PERCONA

www.percona.com

Percona Server for MongoDB 3.6 Documentation

Release 3.6.23-13.0

Mar 30, 2021

CONTENTS

Percona Server for MongoDB is a free, enhanced, fully compatible, open source, drop-in replacement for MongoDB 3.6 Community Edition with enterprise-grade features. It requires no changes to MongoDB applications or code.

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

See also:

MongoDB Documentation: buildInfo Database Command https://docs.mongodb.com/manual/reference/command/buildInfo/#dbcmd.buildInfo

---

Percona Server for MongoDB provides the following features:

- MongoDB's original MMAPv1 storage engine, and the default WiredTiger engine
- Optional *Percona Memory Engine* and *MongoRocks* storage engines
- *External SASL authentication* using OpenLDAP or Active Directory
- *Audit logging* to track and query database interactions of users or applications
- *Hot Backup* for the default WiredTiger and alternative *MongoRocks* storage engine
- *Profiling Rate Limit* to decrease the impact of the profiler on performance

---

Part I

Introduction

CHAPTER

ONE

PERCONA SERVER FOR MONGODB FEATURE COMPARISON

Percona Server for MongoDB 3.6 is based on MongoDB 3.6. *Percona Server for MongoDB* extends MongoDB 3.6 Community Edition to include the functionality that is otherwise only available in MongoDB 3.6 Enterprise.

	PSMDB	MongoDB
Storage Engines	• WiredTiger (default) • *Percona Memory Engine*	• WiredTiger (default) • In-Memory (Enterprise only)
Encryption-at-Rest	• Key server = Hashicorp Vault • Fully opensource	• Key server = KMIP • Enterprise only
Hot Backup	YES (replicaset)	NO
LDAP Authentication	• Simple LDAP Auth • (legacy) *External SASL Authentication*	• Enterprise only • Enterprise only
LDAP Authorization	YES	Enterprise only
Kerberos Authentication	YES	Enterprise only
Audit Logging	YES	Enterprise only
Log redaction	YES	Enterprise only
SNMP Monitoring	NO	Enterprise only

Profiling Rate Limiting

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

Part II

Installation

CHAPTER

TWO

INSTALLING PERCONA SERVER FOR MONGODB

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

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

- *Install on Debian or Ubuntu*
- *Install on RHEL or CentOS*

Alternative Install Instructions

You can also download packages from the Percona website and install them manually using `dpkg` or `rpm`.

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

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

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

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

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 3.4*.

Installing Percona Server for MongoDB on Debian and Ubuntu

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

Package Contents

percona-server-mongodb-36 Installs the `mongo` shell, import/export tools, other client utilities, server software, default configuration, and init.d scripts.

percona-server-mongodb-36-server Contains the `mongod` server, default configuration files, and init.d scripts.

percona-server-mongodb-36-shell Contains the `mongo` shell.

percona-server-mongodb-36-mongos Contains the `mongos` sharded cluster query router.

percona-server-mongodb-36-tools Contains Mongo tools for high-performance MongoDB fork from Percona.

percona-server-mongodb-36-dbg Contains debug symbols for the server.

Installing from Percona Repositories

It is recommended to install *Percona Server for MongoDB* from official Percona repositories.

Configure Percona repositories as described in Percona Software Repositories Documentation.

Install the latest version

Starting from *Percona Server for MongoDB* 3.6.19-7.0, the packages are located in the `psmdb-36` repository. To install the latest version of *Percona Server for MongoDB*, do the following:

1. Enable the repository.

```
$ sudo percona-release enable psmdb-36
```

2. Install the required *Percona Server for MongoDB* package using `apt-get`. For example, to install the full package, run the following:

```
$ sudo apt-get install percona-server-mongodb-36
```

Install a specific version

Percona Server for MongoDB versions earlier than 3.6.19-7.0 are available for download on the Percona website.

1. Select the desired version and your operating system from the drop-down menus.

2. Download the package bundle. The permalink for download is generated for you based on your selections in the format `https://www.percona.com/downloads/percona-server-mongodb-3.6/<release_version>/binary/<OS>/<OS_version>/x86_64/<package_name>`.

   For example, to download *Percona Server for MongoDB* 3.6.14-3.4 for Debian Buster, use the following URL:

```
$ wget https://www.percona.com/downloads/percona-server-mongodb-3.6/percona-
↪server-mongodb-3.6.14-3.4/binary/debian/buster/x86_64/percona-server-mongodb-3.
↪6.14-3.4-r0985495-buster-x86_64-bundle.tar
```

3. Unpack the archive

```
$ tar xfv percona-server-mongodb-3.6.14-3.4-r0985495-buster-x86_64-bundle.tar
```

4. Install the packages

```
$ sudo dpkg -i percona-server-mongodb-36_3.6.14-3.4.buster_amd64.deb \
  percona-server-mongodb-36-dbg_3.6.14-3.4.buster_amd64.deb \
  percona-server-mongodb-36-mongos_3.6.14-3.4.buster_amd64.deb \
  percona-server-mongodb-36-server_3.6.14-3.4.buster_amd64.deb \
  percona-server-mongodb-36-shell_3.6.14-3.4.buster_amd64.deb \
  percona-server-mongodb-36-tools_3.6.14-3.4.buster_amd64.deb
```

Note: If `dpkg` fails at this step due to missing dependencies, run `apt` as follows:

```
$ sudo apt-get install --fix-broken
```

Using Percona Server for MongoDB

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

- **Starting the service**

  Percona Server for MongoDB is started automatically after installation unless it encounters errors during the installation process. You can also manually start it using the following command:

```
$ sudo systemctl start mongod
```

- **Confirming that the service is running**

  Check the service status using the following command:

```
$ sudo systemctl status mongod
```

- **Stopping the service**

  Stop the service using the following command:

```
$ sudo systemctl stop mongod
```

- **Restarting the service**

  Restart the service using the following command:

```
$ sudo systemctl restart mongod
```

Uninstalling Percona Server for MongoDB

To uninstall *Percona Server for MongoDB*, remove all the installed packages. Removing packages with `apt-get remove` will leave the configuration and data files. Removing the packages with `apt-get purge` will remove all the packages with configuration files and data. Depending on your needs you can choose which command better suits you.

1. Stop the server:

```
$ sudo systemctl stop mongod
```

2. Remove the packages.

   - If you want to leave configuration and data files:

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

   - If you want to delete configuration and data files as well as the packages:

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

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

Note: *Percona Server for MongoDB* should work on other RPM-based distributions (for example, Amazon Linux AMI and Oracle Linux), but it is tested only on platforms listed on the Percona Software and Platform Lifecycle page.[1].

[1] We support only the current stable RHEL 6 and CentOS 6 releases, because there is no official (i.e. RedHat provided) method to support or download the latest OpenSSL on RHEL and CentOS versions prior to 6.5. Similarly, and also as a result thereof, there is no official Percona way to support the latest Percona Server builds on RHEL and CentOS versions prior to 6.5. Additionally, many users will need to upgrade to OpenSSL 1.0.1g or later (due to the Heartbleed vulnerability), and this OpenSSL version is not available for download from any official RHEL and CentOS repositories for versions 6.4 and prior. For any officially unsupported system, `src.rpm` packages can be used to rebuild Percona Server for any environment. Please contact our support service if you require further information on this.

Package Contents

Percona-Server-MongoDB-36 Installs the `mongo` shell, import/export tools, other client utilities, server software, default configuration, and init.d scripts.

Percona-Server-MongoDB-36-server Contains the `mongod` server, default configuration files, and init.d scripts.

Percona-Server-MongoDB-36-shell Contains the `mongo` shell.

Percona-Server-MongoDB-36-mongos Contains the `mongos` sharded cluster query router.

Percona-Server-MongoDB-36-tools Contains Mongo tools for high-performance MongoDB fork from Percona.

Percona-Server-MongoDB-36-debuginfo Contains debug symbols for the server.

Installing from Percona Repositories

It is recommended to install *Percona Server for MongoDB* from official Percona repositories

Configure Percona repositories as described in Percona Software Repositories Documentation.

