Percona Backup for MongoDB Documentation

Release 1.3.3

Nov 04, 2020

CONTENTS

Percona Backup for MongoDB is a distributed, low-impact solution for achieving consistent backups of MongoDB sharded clusters and replica sets.

Percona Backup for MongoDB supports Percona Server for MongoDB and MongoDB Community v3.6 or higher with MongoDB Replication enabled.

Note: Percona Backup for MongoDB doesn't work on standalone MongoDB instances. This is because Percona Backup for MongoDB requires an *oplog* to guarantee backup consistency. Oplog is available on nodes with replication enabled.

For testing purposes, you can deploy Percona Backup for MongoDB on a single-node replica set. (Specify the `replication.replSetName` in the configuration file of the standalone server.)

See also:

MongoDB Documentation: Convert a Standalone to a Replica Set https://docs.mongodb.com/manual/tutorial/convert-standalone-to-replica-set/

The Percona Backup for MongoDB project inherited from and replaces *mongodb_consistent_backup*, which is no longer actively developed or supported.

Features

- Backup and restore for both classic non-sharded replica sets and sharded clusters
- Simple command-line management utility
- Replica set and sharded cluster consistency through oplog capture
- Distributed transaction consistency with MongoDB 4.2+
- Simple, integrated-with-MongoDB authentication
- No need to install a coordination service on a separate server.
- Use any S3-compatible storage
- Users with classic, locally-mounted remote filesystem backup servers can use 'file system' instead of 's3' storage type.

Part I

Introduction

CHAPTER

ONE

PERCONA BACKUP FOR MONGODB INTRO

How to use Percona Backup for MongoDB: going back in time (`pbm restore`)

Even in a highly-available architecture, such as with MongoDB replication, backups are still required even though losing one server is not fatal. Whether for a complete or partial data disaster you can use PBM (Percona Backup for MongoDB) to go back in time to the best available backup snapshot.

For example, imagine your web application's update was released on Sunday, June 9th 23:00 EDT but, by 11:23 Monday, someone realizes that the update has a bug that is wiping the historical data of any user who logged in. Nobody likes to have downtime, but it's time to roll back: what's the best backup to use?

```
$ pbm list
```

Output

```
$ pbm list
2019-09-10T07:04:14Z
2019-09-09T07:03:50Z
2019-09-08T07:04:21Z
2019-09-07T07:04:18Z
```

The most recent daily backup is 03:04 EDT (07:04 UTC), which would include 4 hours of damage caused by the bug. Let's restore the one before that:

```
$ pbm restore 2019-06-09T07:03:50Z
```

Next time there is an application release, it might be best to make an extra backup manually just before:

```
$ pbm backup
```

Percona Backup for MongoDB is an uncomplicated command-line tool by design. The full set of commands:

```
$ pbm help
usage: pbm [<flags>] <command> [<args> ...]

Percona Backup for MongoDB

Flags:
  --help                     Show context-sensitive help (also try --help-long
                             and --help-man).
  --mongodb-uri=MONGODB-URI  MongoDB connection string. Default value read from
                             environment variable PBM_MONGODB_URI.
```

```
Commands:
  help [<command>...]
    Show help.

  config [<flags>]
    Set, change or list the config

  backup
    Make backup

  restore <backup_name>
    Restore backup

  delete-backup <backup_name> [<flags>]
    Delete backup(s)

  cancel-backup
    Cancel backup

  list [<flags>]
    Backup list

  version [<flags>]
    PBM version info
```

Part II

Architecture

CHAPTER

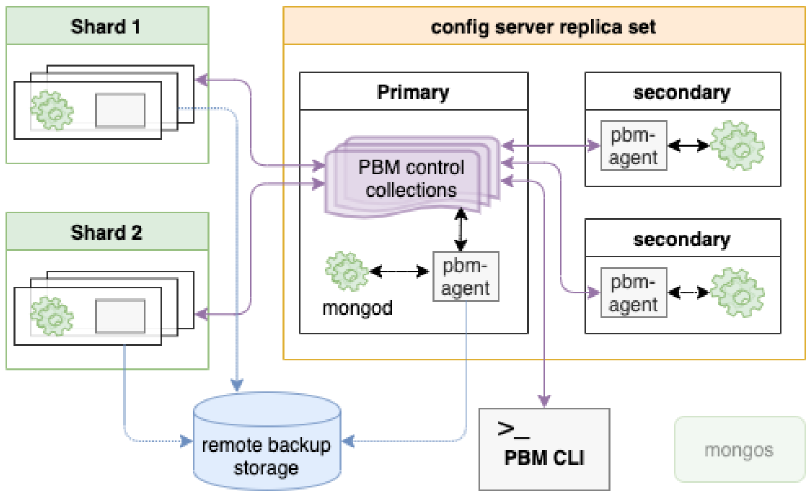
TWO

ARCHITECTURE

Percona Backup for MongoDB consists of the following components:

- *pbm-agent* is a process running on every `mongod` node within the cluster or a replica set that performs backup and restore operations.
- *PBM Command Line Utility (pbm)* is a command-line utility that instructs pbm-agents to perform an operation.

  A single **`pbm-agent`** is only involved with one cluster (or non-sharded replica set). The `pbm` CLI utility can connect to any cluster it has network access to, so it is possible for one user to list and launch backups or restores on many clusters.
- *PBM Control Collections* are special collections in MongoDB that store the configuration data and backup states. Both `pbm` CLI and **`pbm-agent`** use PBM Control collections to check backup status in MongoDB and communicate with each other.
- *Remote Backup Storage* is where Percona Backup for MongoDB saves backups. It can be either an *S3 compatible storage* or a filesystem-type storage.



pbm-agent

Percona Backup for MongoDB requires one instance of **`pbm-agent`** to be attached locally to each `mongod` instance. This includes replica set nodes that are currently secondaries and config server replica set nodes in a sharded cluster.

There is no **`pbm-agent`** config file. Some configuration is required for the service script (e.g. `systemd` unit file) that will run it though. See *Configuring service init scripts*.

The backup and restore operations are triggered when the **pbm-agent** observes updates made to the PBM control collections by the `pbm` CLI. In a method similar to the way replica set members elect a new primary, the **pbm-agent** processes in the same replica set 'elect' the one to do the backup or restore for that replica set.

PBM Command Line Utility (`pbm`)

`pbm` is the command you will use manually in the shell, and it will also work as a command that can be executed in scripts (for example, by `crond`). It manages your backups through a set of sub-commands:

```
$ pbm help
usage: pbm [<flags>] <command> [<args> ...]

Percona Backup for MongoDB

Flags:
  --help                     Show context-sensitive help (also try --help-long
                             and --help-man).
  --mongodb-uri=MONGODB-URI  MongoDB connection string. Default value read from
                             environment variable PBM_MONGODB_URI.

Commands:
  help [<command>...]
    Show help.

  config [<flags>]
    Set, change or list the config

  backup
    Make backup

  restore <backup_name>
    Restore backup

  delete-backup <backup_name> [<flags>]
    Delete backup(s)

  cancel-backup
    Cancel backup

  list [<flags>]
    Backup list

  version [<flags>]
    PBM version info
```

`pbm` modifies the PBM config by saving it in the PBM Control collection for config values. Likewise it starts and monitors backup or restore operations by updating and reading other PBM control collections for operations, log, etc.

`pbm` does not have its own config and/or cache files. Setting the `PBM_MONGODB_URI` environment variable in your shell is a configuration-like step that should be done for practical ease though. (Without `PBM_MONGODB_URI` the `--mongodb-uri` command line argument will need to be specified each time.)

PBM Control Collections

The config and state (current and historical) for backups is stored in collections in the MongoDB cluster or non-sharded replica set itself. These are put in the system `admin` db of the config server replica set to keep them cleanly separated from user db namespaces. (In a non-sharded replica set the `admin` db of the replica set itself is used.)

- *admin.pbmConfig*
- *admin.pbmCmd* (Used to define and trigger operations)
- *admin.pbmLock* (**`pbm-agent`** synchronization-lock structure)
- *admin.pbmBackup* (Log / status of each backup)

The `pbm` command line tool creates these collections as needed. You do not have to maintain these collections, but you should not drop them unnecessarily either. Dropping them during a backup will cause an abort of the backup.

Filling the config collection is a prerequisite to using PBM for executing backups or restores. (See config page later.)

Remote Backup Storage

Percona Backup for MongoDB saves your files to a directory. Conceptually in the case of object store; actually if you are using filesystem-type remote storage. Using `pbm list`, a user can scan this directory to find existing backups even if they never used `pbm` on their computer before.

The files are prefixed with the (UTC) starting time of the backup. For each backup there is one metadata file. For each replicaset in the backup:

- A mongodump-format compressed archive that is the dump of collections
- A (compressed) BSON file dump of the oplog covering the timespan of the backup.

