DEVOPS Course Final Task Report

Instructions for the teaching assistant:

Optional features Implemented:

1. Logging feature
2. Static Analyser

How to test the system:

Each component in the application generates logs under logs/ folder as html page. The file name follows the pattern <Start time>_<Service name>_logs.html. I.e the http_server's log will be stored in *./logs/19_01_2023_22_25_12_httpserver_logs.html.* This generated log also has the capacity to filter the logs in the individual html page.

The Static analyser runs as a stage in the gitlab-ci pipeline. Here we perform flake8 and pylint static analyzer The static analyser runs first, despite the result of the static analysis the code passes to the next stage.

CI/CD Pipeline:

1. Version Management:

The version management tools used are gitlab and github. We use two remote repositories. The gitlab along with gitlab-runner is installed in the local machine while github resides in github.com. The "main" branch is left untouched as per the course instruction. All the changes are updated in the "project" branch.

The project has been configured to push to two remote repositories by creating a new git remote called "all" and setting two remote urls for pushing using the command. The gitlab repository uses http authentication with access token while github uses ssh private key authentication.

```
git remote add all git@github.com:praburocking/devopsfinalprojeect.git
git remote set-url --add --push all
http://localhost:8080/gitlab-instance-c2607042/devopsfinalproject.git
git remote set-url --add --push all git@github.com:praburocking/devopsfinalprojeect.git
git push all project
```

2. Static analysis

The programming error and python coding standard (PEP8) are analysed using the static analyzer. The static analyzers used are Flake8 and Pylint. In the Gitlab CI pipeline the static analysing jobs are configured in such a way(allow_failure: true) that even the code doesn't have the required standard the code will pass through the pipeline, This "static_analysis" step is in the pipeline to generate the reports and it will not block the code in the CI pipeline.

3. Build:

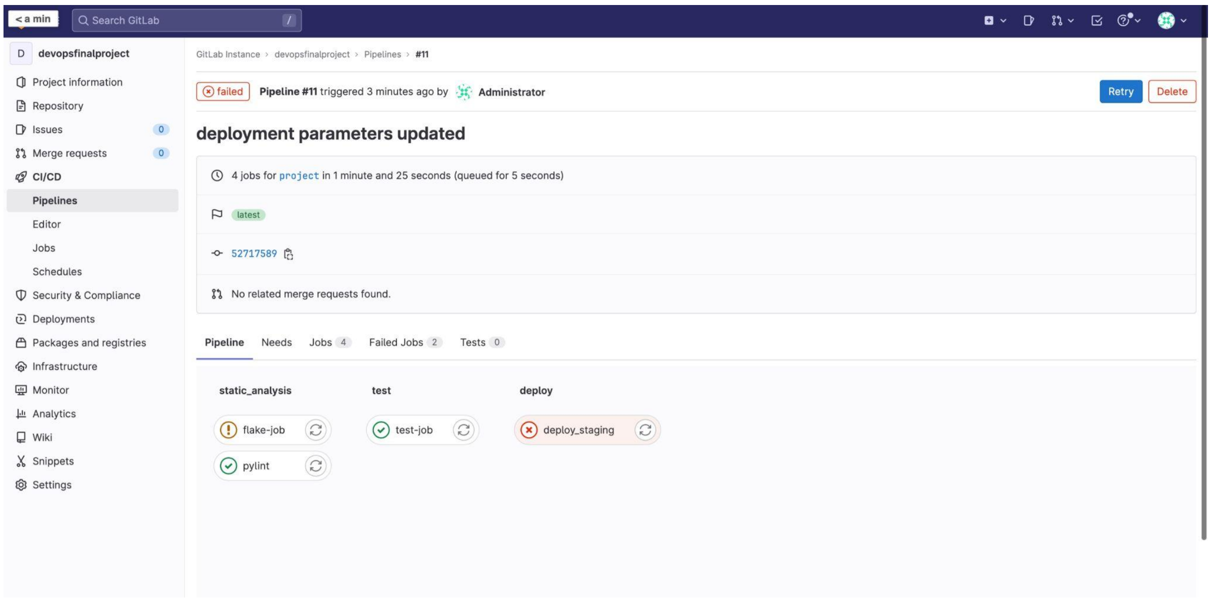
Since the application is built with python, there isn't any special building tool needed and therefore no build stage is needed in the CI/CD pipeline.

4. Tests:
   The test has been implemented for only Http-server component. The test is implemented by using pytest framework with pytest-mock plugins for mocking the file read/write operations.It tests all the endpoint's success and fail cases.

5. Deploy:
   The application is deployed on the DooD(Docker on the Docker) container. With teracy/ubuntu as the base, the Dockerfile for the deployment container can be found in the '/deployment' directory in the repository. Once the pipeline comes to the deployment stage, the gitlab runner, ssh into the running docker container called 'deployment' which is in the same network as local gitlab,gitlab-runner using the password and clone the github repository by using the access token. Once the repository is cloned then we run 'docker compose up' and exit the pipeline.

6. Log and Monitor:
   For monitoring the services, we have a separate service called 'log_monitor' which can be accessed in the port '9999'. When we access the log service in the browser, it will show a list of html files with names like <DATE_TIME>_<SERVICE_NAME>_logs.html. Through this we can check the live log and also the logs of the older runs. These logs files are persisted in ./logs directory in the repository, so if we want to clear the logs of the older runs we can simply delete those files in the ./logs directory. These html logs are created with the 'Pylog2html' package which provides better UI and also some basic functions like filtering and searching.

