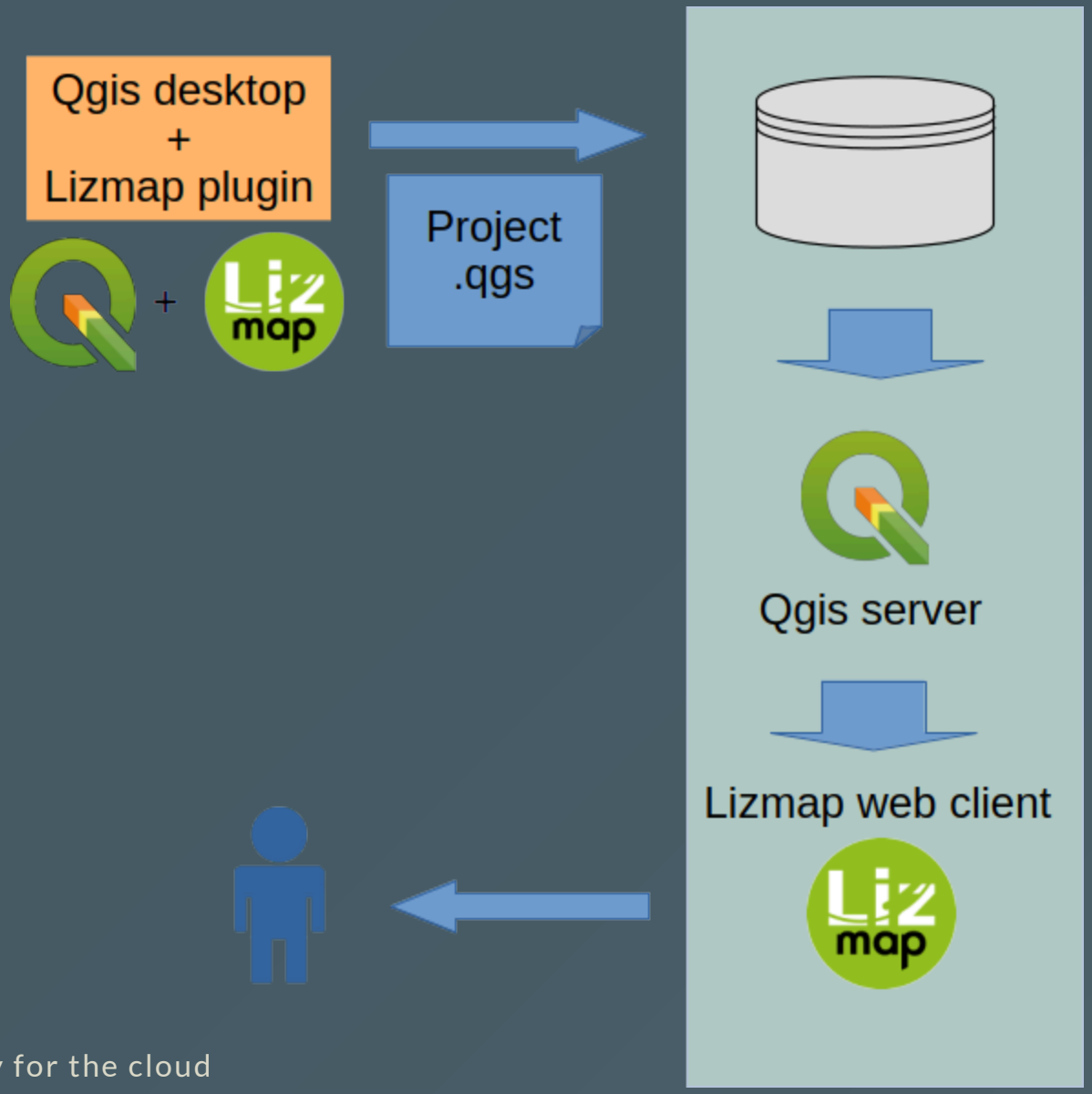
QGIS server for the cloud

Return of experience from GIS hosting services with Lizmap and QGIS server

GIS hosting services with Lizmap and QGIS server



GIS hosting services with Lizmap and QGIS server

Numbers:

- 580 QGIS server instances
- 1500 projects
- 47 physical servers
- 407 PosgreSQL databases
- 377 lizmap instances

Problems to solve:

- Scalability
- Distributed architecture
- Monitoring
- Zeroconf (ideally !)
- Security

Dealing with issues

- Projects with many layers: up to 200 layers per project
- Loading times of several minutes
- Memory issues
- High latency requests (mainly remote services)
- Stuck server instances

When things go wrong

Needs for a better control: Py-QGIS-Server

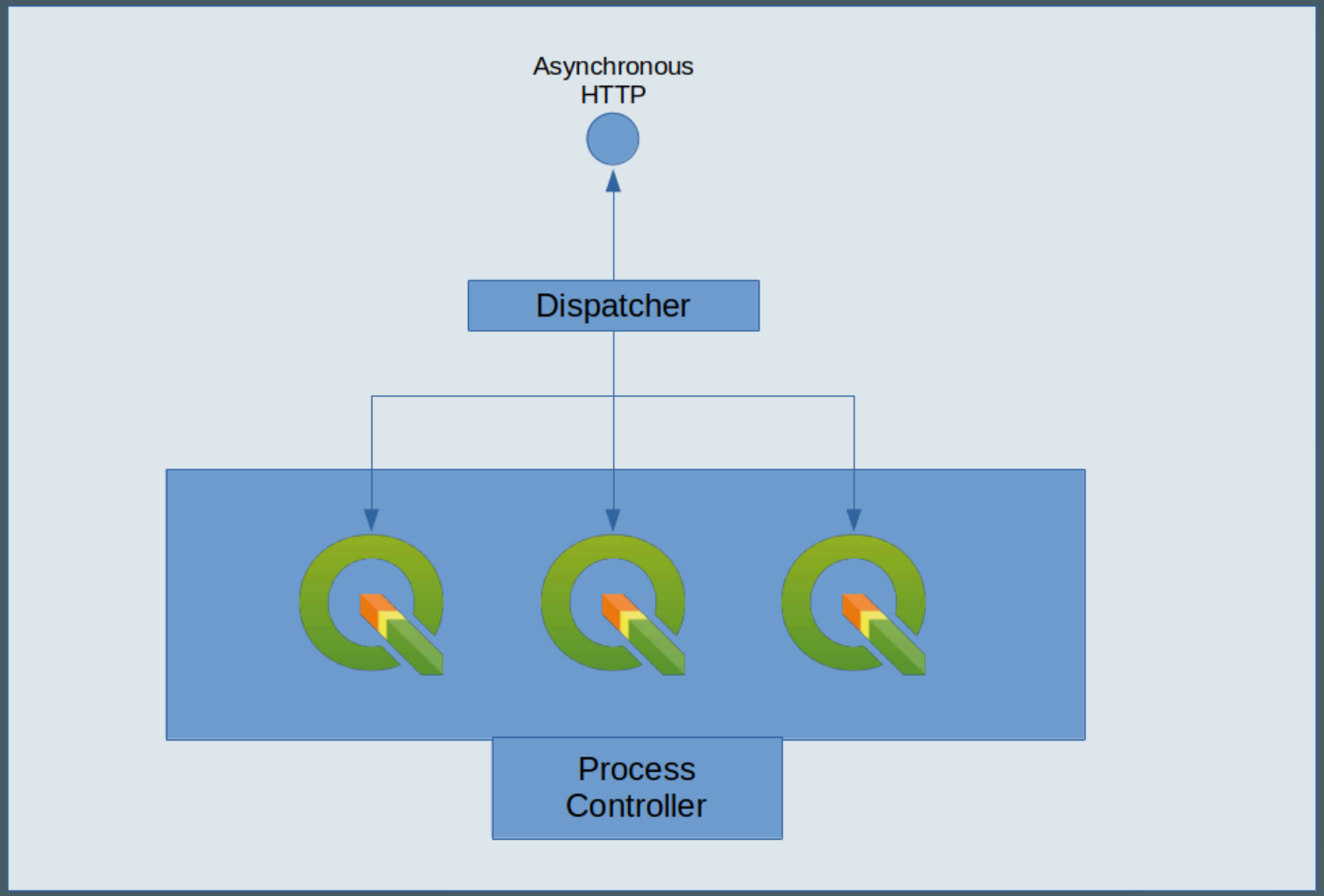
What

- QGIS server embedded in python code
- Manage QGIS server instances in subprocesses

Why

- Deal with scalability
- Deal with cache (#1)
- Deal with processes healthcheck
- Deal with complex Access Control List

Py-QGIS-Server unit



Advantages:

- Compact (one service)
- Easy to deploy
- Simple routing with reverse proxy (Nginx, ,,,)

Downsides

- Need Improve horizontal scalability supports on cloud native environments (i.e minimal reconfiguration)
- Problems with caching *heavy* projects (#1)
- Add more complex routing rules without resorting to complex 3-party tool configuration.

More about caching projects

No sharing of projects between service instance:

- Do the maths !
- No way to use memcache-like tools
- Simple LRU caching does not work well
  - Latencies when loading big projects
  - LRU scheme not suited for current usage.

Better but still some room for improvement

Performances considerations

Horizontal scalability governance

- What is the expected request rate ?
- How request distribute on projects ?
- How many different projects I have to handle ?

Impact on request processing

- How many layers in my project ?
- Data volumetry ?
- Accesses to database backends/external services ?

There is no single strategy to rule all your projects

Guideline hints

From a project management perspective

- Consider Project as an application
- Corollary: QGIS server is an application server
- Control what is published (from a customer perspective)
- Keep some level of flexibility (dynamic caching)