The end time of the oplog slice(s) is the data-consistent point in time of a backup snapshot.

For details about supported backup storage, see *Remote backup storage*.

Part III

Installation and Upgrade

CHAPTER

THREE

INSTALLING PERCONA BACKUP FOR MONGODB

Percona provides and supports installation packages for Percona Backup for MongoDB in the *deb* and *rpm* formats that you can install by using `apt` or `yum` or other interfaces to your package management system.

The Percona Software and Platform Lifecycle page lists Linux distributions for which Percona Backup for MongoDB installation packages are available.

For your convenience, we recommend that you install the `percona-release` utility which makes it easy to install any Percona product on your system.

You may also build and install Percona Backup for MongoDB from source code in case you require a fully controlled installation method.

Regardless of the installation method you choose, the following tools are at your disposal after the installation completes:

Tool	Purpose
pbm	Command-line interface for controlling the backup system
pbm-agent	An agent for running backup/restore actions on a database host

You should install **`pbm-agent`** on every server that has mongod nodes in the MongoDB cluster (or non-sharded replica set). The `pbm` CLI can be installed on any or all servers or desktop computers you wish to use it from, so long as those computers aren't network-blocked from accessing the MongoDB cluster.

See also:

More information about `percona-release` https://www.percona.com/doc/percona-repo-config/percona-release.html

Prerequisites

It is recommended to install Percona Backup for MongoDB from official Percona repositories by using the `percona-release` utility. Starting from v1.3.0, Percona Backup for MongoDB packages are stored in the *pbm* repository.

```
$ sudo percona-release enable pbm release
```

Percona Backup for MongoDB is available for installation from your package management system when you enable the *pbm* repository.

See also:

Configuring Percona repositories https://www.percona.com/doc/percona-repo-config/index.html

Installing Percona Backup for MongoDB Using `apt`

```
$ sudo apt-get update
$ sudo apt-get install percona-backup-mongodb
```

Installing Percona Backup for MongoDB Using `yum`

```
$ sudo yum update
$ sudo yum install percona-backup-mongodb
```

Building from source code

Building the project requires:

- Go 1.11 or above
- make

See also:

Installing and setting up Go tools https://golang.org/doc/install

To build the project (from the project dir):

```
$ go get -d github.com/percona/percona-backup-mongodb
$ cd "$(go env GOPATH)/src/github.com/percona/percona-backup-mongodb"
$ make
```

After **`make`** completes, you can find `pbm` and **`pbm-agent`** binaries in the `./bin` directory:

```
$ cd bin
$ pbm version
```

By running **`pbm version`**, you can verify if Percona Backup for MongoDB has been built correctly and is ready for use.

Output

```
Version:   [pbm version number]
Platform:  linux/amd64
GitCommit: [commit hash]
```

```
GitBranch: master
BuildTime: [time when this version was produced in UTC format]
GoVersion: [Go version number]
```

Percona Backup for MongoDB services and location of configuration files

After Percona Backup for MongoDB is successfully installed on your system, you have `pbm` and **`pbm-agent`** programs on your system.

Configuring service init scripts

To start a **`pbm-agent`**, set the MongoDB connection URI string for the PBM user to the local `mongod` node in the environment file that is included in the *pbm-agent.service* systemd unit file.

With the current systemd unit file (see below), this means setting the "PBM_MONGODB_URI" environment variable in `/etc/default/pbm-agent` (for Debian and Ubuntu) or `/etc/sysconfig/pbm-agent` (for Red Hat or CentOS).

Hint: In Ubuntu and Debian, the path to the `pbm-agent.service` file is `/lib/systemd/system/pbm-agent.service`.

In Red Hat and CentOS, this file is found at the path `/usr/lib/systemd/system/pbm-agent.service`.

```
[Unit]
Description=pbm-agent
After=time-sync.target network.target

[Service]
EnvironmentFile=-/etc/default/pbm-agent
Type=simple
User=pbm
Group=pbm
PermissionsStartOnly=true
ExecStart=/usr/bin/pbm-agent

[Install]
WantedBy=multi-user.target
```

To set the "PBM_MONGODB_URI" environment variable:

1. Create the "pbm user" in the `admin` database. See *Create the PBM user*.
2. Edit the environment file:

```
PBM_MONGODB_URI="mongodb://pbmuser:secretpwd@localhost:27018"
```

See also:

More information about standard MongoDB connection strings *Authentication*

The *Running Percona Backup for MongoDB* section, explains in detail how to start **`pbm-agent`** and provides examples how to use `pbm` commands.

CHAPTER

FOUR

UPGRADING PERCONA BACKUP FOR MONGODB

Similar to installing, the recommended and most convenient way to upgrade Percona Backup for MongoDB is from the Percona repository.

You can upgrade Percona Backup for MongoDB to the **latest version** or to a **specific version**. Since all packages of Percona Backup for MongoDB are stored in the same repository, the following steps apply to both upgrade scenarios:

1. Enable Percona repository.
2. Stop **`pbm-agent`**.
3. Install new version packages (the old ones are automatically removed).
4. Start **`pbm-agent`**.

Important notes

1. Backward compatibility between data backup and restore is supported for upgrades within one major version only (e.g. from 1.1.x to 1.2.y). When you upgrade Percona Backup for MongoDB over several major versions (e.g. from 1.0.x to 1.2.y), we recommend to make a backup right after the upgrade.
2. Starting from version 1.3.0, Percona Backup for MongoDB packages are stored in the *pbm* repository and the *tools* repository for backward compatibility. To upgrade to the version lower than 1.3.0, enable the *tools* repository.
3. Upgrade Percona Backup for MongoDB on all nodes where it is installed.

Enable Percona repository

Install the `percona-release` utility or update it to the latest version as described in Percona Software Repositories Documentation.

Enable the repository running the command as root or via **sudo**

```
$ sudo percona-release enable pbm
```

Note: For `apt`-based systems, run `apt-get update` to update the local cache.

Upgrade Percona Backup for MongoDB using `apt`

Important: Run all commands as root or via **sudo**.

Upgrade to the latest version

1. Stop **pbm-agent**

```
$ sudo systemctl stop pbm-agent
```

2. Install new packages

```
apt-get install percona-backup-mongodb
```

3. Start **pbm-agent**

```
$ sudo systemctl start pbm-agent
```

Upgrade to a specific version

1. List available options:

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

Sample output

```
percona-backup-mongodb | 1.3.1-1.stretch | http://repo.percona.com/tools/apt␣
↪stretch/main amd64 Packages
percona-backup-mongodb | 1.3.0-1.stretch | http://repo.percona.com/tools/apt␣
↪stretch/main amd64 Packages
percona-backup-mongodb | 1.2.1-1.stretch | http://repo.percona.com/tools/apt␣
↪stretch/main amd64 Packages
percona-backup-mongodb | 1.2.0-1.stretch | http://repo.percona.com/tools/apt␣
↪stretch/main amd64 Packages
percona-backup-mongodb | 1.1.3-1.stretch | http://repo.percona.com/tools/apt␣
↪stretch/main amd64 Packages
```

```
percona-backup-mongodb | 1.1.1-1.stretch | http://repo.percona.com/tools/apt␣
↪stretch/main amd64 Packages
percona-backup-mongodb | 1.0.0-1.stretch | http://repo.percona.com/tools/apt␣
↪stretch/main amd64 Packages
percona-backup-mongodb | 1.0-1.stretch | http://repo.percona.com/tools/apt␣
↪stretch/main amd64 Packages
```

2. Stop **`pbm-agent`**:

```
$ sudo systemctl stop pbm-agent
```

3. Install a specific version packages. For example, to upgrade to Percona Backup for MongoDB 1.1.3, run the following command:

```
$ sudo apt-get install percona-backup-mongodb=1.1.3-1.stretch
```

4. Start **`pbm-agent`**:

```
$ sudo systemctl start pbm-agent
```

Upgrade Percona Backup for MongoDB using `yum`

Important: Run all commands as root or via **`sudo`**.

