

Server for MongoDB Documentation 6.0

6.0.5-4 (March 29, 2023)

Table of contents

1. Percona Server for MongoDB 6.0 Documentation	4
1.1 Features	4
1.2 Get started	4
2. Percona Server for MongoDB feature comparison	6
2.1 Profiling Rate Limiting	6
3. Install PSMDB	8
3.1 Install Percona Server for MongoDB	8
3.2 Install Percona Server for MongoDB on Debian and Ubuntu	10
3.3 Install Percona Server for MongoDB on Red Hat Enterprise Linux and derivatives	13
3.4 Install Percona Server for MongoDB from binary tarball	16
3.5 Run Percona Server for MongoDB in a Docker container	18
4. Features	20
4.1 Storage	20
4.2 Backup	28
4.3 Authentication	35
4.4 Encryption	71
4.5 Auditing	83
4.6 Profiling rate limit	88
4.7 Log redaction	90
4.8 Additional text search algorithm - ngram	91
5. Manage PSMDB	92
5.1 Percona Server for MongoDB parameter tuning guide	92
5.2 Upgrade	94
5.3 Uninstall Percona Server for MongoDB	101
6. Release notes	103
6.1 Percona Server for MongoDB 6.0 Release Notes	103
6.2 Percona Server for MongoDB 6.0.5-4 (2023-03-29)	104
6.3 Percona Server for MongoDB 6.0.4-3 (2023-01-30)	106
6.4 Percona Server for MongoDB 6.0.3-2 (2022-12-07)	107
6.5 Percona Server for MongoDB 6.0.2-1 (2022-10-31)	108
7. Glossary	111
7.1 ACID	111
7.2 Atomicity	111
7.3 Consistency	111
7.4 Durability	111

7.5	Foreign Key	111
	Silsolation	111
	Jenkins	111
	Kerberos	111
7.9	Rolling restart	111
7.10	0 Technical preview feature	112
8. C	opyright and licensing information	114
	Documentation licensing	114
	2 Software license	114
9. Tr	rademark policy	116

1. Percona Server for MongoDB 6.0 Documentation

Percona Server for MongoDB is a free, enhanced, fully compatible, source available, drop-in replacement for MongoDB 6.0 Community Edition with enterprise-grade features. It requires no changes to MongoDB applications or code.

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.

This is the documentation for the latest release, Percona Server for MongoDB 6.0.5-4 (Release Notes).

1.1 Features

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
- AWS IAM authentication (a technical preview feature)
- 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

1.2 Get started

• Install Percona Server for MongoDB

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services, contact Percona Sales.

2. Percona Server for MongoDB feature comparison

Percona Server for MongoDB 6.0 is based on MongoDB 6.0. 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	WiredTiger (default)Percona Memory Engine	WiredTiger (default)In-Memory (Enterprise only)
Encryption-at-Rest	- Key servers = Hashicorp Vault, KMIP- Fully open source	Key server = KMIPEnterprise only
Hot Backup	YES (replica set)	NO
LDAP Authentication	(legacy) LDAP authentication with SASL	Enterprise only
LDAP Authorization	YES	Enterprise only
Kerberos Authentication	YES	Enterprise only
AWS IAM authentication	YES	MongoDB Atlas
Audit Logging	YES	Enterprise only
Log redaction	YES	Enterprise only
SNMP Monitoring	NO	Enterprise only
Database profiler	YES with therateLimit argument	YES

2.1 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.

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services, contact Percona Sales.

Install PSMDB

3.1 Install 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.

3.1.1 System requirements

x86_64

Percona Server for MongoDB has the same system requirements as the MongoDB Community Edition.

Starting in MongoDB 5.0, mongod, mongos, and the legacy mongo shell are supported on x86_64 platforms that must meet these minimum micro-architecture requirements:

- Only Oracle Linux running the Red Hat Compatible Kernel (RHCK) is supported. MongoDB does not support the Unbreakable Enterprise Kernel (UEK).
- MongoDB 5.0 and above requires use of the AVX instruction set, available on select Intel and AMD processors.

ARM64

Percona Server for MongoDB requires the ARMv8.2-A or later microarchitecture.

3.1.2 Installation instructions

Choose how you wish to install Percona Server for MongoDB:

- From Percona repositories (recommended):
 - on Debian or Ubuntu
 - on RHEL or CentOS
- From binary tarballs
- Manually from Percona website



Note

Make sure that all dependencies are satisfied

• Run in Docker

3.1.3 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 5.0.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

3.2 Install Percona Server for MongoDB on Debian and Ubuntu

This document describes how to install Percona Server for MongoDB from Percona repositories on DEB-based distributions such as Debian and Ubuntu.



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.

3.2.1 Package Contents

Package	Contains
percona-server- mongodb	The mongosh shell, import/export tools, other client utilities, server software, default configuration, and init.d scripts.
percona-server- mongodb-server	The mongod server, default configuration files, and init.d scripts
percona-server- mongodb-shell	The mongosh 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

3.2.2 Procedure

Configure Percona repository

Percona provides the percona-release configuration tool that simplifies operating repositories and enables to install and update both Percona Backup for MongoDB packages and required dependencies smoothly.

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
```

After you install this package, you have the access to Percona repositories. 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-60 release
```

4. Remember to update the local cache:

```
$ sudo apt update
```

Install Percona Server for MongoDB

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 | 6.0.2-1.buster | http://repo.percona.com/psmdb-60/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 6.0.2-1, run the following command:

```
$ sudo apt install percona-server-mongodb=6.0.2-1.buster percona-server-mongodb-
mongos=6.0.2-1.buster percona-server-mongodb-shell=6.0.2-1.buster percona-server-
mongodb-server=6.0.2-1.buster percona-server-mongodb-tools=6.0.2-1.buster
```

3.2.3 Run Percona Server for MongoDB

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

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

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services, contact Percona Sales.

3.3 Install Percona Server for MongoDB on Red Hat Enterprise Linux and derivatives

This document describes how to install Percona Server for MongoDB on RPM-based distributions such as Red Hat Enterprise Linux and compatible derivatives.



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.

3.3.1 Package Contents

Package	Contains
percona-server- mongodb	The mongosh shell, import/export tools, other client utilities, server software, default configuration, and init.d scripts.
percona-server- mongodb-server	The mongod server, default configuration files, and init.d scripts
percona-server- mongodb-shell	The mongosh 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

3.3.2 Procedure

Percona provides the percona-release configuration tool that simplifies operating repositories and enables to install and update both Percona Backup for MongoDB packages and required dependencies smoothly.

