Point in Time Restoration

In the unlikely event of a catastrophe, you can perform a point in time restoration of your ClustrixDB cluster on specific databases, tables, or the entire cluster. This should be handled with extreme care and we highly recommend contacting Clustrix Support to assist.

Prerequisites

Before you can use Point in Time Restoration there are a few prerequisites that should be in place for your cluster:

1. A binary backup from the ClustrixDB Parallel Backup should be available. Please see ClustrixDB Fast Backup and Restore for more information on this feature.
2. You will need to have binlogging enabled. You can find information on enabling binlogs at Configuring Replication.
3. Ensure your binlogs include the point in time to which you wish to restore.

Steps to restore to a specific Point in Time

Step 1 - Stop Writes to the Database(s) or Table(s) You Wish to Restore

Stop any slave processes that could be interacting with the database(s) or table(s) that need to be restored and disable binlogging. This ensures that the database(s) or table(s) do not take any writes while they are being restored. You also do not want any of the changes you make while setting up Point in Time Restoration to replicate to any slaves, so set the sql_log_bin to false. Setting sql_log_bin to false is for the current transaction only, so you should include that before every operation that modifies a table, just to make sure nothing replicates unexpectedly. The RESTORE operations themselves do not replicate.

```
sql> STOP SLAVE slave name;
sql> SET sql_log_bin = 'false';
```

Optional but highly recommended - Put cluster into read only mode.

```
sql> SET GLOBAL read_only = true;
```

Step 2 - Restore Database or Table

Restore a database (or table) from the most recent backup. You are going to use the lastest backup to restore the database(s) and/or table(s) that you need to recover. You can do this one of two ways. The first method is to drop the database (or table) to be restored and write the database (or table) fresh from backup like this:

```
sql> SET sql_log_bin = 'false';
sql> DROP DATABASE database;
sql> RESTORE database FROM 'ftp://username
@ftp server_address/path_to_backup'
```

If you are dropping and restoring a table you would replace database with database.table in the RESTORE.

You can restore multiple databases and/or tables, just separate each entry with a comma. For example:

```
sql> RESTORE foo, foo2, foo3 FROM 'ftp://username@ftp://server.com/backups/backupfolders/backup_file'
```

The second method is to restore to an alternate database (or table) and rename the table(s) to the original after verifying the data.

```
RESTORE database1.table1 AS database_backu
p.table1 FROM 'ftp://username@ftp
server_address/path_to_backup';
```

Example:

```
sql> RESTORE foo.bar AS foo_backup.bar FROM 'ftp://username@server.com/backups/backupfolders/backup_file'
```

Verify the data by any means you like, then drop the old database and rename the new table(s) to it.

```
sql> SET sql_log_bin = 'false';
sql> DROP TABLE database.table;
sql> RENAME TABLE database_back
up.table1 AS database.table1
```

Example:

```
sql> SET sql_log_bin = 'false';
sql> DROP TABLE foo.bar;
sql> RENAME TABLE foo_backup.bar AS foo.bar;
```

You can only rename tables, not entire databases. You CAN rename a table from one database to another, so if needed you could rename all tables inside a database to a new database but this can be time consuming by hand or require some scripting.

Step 3 - Extract the Binlog Name and Positions

Extract the binlog file and position from the metadata/binlogs folder on the ftp server. This will indicate the **start position** from the binlog from the point where the backup was created. **This is to be used in Step 5 and Step 7.**

On your FTP server run:

```
shell> cat path_to_ftp_backup/metadata
/binlogs
```

The output will be in the format binlogname.binlogfile:binlog-position

Example:

```
shell> cat /ftp/backups/newestBackup/metadata
/binlogs
foobin.000835:12345678
```

Step 4 - Find the Correct Binlog

Find the binlog containing the event you want to recover up to using the show binlog files command from a mysql client pointed to ClustrixDB. To find the correct binlog, locate the closest timestamp **prior to but not exceeding** the start time of the event (or time) that you wish to restore. Your desired event should be in that binlog. **Record this binlog name as it will be used as the stop position in end_logname position in step 5 and the logname in step 10.**