Upgrade to the latest version

1. Stop **`pbm-agent`**

```
$ sudo systemctl stop pbm-agent
```

2. Install new packages

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

3. Start **`pbm-agent`**

```
$ sudo systemctl start pbm-agent
```

Upgrade to a specific version

1. List available versions

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

Sample output

```
Available Packages
percona-backup-mongodb.x86_64    1.0-1.el7            tools-release-x86_64
percona-backup-mongodb.x86_64    1.0.0-1.el7          tools-release-x86_64
percona-backup-mongodb.x86_64    1.1.1-1.el7          tools-release-x86_64
percona-backup-mongodb.x86_64    1.1.3-1.el7          tools-release-x86_64
percona-backup-mongodb.x86_64    1.2.0-1.el7          tools-release-x86_64
percona-backup-mongodb.x86_64    1.2.1-1.el7          tools-release-x86_64
percona-backup-mongodb.x86_64    1.3.0-1.el7          tools-release-x86_64
percona-backup-mongodb.x86_64    1.3.1-1.el7          tools-release-x86_64
```

2. Stop **`pbm-agent`**:

```
$ sudo systemctl stop pbm-agent
```

3. Install a specific version packages. For example, to upgrade Percona Backup for MongoDB to version 1.1.3, use the following command:

```
$ sudo yum install percona-backup-mongodb-1.1.3-1.el7
```

4. Start **`pbm-agent`**:

```
$ sudo systemctl start pbm-agent
```

Part IV

Configuration

CHAPTER

FIVE

AUTHENTICATION

Percona Backup for MongoDB has no authentication and authorization subsystem of its own - it uses MongoDB's, i.e. `pbm` and **`pbm-agent`** only require a valid MongoDB connection URI string for the PBM user.

For the S3-compatible remote storage authentication config, see *Percona Backup for MongoDB config in a Cluster (or Non-sharded Replica set)*.

Create the PBM user

To run Percona Backup for MongoDB a user must be created in the `admin` db that has the role *grants* as shown below.

```
db.getSiblingDB("admin").createRole({ "role": "pbmAnyAction",
      "privileges": [
         { "resource": { "anyResource": true },
           "actions": [ "anyAction" ]
         }
      ],
      "roles": []
   });
db.getSiblingDB("admin").createUser({user: "pbmuser",
       "pwd": "secretpwd",
       "roles" : [
          { "db" : "admin", "role" : "readWrite", "collection": "" },
          { "db" : "admin", "role" : "backup" },
          { "db" : "admin", "role" : "clusterMonitor" },
          { "db" : "admin", "role" : "restore" },
          { "db" : "admin", "role" : "pbmAnyAction" }
       ]
    });
```

User name and password values and other options of the createUser command can be set as you require so long as the roles shown above are granted.

This user must be created on every replicaset, i.e. it must be created on the shard replicasets as well as the config server replicaset.

Note: In a cluster run *db.getSiblingDB("config").shards.find({}, {"host": true, "_id": false})* to list all the host+port lists for the shard replicasets. The replicaset name at the *front* of these "host" strings will have to be placed as a "/?replicaSet=xxxx" argument in the parameters part of the connection URI (see below).

MongoDB connection strings - A Reminder (or Primer)

Percona Backup for MongoDB uses MongoDB Connection URI strings to open MongoDB connections. Neither `pbm` or **`pbm-agent`** accept legacy-style command-line arguments for `--host`, `--port`, `--user`, `--password`, etc. as the `mongo` shell or `mongodump` command does.

```
$ pbm-agent --mongodb-uri "mongodb://pbmuser:secretpwd@localhost:27018/"
$ #Alternatively:
$ export PBM_MONGODB_URI="mongodb://pbmuser:secretpwd@localhost:27018/"
$ pbm-agent
```

```
$ pbm list --mongodb-uri "mongodb://pbmuser:secretpwd@mongocsvr1:27018,
↪mongocsvr2:27018,mongocsvr3:27018/?replicaSet=configrs"
$ #Alternatively:
$ export PBM_MONGODB_URI="mongodb://pbmuser:secretpwd@mongocsvr1:27018,
↪mongocsvr2:27018,mongocsvr3:27018/?replicaSet=configrs"
$ pbm list
```

The connection URI above is the format that MongoDB drivers accept universally since approximately the release time of MongoDB server v3.6. The `mongo` shell accepts it too since v4.0. Using a v4.0+ mongo shell is a recommended way to debug connection URI validity from the command line.

The MongoDB Connection URI specification includes several non-default options you may need to use. For example the TLS certificates/keys needed to connect to a cluster or non-sharded replicaset with network encryption enabled are "tls=true" plus "tlsCAFile" and/or "tlsCertificateKeyFile" (see tls options).

Technical note

As of v1.0 the driver used by Percona Backup for MongoDB is the official v1.1 mongo-go-driver.

The `pbm-agent` connection string

`pbm-agent` processes should connect to their localhost mongod with a standalone type of connection.

The `pbm` connection string

The `pbm` CLI will ultimately connect to the replica set with the *PBM control collections*.

- In a non-sharded replica set it is simply that replica set.
- In a cluster it is the config server replica set.

You do not necessarily have to provide that connection string. If you provide a connection to any live node (shard, configsvr, or non-sharded replicaset member), it will automatically determine the right hosts and establish a new connection to those instead.

Tip: When running `pbm` from an unsupervised script, we recommend using a replica set connection string. A standalone-style connection string will fail if that `mongod` host happens to be down temporarily.

CHAPTER

SIX

PERCONA BACKUP FOR MONGODB CONFIG IN A CLUSTER (OR NON-SHARDED REPLICA SET)

The configuration information is stored in a single document of the *admin.pbmConfig* collection. That single copy is shared by all the **pbm-agent** processes in a cluster (or non-sharded replica set), and can be read or updated using the `pbm` tool.

You can see the whole config by running *db.getSiblingDB("admin").pbmConfig.findOne()*. But you don't have to use the mongo shell; the `pbm` CLI has a "config" subcommand to read and update it.

Percona Backup for MongoDB config contains the following settings:

- *Remote backup storage* configuration is available as of v1.0 or v1.1
- *Point-in-Time Recovery* configuration is available as of v1.3.0
- *Backup restore options* are available as of v1.3.2

Insert the whole Percona Backup for MongoDB config from a YAML file

If you are initializing a cluster or non-sharded replica set for the first time, it is simplest to write the whole config as YAML file and use the **pbm config --file** method to upload all the values in one command.

```
$ pbm config --file pbm_config.yaml
```

Execute whilst connecting to config server replica set if it is a cluster. Otherwise just connect to the non-sharded replica set as normal. (See *MongoDB connection strings - A Reminder (or Primer)* if you are not familiar with MongoDB connection strings yet.) For more information about available config file options, see *Configuration file options*.

Run **pbm config --list** to see the whole config. (Sensitive fields such as keys will be redacted.)

Accessing or updating single config values

You can set a single value at time. For nested values use dot-concatenated key names as shown in the following example:

```
$ pbm config --set storage.s3.bucket="operator-testing"
```

To list a single value you can specify just the key name by itself and the value will be returned (if set)

```
$ pbm config storage.s3.bucket
operator-testing
$ pbm config storage.s3.INVALID-KEY
Error: unable to get config key: invalid config key
```

CHAPTER

SEVEN

REMOTE BACKUP STORAGE

Percona Backup for MongoDB supports the following types of remote backup storages:

- S3-compatible storage
- Filesystem type storage

S3 compatible storage

Percona Backup for MongoDB should work with other S3-compatible storages but was only tested with the following ones:

- Amazon Simple Storage Service
- Google Cloud Storage
- MinIO

Starting from v1.3.2, Percona Backup for MongoDB supports *server-side encryption* for *S3 buckets* with customer managed keys stored in AWS KMS (AWS Key Management Service).

See also:

Protecting Data Using Server-Side Encryption with CMKs Stored in AWS Key Management Service (SSE-KMS)

Remote Filesystem Server Storage

This storage must be a remote fileserver mounted to a local directory. It is the responsibility of the server administrators to guarantee that the same remote directory is mounted at exactly the same local path on all servers in the MongoDB cluster or non-sharded replica set.

Warning: Percona Backup for MongoDB uses the directory as if it were any normal directory, and does not attempt to confirm it is mounted from a remote server. If the path is accidentally a normal local directory, errors will eventually occur, most likely during a restore attempt. This will happen because **`pbm-agent`** processes of other nodes in the same replica set can't access backup archive files in a normal local directory on another server.

Local Filesystem Storage

This cannot be used except if you have a single-node replica set. (See the warning note above as to why). We recommend using any object store you might be already familiar with for testing. If you don't have an object store yet, we recommend using MinIO for testing as it has simple setup. If you plan to use a remote filesytem-type backup server, please see the "Remote Filesystem Server Storage" above.

Example config files

Providing remote storage information within a YAML config file is the easiest way to configure the storage for Percona Backup for MongoDB. For how to insert the config file, see *Insert the whole Percona Backup for MongoDB config from a YAML file*.