Configure Percona repository

1. Install percona-release:

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

Example output

2. Enable the repository:

```
$ sudo percona-release enable psmdb-60 release
```

🧪 s

See also

Percona Software Repositories Documentation

Install Percona Server for MongoDB packages

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 6.0.2-1.el8 psmdb-60-release-x86_64

2. Install a specific version packages. For example, to install *Percona Server for MongoDB* 6.0.2-1, run the following command:

\$ sudo yum install percona-server-mongodb-6.0.2-1.el8

3.3.3 Run Percona Server for MongoDB



Note

If you use 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.

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

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: service mongod status

```
$ sudo systemctl status mongod
```

Stopping the service

Stop the service using the following command: service mongod stop

```
$ sudo systemctl stop mongod
```

Restarting the service

Restart the service using the following command: service mongod restart

```
$ sudo systemctl restart mongod
```

Run after reboot

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

To make it start automatically after reboot, enable it using the systemctl utility:

```
$ sudo systemctl enable mongod
```

Then start the mongod service:

```
$ sudo systemctl start mongod
```

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services, contact Percona Sales.

3.4 Install 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 the following 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.
- percona-mongodb-mongosh-<version>-x86 64.tar.gz is the tarball for mongosh shell.

3.4.1 Preconditions

The following packages are required for the installation.

On Debian and Ubuntu

- libcurl4
- libsasl2-modules
- libsasl2-modules-gssapi-mit

On Red hat Enterprise Linux and derivatives

- libcurl
- cyrus-sasl-gssapi
- cyrus-sasl-plain

3.4.2 Procedure

The steps below describe the installation on Debian 10 ("buster").

1. Fetch and the binary tarballs:

```
$ wget https://www.percona.com/downloads/percona-server-mongodb-6.0/percona-server-
mongodb-6.0.2-1/binary/tarball/percona-server-mongodb-6.0.2-1-x86_64.glibc2.17.tar.gz\
$ wget https://www.percona.com/downloads/percona-server-mongodb-6.0/percona-server-
mongodb-6.0.2-1/binary/tarball/percona-mongodb-mongosh-1.6.0-x86_64.tar.gz
```

2. Extract the tarballs

```
$ tar -xf percona-server-mongodb-6.0.2-1-x86_64.glibc2.17.tar.gz
$ tar -xf percona-mongodb-mongosh-1.6.0-x86_64.tar.gz
```

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

```
$ export PATH=~/percona-server-mongodb-6.0.2-1/bin/:~/percona-mongodb-mongosh-1.6.0/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.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services, contact Percona Sales.

3.5 Run Percona Server for MongoDB in a Docker container

Docker images of Percona Server for MongoDB are hosted publicly on Docker Hub.

For more information about using Docker, see the Docker Docs.



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

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

To run the latest Percona Server for MongoDB 6.0 in a Docker container, run the following command as the root user or via sudo:

```
On x86_64 platforms

$ docker run -d --name psmdb --restart always \
percona/percona-server-mongodb:6.0

On ARM64 platforms

$ docker run -d --name psmdb --restart always \
```

Replace the <TAG> with the desired version (for example, 6.0.4-3-arm64)

The 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:6.0 / percona/percona-server-mongodb:<TAG>-arm64 is the name and version tag of the image to derive the container from.

3.5.1 Connecting from another Docker container

percona/percona-server-mongodb:<TAG>-arm64

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.

3.5.2 Connecting with the mongosh shell

To start another container with the mongosh shell that connects to your Percona Server for MongoDB container, run the following command:

```
\ docker run -it --link psmdb --rm percona/percona-server-mongodb:6.0 mongosh mongodb:// MONGODB_SERVER:PORT/DB_NAME
```

Set MONGODB_SERVER, PORT, and DB_NAME with the IP address of the psmdb container, the port of your MongoDB Server (default value is 27017), and the name of the database you want to connect to.

You can get the IP address by running this command:

```
$ docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' psmdb
```

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services, contact Percona Sales.

4. Features

4.1 Storage

4.1.1 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.

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

Usage

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
```

Configuration

You can configure Percona Memory Engine using either command-line options or corresponding parameters in the /etc/mongod.conf file. The following are the configuration examples:

Configuration file

The configuration file is formatted in YAML

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

Command line

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

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

OPTIONS

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

Command line	Configuration file	Default	Description
inMemorySizeGB()	storage.inMemory.engineConfig.inMemorySizeGB	50% of total memory minus 1024 MB, but not less than 256 MB	Specifies the maximum memory in gigabytes to use for data
inMemoryStatisticsLogDelaySecs()	storage.inMemory.engineConfig.statisticsLogDelaySecs	0	Specifies the number of seconds between writes to statistics log. If 0 is specified then statistics are not logged

Switching storage engines

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.

PROCEDURE

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

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.

1. Stop mongod

```
$ service mongod stop
```

2. Edit the configuration file

```
storage:
  dbPath: <newDataDir>
  engine: inmemory
```

3. Start mongod

```
$ service mongod start
```

Permanent switch to Percona Memory Engine without any valuable data in your database

Clean out the dbPath data directory (by default, /var/lib/mongodb) and edit the configuration file:

1. Stop mongod

```
$ service mongod stop
```

2. Clean out the dbPath data directory

```
$ sudo rm -rf <dbpathDataDir>
```

3. Edit the configuration file

```
storage:
  dbPath: <newDataDir>
  engine: inmemory
```

4. Start mongod

```
$ service mongod start
```

Switch to Percona Memory Engine with data migration and compatibility

Standalone instance

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

1. Export the dataDir contents

```
$ mongodump --out <dumpDir>
```

2. Stop mongod

```
$ service mongod stop
```

3. Clean out the dbPath data directory

```
$ sudo rm -rf <dbpathDataDir>
```

- 4. 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
- 5. Start mongod

```
$ service mongod start
```

6. Restore the database

```
$ mongorestore <dumpDir>
```

Replica set

Use the "rolling restart" process.

- 1. Switch to the Percona Memory Engine on the secondary node. Clean out the dbPath data directory and edit the configuration file:
- 2. Stop mongod

```
$ service mongod stop
```

3. Clean out the dbPath data directory

```
$ sudo rm -rf <dbpathDataDir>
```

4. Edit the configuration file

```
storage:
  dbPath: <newDataDir>
  engine: inmemory
```

5. Start mongod

```
$ service mongod start
```

- 6. Wait for the node to rejoin with the other nodes and report the SECONDARY status.
- 7. Repeat the procedure to switch the remaining nodes to Percona Memory Engine.

DATA AT REST ENCRYPTION

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.

4.1.2 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.