Install the latest version

Starting from *Percona Server for MongoDB* 3.6.19-7.0, the packages are located in the `psmdb-36` repository. To install the latest version of *Percona Server for MongoDB*, do the following:

1. Enable the repository.

```
$ sudo percona-release enable psmdb-36
```

2. Install the required *Percona Server for MongoDB* package using `yum`. For example, to install the full package, run the following:

```
$ sudo yum install Percona-Server-MongoDB-36
```

Install a specific version

Earlier versions of *Percona Server for MongoDB* are located in the `original` repository. To install a specific version:

1. Enable the *original* repository:

```
$ sudo percona-release enable original
```

2. List available packages:

```
$ sudo yum list Percona-Server-MongoDB-36 --showduplicates
```

Sample output

```
     Available Packages
Percona-Server-MongoDB-36.x86_64    3.6.17-4.0.el8    percona-release-x86_64
Percona-Server-MongoDB-36.x86_64    3.6.18-5.0.el8    percona-release-x86_64
```

```
Percona-Server-MongoDB-36.x86_64    3.6.18-6.0.el8    percona-release-x86_64
Percona-Server-MongoDB-36.x86_64    3.6.19-7.0.el8    percona-release-x86_64
```

3. Install a specific version packages. For example, to install *Percona Server for MongoDB* 3.6.17-4.0, use the following command:

```
$ sudo yum install Percona-Server-MongoDB-36-3.6.17-4.0.el8
```

Using Percona Server for MongoDB

Warning: If you have SELinux security module installed, it will conflict with Percona Server for MongoDB. There are several options to deal with this:

- Remove the SELinux packages. This is not recommended, because it may violate security.
- Disable SELinux by setting `SELINUX` in `/etc/selinux/config` to `disabled`. This change takes effect after you reboot.
- Run SELinux in permissive mode by setting `SELINUX` in `/etc/selinux/config` to `permissive`. This change takes effect after you reboot.

  You can also enforce permissive mode at runtime using the `setenforce 0` command. However, this will not affect the configuration after a reboot.

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

- **Starting the service**

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

```
$ sudo systemctl start mongod
```

- **Confirming that service is running**

  Check the service status using the following command:

```
$ sudo systemctl status mongod
```

- **Stopping the service**

  Stop the service using the following command:

```
$ sudo systemctl stop mongod
```

- **Restarting the service**

  Restart the service using the following command:

```
$ sudo systemctl restart mongod
```

Running after reboot

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

For RHEL or CentOS versions 5 and 6, you can use the `chkconfig` utility to enable auto-start as follows:

```
$ chkconfig --add mongod
```

For RHEL or CentOS version 7, you can use the `systemctl` utility as follows:

```
$ systemctl enable mongod
```

Uninstalling Percona Server for MongoDB

To completely uninstall Percona Server for MongoDB you'll need to remove all the installed packages and data files:

1. Stop the Percona Server for MongoDB service:

   ```
   $ sudo systemctl stop mongod
   ```

2. Remove the packages:

   ```
   $ sudo yum remove Percona-Server-MongoDB*
   ```

3. Remove the data and configuration files:

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

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.

Installing Percona Server for MongoDB from Binary Tarball

You can find the link to the binary tarball under the *Generic Linux* menu item on the Percona Server for MongoDB download page.

There are two tarballs available:

- `percona-server-mongodb-3.6.19-8.0-x86_64.glibc2.17.tar.gz` is the general tarball, compatible with any operation system but for Centos 6.
- `percona-server-mongodb-3.6.19-8.0-x86_64.glibc2.12.tar.gz` is the tarball for Centos 6.

1. Fetch and extract the correct binary tarball. For example, if you are running Debian 10 ("buster"):

   ```
   $ wget https://www.percona.com/downloads/percona-server-mongodb-3.6/percona-
   ↪server-mongodb-3.6.19-8.0/binary/tarball/percona-server-mongodb-3.6.19-8.0-x86_
   ↪64.glibc2.17.tar.gz\
   $ tar -xf percona-server-mongodb-3.6.19-8.0-x86_64.glibc2.17.tar.gz
   ```

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

```
export PATH=~/percona-server-mongodb-3.6.19-8.0/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`.

Running *Percona Server for MongoDB* in a Docker Container

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

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

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

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

To run the latest *Percona Server for MongoDB* 3.6 in a Docker container, use the following command:

```
docker run -d \
 --name psmdb \
 --restart always \
 percona/percona-server-mongodb:3.6
```

The previous command does the following:

- The `docker run` command instructs the `docker` daemon to run a container from an image.
- The `-d` option starts the container in detached mode (that is, in the background).
- The `--name` option assigns a custom name for the container that you can use to reference the container within a Docker network. In this case: `psmdb`.
- The `--restart` option defines the container's restart policy. Setting it to `always` ensures that the Docker daemon will start the container on startup and restart it if the container exits.
- `percona/percona-server-mongodb:3.6` is the name and version tag of the image to derive the container from.

  For full list of tags, see https://hub.docker.com/r/percona/percona-server-mongodb/tags/

Connecting from Another Docker Container

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

Connecting with the Mongo Shell

To start another container with the `mongo` shell that connects to your *Percona Server for MongoDB* container, run the following comand:

```
docker run -it --link psmdb --rm percona/percona-server-mongodb:mongo mongo -h psmdb
```

CHAPTER

THREE

UPGRADING FROM MONGODB 3.6 COMMUNITY EDITION TO *PERCONA SERVER FOR MONGODB*

An in-place upgrade is done with existing data in the server. Generally speaking, this is stopping the `mongod` service, removing the old packages, installing the new server and starting it with the same db data directory. An in-place upgrade is suitable for most environments, except the ones that use ephemeral storage and/or host addresses.

Note: MongoDB creates a user that belongs to two groups, which is a potential security risk. This is fixed in *Percona Server for MongoDB*: 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 document 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.

Upgrading on Debian or Ubuntu

1. Stop the `mongod` service:

```
$ sudo systemctl stop mongod
```

2. Check for installed packages:

```
$ dpkg -l | grep mongod

ii  mongodb-org                    3.6.2                                amd64        ␣
↪ MongoDB open source document-oriented database system (metapackage)
ii  mongodb-org-mongos             3.6.2                                amd64        ␣
↪ MongoDB sharded cluster query router
ii  mongodb-org-server             3.6.2                                amd64        ␣
↪ MongoDB database server
ii  mongodb-org-shell              3.6.2                                amd64        ␣
↪ MongoDB shell client
ii  mongodb-org-tools              3.6.2                                amd64        ␣
↪ MongoDB tools
```

3. Remove installed packages:

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

4. Remove log files:

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

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

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

7. Start the `mongod` service:

```
$ sudo systemctl start mongod
```

Upgrading on Red Hat Enterprise Linux or CentOS

1. Stop the `mongod` service:

```
$ sudo systemctl stop mongod
```

2. Check for installed packages:

```
$ rpm -qa | grep mongo

mongodb-org-mongos-3.6.2-1.el6.x86_64
```

```
mongodb-org-shell-3.6.2-1.el6.x86_64
mongodb-org-server-3.6.2-1.el6.x86_64
mongodb-org-tools-3.6.2-1.el6.x86_64
mongodb-org-3.6.2-1.el6.x86_64
```

3. Remove the installed packages:

```
$ sudo yum remove \
mongodb-org-mongos-3.6.2-1.el6.x86_64 mongodb-org-shell-3.6.2-1.el6.x86_64 \
mongodb-org-server-3.6.2-1.el6.x86_64 mongodb-org-tools-3.6.2-1.el6.x86_64 \
mongodb-org-3.6.2-1.el6.x86_64
```

4. Remove log files:

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

5. Install Percona Server for MongoDB *using yum*.
6. Start the `mongod` service:

