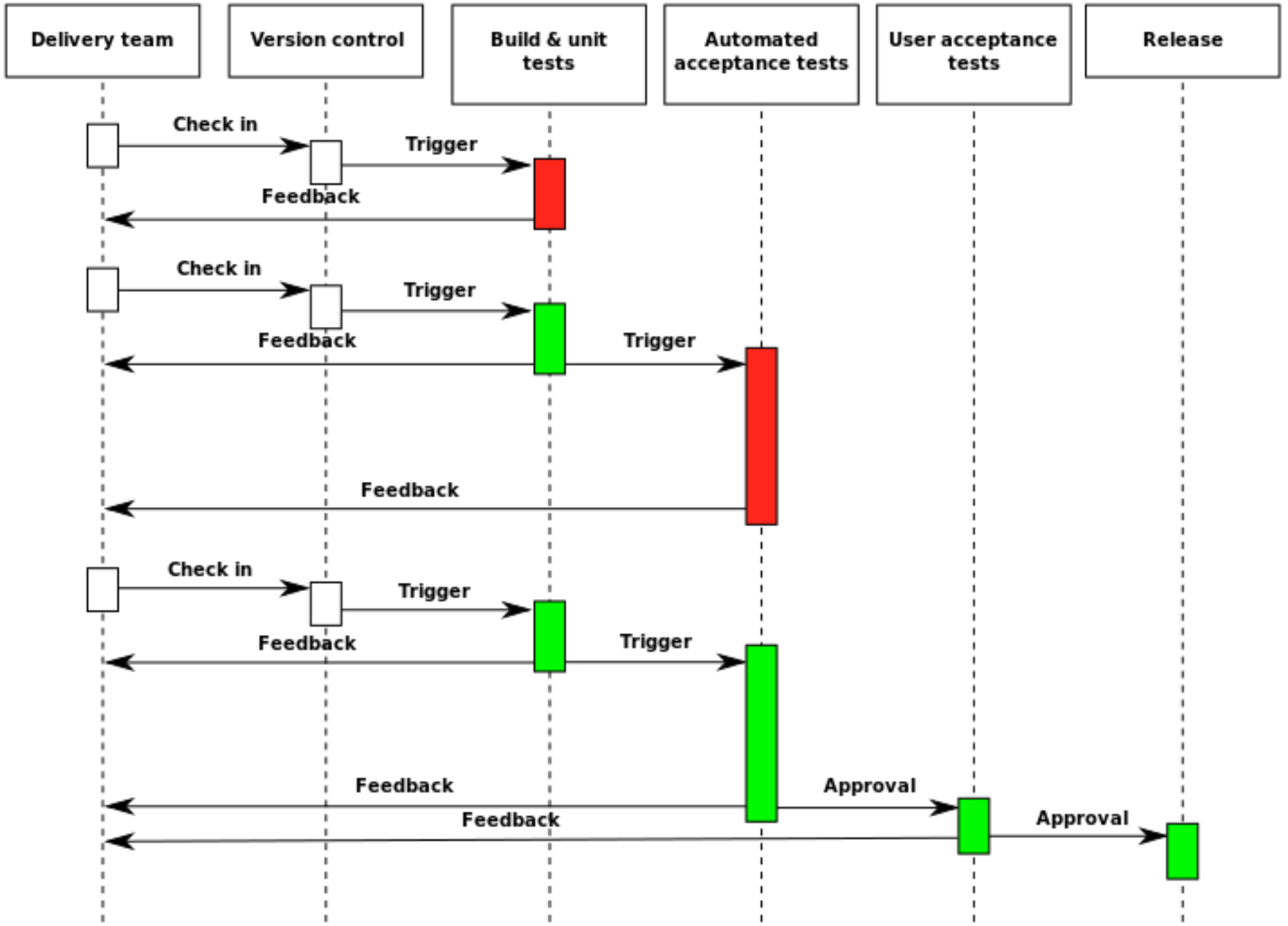
COMP.SE.140

Automation

Kari Systä

About automation

Deployment pipeline (a possible example)



Tampereen yliopisto
Tampere University

Infrastructure as code

From: https://docs.microsoft.com/en-us/azure/devops/learn/what-is-infrastructure-as-code

Infrastructure as Code (IaC) is

- the management of infrastructure (networks, virtual machines, load balancers, and connection topology) in a descriptive model,
- using the same versioning as DevOps team uses for source code.
- Like the principle that the same source code generates the same binary, an IaC model generates the same environment every time it is applied.
- IaC is a key DevOps practice and is used in conjunction with continuous delivery.

