Tampere University

Rest of Cloud Native and project 07.11.2023

Kari Systä

Tampere University

Course status

SISU signups	109
Plus signups	150
Docker/compose exercise	86
Communication exercise	69
Ansible exercise	9

Tampereen yliopisto
Tampere University

Function as a service/ serverless computing

Do you really want to keep
your containers running all the time
if you need to pay for it?

Do you really want to operate
and maintain your containers –
your developers could also
do something else.

Tampereen yliopisto
Tampere University

Serverless computing

Baldini et all: Serverless Computing:

Current Trends and Open Problems, Research Advanges in Cloud Computing, Springer, 2017.

A cloud-native platform

for

- short-running, stateless computation
- event driven applications

which

- scale up and down instantly and automatically

and

- charge for actual usage and high granulatity

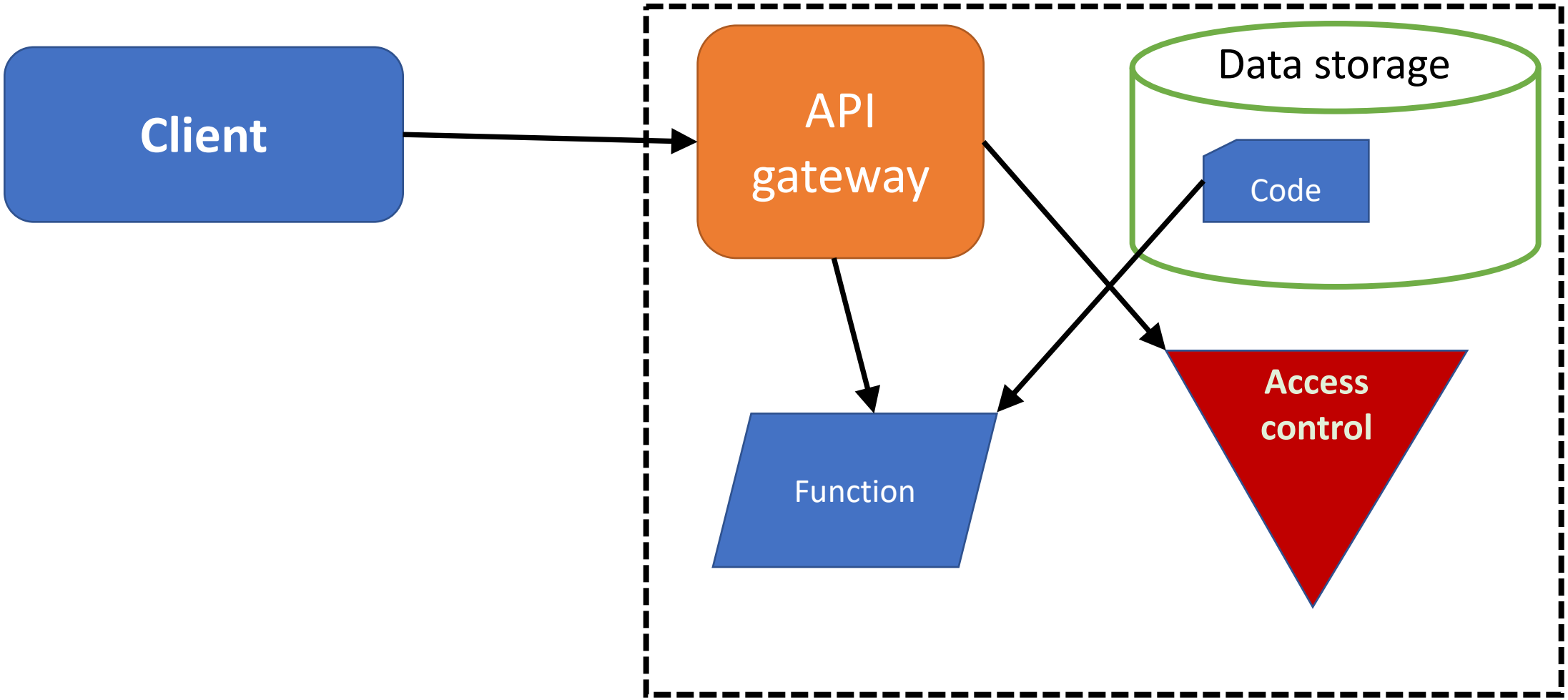
Tampereen yliopisto
Tampere University

https://medium.com/@BoweiHan/an-introduction-to-serverless-and-faas-functions-as-a-service-fb5cec0417b2

