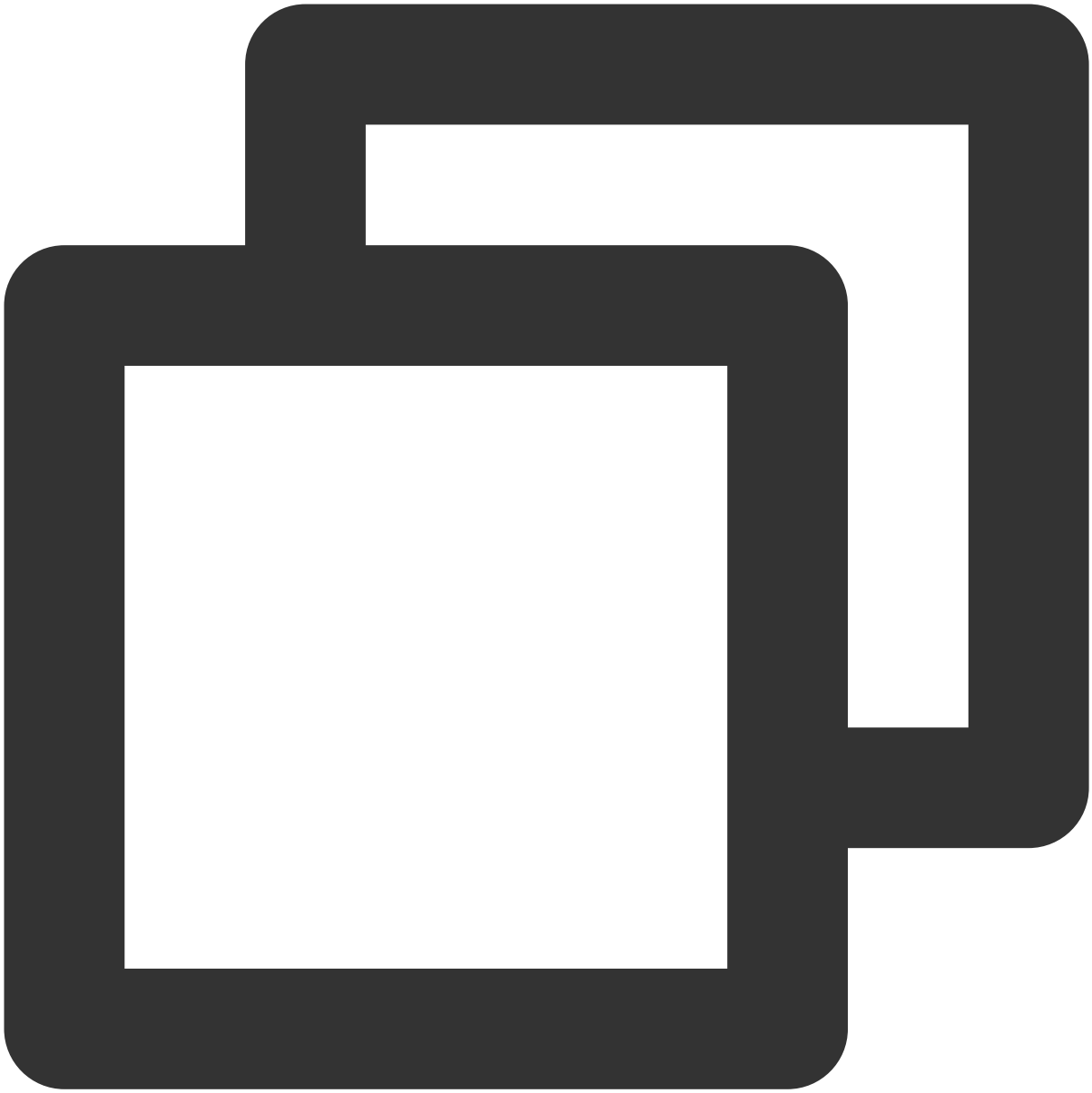
```
$ goosefs fs
```

2. Run the `ls` command to list files in GooseFS. The following example lists all files in the root directory:



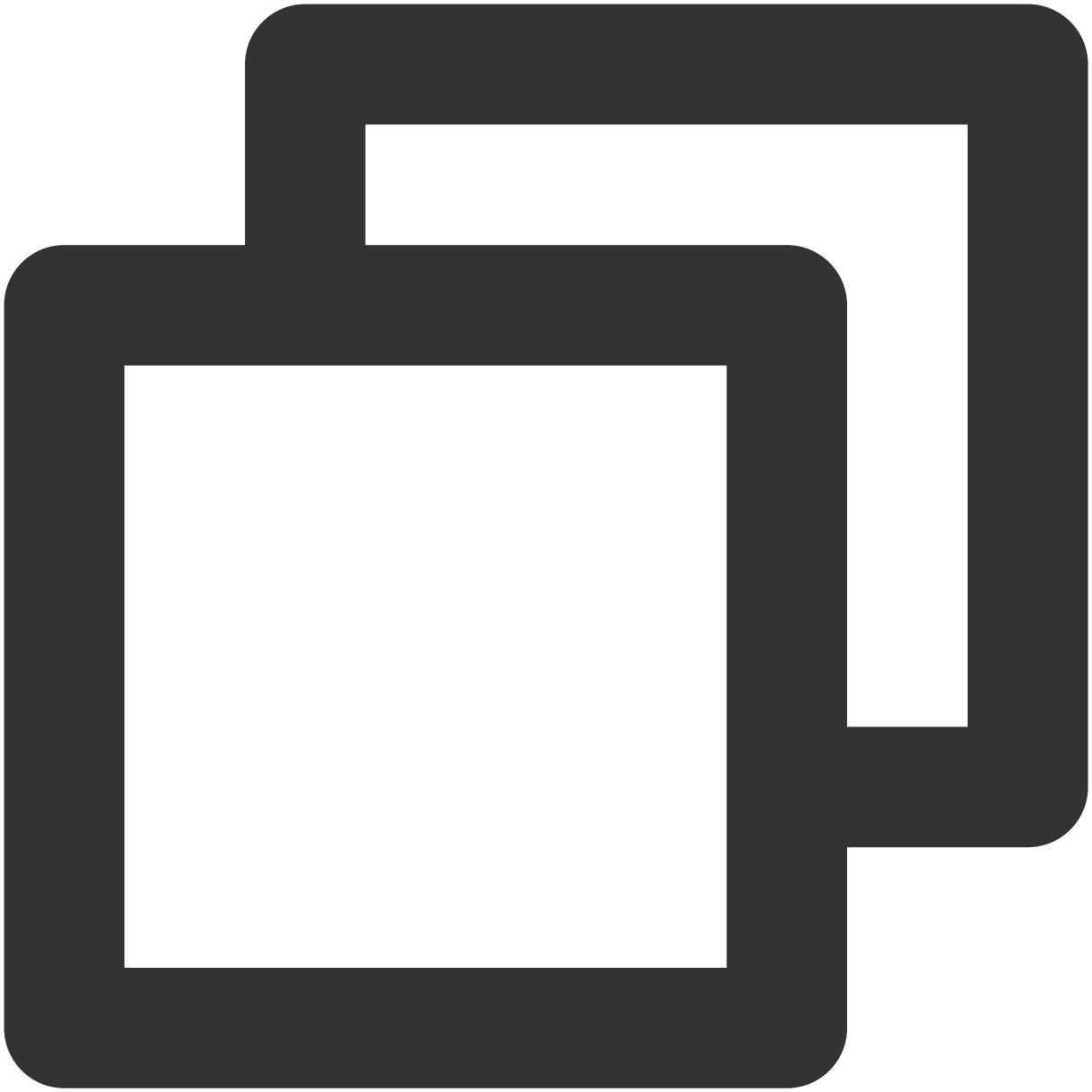
```
$ goosefs fs ls /
```

3. Run the `copyFromLocal` command to copy a local file to GooseFS:



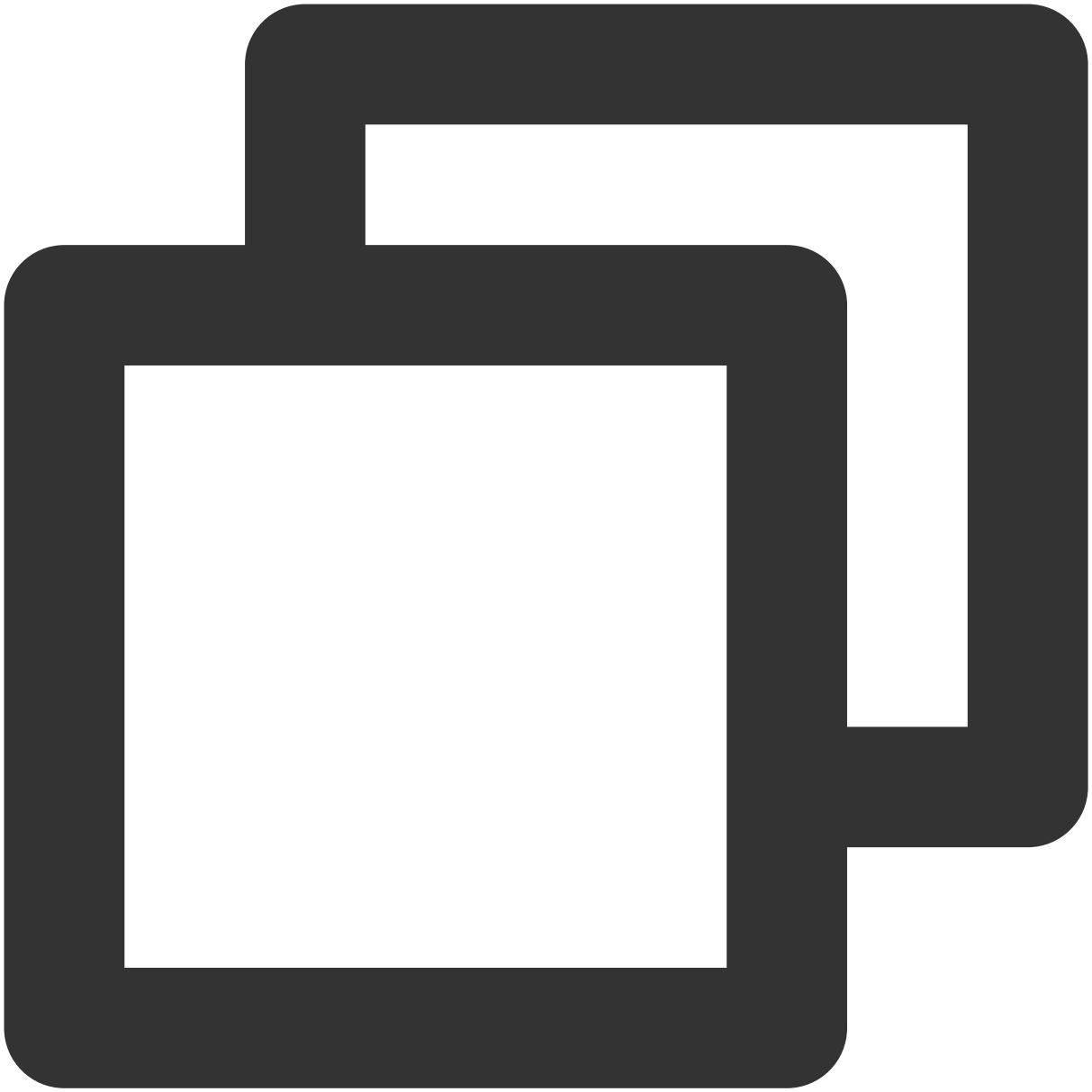
```
$ goosefs fs copyFromLocal LICENSE /LICENSE
Copied LICENSE to /LICENSE
$ goosefs fs ls /LICENSE
-rw-r--r--  hadoop         supergroup                20798       NOT_PERSISTED 03-26
```

4. Run the `cat` command to view the file content:



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

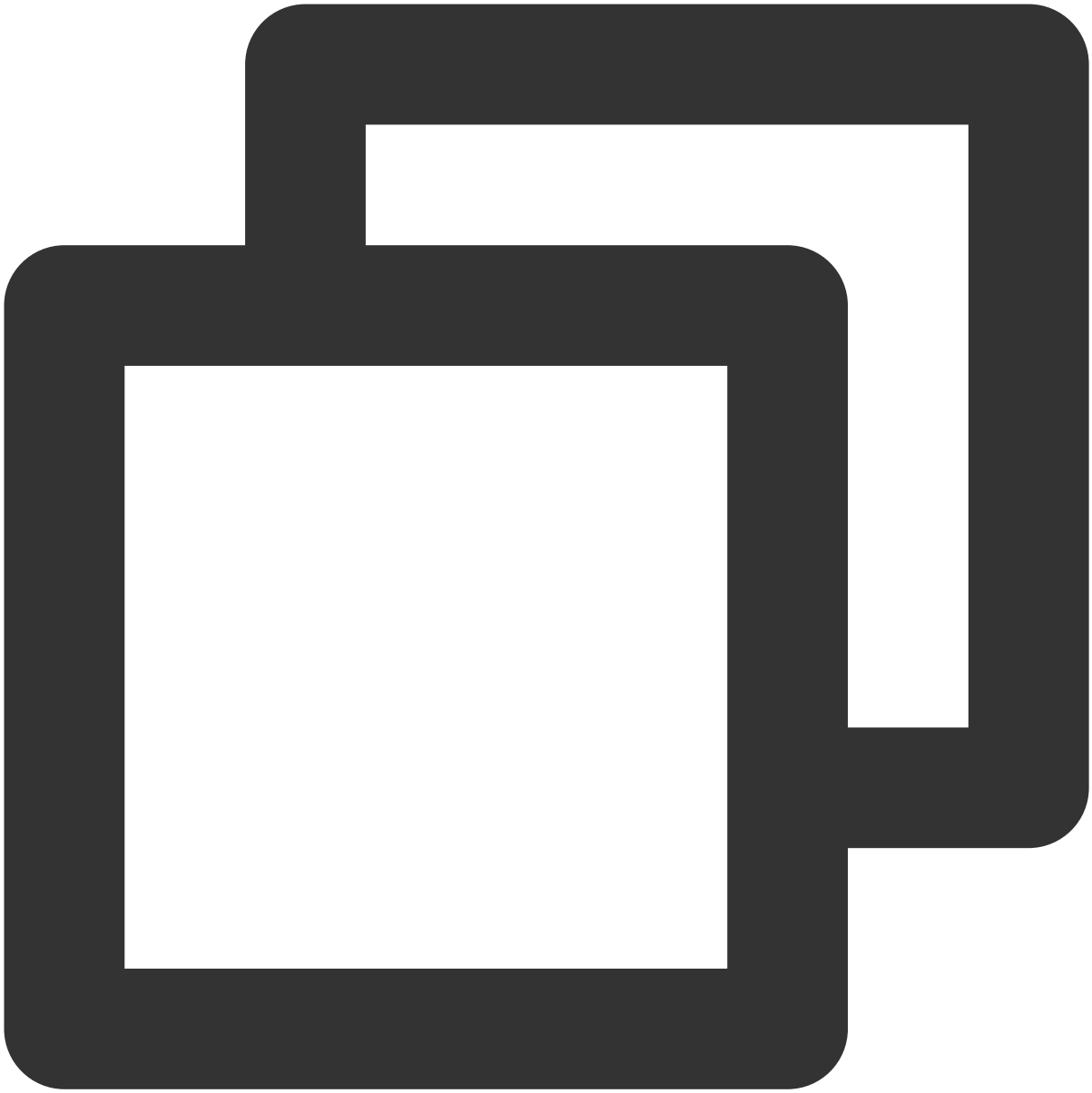
5. By default, GooseFS uses the local disk as the underlying file system. The default file system path is `./underFSStorage`. You can run the `persist` command to store files to the local system persistently as follows:



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

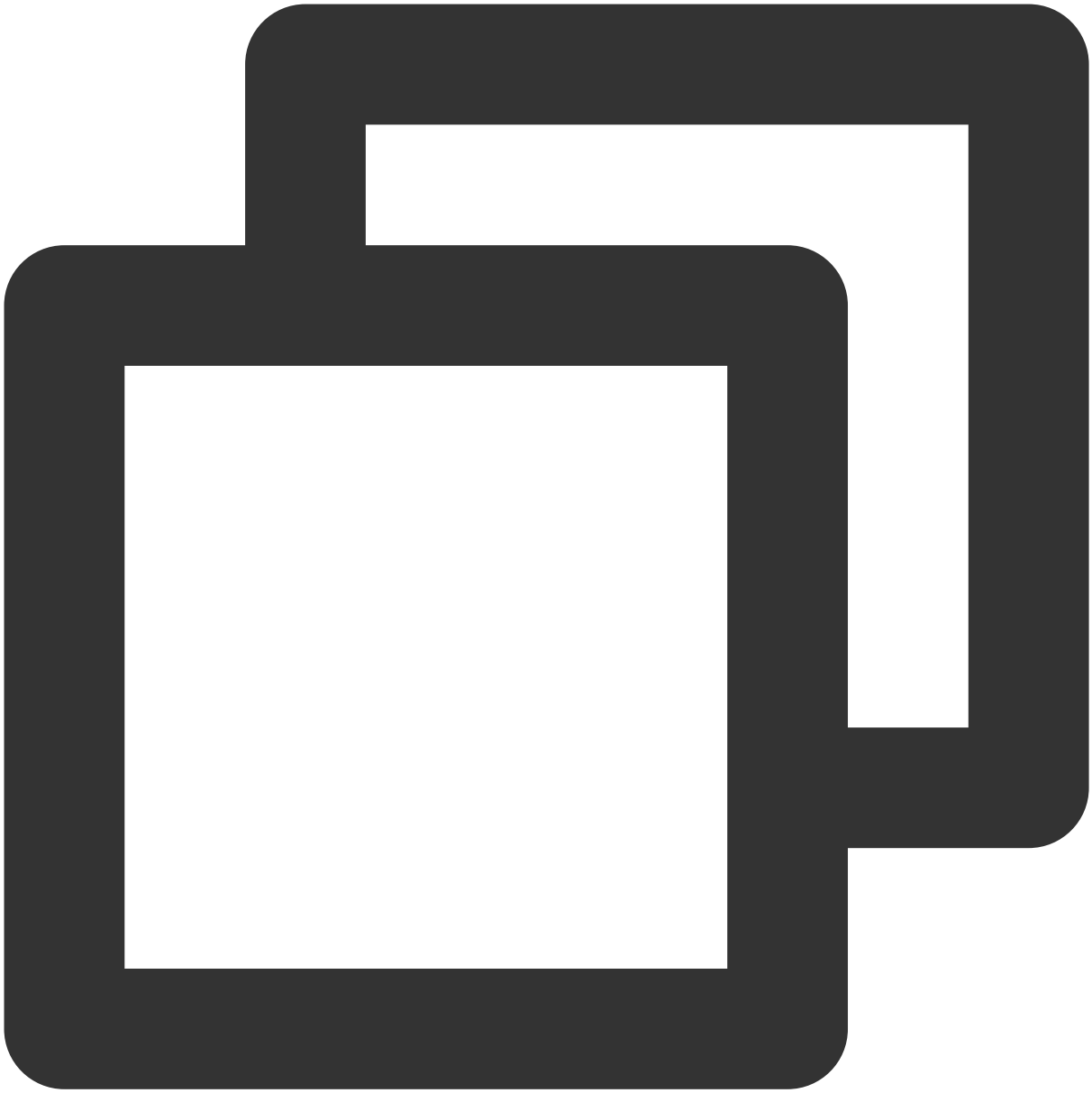
Using GooseFS to Accelerate Uploads/Downloads

1. Check the file status to determine whether a file is cached. The file status `PERSISTED` indicates that the file is in the memory, and `NOT_PERSISTED` indicates not.



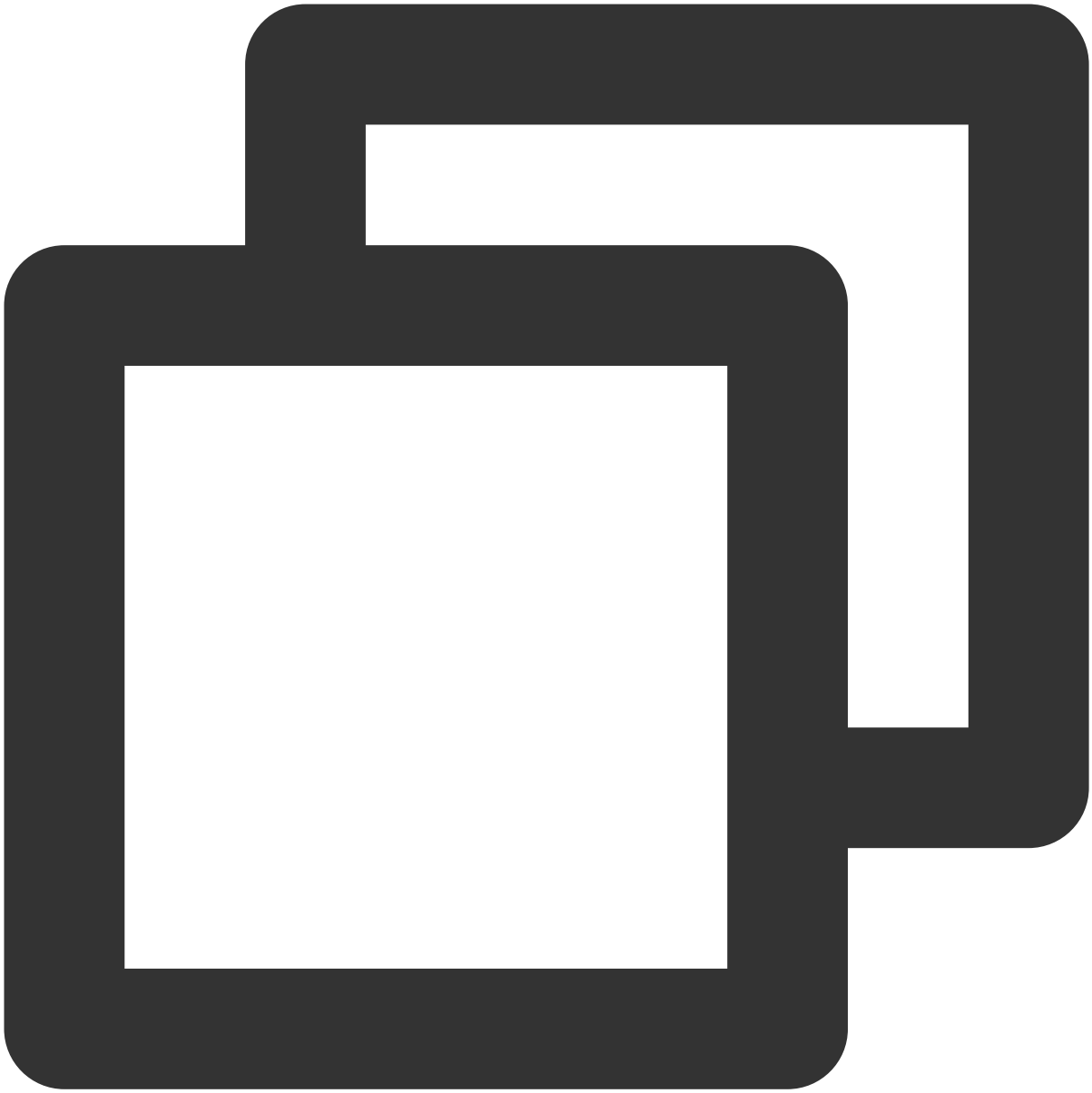
```
$ goosefs fs ls /data/cos/sample_tweets_150m.csv
-r-x------ staff  staff 157046046 NOT_PERSISTED 01-09-2018 16:35:01:002   0% /data/
```

2. Count how many times “tencent” appeared in the file and calculate the time consumed:



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

3. Caching data in memory can effectively speed up queries. An example is as follows:

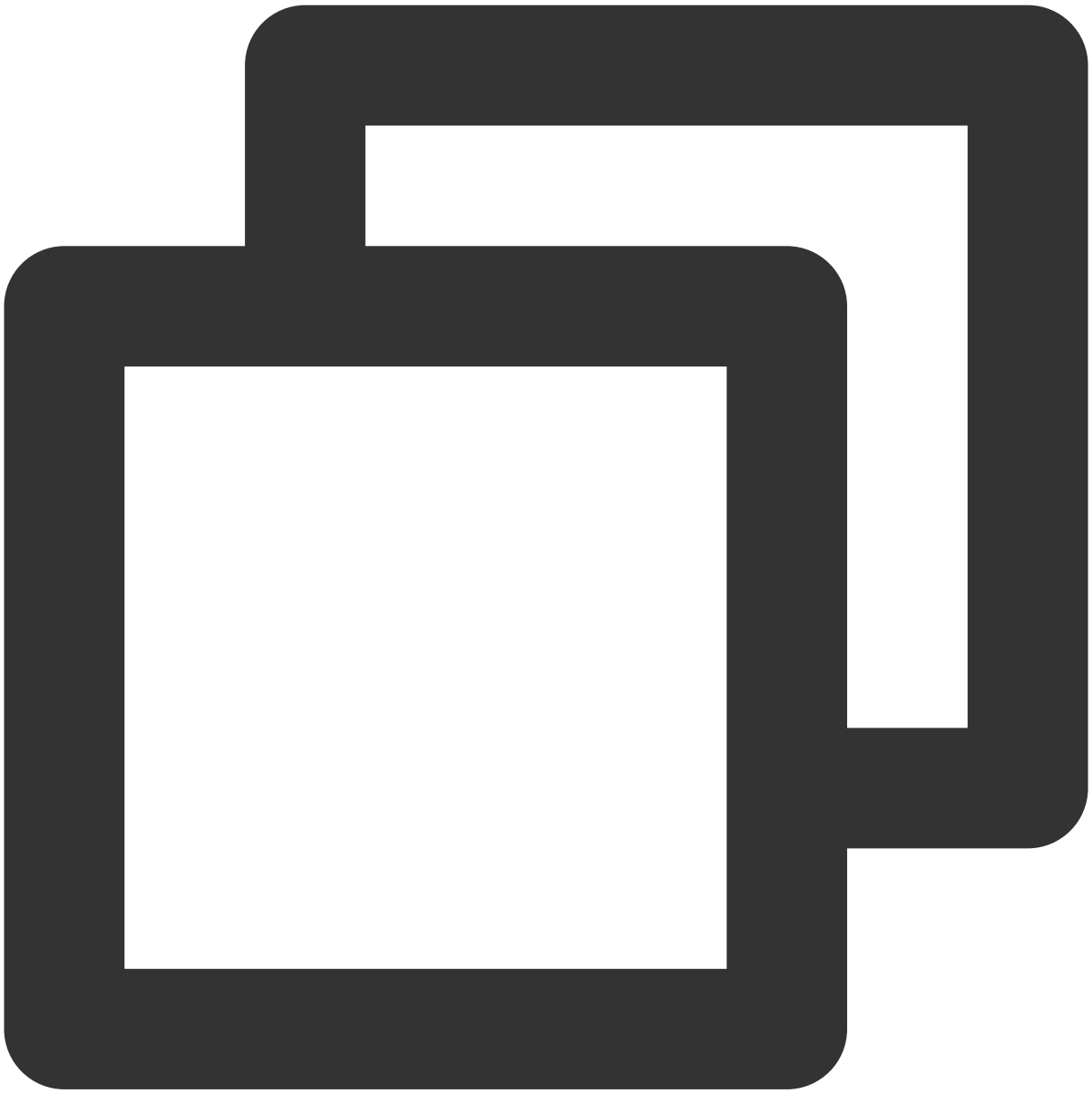


```
$ goosefs fs ls /data/cos/sample_tweets_150m.csv
-r-x------ staff  staff 157046046
ED 01-09-2018 16:35:01:002   0% /data/cos/sample_tweets_150m.csv
$ time goosefs fs cat /data/s3/sample_tweets_150m.csv | grep-c tencent
889
real    0m1.917s
user    0m2.306s
sys         0m0.243s
```

The data above shows that the system delay is reduced from 1.181s to 0.243s, achieving a 10-times improvement.

Shutting Down GooseFS

Run the following command to shut down GooseFS:



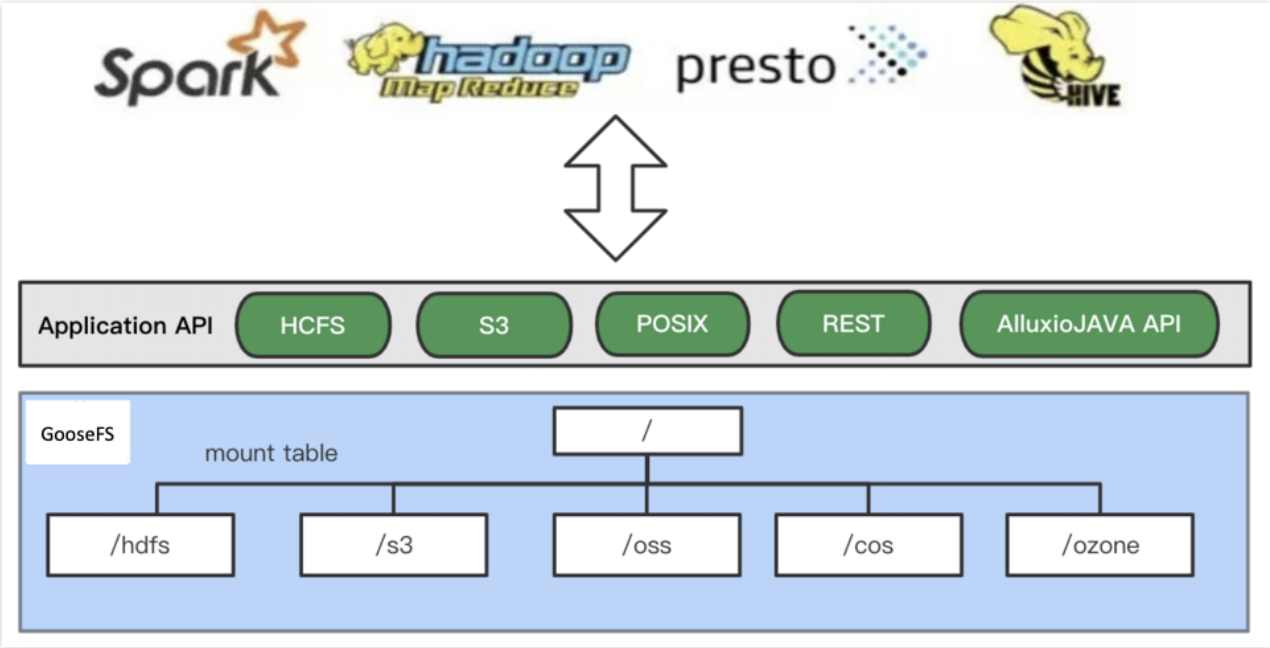
```
$ ./bin/goosefs-stop.sh local
```

Key Features
Caching

Last updated：2024-03-25 16:04:01

Overview

Based on the open-source Alluxio framework, GooseFS provides a powerful distributed cache ability to unify cross-platform data and improve the overall I/O throughput.
It can be implemented as follows:
Remote storage for namespaces: Connect namespaces to different external underlying storage such as COS and CHDFS for unlimited and persistent storage. With CosN's compute-storage separation solution, GooseFS offers scalability and high availability for big data storage.
GooseFS local cache: GooseFS leverages the distributed cache ability of the master and workers to store the cached and temporary data generated at the application layer in the memory and disk of the local application node. Each local node is configurable. The remote persistent storage layer connected to namespaces will use the same client to serve users' reads/writes.
You can configure the cache policy to decide whether to use the local cache or remote storage for namespaces.



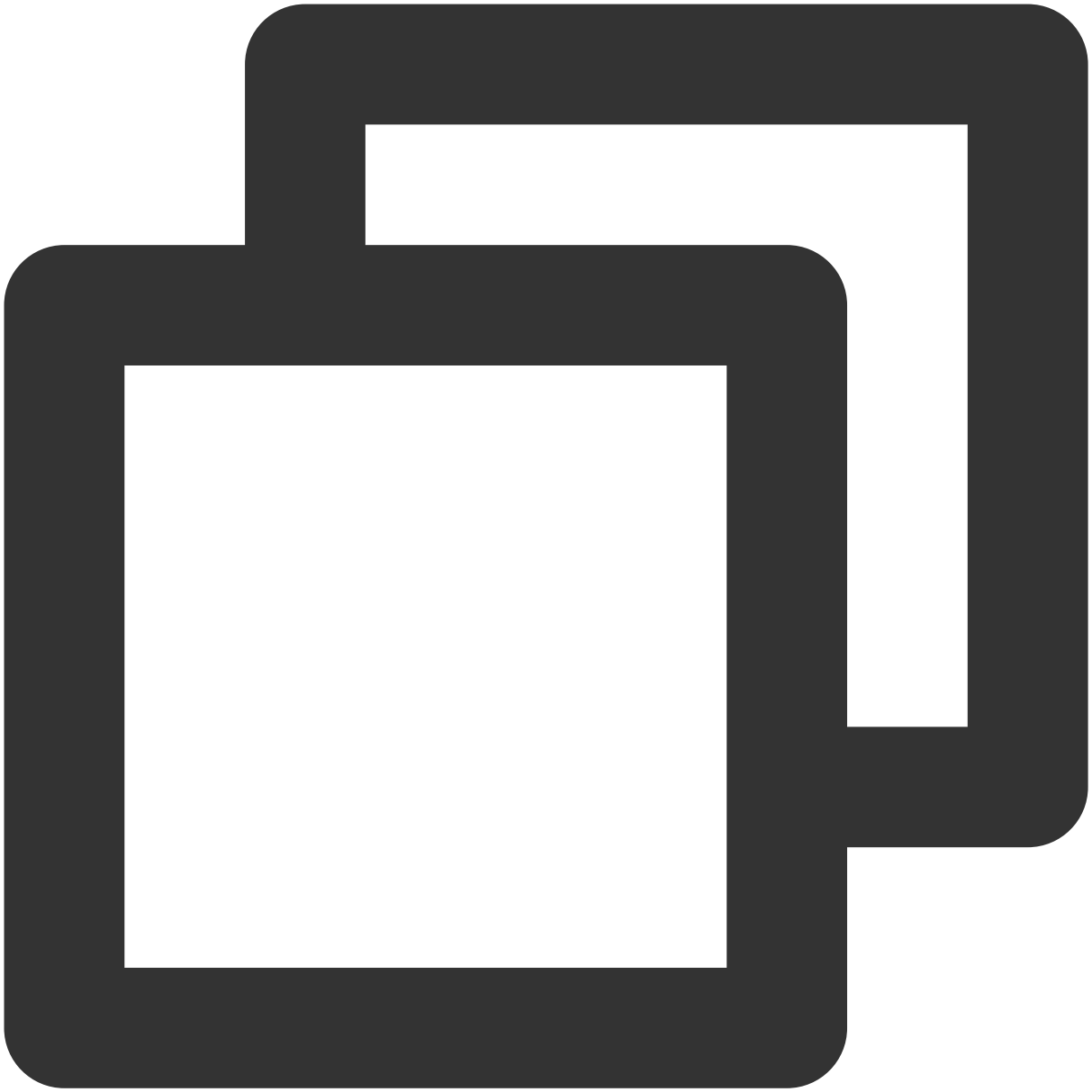
GooseFS supports local cache and remote storage for namespaces to provide an all-around storage separated solution and create data affinity for application nodes.

GooseFS Cache Configuration

You can open the `goosefs-site.properties` file to view the GooseFS cache configurations, which include cache levels (single-level and multi-level storage), cache replacement policies (LRU and LRFU), and more.

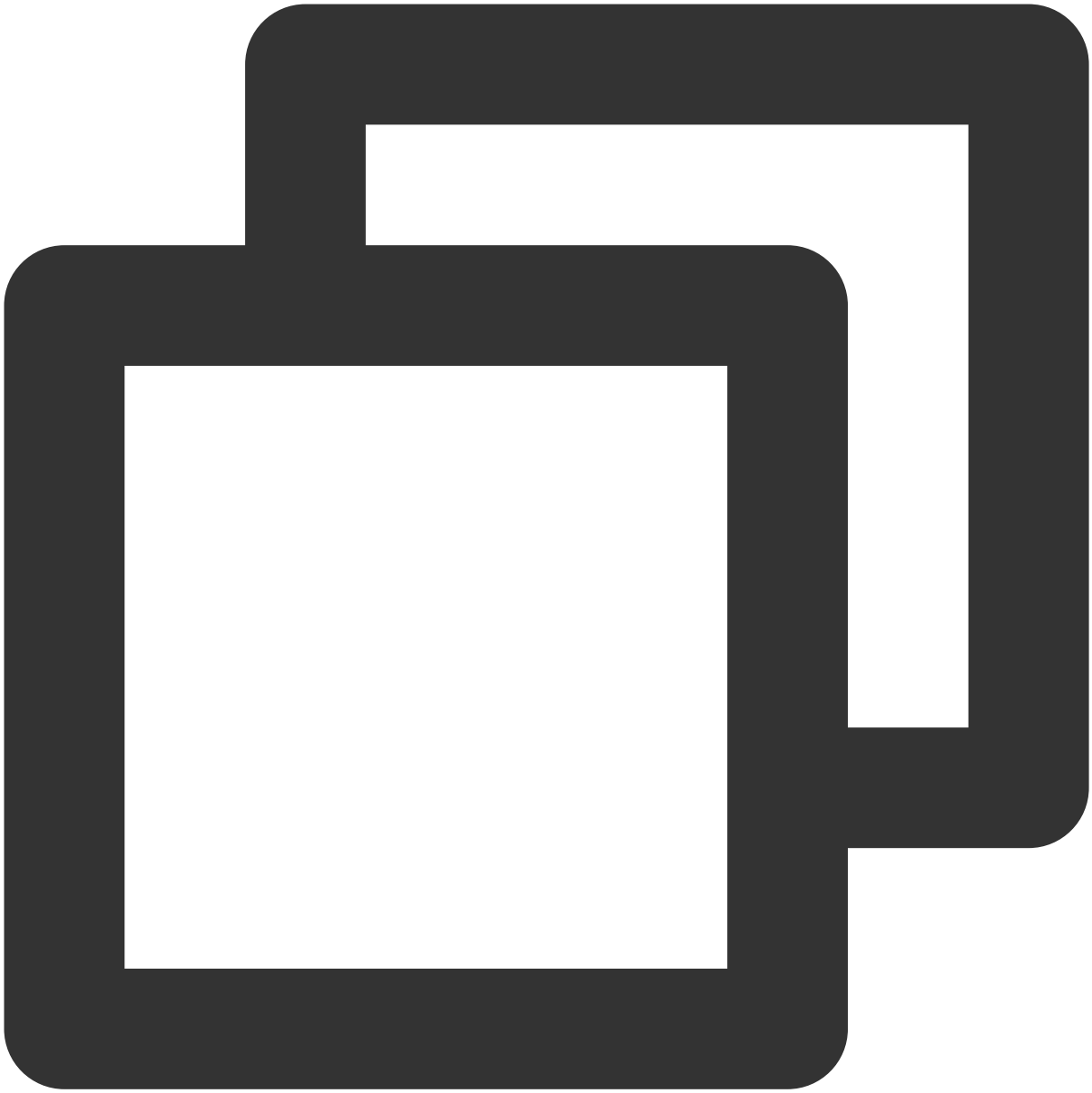
1. Cache level

GooseFS provides single-level and multi-level caches. Single-level cache uses only one memory device, while multi-level cache uses various storage devices to meet the needs of different business loads and provide satisfactory I/O performance. By default, GooseFS uses single-level storage. If there are multiple storage devices, cache replacement will affect performance and thus **single-level storage** is recommended. You can choose a memory device such as MEM, SSD, or HDD according to your I/O requirements.
You can modify `levels` in the `goosefs-site.properties` configuration file to change the cache level, where `1` indicates using only one level of storage and `3` indicates 3 levels.



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

You can also modify `alias` in `goosefs-site.properties` to change the storage device used for a cache level.



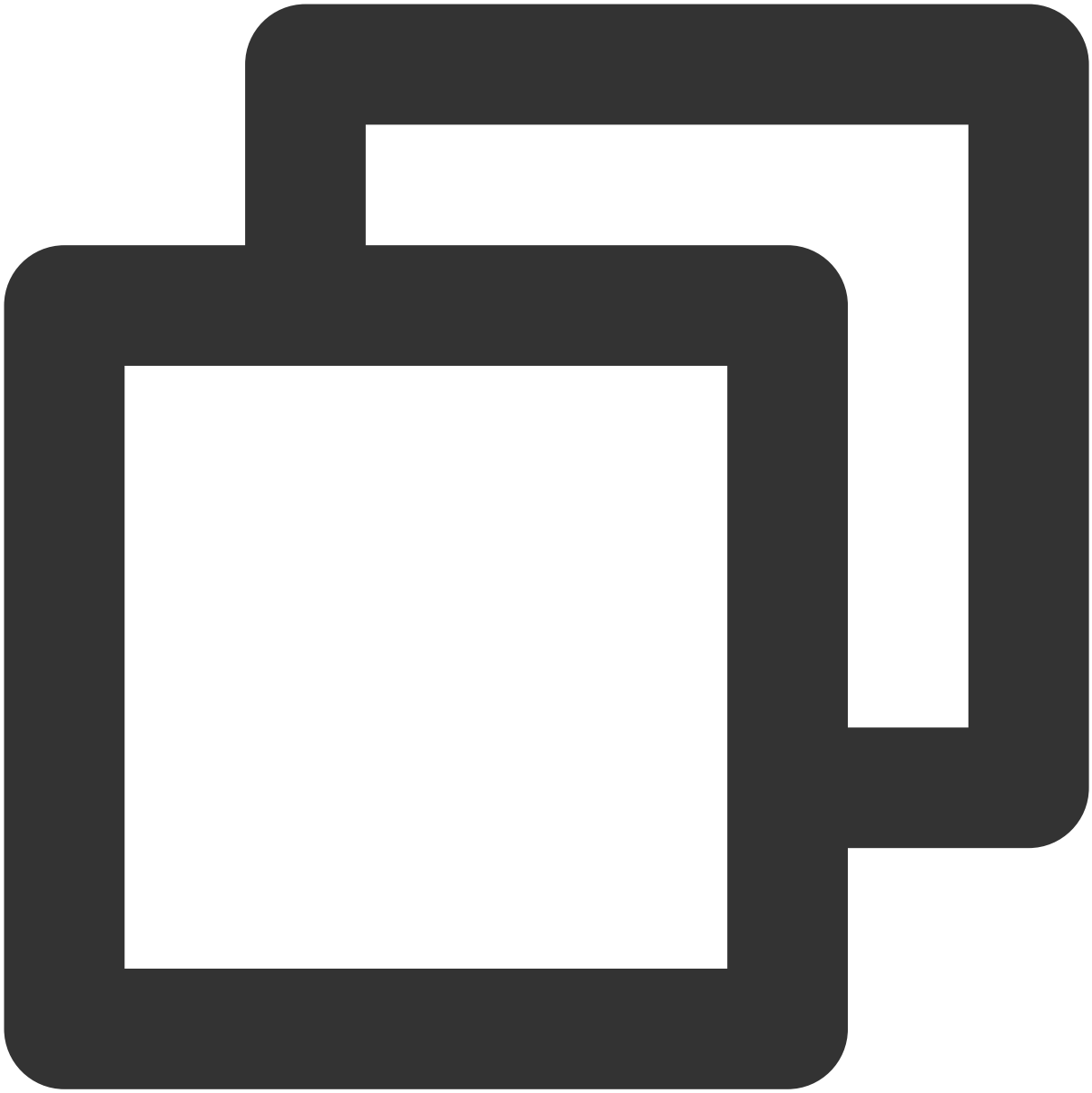
```
goosefs.worker.tieredstore.level{x}.alias=MEM
```

Note:

”x” in level{x} is the index of the storage level. For example, `level0` indicates single-level storage.

1.1 Single-level storage

Common configurations for single-level storage are as follows:



```
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
```

Configuration items:

ramdisk.size: memory size for workers. The value must be smaller than `memory`. GooseFS will allocate memory from the total memory for each worker according to the specified value.
memory.size: total memory of the GooseFS system. The specified size of memory will be automatically used from the storage device. The value must be smaller than the size of the physical memory of the storage device.
dirs.path: directories that GooseFS allocates storage devices for. This parameter must be used together with `dirs.mediumtype`. In the example above, `/data/GooseFSWorker` is attached to the MEM storage device, and `/data4/GooseFSWorker` is attached to an SSD. Note that the directory sequence must correspond to the storage device sequence.
dirs.mediumtype: storage device used for the specified directories. This parameter must be used together with `dirs.path`. By default, `MEM` and `SSD` are valid values. If other storage devices such as HDD are attached, you can configure them as needed.
dirs.quota: space allocated for the specified directories. The values must correspond to the directory sequence. In the example above, each of `/data/GooseFSWorker`, `/data1/GooseFSWorker`, and `/data2/GooseFSWorker` are allocated 16 GB of MEM, and both `data3/GooseFSWorker` and `/data4/GooseFSWorker` are allocated 100 GB of SSD.

1.2 Multi-level storage

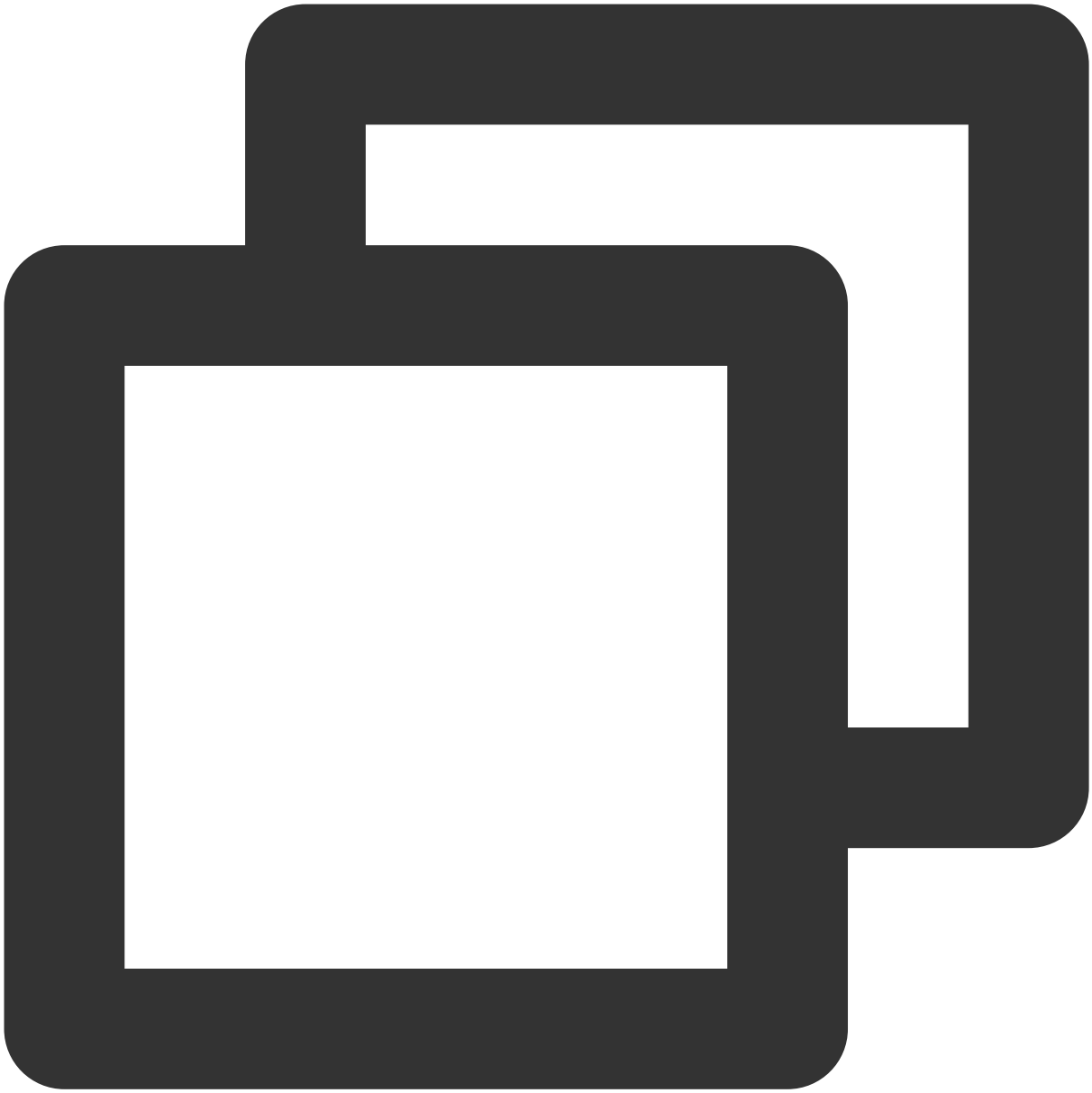
Multi-level storage reads and writes data blocks differently from single-level storage. In single-level mode, data reads/writes use the same storage device. However, in multi-level mode, data is first written to the highest level of storage device. If any data needs to be read, it will be moved to the highest level of storage device. Details are described below.

Data writes: New data blocks will be written to the highest level of storage device by default. If the highest level of storage device is used up, data will be moved to the next storage level. If storage capacities of all storage devices are used up, older data will be replaced according to the cache replacement policies you set. If expired data cannot be replaced and the available memory is used up, data cannot be written.

Data reads: In multi-level storage mode, cold data will be moved to the lower-level of storage device imperceptibly. If the data is read again, it will be moved back to the highest storage level.

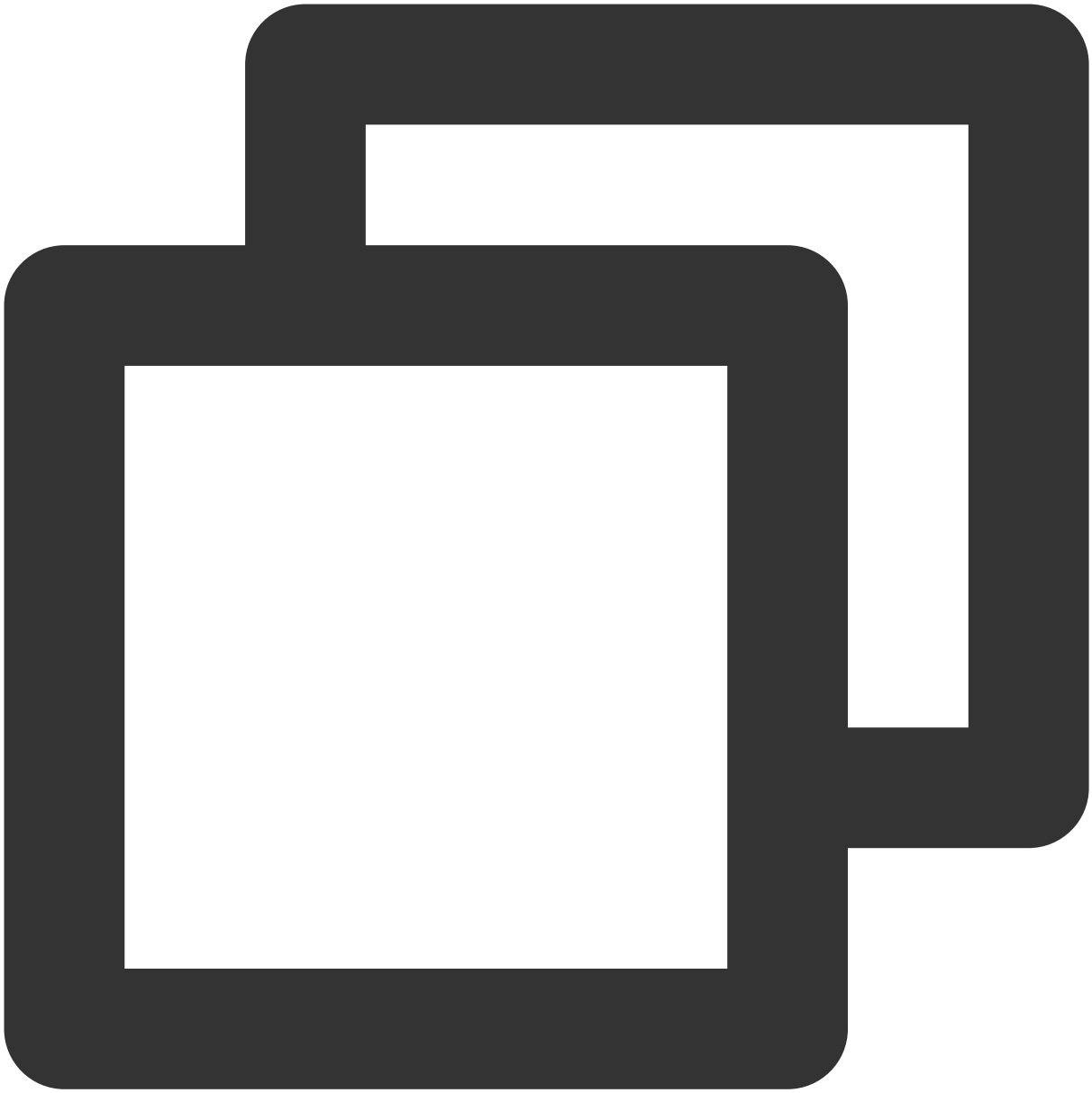
Note:

GooseFS will clear a specified amount of data according to the cache replacement policies configured. The amount can be specified with `goosefs.worker.tieredstore.free.ahead.bytes`, and the default value is `0`.
In multi-level storage mode, data reads may cause data movement from a lower storage level to the highest one and compromise performance. Therefore, **you are advised to use single-level storage** in most cases.
Common configuration items:



```
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
```

In the example above, `x` indicates the index of the cache level. In most cases, you can use three levels (corresponding MEM, SSD, and HDD). The example below uses two levels (MEM and SSD) and allocates 100 GB for each level:

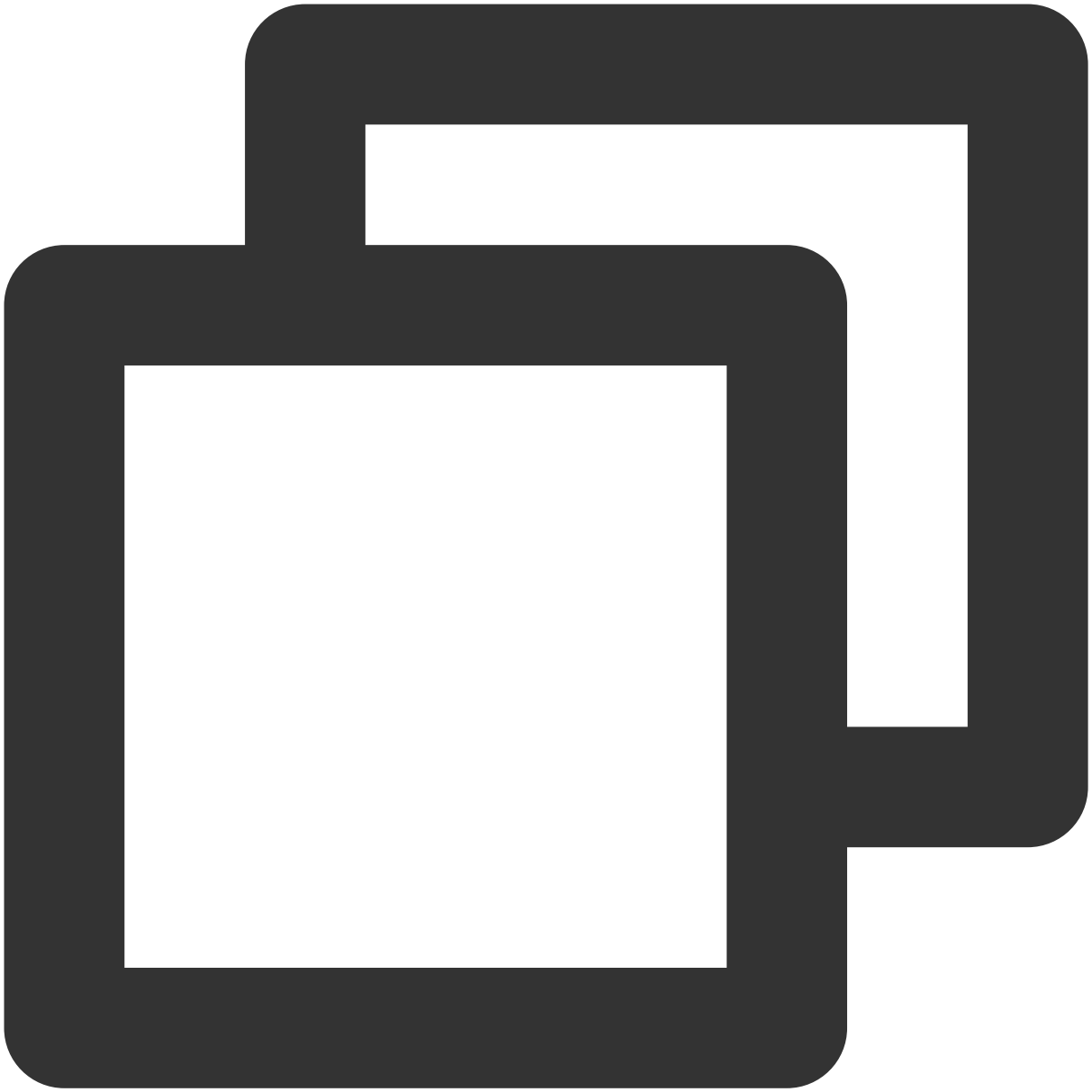


```
goosefs.worker.tieredstore.levels=2
goosefs.worker.tieredstore.level0.alias=MEM
goosefs.worker.tieredstore.level0.dirs.path=/data/GooseFSWorker
goosefs.worker.tieredstore.level0.dirs.mediumtype=MEM
goosefs.worker.tieredstore.level0.dirs.quota=100GB
goosefs.worker.tieredstore.level1.alias=SSD
goosefs.worker.tieredstore.level1.dirs.path=/data1/GooseFSWorker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

You can configure an unlimited number of storage levels with GooseFS, but their names (alias) must be unique. If you use three levels, it will be easy to distinguish them if you name them MEM, SSD, and HDD.

2. Cache replacement policies

GooseFS supports two cache replacement policies:
LRUAnnotator: least-recently-used (LRU) (default)
LRFUAnnotator: combines the least-recently-used (LRU) and least-frequently-used (LFU) schemes. You can set weights of LRU and LFU using `goosefs.worker.block.annotator.lrfu.step.factor` and `goosefs.worker.block.annotator.lrfu.attenuation.factor` to decide how these two policies should take effect together.
If the weight of least-recently-used is set the maximum value, it will function in the same way as `LRUAnnotator`.
You can use `goosefs.worker.block.annotator.class` to specify the cache replacement policies, where the policy names should be correctly set as follows:



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

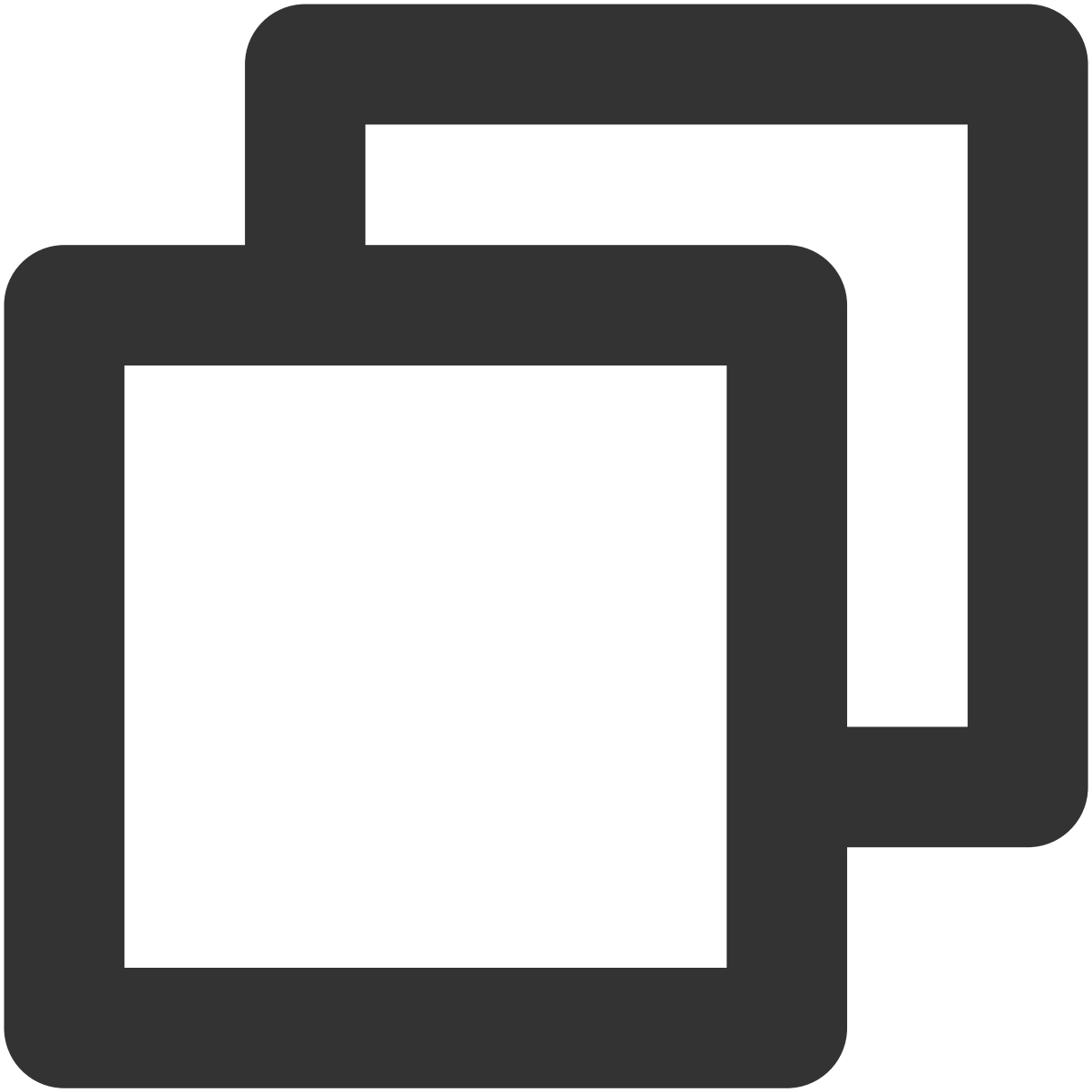
Data Lifecycle

The lifecycle described here is for data in GooseFS rather than data in the remote UFS. You can manage data lifecycle with the following four operations:

free: deletes the specified directories or files from GooseFS (data in the UFS will be intact). This operation is mainly used to free GooseFS cache space for other hotter data.
load: loads directories/files in the UFS to GooseFS. This operation is mainly used to restore cold data for higher I/O performance.
persist: writes GooseFS data to the UFS to store data persistently and avoid data loss.
Time to Live (TTL): sets the lifecycle for directories/files in GooseFS. If data has expired, it will be deleted from GooseFS and the UFS. You can also delete data only in GooseFS or only free the disk space.

1. Freeing data from GooseFS

Run the `free` command to free data from GooseFS. In the example below, the data status becomes 0% in GooseFS after the freeing operation.



```
$ 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
```

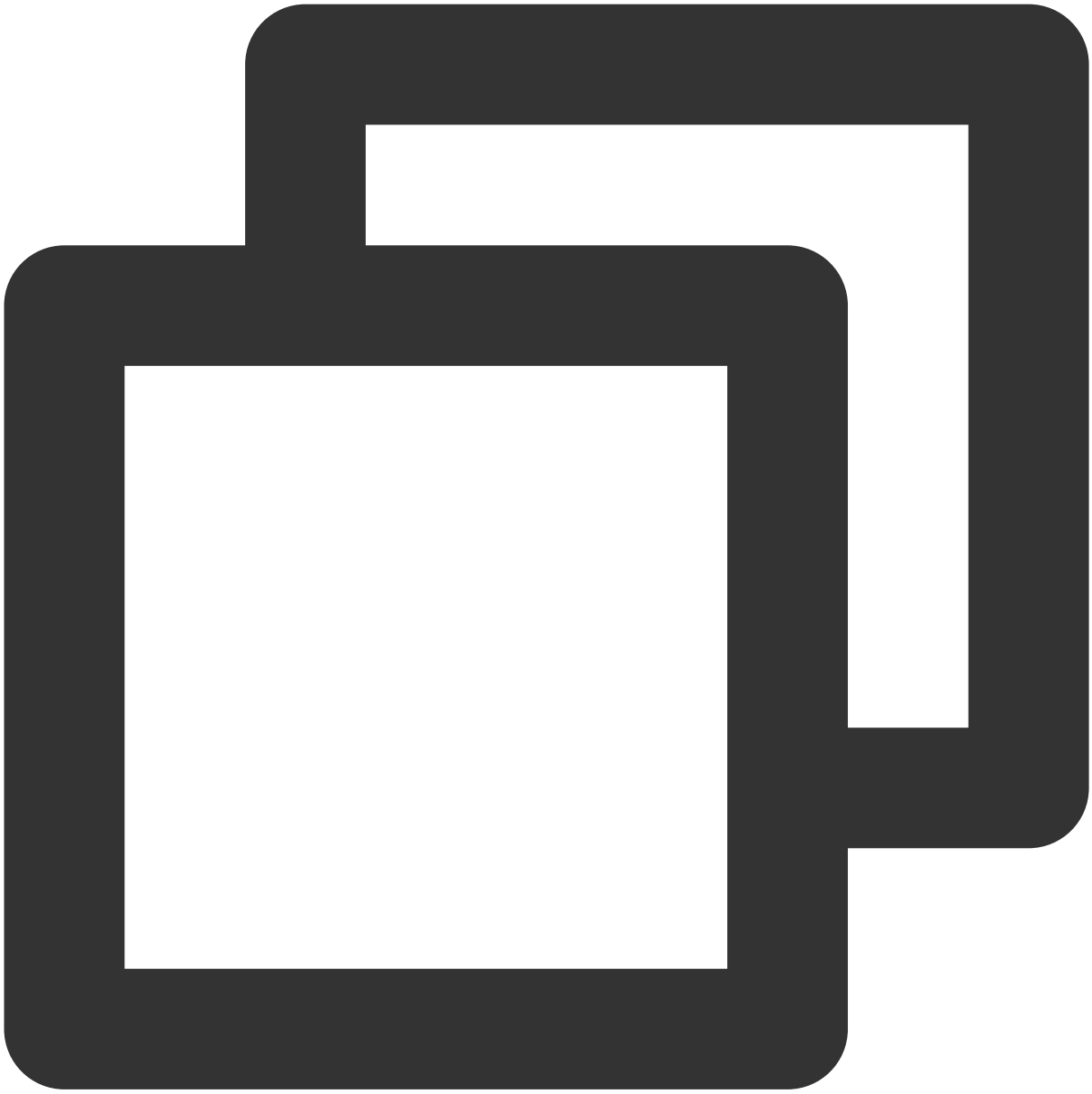
`/data/test.txt` in the example can be replaced with any valid GooseFS path. If data is stored in the remote UFS, you can run the `load` command to reload the data.

Note:

You are advised to set cache replacement policies to automatically clear GooseFS historical data.

2. Load data to GooseFS

Run the `load` command to load data to GooseFS. In the example below, the data has been 100% loaded.



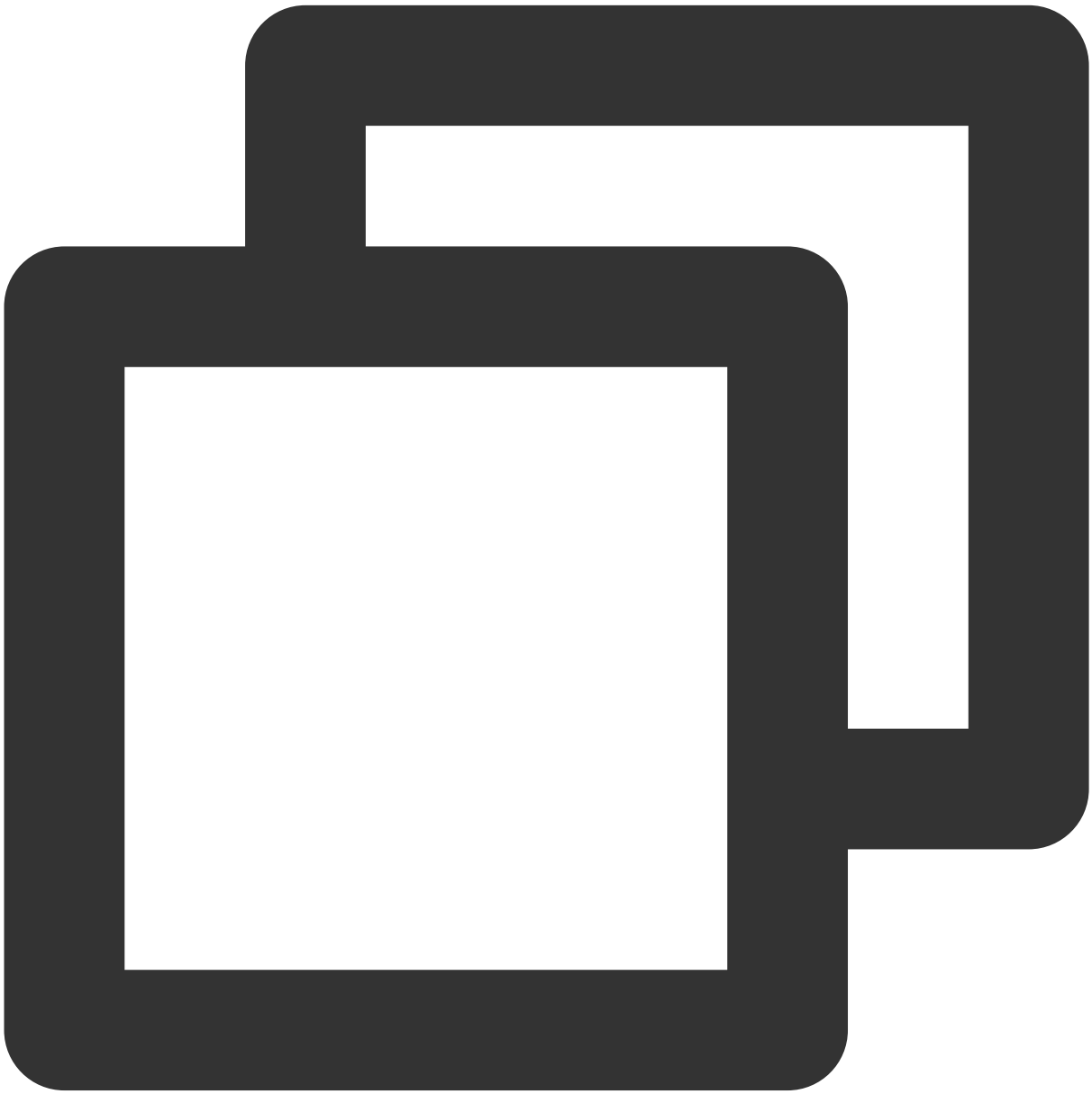
```
$ goosefs fs load /data/test.txt
/data/test.txt loaded
$ goosefs fs ls /data/test.txt
-rw-rw-rw-  hadoop         hadoop                        14       PERSISTED 03-11-202
```

`/data/test.txt` in the example can be replaced with any valid GooseFS path. To load data from a local file system, use the `copyFromLocal` command. Note that data loaded from a local file system will not be stored to the

remote UFS. By default, data will be cached to GooseFS at the first access. Therefore, you don't need to run the `load` command in most cases. However, if you need your data loaded into the cache in advance, you can run this command.

3. Storing GooseFS data persistently

Run the `persist` command to save the data to the remote UFS:



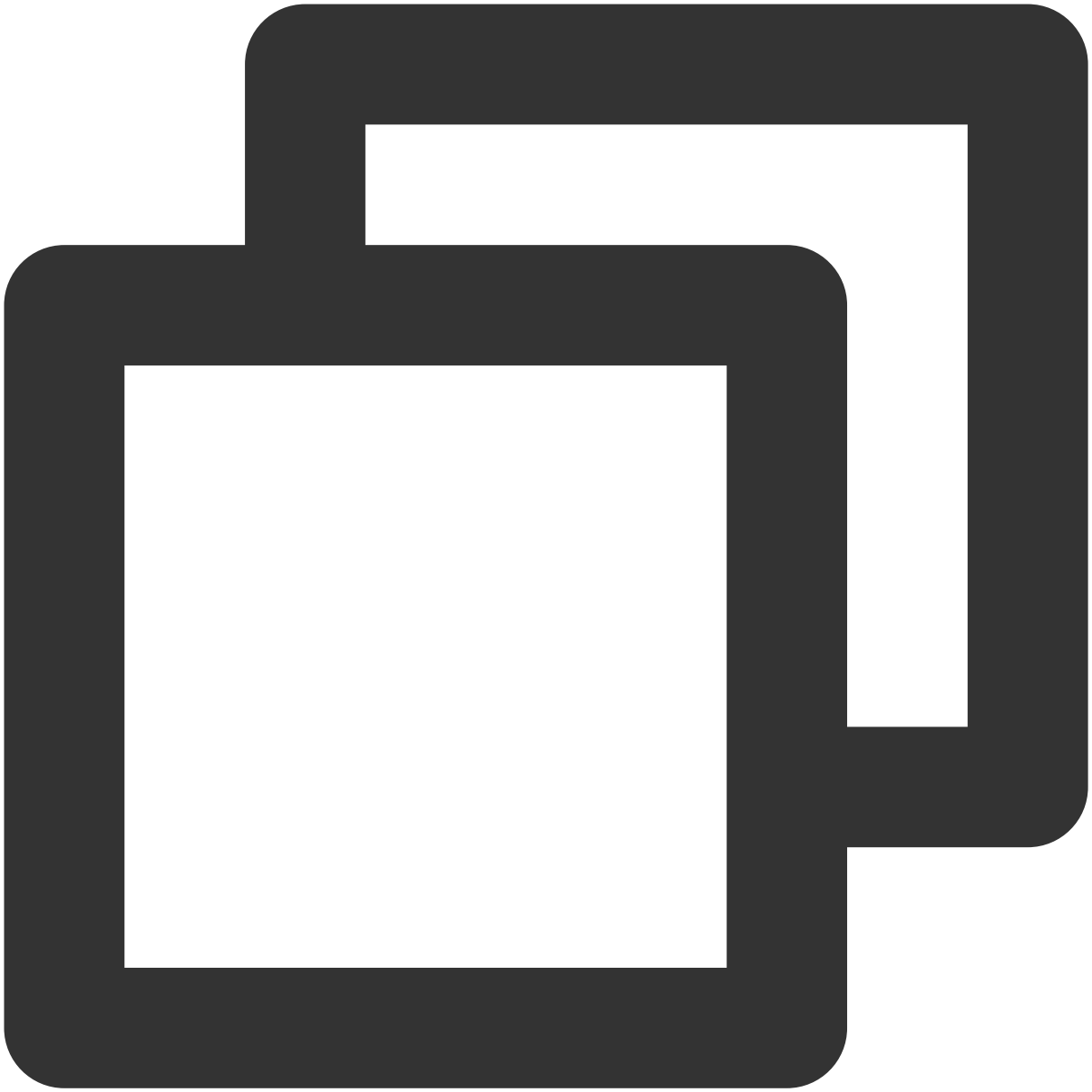
```
$ goosefs fs persist /data/test.txt
$ goosefs fs ls /data/test.txt
-rw-rw-rw-  hadoop         hadoop                       14       PERSISTED 03-11-202
```

`/data/test.txt` in the example above can be replaced with any valid GooseFS path. You are advised to configure cache replacement policies to automatically store data persistently.

4. Adding TTL

GooseFS allows you to add a time to live (TTL) value for any directory or file in a namespace. TTL ensures that your data can be deleted as scheduled to free space for new files and make full use of the local disk. The TTL values of GooseFS files and directories can be the metadata so that these TTL values will still take effect even if the cluster is restarted. The checker program in the background will check TTL values periodically and clear expired data automatically.

The checker's interval and action can be set in `goosefs-site.preperties`. In the example below, the checker's interval is set to 10 minutes, and the action is set to `FREE`.



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

After the configuration, GooseFS background threads will inspect expired files every 10 minutes. If any expired file is found during an inspection cycle, the file's cache will be freed when the next cycle completes. For example, if a file is found expired in the cycle that ends at 00:00:00, the file's cache will be cleared at 00:10:00.

Note:

The default unit of the interval is ms (millisecond). You can use a unit such as s (second), m (minute), or h (hour) when you set the input parameter. For more information, please see the configuration description.

Data Replication

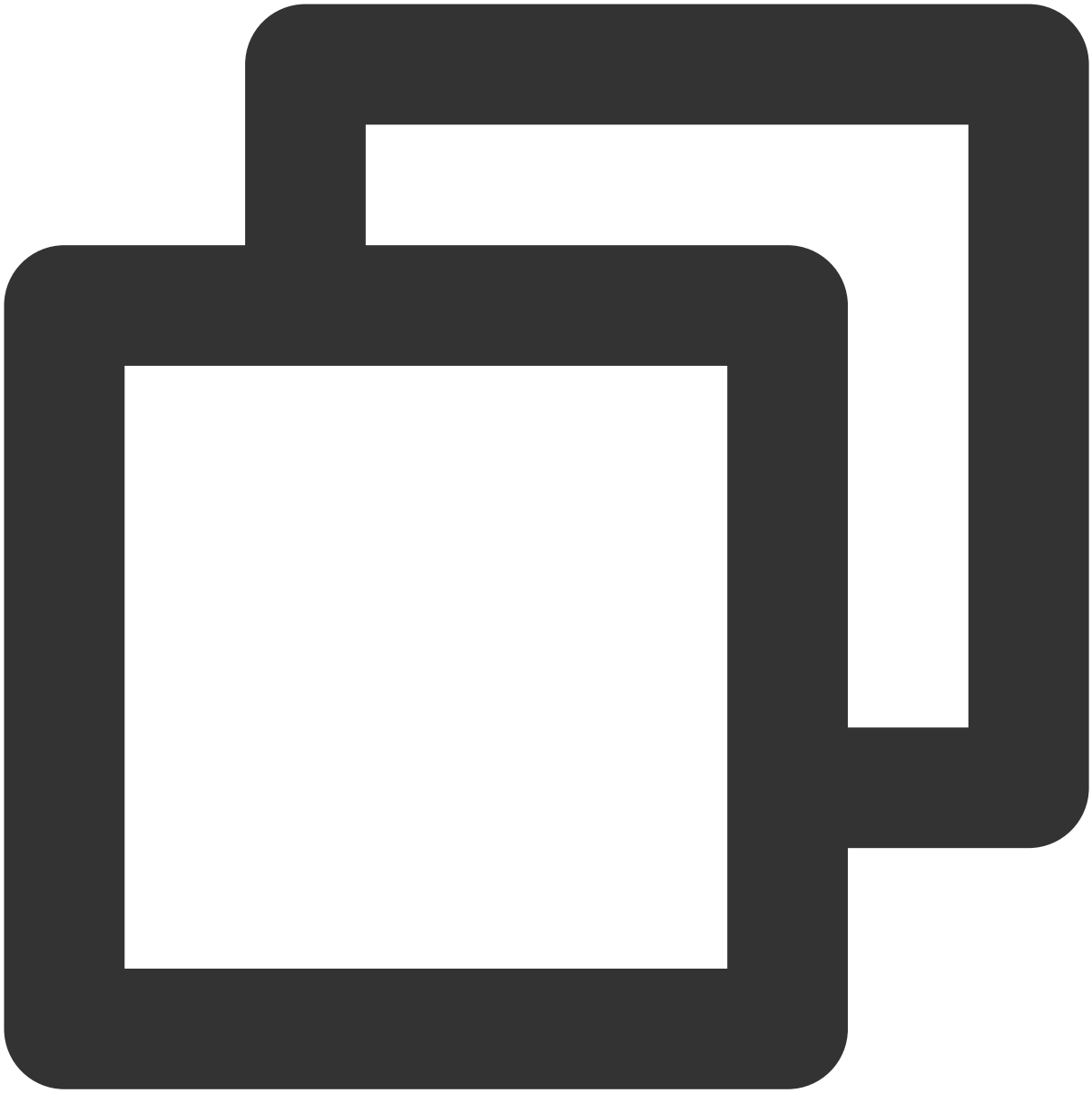
Data replication is classified into active replication and passive replication.

1. Passive replication

In GooseFS, each file has one or more data blocks spread across the cluster. By default, GooseFS automatically adjusts the number of replicas for data blocks according to the load and capacity. Passive replication is performed when:
The same block is read by multiple clients at the same time. As a result, multiple blocks reside in different workers.
When local reads are prioritized, but the data is not found from the local environment, remote reads will be initialized, and the data will be saved to the local worker.
If the number of replicas generated by passive replication is greater than the configured quantity, the excess will be deleted asynchronously, which is unperceivable to users.

2. Active replication

Active replication is implemented by setting the number of replicas for a file. If the number of blocks is smaller than the configured quantity, blocks will be made up asynchronously. If there are excessive blocks, excessive replicas will be deleted. You can run the following command to configure active replication:



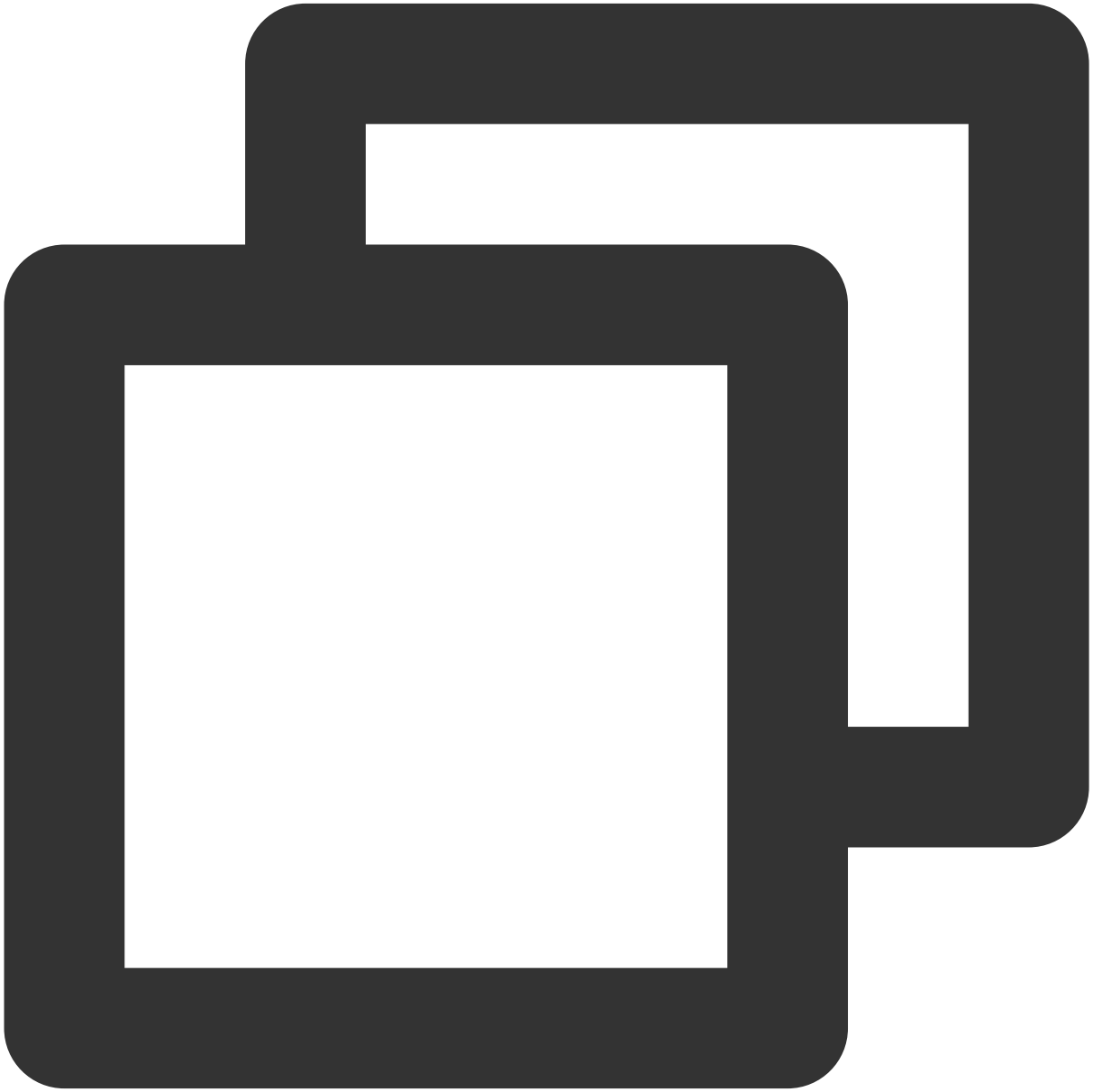
```
$ goosefs fs setReplication  [-R] [--max | --min ] <path>
```

Parameters are described as follows:

max: the maximum number of replicas (default value: `-1`, indicating no upper limit). If this parameter is set to `0`, the file will not be cached in GooseFS. Usually, you can set this parameter to a positive integer. After the configuration, GooseFS will check the replica quantity and delete the excessive ones.

min: the minimum number of replicas (default value: `0`, indicating no replica will be retained when the data expires). Usually, you can set this parameter to a positive integer, which must be **smaller than** `max`. After the configuration,

GooseFS will check the replica quantity and make up the quantity automatically if the replica quantity is smaller than the configured value.
path: a directory or a file path
R: recursively replicates all files and sub-directories of a directory (if `path` is set to a directory) following `min` and `max`.
You can run the `stat` command to view the replication of a file. In the example below, `replicationMax` of the `/data/test.txt` file is set to `-1`, which means that the file will be deleted once it expires.

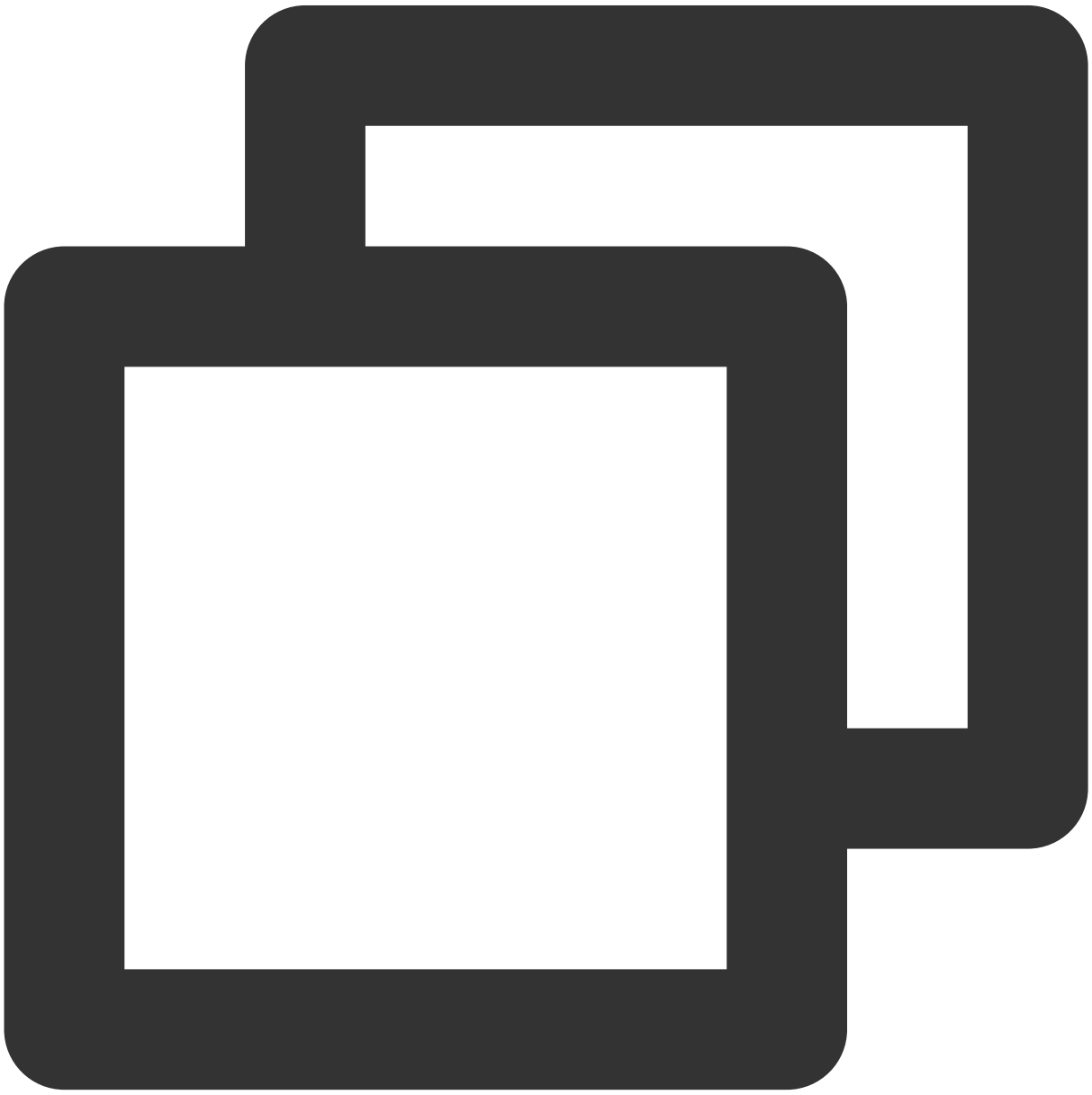


