End Report

Table of Contents

This report provides an overview of the system with its features, architecture, CI/CD pipeline, testing and reflections on learning and mistakes.

Feature Implementation

- Infrastructure and automation implemented in Gitlab including build, test and deploy phases
- Service1 is built with Flask, manipulating the state and remembering all state transitions
- The project follows TDD (Test-Driven Development) paradigm, designing test cases first and adjusting feature implementation
- Nginx handles all API requests and proxies them to service1

Instructions to test the system

1. Start the system:

```
git clone git@compse140.devops-gitlab.rd.tuni.fi:hckhda/exercises.git

git checkout project

docker-compose up –build
```

2. Run the system:

```
curl -X GET http://localhost:8198/state
```

3. Monitor logs:

```
curl -x GET http://localhost:8198/run-log
```

4. Run test on service1:

```
pytest service1/test.py
```

CI/CD Pipeline

Version management and branching

- Gitlab and git for version management
- Branch: project

Build tools

- Docker: Containerization for nginx, Flask and Node services
- Nginx: Reverse proxy and API Gateway

Testing

- Unit tests: using pytest, tests /state transitions, /request and /run-log
- .gitlab-ci.yml automates builds, tests and deployment
- Run test on service1:

```
pytest service1/test.py
```

Deployment

- Automated deployment via Gitlab runs the following:

1. Build: build Docker images
2. Test: run unit tests inside a container
3. Deploy: starts services using Docker Compose

Monitoring

- Docker logs used for real-time monitoring

```
Docker logs -f nginx
```

- Monitor logs:

```
curl -x GET http://localhost:8198/run-log
```

Example runs of pipeline

Failures

1. Exposure 8197:8197 for service1 in docker-compose (pipeline #3339)

- Initially exposed service1 to 0:8197 for dynamic port assignment, and deploy replicas was set to 3. So that the container could scale up to three. But then I didn't see the need of having the scalability within the system, and the requirement to expose to port 8197 for testing.

2. Docker Compose connection issues with docker:dind (pipeline #2553)

- Add docker:dint (Docker-in-Docker) to allow pipeline to run Docker commands inside container
- Install Docker Compose by having "apk add docker-compose" in Alpine Linux-based container

Success

1. Proxy API Gateway /8197 to Nginx (pipeline #3733)
- Routed API requests through Nginx to Flask
- Used correct Ngix reverse proxy configuration

2. Implement GET /run-log, update test (pipeline #3534)
- Improved test assertions
- Ensured returning valid responses

Reflections

- Setting proxy between nginx and service1: implement Nginx as API gateway provided insight into handling request routing and authentication at gateway level
- Logging state transitions improved debugging
- Setting up Gitlab CI/CD to automate builds and testing improved deployment efficiency
- Debugging failing pipelines gave insight into test failures, missing dependencies, container cleanup

- Using Docker-Compose to link nginx, service1, and service2, improved modularity and scalability
- To improve readability and usability, updated .gitlab-ci.yml, and called the configuration in necessary phases:

```
- Define a shared configuration using YAML anchors
- .default-docker-jobs: &default-docker-jobs
-   tags:
-     - macos
-   image: docker:20.10.24
-   services:
-     - docker:dind
-   variables:
-     DOCKER_HOST: tcp://docker:2375
-     DOCKER_TLS_CERTDIR: ""  # Disable TLS for Docker-in-Docker
-   before_script:
-     - echo "Installing Docker Compose in Alpine Linux-based container"
-     - apk add --no-cache docker-compose
-   after_script:
-     - docker-compose down -v
```

What could be done better

- Enhanced security: implemented authentication check in service1 to secure PUT /state endpoint

- Improve test coverage by adding more edge case coverage

- Develop a frontend UI to interact with /state, /run-log, and /request instead of relying on API calls.

Effort and time estimation

Task	**Estimated time**
Create the pipeline infrastructure using gitlab-ci	2 hours
Set up automatic build, test, deploy phases	2 hours
Write pytest for service1 (Flask)	3 hours
Write and adjust API Gateway in service1 to the unit tests	5 hours
Debug and fix issues regarding CI/CD pipeline	3 hours
Set up nginx to proxy to service1	2 hours

Documentation	2 hours
Total	16 hours