QLDS Manager Documentation

Release 2.2.5

Piotr Rzeczkowski

December 20, 2015

Contents

QLDS Manager is a tool dedicated for Quake Live Server administrators

Core functionality allows to:

- Download SteamCMD and QL server files to specified location
- Update QL server files
- Easily configure multiple servers

Additional functionality:

- Generating supervisord configuration (requires supervisord)
- Keep servers alive
- Start/stop/restart server
- Connect to rcon console (requires zmq)

CHAPTER 1

Documenatation

1.1 Installation

1.1.1 Requirements

- Cement
- Python 3+

1.1.2 Optional requirements

- ØMQ - if you want to use RCON
- Supervisor - if you want to start/stop servers

1.1.3 Preferred way

Use pip

```
pip3 install qlds-manager
```

It will install `qldsmanager` script, that will allow you to use Manager as command

1.1.4 Use setuptools manually

For that method to work, you'll need setuptools installed for your version of Python

Go to the QLDS-Manager dir and run

```
python3 setup.py install
```

Just like in pip installation, it will create script to use Manage as command

1.1.5 Manual

You can install all dependencies manually and use QLDS Manager with

```
python3 /path/to/qlds-manager/qldsmanager
```

1.2 Configuration

1.2.1 Manager configuration

Here you can find informations about QLDS Manager configuration

Default values

Description	Location
steamcmd	~/steamcmd
QL files	~/QLserver
Servers configuration	~/.qldsmanager/servers
Rcon configuration	~/.qldsmanager/rcon
Supervisord	~/.local/bin/supervisord
Supervisorctl	~/.local/bin/supervisorctl

Setting new values

To set new paths and file locations, run Manager like:

```
qldsmanager configure <arguments>
```

Note: You can get list of available arguments with

```
qldsmanager configure --help
```

It will display help block

Available arguments are

Argument	Type	Description
`--steamcmd`	Directory	Location for SteamCMD
`--ql`	Directory	Location for QL server files
`--servers`	File	Server list configuration file
`--rcon`	File	Optional rcon list configuration file
`--supervisor`	Executable	Supervisord executable (eg.: `/usr/bin/supervisord`)
`--supervisorctl`	Executable	Supervisorctl executable (eg.: `/usr/bin/supervisorctl`)

Note: You can set more than one argument at a time in any order, eg..:

```
qldsmanager configure --servers ~/qlds_servers --ql ~/qlds_ql_files
```

1.2.2 Configuration syntax

Configuration is held in simple .ini file

The file is parsed before passing it to command like, so you can use or define parameters

Parameters

You can set parameters used later in configuration. For example, if you want to use same passwords in every server, you can put something like that

```
[parameters]
rcon_password = secret-rcon-pass
stats_password = stats-password
private_password = elite-only
```

Now to use those parameters in configuration, call them with `${parameters.*}` where * is parameter name

Note: To get parameter, use

```
${parameters.*}
```

Global parameters

There are 2 pre-defined global parameters, which cannot be overwritten. Those parameters are:

- `${loop}`
- `${global.loop}`

`${loop}` is gives current iteration in server group, so its value increases by 1 after each server parsed in servers group

`${global.loop}` is global iteration pointer, so its value increases by 1 after each server parsed, regardless of group it's in

Self parameters

With the parameters above, you can set some contants. But what if you want to get current server configuration value?

Instead of using `${parameter.*}` use `${server.*}`.

Note: To get server variable, use

```
${server.net_port.*}
```

Math

You can use mathematical expressions in You configuration!

Each expression has to be between >> and <<

Note: Example math expression, where we set zmq_rcon_port 1000 higher than server's port

```
zmq_rcon_port = >>${server.net_port} + 1000<<
```

Sections

In addition to `[parameters]` section, all other sections have to start with pre-defined words. Those words are:

Word	Purpose
`server`	Single server configuration
`defaults`	Server's group defaults/fallback configuration
`extra`	Extras added to server's group

1.2.3 Servers

Configuring server is easy. One section is one server. In section you provide arguments that are passed to command line.

The parameter `net_port` is required!

Single server configuration

Tip: If you don't want the server so start automatically when supervisor starts, add `__autostart = 0` in its configuration. This variable will be ignored in command line but will tell Manager to disable autostart.

First thing you have to notice is section name. It's `server:public1`. `public1` is your local server identifier. You can use it to start/stop server, attach rcon console or even server process. It's used for `fs_homepath` too, so each server gets its own directory with config files.

Then goes the listing. Even if you set `sv_hostname` in server.cfg, commandline overrides it, so you can use some default server.cfg and use Manager for instance-specific variables.

Look at the example below:

Note: Each server's section has to start with word `server`

```
[server:public1]
zmq_rcon_enable = 1
zmq_rcon_password = ${parameters.rcon_password}
zmq_rcon_port = >>${server.net_port} + 1000<<
zmq_stats_enable = 1
zmq_stats_password = ${parameters.stats_password}
zmq_stats_port = ${server.net_port}
net_port = 27960
sv_hostname = QLDS Managed Server
sv_privatepassword = ${parameters.private_password}
```

As you can see, all passwords are passed from parameters and rcon port is 100 higher than server port (27960 + 1000 = 28960)

Server grouping

If you have multiple servers with similar configuration, you can create group for them. To start 3 servers with same configuration as in previous example, you have to create group first. Group name has to start with word `defaults`. Then, in server's configuration name, after its name, add name of the group to extend delimited with colon. Eg.:

Note: Each servers group has to start with word `defaults`

```
[defaults:publics]
zmq_rcon_enable = 1
zmq_rcon_password = ${parameters.rcon_password}
```

```
zmq_rcon_port = >>${server.net_port} + 1000<<
zmq_stats_enable = 1
zmq_stats_password = ${parameters.stats_password}
zmq_stats_port = ${server.net_port}
net_port = >>27959 + ${loop}<<
sv_hostname = QLDS Managed Server #${loop}
sv_privatepassword = ${parameters.private_password}

[server:public1:publics]

[server:public2:publics]

[server:public3:publics]
sv_hostname = Custom Hostname
```

Notice that `net_port` starts with *27959*, not default *27960*. It's because `${loop}` is added to it, and this parameter starts from 1

`public3` configuration will override group's hostname, so you'll end with servers:

Port	Hostname
27960	QLDS Managed Server #1
27961	QLDS Managed Server #2
27962	Custom Hostname

Additional parsing of group variables

Let's say you want all server hostnames to use same schema for naming, even when you override it for single server. Of course you can remember, when overriding, to set hostname according to schema, but there's a better way. The `extra` section.

`extra` section takes prepared server configuration, looks for defined variables and changes them according to its own schema.

Note: The `extra` section has to "extend" servers group

Goal: use schema `-== QLDS Managed [server name] #[number] =-` for servers

First of all, we need servers group:

```
[defaults:publics]
net_port = >>27959 + ${loop}<<
sv_hostname = Server
```

It's simplified as much as it can be. Now, we have to "extend" it. Create `[extra:publics]` section. `publics` is the name of servers group.

```
[extra:publics]
sv_hostname = "-== QLDS Manager ${self} #{$loop} =-"
```

As you can see, there is `${self}` parameter introduced. It's replaced with original value of variable (sv_hostname) in this case, so you" end up with desired hostname