Tampereen yliopisto
Tampere University

Benefits of automation

- Prevent errors
- Is repeatable
- No need to write documentation
- Enables collaboration because everything is explicit in scripts
- Expertise encapsulated in scripts
- Manual work is boring
- Fast and relentless feedback
- Risk management: Automated checking and auditing

Automation includes

- Building
  -> no command-line tools needed
- Testing
  -> run frequently
- Other quality analysis
  -> less manual inspection needed;
- Deployment
  -> VMs and containers created automatically
  -> configuration management
- Database tools
  -> initialization
  -> management
- Scaling

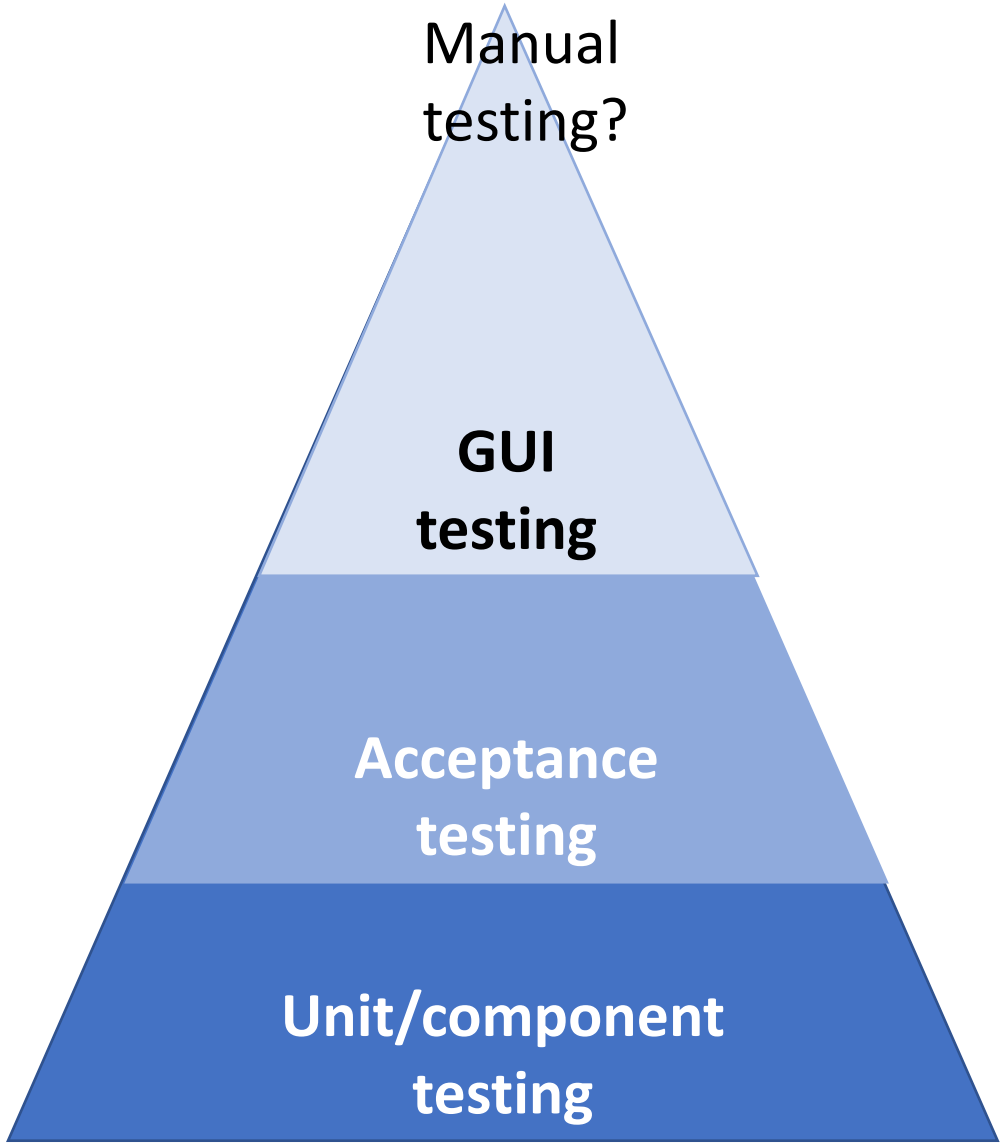
Tampereen yliopisto
Tampere University

Automated tests

- A common practice in CI and CD
- Does not invent the test (usually);
  - test are designed and implemented manually but
  - executed automatically
- Tests need to maintained
- Software needs to be testable
- Not a silver bullet for testing, but necessary helper in CI/CD

