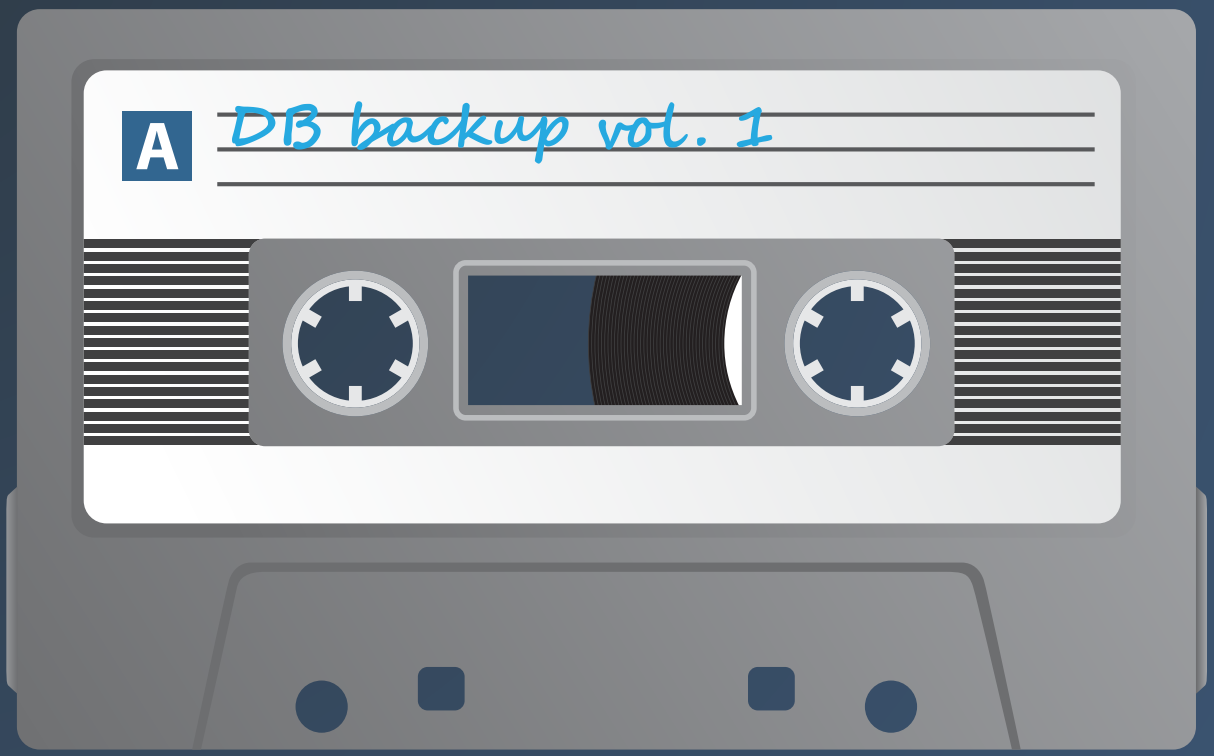
severalnines

The DevOps Guide to Database Backups for MySQL and MariaDB



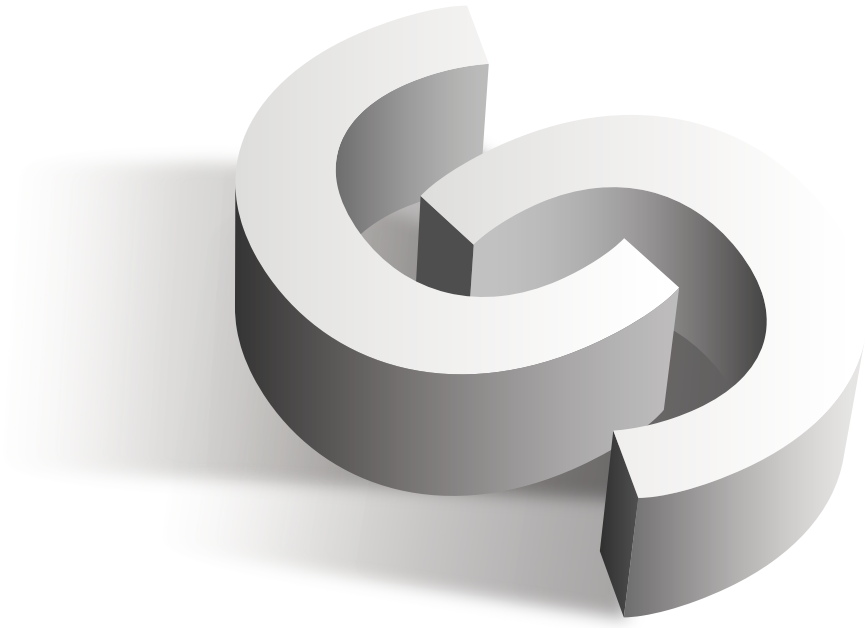


Table of Contents

Table of Contents

Introduction

A key operational aspect of database management is to ensure that backups are performed, so that a database can be restored if disaster strikes. Data loss can happen in a number of circumstances: system crash, hardware failure, power failure, human error (accidental DELETE or DROP) or even natural disaster (flood, earthquake, fire). Some of these are almost impossible to prevent from happening. The DBA or System Administrator is usually the responsible party to ensure that the data is protected, consistent and reliable. Backups are an important part of a recovery strategy for your data.

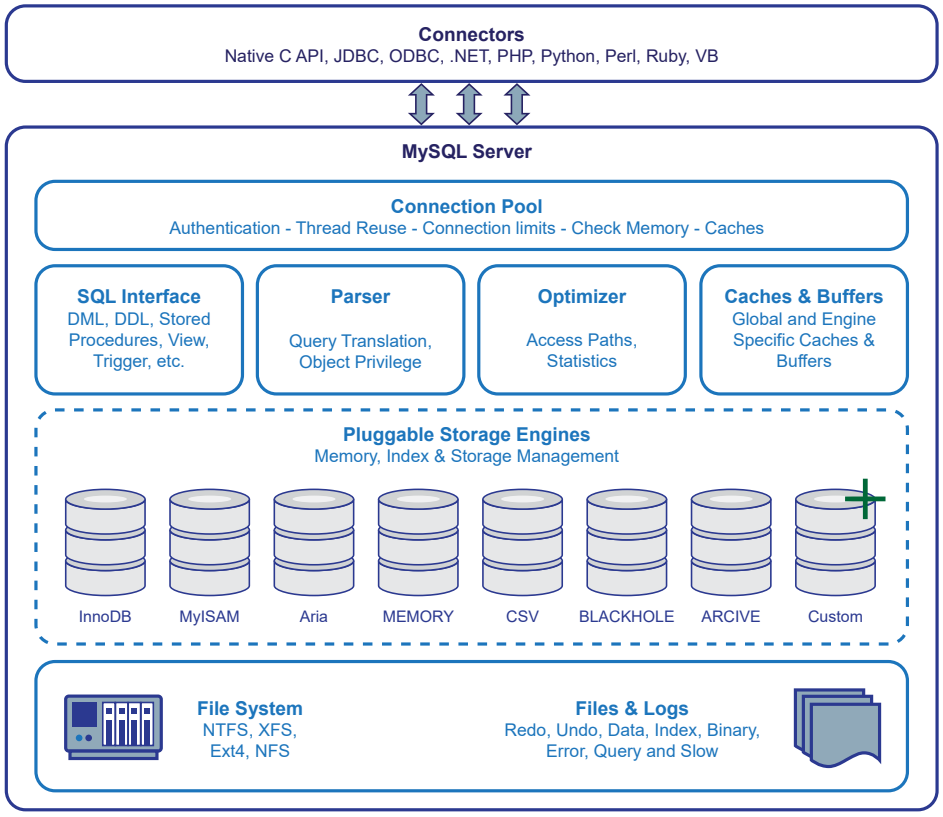
There are a number of ways to backup your database, but it is important to review the RTO and RPO before deciding on a backup strategy. RTO (Recovery Time Objective) is how long you have to recover your data, as it affects the length of your outage. RPO (Recovery Point Objective) is the allowable data loss - how much data can you afford to lose? A tighter RTO and RPO means you will need to spend more money on your infrastructure.

This whitepaper gives an overview of the two most popular backup utilities available for MySQL and MariaDB, namely mysqldump and Percona XtraBackup. We'll also see how database features like binary logging and replication can be leveraged in our backup strategy. We will discuss some best practices that can be applied to high availability topologies in order to make our backups reliable, secure and consistent.

Impact of Storage Engine on Backup Procedure

MySQL and MariaDB enables storage engines to be loaded into and unloaded from a running database server. This modular architecture provides benefits to those who wish to specifically target a particular application need, such as data warehousing, transaction processing, or high availability. The storage engine implements a more specific set of features required for a type of workload, therefore there is less system overhead with the end result being higher database performance.

The following picture shows the storage engine architecture:



Since the data is stored inside the storage engine, we need to understand how the storage engines work to determine the best backup tool. In general, MySQL backup tools perform a special operation in order to retrieve a consistent data- either lock the tables or establish a transaction isolation level that guarantees data read is unchanged.

On MySQL and MariaDB, the following storage engines are enabled by default:

- MyISAM - Default storage engine up until MySQL 5.5.5.
- InnoDB - Default storage engine since MySQL 5.5.5. XtraDB for Percona and MariaDB.
- MEMORY - Hash based, stored in memory, useful for temporary tables.
- BLACKHOLE - anything you write to it disappears.
- CSV - Stores data in text files using comma-separated values format.

- ARCHIVE - store large amounts of unindexed data with a very small overhead.
- Aria - Crash-safe tables with MyISAM heritage. Available in MariaDB distributions.

As you may notice, there are lots of storage engines preloaded into the MySQL/MariaDB server. We are going to look into the most popular ones: MyISAM/Aria, InnoDB/XtraDB and MEMORY.

2.1. MyISAM/Aria

MyISAM was the default storage engine for MySQL versions prior to 5.5.5. It is based on the older ISAM code but has many useful extensions. The major deficiency of MyISAM is the absence of transactions support.

Aria is another storage engine with MyISAM heritage and is a MyISAM replacement in all MariaDB distributions. The main difference is that Aria is crash safe, whereas MyISAM is not. Being crash safe means that an Aria table can recover from unexpected failures in a much better way than a MyISAM table can. In most circumstances, backup operations for MyISAM and Aria are almost identical.

MyISAM uses table-level locking. It stores indexes in one file and data in another. MyISAM tables are generally more compact in size on disk when compared to InnoDB tables. MyISAM uses key buffers for caching indexes and leaves the data caching management to the operating system. With the nature of table-level locking and no transaction support, the recommended way to backup MyISAM tables is to acquire the global read lock by using FLUSH TABLE WITH READ LOCK (FTWRL) to make MySQL read-only temporarily or use LOCK TABLE statement explicitly. Without that, MyISAM backups will be inconsistent.

2.2. InnoDB/XtraDB

InnoDB is the default storage engine for MySQL and MariaDB. It provides the standard ACID-compliant transaction features, along with foreign key support and row-level locking.

Percona's XtraDB is an enhanced version of the InnoDB storage engine for MySQL and MariaDB. It features some improvements that make it perform better in certain situations. It is backwards compatible with InnoDB, so it can be used as a drop-in replacement.

There are a number of key components in InnoDB that directly influences the behaviour of backup and restore operation:

- Transactions
- Crash recovery
- Multiversion concurrency control (MVCC)

2.2.1. Transactions

InnoDB does transactions. A transaction will never be completed unless each individual operation within the group is successful (COMMIT). If any operation within the transaction fails, the entire transaction will fail and any changes will be undone

(ROLLBACK).

The following example shows a transaction in MySQL (assuming autocommit is off):

```
1  BEGIN;
2  UPDATE account.saving SET balance = (balance - 10) WHERE id
   = 2;
3  UPDATE account.current SET balance = (balance + 10) WHERE id
   = 2;
4  COMMIT;
```

A transaction starts with a BEGIN and ends with a COMMIT or ROLLBACK. In the above example, if the MySQL server crashed after the first UPDATE statement completed, that update would be rolled back.

