Liquidshop 3 - The Liquidsoap Workshop - 2023-05-30

Radio France	Building a production ready Liquidsoap stack for radio broadcasting

https://radiofrance.fr

https://github.com/radiofrance/rf-liquidsoap

Youenn Piolet Cloud engineer & DevOps, Team “Fondation” Github: uZer Keybase: https://keybase.io/ypiolet Mastodon: @schematicwizard@merveilles.town

About this presentation

Last week, we open sourced our Liquidsoap scripts:

- https://github.com/radiofrance/rf-liquidsoap

Previous presentations about our infrastructure:

- https://archive.fosdem.org/2020/schedule/event/om_audio_streami
- https://www.liquidsoap.info/liquidshop/1/
- https://www.liquidsoap.info/liquidshop/1/slides/piolet.pdf
- https://youtu.be/UnHfgDmi9_w

Main focus today:

1. Reminder of our **context**
2. Requirements for a **production ready**, Liquidsoap based streaming platform
3. Our Liquidsoap **demo stack** and **scripts**

1. Our context

1.1. Radio France

Information, Education, Entertainment, Culture *Public service with 903 journalists, 9 special reporters 1058 live events, 243 897 visitors in 2019 A national symphony orchestra*

- ~70 Million listeners per month for on-demand content
- ~70 Million monthly web visitors (doesn't include France Info)

Our broadcasting mediums: **FM, DAB+, Internet** (Live radio, podcasts, on demand content...)

1. Our context

1.2. Radio France - Direction du numérique

~200+ coworkers handling the presence of Radio France on the Internet

- Developers
- Infrastructure Engineers
- Designers
- Marketing Teams
- Innovation experts
- Data Engineers

We love open source!

1. Our context

1.3. Radiophonic activity

```
~~~graph-easy --as=boxart
[7 national channels]
[45 local channels]
[26 webradios]
[on demand channels]
~~~
```

∼80x 24/7 radio streams

https://www.acpm.fr/Les-chiffres/Frequentation-Radios/Classement-des-Radios-Digitales/Par-marque/Classement-France

1. Our context

1.4. Liquidsoap in Radio France cloud based environment

1.4.1. We use Liquidsoap like a real time pipeline for audio:

- raw inputs, coming from out studios
- buffers
- encoding: AAC & MP3, multiple qualities
- output: icecast & hls
- monitoring and operations over sources

```
~~~graph-easy --as=boxart
[inputs] - SRT -> [source selection] - encoding -> [mp3, aa
~~~
```

1. Our context

1.4. Liquidsoap in Radio France cloud based environment

1.4.2. We stream the audio we receive as is

- No playlist
- No audio transitions
- No advanced audio processing/filters or normalization for now

1. Our context

1.4. Liquidsoap in Radio France cloud based environment

1.4.3. We kept it simple

For one livestream -> (at least) one Liquidsoap process

Keeping the latency introduced by the pipeline as low as possible.

> *Most of the latency is introduced by the streaming protocols: Icecast, HLS...*

2. Requirements for a Liquidsoap based streaming platform

2.1. A standard production environment

What’s needed for a real time audio streaming production?