```
$ sudo systemctl start mongod
```

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

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

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

Steps to upgrade from MongoDB 3.6 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.

CHAPTER

FOUR

UPGRADING FROM *PERCONA SERVER FOR MONGODB* 3.4 TO 3.6

Important: MongoRocks is deprecated in Percona Server for MongoDB 3.6. If you are using *Percona Server for MongoDB* 3.4 with MongoRocks, and you wish to upgrade to version 3.6, please read the note at the top of the MongoRocks page before upgrading: https://www.percona.com/doc/percona-server-for-mongodb/3.6/mongorocks.html

To upgrade *Percona Server for MongoDB* to version 3.6, you must be running version 3.4. Upgrades from earlier versions are not supported.

Before upgrading your production *Percona Server for MongoDB* deployments, test all your applications in a testing environment to make sure they are compatible with the new version.

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

1. Stop the `mongod` instance.
2. Remove old packages.
3. Install new packages.
4. Start the `mongod` instance.

It is recommended to upgrade *Percona Server for MongoDB* from official Percona repositories using the corresponding package manager for your system. For more information, see *Installing Percona Server for MongoDB*.

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

Upgrading on Debian or Ubuntu

1. Stop the `mongod` instance:

```
sudo service mongod stop
```

2. Remove *Percona Server for MongoDB* 3.4 packages:

```
sudo apt-get remove percona-server-mongodb-34*
```

3. Install *Percona Server for MongoDB* 3.6 packages:

```
sudo apt-get install percona-server-mongodb-36
```

4. If you modified the configuration file and wish to use it with the new version, verify that the `/etc/mongod.conf` file includes the correct options.
5. Start the `mongod` instance:

```
sudo service mongod start
```

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

Upgrading on RHEL and CentOS

1. Stop the `mongod` instance:

```
sudo service mongod stop
```

2. Remove *Percona Server for MongoDB* 3.4 packages:

```
sudo yum remove Percona-Server-MongoDB-34*
```

3. Install *Percona Server for MongoDB* 3.6 packages:

```
sudo yum install Percona-Server-MongoDB-36
```

4. Start the `mongod` instance:

```
sudo service mongod start
```

Note: When you remove old packages on Centos / RHEL, your modified configuration file is placed to `/etc/mongod.conf.rpmsave`. To use your configuration with the new version, do the following before you start the `mongod` service:

- Replace the default `/etc/mongod.conf` file,
- Check the permissions for the `mongod` user to custom paths for database and/or log files.
- Restore the permissions, if needed:

```
chown -R mongod:mongod <dpPathDir> <logDir>
```

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

Part III

Features

CHAPTER

FIVE

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 standard MongoDB engines (the original MMAPv1 and the default WiredTiger), as well as MongoRocks.

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`):

See also:

***MongoDB* Documentation: Configuration File Options**

- storage.engine Options
- storage.wiredTiger Options
- storage.inmemory Options

Data files created by one storage engine are not compatible with other storage 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 using the *`--inMemoryStatisticsLogDelaySecs`* option.

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 permissons 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 replicasets, 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.
  - for a standalone instance or a single-node replicaset, use the `mongodump` and `mongorestore` utilities:

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

Configuring Percona Memory Engine

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

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

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

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

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

option `--inMemorySizeGB`

Config `storage.inMemory.engineConfig.inMemorySizeGB`

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

Specifies the maximum memory in gigabytes to use for data.

option `--inMemoryStatisticsLogDelaySecs`

Config `storage.inMemory.engineConfig.statisticsLogDelaySecs`

Default 0

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

CHAPTER

SIX

MONGOROCKS

Important: MongoRocks is deprecated in Percona Server for MongoDB 3.6.

MongoRocks is deprecated in *Percona Server for MongoDB* 3.6 and it will be fully removed in the next major version of *Percona Server for MongoDB*. Feature compatibility version is set to 3.4 when using *Percona Server for MongoDB* 3.6 with MongoRocks, so 3.6 features, such as retryable writes and causal consistency, cannot be used. Additionally, read concern majority may produce unreliable results.

If you are using MongoRocks with *Percona Server for MongoDB* 3.4 or older, we strongly encourage you to migrate from MongoRocks to WiredTiger before upgrading to *Percona Server for MongoDB* 3.6. Instructions on how to change storage engines are located in this Percona blog post: https://www.percona.com/blog/2017/03/07/how-to-change-mongodb-storage-engines-without-downtime/.

If you install *Percona Server for MongoDB* 3.6 with the RocksDB storage engine, you will receive the following error message when trying to start mongod:

```
[ERROR] There are known issues with MongoDB 3.6 and MongoRocks. To learn about these␣
↪issues and how to enable MongoRocks with Percona Server for MongoDB 3.6, please␣
↪read https://www.percona.com/doc/percona-server-for-mongodb/3.6/mongorocks.html,␣
↪terminating
```

Note: changing feature compatibility version to 3.6 when using MongoRocks will produce the following error: `storage engine does not support upgrading featureCompatibilityVersion to 3.6`

To continue using *Percona Server for MongoDB* 3.6 with MongoRocks, use one of the following two methods:

Add `--useDeprecatedMongoRocks` to `mongod` startup options Update the config file with the following parameter:

```
storage:
engine: rocksdb
useDeprecatedMongoRocks: true
```

MongoRocks is a storage engine for MongoDB based on the RocksDB key-value store optimized for fast storage. It is developed by Facebook and designed to handle write-intensive workloads.

The *MongoRocks* storage engine is available in *Percona Server for MongoDB* along with the standard MongoDB engines (the original MMAPv1 and the default WiredTiger), as well as *Percona Memory Engine*.

Using MongoRocks

As of version 3.2, *Percona Server for MongoDB* runs with WiredTiger by default. If you still would like to use the deprecated *MongoRocks* storage engine, please use the `--storageEngine rocksdb` command-line option accompanied by `--useDeprecatedMongoRocks` when you start `mongod`. Alternatively, you can set the `storage.engine` and `useDeprecatedMongoRocks` variables in the configuration file (by default, `/etc/mongod.conf`) as shown below.

Data created by one storage engine is not compatible with other storage engines, because each one has its own data model. When changing the storage engine, you have to do one of the following:

- If you simply want to temporarily test *MongoRocks*, change to a different data directory with the `--dbpath` command-line option:

```
$ service mongod stop
$ mongod --storageEngine rocksdb --useDeprecatedMongoRocks --dbpath <newDataDir>
```

Note: Make sure that the user running `mongod` has read and write permissions for the new data directory.

- If you want to permanently switch to *MongoRocks* and do not have any valuable data in your database, clean out the default data directory and edit the configuration file:

```
$ service mongod stop
$ rm -rf /var/lib/mongodb/*
$ sed -i '/engine: .*rocksdb/s/#//g' /etc/mongod.conf
$ service mongod start
```

- If there is data that you want to migrate and make compatible with *MongoRocks*, use the `mongodump` and `mongorestore` utilities:

```
$ mongodump --out <dumpDir>
$ service mongod stop
$ rm -rf /var/lib/mongodb/*
$ sed -i '/engine: .*rocksdb/s/#//g' /etc/mongod.conf
$ service mongod start
$ mongorestore <dumpDir>
```

Configuring MongoRocks

You can configure *MongoRocks* using either command-line options or corresponding parameters in the `/etc/mongod.conf` file. The configuration file is formatted in YAML. For example, the following sample configuration is suggested as the default for running *Percona Server for MongoDB* with *MongoRocks*:

```
storage:
  engine: rocksdb
  useDeprecatedMongoRocks: true
  rocksdb:
    cacheSizeGB: 1
    compression: snappy
```

```
    maxWriteMBPerSec: 1024
    crashSafeCounters: false
    counters: true
    singleDeleteIndex: false
```

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

```
mongod --storageEngine=rocksdb \
  --useDeprecatedMongoRocks \
  --rocksdbCacheSizeGB=1 \
  --rocksdbCompression=snappy \
  --rocksdbMaxWriteMBPerSec=1024 \
  --rocksdbCrashSafeCounters=false \
  --rocksdbCounters=true \
  --rocksdbSingleDeleteIndex=false
```

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

option `--rocksdbCacheSizeGB`

Variable `storage.rocksdb.cacheSizeGB`

Type Integer

Default 30% of physical memory

Specifies the amount of memory (in gigabytes) to allocate for block cache. Block cache is used to store uncompressed pages. Compressed pages are stored in kernel's page cache.

To configure block cache size dynamically, set the `rocksdbRuntimeConfigCacheSizeGB` parameter at runtime:

```
db.adminCommand({setParameter:1, rocksdbRuntimeConfigCacheSizeGB: 10})
```

option `--rocksdbCompression`

Variable `storage.rocksdb.compression`

Type String

Default `snappy`

Specifies the block compression algorithm for data collection. Possible values: `none`, `snappy`, `zlib`, `lz4`, `lz4hc`.

option `--rocksdbMaxWriteMBPerSec`

Variable `storage.rocksdb.maxWriteMBPerSec`

Type Integer

Default `1024` (1 GB/sec)

Specifies the maximum speed at which *MongoRocks* writes to storage (in megabytes per second). Decrease this value to reduce read latency spikes during compactions. However, reducing it too much might slow down writes.

To configure write speed dynamically, set the `rocksdbRuntimeConfigMaxWriteMBPerSec` parameter at runtime:

```
db.adminCommand({setParameter:1, rocksdbRuntimeConfigMaxWriteMBPerSec:30})
```

option `--rocksdbCrashSafeCounters`

Variable `storage.rocksdb.crashSafeCounters`

Type Boolean

Default `false`

Specifies whether to correct counters after a crash. Enabling this can affect write performance.

option `--rocksdbCounters`

Variable `storage.rocksdb.counters`

Type Boolean

Default `true`

Specifies whether to use advanced counters for *MongoRocks*. You can disable them to improve write performance.

option `--rocksdbSingleDeleteIndex`

Variable `storage.rocksdb.singleDeleteIndex`

Type Boolean

Default `false`

This is an experimental feature. Enable it only if you know what you are doing.

CHAPTER

SEVEN

HOT BACKUP

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

Making a backup

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

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

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

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

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

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

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

Streaming hot backups to a remote destination

Percona Server for MongoDB enables uploading hot backups to an Amazon S3 or a compatible storage service, such as MinIO.

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

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

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

Field	Type	Description
bucket	string	The only mandatory field. Names are subject to restrictions described in the 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 *MinIO* 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 *minio server* command this should field should contain *HTTP*.
useVirtual-Addressing	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 *credentials* configuration file. If not specified, the profile named **default** is used.
ac-cessKeyId	string	The access key id
secretAc-cessKey	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 `~/.aws/credentials`.

An example of the 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"}})
```

See also:

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

Restoring data from backup

Restoring from backup on a standalone server

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

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

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

Restoring from backup in a replica set

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

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

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

The restore steps are the following:

1. Stop the `mongod` service:

```
$ systemctl stop mongod
```

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

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

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

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

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

```
$ systemctl start mongod
```

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

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

6. Restart the node with the replication enabled
   - Shut down the node.

     ```
     $ systemctl stop mongod
     ```

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

     ```
     $ systemctl start mongod
     ```

7. Initiate a new replica set

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

CHAPTER

EIGHT

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.

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 }
```

Important: *MongoDB* uses the *sampleRate* parameter in place of `rateLimit`.

See also:

***MongoDB* Documentation: The *profile* command** https://docs.mongodb.com/manual/reference/command/profile/#profile

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

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

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

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

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

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

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

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.

CHAPTER

NINE

EXTERNAL AUTHENTICATION

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. External authentication allows the MongoDB server to verify the client's user name and password against a separate service, such as OpenLDAP or Active Directory. This allows users to access the database with the same credentials that they use for their emails or workstations.

Percona Server for MongoDB supports the following external authentication mechanisms:

- *LDAP authentication using SASL*;
- *Kerberos authentication*;
- *Authentication and authorization with direct binding to LDAP*.

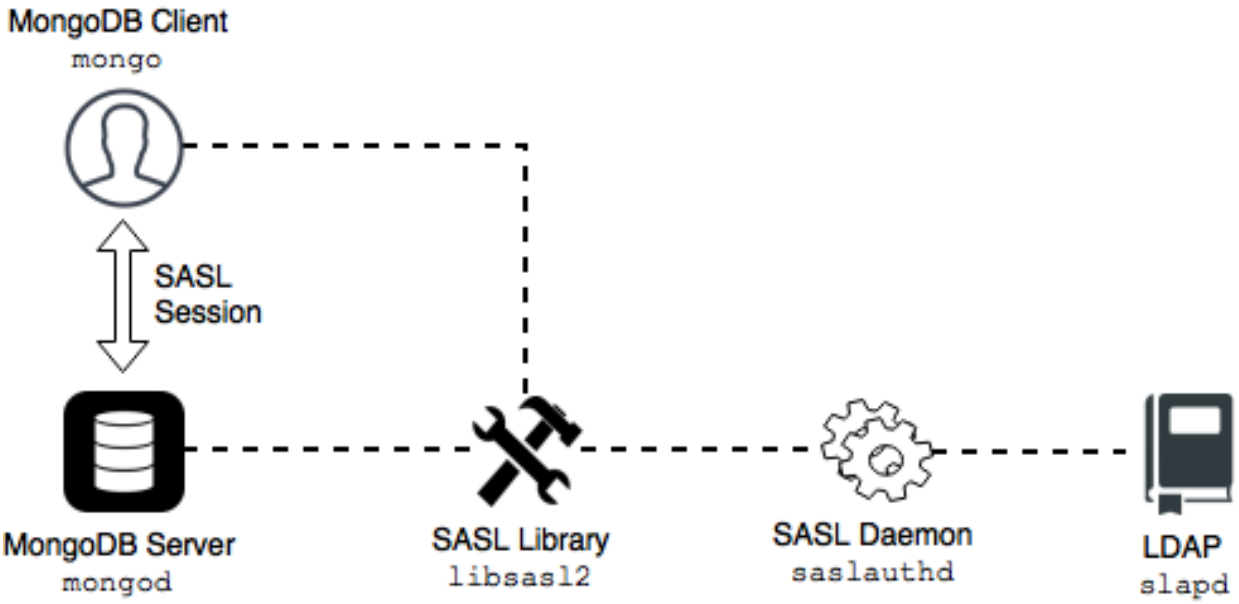
LDAP authentication using SASL

Overview

The following components are necessary for external authentication to work:

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

The following image illustrates this architecture:



An authentication session uses the following sequence:

1. A `mongo` client connects to a running `mongod` instance.
2. The client creates a `PLAIN` authentication request using the SASL library.
3. The client then sends this SASL request to the server as a special Mongo command.
4. The `mongod` server receives this SASL Message, with its authentication request payload.
5. The server then creates a SASL session scoped to this client, using its own reference to the SASL library.
6. Then the server passes the authentication payload to the SASL library, which in turn passes it on to the `saslauthd` daemon.
7. The `saslauthd` daemon passes the payload on to the LDAP service to get a YES or NO authentication response (in other words, does this user exist and is the password correct).
8. The YES/NO response moves back from `saslauthd`, through the SASL library, to `mongod`.
9. The `mongod` server uses this YES/NO response to authenticate the client or reject the request.
10. If successful, the client has authenticated and can proceed.

Environment setup and configuration

This section 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.

The following components are required:

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

The following steps will help you configure your environment:

Assumptions

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

1. You have the LDAP server up and running. The LDAP server is accessible to the server with the *Percona Server for MongoDB* installed.
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` privileges 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:

   For *RedHat and CentOS*:

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

   For *Debian and Ubuntu*:

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

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

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

   For *RedHat/CentOS*, specify the `ldap` value for the `--MECH` option using the following command:

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

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

   ```
   MECH=ldap
   ```

   For *Debian/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"
```

3. Create the `/etc/saslauthd.conf` configuration file and specify these settings required for `saslauthd` to connect to a local OpenLDAP service (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 and bind domain name. This allows the `saslauthd` service to connect to the LDAP service as root. 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=%ucn=%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. 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:OK "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 `mongodb.conf` and placed in a directory where `libsasl2` can find and read it. `libsasl2` is hard-coded to look in certain directories at build time. This location may be different depending on the installation method.

