Showergel

Release 0.2

Martin Kirchgessner

May 14, 2021

CONTENTS:

Showergel is made to live aside Liquidsoap: while a Liquidsoap script creates a radio stream, Showergel provides complementary features like logging or occasional scheduling, with a (minimalist) Web interface. It is made to run on a Linux box (with systemd) dedicated to your radio stream.

Note: With Showergel you still have to write/tune the Liquidsoap script that will fit your radio. That is, the set of sources (playlist, input.harbor, ...) and operators (random, switch, fallback, ...) that fits your programs and schedule. This documentation only provide a starter script, covering Liquidsoap's basics and Showgel's integration.

CHAPTER

ONE

DESIGN PRINCIPLES

Showergel is meant for community and benevolent radios. Therefore we'll keep it small and simple:

- Showergel is intended to run on the same machine as Liquidsoap.
- The REST/Web interface is served by the Bottle framework, because it's enough and allows keeping everything in a single process.
- Showergel's data is stored in SQLite because its architecture fits perfectly our use case (everything in a single, local file).
- Scheduling is delegated to APScheduler, who also needs SQLAlchemy to access SQLite, so we use SQLAlchemy too.
- Showergel will not hold your music and shows collection. For that matter we suggest Beets, you can find examples of its integration with Liquidsoap in Liquidsoap documentation.

1.1 Installing Showergel

Installing Showergel requires

- a Linux box relying on systemd (ie. a recent mainstream distribution), with only user access,
- Python, at least v3.7,
- check pip is available, by calling `pip3 --version` (or `pip --version`)
- a running Liquidsoap radio - version 1.4.4 is Showergel's best friend.

Note: We assume you already know the basics of Liquidsoap and its script language. If you have never played with only Liquidsoap, we advise you read at least its quick start guide. Then you might jump to *Showergel's quickstart.liq*.

Install Showergel by running `pip3 install showergel` (maybe replace `pip3` by `pip`).

Then you'll need to set-up your first instance. Before that, let's explain briefly what it means.

1.1.1 What's a Showergel instance ?

Technically, it is an instance of the `showergel` program. It's an HTTP server: it is able to serve an interface you can open in your browser, and to answer queries called from your Liquidsoap script.

Functionally, an instance is the companion of *one* Liquidsoap stream. The two programs will communicate with each other to produce and follow this stream. An instance relies on some configuration (a `.toml` file) and a database (an SQLite file). It can be installed as a system service, named after information you provide at set-up time.

If you are running multiple Liquidsoap streams on the same machine, you'll have to set up one Showergel instance per stream. In that case,

- each instance should have a different name
- each instance should run on a different port number

We advise you to put all instance's files (configuration, DB, logs) in the same folder as the corresponding `.liq` file. When running multiple Liquidsoaps, create a folder per instance.

1.1.2 Create an instance with the interactive installer

Run the interactive installer by calling `showergel_install`. It will explain on the terminal what is happening and what to do from here. If you stick to defaults, the instance's basename will be `showergel`, so the installer will:

- create a database (`showergel.db`) and a configuration file (`showergel.toml`) in the current directory,
- create a systemd user service called `showergel` ; in other words you can `systemctl --user status|start|stop|restart showergel`.
- enable the service and systemd's lingering so Showergel will start automatically at boot time.

The installer allows you to create another systemd user service, for your Liquidsoap script. This is recommended, because then systemd will automatically launch both Showergel and Liquidsoap, and restart them when a crash happens. If you do so, the installer creates two systemd services with the same basename: for example, `radio_gel` (Showergel service associated to `radio`) and `radio_soap` (wrapper for the Liquidsoap script you provided for `radio`).

If you choose to not create a service, you will have to (re)start Showergel manually by calling `showergel showergel.toml`.

Before exiting, the installer gives a recap of its actions.

Note: Please pay attention to the installer's recap: it lists how to run/stop Showergel and gives pointers to important files. Read it twice and copy this information safely.

Unless you configured another port, Showergel will be available at http://localhost:2345/. Showergel's and Liquidsoap's configurations are coupled. In perticular, they should define which port/URL should be used to contact each other: see *Configuring Showergel* and *Showergel integration in Liquidsoap scripts*.

If you installed Showergel as a system service, don't forget to restart showergel's service after editing its configuration file. If you installed your Liquidsoap script as a system service, restart this service after editing the script - and after running `liquidsoap --check my_script.liq` ! In both cases, log files are in the same folder as instance's other files.

