



**P E R C O N A**

# **Percona Server for MongoDB 5.0 Documentation**

*Release 5.0.10-9*

**Percona LLC and/or its affiliates 2015-2022**

**Aug 09, 2022**



## CONTENTS

<b>I</b>	<b>About Percona Server for MongoDB</b>	<b>3</b>
<b>II</b>	<b>Installation</b>	<b>7</b>
<b>III</b>	<b>Features</b>	<b>17</b>
<b>IV</b>	<b>How to</b>	<b>57</b>
<b>V</b>	<b>Release Notes</b>	<b>95</b>
<b>VI</b>	<b>Reference</b>	<b>107</b>



*Percona Server for MongoDB* is a free, enhanced, fully compatible, source available, drop-in replacement for MongoDB 5.0 Community Edition with enterprise-grade features. It requires no changes to MongoDB applications or code.

---

**Hint:** To see which version of *Percona Server for MongoDB* you are using check the value of the `psmdbVersion` key in the output of the `buildInfo` database command. If this key does not exist, *Percona Server for MongoDB* is not installed on the server.

---

*Percona Server for MongoDB* provides the following features:

- MongoDB's default [WiredTiger](#) engine
- *Percona Memory Engine* storage engine
- *Data at Rest Encryption*
- *External authentication* using OpenLDAP or Active Directory
- *Audit logging* to track and query database interactions of users or applications
- *Hot Backup* for the default [WiredTiger](#)
- *Profiling Rate Limit* to decrease the impact of the profiler on performance

To learn more about the features, available in *Percona Server for MongoDB*, see [Percona Server for MongoDB Feature Comparison](#)

---



## **Part I**

# **About Percona Server for MongoDB**





## PERCONA SERVER FOR MONGODB FEATURE COMPARISON

*Percona Server for MongoDB 5.0* is based on [MongoDB 5.0](#). *Percona Server for MongoDB* extends MongoDB Community Edition to include the functionality that is otherwise only available in MongoDB Enterprise Edition.

	PSMDB	MongoDB
Storage Engines	<ul style="list-style-type: none"> <li>• <a href="#">WiredTiger</a> (default)</li> <li>• <i>Percona Memory Engine</i></li> </ul>	<ul style="list-style-type: none"> <li>• <a href="#">WiredTiger</a> (default)</li> <li>• <a href="#">In-Memory</a> (Enterprise only)</li> </ul>
Encryption-at-Rest	<ul style="list-style-type: none"> <li>• Key servers = <i>Hashicorp Vault, KMIP</i></li> <li>• Fully open source</li> </ul>	<ul style="list-style-type: none"> <li>• Key server = KMIP</li> <li>• Enterprise only</li> </ul>
Hot Backup	<i>YES</i> (replica set)	NO
LDAP Authentication	<ul style="list-style-type: none"> <li>• (legacy) <i>LDAP authentication with SASL</i></li> </ul>	<ul style="list-style-type: none"> <li>• Enterprise only</li> </ul>
LDAP Authorization	<i>YES</i>	Enterprise only
Kerberos Authentication	<i>YES</i>	Enterprise only
Audit Logging	<i>YES</i>	Enterprise only
Log redaction	<i>YES</i>	Enterprise only
SNMP Monitoring	NO	Enterprise only

### 1.1 Profiling Rate Limiting

Profiling Rate Limiting was added to *Percona Server for MongoDB* in v3.4 with the `--rateLimit` argument. Since v3.6, MongoDB Community (and Enterprise) Edition includes a similar option `slowOpSampleRate`. Please see [Profiling Rate Limit](#) for more information.



# **Part II**

## **Installation**



## INSTALLING PERCONA SERVER FOR MONGODB

Percona provides installation packages of *Percona Server for MongoDB* for the most 64-bit Linux distributions. Find the full list of supported platforms on the [Percona Software and Platform Lifecycle](#) page.

The recommended installation method is from *Percona* repositories. Follow the links below for the installation instructions for your operating system.

- [Install on Debian or Ubuntu](#)
- [Install on RHEL or CentOS](#)

### 2.1 Alternative Install Instructions

You can also download packages from the [Percona website](#) and install them manually using **dpkg** or **rpm**.

---

**Note:** In this case, you will have to make sure that all dependencies are satisfied.

---

If you want more control over the installation, you can *install Percona Server for MongoDB from binary tarballs*.

---

**Note:** This method is for advanced users with specific needs that are not addressed by DEB and RPM packages.

---

If you want to run *Percona Server for MongoDB* in a Docker container, see *Running Percona Server for MongoDB in a Docker Container*.

### 2.2 Upgrade Instructions

If you are currently using MongoDB, see *Upgrading from MongoDB*.

If you are running an earlier version of *Percona Server for MongoDB*, see *Upgrading from Version 4.4*.

## 2.3 Uninstall Instructions

To uninstall *Percona Server for MongoDB*, see *Uninstalling Percona Server for MongoDB*.

### 2.3.1 Installing Percona Server for MongoDB on Debian and Ubuntu

Use this document to install *Percona Server for MongoDB* from Percona repositories on DEB-based distributions such as Debian and Ubuntu.

---

**Note:** *Percona Server for MongoDB* should work on other DEB-based distributions, but it is tested only on platforms listed on the [Percona Software and Platform Lifecycle](#) page.

---

- *Package Contents*
- *Installing from Percona repositories*
  - *Configure Percona repository*
  - *Install the latest version*
  - *Install a specific version*
- *Running Percona Server for MongoDB*

#### Package Contents

Package	Contains
percona-server-mongodb	The mongo shell, import/export tools, other client utilities, server software, default configuration, and <code>init.d</code> scripts.
percona-server-mongodb-server	The <b>mongod</b> server, default configuration files, and <code>init.d</code> scripts
percona-server-mongodb-shell	The mongo shell
percona-server-mongodb-mongos	The mongos sharded cluster query router
percona-server-mongodb-tools	Mongo tools for high-performance MongoDB fork from Percona
percona-server-mongodb-dbg	Debug symbols for the server

## Installing from Percona repositories

It is recommended to install *Percona Server for MongoDB* from official Percona repositories using the **percona-release** utility.

### Configure Percona repository

Run the following commands as root or by using the **sudo** command

1. Fetch **percona-release** packages from Percona web:

```
$ wget https://repo.percona.com/apt/percona-release_latest.${lsb_release -sc}_all.  
↳ deb
```

2. Install the downloaded package with **dpkg**:

```
$ sudo dpkg -i percona-release_latest.${lsb_release -sc}_all.deb
```

Once you install **percona-release**, the *Percona* repositories should be available. You can check the repository setup in the `/etc/apt/sources.list.d/percona-release.list` file.

3. Enable the repository:

```
$ sudo percona-release enable psmdb-50 release
```

4. Remember to update the local cache:

```
$ sudo apt update
```

### Install the latest version

Run the following command to install the latest version of *Percona Server for MongoDB*:

```
$ sudo apt install percona-server-mongodb
```

### Install a specific version

To install a specific version of *Percona Server for MongoDB*, do the following:

1. List available versions:

```
$ sudo apt-cache madison percona-server-mongodb
```

---

#### Sample Output

```
percona-server-mongodb | 5.0.2-1.buster | http://repo.percona.com/psmdb-50/apt.  
↳ buster/main amd64 Packages
```

2. Install a specific version packages. You must specify each package with the version number. For example, to install *Percona Server for MongoDB* 5.0.2-1, run the following command:

```
$ sudo apt install percona-server-mongodb=5.0.2-1.buster percona-server-mongodb-
↪mongos=5.0.2-1.buster percona-server-mongodb-shell=5.0.2-1.buster percona-server-
↪mongodb-server=5.0.2-1.buster percona-server-mongodb-tools=5.0.2-1.buster
```

## Running Percona Server for MongoDB

By default, *Percona Server for MongoDB* stores data files in `/var/lib/mongodb/` and configuration parameters in `/etc/mongod.conf`.

### Starting the service

*Percona Server for MongoDB* is started automatically after installation unless it encounters errors during the installation process.

You can also manually start it using the following command:

```
$ sudo systemctl start mongod
```

### Confirming that the service is running

Check the service status using the following command:

```
$ sudo systemctl status mongod
```

### Stopping the service

Stop the service using the following command:

```
$ sudo systemctl stop mongod
```

### Restarting the service

Restart the service using the following command:

```
$ sudo systemctl restart mongod
```

## 2.3.2 Installing Percona Server for MongoDB on Red Hat Enterprise Linux and CentOS

Use this document to install Percona Server for MongoDB on RPM-based distributions such as Red Hat Enterprise Linux and CentOS from Percona repositories.

---

**Note:** *Percona Server for MongoDB* should work on other RPM-based distributions (for example, Amazon Linux AMI and Oracle Linux), but it is tested only on platforms listed on the [Percona Software and Platform Lifecycle](#) page.

---

- *Package Contents*
- *Installing from Percona repositories*
  - *Configure Percona repository*
  - *Install the latest version*
  - *Install a specific version*



- *Running Percona Server for MongoDB*
  - *Running after reboot*

## Package Contents

Package	Contains
percona-server-mongodb	The mongo shell, import/export tools, other client utilities, server software, default configuration, and <code>init.d</code> scripts.
percona-server-mongodb-server	The <b>mongod</b> server, default configuration files, and <code>init.d</code> scripts
percona-server-mongodb-shell	The mongo shell
percona-server-mongodb-mongos	The mongos sharded cluster query router
percona-server-mongodb-tools	Mongo tools for high-performance MongoDB fork from Percona
percona-server-mongodb-debugsource	Debug sources for the server

## Installing from Percona repositories

Percona repositories are managed using the **percona-release** tool. So in order to install *Percona Server for MongoDB*, install **percona-release** first

### Configure Percona repository

1. Install **percona-release**:

```
$ sudo yum install https://repo.percona.com/yum/percona-release-latest.noarch.rpm
```

#### Example of output

```
Retrieving https://repo.percona.com/yum/percona-release-latest.noarch.rpm
Preparing... ##### [100%]
1:percona-release ##### [100%]
```

2. Enable the repository:

```
$ sudo percona-release enable psmdb-50 release
```

See also:

More information about how to use the **percona-release** tool <https://www.percona.com/doc/percona-repo-config/index.html>

### Install the latest version

To install the latest version of *Percona Server for MongoDB*, use the following command:

```
$ sudo yum install percona-server-mongodb
```

### Install a specific version

To install a specific version of *Percona Server for MongoDB*, do the following:

1. List available versions:

```
$ sudo yum list percona-server-mongodb --showduplicates
```

---

#### Sample Output

Available Packages		
percona-server-mongodb.x86_64	5.0.2-1.el8	psmdb-44-release-x86_64

---

2. Install a specific version packages. For example, to install *Percona Server for MongoDB* 5.0.2-1, run the following command:

```
$ sudo yum install percona-server-mongodb-5.0.2-1.el8
```

## Running Percona Server for MongoDB

---

**Note:** If you are using SELinux in enforcing mode, you must customize your SELinux user policies to allow access to certain `/sys` and `/proc` files for OS-level statistics. Also, you must customize directory and port access policies if you are using non-default locations.

Please refer to [Configure SELinux](#) section of MongoDB Documentation for policy configuration guidelines.

---

*Percona Server for MongoDB* stores data files in `/var/lib/mongodb/` by default. The configuration file is `/etc/mongod.conf`.

### Starting the service

*Percona Server for MongoDB* is not started automatically after installation. Start it manually using the following command:

```
$ sudo systemctl start mongod
```

### Confirming that service is running

Check the service status using the following command:

```
$ sudo systemctl status mongod
```

### Stopping the service

Stop the service using the following command:

```
$ sudo systemctl stop mongod
```

### Restarting the service

Restart the service using the following command:

```
$ sudo systemctl restart mongod
```

### Running after reboot

The mongod service is not automatically started after you reboot the system.

You can enable it using the systemctl utility:

```
$ sudo systemctl enable mongod
```

## 2.3.3 Installing Percona Server for MongoDB from Binary Tarball

You can find links to the binary tarballs under the *Generic Linux* menu item on the [Percona website](#)

There are two tarballs available:

- `percona-server-mongodb-<version>-x86_64.glibc2.17.tar.gz` is the general tarball, compatible with any [supported operating system](#) except Ubuntu 22.04.
- `percona-server-mongodb-<version>-x86_64.glibc2.35.tar.gz` is the tarball for Ubuntu 22.04.

### Procedure

1. Fetch and extract the binary tarball. For example, if you are running Debian 10 (“buster”), use the following command:

```
$ wget https://www.percona.com/downloads/percona-server-mongodb-5.0/percona-server-  
↪mongodb-5.0.2-1/binary/tarball/percona-server-mongodb-5.0.2-1-x86_64.glibc2.17.  
↪tar.gz\  
$ tar -xf percona-server-mongodb-5.0.2-1-x86_64.glibc2.17.tar.gz
```

2. Add the location of the binaries to the PATH variable:

```
$ export PATH=~/.percona-server-mongodb-5.0.2-1/bin/:$PATH
```

3. Create the default data directory:

```
$ mkdir -p /data/db
```

4. Make sure that you have read and write permissions for the data directory and run `mongod`.

### 2.3.4 Running *Percona Server for MongoDB* in a Docker Container

Docker images of *Percona Server for MongoDB* are hosted publicly on Docker Hub at <https://hub.docker.com/r/percona/percona-server-mongodb/>.

For more information about using Docker, see the [Docker Docs](#).

---

**Note:** Make sure that you are using the latest version of Docker. The ones provided via `apt` and `yum` may be outdated and cause errors.

---

---

**Note:** By default, Docker will pull the image from Docker Hub if it is not available locally.

---

To run the latest *Percona Server for MongoDB* 5.0 in a Docker container, use the following command:

Run this command as root or by using the **sudo** command

```
$ docker run -d --name psmdb --restart always \
percona/percona-server-mongodb:5.0
```

The previous command does the following:

- The `docker run` command instructs the `docker` daemon to run a container from an image.
- The `-d` option starts the container in detached mode (that is, in the background).
- The `--name` option assigns a custom name for the container that you can use to reference the container within a Docker network. In this case: `psmdb`.
- The `--restart` option defines the container's restart policy. Setting it to `always` ensures that the Docker daemon will start the container on startup and restart it if the container exits.
- `percona/percona-server-mongodb:5.0` is the name and version tag of the image to derive the container from.

See also:

**Docker Documentation: the full list of tags** <https://hub.docker.com/r/percona/percona-server-mongodb/tags/>

### Connecting from Another Docker Container

The *Percona Server for MongoDB* container exposes standard MongoDB port (27017), which can be used for connection from an application running in another container. To link the application container to the `psmdb` container, use the `--link psmdb` option when running the container with your app.

### Connecting with the Mongo Shell

To start another container with the `mongo` shell that connects to your *Percona Server for MongoDB* container, run the following command: `docker run -it --link psmdb --rm percona/percona-server-mongodb:mongo mongo -h psmdb`

# **Part III**

## **Features**



## PERCONA MEMORY ENGINE

*Percona Memory Engine* is a special configuration of [WiredTiger](#) that does not store user data on disk. Data fully resides in the main memory, making processing much faster and smoother. Keep in mind that you need to have enough memory to hold the data set, and ensure that the server does not shut down.

- *Using Percona Memory Engine*
- *Switching storage engines*
- *Configuring Percona Memory Engine*

The *Percona Memory Engine* is available in *Percona Server for MongoDB* along with the default MongoDB engine *WiredTiger*.

### 3.1 Using Percona Memory Engine

As of version 3.2, *Percona Server for MongoDB* runs with [WiredTiger](#) by default. You can select a storage engine using the `--storageEngine` command-line option when you start `mongod`. Alternatively, you can set the `storage.engine` variable in the configuration file (by default, `/etc/mongod.conf`):

```
storage:  
  dbPath: <dataDir>  
  engine: inMemory
```

See also:

**MongoDB Documentation: Configuration File Options**

- [storage.engine Options](#)
- [storage.wiredTiger Options](#)
- [storage.inmemory Options](#)

## 3.2 Switching storage engines

### 3.2.1 Considerations

If you have data files in your database and want to change to Percona Memory Engine, consider the following:

- Data files created by one storage engine are not compatible with other engines, because each one has its own data model.
- When changing the storage engine, the **mongod** node requires an empty `dbPath` data directory when it is restarted. Though Percona Memory Engine stores all data in memory, some metadata files, diagnostics logs and statistics metrics are still written to disk. This is controlled with the `--inMemoryStatisticsLogDelaySecs` option.

Creating a new `dbPath` data directory for a different storage engine is the simplest solution. Yet when you switch between disk-using storage engines (e.g. from [WiredTiger](#) to [Percona Memory Engine](#)), you may have to delete the old data if there is not enough disk space for both. Double-check that your backups are solid and/or the replica set nodes are healthy before you switch to the new storage engine.

### 3.2.2 Procedure

To change a storage engine, you have the following options:

- If you simply want to temporarily test Percona Memory Engine, set a different data directory for the `dbPath` variable in the configuration file. Make sure that the user running **mongod** has read and write permissions for the new data directory.

```
$ service mongod stop
$ # In the configuration file, set the inmemory
$ # value for the storage.engine variable
$ # Set the <newDataDir> for the dbPath variable
$ service mongod start
```

- If you want to permanently switch to Percona Memory Engine and do not have any valuable data in your database, clean out the `dbPath` data directory (by default, `/var/lib/mongodb`) and edit the configuration file:

```
$ service mongod stop
$ rm -rf <dbpathDataDir>
$ # Update the configuration file by setting the new
$ # value for the storage.engine variable
$ # set the engine-specific settings such as
$ # storage.inMemory.engineConfig.inMemorySizeGB
$ service mongod start
```

- If there is data that you want to migrate and make compatible with Percona Memory Engine, use the following methods:

- for replica sets, use the “rolling restart” process. Switch to the Percona Memory Engine on the secondary node. Clean out the `dbPath` data directory and edit the configuration file:

```
$ service mongod stop
$ rm -rf <dbpathDataDir>
$ # Update the configuration file by setting the new
$ # value for the storage.engine variable
$ # set the engine-specific settings such as
```

(continues on next page)



(continued from previous page)

```
$ # storage.inMemory.engineConfig.inMemorySizeGB
$ service mongod start
```

Wait for the node to rejoin with the other nodes and report the SECONDARY status.