S3-compatible remote storage

Amazon Simple Storage Service

```
storage:
  type: s3
  s3:
    region: us-west-2
    bucket: pbm-test-bucket
    prefix: data/pbm/backup
    credentials:
      access-key-id: <your-access-key-id-here>
      secret-access-key: <your-secret-key-here>
    serverSideEncryption:
      sseAlgorithm: aws:kms
      kmsKeyID: <your-kms-key-here>
```

GCS

```
storage:
 type: s3
 s3:
     region: us-east1
     bucket: pbm-testing
     prefix: pbm/test
     endpointUrl: https://storage.googleapis.com
     credentials:
       access-key-id: <your-access-key-id-here>
       secret-access-key: <your-secret-key-here>
```

MinIO

```
storage:
  type: s3
  s3:
    endpointUrl: "http://localhost:9000"
    region: my-region
    bucket: pbm-example
    prefix: data/pbm/test
    credentials:
      access-key-id: <your-access-key-id-here>
      secret-access-key: <your-secret-key-here>
```

Remote filesystem server storage

```
storage:
  type: filesystem
  filesystem:
    path: /data/local_backups
```

For the description of available configuration options, see *Configuration file options*.

Part V

Usage

CHAPTER

EIGHT

RUNNING PERCONA BACKUP FOR MONGODB

Please see *Authentication* if you have not already. This will explain the MongoDB user that needs to be created, and the connection method used by Percona Backup for MongoDB.

Initial setup

1. Determine the right MongoDB connection string for the `pbm` CLI. (See *MongoDB connection strings - A Reminder (or Primer)*)
2. Use the `pbm` CLI to insert the config (especially the Remote Storage location and credentials information). See *Insert the whole Percona Backup for MongoDB config from a YAML file*
3. Start (or restart) the **`pbm-agent`** processes for all mongod nodes.

Start the `pbm-agent` processes

After installing **`pbm-agent`** on the all the servers that have mongod nodes make sure one instance of it is started for each mongod node.

E.g. Imagine you put configsvr nodes (listen port 27019) colocated on the same servers as the first shard's mongod nodes (listen port 27018, replica set name "sh1rs"). In this server there should be two **`pbm-agent`** processes, one connected to the shard (e.g. "mongodb://username:password@localhost:27018/") and one to the configsvr node (e.g. "mongodb://username:password@localhost:27019/").

It is best to use the packaged service scripts to run **pbm-agent**. After adding the database connection configuration for them (see *Configuring service init scripts*), you can start the **pbm-agent** service as below:

```
$ sudo systemctl start pbm-agent
$ sudo systemctl status pbm-agent
```

For reference an example of starting pbm-agent manually is shown below. The output is redirected to a file and the process is backgrounded. Alternatively you can run it on a shell terminal temporarily if you want to observe and/or debug the startup from the log messages.

```
$ nohup pbm-agent --mongodb-uri "mongodb://username:password@localhost:27018/" > /
↪data/mdb_node_xyz/pbm-agent.$(hostname -s).27018.log 2>&1 &
```

Tip: Running as the `mongod` user would be the most intuitive and convenient way. But if you want it can be another user.

When a message *"pbm agent is listening for the commands"* is printed to the **pbm-agent** log file it confirms it connected to its mongod successfully.

How to see the pbm-agent log

With the packaged systemd service the log output to stdout is captured by systemd's default redirection to systemd-journald. You can view it with the command below. See *man journalctl* for useful options such as '–lines', '–follow', etc.

```
~$ journalctl -u pbm-agent.service
-- Logs begin at Tue 2019-10-22 09:31:34 JST. --
Jan 22 15:59:14 akira-x1 systemd[1]: Started pbm-agent.
Jan 22 15:59:14 akira-x1 pbm-agent[3579]: pbm agent is listening for the commands
...
...
```

If you started pbm-agent manually see the file you redirected stdout and stderr to.

Running Percona Backup for MongoDB

Provide the MongoDB URI connection string for `pbm`. This allows you to call `pbm` commands without the `--mongodb-uri` flag.

Use the following command:

```
export PBM_MONGODB_URI="mongodb://pbmuser:secretpwd@localhost:27018/"
```

For more information what connection string to specify, refer to *The pbm connection string* section.

Running `pbm` commands

`pbm` is the command line utility to control the backup system.

Configuring a Remote Storage for Backup and Restore Operations

This must be done once, at installation or re-installation time, before backups can be listed, made, or restored. To configure remote storage, see *Percona Backup for MongoDB config in a Cluster (or Non-sharded Replica set)* and *Remote backup storage*.

Listing all backups

```
$ pbm list
```

Sample output

```
2020-07-10T07:04:14Z
2020-07-09T07:03:50Z
2020-07-08T07:04:21Z
2020-07-07T07:04:18Z
```

Starting a backup

```
pbm backup
```

Starting a backup with compression

```
pbm backup --compression=s2
```

`s2` is the default compression type. Other supported compression types are: `gzip`, `snappy`, `lz4`, `pgzip`. The `none` value means no compression is done during backup.

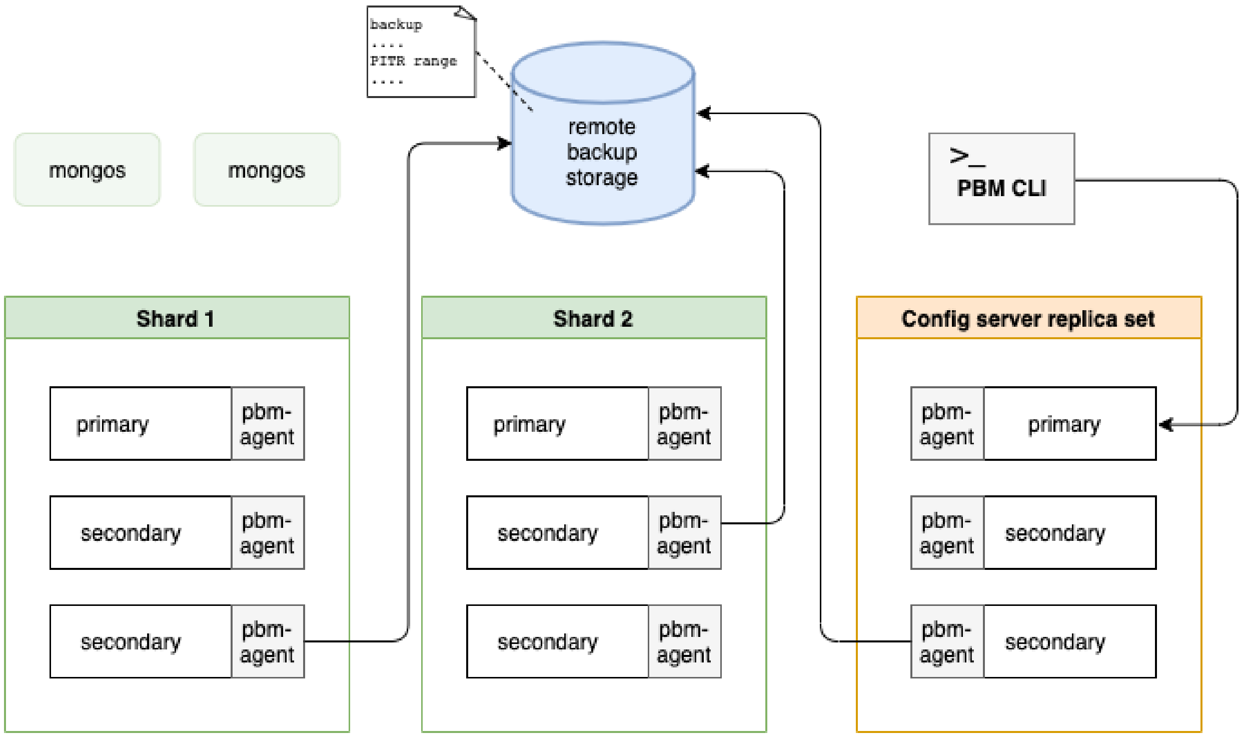
Backup in sharded clusters

Important: For PBM v1.0 (only) before running `pbm backup` on a cluster stop the balancer.

In sharded clusters, one of `pbm-agent` processes for every shard and the config server replica set writes backup snapshots and *oplog slices* into the remote backup storage directly. To learn more about oplog slicing, see *Point-in-Time Recovery*.

The `mongos` nodes are not involved in the backup process.

The following diagram illustrates the backup flow.



Checking an in-progress backup

Run the `pbm list` command and you will see the running backup listed with a 'In progress' label. When that is absent the backup is complete.

Restoring a backup

To restore a backup that you have made using `pbm backup`, use the `pbm restore` command supplying the time stamp of the backup that you intend to restore.

Important: Consider these important notes on restore operation:

1. Percona Backup for MongoDB is designed to be a full-database restore tool. As of version <=1.x it performs a full all-databases, all collections restore and does not offer an option to restore only a subset of collections in the backup, as MongoDB's `mongodump` tool does. But to avoid surprising `mongodump` users, as of versions 1.x Percona Backup for MongoDB replicates mongodump's behavior to only drop collections in the backup. It does not drop collections that are created new after the time of the backup and before the restore. Run a `db.dropDatabase()` manually in all non-system databases (i.e. all databases except "local", "config"

and "admin") before running |pbm-restore| if you want to guarantee that the post-restore database only includes collections that are in the backup.