Sample CI/CD Pipeline:



failed Pipeline #11 triggered 3 minutes ago by Administrator

deployment parameters updated

4 jobs for project in 1 minute and 25 seconds (queued for 5 seconds)

latest

52717589

No related merge requests found.

static_analysis: flake-job, pylint
test: test-job
deploy: deploy_staging

Test passed Log from gitlab CI/CD pipeline:

```
$ export ENV=TEST
$ echo "Running tests"
Running tests
$ cd tests/
$ pytest .
============================== test session starts
==============================
platform linux -- Python 3.11.1, pytest-7.2.1, pluggy-1.0.0
rootdir: /builds/gitlab-instance-c2607042/devopsfinalproject/tests
plugins: mock-3.10.0, anyio-3.6.2
collected 6 items
test_http_server.py ......                                    [100%]
=============================== warnings summary
===============================
test_http_server.py::test_simple_update_state
test_http_server.py::test_update_same_state
test_http_server.py::test_update_same_state
test_http_server.py::test_update_undefined_state

/builds/gitlab-instance-c2607042/devopsfinalproject/venv/lib/python3.11/site-packages/httpx/
_content.py:205: DeprecationWarning: Use 'content=<...>' to upload raw bytes/text content.
  warnings.warn(message, DeprecationWarning)
-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
======================== 6 passed, 4 warnings in 0.18s
========================
$ echo "test completed"
test completed
Job succeeded
```

Test failed Log from gitlab CI/CD pipeline:

```
Running tests
$ cd tests/
$ pytest .
============================== test session starts
==============================
platform linux -- Python 3.11.1, pytest-7.2.1, pluggy-1.0.0
rootdir: /builds/gitlab-instance-c2607042/devopsfinalproject/tests
plugins: mock-3.10.0, anyio-3.6.2
collected 6 items
test_http_server.py F.....                                    [100%]
=================================== FAILURES
===================================
______________________________ test_read_message
_______________________________
mocker = <pytest_mock.plugin.MockerFixture object at 0xffff9b715550>
    def test_read_message(mocker) :
        output_res=["2023-01-22 00:18:50.567344 1 MSG_1 to compse140.o",
                    "2023-01-22 00:18:50.567431 2 MSG_1 to compse140.i",
                    "2023-01-22 00:18:50.567512 3 MSG_2 to compse140.o",
                    "2023-01-22 00:18:50.567581 4 MSG_2 to compse140.i",
                    "2023-01-22 00:18:51.670902 5 MSG_3 to compse140.o",
                    "2023-01-22 00:18:51.671679 6 MSG_3 to compse140.i"]
        mocked_etc_release_data = mocker.mock_open(read_data="\n".join(output_res))
        builtin_open = "builtins.open"
        mocker.patch(builtin_open, mocked_etc_release_data)
        response = client.get("/message")
>       assert response.status_code == 201
E       assert 200 == 201
E        +  where 200 = <Response [200 OK]>.status_code
test_http_server.py:43: AssertionError
----------------------------- Captured stderr call -----------------------------
2023-01-29 22:51:42,534 HTTP SERVER  INFO     expected file path ../logs/temp_file.txt
2023-01-29 22:51:42,534 HTTP SERVER  INFO     GET request received for /message
=============================== warnings summary
===============================
test_http_server.py::test_simple_update_state
test_http_server.py::test_update_same_state
test_http_server.py::test_update_same_state
test_http_server.py::test_update_undefined_state

/builds/gitlab-instance-c2607042/devopsfinalproject/venv/lib/python3.11/site-packages/httpx/
_content.py:205: DeprecationWarning: Use 'content=<...>' to upload raw bytes/text content.
    warnings.warn(message, DeprecationWarning)
-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
=========================== short test summary info
============================
FAILED test_http_server.py::test_read_message - assert 200 == 201
 +  where 200 = <Response [200 OK]>.status_code
=================== 1 failed, 5 passed, 4 warnings in 0.30s ====================
ERROR: Job failed: exit code 1
```

Reflections:

The initial part of setting up the local gitlab and gitlab runner was a difficult part; The other difficult part was to create a deployment space to deploy the application using CI/CD. Initially I wanted to use an alpine Docker container but then due to some difficulty in using it I switched to teracy/ubuntu.

As for the logic of the application, I used two rabbitMQ exchanges one for sending the control message, i.e when the state of the application changes an application state message will be pushed to the control topic from httpserver component which will be consumed by all the other components and act according to the state in the message. The one difficult use case to achieve was to pause the message in the middle of publishing it from Orig component, this was achieved by having an external file in which the httpserver component will write the last updated state and orig component will check for the last updated state in this file every time it publishes a message and if the last state is “PAUSED” then it pauses the message. The other difficult case to achieve was shutting down the services, for this case I don't want to implement a ssh service which can ssh to the host and perform “docker-compose down” as it would be a security threat. For this case I mostly tried to shutdown the services programmatically.

The Things I could have done differently are implement a better logging system, as currently it is just a bunch of web pages.