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

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
```

Switching storage engines

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 <code>dbPath</code> data directory for a different storage engine is the simplest solution. Yet when you switch between disk-using storage engines (e.g. from <code>WiredTiger</code> 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.

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
$ # 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.

 $\bullet \ \text{for a standalone instance or a single-node replica set, use the } \ \text{mongodump} \ \ \text{and} \ \ \text{mongorestore} \ \ \text{utilities:}$

```
$ 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>
```

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.

Configure Percona Memory Engine

Use either the command line or edit the configuration file to configure Percona Memory Engine

```
Command line

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

Configuration file

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

AVAILABLE OPTIONS

Command line	Config file	Default value	Description
inMemorySizeGB()	storage.inMemory.engineConfig.inMemorySizeGB	50% of total memory minus 1024 MB, but not less than 256 MB	Specifies the maximum memory in gigabytes to use for data.
inMemoryStatisticsLogDelaySecs()	<pre>storage.inMemory.engineConfig.statisticsLogDelaySecs</pre>	0	Specifies the number of seconds between writes to statistics log. If 0 is specified then statistics are not logged.

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

4.2 Backup

4.2.1 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.



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.

Make a backup

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

1. Provide access to the backup directory for the mongod user:

```
$ sudo chown mongod:mongod <backupDir>
```

2. 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 $\{ \text{"ok"} : 1 \}$ 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" }
```

Save a backup to a TAR archive

To save a backup as a tar archive, use the archive field to specify the destination path:

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

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	Туре	Description
bucket	string	The only mandatory field. Names are subject to restrictions described in the Bucket Restrictions and Limitations section of Amazon S3 documentation
path	string	The virtual path inside the specified bucket where the backup will be created. If the path 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 false . For more information, see Virtual Hosting of Buckets in the Amazon S3 documentation.
region	string	The name of an AWS region. The default region is US_EAST_1 . For more information see AWS Service Endpoints 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 default is used.
accessKeyId	string	The access key id
secretAccessKey	string	The secret access key

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 \sim /.aws/credentials.

Example credentials file

```
[default]
aws_access_key_id = ABC123XYZ456QQQAAAFFF
aws_secret_access_key = zuf+secretkey0secretkey1secretkey2
[localminio]
aws_access_key_id = ABCABCABCABC55566678
aws_secret_access_key = secretaccesskey1secretaccesskey2secretaccesskey3
```

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"}})
```



AWS Documentation: Providing AWS Credentials

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.



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

1. Stop the mongod service

```
$ systemctl stop mongod
```

2. Clean out the data directory

```
$ rm -rf /var/lib/mongodb/*
```

3. Copy backup files

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

4. Grant permissions to data files for the mongod user

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

5. 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.



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

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 mongosh shell
> mongosh
# Initiate a new replica set
> rs.initiate()
```

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services, contact Percona Sales.

4.2.2 \$backupCursor and \$backupCursorExtend aggregation stages

\$backupCursor and \$backupCursorExtend aggregation stages expose the WiredTiger API which allows making consistent backups. Running these stages allows listing and freezing the files so you can copy them without the files being deleted or necessary parts within them being overwritten.

- \$backupCursor outputs the list of files and their size to copy.
- \$backupCursorExtend outputs the list of WiredTiger transaction log files that have been updated or newly added since the \$backupCursor was first run. Saving these files enables restoring the database to any arbitrary time between the \$backupCursor and \$backupCursorExtend execution times.

They are available in Percona Server for MongoDB starting with version 6.0.2-1.

Percona provides Percona Backup for MongoDB (PBM) – a light-weight open source solution for consistent backups and restores across sharded clusters. PBM relies on these aggregation stages for physical backups and restores. However, if you wish to develop your own backup application, this document describes the \$backupCursor and \$backupCursorExtend aggregation stages.

Usage

You can run these stages in any type of MongoDB deployment. If you need to back up a single node in a replica set, first run the \$backupCursor, then the \$backupCursorExtend and save the output files to the backup storage.

To make a consistent backup of a sharded cluster, run both aggregation stages on one node from each shard and the config server replica set. It can be either the primary or the secondary node. Note that since the secondary node may lag in syncing the data from the primary one, you will have to wait for the exact same time before running the \$backupCursorExtend.

Note that for standalone MongoDB node with disabled oplogs, you can only run the \$backupCursor aggregation stage.

GET A LIST OF ALL FILES TO COPY WITH \$BACKUPCURSOR

```
var bkCsr = db.getSiblingDB("admin").aggregate([{$backupCursor: {}}])
bkCsrMetadata = bkCsr.next().metadata
```

Sample output:

```
{
    metadata: {
        backupId: UUID("35c34101-0107-44cf-bdec-fad285e07534"),
        dbpath: '/var/lib/mongodb',
        oplogStart: { ts: Timestamp({ t: 1666631297, i: 1 }), t: Long("-1") },
        oplogEnd: { ts: Timestamp({ t: 1666631408, i: 1 }), t: Long("1") },
        checkpointTimestamp: Timestamp({ t: 1666631348, i: 1 })
    }
},
```

Store the metadata document somewhere, because you need to pass the backupId parameter from this document as the input parameter for the \$backupCursorExtend stage. Also you need the oplogEnd timestamp. Make sure that the \$backupCursor is complete on all shards in your cluster.

Note

Note that when running <code>\$backupCursor</code> in a standalone node deployment, <code>oplogStart</code>, <code>oplogEnd</code>, <code>checkpointTimesatmp</code> values may be absent. This is because standalone node deployments don't have oplogs.

RUN \$BACKUPCURSOREXTEND TO RETRIEVE THE WIREDTIGER TRANSACTION LOGS

Pass the backupId from the metadata document as the first parameter. For the timestamp parameter, use the maximum (latest) value among the oplogEnd timestamps from all shards and config server replica set. This will be the target time to restore.

```
var bkExtCsr = db.aggregate([{$backupCursorExtend: {backupId: bkCsrMetadata.backupId,
timestamp: new Timestamp(1666631418, 1)}}])
```

Sample output:

```
{ "filename" : "/data/plain_rs/n1/data/journal/WiredTigerLog.00000000042" }
{ "filename" : "/data/plain_rs/n1/data/journal/WiredTigerLog.00000000043" }
{ "filename" : "/data/plain_rs/n1/data/journal/WiredTigerLog.00000000044" }
```

LOOP THE \$BACKUPCURSOR

Prevent the backup cursor from closing on timeout (default – 10 minutes). This is crucial since it prevents overwriting backup snapshot file blocks with new ones if the files take longer than 10 minutes to copy. Use the getMore command for this purpose.

COPY THE FILES TO THE STORAGE

Now you can copy the output of both aggregation stages to your backup storage.

After the backup is copied to the storage, terminate the getMore command and close the cursor.



Note

Save the timestamp that you passed for the \$backupCursorExtend stage somewhere since you will need it for the restore.

This document is based on the blog post Experimental Feature: \$backupCursorExtend in Percona Server for MongoDB by Akira Kurogane

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services, contact Percona Sales.

4.3 Authentication

4.3.1 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 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
- AWS IAM authentication

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 Enable authentication.

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 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 certificates. For testing purposes, you can generate and use self-signed certificates.

x.509 authentication is compatible with 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

LDAP authentication with SASL

Overview

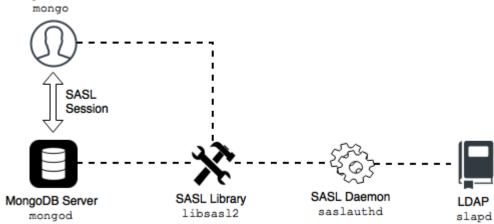
LDAP authentication with SASL means that both the client and the server establish a SASL session using the SASL library. Then authentication (bind) requests are sent to the LDAP server through the SASL authentication daemon (saslauthd) that acts as a remote proxy for the mongod server.

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:

MongoDB Client



An authentication session uses the following sequence:

- 1. A mongosh 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.

Kerberos authentication

Percona Server for MongoDB supports Kerberos authentication starting from release 6.0.2-1.

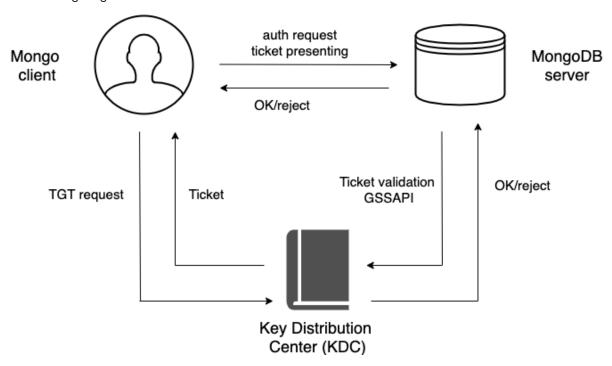
This authentication mechanism involves the use of a Key Distribution Center (KDC) - a symmetric encryption component which operates with tickets. A ticket is a small amount of encrypted data which is used for authentication. It is issued for a user session and has a limited lifetime.

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-Grantng 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 mongod server presenting the ticket.
- 4. The mongod 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.



[SCRAM]: Salted Challenge Response Authentication Mechanism [CA]: Certified Authority *[DN]: Distinguished Name

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services, contact Percona Sales.

4.3.2 Enable SCRAM 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 documents describes how to enable built-in SCRAM 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.

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.

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:

```
$ service mongod restart
```

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

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



MongoDB Documentation: Enable Access Control

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

4.3.3 Set 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.

Environment setup and configuration

The following components are required:

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

The following steps will help you configure your environment:

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, see 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.

CONFIGURING SASLAUTHD

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

```
Debian and Ubuntu

$ sudo apt install -y sasl2-bin

RHEL and derivatives

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

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

Note

Back up the original configuration file before making changes.

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/saslauthdsudo 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
```

RHEL and derivatives

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

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
```

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 0:0K "Success". If it doesn't, then either the user name and password are not in the LDAP service, or sasaluthd is not configured properly.

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 <code>mongodb.conf</code> and placed in a directory where <code>libsasl2</code> can find and read it. <code>libsasl2</code> 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.



SASL documentation

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.

```
$ mongosh --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"} ]} );
```

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
```

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

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services, contact Percona Sales.

4.3.4 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.

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.

Setup procedure

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 <code>-subj</code> 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/0=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/0=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 <u>subj</u> flag, specify the information about clients in the 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
```

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.

CONFIGURE THE 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.

Output:

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

3. Start the mongod service

```
$ sudo systemctl start mongod
```

AUTHENTICATE WITH THE X.509 CERTIFICATE

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

```
$ mongosh --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"
             ],
             "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
```

*[CA]: Certificate Authority

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

4.3.5 Setting up Kerberos authentication

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

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.

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 assign 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.

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.

Debian and Ubuntu

Edit the environment file at the path /etc/default/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.

RHEL and derivatives

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

```
{\tt 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
```

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

4.3.6 AWS IAM authentication



Version added: 6.0.5-4



Important

This is a technical preview feature

IAM (Identity Access Management) is the AWS service that allows you to securely control access to AWS resources. Percona Server for MongoDB supports authentication with AWS IAM enabling you to use the same AWS credentials both for it and other components of your infrastructure. This saves your DBAs from managing different sets of secrets and frees their time on other activities.

You can configure AWS IAM for a password-less authentication. Instead of username and password, the user or the application presents the AWS security credentials for authentication, but the secret key is not sent to Percona Server for MongoDB. This significantly increases the security in your infrastructure.

Percona Server for MongoDB supports two authentication types:

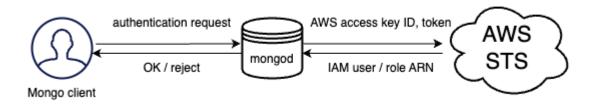
User authentication

This authentication type is typically used by human operators. Every user account in AWS has the ARN (Amazon Resource Name), which uniquely identifies this account and the user associated with it. During authentication, the ARN is used to verify the user's identity.

Role authentication

This type is typically used for applications / mongo clients. For instance, if your application is running on AWS resources like EC2 instance or ECS (Elastic Container Service) which uses the IAM role assigned to it. Another scenario is to allow users to assume the IAM role and in such a way, grant a user the permissions outlined in the IAM role. The ARN of the IAM role is used to authenticate the application in Percona Server for MongoDB.

For either type of AWS IAM authentication, the flow is the following:



- A mongo client (a Mongo shell or an application that talks to Percona Server for MongoDB via a driver) gets
 AWS credentials from either EC2/ECS instance metadata service, environmental variables or MongoDB URI
 connection string.
- 2. The mongo client constructs the authentication request which includes the AWS access key ID, token and the signature and sends it to Percona Server for MongoDB



Important

The mongo client never sends the secret access key to Percona Server for MongoDB.

- 3. Percona Server for MongoDB sends the received credentials to the AWS STS (Security Token Service) for verification
- 4. The AWS STS service validates whether the signature is correct and answers with the user / role ARN that created the signature
- 5. Percona Server for MongoDB looks for the same username as the received ARN in the \$external database and grants privileges to access Percona Server for MongoDB as defined for the respective user.



See also

- AWS documentation:
 - AWS Identity and Access Management
 - Temporary security credentials in IAM
 - Authenticating Requests (AWS Signature Version 4)
- MongoDB documentation: Set Up Passwordless Authentication with AWS IAM

Configuration

For how to configure AWS IAM authentication, see Setting up AWS IAM authentication.