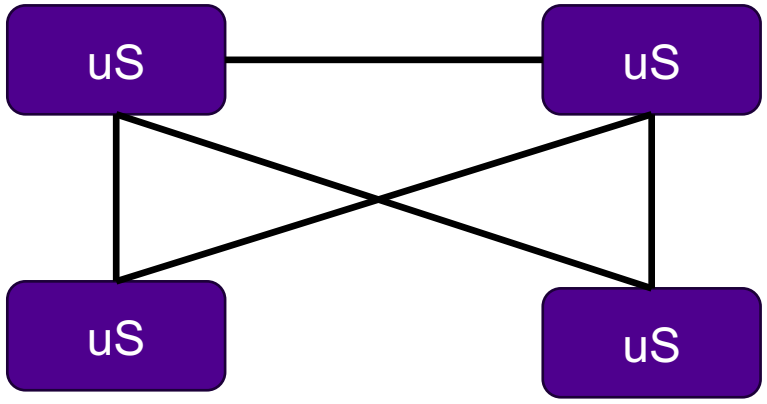
”… you can simply upload modular chunks of functionality into the cloud that are executed independently.
Imagine the possibilities! Instead of scaling a monolithic REST server to handle potential load, you can now split the server into a bunch of functions which can be scaled automatically and independently.”

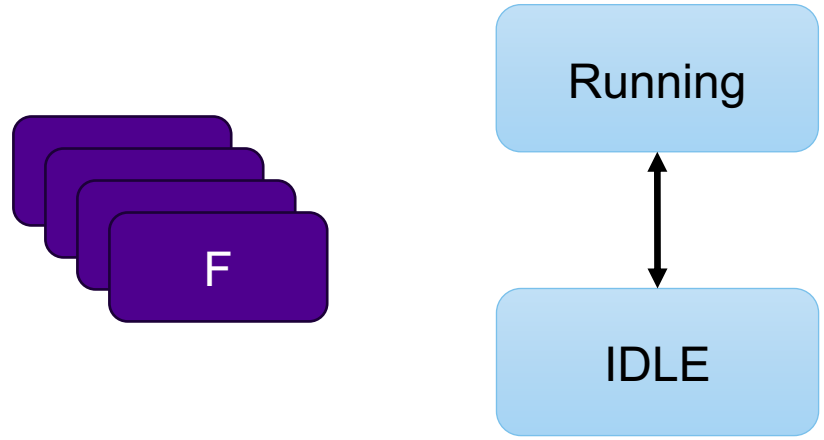
Tampereen yliopisto
Tampere University

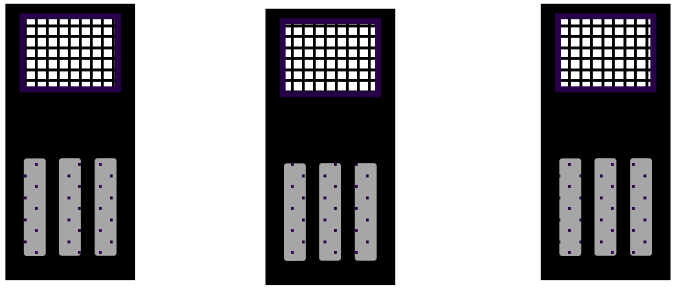
Function as a service?

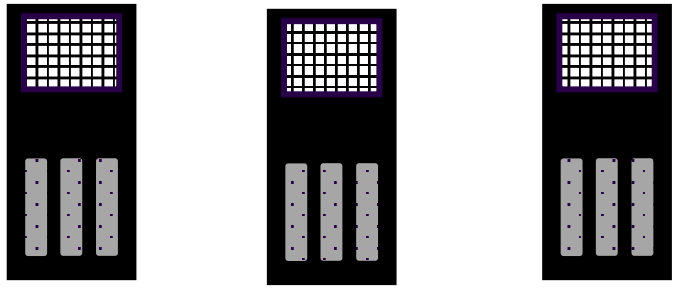


The difference









Claimed FaaS advantages

- Smaller for developer since infrastructure is handled by somebody else => more time for writing application code
- Inherently scalable
- No need to pay for idle resources (temptation to miss-use)
- Available and fault tolerant
- No explicit multi-tenancy
- Forces modular business logic

Tampereen yliopisto
Tampere University

Claimed FaaS disadvantages

- Decreased transparency
- Maybe challenging to debug
- Autoscaling of functions may lead to autoscaling of cost
- Keeping track of huge numbers of functions is tough
- Chaching of requests?