Repeat the procedure to switch the remaining nodes to Percona Memory Engine.

- for a standalone instance or a single-node replica set, use the mongodump and mongorestore utilities:

```
$ mongodump --out <dumpDir>
$ service mongod stop
$ rm -rf <dbpathDataDir>
$ # Update the configuration file by setting the new
$ # value for the storage.engine variable
$ # set the engine-specific settings such as
$ # storage.inMemory.engineConfig.inMemorySizeGB
$ service mongod start
$ mongorestore <dumpDir>
```

### 3.2.3 Switching engines with encrypted data

Using *Data at Rest Encryption* means using the same `storage.*` configuration options as for *WiredTiger*. To change from normal to *Data at Rest Encryption* mode or backward, you must clean up the `dbPath` data directory, just as if you change the storage engine. This is because `mongod` cannot convert the data files to an encrypted format ‘in place’. It must get the document data again either via the initial sync from another replica set member, or from imported backup dump.

## 3.3 Configuring Percona Memory Engine

You can configure the Percona Memory Engine using either command-line options or corresponding parameters in the `/etc/mongod.conf` file. The configuration file is formatted in YAML. For example:

```
storage:
  engine: inMemory
  inMemory:
    engineConfig:
      inMemorySizeGB: 140
      statisticsLogDelaySecs: 0
```

Setting parameters in the previous example configuration file is the same as starting the `mongod` daemon with the following options:

```
$ mongod --storageEngine=inMemory \
--inMemorySizeGB=140 \
--inMemoryStatisticsLogDelaySecs=0
```

The following options are available (with corresponding YAML configuration file parameters):

#### --inMemorySizeGB

**Config** `storage.inMemory.engineConfig.inMemorySizeGB`

**Default** 50% of total memory minus 1024 MB, but not less than 256 MB

Specifies the maximum memory in gigabytes to use for data.

**--inMemoryStatisticsLogDelaySecs**

**Config** `storage.inMemory.engineConfig.statisticsLogDelaySecs`

**Default** 0

Specifies the number of seconds between writes to statistics log. If 0 is specified then statistics are not logged.

## HOT BACKUP

*Percona Server for MongoDB* includes an integrated open-source hot backup system for the default [WiredTiger](#) storage engine. It creates a physical data backup on a running server without notable performance and operating degradation.

**Note:** Hot backups are done on `mongod` servers independently, without synchronizing them across replica set members and shards in a cluster. To ensure data consistency during backups and restores, we recommend using [Percona Backup for MongoDB](#).

---

- *Making a backup*
- *Saving a backup to a TAR archive*
- *Streaming hot backups to a remote destination*
  - *Credentials*
  - *Examples*
- *Restoring data from backup*

### 4.1 Making a backup

To take a hot backup of the database in your current `dbpath`, do the following:

- Make sure to provide access to the backup directory for the `mongod` user

```
chown mongod:mongod <backupDir>
```

- Run the `createBackup` command as administrator on the `admin` database and specify the backup directory.

```
> use admin
switched to db admin
> db.runCommand({createBackup: 1, backupDir: <backup_data_path>})
{ "ok" : 1 }
```

The backup taken is the snapshot of the `mongod` server's `dataDir` at the moment of the `createBackup` command start.

If the backup was successful, you should receive an `{ "ok" : 1 }` object. If there was an error, you will receive a failing `ok` status with the error message, for example:

```
> db.runCommand({createBackup: 1, backupDir: ""})
{ "ok" : 0, "errmsg" : "Destination path must be absolute" }
```

## 4.2 Saving a backup to a TAR archive

---

### Implementation details

This feature was implemented in *Percona Server for MongoDB* 4.2.1-1.

---

To save a backup in the format of *tar* archive, use the *archive* field to specify the destination path:

```
> use admin
...
> db.runCommand({createBackup: 1, archive: <path_to_archive>.tar })
```

## 4.3 Streaming hot backups to a remote destination

*Percona Server for MongoDB* enables uploading hot backups to an [Amazon S3](#) or a compatible storage service, such as [MinIO](#).

This method requires that you provide the *bucket* field in the *s3* object:

```
> use admin
...
> db.runCommand({createBackup: 1, s3: {bucket: "backup20190510", path: <some_optional_
↪path>} })
```

In addition to the mandatory *bucket* field, the *s3* object may contain the following fields:

Field	Type	Description
bucket	string	The only mandatory field. Names are subject to restrictions described in the <a href="#">Bucket Restrictions and Limitations</a> section of <a href="#">Amazon S3 documentation</a>
path	string	The virtual path inside the specified bucket where the backup will be created. If the <i>path</i> is not specified then the backup is created in the root of the bucket. If there are any objects under the specified path, the backup will not be created and an error will be reported.
endpoint	string	The endpoint address and port - mainly for AWS S3 compatible servers such as the <i>MinIO</i> server. For a local MinIO server, this can be “127.0.0.1:9000”. For AWS S3 this field can be omitted.
scheme	string	“HTTP” or “HTTPS” (default). For a local MinIO server started with the <i>minio server</i> command this should field should contain <i>HTTP</i> .
useVirtualAddressing	bool	The style of addressing buckets in the URL. By default ‘true’. For MinIO servers, set this field to <b>false</b> . For more information, see <a href="#">Virtual Hosting of Buckets</a> in the Amazon S3 documentation.
region	string	The name of an AWS region. The default region is <b>US_EAST_1</b> . For more information see <a href="#">AWS Service End-points</a> in the Amazon S3 documentation.
profile	string	The name of a credentials profile in the <i>credentials</i> configuration file. If not specified, the profile named <b>default</b> is used.
accessKeyId	string	The access key id
secretAccessKey	string	The secret access key

### 4.3.1 Credentials

If the user provides the *access key id* and the *secret access key* parameters, these are used as credentials.

If the *access key id* parameter is not specified then the credentials are loaded from the credentials configuration file. By default, it is `~/aws/credentials`.

#### An example of the credentials file

```
[default]
aws_access_key_id = ABC123XYZ456QQAAFF
aws_secret_access_key = zuf+secretkey0secretkey1secretkey2
[localminio]
aws_access_key_id = ABCABCABCABC55566678
aws_secret_access_key = secretaccesskey1secretaccesskey2secretaccesskey3
```

### 4.3.2 Examples

#### Backup in root of bucket on local instance of MinIO server

```
> db.runCommand({createBackup: 1, s3: {bucket: "backup20190901500",
scheme: "HTTP",
endpoint: "127.0.0.1:9000",
useVirtualAddressing: false,
profile: "localminio"}}})
```

#### Backup on MinIO testing server with the default credentials profile

The following command creates a backup under the virtual path “year2019/day42” in the *backup* bucket:

```
> db.runCommand({createBackup: 1, s3: {bucket: "backup",
path: "year2019/day42",
endpoint: "sandbox.min.io:9000",
useVirtualAddressing: false}}})
```

#### Backup on AWS S3 service using default settings

```
> db.runCommand({createBackup: 1, s3: {bucket: "backup", path: "year2019/day42"}}})
```

See also:

**AWS Documentation: Providing AWS Credentials** <https://docs.aws.amazon.com/sdk-for-cpp/v1/developer-guide/credentials.html>

## 4.4 Restoring data from backup

### Restoring from backup on a standalone server

To restore your database on a standalone server, stop the mongod service, clean out the data directory and copy files from the backup directory to the data directory. The mongod user requires access to those files to start the service. Therefore, make the mongod user the owner of the data directory and all files and subdirectories under it, and restart the mongod service.

Run the following commands as root or by using the **sudo** command

```
# Stop the mongod service
$ systemctl stop mongod
# Clean out the data directory
$ rm -rf /var/lib/mongodb/*
# Copy backup files
$ cp -RT <backup_data_path> /var/lib/mongodb/
# Grant permissions to data files for the mongod user
$ chown -R mongod:mongod /var/lib/mongodb/
# Start the mongod service
$ systemctl start mongod
```

## Restoring from backup in a replica set

The recommended way to restore the replica set from a backup is to restore it into a standalone node and then initiate it as the first member of a new replica set.

**Note:** If you try to restore the node into the existing replica set and there is more recent data, the restored node detects that it is out of date with the other replica set members, deletes the data and makes an initial sync.

Run the following commands as root or by using the **sudo** command

The restore steps are the following:

1. Stop the mongod service:

```
$ systemctl stop mongod
```

2. Clean the data directory and then copy the files from the backup directory to your data directory. Assuming that the data directory is `/var/lib/mongodb/`, use the following commands:

```
$ rm -rf /var/lib/mongodb/*  
$ cp -RT <backup_data_path> /var/lib/mongodb/
```

3. Grant permissions to the data files for the mongod user

```
$ chown -R mongod:mongod /var/lib/mongodb/
```

4. Make sure the replication is disabled in the config file and start the mongod service.

```
$ systemctl start mongod
```

5. Connect to your standalone node via the mongo shell and drop the local database

```
> mongo  
> use local  
> db.dropDatabase()
```

6. Restart the node with the replication enabled

- Shut down the node.

```
$ systemctl stop mongod
```

- Edit the configuration file and specify the `replication.replSetName` option
- Start the mongod node:

```
$ systemctl start mongod
```

7. Initiate a new replica set

```
# Start the mongo shell  
> mongo  
# Initiate a new replica set  
> rs.initiate()
```





## PROFILING RATE LIMIT

*Percona Server for MongoDB* can limit the number of queries collected by the database profiler to decrease its impact on performance. Rate limit is an integer between 1 and 1000 and represents the fraction of queries to be profiled. For example, if you set it to 20, then every 20th query will be logged. For compatibility reasons, rate limit of 0 is the same as setting it to 1, and will effectively disable the feature meaning that every query will be profiled.

The MongoDB database profiler can operate in one of three modes:

- 0: Profiling is disabled. This is the default setting.
- 1: The profiler collects data only for *slow* queries. By default, queries that take more than 100 milliseconds to execute are considered *slow*.
- 2: Collects profiling data for all database operations.

Mode 1 ignores all *fast* queries, which may be the cause of problems that you are trying to find. Mode 2 provides a comprehensive picture of database performance, but may introduce unnecessary overhead.

With rate limiting you can collect profiling data for all database operations and reduce overhead by sampling queries. Slow queries ignore rate limiting and are always collected by the profiler.

### 5.1 Comparing to the `sampleRate` option

The `sampleRate` option (= `slowOpSampleRate` config file option) is a similar concept to `rateLimit`. But it works at different profile level, completely ignores operations faster than `slowOpsThresholdMs` (a.k.a. `slowMs`), and affects the log file printing, too.

	<code>sampleRate</code>	<code>rateLimit</code>
Affects profiling level 1	yes	no
Affects profiling level 2	no	yes
Discards/filters slow ops	yes	no
Discards/filters fast ops	no	yes
Affects log file	yes	no
Example value of option	0.02	50

`rateLimit` is a better way to have continuous profiling for monitoring or live analysis purposes. `sampleRate` requires setting `slowOpsThresholdMs` to zero if you want to sample all types of operations. `sampleRate` has an effect on the log file which may either decrease or increase the log volume.

## 5.2 Enabling the Rate Limit

To enable rate limiting, set the profiler mode to 2 and specify the value of the rate limit. Optionally, you can also change the default threshold for slow queries, which will not be sampled by rate limiting.

For example, to set the rate limit to 100 (profile every 100th *fast* query) and the slow query threshold to 200 (profile all queries slower than 200 milliseconds), run the mongod instance as follows:

```
$ mongod --profile 2 --slowms 200 --rateLimit 100
```

To do the same at runtime, use the `profile` command. It returns the *previous* settings and "ok" : 1 indicates that the operation was successful:

```
> db.runCommand( { profile: 2, slowms: 200, ratelimit: 100 } );
{ "was" : 0, "slowms" : 100, "ratelimit" : 1, "ok" : 1 }
```

To check the current settings, run `profile: -1`:

```
> db.runCommand( { profile: -1 } );
{ "was" : 2, "slowms" : 200, "ratelimit" : 100, "ok" : 1 }
```

If you want to set or get just the rate limit value, use the `profilingRateLimit` parameter on the `admin` database:

```
> db.getSiblingDB('admin').runCommand( { setParameter: 1, "profilingRateLimit": 100 } );
{ "was" : 1, "ok" : 1 }
> db.getSiblingDB('admin').runCommand( { getParameter: 1, "profilingRateLimit": 1 } );
{ "profilingRateLimit" : 100, "ok" : 1 }
```

If you want rate limiting to persist when you restart mongod, set the corresponding variables in the MongoDB configuration file (by default, `/etc/mongod.conf`):

```
operationProfiling:
  mode: all
  slowOpThresholdMs: 200
  rateLimit: 100
```

---

**Note:** The value of the `operationProfiling.mode` variable is a string, which you can set to either `off`, `slowOp`, or `all`, corresponding to profiling modes 0, 1, and 2.

---

## 5.3 Profiler Collection Extension

Each document in the `system.profile` collection includes an additional `rateLimit` field. This field always has the value of 1 for *slow* queries and the current rate limit value for *fast* queries.

## AUTHENTICATION

Authentication is the process of verifying a client's identity. Normally, a client needs to authenticate themselves against the MongoDB server user database before doing any work or reading any data from a `mongod` or `mongos` instance.

By default, *Percona Server for MongoDB* provides a SCRAM (Salted Challenge Response Authentication Mechanism) authentication mechanism where clients authenticate themselves by providing their user credentials. In addition, you can integrate *Percona Server for MongoDB* with a separate service, such as OpenLDAP or Active Directory. This enables users to access the database with the same credentials they use for their emails or workstations.

You can use any of these authentication mechanisms supported in *Percona Server for MongoDB*:

- *SCRAM* (default)
- *x.509 certificate authentication*
- *LDAP authentication with SASL*
- *Kerberos Authentication*
- *Authentication and authorization with direct binding to LDAP*

### 6.1 SCRAM

SCRAM is the default authentication mechanism. *Percona Server for MongoDB* verifies the credentials against the user's name, password and the database where the user record is created for a client (authentication database). For how to enable this mechanism, see *Enabling Authentication*.

### 6.2 x.509 certificate authentication

This authentication mechanism enables a client to authenticate in *Percona Server for MongoDB* by providing an x.509 certificate instead of user credentials. Each certificate contains the subject field defined in the DN (Distinguished Name) format. In *Percona Server for MongoDB*, each certificate has a corresponding user record in the `$external` database. When a user connects to the database, *Percona Server for MongoDB* matches the subject value against the usernames defined in the `$external` database.

For production use, we recommend using valid CA (Certified Authority) certificates. For testing purposes, you can generate and use self-signed certificates.

x.509 authentication is compatible with *LDAP authorization* to enable you to control user access and operations in *Percona Server for MongoDB*. For configuration guidelines, refer to *Set up x.509 authentication and LDAP authorization*.

See also:

MongoDB Documentation: [x.509](#)

Percona Blog: [Setting up MongoDB with Member x509 auth and SSL + easy-rsa](#)

## 6.3 LDAP authentication with SASL

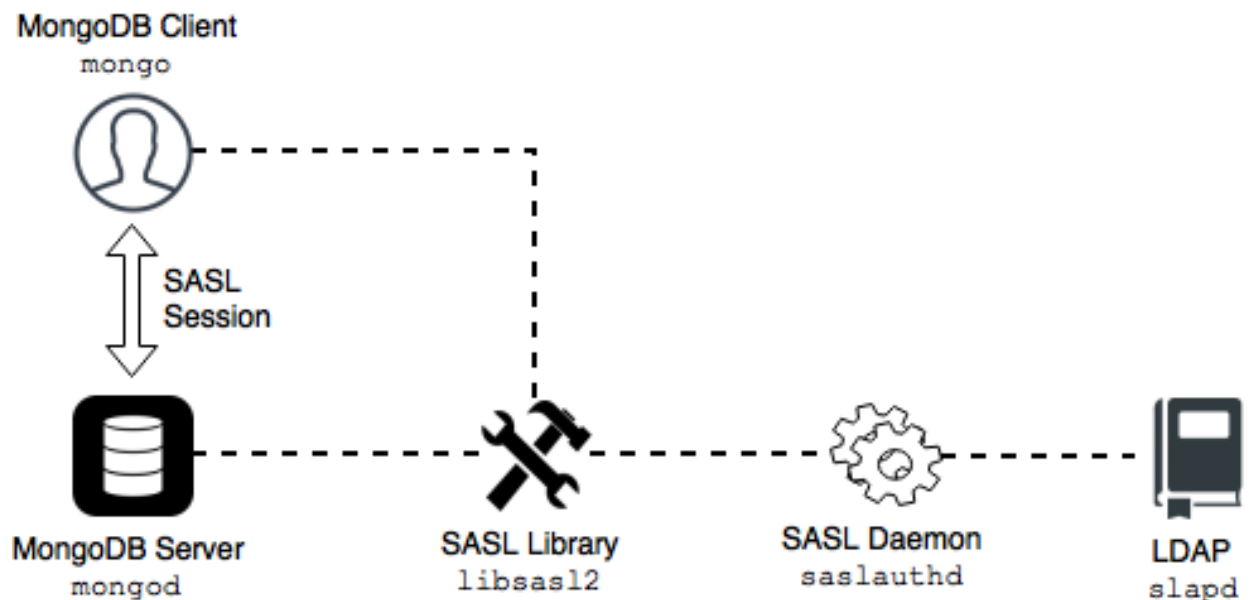
### 6.4 Overview

LDAP authentication with SASL (Simple Authentication and Security Layer) means that both the client and the server establish a SASL session using the SASL library. Then authentication (bind) requests are sent to the LDAP server through the SASL authentication daemon (`saslauthd`) that acts as a remote proxy for the `mongod` server.