Testability

- Testbed can command the software
- Tests can investigate state and results
- Proper architecture and coding style helps
  e.g. Standard getters and setters
- Well-defined APIs


Manual testing?
GUI testing
Acceptance testing
Unit/component testing

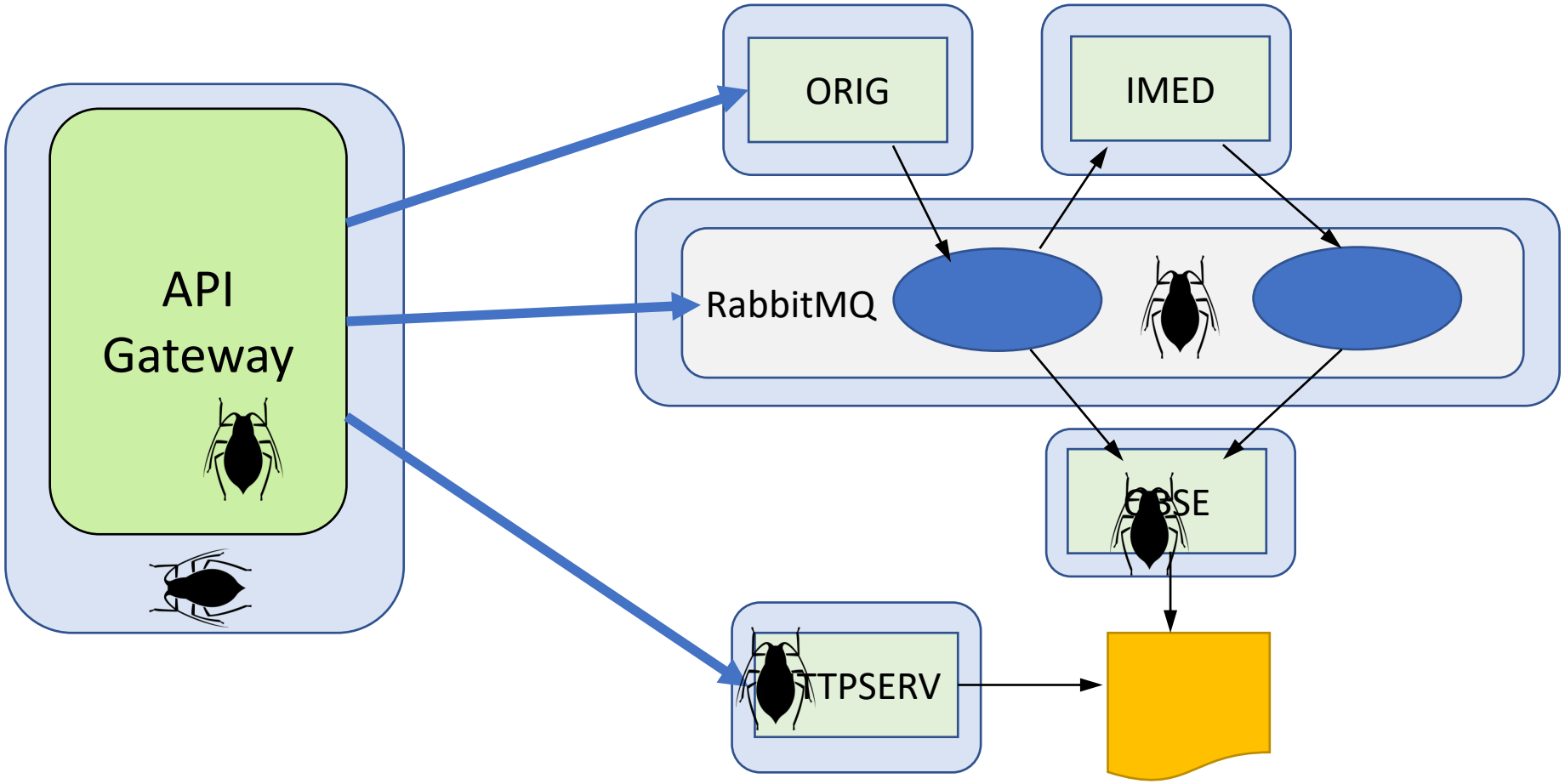
Tampereen yliopisto
Tampere University

Automated acceptance tests

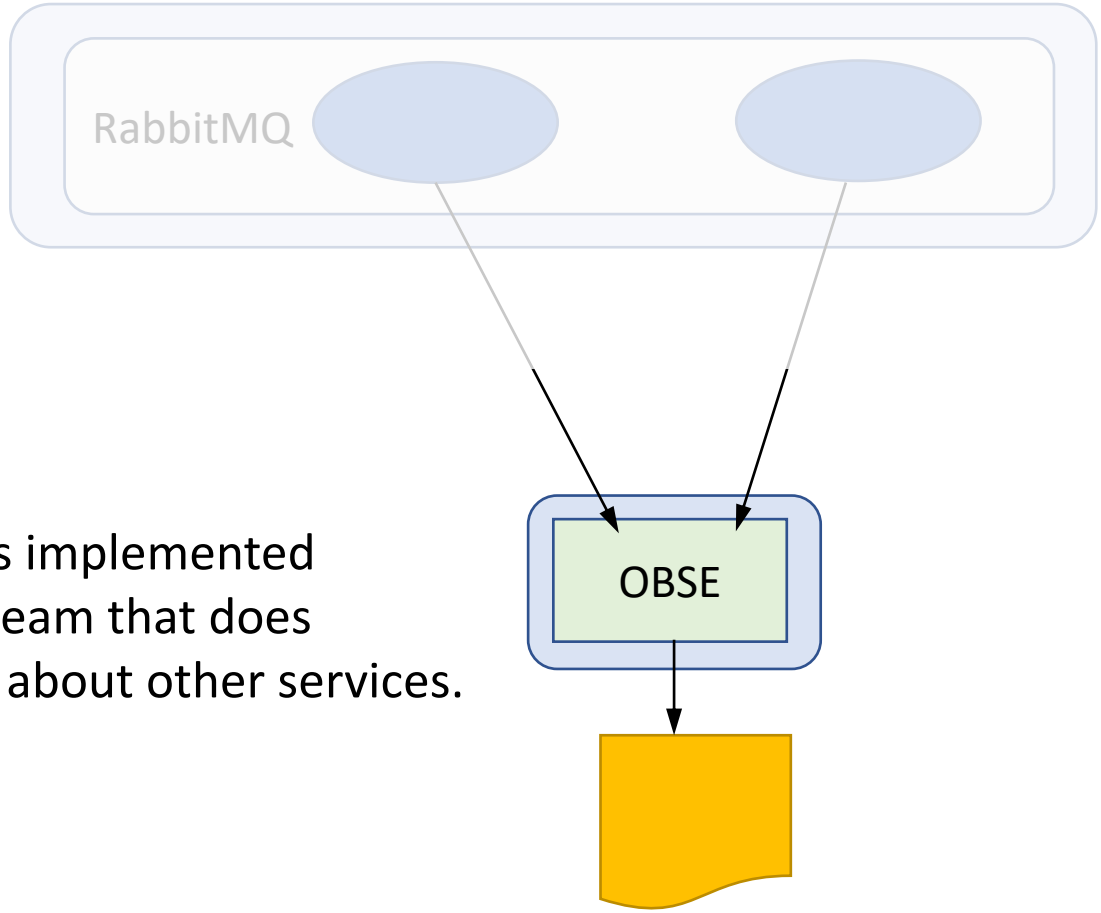
- Acceptance tests do not test everything but is an essential “gate” if deployment is automated.
- Some best practices (according to Humbley and Farley):
  - Test in realistic environment(s)
  - Acceptence tests are owned by the whole team (no separate team for it)
  - Developers should be able to run the tests in their own dev environment)
  - Tie to business value – not to technical solution of the system
- Nonfunctional testing
  - Capacity, scalability
  - Code quality analysis