2.2.2. Crash Recovery

InnoDB maintains a transaction log, also called redo log. The redo log is physically represented as a set of files, typically named `ib_logfile0` and `ib_logfile1`. The log contains a record of every change to InnoDB data. When InnoDB starts, it inspects the data files and the transaction log, and performs two steps:

1. Applies committed transaction log entries to the data files.
2. Performs an undo operation (rollback) on any transactions that modified data but did not commit.

The rollback is performed by a background thread, executed in parallel with transactions from new connections. Until the rollback operation is completed, new connections may encounter locking conflicts with recovered transactions. In most situations, even if the MySQL server was killed unexpectedly in the middle of heavy activity, the recovery process happens automatically. No action is needed from the DBA.

Percona Xtrabackup utilizes InnoDB crash recovery functionality to prepare the internally inconsistent backup (the binary copy of MySQL data directory) into a consistent and usable database again. Details on this is explained in the Backup Tools section.

2.2.3. MVCC

InnoDB is a multiversion concurrency control (MVCC) storage engine which means many versions of a single row can exist at the same time. Due to this nature, unlike MyISAM, InnoDB does not require a global read lock to get a consistent read. It utilizes its ACID-compliant transaction component called isolation. Isolation is the “i” in the acronym ACID - the isolation level determines the capabilities of a transaction to read/write data that is accessed by other transactions.

From highest amount of consistency and protection to the least, the isolation levels supported by InnoDB are:

- SERIALIZABLE
- REPEATABLE READ (default)
- READ COMMITTED
- READ UNCOMMITTED

Covering all isolation levels in this context is unnecessary. In order to get a consistent snapshot of InnoDB tables, one could simply start a transaction with REPEATABLE

READ isolation level. In REPEATABLE READ, a read view is created at the start of the transaction, and this read view is held open for the duration of the transaction. For example, if you execute a SELECT statement at 6 AM, and come back in an open transaction at 6 PM, when you run the same SELECT, then you will see the exact same resultset that you saw at 6 AM. This is part of MVCC capability and it is accomplished using row versioning and UNDO information.

Logical backup like mysqldump is using this approach to generate a consistent backup for InnoDB without explicit table lock that can cause the MySQL server to be read-only. Details on this as in the Backup Tools chapter.

2.3. MEMORY

The MEMORY storage engine (formerly known as HEAP) creates special-purpose tables with contents that are stored in memory. Because the data is vulnerable to crashes, hardware issues, or power outages, only use these tables as temporary work areas or read-only caches for data pulled from other tables.

Despite the in-memory processing for MEMORY tables, they are not necessarily faster than InnoDB tables on a busy server, for general-purpose queries, or under a read/write workload. In particular, the table locking involved with performing updates can slow down concurrent usage of MEMORY tables from multiple sessions.

Due to the transient nature of data from MEMORY tables (data is not persisted to disk), only logical backup is capable of backing up these tables. Backup in physical format is not possible.

2.4. MyISAM, InnoDB and MEMORY Comparison

The following table illustrates the differences between storage engines:

Feature	MyISAM	InnoDB	MEMORY
Storage limits	256TB	64TB	RAM
Transaction support	No	Yes	No
Recovery from crash	Complete rebuild tables/indexes	Recover from redo logs	Vulnerable to crash
MVCC	No	Yes	No
Performance with data growth	Slow down dramatically	Performance remain almost unchanged	Performance remain almost unchanged
Foreign key support	No	Yes	No
Locking granularity	Full table	Row	Full table
Fulltext Search Index	Yes	No	No

Feature	MyISAM	InnoDB	MEMORY
Cluster Indexes	No	Yes	Yes
Data Compression	No	Yes	No
Data caches	No	Yes	N/A
Storage of table	One table stored in 3 separate files: • .FRM for table format • .MYD for data • .MYI for indexes	Table stored in table space - consisting several files (or raw disk partitions)	In memory

The above comparison shows the differences in storage engines characteristics. This helps us understand the way backup procedures should work, and ultimately reduce the risk of recovery failure when it really matters.

Backup Tools

A backup tool is an application that specifically designed to perform backup and restore of your database. In this whitepaper, we will cover mysqldump, Percona Xtrabackup and binary log. We are not going to cover other tools such as MySQL Enterprise Backup (mysqldbackup), mydumper or storage snapshots technologies.

3.1. mysqldump

This standard backup tool comes with every MySQL/MariaDB client package. The mysqldump client utility performs logical backups. It queries the MySQL/MariaDB server and produces a set of SQL statements that can be executed to reproduce the original database object definitions and table data.

3.1.1. How does it work?

Mysqldump can have different behaviour depending on the storage engine used. If there is no option supplied, mysqldump will default to use option `--opt`. This option, enabled by default, is shorthand for the combination of `--add-drop-table --add-locks --create-options --disable-keys --extended-insert --lock-tables --quick --set-charset`. It gives a fast dump operation and produces a dump file that can be quickly reloaded into a MySQL server. However, "`--lock-tables`" locks tables for each database separately, this option does not guarantee that the tables in the dump file are logically consistent between databases. Tables in different databases may be dumped in completely different states.

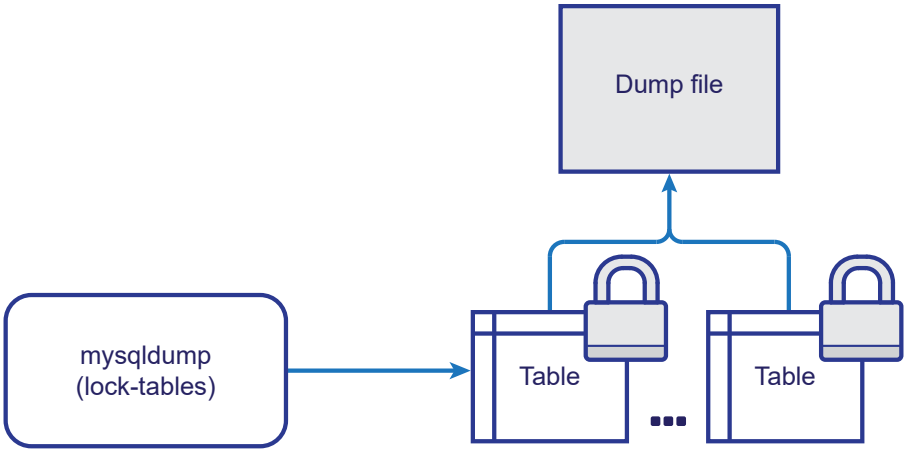
Suppose you run the command with minimal options to backup:

```
1   $ mysqldump --user=root --password --all-databases > all-da-
    tabase.sql
```

For each database schema and table, a dump performs these steps:

1. LOCK TABLE table.
2. SHOW CREATE TABLE table.
3. SELECT * FROM table INTO OUTFILE temporary file.
4. Write the contents of the temporary file to the end of the dump file.
5. UNLOCK TABLES

The process is illustrated in the following diagram:



By default mysqldump doesn't include routines and events in its output - you have to explicitly set `--routines` and `--events` flags. One must be aware of the contents of the database and execute mysqldump with parameters that take this into consideration, as described in the next section.

3.1.1.1. Non-transactional Tables

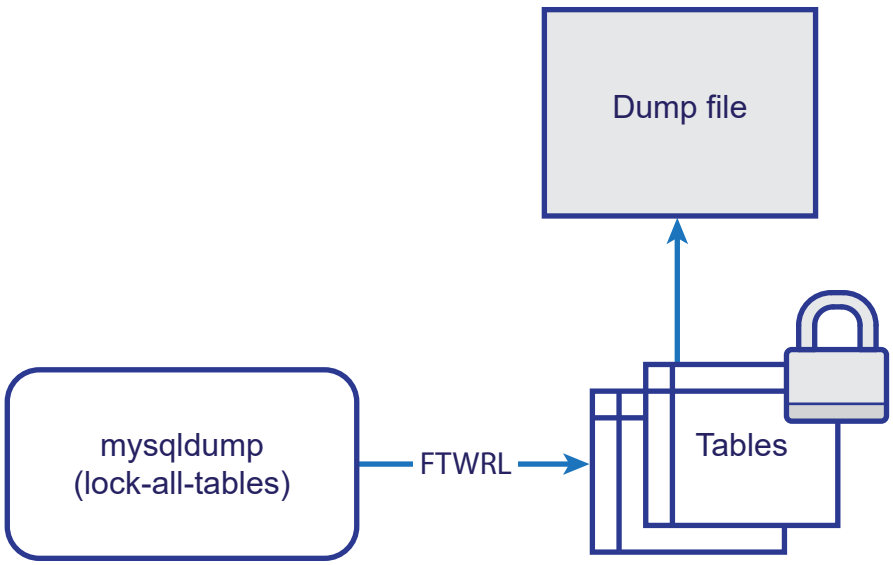
If you run mysqldump against a database that contains non-transactional tables (MyISAM for example), then mysqldump will have to lock all tables in the database to ensure consistency. The following mysqldump command performs a consistent backup on MyISAM tables:

```
1   $ mysqldump --lock-all-tables --user=root --password db1 >
    db1.sql
```

The process flow can be described as per below:

1. FLUSH TABLES WITH READ LOCK - Global read lock and not table locks.
2. For each database schema and table, a dump performs these steps:
    - SHOW CREATE TABLE.
    - SELECT * FROM table INTO OUTFILE temporary file.
    - Write the contents of the temporary file to the end of the dump file.
3. UNLOCK TABLES.

The process is illustrated in the following diagram:



FLUSH TABLES WITH READ LOCK is the only way to guarantee a consistent snapshot of MyISAM tables while the MySQL server is running. This will make the MySQL server become read-only until UNLOCK TABLES is executed.

3.1.1.2. Transactional Tables

For tables on InnoDB storage engine, it is recommended to use `--single-transaction` option. MySQL then produces a checkpoint that allows the dump to capture all data prior to the checkpoint while receiving incoming changes. Those incoming changes do not become part of the dump. That ensures the same point-in-time for all tables. The `--single-transaction` option of mysqldump does not do FLUSH TABLES WITH READ LOCK. It causes mysqldump to setup a REPEATABLE READ transaction for all tables being dumped.

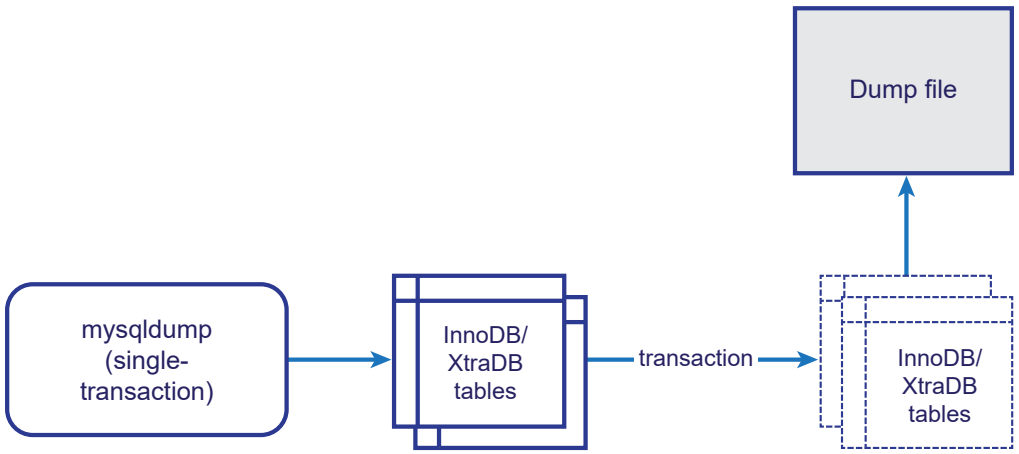
Consider the following mysqldump command for a database that only contains InnoDB tables:

```
$ mysqldump --single-transaction --user=root --password db1
> db1.sql
```

The process flow can be described as per below:

1. START TRANSACTION WITH CONSISTENT SNAPSHOT.
2. For each database schema and table, a dump performs these steps:
    - SHOW CREATE TABLE table.
    - SELECT * FROM table INTO OUTFILE temporary file.
    - Write the contents of the temporary file to the end of the dump file.
3. COMMIT.