To find the timestamps run:

```
sql> show binlog files;
+---------------+-----------+-----------------------+
| File          | Size      | First Event Timestamp |
+---------------+-----------+-----------------------+
| foobin.000835 | 104857600 | 2018-01-24 08:01:16   |
| foobin.000836 | 104857600 | 2018-01-24 08:30:13   |
| foobin.000837 | 104857600 | 2018-01-24 08:51:46   |
```

If the event you're looking for was at 2018-01-24 08:50:14, you would select binlog foobin.000836 as an ending point from that example as it contains your event. foobin.000836 shows the closest timestamp before 8:50 and foobin.000837 begins after that time.

Step 5 - Dump Binlog with repclient

ClustrixDB ships with two tools to interact with binlogs. repclient, which is to dump and read the binlogs, and repserver, which can re-serve these binlogs, in this step you will dump the raw binlog files with repclient. First you are going to create a new directory inside the /clustrix directory and then change directory to the newly created directory, you will then run repclient to extract the correct portion of binlog to replay in Step 6. You are using the start position from Step 3 and the end position from Step 4. Use the following syntax:

```
repclient -dumpbinlog -logname '
binlog_name_from_step_3' -
end_logname 'binlog_name_from_st
ep_4'
```

Example:

```
shell> mkdir /clustrix/binlogs
shell> cd /clustrix/binlogs/
shell> repclient -dumpbinlog -logname 'foobin.000835' -end_logname 'foobin.
000836'
```

This will take some time to run, depending on the amount of logs that need to be dumped. Wait for the command to exit and return you back to a bash prompt. If successful, you will see a set of binlog files in your current working directory.

If you only have one binlog file, this process will not end as repclient reads up until the end of the log, which has not been recorded yet. The best way to handle this is to open another window and monitor the size of the file that is being created in the /clustrix/binlogs/ folder.

Once the file has stopped increasing in size for 10 seconds, you can use ctrl-c to manually stop the process.

For example:

```
shell> ls -lh /clustrix
/binlogs/
```

Step 6 - Change server_id

Change the server_id. This ensures that when you replay the events later. the cluster will not discard them as in master/master replication. The server_id must be unique in your infrastructure.

Show the cluster's server_id and **make a note of it so that you can reset it later.**

```
sql> SHOW GLOBAL VARIABLES LIKE '%
server_id%'
+---------------+------------+
| Variable_name | Value      |
+---------------+------------+
| server_id     | 1044953144 |
+---------------+------------+
1 row in set (0.01 sec)
```

Then change that variable to something unique.

```
sql> SET GLOBAL server_id = '0123456789';
```

Step 7 - Replay Binlog with repserver

Start an instance of repserver, the supplied binlog player utility. This will set up a replication master process to serve binlogs back to the cluster. It is a good idea to run repserver in screen, or a separate terminal as the command will need to run continuously.

```
shell> cd /clustrix
/binlogs
shell> repserver -
port 3307
```

You should see output like the following:

```
root@node001:/clustrix/binlogs$ repserver -port 3307
2018-01-30 22:56:22 INFO mysql/replication/repserver/repserver.c:620 have_stat(): Use Ctrl-C to exit
2018-01-30 22:56:22 INFO mysql/replication/repserver/repserver_proto.c:821 listen_on_port(): IPv4(0.0.0.0:3307)
```

DO NOT CLOSE THIS TERMINAL/SCREEN. Minimize the terminal and move on to the next step in a new window.