```
$ goosefs fs stat /data/test.txt
/data/test.txt is a file path.
```

```
FileInfo{fileId=50331647, fileIdentifier=null, name=test.txt, path=/data/test.txt,
This file does not contain any blocks.
```

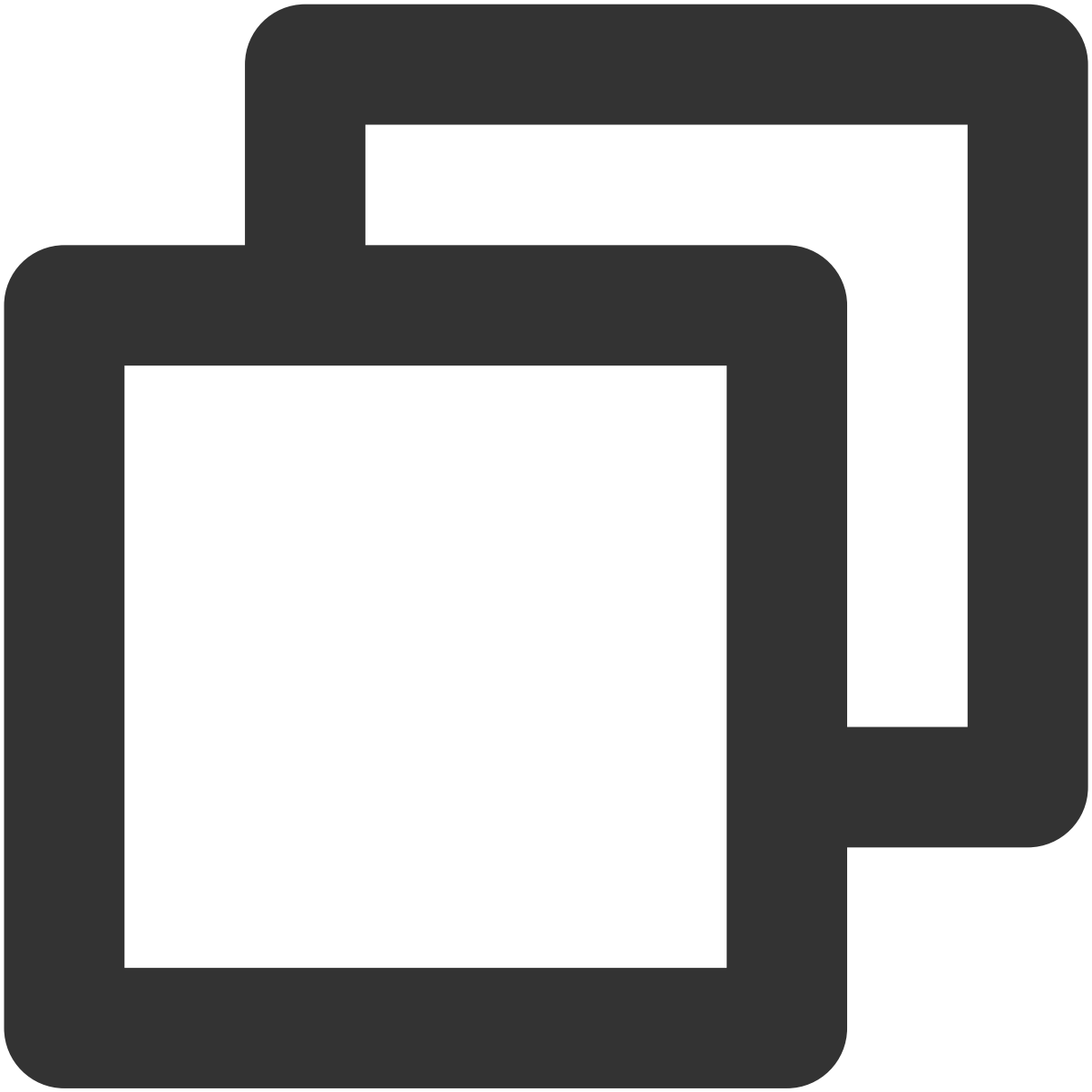
Querying Cache Usage

GooseFS will record the local cache and storage usage. You can run the commands below to view the running status of GooseFS for local cache management and maintenance.
View the cache usage, in bytes:



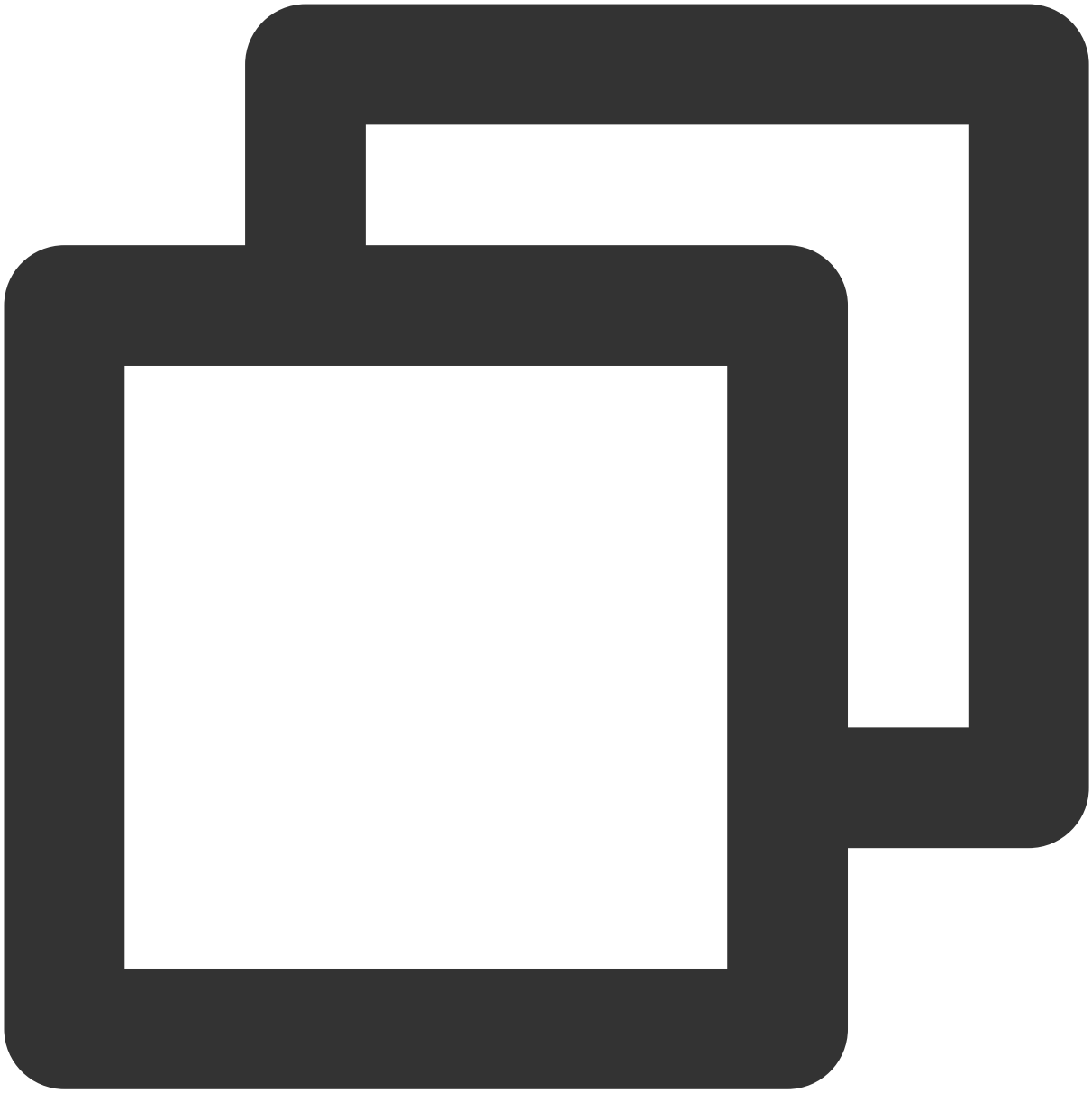
```
$ goosefs fs getUsedBytes
Used Bytes: 0
```

View the total cache capacity, in bytes:



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

View the cache usage report:



```
$ 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
```

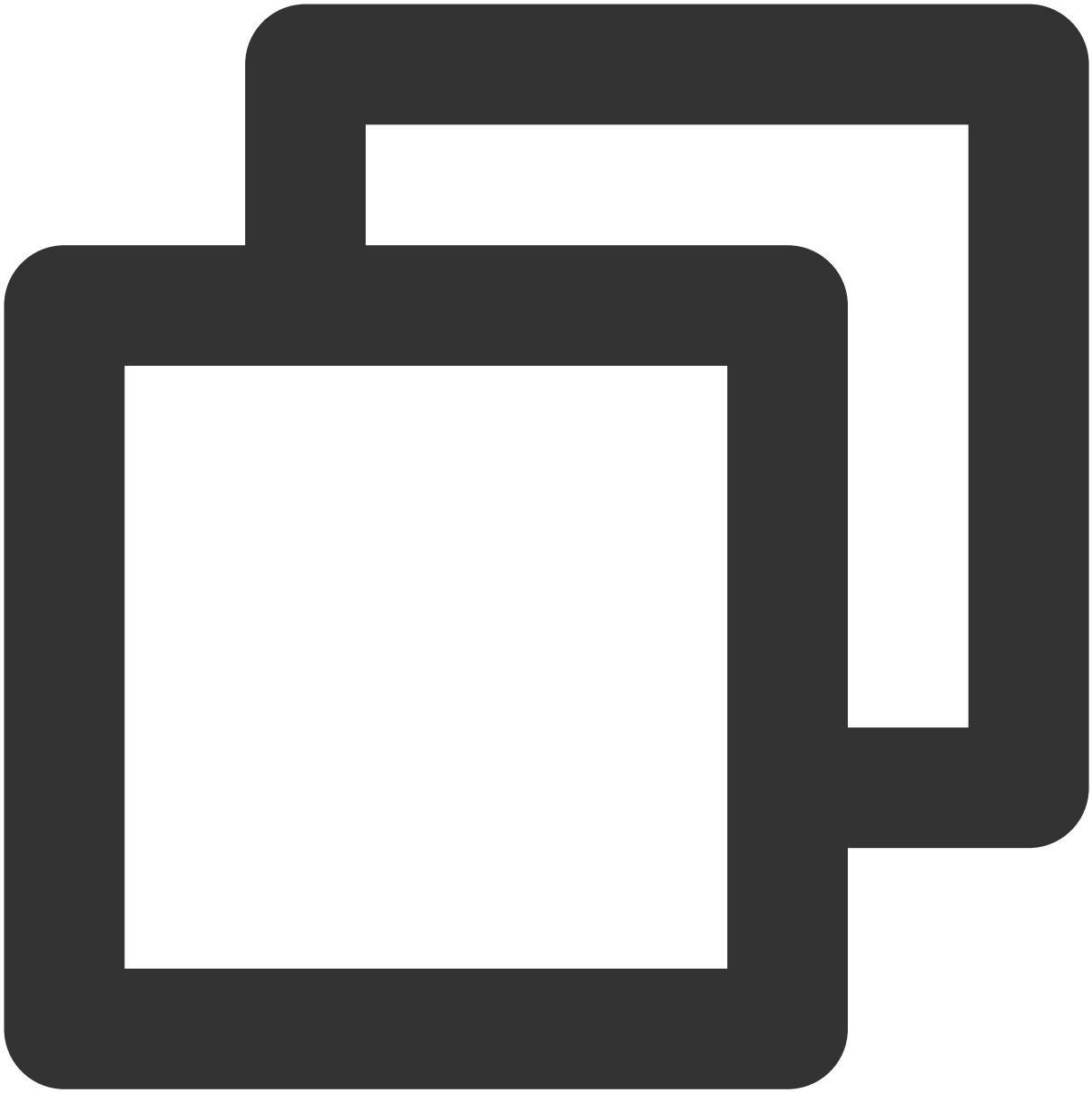
```
    Lost Workers: 0
    Total Capacity: 1500.00GB
        Tier: HDD  Size: 1500.00GB
    Used Capacity: 0B
        Tier: HDD  Size: 0B
    Free Capacity: 1500.00GB
```

Transparent Acceleration

Last updated：2024-03-25 16:04:01

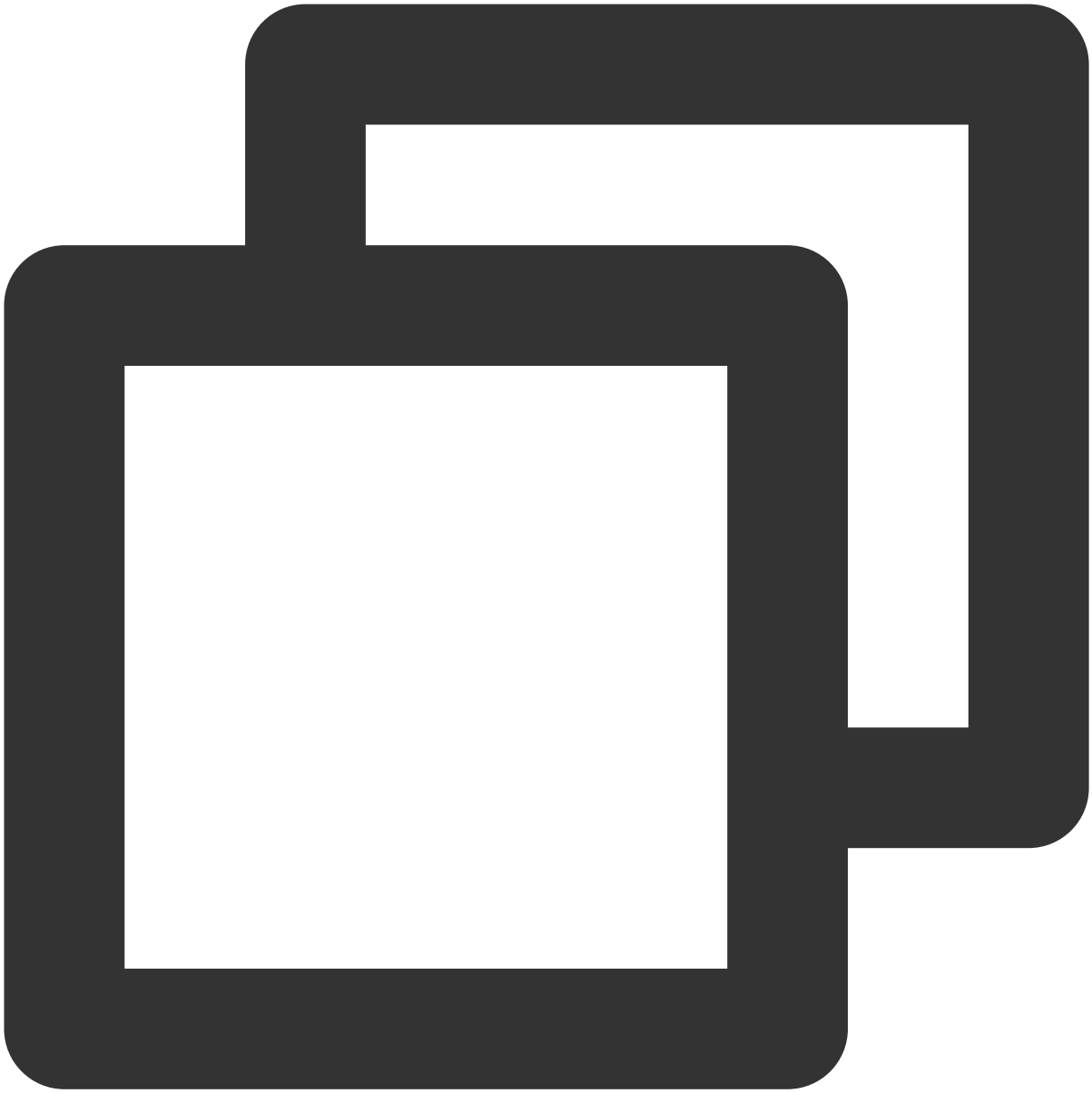
Overview

Transparent acceleration is used to accelerate CosN access to COS. The COS-based CosN is implemented with a standard Hadoop file system, aiming to help integrate big data computing frameworks (such as Hadoop, Spark, and Tez) with COS. You can use CosN to read and write data stored in COS. If you are already using CosN to access COS, you can use GooseFS's client-based path mapping method to access GooseFS with the CosN schema without modifying the definition of the current Hive table to facilitate the comparison tests of GooseFS features/performance. If you are a CHDFS user, you can modify your configurations so that you can access GooseFS with the OFS schema. The mapping between CosN schema and GooseFS schema is described below.
Assuming that the UFS path of the namespace warehouse is `cosn://examplebucket-1250000000/data/warehouse/` , the CosN-to-GooseFS path mapping will be as follows:



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

GooseFS-to-CosN path mapping:



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

The CosN schema maintains the mapping between GooseFS and UFS CosN paths on the client, and converts CosN-path requests to GooseFS-path requests. The mapping is refreshed periodically. You can modify the refresh interval (default: 60s) using `goosefs.user.client.namespace.refresh.interval` in the GooseFS configuration file `goosefs-site.properties` .

Note:

If the accessed CosN path cannot be converted into a GooseFS path, the corresponding Hadoop API call will throw an exception.

Example

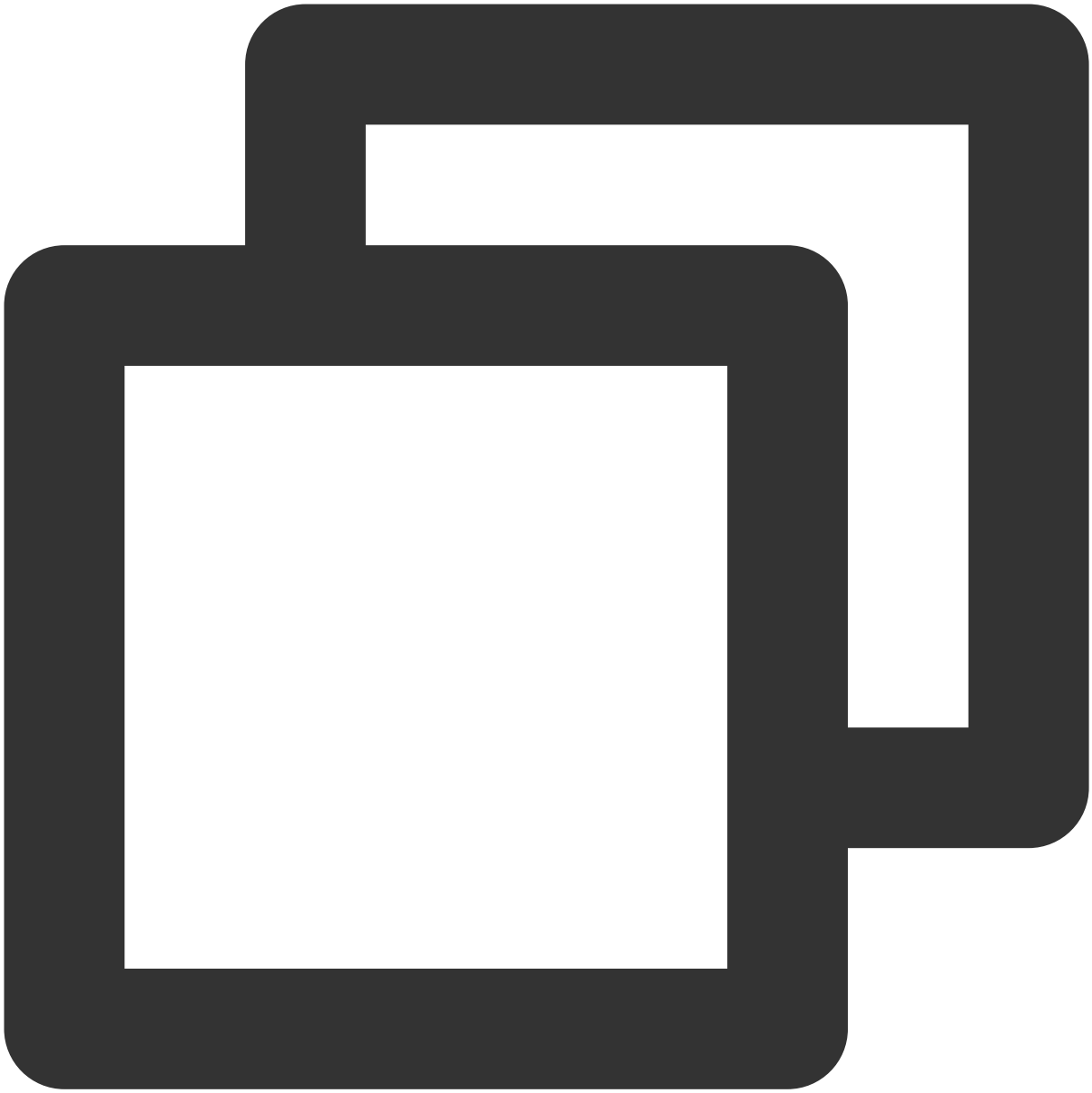
This example shows how to access GooseFS using schemas gfs://, cosn://, and ofs:// on the Hadoop command-line tool and Hive.

1. Prepare the data and computing cluster

Creating a bucket for testing purposes.
Create a folder named `ml-100k` in the root directory of the bucket.
Download the `ml-100k` dataset from Grouplens and upload the `u.user` file to `<Bucket root directory>/ml-100k`.
Purchase an EMR cluster and configure the HIVE component by referring to the EMR documentation.

2. Configure the environment

i. Put the GooseFS client package `goosefs-1.0.0-client.jar` in the `share/hadoop/common/lib/` directory.

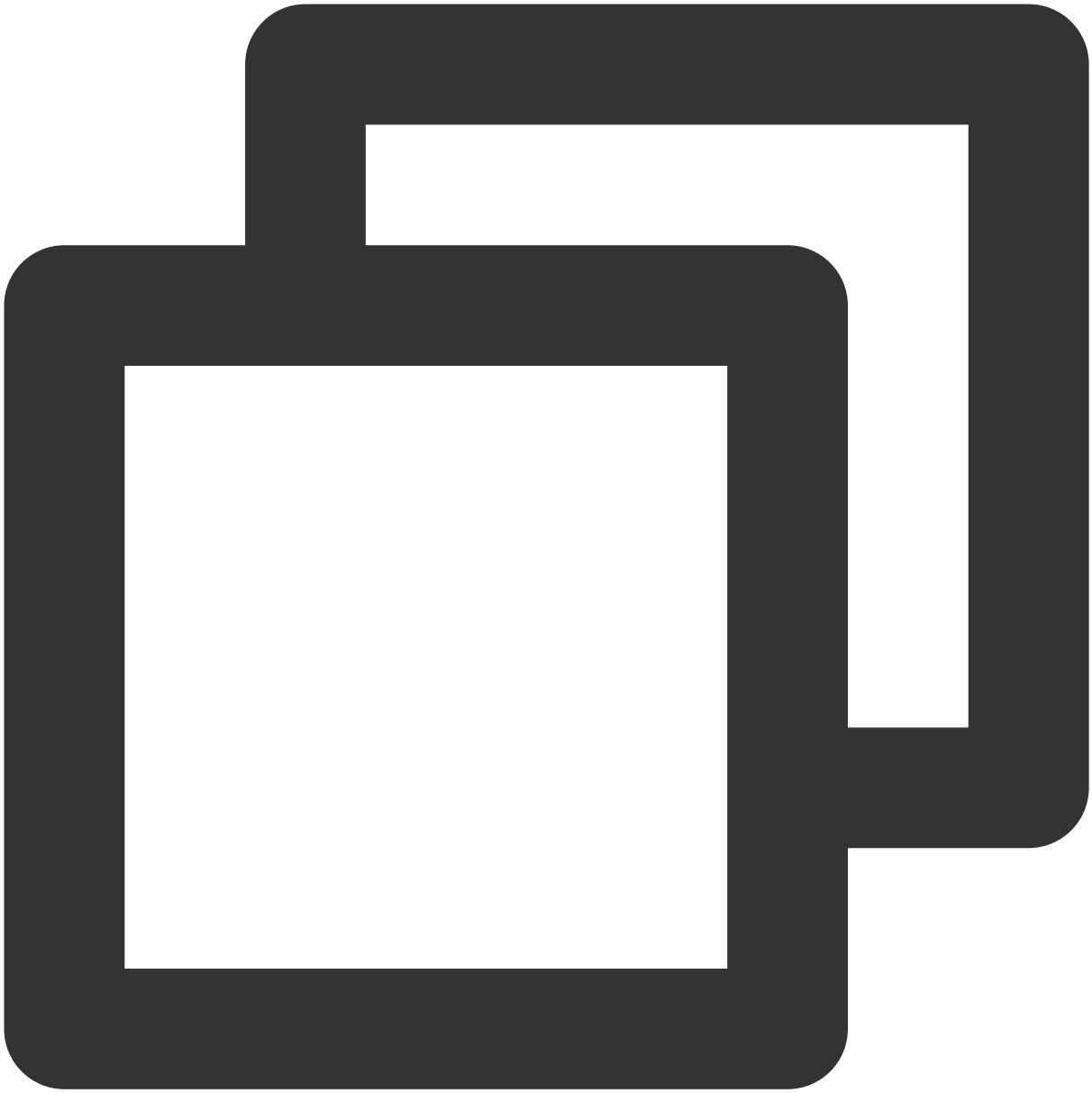


```
cp goosefs-1.0.0-client.jar  hadoop/share/hadoop/common/lib/
```

Note:

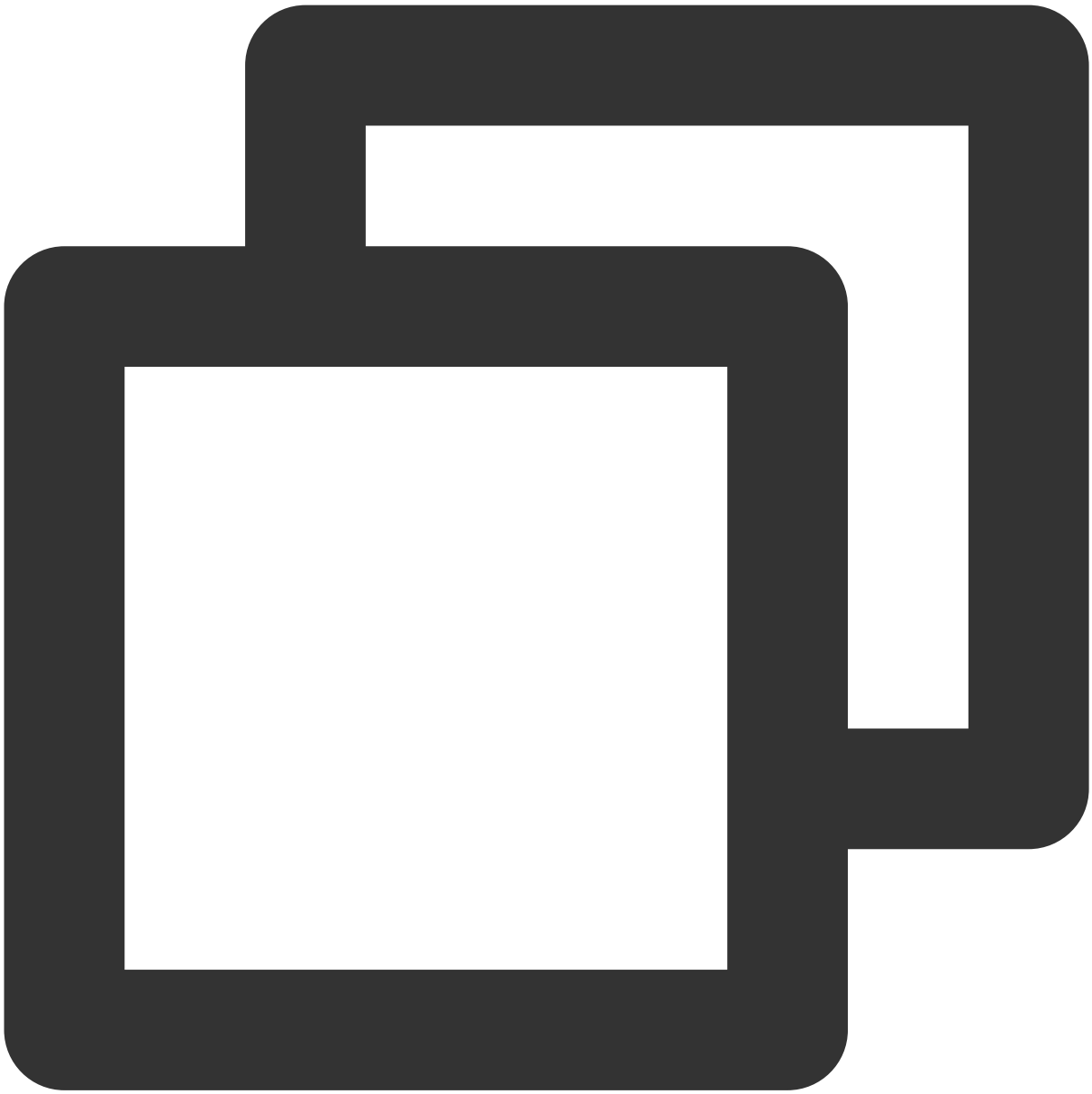
The configuration update and JAR package should be synced to all nodes in the cluster.

ii. Modify the Hadoop configuration file `etc/hadoop/core-site.xml` to specify the GooseFS class implementation.



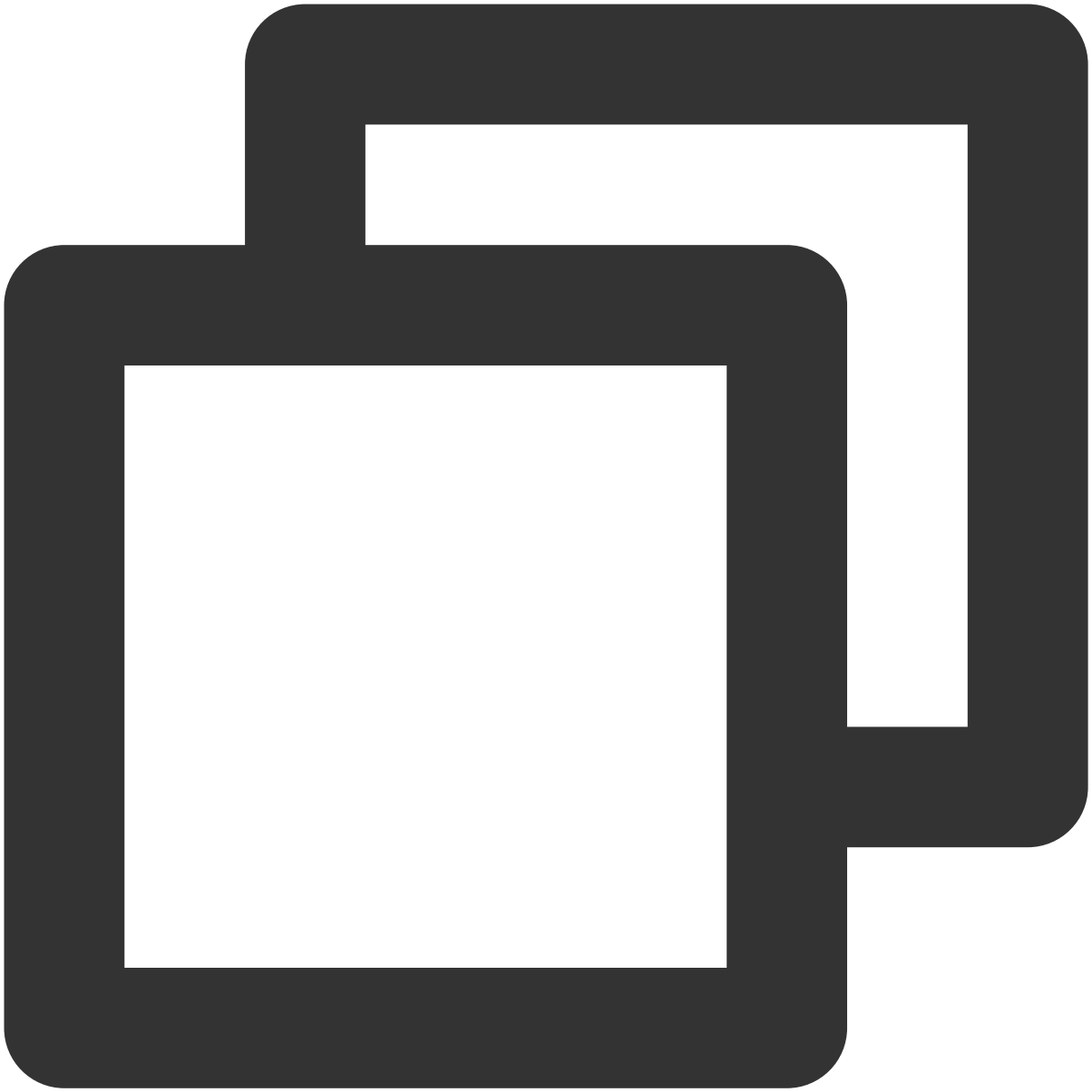
```
<property>
  <name>fs.AbstractFileSystem.gfs.impl</name>
  <value>com.qcloud.cos.goosefs.hadoop.GooseFileSystem</value>
</property>
<property>
  <name>fs.gfs.impl</name>
  <value>com.qcloud.cos.goosefs.hadoop.FileSystem</value>
</property>
```

iii. Run the following Hadoop command to check whether you can access GooseFS using the `gfs://` schema, where <MASTER_IP> is the IP of the master.



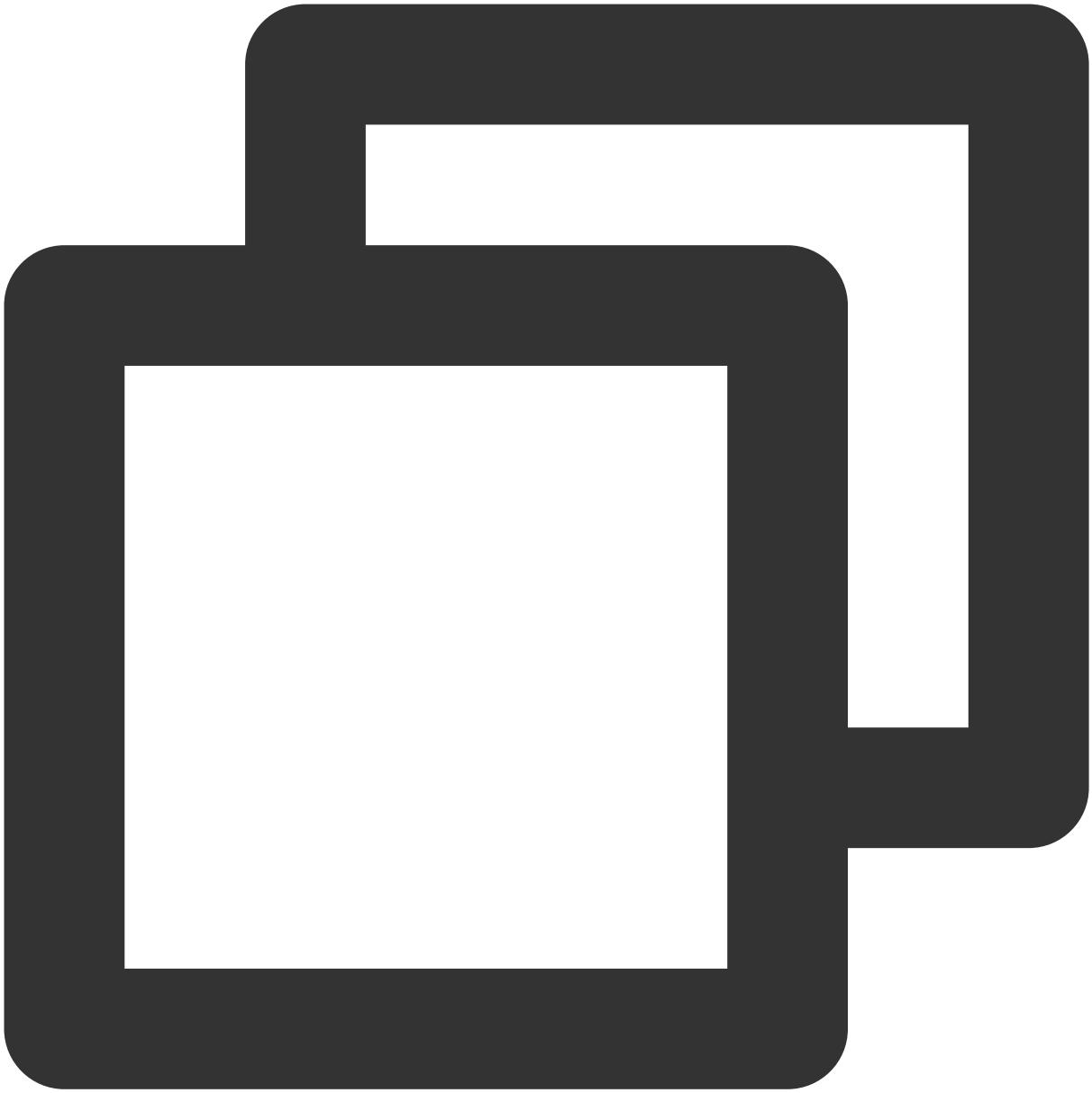
```
hadoop fs -ls gfs://<MASTER_IP>:9200/
```

iv. Put the JAR package of the GooseFS client to the `hive/auxlib/` directory for Hive to load it.



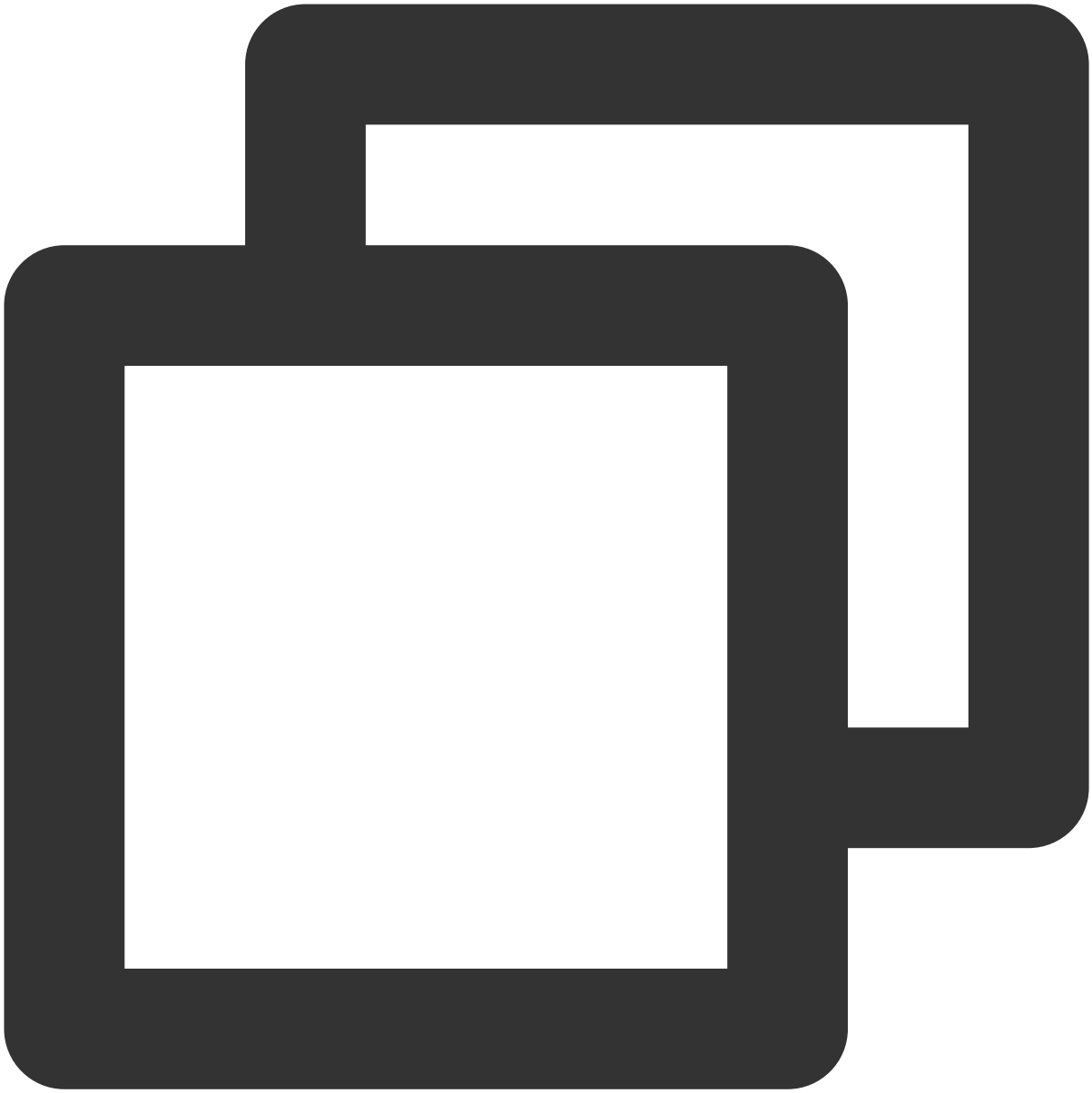
```
cp goosefs-1.0.0-client.jar  hive/auxlib/
```

v. Run the following command to create a namespace whose UFS schema is CosN and list the namespace. Replace `examplebucket-1250000000` with your actual COS bucket, and `SecretId` and `SecretKey` with your actual key information.



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

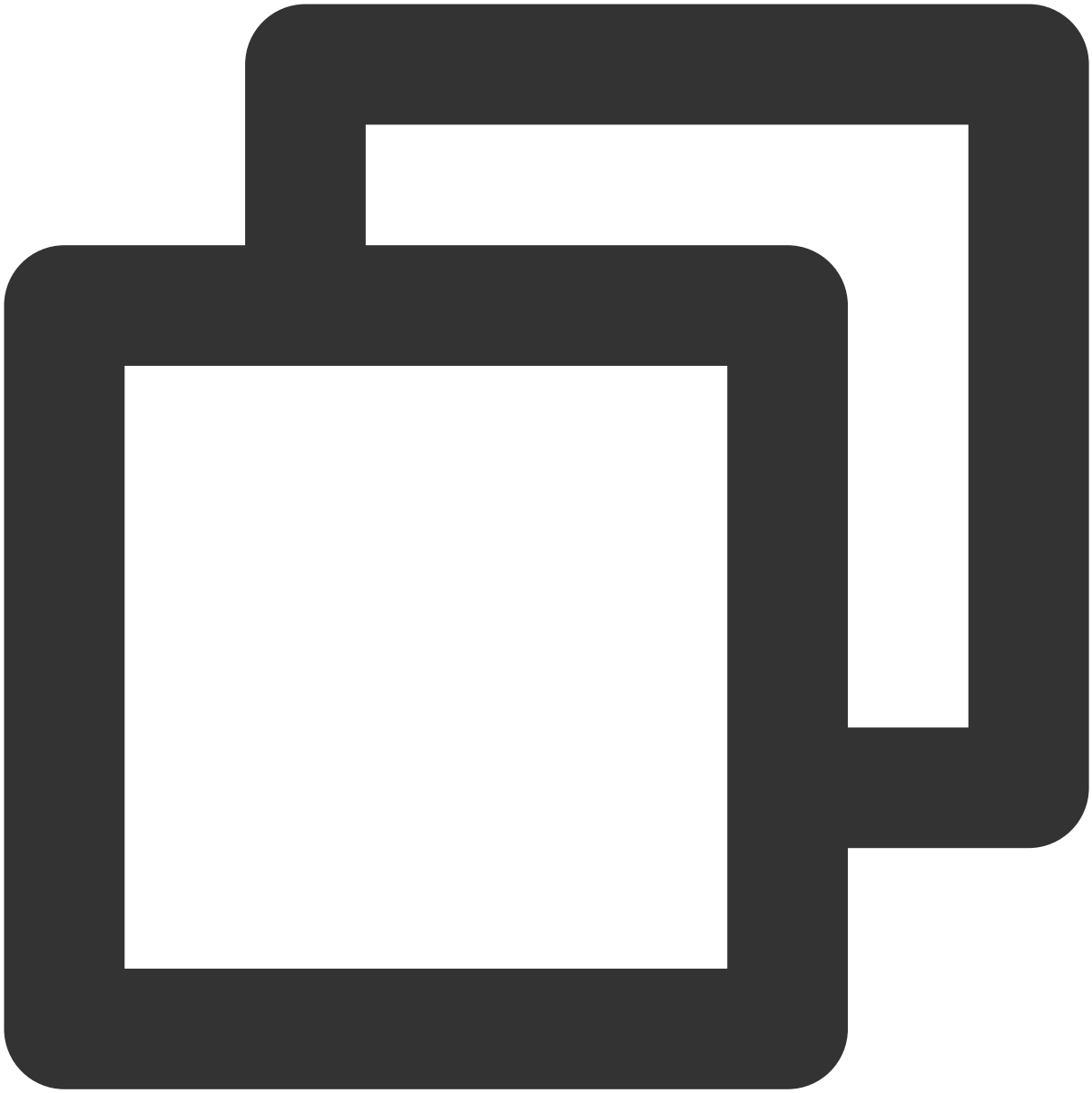
vi. Run the following command to create a namespace whose UFS schema is OFS and list the namespace. Replace `instance-id` with the actual ID of your CHDFS instance, and `1250000000` with your actual `APPID` :



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

3. Create a GooseFS schema and query data

Run the following commands:

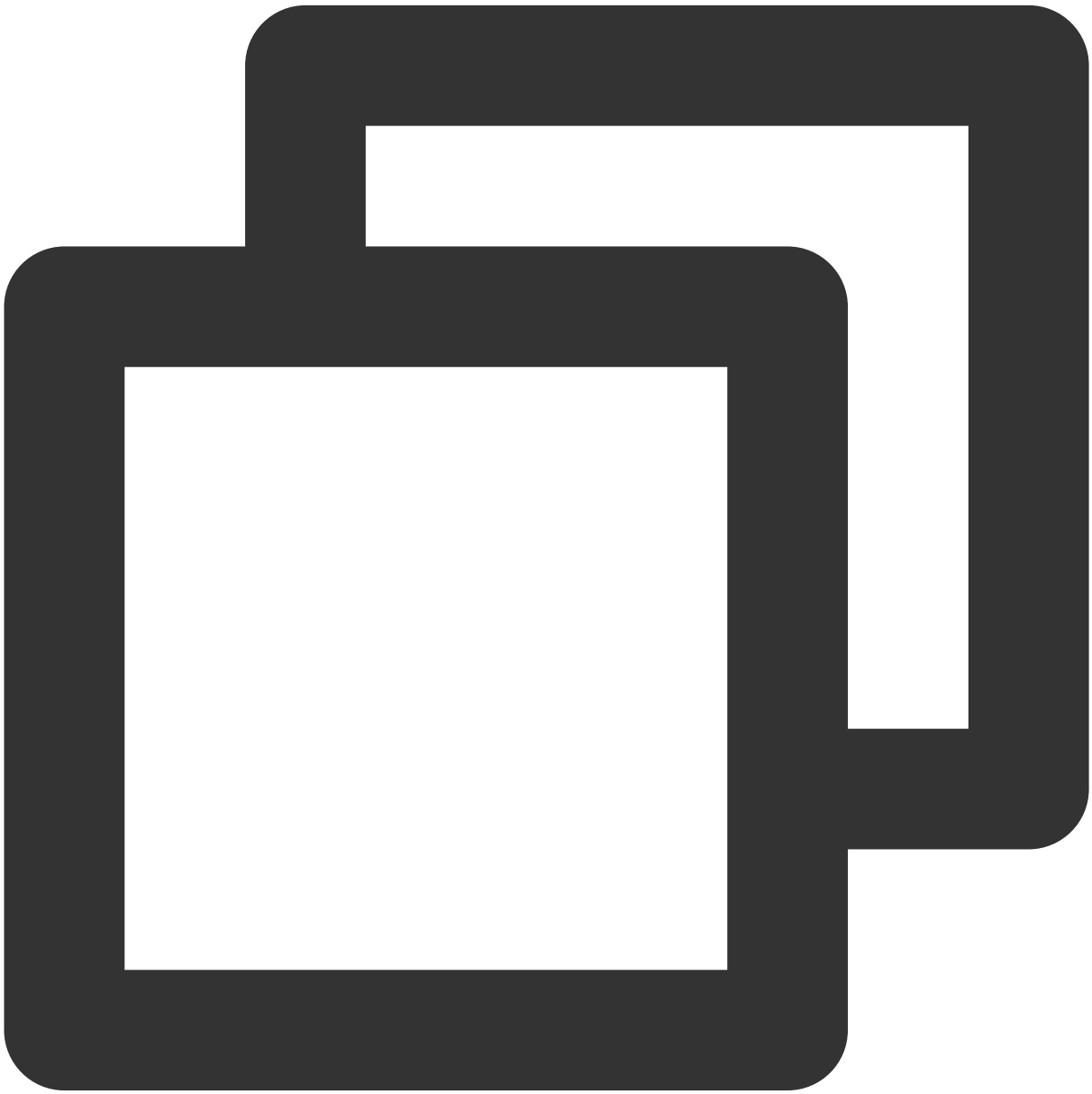


```
create database goosefs_test;

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. Create a CosN schema and query data

Run the following commands:

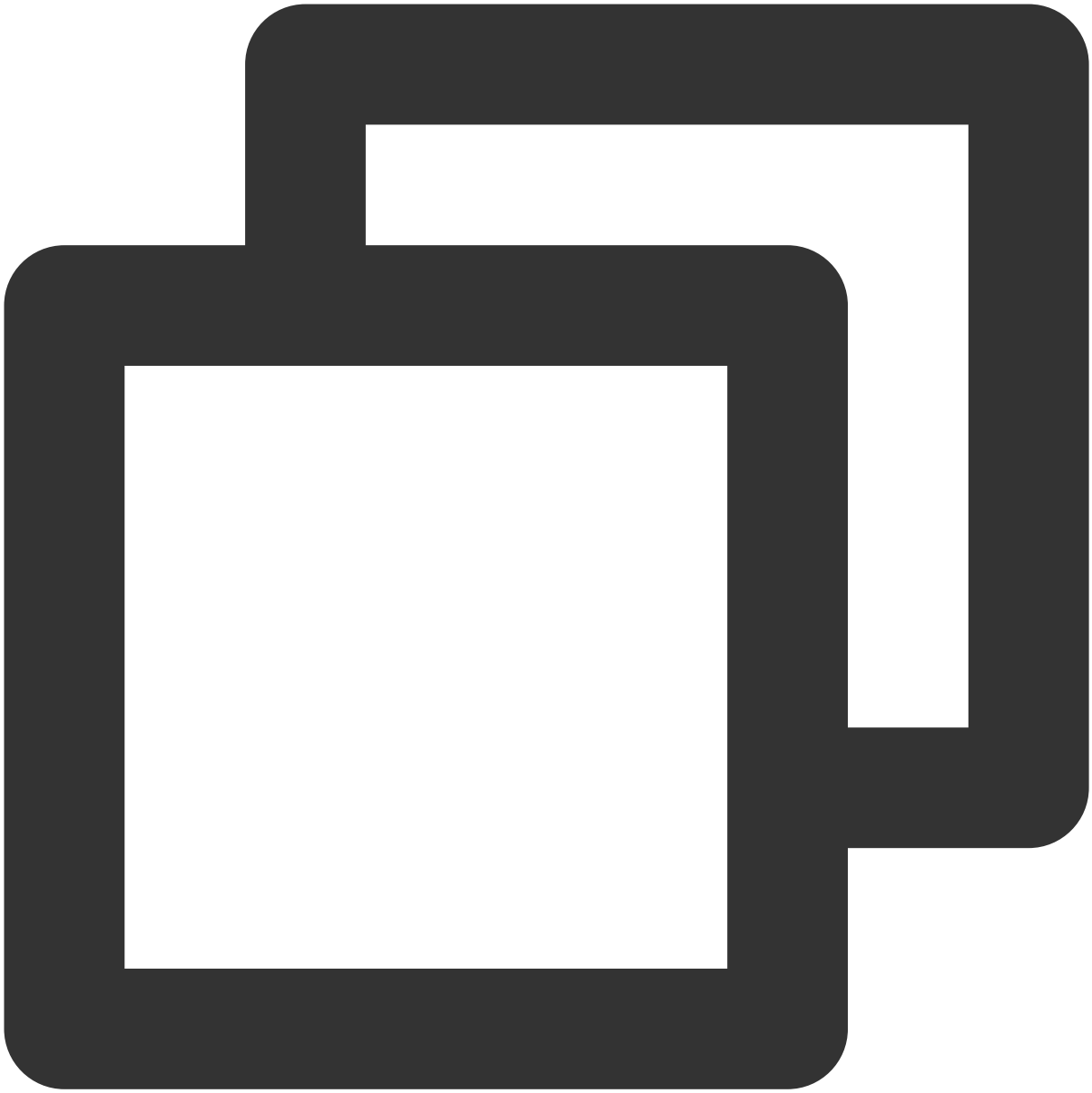


```
CREATE TABLE u_user_cosn (
userid INT,
```

```
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;
```

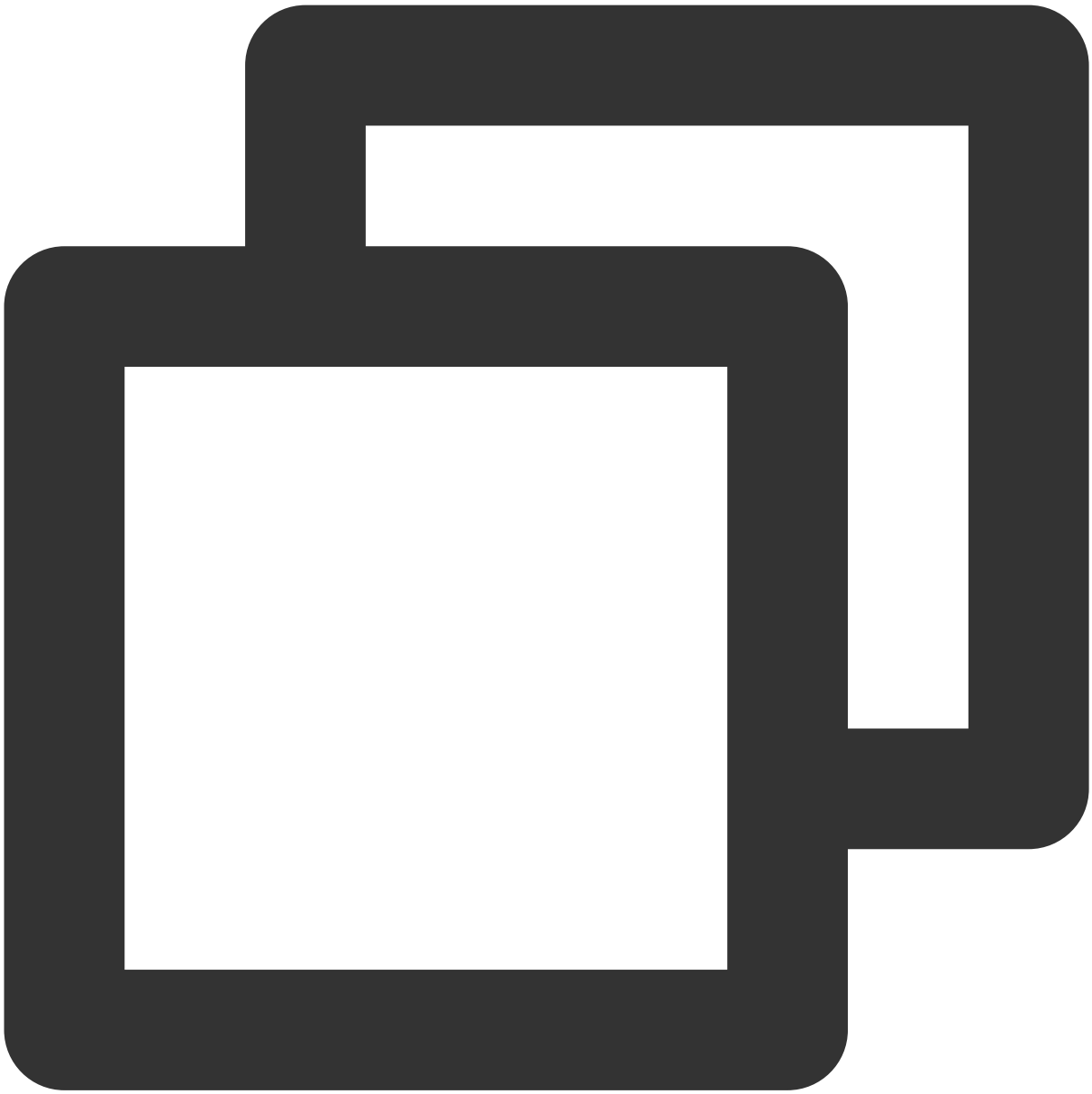
5. Modify the CosN implementation to GooseFS-compatible

Modify `hadoop/etc/hadoop/core-site.xml` :



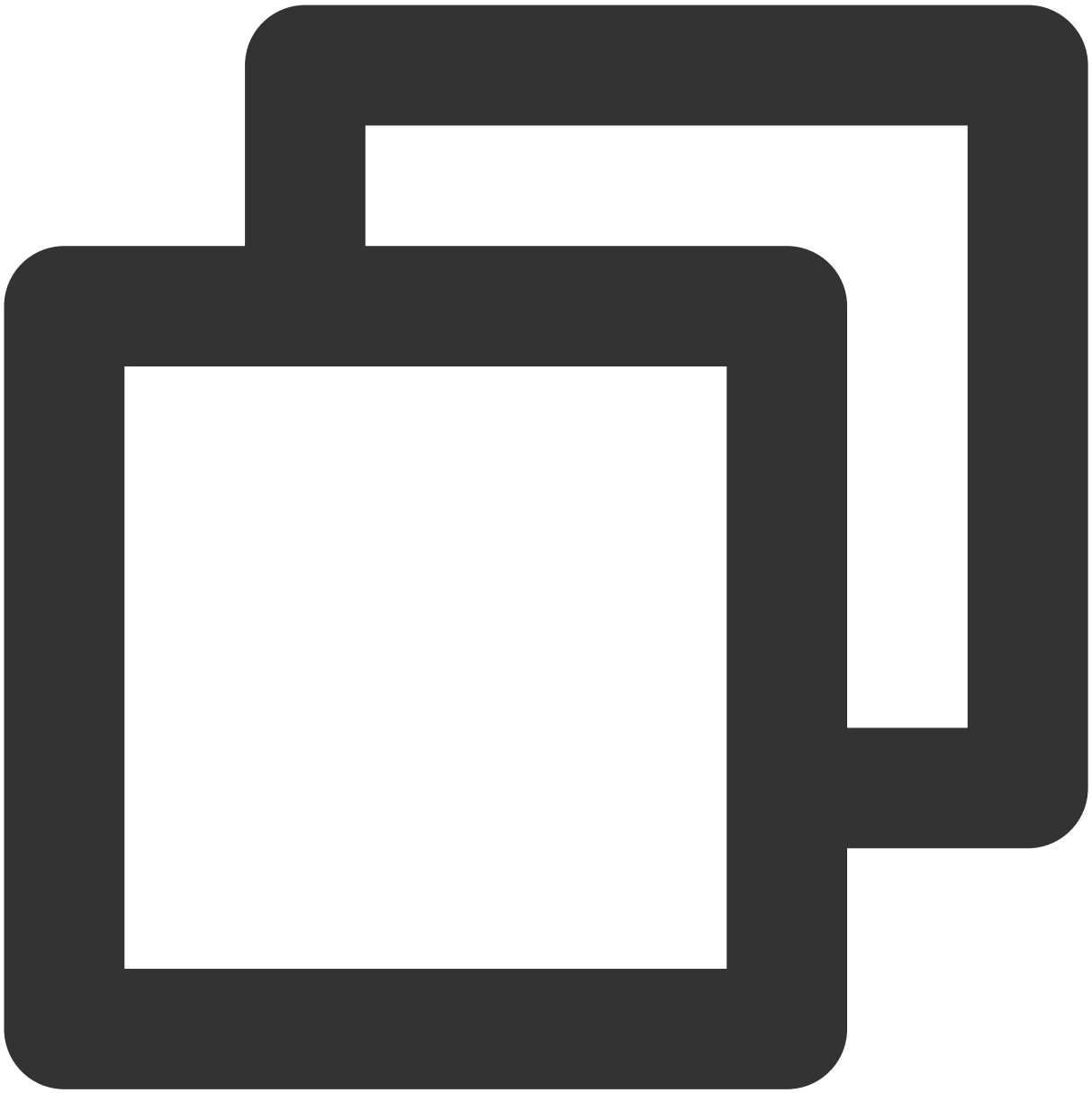
```
 <property>
     <name>fs.AbstractFileSystem.cosn.impl</name>
     <value>com.qcloud.cos.goosefs.hadoop.CosN</value>
</property>
<property>
     <name>fs.cosn.impl</name>
     <value>com.qcloud.cos.goosefs.hadoop.CosNFileSystem</value>
</property>
```

Run the Hadoop command below. If the path cannot be converted into a GooseFS path, an error message will be displayed in the command output.



```
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
```

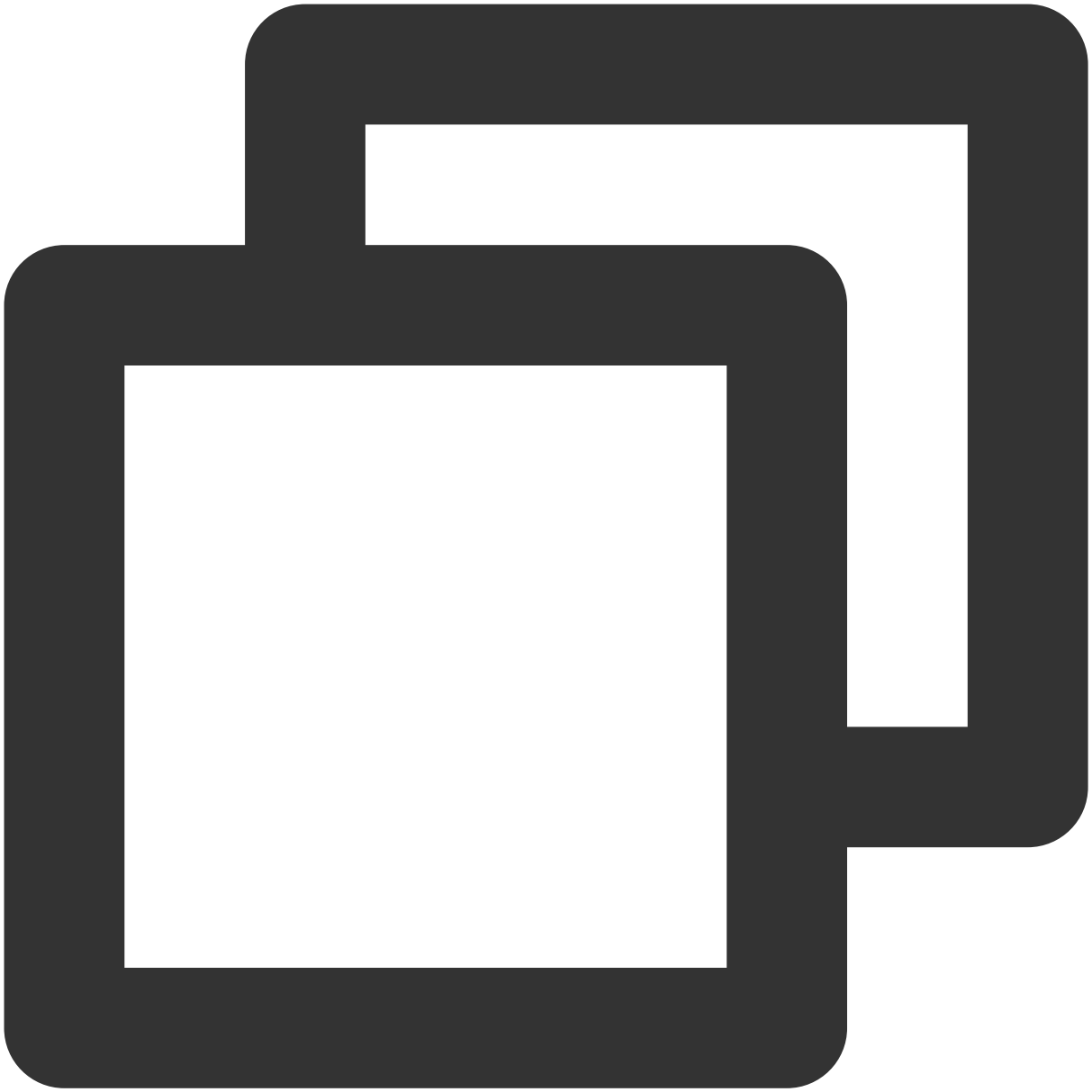
Run the Hive query language again:



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

6. Create an OFS schema and query data

Run the following commands:

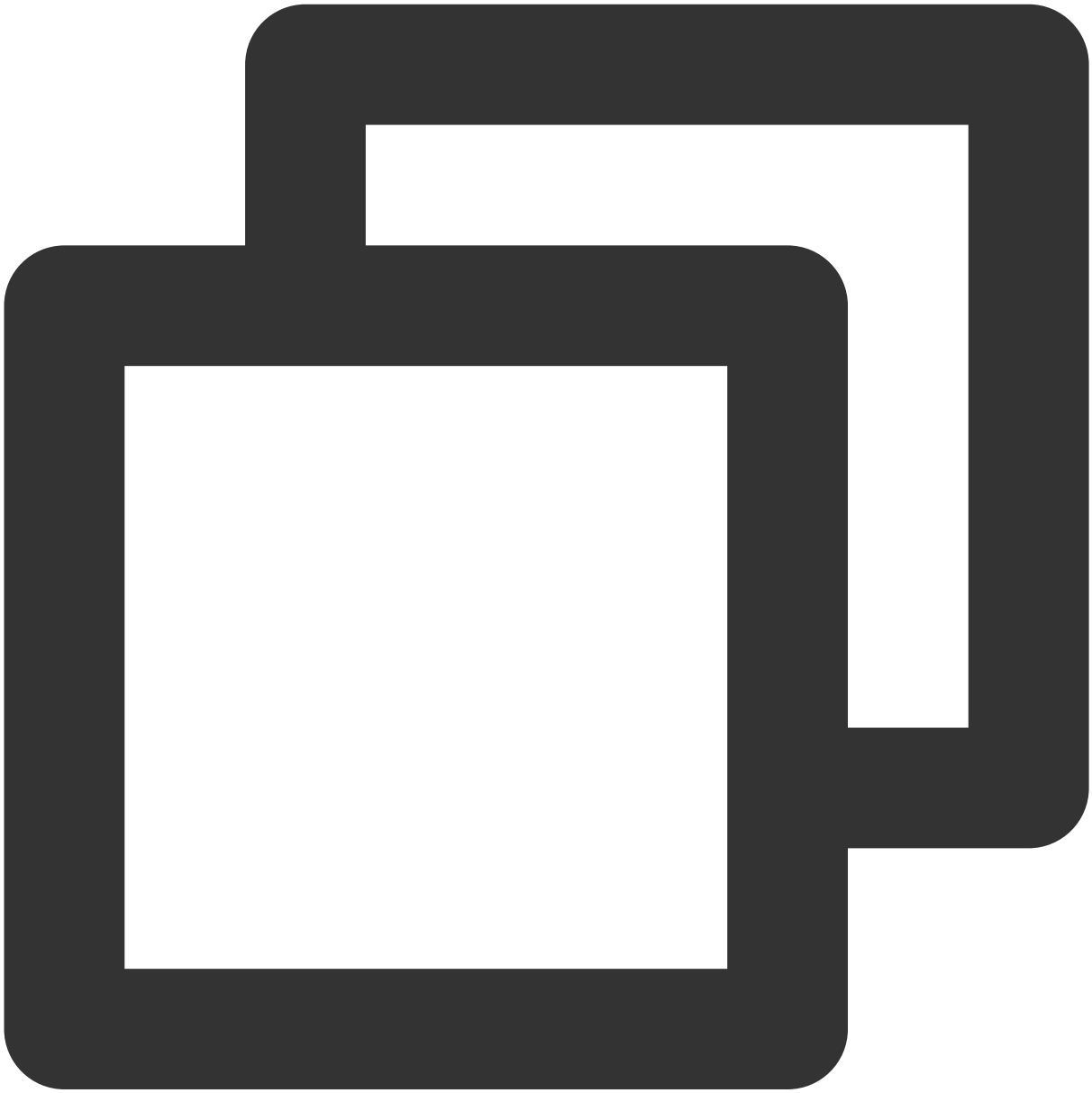


```
CREATE TABLE u_user_ofs (
userid INT,
age INT,
gender CHAR(1),
occupation STRING,
zipcode STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
STORED AS TEXTFILE
LOCATION 'ofs://instance-id.chdfs.ap-guangzhou.myqcloud.com/ofs-test/';
```

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

7. Modify the OFS implementation to GooseFS-compatible

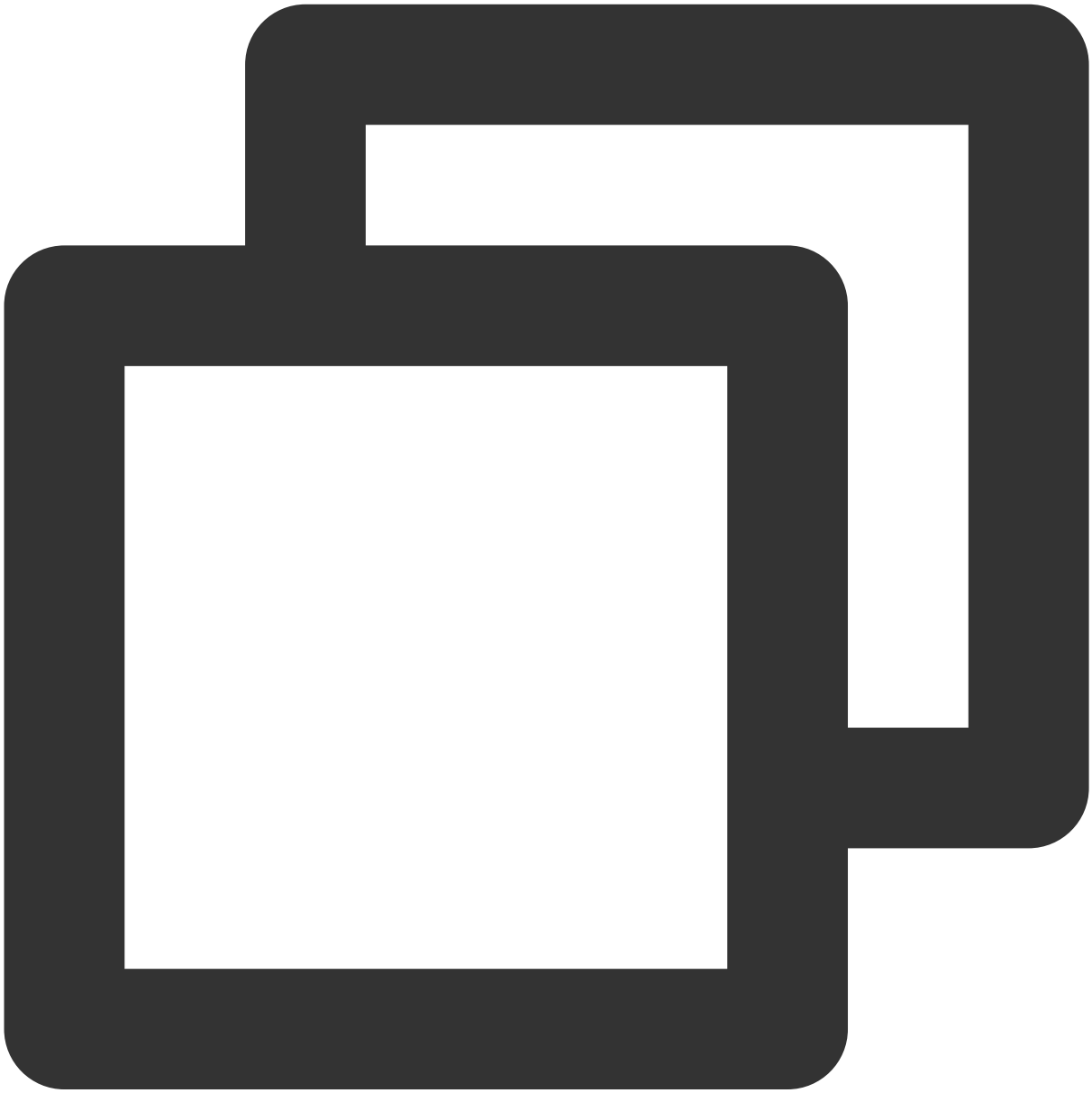
Modify `hadoop/etc/hadoop/core-site.xml` :



```
<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>
```

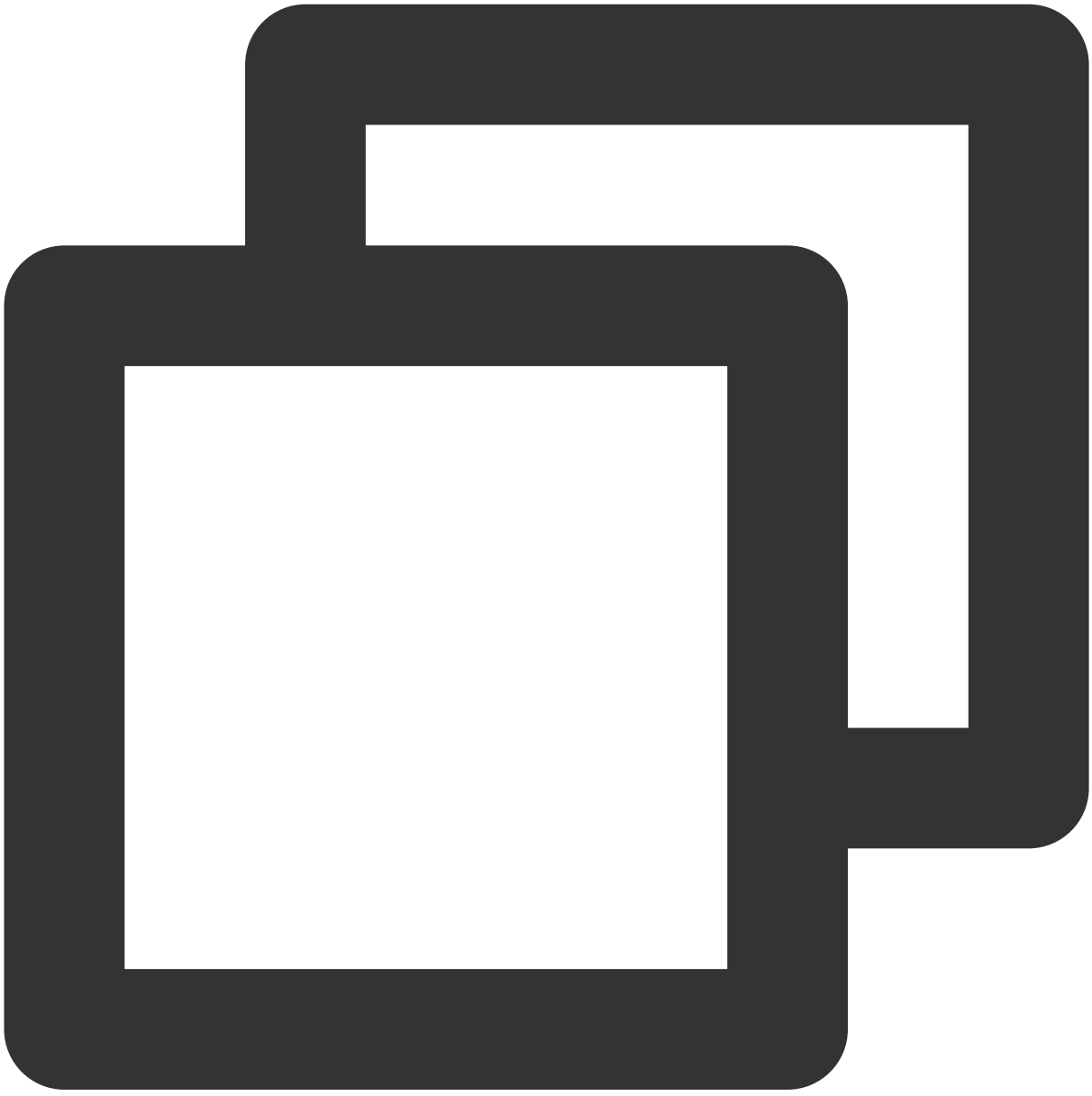
Run the Hadoop command below. If the path cannot be converted into a GooseFS path, an error message will be included in the command output.



```
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
```

Run the Hive query language again:



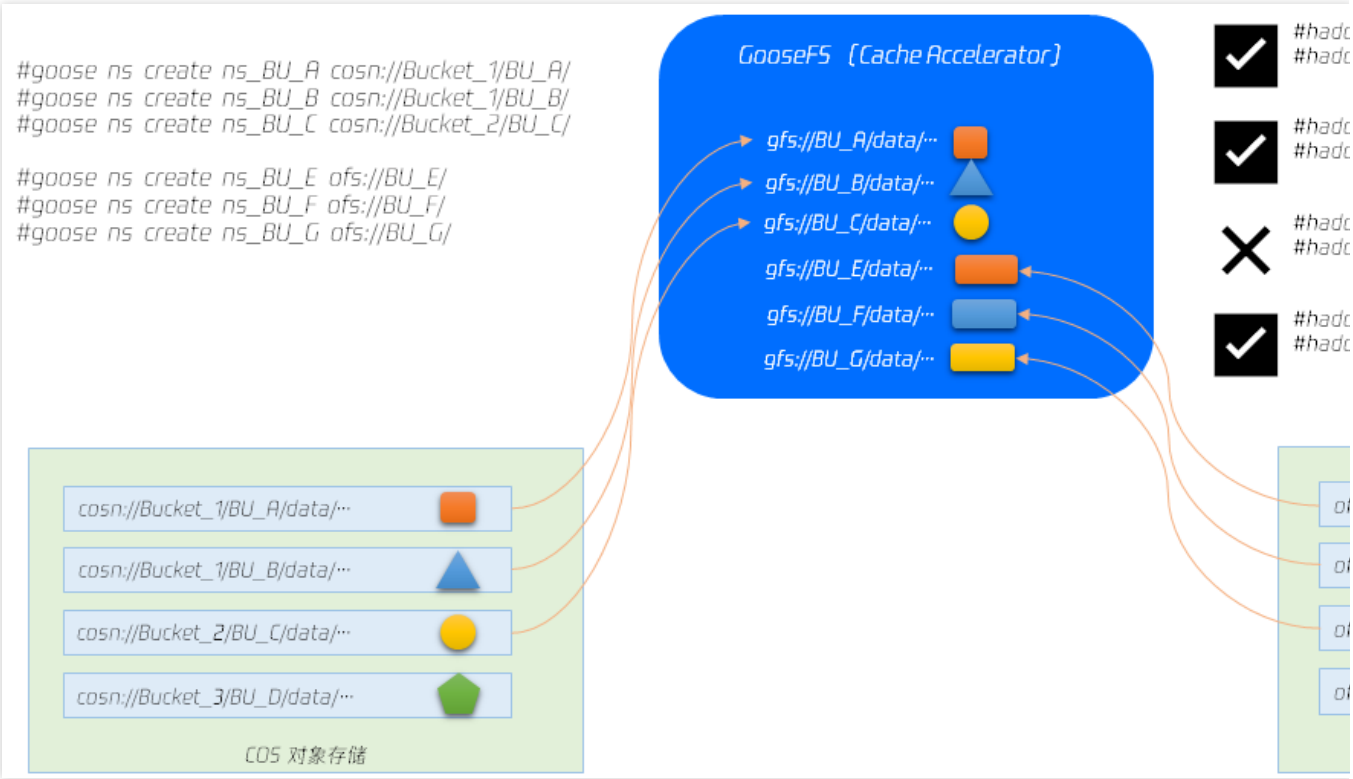
```
select sum(age) from u_user_ofs;
```

Unified Namespace Capability

Last updated：2024-03-25 16:04:01

Overview

The unified namespace capability of GooseFS integrates the access semantics of different underlying storage systems through a transparent naming mechanism, providing users with a unified interactive view for data management.
Leveraging this capability, GooseFS connects and communicates with different underlying storage systems, such as local file systems, Tencent Cloud Object Storage (COS), and Tencent Cloud HDFS (CHDFS), and provides unified access APIs and file protocols for upper-layer businesses. In this way, the business side only needs to call the access APIs provided by GooseFS to access data stored in different underlying storage systems.



The figure above shows the working principle of the unified namespace. You can use GooseFS's namespace creation instruction `create ns` to mount specified file directories of COS and CHDFS to GooseFS, and then use the `gfs://` unified schema to access data. Details are as follows:
COS contains 3 buckets in total: bucket-1, bucket-2, and bucket-3. bucket-1 contains the BU_A and BU_B directories, and both bucket-1 and bucket-2 are mounted to GooseFS.

CHDFS contains 4 directories: BU_E, BU_F, BU_G, and BU_H. All of them are mounted to GooseFS, except BU_H. Among GooseFS file operations, the BU_A and BU_E directories can be accessed using the unified schema `gfs://` , and the files are cached in the local file system of GooseFS.

The BU_A and BU_E directories stored in the underlying file systems (COS and HDFS) have been mounted to GooseFS. If files are cached on GooseFS, they can be accessed through the unified schema `gfs://` (for example, `hadoop fs ls gfs://BU_A`), and they can also be accessed through the namespace of each remote file system (for example, `hadoop fs ls cosn://bucket-1/BU_A`).

If the files are not cached on GooseFS, they cannot be accessed through the unified schema `gfs://` because the files are not cached in the local file system of GooseFS, but they can still be accessed through the namespace of the underlying storage systems.

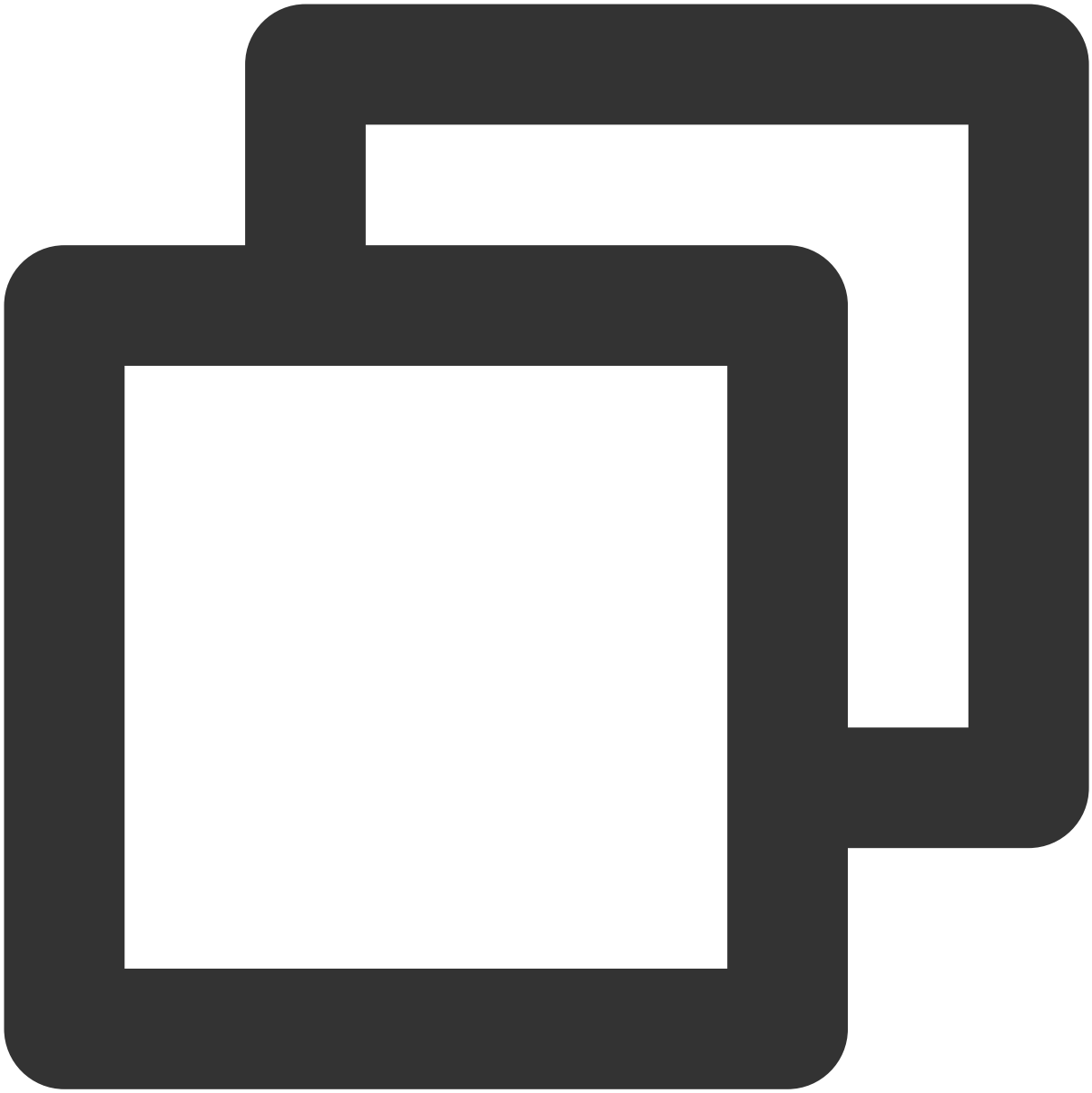
Using the Unified Namespace Capability

You can use the `create ns` instruction to create a namespace in GooseFS and map underlying storage systems to GooseFS. Currently supported underlying storage systems include COS, CHDFS, and local HDFS. The procedure for creating a namespace is similar to that for mounting a file volume disk in a Linux file system. With the namespace created, GooseFS can provide clients with a file system with uniform access semantics. The current operation instruction set for GooseFS namespaces is as follows:

Note:

We recommend that you avoid using permanent keys in the configuration. Configuring sub-account keys or temporary keys can help improve your business security. When authorizing a sub-account, grant only the permissions of the operations and resources that the sub-account needs, which helps avoid unexpected data leakage.

If you must use a permanent key, we recommend that you limit ITS permission scope. This can improve the security by limiting its executable operations, resource scope, and conditions (access IP, etc.).