The process is illustrated in the following diagram:



To ensure a valid dump file while a `--single-transaction` dump is in process no other connection should use the following DDL statements: ALTER TABLE, CREATE TABLE, DROP TABLE, RENAME TABLE, TRUNCATE TABLE. A consistent read is not isolated from those statements, so use of these on a table to be dumped can cause the SELECT that is performed by mysqldump to retrieve incorrect contents for the table or fail.

3.1.1.3. Flush Binary Logs

It is useful to create a new binary log whenever a mysqldump backup is taken. FLUSH LOGS closes and reopens all log files. If binary logging is enabled, the sequence number

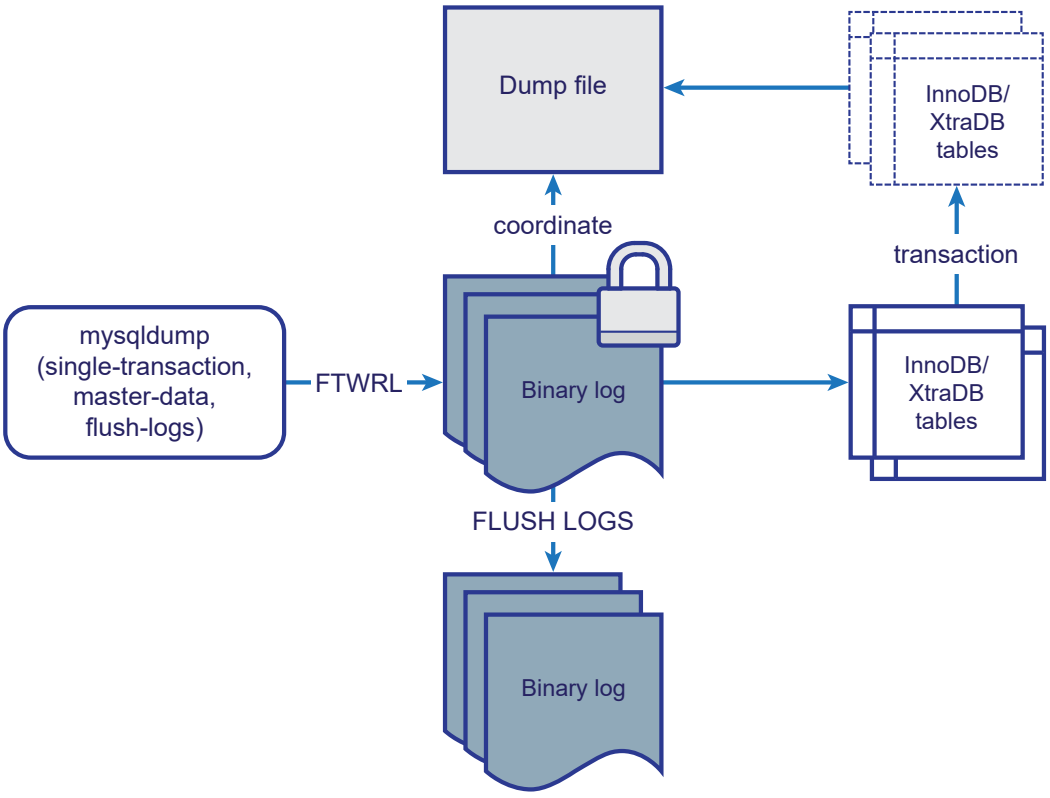
of the binary log file is incremented by one relative to the previous file. Consider the following mysqldump command against InnoDB storage engine to include binary log coordinates:

```
$ mysqldump --single-transaction --master-data --flush-logs
--user=root --password db1 > db1.sql
```

The process flow can be described as per below:

1. START TRANSACTION WITH CONSISTENT SNAPSHOT.
2. FLUSH TABLES WITH READ LOCK.
3. Get the binary log coordinates.
4. FLUSH LOGS.
5. UNLOCK TABLES.
6. For each database schema and table, a dump performs these steps:
    - SHOW CREATE TABLE table.
    - SELECT * FROM table INTO OUTFILE temporary file.
    - Write the contents of the temporary file to the end of the dump file.
7. COMMIT.

The process is illustrated in the following diagram:



With `--master-data` flag, mysqldump has to acquire a global lock for a short period of time and release it back once the binary log coordinates are retrieved. It can then proceed with retrieving data consistently without the need to lock every table.

Mysqldump does not dump the `information_schema` database by default. To dump information_schema, name it explicitly on the command line and also use the `--skip-lock-tables` option. It also never dumps the `performance_schema` database and the MySQL Cluster's ndbinfo information database.

3.1.2. Advantages

mysqldump is probably the most popular backup method for MySQL. Advantages include the convenience and flexibility of viewing or even modifying the output using standard text tools before restoring. You can clone databases for development and DBA work, or produce slight variations of an existing database for testing. It is also more practical to do partial restore, where you just want to restore only certain rows or tables.

Mysqldump is fairly easy to use and pretty straightforward. It is also machine independent and highly portable. The SQL dumps work in Windows and other operating system supported by MySQL. It is also the perfect tool when migrating data between different versions or storage engines of MySQL.

SQL dump files are also compression-friendly. Depending on the compression level and tool used, you can achieve up to 15 times compression of the backup size. The mysqldump command can also generate output in CSV, other delimited text or XML format.

Indirectly, mysqldump also detects any corrupted data files. For example when a mysqldump is taken, the data must be in a good state or an error would be generated during the dump process.

3.1.3. Disadvantages

A mysqldump backup is slower than a physical backup, notably on a large dataset because the server must access the database and convert the physical data into a logical format. Mysqldump is a single-threaded tool and this is its most significant drawback - performance is ok for small databases but it quickly becomes unacceptable if the data set grows to tens of gigabytes.

Mysqldump will do, for each database and for each table, "`SELECT * FROM ..`" and write the content to the mysqldump file. The problem with the "`SELECT * FROM ..`" is the impact if you have a data set that does not fit in the InnoDB Buffer Pool. The active data set (that your application uses) will take a hit when the "`SELECT * FROM ..`" will load up data from disk, store the pages in the InnoDB Buffer Pool, and to do so, expunge pages part of the active data set from the InnoDB Buffer pool. Hence you will get a performance degradation on that node, since the active data set is no longer in RAM but on disk.

Mysqldump backup must be performed against a running MySQL/MariaDB server. Backups in logical format are large, particularly when saved in text format, and often slow to create and restore. With large data sizes, even if the backup step takes a reasonable time, restoring the data can be very slow because replaying the SQL statements involves disk I/O for insertion, index creation, and so on.

There is no way to do an incremental backup with SQL dump. A full backup is executed each time. This can be very time-consuming especially in large databases.

When using mysqldump with a non-transactional storage engine like MyISAM, a dump holds a global read lock on all tables, blocking writes from other connections for the duration of the full backup. The locking can be optional though. However without table lock, there will be no guarantee of backup consistency.

The mysqldump backup does not include any MySQL related logs or configuration files, or other database-related files that are not part of databases.

3.2. Percona Xtrabackup

Percona XtraBackup is the most popular, open-source, MySQL/MariaDB hot backup software that performs non-blocking backups for InnoDB and XtraDB databases. It falls into the physical backup category, which consists of exact copies of the MySQL data directory and files underneath it.

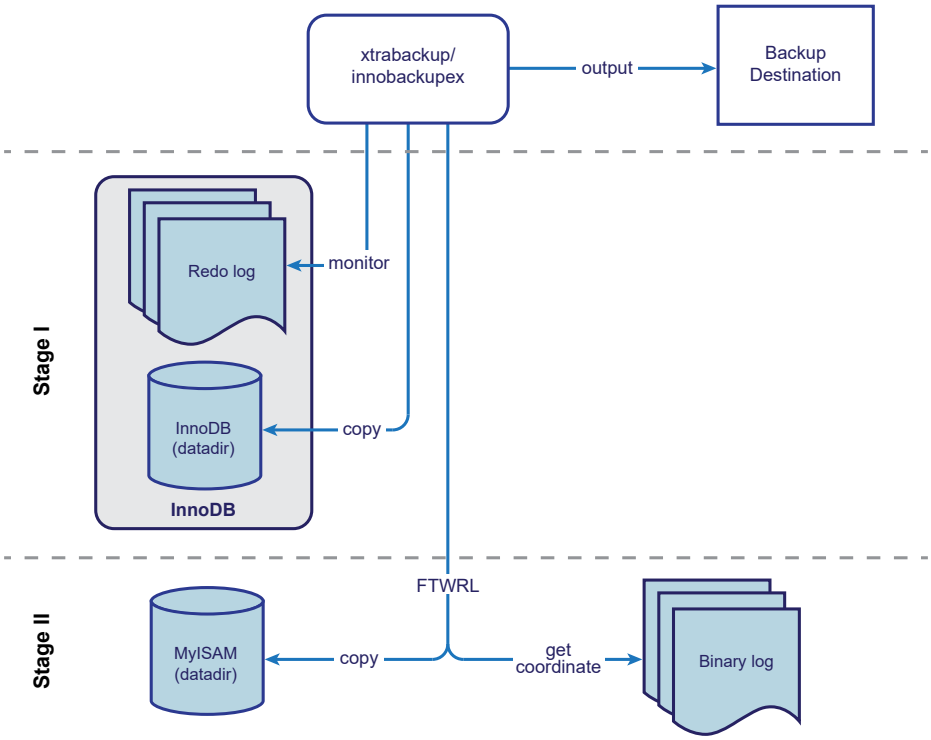
Xtrabackup does not lock your database during the backup process, provided the tables are running on InnoDB or XtraDB storage engine. For large databases (100+ GB), it provides much better restoration time as compared to mysqldump. The restoration process involves preparing MySQL data from the backup files, before replacing or switching it with the current data directory on the target node.

Percona XtraBackup is a combination of the xtrabackup C program, and the innobackupex Perl script. The xtrabackup program copies and manipulates InnoDB and XtraDB data files, and the Perl script enables enhanced functionality, such as interacting with a running MySQL/MariaDB server and backing up MyISAM tables. Percona XtraBackup works with MySQL/MariaDB servers, as well as Percona Server with XtraDB. It is also a recommended tool to perform State Snapshot Transfer (SST) in Galera Cluster. It runs on Linux and FreeBSD.

The xtrabackup and innobackupex tools permit to do operations such as streaming and incremental backups with various combinations of copying the data files, copying the log files, and applying the logs to the data.

3.2.1. How it works?

Percona XtraBackup works by remembering the log sequence number (LSN) when it starts, and then copying away the data files to another location. Copying data takes some time, and if the files are changing, they reflect the state of the database at different points in time. At the same time, XtraBackup runs a background process that keeps an eye on the transaction log (aka redo log) files, and copies changes from it. This has to be done continually because the transaction logs are written in a round-robin fashion, and can be reused after a while. XtraBackup needs the transaction log records for every change to the data files since it began execution.



Since the backup content is inconsistent, Percona Xtrabackup requires an additional process for restoration, called prepare process. During this step, Percona XtraBackup performs crash recovery against the copied data files, using the transaction log file. After this is done, the database is ready to restore and use. The final step is to overwrite (or swap) the content of MySQL datadir on the target server with the directory of the prepared backup.

The innobackupex program adds more convenience and functionality by also allowing to back up MyISAM tables and .frm files. It starts the xtrabackup process, waits until it finishes copying files, and then issues FLUSH TABLES WITH READ LOCK to prevent further changes to MySQL's data and flush all MyISAM tables to disk. During that time, no query will be executed on the host. innobackupex holds this lock, copies the MyISAM files, and then releases the lock.

