SQream DB Administrator Guide

Version 3.3

Overview

This guide is intended for SQream DB administrators.

The guide will go through the DBA main tasks, as well as describing some of the best practices in SQream DB.

This guide is a complementary guide to the SQL Reference.

For further support please contact support@sqreamtech.com or your account manager.

Table of Contents

Administration Guide

This document is mostly a conceptual overview and recommendations on best practices regarding SQream internal behavior. For SQL syntax and supported features, please refer to the SQL Reference Guide.

All keywords in are **case insensitive**.

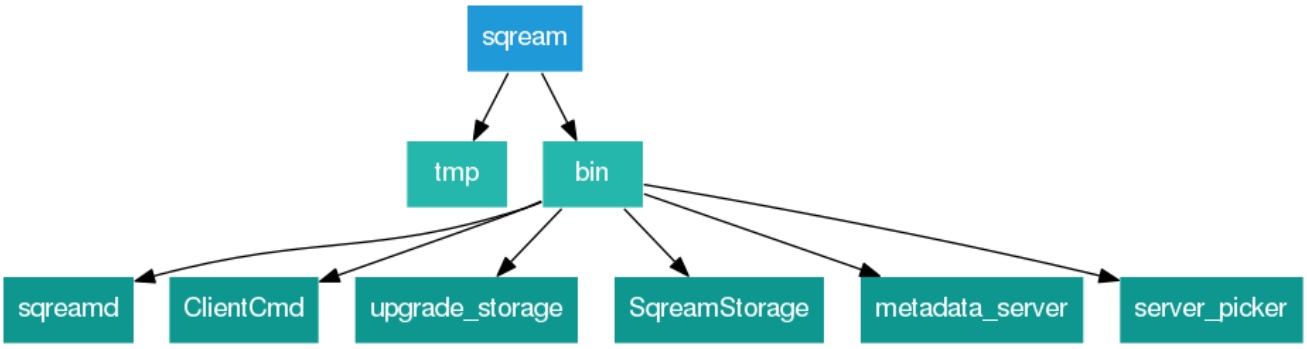
Getting Started

Understanding the SQream DB environment

The SQream environment is usually made up of two folders - the installation directory and the storage cluster.

The installation directory

This folder contains the SQream DB binary applications.