Tampereen yliopisto
Tampere University

Microservices vs. Serverless/FaaS
(They are different – do not call serveless microservices)

- Microservice
  - Small services running in their own process and communicating with lightweight services
  - Can be stateful

- Serverless / FaaS
  - Short term execution triggered by a request, then closes down
  - For stateless computing

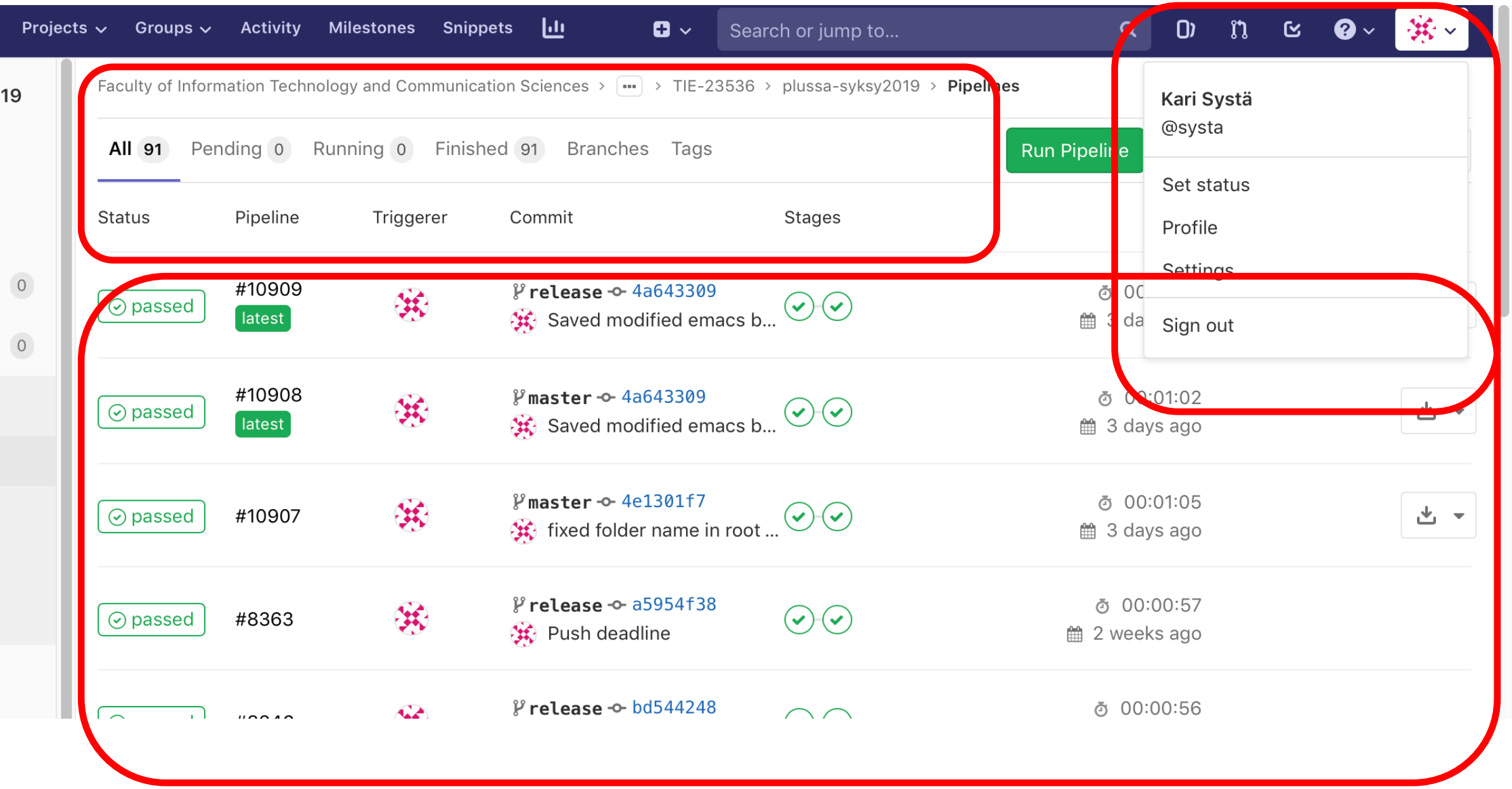
Tampereen yliopisto
Tampere University

Some comparison

	Microservice	Serverless / FaaS
Bug hunting	Easier (but not easy)	Difficult
Infrastructure code	May be complex	Minimal or even non-existent
Scaling	Need to be implemented	Automatic
Performance	Good	Possible cold-start issues
Running cost	May include cost of idle time	Pay only per use

Tampereen yliopisto
Tampere University

Microfront-ends



Information about architectures and implementation tricks on previous years' videos.