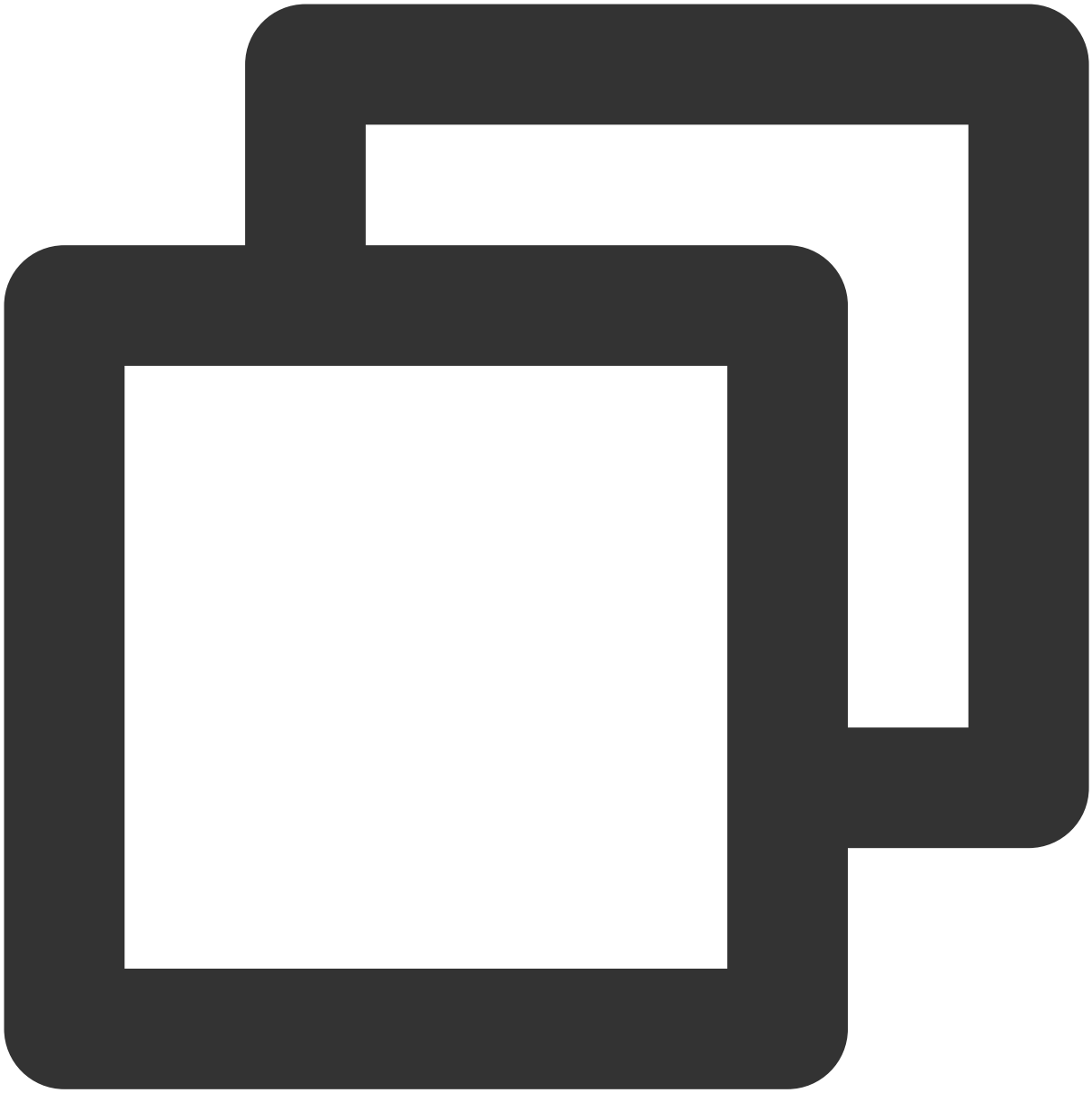
```
$ goosefs ns
Usage: goosefs ns [generic options]
        [create <namespace> <CosN/Chdfs path> <--wPolicy <1-6>> <--rPolicy <1-5>>
        [delete <namespace>]
        [help [<command>]]
        [ls [-r|--sort=option|--timestamp=option]]
        [setPolicy [--wPolicy <1-6>] [--rPolicy <1-5>] <namespace>]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

The instructions are described as follows:

Instruction	Description
create	Creates a namespace and maps a remote storage system (UFS) to the namespace. When creating the namespace, you can set cache read and write policies. You need to pass in an authorized key (`secretId` and `secretKey`).
delete	Deletes a specified namespace.
ls	Lists the detailed information of a specified namespace, including the UFS path, creation time, cache policy, and TTL information.
setPolicy	Sets the cache policy of a specified namespace.
setTtl	Sets TTL for a specified namespace.
stat	Provides the description of a specified namespace, including the mount point, UFS path, creation time, cache policy, TTL information, persistence status, user group, ACL, last access time, and modification time.
unsetPolicy	Resets the cache policy of a specified namespace.
unsetTtl	Resets the TTL of a specified namespace.

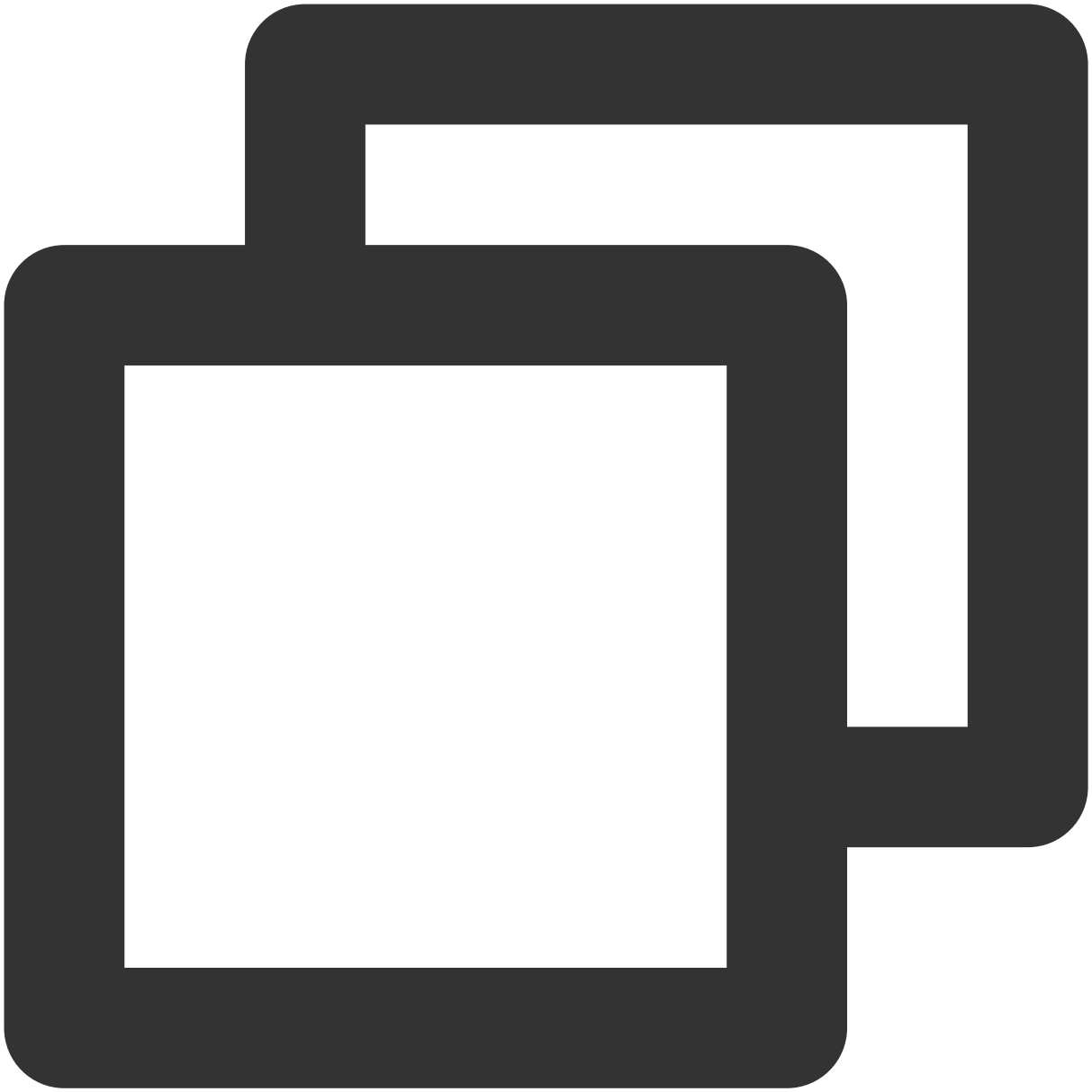
Creating or Deleting Namespaces

By creating a namespace in GooseFS, you can cache frequently accessed hot data from a remote storage system to a local high-performance storage node to provide high-performance data access for local computing businesses. The following shows how to map the COS bucket `example-bucket` , the `example-prefix` directory in the bucket, and the CHDFS to the `test_cos` , `test_cos_prefix` , and `test_chdfs` namespaces respectively.



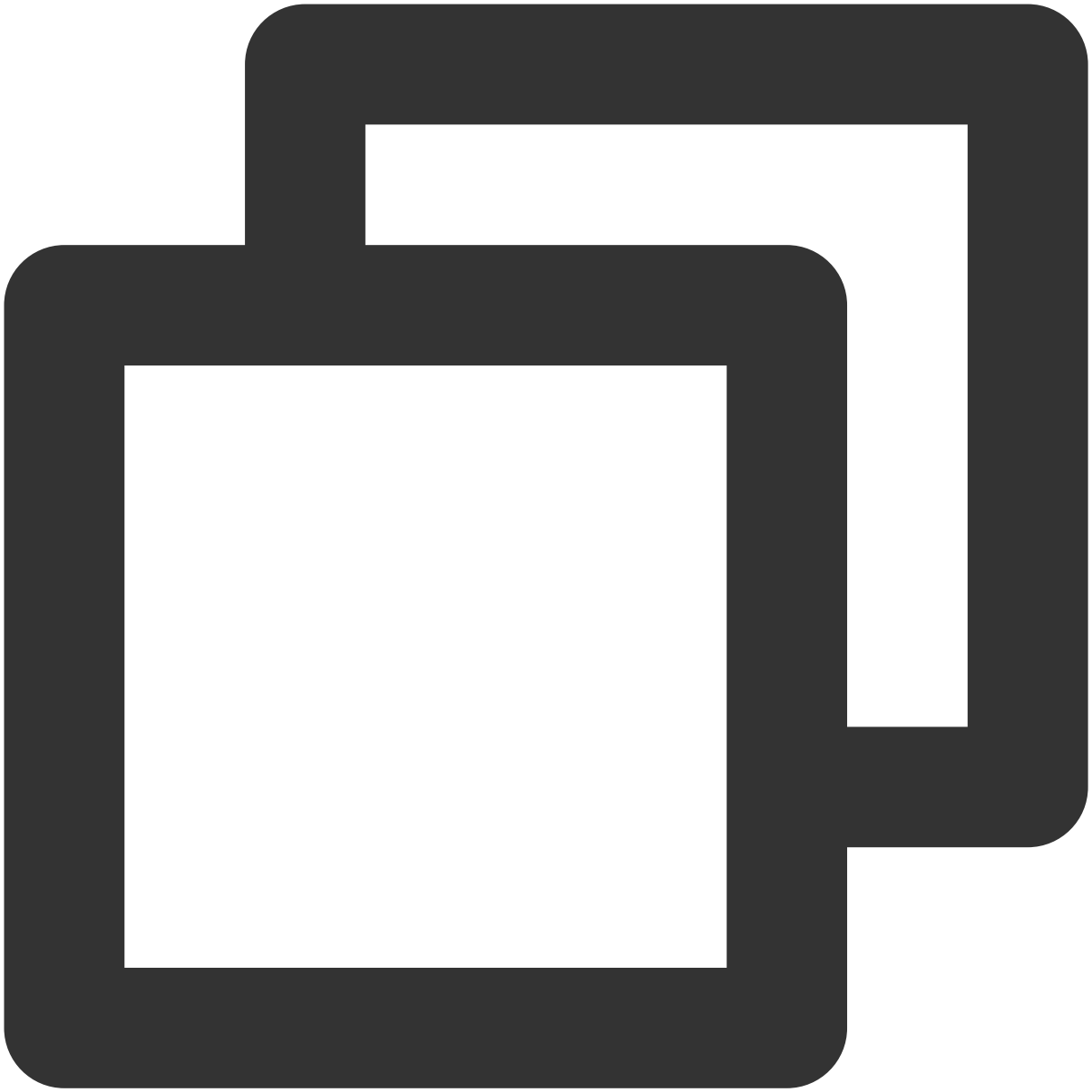
```
# Map the COS bucket `example-bucket` to the `test_cos` namespace
$ goosefs ns create test_cos cosn://example-bucket-1250000000/ --wPolicy 1 --rPolic

# Map the `example-prefix` directory in the COS bucket `example-bucket` to the `tes
$ goosefs ns create test_cos_prefix cosn://example-bucket-1250000000/example-prefix

# Map the CHDFS `f4ma0l3qabc-Xy3` to the `test_chdfs` namespace
$ goosefs ns create test_chdfs ofs://f4ma0l3qabc-Xy3/ --wPolicy 1 --rPolicy 1 --att
```

After successful creation, you can use the `goosefs fs ls` instruction to view the directory details:



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

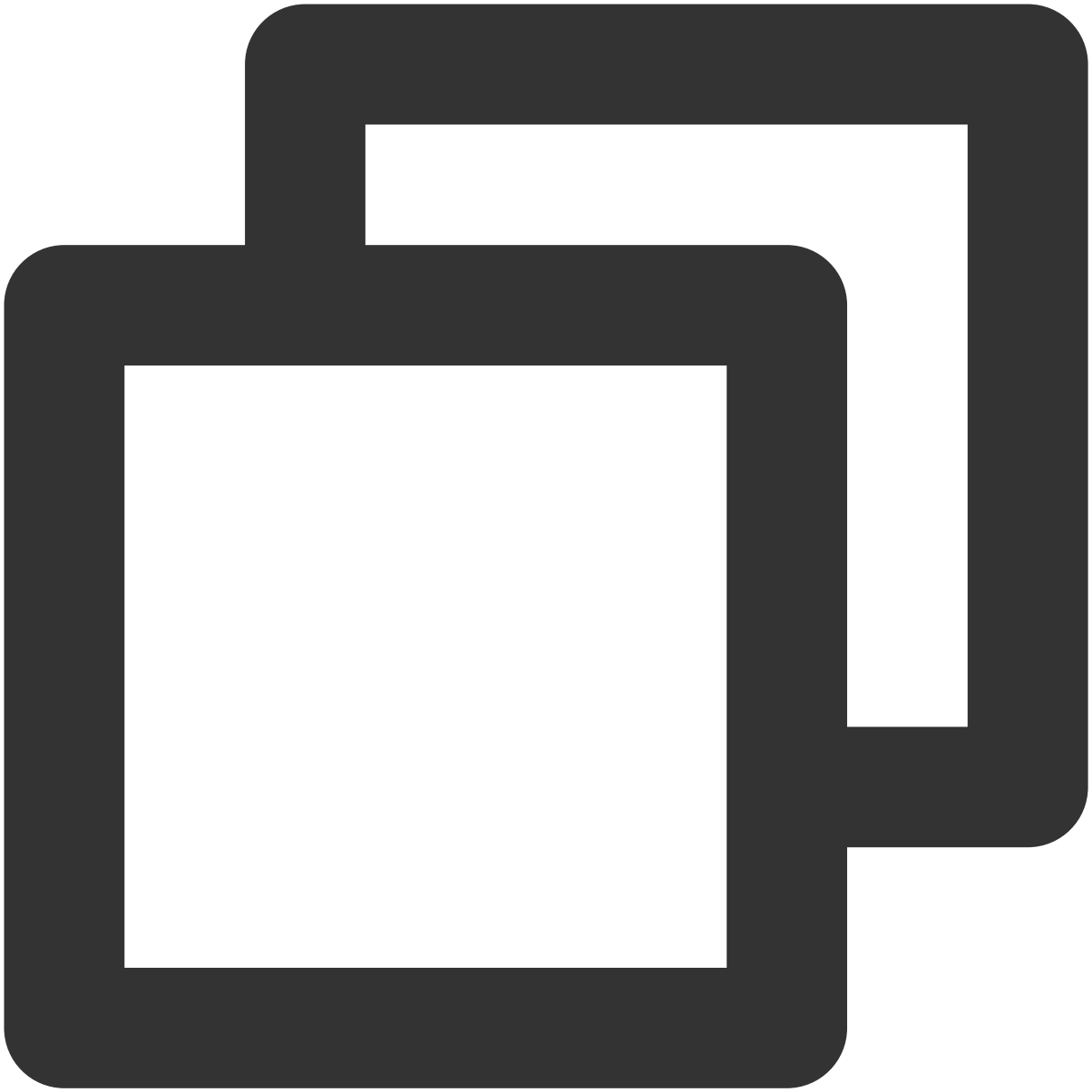
You can use the `delete` instruction to delete unwanted namespaces:



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

Setting the cache policy

You can use `setPolicy` and `unsetPolicy` to set the cache policy of a namespace. The instruction set is as follows:



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

The parameters are described as follows:

wPolicy: write cache policy. Up to 6 write cache policies are supported.

rPolicy: read cache policy. Up to 5 read cache policies are supported.

namespace: specified namespace

The read and write cache policies currently supported by GooseFS are as follows:

Write cache policies

Policy Name	Behavior	Corresponding	Data	Write

		Write_Type	Security	Efficiency
MUST_CACHE (1)	Data is stored only in GooseFS and is not written to the remote storage system.	MUST_CACHE	Unreliable	High
TRY_CACHE (2)	If the cache has space, data is written to GooseFS. Otherwise, data is written directly to underlying storage systems.	TRY_CACHE	Unreliable	Medium
CACHE_THROUGH (3)	Data is cached as much as possible and simultaneously written to remote storage systems.	CACHE_THROUGH	Reliable	Low
THROUGH (4)	Data is not stored in GooseFS, but written directly to the remote storage system.	THROUGH	Reliable	Medium
ASYNC_THROUGH (5)	Data is written to GooseFS and asynchronously purged to remote storage systems.	ASYNC_THROUGH	Weak reliability	High

Note:

`Write_Type` indicates the file cache policy specified when the user calls the SDK or API to write data to GooseFS. It takes effect only for a single file.

If you want to change the configured write cache policy, you need to carefully evaluate the importance of the cached data. If the data is important, we recommend that you ensure that the cached data has been persisted; otherwise, it may be lost. For example, after the write cache is changed from `MUST_CACHE` to `CACHE_THROUGH`, if the `persist` command is not called to persist the data, the data that is about to be eliminated cannot be written to the underlying layer and will be lost.

Read cache policies

Policy Name	Behavior	Metadata Sync	Corresponding Read_Type
NO_CACHE (1)	Data is not cached and is directly read from remote storage systems instead.	NO	NO_CACHE
CACHE (2)	Metadata access behavior: if a cache is hit, the metadata in the	Once	CACHE

	master shall prevail. Metadata is not proactively synchronized from the underlying layer. Data stream access behavior: the data stream `Read_Type` is `CACHE` .		
CACHE_PROMOTE (3)	Metadata access behavior: same as `CACHE` . Data stream access behavior: the data stream `Read_Type` is `CACHE_PROMOTE` .	Once	CACHE_PROMO
CACHE_CONSISTENT_PROMOTE (4)	Metadata access behavior: sync the metadata in the remote storage system (UFS) each time before a read operation. If the data does not exist in the UFS, report the `Not Exists` exception. Data stream access behavior: the data stream `Read_Type` is `CACHE_PROMOTE` . If a cache is hit, data is cached to the hottest cache medium.	Always	CACHE
CACHE_CONSISTENT (5)	Metadata access behavior: same as `CACHE_CONSISTENT_PROMOTE` . Data stream access behavior: the data stream `Read_Type` is `CACHE` . That is, when a cache is hit, data is not moved between different media layers.	Always	CACHE_PROMO

Note:

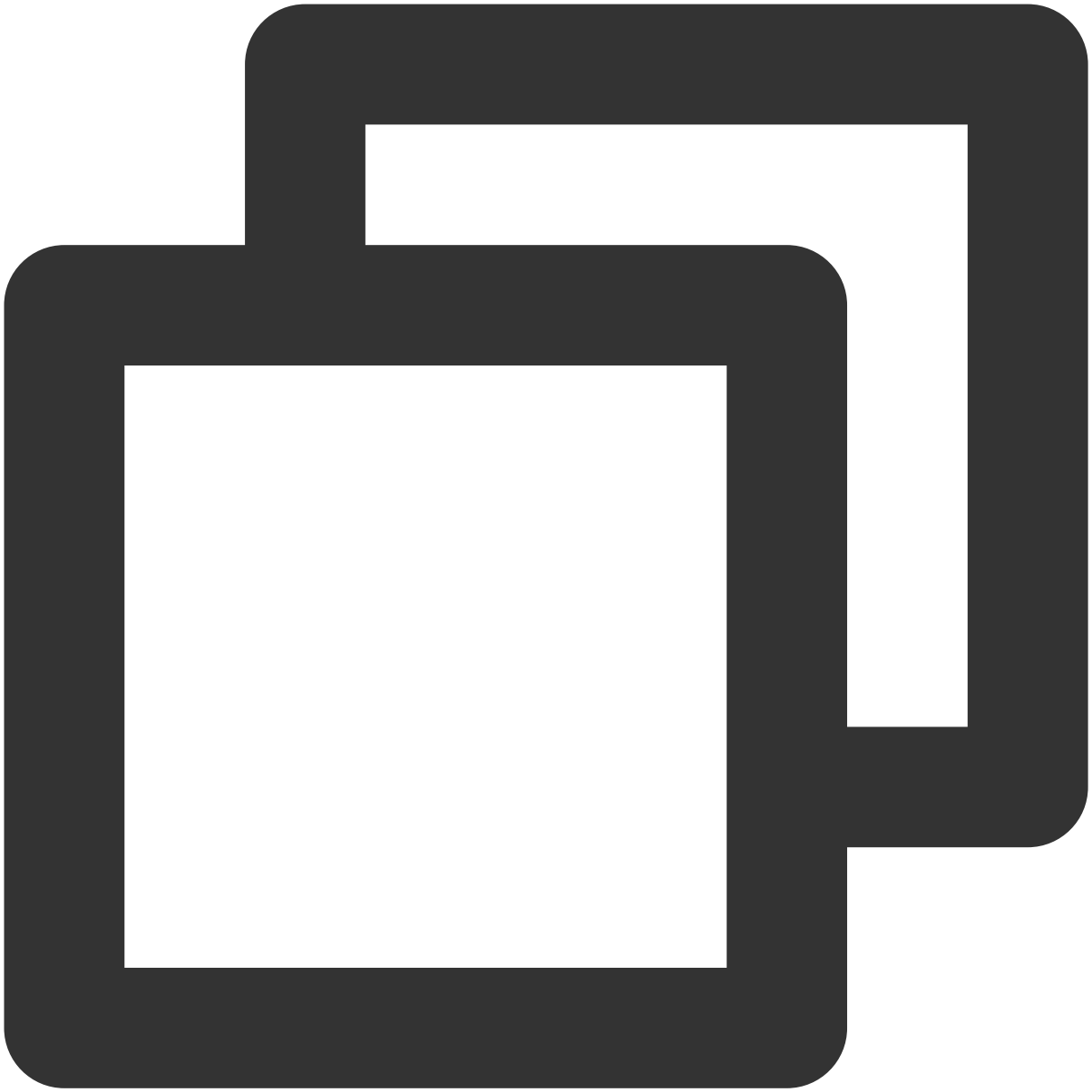
`Read_Type` indicates the file cache policy specified when the user calls the SDK or API to read data from GooseFS. It takes effect only for a single file.

Based on current big data business practices, we recommend the following combinations of read and write cache policies:

Write Cache Policy	Read Cache Policy	Policy Combination Performance
CACHE_THROUGH (3)	CACHE_CONSISTENT (5)	Strong data consistency between the cache and remote storage systems
CACHE_THROUGH	CACHE (2)	Write: strong consistency; read: eventual consistency

(3)		
ASYNC_THROUGH (5)	CACHE_CONSISTENT (5)	Write: eventual consistency; read: strong consistency
ASYNC_THROUGH (5)	CACHE (2)	Read/Write: eventual consistency
MUST_CACHE (1)	CACHE (2)	Data is read from the cache only.

The following example shows how to set the read and write cache policies of the `test_cos` namespace to `CACHE_THROUGH` and CACHE_CONSISTENT` respectively:

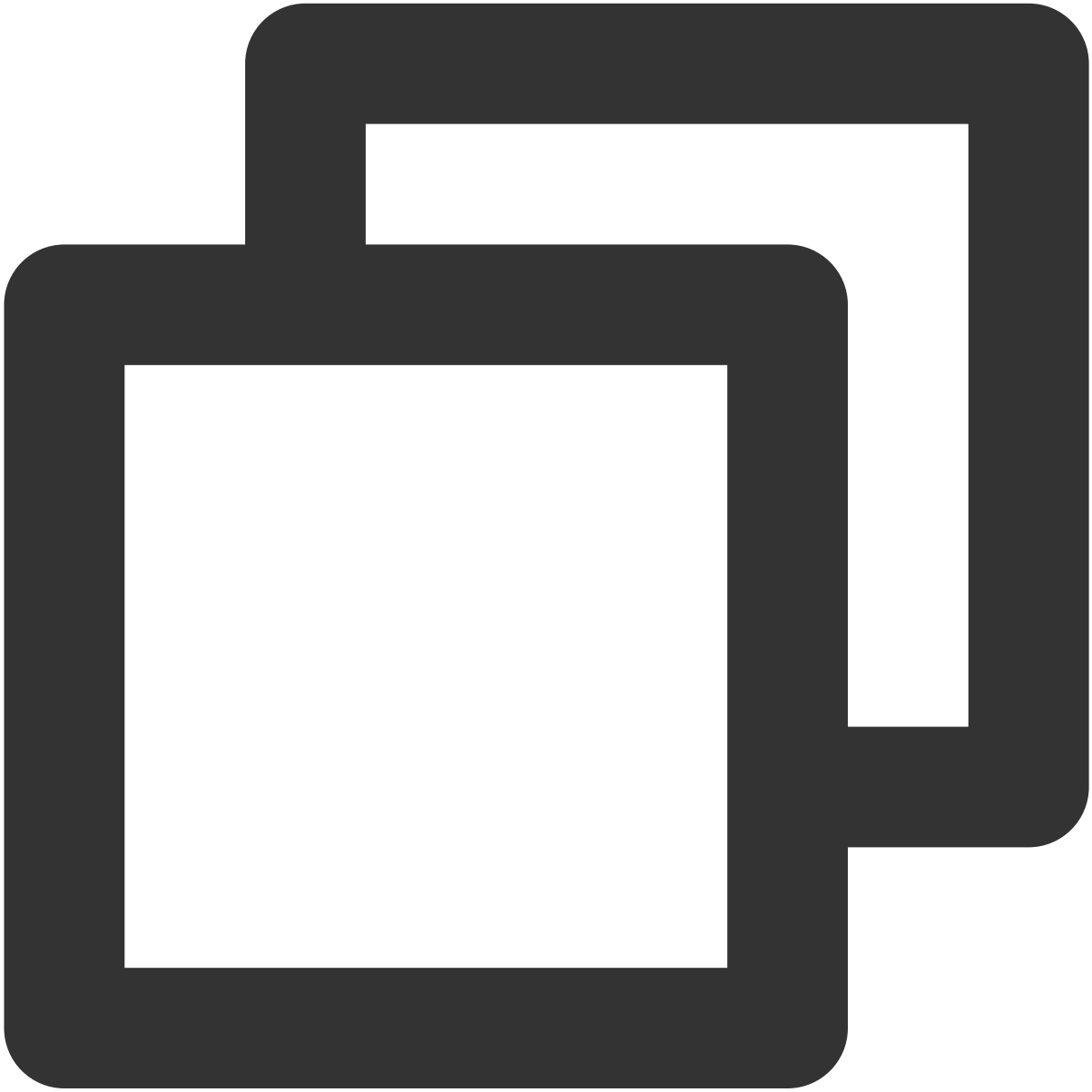


```
$ goosefs ns setPolicy --wPolicy 3 --rPolicy 5 test_cos
```

Note:

In addition to specifying cache policies when creating namespaces, you can also configure global cache policies by setting `Read_Type` or `Write_Type` for specific files when reading or writing files, or by using the `Properties` configuration file. If multiple policies exist at the same time, their priority order is as follows: custom priority > namespace read and write policies > global cache policy configured in the configuration file. For the read policy, the combination of the custom `Read_Type` and the namespace's `DirReadPolicy` takes effect. That is, the custom `Read_Type` is used as the data stream read policy, and the namespace policy is used for metadata.

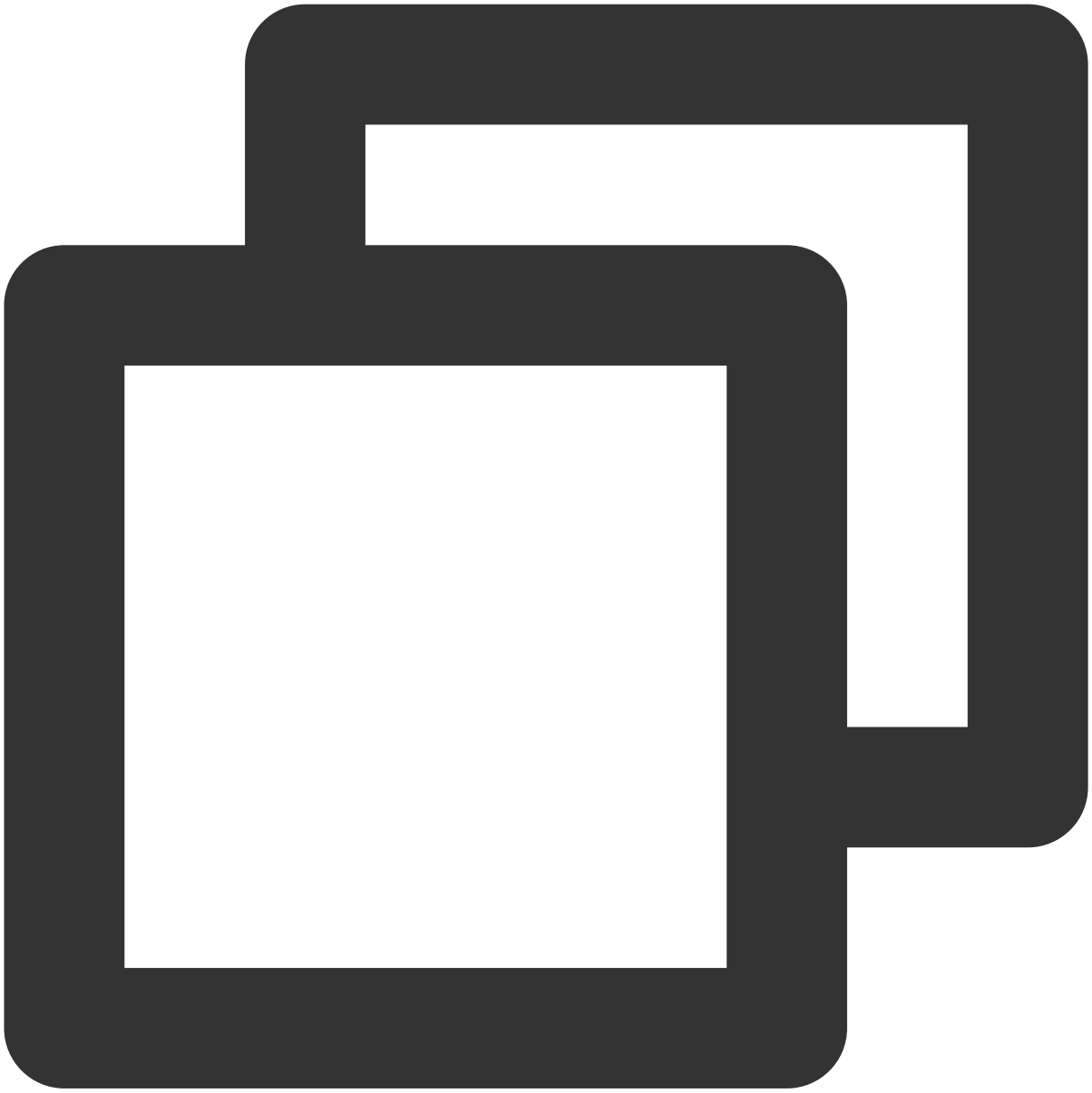
For example, GooseFS contains a COSN namespace whose read policy is `CACHE_CONSISTENT` and the namespace contains a `test.txt` file. When the client reads the `test.txt` file, `Read_Type` is specified as `CACHE_PROMOTE` . Then the entire read behavior is to sync metadata and perform `CACHE_PROMOTE` .
To reset the read and write cache policies, you can use the `unsetPolicy` instruction. The following shows how to reset the read and write cache policies for the `test_cos` namespace:



```
$ goosefs ns unsetPolicy test_cos
```

Setting TTL

Time to Live (TTL) is used to manage data cached on the local nodes of GooseFS. Setting TTL allows a specified operation, such as `delete` or `free`, to be performed on the cached data after a specified period of time. The instruction for setting TTL is as follows:



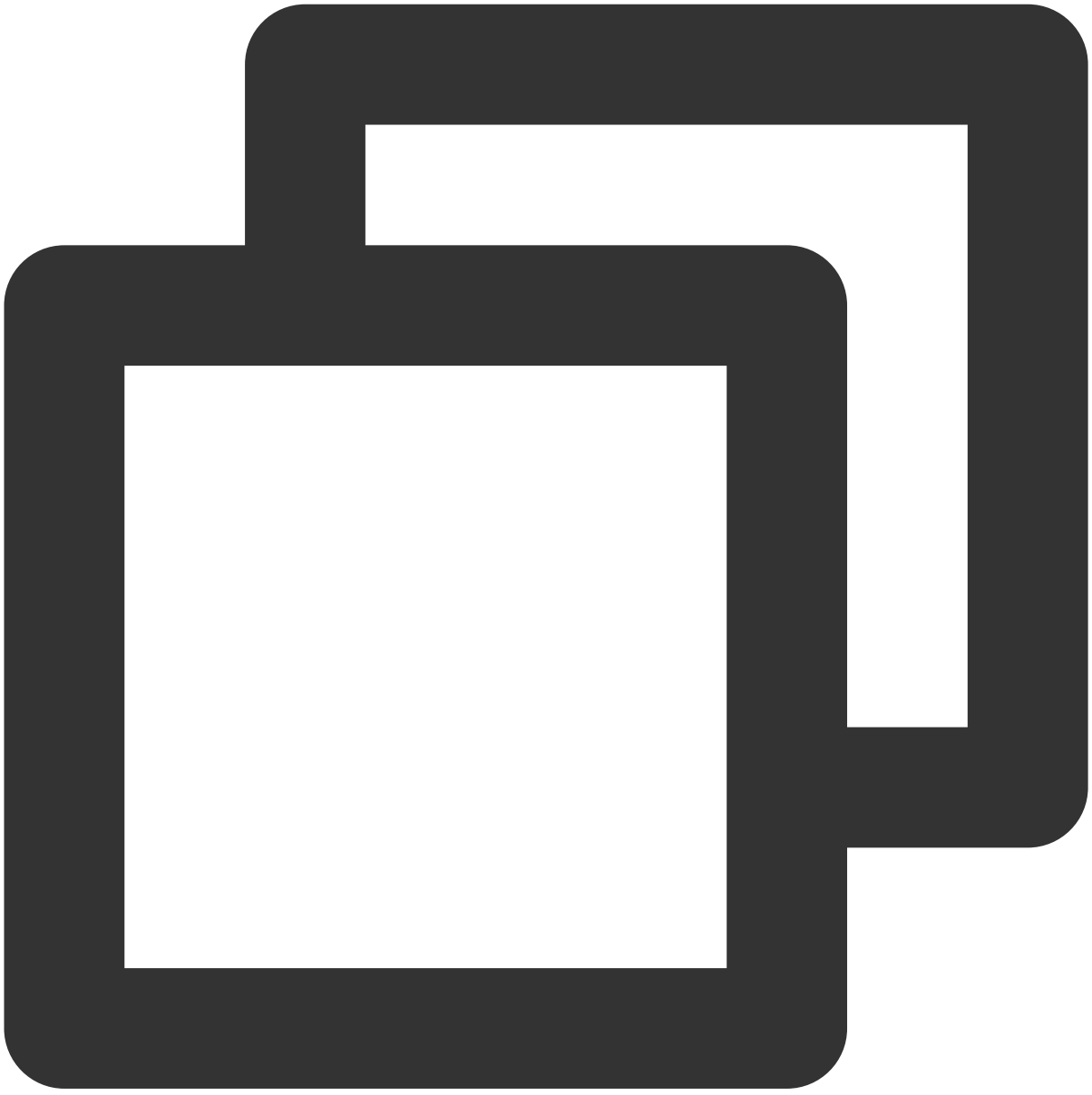
```
$ goosefs ns setTtl [--action delete|free] <namespace> <time to live>
```

The parameters are described as follows:
action: operation performed after the cache duration expires. Currently, `delete` and `free` are supported. The `delete` operation deletes data from the cache and UFS, while the `free` operation deletes data only from the cache.

namespace: specified namespace

time to live: data cache duration, in milliseconds

The following example shows how to set the policy of the `test_cos` namespace to delete data only from the cache after 60 seconds:



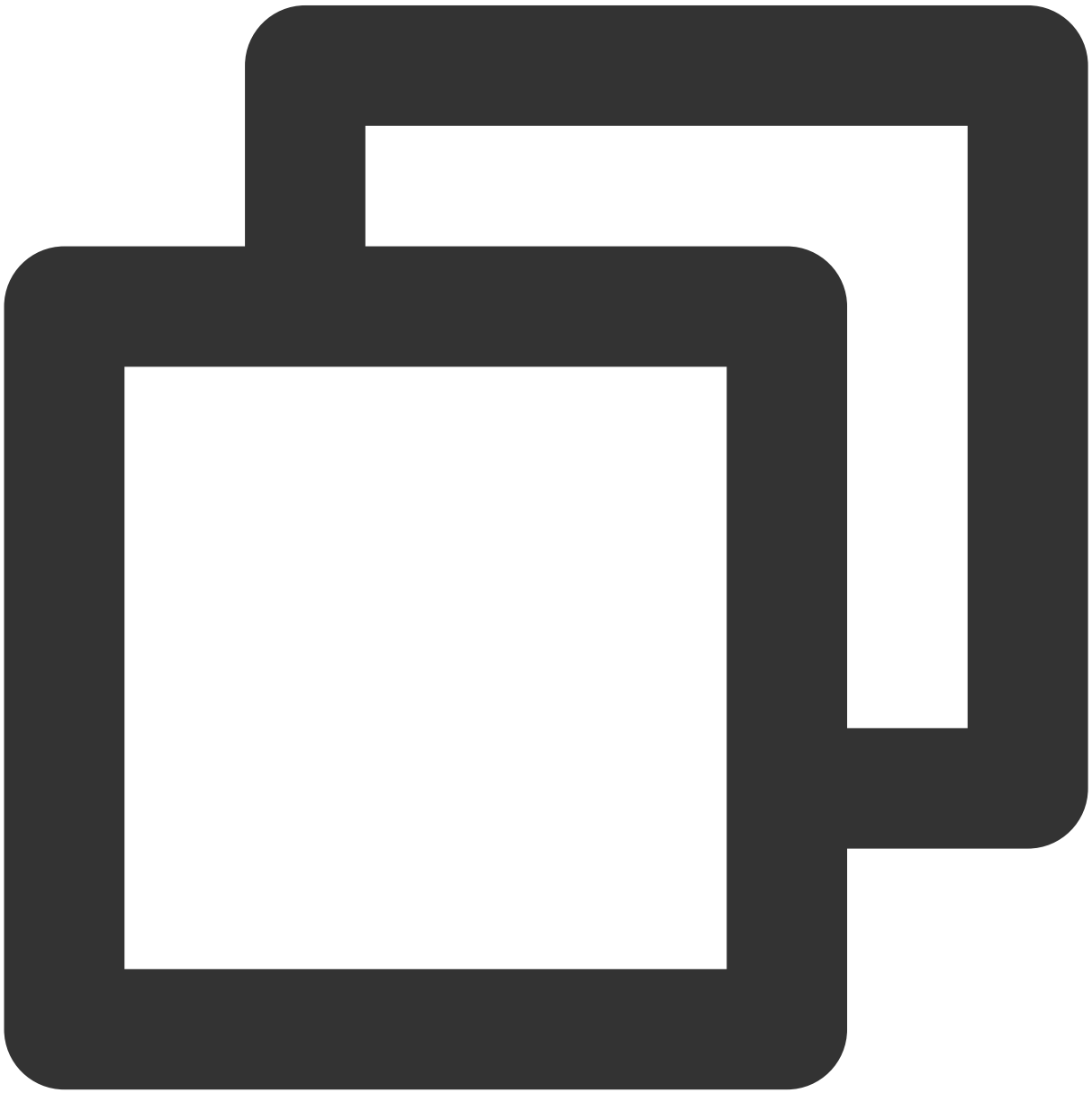
```
$ goosefs ns setTtl --action free test_cos 60000
```

Metadata management

This section describes how GooseFS manages metadata, including metadata synchronization and updates. GooseFS provides users with unified namespace capability. Users can access files on different underlying storage systems using a unified `gfs://` path. You only need to specify the paths of the underlying storage systems. We recommend that you use GooseFS as a unified data access layer to uniformly read and write data from GooseFS to ensure metadata consistency.

Metadata synchronization overview

You can configure the metadata synchronization interval in the `conf/goosefs-site.properties` configuration file:



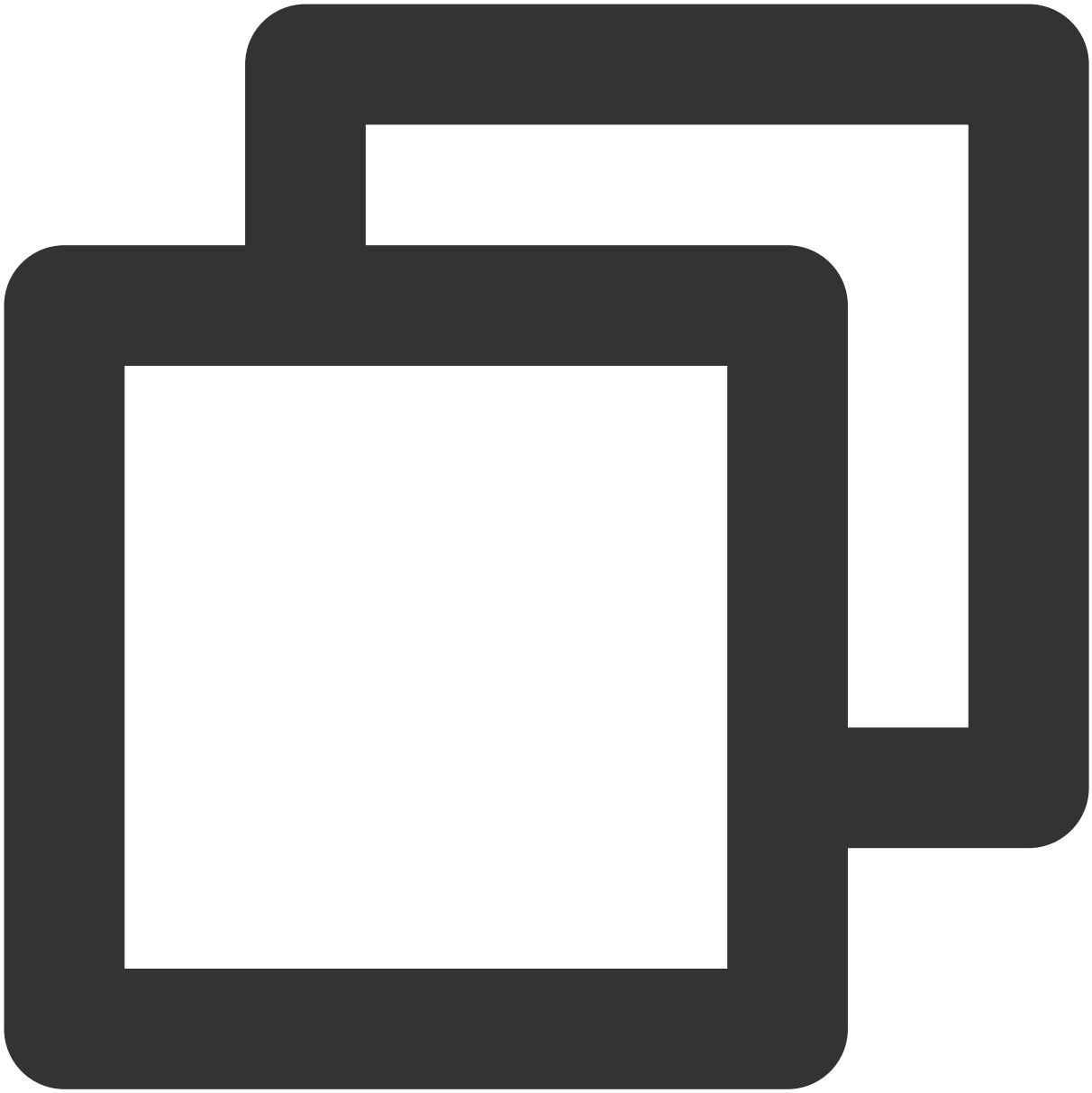
```
goosefs.user.file.metadata.sync.interval=<INTERVAL>
```

The metadata synchronization interval parameter supports 3 types of input values:
-1: metadata is not updated after it is first loaded into GooseFS.
0: GooseFS updates metadata after each read/write operation.
Positive integer: GooseFS periodically updates metadata at a specified interval.
You can choose an appropriate synchronization interval based on your number of nodes, the I/O distance between your GooseFS cluster and the underlying storage system, and the type of the underlying storage system. Usually:
The greater the number of nodes in a GooseFS cluster, the greater the metadata synchronization delay.
The greater the distance between the GooseFS cluster and the underlying storage system, the greater the metadata synchronization delay.
The impact of the underlying storage system on the metadata synchronization delay depends on the QPS load. The higher the QPS load, the less the synchronization delay.

Metadata synchronization management

Configuration methods

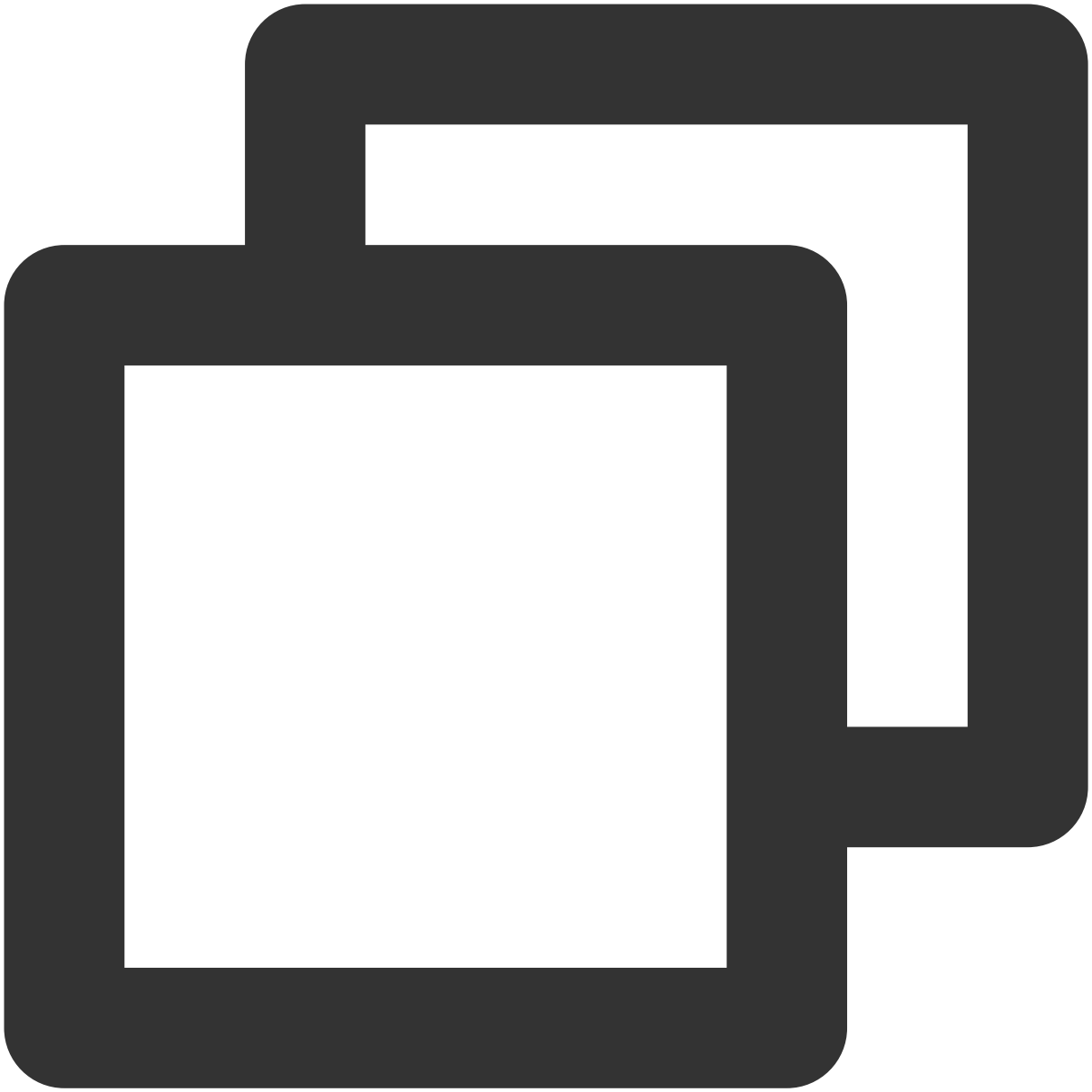
1. Configuration via CLI
You can set the metadata synchronization interval in command line interface (CLI) mode:



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

2. Configuration via the configuration file

For a large-scale GooseFS cluster, you can use the `goosefs-site.properties` configuration file to batch configure the metadata synchronization interval for the master nodes in the cluster, and other nodes will adopt this interval by default.



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

Note:

Many businesses choose to distinguish the purpose of data by directory, and the data access frequencies of different directories are not all the same. You can set different metadata synchronization intervals for different directories. For some directories that change frequently, the metadata synchronization interval can be set to a shorter time (such as 5 minutes). For directories that change little or do not change, the synchronization interval can be set to `-1`, so that GooseFS will not automatically synchronize the metadata of the directories.

Recommended configuration

You can set different metadata synchronization intervals based on business access modes:

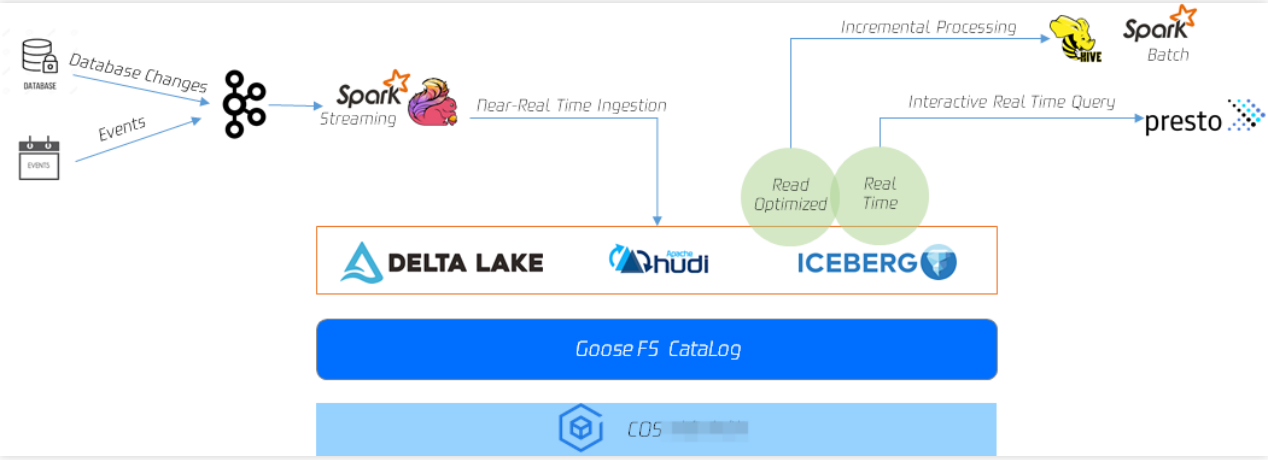
<table>
<tr><th colspan="2">Access Mode</th><th>Metadata Synchronization Interval</th><th>Description</th></tr>
<tr><td colspan="2">All file requests go through GooseFS</td><td>-1</td><td>-</td></tr>
<tr><td rowspan="2">Most file requests go through GooseFS</td><td>HDFS is used as UFS</td><td>Hot update or update by path is recommended</td><td>If the HDFS updates frequently, you are advised to set the update interval to `-1` to prohibit updates.</td></tr>
<tr><td>COS is used as UFS</td><td>Configuring update intervals by path is recommended</td><td rowspan="3">Configuring different update intervals for different directories can alleviate the pressure of metadata synchronization.</td></tr>
<tr><td rowspan="2">File upload requests generally do not go through GooseFS</td><td>HDFS is used as UFS</td><td>Configuring update intervals by path is recommended</td></tr>
<tr><td>COS is used as UFS</td><td>Configuring update intervals by path is recommended</td></tr>
</table>

Table Management

Last updated：2024-03-25 16:04:01

Overview

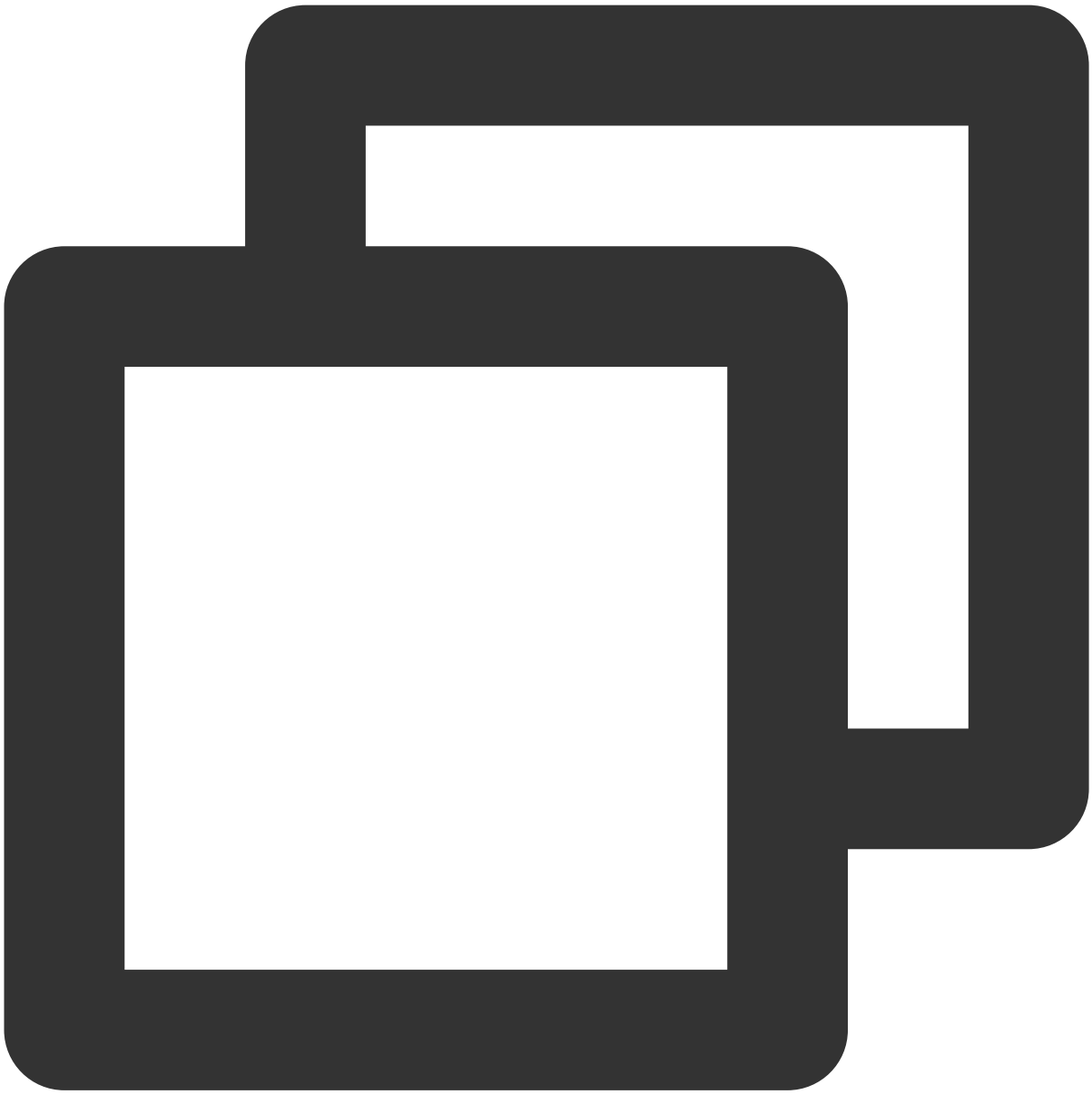
GooseFS's table management feature can be used to manage structured data. It manages database tables of upper-layer computing applications such as Spark SQL, Hive, and Presto. You can connect it to the underlying Hive Metastore. Table management allows SQL engines to read specified data and speeds up data access when the data volume is high.



GooseFS's table management introduces the following features:
Describe metadata. GooseFS Catalog supports caching metadata from a remote Hive Metastore. If SQL engines (such as Spark SQL, Hive, and SQL Presto) are querying data, GooseFS Catalog's metadata caching service can determine the target data's size, location, and structure the way Hive Metastore does.
Cache table-level data in advance. GooseFS Catalog detects the mapping relationship between tables and their storage paths and thus can cache table or partition level data in advance to speed up data access.
Unify metadata queries from multiple storage services. You can use GooseFS Catalog to run upper-layer computing applications and speed up access to various underlying storage services. Moreover, GooseFS Catalog supports cross-storage metadata queries. You only need to enable Catalog on one GooseFS client and can query data stored in different storage systems such as HDFS, COS, and CHDFS.

How It Works

GooseFS table management is implemented with `goosefs table` commands shown below, which can be used to attach/detach databases, query database/table information, load/free data, and more.



```
$ 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>]
```

The commands are described as follows:

attachdb: attaches a remote database to GooseFS. Currently, only Hive Metastore is supported.

detachdb: detaches a database from GooseFS.

free: frees data cache of a specified DB.Table (partition-level is supported).

load: loads data of a specified DB.Table (partition-level is supported). You can use `replication` to specify the number of replicas to cache.

ls: lists metadata of a DB or DB.Table.

stat: queries the statistics on a specified DB.Table, including the number of files, total size, and the percentage cached.

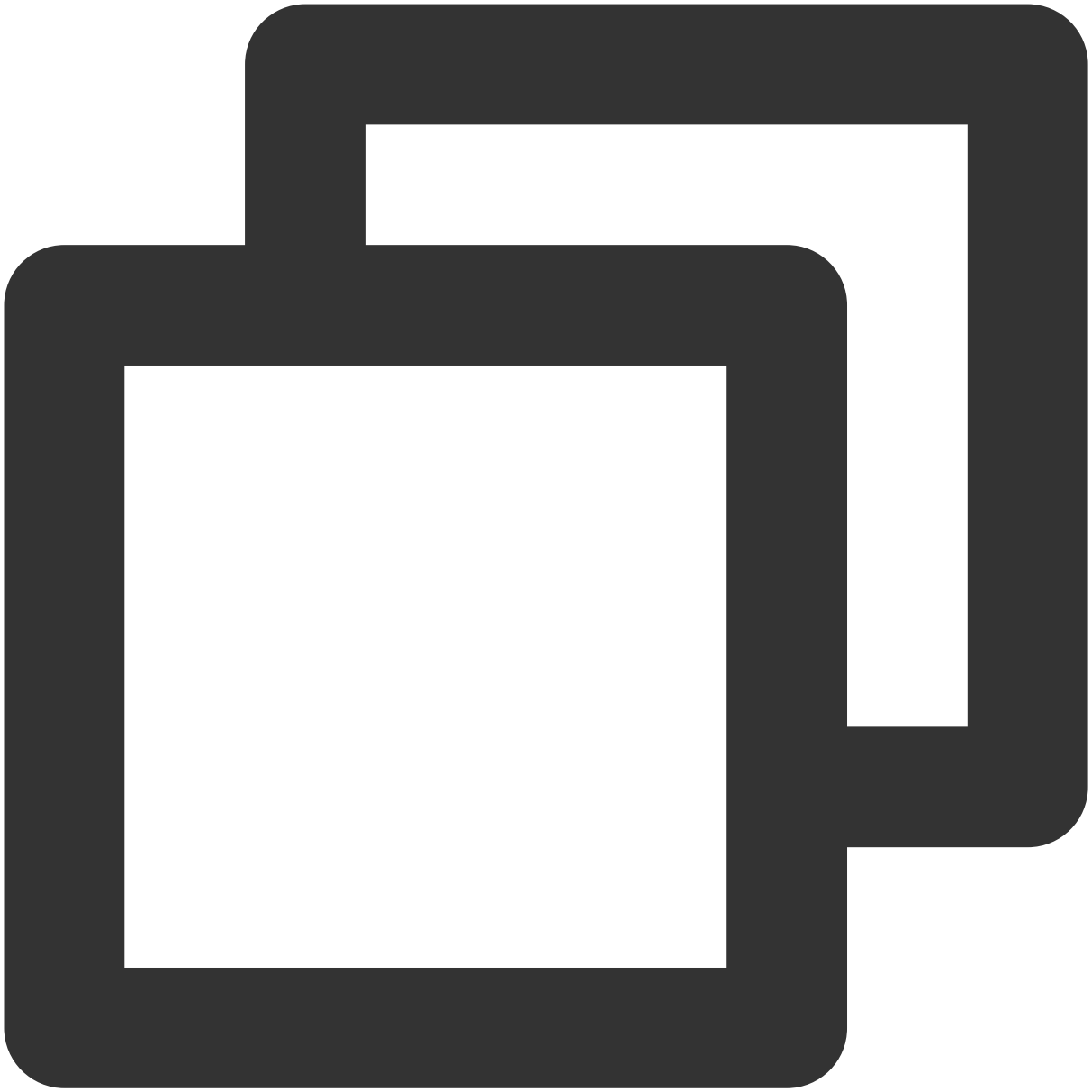
sync: syncs data of a specified database.

transform: transforms a table in a specified database into a new table.

transformStatus: queries table transforming status.

1. Attaching a database

You need to attach the database to GooseFS before loading the table data. Run the following command to attach the database `goosefs_db_demo` from `metastore_host:port` to GooseFS and name this database `test_db` in GooseFS:



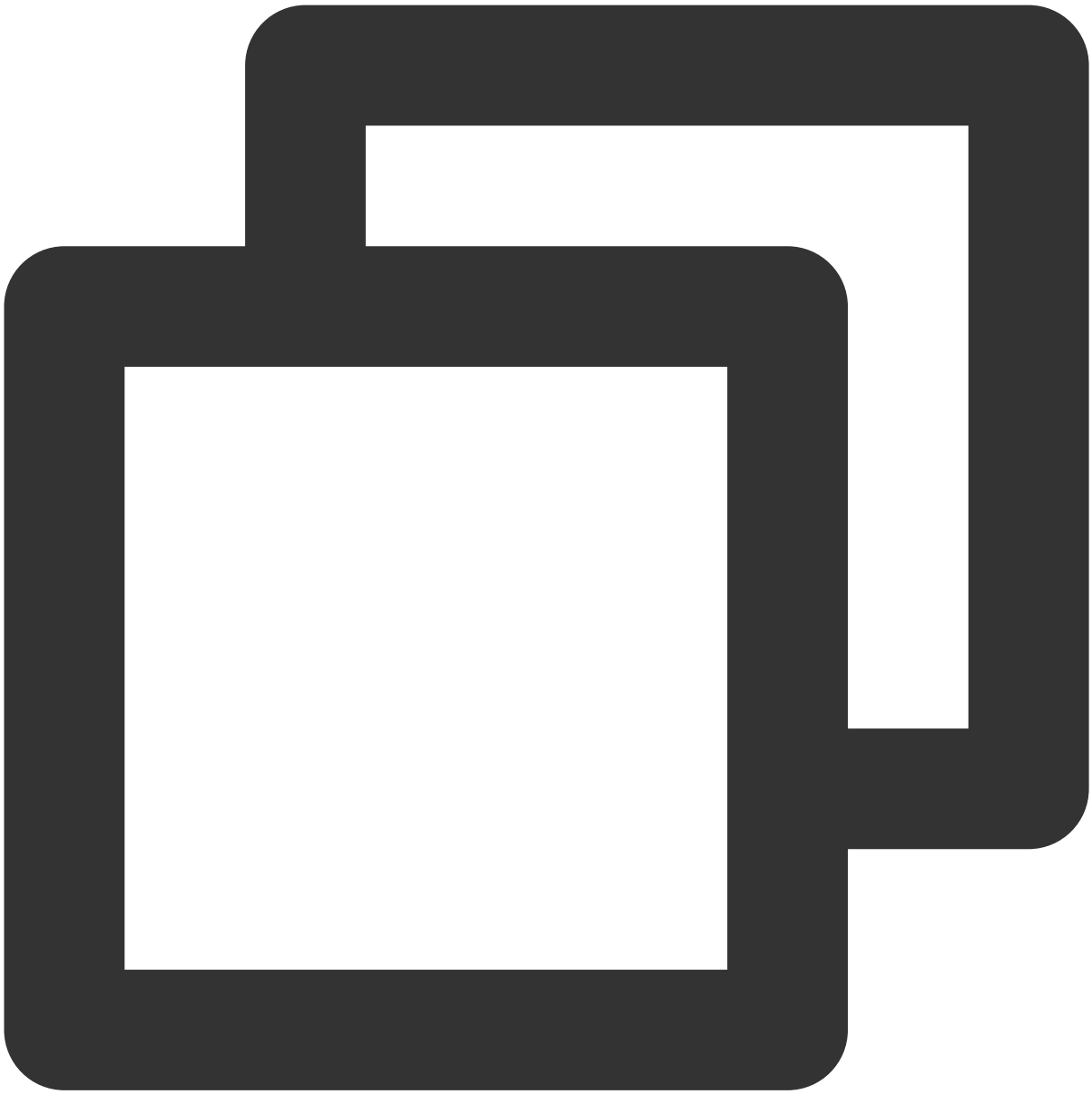
```
$ goosefs table attachdb --db test_db hive thrift://metastore_host:port goosefs_db_

response of attachdb
```

Note:

You can replace `metastore_host:port` with any valid and connectable Hive Metastore address.

2. Viewing table information

After the database is attached, run the `ls` command to view the attached database and table information. The following command shows how to query information about the `web_page` table in `test_db` :

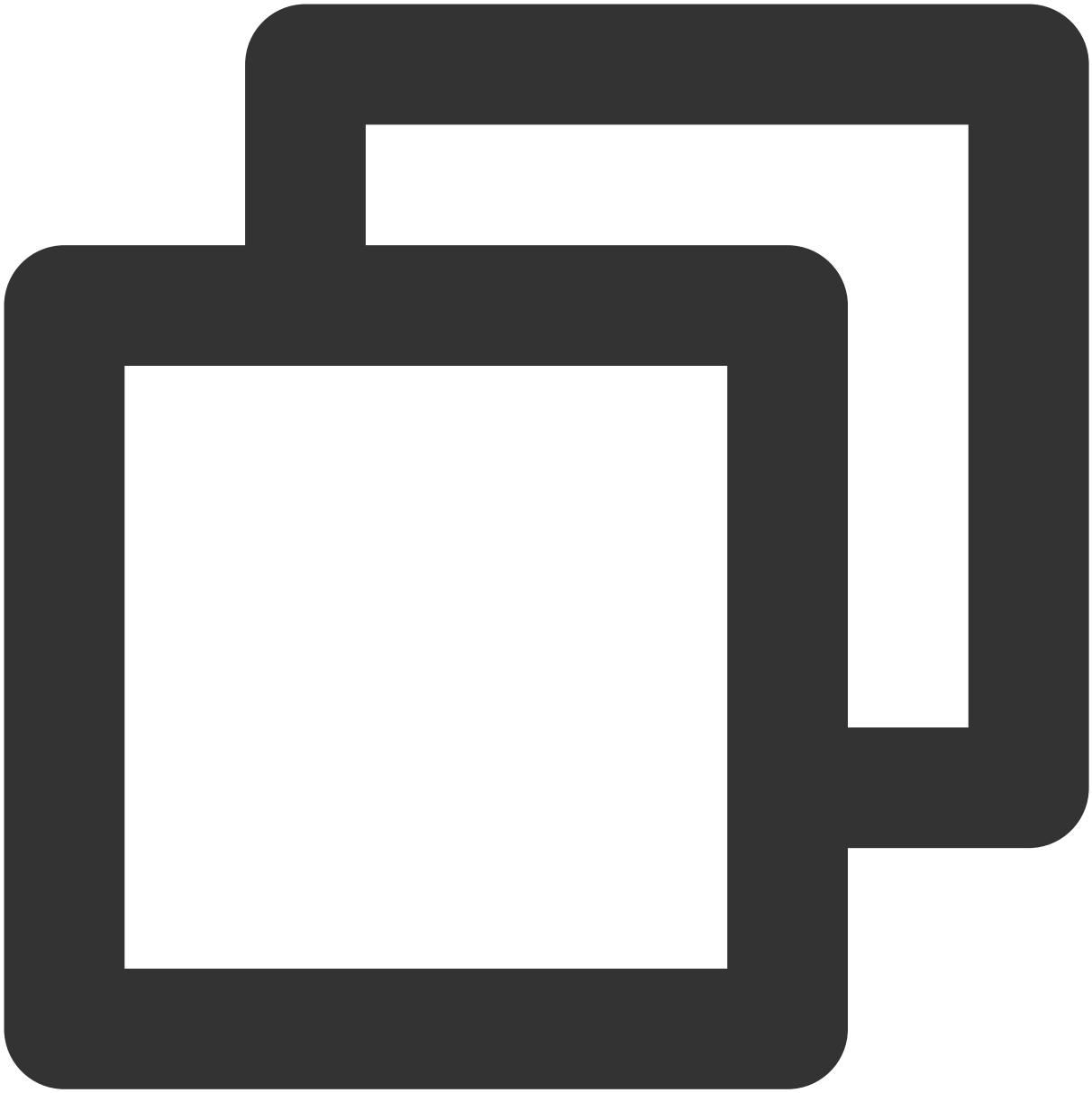


```
$ goosefs table ls test_db web_page

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. Loading table data

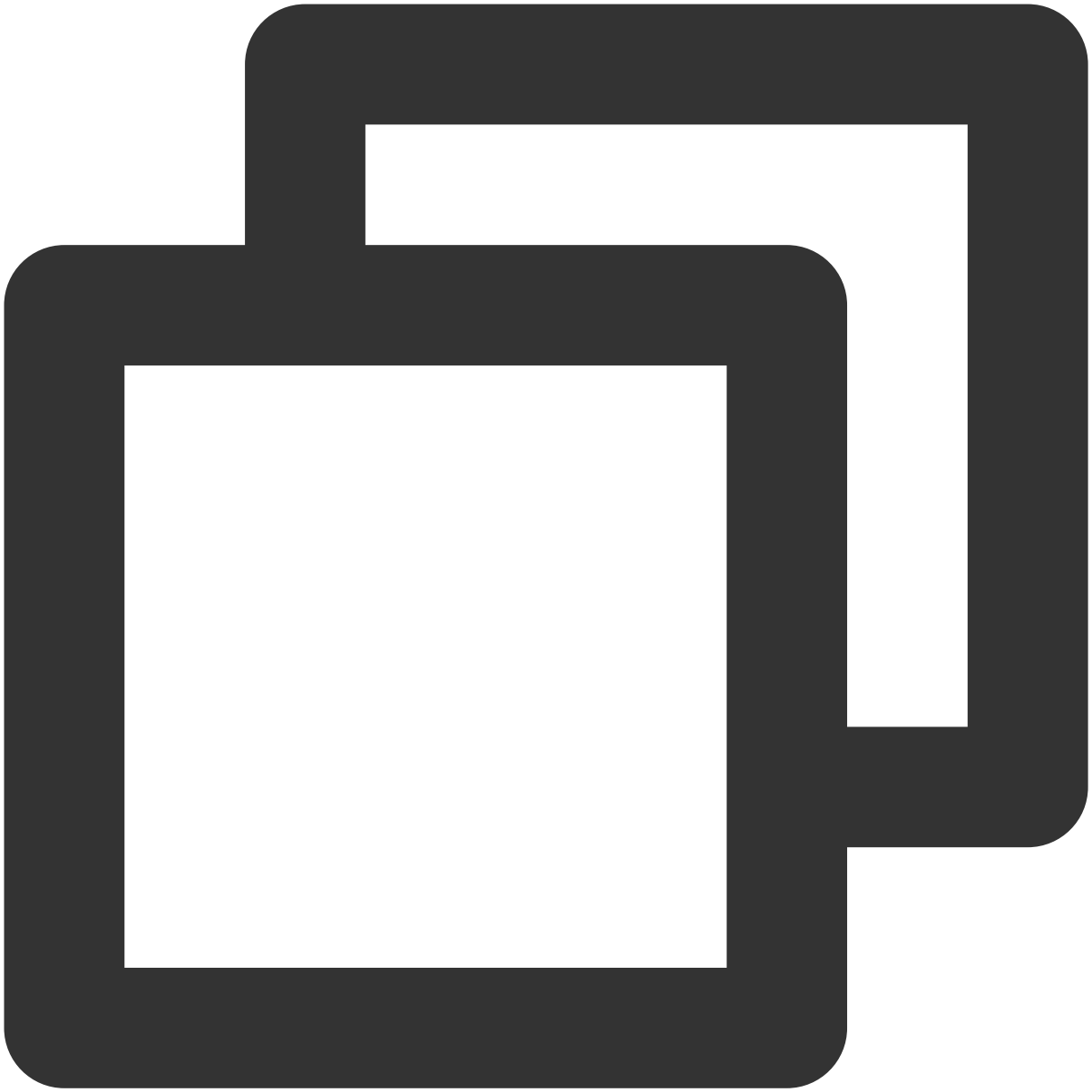
Once the data loading command is delivered, an async job will be initiated in the backend, and GooseFS will return the job ID after the job is started. You can run the `job stat <ID>` command to view the running status of the job, or run the `table stat` command to view the percentage cached. The data loading command is as follows:



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

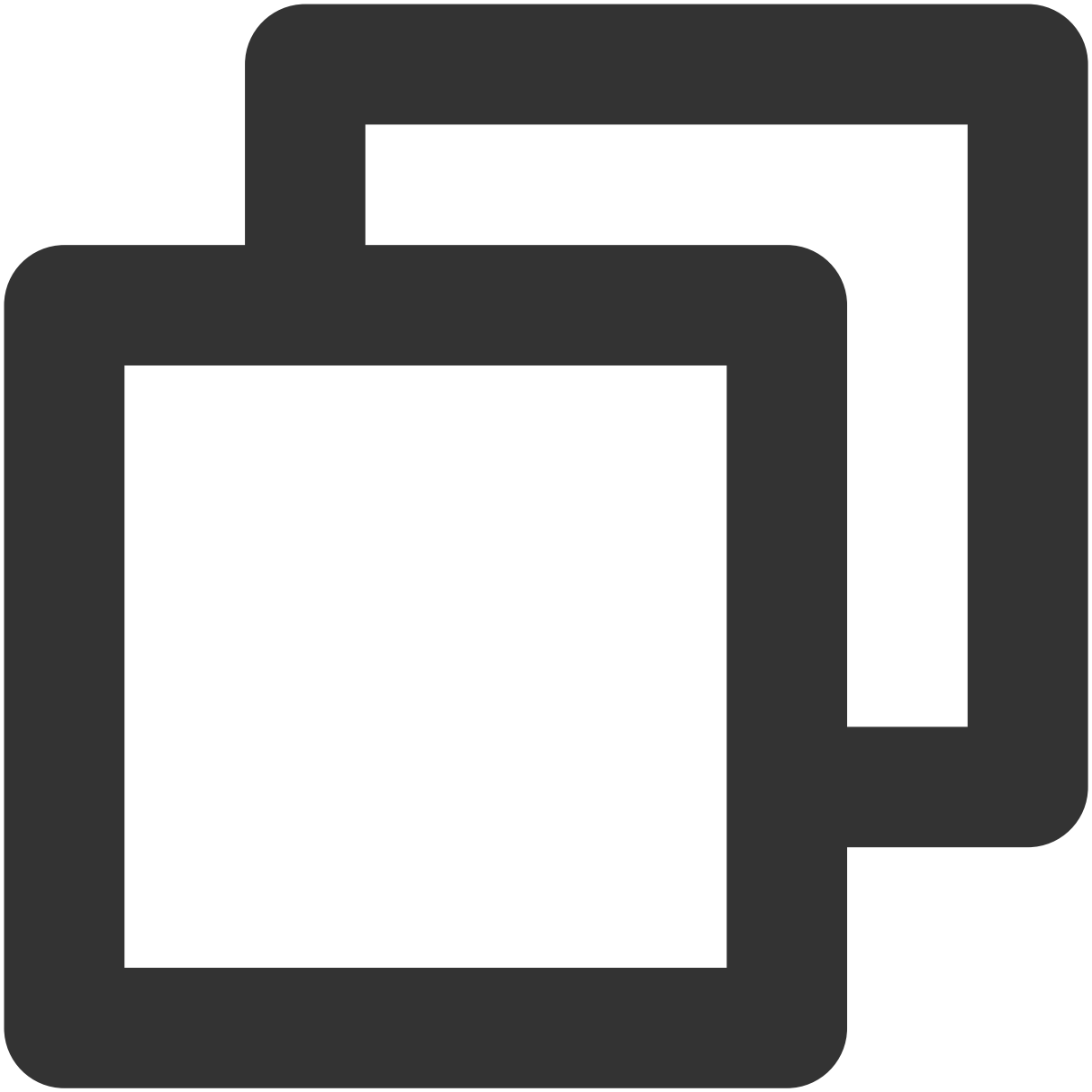
4. Viewing statistics on table loading

Run the `job stat` command to view the execution progress of the table loading job. If the status is `COMPLETED`, the table has been loaded successfully. If the status is `FAILED`, you can view the logs in `master.log` to troubleshoot.



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

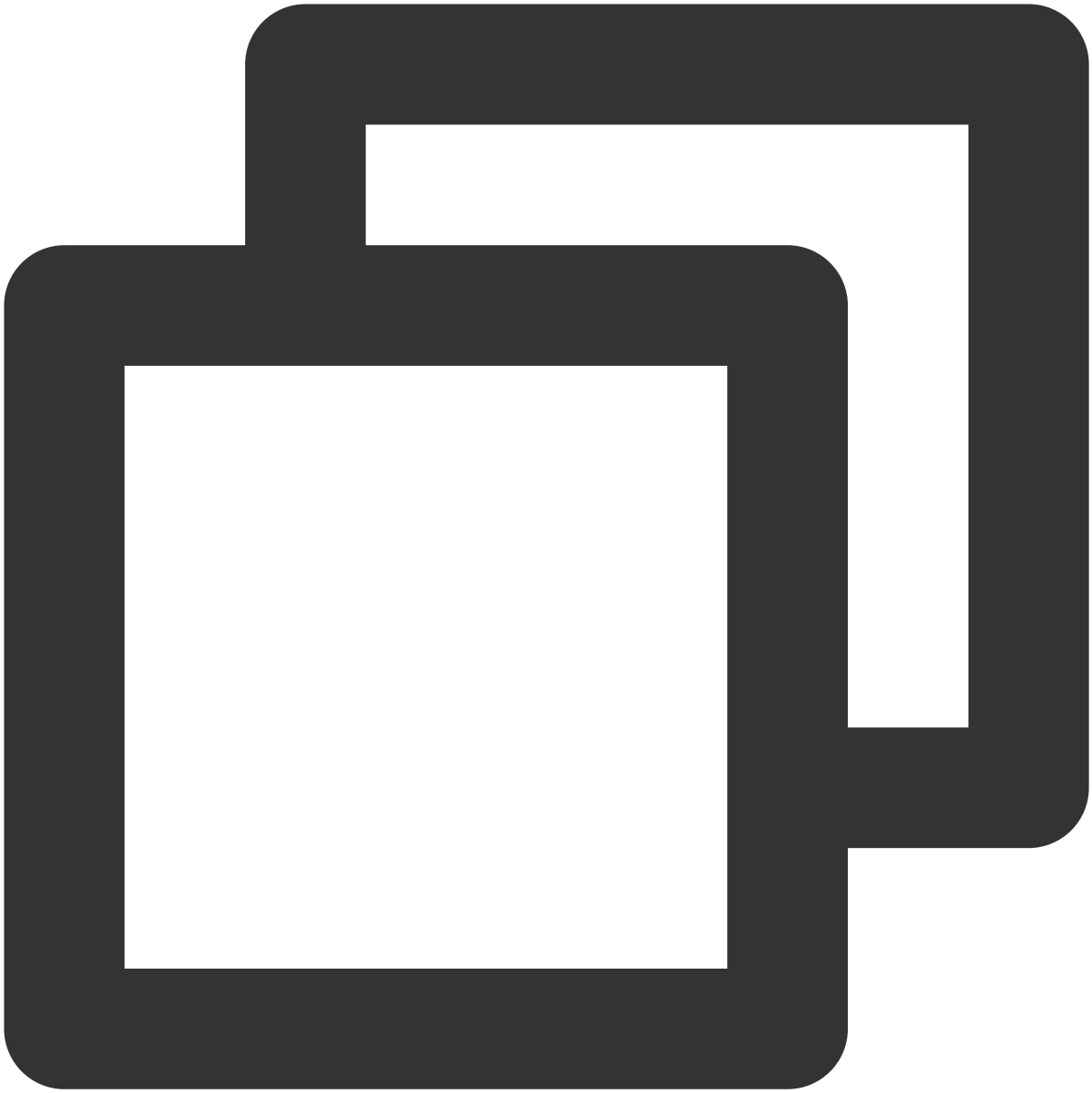
When the table is loaded, run the `stat` command to view the statistics on the table.



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

5. Freeing a table

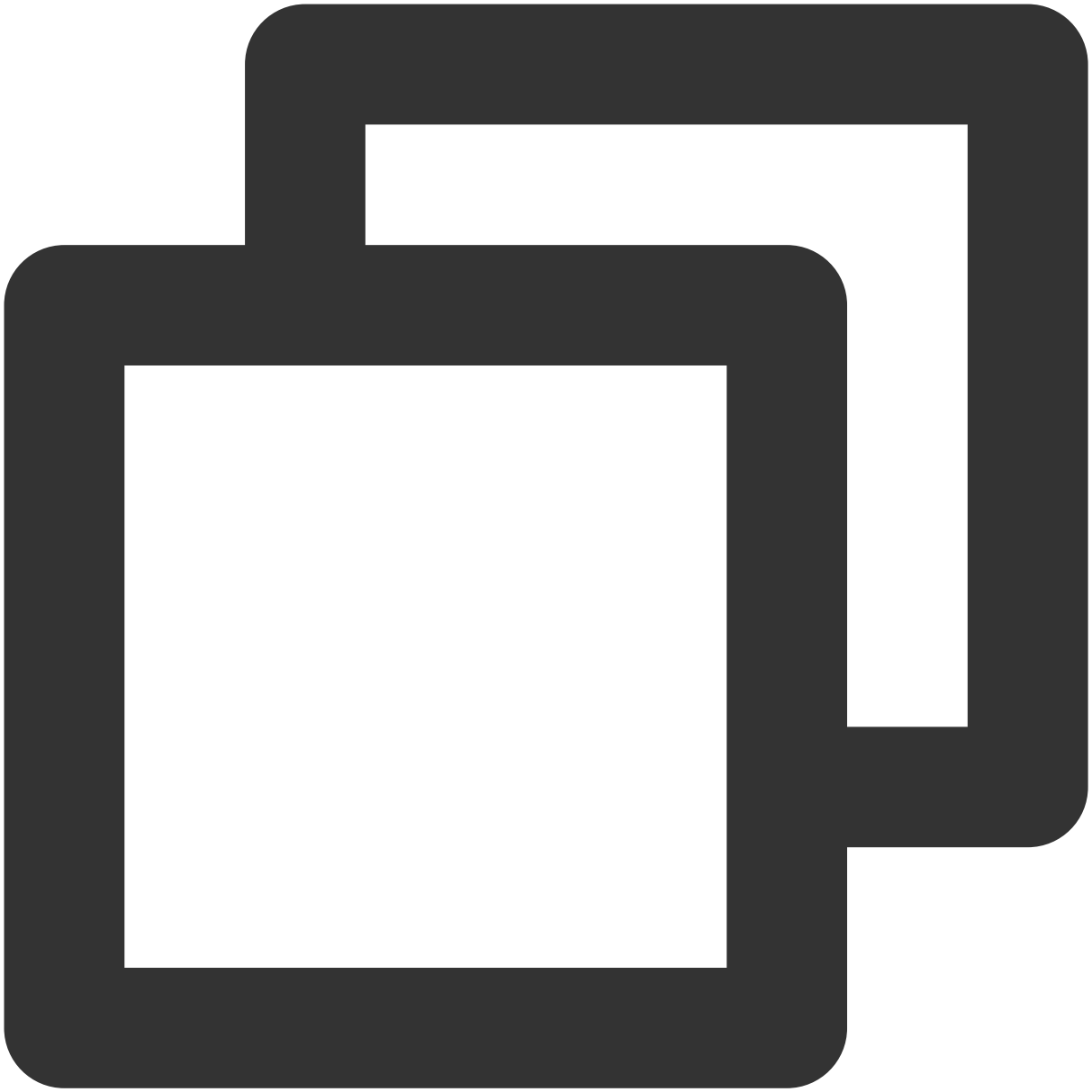
Run the following table to free the cache of a specified table from GooseFS:



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

6. Detaching the database

Run the following command to detach a specified database from GooseFS:



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

GooseFS-FUSE Capability

Last updated：2024-03-25 16:04:01

GooseFS-FUSE can mount a GooseFS system to a local file system on a Unix machine. Using this feature, some standard command-line tools (such as ls, cat, and echo) can directly access data in the GooseFS system. More importantly, applications implemented in different languages, such as C, C++, Python, Ruby, Perl, and Java, can use standard POSIX APIs (such as open, write, and read) to read and write GooseFS without any client-side integration and setup of GooseFS.
GooseFS-FUSE is based on the FUSE project and supports most file system operations. However, due to the inherent properties of GooseFS, such as its one-time, immutable file data model, the mounted file system does not fully conform to POSIX standards and has some limitations. Therefore, read Limitations first to see what this feature does and what its limitations are.

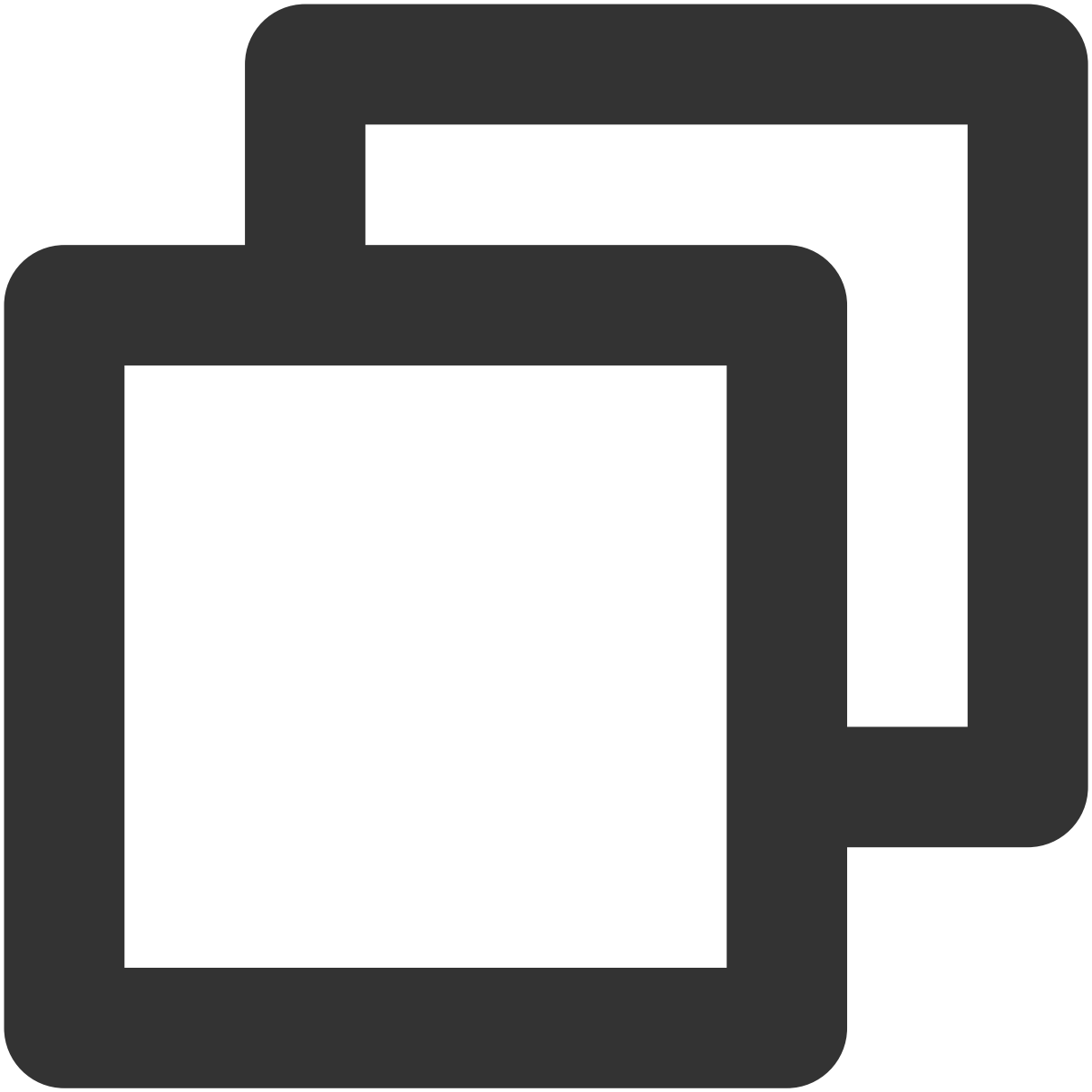
Installation Requirements

JDK 1.8 or later
Linux system: libfuse 2.9.3 or later (Version 2.8.3 can be used, but there will be some alerts.)
MAC system: osxfuse 3.7.1 or later

Usage

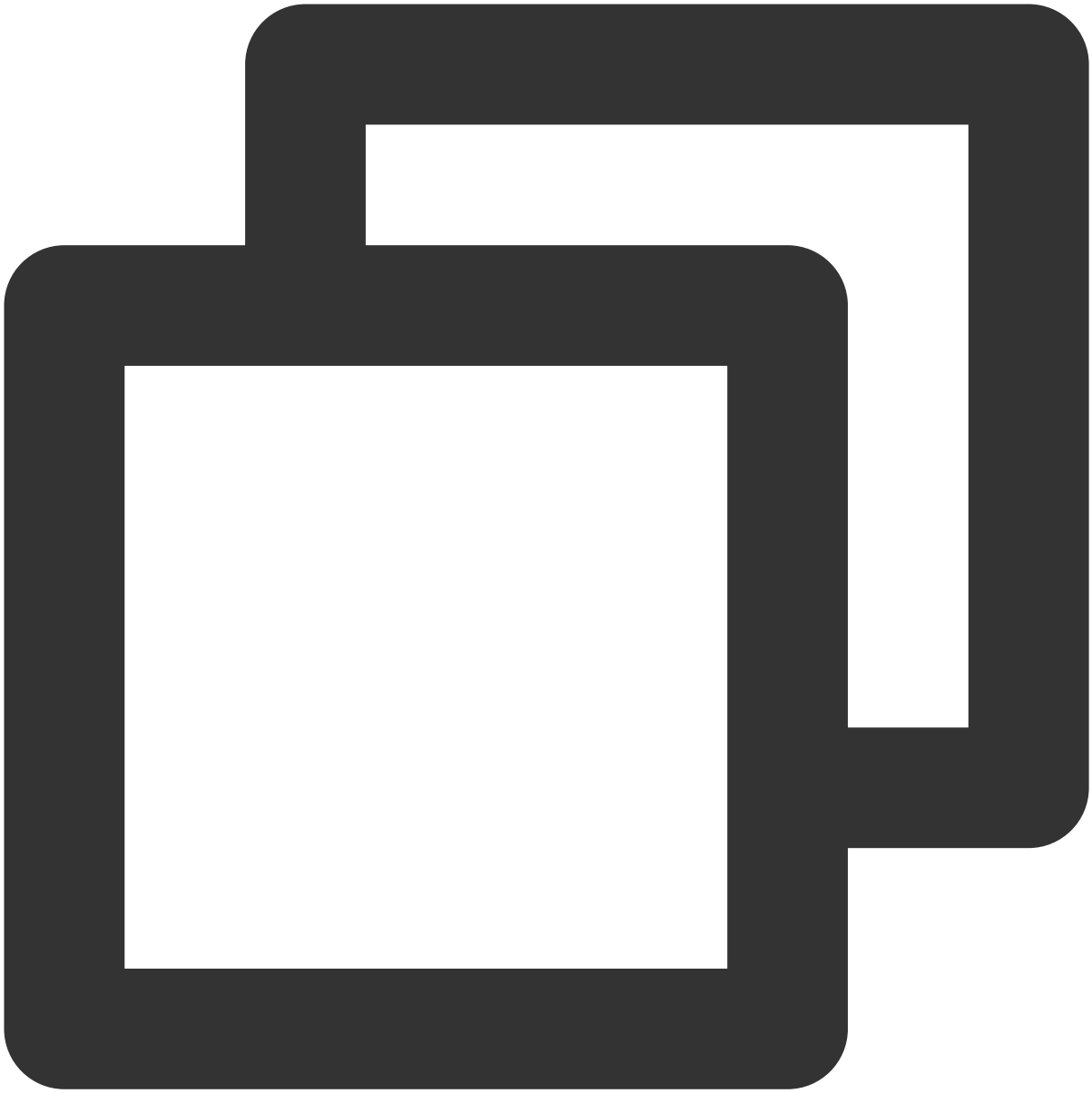
Mounting GooseFS-FUSE

After configuring and starting the GooseFS cluster, start Shell on the node where you want to mount GooseFS, go to the `$GOOSEFS_HOME` directory, and run the following command:



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

This command starts a background Java process that mounts the corresponding GooseFS path to the path specified by `<mount_point>` . For example, mount the GooseFS path `/people` to the `/mnt/people` directory on the local file system.



```
$ 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
```

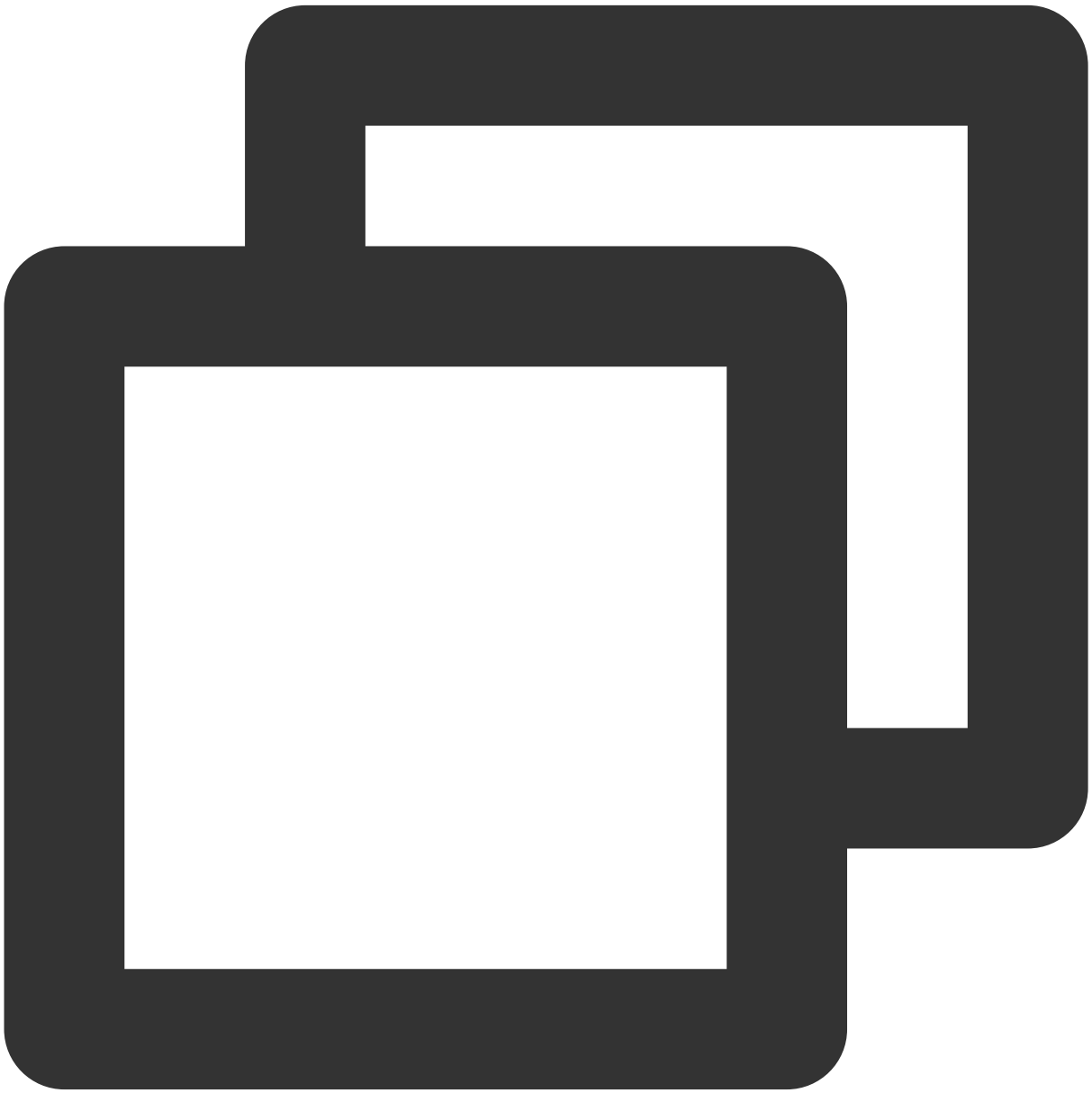
If `GooseFS_path` is not specified, GooseFS-FUSE is mounted to the GooseFS root directory (/) by default. You can run this command multiple times to mount GooseFS to different local directories. All GooseFS-FUSE clients will share the `$GOOSEFS_HOME\\logs\\fuse.log` log file. This log file is useful for troubleshooting.

Note:

`<mount_point>` must be an empty folder in the local file system, and the user that starts the GooseFS-FUSE process must have the read/write permission on the mount target.