In the configuration file, specify the following:

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

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

See Also

- SASL documentation:

Configuring `mongod` server

To enable external authentication, you must 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" ] }
```

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:

```
service mongod restart
```

When everything is configured properly, you can use the *External authentication commands*.

External authentication commands

Use the following command to add an external user to the `mongod` server:

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

The previous example assumes that you have set up the server-wide admin user/role and have successfully authenticated as that user locally.

Note: External users cannot have roles assigned in the admin database.

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})
```

Based on the material from Percona Database Performance Blog

This section is based on the blog post *Percona Server for MongoDB Authentication Using Active Directory* by *Doug Duncan*:
https://www.percona.com/blog/2018/12/21/percona-server-for-mongodb-authentication-using-active-directory/

Kerberos authentication

Percona Server for MongoDB supports Kerberos authentication starting from release 3.6.19-7.0. It is implemented the same way as in MongoDB 3.6 Enterprise.

See also:

MongoDB Documentation:

- Kerberos Authentication

Authentication and authorization with direct binding to LDAP

As of version 3.6.18-5.0, Percona Server for MongoDB supports LDAP Authorization.

This feature has been supported in MongoDB 3.6 Enterprise since its version 3.4.

As of version 3.6.21-10.0, Percona Server for MongoDB supports LDAP referrals as defined in RFC 4511 4.1.10. For security reasons, LDAP referrals are disabled by default. Double-check that using referrals is safe before enabling them.

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

```
setParameter:
   ldapFollowReferrals: true
```

Connection pool

As of version 3.6.21-10.0, *Percona Server for MongoDB* always uses a connection pool to LDAP server to process authentication requests. The connection pool is enabled by default. The default connection pool size is 2 connections.

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

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

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

Alternatively, edit the configuration file:

```
setParameter:
  ldapConnectionPoolSizePerHost: 5
```

Support for multiple LDAP servers

As of version 3.6.21-11.0, you can specify multiple LDAP servers for failover. *Percona Server for MongoDB* sends authentication requests to the first server defined in the list. When this server is down or unavailable, it sends requests to the next server and so on. Note that *Percona Server for MongoDB* keeps sending requests to this server even after the unavailable server recovers.

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

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

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

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

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

See also:

- **MongoDB Documentation:**
  - LDAP Authorization
  - Authenticate and Authorize Users Using Active Directory via Native LDAP
- LDAP referrals

CHAPTER

TEN

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
```

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

Audit Options

The following options control audit logging:

option --auditDestination

Variable auditLog.destination

Type String

Enables auditing and specifies where to send audit events:

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

option --auditFilter

Variable auditLog.filter

Type String

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

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

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

For more information, see *Audit Filter Examples*.

option --auditFormat

Variable auditLog.format

Type String

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

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

option --auditPath

Variable auditLog.path

Type String

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

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

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

Audit Message Syntax

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

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

atype Event type

ts Date and UTC time of the event

local Local IP address and port number of the instance

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

users Users associated with the event

roles Roles granted to the user

param Details of the event associated with the specific type

result Exit code (`0` for success)

Audit Filter Examples

The following examples demonstrate the flexibility of audit log filters.

Basic Filtering

For example, you can log actions only from user *john* on all databases:

- Command line:

```
--auditDestination file --auditFilter '{ "users.user": "john" }'
```

- Config file:

```
auditLog:
  destination: file
  filter: '{ "users.user": "john" }'
```

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 }
```

Enabling Auditing of Authorization Success

By default, only authorization failures for the `authCheck` action are logged by the audit system. To enable logging of authorization successes, set the `auditAuthorizationSuccess` parameter to `true`.

Note: Enabling this parameter is required if you want to filter CRUD operations in the audit log, because CRUD operations are logged under `authCheck` action.

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
```

Warning: Enabling `auditAuthorizationSuccess` can impact performance compared to logging only authorization failures.

CHAPTER

ELEVEN

LOG REDACTION

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

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

Log redaction is important for 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 }
)
```

CHAPTER

TWELVE

ENABLING AUTHENTICATION

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

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 it 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.

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
```

CHAPTER

THIRTEEN

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.

See also:

MongoDB Documentation: Encryption at Rest https://docs.mongodb.com/manual/core/security-encryption-at-rest/#encryption-at-rest

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 in MongoDB. In the current release of *Percona Server for MongoDB*, the data encryption at rest does not include support for KMIP (Key Management Interoperability Protocol), or Amazon AWS key management services.

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.
- Local key management using a keyfile.

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

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

- *Encrypting Rollback Files*
- *Migrating from Key File Encryption to HashiCorp Vault Encryption*

Important Configuration Options

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

- AES256-CBC
- AES256-GCM

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

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

See also:

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

HashiCorp Vault Integration

Starting from version 3.6.13-3.3, *Percona Server for MongoDB* provides HashiCorp Vault integration. 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* https://www.percona.com/blog/2020/04/21/using-vault-to-store-the-master-key-for-data-at-rest-encryption-on-percona-server-fo

How to configure the KV Engine: https://www.vaultproject.io/api/secret/kv/kv-v2.html

---

HashiCorp Vault Parameters

Command line	Config file	Type	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. Note that vault secrets path format must be:
			`<vault_secret_mount>/data/<custom_path>`
			where:
			• `<vault_secret_mount>` is your Vault KV Secrets Engine;
			• `data` is the mandatory path prefix required by Version 2 API;
			• `<custom_path>` is your secrets path
			Example:
			`secret_v2/data/psmdb-test/rs1-27017`
			Note: It is recommended to use different secret paths for every database node.

vaultRotateMasterKey	security.vault.rotateMasterKey	switch	Enables master key rotation
vaultServerCAFile	security.vault.serverCAFile	string	The path to the TLS certificate file
vaultDisableTLSForTesting	security.vault.disableTLSForTesting	switch	Disables secure connection to HashiCorp Vault using SSL/TLS client certificates

Config file example

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

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

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

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.

For a replica set, the 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.

Local key management using a keyfile

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

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

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

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

- **400** - only the owner may read the file.

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

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

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

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

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

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

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

See also:

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

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

All these options can be specified in the configuration file:

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

See also:

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

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

Migrating from Key File Encryption to HashiCorp Vault Encryption

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

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

Assumptions

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

1. Stop `mongod`.

```
$ sudo systemctl stop mongod
```

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

```
# Retrieve the key value from the keyfile
$ sudo cat /data/key/mongodb.key
d0JTFcePmvROyLXwCbAH8fmiP/ZRm0nYbeJDMGaI7Zw=
# Insert the key into vault
$ vault kv put secret/dc/psmongodb1 value=d0JTFcePmvROyLXwCbAH8fmiP/
↪ZRm0nYbeJDMGaI7Zw=
```

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

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

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

4. Start the `mongod` service

```
$ sudo systemctl start mongod
```

CHAPTER

FOURTEEN

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 native 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 normal MongoDB full text search.

Usage

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

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

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

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

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

See also:

***MongoDB* documentation:**

- Text search
- Text indexes
- $text operator

More information about ngram implementation:

- https://github.com/percona/percona-server-mongodb/blob/v3.6/src/mongo/db/fts/ngram-tokenizer.md

Part IV

Reference

CHAPTER

FIFTEEN

SWITCHING STORAGE ENGINES

By default, *Percona Server for MongoDB* runs with WiredTiger. There is also the original MMAPv1 storage engine, as well as optional *Percona Memory Engine* and *MongoRocks* storage engines to choose from. Each one is designed for specific purposes and workloads.

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`).

Data created by one storage engine is not compatible with other storage engines, because each one has its own data model. When changing the storage engine, you have to do one of the following:

See also:

MongoDB Documentation: Configuration File Options

- storage.engine Options
- storage.wiredTiger Options
- storage.inmemory Options

Data files created by one storage engine are not compatible with other storage 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 even when using *Percona Memory Engine*. Though in-memory storage engine stores all data in memory, some metadata files, diagnostics logs and statistics metrics are still written to disk.

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.

If there is data that you want to migrate and make compatible with the new storage engine, use the following methods:

- for replica sets, use the "rolling restart" process.

  Switch to the new storage engine on the secondary node. 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
```

Wait for the node to rejoin with the other replica set members and report the SECONDARY status.

Repeat the procedure on the remaining nodes.

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

```
$ mongodump --out <dumpDir>
$ 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.wiredTiger.engineConfig.cacheSizeGB or
$ # storage.inMemory.engineConfig.inMemorySizeGB
$ service mongod start
$ mongorestore <dumpDir>
```

CHAPTER

SIXTEEN

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.