Tampereen yliopisto
Tampere University

7R's of cloud Micration

Replace
with similar or improved but SaaS

Reuse
in the new SaaS version

Refactor
towards cloud-native architecture

Replatform
by using cloud services

Rehost
to a VM

Retain

Retire

Tampereen yliopisto
Tampere University

Nice video about microservices

- Netflix story (Mastering Chaos - A Netflix Guide to Microservices) <https://www.youtube.com/watch?v=CZ3wIuvmHeM>

Tampereen yliopisto
Tampere University

Stateful vs stateless computation

- If a service has an internal state it is difficult to
  - Scale it
  - Move it to other server or other hosting system

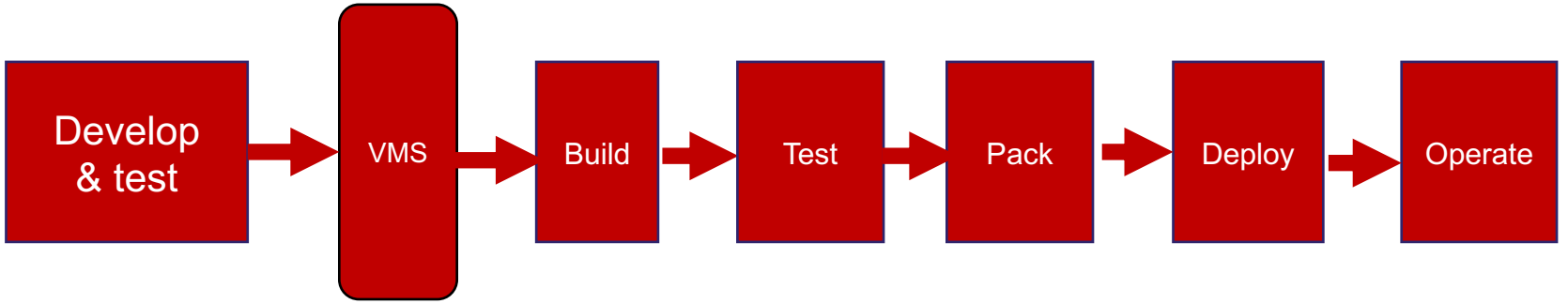
  => Stateless Services are subject to cloud-specific optimizations
- The internal state my be
  - volatile or
  - non-volatile
  - ... in memory, file local to container,