1.1.3 Install for back-end development

Depencencies, installation and packing is done by Poetry. Once Poetry is installed, create a Python3 environment, activate it, and run `poetry install` from a clone of Showergel's repository.

When developping, your Liquidsoap script and Showergel should be launched manually. Run `showergel_install --dev` to create an empty database (`showergel.db`) and a basic configuration file (`showergel.toml`) in the current folder. Read (and edit, maybe) `showergel.toml`, launch Liqudisoap, then run `showergel showergel.toml`. You'll likely want to enable the detailed log by setting `level=DEBUG` in the `logger_root` section of the toml file.

Test with `pytest`. See also *Package and publish Showergel*.

1.1.4 Install for front-end development

The front-end is written in JavaScript packed with Yarn, with VueJS's single-file components. We use the Bulma CSS Framework.

To modify the front-end, you must beforehand install Yarn and Vue_CLI, then run `yarn install` from the repository root. Start the live-building server with `yarn serve`. If you don't have time to install the whole back-end, you can call the demo app by creating a `front/.env` file that contains:

```
VUE_APP_BACKEND_URL=https://arcane-retreat-54560.herokuapp.com
```

Similarly, a fully-working HTML/JS/CSS build is included in this repository, so one doesn't have to install `yarn` and Vue while working on the back-end. Those files are generated by `yarn build`.

Note: Please do **not** commit modifications in the `/showergel/www/` folder. In order to avoid complex and useless conflicts, commits concerning this folder should only happen on the `main` branch.

1.1.5 Deploy to Heroku in demo mode

In demo mode, the application starts by putting fake data in the database. It's enabled by putting `demo = True` in the configuration file's `[listen]` section.

Source repository includes such a configuration, so you can create and push the app right after cloning:

```
heroku create --region eu
git push heroku main
heroku logs --tail
```

We might need to update `requirements.txt` from time to time:

```
poetry export --dev --without-hashes -f requirements.txt --output requirements.txt
```

`--dev` is here because `requirements.txt` is also used by ReadTheDocs to compile the present documentation, which requires a Sphinx extension.

1.2 Configuring Showergel

A configuration file is needed to start Showergel. It tells the program what to do, where data should be written, how to contact Liquidsoap, etc. Showergel's installer creates a minimal configuration, (including comments if you're hurried to tweak it) but you may need more or hesitate: this section will tell you all configuration details and their implications, if any.

1.2.1 What's a `.toml` configuration file

Showergel uses the TOML format. If you have never encountered it, let's start by describing *how* this configuration is written. It is a text file, encoded in UTF-8, that you can edit with the same program as your Liquidsoap script (Notepad, gedit, vim, Eclipse, etc.).

Configuration properties have a *key* (usually a word) and a *value* (a character string, a list, number, ...). Properties are grouped by sections, like `[listen]` or `[db.sqlalchemy]`.

Do not mix properties between sections: usually, the program will look for a property below one perticular section. *Comments* are notes that will be ignored by the program: they start with a `#`.

A TOML file looks like this:

```
[section]
property = "value"
somelist = ["foo", "bar", "baz"]
secret = 9876

[another_section]
# this line starts with a sharp : programs will ignore it
# you can use comments to note why a value is defined
# or to put aside old configuration values
debug = true  # comments can be at end of lines too
```

Showergel's installer creates a basic TOML file you should start with. This page's sections match sections in Showergel's config file.

1.2.2 `[db.sqlalchemy]`

This section should at least include a path to the instance's database file:

```
url = "sqlite:////home/me/path/to/showergel.db"
```

Yes, that's four `/`. Use only 3 if you prefer a relative path (relative to Showergel's working directory), for example `sqlite:///showergel.db`.

If you do not want to write a file, just use `sqlite:///:memory:`. However with that setting Showergel will always forget its data when restarting. This will disrupt metadata logging and users authentication, but can be enough for a short test. It is supported for unit testing and the online demo.

If you already have a DB server at hand, you can use it for Showergel too. You will have to install the DBAPI package youself, and refer to SQLAlchemy's documentation for the URL format - see pages about PostGreSQL or MySQL.

You may display DB requests in Showergel's log by setting the property `echo = true`. This can be useful when debugging.

1.2.3 [interface]

You may change the name displayed in the interface's left bar:

```
name = "Showergel Radio 98.7 FM"
```

1.2.4 [listen]

This defines Showergel's address for your browser and liquidsoap. For example:

```
address = "localhost"
port = 2345
```

Using these (default) values, Showergel's interface will be available at the URL http://localhost:2345/. If you are running multiple instances of Showergel on the same machine, be careful to set a different `port` for each one.