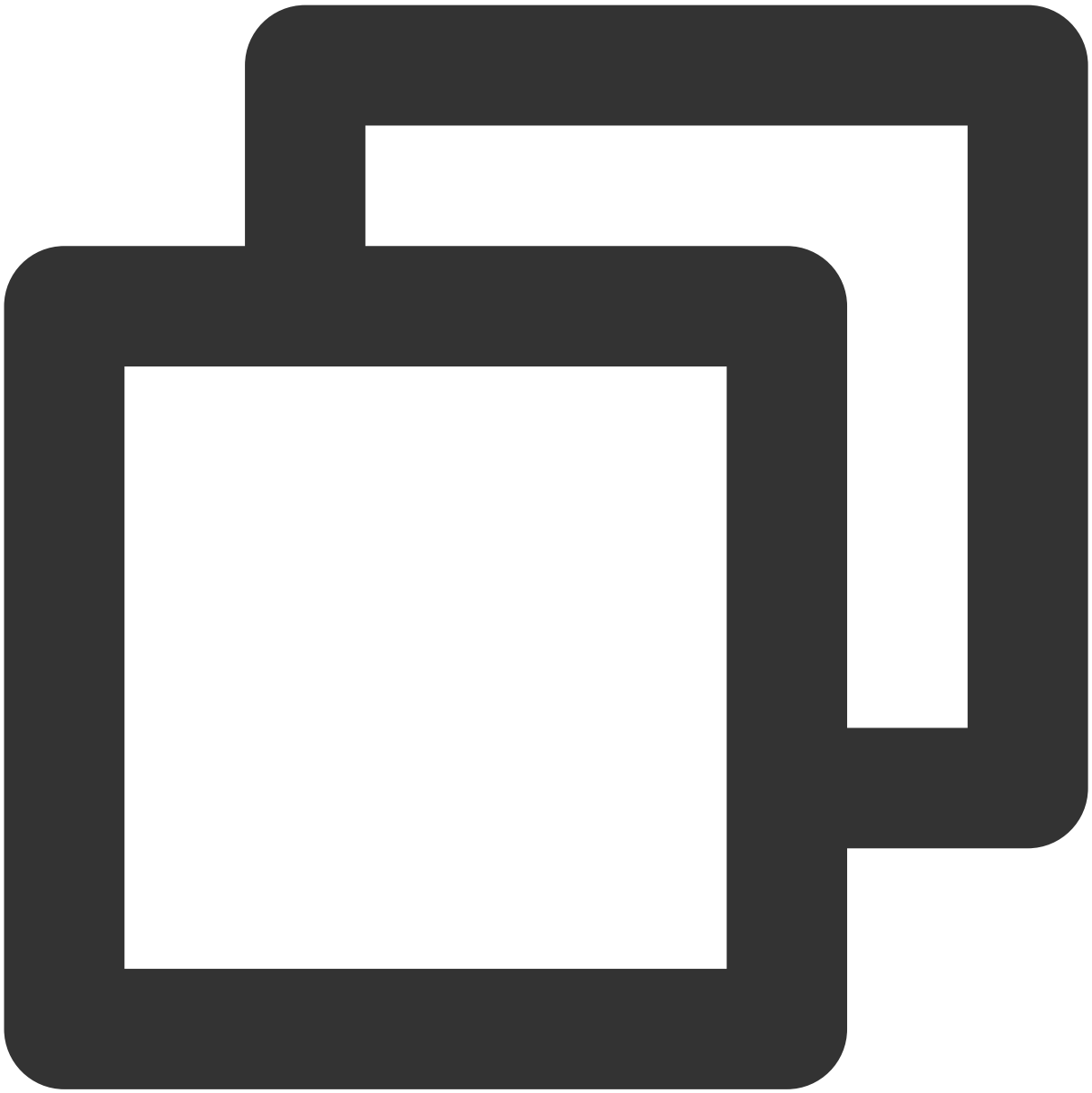
Unmounting GooseFS-FUSE

To unmount GooseFS-FUSE, you need to start Shell on the node, go to the `$GOOSEFS_HOME` directory and run the following command:



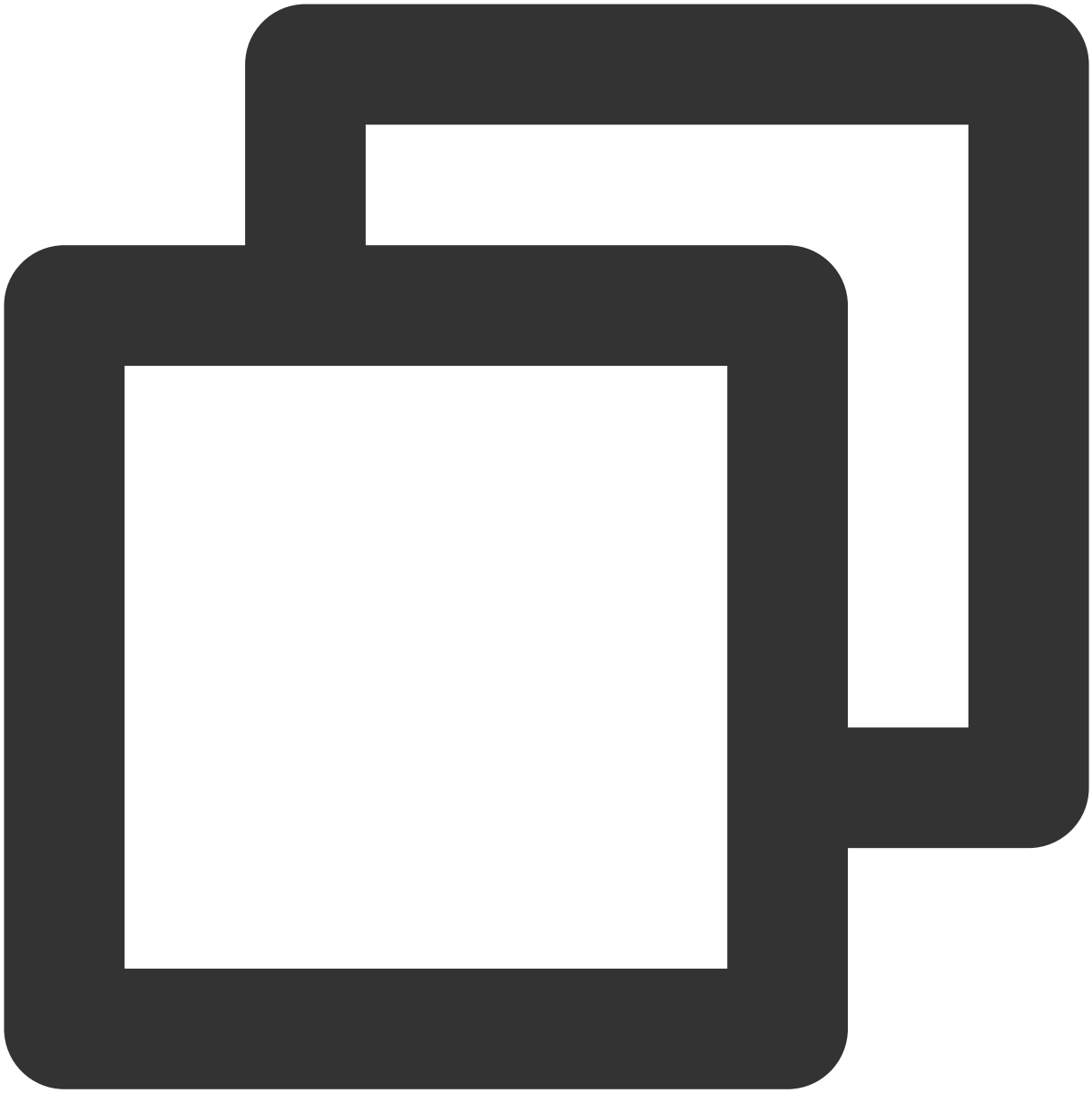
```
$ integration/fuse/bin/goosefs-fuse umount mount_point
```

This command ends the Java background process of GooseFS-FUSE and unmount the file system. The following is an example:



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

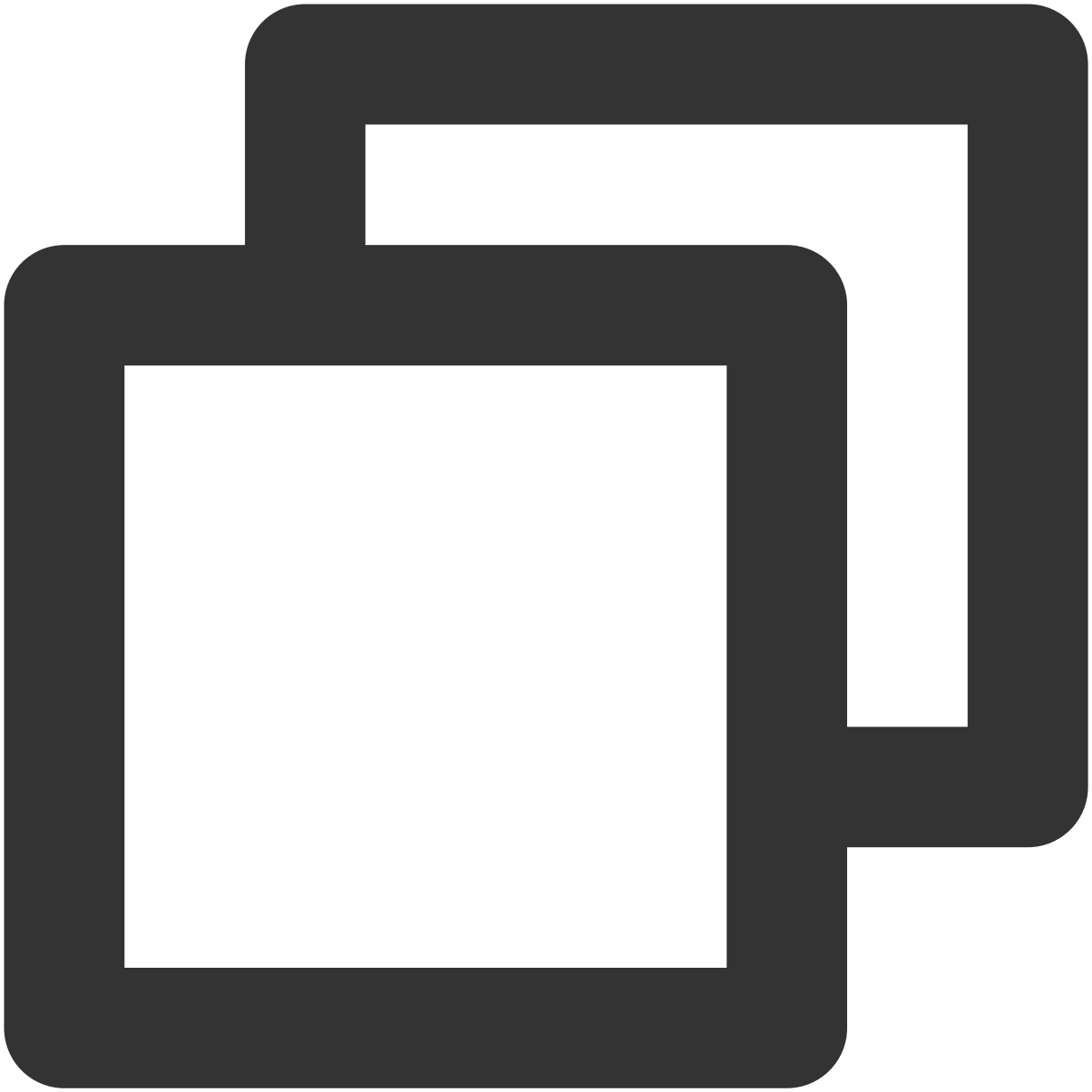
By default, if any read/write operation is not completed, the unmount operation will wait a maximum of 120 seconds. If the read/write operation does not complete after 120 seconds, the FUSE process will be forcibly ended, which will cause the current file read/write to fail. You can add the `-s` parameter to prevent the FUSE process from being forcibly ended, for example:



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

Checking whether GooseFS-FUSE is running

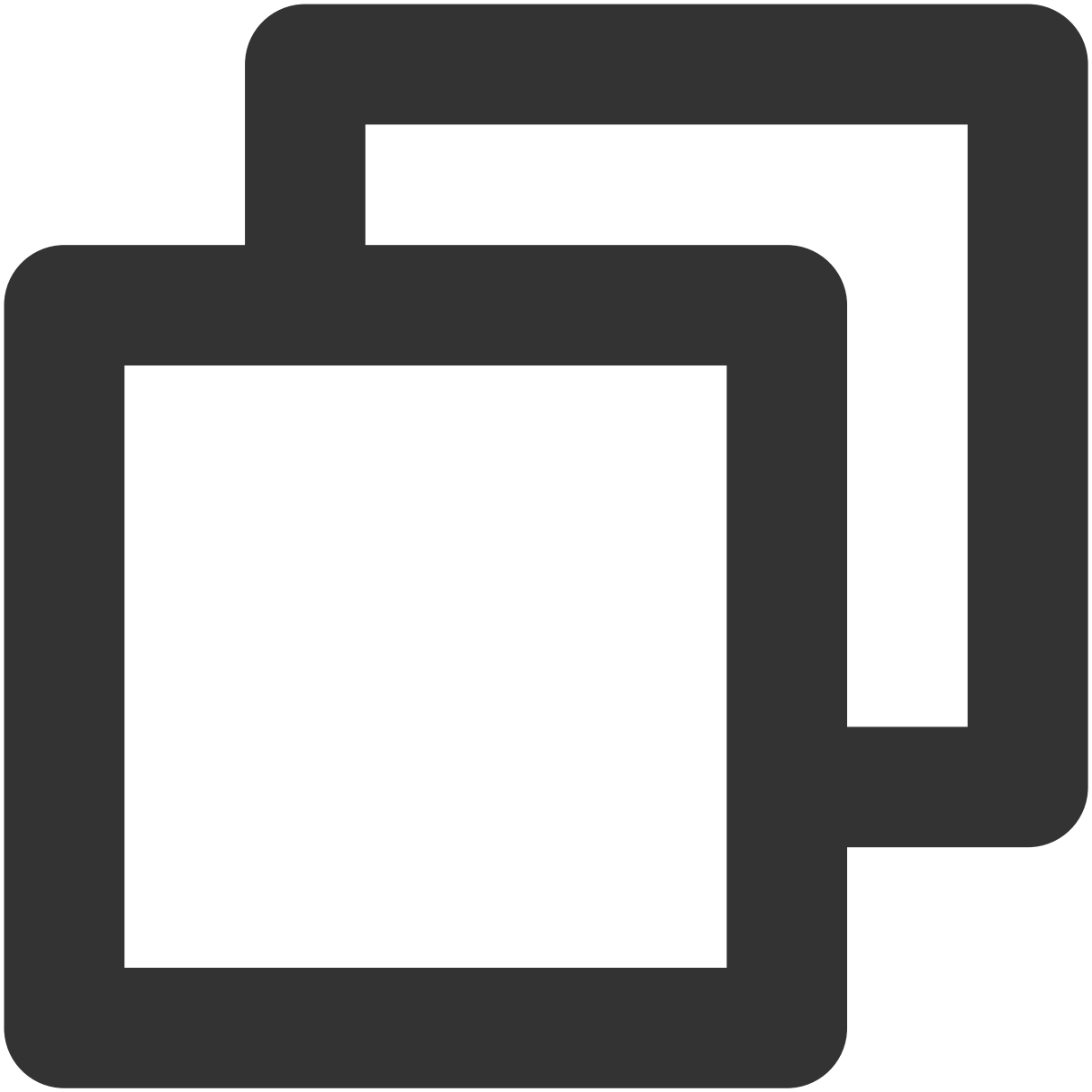
List all mount targets, start Shell on the node, go to the `$GOOSEFS_HOME` directory, and run the following command:



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

This command will output information including `pid` , `mount_point` and `GooseFS_path` .

The following is an example of the output format:



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

GooseFS-FUSE directory structure

The structure of the `conf` directory is as follows:

masters: master server IP configuration file

workers: worker server IP configuration file

goosefs-site.properties: GooseFS configuration file

libexec: library file on which GooseFS-FUSE depends to run

goosefs-fuse-1.4.0: GooseFS-FUSE JAR package to run in the background

log: log directory

Configuration Options

GooseFS-FUSE performs operations based on the standard GooseFS-core-client-fs. If you want it to perform operations like other applications, you can customize the behavior of GooseFS-core-client-fs.

You can edit the `$GOOSEFS_HOME/conf/goosefs-site.properties` configuration file to modify the client options.

Note:

All modifications must be completed before GooseFS-FUSE is started.

Limitations

Currently, GooseFS-FUSE supports most basic file system operations. However, due to some inherent features of GooseFS, you need to be aware of the following:

Files cannot be written randomly and additionally.

Files can be written sequentially only once and cannot be modified. If you want to modify a file, first delete the file and then recreate it or open the file with the `O_TRUNC` identifier and set its length to `0` .
Files being written in the mount point cannot be read.
The file length cannot be truncated.
Commands related to soft/hard link are not supported. GooseFS does not have the concepts of hard link or soft link, so it does not support commands related to them, such as `ln` . In addition, information about hard links is not shown in the output of `ll` .
When Cloud Object Storage (COS) is used as the underlying storage, the `Rename` operation is nonatomic.
Only when the `GooseFS.security.group.mapping.class` option of GooseFS is set to the value of `ShellBasedUnixGroupsMapping` , the user and group information of a file corresponds to the user group of a Unix system. Otherwise, the `chown` and `chgrp` operations do not take effect, and `ll` returns the information of the user and group that started the GooseFS-FUSE process.

Performance Considerations

Due to the combination of FUSE and JNR, the performance of mounting file systems is relatively lower compared to using native file system APIs directly.
Most of the performance problems are due to the fact that there are several copies in the memory each time a read or write operation is performed, and FUSE sets the maximum granularity of write operations to 128 KB. Performance can be greatly improved with the FUSE write-backs cache policy introduced by Kernel 3.15 (however, the libfuse 2.x userspace library currently does not support this feature).

GooseFS-FUSE Parameters

The following are GooseFS-FUSE related parameters:

Parameter	Default Value	Description
goosefs.fuse.cached.paths.max	500	Defines the size of the internal GooseFS-FUSE cache, which maintains the most common conversions between local file system paths and Alluxio file URIs.
goosefs.fuse.debug.enabled	false	Allows FUSE debugging output, which is redirected to the `fuse.out` log file in the directory specified by `goosefs.logs.dir` .
goosefs.fuse.fs.name	goosefs-	Descriptive name used by FUSE to mount a file

	fuse	system.
goosefs.fuse.jnifuse.enabled	true	Use the JNI-FUSE library for higher performance. If the JNI-FUSE library is disabled, JNR-FUSE will be used.
goosefs.fuse.shared.caching.reader.enabled	false	(Experimental) Use a shared gRPC data reader to achieve higher performance in multi-process file reading by using GooseFS JNI-FUSE. Block data will be cached on the client, so the FUSE process requires more memory.
goosefs.fuse.logging.threshold	10s	Record FUSE API calls when the time taken exceeds the threshold.
goosefs.fuse.maxwrite.bytes	131072	Granularity (in bytes) of FUSE write operations. Note that 128 KB is currently the upper limit for the Linux kernel limit.
goosefs.fuse.user.group.translation.enabled	false	Whether to convert GooseFS users and groups to the corresponding Unix users and groups in the FUSE API. When set to `false`, the users and groups of all FUSE files are displayed as the users and groups that mount the GooseFS-FUSE thread.

FAQs

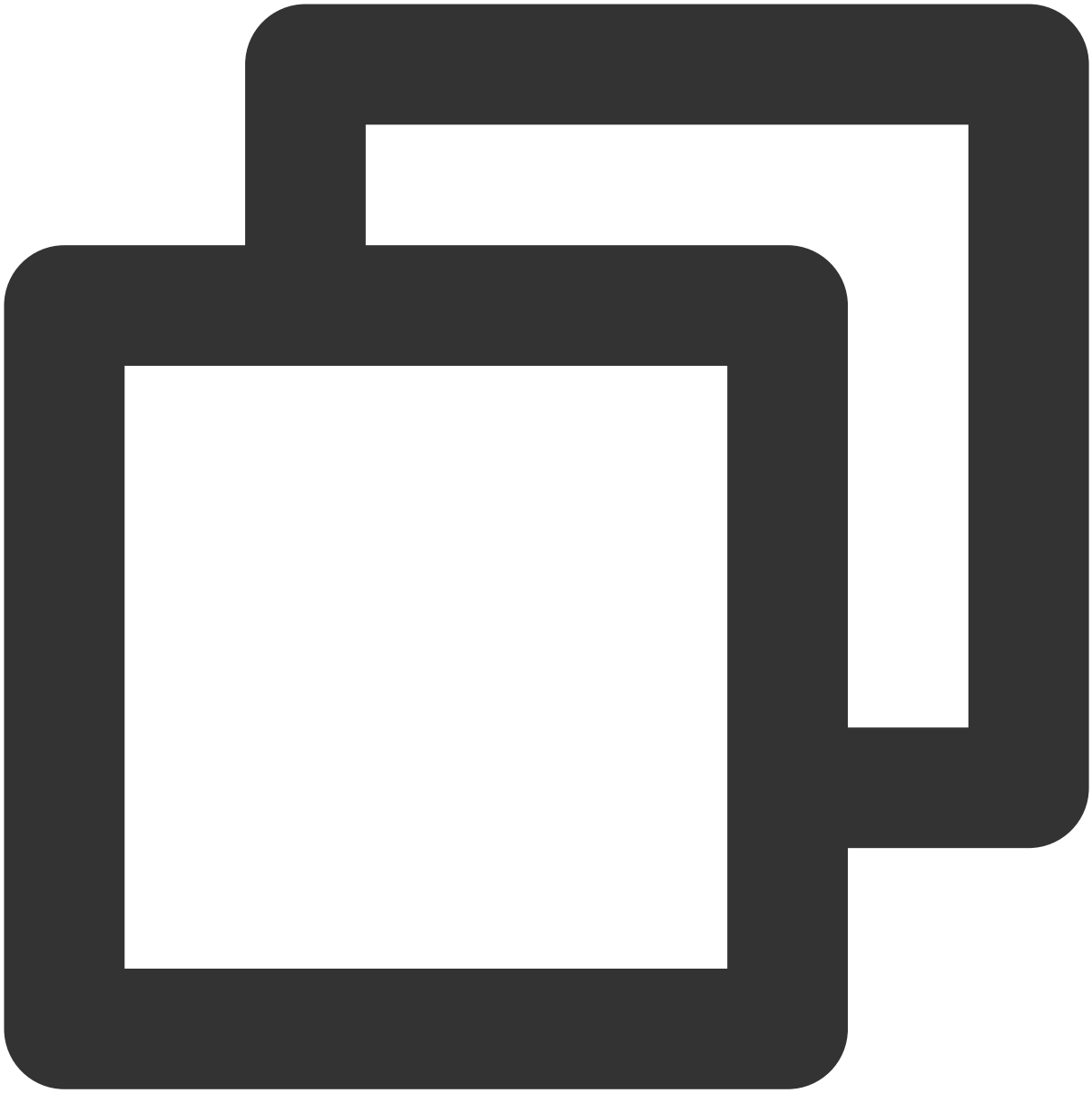
Missing libfuse library file

You need to install libfuse before mounting GooseFS-Fuse.

```
2021-10-13 20:04:42,970 INFO  TieredIdentityFactory - Initialized tiered identity TieredIdentity(node=10.91.27.82, ra
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/libjnifuse9578790
5_/sp1cwpdd1xn9w96x5xkw7qnc0000gn/T/libjnifuse957879065968834264.jnilib, 1): Library not loaded: /usr/local/lib/libfu
  Referenced from: /private/var/folders/5_/sp1cwpdd1xn9w96x5xkw7qnc0000gn/T/libjnifuse957879065968834264.jnilib
  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.java:105)
        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
```

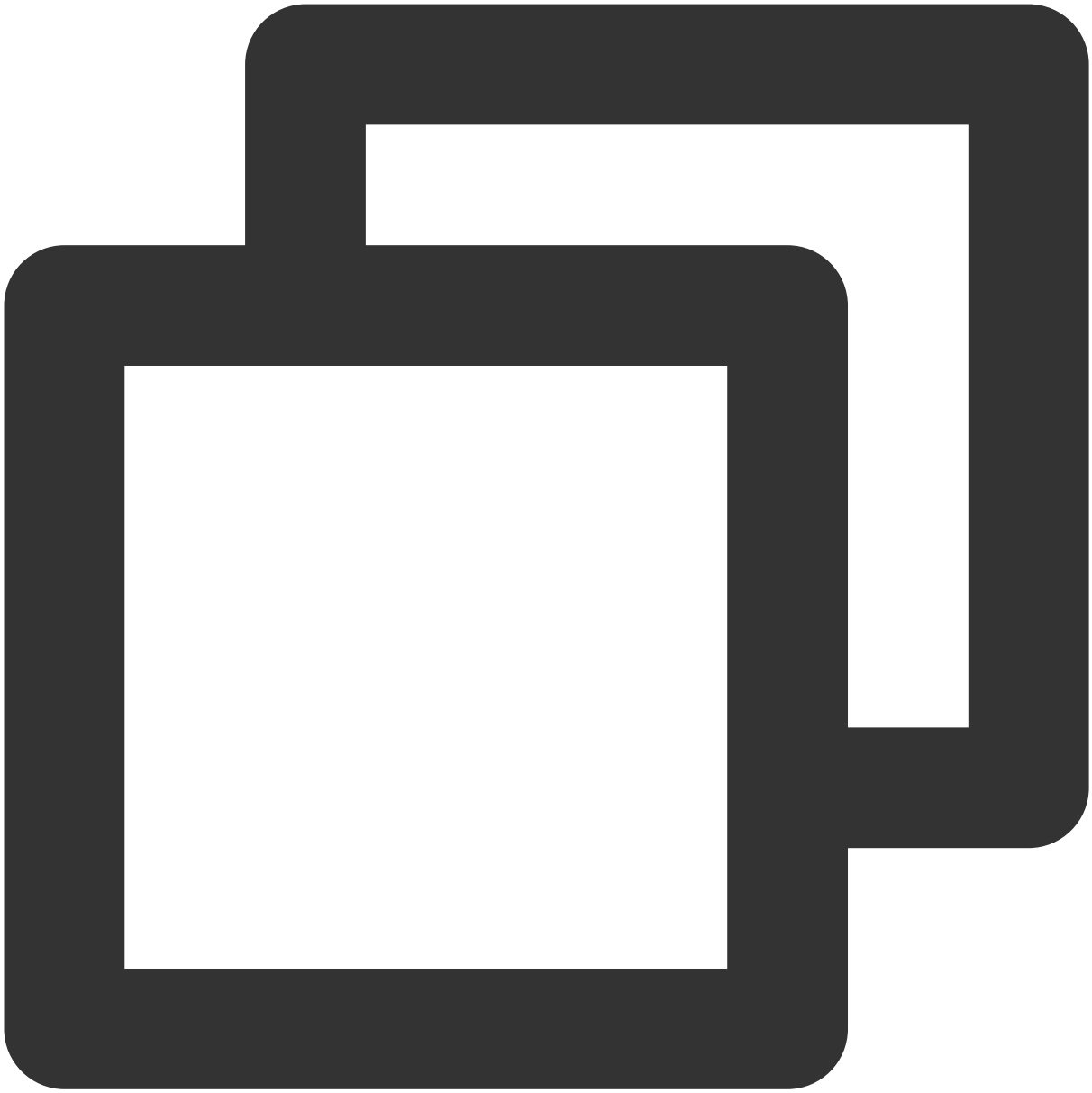
Option 1

Installation command:



```
yum install fuse-devel
```

Check whether the installation is successful:



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

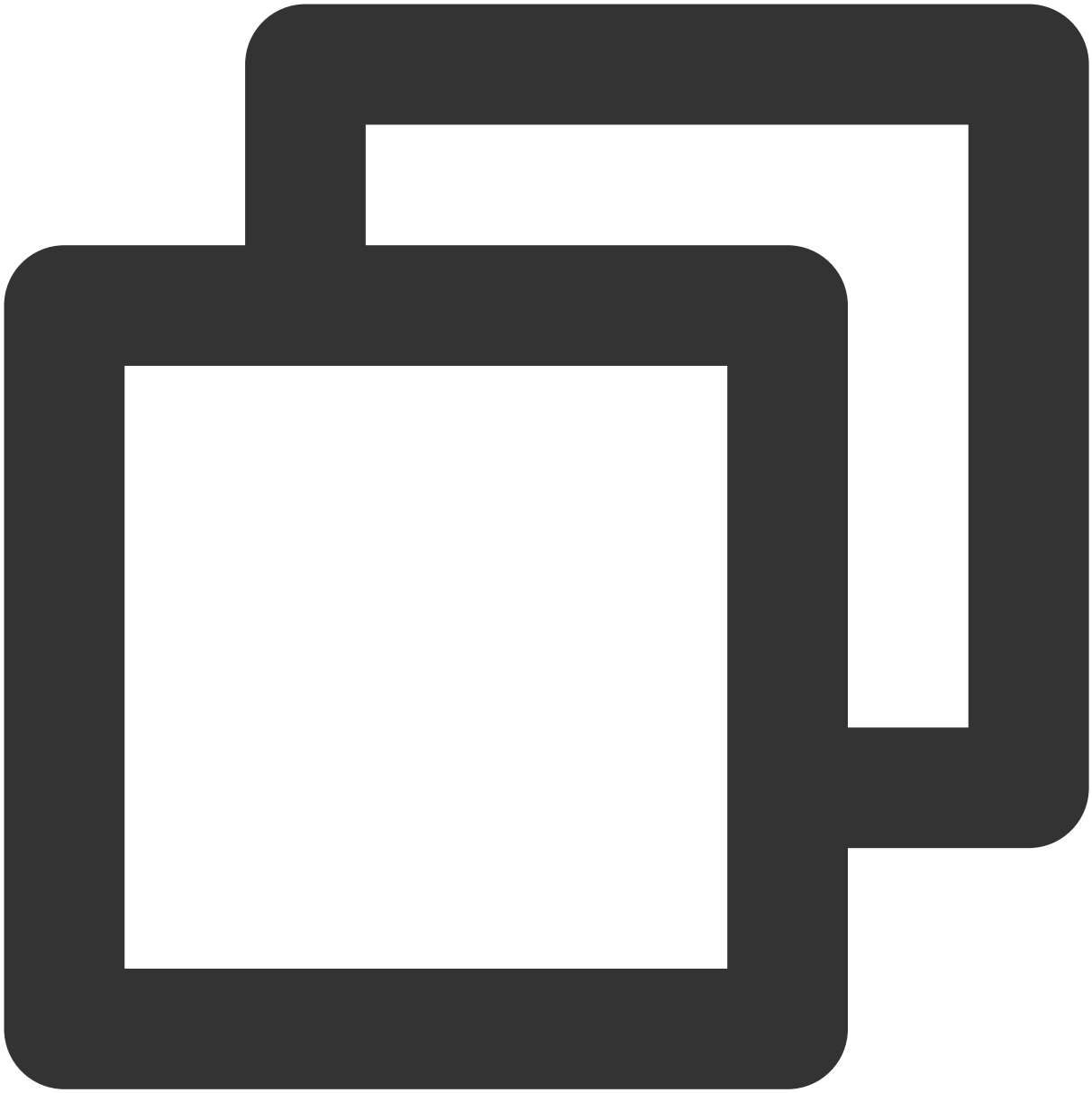
Option 2

Update the old version libfuse.so.2.9.2. The procedure is as follows:

Note:

If libfuse is installed in CentOS 7, libfuse.so.2.9.2 is installed by default.

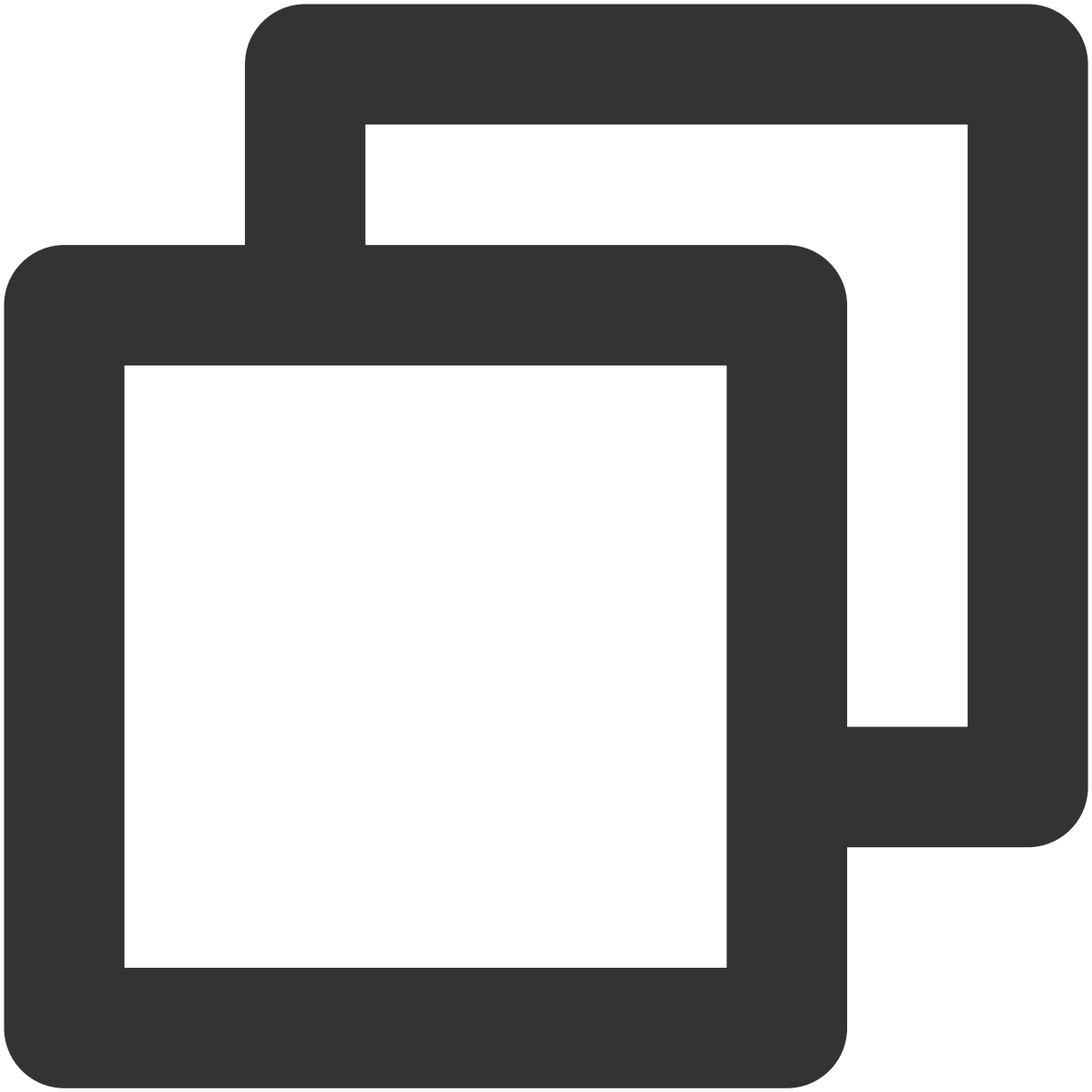
Download the libfuse source code and compile and generate libfuse.so.2.9.7.



```
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
```

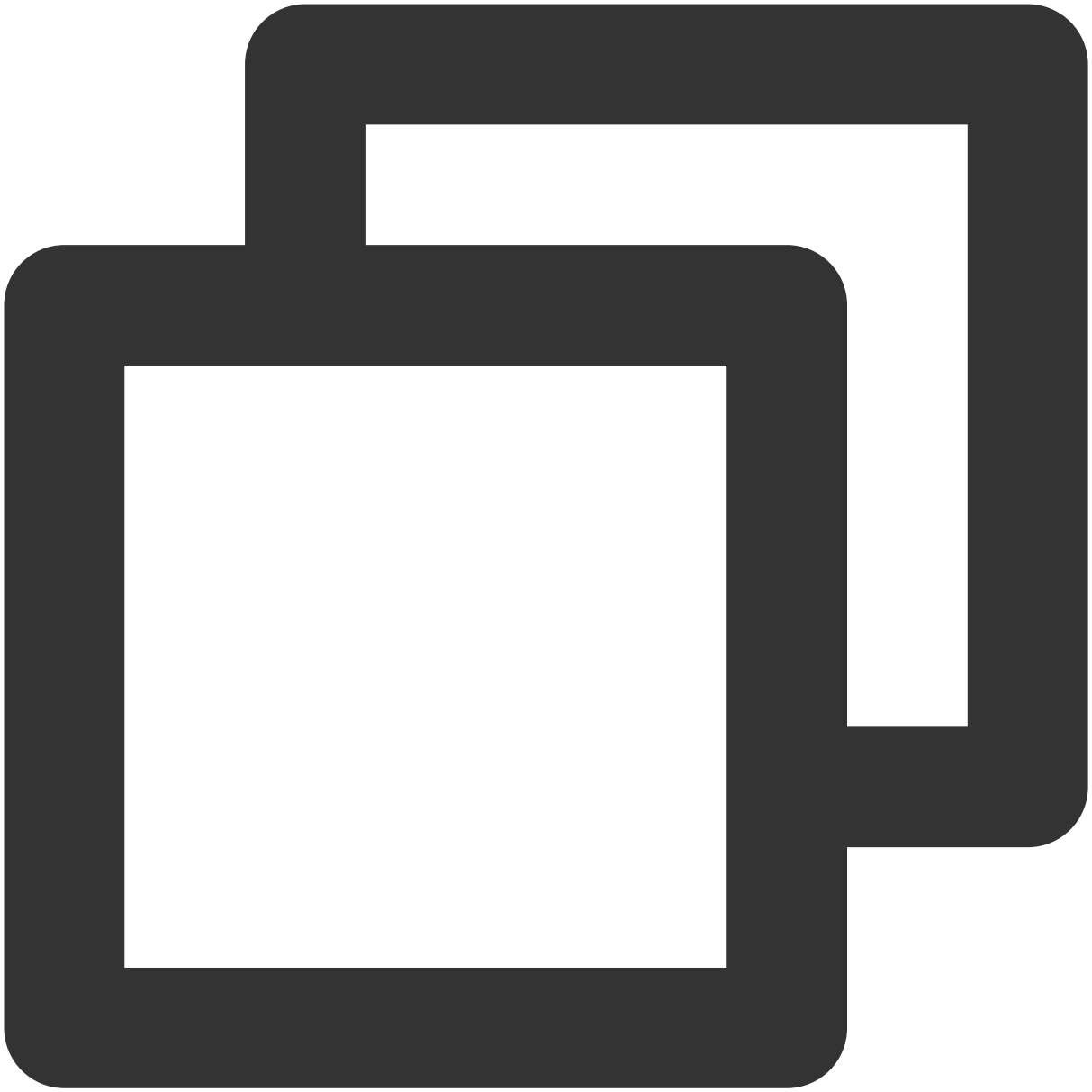
Install libfuse.so.2.9.7 to replace the old version:

1.1 Run the following command to find the links of the libfuse.so.2.9.2 library:



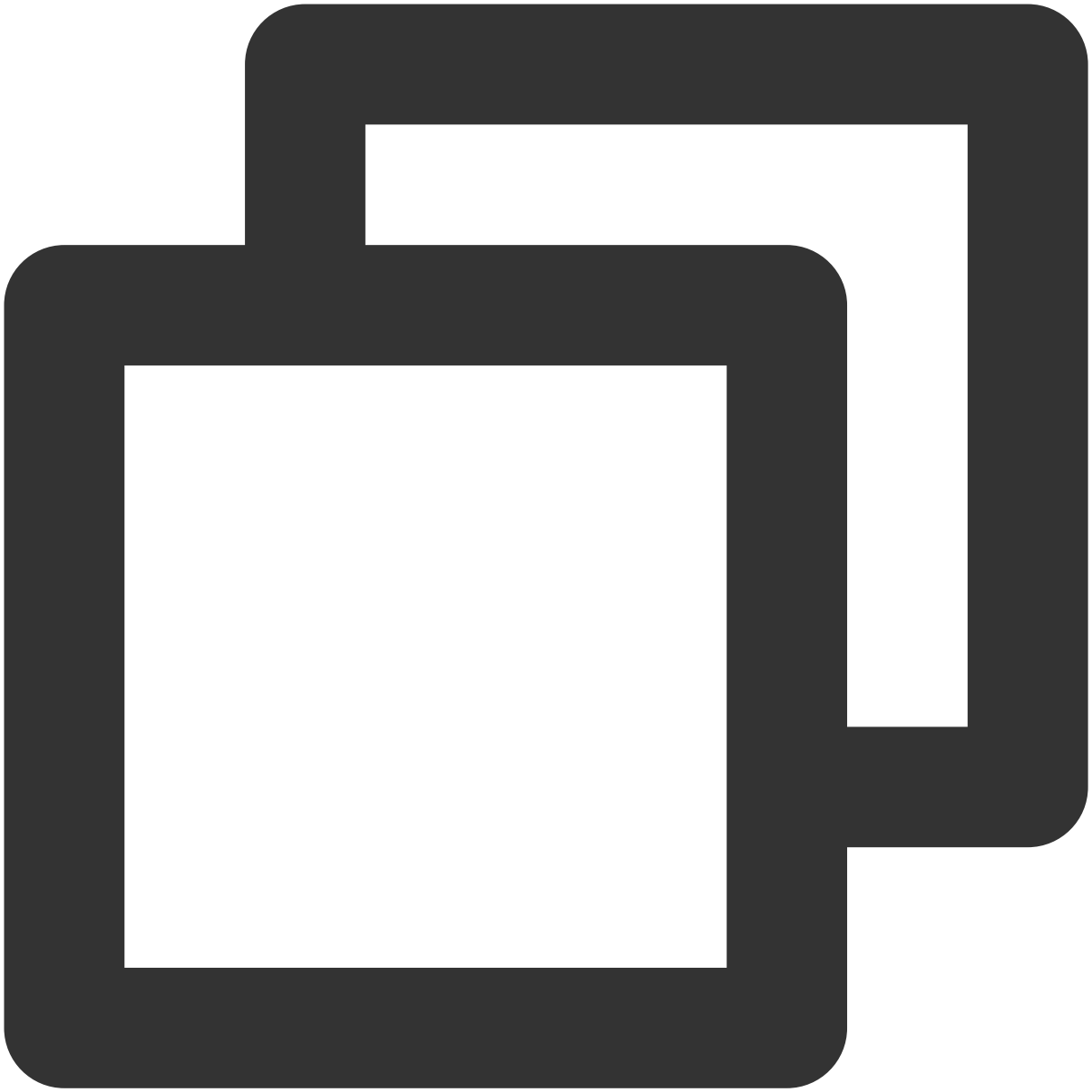
```
find / -name libfuse.so*
```

1.2 Run the following command to copy libfuse.so.2.9.7 to the location of the old version libfuse.so.2.9.2 library:



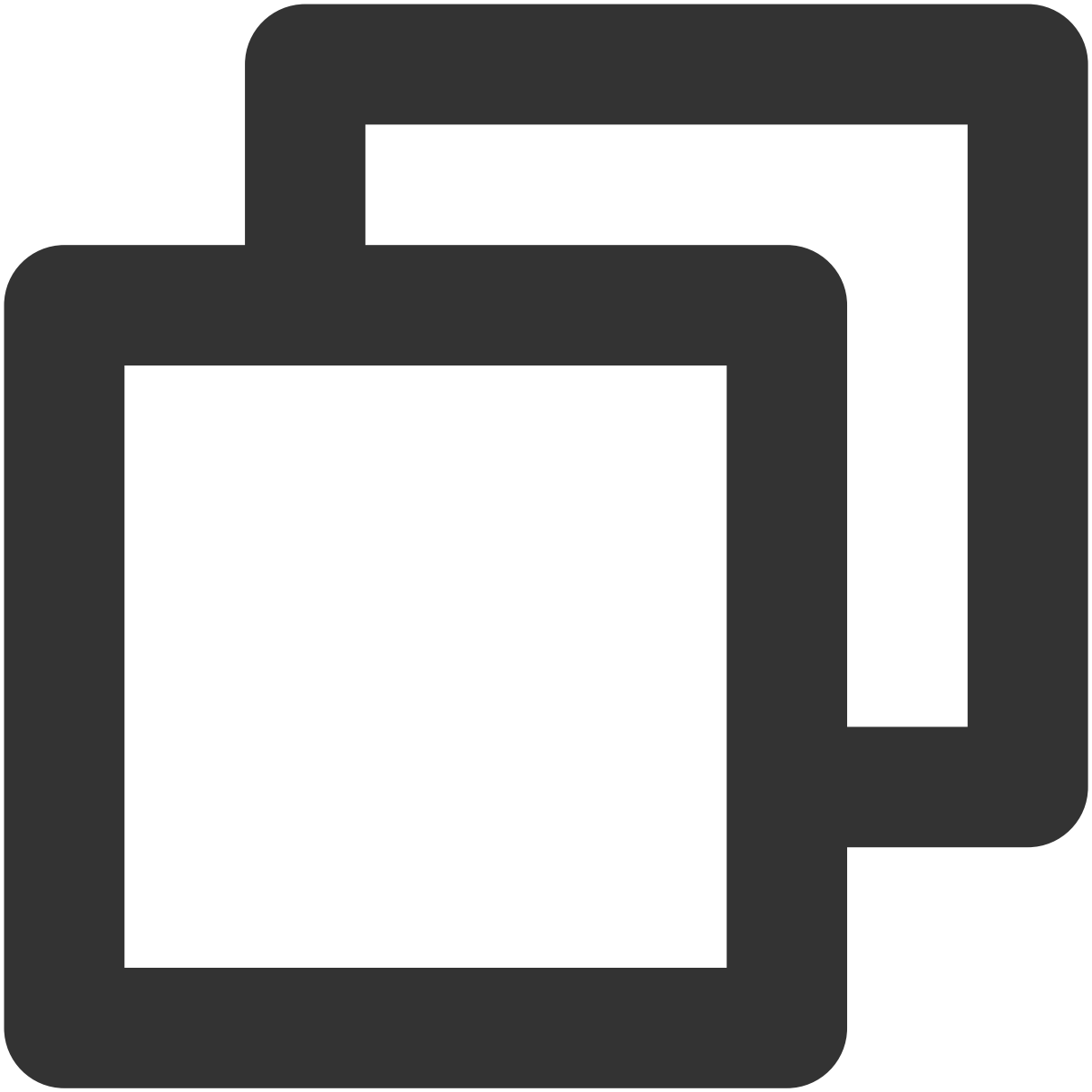
```
cp /usr/local/lib/libfuse.so.2.9.7 /usr/lib64/
```

1.3 Run the following commands to delete all links of the old version libfuse.so library:



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

1.4 Run the following commands to build libfuse.so.2.9.7 library links similar to those of the old version library.



```
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
```

“Write error in swap file” error during editing a file in the mounting point by using VIM

You can change the VIM configuration to use VIM 7.4 or earlier for editing files in the GooseFS-Fuse mounting point. The VIM swap file is used to retain the modifications so that VIM can restore unsaved modifications based on the swap file if the machine crashes or restarts. The above error is due to a random write operation to a VIM swap file,

while GooseFS does not support it. Solution: You can run the `:set noswapfile` command to terminate swap file generation or add `set noswapfile` to the configuration file (“~/.vimrc”).

Deployment Guide
Deploying with a Self-Built Cluster

Last updated：2024-03-25 16:04:01

This document describes how to deploy, configure, and run GooseFS on a single server, in a cluster, and in the Tencent Cloud EMR cluster that has not integrated with GooseFS.

Environment Requirements

Hardware environments

Currently, GooseFS can work with Linux/MacOS running a mainstream x86/x64 framework (**Note: M1-powered MacOS has not been verified**). Configuration requirements for nodes are described as follows:

Master nodes

CPU: A clock rate of 1 GHz or above. Since the master node needs to process a large amount of data, we recommend you use multi-core processors in the production environment.
Physical memory: 1 GB or above. We recommend you use at least 8 GB of physical memory in the production environment.
Disk: 20 GB or above. We recommend you use high-performance NVME SSD as the metadata caching disk (RocksDB mode) in the production environment.

Worker nodes

CPU: A clock rate of 1 GHz or above. Since a worker node also needs to handle a lot of concurrent requests, we recommend you use multi-core processors in the production environment.
Physical memory: 2 GB or above. You can allocate memory for the production environment according to your performance needs. For example, if you need to cache a lot of data blocks to the worker node memory, or use mixed storage (MEM + SSD/HDD), you can allocate the physical memory according to the volume of data that might be cached to the memory. We recommend you use at least 16 GB of physical memory for workers in the production environment, regardless of what caching mode you use.
Disk: An SSD/HDD disk of at least 20 GB. We recommend you allocate disk space for each worker node by estimating the amount of data that might need to be cached to the production environment. Moreover, for better performance, we recommend you use NVME SSD disks for worker nodes.

Software environments

Red Hat Enterprise Linux 5.5 or later, Ubuntu Linux 12.04 LTS or later (supported if batch deployment is not used), CentOS 7.4 or later, TLinux 2.x (Tencent Linux 2.x), and Intel-based MacOS 10.8.3 or later. If you use Tencent Cloud CVM, we recommend you use CentOS 7.4, Tencent (TLinux 2.x), or TencentOS Server 2.4.
OpenJDK 1.8/Oracle JDK 1.8. Note that JDK 1.9 or later versions have not been verified.
Supports SSH tools (including SSH and SCP tools), Expect Shell (for batch deployment), and Yum Package Manager.

Cluster network environments

Master and worker nodes need to be connected via a public/private network. If batch deployment is used, the master node needs to be able to access the public software source properly, or can correctly configure private software sources in the package management system.
The cluster can access COS buckets or CHDFS over a public/private network.

Security group and permission requirements

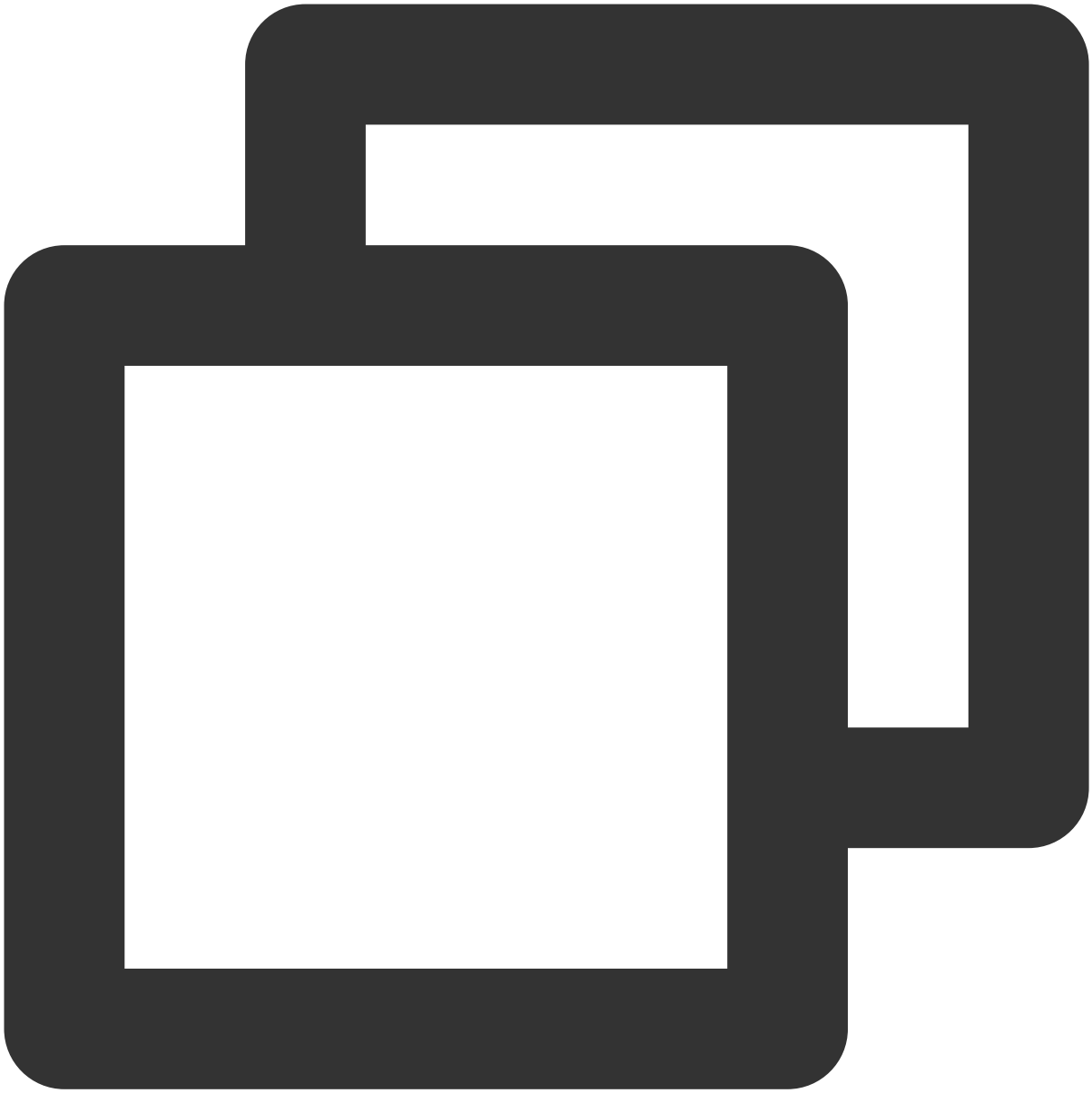
In most cases, you are advised to use a dedicated Linux account to deploy and run GooseFS. For example, in the self-built cluster and EMR environment, you can use the Hadoop user to deploy and run GooseFS. If batch deployment is used, the following permissions are also needed:
Permission to switch to the root account or use `sudo` .
The running and deployment account needs to have permission to read and write to the installation directory.
The master node should have permission to use SSH to log in to all worker nodes in the cluster.
The corresponding node role in the cluster should open the corresponding port(s) (master: 9200 and 9201; worker: 9203 and 9204; job master: 9205, 9206, and 9207; job worker: 9208, 9209, and 9210; proxy: 9211; LogServer: 9212).

Single-Server Deployment and Running (Pseudo-Distribution Framework)

The pseudo-distribution framework is mainly used for trying out and debugging GooseFS. Beginners can try out and debug GooseFS on a host running Linux OS or macOS.

Deployment

1. Download the GooseFS binary distribution package).
2. Decompress the package, go to the GooseFS directory, and perform the operations below.
Create the `conf/goosefs-site.properties` configuration file by copying `conf/goosefs-site.properties.template` .

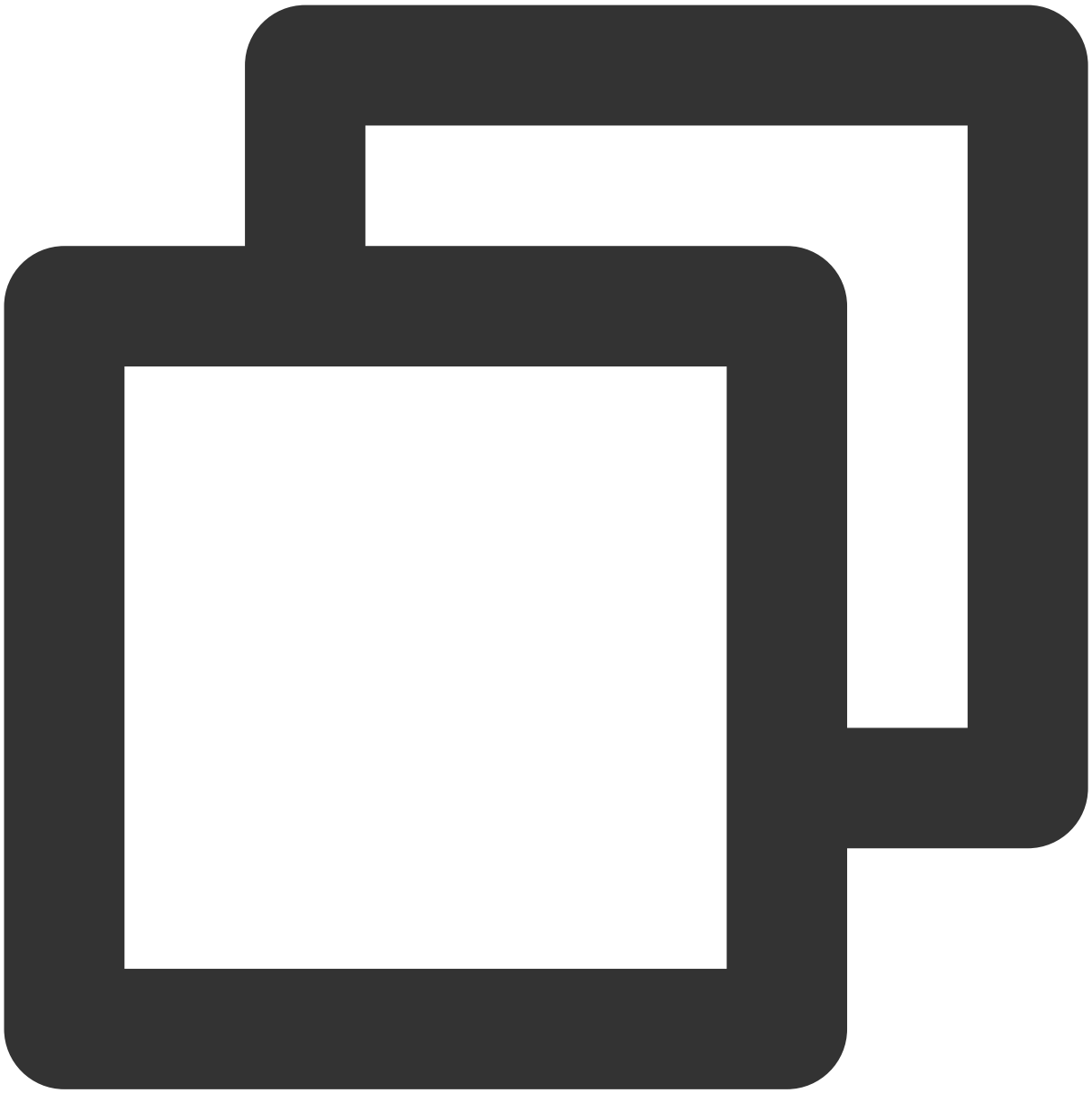


```
$ cp conf/goosefs-site.properties.template conf/goosefs-site.properties
```

Set `goosefs.master.hostname` to `localhost` in `conf/goosefs-site.properties`.
In `conf/goosefs-site.properties`, set `goosefs.master.mount.table.root.ufs` to the directory in the local file system, such as `/tmp` or `/data/goosefs`.
You are advised to set SSH passwordless login for localhost. Otherwise, you need to enter its login password for operations such as formatting and starting.

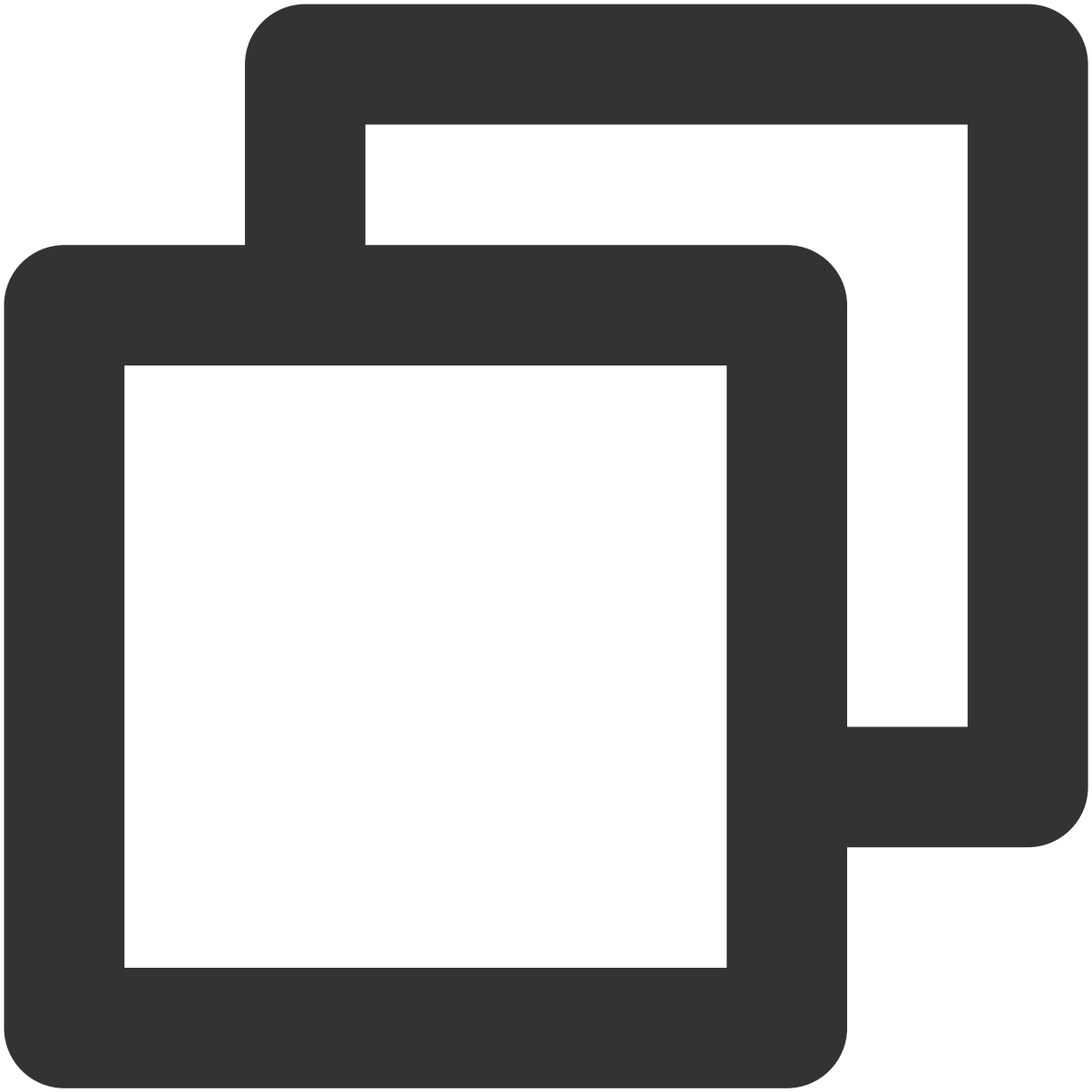
Running

Run the following command to mount a RamFS file system:



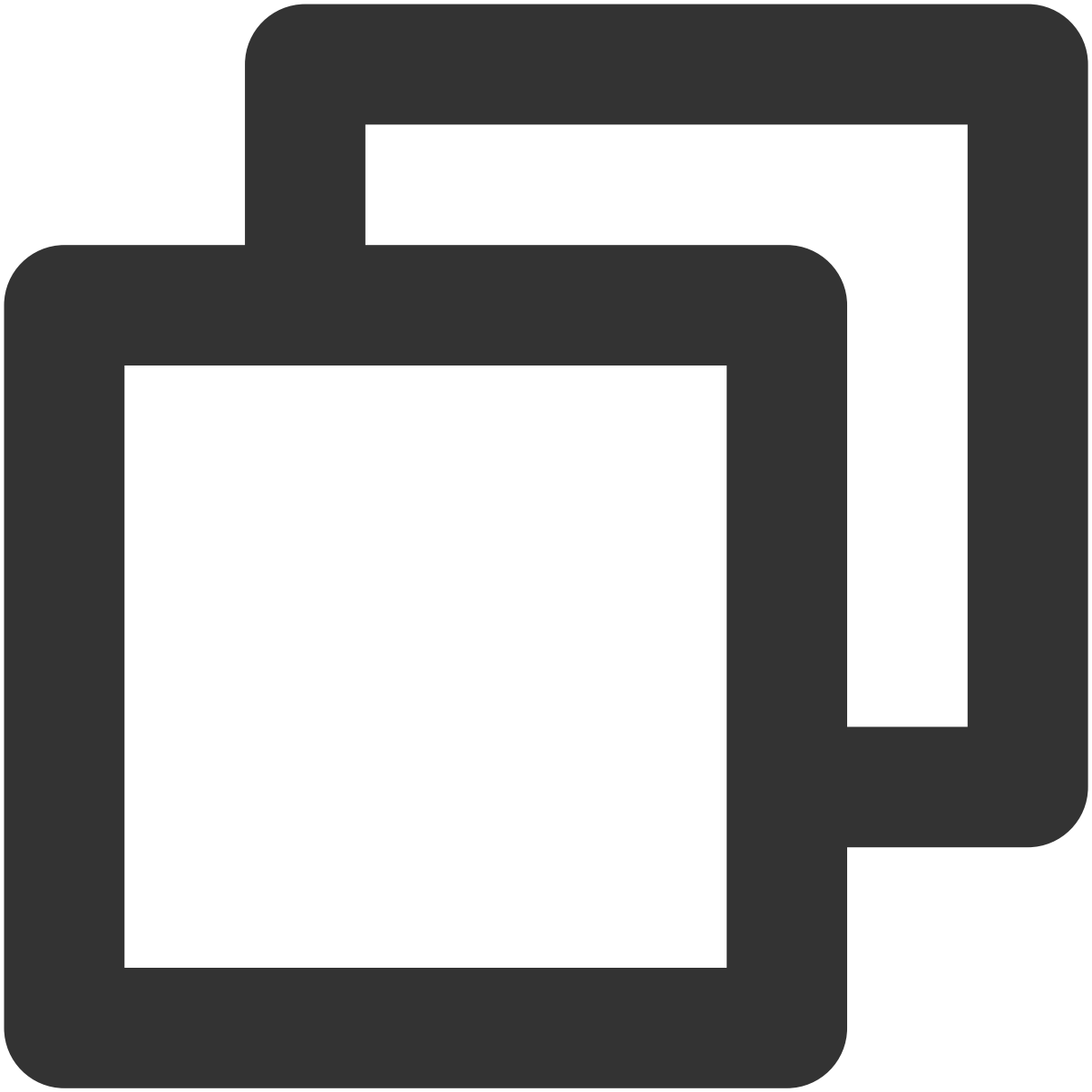
```
$ ./bin/goosefs-mount.sh SudoMount
```

You can also mount it directly when you start the GooseFS cluster as follows:



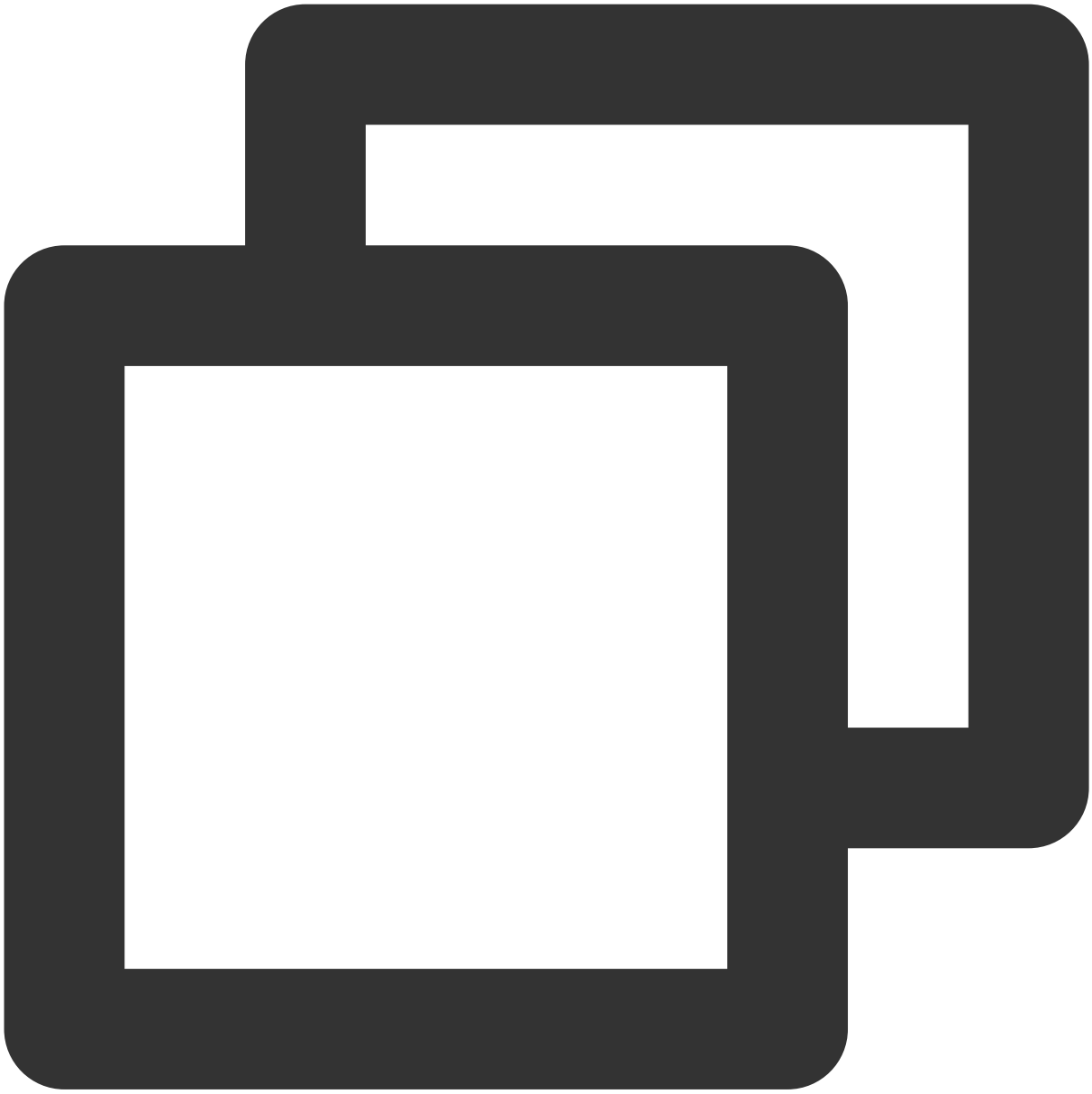
```
$ ./bin/goosefs-start.sh local SudoMount
```

When the cluster is started, run the `jps` command to view all GooseFS processes in the pseudo-distribution mode.



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

After this, you can run the `goosefs` command to perform operations related to namespace, fileSystem, job, and table. For example, you can run the following commands to upload a local file to GooseFS and list files and directories in the root directory of GooseFS:



```
$ 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
```

The GooseFS CLI tool allows you to perform all kinds of operations on namespaces, tables, jobs, file systems, and more to manage and access GooseFS. For more information, please see our documentation or run the `goosefs -h` command to view the help messages.

Cluster Deployment and Running

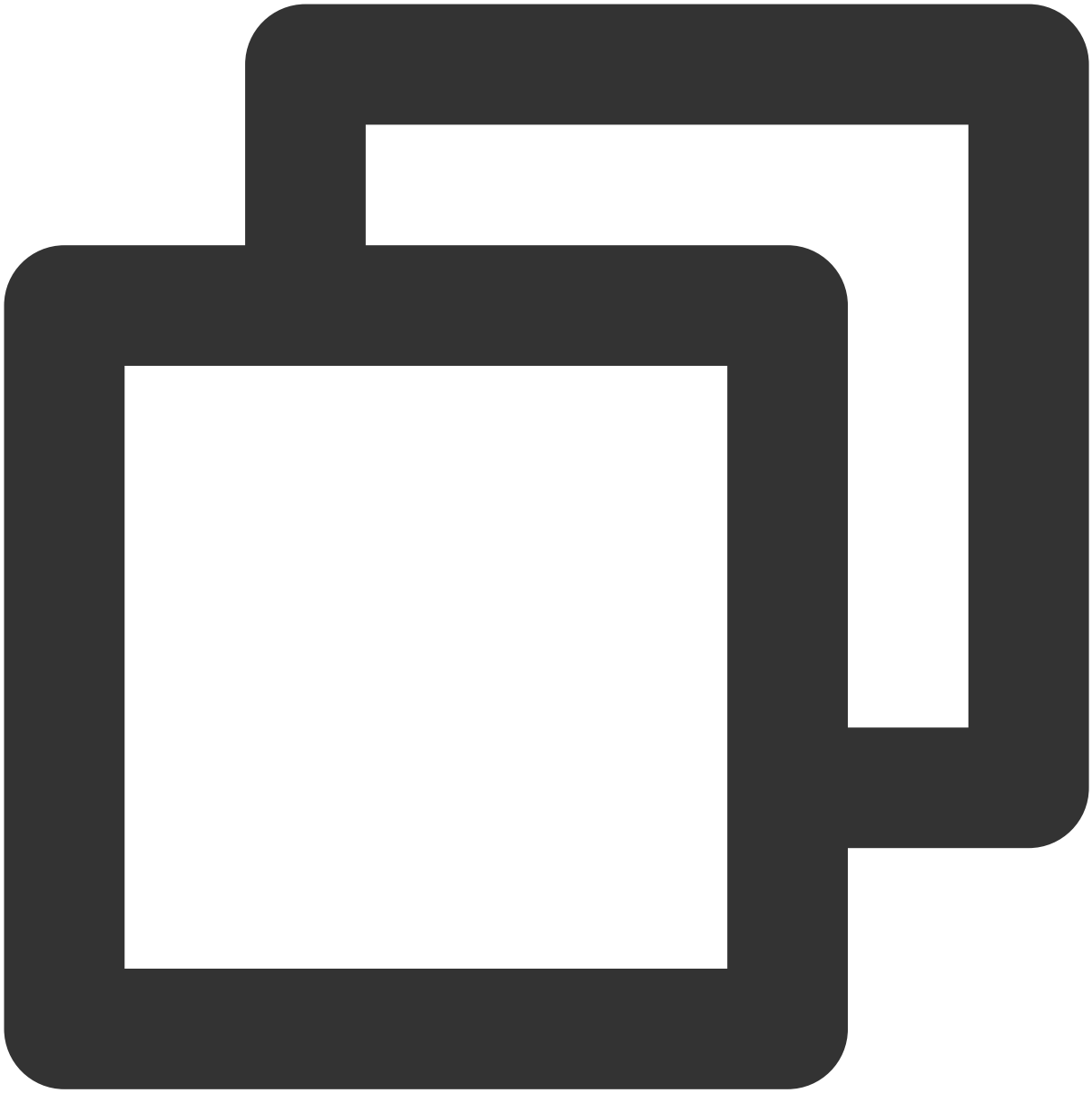
The cluster deployment and running are mainly for the production environment of self-built IDC clusters and Tencent Cloud EMR production environments that have not integrated with GooseFS. The deployments are classified into standalone framework deployment and high-availability (HA) framework deployment.
In the `scripts` directory, you can find scripts to configure SSH passwordless logins or deploy GooseFS in batches, which make it easy to deploy a large-scale GooseFS cluster. See the batch deployment requirements mentioned above to check whether you can use batch deployment.

Introduction to the batch deployment tool

GooseFS provides scripts in the `scripts` directory for you to configure SSH passwordless logins or deploy GooseFS in batches. If the execution conditions are met, you can perform the following steps to batch deploy jobs:
Enter the extracted GooseFS directory by running `cd ${GOOSEFS_HOME}` .
Configure the hostname or IP address in `conf/masters` and `conf/workers` .
Go to the `scripts` directory and configure the `SCRIPT_EXEC_USER` and `SCRIPT_EXEC_PASSWORD` carefully in the `commons.sh` script file. Then, switch to the `root` account or use `sudo` to run `config_ssh.sh` , so that you can configure passwordless logins for the entire cluster.
Run the `validate_env.sh` tool to validate the configuration of the cluster.
Create a soft link to the configuration file directory by running `ln -s conf ~/.goosefs` .
Finally, run `goosefs copy .` and `goosefs copy ~/.goosefs` to distribute the GooseFS binary package and configuration file to all nodes.
After a successful deployment, run `./bin/goosefs-start.sh all SudoMount` to start the entire cluster. By default, all running logs will be recorded to `${GOOSEFS_HOME}/logs` .

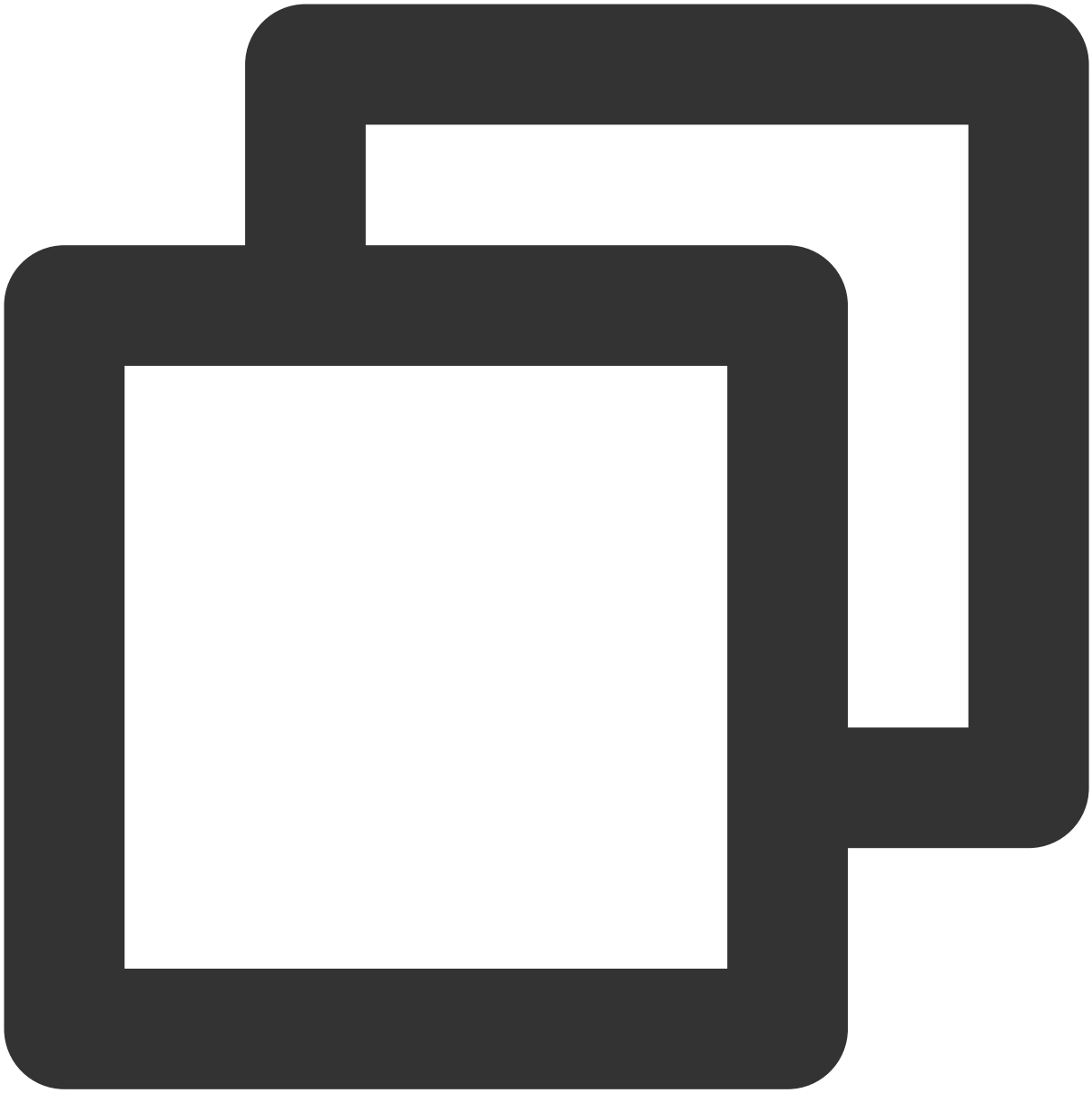
Standalone framework deployment

In the standalone framework, only one master node and multiple worker nodes are deployed in the cluster. You can deploy and run the cluster as follows:
1. Download the GooseFS binary distribution package).
2. Run the `tar zxvf goosefs-x.x.x-bin.tar.gz` command to decompress the package into the installation directory. You can see **Introduction to the batch deployment tool** to deploy and run your cluster in batches, or perform the following steps to deploy it manually.
(1) Copy the `template` file from the `conf` directory to create a configuration file.



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

(2) Configure `goosefs-site.properties` as follows:



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

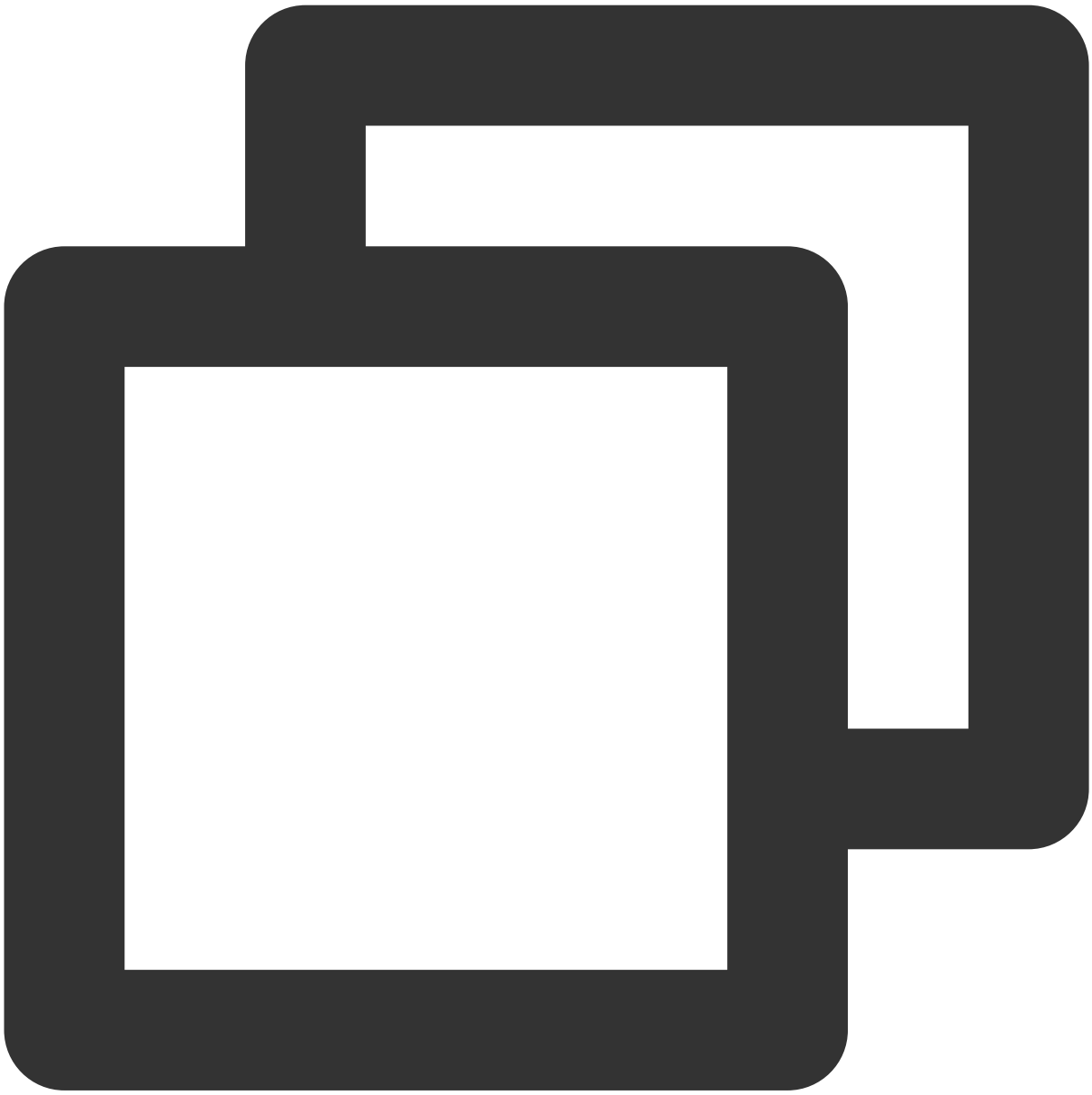
Set `goosefs.master.hostname` to the hostname or IP of the master node, and `goosefs.master.mount.table.root.ufs` to the URI in the under file system (UFS) to which the GooseFS root directory is mounted.

Note:

This URI must be accessible for both the master and worker nodes and therefore it cannot be a local directory.

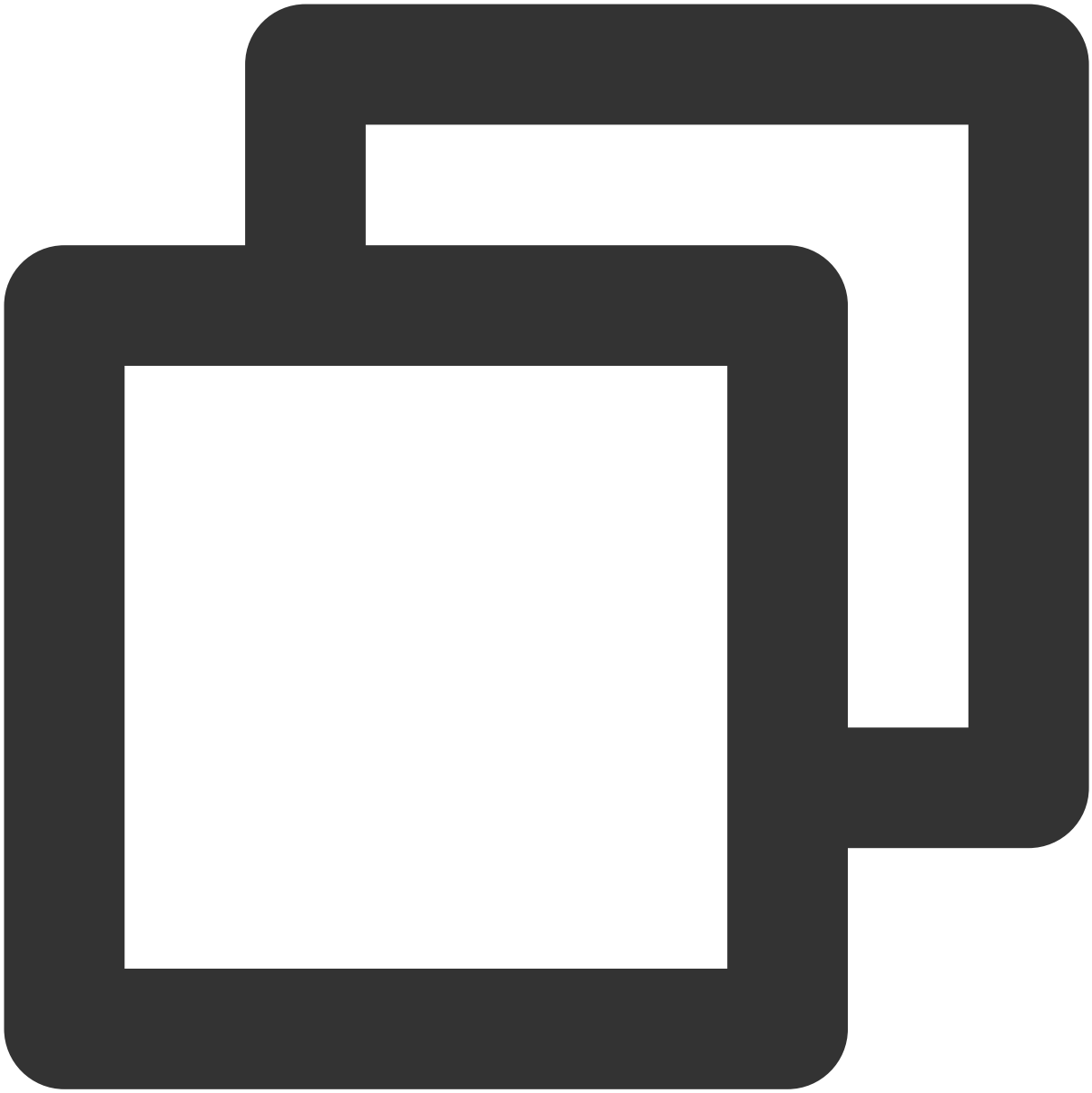
For example, you can mount a COS path to the root directory of GooseFS with `goosefs.master.mount.table.root.ufs=cosn://bucket-1250000000/goosefs/`.

In the `masters` configuration file, specify the hostname or IP of the master node as follows:



```
# 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
```

In the `workers` configuration file, specify the hostname or IP for all worker nodes as follows:



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

After the configuration is completed, run `./bin/goosefs copyDir conf/` to sync the configurations to all nodes.

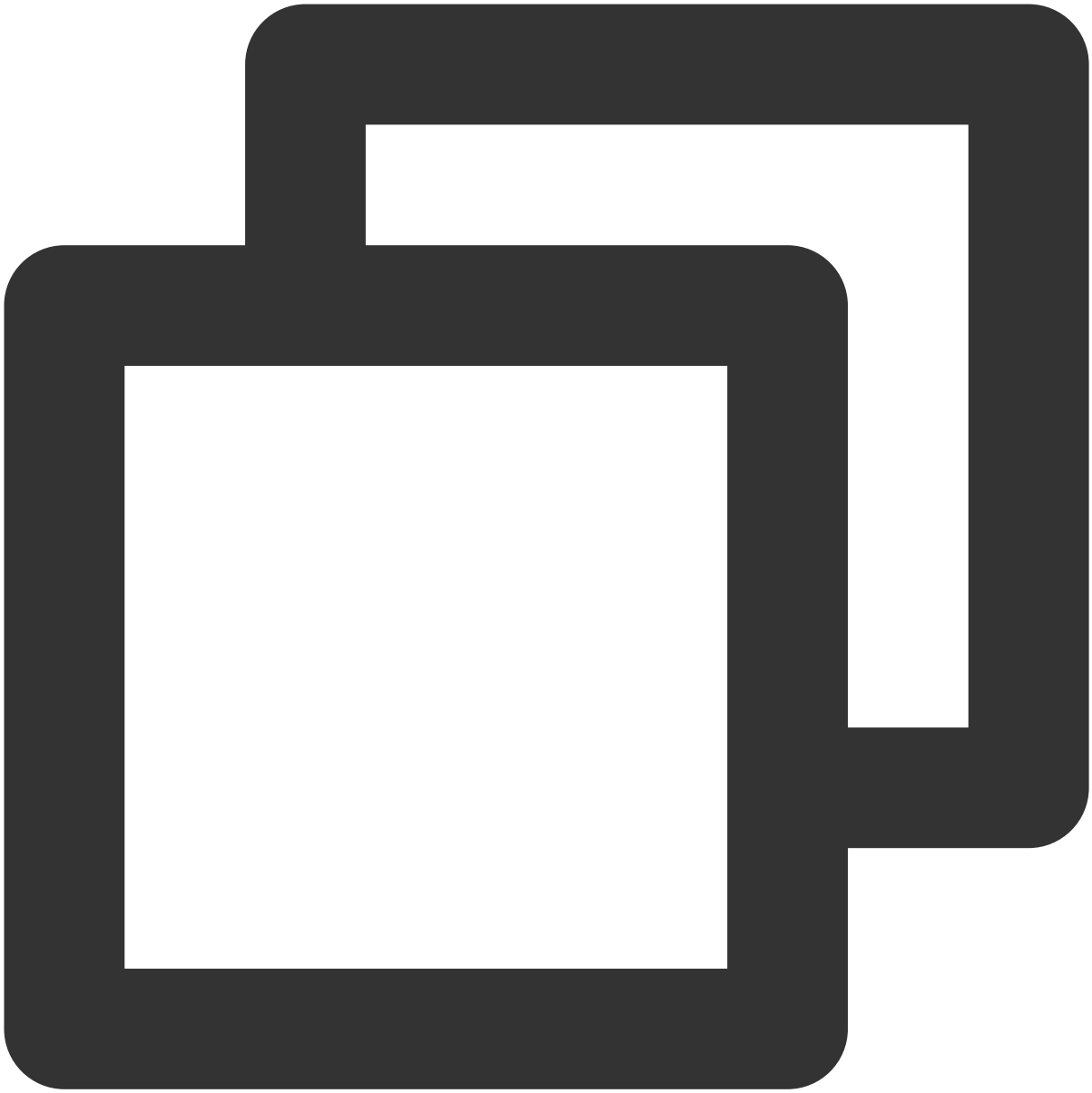
Finally, run `./bin/goosefs-start.sh all` to start the GooseFS cluster.