CHAPTER

SEVENTEEN

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` variables 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> } )
```

variable `cursorTimeoutMillis`

Value Type *integer*

Default `600000` (ten minutes)

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

variable `failIndexKeyTooLong`

Value Type *boolean*

Default `true`

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

variable `internalQueryPlannerEnableIndexIntersection`

Value Type *boolean*

Default `true`

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

variable `ttlMonitorEnabled`

Value Type *boolean*

Default `true`

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

variable `ttlMonitorSleepSecs`

Value Type *integer*

Default `60` (one minute)

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

CHAPTER

EIGHTEEN

CONTACTING, CONTRIBUTING, REPORTING BUGS

If you want to contact developers, use the community forum.

Note: Please search the forum for similar questions and discussions before openning a new thread.

You will need to sign up for an account to post in the forum.

If you want to explore source code and contribute to the project use the GitHub repo.

Note: Search existing pull requests and recent commits for code that may fix what you are planning to suggest.

You will need a public GitHub account and request contributor access to the repo.

If you want to report a bug or feature request, use the PSMDB project in JIRA.

Note: Search JIRA for existing tickets before submitting a bug or feature request.

You will need a JIRA account to report bugs.

CHAPTER

NINETEEN

PERCONA SERVER FOR MONGODB 3.6 RELEASE NOTES

Percona Server for MongoDB 3.6.23-13.0

Date March 30, 2021

Installation Installing Percona Server for MongoDB

Percona Server for MongoDB 3.6.23-13.0 is based on MongoDB 3.6.23 Community Edition and does not include any additional changes.

Percona Server for MongoDB 3.6.22-12.0

Date February 23, 2021

Installation Installing Percona Server for MongoDB

Percona Server for MongoDB 3.6.22-12.0 is an enhanced, open source, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 3.6.22 Community Edition. It supports MongoDB 3.6.22 protocols and drivers.

Bugs Fixed

- PSMDB-817: LDAP ConnectionPoller always uses up CPU of one core (Thanks to user cleiton.domazak for reporting this issue)

Percona Server for MongoDB 3.6.21-11.0

Date December 28, 2020

Installation Installing Percona Server for MongoDB

Percona Server for MongoDB 3.6.21-11.0 is an enhanced, open source, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 3.6.21 Community Edition. It supports MongoDB 3.6.21 protocols and drivers.

Improvements

- PSMDB-745: Add support for multiple LDAP servers for authentication
- PSMDB-761: Add validateLDAPServerConfig config option

Bugs Fixed

- PSMDB-788: Fix LDAP rebind procedure to allow LDAP referrals to work with ldapBindMethod==sasl

Percona Server for MongoDB 3.6.21-10.0

Date November 26, 2020

Installation Installing Percona Server for MongoDB

Percona Server for MongoDB 3.6.21-10.0 is an enhanced, open source, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 3.6.21 Community Edition. It supports MongoDB 3.6.21 protocols and drivers.

Improvements

- PSMDB-758: Add mongobridge as a part of percona-server-mongodb-server package
- PSMDB-755: Add ldapDebug, ldapFollowReferrals and ldapConnectionPoolSizePerHost server parameters
- PSMDB-376: Update SAN recognition for IP addresses on OpenSSL

Bugs Fixed

- PSMDB-766: Redirect openldap debug messages to mongodb log
- PSMDB-544: Binaries perconadecrypt and mongobridge do not have a version

Percona Server for MongoDB 3.6.20-9.0

Date October 22, 2020

Installation Installing Percona Server for MongoDB

Percona Server for MongoDB 3.6.20-9.0 is an enhanced, open source, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 3.6.20 Community Edition. It supports MongoDB 3.6.20 protocols and drivers.

Improvements

- PSMDB-711: Improve audit log performance

Bugs Fixed

- PSMDB-712: User can't be authorized via LDAP due to 'LDAP search failed with error: Referral'
- PSMDB-715: createBackup using AWS remote location fails with "EntityTooLarge"
- PSMDB-707: LDAP authentication randomly fails with the "Bad parameter to an ldap routine" message in the log
- PSMDB-677: `mongosh` cannot authenticate LDAP user

Percona Server for MongoDB 3.6.19-8.0

Date October 9, 2020

Installation Installing Percona Server for MongoDB

Percona Server for MongoDB 3.6.19-8.0 is an enhanced, open source, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 3.6.19 Community Edition. It supports MongoDB 3.6.19 protocols and drivers.

This release fixes security vulnerability CVE-2020-26542.

Improvements

- PSMDB-674: Provide binary tarball with shared libs and glibc suffix

Percona Server for MongoDB 3.6.19-7.0

Date August 13, 2020

Installation Installing Percona Server for MongoDB

Percona Server for MongoDB 3.6.19-7.0 is an enhanced, open source, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 3.6.19 Community Edition. It supports MongoDB 3.6.19 protocols and drivers.

New Features

- PSMDB-582: Added Kerberos authentication to Percona Server for MongoDB

Bugs Fixed

- PSMDB-671: createBackup returns ok:1 for archived backup when there is no disk space available
- PSMDB-656: LDAP - user's permissions remain intact after a user is removed from LDAP
- PSMDB-589: Add ldapUserCacheInvalidationInterval parameter to periodically flush external user cache
- PSMDB-583: Detect a connection loss to LDAP server and reconnect automatically

Percona Server for MongoDB 3.6.18-6.0

Date May 25, 2020

Installation Installing Percona Server for MongoDB

Percona Server for MongoDB 3.6.18-6.0 is an enhanced, open source, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 3.6.18 Community Edition. It supports MongoDB 3.6.18 protocols and drivers.

New Features

- PSMDB-587: Add options to control the size of several WiredTiger hash arrays concerned with data handles and one for session cursor cache
- PSMDB-616: Add "txn_checkpoint_prepare_time" and "txn_checkpoint_tree_helper_time" wiredTiger.transaction stats

Bugs Fixed

- PSMDB-600: Fix leak of dhandle session_inuse counter in __evict_walk through 'error' early loop exit logic

Percona Server for MongoDB 3.6.18-5.0

Date May 19, 2020

Installation Installing Percona Server for MongoDB

Percona Server for MongoDB 3.6.18-5.0 is an enhanced, open source, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 3.6.18-5.0 Community Edition. It supports MongoDB 3.6.18 protocols and drivers.

Improvements

- PSMDB-165: LDAP Authorization. Starting from release 3.6.18-5.0, Percona Server for MongoDB supports LDAP Authorization.

  This feature has been supported in MongoDB 3.6 Enterprise since its version 3.4.

  Note the following limitations of LDAP Authorization in Percona Server for MongoDB:

  - The LDAP (Lightweight Directory Access Protocol) connection pool and all related parameters are not supported.
  - The ldapTimeoutMS parameter is ignored.
  - The ldapUserCacheInvalidationInterval parameter is ignored.
  - The –ldapServers option may only contain a single server (MongoDB 3.6 Enterprise accepts a comma-separated list).

  See our documentation for more information about how *External Authentication* is supported in Percona Server for MongoDB.

  See also:

  MongoDB Documentation:

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

Percona Server for MongoDB 3.6.17-4.0

Date February 13, 2020

Installation *Installing Percona Server for MongoDB*

Percona Server for MongoDB 3.6.17-4.0 is based on MongoDB 3.6.17.

Bugs Fixed

- PSMDB-473: The `logApplicationMessage` command failed even when it was run by the user with extended privileges. The problem has been fixed to allow running the `logApplicationMessage` command by any role that has the *applicationMessage* privilege, such as *clusterManager* or *hostManager*.