The following components are necessary for external authentication to work:

- **LDAP Server:** Remotely stores all user credentials (i.e. user name and associated password).
- **SASL Daemon:** Used as a MongoDB server-local proxy for the remote LDAP service.
- **SASL Library:** Used by the MongoDB client and server to create data necessary for the authentication mechanism.

The following image illustrates this architecture:



An authentication session uses the following sequence:

1. A `mongo` client connects to a running `mongod` instance.
2. The client creates a PLAIN authentication request using the SASL library.
3. The client then sends this SASL request to the server as a special Mongo command.
4. The `mongod` server receives this SASL message, with its authentication request payload.
5. The server then creates a SASL session scoped to this client, using its own reference to the SASL library.
6. Then the server passes the authentication payload to the SASL library, which in turn passes it on to the `saslauthd` daemon.

7. The `saslauthd` daemon passes the payload on to the LDAP service to get a YES or NO authentication response (in other words, does this user exist and is the password correct).
8. The YES/NO response moves back from `saslauthd`, through the SASL library, to `mongod`.
9. The `mongod` server uses this YES/NO response to authenticate the client or reject the request.
10. If successful, the client has authenticated and can proceed.

For configuration instructions, refer to [Setting up LDAP authentication with SASL](#).

## 6.5 Kerberos Authentication

*Percona Server for MongoDB* supports Kerberos authentication starting from release 4.2.6-6.

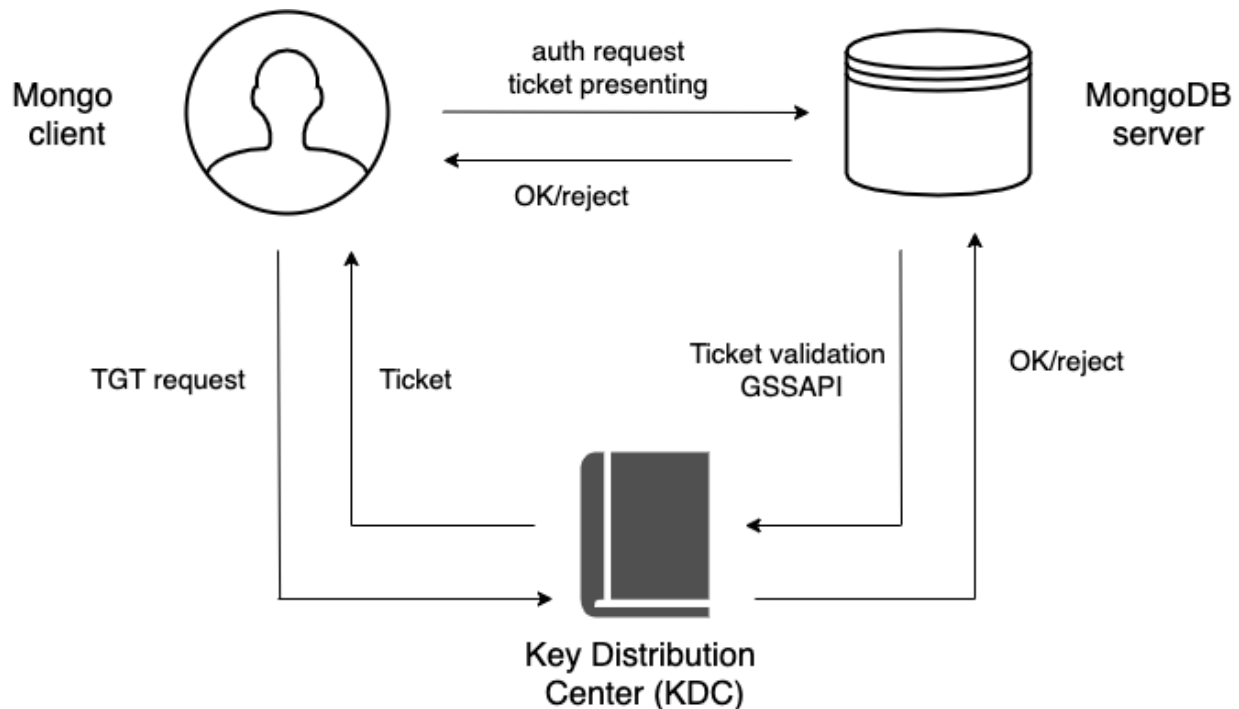
This authentication mechanism involves the use of a *Key Distribution Center (KDC)* - a symmetric encryption component which operates with tickets. A *ticket* is a small amount of encrypted data which is used for authentication. It is issued for a user session and has a limited lifetime.

When using Kerberos authentication, you also operate with principals and realms.

A *realm* is the logical network, similar to a domain, for all Kerberos nodes under the same master KDC.

A *principal* is a user or a service which is known to Kerberos. A principal name is used for authentication in Kerberos. A service principal represents the service, e.g. `mongodb`. A user principal represents the user. The user principal name corresponds to the username in the `$external` database in *Percona Server for MongoDB*.

The following diagram shows the authentication workflow:



The sequence is the following:

1. A mongo client sends the Ticket-Granting Ticket (TGT) request to the Key Distribution Center (KDC)
2. The KDC issues the ticket and sends it to the mongo client.
3. The mongo client sends the authentication request to the mongo server presenting the ticket.

4. The mongo server validates the ticket in the KDC.
5. Upon successful ticket validation, the authentication request is approved and the user is authenticated.

Kerberos authentication in *Percona Server for MongoDB* is implemented the same way as in MongoDB Enterprise.

**See also:**

MongoDB Documentation: [Kerberos Authentication](#)

## LDAP AUTHORIZATION

LDAP authorization allows you to control user access and operations in your database environment using the centralized user management storage – an LDAP server. You create and manage user credentials and permission information in the LDAP server. In addition, you create roles in the `admin` database with the names that exactly match the LDAP group Distinguished Name. These roles define what privileges the users who belong to the corresponding LDAP group.

### 7.1 Supported authentication mechanisms

LDAP authorization is compatible with the following authentication mechanisms:

- *x.509 certificate authentication*
- *Kerberos Authentication*
- *Authentication and authorization with direct binding to LDAP*

### 7.2 Authentication and authorization with direct binding to LDAP

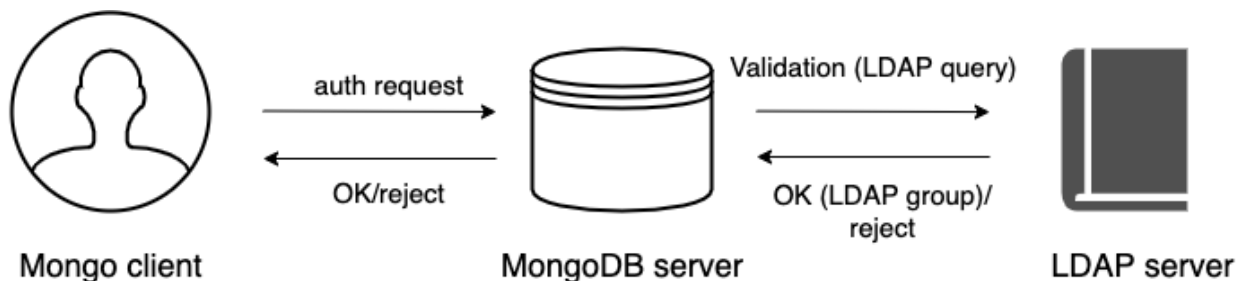
Starting with release 4.2.5-5, you can configure *Percona Server for MongoDB* to communicate with the LDAP server directly to authenticate and also authorize users.

The advantage of using this mechanism is that it is easy to setup and does not require pre-creating users in the dummy `$external` db. Nevertheless, the `--authenticationDatabase` connection argument will still need to be specified as `$external`.

The following example illustrates the connection to *Percona Server for MongoDB* from the `mongo` shell:

```
mongo -u "CN=alice,CN=Users,DC=engineering,DC=example,DC=com" -p --  
↪ authenticationDatabase '$external' --authenticationMechanism PLAIN
```

The following diagram illustrates the authentication and authorization flow:



1. A user connects to the db providing their credentials
2. If required, *Percona Server for MongoDB* transforms the *username* to match the DN in the LDAP server according to the mapping rules specified for the `--ldapUserToDNMapping` parameter.
3. *Percona Server for MongoDB* queries the LDAP server for the user identity and /or the LDAP groups this user belongs to.
4. The LDAP server evaluates the query and if a user exists, returns their LDAP groups.
5. *Percona Server for MongoDB* authorizes the user by mapping the DN of the returned groups against the roles assigned to the user in the `admin` database. If a user belongs to several groups they receive permissions associated with every group.

## 7.2.1 Username transformation

If clients connect to *Percona Server for MongoDB* with usernames that are not LDAP DN, these usernames must be converted to the format acceptable by LDAP.

To achieve this, the `--ldapUserToDNMapping` parameter is available in *Percona Server for MongoDB* configuration.

The `--ldapUserToDNMapping` parameter is a JSON string representing an ordered array of rules expressed as JSON documents. Each document provides a regex pattern (`match` field) to match against a provided username. If that pattern matches, there are two ways to continue:

- If there is the `substitution` value, then the matched pattern becomes the username of the user for further processing.
- If there is the `ldapQuery` value, the matched pattern is sent to the LDAP server and the result of that LDAP query becomes the DN of the user for further processing.

Both `substitution` and `ldapQuery` should contain placeholders to insert parts of the original username – those placeholders are replaced with regular expression submatches found on the `match` stage.

So having an array of documents, *Percona Server for MongoDB* tries to match each document against the provided name and if it matches, the name is replaced either with the substitution string or with the result of the LDAP query.

## LDAP referrals

As of version 4.2.10-11, *Percona Server for MongoDB* supports LDAP referrals as defined in [RFC 4511 4.1.10](#). For security reasons, referrals are disabled by default. Double-check that using referrals is safe before enabling them.

To enable LDAP referrals, set the `ldapFollowReferrals` server parameter to `true` using the *setParameter* command or by editing the configuration file.

```
setParameter:
  ldapFollowReferrals: true
```



## Connection pool

As of version 4.2.10-11, *Percona Server for MongoDB* always uses a connection pool to LDAP server to process bind requests. The connection pool is enabled by default. The default connection pool size is 2 connections.

You can change the connection pool size either at the server startup or dynamically by specifying the value for the `ldapConnectionPoolSizePerHost` server parameter.

For example, to set the number of connections in the pool to 5, use the `setParameter` command:

### Command line

```
$ db.adminCommand( { setParameter: 1, ldapConnectionPoolSizePerHost: 5 } )
```

### Configuration file:

```
setParameter:
  ldapConnectionPoolSizePerHost: 5
```

## Support for multiple LDAP servers

As of version 4.2.12-13, you can specify multiple LDAP servers for failover. *Percona Server for MongoDB* sends bind requests to the first server defined in the list. When this server is down or unavailable, it sends requests to the next server and so on. Note that *Percona Server for MongoDB* keeps sending requests to this server even after the unavailable server recovers.

Specify the LDAP servers as a comma-separated list in the format `<host>:<port>` for the `-ldapServers` option.

You can define the option value at the server startup by editing the configuration file.

```
security:
  authorization: "enabled"
  ldap:
    servers: "ldap1.example.net,ldap2.example.net"
```

You can change `ldapServers` dynamically at runtime using the `setParameter`.

```
$ db.adminCommand( { setParameter: 1, ldapServers:"localhost,ldap1.example.net,ldap2.
↪example.net"} )
{ "was" : "ldap1.example.net,ldap2.example.net", "ok" : 1 }
```

See also:

### MongoDB Documentation:

- [Authenticate and Authorize Users Using Active Directory via Native LDAP](#)
- [LDAP referrals](#)

## 7.3 Configuration

For how to configure LDAP authorization with the native LDAP authentication, see *Setting up LDAP authentication and authorization using NativeLDAP*.

## AUDITING

Auditing allows administrators to track and log user activity on a MongoDB server. With auditing enabled, the server will generate an audit log file. This file contains information about different user events including authentication, authorization failures, and so on.

To enable audit logging, specify where to send audit events using the `--auditDestination` option on the command line or the `auditLog.destination` variable in the configuration file.

If you want to output events to a file, also specify the format of the file using the `--auditFormat` option or the `auditLog.format` variable, and the path to the file using the `--auditPath` option or the `auditLog.path` variable.

To filter recorded events, use the `--auditFilter` option or the `auditLog.filter` variable.

For example, to log only events from a user named *tim* and write them to a JSON file `/var/log/psmdb/audit.json`, start the server with the following parameters:

```
mongod \
--dbpath data/db
--auditDestination file \
--auditFormat JSON \
--auditPath /var/log/psmdb/audit.json \
--auditFilter '{ "users.user" : "tim" }'
```

The options in the previous example can be used as variables in the MongoDB configuration file:

```
storage:
  dbPath: data/db
auditLog:
  destination: file
  format: JSON
  path: /var/log/psmdb/audit.json
  filter: '{ "users.user" : "tim" }'
```

This example shows how to send audit events to the `syslog`. Specify the following parameters:

```
mongod \
--dbpath data/db
--auditDestination syslog \
```

Alternatively, you can edit the MongoDB configuration file:

```
storage:
  dbPath: data/db
```

(continues on next page)

(continued from previous page)

```
auditLog:
  destination: syslog
```

---

**Note:** If you start the server with auditing enabled, it cannot be disabled dynamically during runtime.

---

## 8.1 Audit Options

The following options control audit logging:

### **--auditDestination**

**Variable** auditLog.destination

**Type** String

Enables auditing and specifies where to send audit events:

- **console:** Output audit events to `stdout`.
- **file:** Output audit events to a file specified by the `--auditPath` option in a format specified by the `--auditFormat` option.
- **syslog:** Output audit events to `syslog`.

### **--auditFilter**

**Variable** auditLog.filter

**Type** String

Specifies a filter to apply to incoming audit events, enabling the administrator to only capture a subset of them. The value must be interpreted as a query object with the following syntax:

```
{ <field1>: <expression1>, ... }
```

Audit log events that match this query will be logged. Events that do not match this query will be ignored.

For more information, see [Audit Filter Examples](#).

### **--auditFormat**

**Variable** auditLog.format

**Type** String

Specifies the format of the audit log file, if you set the `--auditDestination` option to `file`.

The default value is JSON. Alternatively, you can set it to BSON.

### **--auditPath**

**Variable** auditLog.path

**Type** String

Specifies the fully qualified path to the file where audit log events are written, if you set the `--auditDestination` option to `file`.

If this option is not specified, then the `auditLog.json` file is created in the server's configured log path. If log path is not configured on the server, then the `auditLog.json` file is created in the current directory (from which `mongod` was started).

**Note:** This file will rotate in the same manner as the system log path, either on server reboot or using the `logRotate` command. The time of rotation will be added to the old file's name.

## 8.2 Audit Message Syntax

Audit logging writes messages in JSON format with the following syntax:

```
{
  atype: <String>,
  ts : { "$date": <timestamp> },
  local: { ip: <String>, port: <int> },
  remote: { ip: <String>, port: <int> },
  users : [ { user: <String>, db: <String> }, ... ],
  roles: [ { role: <String>, db: <String> }, ... ],
  param: <document>,
  result: <int>
}
```

**atype** Event type

**ts** Date and UTC time of the event

**local** Local IP address and port number of the instance

**remote** Remote IP address and port number of the incoming connection associated with the event

**users** Users associated with the event

**roles** Roles granted to the user

**param** Details of the event associated with the specific type

**result** Exit code (0 for success)

## 8.3 Audit Filter Examples

The following examples demonstrate the flexibility of audit log filters.

- *Standard Query Selectors*
- *Regular Expressions*
- *Read and Write Operations*

```
auditLog:
  destination: file
  filter: '{atype: {$in: [
    "authenticate", "authCheck",
```

(continues on next page)

(continued from previous page)

```

    "renameCollection", "dropCollection", "dropDatabase",
    "createUser", "dropUser", "dropAllUsersFromDatabase", "updateuser",
    "grantRolesToUser", "revokeRolesFromUser", "createRole", "updateRole",
    "dropRole", "dropAllRolesFromDatabase", "grantRolesToRole", "revokeRolesFromRole
    ↪",
    "grantPrivilegesToRole", "revokePrivilegesFromRole",
    "replSetReconfig",
    "enableSharding", "shardCollection", "addShard", "removeShard",
    "shutdown",
    "applicationMessage"
  ]}]'
```

### 8.3.1 Standard Query Selectors

You can use query selectors, such as \$eq, \$in, \$gt, \$lt, \$ne, and others to log multiple event types.

For example, to log only the dropCollection and dropDatabase events:

- Command line:

```
--auditDestination file --auditFilter '{ atype: { $in: [ "dropCollection",
↪ "dropDatabase" ] } }'
```

- Config file:

```
auditLog:
  destination: file
  filter: '{ atype: { $in: [ "dropCollection", "dropDatabase" ] } }'
```

### 8.3.2 Regular Expressions

Another way to specify multiple event types is using regular expressions.

For example, to filter all drop operations:

- Command line:

```
--auditDestination file --auditFilter '{ "atype" : /^drop.*/ }'
```

- Config file:

```
auditLog:
  destination: file
  filter: '{ "atype" : /^drop.*/ }'
```

### 8.3.3 Read and Write Operations

By default, operations with successful authorization are not logged, so for this filter to work, enable `auditAuthorizationSuccess` parameter, as described in *Enabling Auditing of Authorization Success*.