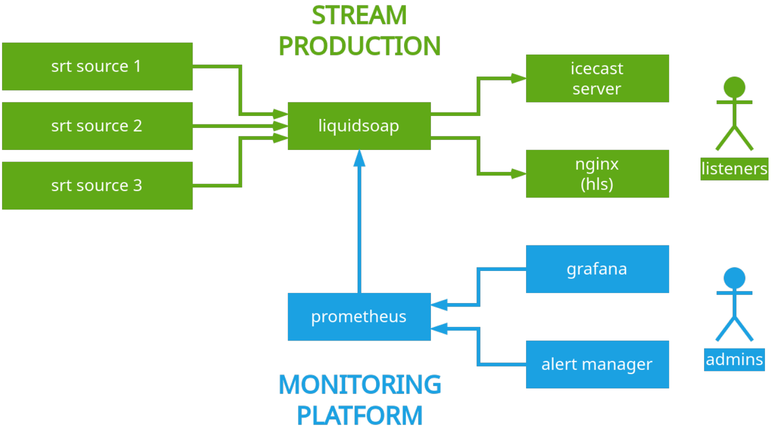

Figure 1: basic.png

2. Requirements for a Liquidsoap based streaming platform

2.1. A standard production environment

2.1.1. Achieving **input** resilience (1/3)

A self switching input fallback mechanism

```
radio_prod = fallback(
  id="fallback_prod",
  track_sensitive=false,
  [
    ...
  ]
)
```

2. Requirements for a Liquidsoap based streaming platform

2.1. A standard production environment

2.1.1. Achieving **input** resilience (2/3)

Protect stream continuity at all cost to avoid client disconnection, with a `safe_blank` source

```
radio_prod = fallback(
  id="fallback_prod",
  track_sensitive=false,
  [
    ...
    (safe_blank:source(audio=pcm,video=none,midi=none))
  ]
)
```

2. Requirements for a Liquidsoap based streaming platform

2.1. A standard production environment

2.1.1. Achieving **input** resilience (3/3)

Multiple paths for the audio coming from the studios

```
input_list = [
  {name="voieA_caller1", is_autofallback=true, port=10000},
  {name="voieA_caller2", is_autofallback=true, port=10001},
  {name="voieB_caller1", is_autofallback=true, port=10002},
  {name="voieB_caller2", is_autofallback=true, port=10003},
  {name="override_caller1", is_autofallback=false, port=100
  {name="override_caller2", is_autofallback=false, port=100
  {name="sat_sat1", is_autofallback=true, port=10006},
]
```

2. Requirements for a Liquidsoap based streaming platform

2.1. A standard production environment

2.1.2. Achieving **output** resilience

Multiple Liquidsoap instances per station you stream

Useful for Liquidsoap resilience
eg. 2 production servers, 2 preprod servers, one instance of Liquidsoap per channel, replicated on every server.
That way, you can perform maintainances or script modifications without service disruption

Multiple streaming servers and protocols

Useful for accessibility, loadbalancing, SLA...

- Icecast: Icecast master/relay architecture
- HLS: using CDN or cache mechanisms

2. Requirements for a Liquidsoap based streaming platform

2.1. A standard production environment

2.1.3. Observability on Liquidsoap

- Service availability (*Is Liquidsoap running?*)
- Input status (*Do we receive our audio sources?*)
- Output status (*Can we produce audio?*)
- Logs
- Network metrics (*Bandwidth usage, latency, jitter…*)
- System metrics (*CPU, memory…*)
- Pipeline metrics (*Buffers, latency, clocks…*)
- Audio metrics (*LUFS levels, is the audio blank?*)

2. Requirements for a Liquidsoap based streaming platform

2.1. A standard production environment

2.1.4. Tools for operators (1/6)

An API to ***get*** *information about Liquidsoap state:*

Liquidsoap's Harbor HTTP API

https://www.liquidsoap.info/doc-dev/harbor_http.html

```
harbor.http.register(port=harbor_http_port, method="GET", '
harbor.http.register(port=harbor_http_port, method="GET", '
```

2. Requirements for a Liquidsoap based streaming platform

2.1. A standard production environment

2.1.4. Tools for operators (2/6)

An API to ***get*** *information about Liquidsoap state:*

```
def write_http_response(code, data) =
  http.response(code=code, headers=[("Content-Type", "appl
end

def handler(h, method) =
  def response(~protocol, ~data, ~headers, uri) =
    let (code, data) = h(protocol, data, headers, uri)
    log.info(label="httplog", "#{code} #{method} #{uri}")
    log.debug(label="httplog", "#{code} #{method} #{uri} -
    write_http_response(code, data)
  end
  response
end
```