```
> db.runCommand({logApplicationMessage: 'find'});
```

Percona Server for MongoDB 3.6.16-3.6

Percona announces the release of Percona Server for MongoDB 3.6.16-3.6 on December 17, 2019. Download the latest version from the Percona website or the Percona Software Repositories.

Percona Server for MongoDB is an enhanced, open source, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 3.6 Community Edition. It supports MongoDB 3.6 protocols and drivers.

Percona Server for MongoDB extends Community Edition functionality by including the *Percona Memory Engine* storage engine, as well as several enterprise-grade features. Also, it includes *MongoRocks* storage engine, which is now deprecated. *Percona Server for MongoDB* requires no changes to MongoDB applications or code.

Percona Server for MongoDB 3.6.16-3.6 is based on MongoDB 3.6.16. and does not include any additional changes.

Percona Server for MongoDB 3.6.15-3.5

Percona announces the release of Percona Server for MongoDB 3.6.15-3.5 on November 26, 2019. Download the latest version from the Percona website or the Percona Software Repositories.

Percona Server for MongoDB is an enhanced, open source, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 3.6 Community Edition. It supports MongoDB 3.6 protocols and drivers.

Percona Server for MongoDB extends Community Edition functionality by including the *Percona Memory Engine* storage engine, as well as several enterprise-grade features. Also, it includes *MongoRocks* storage engine, which is now deprecated. *Percona Server for MongoDB* requires no changes to MongoDB applications or code.

Percona Server for MongoDB 3.6.15-3.5 is based on MongoDB 3.6.15.

This release enables streaming hot backups to Amazon S3 or a compatible storage, such as MinIO. Note that this feature has the Experimental status and is not recommended to be used in a production environment.

For more information about this feature, see the *Hot Backup* section in our documentation.

New Features

- PSMDB-371: Hot backup streaming to a remote destination (Experimental).

Percona Server for MongoDB 3.6.14-3.4

Percona announces the release of Percona Server for MongoDB 3.6.14-3.4 on October 10, 2019. Download the latest version from the Percona website or the Percona Software Repositories.

Percona Server for MongoDB is an enhanced, open source, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 3.6 Community Edition. It supports MongoDB 3.6 protocols and drivers.

Percona Server for MongoDB extends Community Edition functionality by including the *Percona Memory Engine* storage engine, as well as several enterprise-grade features. Also, it includes *MongoRocks* storage engine, which is now deprecated. *Percona Server for MongoDB* requires no changes to MongoDB applications or code.

Percona Server for MongoDB 3.6.14-3.4 is based on MongoDB 3.6.14. In this release, the license of RPM and DEB packages has been changed from AGPLv3 to SSPL.

Bugs Fixed

- PSMDB-447: The license for RPM and DEB packages has been changed from AGPLv3 to SSPL.

Percona Server for MongoDB 3.6.13-3.3

Percona announces the release of Percona Server for MongoDB 3.6.13-3.3 on June 21, 2019. Download the latest version from the Percona website or the Percona Software Repositories.

Percona Server for MongoDB is an enhanced, open source, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 3.6 Community Edition. It supports MongoDB 3.6 protocols and drivers.

Percona Server for MongoDB extends Community Edition functionality by including the *Percona Memory Engine* storage engine, as well as several enterprise-grade features. Also, it includes *MongoRocks* storage engine, which is now deprecated. *Percona Server for MongoDB* requires no changes to MongoDB applications or code.

Percona Server for MongoDB 3.6.13-3.3 introduces the support of HashiCorp Vault key management service. For more information, see *Data at Rest Encryption* in the documentation of *Percona Server for MongoDB*.

New Features

- PSMDB-322: HashiCorp Vault support

Percona Server for MongoDB 3.6.12-3.2

Date May 3, 2019

Download http://www.percona.com/downloads/percona-server-mongodb-3.6/

Installation *Installing Percona Server for MongoDB*

Percona announces the release of Percona Server for MongoDB 3.6.12-3.2 on May 3, 2019. Download the latest version from the Percona website or the Percona Software Repositories.

Percona Server for MongoDB is an enhanced, open source, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 3.6 Community Edition. It supports MongoDB 3.6 protocols and drivers.

Percona Server for MongoDB extends Community Edition functionality by including the *Percona Memory Engine* storage engine, as well as several enterprise-grade features. Also, it includes *MongoRocks* storage engine, which is now deprecated. *Percona Server for MongoDB* requires no changes to MongoDB applications or code.

Bugs Fixed

- PSMDB-343: Building from sources could fail with *openssl* version 1.1.1.

Percona Server for MongoDB 3.6.11-3.1

Date March 15, 2019

Download http://www.percona.com/downloads/percona-server-mongodb-3.6/

Installation *Installing Percona Server for MongoDB*

Percona announces the release of Percona Server for MongoDB 3.6.11-3.1 on March 15, 2019. Download the latest version from the Percona website or the Percona Software Repositories.

Percona Server for MongoDB is an enhanced, open source, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 3.6 Community Edition. It supports MongoDB 3.6 protocols and drivers.

Percona Server for MongoDB extends Community Edition functionality by including the *Percona Memory Engine* storage engine, as well as several enterprise-grade features. Also, it includes *MongoRocks* storage engine, which is now deprecated. *Percona Server for MongoDB* requires no changes to MongoDB applications or code.

Release 3.6.11-3.1 extends the `buildInfo` command with the `psmdbVersion` key to report the version of *Percona Server for MongoDB*. If this key exists then *Percona Server for MongoDB* is installed on the server. This key not available from MongoDB.

Bugs Fixed

- PSMDB-216: The database command `buildInfo` provides the `psmdbVersion` key to report the version of *Percona Server for MongoDB*. If this key exists then *Percona Server for MongoDB* is installed on the server. This key is not available from MongoDB.

Percona Server for MongoDB 3.6.10-3.0

Date February 6, 2019

Download http://www.percona.com/downloads/percona-server-mongodb-3.6/

Installation *Installing Percona Server for MongoDB*

Percona announces the release of Percona Server for MongoDB 3.6.10-3.0 on February 6, 2019. Download the latest version from the Percona website or the Percona Software Repositories. This release is *also available for Ubuntu 18.10 (Cosmic Cuttlefish)*.

Percona Server for MongoDB is an enhanced, open source, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 3.6 Community Edition. It supports MongoDB 3.6 protocols and drivers.

Percona Server for MongoDB extends Community Edition functionality by including the *Percona Memory Engine* storage engine, as well as several enterprise-grade features. Also, it includes *MongoRocks* storage engine, which is now deprecated. *Percona Server for MongoDB* requires no changes to MongoDB applications or code.

In *Percona Server for MongoDB* 3.6.10-3.0, data at rest encryption becomes GA. The data at rest encryption feature now covers the temporary files used for external sorting and the rollback files. You can decrypt and examine the contents of the rollback files using the new **perconadecrypt** command line tool.

In this release, *Percona Server for MongoDB* supports the Ngram full text search engine. Thanks to *@SunguckLee on github* for this contribution. To enable Ngram full text search, create an index passing **ngram** to the `default_language` parameter.

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

New Features

- PSMDB-276: **perconadecrypt** tool is now available for decrypting the encrypted rollback files.
- PSMDB-250: The Ngram full text search engine has been added to *Percona Server for MongoDB*. Thanks to *@SunguckLee on github*

Bugs Fixed

- PSMDB-234: It was possible use a key file for encryption the owner of which was not the owner of the `mongod` process.
- PSMDB-269: In some cases, hot backup was not using the correct path to the `keydb` directory designated for data encryption.
- PSMDB-273: When using data at rest encryption, temporary files for external sorting and rollback files were not encrypted
- PSMDB-272: **mongos** could crash when running the the **createBackup** command.
- PSMDB-233: WiredTiger encryption options were silently ignored at server startup, although a storage engine different from WiredTiger was used.
- PSMDB-257: MongoDB could not be started with a group-readable `key file` owned by `root`.
- PSMDB-266: In some cases, it was possible to add arbitrary collections to the `keydb` directory which may only store encryption data.

Other bugs fixed: PSMDB-239, PSMDB-243

Percona Server for MongoDB 3.6.8-2.0

Date October 31, 2018

Download http://www.percona.com/downloads/percona-server-mongodb-3.6/

Installation *Installing Percona Server for MongoDB*

Percona announces the release of Percona Server for MongoDB 3.6.8-2.0 on October 31, 2018. Download the latest version from the Percona website or the Percona Software Repositories.

Percona Server for MongoDB is an enhanced, open source, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 3.6 Community Edition. It supports MongoDB 3.6 protocols and drivers.

Percona Server for MongoDB extends Community Edition functionality by including the *Percona Memory Engine* storage engine, as well as several enterprise-grade features. Also, it includes *MongoRocks* storage engine, which is now deprecated. *Percona Server for MongoDB* requires no changes to MongoDB applications or code.