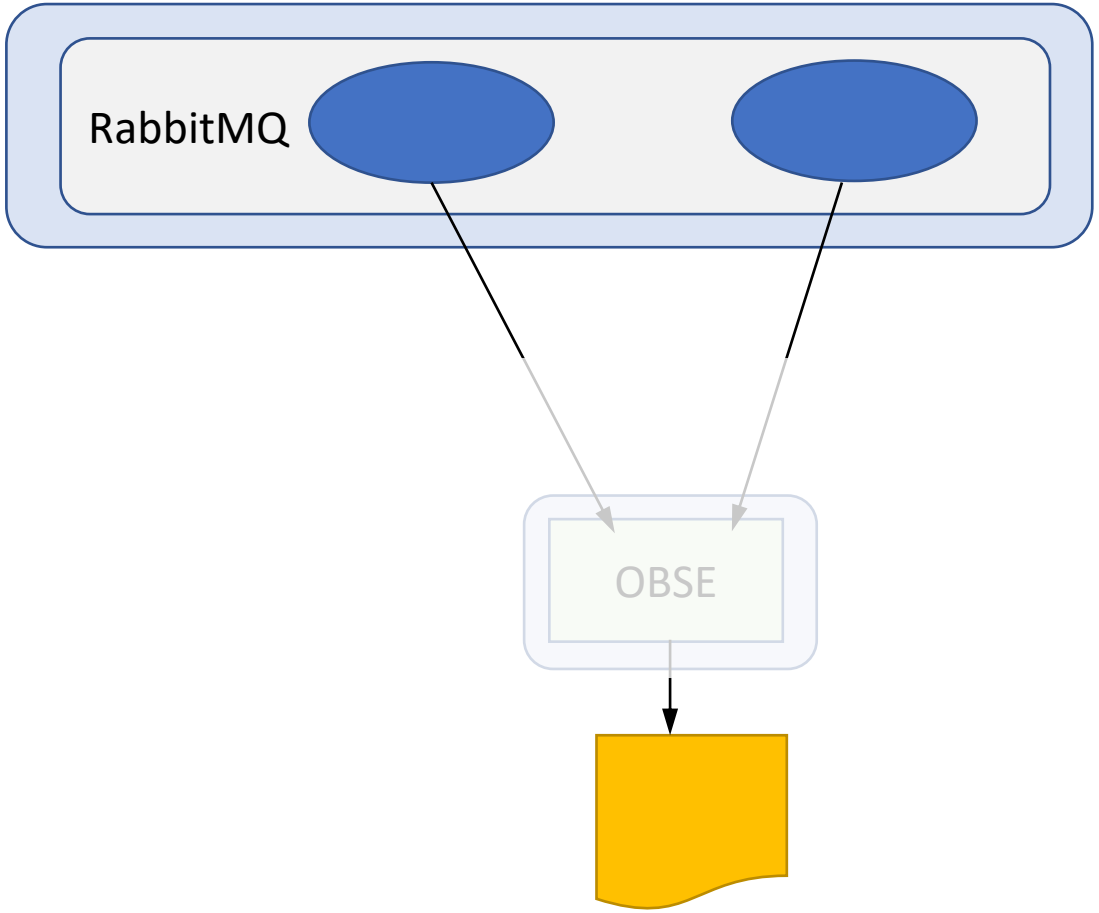
Tampereen yliopisto
Tampere University

Testing cloud-native is difficult
And debugging even more difficult



What if OBSE is implemented
by a separate team that does
not now much about other services.