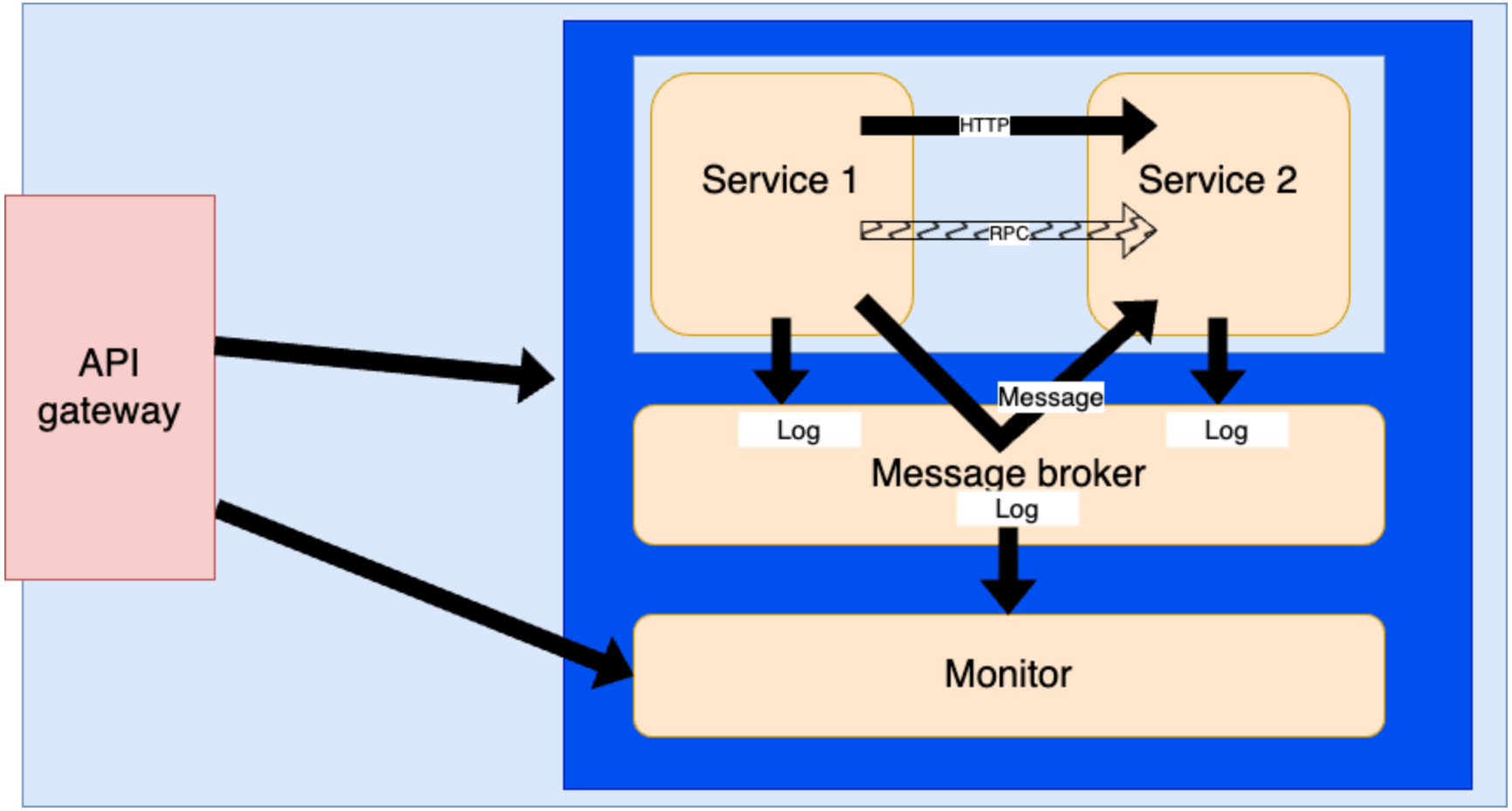
- Serverless / FaaS

Tampereen yliopisto
Tampere University

About the project

Project


Develop & test
VMS
Build
Test
Pack
Deploy
Operate


API gateway
Service 1
HTTP
Service 2
RPC
Message
Log
Log
Message broker
Log
Monitor

Schedule

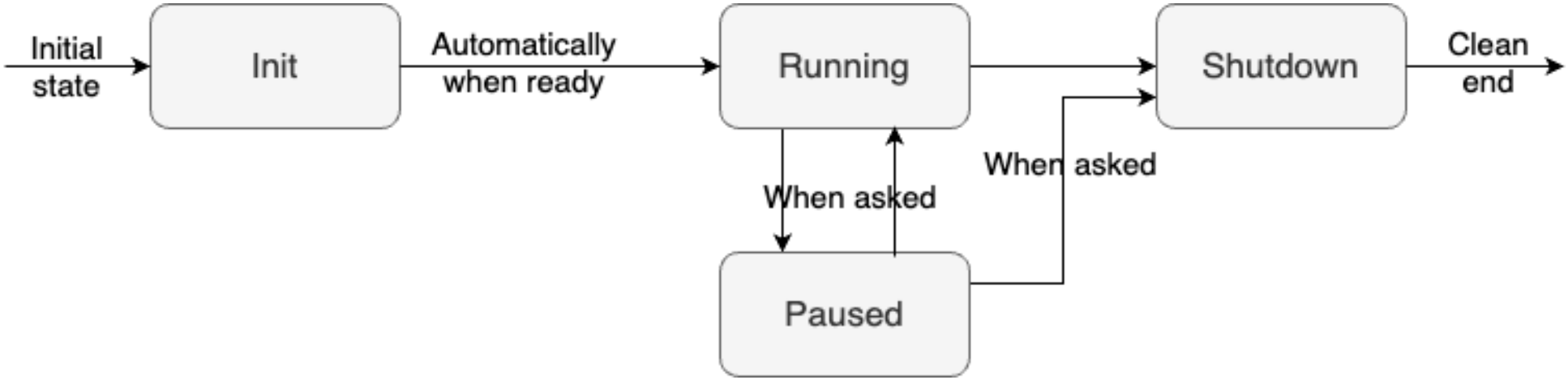
- The instructions disclosed: 07.11.2023
  - Students can start by installing the gitlab-ci
  - New versions to resolve ambiguous parts may be published later.
- Discussions in the lecture: 07.11.2023
  - Students are asked to give clarification questions
- Latest submission if you want course to graded in 2023: 07.12.2023
- Latest submission to pass the course: 31.01.2024

