Instructions for the teaching assistant (Also in README.md, but contains less info)

1. List of optional features implemented:

- Testing of individual services: **Done (test step in pipeline, somewhat extensive test suite, in /test, testing service1, up to you if this is enough, see ./test/test folder)**
- Static analysis step in the pipeline: **Done (eslint step in pipeline, linting service1, see gitlab-ci.yml)**
- Monitoring and logging for troubleshooting: **Done (in browser view, minimal, see browser view)**
- Deployment to external server: **Done (with cPouta Ubuntu 22.04 image, IP 86.50.231.95, see IP address)**

2. Instructions for examiner to test the system, pay attention to optional features.

Testing the system can be done through following addresses:

Example: http://86.50.231.95:8197/state

http://86.50.231.95:8197/something -> API (**username: user1, password: your_mom**)
- /state PUT & GET (PUT needs auth) -> System state management
- /request GET -> Request service data
- /run-log GET -> Run log

http://86.50.231.95:8198/something -> Browser front (**username: user1, password: your_mom**)
- /controlpanel.html -> Main page
- /shutdown POST -> Shutdown containers
- /api GET -> Service request
- /debug-monitor GET -> Monitor system

Alternatively you can build the system locally (if for example the cPouta service has crashed at the time of inspection, lol):

Installation:

1. *Install docker-compose and git on computer*
2. git clone -b project https://github.com/Ka1aschNikoV/course-devops.git
3. cd course-devops
4. sudo docker-compose build --no-cache
5. sudo docker-compose up -d

Finally:
Enter in browser:
- http://localhost:8197 -> API
- http://localhost:8198 -> Browser front

Auth is done through either
- Browser -> insert username password manually in prompt (easier)
- API -> Send authentication header (more cumbersome)

After authenticated, the test command with curl:

curl localhost:8197/state -X PUT -d "PAUSED" -H "Content-Type: text/plain" -H "Accept: text/plain"

6. And finally: sudo docker-compose down

3. Data about the platform you used in development (hardware, CPU architecture, operating system, version of docker and docker-compose)