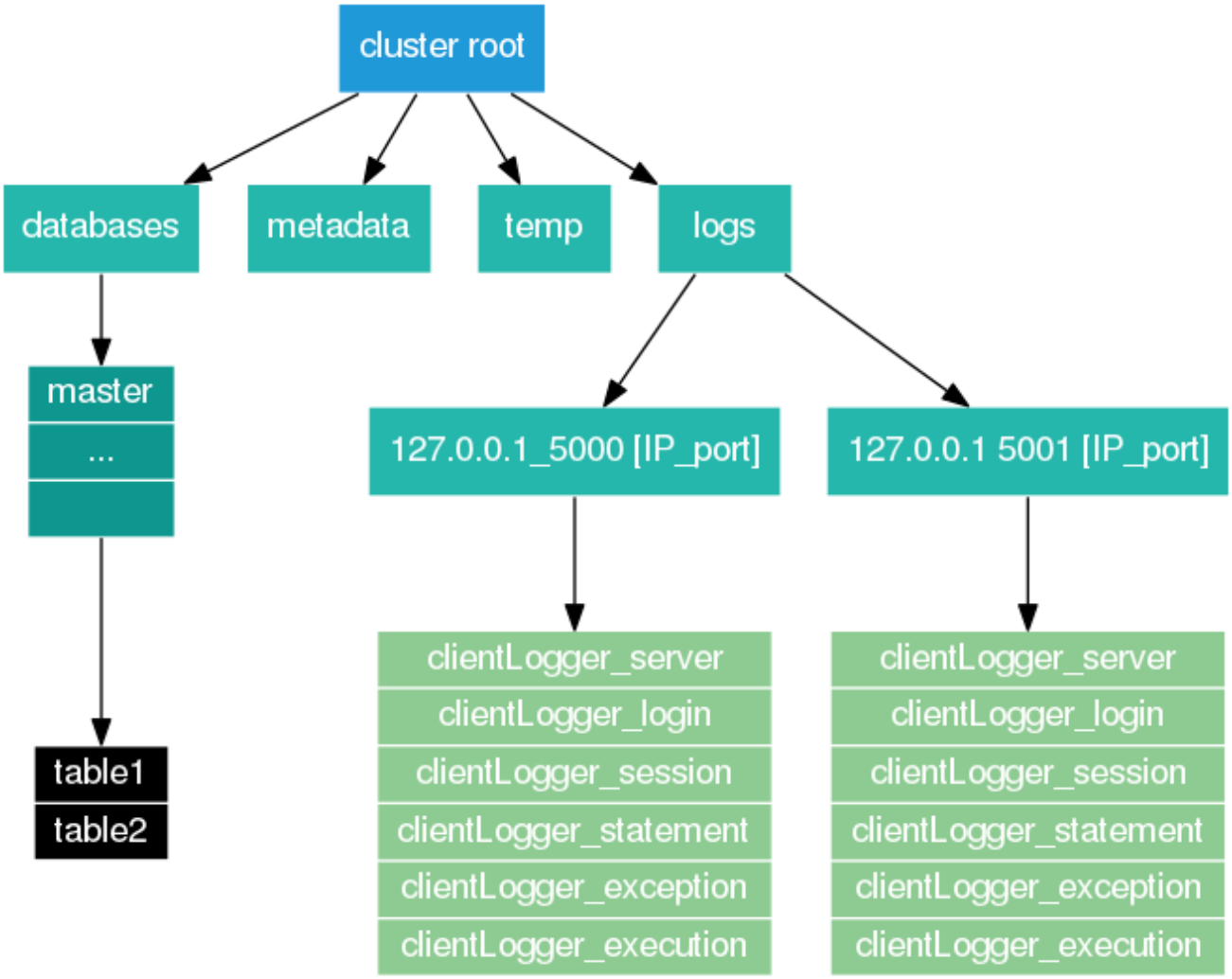


The main applications found in the binary directory are:

Executable Name	Description
`sqreamd`	The sqreamd server daemon
`ClientCmd`	Command line client
`upgrade_storage`	Metadata storage upgrader to be used between versions
`SqreamStorage`	Storage utility to create new clusters and restore access
`metadata_server`	Metadata server for clustered installations
`server_picker`	Load balancer for clustered installations

The storage cluster root

This directory contains the entire database storage.

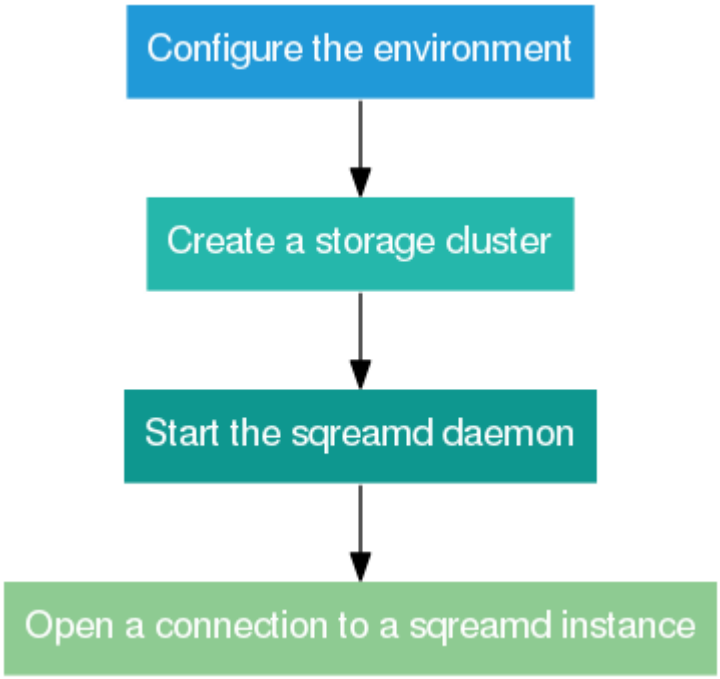


Directory	Description
`databases`	The files containing all databases, tables, columns, chunks, etc.
`metadata`	Internal metadata structures for accessing data from disk
`temp`	Temporary spools
`logs`	The **logs** folder contains subfolders per SQream instance with folder name = [IP_port]. One folder per running SQream instance.
`127.0.0.1_5000 [IP_port]`	Contains all logs files of a specific SQream instance