The backed-up MyISAM and InnoDB tables will eventually be consistent with each other, because after the prepare (recovery) process, InnoDB's data is rolled forward to the point at which the backup completed, not rolled back to the point at which it started. This point in time matches where the FLUSH TABLES WITH READ LOCK was taken, so the MyISAM data and the prepared InnoDB data are in sync. In other words, the actual point-in-time is a moving target until the backup process is complete. For example, if Percona XtraBackup starts at midnight and lasts till 1:15 AM, then the backup's actual point-in-time is 1:15 AM.

3.2.2. Advantages

This method of raw backup is quicker than a logical backup (e.g. mysqldump), because it does not convert the contents of the database into SQL queries. It simply copies data files and the output is more compact than a logical backup. The main advantage of xtrabackup over logical backups is its speed - performance is limited by your disk or network throughput.

Percona Xtrabackup is very flexible. It supports multiple threads to copy the files quicker, or use compression to minimize size of the backup. It is possible to create a backup locally or stream it over the network using SSH tunnel or netcat. It is also possible to create incremental backups which take significantly less disk space, as well as less time to execute.

For large-scale full recovery, Percona Xtrabackup is usually faster to restore. The restore step is basically a simple copy of the prepared binary files.

In addition to database data, the backup can include any related files such as log and configuration files.

3.2.3. Disadvantages

Percona Xtrabackup needs to access the MySQL data directory locally. If you would like to perform a remote backup, the xtrabackup process must be run on the MySQL server locally and stream the backup to a separate host where the backup will be stored - for example via SSH tunnel or netcat. Performing offline backup is not possible since Percona Xtrabackup needs to access the MySQL server to check the version and generate a list of tablespaces.

If your tables are primarily InnoDB tables, then you can perform a virtually non-blocking backup. However, if you have a mix of InnoDB and MyISAM tables, or primarily MyISAM tables, xtrabackup will impact the non-transactional tables during the FLUSH TABLE WITH READ LOCK. Depending on the size of those tables, this may take a while. During

that time, no query will be executed on the host and MySQL is considered read only.

When restoring incremental backups, the overall restoration process is slower as deltas have to be applied one after another and it may take a significant amount of time. Some of the options also make it possible to restore down to table level, however it cannot go down to row level. If one of the incremental backups is corrupted, the rest will not be usable.

Percona Xtrabackup is portable only to other machines that have identical or similar hardware characteristics. If the backup was taken on Linux machine, there is no guarantee that it will work in Windows or BSD machine.

The is no option to FLUSH LOGS when taking the backup. Data from MEMORY tables cannot be backed up in physical format because their contents are not stored on disk.

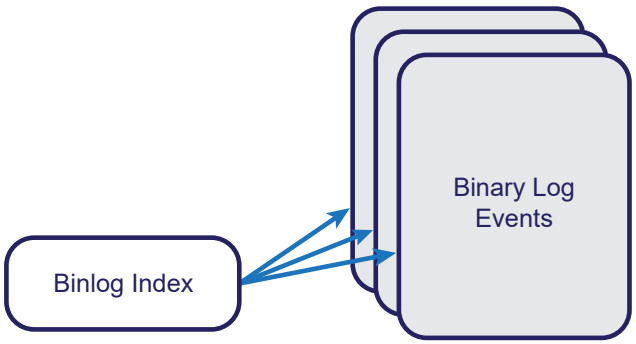
3.3. Binary Log

The binary log records changes made to the database. It is normally used for replication, where the binary log allows the same changes to be made on the slaves as well. The log contains queries like INSERT, UPDATE and DELETE. Binary logs can be used to perform a logical, incremental, hot, non-locking backup and is critical in MySQL replication. It has some performance impact on operations as all the changes in the database are logged in real time. However, the benefits of the binary log in enabling replication and performing restore operations generally overweigh the minor performance impact.

MySQL/MariaDB binary logging is disabled by default. To enable this, you have to configure `log_bin` inside the MySQL configuration file.

3.3.1. How does it work?

The events that specify modifications to data are stored in a series of files called binlog files with names in the form `[hostname]-bin.000001`, together with a binlog index file that keeps track of the existing binlog files. The index file is usually named `[hostname]-bin.index`. These files are by default located under the MySQL data directory. The names of the binlog files and the binlog index file can be controlled using the `log-bin` and `log-bin-index` options. The index file keeps track of all the binlog files used by the server so that the server can correctly create new binlog files when necessary, even after server restarts. Each line in the index file contains the name of a binlog file that is part of the binary log.



You can control the format to use when writing to the binary log using the option binlog-format with following values:

- STATEMENT causes logging to be statement based.
- ROW causes logging to be row based.
- MIXED causes logging to use mixed (STATEMENT or ROW) format.

The FLUSH LOGS command writes all logs to disk and creates a new file to continue writing the binary log. This can be useful when administering recovery images for point-in-time-recovery. Reading from an active open binlog file can have unexpected results, so it is advisable to force an explicit flush before trying to use binlog files for recovery. Flushing binary logs is usually used together with mysqldump where you can instruct it, with a flag, to flush the binary logs when performing backup. This will make the backup consistent and the newly generated binary log will start fresh to record new database changes after the backup.

MySQL provides a tool called mysqlbinlog to work with binary logs. It can be used to display the contents in text format or display the contents of relay log files written by a slave server in a replication setup, since relay logs have the same format as binary logs. You can also use this tool to backup binary logs located locally or remotely.

To backup binary logs using mysqlbinlog, we first have to retrieve the names of the binary logs currently available in the MySQL server, by using the following command:

```
mysql> SHOW BINARY LOGS;
+---------------+-----------+
| Log_name      | File_size |
+---------------+-----------+
| binlog.000130 |     27459 |
| binlog.000131 |     13719 |
| binlog.000132 |     43268 |
+---------------+-----------+
```

Then, you can use the `--read-from-remote-server` option to connect and create a copy of the binary logs in the backup destination:

```
$ mysqlbinlog --read-from-remote-server --raw --host local-
host --user=root --password binlog.000130 --result-file /
storage/backups/binlogs/binlog.000130
```

This tool only allows to backup one binary log at a time, so some iteration might be required to automate the process. Without the `--result-file` option, MySQL will default to write in the current directory using the same name as the original log file. Take note that you can just copy the binary log directory from the filesystem, but keep in mind to skip the active binary log file (i.e., the one that is currently open by the MySQL server). That is why FLUSH LOGS is important before each direct copying.

Start with a list of all the current binary logs:

```
mysql> SHOW BINARY LOGS;
+---------------+-----------+
| Log_name      | File_size |
+---------------+-----------+
| binlog.000002 | 911565723 |
| binlog.000003 |       217 |
| binlog.000004 |       217 |
```

```
| binlog.000005 |       217 |
| binlog.000006 |      8025 |
+---------------+-----------+
5 rows in set (0.00 sec)
```

Then, flush the log files so MySQL creates a new active binary log and keeps the rest inactive:

```
$ mysqladmin --user=root --password flush-logs
```

You should see a new file has been created, in this example, binlog.00007:

```
mysql> SHOW BINARY LOGS;
+---------------+-----------+
| Log_name      | File_size |
+---------------+-----------+
| binlog.000002 | 911565723 |
| binlog.000003 |       217 |
| binlog.000004 |       217 |
| binlog.000005 |       217 |
| binlog.000006 |      8069 |
| binlog.000007 |       194 |
+---------------+-----------+
```

Finally, copy all the binlogs except the recently flushed, binlog.000007 inside MySQL data directory to the backup location accordingly:

```
$ cd /var/lib/mysql
$ cp binlog.000002 binlog.000003 binlog.000004 binlog.000005
binlog.000006 /storage/backups/binlogs
```

3.3.2. Advantages

Binary log allows point-in-time recovery to happen. A backup reflects the state of the database at a certain point in time, but the changes between two backup points are not recorded. What if the server crashes a minute before the next backup should run? You can restore from the last backup, but what about the transactions until the point when the server crashed? By replaying the binary log to a server, repeating changes that were recorded in the binary log, the MySQL server can be brought back to the most up-to-date state of database right before the server crashed.

Because the binary log keeps a record of all changes, you can also use it for auditing purposes to see what happened in the database.

3.3.3. Disadvantages

In real world, though, replaying binlogs is a slow and painful process. Of course, your mileage may vary - it all depends on the amount of modifications to the database. The replaying process which involves the mysqlbinlog utility can be complicated.

Running a server with binary logging enabled comes with a performance impact. Binary

logs can eat up a significant amount of disk space if you have high database traffic, so setting up an appropriate `expire_log_days` value is important. Or you have to purge binary logs more frequently.

If you are using InnoDB tables and the transaction isolation level is READ COMMITTED or READ UNCOMMITTED, only row-based logging can be used. It is possible to change the logging format to STATEMENT, but doing so at runtime leads very rapidly to errors because InnoDB can no longer perform inserts.

3.3.4. Restoring with Binary Logs

The procedure to restore from a point in time using binary logs is:

1. Restore the database from the last completed backup closest to the desired recovery point.
2. Use the mysqlbinlog utility to restore to the desired point in time. The mysqlbinlog utility converts the events in the binary log files from binary format to text so that they can be executed or viewed.

3.3.4.1. Full Restore

The following shows the point-in-time recovery steps with binary log. In this example, we have just restored from the latest backup and would like to bring the database to its latest state just before server crashed.

1. We can determine which binlog file recorded the event by verifying the modified time of the file. In this case, the event should be stored inside

```
1  $ ls -al /var/lib/mysql
2  -rw-rw---- 1 mysql mysql       126 Oct 22 05:18 bin-
   log.000001
3  -rw-rw---- 1 mysql mysql      1197 Oct 22 14:46 bin-
   log.000002
4  -rw-rw---- 1 mysql mysql       126 Oct 22 14:46 bin-
   log.000003
5  -rw-rw---- 1 mysql mysql   6943553 Oct 23 18:38 bin-
   log.000004
```

2. Take note of the binary log file and position for the restored data when Percona Xtrabackup was executed:

```
1  $ cat /var/lib/mysql/xtrabackup_binlog_info
2  binlog.000004   5840269
```

3. Replay the binary log up from the start position and send the output to the MySQL Server:

```
1  $ mysqlbinlog /var/lib/mysql/binlog.000004 --start-po-
   sition=5840269 | mysql -uroot -p
```

The cluster will start to replay the log and catch up until the determined point.

3.3.4.2. Partial Restore

If you want to do partial restore for a truncated table, we can replay the binary logs until right before a TRUNCATE event happened to a server that caused missing rows. Then, export the table to an SQL dump file and import it back to a running MySQL server.

1. Use the mysqlbinlog tool with `--base64-output=decode-rows` to decode the binlog and send the output to a file called `decoded.txt`:

```
$ mysqlbinlog --start-datetime="2016-10-23 17:30:00"
--stop-datetime="2016-10-23 18:30:00" /var/lib/mysql/
binlog.000004 --base64-output=decode-rows --verbose >
decoded.tx
```

2. Find the line number of the TRUNCATE event:

```
$ grep -n truncate decoded.txt
45375:truncate t1
```

3. Look up the position number before the TRUNCATE event. In this case, the binlog should be replayed up until position 6494999 because position 6495077 indicates the unwanted TRUNCATE event:

```
$ head -45375 decoded.txt | tail -15
### INSERT INTO `db1`.`t1`
### SET
###   @1=8875
###   @2='781673564aa9885eeea148ebc6a58b98'
###   @3='r7LdRPhMa4kUpOLQKy033KufSw9CGYsnpInwfT8T-
WRo='
###   @4=5645
###   @5='06:10:45 PM'
# at 6494972
#161023 18:10:45 server id 1  end_log_pos 6494999
Xid = 8978
COMMIT/*!*/;
# at 6494999
#161023 18:10:45 server id 1  end_log_pos 6495077
Query   thread_id=15487 exec_time=0     error_code=0
use `db1`/*!*/;
SET TIMESTAMP=1382551845/*!*/;
truncate t1
```

4. By tailing the last 15 lines before the TRUNCATE event, we can conclude that after restoring the backups, we should replay the binlog from the recorded binlog file and position of the backup set, up until binlog.000004 on position 6494999.
   Replay the binary log up until the determined position and send the output to the MySQL Server:

```
$ mysqlbinlog /var/lib/mysql2/binlog.000004
--start-position=5840269 --stop-position=6494999 |
mysql -uroot -p
```

5. Export the data for the table so we can load it into the running cluster. We will export all columns except the primary key column because the AUTO_INCREMENT values have been repeated since the truncate happened. This will avoid DUPLICATE ENTRY errors:

```
mysql> SELECT data,extra,value,time
INTO OUTFILE '/tmp/truncated_data.sql'
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
FROM db1.t1;
Query OK, 2725 rows affected (0.00 sec)
```

Now you can import the data into the running MySQL server. Log into one of the MySQL server and start the import process:

```
mysql> LOAD DATA LOCAL INFILE 'truncated_data.sql'
INTO TABLE db1.t1
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n' (@col1,@col2,@col3,@col4)
SET data=@col1, extra=@col2, value=@col3, time=@col4;
Query OK, 2725 rows affected (0.26 sec)
Records: 2725  Deleted: 0  Skipped: 0  Warnings: 0
```

Binary logs restoration is not straightforward, but it is a safe bet for your data. It increases the probability of your data being restored to a correct state in almost any kind of data loss scenario.