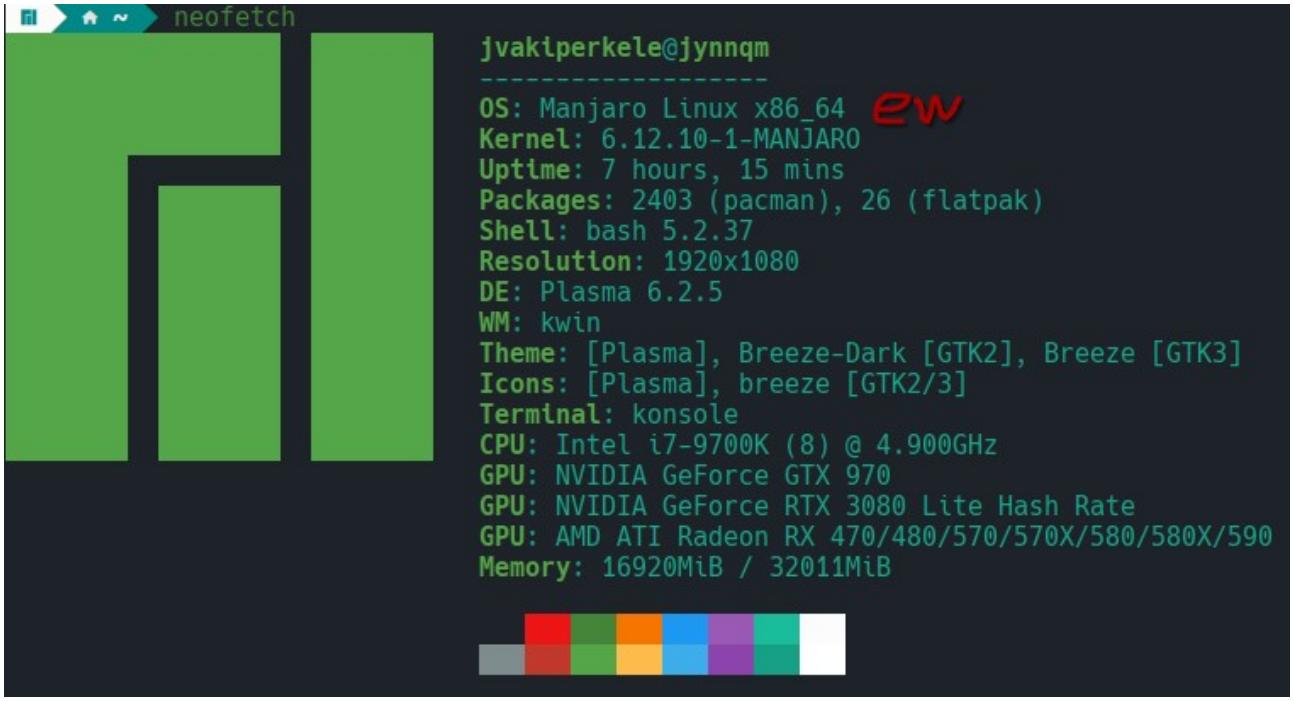


Image 1: Dev environment

- Processor: Coffee Lake Intel Core i7 9700k, x86
- Docker: docker-1:27.3.1-1
- Docker-compose: locally docker-compose-2.32.4-1 (runner and server, version docker-compose-1.29.2-1), also see Reflections

4. Description of the CI/CD pipeline

Briefly document all steps:

• **Version management; use of branches etc**

- Branch "project" was used in both gitlab and github, and is the only branch to be inspected. Push into GitLab repo triggers gitlab-runner actions

• **Building tools**

- Locally using docker compose up --build, on deployment server docker-compose up -d. gitlab-runner was used on the host computer, containerized with gitlab-runner image, run on host computer with sudo docker compose up --build. Runner does 4 jobs, eslint-job, build-job, test-job, deploy-server

• **Testing; tools and test cases**

- Testing with mocha/chai. Testing is done with its own test container, and tests cover all API's and services in some form. Testing is quite general, since I did not want to spend too much time with formatting tests to fit my purposes, the most complex one is probably the gatewayGetState service1 and service2 content detection. In general, the tests cover basic outputs from success/fail status codes and returned text/plain values. Test driven development is more evident in initial commits with a genuine effort made, there were some excessive spam commits and accidental secret leaks when debugging the gitlab-runner + docker combo later on, below is a snippet of the earlier commit history.