Setting up SQream DB

If you would like to deploy SQream DB on AWS or Azure please contact SQream directly at info@sqream.com.

Make sure that you are running on a 64-bit Linux operating system, and have an Nvidia GPU installed.
See the Hardware Requirements page to verify your system meets the minimum requirements for running SQream DB.



Create the cluster

1. Enter the `sqream` directory and create a storage cluster
2. `./bin/SqreamStorage -C -r <full path to new cluster>`

Example: Create storage cluster

If your main storage is on `/mnt/storage`, you can run
`./bin/SqreamStorage -C -r /mnt/storage/sqream_storage`

Configure the instances

Each SQream DB daemon must run against a configuration file.

Below is a sample **minimal** configuration file. It is recommended that it be placed in `/etc/sqream/sqream_config.json`. This file will start the server on port **5000**, against GPU **#0**.

Example minimal configuration JSON

```
{
    "compileFlags":  {
    },
    "runtimeFlags":  {
    },
    "runtimeGlobalFlags":  {
    },
    "server":  {
        "licensePath"  :  "/etc/sqream/license.enc",
        "port":  5000,
        "cluster":  "/mnt/storage/sqream_storage",
        "gpu":  0,
        "ssl_port":  5100
    }
}
```

- JSON files can not contain any comments
- When altering the JSON file, pay close attention to the field-separating commas at the end of the lines.

Permanent instance options may be placed in this file, based on consultation with SQream support or through the configuration utility.

Configuring instances parameters

Table 20. Common parameters setting

Flag Name	Description	Default Value	Range of Values	Remark
`spoolmemorygb`	Select what size spool SQream DB can use for writing intermediate results to RAM, in GB	`128gb`	`1-machine ram size`	Should consider total ram size and number of SQream instances
`insertParsers`	Set the number of parsing threads to be launched for each file, during the bulk load process	`4`	`1-32`	Should consider total ram size and number of SQream instances
`insertCompressors`	Set the number of compression threads to be launched for each file, during the bulk load process	`4`	`1-32`	Should consider total ram size and number of SQream instances
`statementLockTimeout`	Set the number of seconds SQream will wait for a lock before returning an error	`3`	`1-no limit`	
`showFullExceptionInfo`	Show complete error message	`FALSE`	`TRUE/FALSE`	Enabling this setting will often show more detailed error message
`initialSubscribedServices`	List of services the instance will be subscribed to	`sqream`	`list of strings`	Example: "sqream,etl_service,query_service"
`useLogMaxFileSize`	Defines whether SQream logs should be cycle every logMaxFileSizeMB size	`TRUE`	`TRUE/FALSE`	
`logMaxFileSizeMB`	Set the size of SQream logs cycle	`20`	`>0`	Set in MB

Starting and Stopping the SQream DB daemons

The procedure for starting and stopping the SQream DB daemons varies depending on your installation type:

- SQream based on Docker
- SQream based on RPM

SQream based on Docker

Use the start and stop options of the sqreamd command as described in the sqream-console commands in Start the SQream Console.

SQream based on RPM

A SQream RPM installation uses the Monit service to control the SQream services.

