Takeaways of building a research-oriented system

Collection of little things learned the hard way

Context

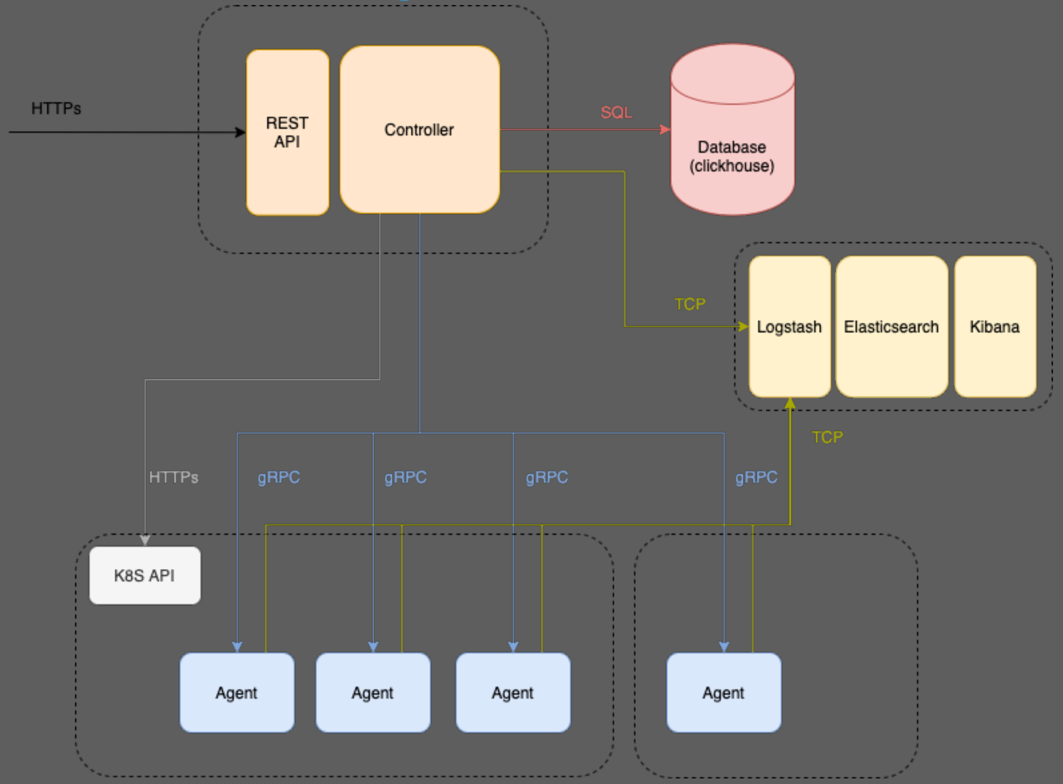
- Hi, I’m a research engineer ! (now in a PhD)
- I’m buidling a system to discover the topology of Internet (not published yet)
  - ✓ Modern
  - ✓ Scalable
  - ✓ Fault-tolerent

