# Project document

## Instructions for the teaching assistant

### Implemented optional features

- Static analysis step in the pipeline. Using linting and  ESLint's recommended rules.

### Instructions for examiner to test the system.

This project was developed and tested on:

- Operating System: Ubuntu (WSL 2 on Windows 10)
- CPU architecture: x86_64 (AMD64)
- Hardware:  Intel core i7-1165G7
- Docker version: 26.1.4
- Docker compose version: v2.27.1-desktop.1

The project can be run with native Linux. This enables direct hardware access, direct access to host networking and no dependency on Windows Docker desktop as Docker runs natively.

### Description of the CI/CD pipeline

Version management

- Branching:
  Code is placed in 'project' branch.
- The pipeline runs when pushing to the repository

Build

- Tools: Docker, Docker Compose

Testing

- Tools: Jest
- Test cases
  - GET /state :
    System starts in the correct **INIT** state
  - State transitions
    Transfers to RUNNING after authentication
    Using PUT /state, transfers to 'PAUSED' and 'RUNNING'
  - GET /run-log:
    After authentication, retrieves run log containing the correct state in text format
  - GET /request

After authentication, when the state is ‘RUNNING’ retrieve service information in text form containing ‘service1’ and ‘service2’

Packaging

- Docker images:
  Node.js (latest), builds JavaScript based services (service1, api gateway).
  Rust (latest), builds Rust based service (service2).
- The services are packaged as Docker containers.
- The nginx service is included for load balancing

Deployment

- Deployment method: Local deployment using Docker Compose
- Command: docker-compose up -d
- The deployment can be verified locally using
  docker ps
- The deployment can be stopped using
  docker-compose down

Operations; Monitoring

- Logging & Monitoring:
  Nginx Logs: mounted to ./nginx_logs/
  Docker Logging: Can be checked with: docker logs <container_name>

# Example runs of the pipeline

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

Logs of failed tests #3084 and succeeded tests #3090 are attached in the end.
This demonstrates development of new feature run-log and running the pipeline:

-> Implementing tests -> Development until test succeeds -> implementing test for new feature

# Reflections

I had no previous experience on building pipelines, so finding information about different stages and how to implement them took some time, even though I think I understood the basics of the steps and their purpose.

With the gateway api I struggled a lot with verifying the basic authentication status and requiring the user to login again after initializing the state. This was because the browser cache. I tried sending Unauthorized status code until user logs in again, but refreshing the page gave access to user without having to log in again, because of the cache. I wasn't able to find a solution for forcing a re-log in with nginx basic authentication. Token based authentication would have given me the solution, but I think basic authentication was assumed with this project. Also mounting the logs and docker socket took a lot of time to figure out, and doesn't seem like a secure way to implement the features

Maybe most important thing I wanted to do differently, but failed was using a docker based e Gitlab runner. I had issues with Docker in Docker set up, even though I mounted the docker socket and directed docker commands to docker socket. I think my development environment (WSL) possibly caused some problems I could't understand, and I ended up setting up shell based Gitlab runner.

I could have done some things with my implementation differently, like using env variables and .env file for port numbers, links, etc. Also, each feature could have been implemented in their own branches, and merged to the project branch once feature is ready and tested with the pipeline. Using linting or static analysis from the beginning would have been useful, as I ended up cleaning up code in the end.

Would be very helpful for learning to see what kind of solutions other people implemented, as for example my solutions with basic authentication didn't work exactly like was described in the instructions.

I used estimately seven days (~52 hours) for this project.

# Pipeline step test logs

1. Failing test log

```
$ echo "Running tests..."
Running tests...
$ npm test
> devops@1.0.0 test
> jest
FAIL tests/api.test.js
● Console
console.log
```

```
State reset to INIT: 200
at Object.log (tests/api.test.js:44:17)
console.log
State reset to INIT: 200
at Object.log (tests/api.test.js:44:17)
console.log
Authentication succeeded: 200
at log (tests/api.test.js:20:17)
console.log
State reset to INIT: 200
at Object.log (tests/api.test.js:44:17)
console.log
State reset to INIT: 200
at Object.log (tests/api.test.js:44:17)
● GET /run-log › should return run log containig state RUNNING
expect(received).toContain(expected) // indexOf
Expected substring: "RUNNING"
Received string: "TODO"
129 | });
130 | expect(res.status).toBe(200);
> 131 | expect(res.data).toContain('RUNNING');
| ^
132 |
133 | });
134 | });
at Object.toContain (tests/api.test.js:131:26)
PASS tests/service1.test.js
Test Suites: 1 failed, 1 passed, 2 total
Tests: 1 failed, 4 passed, 5 total
Snapshots: 0 total
Time: 0.731 s, estimated 1 s
Ran all test suites.
Running after_script 00:23
Running after script…
```

