



28.1.2025

# COMP.SE.140 FINAL PROJECT

JUHANA KIVELÄ

<https://github.com/JuhanaKivela/DevOps-Exercises/tree/project>

<https://compse140.devops-gitlab.rd.tuni.fi/nnjuki/comp.se.140-pipeline>



# 1. Instructions for the teaching assistant

## 1.1. Implemented optional features

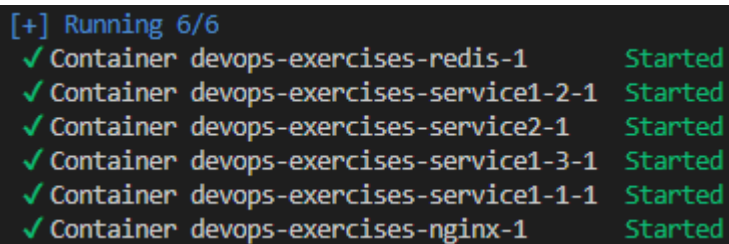
The project does not contain any of the optional features.

## 1.2. Instructions for examiner to test the system

**To run the project**, the GitHub repo can be cloned, and Docker Compose can be used to start all the services:

```
git clone --branch project https://github.com/JuhanaKivela/DevOps-Exercises.git
cd DevOps-Exercises
docker-compose build --no-cache
docker-compose up -d
```

This will launch 6 services. Note that it will take ~30 seconds for the services to boot up.

A terminal window with a dark background showing the output of a Docker Compose command. It displays a status summary followed by a list of six containers, each with a green checkmark, its name, and the word 'Started' in green.

```
[+] Running 6/6
✓ Container devops-exercises-redis-1      Started
✓ Container devops-exercises-service1-2-1 Started
✓ Container devops-exercises-service2-1   Started
✓ Container devops-exercises-service1-3-1 Started
✓ Container devops-exercises-service1-1-1 Started
✓ Container devops-exercises-nginx-1      Started
```

**The tests** include mechanism to wait for services to boot up before the tests are run. Tests can be run with the following commands:

```
cd tests
mvn test
```

The project can be also tested with e.g. Postman.

No authentication needed:

GET <http://localhost:8197/run-log>

GET <http://localhost:8197/status>

GET <http://localhost:8197/request>

Authentication needed:

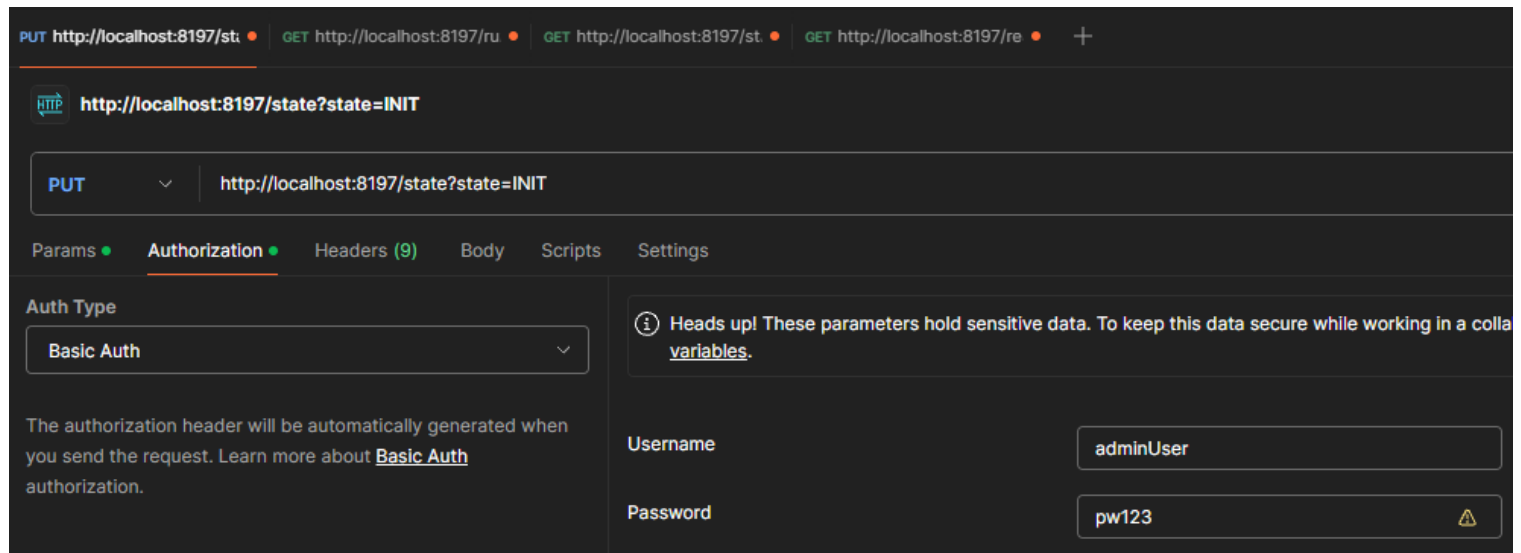
PUT <http://localhost:8197/state?state=INIT>