Performing Backup Efficiently

All of the backup methods have their pros and cons. They also have their requirements when it comes to how they affect regular workloads. As usual, your backup strategy will depend on the business requirements, the environment you operate in and resources at your disposal.

Backup should be planned according to the restoration requirement. Data loss can be full or partial. For instance, you do not always need to recover the whole data. In some cases, you might just want to do a partial recovery by restoring missing tables or rows.

Percona Xtrabackup can be used in many backup scenarios. Having mysqldump is also important for partial recovery, where corrupted databases can be corrected by analysing at the contents of the dump. Binary logs allow us to achieve point-in-time recovery, e.g., up to right before the MySQL server went down.

In the next sections, we'll look at the different factors that contribute to efficient backup and restore procedures.

4.1. Backup Credentials

Performing backup requires a valid user to access the MySQL Server. You can, of course use the MySQL root user to perform backup and restore since it holds super-user privileges and it is intended for low-level administration purposes. It is recommended to create a dedicated backup user that holds specific privileges for backup purposes, as shown in the following commands.

To create a user, simple use the following CREATE USER statement:

```
1   mysql> CREATE USER 'backupuser'@'localhost' IDENTIFIED BY
    'backuppassword';
```

Then, GRANT the user with specific privileges for backup purposes and FLUSH the user privileges table:

```
1   mysql> GRANT SELECT, INSERT, CREATE, RELOAD, PROCESS, SUPER,
    LOCK TABLES, REPLICATION CLIENT, EVENT, CREATE TABLESPACE
2   ON *.*
3   TO 'backupuser'@'localhost';
4   mysql> FLUSH PRIVILEGES;
```

To simplify the backup and restore process, it is recommended to store these credentials where MySQL can look for it. In this way, we're avoiding to explicitly specify the password in the command line which will return the following warning:

```
1   mysql: [Warning] Using a password on the command line inter-
    face can be insecure.
```

By default, mysqldump loads all options under the `[mysqldump]` directive while Percona Xtrabackup reads `[xtrabackup]` directive inside the MySQL configuration file, or the user's option file. Setting this up before performing any backup operations will reduce the complexity of the backup commands, since we do not have to specify the loaded options anymore.

Inside the MySQL configuration file (my.cnf), adding the following lines will do the trick:

```
1  [mysqldump]
2  user=backupuser
3  password=backuppassword
4  host=localhost
5
6  [xtrabackup]
7  user=backupuser
8  password=backuppassword
9  host=localhost
```

Now you can perform a mysqldump command without the need to specify host and user credentials:

```
1  $ mysqldump --single-transaction db1 > db1.sql
```

4.2. Storage Engine

You can verify if your databases are running on a mixed storage engine by using the following query:

```
1  SELECT TABLE_SCHEMA, TABLE_NAME, ENGINE
2  FROM information_schema.TABLES
3  WHERE TABLE_SCHEMA
4  NOT IN ('mysql','information_schema','performance_schema');
```

If you have storage engines that do not support transaction (e.g. MyISAM, Aria, MEMORY), mysqldump and Percona Xtrabackup will likely have to lock the tables while the backup is taken. Lock tables will make the MySQL server read-only to ensure consistency during the backup. This is crucial factor to determine the most efficient way to perform a logical backup, where extra options are necessary in the backup command line.

To get a consistent backup on non-transactional storage engine, one must do:

```
1  $ mysqldump --lock-all-tables --all-databases > backup.sql
```

If the storage engine supports transaction (e.g. InnoDB), mysqldump does not require table locking to some extent. For InnoDB, it is sufficient to use "`--single-transaction`" to get a consistent backup. When using this option, you should keep in mind that only InnoDB tables are dumped in a consistent state. For example, any MyISAM or MEMORY tables dumped while using this option may still change state:

```
$ mysqldump --single-transaction --all-databases > backup.sql
```

However, if the "`--master-data`" option is appended, mysqldump still requires a global read lock to get the precise binlog coordinates prior to starting the REPEATABLE-READ transaction. The "`--master-data`" option triggers this lock, it is then released once the binlog coordinates have been obtained. Percona XtraDB 5.6 introduces a new option called LOCK TABLES FOR BACKUP to overcome this limitation, which allows a virtually lock-less backup through "`--lock-for-backup`" flag:

```
$ mysqldump --single-transaction --lock-for-backup --all-databases > backup.sql
```

If you do have a hybrid mix of storage engines, Percona Xtrabackup handles this with more efficiency. The locking will only happen during the MyISAM phase of the backup. The bottomline is that one should avoid using MyISAM tables if possible, except for the mysql system tables.

For Aria storage engine, there is a limitation in Percona Xtrabackup. The issue is that the engine uses recovery log files and an `aria_log_control` file that are not backed up by xtrabackup. Starting MariaDB without the `aria_log_control` file, MariaDB will mark all the Aria tables as corrupted with this error when doing a CHECK on the table:

```
"Table is from another system and must be zerofilled or repaired to be usable on this system."
```

This means that the Aria tables from an xtrabackup backup must be repaired before being usable (this can take quite long time depending on the size of the table). Another option is to perform a check on all Aria tables present in the backup after the prepare phase:

```
$ aria_chk --zerofill [tablename]
```

4.3. Dataset Size

You can get the total database size by using the following query:

```
SELECT SUM(ROUND(((data_length + index_length) / 1024 / 1024 / 1024), 2)) AS "Size in GB"
FROM information_schema.TABLES;
```

This will give you a ballpark figure. The column `index_length` is not used in mysqldump because it does not dump indexes, only data.

Bigger dataset usually means longer backup time. Using a simple rule of thumb, if you have database with less than 10GB in size and it fits into the InnoDB buffer pool, using mysqldump with binary logs enabled is a safe bet. The reason is that for most workloads, you will not notice much performance degradation despite the restoration time might be vary. For a dataset with hundreds of gigabytes of data, mysqldump is too slow to be useful and it can literally take days to restore a couple of hundred of

gigabytes. Usually when you need to restore from a backup, you are in some sort of emergency, and the restore process that takes days is not an option.

Percona Xtrabackup performs binary backups of heavily loaded MySQL servers in a relatively short time. For faster full recovery of a large dataset, this is the recommended way. The backup needs to be prepared beforehand with `--prepare` option by replaying the InnoDB redo log to a point where the backup ends:

```
1  $ xtrabackup --backup --target-dir=/storage/backups/full #
   backup
2  $ xtrabackup --prepare --target-dir=/storage/backups/full #
   prepare
```

Ensure you see the last line contains "`completed OK!`". It indicates the backup is prepared and is ready to be restored. It is important to note that the MySQL server needs to be shut down before restore is performed. You can't restore to a datadir of a running mysqld instance (except when importing a partial backup). With a single copy command, you should then be ready to start the MySQL server with the prepared data (assuming in my.cnf, you have "`datadir=/var/lib/mysql`"):

```
1  $ systemctl stop mysql
2  $ xtrabackup --copy-back --target-dir=/data/backups/
3  $ chown -R mysql:mysql /var/lib/mysql
4  $ systemctl start mysql
```

Take note that when restoring Xtrabackup incremental backups, the overall restoration process is slower as deltas have to be applied one after another (using "`--apply-log-only`" option).

With a small dataset, many might choose mysqldump instead because it is more straightforward to restore. However, with large data sizes, even if the mysqldump process takes a reasonable time, restoring the data can be very slow. Replaying the SQL statements involves disk I/O for insertion, index creation, and so on.

4.4. Recovery Objective

MySQL backup can be used to recover the whole dataset on a new server, set up a slave for asynchronous replication or upgrade to another major MySQL version. For this reason, having a schedule with full backups is a good practice. A common mistake by sysadmins is to forget to copy the binary logs as part of the backup files. Also, backing up using mysqldump for staging up a slave in the future requires additional options on the backup command, as shown in the next section - High Availability Setup.

The following commands suffice to take a full backup on mysqldump and Percona Xtrabackup:

```
1  $ mysqldump --single-transaction --triggers --events --rou-
   tines --all-databases > full-backup.sql
2  $ xtrabackup --backup --target-dir=/storage/full-backup
```

If Global Transaction Identifier (GTID) with InnoDB (GTIDs aren't available with MyISAM) is enabled, one should use the `--set-gtid-purged=OFF` option for portability:

```
$ mysqldump --single-transaction --set-gtid-purged=OFF
--triggers --events --routines --all-databases > full-back-
up.sql
```

If you are using binary columns to store blobs, it is recommended to use `--hex-blob`, to safeguard against special characters that might be there. Mysqldump will use hexadecimal notation instead, for example, 'abc' becomes 0x616263:

```
$ mysqldump --single-transaction --set-gtid-purged=OFF
--triggers --events --routines --hex-blob --all-databases >
full-backup.sql
```

In some occasions, you might need to use the backup for partial recovery, restoring only a single row, table or database. Having mysqldump is more practical since you can generate a dump file per database and directly view/modify the content of the dump file via text editor. It is recommended to backup data and schema separately and disable "`--extended-insert`" to get a more organized view of SQL statements in the dump file. The following commands perform mysqldump against InnoDB storage engine, and generates separate dump files per database:

```
$ mysqldump --single-transaction --extended-insert=0
--no-create-info [db_name] > [db_name]-data.sql
$ mysqldump --no-data [db_name] > [db_name]-schema.sql
```

Percona Xtrabackup also comes with an option called "`--export`", which basically allows restoring individual tables. However the destination server must be running either Percona Server with XtraDB or MySQL 5.6 with `innodb_file_per_table` enabled. Restoring partial backup with Xtrabackup should be done by importing the tablespace, not by using the `--copy-back` option. During the prepare stage, one would perform the following:

```
$ innobackupex --apply-log --export /path/to/partial/backup
```

You should see three files being created on the exported backup, as per below:

```
$ find /storage/backups/mysql/ -name mytable.*
/data/backups/mysql/db1/mytable.exp
/data/backups/mysql/db1/mytable.ibd
/data/backups/mysql/db1/mytable.cfg
```

Then, import the table by discarding the current tablespace, copy them to the target database directory and import the copied tablespace:

```
mysql> ALTER TABLE mydatabase.mytable DISCARD TABLESPACE;
$ copy /data/backups/mysql/db1/mytable.exp /var/lib/mysql/
db1
$ copy /data/backups/mysql/db1/mytable.ibd /var/lib/mysql/
db1
$ copy /data/backups/mysql/db1/mytable.cfg /var/lib/mysql/
db1 # if importing to MySQL 5.6 only
mysql> ALTER TABLE db1.mytable IMPORT TABLESPACE;
```

Once this is executed, data in the imported table will be available.

4.5. High Availability Setup

It is common nowadays to have a high availability setup using either MySQL Replication or Galera Cluster. It is not necessary to backup all members in the replication chain or cluster. Since all nodes are expected to hold the same data (unless the dataset is sharded across different nodes), it is recommended to perform backup on only one node (or one per shard).