2. Whilst the restore is running, prevents clients from accessing the database. The data will naturally be incomplete whilst the restore is in progress, and writes the clients make cause the final restored data to differ from the backed-up data.
3. If you enabled *Point-in-Time Recovery*, disable it before running `pbm restore`. This is because Point-in-Time Recovery incremental backups and restore are incompatible operations and cannot be run together.

```
$ pbm restore 2019-06-09T07:03:50Z
```

New in version 1.3.2: The Percona Backup for MongoDB config includes the restore options to adjust the memory consumption by the **`pbm-agent`** in environments with tight memory bounds. This allows preventing out of memory errors during the restore operation.

```
restore:
  batchSize: 500
  numInsertionWorkers: 10
```

The default values were adjusted to fit the setups with the memory allocation of 1GB and less for the agent.

Note: The lower the values, the less memory is allocated for the restore. However, the performance decreases too.

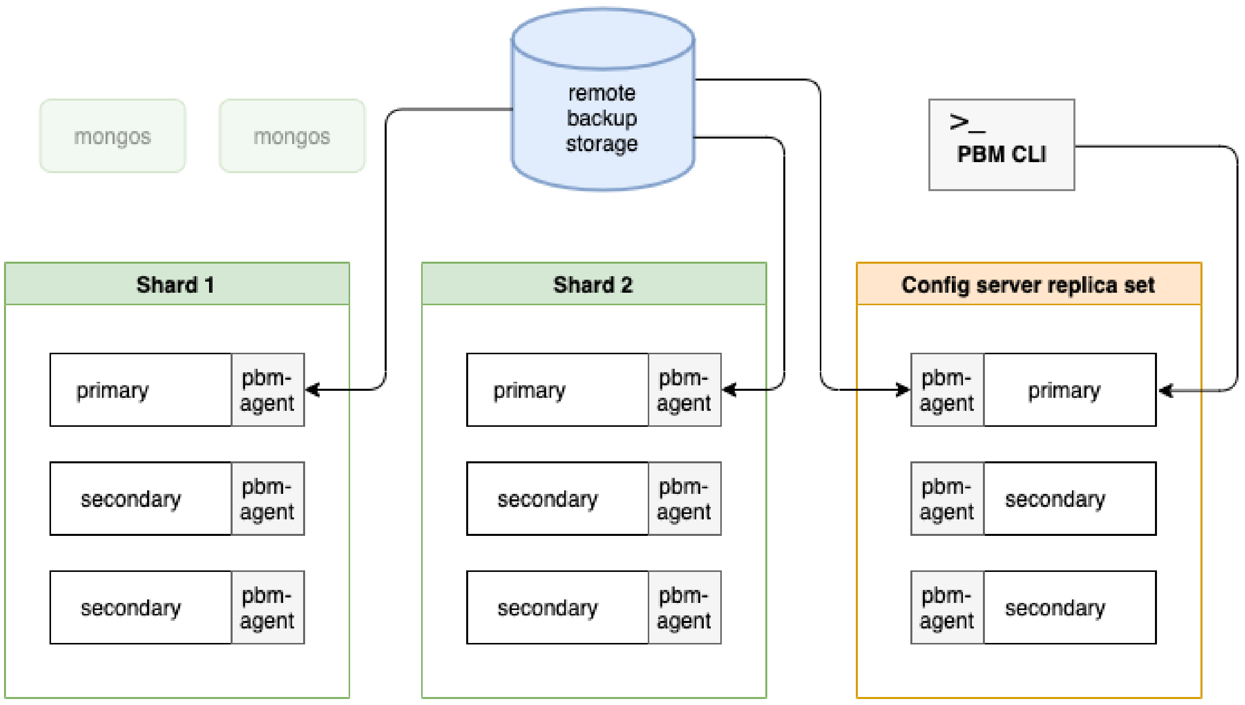
Restoring a backup in sharded clusters

Important: As preconditions for restoring a backup in a sharded cluster, complete the following steps:

1. Stop the balancer.
2. Shut down all `mongos` nodes to stop clients from accessing the database while restore is in progress. This ensures that the final restored data doesn't differ from the backed-up data.
3. Disable point-in-time recovery if it is enabled. To learn more about point-in-time recovery, see *Point-in-Time Recovery*.

Note that you can restore a sharded backup only into a sharded environment. It can be your existing cluster or a new one. To learn how to restore a backup into a new environment, see *Restoring a backup into a new environment*.

During the restore, `pbm-agents` write data to primary nodes in the cluster. The following diagram shows the restore flow.



After a cluster's restore is complete, restart all `mongos` nodes to reload the sharding metadata.

Restoring a backup into a new environment

To restore a backup from one environment to another, consider the following key points about the destination environment:

- Replica set names (both the config servers and the shards) in your new destination cluster and in the cluster that was backed up must be exactly the same.
- Percona Backup for MongoDB configuration in the new environment must point to the same remote storage that is defined for the original environment, including the authentication credentials if it is an object store. Once you run `pbm list` and see the backups made from the original environment, then you can run the `pbm restore` command.

  Of course, make sure not to run `pbm backup` from the new environment whilst the Percona Backup for MongoDB config is pointing to the remote storage location of the original environment.

Canceling a backup

You can cancel a running backup if, for example, you want to do another maintenance and don't want to wait for the large backup to finish first.

To cancel the backup, use the `pbm cancel-backup` command.

```
$ pbm cancel-backup
Backup cancellation has started
```

After the command execution, the backup is marked as canceled in the `pbm list` output:

```
$ pbm list
...
2020-04-30T18:05:26Z  Canceled at 2020-04-30T18:05:37Z
```

Deleting backups

Use the `pbm delete-backup` command to delete a specified backup or all backups older than the specified time.

The command deletes the backup regardless of the remote storage used: either S3-compatible or a filesystem-type remote storage.

Note: You can only delete a backup that is not running (has the "done" or the "error" state).

To delete a backup, specify the `<backup_name>` from the the `pbm list` output as an argument.

```
$ #Get the backup name
$ pbm list
Backup history:
  2020-04-20T10:55:42Z
  2020-04-20T13:07:34Z
  2020-04-20T13:13:20Z
  2020-04-20T13:45:59Z

$ #Delete a backup
$ pbm delete-backup 2020-04-20T13:45:59Z
```

By default, the `pbm delete-backup` command asks for your confirmation to proceed with the deletion. To bypass it, add the `-f` or `--force` flag.

```
$ pbm delete-backup --force 2020-04-20T13:45:59Z
```

To delete backups that were created before the specified time, pass the `--older-than` flag to the `pbm delete-backup` command. Specify the timestamp as an argument for the `pbm delete-backup` command in the following format:

- `%Y-%M-%DT%H:%M:%S` (e.g. 2020-04-20T13:13:20) or
- `%Y-%M-%D` (e.g. 2020-04-20).

```
$ #Get the backup name
$ pbm list
Backup history:
  2020-04-20T20:55:42Z
  2020-04-20T23:47:34Z
  2020-04-20T23:53:20Z
  2020-04-21T02:16:33Z
$ #Delete backups created before the specified timestamp
$ pbm delete-backup -f --older-than 2020-04-21
Backup history:
  2020-04-21T02:16:33Z
```

CHAPTER

NINE

POINT-IN-TIME RECOVERY

Point-in-Time Recovery is restoring a database up to a specific moment. Point-in-Time Recovery includes restoring the data from a backup snapshot and replaying all events that occurred to this data up to a specified moment from *oplog slices*. Point-in-Time Recovery helps you prevent data loss during a disaster such as crashed database, accidental data deletion or drop of tables, unwanted update of multiple fields instead of a single one.

Point-in-Time Recovery is available in Percona Backup for MongoDB starting from v1.3.0. Point-in-Time Recovery is enabled via the `pitr.enabled` config option.

```
$ pbm config --set pitr.enabled=true
```

Incremental backups

When Point-in-Time Recovery is enabled, **`pbm-agent`** periodically saves consecutive slices of the *oplog*. A method similar to the way replica set nodes elect a new primary is used to select the **`pbm-agent`** that saves the oplog slices. (Find more information in *pbm-agent*.)

A slice covers a 10 minute span of oplog events. It can be shorter if Point-in-Time Recovery is disabled or interrupted by the start of a backup snapshot operation.

The oplog slices are stored in the `pbmPitr` subdirectory in the *remote storage defined in the config*. A slice name reflects the start and end time this slice covers.