2. Requirements for a Liquidsoap based streaming platform

2.1. A standard production environment

2.1.4. Tools for operators (3/6)

*An API to **get** information about Liquidsoap state:*

Basic example: readiness

```
## GET /readiness
def get_readiness(_, _, _, _) =
    (200, '')
end
```

2. Requirements for a Liquidsoap based streaming platform

2.1. A standard production environment

2.1.4. Tools for operators (4/6)

An API to ***get*** *information about Liquidsoap state:*

Advanced example: GET current livesource

```
## GET /livesource
def get_livesource(_, _, _, _) =
  preferred = json.stringify(!preferred_live_source)
  inputs = json.stringify(list.map(fun (s) -> s.name, input
  real = json.stringify(!real_live_source)
  blank = json.stringify(!is_blank)
  (
    200,
    '{"preferred_output": #{preferred}, "inputs": #{inputs
  )
end
```

2. Requirements for a Liquidsoap based streaming platform

2.1. A standard production environment

2.1.4. Tools for operators (5/6)

An API to ***perform operations****, like source selection:*

```
harbor.http.register(port=harbor_http_port, method="GET", '
harbor.http.register(port=harbor_http_port, method="POST",
```

2. Requirements for a Liquidsoap based streaming platform

2.1. A standard production environment

2.1.4. Tools for operators (6/6)

*An API to **perform operations**, like source selection:*

```
## POST /livesource
def post_livesource(_, data, _, _) =
  if not list.exists(fun (s) -> s.name == data, input_sourc
    (400, '{"error_message": "input #{data} does not exist'
  else
    preferred_live_source := data
    # write livesourcestate on disk to persist across resta
    ignore(
      file.write(data=data, append=false, perms=0o644, live
    )
    (200, '{"preferred_output": #{json.stringify(data)}}')
  end
end
```

2. Requirements for a Liquidsoap based streaming platform

2.1. A standard production environment

2.1.5. Alerts

If something goes wrong, we need to be aware quickly.

2.1.6. Runbooks

If something goes wrong, we need to know what to do.

2. Requirements for a Liquidsoap based streaming platform

2.2. A “cloud native” environment

2. Requirements for a Liquidsoap based streaming platform

2.2. A “cloud native” environment

2.2.1. Works without human interactions

The stack should work without needing human interactions.

- Autofallback loop in Liquidsoap (*as shown previously*)
- Initial state should be the nominal running state
- Autorestart on failure

2. Requirements for a Liquidsoap based streaming platform

2.2. A “cloud native” environment

2.2.2. Using standard tools around Liquidsoap (1/2)

```
~~~graph-easy --as=boxart
[Metrics: Prometheus]
[Dashboards: Grafana]
[Alerts: Alertmanager]
[Logs: Stdout + Vector/Filebeat to centralize logs...]
~~~
```

2.2.2. Using standard tools around Liquidsoap (2/2)

Liquidsoap includes a Mirage Prometheus server.

```
settings.prometheus.server.set(true)
settings.prometheus.server.port.set(6001)

# Metric definition
audit_lufs_metric_create = prometheus.gauge(
  labels=["radio", "type", "name"],
  help="Audio LUFS Analysis",
  "liquidsoap_output_lufs_5s"
)

# Metric instance
set_metric_audio_lufs =
  audit_lufs_metric_create(label_values=[radio_name,"output

# Source processing
```

2.2.3. Industrialization, templating and reproductibility (1/2)

Splitting Liquidsoap configuration in parts improves readability:

```
scripts/
  00-live.liq        # <-- this is the entrypoint
  10-settings.liq
  20-prometheus.liq
  30-formats.liq
  40-icecast.liq
  50-hls.liq
  60-core.liq
  90-http.liq
```

In `00-live.liq`:

```
#!/usr/bin/liquidsoap

%include "10-settings.liq"
%include "20-prometheus.liq"
%include "30-formats.liq"
```