2. Succeeding test log

```
Running with gitlab-runner 17.7.0 (3153ccc6)
on devops t3_LjRtK, system ID: s_fac3deba48b3
Preparing the "shell" executor 00:00
Using Shell (bash) executor...
```

```
Preparing environment 00:00
Running on LAPTOP-7F9TPCUD...
Getting source from Git repository 00:01
Fetching changes with git depth set to 20...
Reinitialized existing Git repository in
/home/suvi/devops/builds/t3_LjRtK/0/suvij/devops/.git/
Checking out 61395454 as detached HEAD (ref is project)...
Skipping Git submodules setup
Executing "step_script" stage of the job script 00:27
$ echo "Docker Compose up..."
Docker Compose up...
$ docker-compose up -d
Container service2 Running
Container api_gateway Recreate
Container devops-service1-2 Recreate
Container devops-service1-1 Recreate
Container devops-service1-3 Recreate
Container devops-service1-3 Recreated
Container devops-service1-1 Recreated
Container devops-service1-2 Recreated
Container api_gateway Recreated
Container nginx Recreate
Container nginx Recreated
Container api_gateway Starting
Container devops-service1-3 Starting
Container devops-service1-3 Started
Container devops-service1-2 Starting
Container api_gateway Started
Container devops-service1-2 Started
Container devops-service1-1 Starting
Container devops-service1-1 Started
Container nginx Starting
Container nginx Started
$ npm install
npm warn deprecated inflight@1.0.6: This module is not supported,
and leaks memory. Do not use it. Check out lru-cache if you want a
good and tested way to coalesce async requests by a key value, which
is much more comprehensive and powerful.
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no
longer supported
added 356 packages, and audited 357 packages in 2s
```

```
49 packages are looking for funding
run `npm fund` for details
found 0 vulnerabilities
$ echo "Running tests..."
Running tests...
$ npm test
> devops@1.0.0 test
> jest
PASS tests/api.test.js
● Console
console.log
State reset to INIT: 200
at Object.log (tests/api.test.js:44:17)
console.log
State reset to INIT: 200
at Object.log (tests/api.test.js:44:17)
console.log
Authentication succeeded: 200
at log (tests/api.test.js:20:17)
console.log
State reset to INIT: 200
at Object.log (tests/api.test.js:44:17)
console.log
State reset to INIT: 200
at Object.log (tests/api.test.js:44:17)
PASS tests/service1.test.js
Test Suites: 2 passed, 2 total
Tests: 5 passed, 5 total
Snapshots: 0 total
Time: 0.723 s, estimated 1 s
Ran all test suites.
Running after_script 00:23
Running after script...
$ echo "Shutting down services..."
Shutting down services...
$ docker-compose down
Container nginx Stopping
Container nginx Stopped
Container nginx Removing
Container nginx Removed
Container api_gateway Stopping
```

```
Container devops-service1-1 Stopping
Container devops-service1-3 Stopping
Container devops-service1-2 Stopping
Container devops-service1-3 Stopped
Container devops-service1-3 Removing
Container devops-service1-3 Removed
Container devops-service1-1 Stopped
Container devops-service1-1 Removing
Container devops-service1-1 Removed
Container devops-service1-2 Stopped
Container devops-service1-2 Removing
Container devops-service1-2 Removed
Container service2 Stopping
Container service2 Stopped
Container service2 Removing
Container service2 Removed
Container api_gateway Stopped
Container api_gateway Removing
Container api_gateway Removed
Network devops_backend Removing
Network devops_frontend Removing
Network devops_backend Removed
Network devops_frontend Removed
Cleaning up project directory and file based variables 00:00
Job succeeded
```