The `pbm list` output includes both backup snapshots and valid time ranges for recovery. It also shows the Point-in-Time Recovery status.

```
$ pbm list

Backup snapshots:
     2020-07-10T12:19:10Z
     2020-07-14T10:44:44Z
     2020-07-14T14:26:20Z
     2020-07-17T16:46:59Z
PITR <on>:
     2020-07-14T14:26:40 - 2020-07-16T17:27:26
     2020-07-17T16:47:20 - 2020-07-17T16:57:55
```

Note: If you just enabled Point-in-Time Recovery, the time range list in the `pbm list` output is empty. It requires 10 minutes for the first chunk to appear in the list.

Restore to the point in time

A restore and Point-in-Time Recovery incremental backups are incompatible operations and cannot be run simultaneously. You must disable Point-in-Time Recovery before restoring a database:

```
$ pbm config --set pitr.enabled=false
```

Run `pbm restore` and specify the time stamp from the valid range:

```
$ pbm restore --time="2020-07-14T14:27:04"
```

See also:

Restoring a backup.

A restore operation changes the time line of oplog events. Therefore, all oplog slices made after the restore time stamp and before the last backup become invalid. After the restore is complete, make a new backup to serve as the starting point for oplog updates:

```
$ pbm backup
```

Re-enable Point-in-Time Recovery to resume saving oplog slices:

```
$ pbm config --set pitr.enabled=true
```

Delete a backup

When you *delete a backup*, all oplog slices that relate to this backup will be deleted too. For example, you delete a backup snapshot 2020-07-24T18:13:09 while there is another snapshot 2020-08-05T04:27:55 created after it. **`pbm-agent`** deletes only oplog slices that relate to 2020-07-24T18:13:09.

The same applies if you delete backups older than the specified time.

Note: When Point-in-Time Recovery is enabled, the most recent backup snapshot and oplog slices that relate to it won't be deleted.

Part VI

Troubleshooting

CHAPTER

TEN

TROUBLESHOOTING PERCONA BACKUP FOR MONGODB

Percona Backup for MongoDB provides troubleshooting tools to operate data backups.

pbm-speed-test

pbm-speed-test allows field-testing compression and backup upload speed. You can use it:

- to check performance before starting a backup;
- to find out what slows down the running backup.

The full set of commands:

```
$ /usr/bin/pbm-speed-test --help
usage: pbm-speed-test --mongodb-uri=MONGODB-URI [<flags>] <command> [<args> ...]

Percona Backup for MongoDB compression and upload speed test

Flags:
      --help                       Show context-sensitive help (also try
                                   --help-long and --help-man).
      --mongodb-uri=MONGODB-URI   MongoDB connection string
  -c, --sample-collection=SAMPLE-COLLECTION
                                   Set collection as the data source
  -s, --size-gb=SIZE-GB            Set data size in GB. Default 1
      --compression=s2             Compression type
                                   <none>/<gzip>/<snappy>/<lz4>/<s2>/<pgzip>

Commands:
  help [<command>...]
    Show help.

  compression
    Run compression test

  storage
    Run storage test
  version [<flags>]
    PBM version info
```

By default, **pbm-speed-test** operates with fake semi random data documents. To run **pbm-speed-test** on a real collection, you require a connection to the MongoDB. See *MongoDB connection strings - A Reminder (or Primer)* for details.

Compression test

```
/usr/bin/pbm-speed-test compression --compression=s2 --size-gb 10
Test started ....
10.00GB sent in 8s.
Avg upload rate = 1217.13MB/s.
```

`pbm-speed-test compression` uses the compression library from the config file and sends a fake semi random data document (1 GB by default) to the black hole storage. (Use the `pbm config` command to change the compression library).

To test compression on a real collection, pass the `--sample-collection` flag with the <my_db.my_collection> value.

Run `pbm-speed-test compression --help` for the full set of supported flags:

```
/usr/bin/pbm-speed-test compression --help
usage: pbm-speed-test compression

Run compression test

Flags:
      --help                     Show context-sensitive help (also try
                                 --help-long and --help-man).
      --mongodb-uri=MONGODB-URI  MongoDB connection string
  -c, --sample-collection=SAMPLE-COLLECTION
                                 Set collection as the data source
  -s, --size-gb=SIZE-GB          Set data size in GB. Default 1
      --compression=s2           Compression type
                                 <none>/<gzip>/<snappy>/<lz4>/<s2>/<pgzip>
```

Upload speed test

```
/usr/bin/pbm-speed-test storage --compression=s2
Test started
1.00GB sent in 1s.
Avg upload rate = 1744.43MB/s.
```

`pbm-speed-test storage` sends the semi random data (1 GB by default) to the remote storage defined in the config file. Pass the `--size-gb` flag to change the data size.

To run the test with the real collection's data instead of the semi random data, pass the `--sample-collection` flag with the <my_db.my_collection> value.

Run `pbm-speed-test storage --help` for the full set of available flags:

```
/usr/bin/pbm-speed-test storage --help
usage: pbm-speed-test storage

Run storage test

Flags:
      --help                     Show context-sensitive help (also try --help-long␣
↪and --help-man).
      --mongodb-uri=MONGODB-URI  MongoDB connection string
  -c, --sample-collection=SAMPLE-COLLECTION
                                 Set collection as the data source
  -s, --size-gb=SIZE-GB          Set data size in GB. Default 1
      --compression=s2           Compression type <none>/<gzip>/<snappy>/<lz4>/<s2>/
↪<pgzip>
```

Backup progress logs

If you have a large backup you can track backup progress in **pbm-agent** logs. A line is appended every minute showing bytes copied vs. total size for the current collection.

```
# Start a backup
$ pbm backup
#Check backup progress
journalctl -u pbm-agent.service
2020/05/06 21:31:12 Backup 2020-05-06T18:31:12Z started on node rs2/localhost:28018
2020-05-06T21:31:14.797+0300 writing admin.system.users to archive on stdout
2020-05-06T21:31:14.799+0300 done dumping admin.system.users (2 documents)
2020-05-06T21:31:14.800+0300 writing admin.system.roles to archive on stdout
2020-05-06T21:31:14.807+0300 done dumping admin.system.roles (1 document)
2020-05-06T21:31:14.807+0300 writing admin.system.version to archive on stdout
2020-05-06T21:31:14.815+0300 done dumping admin.system.version (3 documents)
2020-05-06T21:31:14.816+0300 writing test.testt to archive on stdout
2020-05-06T21:31:14.829+0300 writing test.testt2 to archive on stdout
2020-05-06T21:31:14.829+0300 writing config.cache.chunks.config.system.sessions to␣
↪archive on stdout
2020-05-06T21:31:14.832+0300 done dumping config.cache.chunks.config.system.sessions␣
↪(1 document)
2020-05-06T21:31:14.834+0300 writing config.cache.collections to archive on stdout
2020-05-06T21:31:14.835+0300 done dumping config.cache.collections (1 document)
2020/05/06 21:31:24 [##......................]   test.testt  130841/1073901  (12.2%)
2020/05/06 21:31:24 [##########..............]  test.testt2   131370/300000  (43.8%)
2020/05/06 21:31:24
2020/05/06 21:31:34 [#####...................]   test.testt  249603/1073901  (23.2%)
2020/05/06 21:31:34 [###################.....]  test.testt2   249603/300000  (83.2%)
2020/05/06 21:31:34
2020/05/06 21:31:37 [########################]  test.testt2  300000/300000  (100.0%)
```

Part VII

Uninstall Percona Backup for MongoDB

CHAPTER

ELEVEN

UNINSTALLING PERCONA BACKUP FOR MONGODB

To uninstall Percona Backup for MongoDB perform the following steps:

1. Check no backups are currently in progress in the output of `pbm list`.
2. Before the next 2 steps make sure you know where the remote backup storage is, so you can delete backups made by Percona Backup for MongoDB. If it is S3-compatible object storage you will need to use another tool such as Amazon AWS's "aws s3", Minio's `mc`, the web AWS Management Console, etc. to do that once Percona Backup for MongoDB is uninstalled. Don't forget to note the connection credentials before they are deleted too.
3. Uninstall the **`pbm-agent`** and `pbm` executables. If you installed using a package manager, see *Installing Percona Backup for MongoDB* for relevant package names and commands for your OS distribution.
4. Drop the *PBM control collections*.
5. Drop the PBM database user. If this is a cluster the dropUser command will need to be run on each shard as well as in the config server replica set.
6. (Optional) Delete the backups from the remote backup storage.

Part VIII

Reference

CHAPTER

TWELVE

RELEASE NOTES

Percona Backup for MongoDB 1.3.3

Date November 4, 2020

Installation Installing Percona Backup for MongoDB

Percona Backup for MongoDB is a distributed, low-impact solution for consistent backups of MongoDB sharded clusters and replica sets. This is a tool for creating consistent backups across a MongoDB sharded cluster (or a single replica set), and for restoring those backups to a specific point in time.