[AWS]: Amazon Web Service [IAM]: Identity Access Management

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services, contact Percona Sales.

4.3.7 Setting up AWS IAM authentication

This document provides guidelines how to configure Percona Server for MongoDB to use AWS IAM authentication. The use of this authentication method enables you to natively integrate Percona Server for MongoDB with AWS services, increase security of your infrastructure by setting up password-less authentication and offload your DBAs from managing different sets of secrets. To learn more, see AWS IAM authentication

To configure AWS IAM authentication means to set up your AWS environment and configure Percona Server for MongoDB. The AWS environment setup is out of scope of this document. Consult the AWS documentation to perform the following setup steps:

- 1. Configure the AWS resource to work with IAM.
- 2. For user authentication:
 - Create the IAM user and copy its ARN (Amazon Resource Name)

For role authentication:

- · Create the IAM role
- Attach the IAM role to the AWS resource.
- Copy the ARN of the IAM role.

Configure Percona Server for MongoDB

The steps are the following:

- 1. Create users in the \$external database with the username as the IAM user/role ARN
- 2. Enable authentication and specify the authentication mechanism as MONGODB-AWS.

CREATE USERS IN \$EXTERNAL DATABASE

During the authentication, Percona Server for MongoDB matches the ARN of the IAM user or role retrieved from AWS STS against the user created in the \$external database. Thus, the username for this user must include their ARN and have the following format:

User authentication

arn:aws:iam::<ARN>:user/<user_name>

Role authentication

arn:aws:iam::<ARN>:role/<role_name>

Create a user and assign the required roles to them. Specify the ARN and names in the following example commands:

ENABLE AUTHENTICATION

Run the following commands as root or via sudo

1. Stop the mongod service

```
$ sudo systemctl stop mongod
```

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

```
security:
   authorization: enabled

setParameter:
   authenticationMechanisms: MONGODB-AWS
```

3. Start the mongod service

```
$ sudo systemctl start mongod
```

Authenticate in Percona Server for MongoDB using AWS IAM

To test the authentication, use either of the following methods:

MongoDB connection string

Replace <aws_access_key_id>, <aws_secret_access_key> and psmdb.example.com with actual values in the following command:

```
$ mongosh 'mongodb://<aws_access_key_id>:<aws_secret_access_key>:@psmdb.example.com/admin?
authSource=$external&authMechanism=MONGODB-AWS'
```

To pass temporary credentials and AWS token, replace <aws_access_key_id>, <aws_secret_access_key>, <aws_session token> and psmdb.example.com in the following command:

```
$ mongosh 'mongodb://<aws_access_key_id>:<aws_secret_access_key>:@psmdb.example.com/admin?
authSource=$external&authMechanism=MONGODB-
AWS&authMechanismProperties=AWS_SESSION_TOKEN:<aws_session_token>'
```

Environment variables

Set AWS environment variables:

```
export AWS_ACCESS_KEY_ID='<aws_access_key_id>'
export AWS_SECRET_ACCESS_KEY='<aws_secret_access_key>'
export AWS_SESSION_TOKEN='<aws_session_token>'
```

Connect to Percona Server for MongoDB:

```
$ mongosh 'mongodb://psmdb.example.com/testdb?authSource=$external&authMechanism=MONGODB-
AWS'
```

AWS resource metadata

If your application is running on the AWS resource, it receives the credentials from the resource metadata. To connect to Percona Server for MongoDB, run the command as follows:

```
$ mongosh --authenticationMechanism=MONGODB-AWS --authenticationDatabase='$external'
```

Upon successful authentication, the result should look like the following:

[AWS]: Amazon Web Service [IAM]: Identity Access Management

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services, contact Percona Sales.

4.3.8 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.

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

Authentication and authorization with direct binding to LDAP

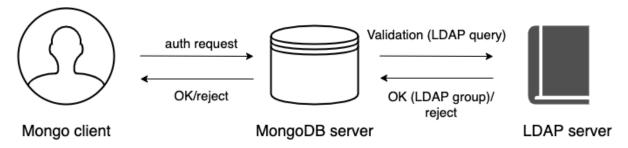
Starting with release 6.0.2-1, you can configure Percona Server for MongoDB to communicate with the LDAP server directly to authenticate and also authorize users.

The advantage of using this mechanism is that it is easy to setup and does not require pre-creating users in the dummy \$external database. 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 mongosh shell:

```
$ mongosh -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 database providing their credentials
- 2. If required, Percona Server for MongoDB transforms the username to match the user 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.

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 <code>ldapQuery</code> 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 6.0.2-1, 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 <code>ldapFollowReferrals</code> server parameter to <code>true</code> using the setParameter command or by editing the configuration file.

```
setParameter:

ldapFollowReferrals: true
```

CONNECTION POOL

As of version 6.0.2-1, 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 <code>ldapConnectionPoolSizePerHost</code> 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 6.0.2-1, 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 <code>ldapServers</code> dynamically at runtime using the <code>setParameter</code>.

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

1

See also

MongoDB Documentation:

- Authenticate and Authorize Users Using Active Directory via Native LDAP
- LDAP referrals

Configuration

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

*[DN]: Distinguished Name

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services, contact Percona Sales.

4.3.9 Set 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.

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.

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

 ${\tt objectClass:} \ {\tt groupOfNames}$

cn: otherusers

member: cn=bob,dc=percona,dc=com

Setup procedure

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 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 and derivatives

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

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 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:

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:

```
$ mongosh -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 , 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:

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:

```
$ mongosh -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 the userToDNMapping parameter:

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

Modify the given example configuration 6億の「Match your deployment, For example,comephaee023 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

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 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 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

1. Edit the /etc/mongod.conf 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"
```

2. Restart the mongod service:

```
$ sudo systemctl restart mongod
```

3. Test the access to Percona Server for MongoDB:

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

Username substitution

1. Edit the /etc/mongod.conf configuration file:

2. Restart the mongod service:

```
$ sudo systemctl restart mongod
```

3. Test the access to Percona Server for MongoDB:

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

LDAP query

1. Edit the /etc/mongod.conf configuration file:

Modify one of this example configuration to match your deployment.

This document is based on the following posts from Percona Database Performance Blog:

- 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

[DN]: Distinguished Name [AD]: Active Directory

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services, contact Percona Sales.

4.4 Encryption

4.4.1 Data at rest encryption

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 6.0.2-1, 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 for this purpose 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.

6

Important

You can only enable data at rest encryption and provide all encryption settings on an empty database, when you start the <code>mongod</code> 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.

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

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
encryptionKeyFile	The path to the encryption key file
encryptionCipherMode	The cipher mode for decryption. The supported values are AES256-CBC or AES256-GCM
inputPath	The path to the encrypted rollback file
outputPath	The path to save the decrypted rollback file

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services, contact Percona Sales.