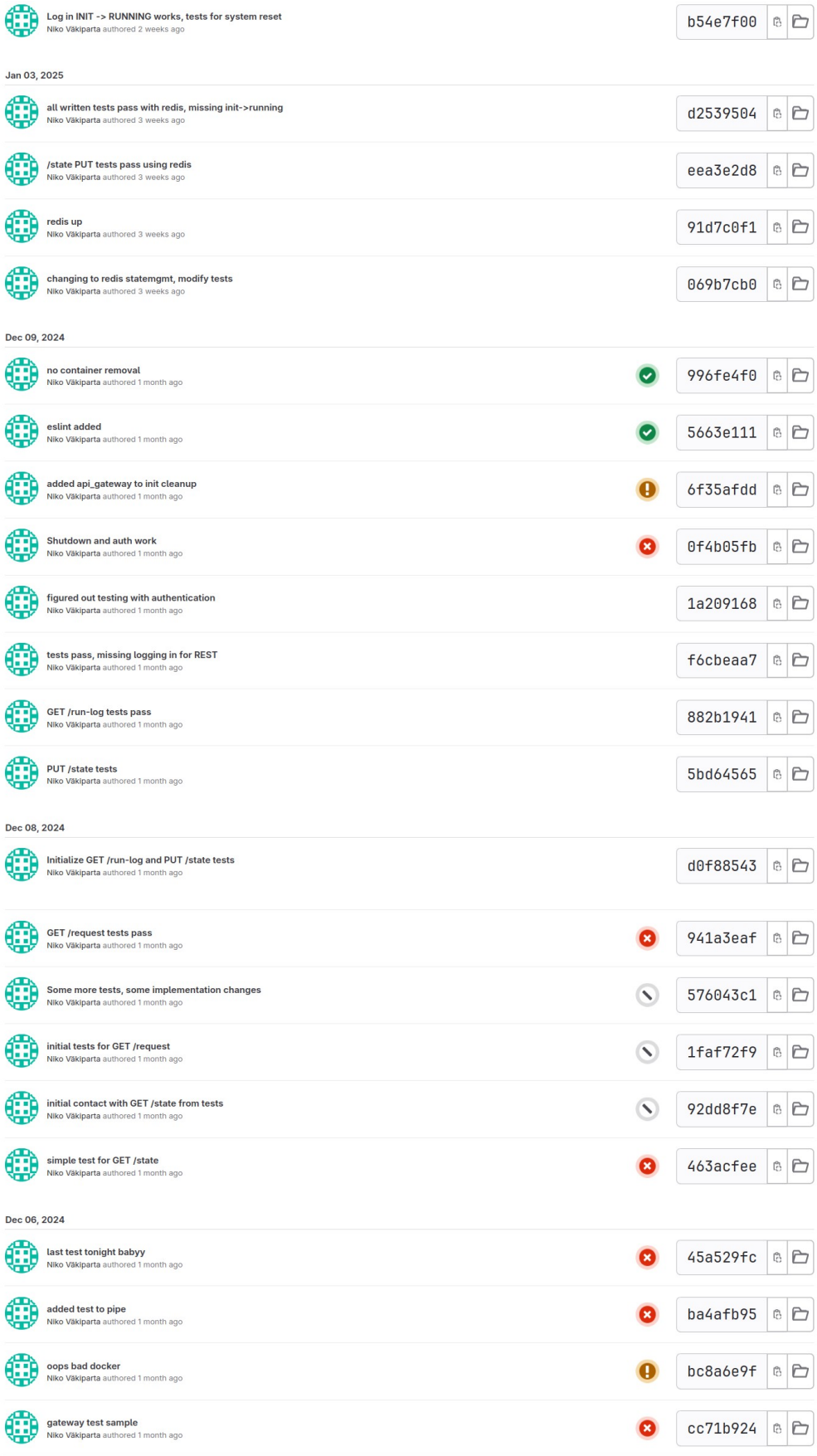


Image 2: Snippet of commit history

• **Packing**

- Not done, images are sent to GitLab container registry in build, and are pulled in deployment step, files are scp'd over from gitlab-runner to deployment

• **Deployment**

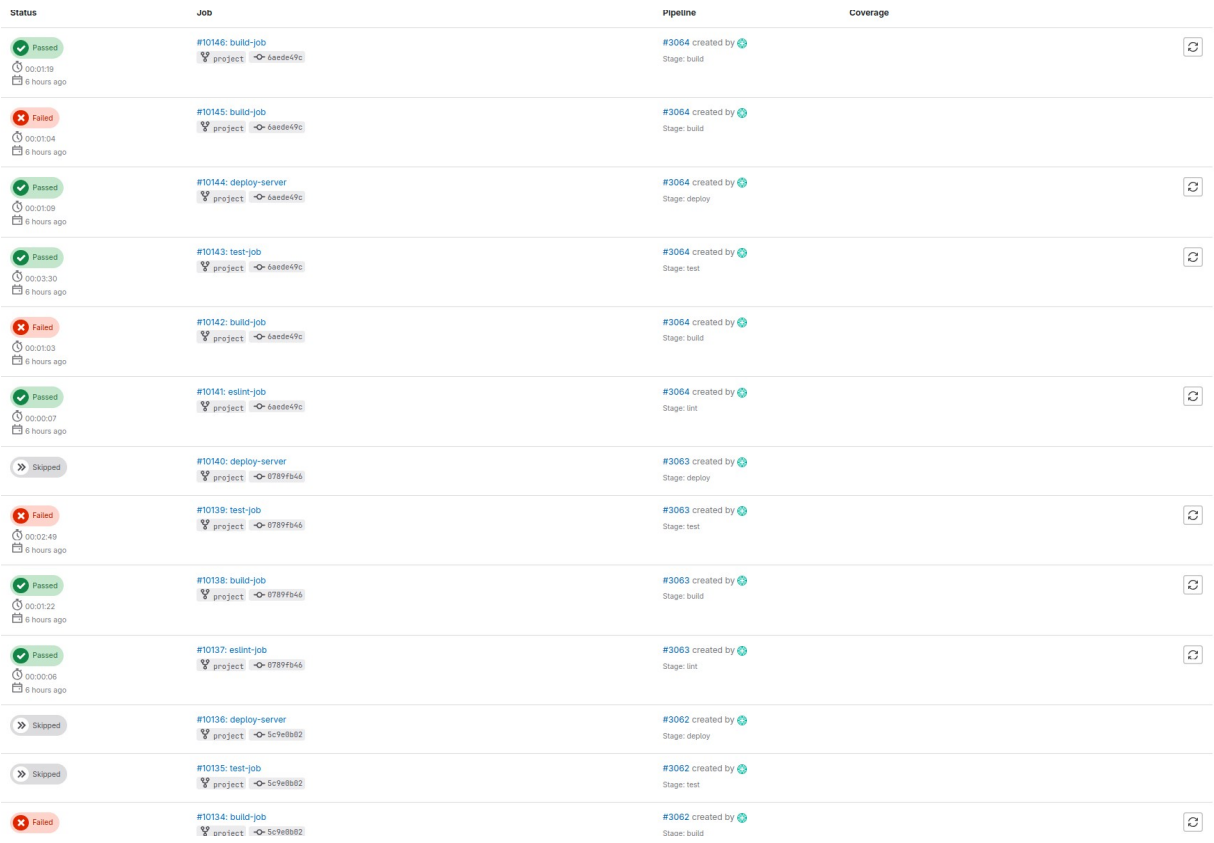
- Done as instructed on cPouta server, with IP 86.50.231.95