To start the SQream demons (Metadata, Server Picker and SQreamD):

```
$ sudo systemctl start monit
```

To check the SQream demons (Metadata, Server Picker and SQreamD):

- Option 1:

  ```
  $ sqream_status
  ```

  Expected output:

  ```
  SQreamDB monit.service: ACTIVE pid=6406                    [  OK  ]
  SQreamDB metadataserver.service: ACTIVE pid=6427           [  OK  ]
  SQreamDB serverpicker.service: ACTIVE pid=6579             [  OK  ]
  SQreamDB sqream1.service: ACTIVE pid= 7440                 [  OK  ]
  ```

- Option 2:

  ```
  $ ps axuwww | grep -v 'grep\|tail\|monitor' | grep -i 'sqreamd\|metadata_
  server\|server_pick\|monit'
  ```

  Expected output:

```
root      7690  0.0  0.0 191916  2464 ?        Ss   15:27   0:00 /bin/su -
sqream -c /bin/nohup /usr/local/sqream/bin/metadata_server &>>
/var/log/sqream/metadataserver.log
sqream    7693  0.0  0.0 115304  1680 ?        Ss   15:27   0:00 -bash -c
/bin/nohup /usr/local/sqream/bin/metadata_server &>>
/var/log/sqream/metadataserver.log
sqream    7708  0.2  1.8 1110888704 183644 ?   Sl   15:27   0:02
/usr/local/sqream/bin/metadata_server
root      8105  0.0  0.0 191916  2456 ?        Ss   15:27   0:00 /bin/su -
sqream -c exec /usr/local/sqream/bin/server_picker 127.0.0.1 3105 &>>
/var/log/sqream/serverpicker.log
sqream    8116  0.2  1.5 34887308 156592 ?     Ssl  15:27   0:02
/usr/local/sqream/bin/server_picker 127.0.0.1 3105
root      8450  0.2  0.0 131124  3452 ?        Ssl  15:34   0:01
/usr/bin/monit -I
root      8453  0.0  0.0 191916  2460 ?        Ss   15:34   0:00 /bin/su -
sqream -c exec /usr/local/sqream/bin/sqreamd -config /etc/sqream/sqream1\_
config.json &>> /var/log/sqream/sqream1.log
sqream    8456  0.6  5.5 1111154756 542644 ?   Ssl  15:34   0:04
/usr/local/sqream/bin/sqreamd -config /etc/sqream/sqream1_config.json
```

To stop all SQream services:

```
$ sudo /usr/local/sqream/config/sqream-stop.sh
```

Expected output:

```
==>Stopping SQreamDB Services
UNIT                    LOAD   ACTIVE SUB    DESCRIPTION
   • metadataserver.service loaded Metadata Server For SQreamDB
   • serverpicker.service   loaded Server Picker - Load Balancer For SQreamDB
   • sqream1.service        loaded SQream SQL Server

LOAD   = Reflects whether the unit definition was properly loaded.
ACTIVE = The high-level unit activation state, i.e. generalization of SUB.
SUB    = The low-level unit activation state, values depend on unit type.
3 loaded units listed. Pass --all to see loaded but inactive units, too.
To show all installed unit files use 'systemctl list-unit-files'.
==>SQreamDB Services have been stopped
```

To stop a specific SQream service:

1. Stop the Monit service:

   ```
   $ sudo systemctl stop monit
   ```

2. Stop the specific SQream service, for example stopping sqream1:

   - Option 1:

     ```
     $ sudo systemctl stop sqream1
     ```

- Option 2:

```
$ pkill -9 sqreamd
```

To restart a service that was stopped:

- Option 1:

  Start the Monit service and monit will start all stopped services:

```
$ sudo systemctl start monit
```

- Option 2:

  Start the stopped service manually:

```
$ sudo systemctl start sqream1.service
```

Connect to the SQream DB server with ClientCmd

To connect to the database server, you may use the ClientCmd command line interface by running

ClientCmd command line arguments

```
./bin/ClientCmd --user=<username> --password=<password> --database=<database
name> --host=<host> --port=<port> --service=<service_name>
```

Connect to the `master` database:

Connecting to `master` on a local host running SQream DB on port 5000 via service 'etl_service'

```
./bin/ClientCmd --user=sqream --password=sqream --database=master --
host=127.0.0.1 --port=5000 --service=etl_service
```

Using SSL Server Authentication with SQream

From version 2.1, SQream DB supports secure sockets layer (SSL) encryption and authentication for connections to its cluster via JDBC and ODBC drivers.