HA framework deployment

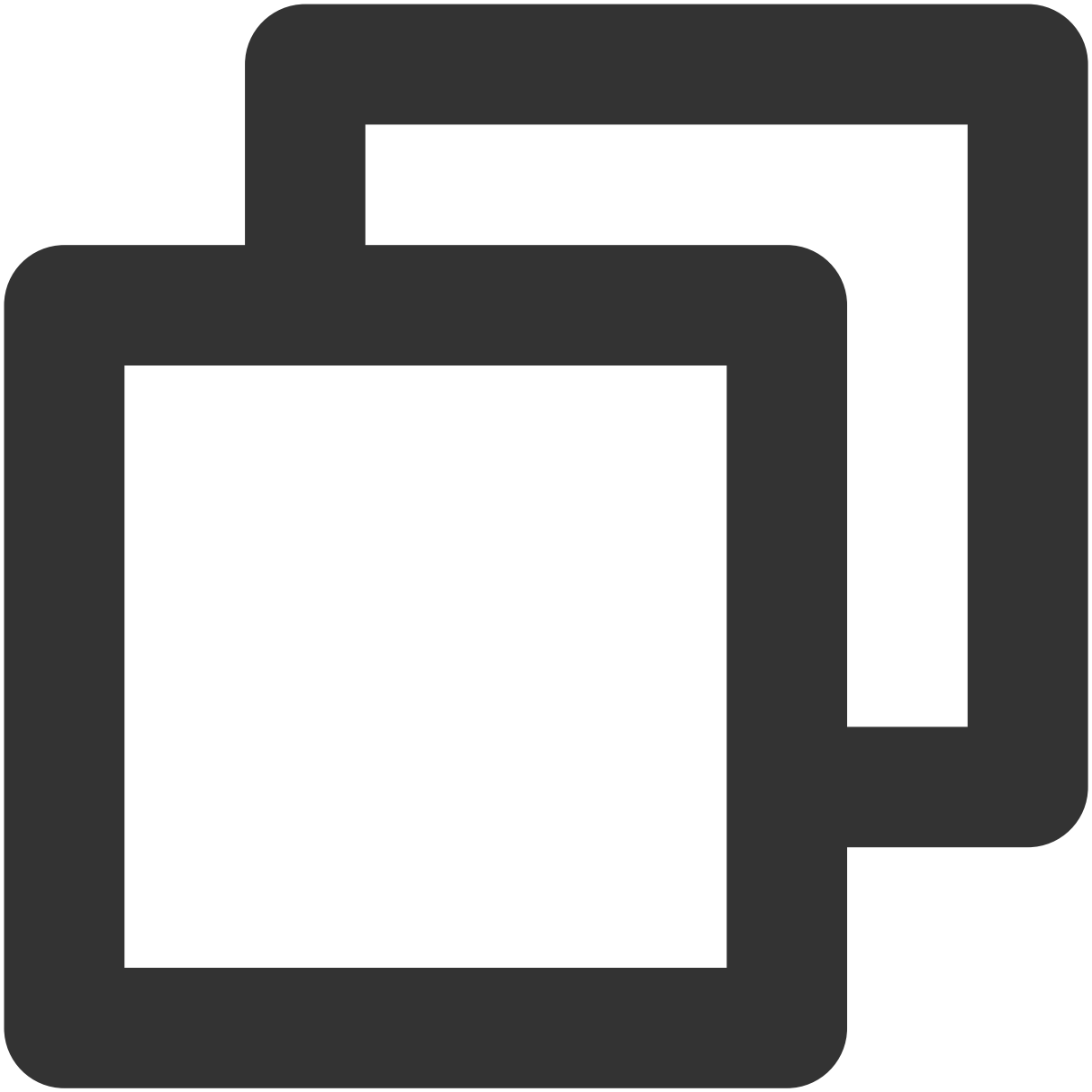
The standalone framework that uses only one master node might lead to a single point of failure (SPOF). Therefore, you are advised to deploy multiple master nodes in the production environment to adopt an HA framework. One of the master nodes will become the leader node that provides services, while other standby nodes will share journals synchronously to maintain the same state as the leader node. If the leader node fails, one of the standby nodes will automatically replace the leader node to continue providing services. In this way, you can avoid SPOFs and make the framework more highly available.
Currently, GooseFS supports using Raft logs or ZooKeeper to ensure the strong consistency of the leader and standby nodes. The deployment of each mode is described below.

HA framework based on Raft embedded logs

First, create a configuration file using a template.



```
$ 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>
```

Note:

The configuration items are described as follows:

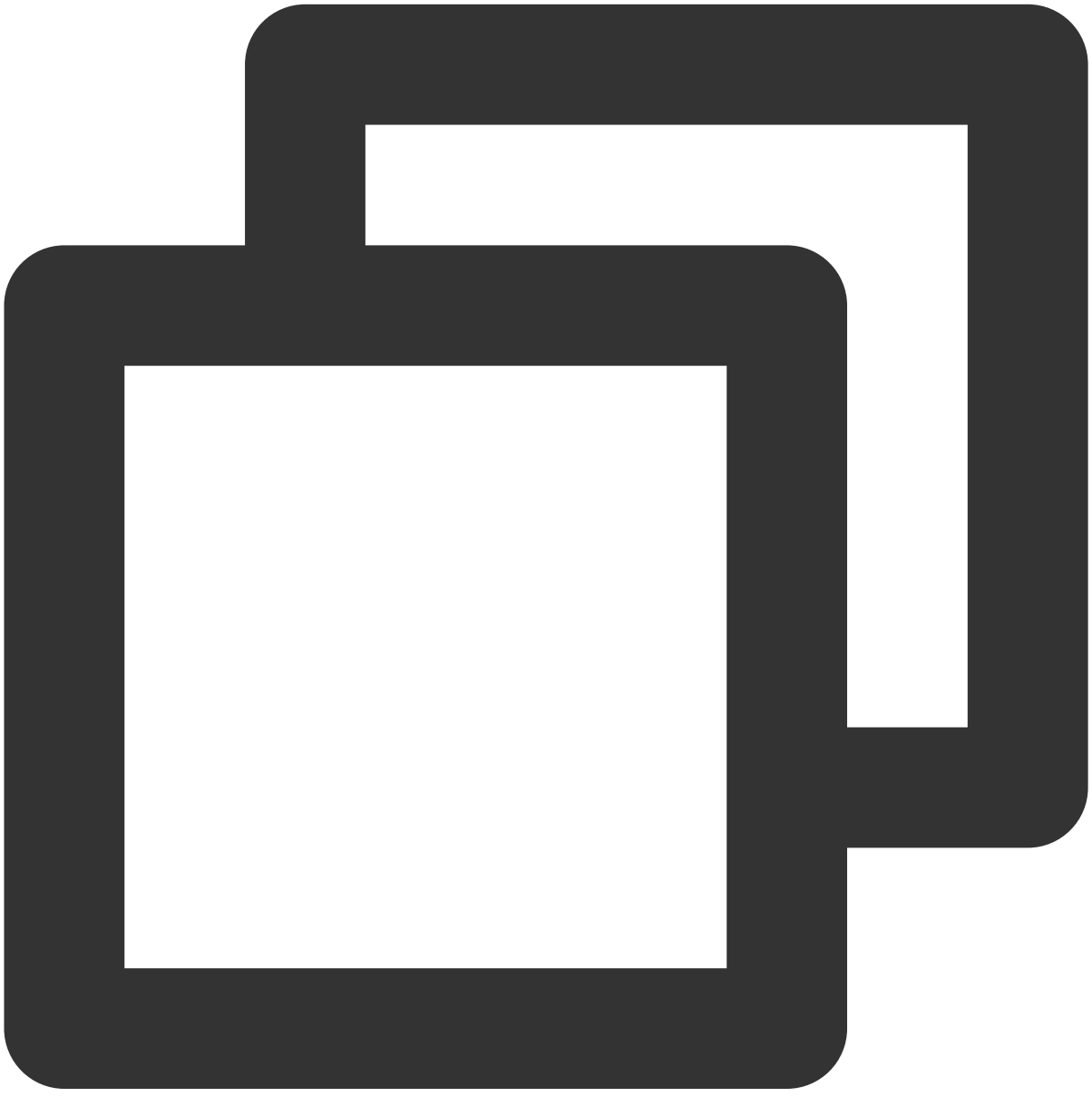
Set `goosefs.master.mount.table.root.ufs` to the underlying storage URI that is mounted to the GooseFS root directory.

Set `goosefs.master.embedded.journal.addresses` to the `ip:embedded_journal_port` or `host:embedded_journal_port` of all standby nodes. The default value of `embedded_journal_port` is

9202. For example, `goosefs.master.embedded.journal.addresses` can be set to 192.168.1.1:9202, 192.168.1.2:9202, and 192.168.1.3:9202.

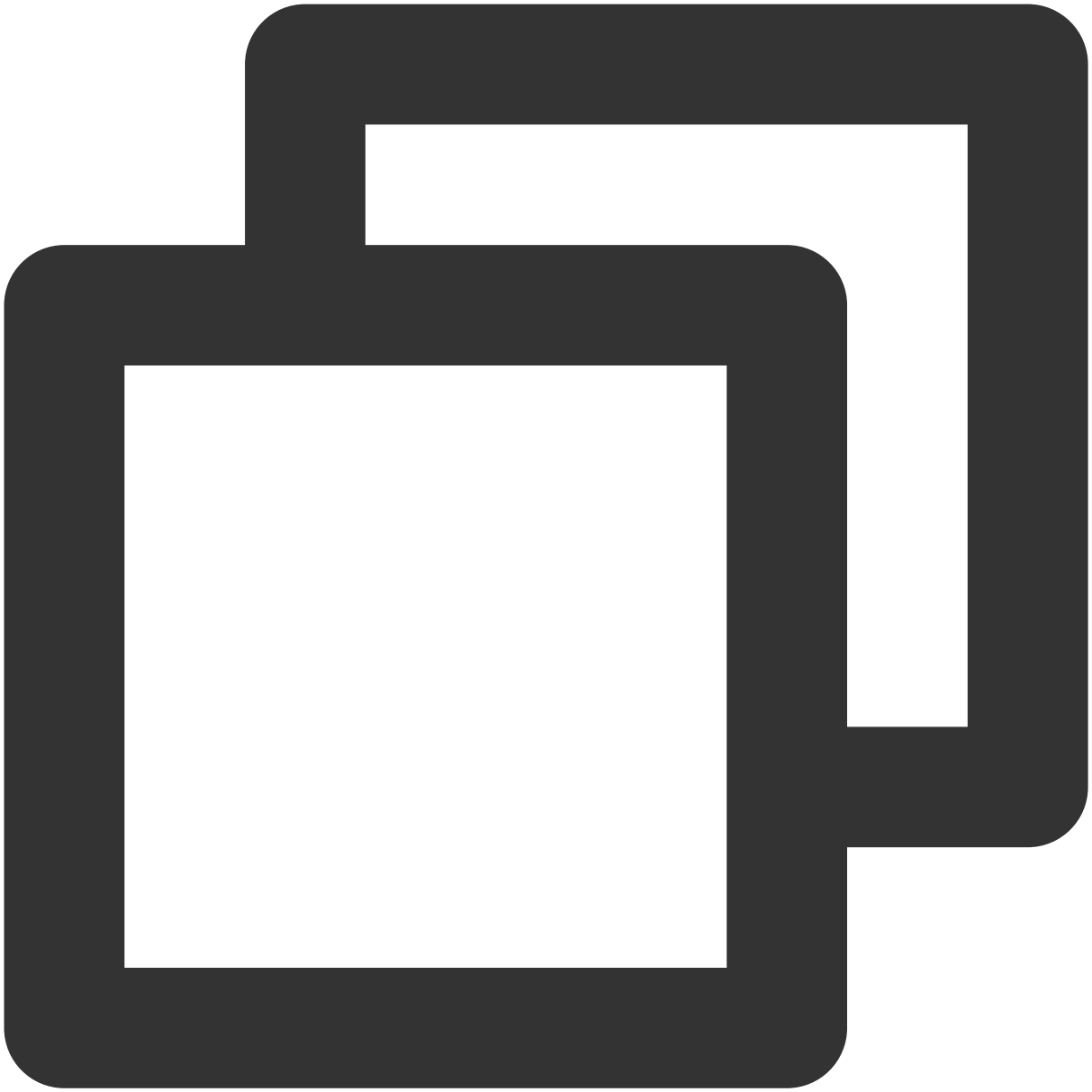
The deployment based on Raft embedded logs uses copycat to select a leader node. Therefore, if you use Raft for an HA framework, do not mix it with ZooKeeper.

After the configurations are completed, run the following command to sync all configurations:

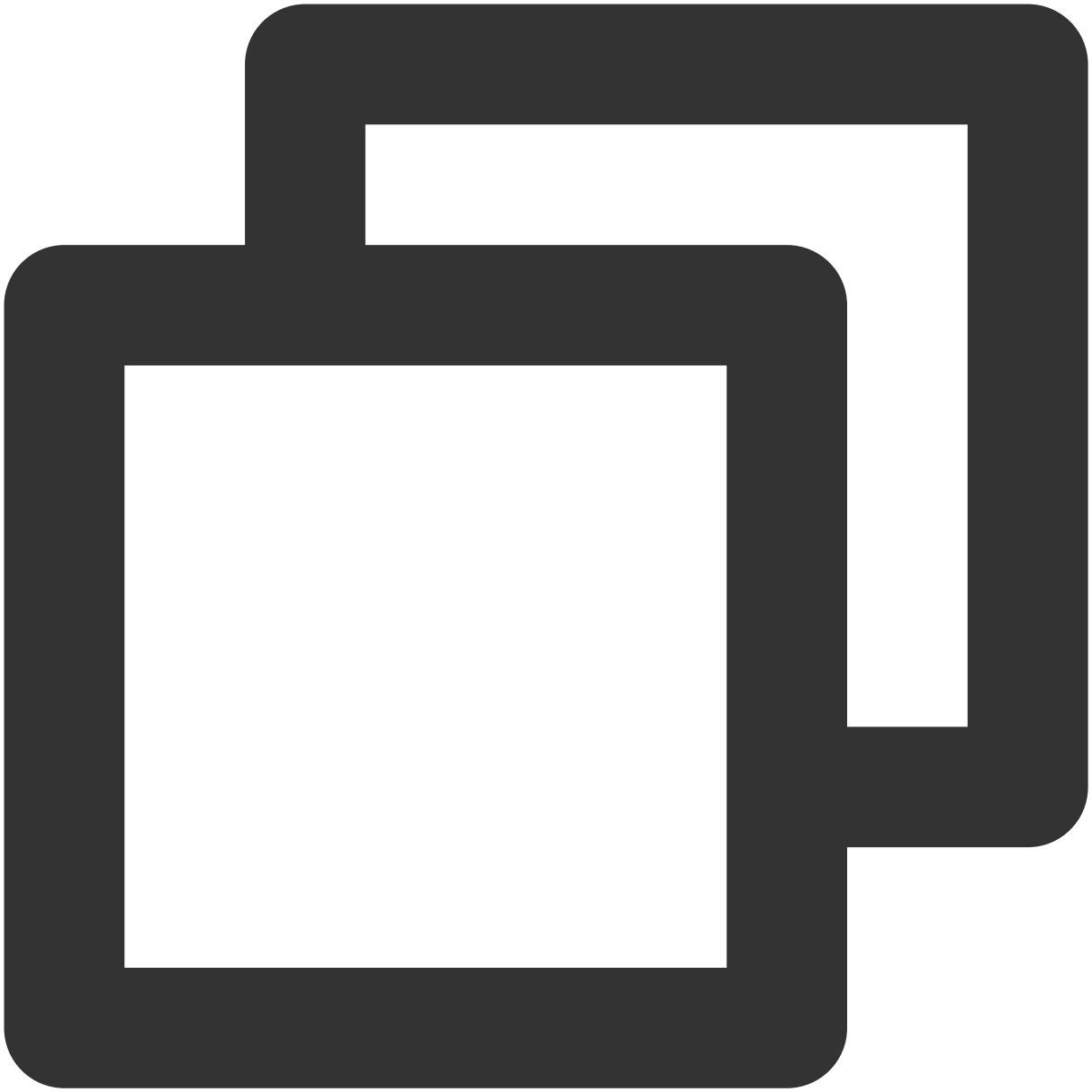


```
$ ./bin/goosefs copyDir conf/
```

Format and then start the GooseFS cluster:

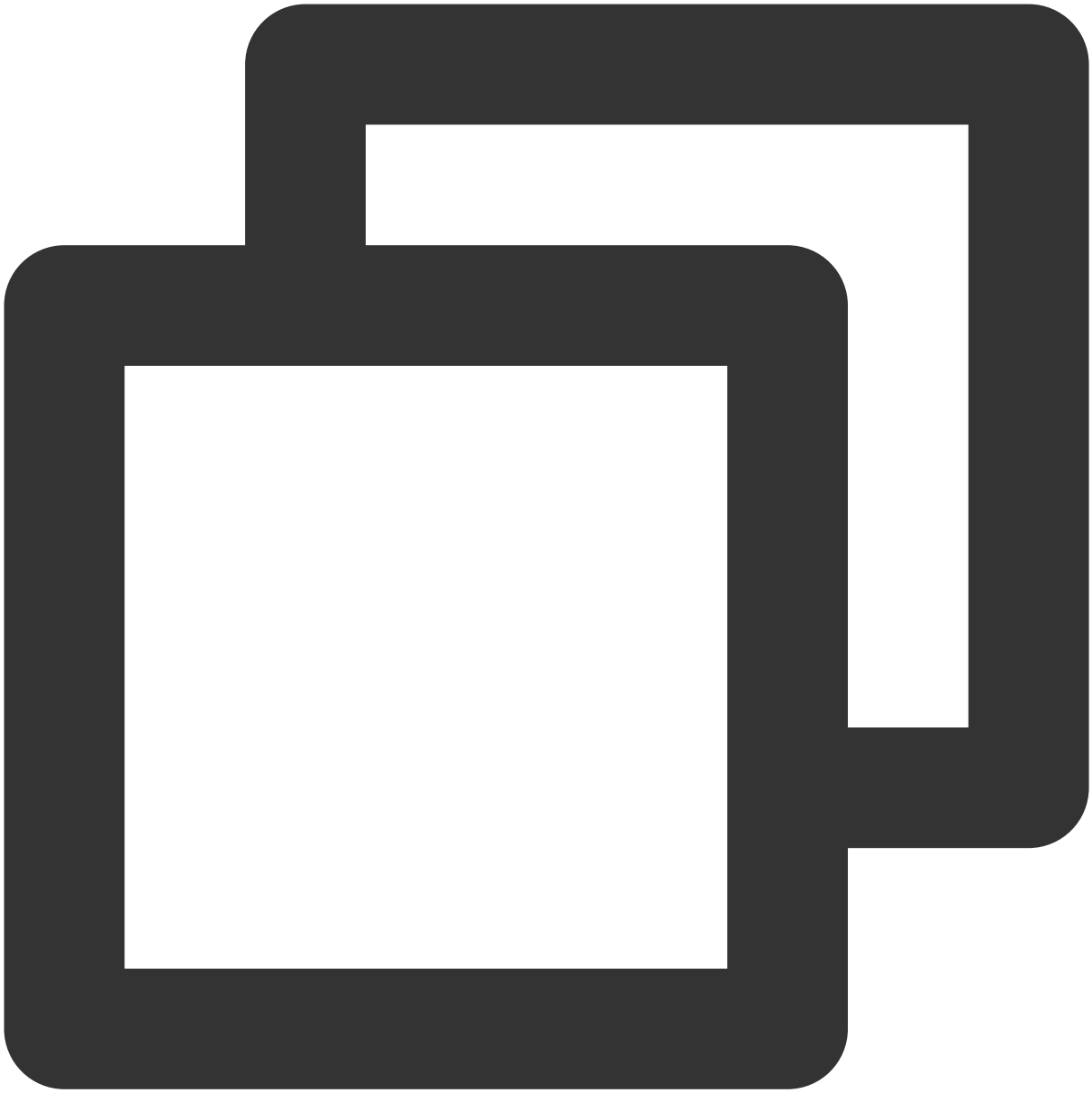


```
$ ./bin/goosefs format
```



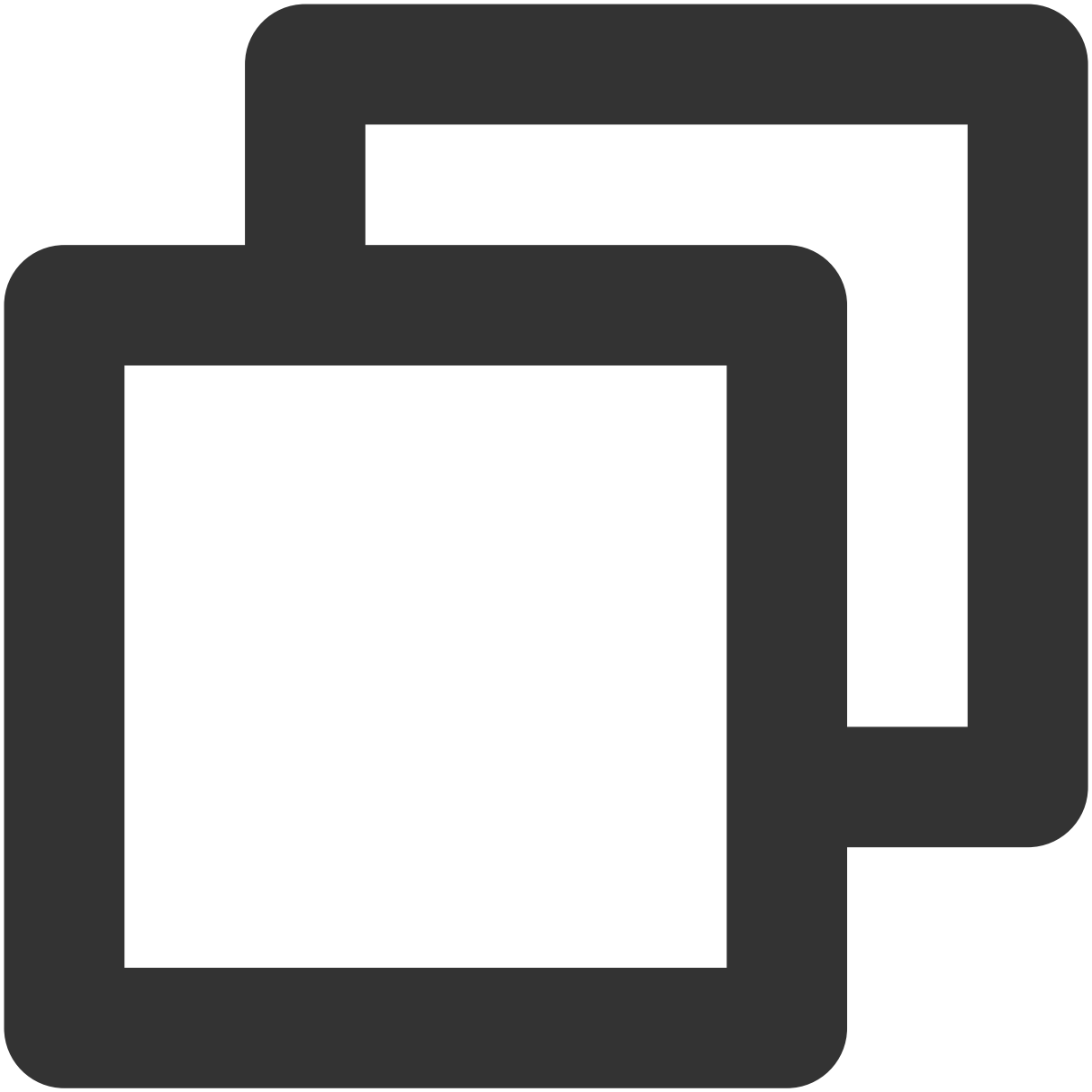
```
$ ./bin/goosefs-start.sh all
```

If the system prompts that you don't have the permission, restart as follows:



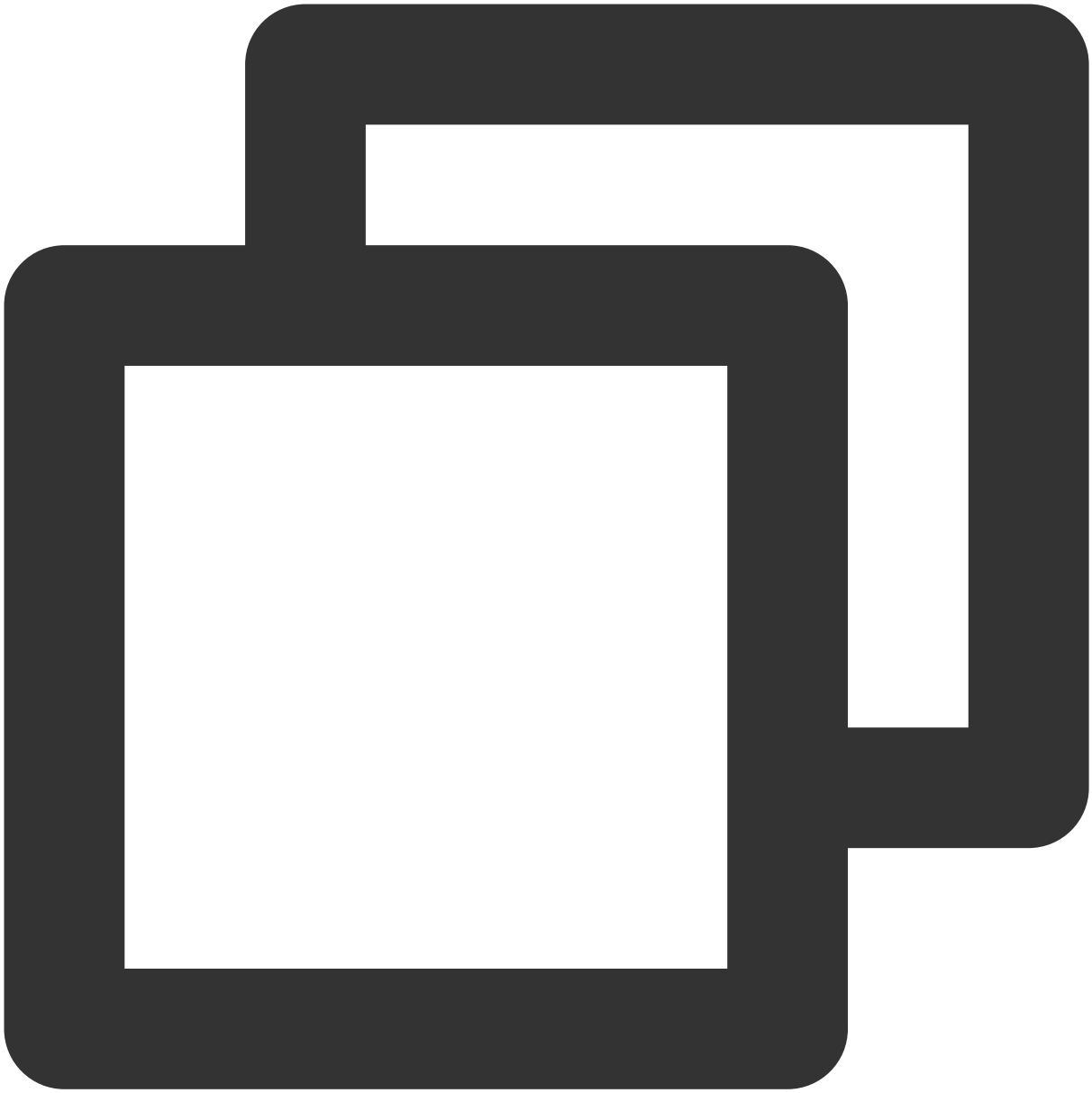
```
$ ./bin/goosefs-stop.sh all
$ ./bin/goosefs-start.sh all SudoMount
```

Run the following command to view the current leader node:



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

Run the following command to view the cluster status:



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

Deployment based on ZooKeeper and shared logs

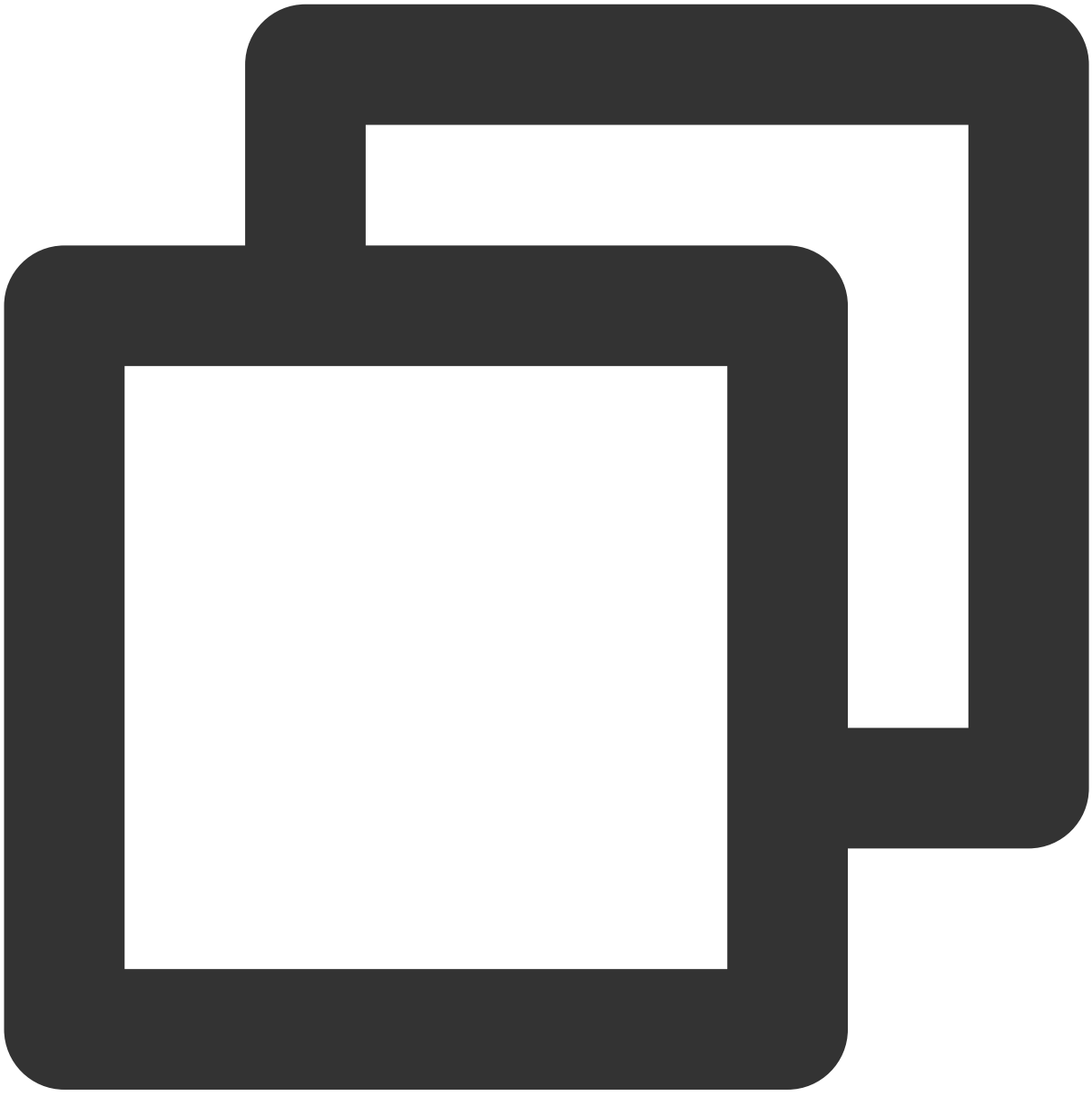
To set up an HA GooseFS framework based on ZooKeeper, you need to:

Have a ZooKeeper cluster. GooseFS uses ZooKeeper to select the leader node, and the client and worker nodes use ZooKeeper to query the leader node.

Have a highly available and strongly consistent shared file system, which must be accessible for all GooseFS master nodes. The leader will write the logs to the file system, and standby nodes will read logs from the file system constantly

and replay logs to ensure state consistency. You are advised to use HDFS or COS (for example, `hdfs://10.0.0.1:9000/goosefs/journal` or `cosn://bucket-1250000000/journal`) as the shared file system.

The configurations are as follows:



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

Note that `JOURNAL_URI` in ZK mode must be a URI of the shared storage system. Then, use `./bin/goosefs copyDir conf/` to sync the configurations to all nodes in the cluster, and use `./bin/goosefs-start.sh all` to start the cluster.

GooseFS Process List

After the `goosefs-start.sh all` script is executed and GooseFS is started, the following processes will run in the cluster:

Process	Description
GooseFSMaster	Default RPC port: 9200; web port: 9201
GooseFSWorker	Default RPC port: 9203; web port: 9204
GooseFSJobMaster	Default RPC port: 9205; web port: 9206
GooseFSProxy	Default web port: 9211
GooseFSJobWorker	Default RPC port: 9208; web port: 9210

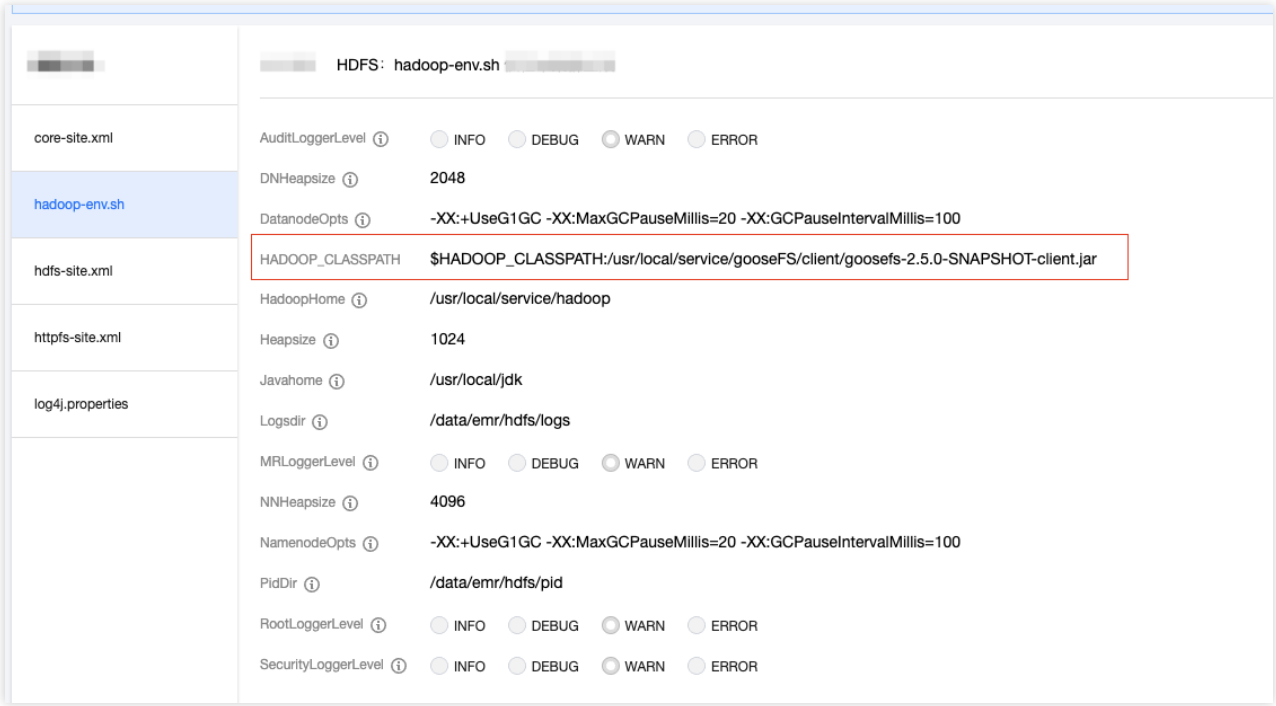
Deploying with Tencent Cloud EMR

Last updated：2024-03-25 16:04:01

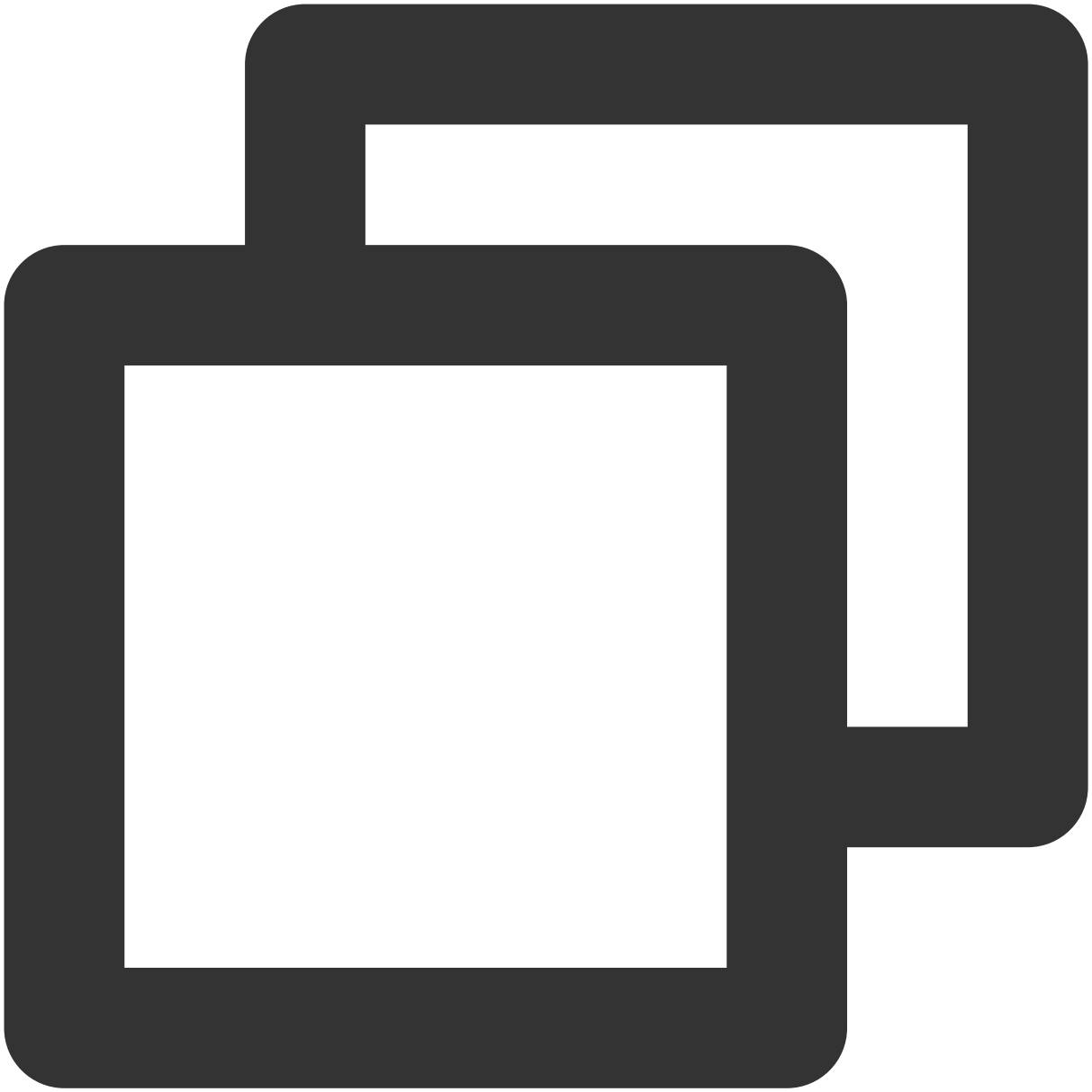
The GooseFS-integrated Tencent Cloud EMR will be released in the latest version, which allows you to use GooseFS as an EMR component without having to deploy it for the EMR environment separately.
For Tencent Cloud EMR clusters that have not integrated with GooseFS, you can refer to this document to deploy the EMR environment for GooseFS.
First, select a framework that suits the production environment by referring to Cluster Deployment and Running and then deploy the cluster.
After that, configure according to the GooseFS-supportive EMR component. This document describes the configuration for Hadoop MapReduce, Spark, and Flink supports.

Hadoop MapReduce Support

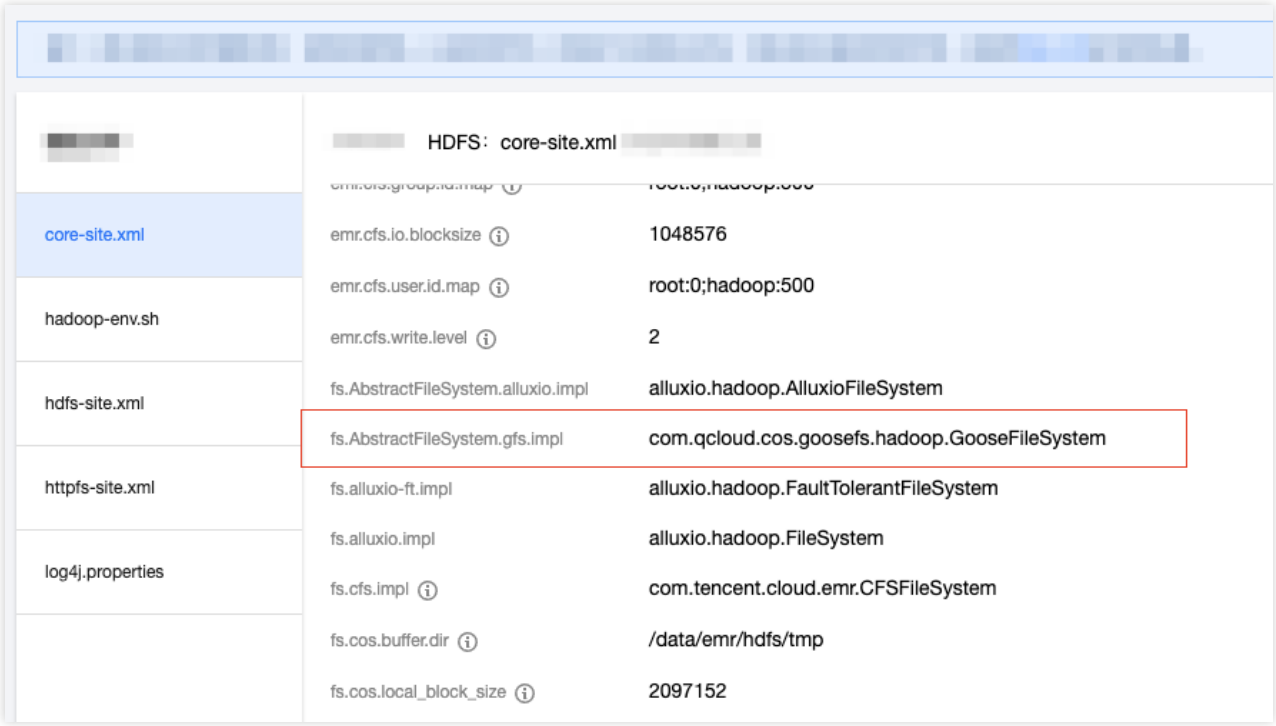
To allow Hadoop MapReduce jobs to read and write GooseFS data, add the GooseFS client's dependent path to `HADOOP_CLASSPATH` in `hadoop-env.sh`. This can be performed in the EMR console as follows:



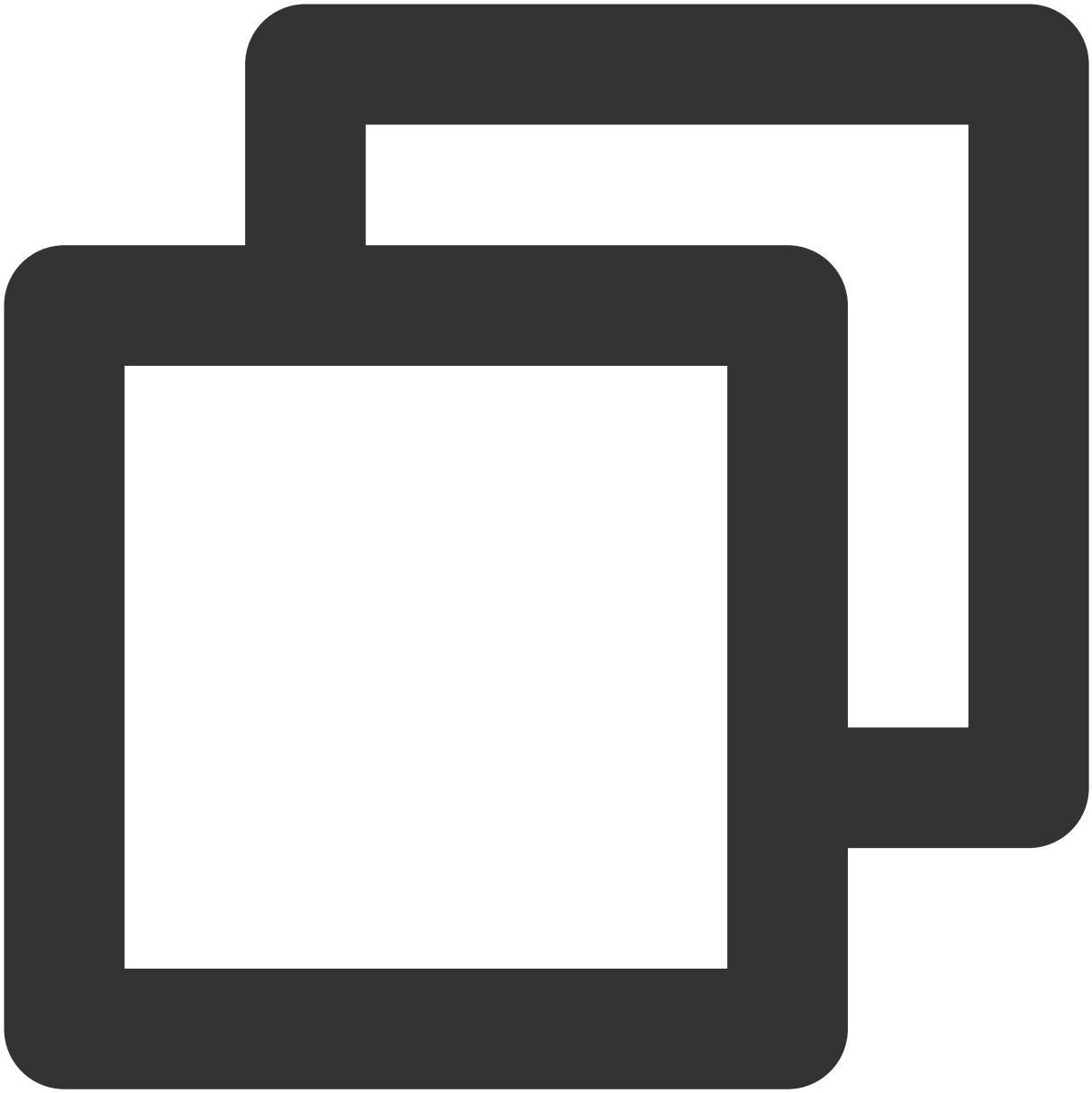
Then, configure the HCFS implementation of GooseFS in `core-site.xml` in the EMR console.
Configure `fs.AbstractFileSystem.gfs.impl` as follows:



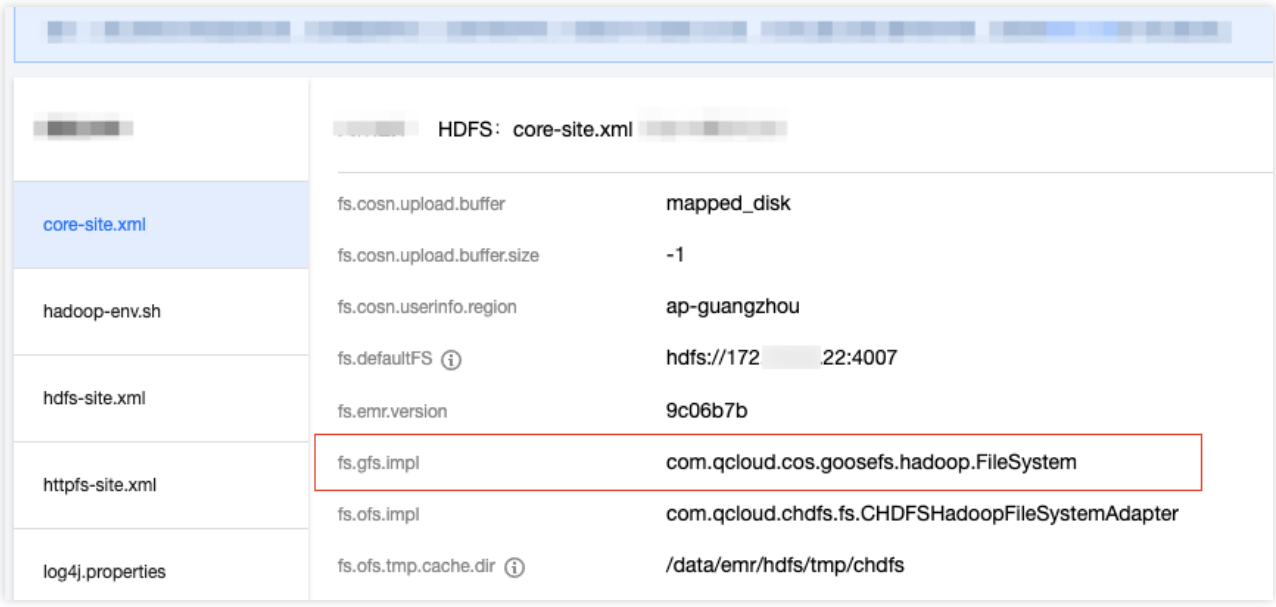
```
com.qcloud.cos.goosefs.hadoop.GooseFileSystem
```



Configure `fs.gfs.impl` as follows:



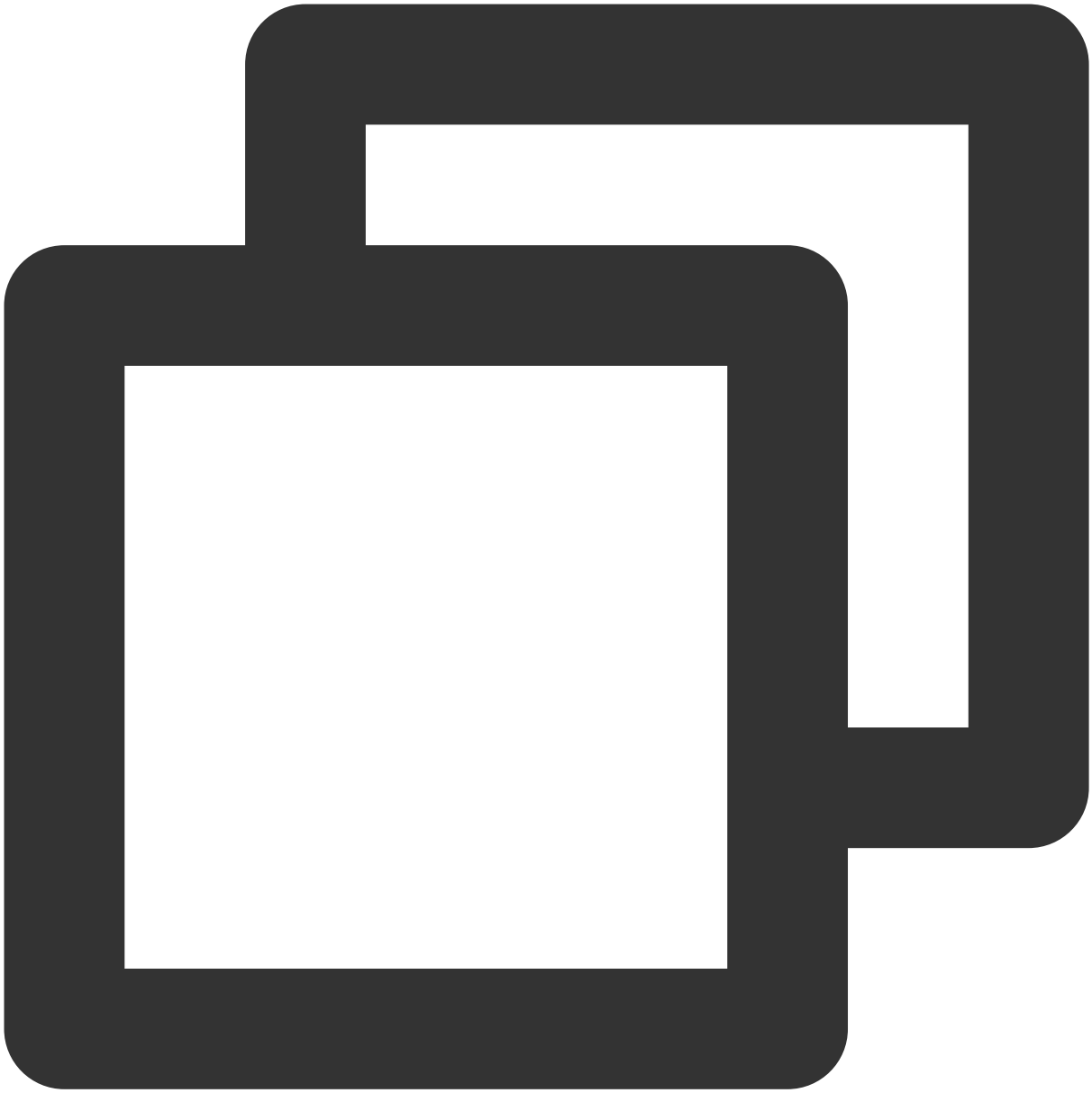
```
com.qcloud.cos.goosefs.hadoop.FileSystem
```



After the configurations are delivered, you can restart YARN-related components for the configurations to take effect.

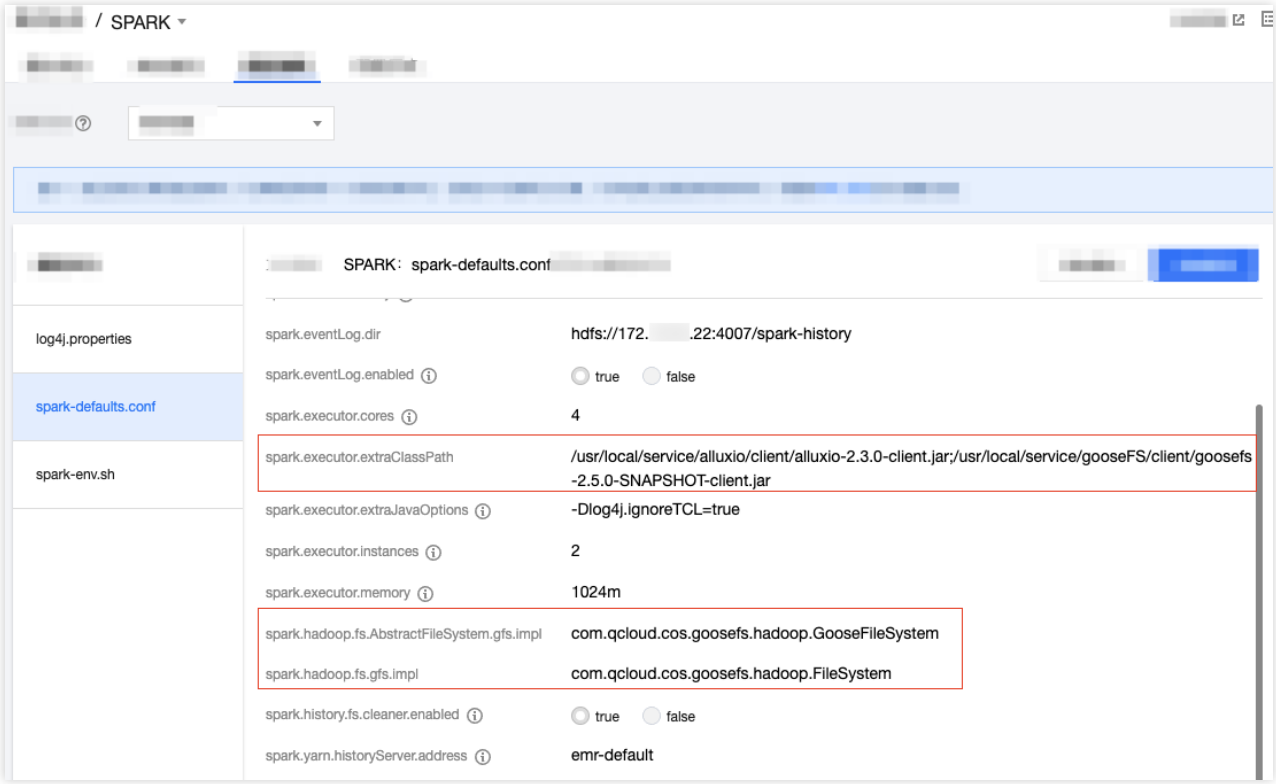
Spark Support

To allow Spark to access GooseFS, you also need to set `spark.executor.extraClassPath` to the GooseFS client's dependency package in the `spark-defaults.conf` file as follows:



```
...
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
...
```

This configuration can also be done and delivered on the Spark component configuration page in the EMR console



Flink Support

EMR's Flink adopts the Flink on YARN deployment mode. Therefore, you need to ensure that `fs.hdfs.hadoopconf` is set to the Hadoop configuration path in `${FLINK_HOME}/flink-conf.yaml`. For Tencent Cloud EMR clusters, the value is `/usr/local/service/hadoop/etc/hadoop` in most cases.
You don't need to configure other configuration items. You can use Flink on YARN to submit the Flink jobs, which can access GooseFS through `gfs://master:port/<path>`.

Note:

If Flink needs to access GooseFS, the master and port must be specified.

Hive, Impala, HBase, Sqoop, and Oozie Support

If the environment support for Hadoop MapReduce has been configured, components such as Hive, Impala, and HBase can be used directly without requiring separate configuration.

Deploying with Tencent Cloud TKE

Last updated：2024-03-25 16:04:01

To deploy GooseFS using TKE, the open source Fluid component is required, which is available in the TKE application market. You can deploy as follows:
1. Use Fluid helm chart to deploy controllers.
2. Use kubectl to create the dataset and GooseFS runtime.

Preparations

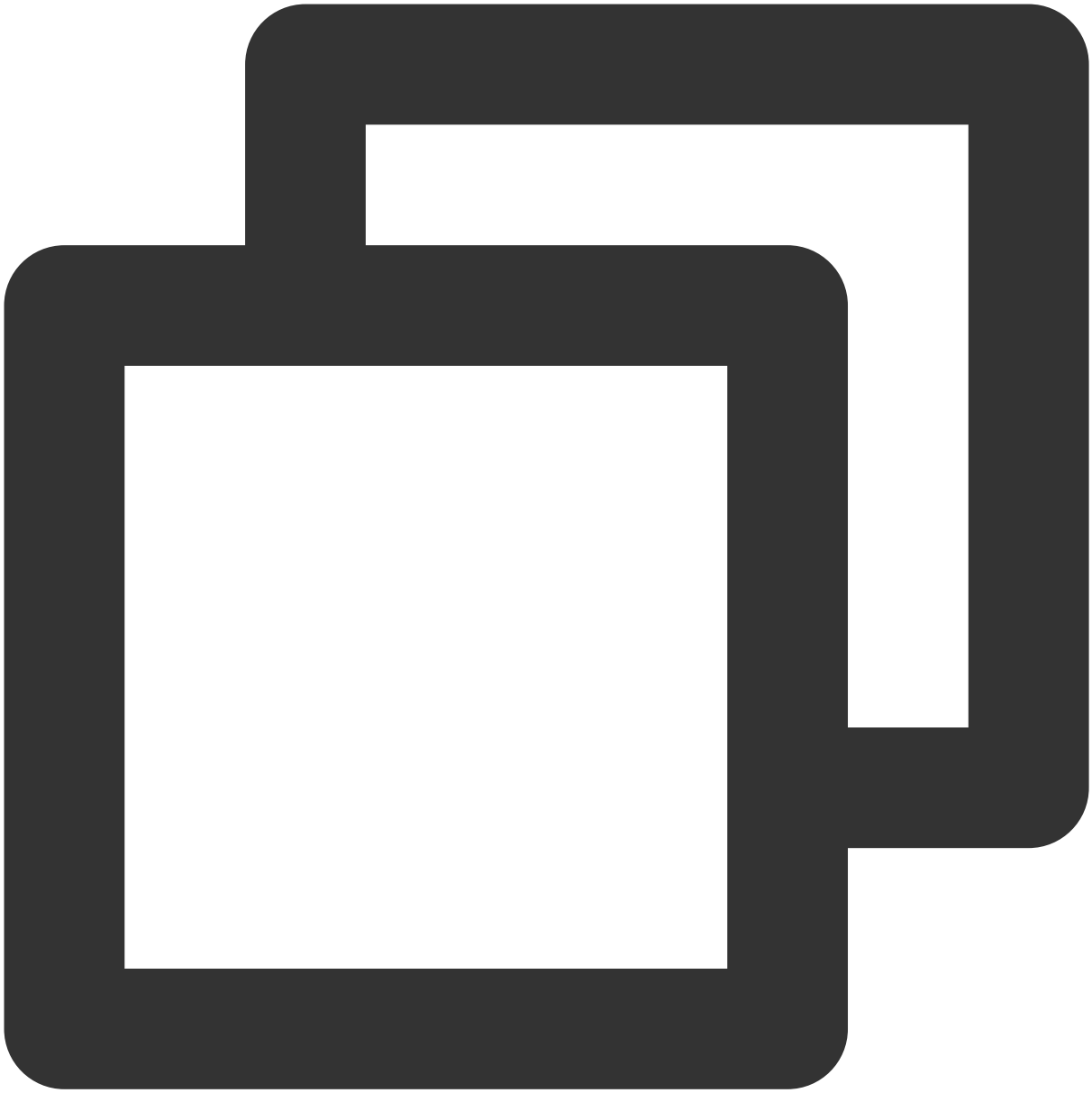
1. Have a Tencent Cloud TKE cluster.
2. Have installed kubectl of v1.18 or a later version.

Installation

1. Find Fluid from the TKE Application Market.
2. Install Fluid controllers.
3. Check the controllers. You can click **Cluster** in the left sidebar and find the desired cluster. If there are two controllers, Fluid has been installed successfully.

Operation Example

1. Authorize cluster access



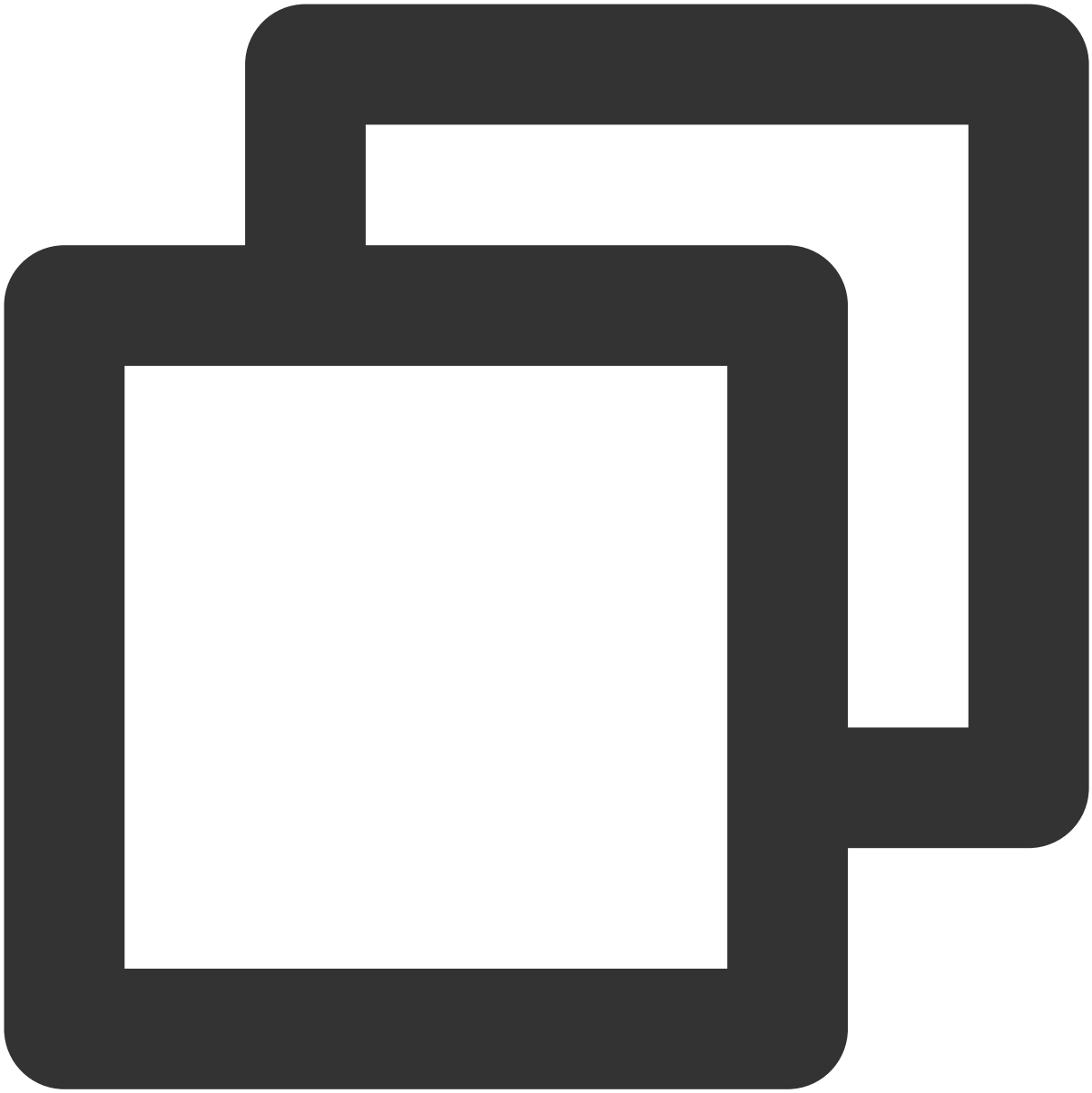
```
[root@master01 run]# export KUBECONFIG=xxx/cls-xxx-config (Download the cluster cre
```

Note:

You need to grant the cluster's API Server access to the public network.

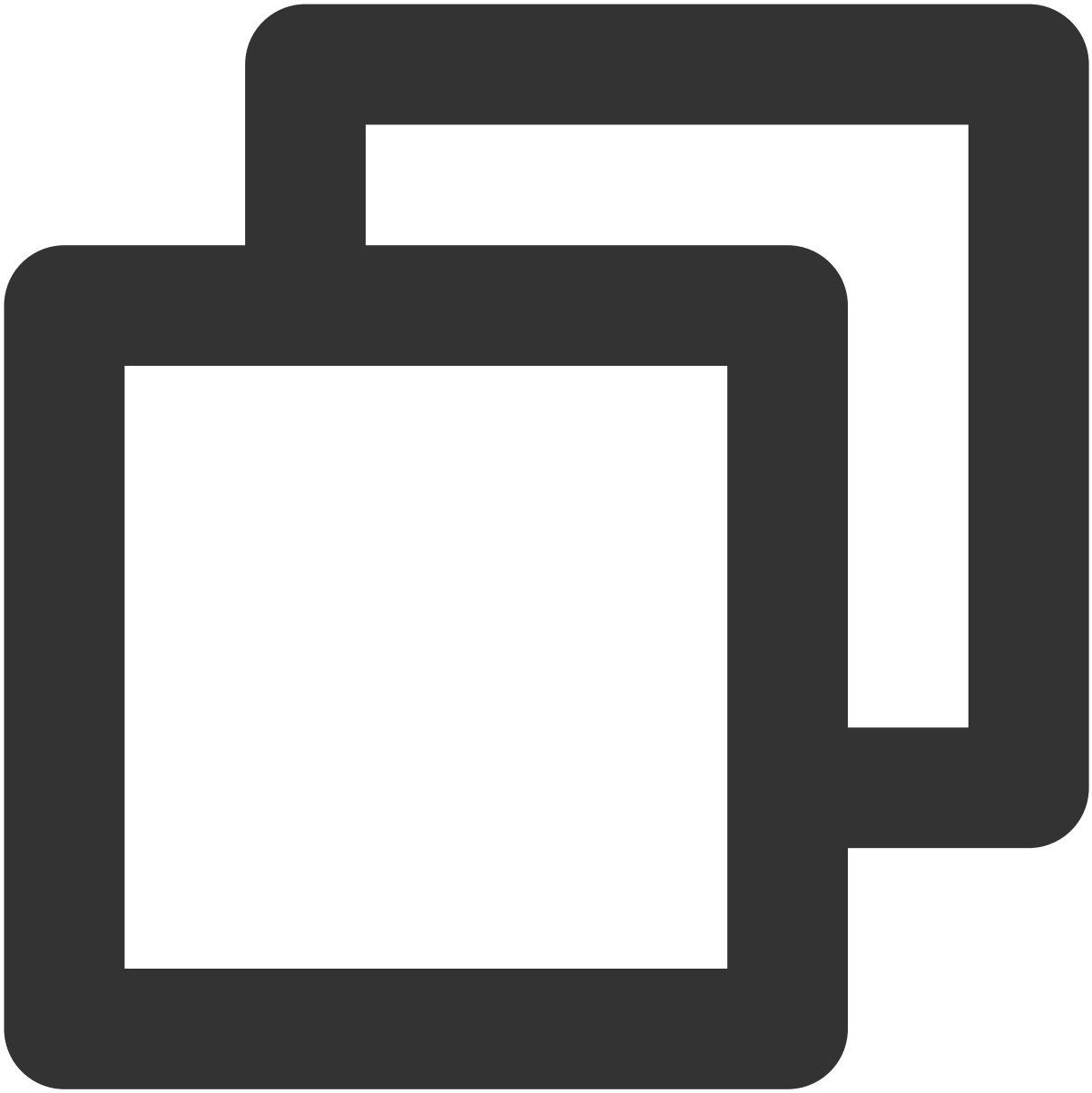
2. Create a UFS dataset (using COS as an example)

Create `secret.yaml` for encryption. The template is as follows:



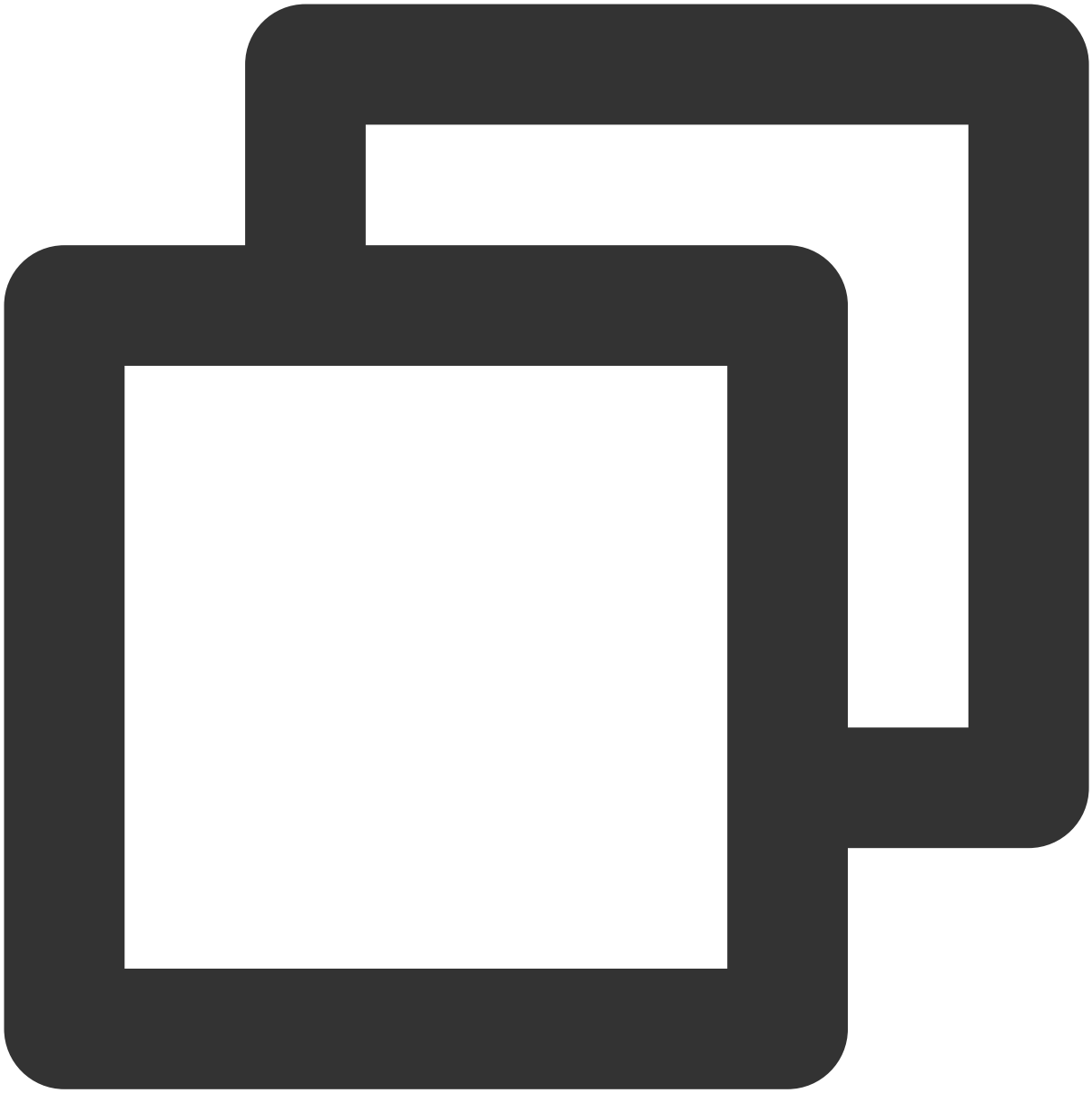
```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
stringData:
  fs.cosn.userinfo.secretKey: xxx
  fs.cosn.userinfo.secretId:xxx
```

Create `secret` :



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

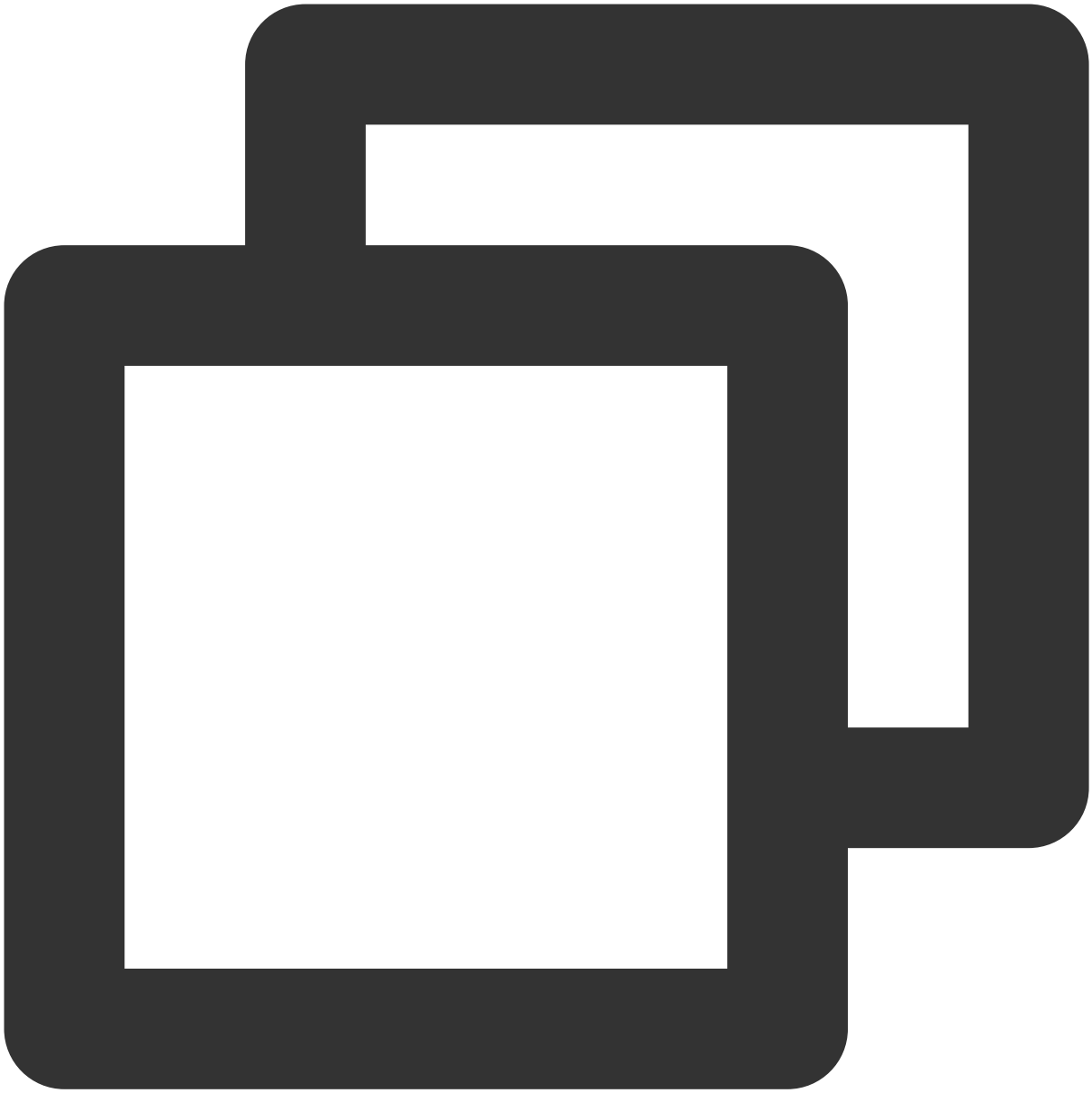
`dataset.yaml` template:



```
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
```

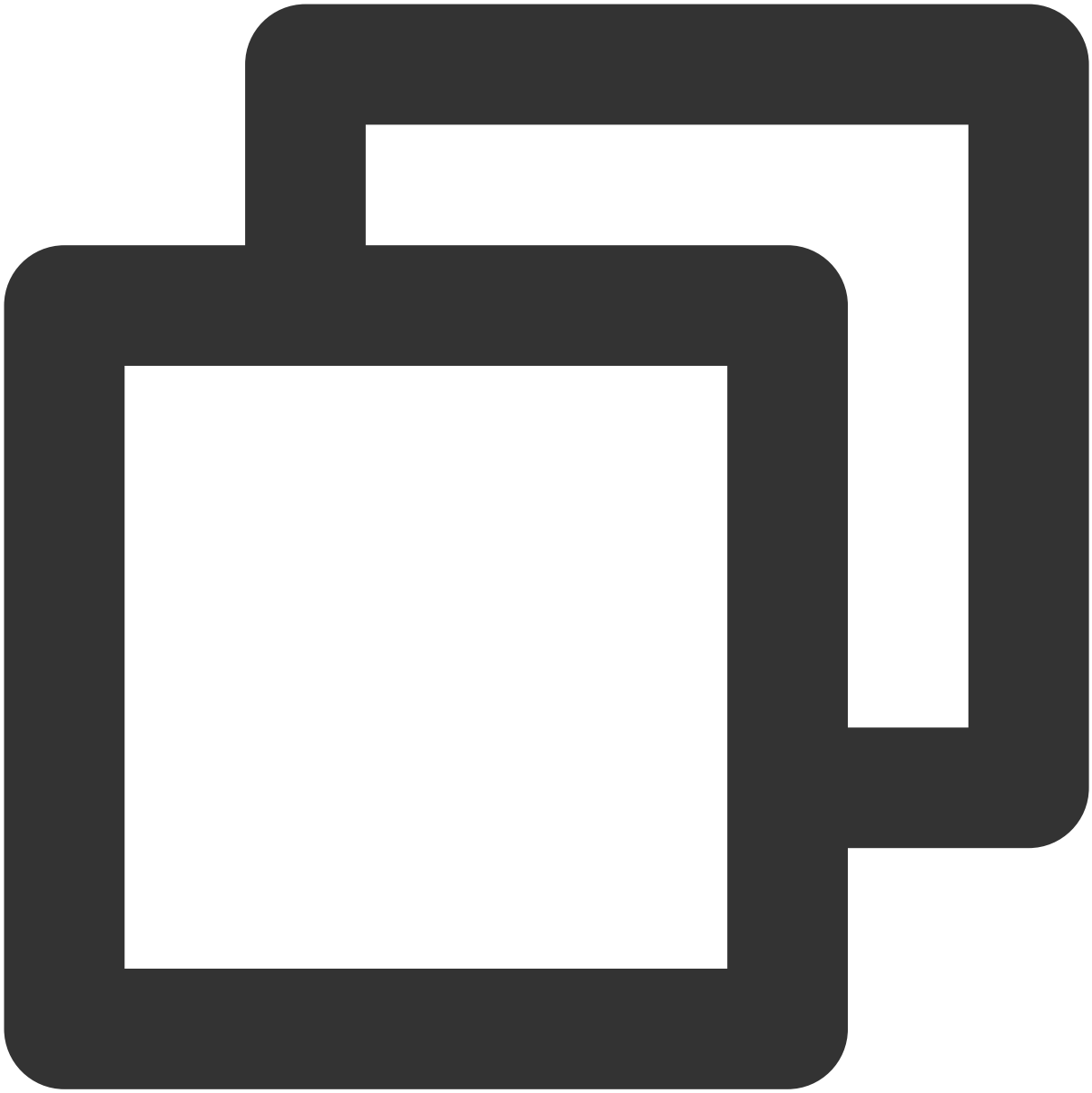
```
    fs.AbstractFileSystem.cosn.impl: org.apache.hadoop.fs.CosN
    fs.cosn.userinfo.appid: "${your appid}"
  encryptOptions:
    - name: fs.cosn.userinfo.secretKey
      valueFrom:
        secretKeyRef:
          name: mysecret
          key: fs.cosn.userinfo.secretKey
    - name: fs.cosn.userinfo.secretId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

Create a dataset:



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

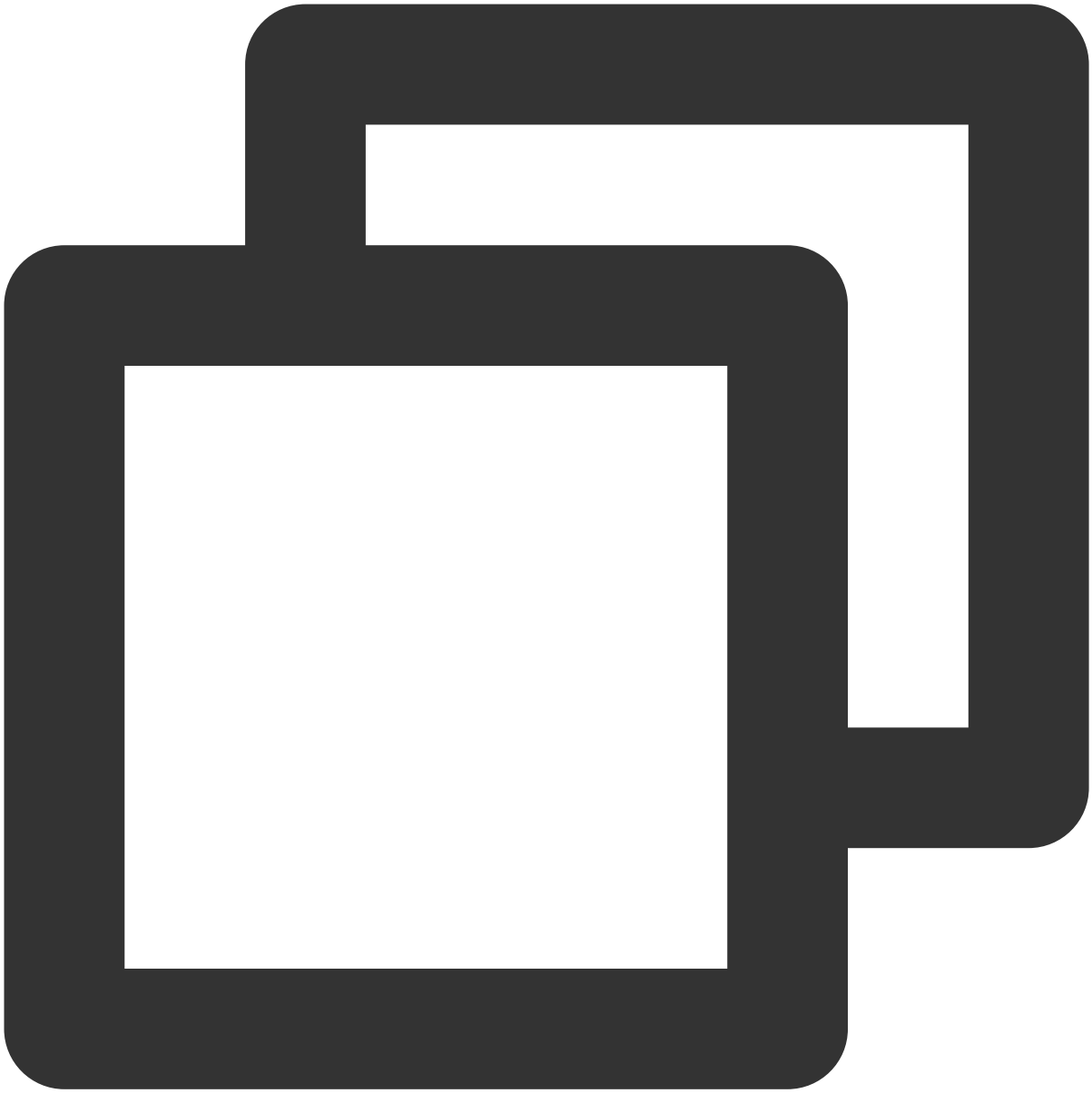
Query the dataset status (NotBond):



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

3. Create runtime

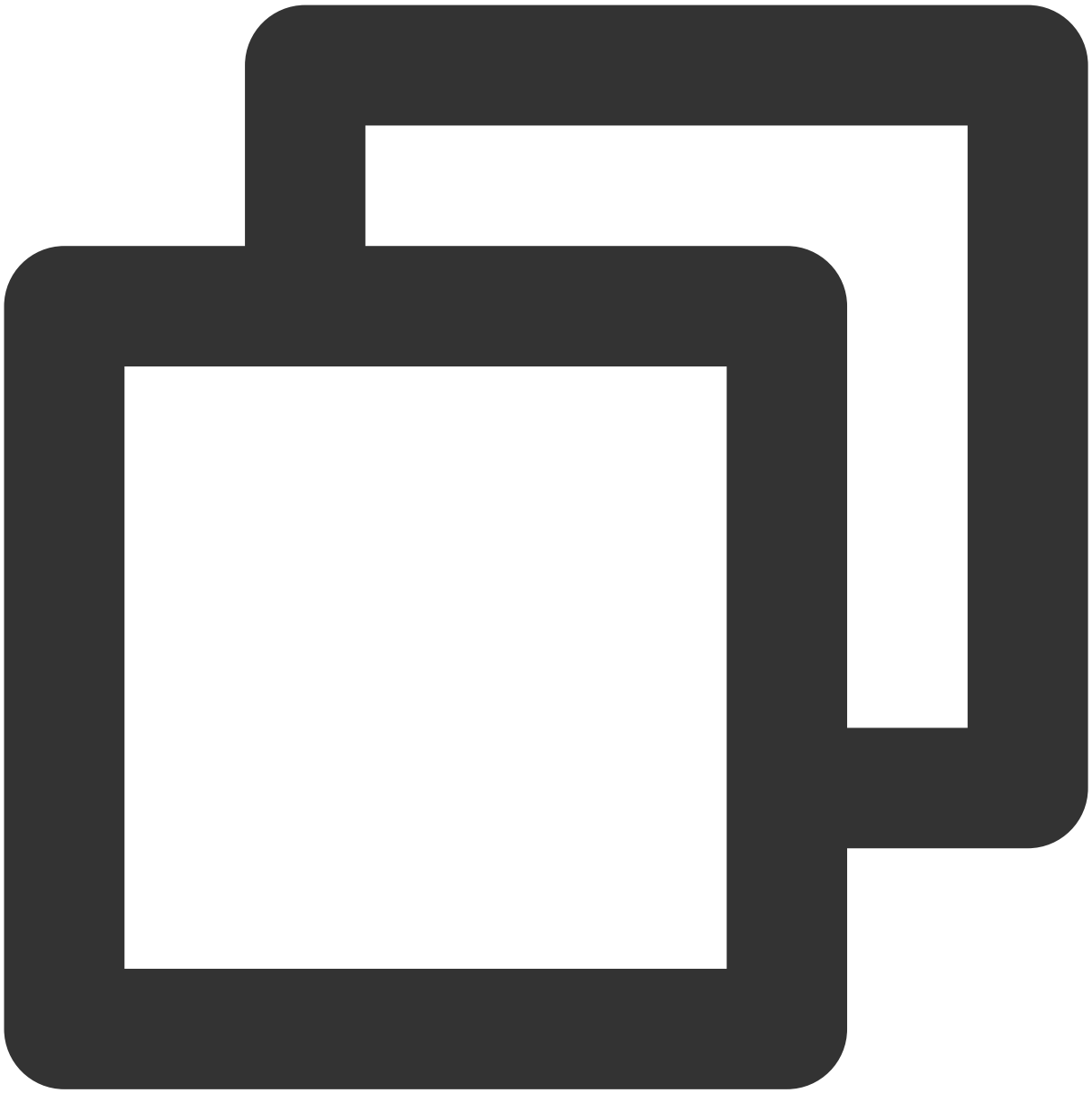
`untime.yaml` template:



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

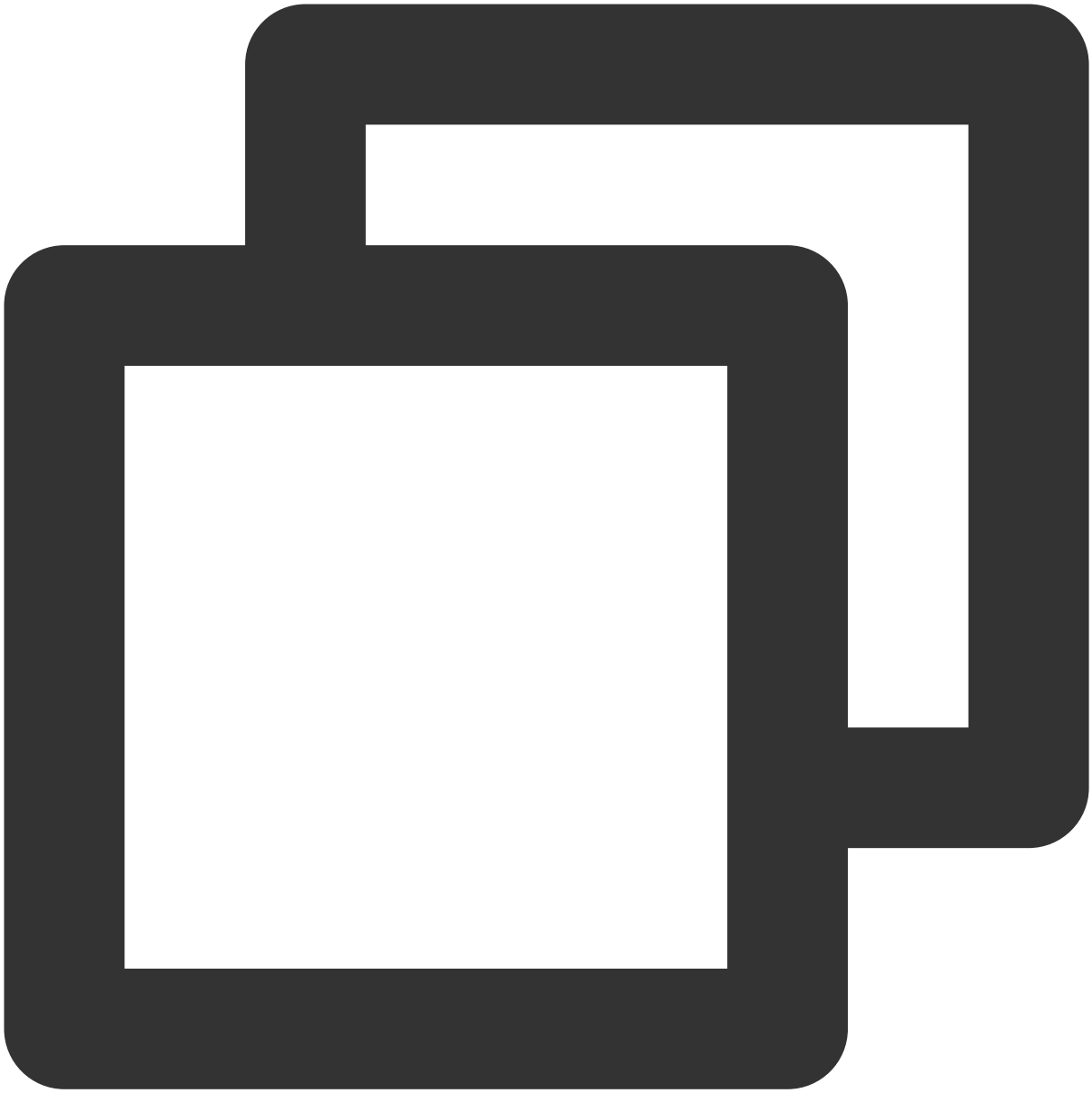
```
    imageTag: ${tag}
  tieredstore:
    levels:
      - mediumtype: MEM
        path: /dev/shm
        quota: 1Gi
        high: "0.95"
        low: "0.7"
  properties:
    # goosefs user
    goosefs.user.file.writetype.default: MUST_CACHE
  master:
    replicas: 1
    journal:
      volumeType: hostpath
    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
```

Create runtime:



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

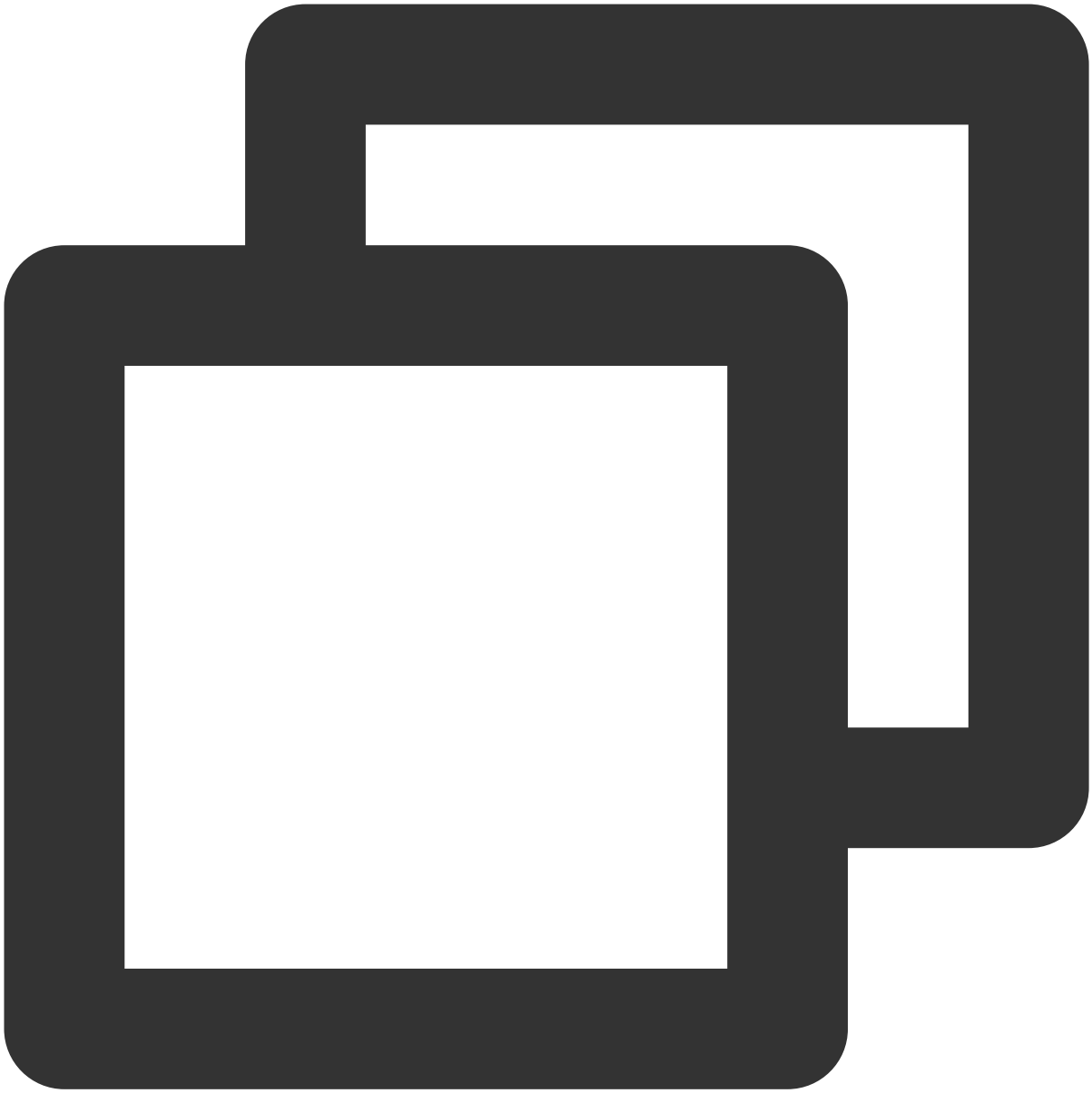
Check the GooseFS component status:



```
[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
```

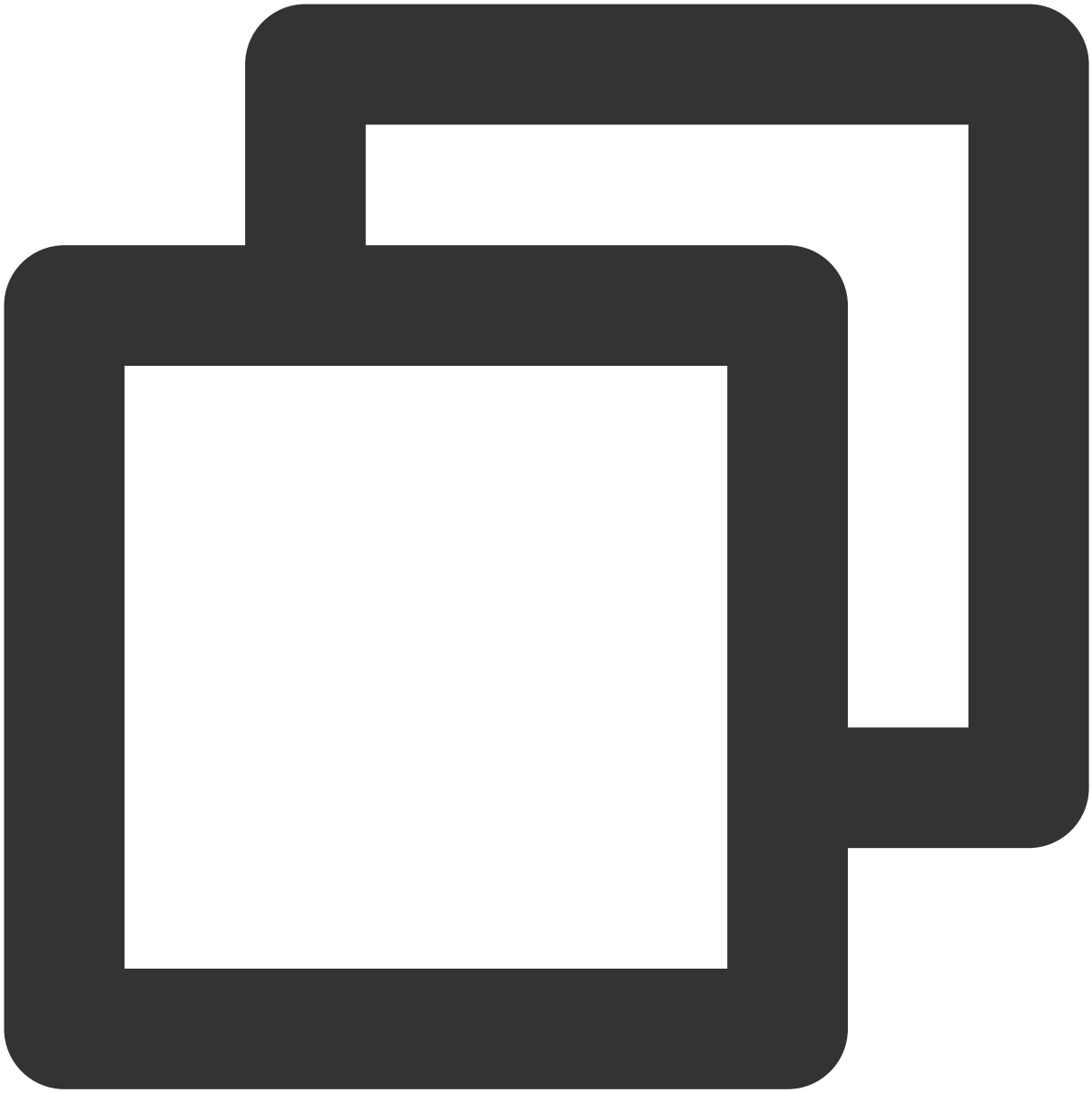
4. Load data

`dataload.yaml` data loading component:



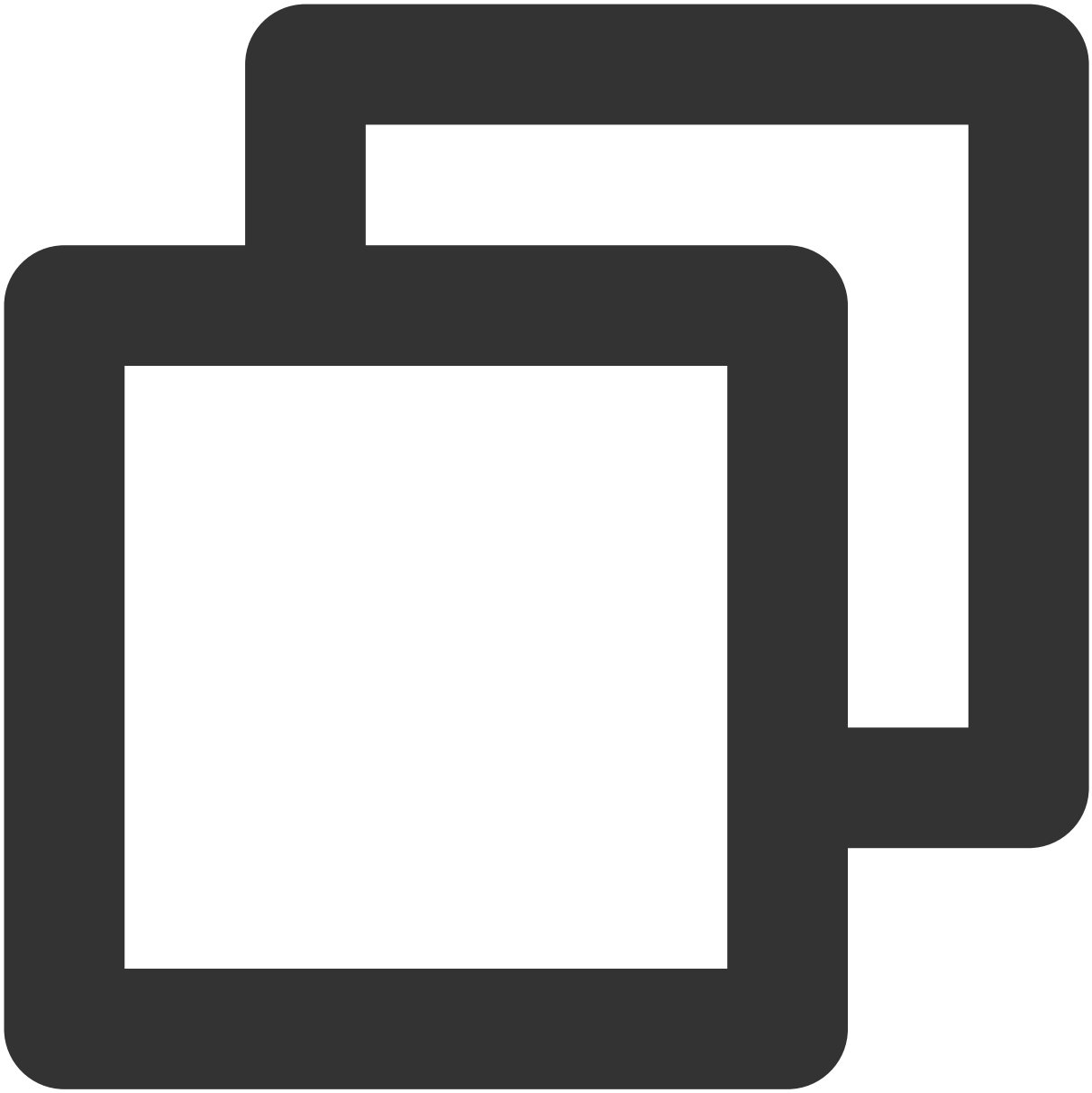
```
apiVersion: data.fluid.io/v1alpha1
kind: DataLoad
metadata:
  name: slice1-dataload
spec:
  dataset:
    name: slice1
    namespace: default
```

The dataset status is “Bond”, and 0% is cached.



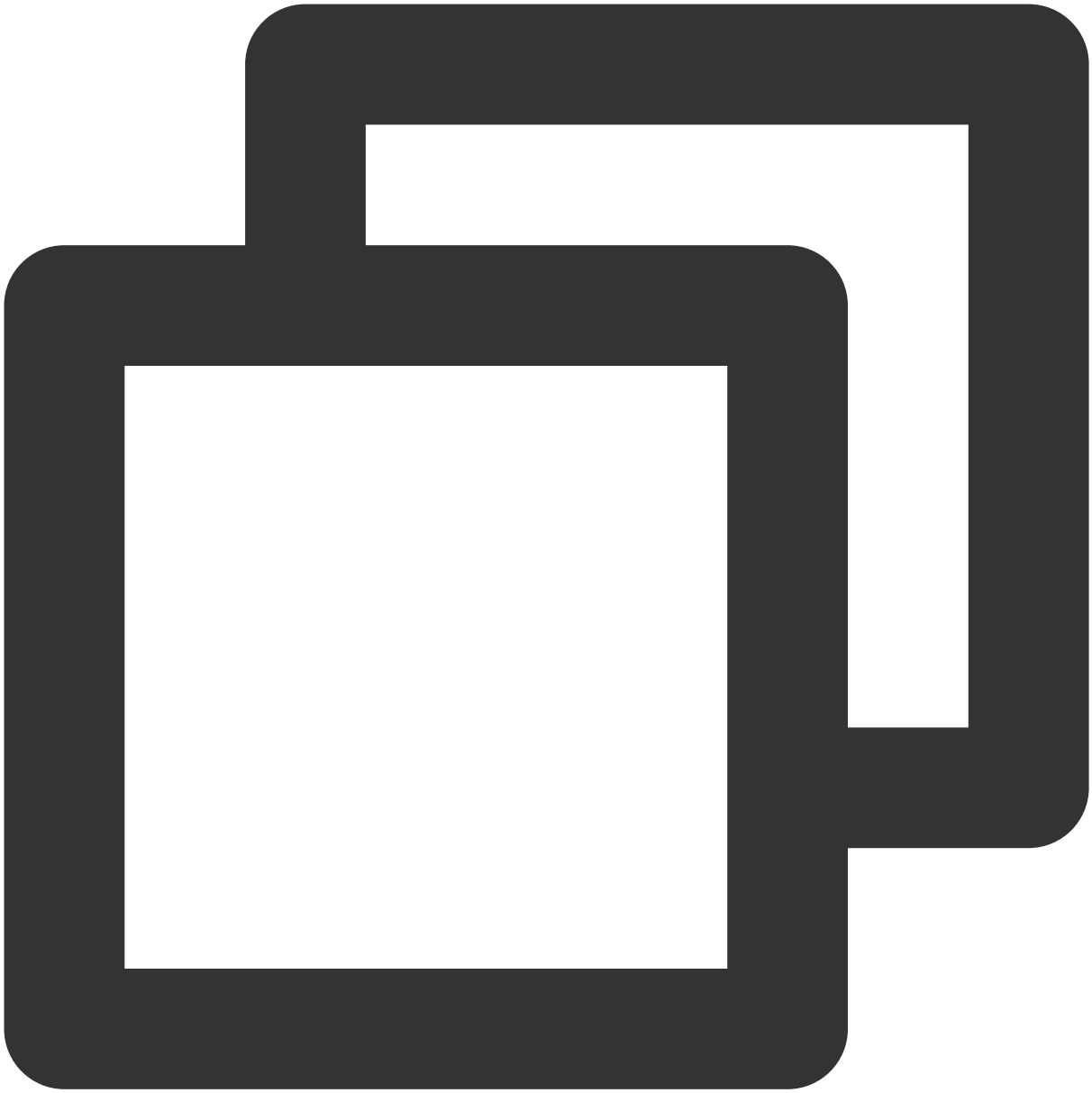
```
[root@master01 run]# kubectl get dataset
NAME     UFS TOTAL SIZE   CACHED   CACHE CAPACITY   CACHED PERCENTAGE   PHASE   AGE
slice1   97.67MiB         0.00B    4.00GiB          0.0%                Bound   31m
```

Start loading data:



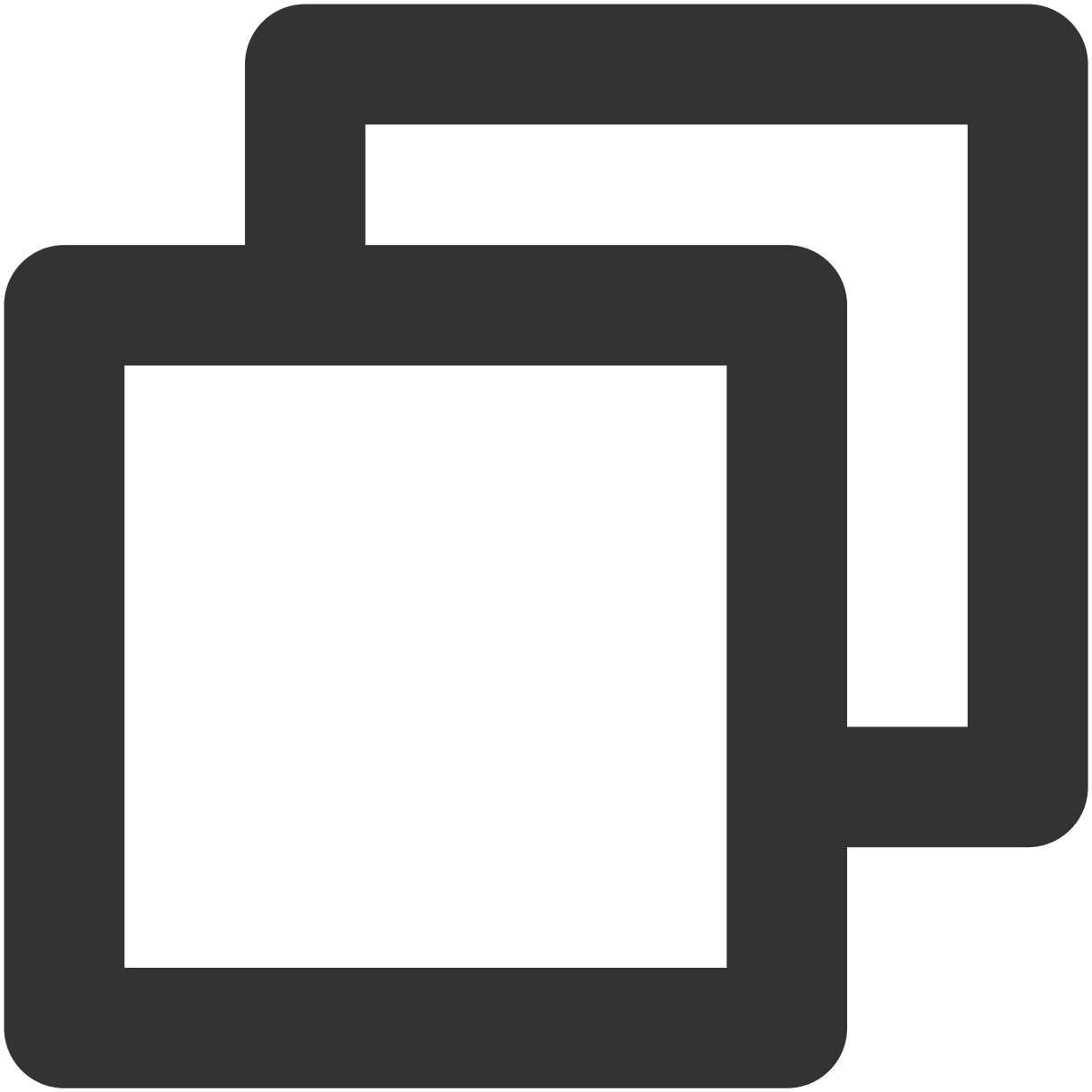
```
[root@master01 run]# kubectl apply -f dataload.yaml
dataload.data.fluid.io/slice1-dataload created
```

View data loading progress:



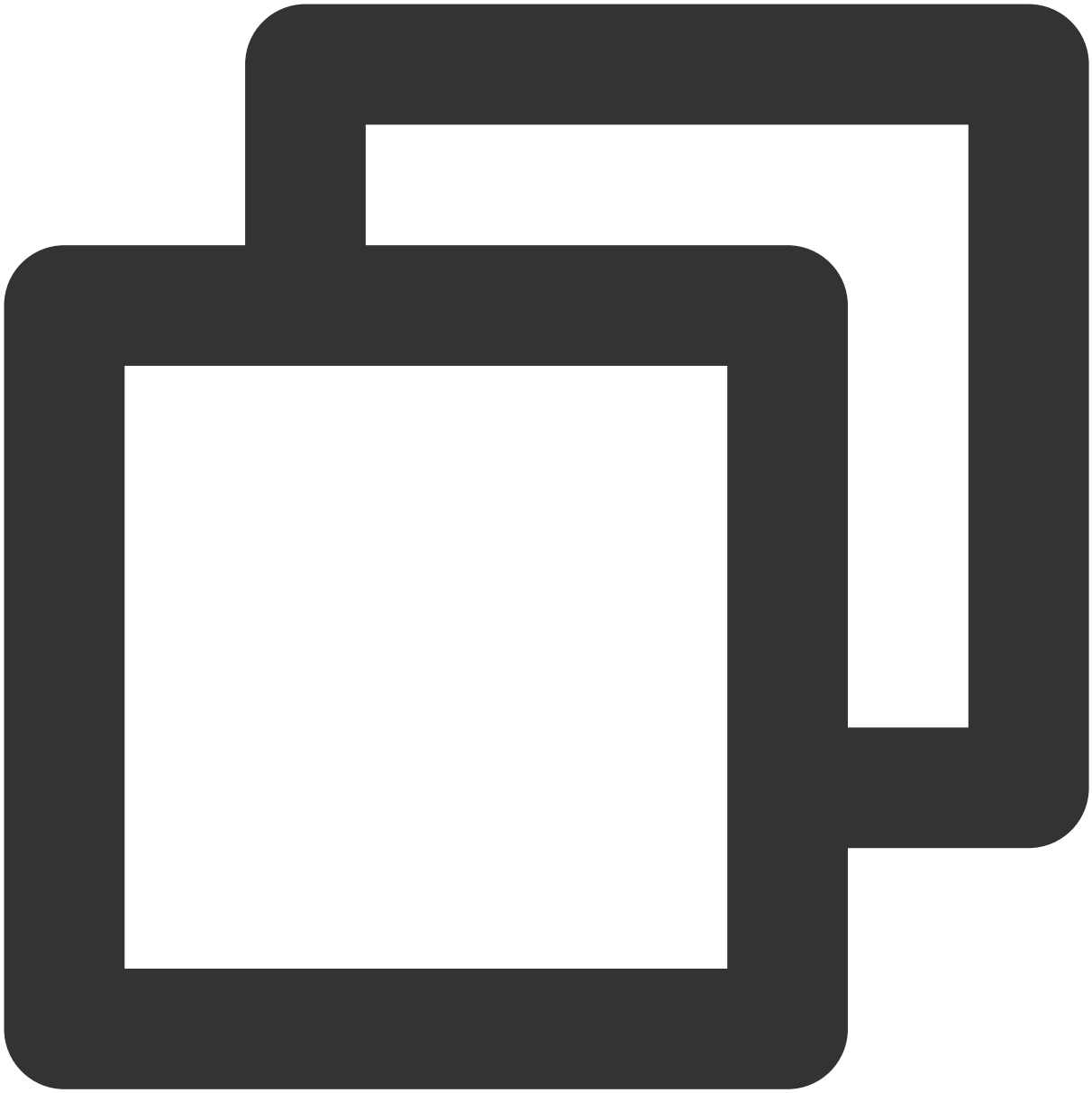
```
[root@master01 run]# kubectl get dataset --watch
NAME     UFS TOTAL SIZE   CACHED     CACHE CAPACITY   CACHED PERCENTAGE   PHASE   A
slice1   97.67MiB         52.86MiB   4.00GiB          54.1%               Bound   3
slice1   97.67MiB         53.36MiB   4.00GiB          54.6%               Bound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

100% of the data is cached:



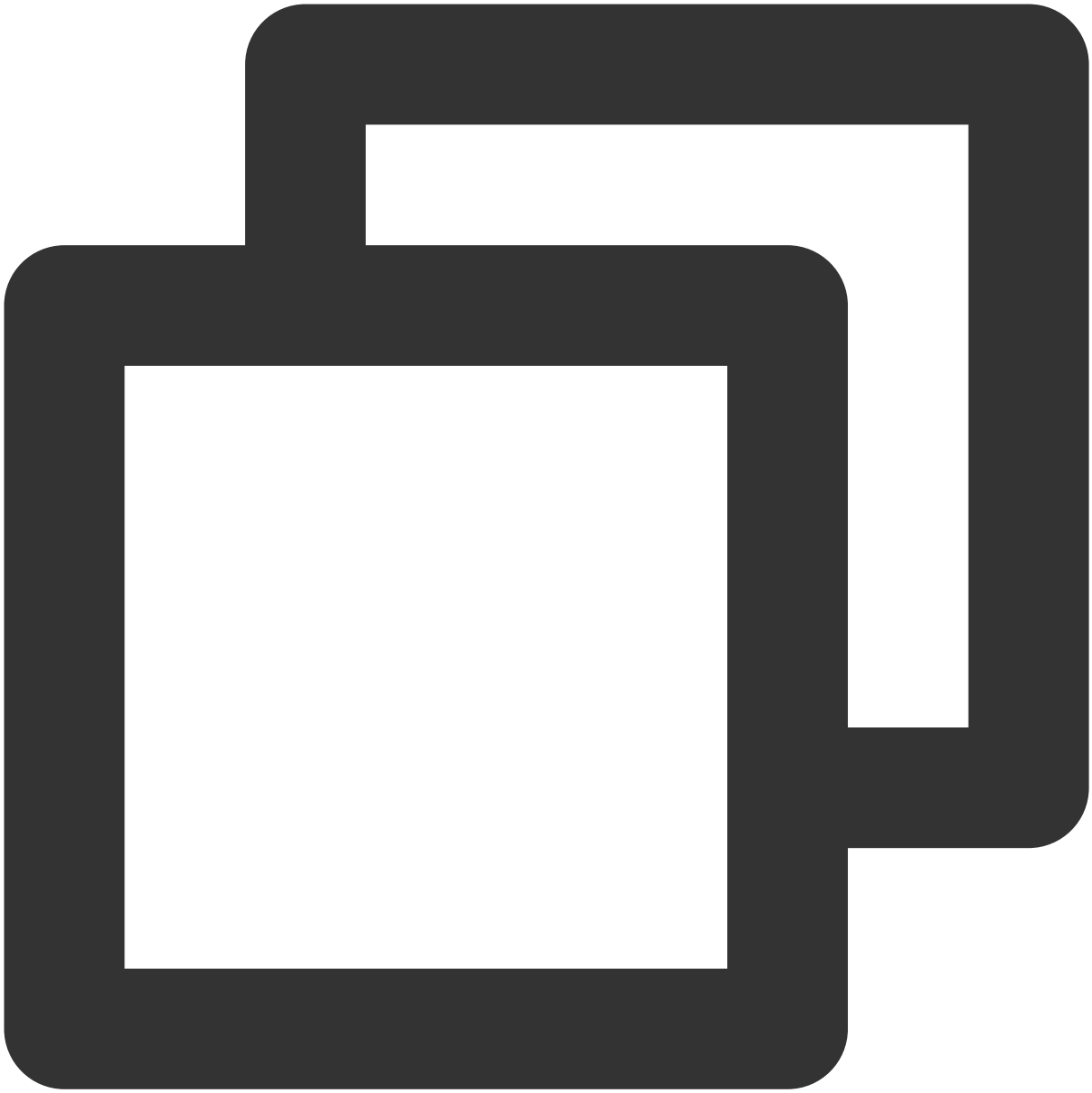
```
[root@master01 run]# kubectl get dataset --watch
NAME     UFS TOTAL SIZE   CACHED     CACHE CAPACITY   CACHED PERCENTAGE   PHASE   A
slice1   97.67MiB         97.67MiB   4.00GiB          100.0%              Bound   4
```

5. View data



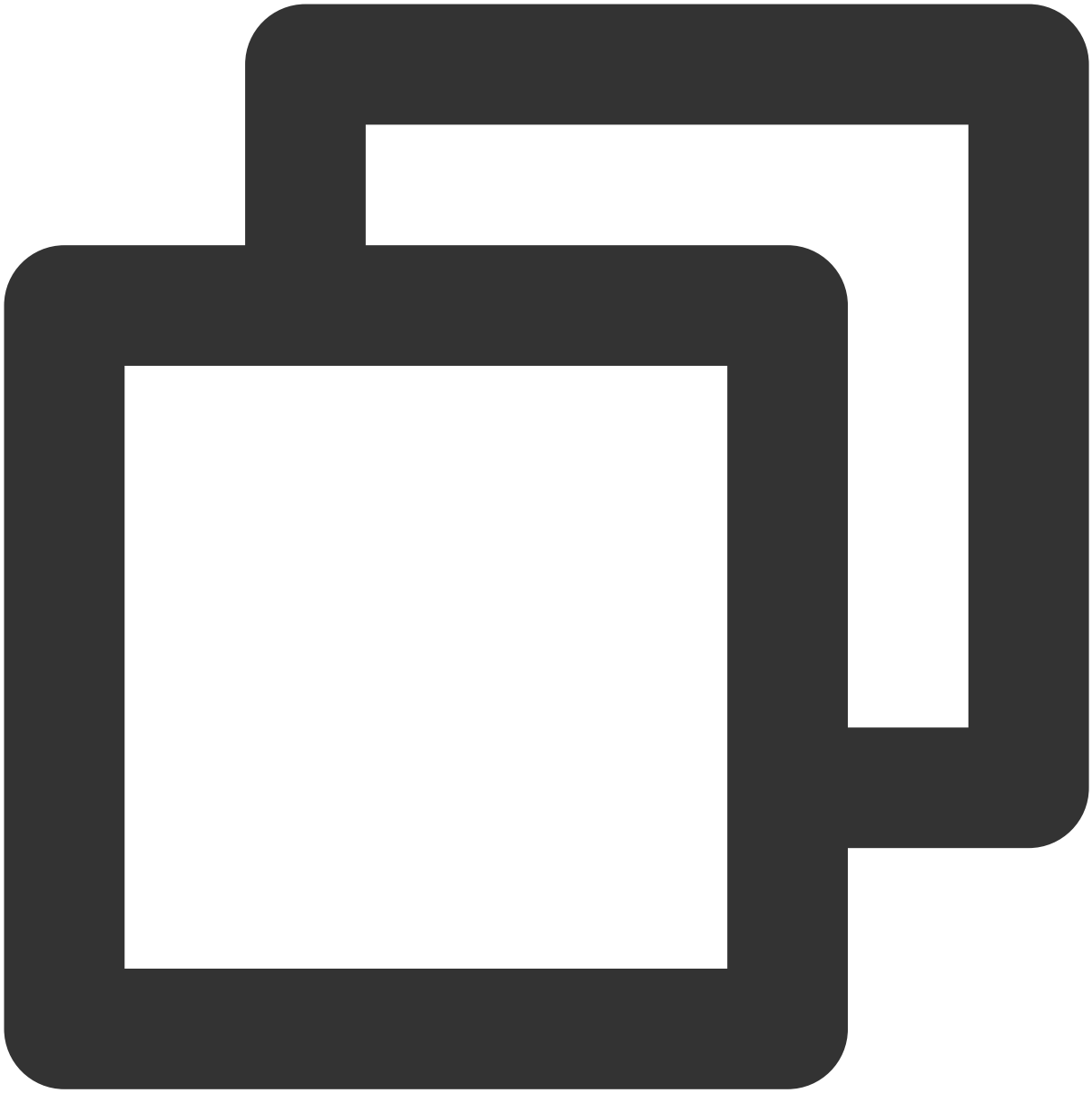
```
[root@master01 run]# kubectl get pods
NAME                                READY   STATUS      RESTARTS   AGE
slice1-dataload-loader-job-km6mg    0/1     Completed   0          12m
slice1-fuse-xsvwj                   1/1     Running     0          17m
slice1-master-0                     2/2     Running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

Go to the GooseFS master container:



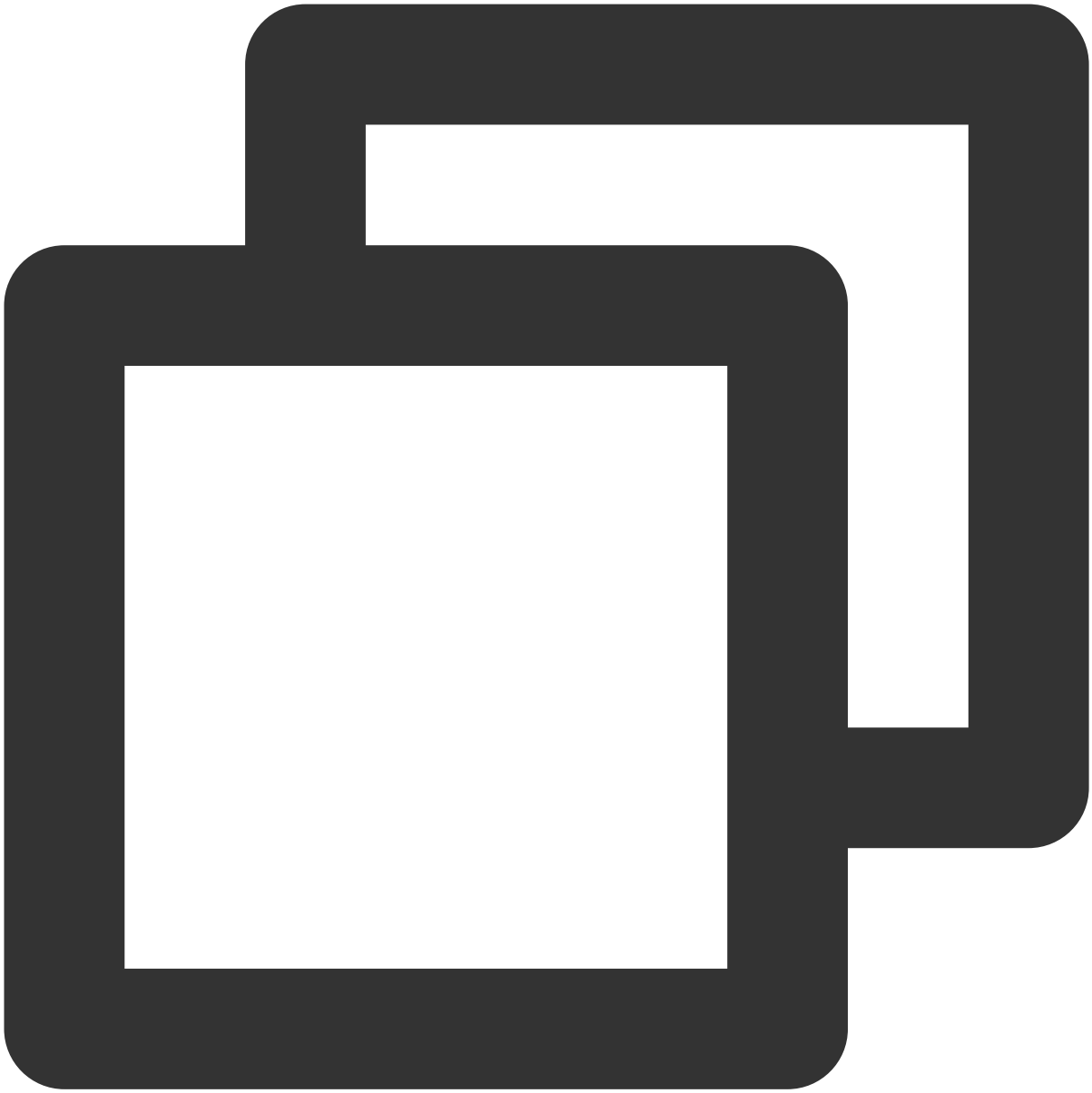
```
[root@master01 run]# kubectl exec -it slice1-master-0 -- /bin/bash
Defaulting container name to goosefs-master.
```

List GooseFS directories:



```
[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
```

View a specific file:



```
[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
```

Deploying with Docker

Last updated：2024-03-25 16:04:01

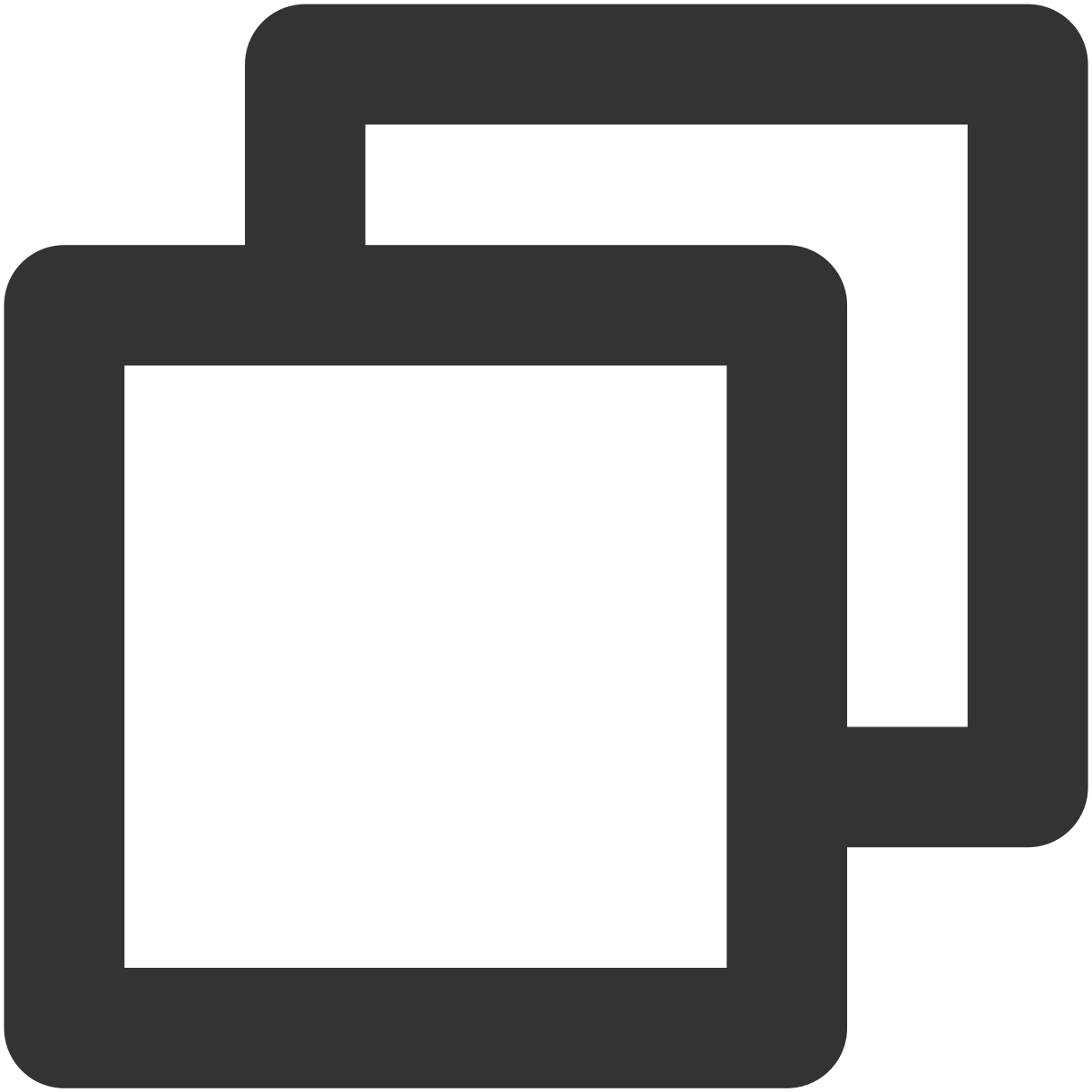
This document describes how to use Docker to deploy GooseFS.

Preparations

1. You have installed Docker 19.03.14 or a later version.
2. You have installed CentOS 7 or a later version
3. You have obtained a GooseFS docker image, such as goosefs:v1.0.0.

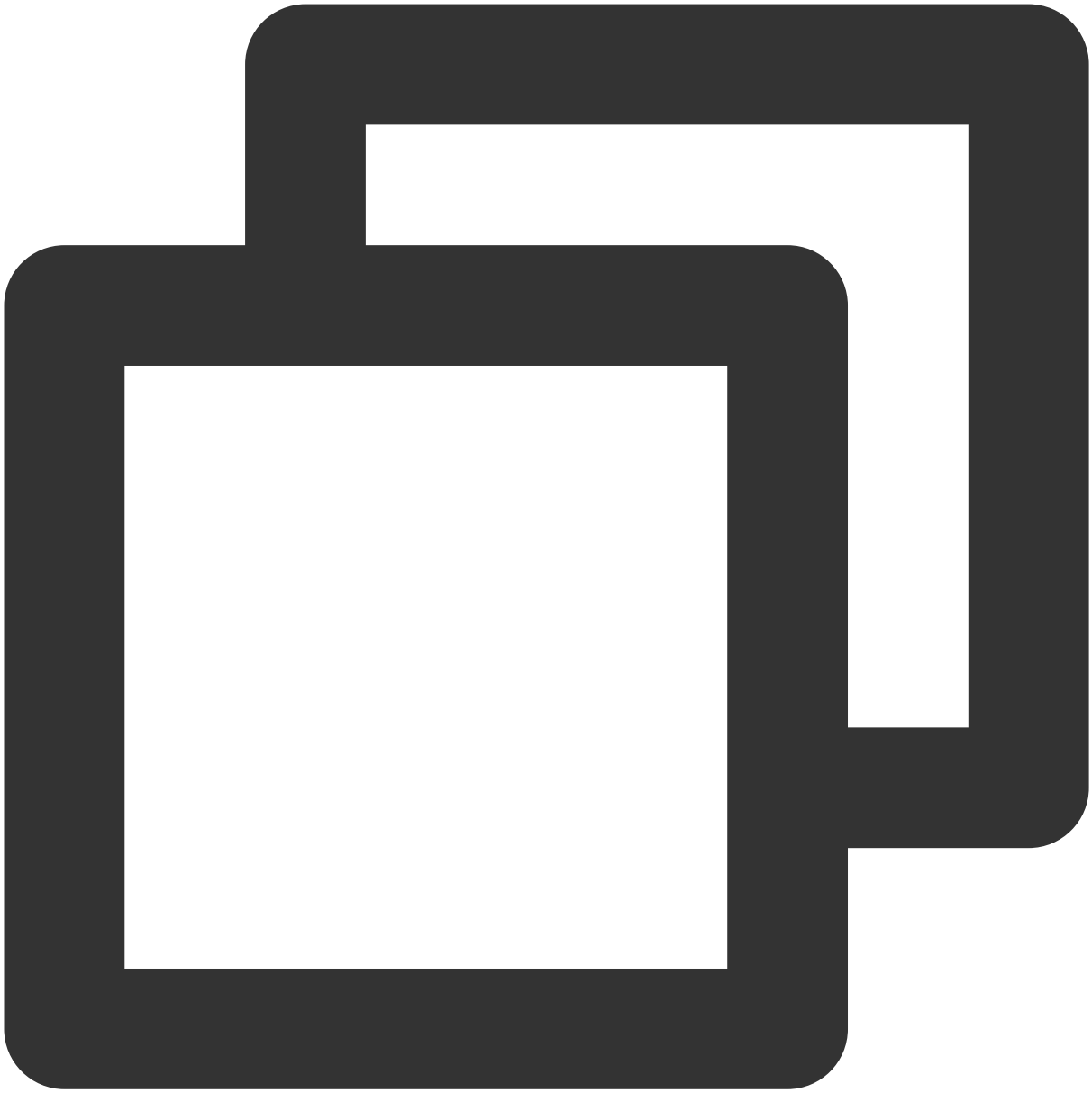
Installation

1. Create a directory called `ufs` to mount the local directory to the root directory of GooseFS:



```
mkdir /tmp/goosefs_ufs
```

2. Run the master process:



```
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
```

Note

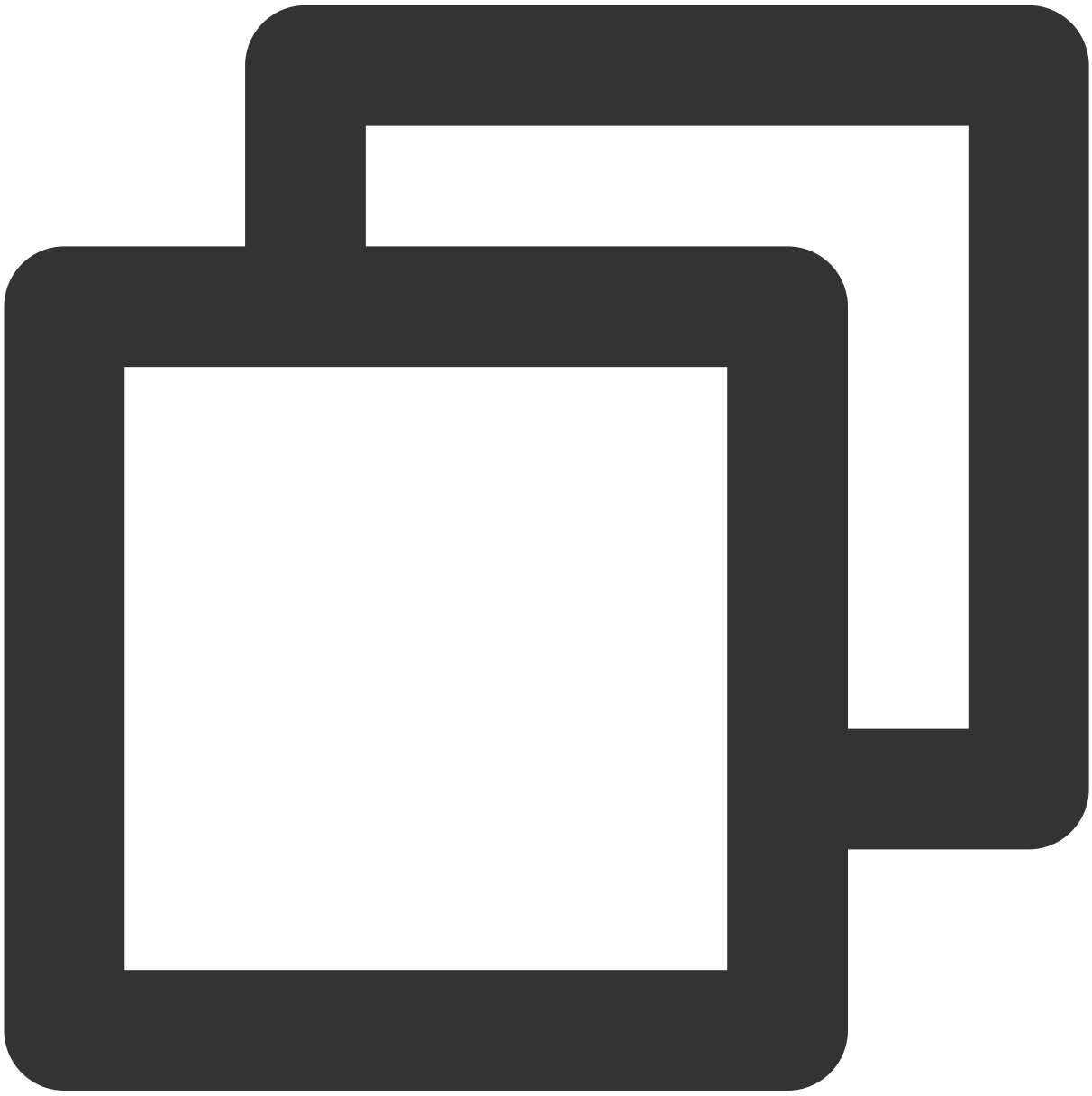
-Dgoosefs.master.hostname: sets the master address.

-Dgoosefs.master.mount.table.root.ufs: sets the mount point in the root directory of GooseFS.

-v /tmp/goosefs_ufs:/opt/data: maps the local directory to the docker container.

--net=host: Docker uses the host's network.

3. Run the worker process.

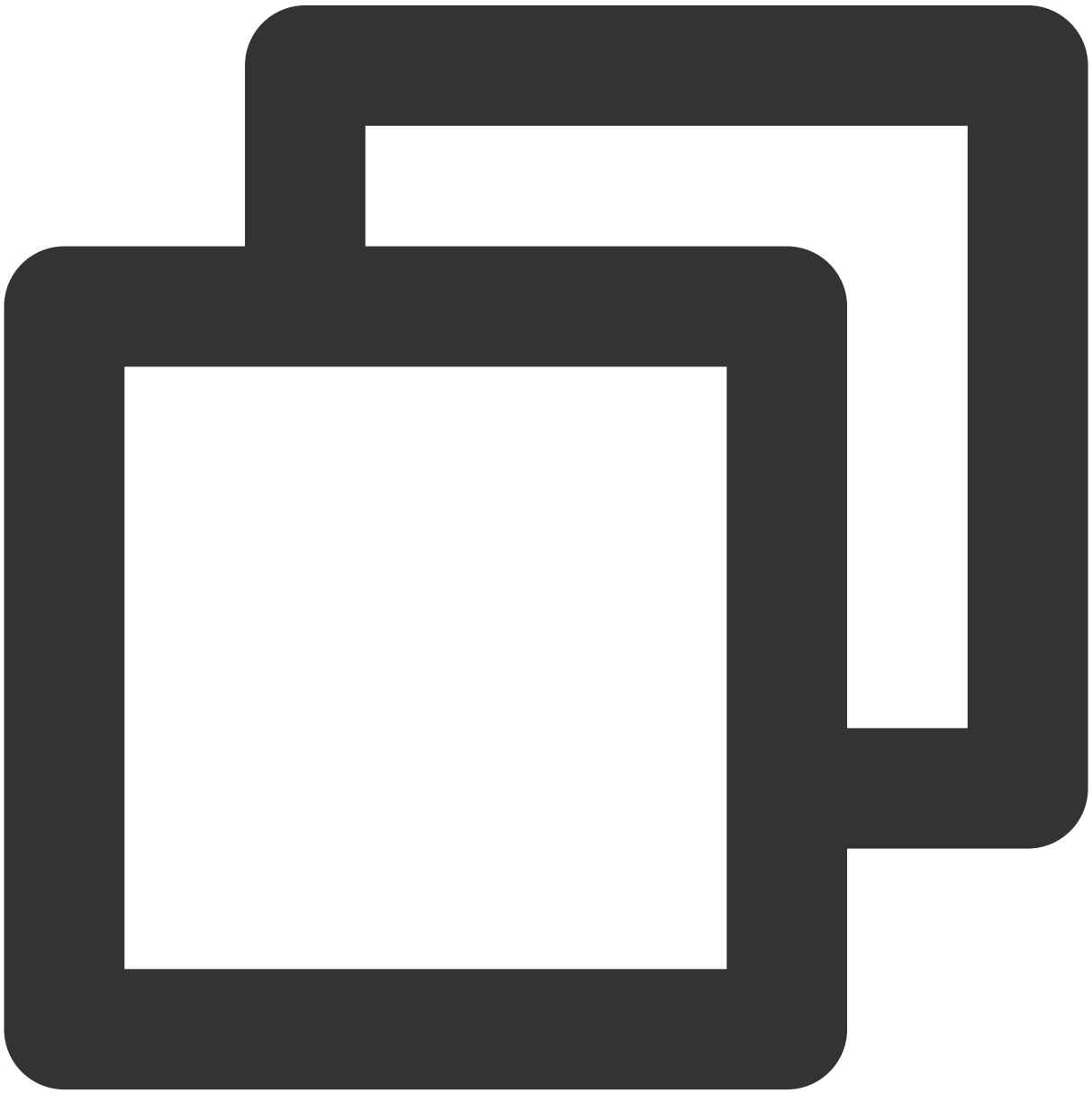


```
docker run -d --rm \\
--net=host \\
--name=goosefs-worker1 \\
--shm-size=1G \\
```

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

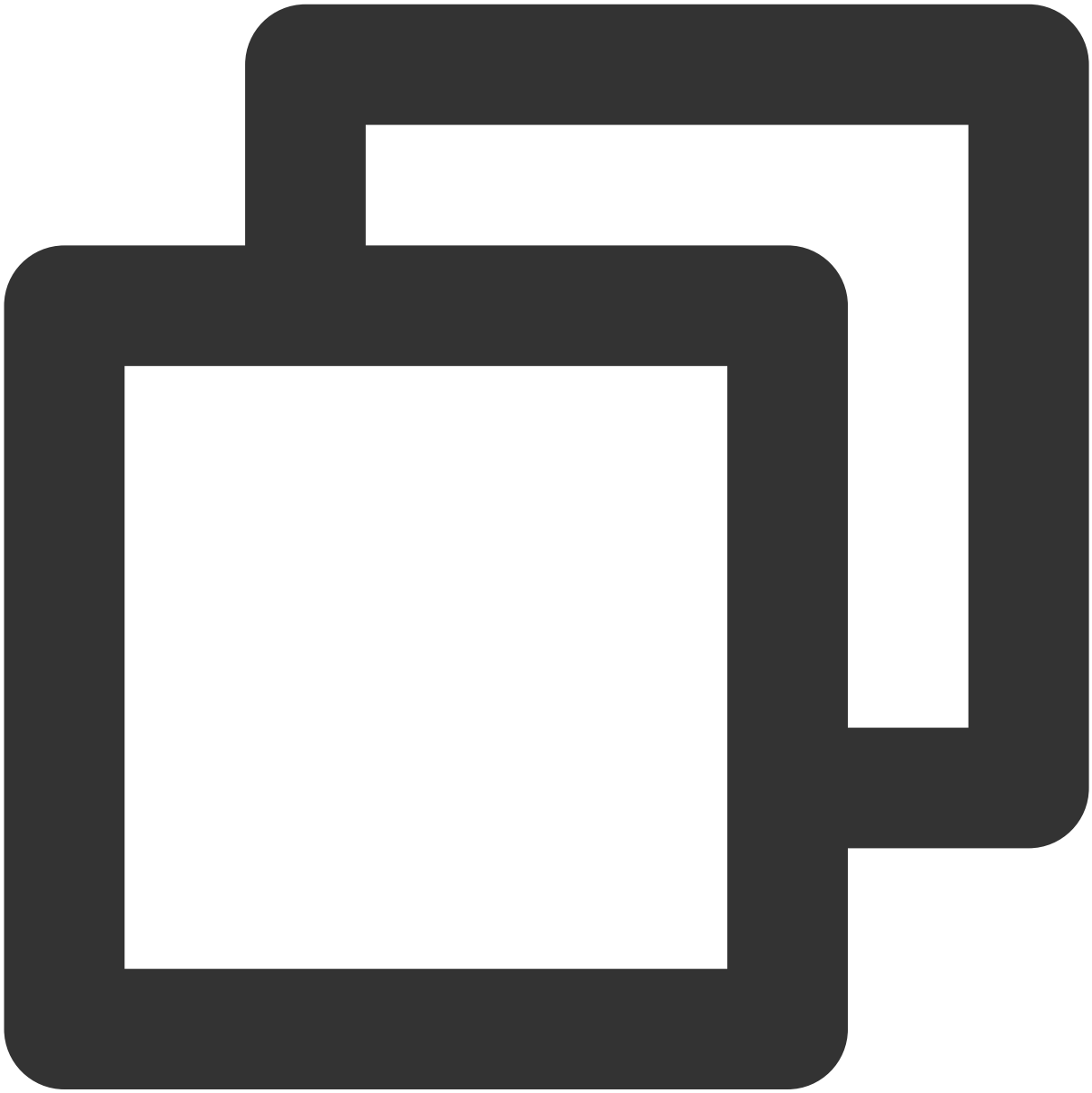
Operation Example

1. View the container:



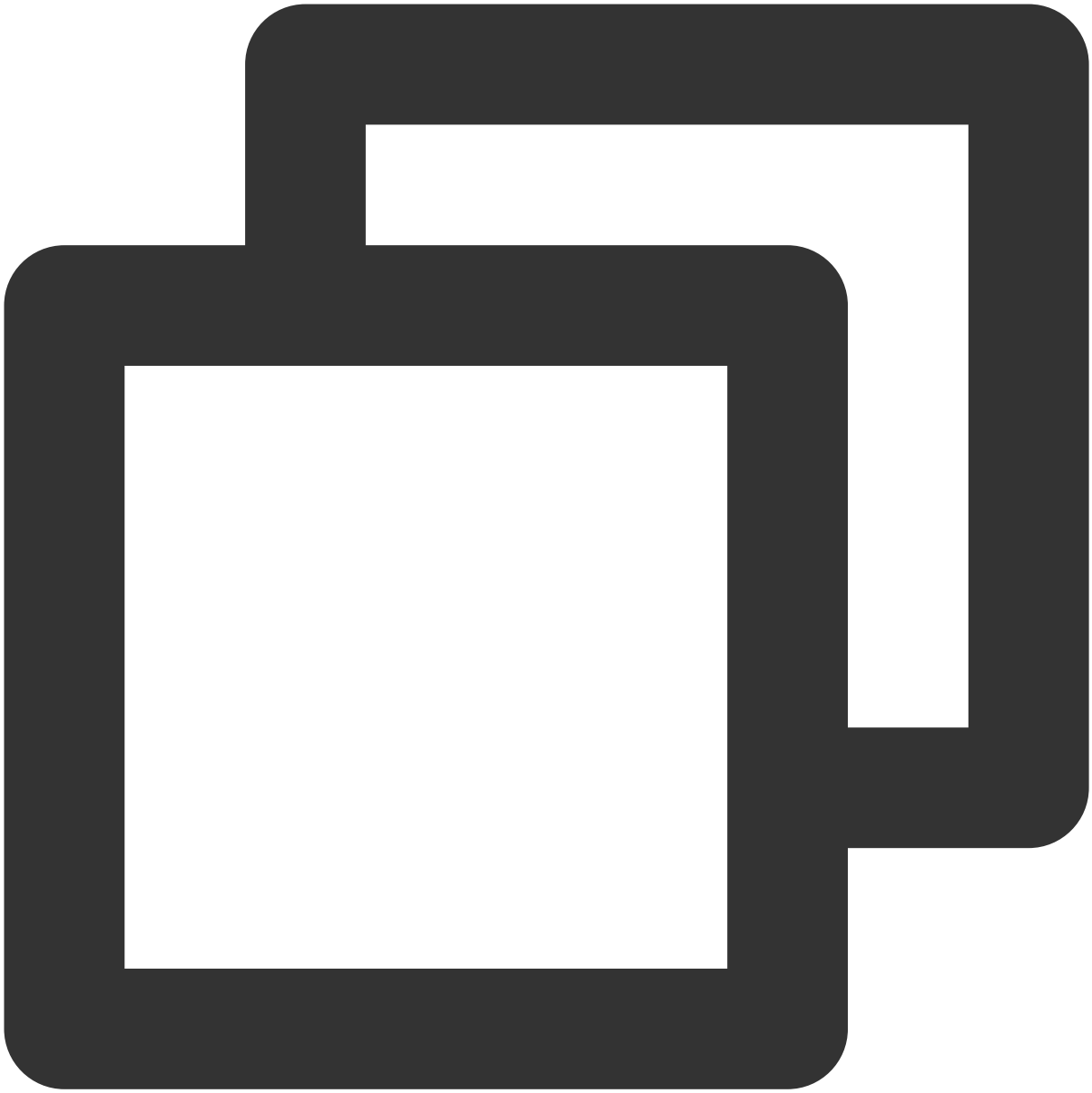
```
[root@VM-0-7-centos ~]# docker ps | grep goosefs
0bda1cac76f4        goosefs:v1.0.0      "/entrypoint.sh mast…"   32 minutes ago
b6260f9a0134        goosefs:v1.0.0     "/entrypoint.sh work…"   About an hour ago
```

2. Go to the container:



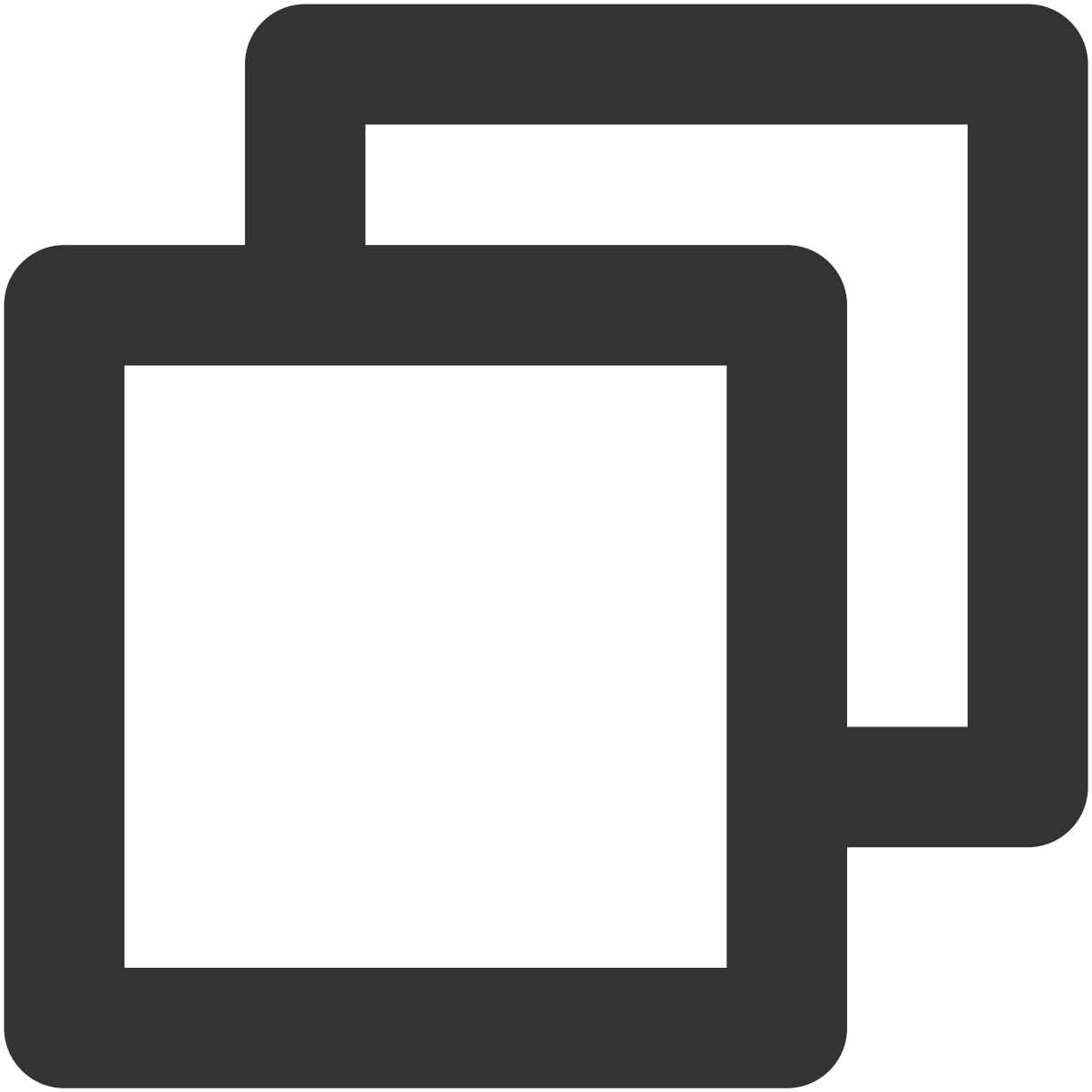
```
docker exec -it 0bda1cac76f4 /bin/bash
```

3. Mount the COS directory:



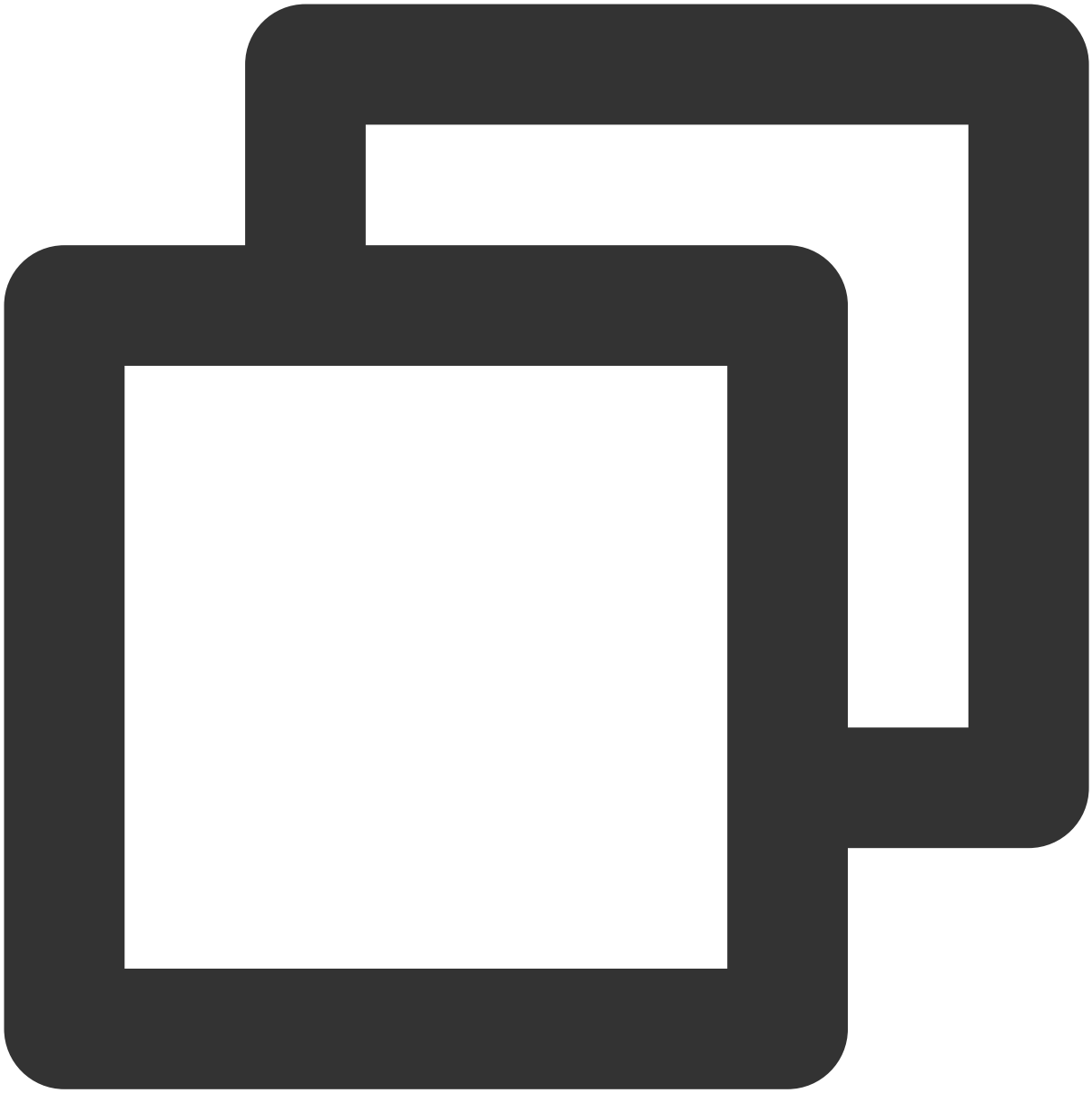
```
goosefs fs mount --option fs.cosn.userinfo.secretId={secretId} \\
    --option fs.cosn.userinfo.secretKey={secretKey} \\
    --option fs.cosn.bucket.region=ap-beijing \\
    --option fs.cosn.impl=org.apache.hadoop.fs.CosFileSystem \\
    --option fs.AbstractFileSystem.cosn.impl=org.apache.hadoop.fs.CosN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

4. View the directory:



```
[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. View the worker node:



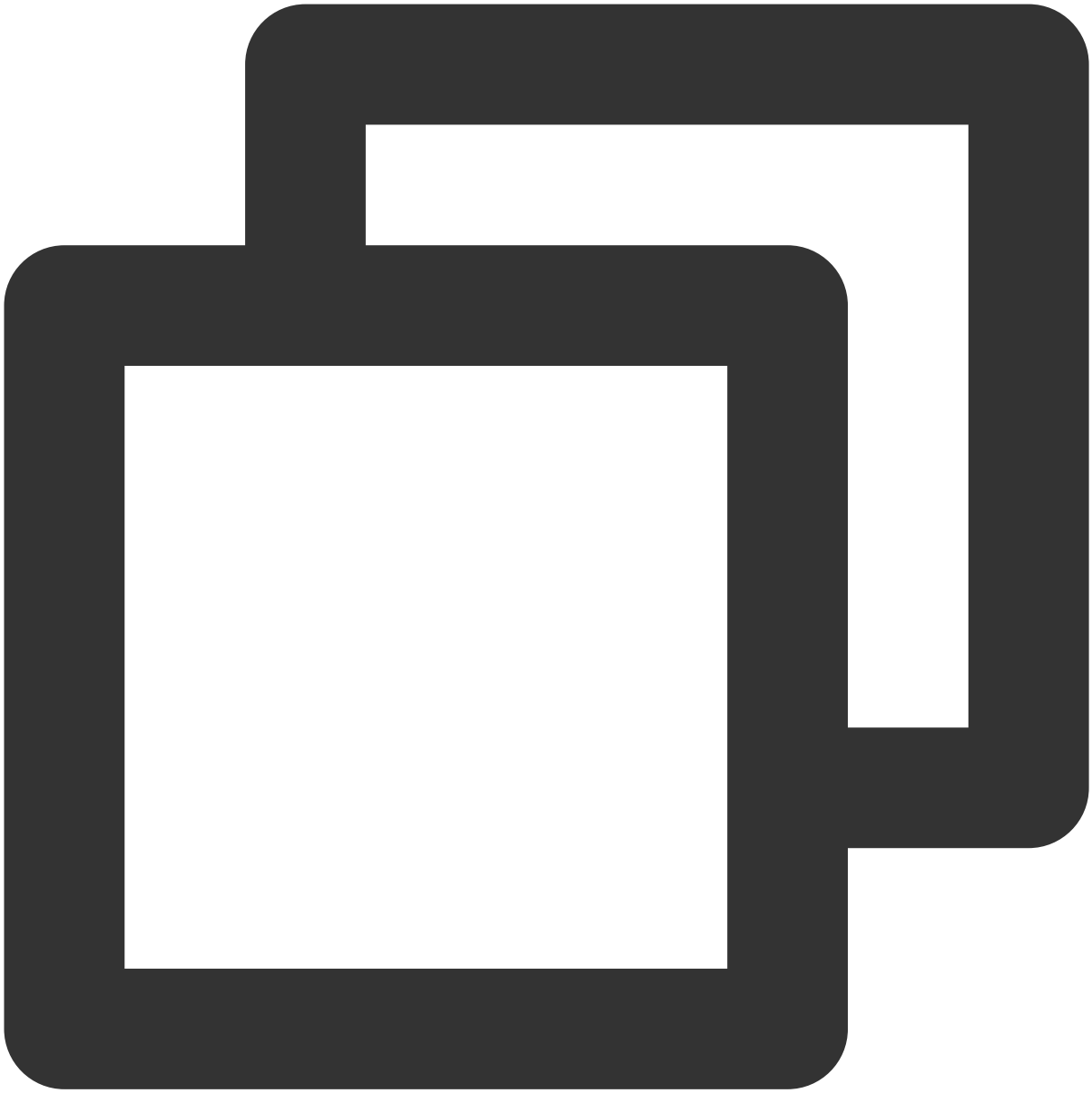
```
[goosefs@VM-0-7-centos goosefs-1.0.0-SNAPSHOT-noUI-noHelm]$ goosefs fsadmin report
Capacity information for all workers:
   Total Capacity: 1024.00MB
       Tier: MEM  Size: 1024.00MB
   Used Capacity: 0B
       Tier: MEM  Size: 0B
   Used Percentage: 0%
   Free Percentage: 100%

Worker Name      Last Heartbeat   Storage       MEM
```

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

OPS Guide
Logging Guide
GooseFS Logs

Last updated：2024-03-25 16:04:01

When GooseFS's master and workers or computing frameworks such as Spark request GooseFS through a GooseFS client, logs will be recorded for troubleshooting. GooseFS outputs logs based on Log4j. Therefore, you can modify `log4j.properties` to change the log output configuration, such as the storage path, log level, or whether to record RPC logs. You can go to the GooseFS configuration directory and modify `log4j.properties` as follows:



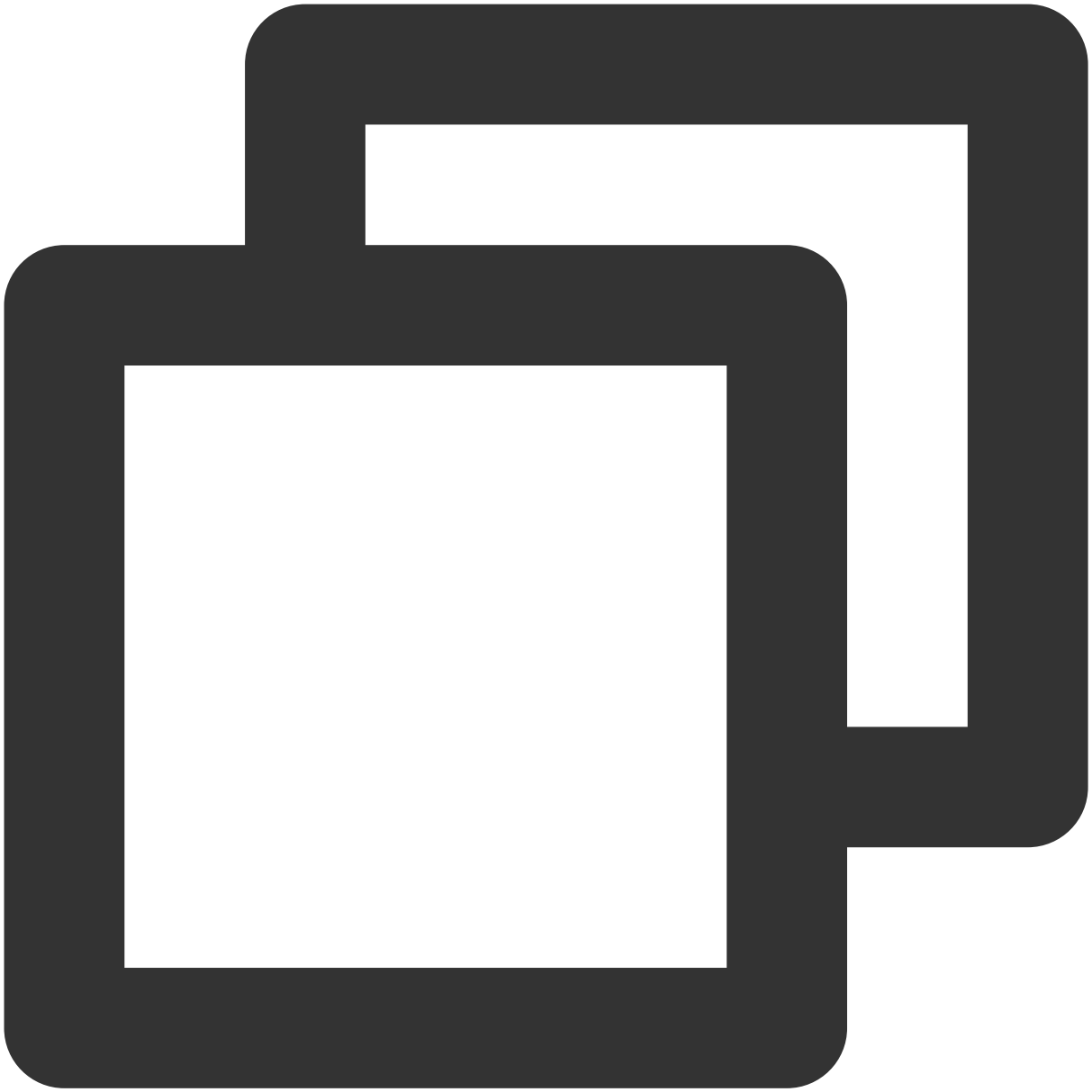
```
$ cd /usr/local/service/goosefs/conf
$ cat log4j.properties

# 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

...
```

The log configuration of GooseFS is described below.

Storage Location

By default, logs collected by GooseFS are stored in `${GOOSEFS_HOME}/logs`, where master logs are stored in `logs/master.log` and worker logs in `logs/worker.log`. Note that errors thrown by node processes are recorded in `master.out` or `worker.out`, which are empty if there is no error. If any error occurs, you can use these files to troubleshoot.
Below are common configuration items for the master log storage:



```
# 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
```

Parameters are described as follows:

MASTER_LOGGER: configures master log output.

MASTER_LOGGER.File: sets the log storage path. You can modify the value to customize a storage path.
MASTER_LOGGER.MaxFileSize: sets the maximum size of a single log file.
MASTER_LOGGER.MaxBackupIndex: sets the maximum number of log files.
MASTER_LOGGER.layout: specifies the layout of the output log.
MASTER_LOGGER.layout.ConversionPattern: specifies the format of the output log.

Note:

.log files are rolled. You can back them up to a UFS such as COS. However, .out files are not rolled and thus need to be deleted manually if needed.

For more information about Log4j parameters, please see Log4j Configuration.

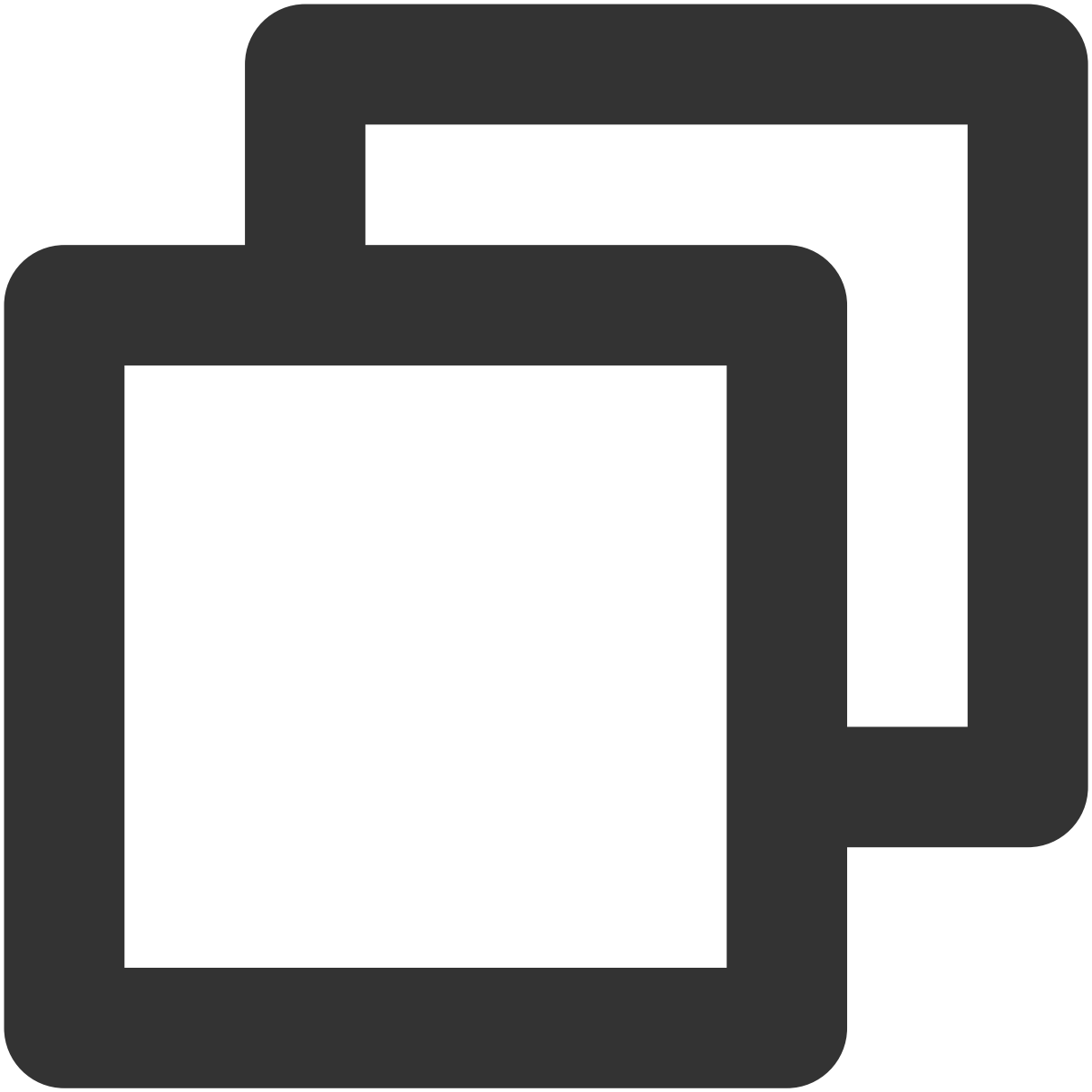
GooseFS stores only logs generated by itself. For logs generated by upper-layer computing applications, view the specific application's configuration for the log location. For the log configurations of common computing applications, please see Apache Hadoop, Apache HBase, Apache Hive, and Apache Spark.

Log Levels

GooseFS has the following five log levels:
TRACE: finer-grained calling logs that are suitable for debugging method/class calls.
DEBUG: fine-grained calling logs that are useful for debugging.
INFO: important information about request handling
WARN: warning information (the task can still run, but there might be potential problems)
ERROR: error message (the running of the task is affected)

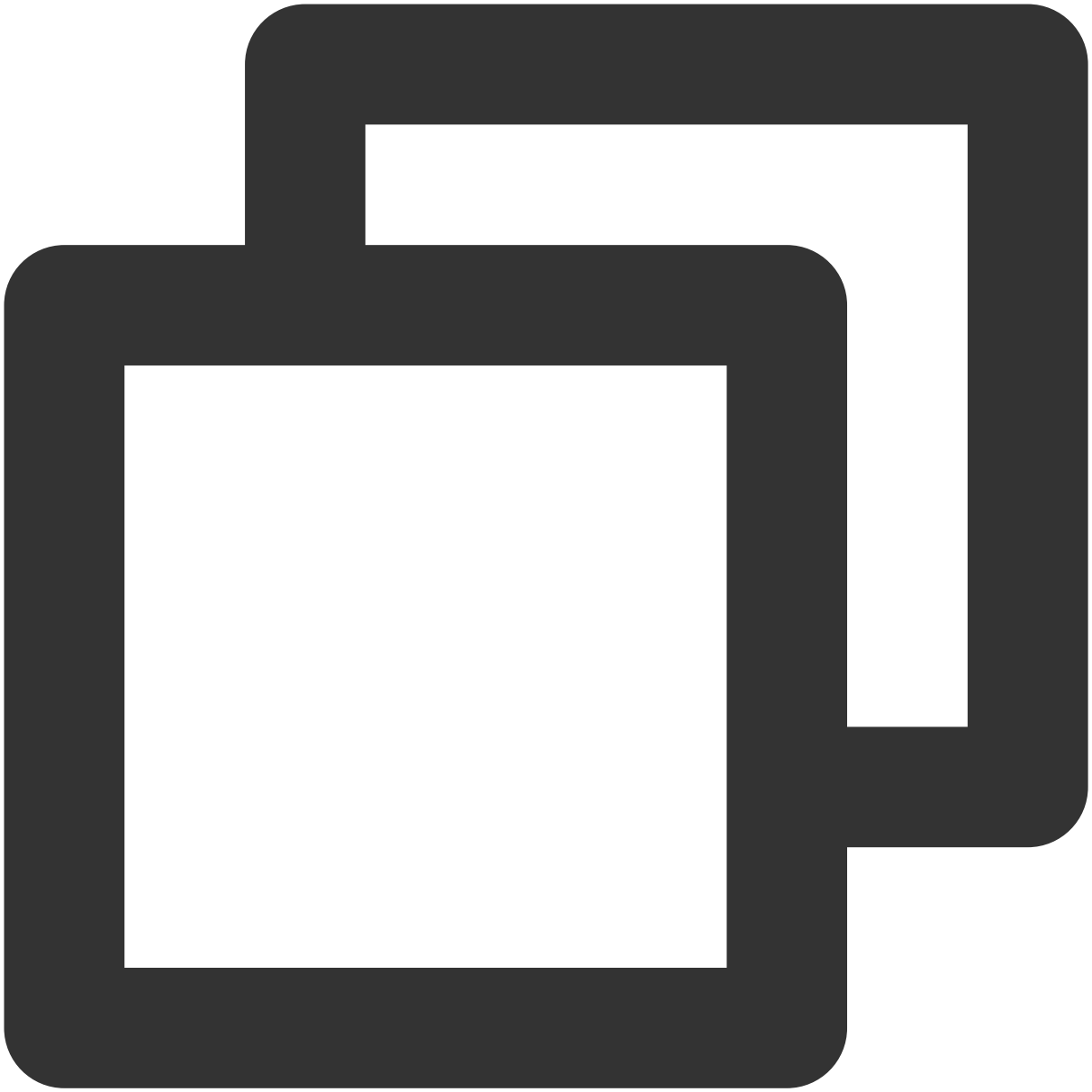
The five log levels are ordered according to how detailed the logs are (the first level is the most detailed). A higher-level log also records log messages recorded in a lower-level one. By default, the log level of GooseFS is set to INFO, which records log messages of INFO, WARN, and ERROR.

You can go to GooseFS's configuration directory and modify `log4j.properties`. The following example changes all log levels of GooseFS to DEBUG:



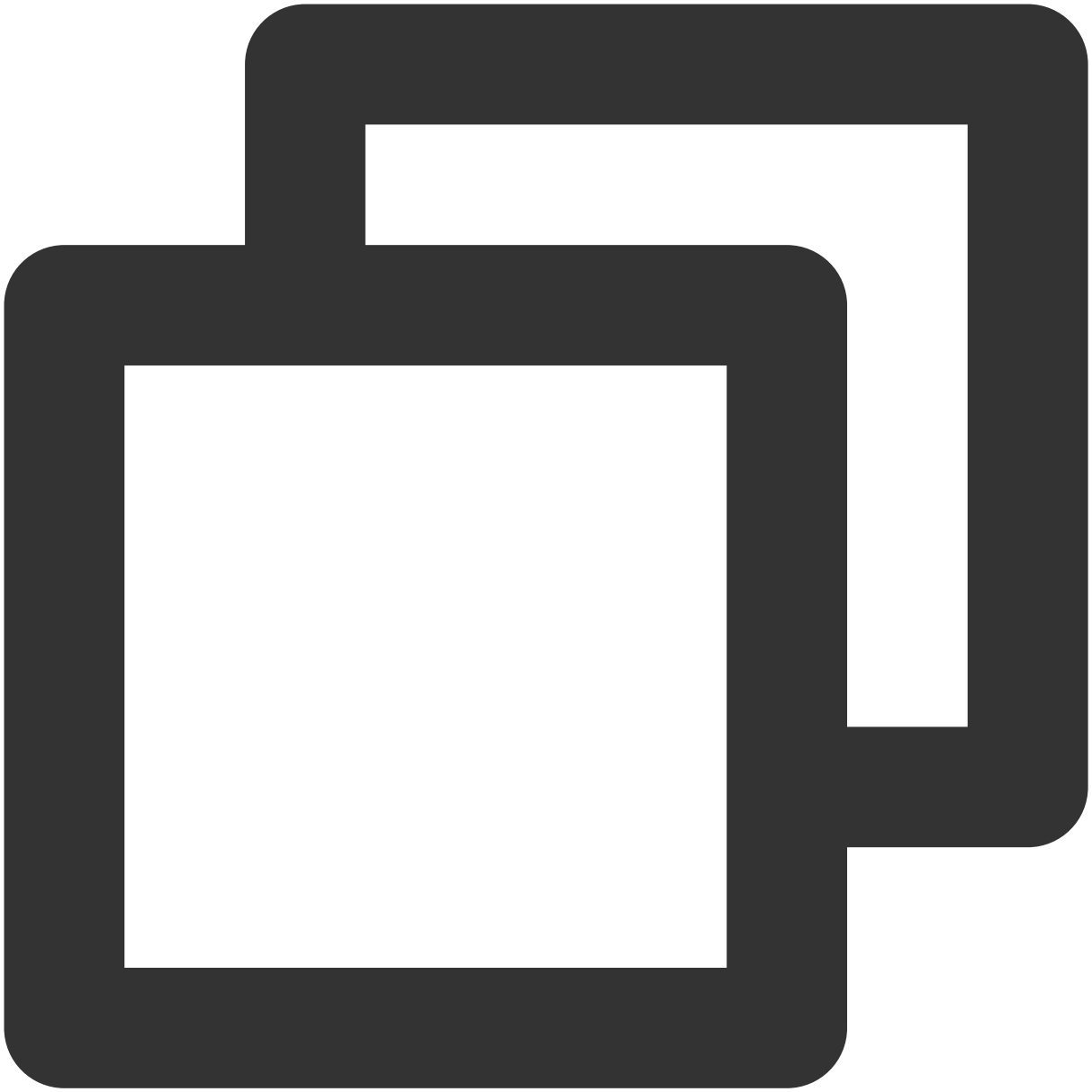
```
log4j.rootLogger=DEBUG, ${goosefs.logger.type}, ${goosefs.remote.logger.type}
```

To modify the log level of a specified class, you can declare it in the configuration file. The following example sets the log level of the `GooseFSFileInStream` class to DEBUG:



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

In most cases, you are advised to change the log level in the logging configuration file. However, sometimes you might need to change the logging parameters when the cluster is running. In this case, you can run the `goosefs logLevel` command to modify the log level. The following are configuration items supported by `logLevel`:



```
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
```

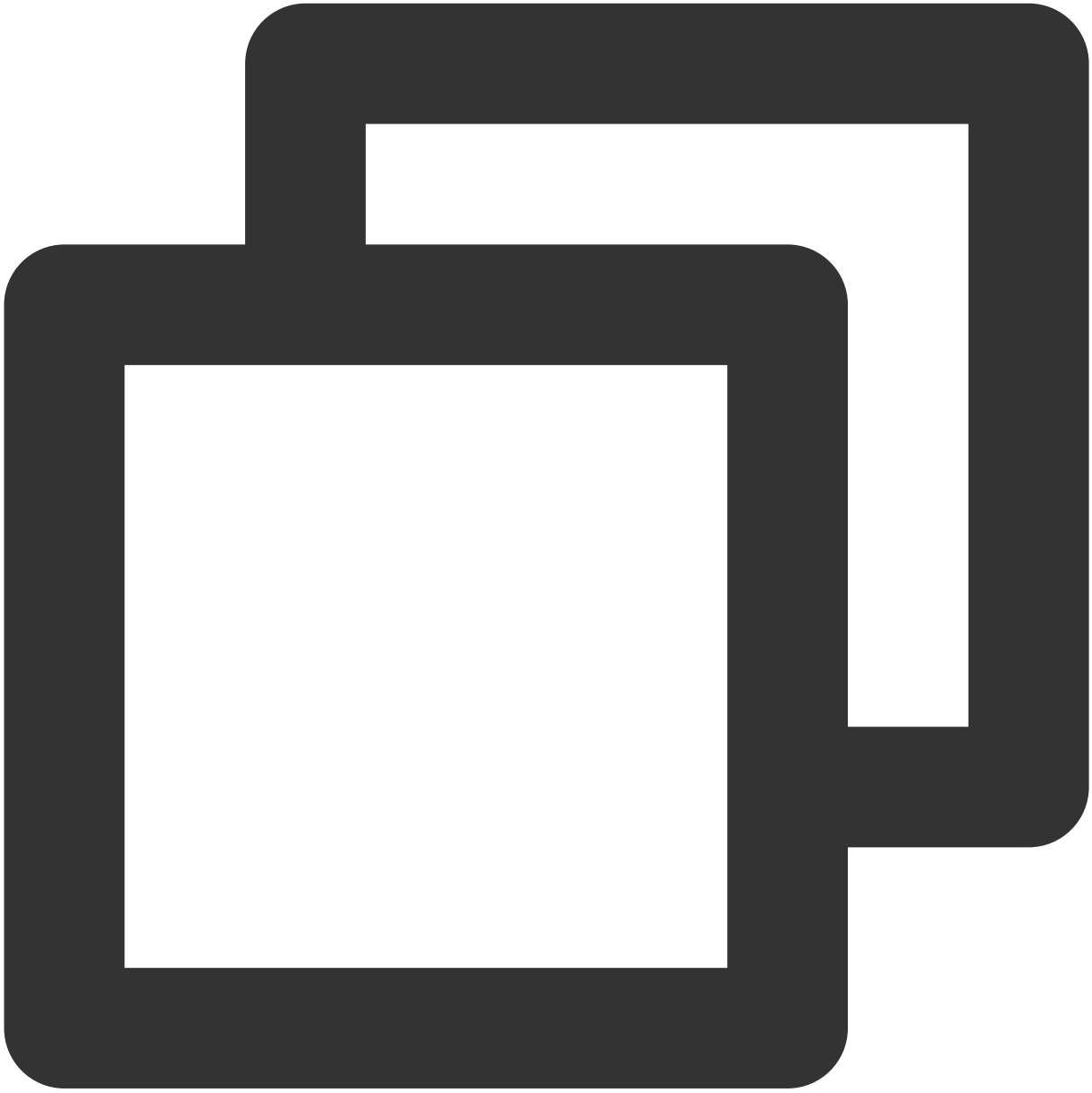
The configuration items are described as follows:

level: log level, which can be TRACE, DEBUG, INFO, WARN, or ERROR

logName: the logger's name, such as com.qcloud.cos.goosefs.underfs.hdfs.HdfsUnderFileSystem

target: targets to apply the change to, which can be the master or workers (specified by IP:PORT). By default, the change applies to the master and all workers.

You can change the log level when the system is running as needed to troubleshoot. The following example changes the log level of the `com.qcloud.cos.goosefs.underfs.hdfs.HdfsUnderFileSystem` class to DEBUG for all workers, and changes it back to INFO when the debugging is complete:



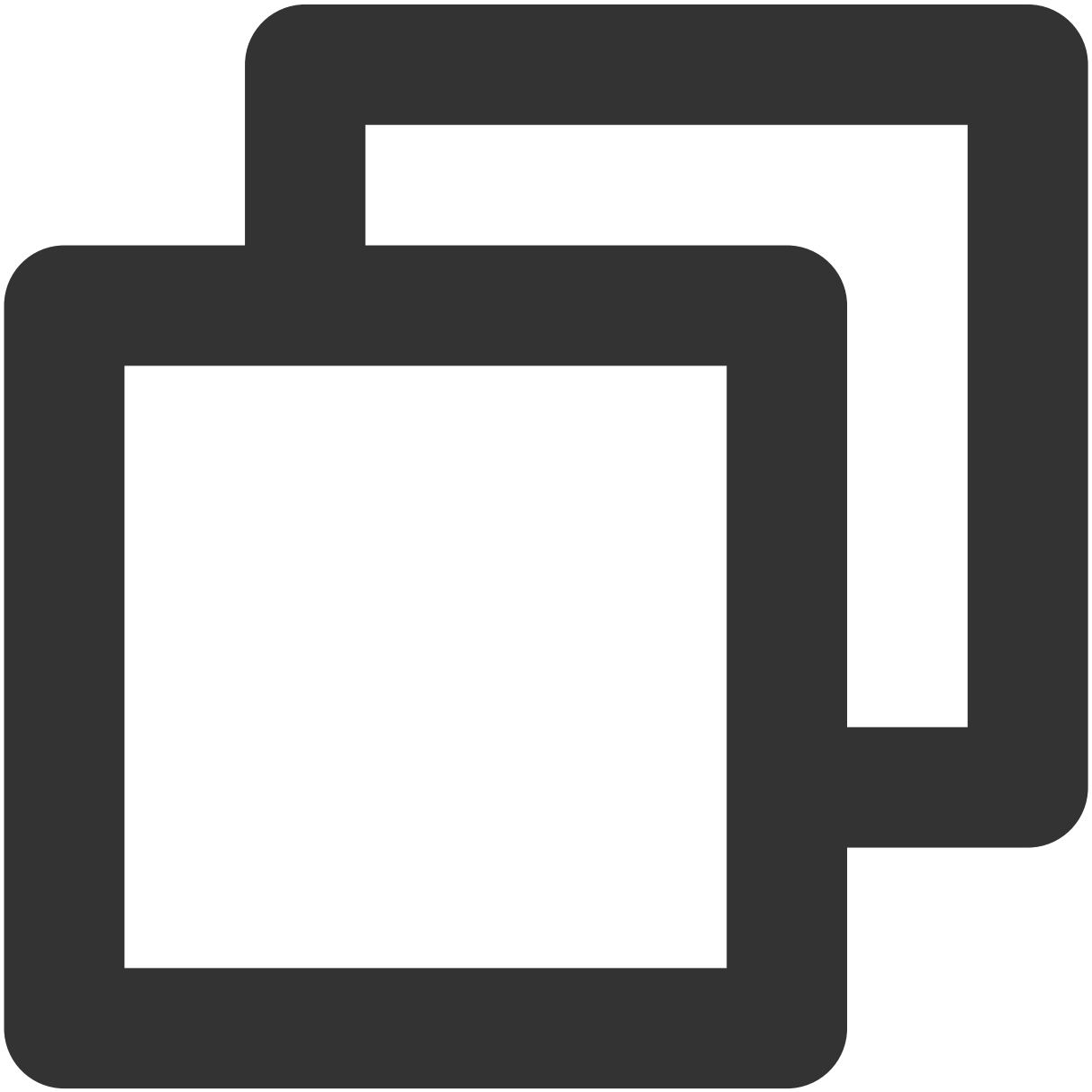
```
$ goosefs logLevel --logName=com.qcloud.cos.goosefs.underfs.hdfs.HdfsUnderFileSyst

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

Advanced Configurations

GooseFS allows you to configure GC event logs, FUSE logs, RPC logs, and UFS operation logs, and perform operations such as log segmentation and log filtering. The following describes how to use these advanced configurations.