4.4.2 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	Configuration file	Туре	Description
vaultServerName	security.vault.serverName	string	The IP address of the Vault server
vaultPort	security.vault.port	int	The port on the Vault server
vaultTokenFile	security.vault.tokenFile	string	The path to the vault token file. The token file is used by MongoDB 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: s.uTrHtzsZnEE7KyHeA797CkWA
vaultSecret	security.vault.secret	string	The path to the Vault secret. The Vault secret path format must be <secrets_engine_mount_path>/ data/<custom_path> where: - <secrets_engine_mount_path> is the path to the Key/Value Secrets Engine v2; - data is the mandatory path prefix required by Version 2 API; - <custom_path> is the path to the specific secret. Example: secret_v2/data/psmdb-test/ rs1-27017 Starting with version 6.0.5-4, a distinct Vault secret path for every replica set member is no longer mandatory. In earlier versions, it is recommended to use different secret paths for every database node in the entire deployment to avoid issues during the master key</custom_path></secrets_engine_mount_path></custom_path></secrets_engine_mount_path>
vaultSecretVersion	security.vault. secretVersion	unsigned long	rotation. (Optional) The version of the Vault secret to use
vaultRotateMasterKey	security.vault. rotateMasterKey	switch	When enabled, rotates the master key and exits
vaultServerCAFile	security.vault. serverCAFile	string	The path to the TLS certificate file
vaultDisableTLSForTesting	security.vault. disableTLSForTesting	switch	Disables secure connection to 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.

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>.

TARGETING A NAMESPACE IN VAULT CONFIGURATION

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
- Secure Multi-Tenancy with Namespaces

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

Starting with version 6.0.5-4, you can store the master key at the same path on every replica set member in your entire deployment. Vault assigns different versions to the master keys stored at the same path. The path and the version serve as the unique identifier of a master key. The mongod server stores that identifier and uses it to retrieve the correct master key from the Vault server during the restart.

In versions 6.0.4-3 and earlier, every mongod node in a replica set in your entire deployment must have a distinct path to the master keys on a Vault server.

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.

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services, contact Percona Sales.

4.4.3 Using the Key Management Interoperability Protocol (KMIP)

Percona Server for MongoDB adds support for 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 key management systems and the database server. KMIP provides the following benefits:

- Streamlines encryption key management
- Eliminates redundant key management processes

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.

Starting with version 6.0.2-1, the kmipKeyIdentifier option is no longer mandatory. When left blank, the database server creates a key on the KMIP server and uses that for encryption. When you specify the identifier, the key with such an ID must exist on the key storage.

KMIP parameters

Option	Туре	Description
kmipServerName	string	The hostname or IP address of the KMIP server. Multiple KMIP servers are supported as the comma-separated list, e.g. kmipl.example.com, kmip2.example.com
kmipPort	number	The port used to communicate with the KMIP server. When undefined, the default port 5696 will be used
kmipServerCAFile	string	The path to the CA certificate file. CA file is used to validate secure client connection to the KMIP server
kmipClientCertificateFile	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
kmipKeyIdentifier	string	Optional. The identifier of the KMIP key. If the key does not exist, the database server creates a key on the KMIP server with the specified identifier. When you specify the identifier, the key with such an ID must exist on the key storage. You can only use this setting for the first time you enable encryption
kmipRotateMasterKey	boolean	Controls master keys rotation. When enabled, generates the new master key version and re-encrypts the keystore. Requires the uniquekmipKeyIdentifier for every mongod node
 kmipClientCertificatePassword	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 4.2.21-21

Key rotation

Percona Server for MongoDB supports the master key rotation. This enables users to comply with data security regulations when using KMIP.

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>
        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> \
   --kmipKeyIdentifier <kmip_identifier>
```

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services, contact Percona Sales.

4.4.4 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.

All these options can be specified in the configuration file:

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

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services, contact Percona Sales.

4.4.5 Migrate from key file encryption to HashiCorp Vault encryption

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



This is a simple guideline and it should be used for testing purposes only. We recommend to contact 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.
- 3. Retrieve the key value from the keyfile

```
$ sudo cat /data/key/mongodb.key
d0JTFcePmvR0yLXwCbAH8fmiP/ZRm0nYbeJDMGaI7Zw=
```

4. 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.
```

5. 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
```

6. Start the mongod service

```
$ sudo systemctl start mongod
```

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

4.5 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
auditLog:
  destination: syslog
```



If you start the server with auditing enabled, you cannot disable auditing dynamically during runtime.

4.5.1 Audit options

The following options control audit logging:

Configuration file	Туре	Description
auditLog.destination	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 theauditPath option in a format specified by theauditFormat option syslog: Output audit events to syslog
auditLog.filter	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</expression1></field1>
auditLog.format	string	Specifies the format of the audit log file, if you set theauditDestination option to file. The default value is JSON. Alternatively, you can set it to BSON
auditLog.path	string	Specifies the fully qualified path to the file where audit log events are written, if you set theauditDestination 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
	auditLog.filter auditLog.format	auditLog.filter string auditLog.format string

4.5.2 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>
}
```

Parameter	Description
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)

4.5.3 Audit filter examples

The following examples demonstrate the flexibility of audit log filters.

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" ] } }'
```

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.*/ }'
```

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 }
```

4.5.4 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
```

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services, contact Percona Sales.

4.6 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.

4.6.1 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.

	sampleRate	rateLimit
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.

4.6.2 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 <code>mongod</code>, set the corresponding variables in the MongoDB configuration file (by default, <code>/etc/mongod.conf</code>):

```
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.

4.6.3 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.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services, contact Percona Sales.

4.7 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.



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

Log redaction is important for comlying 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 }
)
```

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services, contact Percona Sales.

4.8 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.

4.8.1 Usage

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

```
> 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:

```
> 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.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services, contact Percona Sales.

5. Manage PSMDB

5.1 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> } )
```

5.1.1 Parameters

cursorTimeoutMillis

Type: integer

Default: 600000 (ten minutes)

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

failIndexKeyTooLong

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.

internalQueryPlannerEnableIndexIntersection

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.

ttlMonitorEnabled

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.

ttlMonitorSleepSecs

Type: integer

Default: 60 (one minute)

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

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

5.2 Upgrade

5.2.1 Upgrading from Percona Server for MongoDB 5.0 to 6.0

To upgrade Percona Server for MongoDB to version 6.0, you must be running version 5.0. 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 6.0

We recommend to upgrade Percona Server for MongoDB from official Percona repositories using perconarelease repository management tool and the corresponding package manager for your system.

This document describes this method for the in-place upgrade (where your existing data and configuration files are preserved).