For example, to filter read and write operations on all the collections in the `test` database:

**Note:** The dot (.) after the database name in the regular expression must be escaped with two backslashes (\\).

- Command line:

```
--setParameter auditAuthorizationSuccess=true --auditDestination file --auditFilter
↪ '{ atype: "authCheck", "param.command": { $in: [ "find", "insert", "delete",
↪ "update", "findandmodify" ] }, "param.ns": /^test\\.\/ } }'
```

- Config file:

```
auditLog:
  destination: file
  filter: '{ atype: "authCheck", "param.command": { $in: [ "find", "insert", "delete
↪ ", "update", "findandmodify" ] }, "param.ns": /^test\\.\/ } }'

setParameter: { auditAuthorizationSuccess: true }
```

## 8.4 Enabling Auditing of Authorization Success

By default, only authorization failures for the `authCheck` action are logged by the audit system. `authCheck` is for authorization by role-based access control, it does not concern authentication at logins.

To enable logging of authorization successes, set the `auditAuthorizationSuccess` parameter to `true`. Audit events will then be triggered by every command, including CRUD ones.

**Warning:** Enabling the `auditAuthorizationSuccess` parameter heavily impacts the performance compared to logging only authorization failures.

You can enable it on a running server using the following command:

```
db.adminCommand( { setParameter: 1, auditAuthorizationSuccess: true } )
```

To enable it on the command line, use the following option when running `mongod` or `mongos` process:

```
--setParameter auditAuthorizationSuccess=true
```

You can also add it to the configuration file as follows:

```
setParameter:
  auditAuthorizationSuccess: true
```





## LOG REDACTION

*Percona Server for MongoDB* can prevent writing sensitive data to the diagnostic log by redacting messages of events before they are logged. To enable log redaction, run `mongod` with the `--redactClientLogData` option.

---

**Note:** Metadata such as error or operation codes, line numbers, and source file names remain visible in the logs.

---

Log redaction is important for complying with security requirements, but it can make troubleshooting and diagnostics more difficult due to the lack of data related to the log event. For this reason, debug messages are not redacted even when log redaction is enabled. Keep this in mind when switching between log levels.

You can permanently enable log redaction by adding the following to the configuration file:

```
security:
  redactClientLogData: true
```

To enable log redaction at runtime, use the `setParameter` command as follows:

```
db.adminCommand(
  { setParameter: 1, redactClientLogData : true }
)
```



## DATA AT REST ENCRYPTION

- *Encrypting Rollback Files*

Data at rest encryption for the WiredTiger storage engine in *MongoDB* was introduced in MongoDB Enterprise version 3.2 to ensure that encrypted data files can be decrypted and read by parties with the decryption key.

### Differences from Upstream

The data encryption at rest in *Percona Server for MongoDB* is introduced in version 3.6 to be compatible with data encryption at rest interface in *MongoDB*. In the current release of *Percona Server for MongoDB*, the data encryption at rest does not include support for Amazon AWS key management service. Instead, *Percona Server for MongoDB* is *integrated with HashiCorp Vault*. Starting with release 5.0.7-6, *Percona Server for MongoDB* supports the secure transfer of keys using *Key Management Interoperability Protocol (KMIP)*. This allows users to store encryption keys in their favorite KMIP-compatible key manager when they set up encryption at rest.

Two types of keys are used for data at rest encryption:

- Database keys to encrypt data. They are stored internally, near the data that they encrypt.
- The master key to encrypt database keys. It is kept separately from the data and database keys and requires external management.

To manage the master key, use one of the supported key management options:

- Integration with an external key server (recommended). *Percona Server for MongoDB* is *integrated with HashiCorp Vault* and supports the secure transfer of keys using *Key Management Interoperability Protocol (KMIP)*.
- *Local key management using a keyfile.*

Note that you can use only one of the key management options at a time. However, you can switch from one management option to another (e.g. from a keyfile to HashiCorp Vault). Refer to *Migrating from Key File Encryption to HashiCorp Vault Encryption* section for details.

---

**Important:** You can only enable data at rest encryption and provide all encryption settings on an empty database, when you start the `mongod` instance for the first time. You cannot enable or disable encryption while the *Percona Server for MongoDB* server is already running and / or has some data. Nor can you change the effective encryption mode by simply restarting the server. Every time you restart the server, the encryption settings must be the same.

---

## 10.1 HashiCorp Vault Integration

*Percona Server for MongoDB* is integrated with HashiCorp Vault. HashiCorp Vault supports different secrets engines. *Percona Server for MongoDB* only supports the HashiCorp Vault back end with KV Secrets Engine - Version 2 (API) with versioning enabled.

### See also:

Percona Blog: [Using Vault to Store the Master Key for Data at Rest Encryption on Percona Server for MongoDB](#)

HashiCorp Vault Documentation: [How to configure the KV Engine](#)

### HashiCorp Vault Parameters

Command line	Config file	Type	Description
<code>vaultServerName</code>	<code>security.vault.serverName</code>	string	The IP address of the Vault server
<code>vaultPort</code>	<code>security.vault.port</code>	int	The port on the Vault server
<code>vaultTokenFile</code>	<code>security.vault.tokenFile</code>	string	The path to the vault token file. The token file is used by <i>MongoDB</i> to access HashiCorp Vault. The vault token file consists of the raw vault token and does not include any additional strings or parameters. Example of a vault token file:  <code>s.uTrHtzsZnEE7KyHeA797CkWA</code>
<code>vaultSecret</code>	<code>security.vault.secret</code>	string	The path to the vault secret. Every replica set member must have its own distinct vault secret. It is recommended to use different secret paths for every database node. Note that vault secrets path format must be:  <code>&lt;vault_secret_mount&gt;/data/ ↳&lt;custom_path&gt;</code> where: <ul style="list-style-type: none"> <li>• <code>&lt;vault_secret_mount&gt;</code> is your Vault KV Secrets Engine;</li> <li>• <code>data</code> is the mandatory path prefix required by Version 2 API;</li> <li>• <code>&lt;custom_path&gt;</code> is your secrets path</li> </ul> Example:  <code>secret_v2/data/psmdb-test/ ↳rs1-27017</code>
<code>vaultRotateMasterKey</code>	<code>security.vault.rotateMasterKey</code>	switch	Enables master key rotation
<code>vaultServerCAFile</code>	<code>security.vault.serverCAFile</code>	string	The path to the TLS certificate file
<code>vaultDisableTLSForTesting</code>	<code>security.vault.disableTLSForTesting</code>	switch	Disables secure connection to HashiCorp Vault using SSL/TLS client certificates

---

### Config file example

```
security:
  enableEncryption: true
vault:
  serverName: 127.0.0.1
  port: 8200
  tokenFile: /home/user/path/token
  secret: secret/data/hello
```

---

During the first run of the *Percona Server for MongoDB*, the process generates a secure key and writes the key to the vault.

During the subsequent start, the server tries to read the master key from the vault. If the configured secret does not exist, vault responds with HTTP 404 error.

## 10.1.1 Namespaces

Namespaces are isolated environments in Vault that allow for separate secret key and policy management.

You can use Vault namespaces with *Percona Server for MongoDB*. Specify the namespace(s) for the `security.vault.secret` option value as follows:

```
<namespace>/secret/data/<secret_path>
```

For example, the path to secret keys for namespace `test` on the secrets engine `secret` will be `test/secret/<my_secret_path>`.

---

**Note:** You have the following options of how to target a particular namespace when configuring Vault:

1. Set the `VAULT_NAMESPACE` environment variable so that all subsequent commands are executed against that namespace. Use the following command to set the environment variable for the namespace `test`:

```
$ export VAULT_NAMESPACE=test
```

2. Provide the namespace with the `-namespace` flag in commands
- 

### See also:

HashiCorp Vault Documentation:

- **Namespaces** <https://www.vaultproject.io/docs/enterprise/namespaces>
- **Secure Multi-Tenancy with Namespaces** <https://learn.hashicorp.com/tutorials/vault/namespaces>

### 10.1.2 Key Rotation

Key rotation is replacing the old master key with a new one. This process helps to comply with regulatory requirements.

To rotate the keys for a single `mongod` instance, do the following:

1. Stop the `mongod` process
2. Add `--vaultRotateMasterKey` option via the command line or `security.vault.rotateMasterKey` to the config file.
3. Run the `mongod` process with the selected option, the process will perform the key rotation and exit.
4. Remove the selected option from the startup command or the config file.
5. Start `mongod` again.

Rotating the master key process also re-encrypts the keystore using the new master key. The new master key is stored in the vault. The entire dataset is not re-encrypted.

#### Key rotation in replica sets

Every `mongod` node in a replica set must have its own master key. The key rotation steps are the following:

1. Rotate the master key for the secondary nodes one by one.
2. Step down the primary and wait for another primary to be elected.
3. Rotate the master key for the previous primary node.

## 10.2 Local key management using a keyfile

The key file must contain a 32 character string encoded in base64. You can generate a random key and save it to a file by using the `openssl` command:

```
$ openssl rand -base64 32 > mongodb-keyfile
```

Then, as the owner of the `mongod` process, update the file permissions: only the owner should be able to read and modify this file. The effective permissions specified with the `chmod` command can be:

- **600** - only the owner may read and modify the file
- **400** - only the owner may read the file.

```
$ chmod 600 mongodb-keyfile
```

Enable the data encryption at rest in *Percona Server for MongoDB* by setting these options:

- `--enableEncryption` to enable data at rest encryption
- `--encryptionKeyFile` to specify the path to a file that contains the encryption key

```
$ mongod ... --enableEncryption --encryptionKeyFile <fileName>
```

By default, *Percona Server for MongoDB* uses the AES256-CBC cipher mode. If you want to use the AES256-GCM cipher mode, then use the `encryptionCipherMode` parameter to change it.

If `mongod` is started with the `--relaxPermChecks` option and the key file is owned by `root`, then `mongod` can read the file based on the group bit set accordingly. The effective key file permissions in this case are:

- **440** - both the owner and the group can only read the file, or
- **640** - only the owner can read and the change the file, the group can only read the file.

See also:

**MongoDB Documentation: Configure Encryption** <https://docs.mongodb.com/manual/tutorial/configure-encryption/#local-key-management>

**Percona Blog: WiredTiger Encryption at Rest with Percona Server for MongoDB** <https://www.percona.com/blog/2018/11/01/wiredtiger-encryption-at-rest-percona-server-for-mongodb/>

All these options can be specified in the configuration file:

```
security:
  enableEncryption: <boolean>
  encryptionCipherMode: <string>
  encryptionKeyFile: <string>
  relaxPermChecks: <boolean>
```

See also:

**MongoDB Documentation: How to set options in a configuration file** <https://docs.mongodb.com/manual/reference/configuration-options/index.html#configuration-file>

## 10.3 Migrating from Key File Encryption to HashiCorp Vault Encryption

The steps below describe how to migrate from the key file encryption to using HashiCorp Vault.

**Note:** This is a simple guideline and it should be used for testing purposes only. We recommend to use Percona Consulting Services to assist you with migration in production environment.

### Assumptions

We assume that you have installed and configured the vault server and enabled the KV Secrets Engine as the secrets storage for it.

1. Stop mongod.

```
$ sudo systemctl stop mongod
```

2. Insert the key from keyfile into the HashiCorp Vault server to the desired secret path.

```
# Retrieve the key value from the keyfile
$ sudo cat /data/key/mongodb.key
d0JTFcePmvR0yLXwCbAH8fmiP/ZRm0nYbeJDMGaI7Zw=
# Insert the key into vault
$ vault kv put secret/dc/psmongodb1 value=d0JTFcePmvR0yLXwCbAH8fmiP/
↪ ZRm0nYbeJDMGaI7Zw=
```

---

**Note:** Vault KV Secrets Engine uses different read and write secrets paths. To insert data to vault, specify the secret path without the data/ prefix.

---

3. Edit the configuration file to provision the HashiCorp Vault configuration options instead of the key file encryption options.

```
security:
  enableEncryption: true
vault:
  serverName: 10.0.2.15
  port: 8200
  secret: secret/data/dc/psmongodb1
  tokenFile: /etc/mongodb/token
  serverCAFile: /etc/mongodb/vault.crt
```

4. Start the mongod service

```
$ sudo systemctl start mongod
```

## 10.4 Using the Key Management Interoperability Protocol (KMIP)

This feature is **technical preview** quality.

*Percona Server for MongoDB* adds support for the secure transfer of keys using the [OASIS Key Management Interoperability Protocol \(KMIP\)](#). The KMIP implementation was tested with the [PyKMIP server](#) and the [HashiCorp Vault Enterprise KMIP Secrets Engine](#).

KMIP enables the communication between a key management system and the database server. KMIP provides the following benefits:

- Streamlines encryption key management
- Eliminates redundant key management processes

Starting with version 5.0.9-8, you can specify multiple KMIP servers for failover. On startup, *Percona Server for MongoDB* connects to the servers in the order listed and selects the one with which the connection is successful.

---

### KMIP parameters



Option	Type	Description
<code>--kmipServerName</code>	string	The hostname or IP address of the KMIP server. As of version 5.0.9-8, multiple KMIP servers are supported as the comma-separated list, e.g. <code>kmip1.example.com,kmip2.example.com</code>
<code>--kmipPort</code>	number	The port used to communicate with the KMIP server. When undefined, the default port 5696 will be used.
<code>--kmipServerCAFile</code>	string	The path to the TLS certificate file. CA file is used to validate secure client connection to the KMIP server.
<code>--kmipClientCertificateFile</code>	string	The path to the PEM file with the KMIP client private key and the certificate chain. The database server uses this PEM file to authenticate the KMIP server.
<code>--kmipKeyIdentifier</code>	string	Mandatory. The name of the KMIP key. If the key does not exist, the database server creates a key on the KMIP server with the specified identifier.
<code>--kmipRotateMasterKey</code>	boolean	Controls master keys rotation. When enabled, generates the new master key version and re-encrypts the keystore. Available as of version 5.0.8-7. Requires the unique <code>--kmipKeyIdentifier</code> for every mongod node.
<code>--kmipClientCertificatePassword</code>	string	The password for the KMIP client private key or certificate. Use this parameter only if the KMIP client private key or certificate is encrypted. Available starting with version 5.0.9-8.

## 10.5 Key rotation

Starting with release 5.0.8-7, the support for [master key rotation](#) is added. This enables users to comply with data security regulations when using KMIP.

**Note:** To make KMIP master key rotation, make sure that every mongod has a unique `--kmipKeyIdentifier` value.

## 10.6 Configuration

### Considerations

Make sure you have obtained the root certificate, and the keypair for the KMIP server and the mongod client. For testing purposes you can use the [OpenSSL](#) to issue self-signed certificates. For production use we recommend you use the valid certificates issued by the key management appliance.

To enable data-at-rest encryption in *Percona Server for MongoDB* using KMIP, edit the `/etc/mongod.conf` configuration file as follows:

```
security:
  enableEncryption: true
  kmip:
    serverName: <kmip_server_name>
    port: <kmip_port>
    clientCertificateFile: </path/client_certificate.pem>
    clientKeyFile: </path/client_key.pem>
    serverCAFile: </path/ca.pem>
    keyIdentifier: <key_name>
```

Alternatively, you can start *Percona Server for MongoDB* using the command line as follows:

```
$ mongod --enableEncryption \
--kmipServerName <kmip_servername> \
--kmipPort <kmip_port> \
--kmipServerCAFile <path_to_ca_file> \
--kmipClientCertificateFile <path_to_client_certificate> \
--kmipClientKeyFile <path_to_client_private_key> \
--kmipKeyIdentifier <kmip_identifier>
```

## Important Configuration Options

*Percona Server for MongoDB* supports the `encryptionCipherMode` option where you choose one of the following cipher modes:

- AES256-CBC
- AES256-GCM

By default, the AES256-CBC cipher mode is applied. The following example demonstrates how to apply the AES256-GCM cipher mode when starting the **mongod** service:

```
$ mongod ... --encryptionCipherMode AES256-GCM
```

See also:

**MongoDB Documentation: encryptionCipherMode Option** <https://docs.mongodb.com/manual/reference/program/mongod/#cmdoption-mongod-encryptionciphermode>

## 10.7 Encrypting Rollback Files

Starting from version 3.6, *Percona Server for MongoDB* also encrypts rollback files when data at rest encryption is enabled. To inspect the contents of these files, use **perconadecrypt**. This is a tool that you run from the command line as follows:

```
$ perconadecrypt --encryptionKeyFile FILE --inputPath FILE --outputPath FILE [--
↪ encryptionCipherMode MODE]
```

When decrypting, the cipher mode must match the cipher mode which was used for the encryption. By default, the `--encryptionCipherMode` option uses the AES256-CBC mode.

---

### Parameters of **perconadecrypt**

Option	Purpose
<code>-encryptionKeyFile</code>	The path to the encryption key file
<code>-encryptionCipher-Mode</code>	The cipher mode for decryption. The supported values are AES256-CBC or AES256-GCM
<code>-inputPath</code>	The path to the encrypted rollback file
<code>-outputPath</code>	The path to save the decrypted rollback file

---

## ADDITIONAL TEXT SEARCH ALGORITHM - *NGRAM*

The *ngram* text search algorithm is useful for searching text for a specific string of characters in a field of a collection. This feature can be used to find exact sub-string matches, which provides an alternative to parsing text from languages other than the list of European languages already supported by MongoDB Community's full text search engine. It may also turn out to be more convenient when working with the text where symbols like dash('-'), underscore('\_'), or slash("/") are not token delimiters.

Unlike MongoDB full text search engine, *ngram* search algorithm uses only the following token delimiter characters that do not count as word characters in human languages:

- Horizontal tab
- Vertical tab
- Line feed
- Carriage return
- Space

The *ngram* text search is slower than MongoDB full text search.