GC event logs

GooseFS records GC event logs in .out files. You can add the following configuration to the `conf/goosefs-env.sh` file:

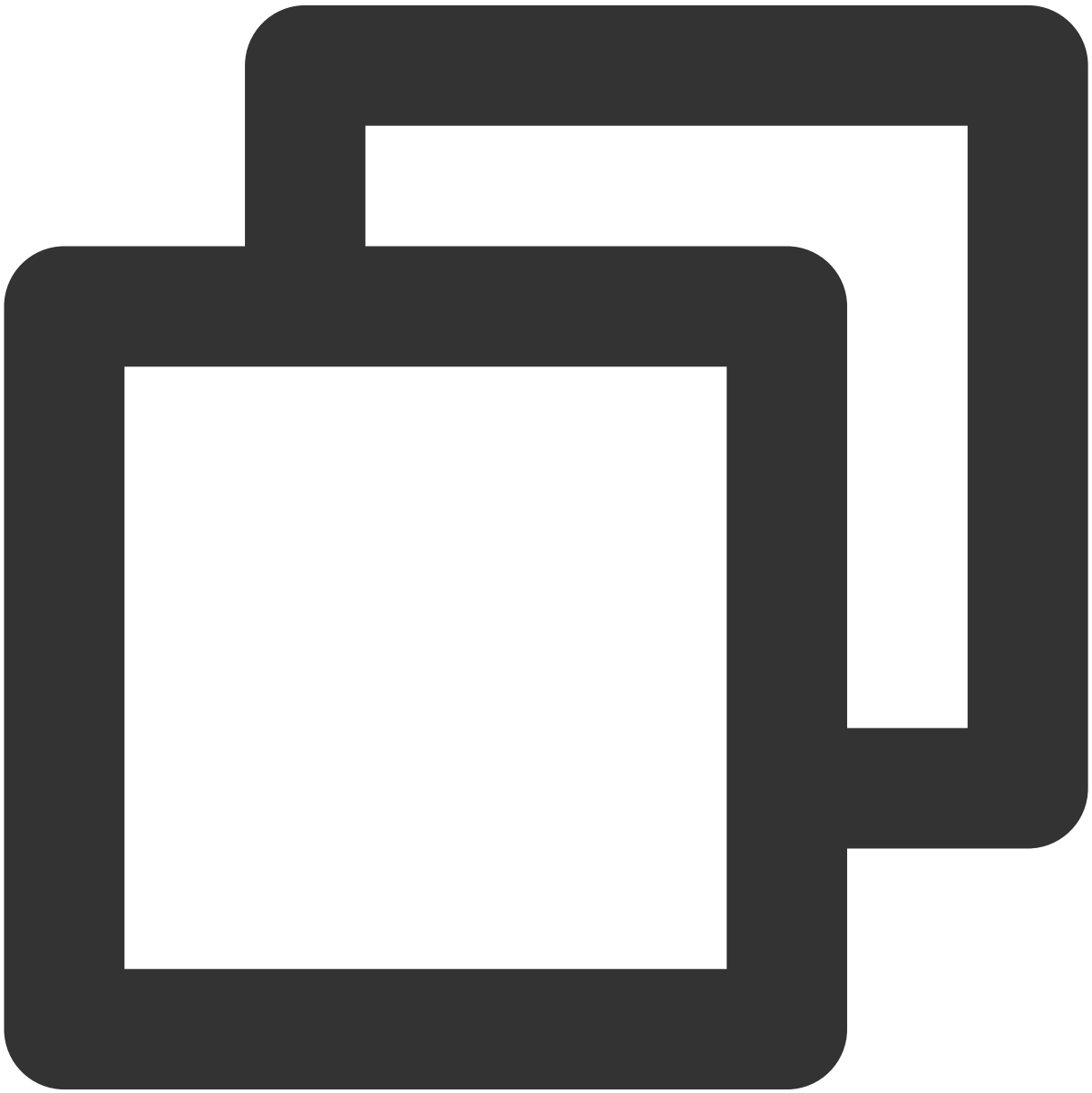


```
GOOSEFS_JAVA_OPTS+=" -XX:+PrintGCDetails -XX:+PrintTenuringDistribution -XX:+PrintG
```

`GOOSEFS_JAVA_OPTS` is the Java VM parameter for all GooseFS nodes. You can also use `GOOSEFS_MASTER_JAVA_OPTS` and `GOOSEFS_WORKER_JAVA_OPTS` to specify the VM parameter for the master and workers, respectively.

FUSE logs

You can set the log level for FUSE in the `conf/log4j.properties` file:



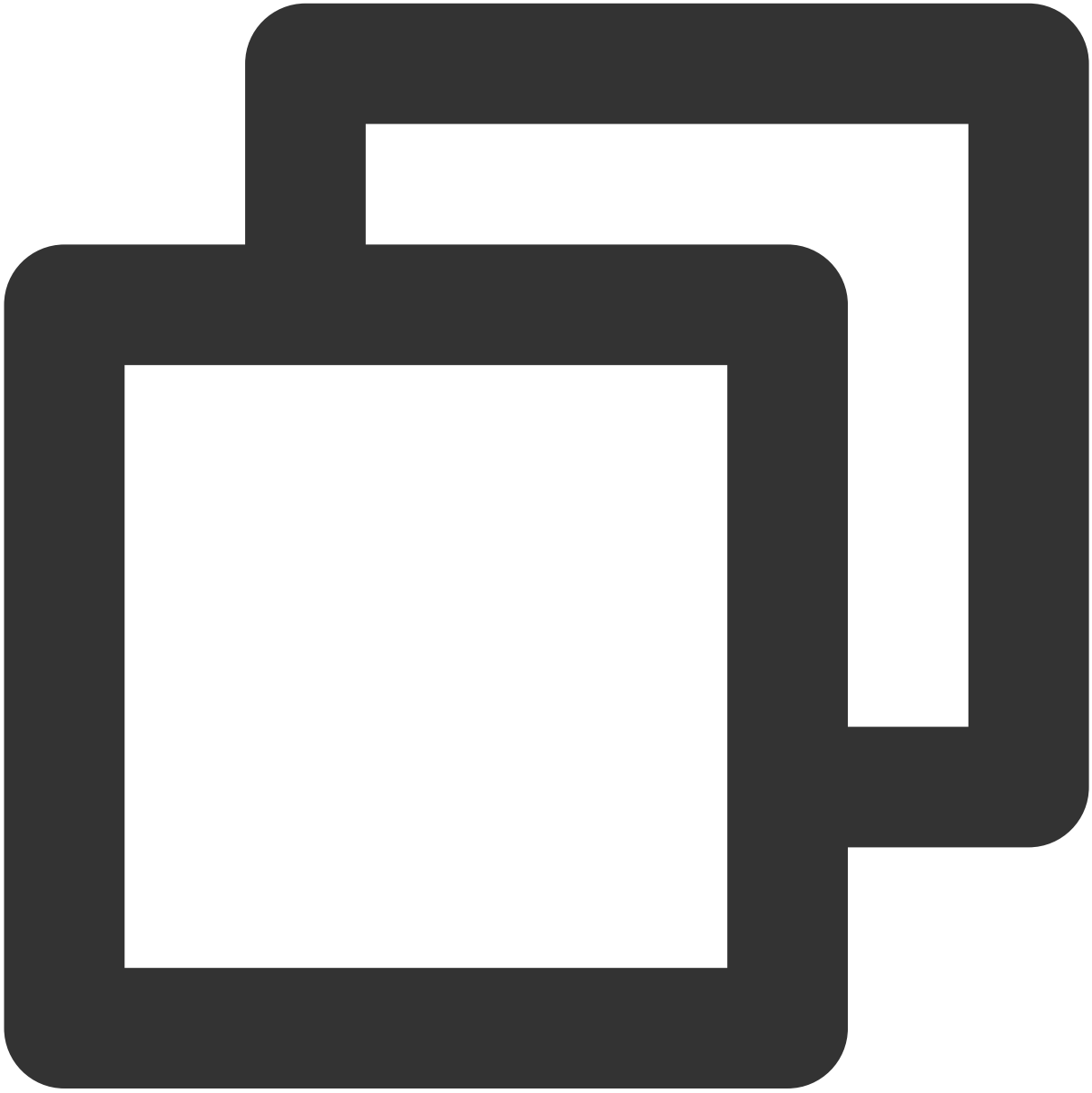
```
goosefs.logger.com.qcloud.cos.goosefs.fuse.GoosefsFuseFileSystem=DEBUG
```

After enabling it, you can view the FUSE logs in `logs/fuse.log` .

RPC logs

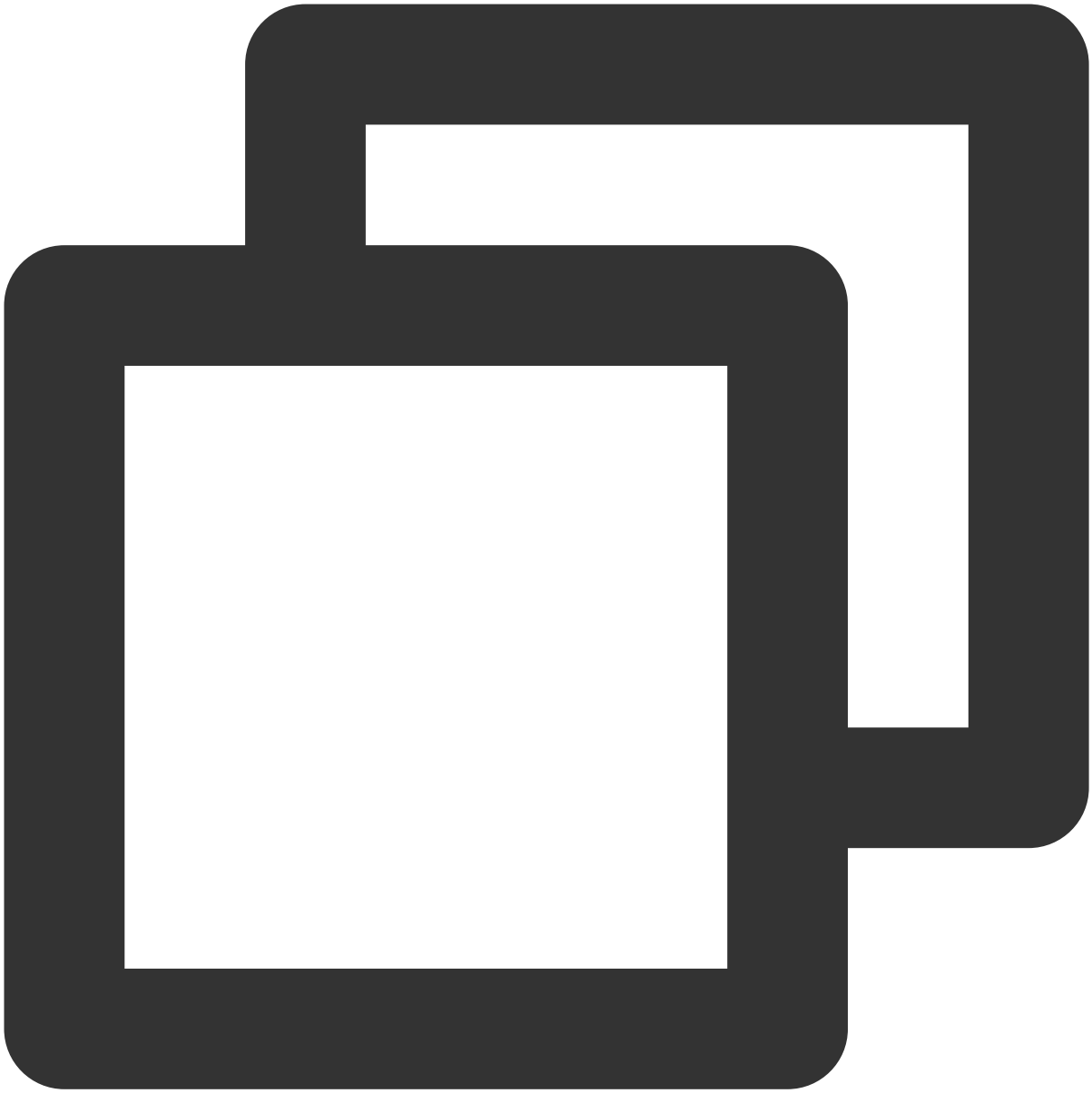
You can use the `conf/log4j.properties` file to configure the RPC logs for the client or the master.

In `log4j.properties` , configure the RPC request logs for the client:



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

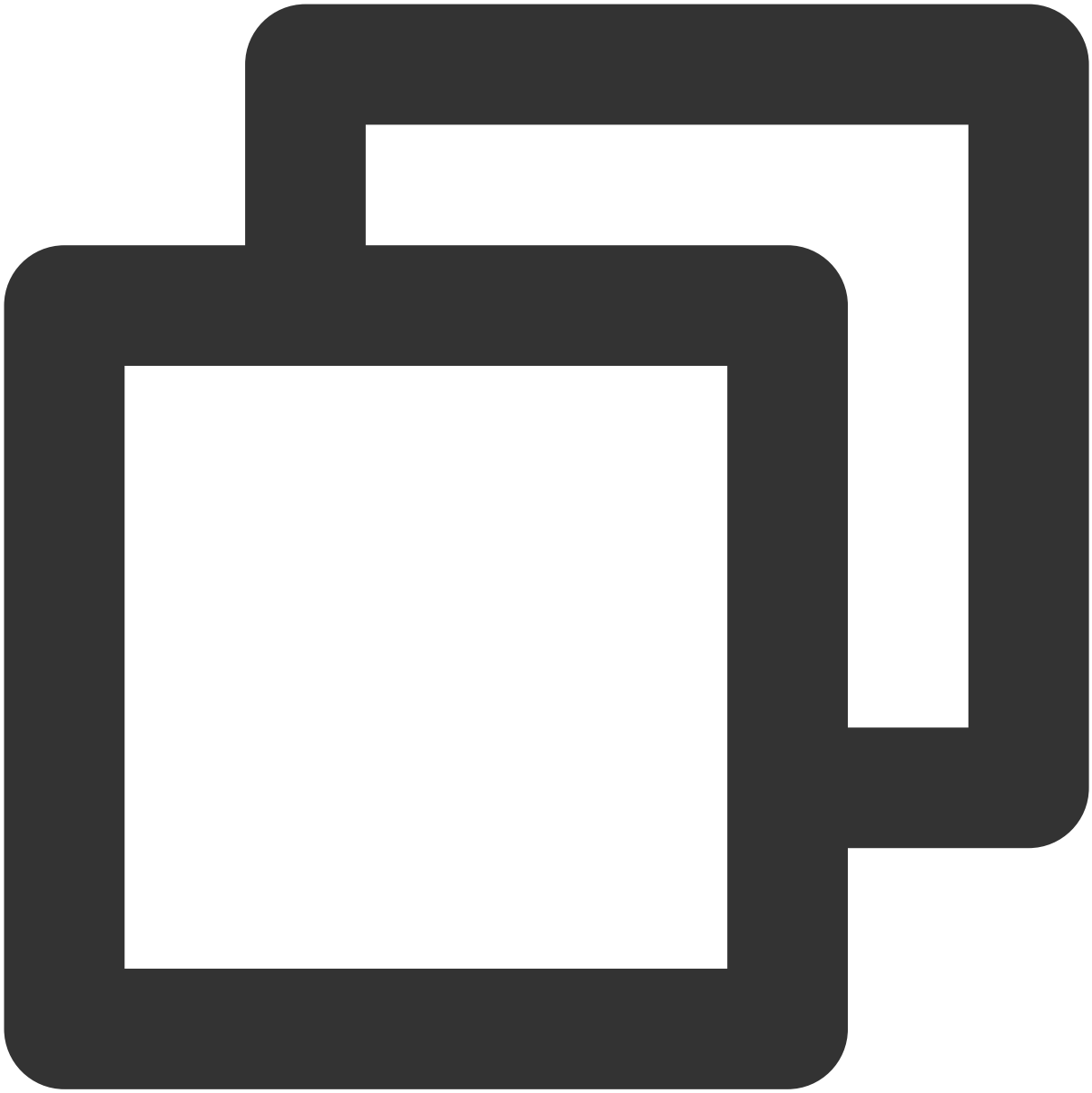
Run the `logLevel` command to configure the RPC request logs for the master:



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

UFS operation logs

To configure UFS operation logs, you can set the `log4j.properties` file. Alternatively, you can run the `logLevel` command as follows:



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

Log segmentation

GooseFS allows you to store different types of logs in different locations. If all logs are stored in the .log files, the following problems may occur:

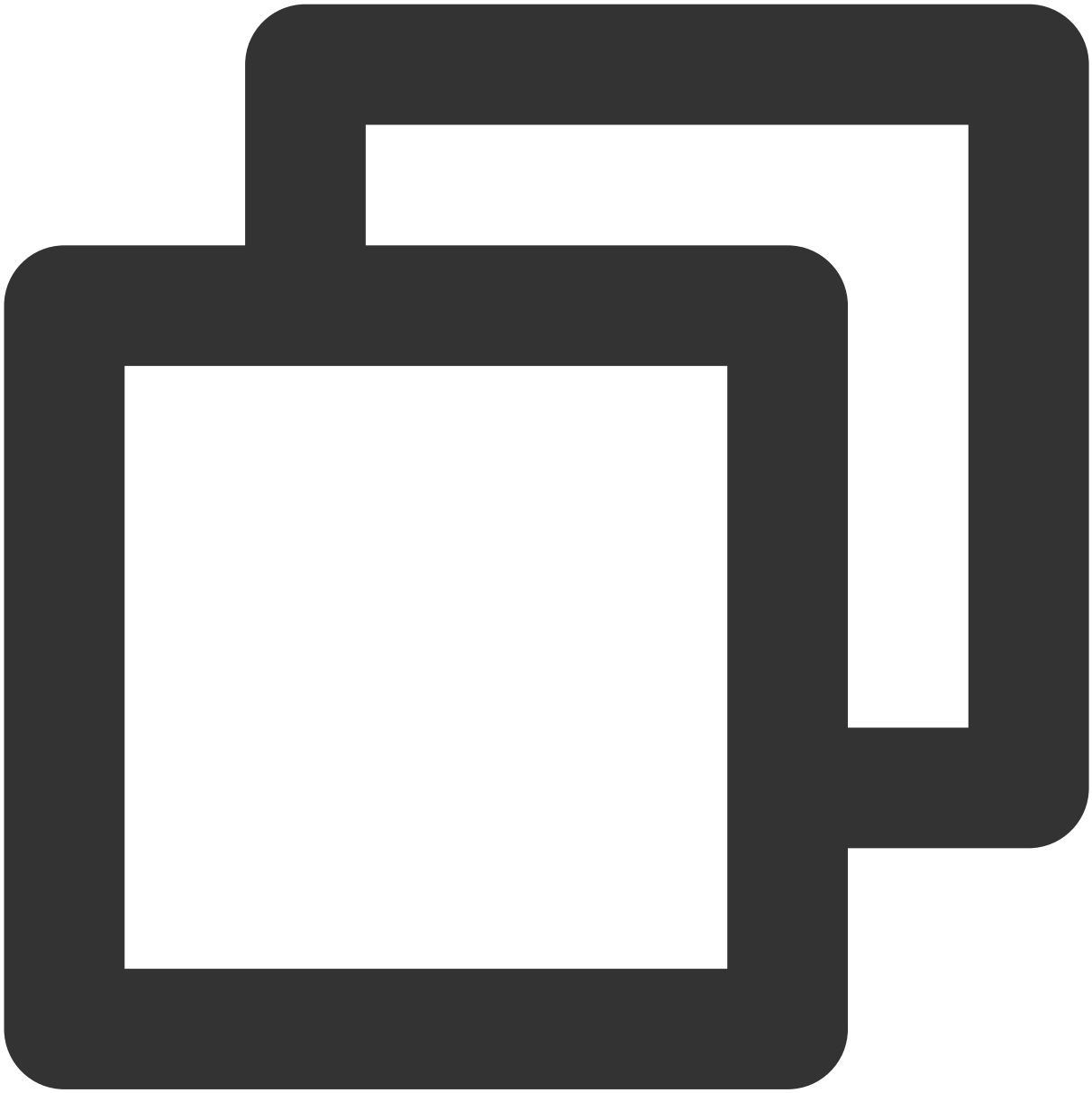
If the cluster is large or the throughput is high, `master.log` or `worker.log` may become extremely large, or lots of logs will be rolled.

Log analysis will become difficult if there are too many logs.

Lots of logs are stored in the local node and consume storage.

To solve the problems above, you can configure `log4j.properties` to set locations for specific types of logs.

The following example stores `StateLockManager` logs in `statelock.log` :

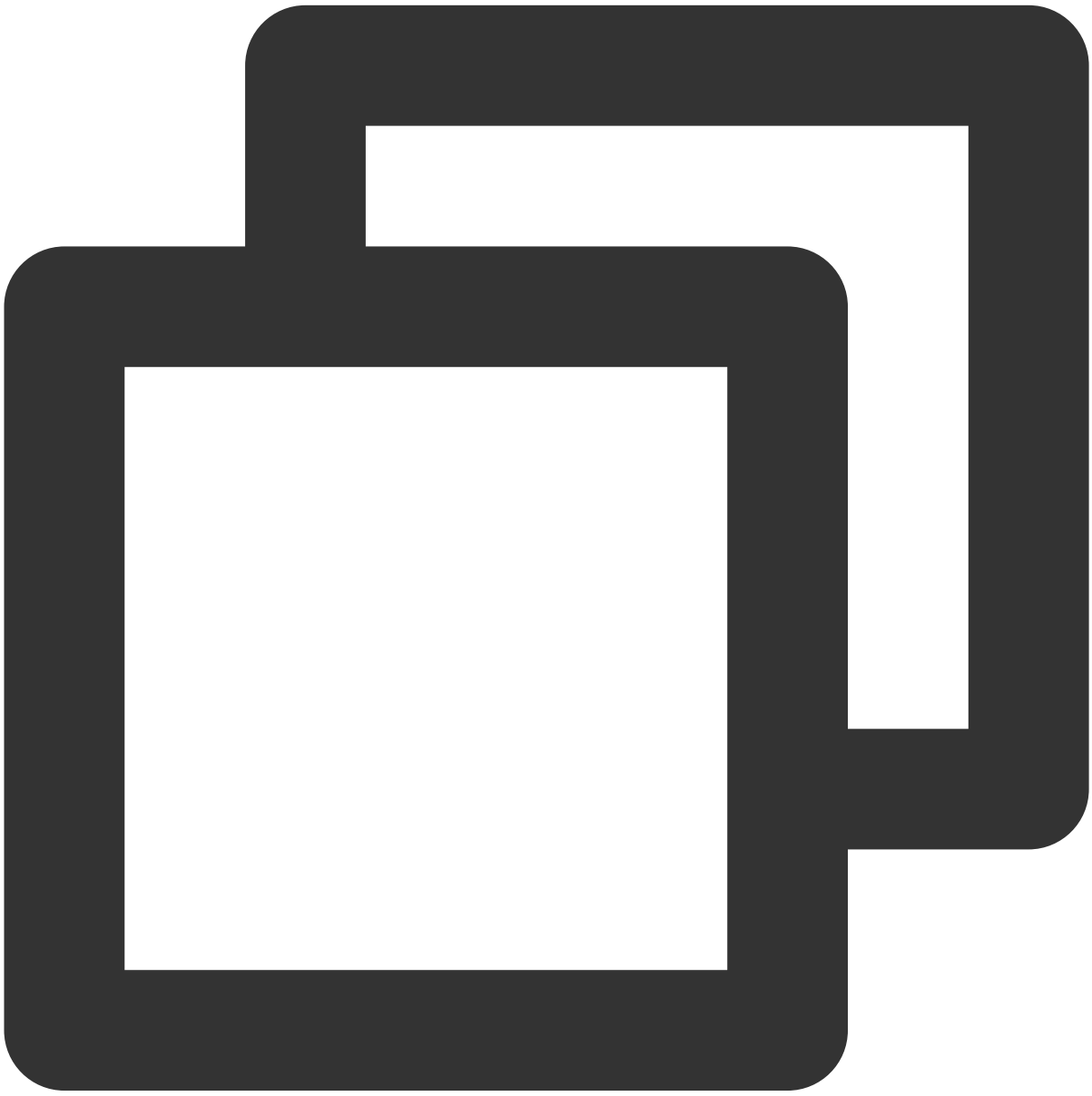


```
log4j.category.com.qcloud.cos.goosefs.master.StateLockManager=DEBUG, State_LOCK_LOG
log4j.additivity.com.qcloud.cos.goosefs.master.StateLockManager=false
log4j.appender.State_LOCK_LOGGER=org.apache.log4j.RollingFileAppender
log4j.appender.State_LOCK_LOGGER.File=<GOOSEFS_HOME>/logs/statelock.log
log4j.appender.State_LOCK_LOGGER.MaxFileSize=10MB
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} -
```

Log filtering

GooseFS allows you to set conditions to filter and record logs instead of recording all logs. For example, during performance testings, some RPC logs need to be recorded. However, not all logs but only those with high latency are needed. In this case, you can configure the `log4j.properties` file to add log filtering conditions. The following example filters logs for requests that have an RPC latency of more than 200ms and FUSE latency of over 1s:



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

Monitoring Guide
Getting GooseFS Monitoring Metrics

Last updated：2024-03-25 16:04:01

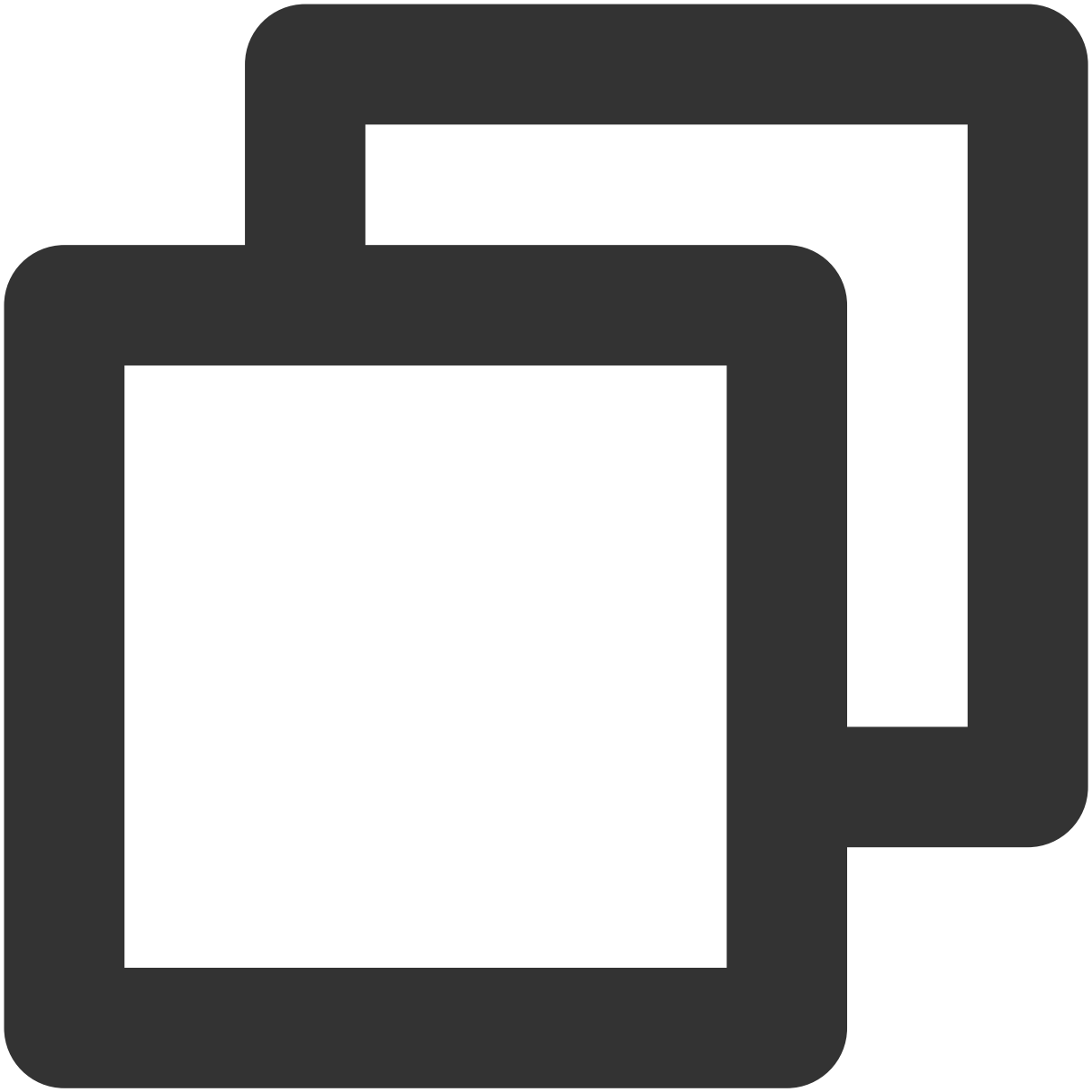
GooseFS records monitoring data based on the Coda Hale Metrics Library. It allows you to obtain metric data in different ways such as the CLI, console, and files. GooseFS currently supports the following metric obtaining methods:
MetricsServlet: monitoring metrics are provided to users in JSON format.
CsvSink: monitoring metrics are provided in CSV files. After configuration, monitoring metrics are periodically generated in CSV files.
PrometheusMetricsServlet: monitoring metrics are provided to users in the format defined by Prometheus.
The above monitoring metrics can be specified in the configuration file. The default path of the GooseFS monitoring metric configuration file is `$ GooseFS_HOME/conf/metrics.properties`. You can use `goosefs.metrics.conf.file` to specify a custom monitoring configuration file. GooseFS provides users with a default template `metrics.properties.template` that contains all configurable attributes.

Getting Monitoring Metrics

The following describes three basic methods for obtaining monitoring metrics:

1. Pulling monitoring metrics via JSON format

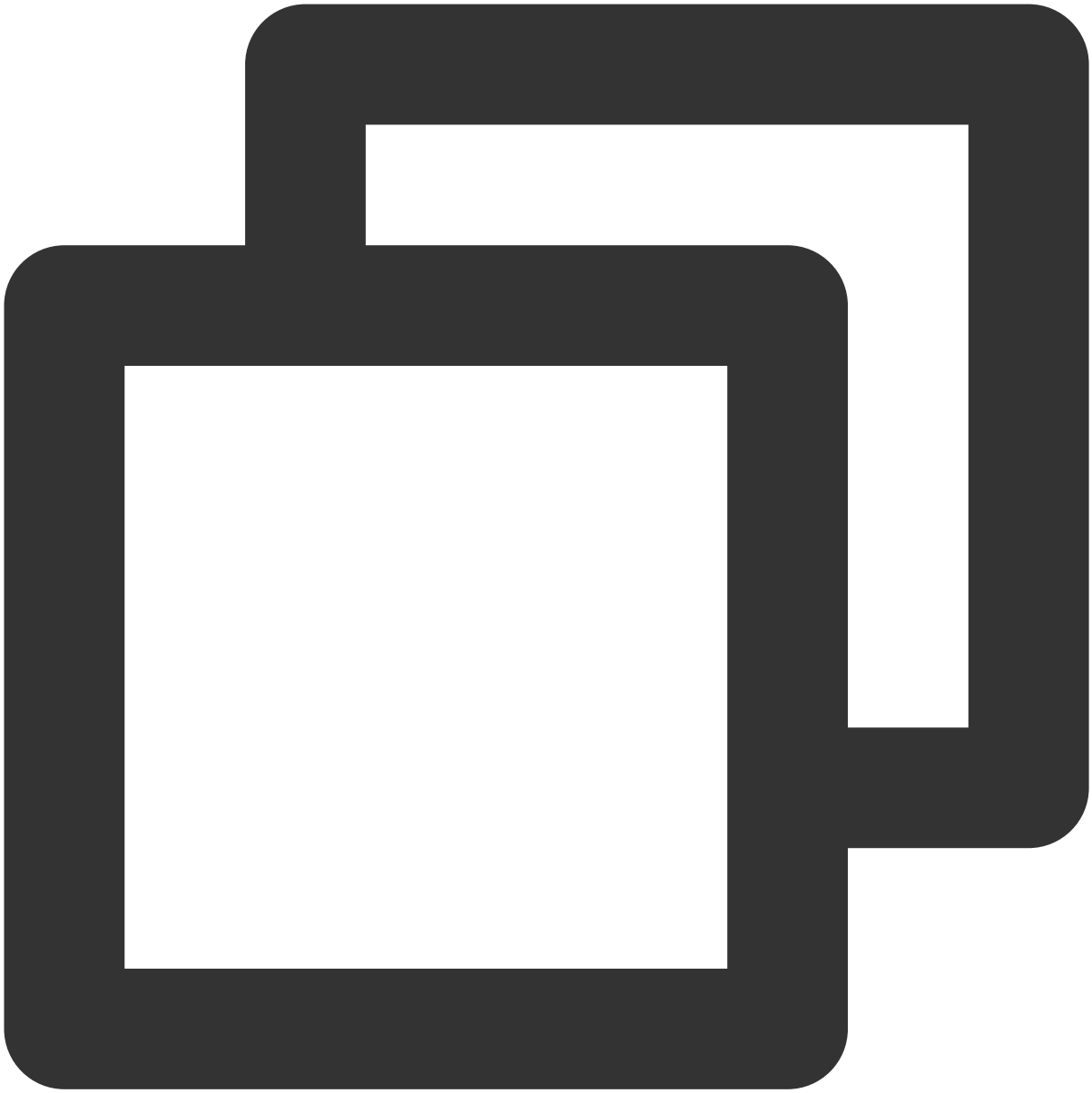
GooseFS adopts the method of pulling monitoring metrics in JSON format by default, which corresponds to the MetricsServlet configuration item. You can use the CLI to initiate an HTTP request to the leading master node of GooseFS to pull all desired monitoring metrics. The metrics port is 9201 for the master node and 9204 for a worker node. The request format is as follows:



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

In this format, <LEADING_MASTER_HOSTNAME> must be a valid master node IP address, and <MASTER_WEB_PORT> must be an enabled port.

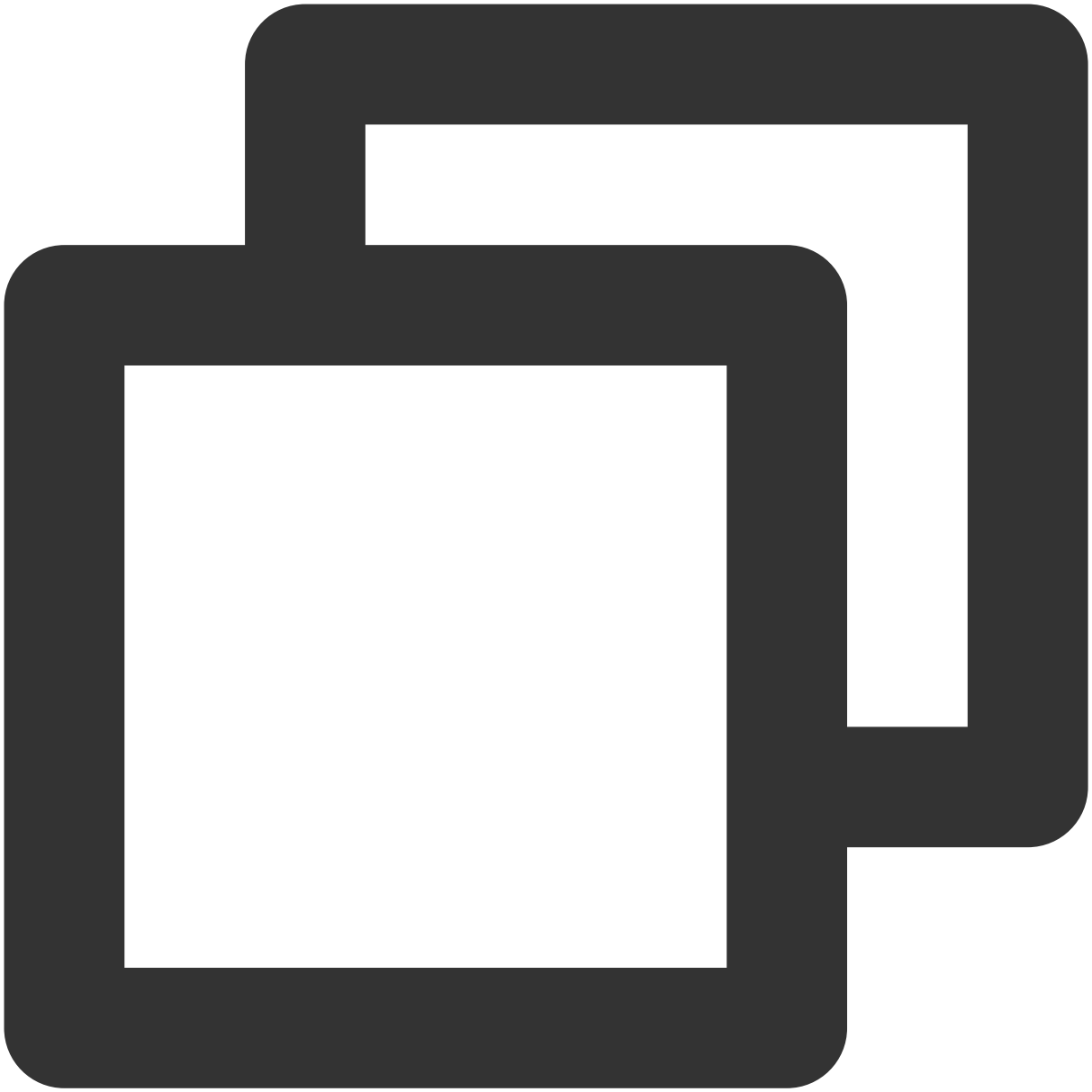
To get the monitoring metrics of a specific worker node, use the following command:



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

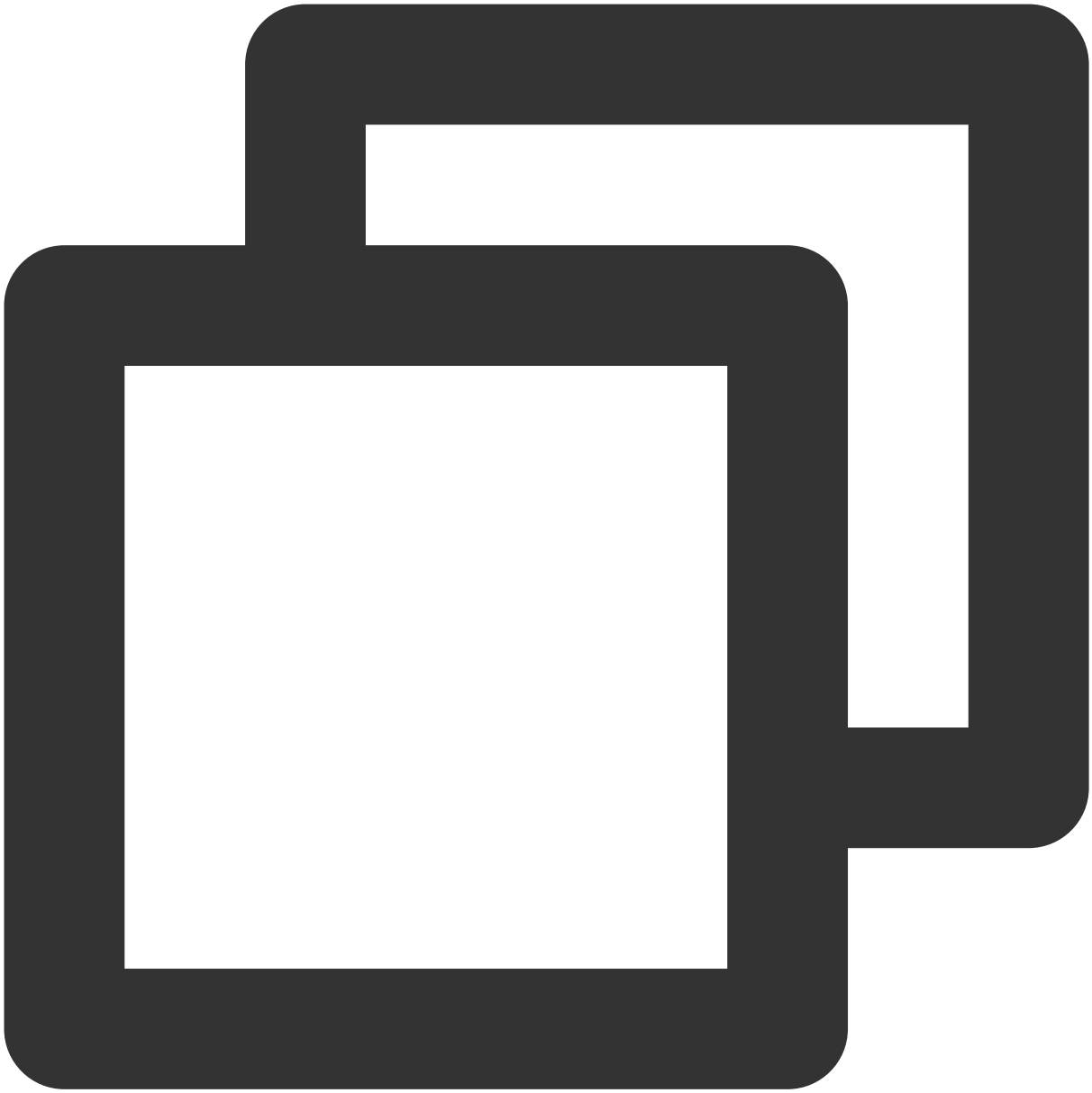
2. Getting monitoring metrics via CSV files

GooseFS allows you to export data to CSV files. You can use this capability to get monitoring metrics. First, you need to prepare a directory to store monitoring metrics:



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

After the storage path is prepared, modify the `conf/metrics.properties` configuration file to enable the CsvSink capability:



```
sink.csv.class=goosefs.metrics.sink.CsvSink # Enable the CsvSink capability

sink.csv.period=1 # Set the monitoring metric export period
sink.csv.unit=senconds # Set the unit of the monitoring metric export period

sink.csv.directory=/tmp/goosefs-metrics # Set the monitoring metric export path
```

After the configuration, you need to restart the node for the configuration to take effect. After the configuration takes effect, the monitoring metrics will be periodically exported to CSV files and stored to the specified path.

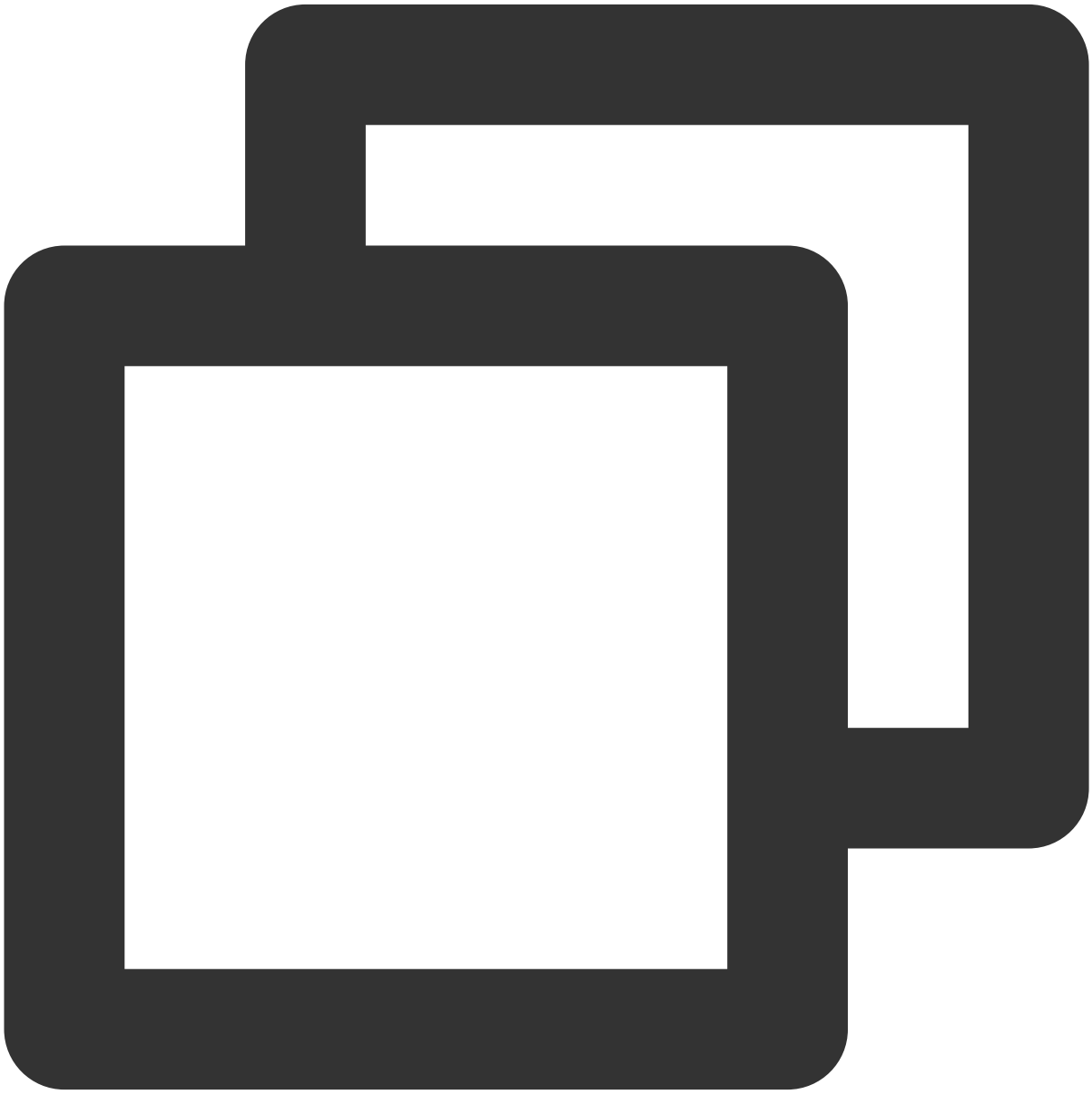
Note:

GooseFS provides a monitoring configuration template. For more information, see the `conf/metrics.properties.template` file.

If GooseFS is deployed on a cluster, ensure that the specified metric storage path can be read by all nodes.

3. Pulling Prometheus monitoring metrics

You can run the following commands to view Prometheus monitoring metrics on the GooseFS master node (metrics port: 9201) and worker node (metrics port: 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
...
```

Monitoring GooseFS Based on Prometheus

Last updated：2024-03-25 16:04:01

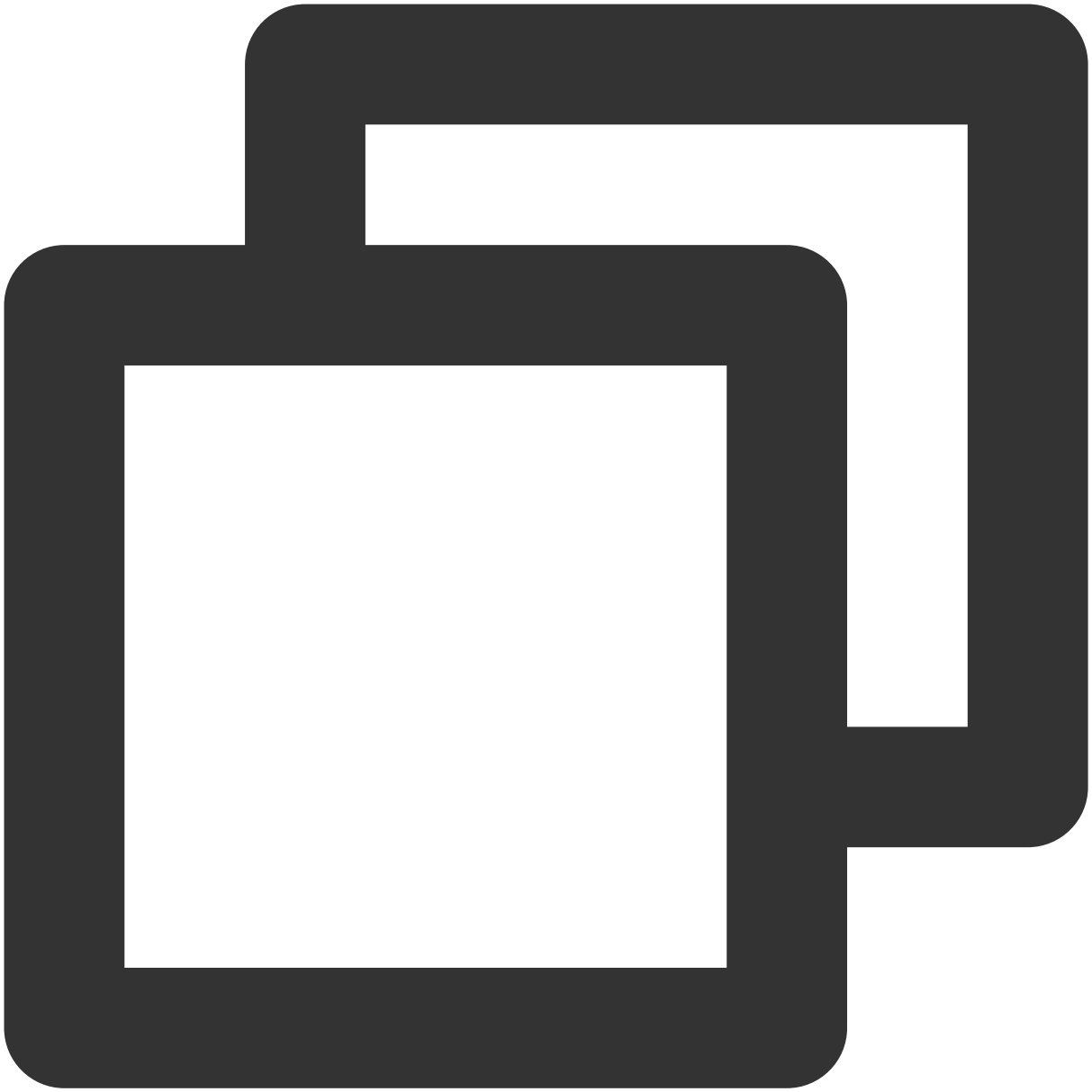
GooseFS allows you to output metrics to different monitoring systems, including Prometheus. Prometheus is an open-source monitoring framework that Tencent Cloud Observability Platform has integrated with. This document introduces GooseFS metrics and how to report them to the self-built/in-cloud Prometheus.

Preparations

Before setting up the Prometheus monitoring system, you need to:
Configure a GooseFS cluster.
Download the Prometheus installation package (official or Tencent Cloud version).
Download and configure Grafana.

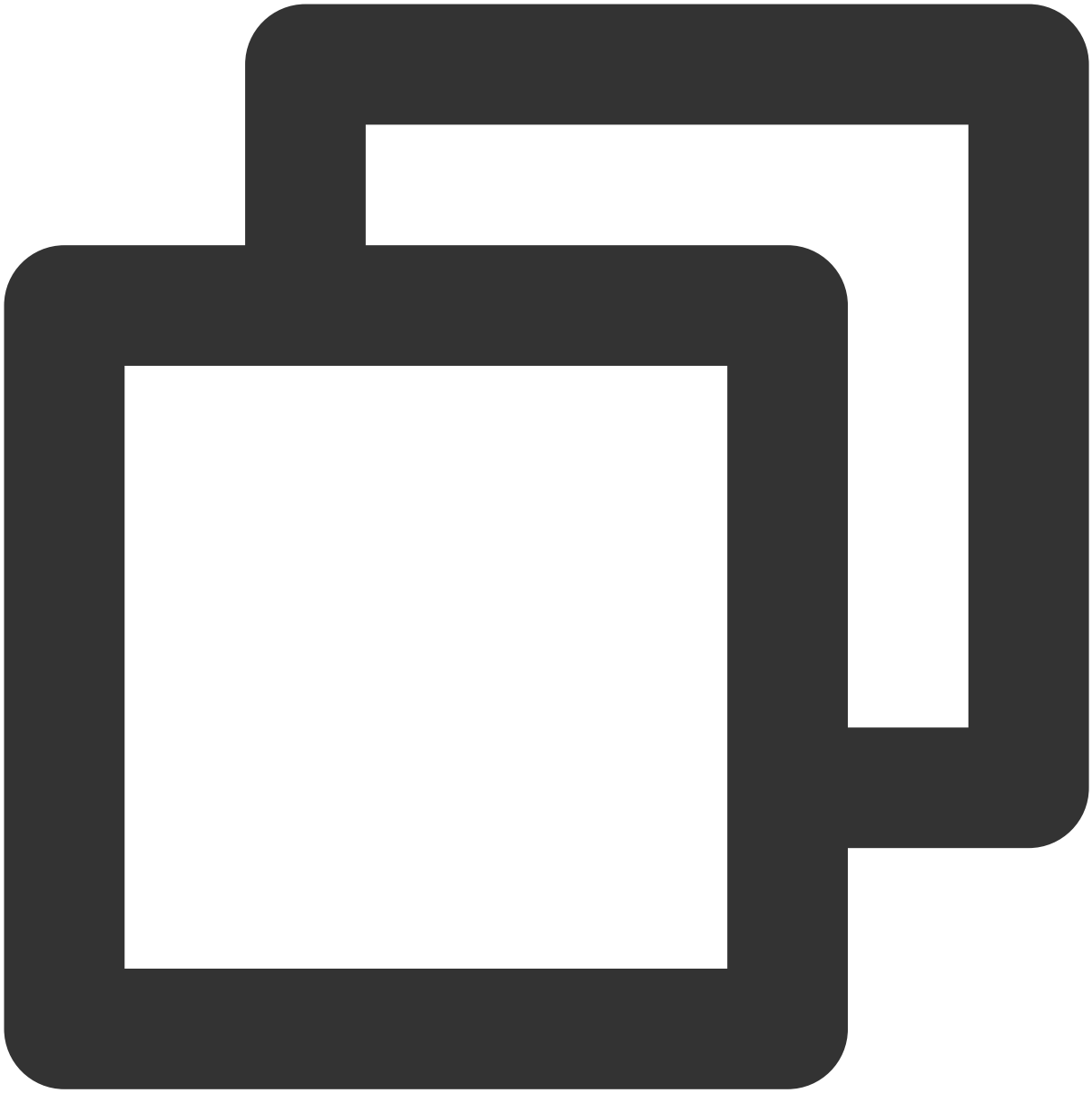
Enabling GooseFS Metric Report

1. Edit the GooseFS configuration file `conf/goosefs-site.properties` by adding the configuration items below. Then, use `goosefs copyDir conf/` to copy all worker nodes and use `./bin/goosefs-start.sh all` to restart the cluster.



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

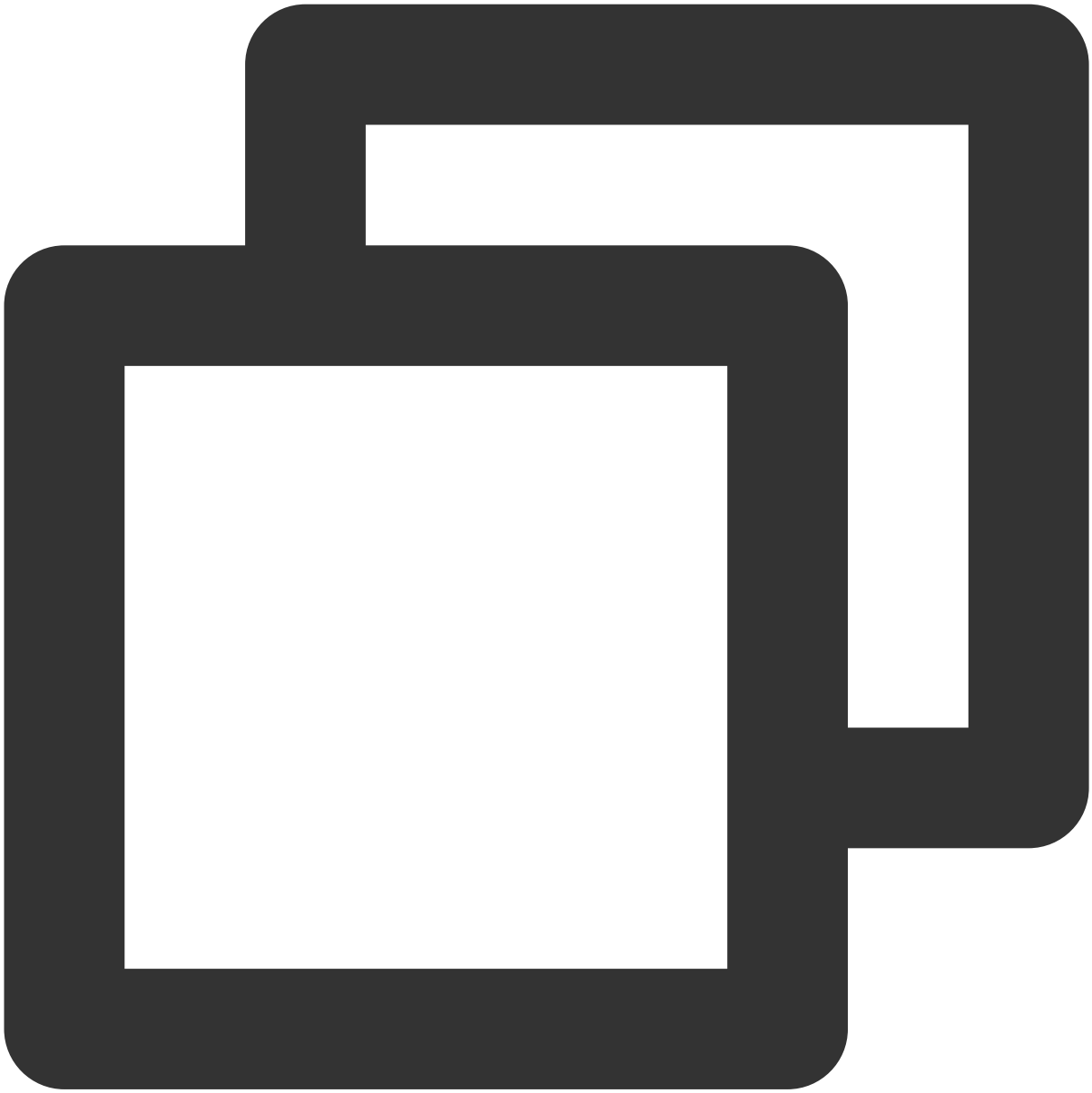
2. You can run the following command to view the Prometheus master metrics (port: 9201) and worker metrics (port: 9204):



```
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
...
```

Reporting Metrics to Self-Built Prometheus

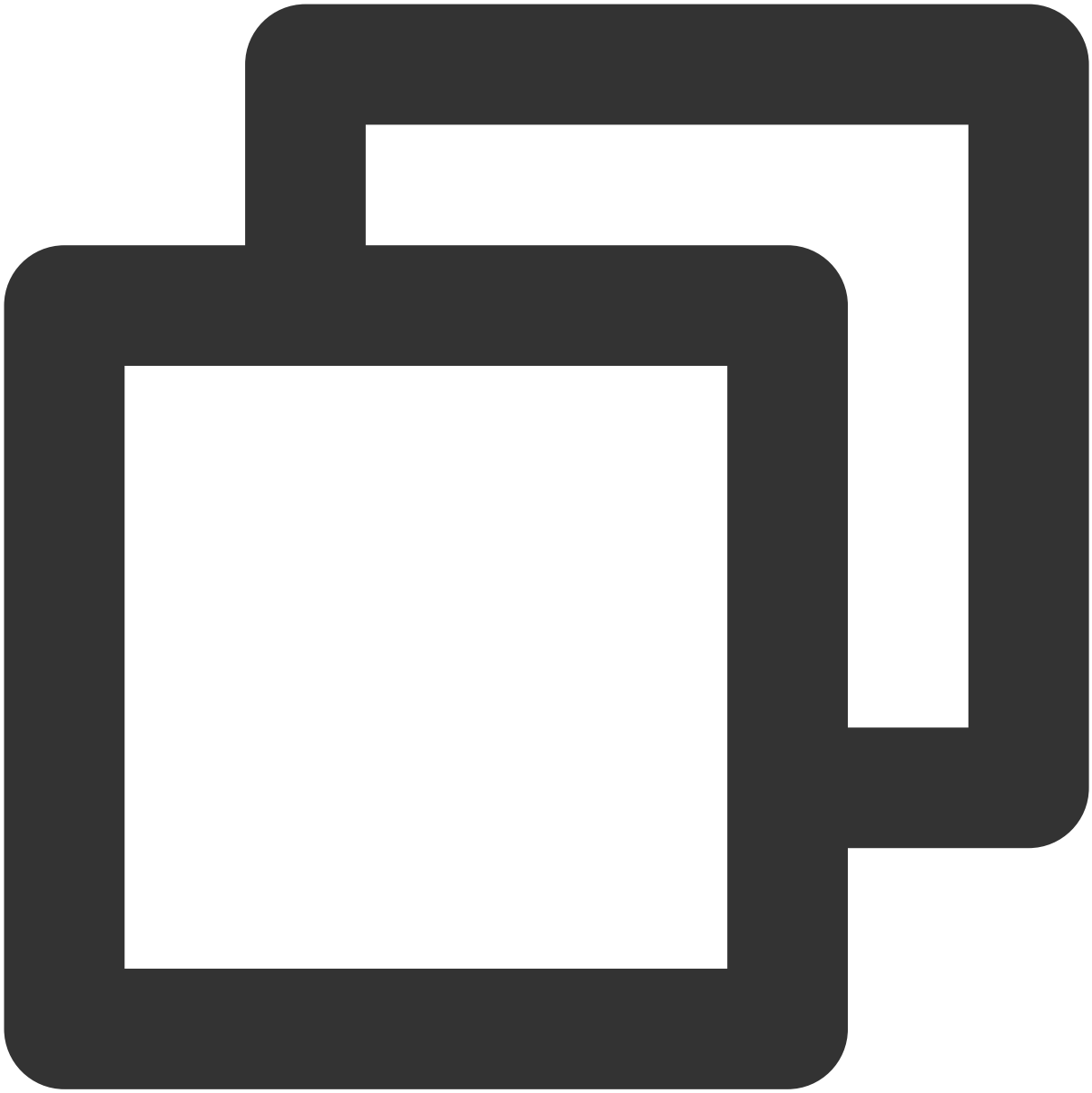
1. Download the Prometheus installation package, decompress it, and modify `promethus.yml` as follows:



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

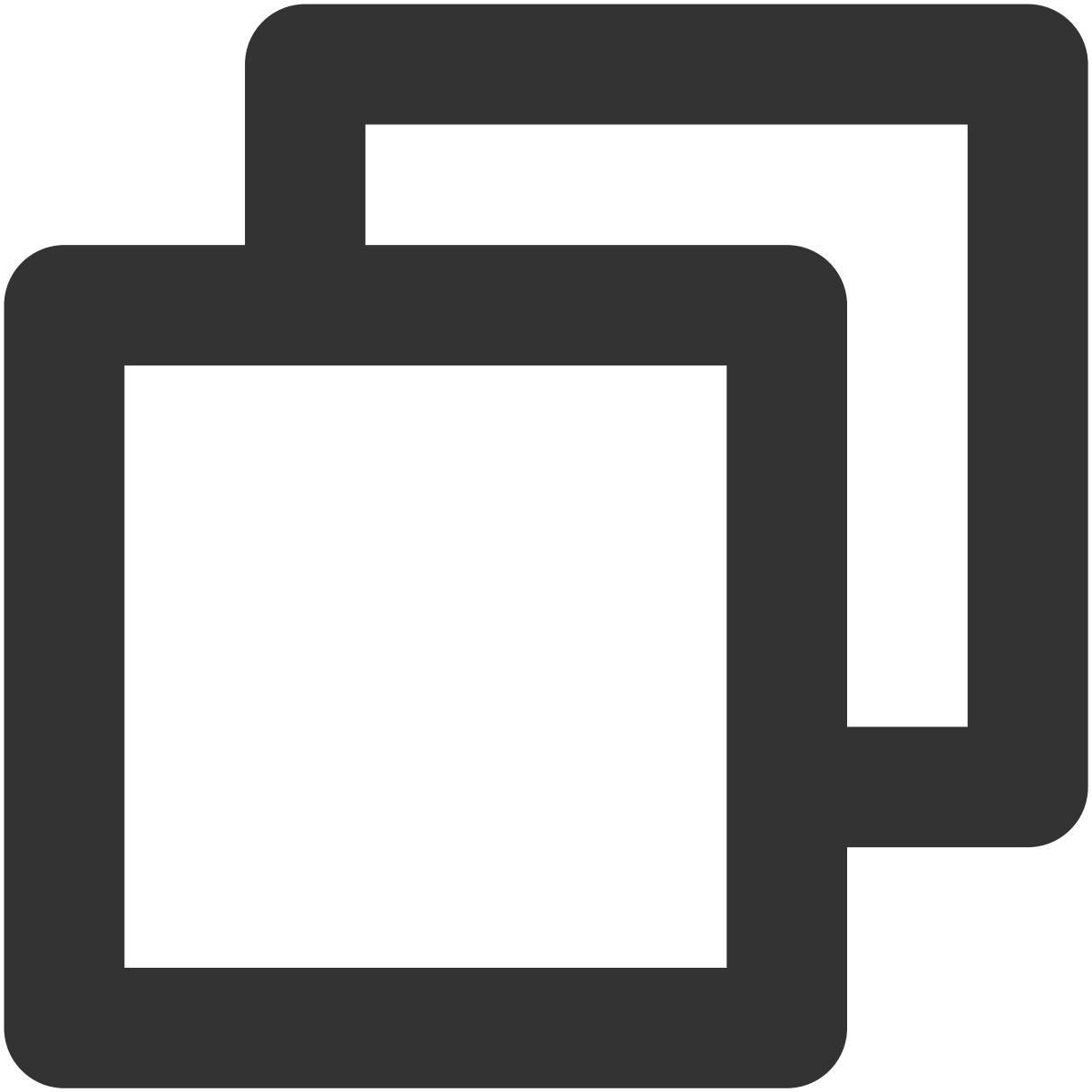
```
    file_sd_configs:
    - refresh_interval: 1m
    files:
    - "targets/cluster/masters/*.yml"
- job_name: 'goosefs workers'
    metrics_path: /metrics/prometheus
    file_sd_configs:
    - refresh_interval: 1m
    files:
    - "targets/cluster/workers/*.yml"
```

2. Create `targets/cluster/masters/masters.yml` and add the IP and port of the master:



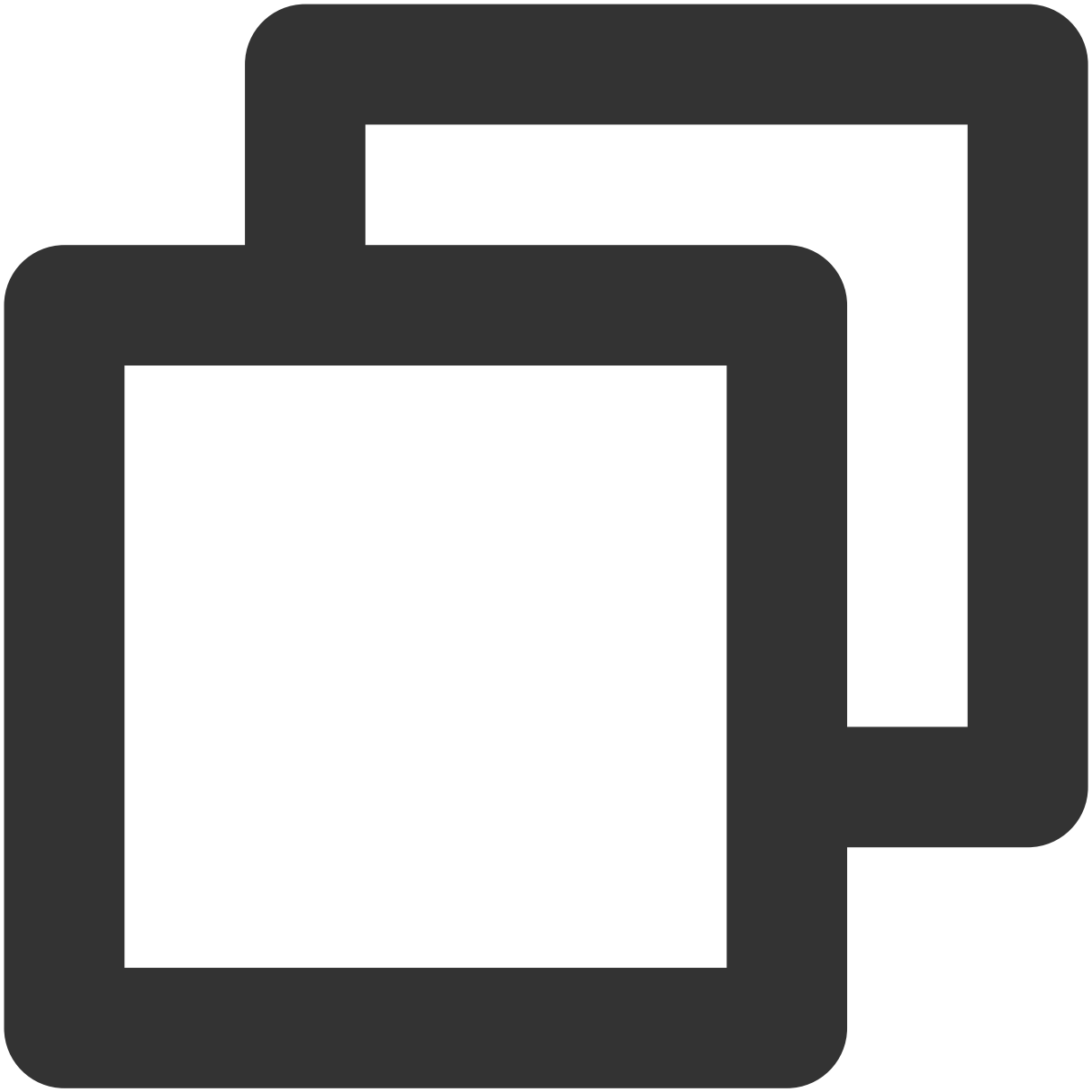
```
- targets:
- "&lt;TARGERTS_MASTER_IP>:&lt;TARGERTS_MASTER_PORT>"
```

3. Create `targets/cluster/workers/workers.yml` and add the IP and port of the worker:



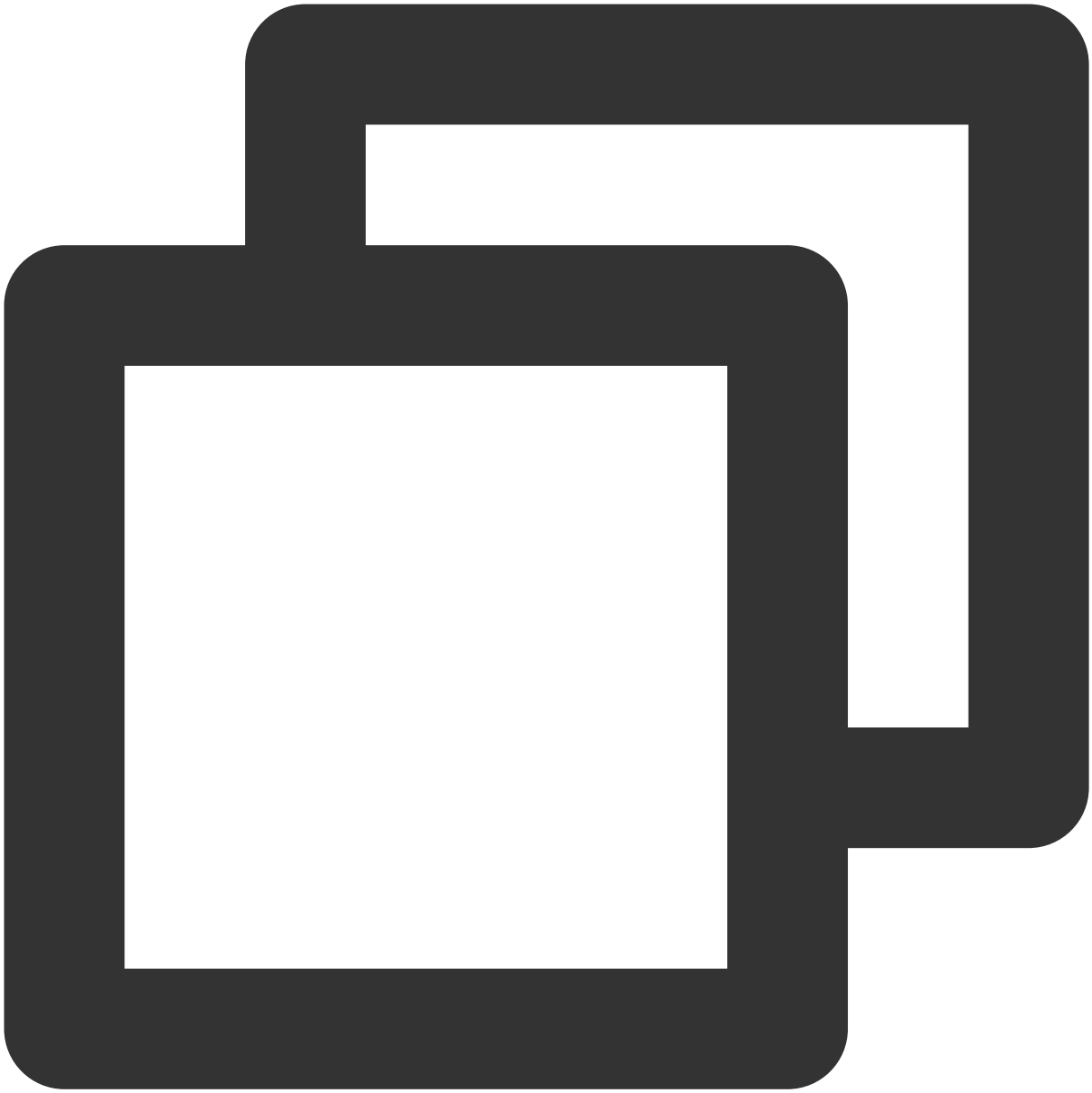
```
- targets:
- "&lt;TARGERTS_WORKER_IP>:&lt;TARGERTS_WORKER_PORT>"
```

4. Start Prometheus. `--web.listen-address` specifies the Prometheus listen address. The default port number is 9090.



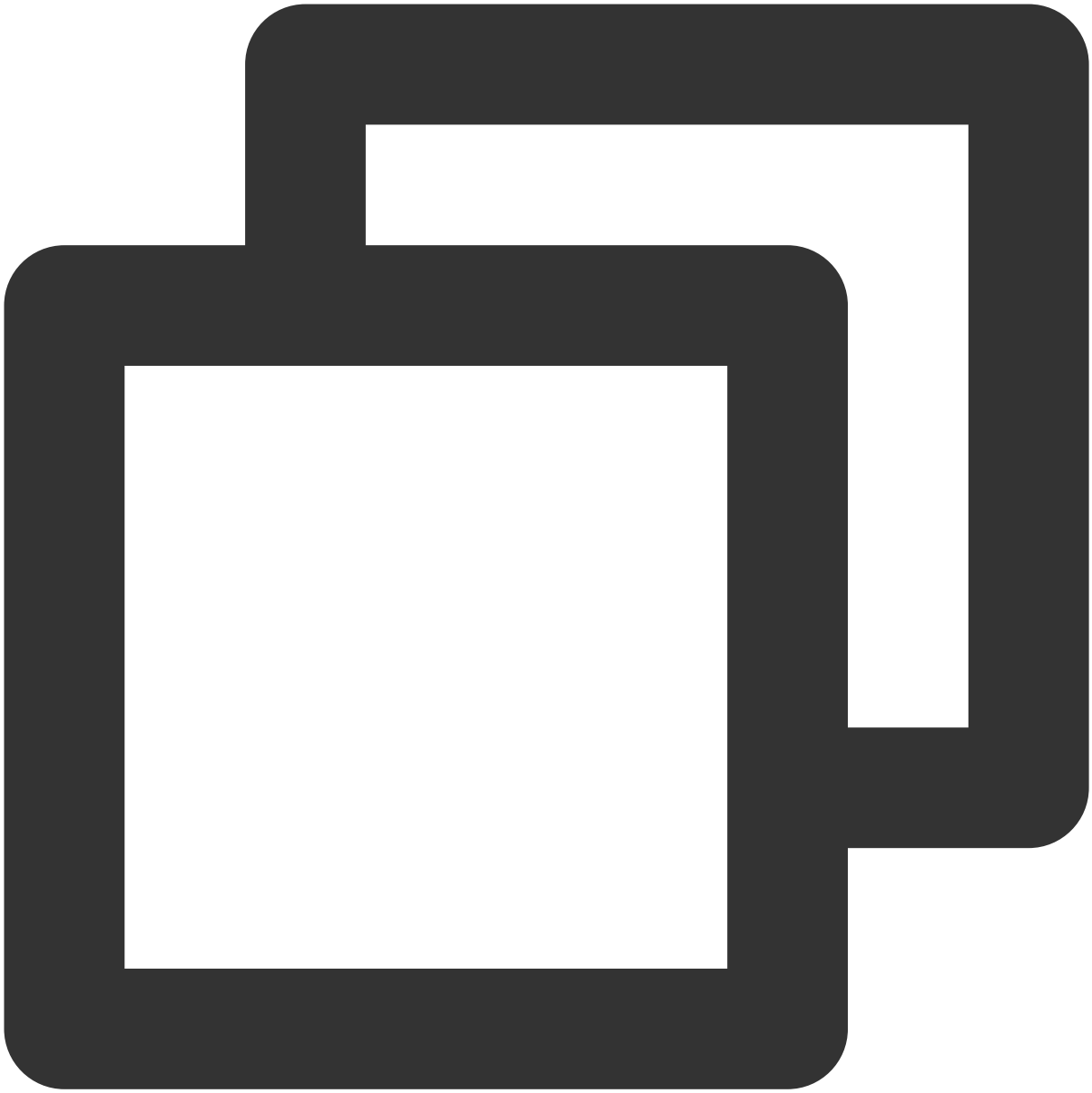
```
nohup ./prometheus --config.file=prometheus.yml --web.listen-address="<LISTEN_IP>
```

5. View the GUI:



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

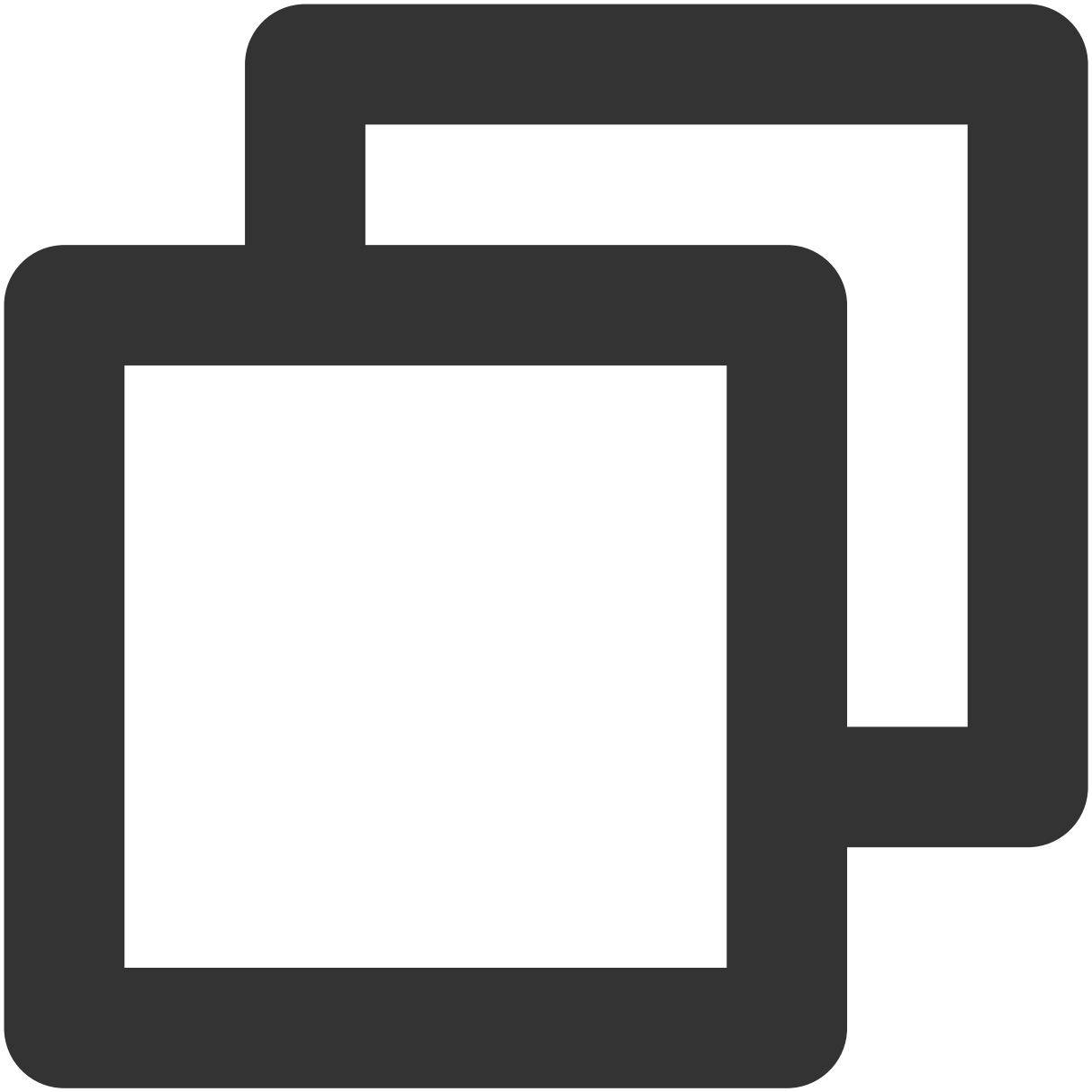
6. View the target instance:



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

Reporting Metrics to Tencent Cloud Prometheus

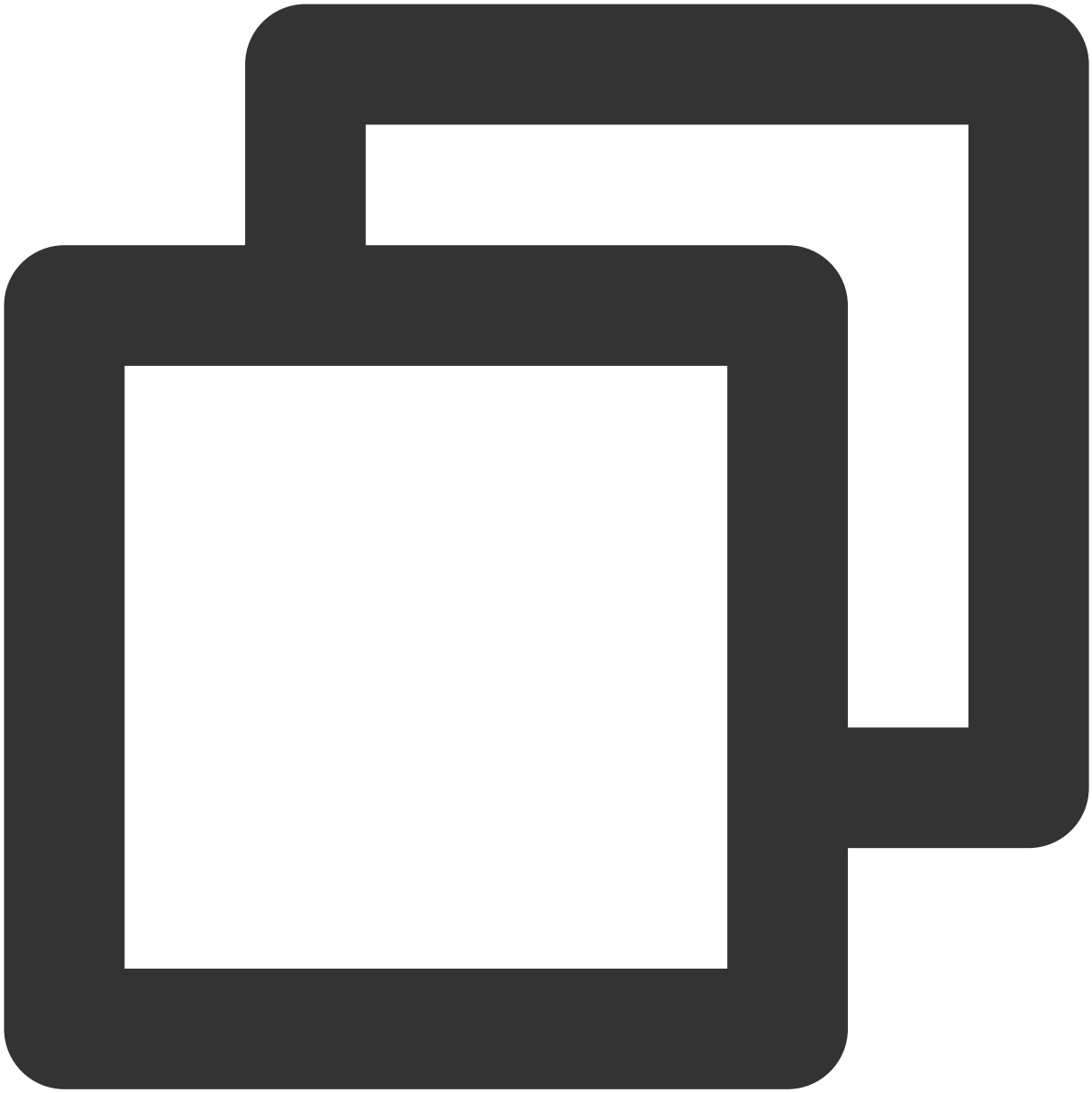
1. Install the Prometheus agent on the master as instructed in the installation guide.



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

2. Configure jobs for the master and worker:

Method 1:



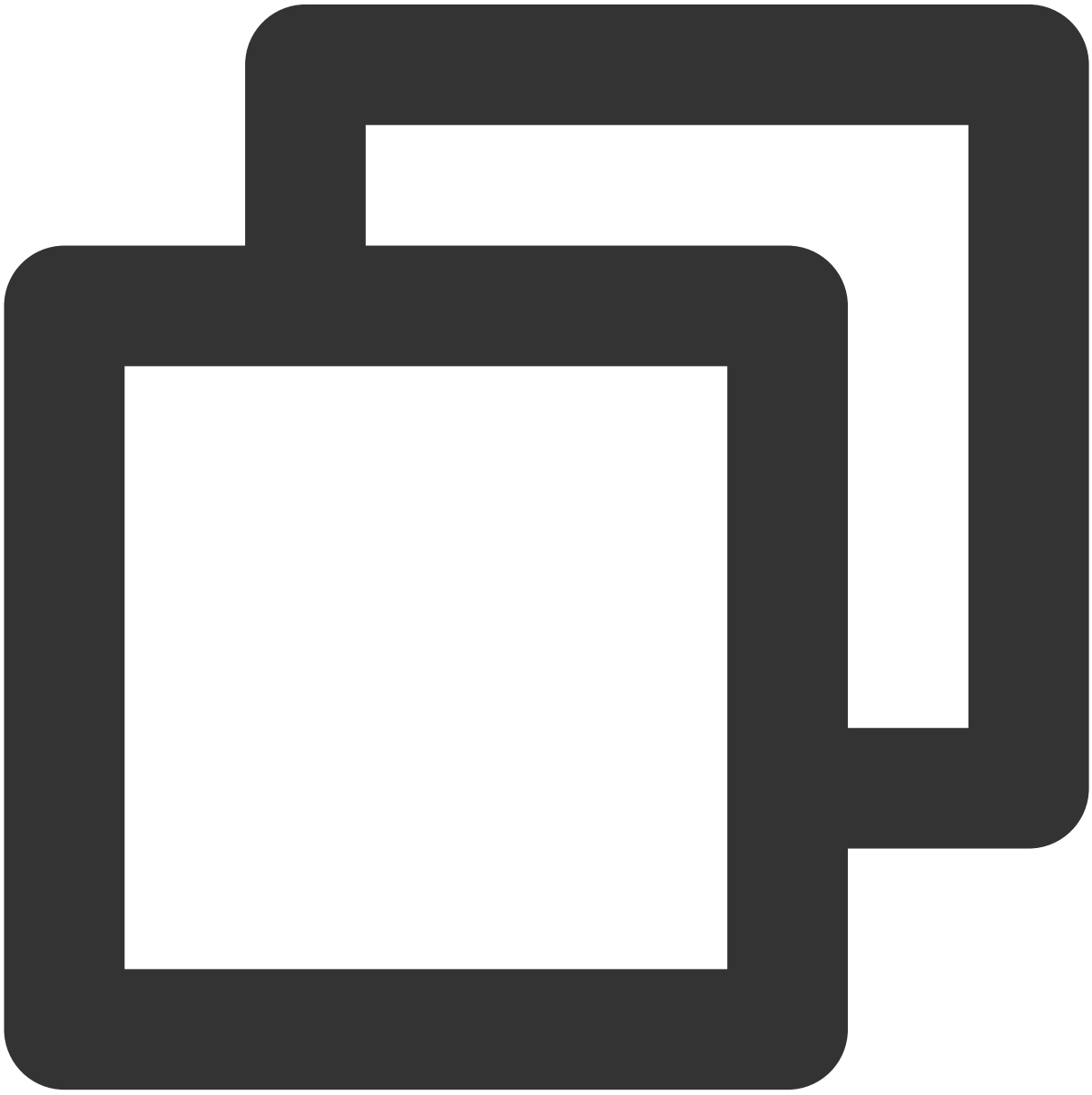
```
job_name: goosefs-masters
honor_timestamps: true
metrics_path: /metrics/prometheus
scheme: http
file_sd_configs:
- files:
  - /usr/local/services/prometheus/targets/cluster/masters/*.yml
  refresh_interval: 1m
job_name: goosefs-workers
honor_timestamps: true
metrics_path: /metrics/prometheus
```

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

Note:

Do not use spaces in the value of `job_name`. However, the value of `job_name` in single-machine Prometheus can contain spaces.

Method 2:



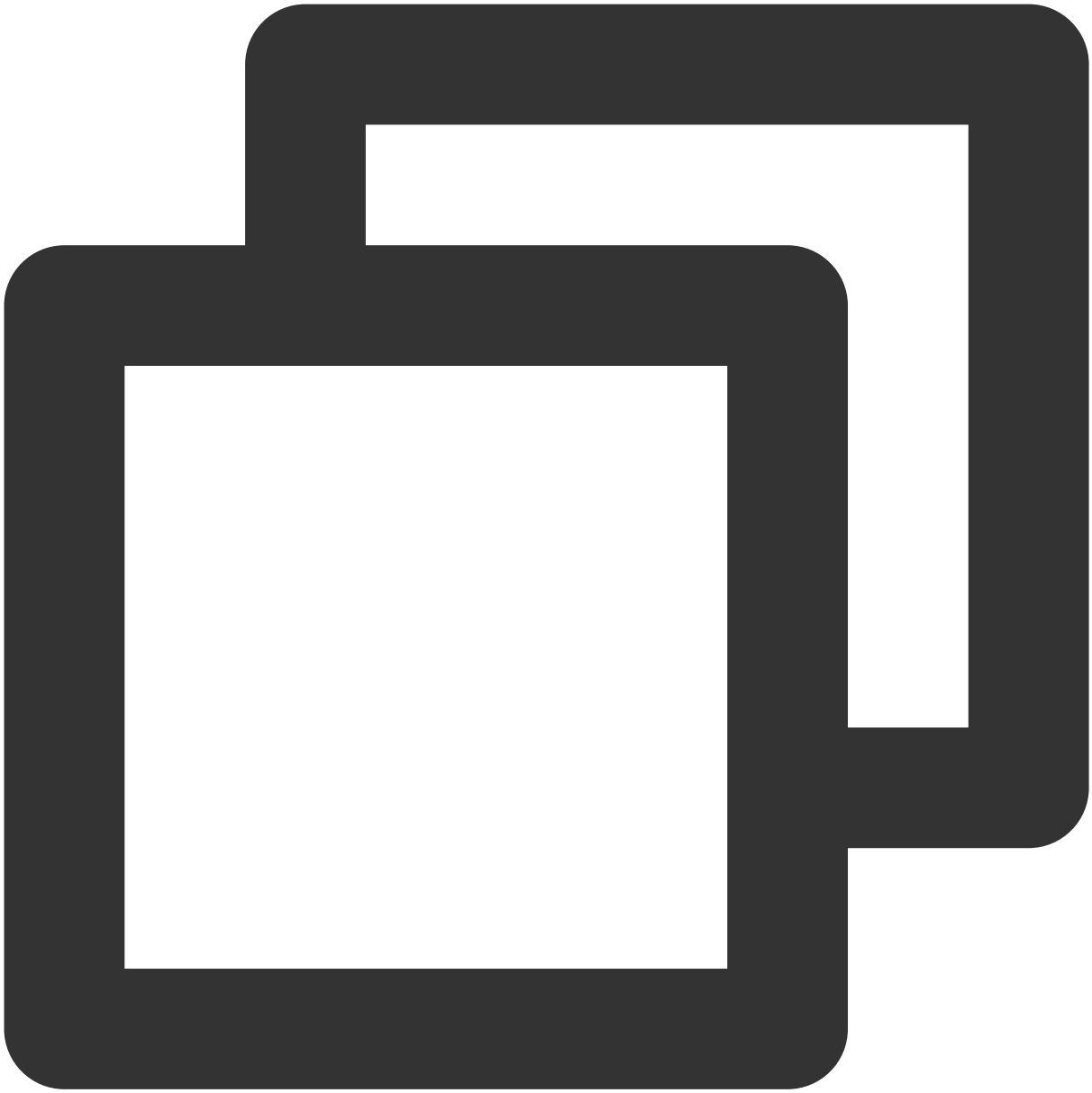
```
job_name: goosefs masters
honor_timestamps: true
metrics_path: /metrics/prometheus
scheme: http
static_configs:
- targets:
 - "<TARGERTS_MASTER_IP>:<TARGERTS_MASTER_PORT>"
 refresh_interval: 1m