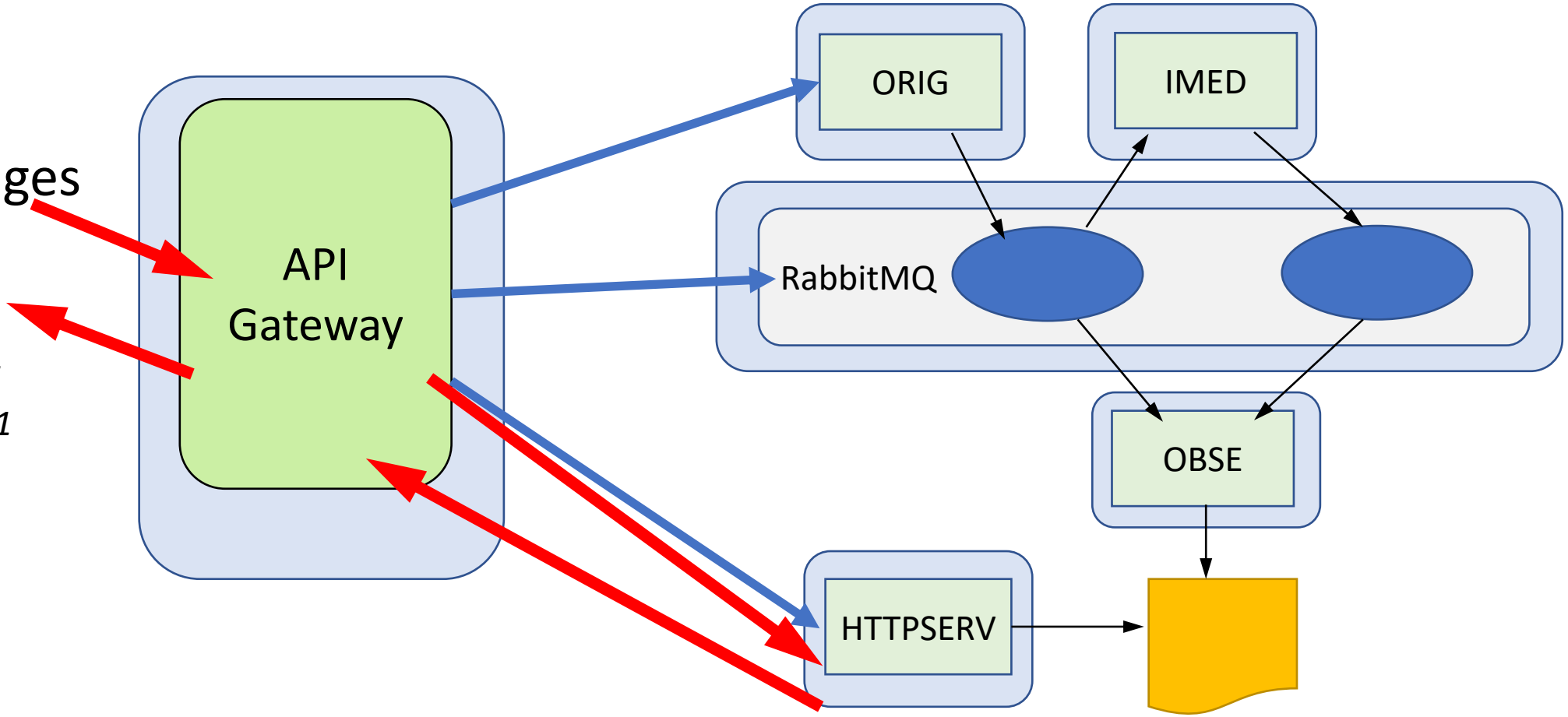

RabbitMQ
OBSE

Tampereen yliopisto
Tampere University

GET /messages

2020-10-01T06:35:01.373Z
Topic my.o: MSG_1

Correct?