• **Operating; monitoring**

- Minimal monitoring implemented in browser, operation through browser or 8197 port API endpoints
- Pipeline is operated on separate machine with a containerized gitlab-runner, run with **sudo docker-compose up --build**. Since config.toml is a risky share, I included it as a picture instead with token blurred (./gitlab-runner/config/image_of_config_toml.png). Pipeline uses docker executor because shell executor was more cumbersome.

Example runs of the pipeline

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



Status	Job	Pipeline	Coverage
Passed 00:01:19 6 hours ago	#10146: build-job project 6aede49c	#3064 created by Stage: build	
Failed 00:01:04 6 hours ago	#10145: build-job project 6aede49c	#3064 created by Stage: build	
Passed 00:01:09 6 hours ago	#10144: deploy-server project 6aede49c	#3064 created by Stage: deploy	
Passed 00:03:30 6 hours ago	#10143: test-job project 6aede49c	#3064 created by Stage: test	
Failed 00:01:03 6 hours ago	#10142: build-job project 6aede49c	#3064 created by Stage: build	
Passed 00:00:07 6 hours ago	#10141: eslint-job project 6aede49c	#3064 created by Stage: lint	
Skipped	#10140: deploy-server project 0789fb46	#3063 created by Stage: deploy	
Failed 00:02:49 6 hours ago	#10139: test-job project 0789fb46	#3063 created by Stage: test	
Passed 00:01:22 6 hours ago	#10138: build-job project 0789fb46	#3063 created by Stage: build	
Passed 00:00:06 6 hours ago	#10137: eslint-job project 0789fb46	#3063 created by Stage: lint	
Skipped	#10136: deploy-server project 5c9e0b02	#3062 created by Stage: deploy	
Skipped	#10135: test-job project 5c9e0b02	#3062 created by Stage: test	
Failed	#10134: build-job project 5c9e0b02	#3062 created by Stage: build	