This release introduces data at rest encryption for the WiredTiger storage engine. Data at rest encryption for WiredTiger in *Percona Server for MongoDB* is compatible with the upstream implementation. In this release of *Percona Server for MongoDB*, this feature is of **BETA** quality and should not be used in a production environment.

Note that *Percona Server for MongoDB* 3.6.8-2.0 is based on MongoDB 3.6.8 which is distributed under the GNU AGPLv3 license. Subsequent releases of *Percona Server for MongoDB* will change its license to SSPL (Server Side Public License) when we move to the SSPL codebase released by MongoDB. For more information, see Percona Statement on MongoDB Community Server License Change.

This release also contains a fix for bug PSMDB-238

Known Issues

- PSMDB-233: When starting *Percona Server for MongoDB* 3.6 with WiredTiger encryption options but using a different storage engine, the server starts normally and produces no warnings that these options are ignored.
- PSMDB-239: WiredTiger encryption is not disabled with using the *Percona Memory Engine* storage engine.
- PSMDB-245: KeyDB's WiredTiger logs are not properly rotated without restarting the server.

Percona Server for MongoDB 3.6.7-1.5

Date September 17, 2018

Download http://www.percona.com/downloads/percona-server-mongodb-3.6/

Installation *Installing Percona Server for MongoDB*

Percona Server for MongoDB is an enhanced, open source, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 3.6 Community Edition. It supports MongoDB 3.6 protocols and drivers.

Percona Server for MongoDB extends Community Edition functionality by including the *Percona Memory Engine* storage engine, as well as several enterprise-grade features. Also it includes *MongoRocks* storage engine, which is now deprecated. *Percona Server for MongoDB* requires no changes to MongoDB applications or code.

This release is based on MongoDB 3.6.7 and does not include any additional changes.

Percona Server for MongoDB 3.6.6-1.4

Date August 3, 2018

Download http://www.percona.com/downloads/percona-server-mongodb-3.6/

Installation *Installing Percona Server for MongoDB*

Percona Server for MongoDB is an enhanced, open source, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 3.6 Community Edition. It supports MongoDB 3.6 protocols and drivers.

Percona Server for MongoDB extends Community Edition functionality by including the *Percona Memory Engine* storage engine, as well as several enterprise-grade features. Also it includes *MongoRocks* storage engine, which is now deprecated. *Percona Server for MongoDB* requires no changes to MongoDB applications or code.

This release is based on MongoDB 3.6.6 and does not include any additional changes.

Percona Server for MongoDB 3.6.5-1.3

Date July 11, 2018

Download http://www.percona.com/downloads/percona-server-mongodb-3.6/

Installation *Installing Percona Server for MongoDB*

Percona Server for MongoDB is an enhanced, open source, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 3.6 Community Edition. It supports MongoDB 3.6 protocols and drivers.

Percona Server for MongoDB extends Community Edition functionality by including the *Percona Memory Engine* storage engine, as well as several enterprise-grade features. Also it includes *MongoRocks* storage engine, which is now deprecated. *Percona Server for MongoDB* requires no changes to MongoDB applications or code.

This release is based on MongoDB 3.6.5 and does not include any additional changes.

Percona Server for MongoDB 3.6.4-1.2

Date May 23, 2018

Download http://www.percona.com/downloads/percona-server-mongodb-3.6/

Installation *Installing Percona Server for MongoDB*

Percona Server for MongoDB is an enhanced, open source, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 3.6 Community Edition. It supports MongoDB 3.6 protocols and drivers.

Percona Server for MongoDB extends Community Edition functionality by including the *Percona Memory Engine* storage engine, as well as several enterprise-grade features. Also it includes *MongoRocks* storage engine, which is now deprecated. *Percona Server for MongoDB* requires no changes to MongoDB applications or code.

This release is based on MongoDB 3.6.4 and provides the following additional changes:

- PSMDB-205: `mongod` failed to initialize if audit filter was set to record Action type events specified with the `$in` expression.
- PSMDB-207: a premature initialization of the feature compatibility version in global parameters was fixed for the RocksDB storage engine.
- PSMDB-209: CentOS 6 and CentOS 7 RPM packages contained config file with a wrong link to the online Percona Memory Engine documentation.

Note: As mentioned in the *Percona Server for MongoDB* 3.6.3-1.1 Release Notes, MongoRocks is deprecated in *Percona Server for MongoDB* 3.6.

Percona Server for MongoDB 3.6.3-1.1

Date April 24, 2018

Download http://www.percona.com/downloads/percona-server-mongodb-3.6/

Installation *Installing Percona Server for MongoDB*

Percona Server for MongoDB is an enhanced, open source, and highly-scalable database that is a fully-compatible, drop-in replacement for MongoDB 3.6 Community Edition. It supports MongoDB 3.6 protocols and drivers.

Percona Server for MongoDB extends Community Edition functionality by including the *Percona Memory Engine* storage engine, as well as several enterprise-grade features. *Percona Server for MongoDB* requires no changes to MongoDB applications or code.

This release is based on MongoDB 3.6.3 and provides the following additional changes:

- MongoRocks is deprecated in Percona Server for MongoDB 3.6 and it will be fully removed in the next major version of *Percona Server for MongoDB*. Feature compatibility version is set to 3.4 when using *Percona Server for MongoDB* 3.6 with MongoRocks, so 3.6 features, such as retryable writes and causal consistency, cannot be used. Additionally, read concern majority may produce unreliable results.
- PSMDB-191: Fixed a bug in *MongoRocks* engine initialization code which caused wrong initialization of `_maxPrefix` value. This could lead to reuse of dropped prefix and to accidental removal of data from the collection using reused prefix.

  In some specific conditions data records could disappear at arbitrary moment of time from the collections or indexes created after server restart.

  This could happen as the result of the following sequence of events:

  1. User deletes one or more indexes or collections. These should be the ones using maximum existing prefixes values.
  2. User shuts down the server before *MongoRocks* compaction thread executes compactions of deleted ranges.
  3. User restarts the server and creates new collections. Due to the bug those new collections and their indexes may get the same prefix values which were deleted and not yet compacted. User inserts some data into the new collections.
  4. After the server restart *MongoRocks* compaction thread continues executing compactions of the deleted ranges and this process may eventually delete data from the collections sharing prefixes with deleted ranges.
- PSMDB-178: RocksSnapshotManager was reworked to match the new model of interaction between MongoDB and storage engine's snapshot manager.

Percona Server for MongoDB 3.6.1-1.0 Beta

Date January 23, 2018

Download http://www.percona.com/downloads/percona-server-mongodb-3.6/

Installation *Installing Percona Server for MongoDB*

Note: This version is considered *BETA* quality and it's not intended to be used in production.

This release is based on MongoDB 3.6.1 and provides the following features:

- MongoDB's original MMAPv1 storage engine, and the default WiredTiger engine
- Optional *Percona Memory Engine* and *MongoRocks* storage engines
- *External SASL authentication* using OpenLDAP or Active Directory
- *Audit logging* to track and query database interactions of users or applications

- *Hot Backup* for the default WiredTiger and alternative *MongoRocks* storage engine
- *Profiling Rate Limit* to decrease the impact of the profiler on performance

Known Issues

- PSMDB-184 - MongoRocks reindexing collection or repairing database commands can cause inconsistency when doing immediate majority read in replica set.
- PSMDB-185 - MongoDB has its own notion of time, including an expected commit order for transactions that may be inconsistent with the order assigned by MongoRocks. This can cause majority read inconsistencies in highly concurrent environments in replica set mode.

CHAPTER

TWENTY

GLOSSARY

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

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

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

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

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

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

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

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

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.

CHAPTER

TWENTYONE

COPYRIGHT AND LICENSING INFORMATION

Documentation Licensing

Software License

Percona Server for MongoDB is source-available software.

CHAPTER

TWENTYTWO

TRADEMARK POLICY

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

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

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

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

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

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

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

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

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

In the event of doubt as to any of the conditions or exceptions outlined in this Trademark Policy, please contact trademarks@percona.com for assistance and we will do our very best to be helpful.

INDEX

Symbols

A

C

D

F

I

J

R

T