### 11.1 Usage

To use *ngram*, create a text index on a collection setting the `default_language` parameter to **ngram**:

```
mongo > db.collection.createIndex({name:"text"}, {default_language: "ngram"})
```

*ngram* search algorithm treats special characters like individual terms. Therefore, you don't have to enclose the search string in escaped double quotes (\") to query the text index. For example, to search for documents that contain the date 2021-02-12, specify the following:

```
mongo > db.collection.find({ $text: { $search: "2021-02-12" } })
```

However, both *ngram* and MongoDB full text search engine treat words with the hyphen-minus - sign in front of them as negated (e.g. "-coffee") and exclude such words from the search results.

**See also:**

**MongoDB documentation:**

- [Text search](#)
- [Text indexes](#)
- [\\$text operator](#)

**More information about ngram implementation:**

- <https://github.com/percona/percona-server-mongodb/blob/v5.0/src/mongo/db/fts/ngram-tokenizer.md>

## **Part IV**

## **How to**



## ENABLING AUTHENTICATION

By default, *Percona Server for MongoDB* does not restrict access to data and configuration.

Enabling authentication enforces users to identify themselves when accessing the database. This document describes how to enable built-in authentication mechanism. *Percona Server for MongoDB* also supports the number of external authentication mechanisms. To learn more, refer to [Authentication](#).

You can enable authentication either automatically or manually.

### 12.1 Automatic setup

To enable authentication and automatically set it up, run the `/usr/bin/percona-server-mongodb-enable-auth.sh` script as root or using `sudo`.

This script creates the `dba` user with the `root` role. The password is randomly generated and printed out in the output. Then the script restarts *Percona Server for MongoDB* with access control enabled. The `dba` user has full superuser privileges on the server. You can add other users with various roles depending on your needs.

For usage information, run the script with the `-h` option.

### 12.2 Manual setup

To enable access control manually:

1. Add the following lines to the configuration file:

```
security:
  authorization: enabled
```

2. Run the following command on the admin database:

```
> db.createUser({user: 'USER', pwd: 'PASSWORD', roles: ['root'] });
```

3. Restart the mongod service:

```
$ systemctl restart mongod
```

4. Connect to the database as the newly created user:

```
$ mongo --port 27017 -u 'USER' -p 'PASSWORD' --authenticationDatabase "admin"
```

See also:

***MongoDB* Documentation: Enable Access Control** <https://docs.mongodb.com/v5.0/tutorial/enable-authentication/>



## SETTING UP LDAP AUTHENTICATION WITH SASL

This document describes an example configuration suitable only to test out the external authentication functionality in a non-production environment. Use common sense to adapt these guidelines to your production environment.

To learn more about how the authentication works, see *LDAP authentication with SASL*.

### 13.1 Environment setup and configuration

The following components are required:

- `slapd`: OpenLDAP server.
- `libsasl2` version 2.1.25 or later.
- `saslauthd`: SASL Authentication Daemon (distinct from `libsasl2`).

The following steps will help you configure your environment:

- *Configuring `saslauthd`*
  - *OpenLDAP server*
  - *Microsoft Windows Active Directory*
- *Sanity check*
- *Configuring `libsasl2`*
- *Configuring `mongod` Server*
  - *Create a root user*
  - *Enable external authentication*
  - *Add an external user to Percona Server for MongoDB*

## Assumptions

Before we move on to the configuration steps, we assume the following:

1. You have the LDAP server up and running and have configured users on it. The LDAP server is accessible to the server with *Percona Server for MongoDB* installed. This document focuses on OpenLDAP server. If you use Microsoft Windows Active Directory, refer to the [Microsoft Windows Active Directory](#) section for `saslauthd` configuration.
2. You must place these two servers behind a firewall as the communications between them will be in plain text. This is because the SASL mechanism of PLAIN can only be used when authenticating and credentials will be sent in plain text.
3. You have sudo privilege to the server with the *Percona Server for MongoDB* installed.

### 13.1.1 Configuring saslauthd

1. Install the SASL packages. Depending on your OS, use the following command:

---

#### RedHat and CentOS

```
$ sudo yum install -y cyrus-sasl
```

---

**Note:** For *Percona Server for MongoDB* versions earlier than 4.0.26-21, 4.4.8-9, 4.2.16-17, also install the `cyrus-sasl-plain` package.

---

---

#### Debian and Ubuntu

```
$ sudo apt install -y sasl2-bin
```

---

2. Configure SASL to use `ldap` as the authentication mechanism.

---

**Note:** Back up the original configuration file before making changes.

---

---

#### RedHat and CentOS

Specify the `ldap` value for the `--MECH` option using the following command:

```
$ sudo sed -i -e s/^MECH=pam/MECH=ldap/g /etc/sysconfig/saslauthd
```

Alternatively, you can edit the `/etc/sysconfig/saslauthd` configuration file:

```
MECH=ldap
```

---

---

#### Debian and Ubuntu

Use the following commands to enable the `saslauthd` to auto-run on startup and to set the `ldap` value for the `--MECHANISMS` option:

```
$ sudo sed -i -e s/^MECH=pam/MECH=ldap/g /etc/sysconfig/saslauthd
sudo sed -i -e s/^
↪MECHANISMS="pam"/MECHANISMS="ldap"/g /etc/default/saslauthd
$ sudo sed -i -e s/^START=no/START=yes/g /etc/default/saslauthd
```

Alternatively, you can edit the `/etc/default/sysconfig/saslauthd` configuration file:

```
START=yes
MECHANISMS="ldap"
```

3. Create the `/etc/saslauthd.conf` configuration file and specify the settings that `saslauthd` requires to connect to a local LDAP service:

### OpenLDAP server

The following is the example configuration file. Note that the server address **MUST** match the OpenLDAP installation:

```
ldap_servers: ldap://localhost
ldap_mech: PLAIN
ldap_search_base: dc=example,dc=com
ldap_filter: (cn=%u)
ldap_bind_dn: cn=admin,dc=example,dc=com
ldap_password: secret
```

Note the LDAP password (`ldap_password`) and bind domain name (`ldap_bind_dn`). This allows the `saslauthd` service to connect to the LDAP service as admin. In production, this would not be the case; users should not store administrative passwords in unencrypted files.

### Microsoft Windows Active Directory

In order for LDAP operations to be performed against a Windows Active Directory server, a user record must be created to perform the lookups.

The following example shows configuration parameters for `saslauthd` to communicate with an Active Directory server:

```
ldap_servers: ldap://localhost
ldap_mech: PLAIN
ldap_search_base: CN=Users,DC=example,DC=com
ldap_filter: (sAMAccountName=%u)
ldap_bind_dn: CN=ldapmgr,CN=Users,DC=<AD Domain>,DC=<AD TLD>
ldap_password: ld@pmgr_Pa55word
```

In order to determine and test the correct search base and filter for your Active Directory installation, the Microsoft [LDP GUI Tool](#) can be used to bind and search the LDAP-compatible directory.

4. Start the `saslauthd` process and set it to run at restart:

```
$ sudo systemctl start saslauthd
$ sudo systemctl enable saslauthd
```

5. Give write permissions to the `/run/saslauthd` folder for the `mongod`. Either change permissions to the `/run/saslauthd` folder:

```
$ sudo chmod 755 /run/saslauthd
```

Or add the mongod user to the sasl group:

```
$ sudo usermod -a -G sasl mongod
```

### 13.1.2 Sanity check

Verify that the saslauthd service can authenticate against the users created in the LDAP service:

```
$ testsaslauthd -u christian -p secret -f /var/run/saslauthd/mux
```

This should return `0:OK "Success"`. If it doesn't, then either the user name and password are not in the LDAP service, or saslauthd is not configured properly.

### 13.1.3 Configuring libsasl2

The mongod also uses the SASL library for communications. To configure the SASL library, create a configuration file.

The configuration file **must** be named `mongodb.conf` and placed in a directory where `libsasl2` can find and read it. `libsasl2` is hard-coded to look in certain directories at build time. This location may be different depending on the installation method.

In the configuration file, specify the following:

```
pwcheck_method: saslauthd
saslauthd_path: /var/run/saslauthd/mux
log_level: 5
mech_list: plain
```

The first two entries (`pwcheck_method` and `saslauthd_path`) are required for `mongod` to successfully use the `saslauthd` service. The `log_level` is optional but may help determine configuration errors.

**See also:**

[SASL documentation](#):

### 13.1.4 Configuring mongod Server

The configuration consists of the following steps:

- Creating a user with the **root** privileges. This user is required to log in to *Percona Server for MongoDB* after the external authentication is enabled.
- Editing the configuration file to enable the external authentication

## Create a root user

Create a user with the **root** privileges in the **admin** database. If you have already created this user, skip this step. Otherwise, run the following command to create the admin user:

```
> use admin
switched to db admin
> db.createUser({"user": "admin", "pwd": "$3cr3tP4ssw0rd", "roles": ["root"]})
Successfully added user: { "user" : "admin", "roles" : [ "root" ] }
```

## Enable external authentication

Edit the `etc/mongod.conf` configuration file to enable the external authentication:

```
security:
  authorization: enabled

setParameter:
  authenticationMechanisms: PLAIN,SCRAM-SHA-1
```

Restart the mongod service:

```
$ sudo systemctl restart mongod
```

## Add an external user to Percona Server for MongoDB

User authentication is done by mapping a user object on the LDAP server against a user created in the `$external` database. Thus, at this step, you create the user in the `$external` database and they inherit the roles and privileges. Note that the username must exactly match the name of the user object on the LDAP server.

Connect to *Percona Server for MongoDB* and authenticate as *the root user*.

```
$ mongo --host localhost --port 27017 -u admin -p '$3cr3tP4ssw0rd' --
↪authenticationDatabase 'admin'
```

Use the following command to add an external user to *Percona Server for MongoDB*:

```
> db.getSiblingDB("$external").createUser( {user : "christian", roles: [ {role: "read", ↪
↪db: "test"} ]} );
```

## 13.2 Authenticate as an external user in Percona Server for MongoDB

When running the mongo client, a user can authenticate against a given database using the following command:

```
> db.getSiblingDB("$external").auth({ mechanism:"PLAIN", user:"christian", pwd:"secret", ↪
↪digestPassword:false})
```

Alternatively, a user can authenticate while connecting to *Percona Server for MongoDB*:

```
$ mongo --host localhost --port 27017 --authenticationMechanism PLAIN --
↪authenticationDatabase \$external -u christian -p
```

---

**Based on the material from Percona Database Performance Blog**

This section is based on the blog post [Percona Server for MongoDB Authentication Using Active Directory](#) by *Doug Duncan*:

---

## SETTING UP LDAP AUTHENTICATION AND AUTHORIZATION USING NATIVELDAP

This document describes an example configuration of LDAP authentication and authorization using direct binding to an LDAP server (Native LDAP). We recommend testing this setup in a non-production environment first, before applying it in production.

### 14.1 Assumptions

1. The setup of an LDAP server is out of scope of this document. We assume that you are familiar with the LDAP server schema.
2. You have the LDAP server up and running and it is accessible to the servers with Percona Server for MongoDB installed.
3. This document primarily focuses on OpenLDAP used as the LDAP server and the examples are given based on the OpenLDAP format. If you are using Active Directory, refer to the [Active Directory configuration](#) section.
4. You have the sudo privilege to the server with the Percona Server for MongoDB installed.

### 14.2 Prerequisites

- In this setup we use anonymous binds to the LDAP server. If your LDAP server disallows anonymous binds, create the user that *Percona Server for MongoDB* will use to connect to and query the LDAP server. Define this user's credentials for the `security.ldap.bind.queryUser` and `security.ldap.bind.queryPassword` parameters in the `mongod.conf` configuration file.
- In this setup, we use the following OpenLDAP groups:

```
dn: cn=testusers,dc=percona,dc=com
objectClass: groupOfNames
cn: testusers
member: cn=alice,dc=percona,dc=com

dn: cn=otherusers,dc=percona,dc=com
objectClass: groupOfNames
cn: otherusers
member: cn=bob,dc=percona,dc=com
```

## 14.3 Setup procedure

### 14.3.1 Configure TLS/SSL connection for *Percona Server for MongoDB*

By default, *Percona Server for MongoDB* establishes the TLS connection when binding to the LDAP server and thus, it requires access to the LDAP CA certificates. To make *Percona Server for MongoDB* aware of the certificates, do the following:

1. Place the certificate in the `certs` directory. The path to the `certs` directory is:
  - On Debian / Ubuntu: `/etc/ssl/certs/`
  - On RHEL / CentOS: `/etc/openldap/certs/`
2. Specify the path to the certificates in the `ldap.conf` file:

---

#### Debian / Ubuntu

```
tee -a /etc/openldap/ldap.conf <<EOF
TLS_CACERT /etc/ssl/certs/my_CA.crt
EOF
```

---

---

#### RHEL / CentOS

```
tee -a /etc/openldap/ldap.conf <<EOF
TLS_CACERT /etc/openldap/certs/my_CA.crt
EOF
```

---

### 14.3.2 Create roles for LDAP groups in *Percona Server for MongoDB*

*Percona Server for MongoDB* authorizes users based on LDAP group membership. For every group, you must create the role in the `admin` database with the name that exactly matches the DN of the LDAP group.

*Percona Server for MongoDB* maps the user's LDAP group to the roles and determines what role is assigned to the user. *Percona Server for MongoDB* then grants privileges defined by this role.

To create the roles, use the following command:

```
var admin = db.getSiblingDB("admin")
db.createRole(
  {
    role: "cn=testusers,dc=percona,dc=com",
    privileges: [],
    roles: [ "readWrite" ]
  }
)

db.createRole(
  {
    role: "cn=otherusers,dc=percona,dc=com",
    privileges: [],
```

(continues on next page)



(continued from previous page)

```

    roles: [ "read" ]
  }
)

```

### 14.3.3 Percona Server for MongoDB configuration

#### Access without username transformation

This section assumes that users connect to *Percona Server for MongoDB* by providing their LDAP DN as the username.

1. Edit the *Percona Server for MongoDB* configuration file (by default, `/etc/mongod.conf`) and specify the following configuration:

```

security:
  authorization: "enabled"
  ldap:
    servers: "ldap.example.com"
    transportSecurity: tls
    authz:
      queryTemplate: "dc=percona,dc=com??sub?(&(objectClass=groupOfNames)(member=
↪{PROVIDED_USER}))"

setParameter:
  authenticationMechanisms: "PLAIN"

```

The `{PROVIDED_USER}` variable substitutes the provided username before authentication or username transformation takes place.

Replace `ldap.example.com` with the hostname of your LDAP server. In the LDAP query template, replace the domain controllers `percona` and `com` with those relevant to your organization.

2. Restart the `mongod` service:

```
$ sudo systemctl restart mongod
```

3. Test the access to *Percona Server for MongoDB*:

```

mongo -u "cn=alice,dc=percona,dc=com" -p "secretpwd" --authenticationDatabase '
↪$external' --authenticationMechanism 'PLAIN'

```

#### Access with username transformation

If users connect to *Percona Server for MongoDB* with usernames that are not LDAP DN, you need to transform these usernames to be accepted by the LDAP server.

Using the `--ldapUserToDNMapping` configuration parameter allows you to do this. You specify the match pattern as a regexp to capture a username. Next, specify how to transform it - either to use a substitution value or to query the LDAP server for a username.

If you don't know what the substitution or LDAP query string should be, please consult with the LDAP administrators to figure this out.

Note that you can use only the query or the substitution stage, the combination of two is not allowed.

## Substitution

1. Edit the *Percona Server for MongoDB* configuration file (by default, `/etc/mongod.conf`) and specify the `userToDNMapping` parameter:

```
security:
  authorization: "enabled"
  ldap:
    servers: "ldap.example.com"
    transportSecurity: tls
    authz:
      queryTemplate: "dc=percona,dc=com??sub?(&
↪(objectClass=groupOfNames)(member={USER}))"
      userToDNMapping: >-
        [
          {
            match: "([^\@]+\@percona\\.com",
            substitution: "CN={0},DC=percona,DC=com"
          }
        ]

setParameter:
  authenticationMechanisms: "PLAIN"
```

The `{USER}` variable substitutes the username transformed during the `userToDNMapping` stage.

Modify the given example configuration to match your deployment.

2. Restart the `mongod` service:

```
$ sudo systemctl restart mongod
```

3. Test the access to *Percona Server for MongoDB*:

```
mongo -u "alice@percona.com" -p "secretpwd" --authenticationDatabase '
↪$external' --authenticationMechanism 'PLAIN'
```

---

## LDAP query

1. Edit the *Percona Server for MongoDB* configuration file (by default, `/etc/mongod.conf`) and specify `userToDNMapping` parameter:

```
security:
  authorization: "enabled"
  ldap:
    servers: "ldap.example.com"
    transportSecurity: tls
    authz:
      queryTemplate: "dc=percona,dc=com??sub?(&
↪(objectClass=groupOfNames)(member={USER}))"
      userToDNMapping: >-
        [
          {
```

(continues on next page)

(continued from previous page)

```

        match: "([^\@]+)@percona\\.com",
        ldapQuery: "dc=percona,dc=com??sub?(&
↪(objectClass=organizationalPerson)(cn={0}))"
    }
]

setParameter:
  authenticationMechanisms: "PLAIN"

```

The {USER} variable substitutes the username transformed during the userToDNMapping stage.

Modify the given example configuration to match your deployment. For example, replace `ldap.example.com` with the hostname of your LDAP server. Replace the domain controllers (DC) `percona` and `com` with those relevant to your organization. Depending on your LDAP schema, further modifications of the LDAP query may be required.