Step 8 - Create a New Slave

Create a new slave on ClustrixDB to read from repserver. The MASTER_LOG_FILE and MASTER_LOG_POS are from step 3 above.

```
sql> CREATE SLAVE 'foo_slave'
MASTER_LOG_FILE = 'foobin.000836'
, MASTER_LOG_POS = 12345678
, MASTER_HOST = 'localhost'
, MASTER_USER = 'root'
, MASTER_PORT = 3307;
```

Step 9 - Setup Replication Policy

IF YOU ARE NOT RESTORING THE ENTIRE CLUSTER TO POINT-IN-TIME: You will need to set up the system.mysql_slave_replication_policy and tabl e_replication_policy tables to include ONLY the database(s) and table(s) you want replicated. To do this, first record all values already in these tables **(thes e will need to be reset later)**. If you are restoring the whole cluster, you can go ahead and skip to Step 10.

```
sql> show variables like '%replication_policy%';
sql> select * from system.mysql_slave_replication_policy;
sql> select * from system.mysql_table_replication_policy;
```

RECORD THE OUTPUT FROM THESE TABLES and then drop any data that is in them. Do this for each database and table that you want restored to a point in time:

```
sql> DELETE from system.mysql_slave_replication_policy;
sql> DELETE from system.mysql_table_replication_policy;
sql> SET GLOBAL mysql_default_db_replication_policy = FALSE;
sql> SET GLOBAL mysql_default_table_replication_policy = FALSE;
```

To restore an entire database, follow this syntax:

```
INSERT INTO system.mysql_slave_replication_policy (dbname, allow) VALUES ('
dbname', TRUE)
```

Example:

```
sql> INSERT INTO system.mysql_slave_replication_policy (dbname, allow) VALUES ('foo', TRUE);
```

To restore tables, follow this syntax:

```
INSERT INTO system.mysql_table_replication_policy VALUES ('
slave name', 'dbname', 'tablename', TRUE)
```

Example:

```
sql> INSERT INTO system.mysql_table_replication_policy VALUES ('foo_slave', 'foo', 'bar', TRUE);
```

You can find more detailed information on setting up new slaves in Using ClustrixDB as a Replication Slave.

Step 10 - Determine Binlog Stop Position

Determine the stop position of the slave. For this step there is no "one size fits all" solution, as the required information location can change depending the type of query that was run, or the type of event from which you are recovering. The most basic method is to run a join on several tables to get the required information. The information needing to be supplied are binlog name, and the time to which you wish to restore. This query will bring you to within 500 transactions prior to the time you specify.

```
sql> SELECT commit_timestamp, mysql_binlog_index.commit_id, name, sequence, offset
     FROM system.mysql_binlog_index
     JOIN system.binlogs using(log_id)
     JOIN _replication.binlog_name_replication_commits using(commit_id)
     WHERE name = 'binlog_name'
     AND from_unixtime(mysql_binlog_index.commit_id>>32) < 'timestamp'
     ORDER BY mysql_binlog_index.commit_id DESC LIMIT 1;
```

The timestamp should be formated as '2018-02-01 14:00' and it represents the time to which you wish to recover. You also need to replace two instances of binlog_name with the name of your binlog, in this case foobin. This will give you the offset, which is required for the next step.

Example:

```
sql> SELECT commit_timestamp, mysql_binlog_index.commit_id, name, sequence,
offset
     FROM mysql_binlog_index
     JOIN binlogs using(log_id)
     JOIN _replication.foobin_replication_commits using(commit_id)
     WHERE name='foobin'
     AND from_unixtime(mysql_binlog_index.commit_id>>32) < '2018-02-28 12:00'
     ORDER BY mysql_binlog_index.commit_id DESC LIMIT 1;
+---------------------+---------------------+--------+----------+--------+
| commit_timestamp    | commit_id           | name   | sequence | offset |
+---------------------+---------------------+--------+----------+--------+
| 2018-02-01 01:27:31 | 5839789947966173188 | foobin |     4443 |    106 |
+---------------------+---------------------+--------+----------+--------+
1 row in set (3.13 sec)
```

Record the output of the offset column. This will be used as your MASTER_LOG_POS in Step 11. In most cases this will give you enough information to do your point in time restoration and you can **PROCEED TO STEP 11.**

Step 10A - Extract the Exact Binlog Stop Position with mysqlbinlog or repclient

This step is optional: If you do not need recovery to the exact transaction, please skip to Step 11.

IT IS HIGHLY RECOMMENDED THAT YOU CONTACT CLUSTRIX SUPPORT AND LET THEM PERFORM THE NEXT OPERATION!