For some features (user authentication, metadata logging), Showergel's URL must be set in your Liquidsoap script too. See *Showergel integration in Liquidsoap scripts*.

Warning: The `address` property may be an IP address. If you change the address, ensure it is only accessible on a private network.

To have a more detailed server log you can add `debug = true`.

1.2.5 [liquidsoap]

This section defines how Showgel can contact Liquidsoap:

```
method = "telnet"
host = "localhost"
port = 1234
```

This should match Liquidsoap's telnet parameters - see *Showergel integration in Liquidsoap scripts*.

Other values can be set as `method`:

- `none` if you don't want to enable Showergel's "current track" display.
- `demo` will simulate a Liquidsoap connection. In that case `host` and `port` are ignored. This is used by Showergel's online demo.
- anything else, or if the parameter is missing, will simulate a Liquidsoap connection by generating different data each time it's called. This should only be used for Showergel's unit tests.

1.2.6 [metadata_log]

This section configures how Showergel stores tracks' metadata. It may contain `extra_fields`: a list of metadata fields that should be stored, when available.

```
[metadata_log]
extra_fields = [
    "genre",
    "language",
    "year",
    "track*",
]
```

A `*` in the field name represents any characters or nothing. In the example above, `track*` will match `track`, but also `track_number` or `tracktotal`. Field names are stored as they are, so Showergel will store `track_number` or `tracktotal`.

1.2.7 Logging configuration

This follows Python's configuration dictionary schema for logging.

1.3 Showergel integration in Liquidsoap scripts

Most Showergel features require your Liquidsoap script to implement a link with Showergel. This page presents these links, as seen from Liquidsoap. **All examples below assume your Liquidsoap script starts with a variable defining Showergel's URL, as:**

```
SHOWERGEL = "http://localhost:2345"
```

1.3.1 Showergel's quickstart.liq

We showcase the complete Showergel integration in a "quick-start script" you can download `here`. It is tested against Liquidsoap 1.4.4. This `.liq` file defines typical radio sources and is heavily commented: you can use it to start your first stream, or pick portions that would improve your existing script (it also contain a few Liquidsoap tricks!).

> **Warning:** Showergel and Liquidsoap are **not** secured against malicious access. In the worst case, this could result in innapropriate control of your radio's program. Please isolate the machine running Showergel on both physical and network levels.

Sections below discuss implementation details on integrating each Showergel feature.

1.3.2 Display/skip current track

You just need to enable Liquidsoap's telnet server. For example:

```
set("server.telnet",true)
set("server.telnet.bind_addr","127.0.0.1")
set("server.telnet.port",1234)
```

`127.0.0.1` is the IP address of `localhost`. The `port` value should match the one in Showergel's configuration's *[liquidsoap]* section. If you are running multiple instances of Liquidsoap on the same machine, be careful to set a different `port` for each one.

> **Warning:** Do not use a public IP address as `bind_addr`. This would open your Liquidsoap instance to the Internet, and someone might connect and mess up your programs.

1.3.3 Logging metadata

You need to define a function that will post metadata to Showergel.

For Liquidsoap version 1.x, you have to insert this function in your stream using the on_metadata or on_track operators, as follows:

```
def post_to_showergel(m)
    response = http.post("#{SHOWERGEL}/metadata_log",
        headers=[("Content-Type", "application/json")],
        data=json_of(m))
    log(label="posted_to_showergel", string_of(response))
end

radio = on_track(post_to_showergel, source)
```

Once this is defined, be careful to use `radio` as your outputs' source (instead of `source`). Otherwise `post_to_showergel` will never be called and nothing will be logged.

For Liquidsoap version 2.x, you just have to call source.on_track or source.on_metadata :

```
def post_to_showergel(m)
    response = http.post("#{SHOWERGEL}/metadata_log",
        headers=[("Content-Type", "application/json")],
        data=json_of(m))
    log(label="posted_to_showergel", string_of(response))
end

source.on_track(radio, post_to_showergel)
```

The line that starts with `log` is optional, it may help when debugging.

> **Warning:** Many Liquidsoap operators repeat previous track's metadata when switching from a source to a another. This concerns operators whose `replay_metadata` parameter defaults to `true`. This often yields duplicate entries in the log, although Showergel does its best to ignore duplicates.
>
> In other words, if you get duplicates in the metadata log, you might avoid them by adding `replay_metadata=false` to your `fallback`/`random`/`rotate`/`switch` operators. Especially if they're track-insensitive.