- Restart the mongod service:

```
$ sudo systemctl restart mongod
```

- Test the access to *Percona Server for MongoDB*:

```
mongo -u "alice" -p "secretpwd" --authenticationDatabase '$external' --
↪authenticationMechanism 'PLAIN'
```

### 14.3.4 Active Directory configuration

Microsoft Active Directory uses a different schema for user and group definition. To illustrate *Percona Server for MongoDB* configuration, we will use the following AD (Active Directory) users:

```

dn:CN=alice,CN=Users,DC=testusers,DC=percona,DC=com
userPrincipalName: alice@testusers.percona.com
memberOf: CN=testusers,CN=Users,DC=percona,DC=com

dn:CN=bob,CN=Users,DC=otherusers,DC=percona,DC=com
userPrincipalName: bob@otherusers.percona.com
memberOf: CN=otherusers,CN=Users,DC=percona,DC=com

```

The following are respective AD groups:

```

dn:CN=testusers,CN=Users,DC=percona,DC=com
member:CN=alice,CN=Users,DC=testusers,DC=example,DC=com

dn:CN=otherusers,CN=Users,DC=percona,DC=com
member:CN=bob,CN=Users,DC=otherusers,DC=example,DC=com

```

Use one of the given *Percona Server for MongoDB* configurations for user authentication and authorization in Active Directory:

---

#### No username transformation

- Edit the mongod configuration file:

```
ldap:
  servers: "ldap.example.com"
  authz:
    queryTemplate: "DC=percona,DC=com??sub?(&(objectClass=group)(member:1.
↪2.840.113556.1.4.1941:={PROVIDED_USER}))"

  setParameter:
    authenticationMechanisms: "PLAIN"
```

- Restart the mongod service:

```
$ sudo systemctl restart mongod
```

- Test the access to *Percona Server for MongoDB*:

```
mongo -u "CN=alice,CN=Users,DC=testusers,DC=percona,DC=com" -p "secretpwd" ↪
↪--authenticationDatabase '$external' --authenticationMechanism 'PLAIN'
```

---

### Username substitution

- Edit the mongod configuration file:

```
ldap:
  servers: "ldap.example.com"
  authz:
    queryTemplate: "DC=percona,DC=com??sub?(&(objectClass=group)(member:1.
↪2.840.113556.1.4.1941:={USER}))"
    userToDNMapping: >-
      [
        {
          match: "([^\@]+)@([^\.\.]+)\.percona\.com",
          substitution: "CN={0},CN=Users,DC={1},DC=percona,DC=com"
        }
      ]

  setParameter:
    authenticationMechanisms: "PLAIN"
```

- Restart the mongod service:

```
$ sudo systemctl restart mongod
```

- Test the access to *Percona Server for MongoDB*:

```
mongo -u "alice@percona.com" -p "secretpwd" --authenticationDatabase '
↪$external' --authenticationMechanism 'PLAIN'
```

---

### LDAP query

- Edit the mongod configuration file:

```
ldap:
  servers: "ldap.example.com"
  authz:
    queryTemplate: "DC=percona,DC=com??sub?(&(objectClass=group)(member:1.
↪2.840.113556.1.4.1941:={USER}))"
    userToDNMapping: >-
      [
        {
          match: "(.+)",
          ldapQuery: "dc=example,dc=com??sub?(&
↪(objectClass=organizationalPerson)(userPrincipalName={0}))"
        }
      ]

  setParameter:
    authenticationMechanisms: "PLAIN"
```

- Restart the mongod service:

```
$ sudo systemctl restart mongod
```

- Test the access to *Percona Server for MongoDB*:

```
mongo -u "alice" -p "secretpwd" --authenticationDatabase '$external' --
↪authenticationMechanism 'PLAIN'
```

---

Modify one of this example configuration to match your deployment.

---

### Based on the material from Percona Database Performance Blog

This document is based on the following blog posts:

- [Percona Server for MongoDB LDAP Enhancements: User-to-DN Mapping](#) by Igor Solodovnikov
  - [Authenticate Percona Server for MongoDB Users via Native LDAP](#) by Ivan Groenewold
-



## SET UP X.509 AUTHENTICATION AND LDAP AUTHORIZATION

*x.509 certificate authentication* is one of the supported authentication mechanisms in *Percona Server for MongoDB*. It is compatible with *LDAP authorization* to enable you to control user access and operations in your database environment.

This document provides the steps on how to configure and use x.509 certificates for authentication in *Percona Server for MongoDB* and authorize users in the LDAP server.

### 15.1 Considerations

1. For testing purposes, in this tutorial we use [OpenSSL](#) to issue self-signed certificates. For production use, we recommend using certificates issued and signed by the CA in *Percona Server for MongoDB*. Client certificates must meet the [client certificate requirements](#).
2. The setup of the LDAP server and the configuration of the LDAP schema is out of scope of this document. We assume that you have the LDAP server up and running and accessible to *Percona Server for MongoDB*.

### 15.2 Setup procedure

#### 15.2.1 Issue certificates

1. Create a directory to store the certificates. For example, `/var/lib/mongocerts`.

```
$ sudo mkdir -p /var/lib/mongocerts
```

2. Grant access to the `mongod` user to this directory:

```
$ sudo chown mongod:mongod /var/lib/mongocerts
```

#### Generate the root Certificate Authority certificate

The root Certificate Authority certificate will be used to sign the SSL certificates.

Run the following command and in the `-subj` flag, provide the details about your organization:

- C - Country Name (2 letter code);
- ST - State or Province Name (full name);
- L - Locality Name (city);
- O - Organization Name (company);

- CN - Common Name (your name or your server's hostname) .

```
$ cd /var/lib/mongocerts
$ sudo openssl req -nodes -x509 -newkey rsa:4096 -keyout ca.key -out ca.crt -subj "/C=US/
↳ST=California/L=SanFrancisco/O=Percona/OU=root/CN=localhost"
```

## Generate server certificate

1. Create the server certificate request and key. In the `-subj` flag, provide the details about your organization:

- C - Country Name (2 letter code);
- ST - State or Province Name (full name);
- L - Locality Name (city);
- O - Organization Name (company);
- CN - Common Name (your name or your server's hostname) .

```
$ sudo openssl req -nodes -newkey rsa:4096 -keyout server.key -out server.csr -subj
↳"/C=US/ST=California/L=SanFrancisco/O=Percona/OU=server/CN=localhost"
```

2. Sign the server certificate request with the root CA certificate:

```
$ sudo openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -set_serial 01 -
↳out server.crt
```

3. Combine the server certificate and key to create a certificate key file. Run this command as the `root` user:

```
$ cat server.key server.crt > server.pem
```

## Generate client certificates

1. Generate client certificate request and key. In the `-subj` flag, specify the information about clients in the DN format.

```
$ openssl req -nodes -newkey rsa:4096 -keyout client.key -out client.csr -subj "/"
↳DC=com/DC=percona/CN=John Doe"
```

2. Sign the client certificate request with the root CA certificate.

```
$ openssl x509 -req -in client.csr -CA ca.crt -CAkey ca.key -set_serial 02 -out
↳client.crt
```

3. Combine the client certificate and key to create a certificate key file.

```
$ cat client.key client.crt > client.pem
```



## 15.2.2 Set up the LDAP server

The setup of the LDAP server is out of scope of this document. Please work with your LDAP administrators to set up the LDAP server and configure the LDAP schema.

## 15.2.3 Configure mongod server

The configuration consists of the following steps:

- Creating a role that matches the user group on the LDAP server
- Editing the configuration file to enable the x.509 authentication

**Note:** When you use x.509 authentication with LDAP authorization, you don't need to create users in the `$external` database. User management is done on the LDAP server so when a client connects to the database, they are authenticated and authorized through the LDAP server.

### Create roles

At this step, create the roles in the `admin` database with the names that exactly match the names of the user groups on the LDAP server. These roles are used for user *LDAP authorization* in *Percona Server for MongoDB*.

In our example, we create the role `cn=otherusers,dc=percona,dc=com` that has the corresponding LDAP group.

```
var admin = db.getSiblingDB("admin")
db.createRole(
  {
    role: "cn=otherusers,dc=percona,dc=com",
    privileges: [],
    roles: [
      "userAdminAnyDatabase",
      "clusterMonitor",
      "clusterManager",
      "clusterAdmin"
    ]
  }
)
```

Output:

```
{
  "role" : "cn=otherusers,dc=percona,dc=com",
  "privileges" : [ ],
  "roles" : [
    "userAdminAnyDatabase",
    "clusterMonitor",
    "clusterManager",
    "clusterAdmin"
  ]
}
```

## Enable x.509 authentication

1. Stop the mongod service

```
$ sudo systemctl stop mongod
```

2. Edit the /etc/mongod.conf configuration file.

```
net:
  port: 27017
  bindIp: 127.0.0.1
  tls:
    mode: requireTLS
    certificateKeyFile: /var/lib/mongocerts/server.pem
    CAFile: /var/lib/mongocerts/ca.crt

security:
  authorization: enabled
  ldap:
    servers: "ldap.example.com"
    transportSecurity: none
    authz:
      queryTemplate: "dc=percona,dc=com??sub?(&(objectClass=groupOfNames) (member=
↵{USER}))"
```

```
setParameter:
  authenticationMechanisms: PLAIN,MONGODB-X509
```

Replace `ldap.example.com` with the hostname of your LDAP server. In the LDAP query template, replace the domain controllers `percona` and `com` with those relevant to your organization.

3. Start the mongod service

```
$ sudo systemctl start mongod
```

### 15.2.4 Authenticate with the x.509 certificate

To test the authentication, connect to *Percona Server for MongoDB* using the following command:

```
$ mongo --host localhost --tls --tlsCAFile /var/lib/mongocerts/ca.crt --
↵tlsCertificateKeyFile <path_to_client_certificate>/client.pem --
↵authenticationMechanism MONGODB-X509 --authenticationDatabase='$external'
```

The result should look like the following:

```
> db.runCommand({connectionStatus : 1})
{
  "authInfo" : {
    "authenticatedUsers" : [
      {
        "user" : "CN=John Doe,DC=percona,DC=com",
        "db" : "$external"
      }
    ]
  }
}
```

(continues on next page)

(continued from previous page)

```
    ],
    "authenticatedUserRoles" : [
      {
        "role" : "cn=otherreaders,dc=percona,dc=com",
        "db" : "admin"
      },
      {
        "role" : "clusterAdmin",
        "db" : "admin"
      },
      {
        "role" : "userAdminAnyDatabase",
        "db" : "admin"
      },
      {
        "role" : "clusterManager",
        "db" : "admin"
      },
      {
        "role" : "clusterMonitor",
        "db" : "admin"
      }
    ]
  },
  "ok" : 1
}
```



## SETTING UP KERBEROS AUTHENTICATION

This document provides configuration steps for setting up *Kerberos Authentication* in *Percona Server for MongoDB*.

### 16.1 Assumptions

The setup of the Kerberos server itself is out of scope of this document. Please refer to the [Kerberos documentation](#) for the installation and configuration steps relevant to your operation system.

We assume that you have successfully completed the following steps:

- Installed and configured the Kerberos server
- Added necessary [realms](#)
- Added service, admin and user [principals](#)
- Configured the A and PTR DNS records for every host running *mongod* instance to resolve the hostnames onto Kerberos realm.

**See also:**

MongoDB Documentation: [Kerberos Authentication](#)

### 16.2 Add user principals to *Percona Server for MongoDB*

To get authenticated, users must exist both in the Kerberos and *Percona Server for MongoDB* servers with exactly matching names.

After you defined the user principals in the Kerberos server, add them to the `$external` database in *Percona Server for MongoDB* and assigned required roles:

```
use $external
db.createUser({user: "demo@PERCONATEST.COM",roles: [{role: "read", db: "admin"}]})
```

Replace `demo@PERCONATEST.COM` with your username and Kerberos realm.

## 16.3 Configure Kerberos keytab files

A keytab file stores the authentication keys for a service principal representing a `mongod` instance to access the Kerberos admin server.

After you have added the service principal to the Kerberos admin server, the entry for this principal is added to the `/etc/krb5.keytab` keytab file.

The `mongod` server must have access to the keytab file to authenticate. To keep the keytab file secure, restrict the access to it only for the user running the `mongod` process.

1. Stop the `mongod` service

```
$ sudo systemctl stop mongod
```

2. [Generate the keytab file](#) or get a copy of it if you generated the keytab file on another host. Save the keyfile under a separate path (e.g. `/etc/mongodb.keytab`)

```
$ cp /etc/krb5.keytab /etc/mongodb.keytab
```

3. Change the ownership to the keytab file

```
$ sudo chown mongod:mongod /etc/mongodb.keytab
```

4. Set the `KRB5_KTNAME` variable in the environment file for the `mongod` process.

---

### On Debian and Ubuntu

Edit the environment file at the path `/etc/default/mongodb` and specify the `KRB5_KTNAME` variable:

```
KRB5_KTNAME=/etc/mongodb.keytab
```

If you have a different path to the keytab file, specify it accordingly.

---

### On RHEL and derivatives

Edit the environment file at the path `/etc/sysconfig/mongod` and specify the `KRB5_KTNAME` variable:

```
KRB5_KTNAME=/etc/mongodb.keytab
```

If you have a different path to the keytab file, specify it accordingly.

5. Restart the `mongod` service

```
$ sudo systemctl start mongod
```

## 16.4 Percona Server for MongoDB configuration

Enable external authentication in *Percona Server for MongoDB* configuration. Edit the `etc/mongod.conf` configuration file and specify the following configuration:

```
security:
  authorization: "enabled"

setParameter:
  authenticationMechanisms: GSSAPI
```

Restart the mongod service to apply the configuration:

```
$ sudo systemctl start mongod
```

## 16.5 Test the access to Percona Server for MongoDB

1. Obtain the Kerberos ticket for the user using the `kinit` command and specify the user password:

```
$ kinit demo
Password for demo@PERCONATEST.COM:
```

2. Check the user ticket:

```
$ klist -l
Principal name          Cache name
-----
demo@PERCONATEST.COM    FILE:/tmp/<ticket>
```

3. Connect to *Percona Server for MongoDB*:

```
$ mongo --host <hostname> --authenticationMechanism=GSSAPI --authenticationDatabase=
→ '$external' --username demo@PERCONATEST.COM
```

The result should look like the following:

```
> db.runCommand({connectionStatus : 1})
{
  "authInfo" : {
    "authenticatedUsers" : [
      {
        "user" : "demo@PERCONATEST.COM",
        "db" : "$external"
      }
    ],
    "authenticatedUserRoles" : [
      {
        "role" : "read",
        "db" : "admin"
      }
    ]
  },
}
```

(continues on next page)

(continued from previous page)

```
}  "ok" : 1
```



## PERCONA SERVER FOR MONGODB PARAMETER TUNING GUIDE

*Percona Server for MongoDB* includes several parameters that can be changed in one of the following ways:

- The `setParameter` admonitions in the configuration file for persistent changes in production:

```
setParameter:
  cursorTimeoutMillis: <int>
  failIndexKeyTooLong: <boolean>
  internalQueryPlannerEnableIndexIntersection: <boolean>
  ttlMonitorEnabled: <boolean>
  ttlMonitorSleepSecs: <int>
```

- The `--setParameter` option arguments when running the `mongod` process for development or testing purposes:

```
$ mongod \
--setParameter cursorTimeoutMillis=<int> \
--setParameter failIndexKeyTooLong=<boolean> \
--setParameter internalQueryPlannerEnableIndexIntersection=<boolean> \
--setParameter ttlMonitorEnabled=<boolean> \
--setParameter ttlMonitorSleepSecs=<int>
```

- The `setParameter` command on the `admin` database to make changes at runtime:

```
> db = db.getSiblingDB('admin')
> db.runCommand( { setParameter: 1, cursorTimeoutMillis: <int> } )
> db.runCommand( { setParameter: 1, failIndexKeyTooLong: <boolean> } )
> db.runCommand( { setParameter: 1, internalQueryPlannerEnableIndexIntersection:
↪<boolean> } )
> db.runCommand( { setParameter: 1, ttlMonitorEnabled: <int> } )
> db.runCommand( { setParameter: 1, ttlMonitorSleepSecs: <int> } )
```

## 17.1 Parameters

### 17.1.1 cursorTimeoutMillis

**Value Type** *integer*

**Default** 600000 (ten minutes)

Sets the duration of time after which idle query cursors are removed from memory.

### 17.1.2 failIndexKeyTooLong

**Value Type** *boolean*

**Default** `true`

Versions of MongoDB prior to 2.6 would insert and update documents even if an index key was too long. The documents would not be included in the index. Newer versions of MongoDB ignore documents with long index key. By setting this value to `false`, the old behavior is enabled.

### 17.1.3 internalQueryPlannerEnableIndexIntersection

**Value Type** *boolean*

**Default** `true`

Due to changes introduced in MongoDB 2.6.4, some queries that reference multiple indexed fields, where one field matches no documents, may choose a non-optimal single-index plan. Setting this value to `false` will enable the old behavior and select the index intersection plan.

### 17.1.4 ttlMonitorEnabled

**Value Type** *boolean*

**Default** `true`

If this option is set to `false`, the worker thread that monitors TTL Indexes and removes old documents will be disabled.

### 17.1.5 ttlMonitorSleepSecs

**Value Type** *integer*

**Default** `60` (one minute)

Defines the number of seconds to wait between checking TTL Indexes for old documents and removing them.

## UPGRADING *PERCONA SERVER FOR MONGODB*

An in-place upgrade is done by keeping the existing data in the server. It involves changing out the MongoDB binaries. Generally speaking, the upgrade steps include:

1. Stopping the mongod service
2. Removing the old binaries
3. Installing the new server version binaries
4. Restarting the mongod service with the same dbpath data directory.

An in-place upgrade is suitable for most environments except the ones that use ephemeral storage and/or host addresses.