Warning

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

Upgrade on Debian and Ubuntu

1. Stop the mongod service:

```
$ sudo systemctl stop mongod
```

2. Enable Percona repository for Percona Server for MongoDB 6.0:

```
$ sudo percona-release enable psmdb-60
```

3. Update the local cache:

```
$ sudo apt update
```

4. Install Percona Server for MongoDB 6.0 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.

Upgrade on Red Hat Enterprise Linux and derivatives

1. Stop the mongod service:

```
$ sudo systemctl stop mongod
```

2. Enable Percona repository for Percona Server for MongoDB 6.0:

```
$ sudo percona-release enable psmdb-60
```

3. Install Percona Server for MongoDB 6.0 packages:

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

4. Start the mongod instance:

```
$ sudo systemctl start mongod
```

After the upgrade, Percona Server for MongoDB is started with the feature set of 5.0 version. Assuming that your applications are compatible with the new version, enable 6.0 version features. Run the following command against the admin database:

```
> db.adminCommand( { setFeatureCompatibilityVersion: "6.0" } )
```

1

See also

MongoDB Documentation:

- Upgrade a Standalone
- Upgrade a Replica Set
- Upgrade a Sharded Cluster

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services, contact Percona Sales.

5.2.2 Upgrade 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 6.0 Community Edition
- Minor upgrade of Percona Server for MongoDB

Upgrading from MongoDB 6.0 Community Edition



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.

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.

Upgrade on Debian and Ubuntu

1. Stop the mongod service:

```
$ sudo systemctl stop mongod
```

2. Check for installed packages:

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

Output:

ii mongodb-org	6.0.2	amd64	MongoDB
<pre>open source document-oriented ii mongodb-org-database</pre>	database system (metapackage) 6.0.2	amd64	MongoDB
open source document-oriented	****	alliu04	Mongobb
ii mongodb-org-database-tools	-extra 6.0.2	amd64	Extra
MongoDB database tools ii mongodb-org-mongos	6.0.2	amd64	MongoDB
sharded cluster query router	0.0.2	amao i	Hongozz
ii mongodb-org-server	6.0.2	amd64	MongoDB
database server ii mongodb-org-shell	6.0.2	amd64	MongoDB
shell client			-
ii mongodb-org-tools tools	6.0.2	amd64	MongoDB

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

- 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 start mongod
```

Upgrade on Red Hat Enterprise Linux and derivatives

1. Stop the mongod service:

```
$ sudo systemctl stop mongod
```

2. Check for installed packages:

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

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

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.
- 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.

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

Steps to upgrade from MongoDB 6.0 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 Switching storage engines for more information on migration of encrypted data. Contact us for working at the detailed migration steps, if further assistance is needed.

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

5.3 Uninstall 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 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.

Keep the configuration and data files

\$ sudo apt remove percona-server-mongodb*

Delete configuration and data files

\$ sudo apt purge percona-server-mongodb*

Uninstall on Red Hat Enterprise Linux and derivatives

1. Stop the mongod 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.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

6. Release notes

6.1 Percona Server for MongoDB 6.0 Release Notes

- Percona Server for MongoDB 6.0.5-4 (2023-03-29)
- Percona Server for MongoDB 6.0.4-3 (2023-01-30)
- Percona Server for MongoDB 6.0.3-2 (2022-12-07)
- Percona Server for MongoDB 6.0.2-1 (2022-10-31)

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services, contact Percona Sales.

6.2 Percona Server for MongoDB 6.0.5-4 (2023-03-29)

Release date	March 29, 2023
Installation	Installing Percona Server for MongoDB

Percona Server for MongoDB 6.0.5-4 is an enhanced, source-available, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 6.0.5 Community Edition.

It is rebased on MongoDB 6.0.5 Community edition and supports the upstream protocols and drivers.

6.2.1 Release Highlights

- Added support for authentication using AWS IAM enables you to natively integrate Percona Server for MongoDB with AWS services, increase security of your infrastructure by setting up password-less authentication and offload your DBAs from managing different sets of secrets. This is a technical preview feature
- Improved master key rotation for data at rest encrypted with HashiCorp Vault enables you to use the same secret key path on every server in your entire deployment thus significantly simplifying the secrets management and key rotation process.

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

- SERVER-61909 Fixed a hang when inserting or deleting a document with large number of index entries
- SERVER-66469 Fixed the issue with filtering time-series collections that contain the date values earlier than Unix epoch (1970)
- SERVER-68122 Fixed the issue with adding a new unencrypted node into an encrypted replica set by removing options which might not apply for this node.
- SERVER-73232 Changed the default log verbosity level for _killOperations to D2.
- SERVER-73266 Fixed deadlock that can occur during index creation
- SERVER-73009 Resolved the issue with the sort order on clustered collections where requested decreasing order returned returned results in increasing order
- SERVER-72512 Fixed the issue with indexes reported as valid while being inconsistent by improving the validation of those indexes
- SERVER-71219 Fixed the migration of distributed transactions by registering the migration source operation observer hook in all paths where transactions transition into the prepared state.

Find the full list of new features and improvements in MongoDB 6.0.5 Community edition release notes.

6.2.2 New Features

• PSMDB-1033: Add authentication with AWS IAM

6.2.3 Improvements

• PSMDB-1148: Improve the master key rotation when using a single master key for data-at-rest encryption with Vault in the entire deployment

6.2.4 Bugs Fixed

- PSMDB-1201: Improved the error message if the attempt to save an encryption key to a KMIP server failed
- PSMDB-1203: Gracefully terminate mongod if the master encryption key can't be saved to a KMIP server
- PSMDB-1204: Fixed the handling of attributes list for LDAP authentication with OpenLDAP during the user to DN mapping stage

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services, contact Percona Sales.

6.3 Percona Server for MongoDB 6.0.4-3 (2023-01-30)

Release date	January 30, 2023
Installation	Installing Percona Backup for MongoDB

Percona Server for MongoDB 6.0.4-3 is an enhanced, source-available, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 6.0.4 Community Edition.

It is rebased on MongoDB 6.0.4 Community edition and supports the upstream protocols and drivers.

6.3.1 Release Highlights

- Percona Server for MongoDB is now available on Red Hat Enterprise Linux 9 and compatible derivatives
- A Docker image for Percona Server for MongoDB (Release candidate) is now available for ARM64 architectures. The support of ARM64 will be extended in subsequent releases.

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

- SERVER-72416 Fixed the issue with incorrect projection parsing when a collection level collation is specified
- SERVER-71759 Changed the yielding policy of dataSize command to YIELD_AUTO for both when the command is called with estimate:true or false
- SERVER-70237 Fixed the issue with a BSON object exceeding the max allowed size during chunks merge in a shard
- SERVER-72222 Fixed the incorrect behavior of the mapReduce command with single reduce optimization in sharded clusters

Find the full list of new features and improvements in MongoDB 6.0.4 Community edition release notes.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services, contact Percona Sales.

6.4 Percona Server for MongoDB 6.0.3-2 (2022-12-07)

Release date	December 7, 2022
Installation	Installing Percona Backup for MongoDB

Percona Server for MongoDB 6.0.3-2 is an enhanced, source-available, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 6.0.3 Community edition.

Percona Server for MongoDB 6.0.3-2 fully supports MongoDB 6.0 protocols and drivers and does not require any code modifications.

6.4.1 Release Highlights

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

- SERVER-66289 Fixed the issue with how the server handles batches of writes when running \$out with secondary read preference by updating write size estimation logic in <code>DocumentSourceWriter</code>
- SERVER-68371 Allowed search queries to pass through query analysis when Client-Side Field Level Encryption is enabled for the MongoClient
- SERVER-68115 Prevented dropping empty path component from elemMatch path during index selection
- SERVER-68394 Prevented yielding strong locks upon startup recovery when <code>_id</code> index is missing

Find the full list of new features and improvements in MongoDB 6.0.3 Community edition release notes.

6.4.2 Improvements

• PSMDB-1181: Add backup cursor parameters to cursor's metadata

6.4.3 Bugs Fixed

• PSMDB-1175: Fixed Percona Server for MongoDB behavior when calling \$backupCursor with disableIncrementalBackup option

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services, contact Percona Sales.

6.5 Percona Server for MongoDB 6.0.2-1 (2022-10-31)

Release date	October 31, 2022
Installation	Installing Percona Backup for MongoDB

We are pleased to announce the availability of Percona Server for MongoDB (PSMDB) 6.0.2-1 – the new major version of the source available, drop-in replacement of MongoDB 6.0 Community edition. It is available for download from Percona website and for installation from Percona Software Repositories.

Percona Server for MongoDB 6.0.2-1 fully supports MongoDB 6.0 protocols and drivers and does not require any code modifications.

6.5.1 Release Highlights

- Data-at-rest encryption using the Key Management Interoperability Protocol (KMIP) is generally available enabling you to use it in your production environment
- \$backupCursor and \$backupCursorExtend aggregation stages functionality is generally available, enabling your application developers to use it for building custom backup solutions.



Percona provides Percona Backup for MongoDB - the open source tool for consistent backups and restores in MongoDB sharded clusters.

 Percona Server for MongoDB packages now include mongosh of the previous mongo shell. The previous legacy mongo shell was deprecated in MongoDB 5.0 and has been removed in version 6.0. Some older methods are unavailable or have been replaced with newer ones for the new mongosh. Please review any methods for compatibility.

If you install Percona Server for MongoDB from tarballs, you must install mongosh from a separate tarball.

Percona Server for MongoDB 6.0.2-1 includes all the features of MongoDB 6.0.2 Community Edition, among which are the following:

- Enhanced time series collections enable you to:
 - Get deeper data analysis insights by building compound and secondary indexes on time, metadata and measurement fields.
 - Distribute the load among nodes in the cluster by sharding new and existing time series collections.
 - Benefit from faster reads and improved performance by applying the sorting on the most recent entry instead of the whole collection.
- The following change streams optimizations help you enhance your event-driven solutions:
 - Improve in-app notifications, reference deleted documents or feed the updated version of the entire doc to the downstream system using the before and after states of a document that was changed.
 - React not only to data changes but also to database change events like creating or dropping of collections with the Data Definition Language (DDL) support.
- New aggregation stages like \$densify, \$documents, \$fill and operators like \$bottom, \$firstN, \$lastN, \$maxN / \$minN and others enable you to off load work from your developers to the database. These operators allow automating key commands, getting required data insights by combining individual operators into aggregation pipelines. As a result, your developers spend less time on writing complex code or manipulating data manually and can focus on other activities.
- Cluster-wide configuration parameters and commands save your DBAs' time on cluster administration.
- The Stable API (formerly known as versioned API) features the extended set of new database commands and aggregation operators which enables you to improve communication of your apps and MongoDB.
- Speed up data processing and save on storage costs with clustered collections. Clustered collections don't require secondary indexes and thus, result in faster queries. A single read/write for the index and the document improves performance for inserts, updates, deletes and queries. With less storage space required by clustered connections, bulk updates and inserts are performed faster. And by turning clustered indexes to TTL indexes with a single field, you benefit from simplified delete operations and reduced storage costs.

Percona Server for MongoDB also includes the following bug fixes and enhancements provided upstream:

- SERVER-68511 Fixed the issue that caused inconsistency in sharding metadata when running the movePrimary command on the database that has the Feature Compatibility Version (FCV) set to 4.4 or earlier. Affects MongoDB versions 5.0.0 through 5.0.10 and MongoDB 6.0.0. Upgrade to the fixed version of MongoDB 6.0.2 / Percona Server for MongoDB 6.0.2-1 as soon as possible.
- SERVER-66072 Fixed dependency analysis for <code>\$match</code> aggregation stage with aggregation expressions with the <code>\$rand</code> operator
- SERVER-68130 Fixed the AutoSplitVector's behavior to predict the BSON object size when generating the response
- WT-9870 Fixed the global time window state before performing the rollback to stable operation by updating the pinned timestamp as part of the transaction setup.
- SERVER-68628 Fixed the issue when retrying a failed resharding operation after a primary failover could lead to server crash or lost writes.
- SERVER-68925 Detect and resolve table logging inconsistencies for WiredTiger tables at startup

Find the full list of new features and improvements in MongoDB 6.0 Community Edition release notes.

Percona Server for MongoDB 6.0.2-1 extends this feature set by providing enterprise-level enhancements for free.

To upgrade to PSMDB, see our upgrade instructions.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

7. Glossary

7.1 ACID

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

7.2 Atomicity

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

7.3 Consistency

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

7.4 Durability

Once a transaction is committed, it will remain so.

7.5 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.

7.6 Isolation

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

7.7 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).

7.8 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.

7.9 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.# Glossary

7.10 Technical preview feature

Technical preview features are not yet ready for enterprise use and are not included in support via SLA. They are included in this release so that users can provide feedback on their experience with the feature prior to its full release in a future GA release (or removal of the feature if it is deemed not useful). This functionality can change (APIs, CLIs, etc.) from tech preview to GA.

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services, contact Percona Sales.

8. Copyright and licensing information

8.1 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.

8.2 Software license

Percona Server for MongoDB is source-available software.

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services, contact Percona Sales.

9. 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 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 trademarks@percona.com for assistance and we will do our very best to be helpful.

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services, contact Percona Sales.