End Report for DevOps Task 1 - University

Instructions for Teaching Assistants and Examiners

The project can be executed using the following steps:

1. Clone the `project` branch:

```
git clone -b project <repository_url>
```

2. Navigate to the project folder:

```
cd <created_folder>
```

3. Build the Docker containers without cache:

```
docker-compose build --no-cache
```

4. Start the application in detached mode:

```
docker-compose up -d
```

5. Test the APIs using `curl` commands:
    - Check the state:

```
curl http://localhost:8197/state
```

- Change the state to `RUNNING`:

```
curl -X PUT http://localhost:8197/state -d "RUNNING" -H "Content-Type: text/plain"
```

- Send a request:

```
curl http://localhost:8197/request
```

- View the run log:

```
curl http://localhost:8197/run-log
```

List of Implemented Optional Features

1. API endpoints tested with an automated test script.
2. CI/CD pipeline setup for automated builds, tests, and deployments.
3. Integrated monitoring/logging features via API endpoint `/run-log`.
4. Multi-service architecture with load balancing.

System Testing Instructions

1. Use the provided `test_api.py` script to validate API functionality.
    - Run the following command to execute tests:

```
docker-compose exec service1 python3 /app/tests/test_api.py
```

  - Ensure all tests pass successfully.
2. API testing examples:
  - Check the initial state:

```
curl http://localhost:8197/state
```

  - Validate state transitions using `PUT` requests.
  - Simulate user requests using `/request` endpoint.

Platform Data

- **Hardware**: Lenovo ThinkPad E14 Gen 6
- **Operating System**: Ubuntu 22.04 LTS
- **Docker Version**: 24.0.1
- **Docker Compose Version**: 2.15.1

CI/CD Pipeline Documentation

Key Steps:

1. **Version Management**: Utilizes `git` with separate branches for development and submission (`exercise4` and `project`).
2. **Build**: Docker Compose used to build images for services (`service1`, `nginx`, `redis`).
3. **Test**: Automated test execution with `test_api.py`.
4. **Deployment**: Services launched via Docker Compose.
5. **Monitoring**: Logs available via `/run-log` endpoint.

Example Pipeline Logs:

1. **Passing Tests**:
   - All services started successfully.
   - Tests executed without errors.
   - Logs captured for reference.
2. **Failing Tests**:
   - Initial misconfiguration of ports and state handling.
   - Logs indicate failure to connect to specific endpoints.

Reflections

Key Learnings

- Effective CI/CD implementation using Docker and GitLab Runner.
- Resolving service orchestration challenges in a multi-container environment.

Challenges Faced

- Debugging container network issues.
- Ensuring API behavior matched test cases after changes.

Potential Improvements

- Incorporate more comprehensive monitoring (e.g., Prometheus, Grafana).
- Automate cleanup processes for orphaned containers.

Effort Estimate

- Approx. **50 hours**:
    - Initial setup and configuration: 15 hours
    - CI/CD pipeline implementation: 15 hours
    - Debugging and testing: 20 hours

Prepared by: Erfan Niketeghad
Date: January 22, 2025