2. Requirements for a Liquidsoap based streaming platform

2.2. A “cloud native” environment

2.2.3. Industrialization, templating and reusability (2/2)

You can make your multipart main script reusable and personalized at runtime with variables for each livestream you want to build (each Liquidsoap service you need to run):

```
    scripts/
    config/
       fip.liq
       franceculture.liq
       franceinter.liq
liquidsoap -c /config/fip.liq /scripts/00-live.liq
```

This is a good way to achieve something close to many industrialization tools like `ansible`, `chef`, `puppet`: a template folder + inventory splitting, improving readability, scalability and reusability.

If you have too many variables, you could even use an external templating tool like `jinja2`, `jsonnet` to generate your inventory.

2. Requirements for a Liquidsoap based streaming platform

2.2. A “cloud native” environment

2.2.4. Version control, release management, lifecycle, integration

It’s always a good practice to: - use versionning (like git) - describe a specific version of a component with name, tag or release version

```
~~~graph-easy --as=boxart
[version 1.0.0: Major feature... ]
[version 1.0.1: Bugfix... ]
[version 1.1.0: Minor feature... ]
~~~
```

Liquidsoap scripts/templates can be seen like a piece of software, with it’s own lifecycle and requirements.

Taking profit from common industrialization tools to implement continuous integration, continuous deployment, gitops, etc.

Using variables and/or a separated inventory makes it easy!

2. Requirements for a Liquidsoap based streaming platform

2.2. A “cloud native” environment

2.2.5. Containers?

Processing an audio livestream with Liquidsoap is almost stateful. We can find some ideas for mitigation with multiple parallel liquidsoap process but…

- Process interruption == output discontinuity
- Sample level synchronization?
- Discontinuity in encoder level containers / output codec containers?

  Not the best for `Kubernetes` or other containerized failable platforms, but still doable!

It is still interesting to use `containers`:

- Manipulation of the Liquidsoap scripts as an artifact or a volume
- Variable values can be set in the environment or in a volume
- Easy versionning of Liquidsoap
- Easy to manipulate system dependencies (`ffmpeg` and other libraries…)

2. Requirements for a Liquidsoap based streaming platform

2.2. A “cloud native” environment

2.2.6. Basic architecture

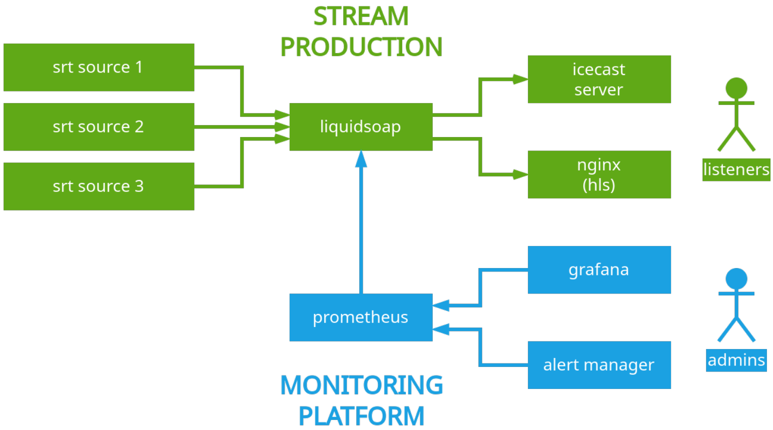

Figure 2: basic.png

3. Our Liquidsoap demo stack and scripts

3.1. Filestructure (1/3)

The . folder:

```
example                # Some configuration examples you ca
scripts                # Our liquidsoap templates
 docker-compose.yml     # Run the demo stack
 Makefile               # Tools to operate the stack
 README.md              # Extensive documentation
```