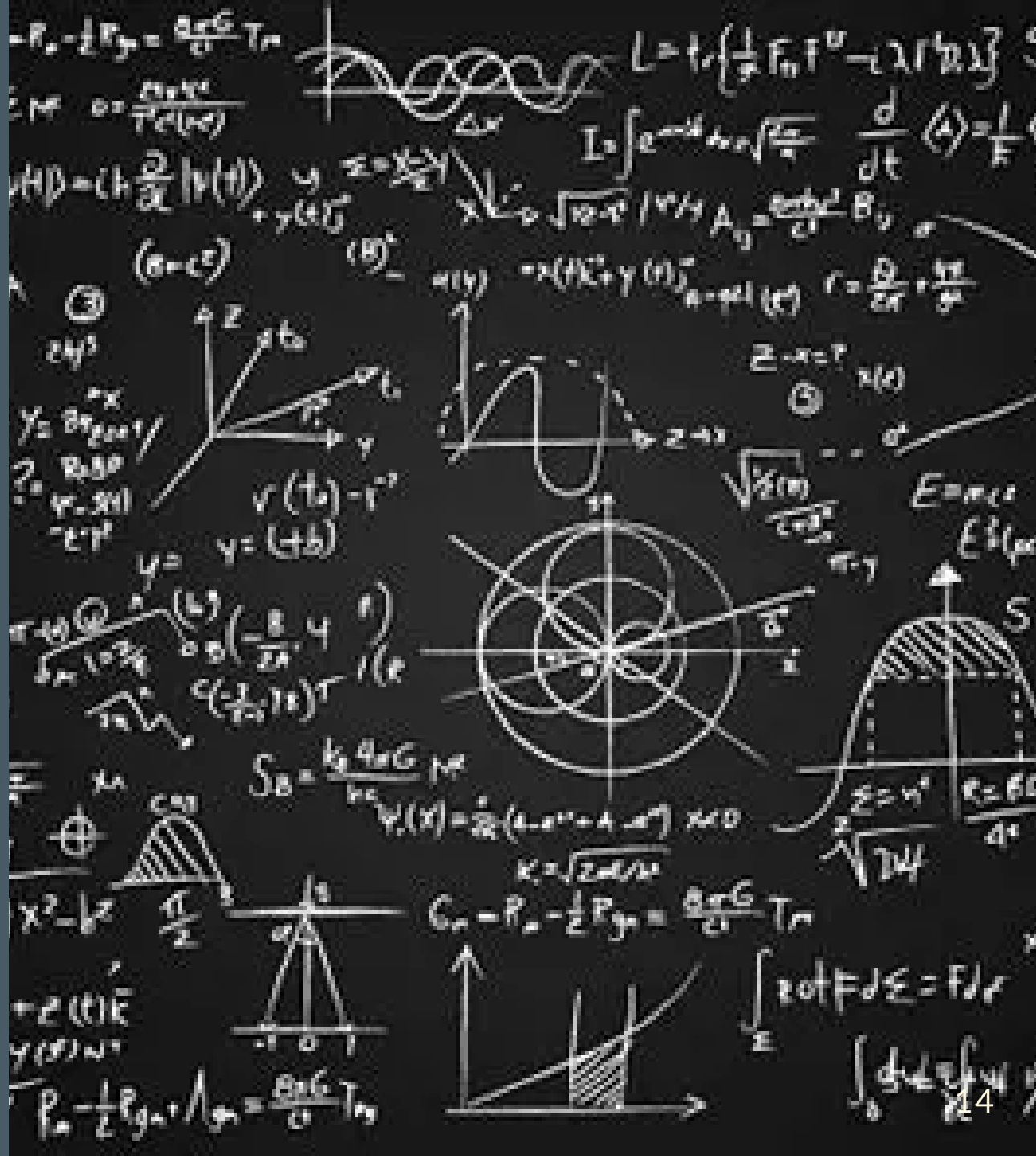
From a deployement perspective

- Projects partionning (Routing)
- Handle access controls at routing level
- Easy to scale even with simple Docker Compose stack.