1.3.4 Authenticating users on harbor

Liquidsoap's input.harbor can require authentication by giving `user` and `password` parameters. But this implies

- storing the clear password in your `.liq` script
- sharing those credentials
- restarting the Liquidsoap stream when you want to update those credentials

This is not enough secured and unconvenient.

Instead, you can rely on Showergel to hold the list of users and their (encrypted) passwords. Then you will be able to add/edit crendentials from Showergel's web interface. This method requires creating an authentication function (in your `.liq`) passed to `intput.harbor`'s `auth` parameter (instead of `user` and `password`).

For Liquidsoap version 1.x, this function can be written as:

```
def auth_function(user, password) =
    let (status, _, _) = http.post("#{SHOWERGEL}/login",
        headers=[("Content-Type", "application/json")],
        data=json_of([("username", user), ("password", password)])
    )
    let (_, code, _) = status
    if code == 200 then
        log("Access granted to #{user}")
        true
    else
        log("Access denied to #{user}")
        false
    end
end

harbor = input.harbor(auth=auth_function, ...
```

For Liquidsoap version 2.x, this function can be written as:

```
def auth_function(user, password) =
    response = http.post("#{SHOWERGEL}/login",
        headers=[("Content-Type", "application/json")],
        data=json_of([("username", user), ("password", password)])
    )
    if response.status_code == 200 then
        log("Access granted to #{user}")
        true
    else
        log("Access denied to #{user}")
        false
    end
end

harbor = input.harbor(auth=auth_function, ...
```

1.4 RESTful interface

GET /metadata_log
Without parameters, `GET /metadata_log` returns the 10 most recent metadata items played. The following query parameters can be provided

Query Parameters

- **start** (*string*) – (ISO 8601 format) define an inclusive query interval
- **end** (*string*) – define an inclusive query interval
- **limit** (*int*) – restricts the number of results (and defaults to 10). It is ignored if `start` and `end` are provided.
- **chronological** (*bool*) – may be set to anything non-empty (use `1` or `true`), otherwise results are sorted recent first. Doesn't affect the interpretation of `start` and `end`.

Response JSON Object

- **metadata_log** – logged metadata matching the query parameters.

POST /metadata_log
Should be called by Liquidsoap to save tracks' metadata. See *Logging metadata*.

POST /login
Should be called by Liquidsoap to authenticate harbor users. It returns the matched user information as a JSONobject, or a 404 error. See *Authenticating users on harbor*.

GET /users

Response JSON Array of Objects

- **username** –
- **created_at** –
- **modified_at** –

PUT /users
User registration

Request JSON Object

- **username** –
- **password** –

Response JSON Object

- **username** –
- **created_at** –
- **modified_at** –

POST /users/(*username*)
Update user attributes

Request JSON Object

- **password** – optional

Response JSON Object

- **username** –

- **created_at** –
- **modified_at** –

DELETE /users/(*username*)
Delete someone's user account

Response JSON Object

- **deleted** – username

GET /live
The returned JSON object might contain many more fields, depending on what's in the current track's metadata. You can reasonably expect `title` and `artist`.

Response JSON Object

- **source** – name of the currently playing source
- **on_air** – current track start time
- **status** – status of the current source ("playing" or "connected to . . . ")
- **server_time** – server's datetime
- **remaining** – *maybe* remaining duration of current source, in seconds

GET /parameters
This returns values from the `[interface]` section of the configuration file.

Response JSON Object

- **name** – instance name (appears as interface's title)
- **version** – showergel's version

DELETE /live
Skips current track: this sends a skip command to the first Liquidsoap output.

OPTIONS /(:re:.*)
this only ensures we support `OPTIONS` requests (instead of returning 405). CORS headers will be returned iff `send_cors` is enabled.

1.5 Package and publish Showergel

1.5.1 Pre-publication check-list

- Run `poetry update`
- Run `pytest` - all tests should pass
- From the `front` folder, run `yarn upgrade`.

If this is done right after a feature freeze, the upcoming version should have a pre-release marker: `-alpha.0`. Run this pre-release for a few days in a realistic setup.

Build and check the documentation: `cd docs; make html`.

1.5.2 Update version markers

- `version` in `pyproject.toml`
- `version` in `front/package.json`
- `release` in `docs/conf.py`

1.5.3 Package

- From the `front` folder, run `yarn build`
- `poetry export --dev --without-hashes -f requirements.txt --output requirements.txt`
- `git commit,push,tag`
- `poetry build`
- `poetry publish`

Showergel is released under the GPL3 license.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

HTTP ROUTING TABLE

/(:re:.*)

/live

/login

/metadata_log

/parameters

/users