Project includes

1. Install the pipeline infrastructure using gitlab-ci. This means that you should:
   - Register your gitlab-runner to the new GitLab service
   - Define the pipeline using `.gitlab-ci.yml` for the application you implemented for the message-queue exercise. The result of the pipeline should be a running system, so the containers should be started automatically. (In other words: “git push => the system is up and running)
   - Test the pipeline with the current version of the application.
2. Create, setup and test an automatic testing framework
   - First, you need to select the testing tools. We do not require any specific tool, even your own test scripts can be used.
   - Create test to the existing functionality of the application (see “Application and its new features” below)
3. Implement the changes and additional functionalities to the mesaaging exercise

The application

1. The system should run “forever” unless explicitly stopped. This means that services 2 should not stop sending after 20 rounds.
2. The system should have the following states:



3. The most notable new component is an API gateway service that provides the external interface to the system. This service should be exposed from port 8083. The API gateway should provide the following REST-like API
(See next slide)

API gateway

GET /messages

Returns all message registered by the Monitor-service

Example response (part of):

- ***SND 1 2022-10-01T06:35:01.373Z 192.168.2.22:8000***
  SND 1 2022-10-01T06:35:01.373Z 192.168.2.22:8000 192.168.2.21:78390

PUT /state (payload “INIT”, “PAUSED”, “RUNNING”, “SHUTDOWN”)

PAUSED = Service 1 is not sending messages

RUNNING = Service 1 sends messages

If the new state is equal to previous nothing happens.

There are two special cases:

INIT = everything (except log information for /run-log and /messages) is set to the initial state and Service starts sending again, and state is set to RUNNING

SHUTDOWN = all containers are stopped

GET /state

get the value of state

API gateway

GET /run-log

Get information about state changes

Example output:

```
2023-11-01T06.35:01.380Z: INIT->RUNNING
2023-11-01T06:40:01.373Z: RUNNING->PAUSED
2023-11-01T06:40:01.373Z: PAUSET->RUNNING
```

GET /message-log

Forward the request to Monitor and return the result

GET /mqstatistic (optional) (in JSON)

Return core overall statistics (the five (5) most important in your mind) of the RabbitMQ, and in addition for each queue return "message delivery rate", "messages publishing rate", "messages delivered recently", "message published lately. (For getting the information see https://www.rabbitmq.com/monitoring.html)

Output should be syntactically correct and intuitive JSON document containing overall and per queue statistics.

Tampereen yliopisto
Tampere University

End report

1. **Instructions for the teaching assistant**

   Implemented optional features

   List of optional features implemented.

   Instructions for examiner to test the system.

   Pay attention to optional features.

2. **Description of the CI/CD pipeline**

   Briefly document all steps:

   Version management; use of branches etc

   Building tools

   Testing; tools and test cases

   Packing

   Deployment

   Operating; monitoring

3. **Example runs of the pipeline**

   Include some kind of log of both failing test and passing.

4. **Reflections**

   Main learnings and worst difficulties

   Especially, if you think that something should have been done differently, describe it here.