Bugs Fixed

- PBM-575: `mongodump` connects to the primary node

Percona Backup for MongoDB 1.3.2

Date October 14, 2020

Installation Installing Percona Backup for MongoDB

Percona Backup for MongoDB is a distributed, low-impact solution for consistent backups of MongoDB sharded clusters and replica sets. This is a tool for creating consistent backups across a MongoDB sharded cluster (or a single replica set), and for restoring those backups to a specific point in time.

New Features

- PBM-426: Add AWS KMS key encryption/decryption for S3 buckets

  Config format

  ```
  storage:
    s3:
      serverSideEncryption:
        sseAlgorithm: "aws:kms"
        kmsKeyID: "........"
  ```

  (Thanks to user pedroalb for reporting this issue)

Improvements

- PBM-568: Print uploadPartSize value to log during backup
- PBM-560: Use s2 compression as default for `pbm-speed-test` instead of gzip

Bugs Fixed

- PBM-485: Fix backups to S3 failing with `MaxUploadParts` limit by auto-adjusting `uploadPartSize` value (Thanks to user pedroalb for reporting this issue)
- PBM-559: pbm-agent runs out of memory while doing restore of large backup (Thanks to user Simon Bernier St-Pierre for reporting this issue)
- PBM-562: Correct calculation of available PITR time ranges by pbm list
- PBM-561: Fix setting of numeric options in config
- PBM-547: Allow deleting backups from local filesystem by moving delete operations to pbm-agents

Percona Backup for MongoDB 1.3.1

Date September, 2020

Installation Installing Percona Backup for MongoDB

Percona Backup for MongoDB is a distributed, low-impact solution for consistent backups of MongoDB sharded clusters and replica sets. This is a tool for creating consistent backups across a MongoDB sharded cluster (or a single replica set), and for restoring those backups to a specific point in time.

Bugs Fixed

- PBM-542: Fix backup folder permissions on filesystem storage for Point-in-Time recovery

Percona Backup for MongoDB 1.3.0

Date August 26, 2020

Installation Installing Percona Backup for MongoDB

Percona Backup for MongoDB is a distributed, low-impact solution for consistent backups of MongoDB sharded clusters and replica sets. This is a tool for creating consistent backups across a MongoDB sharded cluster (or a single replica set), and for restoring those backups to a specific point in time.

New Features

- PBM-455: Add oplog archiver thread for PITR
- PBM-491: Modify “pbm restore” to accept arbitrary point in time when PITR oplog archives available

Improvements

- PBM-526: Add pbm version information to the backup metadata

Percona Backup for MongoDB 1.2.1

Date July 27, 2020

Installation Installing Percona Backup for MongoDB

Percona Backup for MongoDB is a distributed, low-impact solution for consistent backups of MongoDB sharded clusters and replica sets. This is a tool for creating consistent backups across a MongoDB sharded cluster (or a single replica set).

Bugs Fixed

- PBM-509: Include "pbm-speed-test" binary for debian packages

Percona Backup for MongoDB 1.2.0

Date May 13, 2020

Installation Installing Percona Backup for MongoDB

Percona Backup for MongoDB is a distributed, low-impact solution for consistent backups of MongoDB sharded clusters and replica sets. This is a tool for creating consistent backups across a MongoDB sharded cluster (or a single replica set).

New Features

- PBM-348: Add ability to delete old backups
- PBM-447: pbm-speed-test: Add a tool to field-test compression and upload speeds

Improvements

- PBM-431: Raise dump output speed through compression tuning, parallelization
- PBM-461: s2 is set as the default compression mechanism
- PBM-429: Periodic backup progress messages added to pbm-agent logs
- PBM-140: Added ability to cancel a backup

Bugs Fixed

- PBM-451: Resync didn't work if storage type was set to filesystem

Percona Backup for MongoDB 1.1.3

Date April 14, 2020

Installation *Installing Percona Backup for MongoDB*

Improvements

- PBM-424: Remove the `--mongodb-uri` arg from `pbm-agent.service` unit file
- PBM-419: Resolve restore-blocking issues related to admin.system.version
- PBM-417: Improve pbm control collection etc. metadata for restores

Bugs Fixed

- PBM-425: pbm-agent could fail when restoring
- PBM-430: S3 store resync didn't work if the store had a prefix
- PBM-438: `pbm list --size=5` worked in reverse

Percona Backup for MongoDB 1.1.1

Date January 31, 2020

Installation *Installing Percona Backup for MongoDB*

Percona Backup for MongoDB is a distributed, low-impact solution for consistent backups of MongoDB sharded clusters and replica sets. This is a tool for creating consistent backups across a MongoDB sharded cluster (or a single replica set), and for restoring those backups to a specific point in time. The project was inspired by (and intends to replace) the Percona-Lab/mongodb_consistent_backup tool.

Percona Backup for MongoDB supports Percona Server for MongoDB or MongoDB Community Server version 3.6 or higher with MongoDB replication enabled. Binaries for the supported platforms as well as the tarball with source code are available from the Percona Backup for MongoDB download page. For more information about Percona Backup for MongoDB and the installation steps, see the *documentation*.

Bugs Fixed

- PBM-407: Very large collections experienced timeout due to full-collection scan for a preliminary count
- PBM-414: The upload on Google cloud storage was broken with "InvalidArgument: Invalid argument. status code: 400"
- PBM-409: Restore failed with "incompatible auth version with target server"

Percona Backup for MongoDB 1.1.0

Percona is happy to announce the release of Percona Backup for MongoDB 1.1.0 on January 16, 2020.

Percona Backup for MongoDB is a distributed, low-impact solution for consistent backups of MongoDB sharded clusters and replica sets. This is a tool for creating consistent backups across a MongoDB sharded cluster (or a single replica set), and for restoring those backups to a specific point in time. The project was inspired by (and intends to replace) the Percona-Lab/mongodb_consistent_backup tool.

Percona Backup for MongoDB supports Percona Server for MongoDB or MongoDB Community Server version 3.6 or higher with MongoDB replication enabled. Binaries for the supported platforms as well as the tarball with source code are available from the Percona Backup for MongoDB download page. For more information about Percona Backup for MongoDB and the installation steps, see the *documentation*.

Percona Backup for MongoDB 1.1.0 introduces the new `pbm config` command to enable configuring the store from the command line in addition to the configuration file. This command effectively replaces *pbm store* which was only able to read store configuration from the configuration file.

```
$ pbm config --set storage.s3.bucket="operator-testing"
```

New Features

- PBM-344: New *pbm config* command to support configuring the store from the command line.

Improvements

- PBM-361: Improved the processing of timestamps when using oplog.

Bugs Fixed

- PBM-214: `pbm-agent` could crash with restore command running forever, if the primary node became unavailable during the *restore* operation.
- PBM-279: `pbm-agent` could be started with an invalid config file.
- PBM-338: Backups that failed could appear in the output of the *pbm list* command.
- PBM-362: The `pbm backup` could fail when called from the primary node if there were no healthy secondaries.
- PBM-369: ReplicaSets could not establish connections when TLS was used in the cluster.

Percona Backup for MongoDB 1.0.0

Percona is happy to announce the GA release of our latest software product Percona Backup for MongoDB 1.3 on September 19, 2019.

Percona Backup for MongoDB is a distributed, low-impact solution for consistent backups of MongoDB sharded clusters and replica sets. This is a tool for creating consistent backups across a MongoDB sharded cluster (or a single replica set), and for restoring those backups to a specific point in time. The project was inspired by (and intends to replace) the Percona-Lab/mongodb_consistent_backup tool.

Percona Backup for MongoDB supports Percona Server for MongoDB or MongoDB Community Server version 3.6 or higher with MongoDB replication enabled. Binaries for the supported platforms as well as the tarball with source code are available from the Percona Backup for MongoDB download page. For more information about Percona Backup for MongoDB and the installation steps, see the *documentation*.

Percona Backup for MongoDB 1.0.0 features the following:

- The architecture and the authentication of Percona Backup for MongoDB have been simplified compared to the previous release.

- Stores backup data on Amazon Simple Storage Service or compatible storages, such as MinIO.
- The output of `pbm list` shows all backups created from the connected MongoDB sharded cluster or replica set.

Percona Backup for MongoDB 0.5.0

Percona is pleased to announce the early release of Percona Backup for MongoDB 0.5.0 of our latest software product on June 17, 2019. The GA version of Percona Backup for MongoDB is scheduled to be released later in 2019.

Percona Backup for MongoDB is a distributed, low-impact solution for consistent backups of MongoDB sharded clusters and replica sets. This is a tool for creating consistent backups across a MongoDB sharded cluster (or a single replica set), and for restoring those backups to a specific point in time. Percona Backup for MongoDB uses a distributed client/server architecture to perform backup/restore actions.