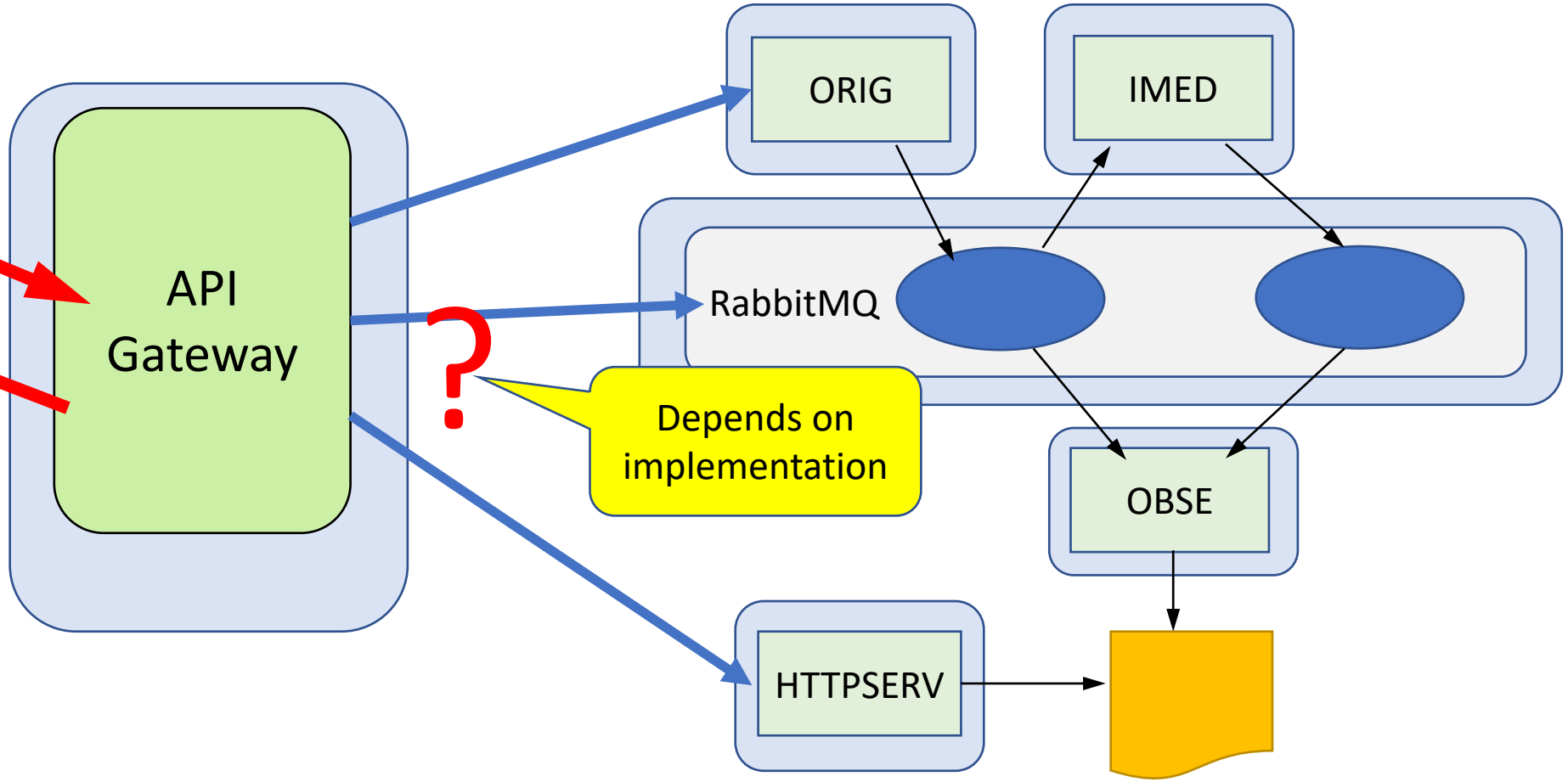


Tampereen yliopisto
Tampere University

GET /run-log

2020-11-01T06:35:01.373Z: INIT
2020-11-01T06:40:01.373Z: PAUSED
2020-11-01T06:40:01.373Z: RUNNING

Correct?



Tampereen yliopisto
Tampere University

Testing microservices

(https://www.infoq.com/articles/twelve-testing-techniques-microservices-intro/)

Key takeaways

- Because a microservice architecture relies more on over-the-wire (remote) dependencies and less on in-process components, your testing strategy and test environments need to adapt to these changes.
- When testing monoliths using existing techniques like service virtualization, you do not have to test everything together; instead, you can divide and conquer, and test individual modules or coherent groups of components.
- When working with microservices, there are also several more options available, because microservices are deployed typically in environments that use containers like Docker.
- You will need to manage the interdependent components in order to test microservices in a cost and time effective way. You can use test doubles in your microservice tests that pretend to be real dependencies for the purpose of the test.

Automation challenges

- ”…provisioning scripts were considered error-prone and, according to developers, they did not work in some environments…”
- ”…automation of the network in was said to be difficult in addition to dealing with legacy system…”
- ”Networks are pretty hard. Some of the databases are pretty hard too because the old relational databases haven’t been designed to be clustered…”

Tampereen yliopisto
Tampere University

Automation scripts are programs Infrastructure as code

- ”Infrastructure as code (IaC) is the process of managing and provisioning computer data centers through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools.”
- three approaches to IaC: declarative (functional) vs. imperative (procedural) vs. intelligent (environment aware)

```
tasks:
- name: ensure apache is at the
        latest version
  yum:
    name: httpd
    state: latest
- name: ensure that postgresql is started
    service:
      name: postgresql
      state: started
```

```
apt-get install …
```

Tampereen yliopisto
Tampere University

Infrastructure as code

All SW engineering principles should be applied.

- Testing
- Maintenance
- Documentation
- Version and configuration management

- Bugs may stop the whole engine

Tampereen yliopisto
Tampere University

Huge number or tools available

- https://digital.ai/periodic-table-of-devops-tools
- https://landscape.cncf.io