For MySQL Replication, the backup should be performed on a slave provided it does not lag behind during the backup time. If you have binary logging enabled on the slave (e.g. GTID replication), it is recommended to append "`--master-data`" including "`--apply-slave-statements`" in the mysqldump command options. This is to simplify the process of staging up the new slave. These two options are helpful in setting up the slave during the restoration of mysqldump, skipping the part that you have to explicitly execute to run the slave. If you look at the content of the dump file, you should see the following lines:

```
1    STOP SLAVE;
2    SET @@GLOBAL.GTID_PURGED= .. ; -- if GTID is enabled
3    CHANGE MASTER .. ;
4    <dump content>
5    START SLAVE;
```

If the backup is taken using Percona Xtrabackup, the default options will automatically include a file under the backup directory called `xtrabackup_binlog_info` (as well as `xtrabackup_info`) which contains the binary log file, position and GTID of the last change (if enabled). Take note that Percona Xtrabackup requires the same major version of MySQL servers on the new slave. For example, if the backup was taken on MySQL 5.5, the target server must be running on MySQL 5.5 as well. If you would like to mix the MySQL versions in a single replication chain, you should use mysqldump instead.

In case of Galera Cluster, the backup might occasionally stall the cluster during the process. Fortunately, you can perform the backup in desynchronization mode with "wsrep_desync=ON". When you allow the node to desync from the cluster momentarily, the cluster performance won't get degraded during the duration of desync, which is suitable for backup workloads. However there is a risk that if the node does not get back in sync before desync is disabled, it still may cause some performance impact on the cluster.

For that particular reason, one might do:

```
1    $ mysql -uroot -p -e 'SET GLOBAL wsrep_desync=ON'
2    $ mysqldump --single-transaction --all-databases --events
     --triggers > full_backup.sql
3    $ mysql -uroot -p -e 'SET GLOBAL wsrep_desync=OFF'
```

The above is also true for Percona Xtrabackup, and you can also use `--galera-info` with Percona Xtrabackup. It then creates the `xtrabackup_galera_info` file which contains information about the local node state at the time of the backup:

```
1   $ innobackupex --galera-info /storage/backups/galera
```

Another option is to attach an asynchronous slave to the Galera Cluster for a loosely-coupled setup, which brings additional benefits as explained in the Dedicated Backup Server section further below. Enabling binary logging might be unnecessary in Galera Cluster because you have an exact copy of the data on the other cluster nodes. However, in case if you would like to have an asynchronous slave attached to one of the Galera nodes, it's recommended to to enable only on one designated master.

It is mandatory to enable `log_slave_updates` on a Galera node so events originating from the other cluster nodes are captured when local slave threads apply writesets.

4.6. Delta Size (changes between two backup points)

If your database workload is write-intensive, you might find the difference in size between the two latest full backups to be fairly huge, for example 1GB for a 10GB dataset per day. Performing regular full backups on databases with this kind of workload will likely introduce performance degradation, and it might be more efficient to perform incremental backups. Ultimately, this kind of workload will bring the database to a state where the backup size is rapidly growing and physical backup might be the only way to go. Percona Xtrabackup is very handy in this situation and incremental backup is supported right out-of-the-box.

To make an incremental backup, one must begin with a full backup as shown in the following example:

```
1   $ xtrabackup --backup --target-dir=/storage/backups/full
```

Then, proceed with incremental backups later on:

```
1   $ xtrabackup --backup --target-dir=/storage/backups/inc1
    --incremental-basedir=/storage/backups/full
2   $ xtrabackup --backup --target-dir=/storage/backups/inc2
    --incremental-basedir=/storage/backups/inc1
3   $ xtrabackup --backup --target-dir=/storage/backups/inc3
    --incremental-basedir=/storage/backups/inc2
```

When restoring the incremental backups, use the `--apply-log-only` option during the prepare stage except for the last one:

```
1   $ xtrabackup --prepare --apply-log-only --target-dir=/stor-
    age/backups/base
2   $ xtrabackup --prepare --apply-log-only --target-dir=/stor-
    age/backups/inc1
3   $ xtrabackup --prepare --apply-log-only --target-dir=/stor-
    age/backups/inc2
4   $ xtrabackup --prepare --target-dir=/storage/backups/inc3
```

Once prepared, the backup can be restored using `--copy-back` option.

If you choose to use mysqldump to do a full backup, having binary logs is

recommended for incremental backup. Thus, making sure the binary logs are backed up properly is critical for recovery. You can use the mysqlbinlog utility to achieve this. Flushing binary logs is also required for each full backup interval and ensures the binary log will not expire during this interval. For example, if you schedule a mysqldump every Sunday, ensure `expire_log_days` is more than 7 days. Take note that binary logs can be very huge for write-intensive workloads, and they come with a price of longer recovery time.

4.7. Backup Size

If you take a plain backup, the backup size is usually similar size to the size of the actual database. If you have a limited storage space backed by an outdated disk subsystem, compression is your friend. Take note that performing compression is a CPU intensive process and can directly impact the performance of your MySQL server. However, some environments have a period of low database traffic and using compression can save you a lot of space. It is a bit of tradeoff of processing power over storage space, reducing the risk of server crash caused by a full disk.

There are lots of compression tools available out there, namely gzip, bzip2, zip, rar and 7z. These tools can do both compression and archiving (packing multiple files into one). Here are some typical ratings in terms of speed, availability and typical compression ratio:

Domain	Justification	Rating
Compression speed	Fast > slow	gzip, zip > bzip2 > 7z > rar
Compression ratio	Better > worse	7z > rar, bzip2 > gzip > zip
Decompression speed	Fast > slow	gzip, zip > 7z > rar > bzip2
Availability	UNIX	gzip > bzip2 > zip > 7z > rar
	Windows	zip > rar > 7z > gzip, bzip2

The above ratings are somewhat subjective, but in general terms, they give a good indication of what to expect. The two most popular tools for compression are gzip and bzip2, which are widely available in UNIX environment. As you can see, bzip2 offers better compression ratio but is slower while gzip is overall faster. If having a smaller backup size is important in your environment, use bzip2. Otherwise, gzip is a good choice.

Normally, mysqldump can have very good compression rates as it is a flat text file. Depending on the compression tool and ratio, a compressed mysqldump can be up to 6 times smaller than the original backup size. To compress the backup, you can pipe the mysqldump output to a compression tool and redirect it to a destination file:

```
$ mysqldump --single-transaction --all-databases | gzip > /storage/backups/all-databases.sql.gz
$ mysqldump --single-transaction --all-databases | bzip2 > /storage/backups/all-databases.sql.bz2
```

If you want a smaller dump size, you can also skip several things like comments, lock tables statement (if InnoDB), skip GTID purged and triggers:

```
$ mysqldump --single-transaction --skip-comments --skip-triggers --skip-lock-tables --set-gtid-purged OFF --all-databases | gzip > /storage/backups/all-databases.sql.gz
```

With Percona Xtrabackup, you can use the streaming mode (innobackupex), which sends the backup to STDOUT in special tar or xbstream format instead of copying files to the backup directory. Having a compressed backup could save you up to 50% of the original backup size, depending on the dataset. Append the `--compress` option in the backup command as per below:

```
$ innobackupex --stream=tar  ./ > backup.xbstream
```

By using the xbstream in streaming backups, you can additionally speed up the compression process by using the `--compress-threads` option. This option specifies the number of threads created by xtrabackup for parallel data compression. The default value for this option is 1. To use this feature, simply add the option to a local backup, for example:

```
$ innobackupex --stream=xbstream --compress --compress-threads=4 ./ > /storage/backups/backup.xbstream
```

Before applying logs during the preparation stage, compressed files will need to be decompressed using xbstream:

```
$ xbstream -x < /storage/backups/backup.xbstream
```

Then, use qpress to extract each file ending with .qp in their respective directory before running `--apply-log` command to prepare the MySQL data.

4.8. Encryption

If your MySQL server or backup destination is located in an exposed infrastructure like public cloud, hosting provider or connected through an untrusted WAN network, it is probably a good idea to enforce encryption to enhance the security of backup data. A simple use case to enforce encryption is where you want to push the backup to an off-site backup storage located in the public cloud.

When creating an encrypted backup, one thing to have in mind is that it usually takes more time to recover. The backup has to be decrypted prior to any recovery activities. With a large dataset, this could introduce some delays to the RTO. On the other hand, if you are using private key for encryption, make sure to store the key in a safe place.

If the private key is missing, the backup will be useless and unrecoverable. If the key is stolen, all created backups that use the same key would be compromised as they are no longer secured. You can use the popular GnuPG or OpenSSL to generate the private or public keys.

To perform mysqldump encryption using GnuPG, generate a private key and follow the wizard accordingly:

```
1   $ gpg --gen-key
```

Create a plain mysqldump backup as usual:

```
1   $ mysqldump --routines --events --triggers --single-transac-
    tion db1 | gzip > db1.tar.gz
```

Encrypt the dump file and remove the older plain backup:

```
1   $ gpg --encrypt -r 'admin@email.com' db1.tar.gz
2   $ rm -f db1.tar.gz
```

GnuPG will automatically append .gpg extension on the encrypted file. To decrypt, simply run the gpg command with `--decrypt` flag:

```
1   $ gpg --output db1.tar.gz  --decrypt db1.tar.gz.gpg
```

To create an encrypted mysqldump using OpenSSL, one has to generate a private key and a public key:

```
1   $ openssl req -x509 -nodes -newkey rsa:2048 -keyout dump.
    priv.pem -out dump.pub.pem
```

This private key (dump.priv.pem) must be kept in a safe place for future decryption. For mysqldump, an encrypted backup can be created by piping the content to openssl, for example:

```
1   $ mysqldump --routines --events --triggers --single-transac-
    tion database | openssl smime -encrypt -binary -text -aes256
    -out database.sql.enc -outform DER dump.pub.pem
```

To decrypt, simply use the private key (dump.priv.pem) alongside the `-decrypt` flag:

```
1   $ openssl smime -decrypt -in database.sql.enc -binary -in-
    form DEM -inkey dump.priv.pem -out database.sql
```

Percona XtraBackup can be used to encrypt or decrypt local or streaming backups with xbstream option in order to add another layer of protection to the backups. Encryption is done with the libgcrypt library. Both `--encrypt-key` option and `--encrypt-key-file` option can be used to specify the encryption key. Encryption keys can be generated with commands like:

```
$ openssl rand -base64 24
bWuYY6FxIPp3Vg5EDWAxoXlmEFqxUqz1
```

This value then can be used as the encryption key. Example of the innobackupex command using the `--encrypt-key`:

```
$ innobackupex --encrypt=AES256 --encrypt-key="bWuYY6FxIP-
p3Vg5EDWAxoXlmEFqxUqz1" /storage/backups/encrypted
```

The output of the above OpenSSL command can also be redirected to a file and can be treated as a key file:

```
$ openssl rand -base64 24 > /etc/keys/pxb.key
```

Use it with the `--encrypt-key-file` option instead:

```
$ innobackupex --encrypt=AES256 --encrypt-key-file=/etc/keys/
pxb.key /storage/backups/encrypted
```

To decrypt, simply use the `--decrypt` option with appropriate `--encrypt-key` or `--encrypt-key-file`:

```
$ innobackupex --decrypt=AES256 --encrypt-key="bWuYY-
6FxIPp3Vg5EDWAxoXlmEFqxUqz1" /storage/backups/encrypt-
ed/2016-10-18_10-20-14/
```

4.9. Dedicated Backup Server

A dedicated backup server is a good approach to minimizing the risks of touching production servers. A backup server is usually an isolated slave connected to the production servers via asynchronous replication. A good backup server consists of plenty of disk space for backup storage, with the ability to do storage snapshots. It does not really need to be as powerful as the servers handling production workload, but still must be decent enough to avoid severe replication lag.