This document provides upgrade instructions for the following use cases:

- *Upgrading from MongoDB 5.0 Community Edition;*
- *Minor upgrade of Percona Server for MongoDB*

### 18.1 Upgrading from MongoDB 5.0 Community Edition

---

**Note:** MongoDB creates a user that belongs to two groups, which is a potential security risk. This is fixed in *Percona Server for MongoDB*: the user is included only in the mongod group. To avoid problems with current MongoDB setups, existing user group membership is not changed when you migrate to *Percona Server for MongoDB*. Instead, a new mongod user is created during installation, and it belongs to the mongod group.

---

This section describes an in-place upgrade of a mongod instance. If you are using data at rest encryption, refer to the *Upgrading to Percona Server for MongoDB with data at rest encryption enabled* section.

#### 18.1.1 Prerequisites

Before you start the upgrade, update the MongoDB configuration file (/etc/mongod.conf) to contain the following settings.

```
processManagement:  
  fork: true  
  pidFilePath: /var/run/mongod.pid
```

**Troubleshooting tip:** The pidFilePath setting in mongod.conf must match the PIDFile option in the systemd mongod service unit. Otherwise, the service will kill the mongod process after a timeout.

**Warning:** Before starting the upgrade, we recommend to perform a full backup of your data.

### Upgrading on Debian or Ubuntu

1. Stop the mongod service:

```
$ sudo systemctl stop mongod
```

2. Check for installed packages:

```
$ sudo dpkg -l | grep mongod
```

#### Output

```
ii  mongodb-org                    5.0.2                amd64                └─
    ↳ MongoDB open source document-oriented database system (metapackage)
ii  mongodb-org-database           5.0.2                amd64                └─
    ↳ MongoDB open source document-oriented database system (metapackage)
ii  mongodb-org-database-tools-extra 5.0.2                amd64                └─
    ↳ Extra MongoDB database tools
ii  mongodb-org-mongos             5.0.2                amd64                └─
    ↳ MongoDB sharded cluster query router
ii  mongodb-org-server             5.0.2                amd64                └─
    ↳ MongoDB database server
ii  mongodb-org-shell              5.0.2                amd64                └─
    ↳ MongoDB shell client
ii  mongodb-org-tools              5.0.2                amd64                └─
    ↳ MongoDB tools
```

3. Remove the installed packages:

```
$ sudo apt remove \
mongodb-org \
mongodb-org-mongos \
mongodb-org-server \
mongodb-org-shell \
mongodb-org-tools
```

4. Remove log files:

```
$ sudo rm -r /var/log/mongodb
```

5. Install *Percona Server for MongoDB* using *apt*.

6. Verify that the configuration file includes the correct options. For example, *Percona Server for MongoDB* stores data files in `/var/lib/mongodb` by default. If you used another `dbPath` data directory, edit the configuration file accordingly

7. Start the mongod service:

```
$ sudo systemctl mongod start
```

### Upgrading on Red Hat Enterprise Linux or CentOS

1. Stop the mongod service:

```
$ sudo systemctl stop mongod
```

2. Check for installed packages:

```
$ sudo rpm -qa | grep mongo
```

#### Output

```
mongodb-org-shell-5.0.2-1.el8.x86_64
mongodb-org-database-5.0.0-1.el8.x86_64
mongodb-org-5.0.0-1.el8.x86_64
mongodb-database-tools-100.4.1-1.x86_64
mongodb-org-server-5.0.2-1.el8.x86_64
mongodb-org-mongos-5.0.2-1.el8.x86_64
mongodb-org-tools-5.0.0-1.el8.x86_64
```

3. Remove the installed packages:

```
$ sudo yum remove \
mongodb-org-tools-5.0.0-1.el8.x86_64 \
mongodb-org-shell-5.0.2-1.el8.x86_64 \
mongodb-org-database-5.0.0-1.el8.x86_64 \
mongodb-org-5.0.0-1.el8.x86_64 \
mongodb-database-tools-100.4.1-1.x86_64 \
mongodb-org-server-5.0.2-1.el8.x86_64 \
mongodb-org-mongos-5.0.2-1.el8.x86_64 \
```

4. Remove log files:

```
$ sudo rm -r /var/log/mongodb
```

5. Install Percona Server for MongoDB *using yum*.

**Note:** When you remove old packages, your existing configuration file is saved as `/etc/mongod.conf.rpmsave`. If you want to use this configuration with the new version, replace the default `/etc/mongod.conf` file. For example, existing data may not be compatible with the default WiredTiger storage engine.

# Start the mongod service:

```
$ sudo systemctl start mongod
```

To upgrade a replica set or a sharded cluster, use the *rolling restart* method. It allows you to perform the upgrade with minimum downtime. You upgrade the nodes one by one, while the whole cluster / replica set remains operational.

**See also:**

**MongoDB Documentation:**

- [Upgrade a Replica Set](#)
- [Upgrade a Sharded Cluster](#)

## 18.2 Minor upgrade of *Percona Server for MongoDB*

To upgrade *Percona Server for MongoDB* to the latest version, follow these steps:

1. Stop the *mongod* service:

```
$ sudo systemctl stop mongod
```

2. Install the latest version packages. Use the command relevant to your operating system.

---

**On Debian and Ubuntu:**

```
$ sudo apt install percona-server-mongodb
```

---

---

**On Red Hat Enterprise Linux or CentOS:**

```
$ sudo yum install percona-server-mongodb
```

---

3. Start the *mongod* service:

```
$ sudo systemctl start mongod
```

To upgrade a replica set or a sharded cluster, use the *rolling restart* method. It allows you to perform the upgrade with minimum downtime. You upgrade the nodes one by one, while the whole cluster / replica set remains operational.

## 18.3 Upgrading to *Percona Server for MongoDB* with data at rest encryption enabled

Steps to upgrade from MongoDB 5.0 Community Edition with data encryption enabled to *Percona Server for MongoDB* are different. *mongod* requires an empty `dbPath` data directory because it cannot encrypt data files in place. It must receive data from other replica set members during the initial sync. Please refer to the `switch_storage_engines` for more information on migration of encrypted data. [Contact us](#) for working at the detailed migration steps, if further assistance is needed.

## UPGRADING FROM *PERCONA SERVER FOR MONGODB 4.4 TO 5.0*

To upgrade *Percona Server for MongoDB* to version 5.0, you must be running version 4.4. Upgrades from earlier versions are not supported.

Before upgrading your production *Percona Server for MongoDB* deployments, test all your applications in a testing environment to make sure they are compatible with the new version. For more information, see [Compatibility Changes in MongoDB 5.0](#)

The general procedure for performing an in-place upgrade (where your existing data and configuration files are preserved) includes the following steps:

1. Stop the mongod instance
2. Enable Percona repository for *Percona Server for MongoDB 5.0*
3. Install new packages. Old packages are considered obsolete and automatically removed
4. Start the mongod instance

It is recommended to upgrade *Percona Server for MongoDB* from official Percona repositories using the [percona-release](#) tool and the corresponding package manager for your system. For more information, see [Installing Percona Server for MongoDB](#).

**Warning:** Perform a full backup of your data and configuration files before upgrading.

---

### Upgrading on Debian or Ubuntu

1. Stop the mongod instance:

```
$ sudo systemctl stop mongod
```

2. Enable Percona repository for *Percona Server for MongoDB 5.0*:

```
$ sudo percona-release enable psmdb-50
```

3. Update the local cache:

```
$ sudo apt update
```

4. Install *Percona Server for MongoDB 5.0* packages:

```
$ sudo apt install percona-server-mongodb
```

5. Start the mongod instance:

```
$ sudo systemctl start mongod
```

For more information, see *Installing Percona Server for MongoDB on Debian and Ubuntu*.

---

### Upgrading on RHEL and CentOS

1. Stop the mongod instance:

```
$ sudo systemctl stop mongod
```

2. Enable Percona repository for *Percona Server for MongoDB* 5.0:

```
$ sudo percona-release enable psmdb-50
```

3. Install *Percona Server for MongoDB* 5.0 packages:

4. Start the mongod instance:

```
$ sudo systemctl start mongod
```

For more information, see *Installing Percona Server for MongoDB on Red Hat Enterprise Linux and CentOS*.

---

After the upgrade, *Percona Server for MongoDB* is started with the feature set of 4.4 version. Assuming that your applications are compatible with the new version, enable 5.0 version features. Run the following command against the admin database:

```
db.adminCommand( { setFeatureCompatibilityVersion: "5.0" } )
```

**See also:**

### MongoDB Documentation:

- [Upgrade a Standalone](#)
- [Upgrade a Replica Set](#)
- [Upgrade a Sharded Cluster](#)



## UNINSTALLING *PERCONA SERVER FOR MONGODB*

To completely remove *Percona Server for MongoDB* you need to remove all the installed packages, data and configuration files. If you need the data, consider making a backup before uninstalling Percona Server for MongoDB.

Follow the instructions, relevant to your operating system:

---

### Uninstall on Debian and Ubuntu

You can remove *Percona Server for MongoDB* packages with one of the following commands:

- **apt-get remove** will only remove the packages and leave the configuration and data files.
- **apt purge** will remove all the packages with configuration files and data.

Choose which command better suits you depending on your needs.

1. Stop the **mongod** server:

```
$ sudo systemctl stop mongod
```

2. Remove the packages. There are two options.

- To keep the configuration and data files, run:

```
$ sudo apt remove percona-server-mongodb*
```

- To delete both the configuration and data files and the packages, run:

```
$ sudo apt purge percona-server-mongodb*
```

---

### Uninstall on RHEL and CentOS

1. Stop the Percona Server for MongoDB service:

```
$ sudo systemctl stop mongod
```

2. Remove the packages:

```
$ sudo yum remove percona-server-mongodb*
```

3. Remove the data and configuration files:

```
$ sudo rm -rf /var/lib/mongodb  
$ sudo rm -f /etc/mongod.conf
```

**Warning:** This will remove all the packages and delete all the data files (databases, tables, logs, etc.). You might want to back up your data before doing this in case you need the data later.

---

**Part V**

**Release Notes**



## PERCONA SERVER FOR MONGODB 5.0 RELEASE NOTES

### 21.1 *Percona Server for MongoDB 5.0.10-9 (2022-08-09)*

**Date** August 09, 2022

**Installation** *Installing Percona Server for MongoDB*

Percona Server for MongoDB 5.0.10-9 is an enhanced, source available, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 5.0.10 Community Edition.

It is rebased on [MongoDB 5.0.10 Community Edition](#) and supports MongoDB 5.0.10 protocols and drivers.

#### 21.1.1 Release Highlights

This release includes bug fixes and improvements provided by MongoDB and included in Percona Server for MongoDB. It specifically includes multiple fixes relate to sharding and the resharding operation. Bugs of note are the following:

- [SERVER-66418](#) - Fixed the issue with bad projection created during dependency analysis due to string order assumption. It resulted in the `PathCollision` error. The issue is fixed by improving dependency analysis for projections by folding dependencies into ancestor dependencies where possible.
- [SERVER-65821](#) - Fixed the deadlock situation in cross shard transactions that could occur when the FCV (Feature Compatibility Version) was set after the “prepared” state of the transactions. That ended up with both the `setFCV` thread and the `TransactionCoordinator` hung.
- [SERVER-65131](#) - This is a v6.0 backport fix to v5.0 that disables opportunistic read targeting (except for specified hedged reads) in order to prevent possible performance problems associated with uneven read distribution across the secondaries.
- [SERVER-66433](#) - Backported the check for user errors in case deadline on the migration destination manager is hit while waiting for a range to be cleared up. This prevents the balancer from getting blocked.
- [SERVER-67457](#) - Fixed the situation where the shards may skip certain steps in the resharding process but respond to the resharding coordinator as if they had. This can cause the resharding coordinator to continue to wait for updated states in the `config.reshardingOperations` document on the primary config server and stall indefinitely.
- [SERVER-66866](#) - Fixed the potential for the range deleter to wait unnecessarily during concurrent migrations between batches.
- [SERVER-66727](#) Fixed the issue where 2 very different measurements (one in the past and one in future) could be incorrectly included in the same buckettime-series bucket.
- [SERVER-64433](#) - Resolved potential problem when adding or removing shards and incorrectly gossiping a new topology time without it being majorly committed.

Find the full list of changes in the [MongoDB 5.0.10 Community Edition release notes](#).

### 21.1.2 Packaging Notes

Debian 9 (“Stretch”) is no longer supported.

**See also:**

Percona Blog: [OS Platform End of Life \(EOL\) Announcement for Debian Linux 9](#)

## 21.2 Percona Server for MongoDB 5.0.9-8 (2022-06-20)

**Date** June 20, 2022

**Installation** *Installing Percona Server for MongoDB*

Percona Server for MongoDB 5.0.9-8 is an enhanced, source available, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 5.0.9 Community Edition. It supports MongoDB 5.0.9 protocols and drivers.

### 21.2.1 Release Highlights

- Support of *multiple KMIP servers* adds failover to your data-at-rest encryption setup.
- Allow users to set KMIP client certificate password through a flag to simplify the migration from MongoDB Enterprise to *Percona Server for MongoDB*.
- *Percona Server for MongoDB* is now available on Ubuntu 22.04 (Jammy Jellyfish).
- Improvements to initial syncs from a secondary sync source.

The bug fixes, provided by MongoDB and included in Percona Server for MongoDB, are the following:

- [SERVER-65137](#) - Detect namespace changes when refreshing Collection after yielding to maintain data consistency and avoid stale catalogs.
- [SERVER-64822](#) - Fixed the issue with releasing the critical state too early when sharding an empty collection. This could result in unwanted writes to that collection.
- [WT-9096](#) - Fixed the wrong key/value returning during search near when the key doesn't exist
- [SERVER-66041](#) - Prevented shard imbalances due to chunk cloner disregarding oversized chunks that contain only 1 oversized document

Find the full list of changes in the [MongoDB 5.0.9 Community Edition release notes](#).

### 21.2.2 Improvements

- [PSMDB-1045](#): Add support for several KMIP servers
- [PSMDB-1043](#): Remove the `kmipClientKeyFile` option and include both the client private key and public certificate in the file specified by the `kmipClientCertificateFile` option.
- [PSMDB-1044](#): Make the `kmipPort` option not mandatory and assign the default value
- [PSMDB-1054](#): Add the ability to specify the password for the KMIP client keys and certificates to simplify migration from MongoDB Enterprise.

## 21.3 Percona Server for MongoDB 5.0.8-7

**Date** May 10, 2022

**Installation** [Installing Percona Server for MongoDB](#)

Percona Server for MongoDB 5.0.8-7 is an enhanced, source available, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 5.0.8 Community Edition. It supports MongoDB 5.0.8 protocols and drivers.

### 21.3.1 Release Highlights

*Percona Server for MongoDB* now supports master key rotation for data encrypted using the [Keys Management Interoperability Protocol \(KMIP\)](#) protocol (tech preview feature). This improvement allows users to comply with regulatory standards for data security.

Other improvements and bug fixes introduced by MongoDB and included in *Percona Server for MongoDB* are the following:

- [WT-8924](#) - Fixed a bug that causes replication to stall on secondary replica set members in a sharded cluster handling cross-shard transactions. It is caused by WiredTiger to erroneously return a write conflict when deciding if an update to a record is allowed. If MongoDB decides to retry the operation that caused the conflict in WiredTiger, it will enter an indefinite retry loop, and oplog application will stall on secondary nodes.

If this bug is hit, the secondary nodes will experience indefinite growth in replication lag. Restart the secondary nodes to resume replication.

This bug affects MongoDB 4.4.10 through 4.4.13 and 5.0.4 to 5.0.7.

Update to the latest version to avoid the secondary replication stall and lag issues.

- [SERVER-63531](#) - Fixed the issue with a `commitQuorum` incorrectly allowing non-voting `buildIndexes:false` nodes to be included in the `commitQuorum` count. It also fixed an error message relating to data-bearing nodes and quorum regarding to non-voting nodes.
- [SERVER-63387](#) - Fixed the order of backup blocks returned to the user to match the order retrieved from WiredTiger.
- [SERVER-65261](#) - Fixed the issue reporting an incorrect number of documents deleted from a capped collection when utilizing a collection scan to perform the delete action.

Find the full list of changes in the [MongoDB 5.0.8 Community Edition release notes](#).

### 21.3.2 Improvements

- [PSMDB-1011](#): Add KMIP master key rotation

### 21.3.3 Bugs Fixed

- [PSMDB-1030](#): Fix descriptions and mutual dependencies of KMIP related options for mongod and perconadecrypt

## 21.4 Percona Server for MongoDB 5.0.7-6

**Date** April 20, 2022

**Installation** [Installing Percona Server for MongoDB](#)

Percona Server for MongoDB 5.0.7-6 is an enhanced, source-available and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 5.0.7 Community Edition. It supports MongoDB 5.0.7 protocols and drivers.

---

**Note:** This is the first version that is production capable as earlier upstream versions of MongoDB had the number of potential serious bugs.

---

### 21.4.1 Release Highlights

*Percona Server for MongoDB* now supports *Keys Management Interoperability Protocol (KMIP)* so that users can store encryption keys in their favorite KMIP-compatible key manager to set up encryption at rest. This is a tech preview feature.

The list of bug fixes introduced by MongoDB and included in Percona Server for MongoDB is the following:

- [SERVER-63203](#) - Fixed the issue where having a large number of split points causes the chunk splitter to not function correctly and huge chunks would not be split without manual intervention. This can be caused when having small shard key ranges and a very high number of documents and where more than 8192 split points would be needed.
- [SERVER-64517](#) - Recover the RecoverableCriticalSection service after the initialSync and startupRecovery stages have completed. This prevents a started up shard to miss an in-memory critical section of a resharded collection.
- [SERVER-64403](#) - Fixed an issue that occurred during the attempt to perform the collation-encoding of a document with a missing sort attribute. In this case an invariant is violated and mongod crashes.
- [SERVER-63722](#) - Fixed an issue when the rename collection (sharding step) participants can get stuck.
- [SERVER-61769](#) - Fixed an issue with idle cursors remaining open when the client attempts to run an aggregation with \$out or \$merge in a transaction on a sharded cluster.
- [SERVER-60412](#) - Check if the host has cgroups v2 enabled and read the memory limits according to that.
- [WT-7922](#) - Report an error when the WiredTiger version file is empty or missing during a startup.