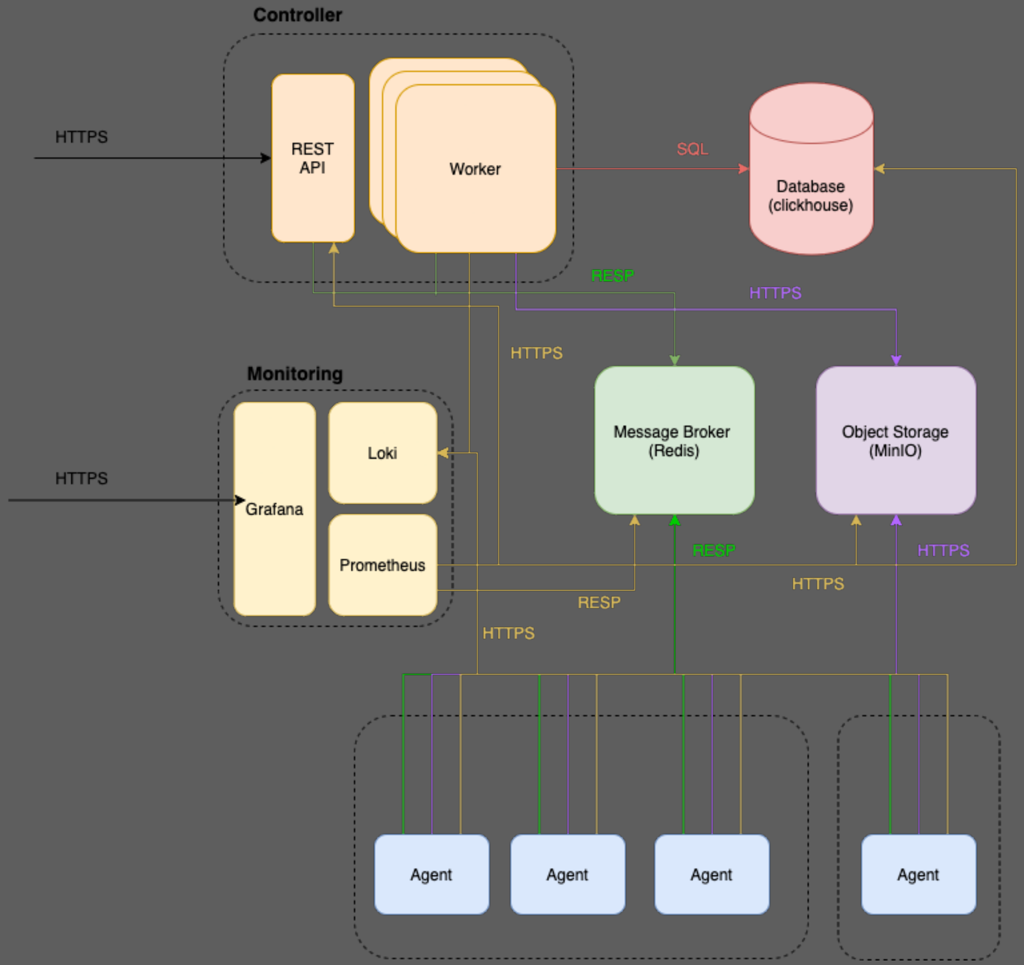
System design and building

Research ... About design

- Proof of Concept definition
- Constraints extraction
- High-level design
- Technologies selection
- Test cases

Research … About design





When technologies influence design

- Docker / Docker Compose
  - ✓ Take you away from monolithic designs
  - ✓ Allow you to incorporate open-source third-parties instead of homecook bricks
  - ✓ Make you think about flows and security

APIs

- How the user will interact with your system / program ?
  - Command-line ? (Typer, Click)
  - Web framework ? (FastAPI, Connexion) ?
  - Website ?

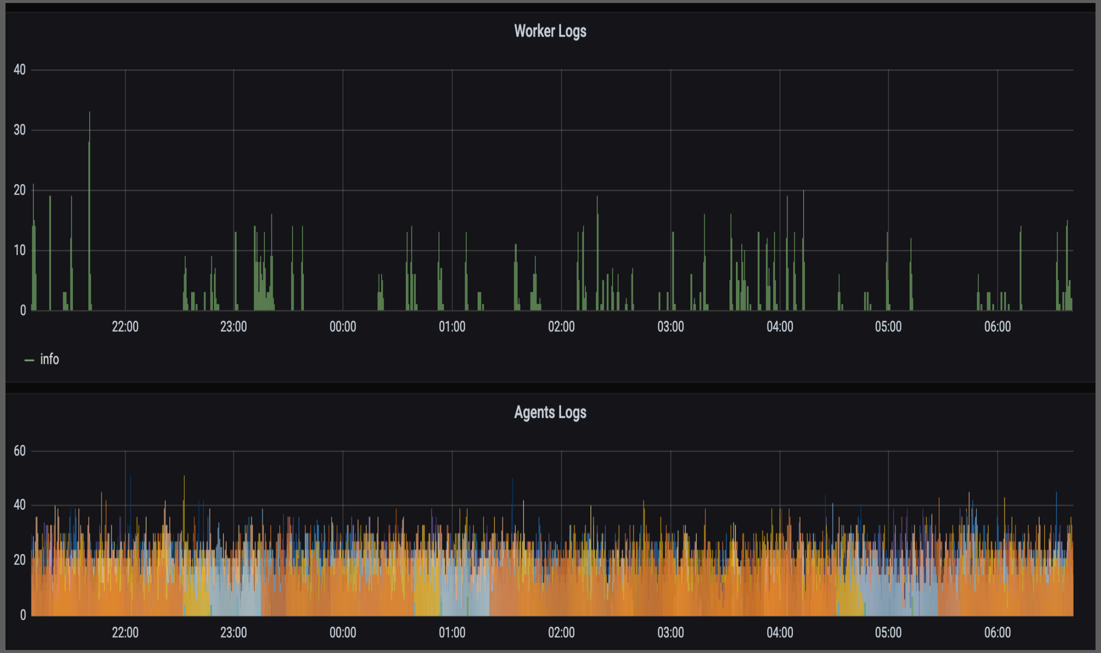
Monitoring

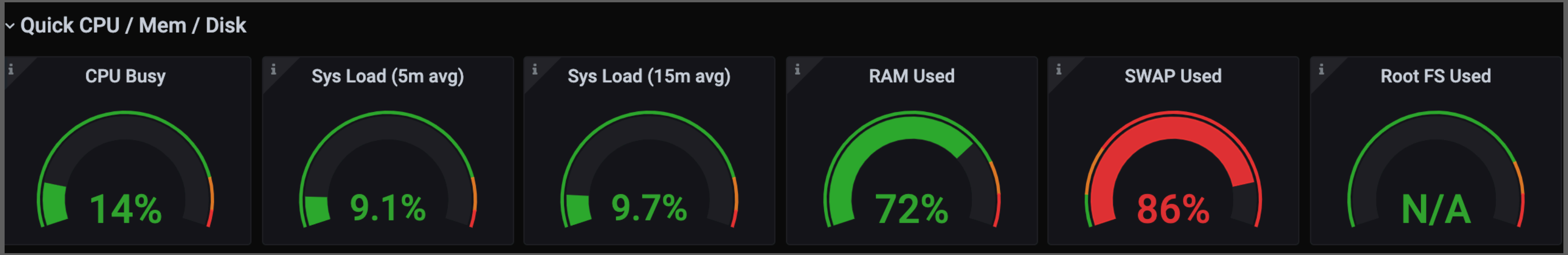
- The forgotten yet essential brick
- Metrics vs Logs
- Monitor your system