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
```

Note:

If you use method 2 to configure jobs, you don't need to create the `masters.yml` and `workers.yml` files under `targets/cluster/masters/` .

Using Grafana to View Metrics

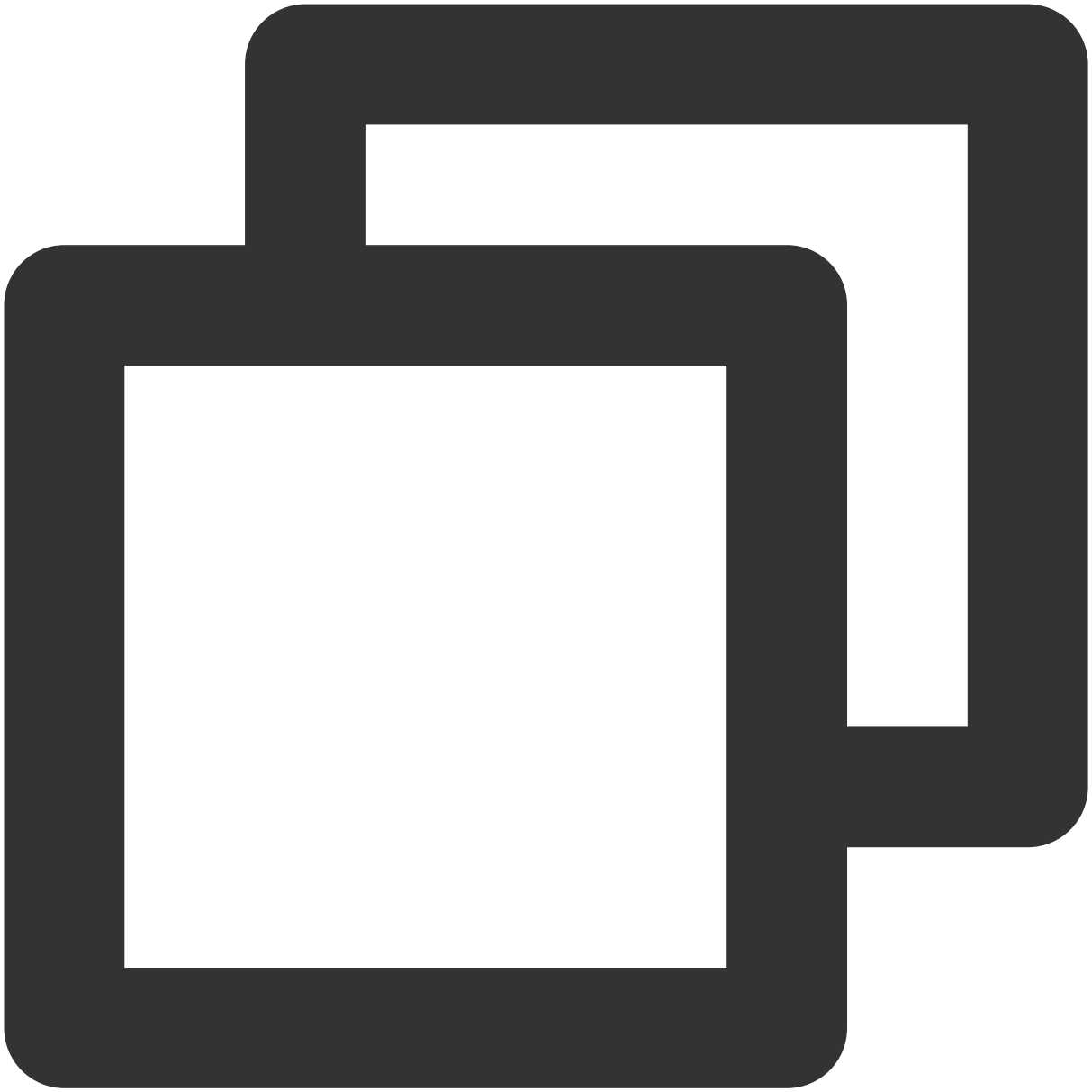
1. Start Grafana.



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

2. Open the login page `http://<GRAFANA_IP>:<GRAFANA_PORT>` . By default, Grafana will be listening on port 3000, and both the “username” and “password” are “admin”. You can change the password after your initial login.

3. Create a Prometheus data source.



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

4. Import Grafana dashboards for GooseFS. Select JSON import (Download JSON) and use it for the data source created above.

Note:

You need to set the password when purchasing the in-cloud Prometheus. The configuration of the in-cloud Grafana monitoring GUI is similar to that mentioned above. Note that the configuration of `job_name` should be consistent.

5. You can export a modified dashboard.

Cluster Configuration Practices

Last updated：2024-03-25 16:04:01

The GooseFS data accelerator provides multiple deployment modes. This document describes the cluster deployment modes commonly used in big data scenarios:

Production environment configuration practice in AI scenarios

Production environment configuration practice in big data scenarios

Data Security
Using Apache Ranger to Manage GooseFS Access Permissions

Last updated：2024-03-25 16:04:01

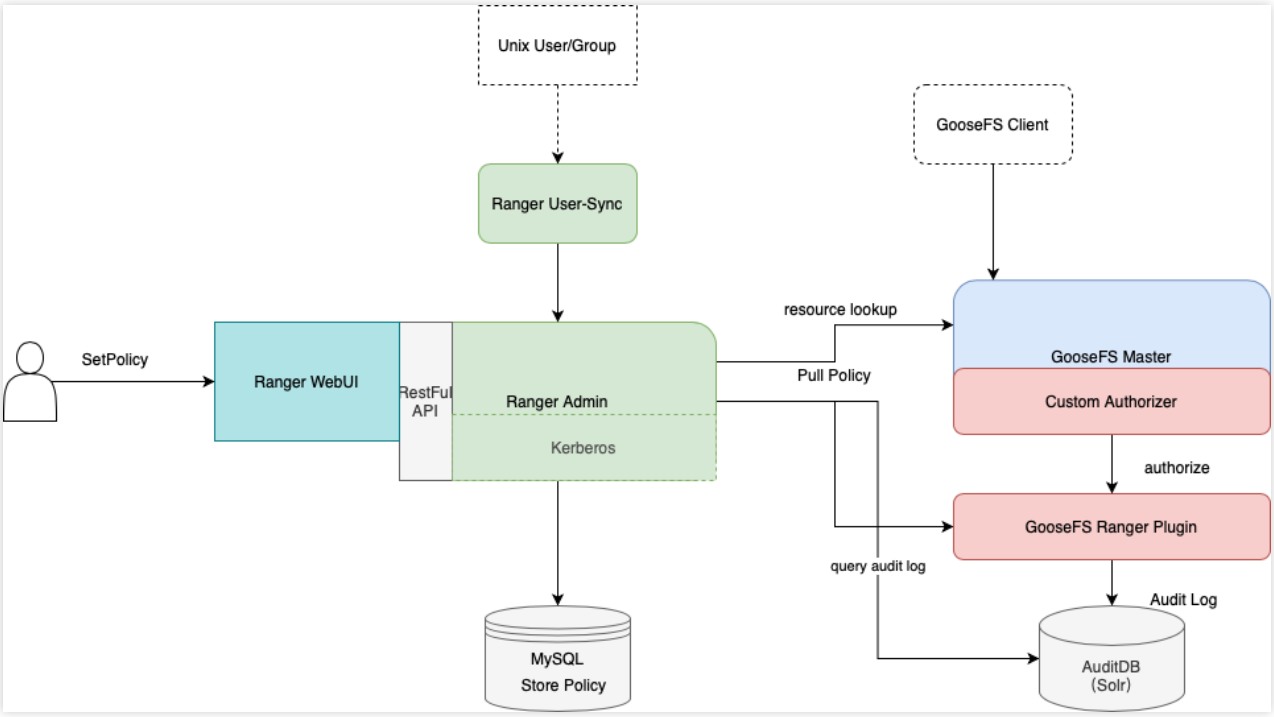
Overview

Apache Ranger is a standardized authentication component that manages access permissions across the big data ecosystem. GooseFS, as an acceleration storage system used for big data and data lakes, can be integrated into the comprehensive Apache Ranger authentication platform. This document describes how to use Apache Ranger to manage access permissions for GooseFS.

Advantages

GooseFS is a cloud-native accelerated storage system that has supported Apache Range access permission management nearly in the same way as it supported HDFS. Therefore, HDFS big data users can easily migrate to GooseFS and reuse HDFS Ranger permission policies.
Compared with HDFS with Ranger, GooseFS with Ranger offers an authentication option of “Ranger + native ACL” that allows you to use the native ACL authentication when Ranger authentication fails, which solves the problem of imperfect Ranger authentication policy configurations.

Framework of GooseFS with Ranger



To integrate GooseFS into Ranger, we developed the GooseFS Ranger plugin that should be deployed on the GooseFS master node and Ranger Admin. The plugin does the following operations:

On the GooseFS master node:

Provides the `Authorizer` API to authenticate each metadata request on the GooseFS master node.

Connects to Ranger Admin to obtain user-configured authentication policies.

On the Ranger Admin:

Supports GooseFS resource lookup for Ranger Admin.

Verifies configurations.

Deployment

Preparations

Before you start, ensure that you have deployed and configured Ranger components (including Ranger Admin and Ranger UserSync) in the environment and can open and use the Ranger web UI normally.

Component Deployment

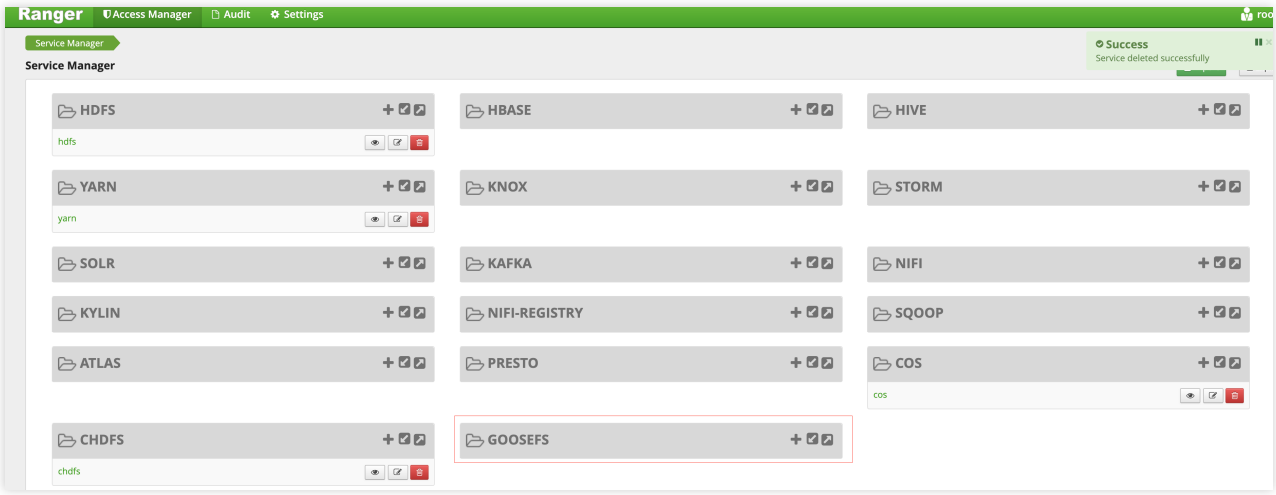
Deploying GooseFS Ranger plugin to Ranger Admin and registering service

Note:

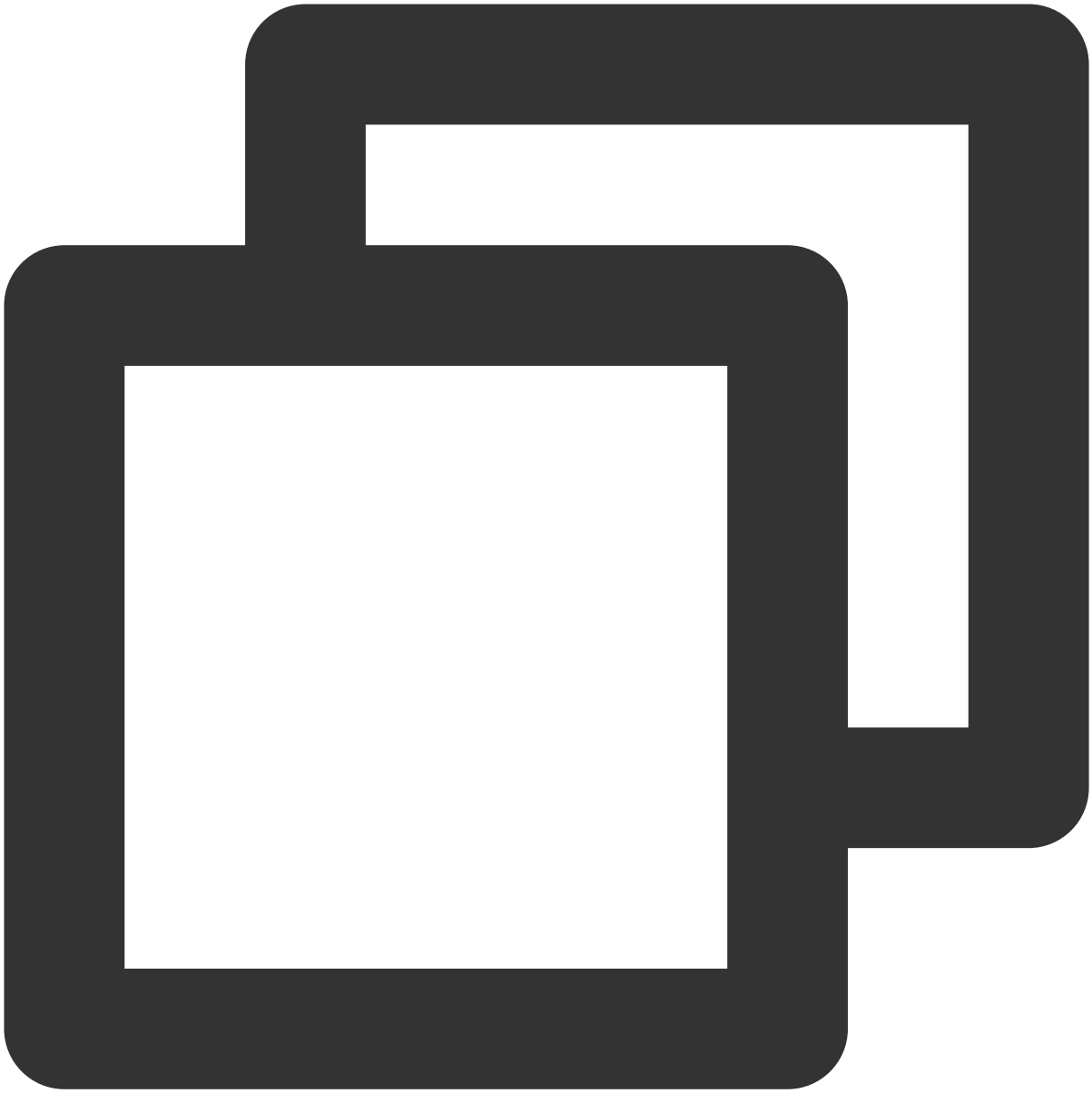
Click here to download the GooseFS Ranger plugin.

Deploy as follows:

1. Create a GooseFS directory in the Ranger service definition directory. Note that you should at least have execute and read permissions on the directory.
1. If you use a Tencent Cloud EMR cluster, the Ranger service definition directory is `/usr/local/service/ranger/ews/webapp/WEB-INF/classes/ranger-plugins`.
2. If you use a self-built Hadoop cluster, you can search for the path of Ranger-integrated components (such as HDFS) in Ranger to locate the path.



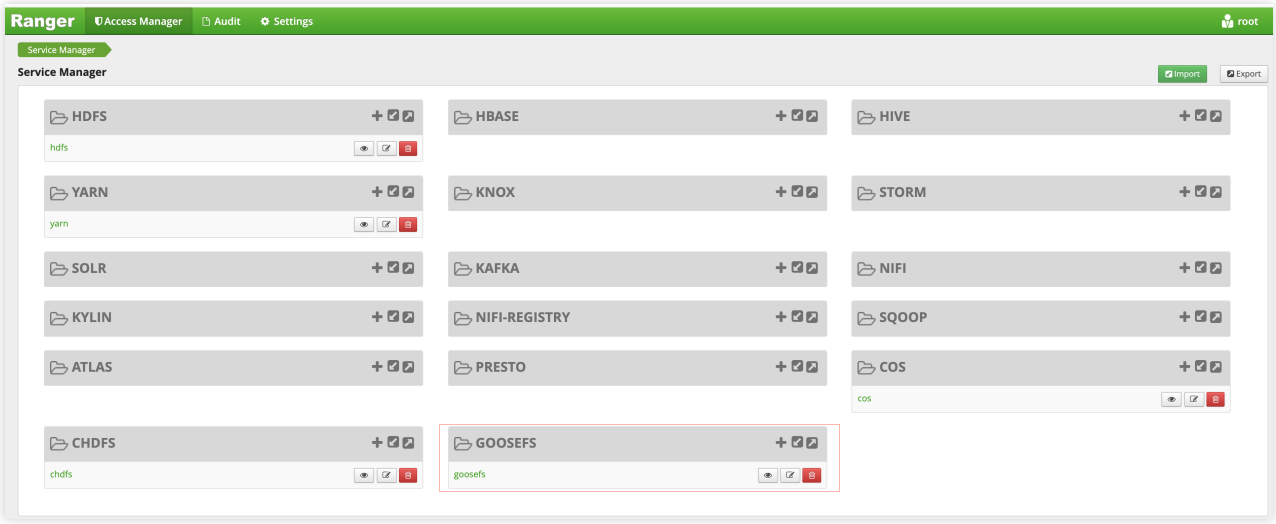
3. Put `goosefs-ranger-plugin-${version}.jar` and `ranger-servicedef-goosefs.json` in the GooseFS directory. Note that you should have read permission.
4. Restart Ranger.
5. In Ranger, run the following commands to register the GooseFS service:



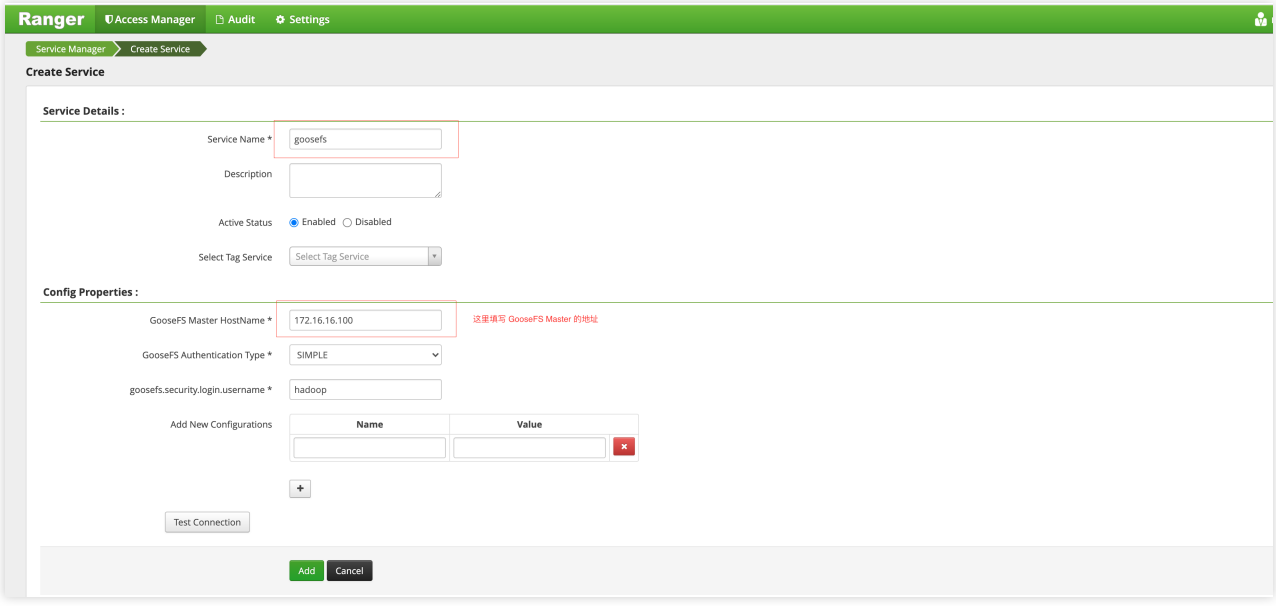
```
# Create the service. The Ranger admin account and password, as well as the Ranger
# For the Tencent Cloud EMR cluster, the admin is the root, and the password is the
adminUser=root
adminPasswd=xxxx

rangerServerAddr=10.0.0.1:6080

curl -v -u${adminUser}:${adminPasswd} -X POST -H "Accept:application/json" -H "Cont

# When the service is successfully registered, a service ID will be returned, which
# To delete the GooseFS service, pass the service ID returned to run the following
```

```
serviceId=104
curl -v -u${adminUser}:${adminPasswd} -X DELETE -H "Accept:application/json" -H "Co
```

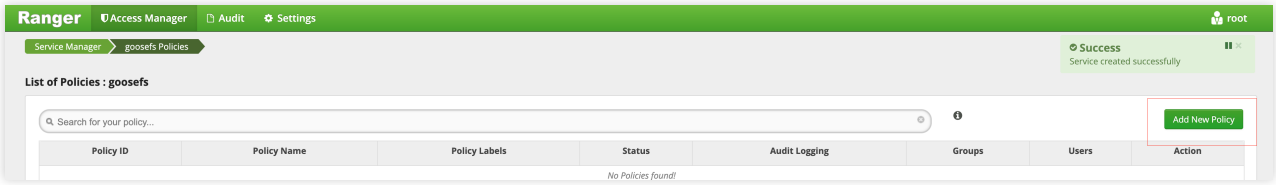
6. After the GooseFS service is created, you can view it in the Ranger console.



7. Click **+** to define the GooseFS service instance.



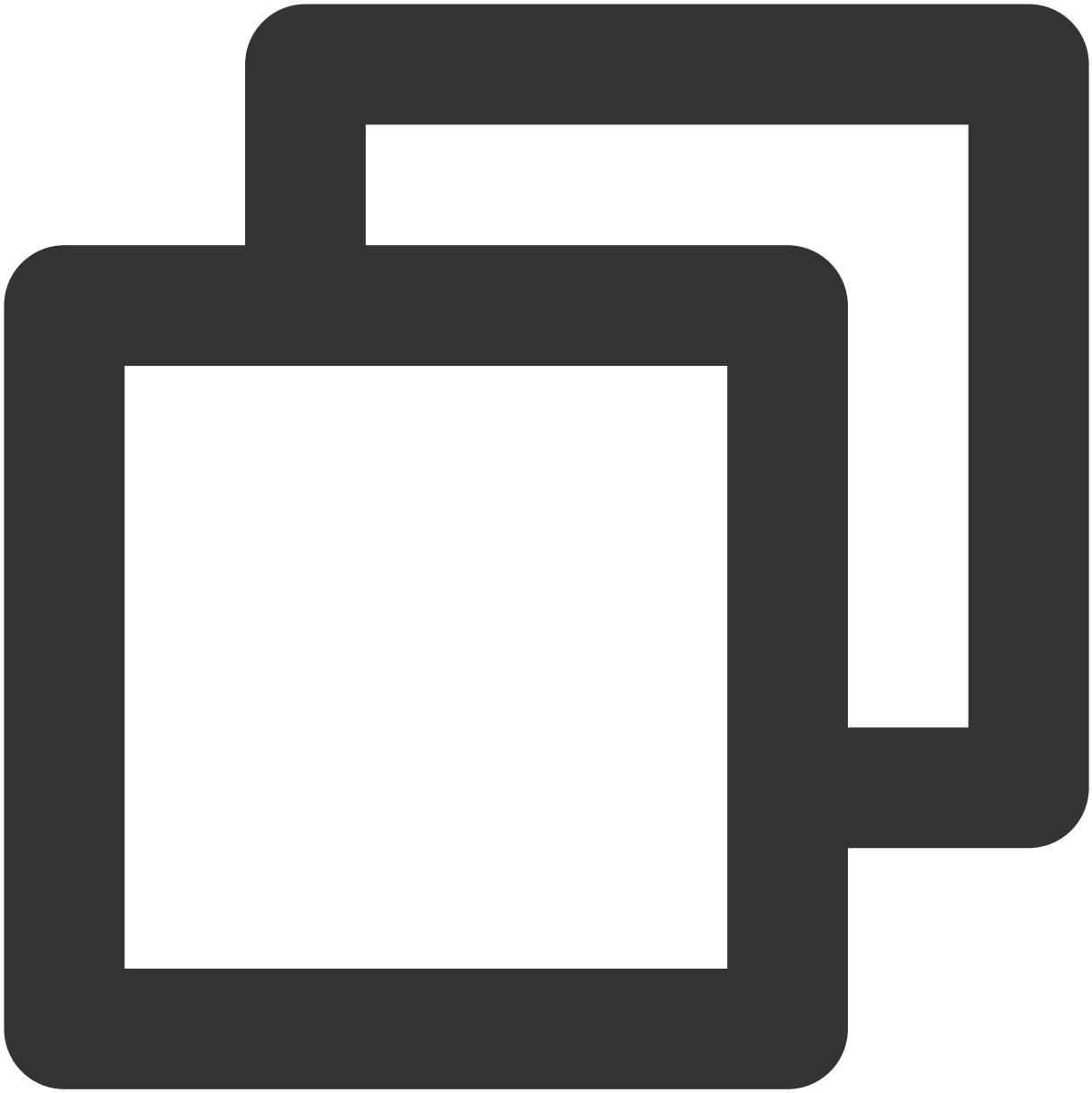
8. Click the created GooseFS instance and add a policy.



Deploying GooseFS Ranger plugin and enabling Ranger authentication

1. Put `goosefs-ranger-plugin-${version}.jar` in the `\\${GOOSEFS_HOME}/lib` directory. You should at least have read permission.
2. Put `ranger-goosefs-audit.xml` , `ranger-goosefs-security.xml` , and `ranger-policymgr-ssl.xml` to the `\\${GOOSEFS_HOME}/conf` directory and configure the required parameters as follows:

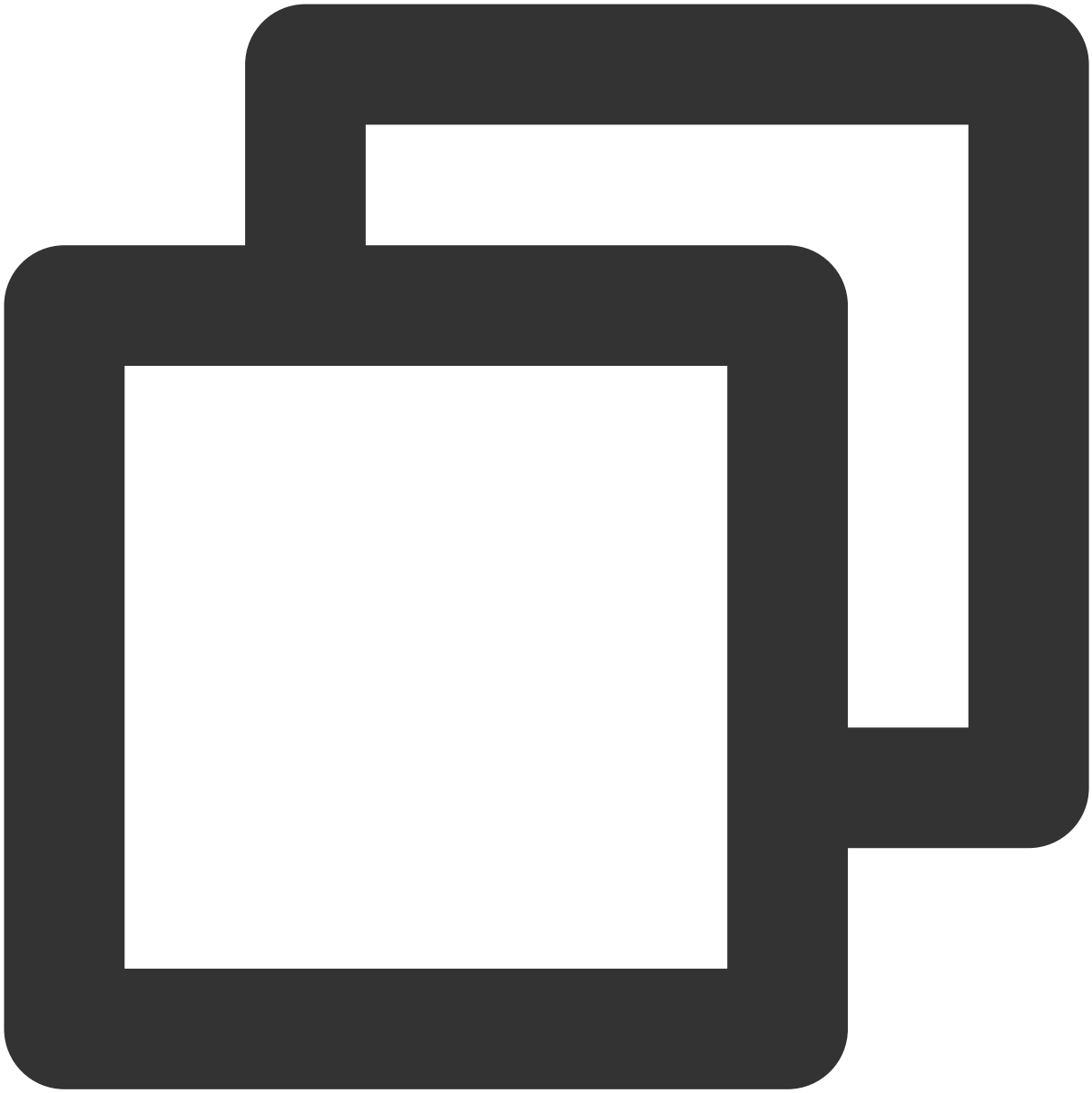
`ranger-goosefs-security.xml` :



```
<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` (you can skip it if audit is disabled):



```
<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>
```

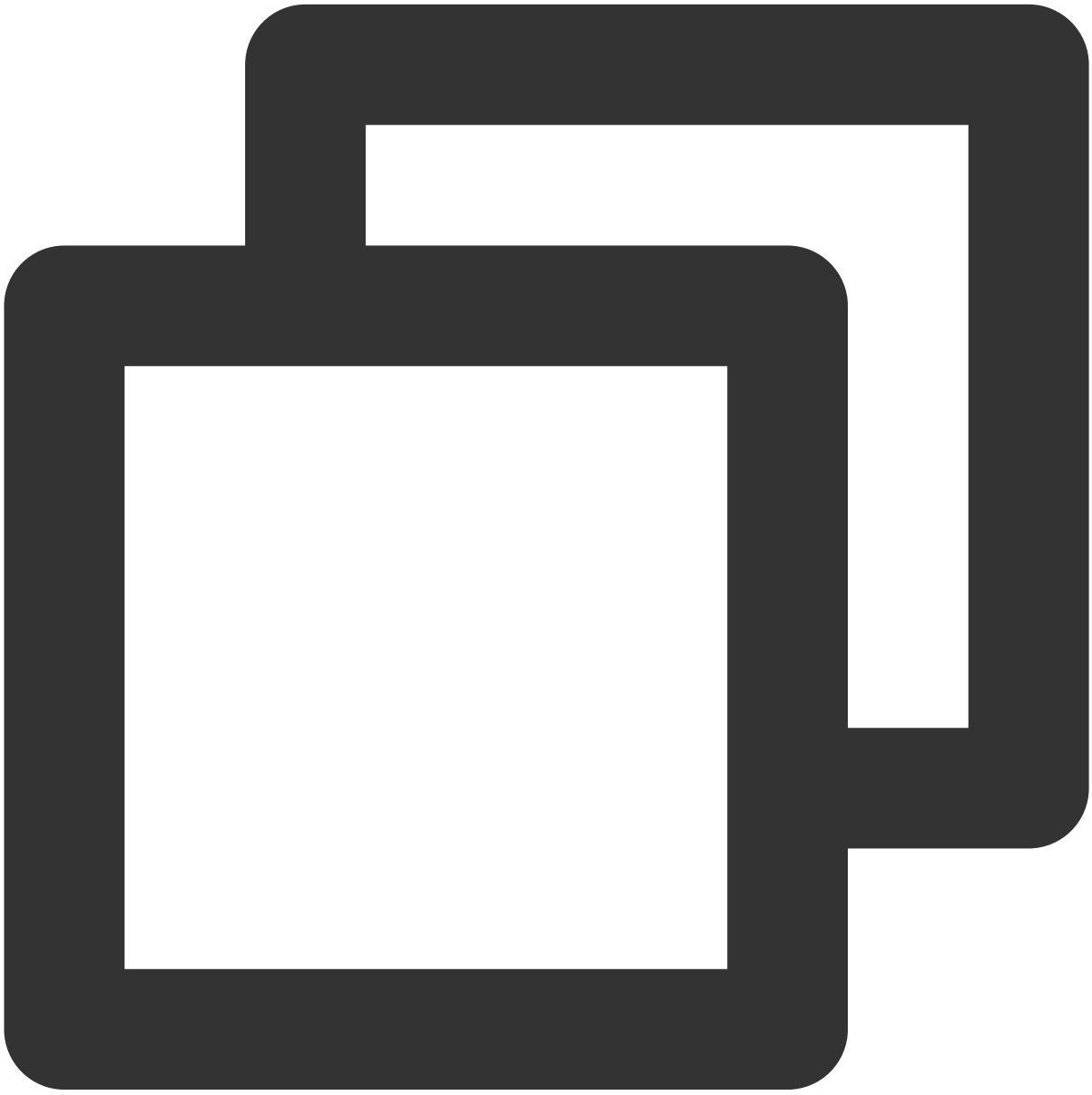
```
  <value>true</value>
   </property>

   <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>
```

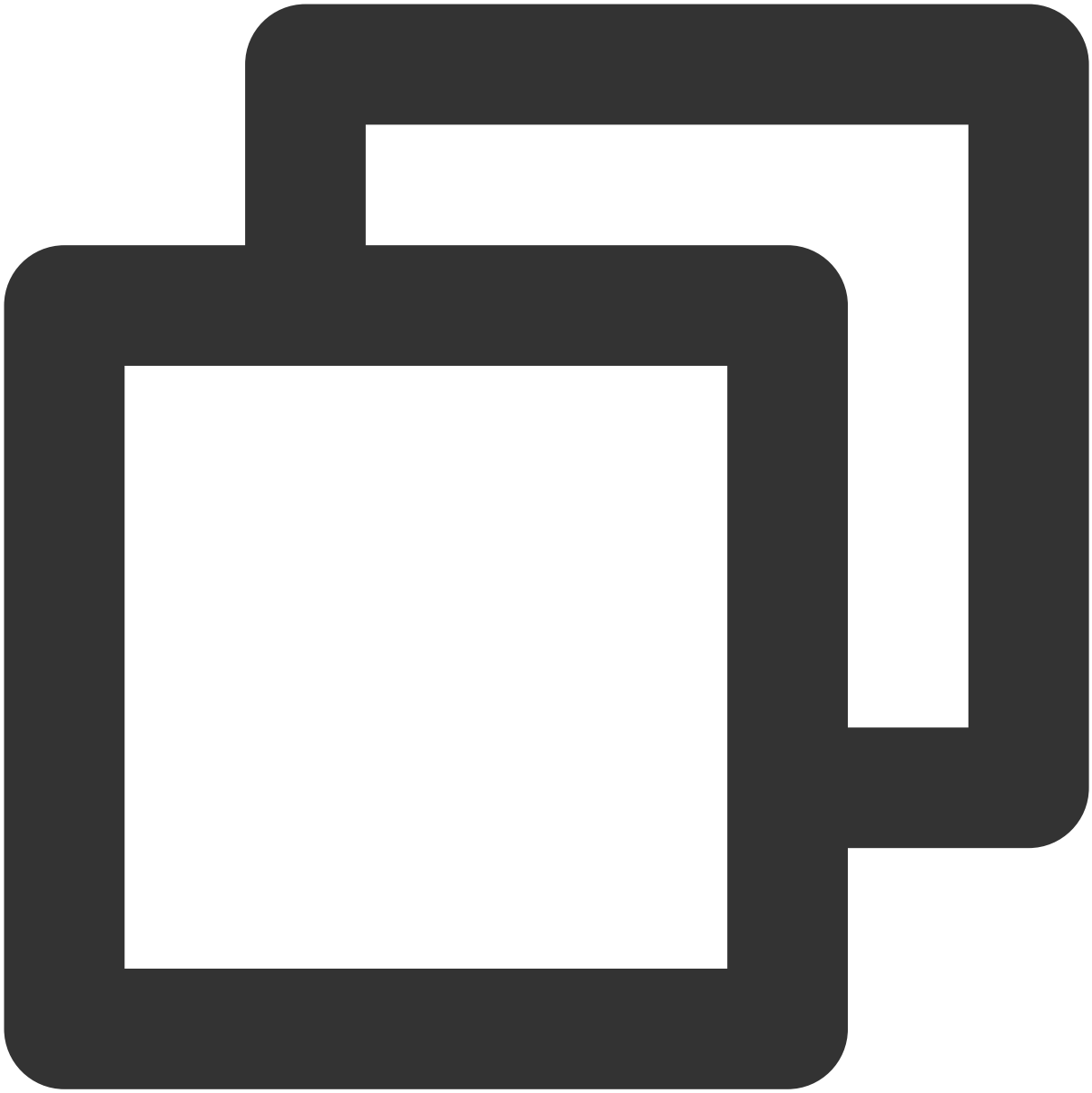
`ranger-policymgr-ssl.xml`



```
<configuration>
  <property>
```

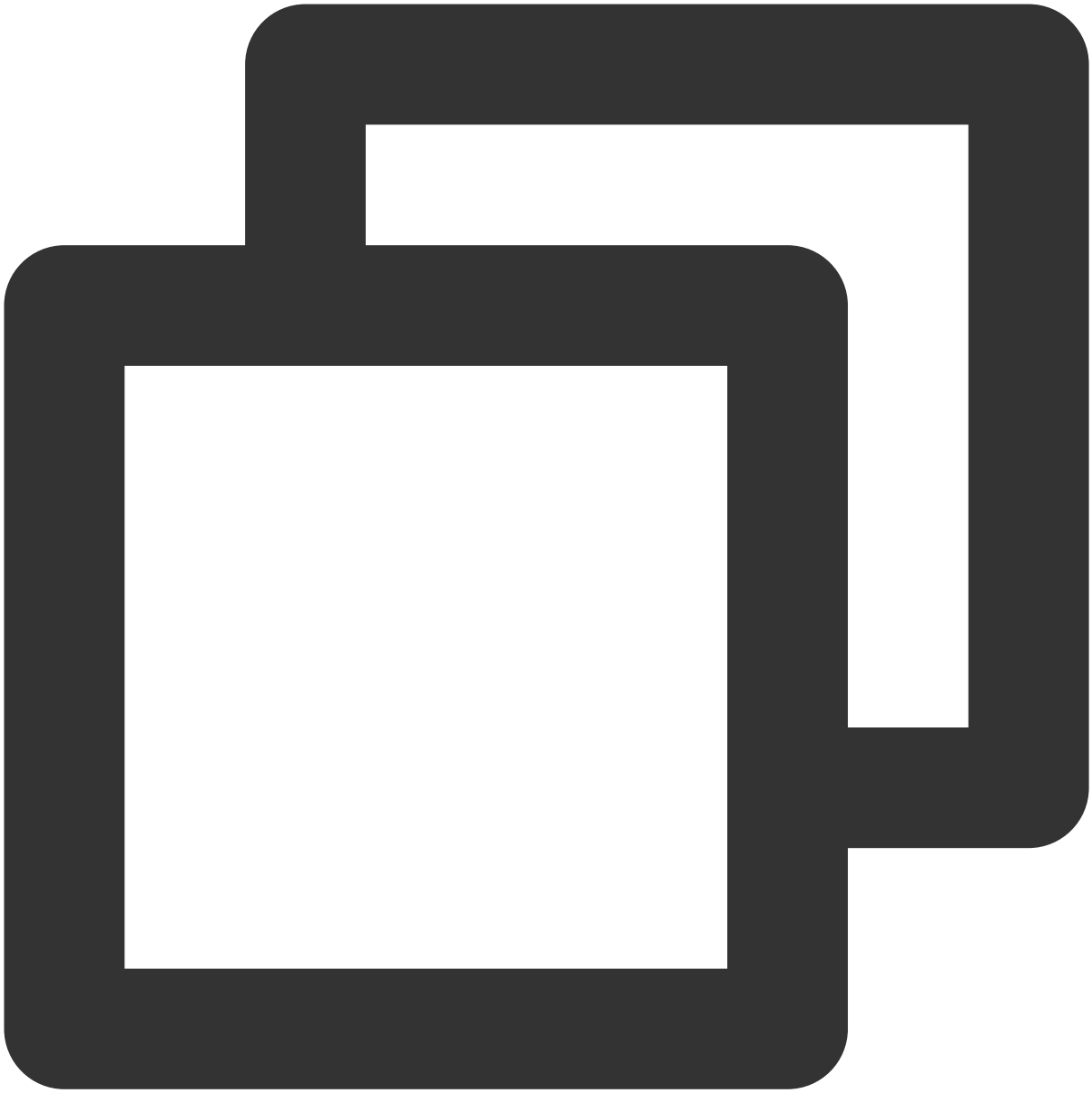
```
    <name>xasecure.policymgr.clientssl.keystore</name>
    <value>hadoopdev-clientcert.jks</value>
  </property>

  <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. Add the following configurations to `goosefs-site.properties` :



```
...
goosefs.security.authorization.permission.type=CUSTOM
goosefs.security.authorization.custom.provider.class=org.apache.ranger.authorizatio
...
```

4. In `\\${GOOSEFS_HOME}/libexec/goosefs-config.sh`, add `goosefs-ranger-plugin-${version}.jar` to the GooseFS class paths:



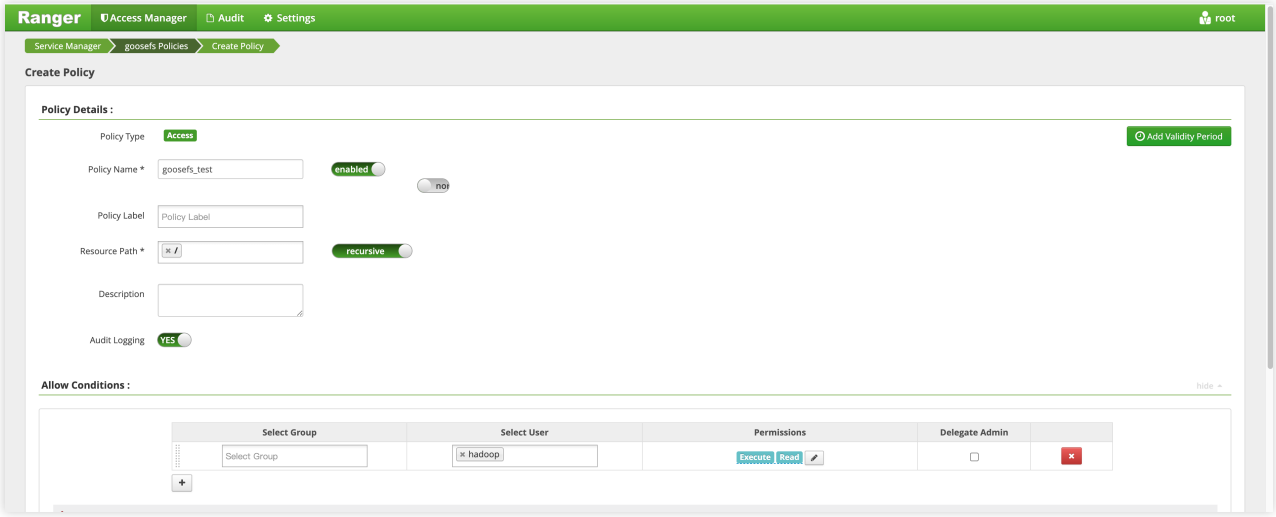
```
...
GOOSEFS_RANGER_CLASSPATH="${GOOSEFS_HOME}/lib/ranger-goosefs-plugin-${version}.jar"
GOOSEFS_SERVER_CLASSPATH=${GOOSEFS_SERVER_CLASSPATH}:${GOOSEFS_RANGER_CLASSPATH}
...
```

After these, the configuration is complete.

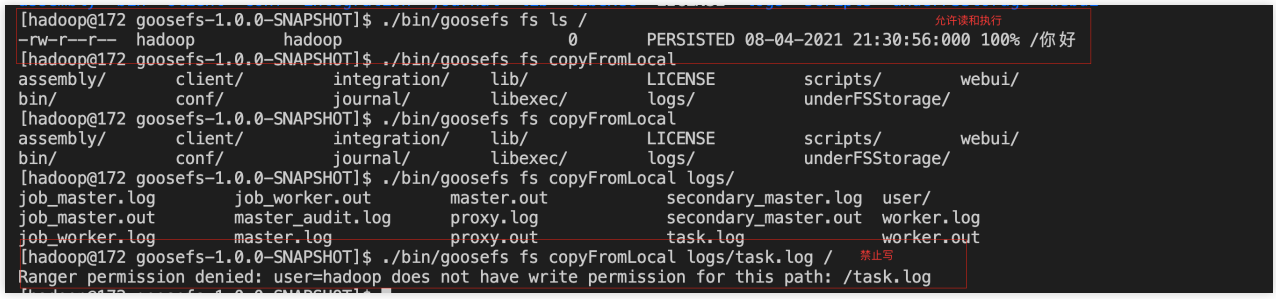
Verification

You can add a policy that allows Hadoop users to read and execute but not write to the GooseFS root directory as follows:

1. Add a policy.



2. Verify the policy to see if the policy takes effect.



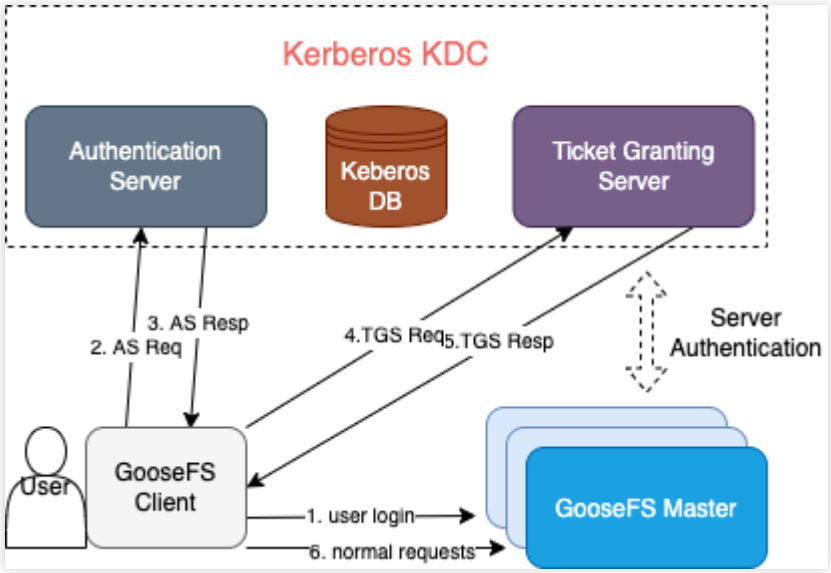
Connecting GooseFS to Kerberos Authentication

Last updated：2024-03-25 16:04:01

Overview

Kerberos is a unified authentication service widely used in the big data ecosystem. GooseFS, as the storage acceleration service for big data and data lake scenarios, allows you to connect cluster nodes and user access requests to Kerberos. This document describes how to configure GooseFS connection to Kerberos and how to use a Hadoop Delegation Token for authentication.

Connecting GooseFS to the Kerberos Authentication Architecture



Strengths of Kerberos Authentication for GooseFS

The authentication architecture and process are basically the same as those of HDFS connection to Kerberos. Applications for which the Kerberos authentication process is enabled in HDFS can be migrated to GooseFS easily. The Hadoop Delegation Token authentication mechanism is supported, so GooseFS is well compatible with Hadoop-based application jobs.

Configuring GooseFS connection to Kerberos

Prerequisites

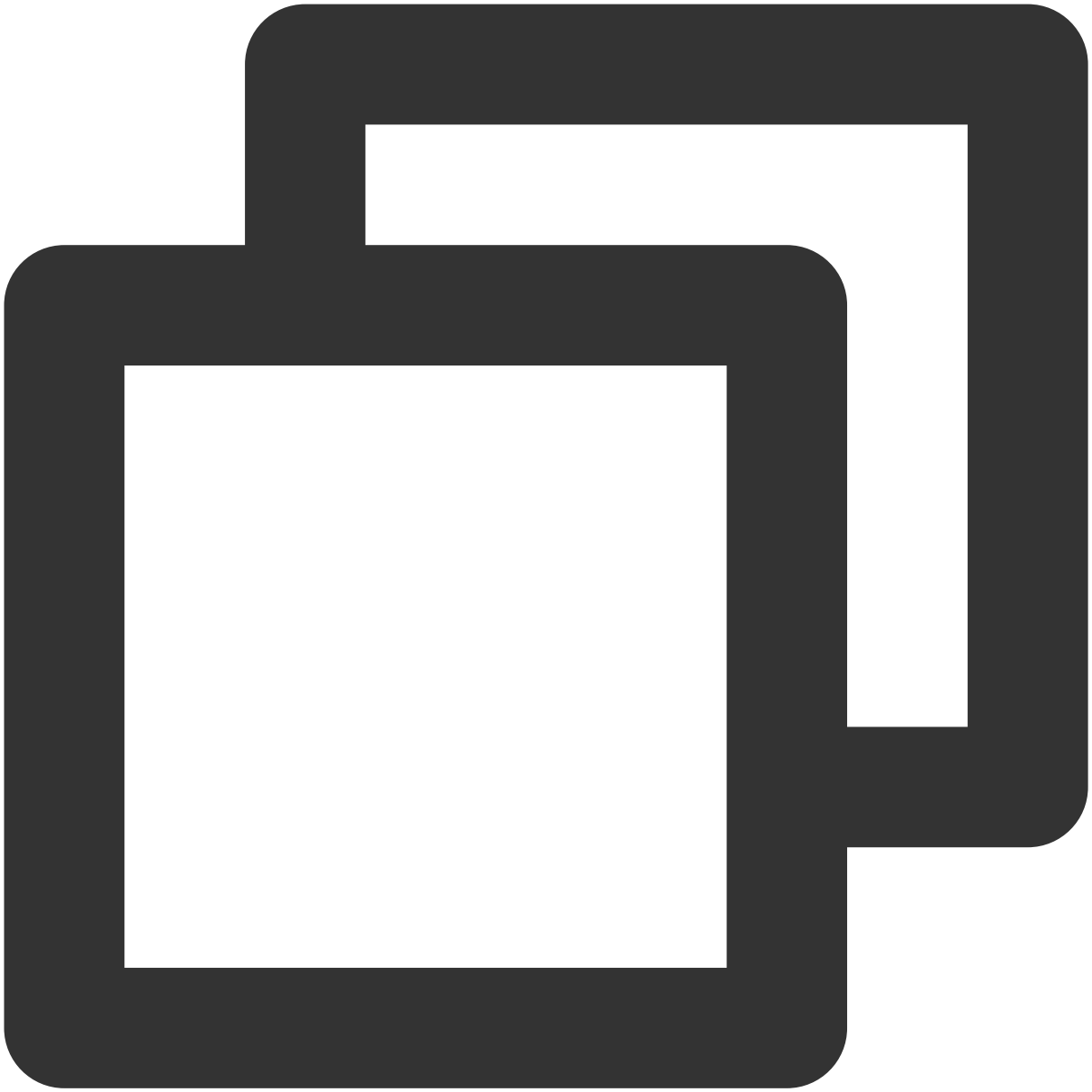
GooseFS 1.3.0 or later.
Make sure that the Kerberos KDC service exists in your environment and GooseFS and the application client can access its port normally.

Creating the GooseFS identity information in Kerberos KDC

First, create Kerberos identities for the GooseFS cluster server and client in Kerberos KDC before configuring the connection. Here, `kadmin.local` is used on the **Kerberos KDC server** for creation:

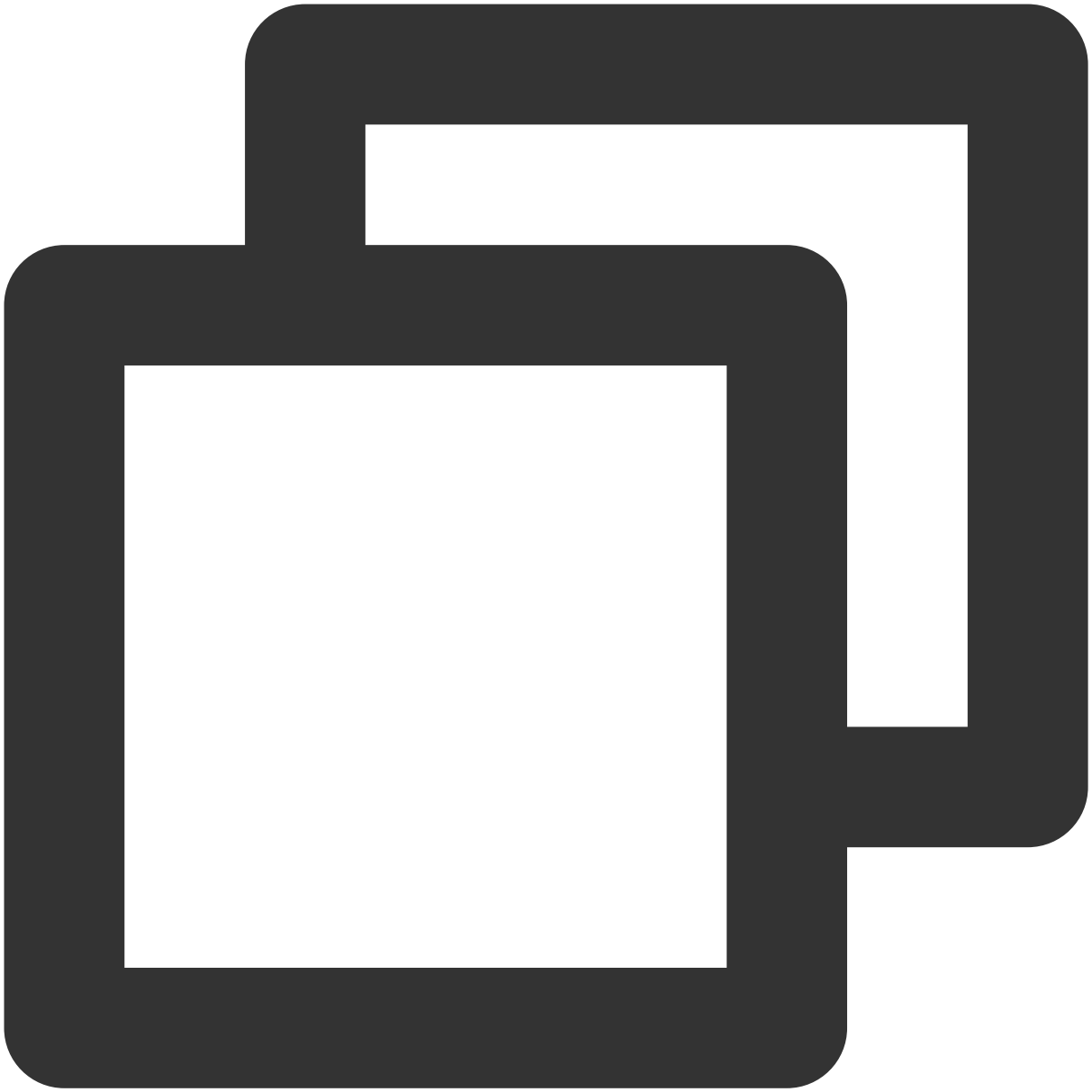
Note:

You need to run the `kadmin.local` tool with the `root/sudo` privileges.



```
$ sudo kadmin.local
```

If the command is executed successfully, you will enter the interactive shell environment of kadmin:



```
Authenticating as principal root/admin@GOOSEFS.COM with password.
kadmin.local:
```

Here, `kadmin.local` indicates the command prompt of the interactive execution environment.

Creating GooseFS server/client identities

The following takes a simple test cluster and its use case as an example to describe how to configure Kerberos:

1. Cluster environment description:

A one-master-dual-worker standalone architecture is used:

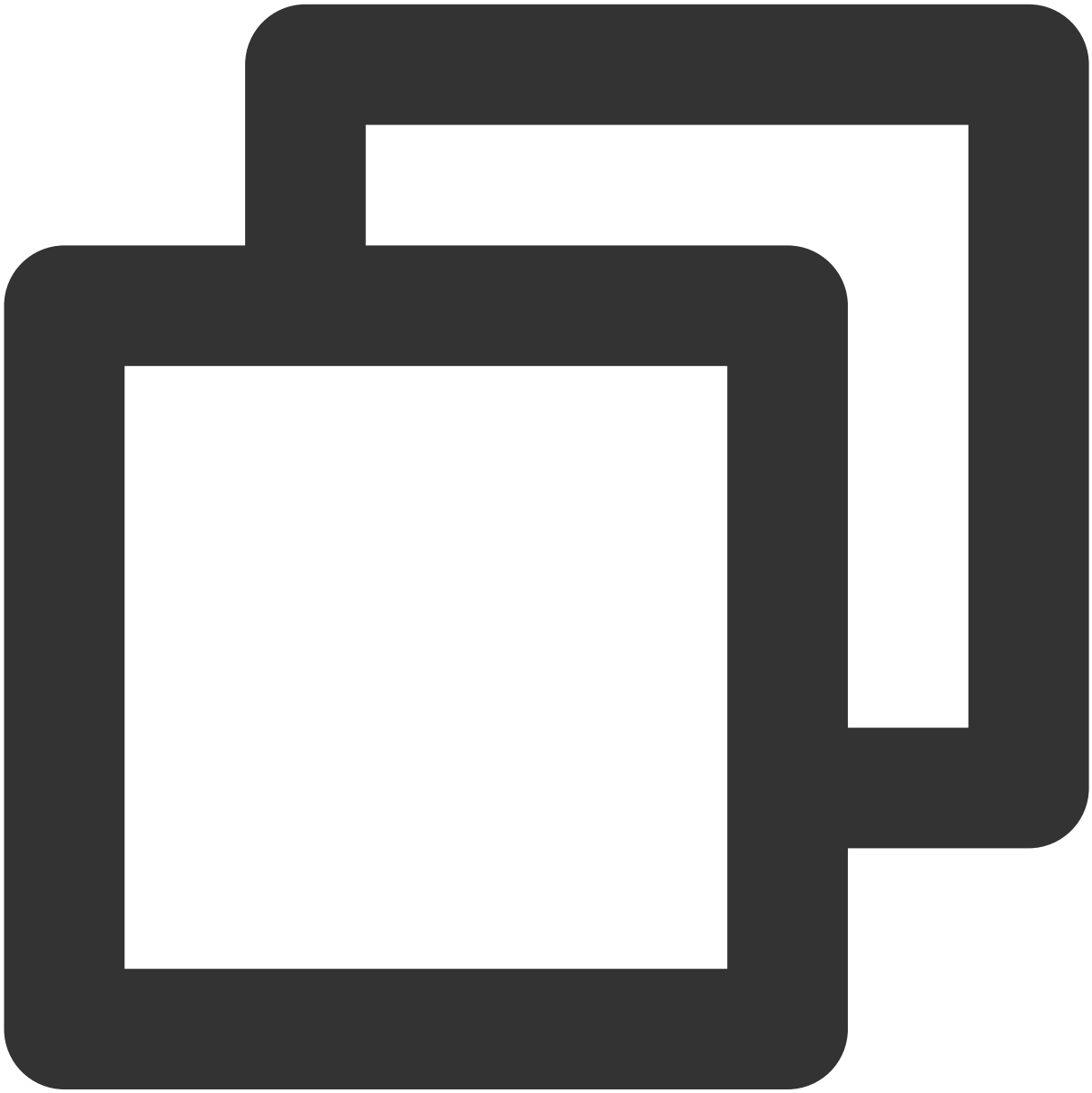
Master (JobMaster): 172.16.0.1

Worker 1 (JobWorker1): 172.16.0.2

Worker 2 (JobWorker2): 172.16.0.3

Client: 172.16.0.4

2. Create the server and client identities in `kadmin.local` :

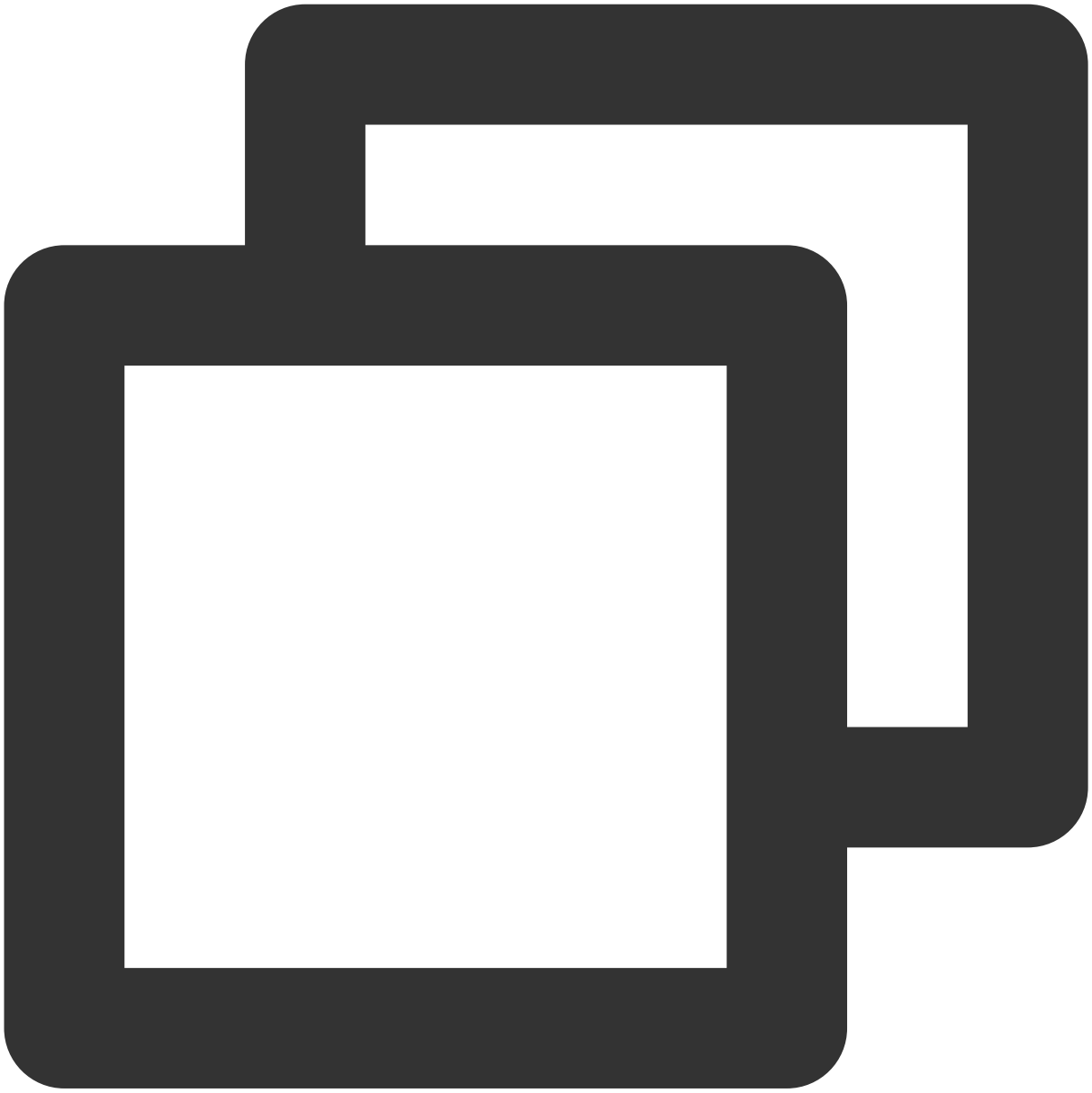


```
kadmin.local: addprinc -randkey goosefs/172.16.0.1@GOOSEFS.COM
kadmin.local: addprinc -randkey client/172.16.0.4@GOOSEFS.COM
```

Note:

Here, `-randkey` is used as no matter whether you log in to GooseFS on the server or client, a `.keytab` file instead of a plaintext password will be used for authentication. If the identity information needs to be used for login with a password, you can remove this field.

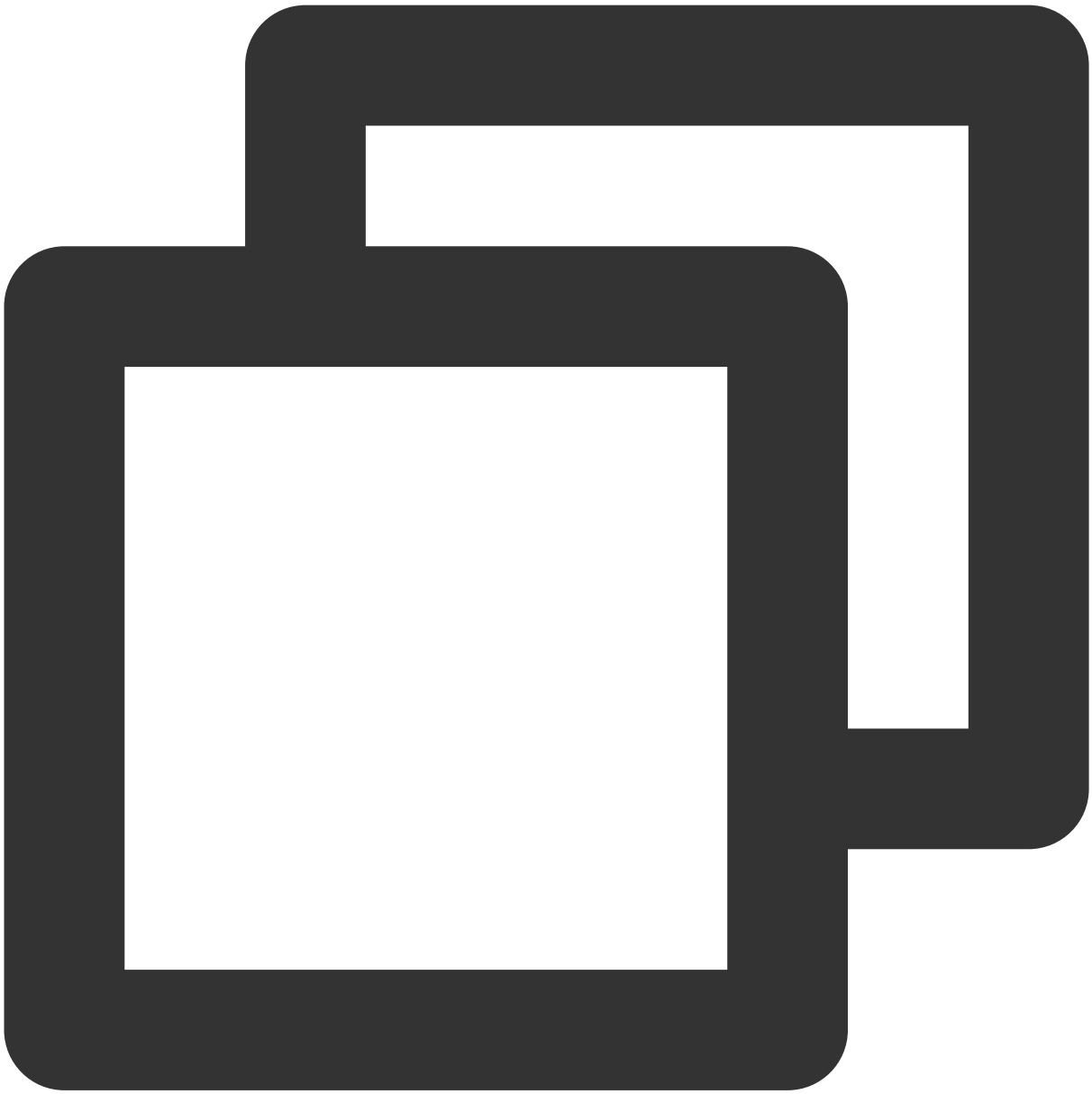
3. Generate and export the `.keytab` file for each identity:



```
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
```

Configuring Kerberos authentication for GooseFS server/client access

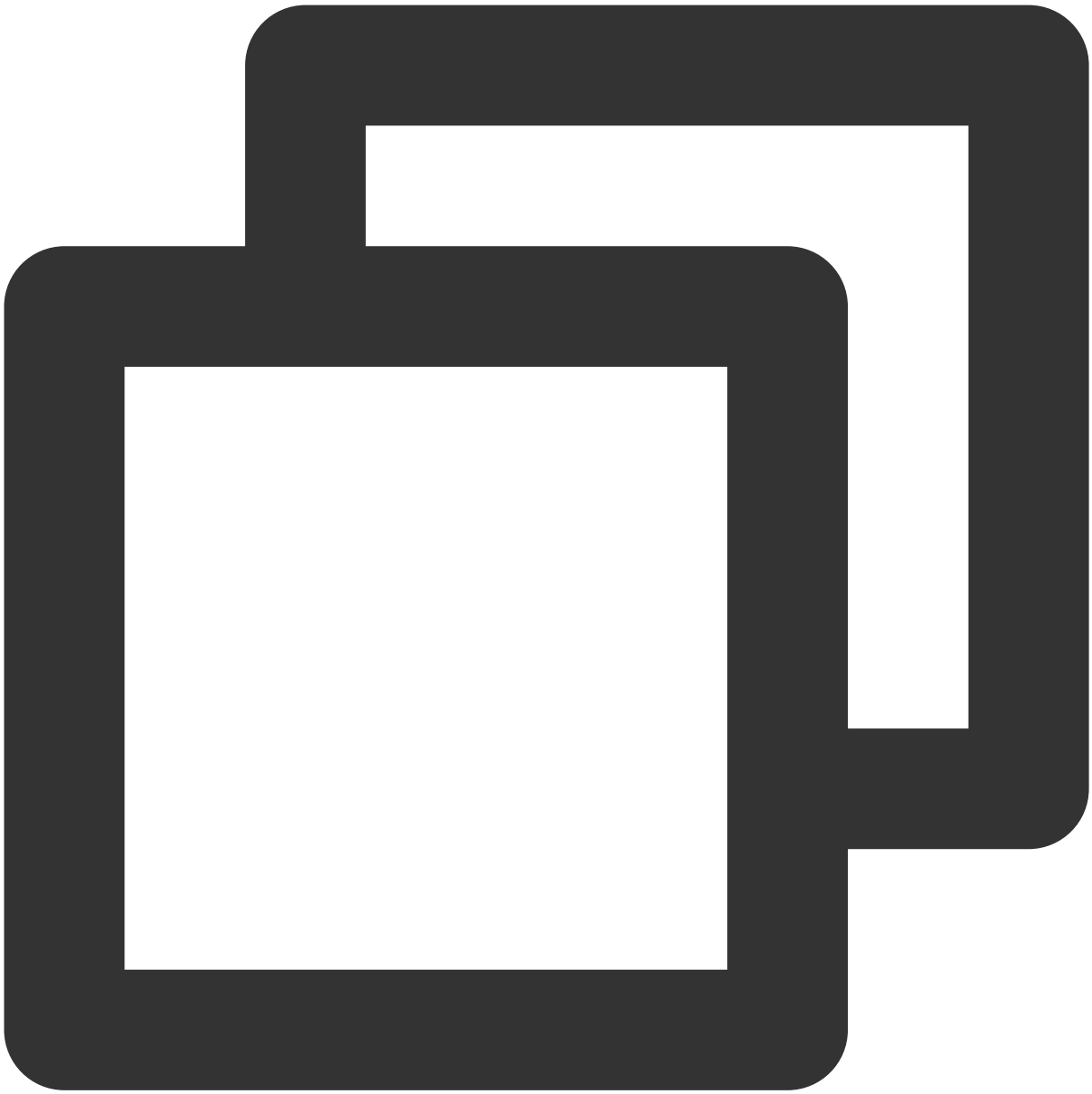
1. Distribute the `.keytab` files exported above to the corresponding servers. Here, we recommend you use the path `${GOOSEFS_HOME}/conf/` .



```
$ 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/
```

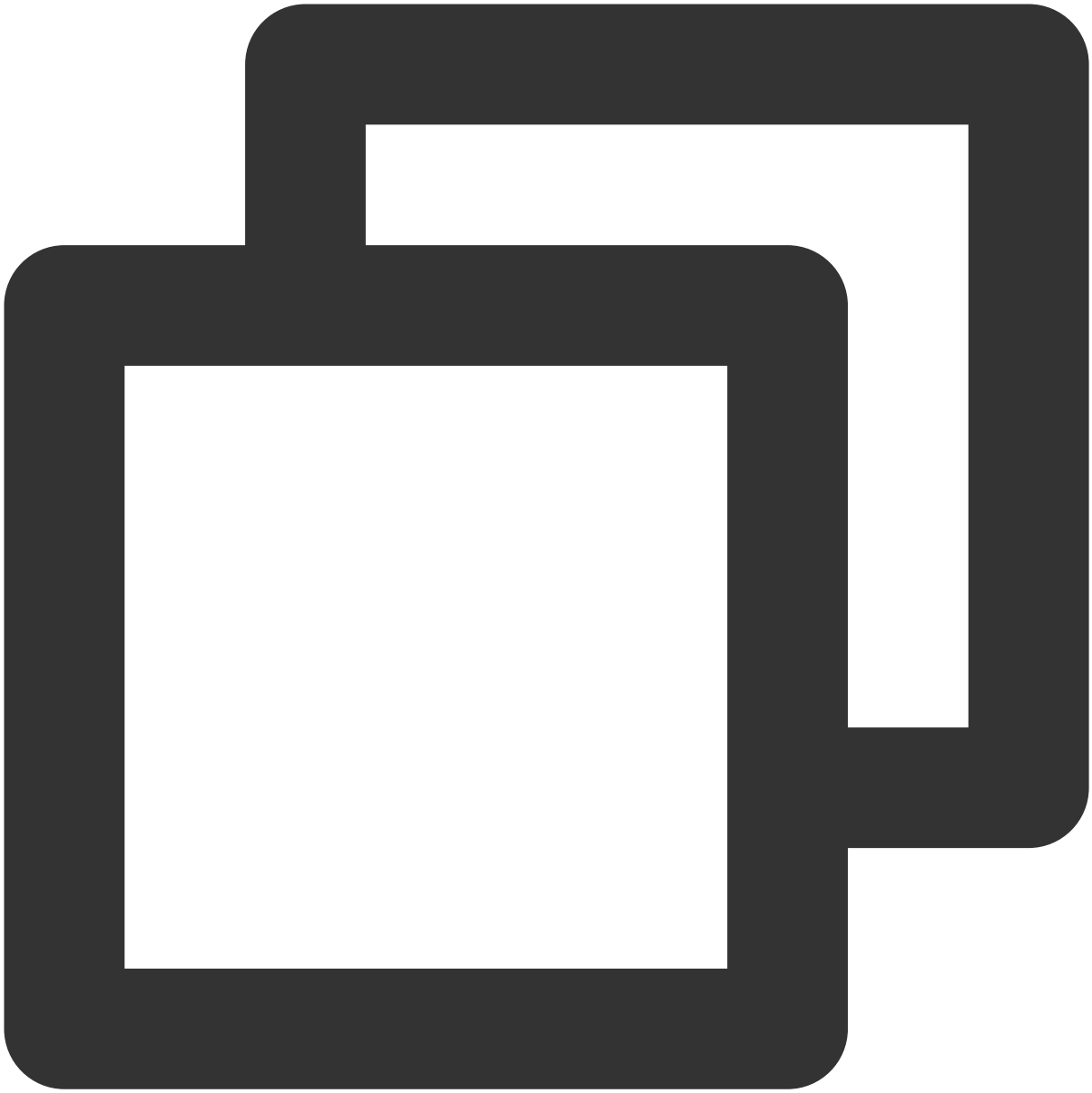
2. On the corresponding server, change the user/user group to which the server principal keytab file belongs to the user/user group used during GooseFS server startup, so as to grant GooseFS the required read permission during

startup.



```
$ chown <GooseFS_USER>:<GOOSEFS_USERGROUP> goosefs_172_16_0_1.keytab
$ # Modify the Unix access permission bit
$ chmod 0440 goosefs_172_16_0_1.keytab
```

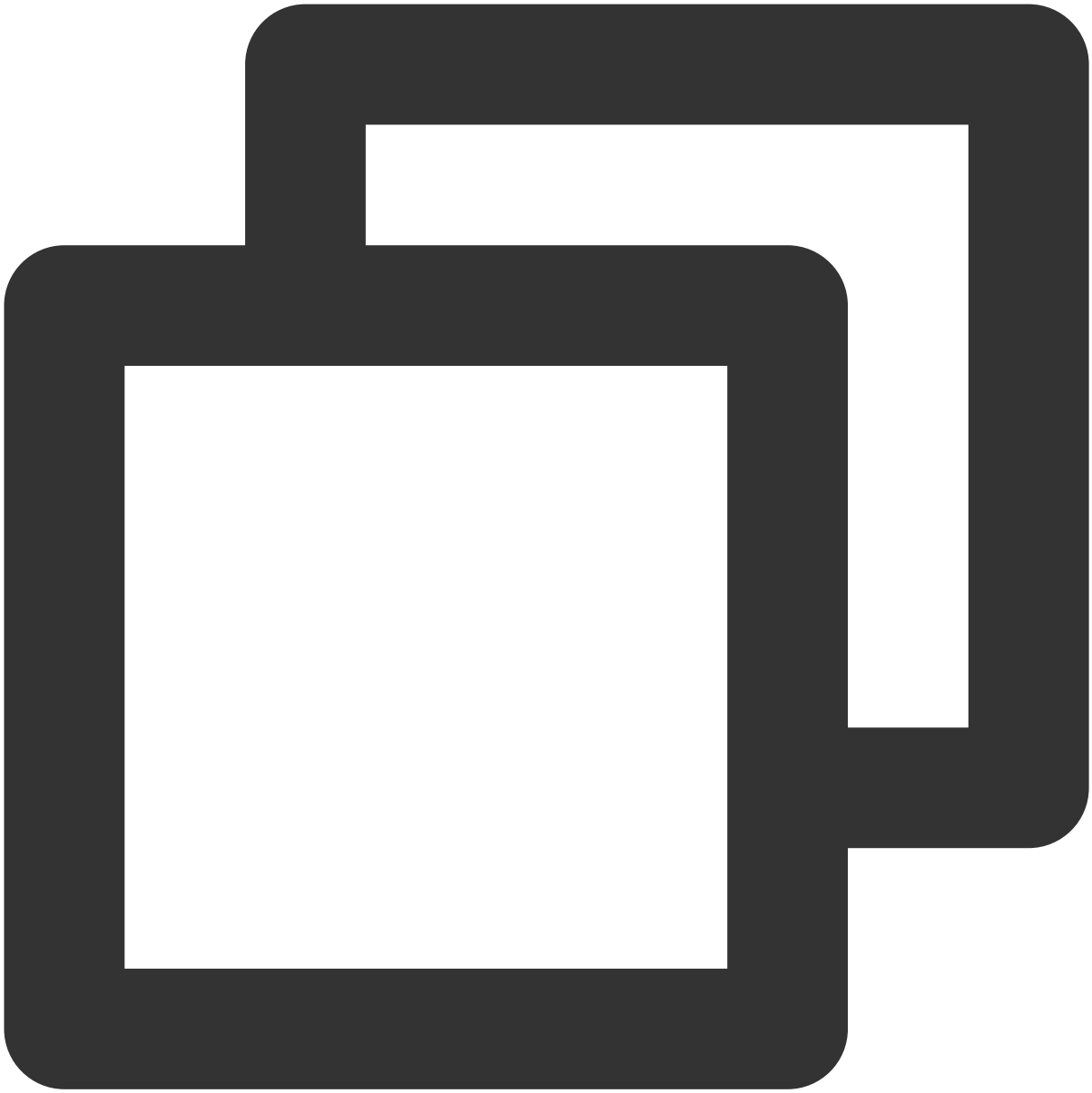
3. Change the user/user group to which the client keytab file belongs to the client account initiating GooseFS requests, so as to guarantee that the client has the required permission to read the file.



```
$ chown <client_user>:<client_usergroup> client_172_16_0_4.keytab
$ # Modify the Unix access permission bit
$ chmod 0440 client_172_16_0_4.keytab
```

GooseFS configuration on the server and client

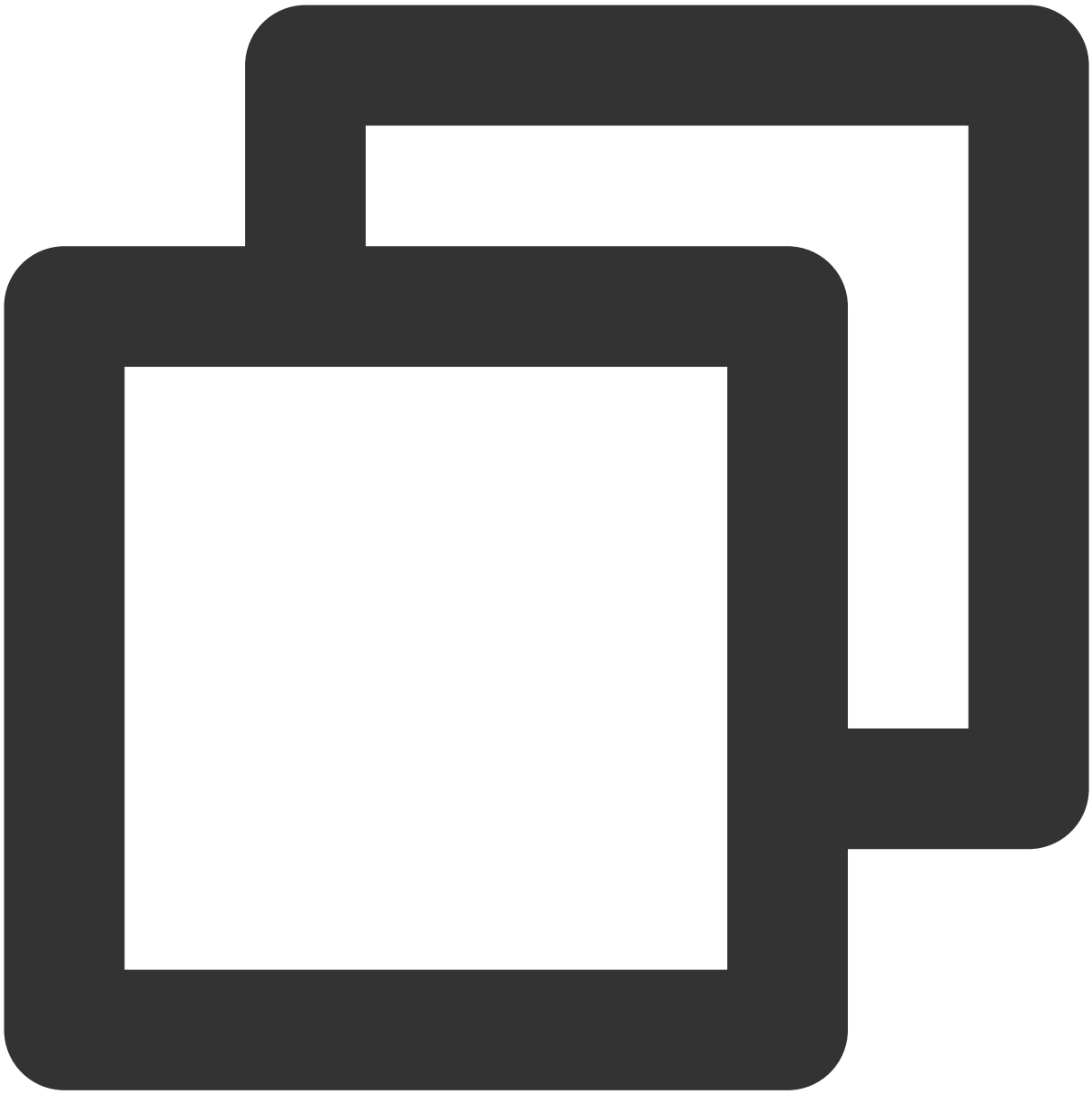
1. `goosefs-site.properties` on the master/worker server:



```
# 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_
```

After configuring authentication on the GooseFS server, restart the entire cluster for the configuration to take effect.

2. `goosefs-site.properties` on the client:



```
# 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
```

Note:

You need to specify the server principal on the client, as in the Kerberos authentication system, KDC needs to know the service accessed by the client currently, and GooseFS uses the server principal to identify the service requested by the client currently.

At this point, GooseFS has been connected to the basic authentication service of Kerberos, and all requests initiated by the client will be authenticated by Kerberos subsequently.