Taking a MySQL backup on a dedicated backup server will simplify your backup plans. Since it uses loosely-coupled asynchronous replication, it will unlikely cause additional overhead to the production database. However, this server is exposed to a single point of failure with the possibility of inconsistent backup if the backup server regularly lags behind. A best practice when automating the process is to ensure the backup server has caught up with the designated master prior to executing the backup. To check how behind the slave is, you can use the following statement and look for "Seconds_Behind_Master" value:

```
SHOW SLAVE STATUS\G
```

If the backup server is dedicated for backup storage, you can stream the backup over network to this server using a combination of compression (gzip, tar and xbstream) alongside network interaction tools like SSH, rsync or netcat. With mysqldump, you can

use gzip and SSH to compress and stream the created backup to the another server:

```
$ mysqldump --single-transaction --triggers --events --routines --hex-blob --set-gtid-purged=OFF --all-databases | gzip -c | ssh root@storage 'cat > /storage/backups/full-backup.sql.gz'
```

Or, use mysqldump to connect to the target MySQL server remotely and perform the dump (provided you are running the same MySQL client version as that of the target server):

```
$ mysqldump --host slave1 --port 3306 --single-transaction --triggers --events --routines --hex-blob --set-gtid-purged=OFF --all-databases | gzip -c > /storage/backups/full-backup.sql.gz
```

With Percona Xtrabackup, you can use `--stream` option (available in innobackupex) to send it to another server instead of storing it locally. There are two streaming tools supported, tar and xbstream:

```
# using tar
$ innobackupex --stream=tar ./ | ssh root@storage "cat - > /storage/backups/fullbackup.tar"
# using xbstream
$ innobackupex --compress --stream=xbstream /storage/backups/ | ssh root@storage "xbstream -x -C /storage/backups/"
```

The main advantage with this setup is that it simplifies the management of backup storage by consolidating backups in one centralized location. By keeping data in one place, it's easier to manage both the hardware and the data itself. That means closer control on data protection, version control and security with a consistent set of data. It also means better control of hardware configuration, capacity and performance. By focusing your efforts in one place, it should also reduce expenditure and risk. Other benefits in having a dedicated backup server is that you can use it as a sandbox to perform regular backup verifications, create a staging MySQL server to extract partial data or prepare the restore data before copying the MySQL data directory onto the target server.

Backup Management

Making sure that backups run successfully every day can be a chore – checking whether each job has completed successfully, re-running jobs that failed, swapping out disks and removing data off site. All these tasks take up time and add zero business value. Then there is also the task of restoring data or configurations, when there is a problem with the database or request from the application developer or QA.

Performing a backup is easy. The harder part is to ensure the backups are organized, usable, available and manageable from the Ops perspective. The number of backup files will grow, database sizes will grow over months and years, and backup procedures will become more complex - especially with all the utilities required to make it all work. Therefore, we must carefully plan our backup strategy from the beginning in order to avoid issues further down the line.

5.1. Backup Scheduling

Performing regular backups of your database is imperative for high availability and disaster recovery. If for any reason you lost your entire cluster and had to do a full restore from backup, you would need a reliable and up-to-date backup to start from.

Some recommendations to consider for a good backup strategy:

- You should be able to completely recover from a catastrophic failure from at least two previous full backups. Just in case the most recent full backup is damaged, lost, or corrupt.
- Your backup should contain at least one full backup within a chosen cycle, normally weekly.
- Store backups away from the current data location, preferably off site.
- Use a mixture of mysqldump and Xtrabackup for extra safety, and not rely on only one method.
- Test restore your backups on a regular basis, e.g. at minimum every two months if not more frequently.

A weekly full backup combined with daily incremental backup is normally enough. Keep a number of backups for a period of time, for instance, each weekly backup stored for at least one month. This allows you to recover an older database in case of emergencies, or if for some reason, you have corrupted backup files.

Apart from regular backup schedules, you might also need to backup your data occasionally before making significant changes, for example, schema, software or hardware changes. In conjunction with binary logging, you will then avoid data loss and you can at least revert to the position just before the failed change (e.g an erroneous drop table).

If using binary logs, we recommend you set `expire_log_days=X+1` in my.cnf, where X are the number of days between full backups.

You should also schedule for backup verification, to verify that backups are usable and restorable. Once in a month, you may try restore any random backup from all multiplexed devices (i.e., local server/external server/SAN/tape).

5.2. Backup Verification and Integrity

To verify that your backup has been successful, restore the backup data on a different server and run the MySQL daemon (mysqld) on the new data directory. Nothing is better than testing a restore, and it should be a periodic procedure. You should be able to start mysqld without problems. Once you have mysqld running, you need to test each table's usability. You can then execute SHOW statements to verify the database and table structures, and execute queries to further verify details of the database.

You may add other verification criterias to trigger alerts. For example, the size of the backup file should be more than x GB (depends upon standard backup size you get). An alarm is triggered if it is lesser than that. When copying or moving the backup files from one location to another, checksum the file to verify its integrity. You can use the ***md5sum*** command to calculate the checksum and compare before and after the operation, for example:

```
## On local server
$ md5sum backup.sql
71e41ff4ebf84db6f07eb73bddcd6073  backup.sql
## Copy to storage server
$ scp sbtest.sql root@storage:/backups/dump/
## On storage server
$ md5sum /backups/dump/backup.sql
71e41ff4ebf84db6f07eb73bddcd6073  /backups/dump/backup.sql
```

There are also some tools available in the MySQL ecosystem to verify the integrity of a backup, as shown in the next sections.

5.2.1. mysqlcheck

MySQL provides utility tools to check for database consistency and check for errors. One of it is ***mysqlcheck***, which uses the SQL statements CHECK TABLE, REPAIR TABLE, ANALYZE TABLE, and OPTIMIZE TABLE in a convenient way for the user. It determines which statements to use for the operation you want to perform, and then sends the statement to the server to be executed. Mysqlcheck is also invoked by the `mysql_upgrade` script to check tables and repair them if necessary.

It is sufficient to invoke the `--analyze` option when performing the check:

```
$ mysqlcheck --analyze --databases db1 --user root --password
```

5.2.2. mysqldbcompare

MySQL provides a utility called ***mysqldbcompare***, to compare the objects and data from two databases to find differences. This tool is only available as part of mysql-utilities package. It identifies objects having different definitions in the two databases and presents them in a diff-style format. However, the data must not change during the comparison as unexpected errors may then occur.

If you are using mysqldump to backup a single database in MySQL replication or Galera Cluster with asynchronous slave, you can use one of the slave servers for backups and also periodically test the restore. The process of testing can be done as per example

below:

1. Stop the slave process (so database does not get updated)
2. Run the backup command using mysqldump for the selected database
3. Create a new database for restore purpose, for example restore_db1
4. Restore the data from backup into restore_db1
5. Use mysqldbcompare to compare the two databases
6. Drop restore_db1 database
7. Start the slave process again

By running ***mysqldbcompare***, you should see output similar as per below:

```
1   $ mysqldbcompare --server1=root:password@localhost --diff-
    type=sql db1:restore_db1
2   # WARNING: Using a password on the command line interface
    can be insecure.
3   # server1 on localhost: ... connected.
4   # server2 on localhost: ... connected.
5   # Checking databases sbtest on server1 and backup_test on
    server2
6   #
7   #                                                  Defn
    Row     Data
8   # Type      Object Name                            Diff
    Count   Check
9   # ------------------------------------------------------------
    ---------------
10  # TABLE     sbtest1                                pass
    pass    -
11  #           - Compare table checksum
    FAIL
12  #           - Find row differences
    pass
13  # TABLE     sbtest10                               pass
    pass    -
14  #           - Compare table checksum
    FAIL
15  #           - Find row differences
    pass
16  # TABLE     sbtest9                                pass
    pass    -
17  #           - Compare table checksum
    FAIL
18  #           - Find row differences
    pass
19
20
21  # Databases are consistent.
22  #
23  # ...done
```

At the end, you can see the result whether the databases are consistent. At this point, the backup is verified to be working and you can safely store it to an appropriate backup location.

5.2.3. pt-table-checksum

Another way to verify if the backup is consistent is by setting up the replication and running ***pt-table-checksum***. This can be used to verify any type of backups, but before setting up replication, the backup should be prepared and be able to run. This means that incremental backups should be merged with full backups, encrypted backups should be decrypted and so on. It performs an online replication consistency check by executing checksum queries on the master, which produces different results on replicas/ slaves that are inconsistent with the master.

You can perform the validation process as follows:

1. Run pt-table-checksum on master to check slave consistency
2. Backup one of the slaves
3. Wipe the data off the slave
4. Perform restoration on the slave
5. Re-validate slave consistency using pt-table-checksum

Example output of the pt-table-checksum:

```
1   $ ./pt-table-checksum
2   TS             ERRORS  DIFFS     ROWS  CHUNKS SKIPPED
    TIME TABLE
3   04-30T11:31:50      0      0   633135       8       0
    5.400 exampledb.aka_name
4   04-30T11:31:52      0      0   290859       1       0
    2.692 exampledb.aka_title
5   Checksumming exampledb.user_info:  16% 02:27 remain
6   Checksumming exampledb.user_info:  34% 01:58 remain
7   Checksumming exampledb.user_info:  50% 01:29 remain
8   Checksumming exampledb.user_info:  68% 00:56 remain
9   Checksumming exampledb.user_info:  86% 00:24 remain
10  04-30T11:34:38      0      0 22187768     126       0
    165.216 exampledb.user_info
11  04-30T11:38:09      0      0        0       1       0
    0.033 mysql.time_zone_name
12  04-30T11:38:09      0      0        0       1       0
    0.052 mysql.time_zone_transition
13  04-30T11:38:09      0      0        0       1       0
    0.054 mysql.time_zone_transition_type
14  04-30T11:38:09      0      0        8       1       0
    0.064 mysql.user
```

If all the values in the DIFFS column are 0, that means that the backup is consistent with the current setup. At this point, the backup is verified to be working and you can safely store it to an appropriate backup location.

5.3. Backup Availability

A local backup is performed on the same host where the MySQL/MariaDB server runs, whereas a remote backup is executed from a different host. Mysqldump can perform a backup against a local or remote MySQL server and the backup output is stored in the location where the process is initiated. On the other hand, Percona Xtrabackup needs to access the filesystem and MySQL data directory. So the backup has to be initiated locally on the MySQL server, with options to store the backup locally or stream it over the network to another host.

5.3.1. Onsite Storage

Onsite storage usually entails storing important data on a periodic basis on local storage devices, such as hard drives, DVDs, magnetic tapes, or CDs. It has some advantages over offsite storage, including:

- Immediate access to data.
- Less expensive.
- Internet access is not needed.

If the backup should be stored locally on the MySQL server, try to avoid using compression since this process is CPU intensive and can directly impact the performance of your MySQL server. However, some environments have a period of low database traffic and using compression can save you a lot of space. It is a bit of tradeoff of processing power over storage space.

5.3.2. Offsite Storage

Offsite storage has some advantages over onsite storage, including:

- Access to data from any location, via Internet or FTP
- Data will be preserved in case you lose your data center
- Backup data can be shared with a number of different remote locations.

The major bottleneck here however is the data transfer speed. Unless it operates on a high speed LAN backbone, the remote backup can be ineffective as it is tied to the maximum upstream speed of the network. To save bandwidth, one would compress the backup on the MySQL server before transferring it over the network.

5.3.3. Hybrid Storage

With the booming public and private cloud storage industry, with options like Dropbox, Google Drive and Amazon S3, we have a new category called hybrid storage. This technology allows the files to be stored locally, with changes automatically synced to a remote storage in the cloud.

This is handy in some cases where you can get the best of both worlds, onsite and offsite. However, the bandwidth in data center is usually costly to support such seamless integration. You do not want the syncing process to eat the allocated bandwidth and compromise the reliability of your production database server. Security is also another concern where you have to enforce encryption over the line to the public cloud.

A good use case of hybrid storage is to have a dedicated backup server, with a dedicated bandwidth line to sync up backup files into the cloud.