   Amount effort (hours) used

   Give your estimate

Grading

As already been communicated this project affects 40% of in the evaluation of the overall course. For that 40% we use the following table

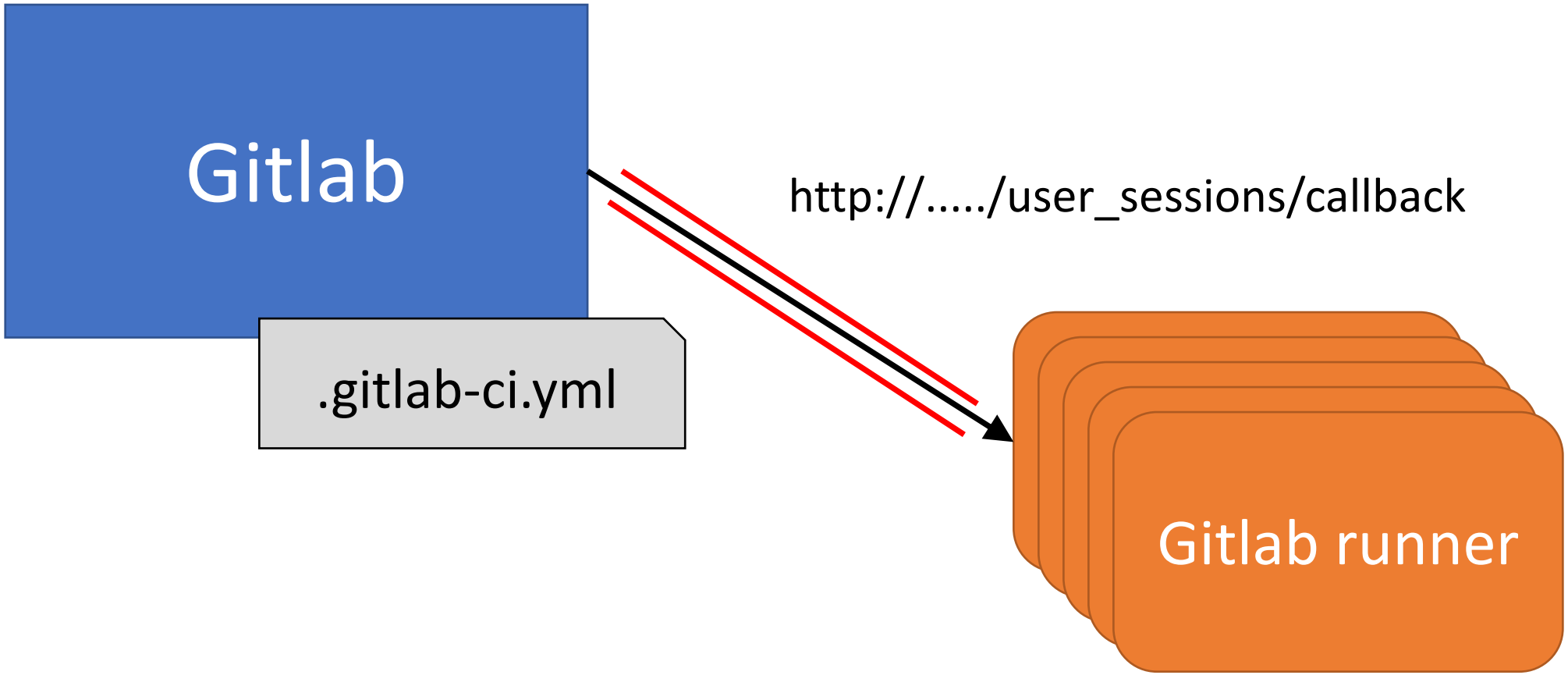
The system work according to requirements	0..20 %
The CI/CD pipeline is clean and complete	0..10 %
Implementation of optional features (each optional feature is worth of 5%)	0..25%
Overall quality (clean code, good comments, ….)	0..5%
Quality of the end report	0..5% (+ up to 5% compensation of a good analysis of your solution and description of a better way to implement.)

Note: optional points can compensate problems elsewhere, but the total sum is capped at 50%. That means that max 10% can be used to compensate lost points in exercises and exam.