3. Our Liquidsoap demo stack and scripts

3.1. Filestructure (2/3)

The ./example folder:

```
example
  alertmanager                    # Alertmanager configura
    config.yml

  grafana/provisioning            # Grafana configuration
    dashboards                    # Simple dashboards for
      dashboard.yml
      docker_containers.json
      docker_host.json
      levels.json
      liquidsoap.json
      services.json
    datasources
      datasources.yml              # Tell Grafana to speak

  liquidsoap                      # Liquidsoap inventory e
```

3. Our Liquidsoap demo stack and scripts

3.1. Filestructure (3/3)

The ./scripts folder: Liquidsoap scripts we use in production today

```
scripts
  formats                             # Encoder profiles
    hls-aac.liq
    hls-libfdk-aac.liq
    icecast-aac.liq
    icecast-libfdk-aac.liq
    icecast-mp3.liq
  00-live.liq                         # Entrypoint, the main
  10-settings.liq                     # Default values
  20-prometheus.liq                   # Create metrics
  30-formats.liq                      # Include formats profi
  40-icecast.liq                      # Output an Icecast str
  50-hls.liq                          # Output an HLS stream
  60-core.liq                         # Source instantiation
  90-http.liq                         # The HTTP API
```

3. Our Liquidsoap demo stack and scripts

3.2. The docker-compose (1/5)

Tests

```
services:
  # Test validity of liquidsoap configuration
  liquidsoap-test:
    image: savonet/liquidsoap:v2.1.4
```

3. Our Liquidsoap demo stack and scripts

3.2. The docker-compose (2/5)

Liquidsoap + sources

```
services:

  # Run liquidsoap and create "myradio" stream
  liquidsoap-myradio:
    image: savonet/liquidsoap:v2.1.4

  # Feed liquidsoap with an example SRT source (https://mod
  source-voieA-caller1:
    image: savonet/liquidsoap:v2.1.4

  # Feed liquidsoap with an example SRT source (https://p-n
  source-voieB-caller1:
    image: savonet/liquidsoap:v2.1.4

  # Feed liquidsoap with an example SRT source (https://dat
  source-override-caller1:
```

3. Our Liquidsoap demo stack and scripts

3.2. The docker-compose (3/5)

Streaming services

```
services:

  # Streaming services: icecast
  icecast:
    image: moul/icecast

  # Streaming services: hls (nginx)
  hls:
    image: nginx:alpine
```

3. Our Liquidsoap demo stack and scripts

3.2. The docker-compose (4/5)

Monitoring services

```
# Monitoring
grafana:
  image: grafana/grafana:latest
prometheus:
  image: prom/prometheus:latest

# Alerting
alertmanager:
  image: prom/alertmanager:latest

# Container metrics
cadvisor:
  image: gcr.io/cadvisor/cadvisor:latest
redis:
  image: redis:latest
```

3. Our Liquidsoap demo stack and scripts

3.2. The docker-compose (5/5)

Docker volumes!

```
volumes:
  data_grafana: {}
  data_hls: {}
  data_liquidsoap: {}
  data_prometheus: {}
```

3. Our Liquidsoap demo stack and scripts

3.3. The Makefile

```
help              Display this message
artifact          Build binary artifact
test              Run test on the liquidsoap configuration
reload            Update containers if needed and restart l
start             Start everything
stop              Stop all containers
status            Show status of docker containers
clean             Stop and remove all containers, networks
logs              Show logs
info              Show useful default URLs and service port
```

3.4. Demo time!

wow

2147483647. Future, conclusions and Q&A

Room for some improvements:

- Variable naming
- Liquidsoap script organization
- More templating, maybe for a more *common* usage
- Extensive documentation
- Known issues (see `CHANGELOG.md`)
- New Grafana dashboards
- Inform Tony I'm using https://datafruits.fm to feed my examples *before* the presentation. Sorry Tony...!

Still missing:

- `.github-ci.yml` and tests on Github (we were using Gitlab for now)
- Finish `docker-compose.yml` for alerts & cadvisor
- `CHANGELOG.md` automation
- Github Stars