To use this option, the server must first be set-up to accept SSL connections.

Enabling SSL server authentication within SQream Instance:

Configure the SERVER flag ssl_port in the SQream instance configuration file to the needed port number.

For Example:

```
{
"server":{
    "port":  5001,
    "ssl_port":  5100,
    "cluster":  "/path/sqream_cluster",
    "gpu":  0,
    "licensePath":  "/path/license.enc"
}
}
```

Restart the SQream DB daemons after making this configuration change

Configure SSL Authentication for JDBC/ODBC drivers

Adding SSL to JDBC

1. Add ssl=true in the connection string
2. Change the port to the SSL port

When connecting via load balancer, the default endpoint for SSH is port 3109

Example for direct connection

```
jdbc:Sqream://hostname:5100/master;user=sqream;password=mypassword;ssl=true;
```

Example for connection to load balancer

```
jdbc:Sqream://hostname:3109/master;user=sqream;password=mypassword;service=sqre
am;cluster=true;ssl=true;
```

Adding SSL to ODBC

In Windows, make sure the SSL checkbox is selected in the DSN settings.

In Linux, add Ssl=true to the connection string

Linux ODBC connection string sample

```
Driver=
{libODBCDrv.so}:Server=hostname:Port=5100:Database=master:User=sqream:Password=
mypassword:Ssl=true:Service=sqream
```

Highly available installations

Contact your SQream representative for further information about installing our highly available solutions.

Operations

Upgrading a version

Here are the necessary steps that ensure a smooth upgrade of your SQream DB version

1. Stop SQream instances on all servers
2. On each node that SQream is installed, unpack the new tarball alongside the old SQream DB directory.
   For example:

```
$ cd /home/sqream
$ mv sqream sqream-old
$ tar xf sqream-<version>.tar.gz
```

3. Repeat the above step for each node that the SQream DB executables exist
4. It may be necessary to run the metadata upgrade utility.
   (This may take a few moments)

```
$ cd sqream/bin
$ ./upgrade_storage <path to sqream storage cluster>
```

5. Restart the services

Key administration concepts

See Concepts above

Monitoring the system

Because SQream DB can be run in a distributed setting, all nodes should be monitored to ensure smooth operation. It is possible to monitor SQream DB with third party tools like Zabbix, Nagios and others, but also through the OS and SQream DB directly.

From the OS

See Identifying which SQream daemons are running

From each node

See connections to the server

You can monitor existing connections to the database by using the `show_connections()` utility function:

```
SELECT  show_connections();
```

Table 21. Sample result from `show_connections()`

ip	conn_id	conn_start_time	stmt_id	stmt_start_time	stmt
192.168.0.93	19	2017-06-22 18:56:54	14	2017-06-22 18:56:54	select show_connections()
192.168.0.93	17	2017-06-22 18:56:48	-1	2017-06-22 18:56:48	

Show server / cluster status

The `show_server_status()` utility function can be used to see which statements are running across the cluster, across all databases.

If no queries are running, this query will return 0 rows in the result set.

```
SELECT  show_server_status();
```

Table 22. Sample result from `show_server_status()`

service_id	connection_id	server ip	server port	database_name	user_name	client ip	statement id	statement	statement start time	statement status	statement status start
sqream	32	192.168.0.93	5000	faa	sqream	192.168.0.1	25	SELECT Year	Carrier	destCityName	COUNT(DISTINCT originCityName) from ontime JOIN l_airport_i

Possible statement status values

Status	Description
Executing	The statement is in execution, awaiting results
Preparing	The statement is compiling, and is awaiting execution
Waiting	The statement is waiting in the queue for execution

The DBA can use the show server status output as a baseline for identifying locks and if needed to stop running statements (based on the **server ip : server port** and **statement_id** columns).

By running a query

Running a query, even the most basic one, should give you an indication if a server is up. If you immediately get "Connection refused" or similar, the server is down.

```
SELECT  1;
```

Stopping existing statements

The **stop_statement()** utility function can be used to cancel or stop a running statement before it finishes.