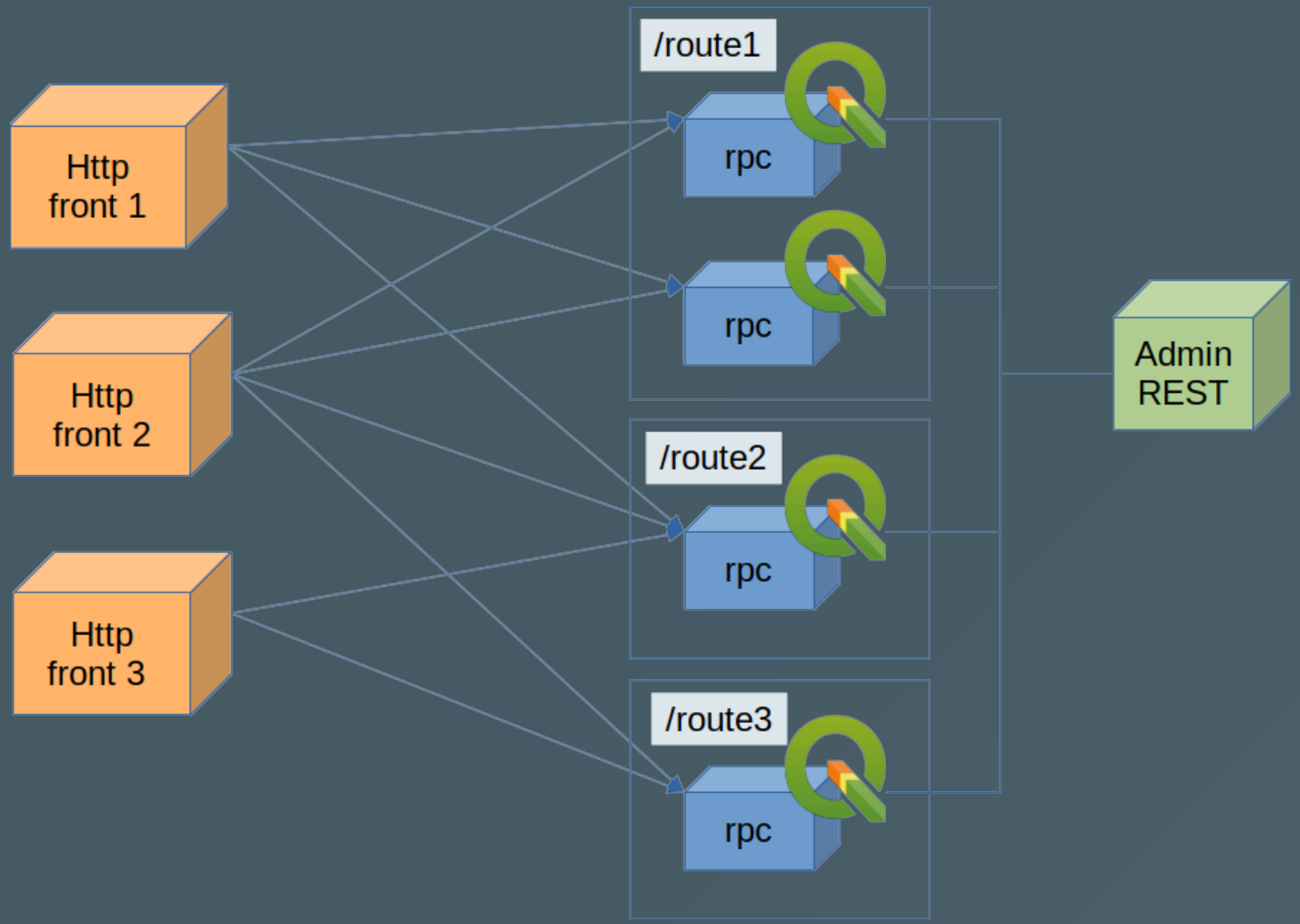
Roadmap to Py-QGIS-Server v2

Microservices with gRPC protocol:

- Built-in Load balancing
- Built-in healtcheck support
- Dedicated administration service
- Bi-directional streaming support
- Indepedent from HTTP front-end



Py-QGIS-server 2 overview



Roadmap to Py-QGIS-Server v2

Revisit QGIS server project caching

- Full control via REST API

Adjust to your infrastructure

- Multiple QGIS server processes by service unit
- Multiple service units, keep QGIS processes low by unit

Dynamic backend configuration

- Remove/Add backend on the fly

Roadmap to Py-QGIS-Server v2

Cloud friendly

Easy to configure

- Environment
- Config file
- Secrets

Easy to scale

- Zeroconf scalalability:
  - `docker service scale qgis-rpc=<n>`
  - `docker compose scale qgis-rpc=<n>`

Basic Docker compose

```
services:
  qgis-rpc:
    image: 3liz/qgis-services:qgis-ltr
    environment:
      CONF_DISPLAY_XVFB: ON
      CONF_WORKER__NAME: worker
      CONF_WORKER__PROJECTS__SEARCH_PATHS: >-
        {
          "/": "/qgis-projects/france_parts"
        }
    volumes:
    - { type: bind, source: "../../tests/data", target: /qgis-projects }
    command: ["qgis-server-rpc", "serve"]
    scale: 2
  web:
    image: 3liz/qgis-services:qgis-ltr
    environment:
      CONF_BACKENDS__BASIC__TITLE: "Basic backends"
      CONF_BACKENDS__BASIC__ADDRESS: "qgis-rpc"
      CONF_BACKENDS__BASIC__ROUTE: "/basic"
    ports:
    - 80:80
    command: ["qgis-server-http", "serve"]
```

Current status

Testing

- Feature ready
- Testing on our Lizmap cloud infrastructure
- Code available on github: https://github.com/3liz/py-qgis-server2

Conclusion

- Revisiting QGIS server as an application server
- Cloud friendly integration (QGIS servers as micro-services)
- Address the problem of serving multiple use cases
  with many different kind of QGIS projects.

Thank you

Questions ?