The project was inspired by (and intends to replace) the Percona-Lab/mongodb_consistent_backup tool.

Percona Backup for MongoDB supports Percona Server for MongoDB or MongoDB Community Server version 3.6 or higher with MongoDB replication enabled. Binaries for the supported platforms as well as the tarball with source code are available from the Percona Backup for MongoDB download page <https://www.percona.com/downloads/percona-backup-mongodb/LATEST/>‘_. For more information about Percona Backup for MongoDB and the installation steps, see the documentation.

Percona Backup for MongoDB 0.5.0 features the following:

- Enables storing backup metadata on Amazon Simple Storage Service storages.
- The API of Percona Backup for MongoDB introduces HTTP basic authentication to prevent an unauthorized user from running backups or restoring data if they manage to access the API port.
- To optimize the usage of network resources, the pbm-agent on `mongos` is not needed any more and backup-coordinator automatically establishes connection to the appropriate `mongos` instance.
- The output of `pbmctl list nodes` now includes the replica set name and informs the backup status of the node.

Percona doesn’t recommend this release for production as its API and configuration fields are still likely to change. It only features a basic API level security. Please report any bugs you encounter in our bug tracking system.

New Features and Improvements

- 93: Support storage of backup metadata on AWS S3.
- 99: **`pbm-agent`** is deprecated on `mongos`.
- 105: Log a warning if a Primary node-type is used for a backup
- 122: Include the replica set name to the output of `pmbctl list nodes`
- 130: Add HTTP Basic Authentication to gRPC servers (API and RPC)
- 139: Support listing backup status in the output of `pmbctl list nodes`
- 170: Enable setting the ‘stopOnError’ attribute in `mongorestore` to ensure consistency of the data being restored.

CHAPTER

THIRTEEN

CONFIGURATION FILE OPTIONS

This page describes configuration file options available in Percona Backup for MongoDB. For how to use configuration file, see *Percona Backup for MongoDB config in a Cluster (or Non-sharded Replica set)*.

Remote backup storage options

Percona Backup for MongoDB supports two types of remote storages: S3-compatible storages and filesystem. Percona Backup for MongoDB should work with other S3-compatible storages but was only tested with the following ones:

- Amazon Simple Storage Service,
- Google Cloud Storage,
- MinIO.

S3 type storage options

```
storage:
  type: s3
  s3:
    region: <string>
    bucket: <string>
    prefix: <string>
    credentials:
      access-key-id: <your-access-key-id-here>
      secret-access-key: <your-secret-key-here>
    serverSideEncryption:
      sseAlgorithm: aws:kms
      kmsKeyID: <your-kms-key-here>
```

option `storage.s3.provider`

Type string

Required NO

The storage provider's name. Supported values: aws, gcs

option `storage.s3.bucket`

Type string

Required YES

The name of the storage *bucket*. See the AWS Bucket naming rules and GCS bucket naming guidelines for bucket name requirements

option `storage.s3.region`

Type string

Required YES (for AWS and GCS)

The location of the storage bucket. Use the AWS region list and GCS region list to define the bucket region

option `storage.s3.prefix`

Type string

Required NO

The path to the data directory on the bucket. If undefined, backups are stored in the bucket root directory

option `storage.s3.endpointUrl`

Type string

Required YES (for MinIO and GCS)

The URL to access the bucket. The default value for GCS is `https://storage.googleapis.com`

option `storage.s3.credentials.access-key-id`

Type string

Required YES

Your access key to the storage bucket

option `storage.s3.credentials.secret-access-key`

Type string

Required YES

The key to sign your programmatic requests to the storage bucket

option `storage.s3.uploadPartSize`

Type int

Required NO

The size of data chunks to be uploaded to the bucket. Default: 10MB.

Percona Backup for MongoDB automatically increases the `uploadPartSize` value if the size of the file to be uploaded exceeds the max allowed file size. (The max allowed file size is calculated with the default values of uploadPartSize * maxUploadParts and is appr. 97,6 GB)

The `uploadPartSize` value is printed in the *pbm-agent log*.

By setting this option, you can manually adjust the size of data chunks if Percona Backup for MongoDB failed to do it for some reason. The defined `uploadPartSize` value overrides the default value and is used for calculating the max allowed file size

Server-side encryption options

option `serverSideEncryption.sseAlgorythm`

Type string

The key management mode used for server-side encryption.

Supported value: `aws:kms`

option `serverSideEncryption.kmsKeyID`

Type string

Your customer-managed key

Filesystem storage options

```
storage:
  type: filesystem
  filesystem:
    path: <string>
```

option `storage.filesystem.path`

Type string

Required YES

The path to the backup directory

Point-in-time recovery options

```
pitr:
  enabled: <boolean>
```

option `pitr.enabled`

Type boolean

Enables point-in-time recovery

Backup restore options

```
restore:
  batchSize: <int>
  numInsertionWorkers: <int>
```

option `batchSize`

Type int

Default Value 500

The number of documents to buffer.

option `numInsertionWorkers`

Type int

Default Value 10

The number of workers that add the documents to buffer.

CHAPTER

FOURTEEN

SUBMITTING BUG REPORTS OR FEATURE REQUESTS

If you find a bug in Percona Backup for MongoDB, you can submit a report to the JIRA issue tracker for Percona Backup for MongoDB.

Start by searching the open tickets for a similar report. If you find that someone else has already reported your problem, then you can upvote that report to increase its visibility.

If there is no existing report, submit a report following these steps:

1. Sign in to JIRA issue tracker. You will need to create an account if you do not have one.
2. In the *Summary*, *Description*, *Steps To Reproduce*, *Affects Version* fields describe the problem you have detected. For PBM the important diagnostic information is: log files from the pbm-agents; a dump of the PBM control collections.

As a general rule of thumb, try to create bug reports that are:

- *Reproducible*: describe the steps to reproduce the problem.
- *Specific*: include the version of Percona Backup for MongoDB, your environment, and so on.
- *Unique*: check if there already exists a JIRA ticket to describe the problem.
- *Scoped to a Single Bug*: only report one bug in one JIRA ticket.

CHAPTER

FIFTEEN

GLOSSARY

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

Amazon S3 Amazon S3 (Simple Storage Service) is an object storage service provided through a web service interface offered by Amazon Web Services.

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

Bucket A bucket is a container on the s3 remote storage that stores backups.

Collection A collection is the way data is organized in MongoDB. It is analogous to a table in relational databases.

Consistency In the context of backup and restore, consistency means that the data restored will be consistent in a given point in time. Partial or incomplete writes to disk of atomic operations (e.g. to table and index data structures separately) won't be served to the client after the restore. The same applies to multi-document transactions, that started but didn't complete by the time the backup was finished.

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

GCP GCP (Google Cloud Platform) is the set of services, including storage service, that runs on Google Cloud infrastructure.

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).

MinIO MinIO is a cloud storage server compatible with *Amazon S3*, released under Apache License v2.

Oplog Oplog (operations log) is a fixed-size collection that keeps a rolling record of all operations that modify data in the database.

Oplog slice A compressed bundle of *oplog* entries stored in the Oplog Store database in MongoDB. The oplog size captures an approximately 10-minute frame. For a snapshot, the oplog size is defined by the time that the slowest replica set member requires to perform mongodump.

pbm-agent A `pbm-agent` is a *PBM* process running on the mongod node for backup and restore operations. A pbm-agent instance is required for every mongod node (including replica set secondary members and config server replica set nodes).

pbm CLI Command-line interface for controlling the backup system. PBM CLI can connect to several clusters so that a user can manage backups on many clusters.

PBM Control collections PBM Control collections are *collections* with config, authentication data and backup states. They are stored in the admin db in the cluster or non-sharded replica set and serve as the communication channel between *pbm-agent* and *pbm CLI*. *pbm CLI* creates a new pbmCmd document for a new operation. *pbm-agents* monitor it and update as they process the operation.

Percona Backup for MongoDB Percona Backup for MongoDB (PBM) is a low-impact backup solution for MongoDB non-sharded replica sets and clusters. It supports both *Percona Server for MongoDB* and MongoDB Community Edition.

Percona Server for MongoDB Percona Server for MongoDB is a drop-in replacement for MongoDB Community Edition with enterprise-grade features.

Point-in-Time Recovery Point-in-Time Recovery is restoring the database up to a specific moment in time. The data is restored from the backup snapshot and then events that occurred to the data are replayed from oplog.

Replica set A replica set is a group of mongod nodes that host the same data set.

S3 compatible storage This is the storage that is built on the *S3* API.

Server-side encryption Server-side encryption is the encryption of data by the remote storage server as it receives it. The data is encrypted when it is written to S3 bucket and decrypted when you access the data.

INDEX