5.4. Backup Housekeeping

To keep backup storage space at an optimum level, you should regularly delete backups that are no longer needed for recovery. Ideally, a full backup with associated incremental backups and binary logs can be purged after you exceed the `expire_log_days` value. However, you have to ensure that your backup files are verified and restorable before purging them. You can also use `PURGE BINARY LOGS` statement to remove the older logs once they are copied:

```
1   mysql> PURGE BINARY LOGS TO ‘binlog.000006’;
```

Occasionally, for Percona Xtrabackup, one would decompress and prepare a backup in the MySQL server after the recovery process has been carried out. The prepared data could eat up significant disk space and this can lead to operating system instability. A good approach for this is to perform this exercise (decompress and prepare) in a “dump” directory, and schedule a garbage collector command or script to clear the data on a daily basis.

5.5. Backup Failover

Backups are usually scheduled in non-peak hours, and maintenance activities might also be scheduled during the same period. This can cause scheduled backup jobs to overlap with some of these activities, therefore affecting database availability. In some scenarios, the scheduled backup on a target host might be skipped if the target host is down, is unreachable or under maintenance.

To overcome this, ensure you have a safeguard mechanism to check if the backup process has completed correctly, or otherwise, move to the next available node for that particular backup set. This requires some logic extension to the backup command or script that you are using. It is also a good approach to use another independent server (for example, monitoring server) to trigger the backup process on another server. You can also review your backup schedule to avoid overlap with maintenance activities.

ClusterControl as Backup Manager

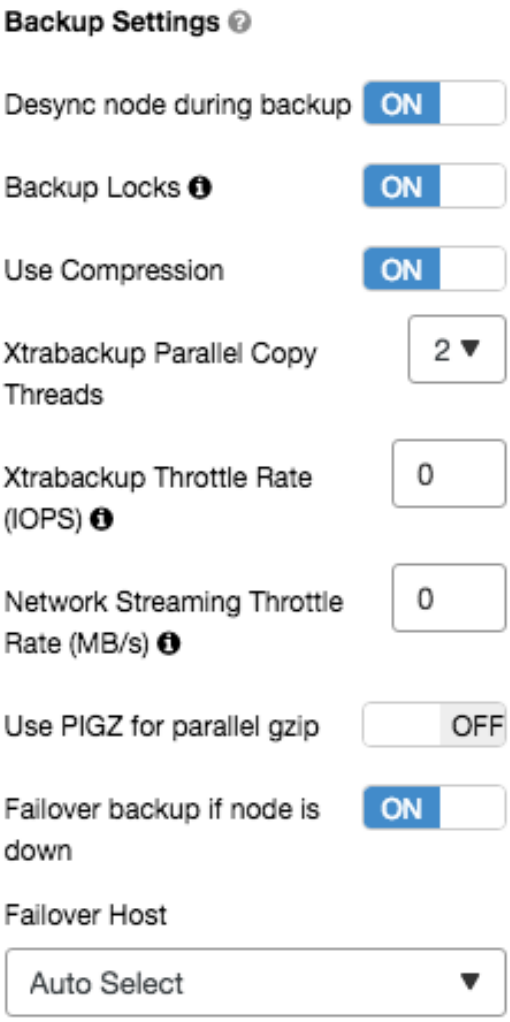
Backup managers are third party tools to simplify the backup operation and management. They do not add features to the current backup - they organize, optimize and use what it is available at operating system or database level. Managing backups can become complex when you have to deal with large datasets, growing database workloads, or multiple servers.

ClusterControl is an agentless database automation tool from Severalnines. It helps manage, monitor and deploy MySQL/MariaDB, MongoDB and PostgreSQL, therefore reducing complexity for operations. ClusterControl backup management provides a simplified and straightforward web interface, from which you can manage multiple systems and clusters. This is to avoid dealing with the long list of command line options when executing backups - offline or online, hot backup or cold backup, full or incremental, mysqldump or Percona Xtrabackup, and so on. The command line arguments for the respective methods are optimised based on the workloads, and would comply to the MySQL backup best practices.

The main features are:

- Generate a mysqldump/Percona Xtrabackup backup immediately.
- Schedule mysqldump/Percona Xtrabackup operation.
- Backup failover - Performs backup on the next available node if the target node is down or unreachable.
- Full, partial (selective databases), incremental backup (backup the deltas since last backup).
- Adapts to the topology - If the MySQL server produces binary log, binlog file and position is automatically captured.
- Centralized backup or on-site backup storage.
- Network and disk IOPS throttling.
- Restore from the created backup or backup created externally.
- Backup reports - Shows backup summary, status, size, location, target node and database, error log (if failed).
- Daily/weekly/monthly operational reports on backup summary.
- Backup notifications via alarms and email.
- Manages incremental backups, and groups the combination of full and incremental backups in a backup set.
- Check the backup destination free space prior to perform a backup.
- Auto-purge backup after exceeding the purge interval.

It is possible to easily specify the backup settings from ClusterControl.



There are interesting features that are adapted to the database topology used. For Galera clusters, ClusterControl can desync a node during backup, so it won't affect the running database cluster. It can also automatically failover the backup to the other host, in case the primary backup host fails. Backup files can be stored locally on the node where the backup is taken, or they can also be streamed to the controller node and compressed on the fly. Incremental backups are grouped with the appropriate full backup, into backup sets. This is a neat way to organize backups, and reduce the complexity of the recovery process.

All incremental backups after a full backup will be part of the same backup set, as seen in the screenshot below:

Backup List	Status	Time	Host	Method
Backup Set: 124 Restore More	✔ Completed	at 11/10/2016 @ 12:00PM	192.168.55.171	mysqldump
Backup Set: 120 Restore More	✔ Completed	at 11/09/2016 @ 3:00PM	192.168.55.171	xtrabackup (full)
└ **Backup: 122** Log	✔ Completed	at 11/09/2016 @ 3:30PM	192.168.55.171	xtrabackup (incr)
└ **Backup: 121** Log	! Failed	at 11/09/2016 @ 3:16PM	192.168.55.171	xtrabackup (incr)

By clicking on the Restore button, you are two clicks away from a full restoration of the completed backups. ClusterControl will automatically perform all the necessary backup preparation processes for Percona Xtrabackup and do the final copy-back before re-bootstrapping the cluster. You will end up with a running and fully restored cluster, where you can immediately proceed to do point-in-time recovery using binary logs if necessary.

ClusterControl also provides operational reports, for all database systems managed by ClusterControl. The following screenshot is an example of the report:

Backup Summary

This section describe backups for the report period for each cluster managed by ClusterControl.

Cluster Name	Cluster Type	Last Backup	# Successful Backups	# Failed Backups	Success Rate
cluster_1 (1)	galera	Successful, more than one day ago (May 15 13:16:07)	3	0	100.00%

cluster_1 (1) - Backup Details

This section describes the backups that have been executed during the report period.

Backup Id	Status	Backup Time	Backup Method	Backup Type	Backup Host	Database	Storage Location	Size
5	running	May 16 13:00:16	xtrabackup	full	172.30.4.115			
4	completed	May 15 13:16:07	xtrabackup	full	172.30.4.115		172.30.4.50:/tmp/backup/BACKUP-4/backup-full-2016-05-15_130007.xbstream.gz	5.55 GiB
3	completed	May 14 13:15:57	xtrabackup	full	172.30.4.115		172.30.4.50:/tmp/backup/BACKUP-3/backup-full-2016-05-14_130018.xbstream.gz	5.55 GiB
2	completed	May 13 14:00:00	xtrabackup	full	172.30.4.115		172.30.4.50:/tmp/backup/BACKUP-2/backup-full-2016-05-13_130019.xbstream.gz	5.92 GiB

It contains two sections and basically gives you a short summary of when the last backup was created, if it completed successfully or failed. You can also check the list of backups executed on the cluster with their state, type and size. This is as close you can get to check that backups work correctly without running a full recovery test. However, we definitely recommend that such tests are regularly performed.

Conclusion

Things fail. It is wise to take measures that prevent failures - redundant hardware, mirrored storage, replication and clustering, failover technology and multi datacenter architectures are some of them. These can minimize the need for a full recovery of your data, but no amount of planning can prevent an unexpected failure. A sound backup and recovery plan is your insurance policy, and one you probably need if you value your data. The amount of data handled by a database server is also growing. Not too long ago, it was common to talk in terms of tens or a few hundreds of gigabytes on a single server. Now it is common with half a terabyte and upwards - on a single server. Business are generating more data in general, and commodity servers nowadays have plenty of RAM, CPU and SSD storage to handle higher data volumes. Therefore, an efficient backup strategy is key to business continuity.

About Severalnines

Severalnines provides automation and management software for database clusters. We help companies deploy their databases in any environment, and manage all operational aspects to achieve high-scale availability.

Severalnines' products are used by developers and administrators of all skills levels to provide the full 'deploy, manage, monitor, scale' database cycle, thus freeing them from the complexity and learning curves that are typically associated with highly available database clusters. The company has enabled over 8,000 deployments to date via its popular ClusterControl solution. Currently counting BT, Orange, Cisco, CNRS, Technicolour, AVG, Ping Identity and Paytrail as customers. Severalnines is a private company headquartered in Stockholm, Sweden with offices in Singapore and Tokyo, Japan. To see who is using Severalnines today visit, http://severalnines.com/customers.

Deploy

Manage

Monitor

Scale

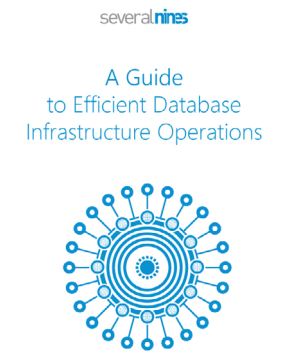
Related Resources from Severalnines

Whitepapers

Management and Automation of Open Source Databases

Proprietary databases have been around for decades with a rich third party ecosystem of management tools. But what about open source databases? This whitepaper discusses the various aspects of open source database automation and management as well as the tools available to efficiently run them.

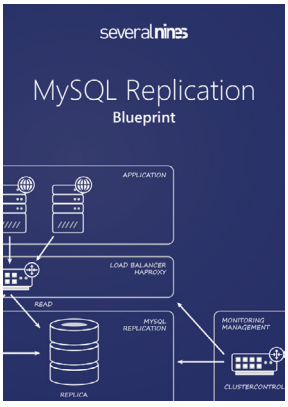
Download here



A Guide to Efficient Database Infrastructure Operations

Taking control of their data is every company's number one job. Database operations encompass a number of functions, including the initial deployment of a solution, configuration management, performance monitoring, SLA management, backups, patches, version upgrades and scaling.
In this white paper, we will discuss the operational aspects of running database infrastructures, and how companies can make these more efficient.

Download here

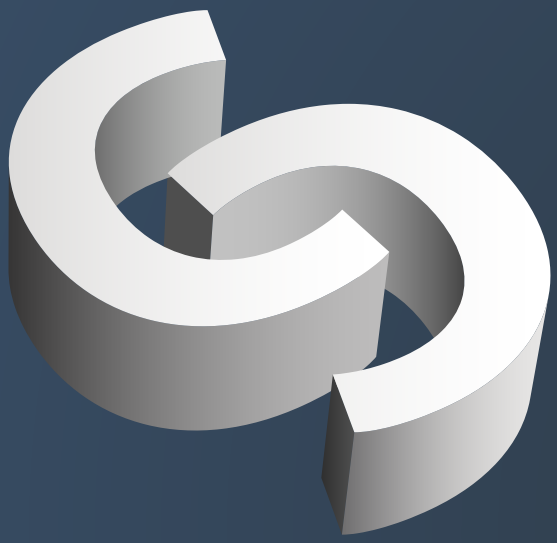


MySQL Replication Blueprint

The MySQL Replication Blueprint whitepaper includes all aspects of a Replication topology with the ins and outs of deployment, setting up replication, monitoring, upgrades, performing backups and managing high availability using proxies.

Download here

severalnines



Deploy

Manage

Monitor

Scale