Usage

- Identify the running statement ID and server IP and port (see **show server status** or **show_connections** above)
- From the same server/port combination - run the stop_statement command:

```
SELECT  stop_statement(42);
```

Logs

SQream DB generates the following log files for the DBA everyday work

Log files:

- **clientLogger_statement** - Keeps track of statements being run in the server, including success/failure, number of rows returned from the query and number of rows processed by the query.
- **clientLogger_server** - Keeps track of server start/stop times
- **clientLogger_session** - Keeps track of different sessions
- **clientLogger_login** - Tracks user login times and IP addresses
- **clientLogger_exception** - Tracks system exceptions
- **clientLogger_execution** - Tracks heavy statements execution statistics (over nodeInfoLoggingSec flag)
- **clientLogger_debug** - Logs debug information. Default debug log level = 4 (INFO). The debug log level can be changed running a statement (e.g., SET logDebugLevel = 6;). No need to restart the SQream instance.
- **metadata_server.log** - Tracks metadata server status and executions queue

File names: Log file names contain a day and time stamp. For example: **clientLogger_statement_20181108_000000.log**

Statement log messages are cut at a length of 10,000 characters. This is to allow loading statements logs files (CSV) into External Tables of SQream. Remember: Row length is limited to 10,240 characters.

Log file configuration:

From V2.18.1 the following default behavior is configured:

- Each SQream instance generates a separate set of log files into a dedicated folder (folder name contains IP and port). For example: /home/sqream/sqream_storage/sqreamdb/logs/**127.0.0.1_5000**.
- Log file rotation: One separate log file per calendar day. New file is generated at midnight.

Related runtimeGlobalflags:

- **useClientLog**: This flag is set the TRUE (by default) to activate the new logging framework.
- **useLogFileRotateHourOfDay**: Enable/disable the rotation of log files. By default this flag is set to TRUE to allow seperate files for separate days.
- **logFileRotateHourOfDay**: Rollover time parameter. By default this flag is set to 0

(=rotation at midnight). Set in the hour of the day. Possible range of values: 0-23.

- **enableLogDebug**: Activate / deactivate the debug log. Default value = true;
- **logDebugLevel**: Supported Log Levels are: 0 = [SYSTEM], 1 - [FATAL], 2 - [ERROR], 3 - [WARN], 4 - [INFO], 5 - [DEBUG], 6 - [TRACE]. Default = 4 (INFO);

Support Utilities

Report Collection

This support utility is used to collect logs and/or the leveldb at a client's site. The generated tar file can then be sent to the SQream support team for further investigation.

- Output file format: report_[date]_[time].tar

Can run as a utility function or executable, allowing to collect information also if the SQream server is not running. Supported collection modes:

- log = only log files are collected
- db = only leveldb
- db_and_log = both log files and leveldb are collected

Syntax and example when running as a utility function:

```
SELECT  report_collection('</pathToOutputFolder>',  '<mode>');

SELECT  report_collection('/home/sqream/log_collection','log');  )
```

Syntax and example for executable. The executable works only when no SQream instance is running:

```
./bin/report_collection <pathToSqreamDb> </pathToOutputFolder>
<mode>./bin/report_collection /home/sqream/sqream_storage/ /home/sqream/log_
collection log
```

Export Reproducible Sample Data

This support utility is used to collect data in order to reproduce a problematic query in a support lab (not only onsite at customer's site). Typically used for query issues that are data related. It runs a query, collects the data and stores the data in a small SQream DB (compressed into a single tar file). This file together with the query can be used in a remote system (support lab) to recreate and investigate the issue.

The output folder contains both the final tar file and data before its compression. It is sufficient to send the tar file. Works as utility function. No executable.

Syntax and example:

```
select  export_reproducible_sample('</pathToOutputFolder>',  'sql query1',
'sql query 2',  ..);

select  export_reproducible_sample('home/sqream/data_collection',  'select *
from t');
```

Copyright