PUT <http://localhost:8197/state?state=RUNNING>

PUT http://localhost:8197/state?state=PAUSED

PUT http://localhost:8197/state?state=SHUTDOWN

The authentication can be handled with Postman basic auth:



### 1.3 Info about my system

CPU	AMD Ryzen 9 3900 XT, 12 Core 3.8 GHz, Zen 2 architecture
Motherboard	MSI X570-A Pro AM4 ATX
RAM	32 GB DDR4 (2x 16GB) 3200MHz
Hard drive	Samsung M.2 970 EVOPlus 1TB
OS	MS Windows 10 Home
Docker Compose	Docker Compose version v2.24.6-desktop.1
Docker	Docker version 25.0.3, build 4debf41

## 2. Description of the CI/CD pipeline

I tried to create the CI/CD pipeline and test cases as early as possible and start utilizing them from early on. I firstly tried to use Node.js + Mocha + Chai for the API testing, but I found that it had slightly less features and customization than what I was used to. So I switched to Java and Junit, which gave me a lot of freedom on the testing implementation. I quickly created the first test case for the TDD development and from there it was easy to develop the feature and then create new test case for the next feature again.

**TDD:** Add API tests for PUT /state

Juhana Kivelä committed last month

Add system status check on Service1

Juhana Kivelä committed last month

**TDD:** Add API test for /run-log


Juhana Kivelä committed last month

Implement state change logging and GET /run-log endpoint


Juhana Kivelä committed last month

Commits on Jan 6, 2025

nginx same ip calls are redirected to a single service

 JuhanaKivela committed 3 weeks ago

Service1 can't return get state or get run-log when paused

 JuhanaKivela committed 3 weeks ago

**TDD:** Iterate tests

 JuhanaKivela committed 3 weeks ago

Commits on Jan 7, 2025

**TDD:** Iterate test for run-log

 JuhanaKivela committed 3 weeks ago

**TDD:** Consider auth in tests

 JuhanaKivela committed 3 weeks ago

Implement auth for PUT methods

 JuhanaKivela committed 3 weeks ago

*In the "TDD:" starting commits I developed new test case for the upcoming feature.*

I created one test case per one API endpoint. On top of that there are couple of extra test cases testing the state changes and authentication. The tests also wait for Service1 to boot up before they are run. Otherwise the API tests would all return 502 and mark them as failed. The PUT /state SHUTDOWN test case is skipped as I was unable to completely implement it. It only shuts down 3 out of 6 containers.

The new development mostly happened for the Service1 and since I had built it in Java, I used Apache Maven as my main build tool. It is already familiar to me from previous

projects, and I had good experience with it, so I decided to stick with it on this project too.

I mainly used GitHub's "project" branch for the app source code and GitLab's "main" branch for the pipeline (.gitlab-ci.yml). I only switched to different branches if I wanted to save some experimental approach for later use. Since many of the implementations were new to me and required some testing & research to get working.

Since the app source code is in a different location than CI/CD pipeline, the building phase of it mainly consists of cloning the GitHub repo and giving the files to the test\_job. The test\_job runs the tests and fails if there are any failed test cases. If testing is passed, it goes forward to deploy\_job, which just runs docker-compose up. This is fairly simple implementation, but I was unsure on how to develop it differently since we're using Docker Compose.

There are not any operating or monitoring parts in the CI/CD pipeline. I'm not too familiar with them and in this project I had to leave them out of the project scope.

## 2.1 Example runs of the pipeline

Passing pipeline:

[build\\_job](#)

[test\\_job](#)

[deploy\\_job](#)

Failing pipeline:

[build\\_job](#) (passed)

[test\\_job](#) (failed, 4 passed test cases and 2 failed)

[deploy\\_job](#) (not run)

## 3. Reflections

One of the big painpoints for me was the authentication. Since originally it was requested that the port 8198 should have credentials asked before the main page is displayed and I handled that in Nginx since it was hard to do on Service1. But later on it was also requested that only specific endpoints on port 8197 require authentication. I tried implementing it on Nginx at the beginning, but it became too messy and complicated, so I switched to basic auth on Service1 side. This also meant that now the auth is handled on two separate places for two different ports. Better solution could

have been using some separate service or trying to put all the authentication in Service1.

One interesting issue was that the three instances of Service1 had all different logs and system state. So they were all stateful, but in different states. I had not encountered this kind of issue on previous sw projects, so it was interesting to learn how to handle this kind of situation. I decided to go with external service Redis, which holds the state and logs. This way all instances of Service1's can just read values from it or add logs to it. I believe this was correct decision as it's simple and effective way to sync the state between services.

In the beginning of the project I could not get a clear understanding of how to handle the development when pipeline and app source code are placed in separate repositories. Since I wanted the pipeline to trigger every time code is pushed to apps source code repo. But at least websocket could not be implement most likely due to TUNI's GitLab restrictions. So I had to manually trigger the pipeline every time I want to run it. This would not be optimal situation in a professional environment, but I believe in there the .gitlab-ci.yml would be placed on the same repo as the app source code, which then solves the issue.

I spent roughly 60 hours on this project. Around 10h was researching and understanding new concepts, 10h were used to set up the pipeline and runner for my local machine and 40h to develop the app.