Tampereen yliopisto
Tampere University

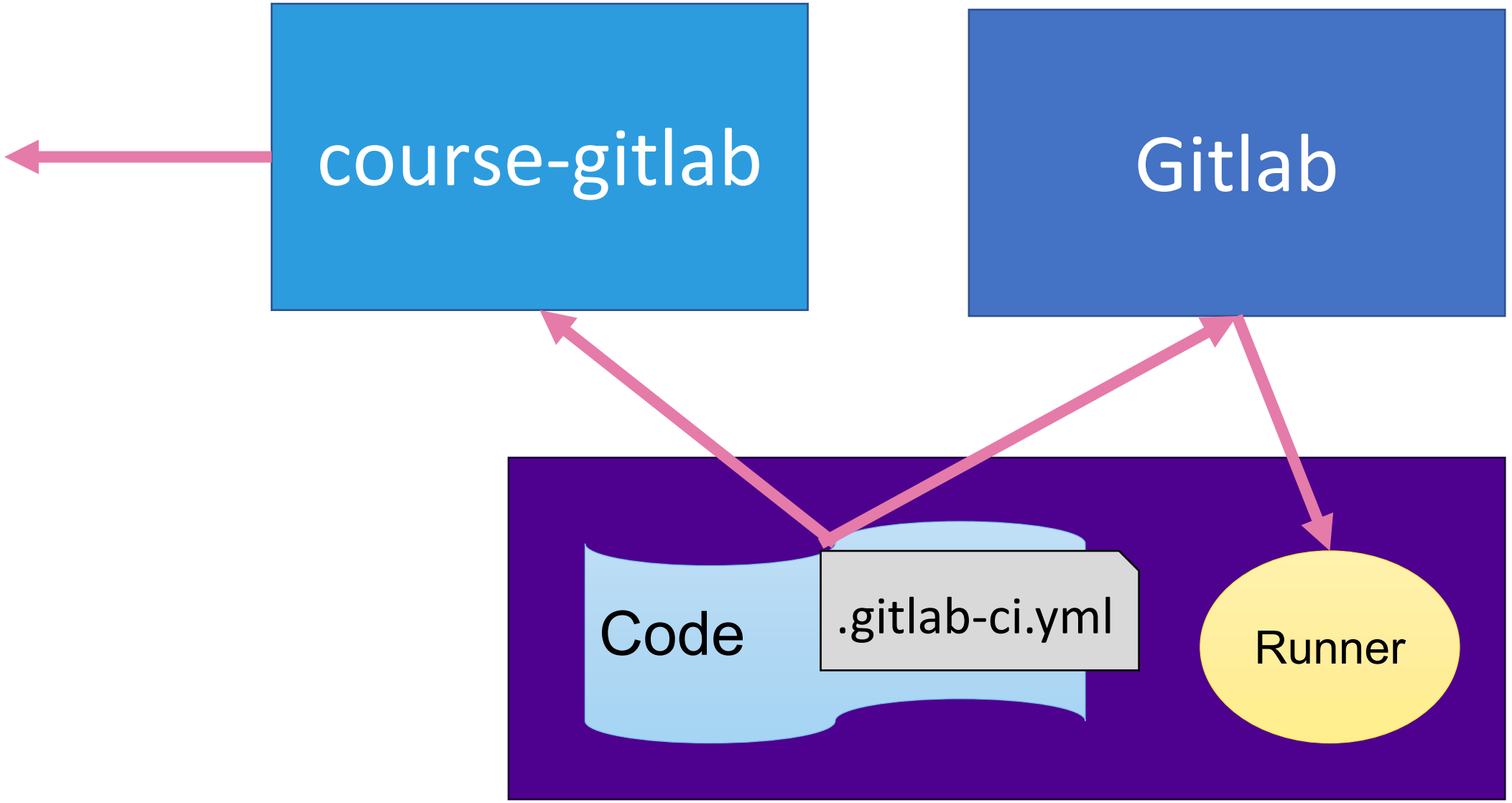
Gitlab CI

https://docs.gitlab.com/ce/ci/



Tampere University

Your version management with GitLab CI



Tampereen yliopisto
Tampere University

Types of runners

Shared Runners

- These runners are useful for jobs multiple projects which have similar requirements. Instead of using multiple runners for many projects, you can use a single or a small number of Runners to handle multiple projects which will be easy to maintain and update.

Specific Runners

- These runners are useful to deploy a certain project, if jobs have certain requirements or specific demand for the projects. Specific runners use *FIFO* (First In First Out) process for organizing the data with first-come first-served basis.

Tampereen yliopisto
Tampere University

How to install .gitlab-ci.yml?

```
git add .gitlab-ci.yml
git commit -m "Add .gitlab-ci.yml”
git push origin master
```

passed	#2913		master 43dda676  more tests		00:00:36 1 month ago
passed	#2912		master 32e0f29b  more tests		00:00:36 1 month ago
failed	#2911		master 8bf6c037  more tests		00:00:16 1 month ago

```
Sphinx error:
Missing config path exercises/hello__hello/config.yaml
make: *** [html] Error 1
Makefile:60: recipe for target 'html' failed

*** ERROR in compile-rst

▼
▼
ERROR: Job failed: exit code 1
```