(i.e., custom code and thirtd-parties), but also underlay

- Alerts to a Slack channel or email

Stacks often used: ELK, Prometheus/Grafana





Security

- Reverse proxy is useful (TLS termination, certificate management, …)
- Let's encrypt
- Of course, no plain-text password in database! (e.g., Bcrypt)
- APIs Authentication : BasicAuth, JWT. (/!\ Timing attacks, DDOS, …)

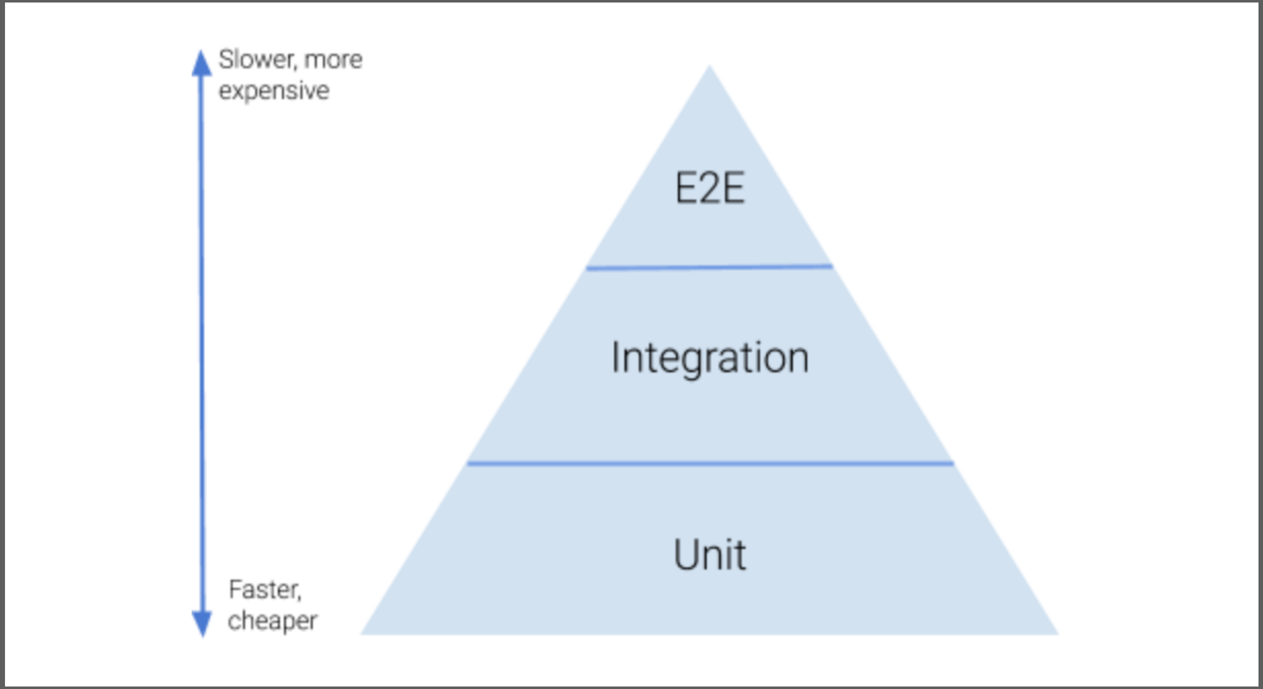
(Python) code within a system

Good practices

- Even when working alone, think about your collegues (or you of the future)
  - ✓ Python packaging and dependency management (poetry, pipenv)
  - ✓ Lint the code (flake8, pylint)
  - ✓ Format automatically (black)
  - ✓ Test ! The sooner the better (pytest)
  - ✓ Security (Bandit)
  - ✓ Documentation

Tests

- Pyramid of tests : guide != law
- Test what it makes sense
- Don’t chase a test coverage



Documentation at multiple levels

- Code Comments (but wisely)
- Tests are a form of documentation
- Custom library documentation (ReadTheDoc)
- API documentation (Swagger) for technical users
- Classical end-user high-level documentation (website ?)

Version control

- Github private repository are free
- Track features and issues (Github issues/project, Trello)
- Use of git tags and sementic versionning
- Branching model

CI/CD

- CI: Verification / Testing at commit and pull requests
- CD: Automatic Docker image push

- Different envionments : Dev/Test/Prod
- More advanced use cases : Blue/Green, Canary

Often used: Github Actions, Gitlab CI/CD, TravisCI, Jenkins

Fault-tolerence

- Python package: Tenacity (https://github.com/jd/tenacity)
- Every interface with other components are in the same library

Code use-case in depth : Async

- Begin to be wildly used in Python ecosytem
- Very convenient for scalability of distributed systems
- Fully integrated in FastAPI, Typer, …

Questions ? ☺