If you need to recover to a specific transaction, manually inspect the binlog using mysqlbinlog or repclient. This is a fairly simple task if you are using statement-based replication (SBR), but can be rather tricky for row-based replication (RBR) as any non-DDL events show up in RBR as encoded hex. If the event you are trying to recover from is a DDL (ALTER, DROP, CREATE) this will always be SBR. To locate a specific transaction in an RBR stream that is not a DDL please contact Clustrix Support for assistance. To inspect binlogs use the filename from step 4 in the logname argument, like the example below:

mysqlbinlog cannot decode ClustrixDB RBR events. Use ClustrixDB repclient instead.

If you are trying to recover from a dropped database named foobase, follow this sample.

```
shell> mysqlbinlog /clustrix/binlogs/foobin.000836 | grep -b10
"foobase" | less
```

You will see an output like the below:

```
12429169-/*!*/;
12429176-# at 45116760
12429190-# at 45116813
12429204-# at 45116853
12429218-#130211 16:54:27 server id 2101898466  end_log_pos 45116913    Query   thread_id=1705364482
exec_time=0     error_code=0
12429331-SET TIMESTAMP=1360630467/*!*/;
12429362-COMMIT
12429369-/*!*/;
12429376-# at 45116913
12429390-#130211 16:54:40 server id 2101898466  end_log_pos 45117003    Query   thread_id=1705364482
exec_time=0     error_code=0
12429503:use foobase/*!*/;
12429521-SET TIMESTAMP=1360630480/*!*/;
12429552-SET @@session.sql_mode=0/*!*/;
12429583:drop database foobase
12429605-/*!*/;
12429612-# at 45117003
12429626-#130211 16:54:40 server id 2101898466  end_log_pos 45117030    Xid = 5843863418521626628
12429713-COMMIT/*!*/;
12429726-# at 45117030
12429740-#130211 16:54:54 server id 1044953188  end_log_pos 45117089    Query   thread_id=1658178562
exec_time=0     error_code=0
12429853-SET TIMESTAMP=1360630494/*!*/;
12429884-SET @@session.sql_mode=524288/*!*/;
12429920-BEGIN
12429926-/*!*/;
```

As the event you are trying to recover from is a DROP DATABASE foobase in the example above, you would look for that command like this: "12429583: drop database foobase" and then look for a transaction position before this. So from the above example you want the line that looks like this:

> *12429376-# at 45116913*

The number after the "at" is the number to record: 45116913

Below is a condensed code snippet showing the drop database and the line you are looking to extract. **Record this information.**

```
12429376-# at 45116913
12429390-#130211 16:54:40 server id 2101898466  end_log_pos 45117003    Query   thread_id=1705364482
exec_time=0     error_code=0
12429503:use foobase/*!*/;
12429521-SET TIMESTAMP=1360630480/*!*/;
12429552-SET @@session.sql_mode=0/*!*/;
12429583:drop database foobase
```

Step 11 - Read Binlogs with New Slave

Now you need to replay the binlogs up to the point you specified in the previous step you do this by starting the new slave, and **ONLY** that slave. The UNTIL command is run with the stop position extracted in Step 10.

```
sql> START SLAVE 'foo' UNTIL MASTER_LOG_FILE = 'foobin.000836', MASTER_LOG_POS = '106';
```

The slave will run up until the transaction before (if LOG_POS is defined properly) the catastrophic ALTER, DROP, or CREATE found in the binlog.

Step 12 - Clean Up

This last step is the cleanup of all the settings and changes made previously:

- Set mysql_default_db_replication_policy and mysql_default_table_replication policy variables to the values found in step 9.
- Stop and drop the slave you created earlier foo.
- Re-start any other slaves that were previously running.
- Set the server_id back to what it originally was. **This was recorded in Step 1.**

  ```
  sql> SET GLOBAL server_id = '1044953144';
  ```

- Re-enable writing to binlogs:

  ```
  sql> SET sql_log_bin = 'true';
  ```

- If you put the cluster into read only mode, enable writes again.

```
sql> SET GLOBAL read_only = false;
```

Go back to the window (or screen) running repserver and stop that with ctrl-c.