Image 3: Pipeline successes and fails

```
Reinitialized existing Git repository in /builds/dpniva/devops-project/.git/
Checking out e8609520 as detached HEAD (ref is project)...
Removing node_modules/
Skipping Git submodules setup
Executing "step_script" stage of the job script                                00:03
Using docker image sha256:89871f29e084e3df96860403fff8bc63410d1e77efec0a13086aeb0af459aab6 for node:latest with digest node@sha256:3b73c4b366d490f76908dda253bb4516bbb3398948fd880d8682c5ef16427eca ...
$ npm install
added 261 packages, and audited 262 packages in 2s
61 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
$ npm run lint
> devops-project@1.0.0 lint
> eslint .
/builds/dpniva/devops-project/service1/index.js
   47:7   warning  Expected return with your callback function  callback-return
   52:1   error    Trailing spaces not allowed                  no-trailing-spaces
   58:10  warning  'err' is defined but never used              no-unused-vars
   61:1   error    Trailing spaces not allowed                  no-trailing-spaces
   73:1   error    Trailing spaces not allowed                  no-trailing-spaces
  123:1   error    Trailing spaces not allowed                  no-trailing-spaces
  312:29  warning  'stderr' is defined but never used           no-unused-vars
  369:19  warning  'ip' is assigned a value but never used      no-unused-vars
  369:29  warning  'time' is assigned a value but never used    no-unused-vars
  405:10  warning  'err' is defined but never used              no-unused-vars
✖ 10 problems (4 errors, 6 warnings)
  4 errors and 0 warnings potentially fixable with the `--fix` option.
Cleaning up project directory and file based variables                         00:00
ERROR: Job failed: exit code 1
```

Queued: 2 seconds
Timeout: 10m (from runner)
Runner: #241 (DG9L_BHf) bruh

Commit e8609520
spam prevention

Pipeline #3139 Failed for project
lint

Related jobs
eslint-job

```
Pulling test     ... already exists
Pulling test     ... already exists
Pulling test     ... pulling fs layer
Pulling test     ... pulling fs layer
Pulling test     ... downloading (15.4%)
Pulling test     ... downloading (100.0%)
Pulling test     ... verifying checksum
Pulling test     ... download complete
Pulling test     ... downloading (100.0%)
Pulling test     ... verifying checksum
Pulling test     ... download complete
Pulling test     ... extracting (100.0%)
Pulling test     ... extracting (100.0%)
Pulling test     ... pull complete
Pulling test     ... extracting (100.0%)
Pulling test     ... extracting (100.0%)
Pulling test     ... pull complete
Pulling test     ... digest: sha256:0c8ac812016d3538df...
Pulling test     ... status: downloaded newer image fo...
Pulling test     ... done
redis is up-to-date
Recreating test_container ...
Recreating service2       ...
Recreating backend2-1     ...
Recreating backend1-1     ...
Recreating backend3-1     ...
Recreating backend3-1     ... done
Recreating backend1-1     ... done
Recreating backend2-1     ... done
Recreating nginx_frontend ...
Recreating nginx_frontend ... done
Recreating service2       ... done
Recreating test_container ... done
Cleaning up project directory and file based variables                         00:00
Job succeeded
```

Finished: 14 minutes ago
Queued: 0 seconds
Timeout: 10m (from runner)
Runner: #241 (DG9L_BHf) bruh

Commit dd81f4f4
spam prevention

Pipeline #3145 Passed for project
deploy

Related jobs
deploy-server

Image 4&5: Single success and fail jobs

5. Reflections

Main learnings and worst difficulties

- Going into this I thought this was a mundane task, but it ended up being very interesting and motivated me to investigate more, very good
- Hardest part was to figure out how connectivity between containers works in non-host environments.
- A lot was learned about version control, actions, linux, containers, permissions, ssh, external servers
- AI insights are in llm.txt in root

I couldn't use docker-compose locally, since I wanted to use docker-desktop package for convenience, and currently on Arch docker-compose conflicts with docker-desktop. However, gitlab-runner container and deployment server use docker-compose, and docker-compose was used locally after development to make sure there are no problemos.

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

- If I had even more time I would've made a **custom login service:**

The current one with nginx basic auth doesn't serve user login detection well, and sometimes causes a **race condition** between service1 load balanced containers, although this only shows itself in the run-log file occasionally as a double input from INIT->RUNNING, and doesn't effect other functionality. This is likely an issue with how the fs.watchFile updates when it looks for logged in users. Also, in general the authentication only considers the existence of a logged in user, thus doesn't support simultaneous use, which isn't ideal for anything real-world. The system is also quite slow (mocha test timeout is 7000 lol) because the sharing of state with service1 containers and using redis state management + shared container files takes time.

Amount effort (hours) used

- I spent about 90-100 hours total, and had fun through all of it, and learned a lot