Find the full list of changes in the [MongoDB 5.0.7 Community Edition Release notes](#).



## 21.4.2 New Features

- **PSMDB-971**: Added support for KMIP (Keys Management Interoperability Protocol) encryption. Now users can store encryption keys in their favorite KMIP-compatible key manager to set up encryption at rest.

## 21.4.3 Bugs Fixed

- **PSMDB-1010**: Fixed the parameters order in the LOGV2\_DEBUG statement for LDAP logging.
- **PSMDB-957**: Fixed server crash caused by LDAP misconfiguration. Now the server logs an error message and exits.

## 21.5 Percona Server for MongoDB 5.0.6-5

**Date** February 10, 2022

**Installation** [Installing Percona Server for MongoDB](#)

Percona Server for MongoDB 5.0.6-5 is an enhanced, open source, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 5.0.6 Community Edition. It supports MongoDB 5.0.6 protocols and drivers.

**Warning:** Inconsistent data is observed after the upgrade from MongoDB 4.4.3 and 4.4.4 to versions 4.4.8+ and 5.0.2+. This issue is fixed upstream in versions 4.4.11 and 5.0.6. *Percona Server for MongoDB* also includes the fix in versions 4.4.12-12 and 5.0.6-5

See the upgrade recommendations below:

- Clusters on versions 4.4.0 and 4.4.1 are safe to upgrade to 4.4.8+ or 5.0.2+ but should upgrade to recommended versions 4.4.11+ or 5.0.5+
- Clusters on versions 4.4.2, 4.4.3, or 4.4.4 should **downgrade** to 4.4.1 and then upgrade to versions 4.4.11+ or 5.0.5+.
- Clusters running versions 4.4.5 - 4.4.7 can and should upgrade to versions 4.4.11+ or 5.0.5+.

Note that clusters running versions 4.4.2 - 4.4.8 are affected by the bug [WT-7995](#). See [WT-7995](#) for specific explanation and instructions on running the `validate` command to check for data inconsistencies. These data inconsistencies can lead to data loss if not identified and repaired at this point between versions 4.4.8 and 4.4.9.

If the `validate` command output reports any failures, resync the impacted node from an unaffected node. **The validate command must be run against all collections in the database. This process can be resource intensive and can negatively impact performance.**

### 21.5.1 Release Highlights

- **WT-8395** - Fixed an issue with inconsistent data observed during the direct upgrade from from 4.4.3 and 4.4.4 to 4.4.8+ and 5.0.2+. Data inconsistency was caused by the incorrect checkpoint metadata to sometimes be recorded by MongoDB versions 4.4.3 and 4.4.4. WiredTiger used this metadata during node startup that could lead to data corruption and could cause the DuplicateKey error. The fix requires the upgrade to versions 4.4.11+ or 5.0.5+.
- **SERVER-62245** - Fixed an issue unavailability of a shard in sharded clusters. Affects MongoDB versions 5.0.0 - 5.0.5. Requires the following steps:

- Restarting the shards as replica sets
  - Checking that range deletion tasks are consistent with migration coordinators
  - Majority-deleting all migration coordinators with a definitive decision
  - Restarting the nodes as shards.
- [SERVER-61194](#) - Fixed time-series bucket OID collisions by adding the difference between the actual timestamp and the rounded timestamp to the instance portion of the OID.
  - [SERVER-62147](#) - Fixed broken OP\_QUERY exhaust cursor implementation.

## 21.5.2 Bugs Fixed

- [PSMDB-950](#): Fixed LDAP authentication using mongo CLI for *Percona Server for MongoDB* installed from a tarball.

## 21.6 Percona Server for MongoDB 5.0.5-4

**Date** December 28, 2021

**Installation** [Installing Percona Server for MongoDB](#)

Percona Server for MongoDB 5.0.5-4 is an enhanced, source-available, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 5.0.5 Community Edition. It supports MongoDB 5.0.5 protocols and drivers.

---

**Important:** One of the main new features for MongoDB v5.0 is live [resharding](#). Be aware that there have been multiple bugs related to this new feature through the releases to date (5.0.0 through 5.0.5). Please test thoroughly in your development and staging environments and use this feature on live production systems with caution.

---

### 21.6.1 Release Highlights

The bug fixes and improvements, provided by MongoDB and included in Percona Server for MongoDB, are the following:

- [SERVER-59858](#) - Added histograms to track latency for tasks scheduled on the reactor thread.
- [SERVER-61483](#) - Fixed an issue when resharding a collection that could cause data inconsistency (lost writes) due to incorrect actions by the ReshardingCoordinator and attempts to commit anyway. Also could cause `fassert()` to config server primary.
- [SERVER-61482](#) - Fixed an issue with stalls on the config server. Updates to config server during resharding may wait too long for oplog slot thus stalling replication on config server indefinitely.
- [SERVER-61633](#) Fixed a resharding issue relating to RecipientStateMachine that caused the server to crash

Find the full list of changes in the [MongoDB 5.0.5 Community Edition release notes](#).

## 21.6.2 Bugs Fixed

- [PSMDB-756](#): Fixed an issue with unmet dependencies for installing MongoDB on Debian (Thanks to Stefan Schlesi for reporting this issue)

## 21.7 *Percona Server for MongoDB 5.0.4-3 (Release Candidate)*

**Date** December 8, 2021

**Installation** [Installing Percona Server for MongoDB](#)

Percona Server for MongoDB 5.0.4-3 is an enhanced, source-available, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 5.0.4 Community Edition.

It is rebased on [MongoDB 5.0.4 Community Edition](#) and supports MongoDB 5.0.4 protocols and drivers.

---

**Note:** Due to many core changes to WiredTiger and the core server introduced to facilitate new features such as resharding, time series collections, and API versioning, MongoDB 5.0.x is still unstable. We do not recommend using this release candidate for production environments.

---

### 21.7.1 Release Highlights

The bug fixes, provided by MongoDB and included in Percona Server for MongoDB, are the following:

- Fixed delays in establishing egress connections on mongos due to delayed responses from `libcrypto.so`
- Allowed replication state changes to interrupt lock acquisition. This interruption fixes deadlocks that may occur when a primary node steps down with profiling enabled.

Find the full list of changes in the [MongoDB 5.0.4 Community Edition release notes](#).

## 21.8 *Percona Server for MongoDB 5.0.3-2 (Release Candidate)*

**Date** October 14, 2021

**Installation** [Installing Percona Server for MongoDB](#)

Percona Server for MongoDB 5.0.3-2 is an enhanced, source-available, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 5.0.3 Community Edition. It supports MongoDB 5.0.3 protocols and drivers.

---

**Note:** Due to many core changes to WiredTiger and the core server introduced to facilitate new features such as resharding, time series collections, and API versioning, MongoDB 5.0.x is still unstable. We do not recommend using this release candidate for production environments.

---

**Warning:** Beginning with MongoDB 5.0.1, several data impacting or corrupting bugs were introduced. Details are listed below.

These bugs are fixed in MongoDB 5.0.3. Percona Server for MongoDB 5.0.3-2 includes the upstream fixes of these bugs.

Unless you are already running a version of MongoDB 5.0.x in your production environments, consider waiting a while longer for any other issues to be worked through.

If you are running any version of MongoDB 5.0.x already, please upgrade to MongoDB 5.0.3 or Percona Server for MongoDB 5.0.3-2 as soon as possible. Please follow closely all MongoDB Inc upstream recommendations regarding avoiding unclean shutdowns, check collections with the `validate` command (watch for performance impacts), and make sure and resync any impacted nodes from unaffected nodes as recommended in [WT-7995](#)

## 21.8.1 Release Highlights

The bug fixes, provided by MongoDB and included in Percona Server for MongoDB, are the following:

- [WT-7995](#) - Checkpoint thread can read and persist inconsistent version of data to disk. Can cause Duplicate Key error on startup and prevent the node from starting. Unclean shutdowns can cause data inconsistency within documents, deleted documents to still exist, incomplete query results due to lost or inaccurate index entries, and/or missing documents. Affects MongoDB versions 5.0.0 through 5.0.2. Upgrade to fixed version of MongoDB 5.0.3 / Percona Server for MongoDB 5.0.3-2 as soon as possible.
- [WT-7984](#) - Bug that could cause Checkpoint thread to omit a page of data. If the server experiences an unclean shutdown, an inconsistent checkpoint is used for recovery and causes data corruption. Fixed in version 5.0.3.

Requires the `validate` command to be run and possible data remediation via complete initial sync.

Find the full list of changes in the [MongoDB 5.0.3 Community Edition release notes](#).

## 21.8.2 Improvements

- [PSMDB-918](#): Disable the deletion of the `mongod` user in RPM packages - This preserves the permissions to the MongoDB data directory for the `mongod` user since its user ID and group ID remain unchanged.

## 21.9 Percona Server for MongoDB 5.0.2-1 (Release Candidate)

**Date** August 16, 2021

**Installation** [Installing Percona Server for MongoDB](#)

---

**Note:** With a lot of new features and modifications introduced, we recommend using this release candidate in testing environments only.

---

**Warning:** Percona Server for MongoDB 5.0.2-1 is not recommended for production use due to the following critical issues: [WT-7984](#) and [WT-7995](#). These issues are fixed in MongoDB 5.0.3 Community Edition and [Percona Server for MongoDB 5.0.3-2 \(Release Candidate\)](#).

We recommend you to upgrade to Percona Server for MongoDB 5.0.3-2 and run the `validate` command on every collection on every replica set node.

Read more about the post-upgrade steps in [WT-7984](#) and [WT-7995](#).

We are pleased to announce the release candidate of Percona Server for MongoDB 5.0.2-1. It is available for [download](#) from [Percona website](#) and for installation from [Percona Software Repositories](#).

Percona Server for MongoDB 5.0.2-1 is an enhanced, source-available, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 5.0.2 Community Edition. It includes [all features of MongoDB 5.0.2 Community Edition](#). The most notable among these are the following:

- [Resharding](#) allows you to select a new shard key for a collection and then works in the background to correct any data distribution problems caused by bad shard keys and improve performance.
- [Time Series Collections](#) are aimed at storing sequences of measurements over a period of time. These specialized collections will store data in a highly optimized way that will improve query efficiency, allow data analysis in real-time, and optimize disk usage.
- [Resumable Index Builds](#) means that the index build for a collection continues if a primary node in a replica set is switched to another server or when a server restarts. The build process is saved to disk and resumes from the saved position. This allows DBAs to perform maintenance and not worry about losing the index build in the process.
- [Window operators](#) allow operations on a specified span of documents known as window. `$setWindowFields` is a new pipeline stage to operate with these documents.
- [Versioned API](#) allows specifying which API version your application communicating with MongoDB runs against. Versioned API detaches the application's lifecycle from that of the database. As a result, you modify the application only to introduce new features instead of having to maintain compatibility with the new version of MongoDB.

In addition, [new aggregation operators](#) such as `$count`, `$dateAdd`, `$dateDiff`, `$dateSubtract`, `$sampleRate` and `$rand` are available with this release.

Percona Server for MongoDB 5.0.2-1 extends this feature set by providing *enterprise-level enhancements* for free.

Percona Server for MongoDB 5.0.2-1 fully supports MongoDB 5.0.2 Community Edition protocols and drivers and requires no changes to MongoDB applications or code.



# **Part VI**

## **Reference**





## CONTACTING, CONTRIBUTING, REPORTING BUGS

If you want to contact developers, use the [community forum](#).

---

**Note:** Please search the forum for similar questions and discussions before opening a new thread.  
You will need to sign up for an account to post in the forum.

---

If you want to explore source code and contribute to the project use the [GitHub repo](#) and the [Contributing guide](#).

---

**Note:** Search existing pull requests and recent commits for code that may fix what you are planning to suggest.  
You will need a public GitHub account and request contributor access to the repo.

---

If you want to report a bug or feature request, use the [PSMDB project in JIRA](#).

---

**Note:** Search JIRA for existing tickets before submitting a bug or feature request.  
You will need a JIRA account to [report bugs](#).

---



## GLOSSARY

**ACID** Set of properties that guarantee database transactions are processed reliably. Stands for *Atomicity*, *Consistency*, *Isolation*, *Durability*.

**Atomicity** Atomicity means that database operations are applied following a “all or nothing” rule. A transaction is either fully applied or not at all.

**Consistency** Consistency means that each transaction that modifies the database takes it from one consistent state to another.

**Durability** Once a transaction is committed, it will remain so.

**Foreign Key** A referential constraint between two tables. Example: A purchase order in the purchase\_orders table must have been made by a customer that exists in the customers table.

**Isolation** The Isolation requirement means that no transaction can interfere with another.

**Jenkins** *Jenkins* is a continuous integration system that we use to help ensure the continued quality of the software we produce. It helps us achieve the aims of:

- no failed tests in trunk on any platform,
- aid developers in ensuring merge requests build and test on all platforms,
- no known performance regressions (without a damn good explanation).

**Kerberos** Kerberos is an authentication protocol for client/server authentication without sending the passwords over an insecure network. Kerberos uses symmetric encryption in the form of tickets - small pieces of encrypted data used for authentication. A ticket is issued for the client and validated by the server.

**Rolling restart** A rolling restart (rolling upgrade) is shutting down and upgrading nodes one by one. The whole cluster remains operational. There is no interruption to clients assuming the elections are short and all writes directed to the old primary use the retryWrite mechanism.



## COPYRIGHT AND LICENSING INFORMATION

### 24.1 Documentation Licensing

This software documentation is (C)2016-2022 Percona LLC and/or its affiliates and is distributed under the [Creative Commons Attribution 4.0 International License](#).

### 24.2 Software License

*Percona Server for MongoDB* is [source-available](#) software.



## TRADEMARK POLICY

This Trademark Policy is to ensure that users of Percona-branded products or services know that what they receive has really been developed, approved, tested and maintained by Percona. Trademarks help to prevent confusion in the marketplace, by distinguishing one company's or person's products and services from another's.

Percona owns a number of marks, including but not limited to Percona, XtraDB, Percona XtraDB, XtraBackup, Percona XtraBackup, Percona Server, and Percona Live, plus the distinctive visual icons and logos associated with these marks. Both the unregistered and registered marks of Percona are protected.

Use of any Percona trademark in the name, URL, or other identifying characteristic of any product, service, website, or other use is not permitted without Percona's written permission with the following three limited exceptions.

*First*, you may use the appropriate Percona mark when making a nominative fair use reference to a bona fide Percona product.

*Second*, when Percona has released a product under a version of the GNU General Public License ("GPL"), you may use the appropriate Percona mark when distributing a verbatim copy of that product in accordance with the terms and conditions of the GPL.

*Third*, you may use the appropriate Percona mark to refer to a distribution of GPL-released Percona software that has been modified with minor changes for the sole purpose of allowing the software to operate on an operating system or hardware platform for which Percona has not yet released the software, provided that those third party changes do not affect the behavior, functionality, features, design or performance of the software. Users who acquire this Percona-branded software receive substantially exact implementations of the Percona software.

Percona reserves the right to revoke this authorization at any time in its sole discretion. For example, if Percona believes that your modification is beyond the scope of the limited license granted in this Policy or that your use of the Percona mark is detrimental to Percona, Percona will revoke this authorization. Upon revocation, you must immediately cease using the applicable Percona mark. If you do not immediately cease using the Percona mark upon revocation, Percona may take action to protect its rights and interests in the Percona mark. Percona does not grant any license to use any Percona mark for any other modified versions of Percona software; such use will require our prior written permission.

Neither trademark law nor any of the exceptions set forth in this Trademark Policy permit you to truncate, modify or otherwise use any Percona mark as part of your own brand. For example, if XYZ creates a modified version of the Percona Server, XYZ may not brand that modification as "XYZ Percona Server" or "Percona XYZ Server", even if that modification otherwise complies with the third exception noted above.

In all cases, you must comply with applicable law, the underlying license, and this Trademark Policy, as amended from time to time. For instance, any mention of Percona trademarks should include the full trademarked name, with proper spelling and capitalization, along with attribution of ownership to Percona Inc. For example, the full proper name for XtraBackup is Percona XtraBackup. However, it is acceptable to omit the word "Percona" for brevity on the second and subsequent uses, where such omission does not cause confusion.

In the event of doubt as to any of the conditions or exceptions outlined in this Trademark Policy, please contact [trade-marks@percona.com](mailto:trade-marks@percona.com) for assistance and we will do our very best to be helpful.





## Symbols

--auditDestination  
    command line option, [40](#)  
--auditFilter  
    command line option, [40](#)  
--auditFormat  
    command line option, [40](#)  
--auditPath  
    command line option, [40](#)  
--inMemorySizeGB  
    command line option, [21](#)  
--inMemoryStatisticsLogDelaySecs  
    command line option, [22](#)

## A

ACID, [111](#)  
Atomicity, [111](#)

## C

command line option  
    --auditDestination, [40](#)  
    --auditFilter, [40](#)  
    --auditFormat, [40](#)  
    --auditPath, [40](#)  
    --inMemorySizeGB, [21](#)  
    --inMemoryStatisticsLogDelaySecs, [22](#)  
Consistency, [111](#)

## D

Durability, [111](#)

## F

Foreign Key, [111](#)

## I

Isolation, [111](#)

## J

Jenkins, [111](#)

## K

Kerberos, [111](#)

## R

Rolling restart, [111](#)