COMP.SE.140, Final Project

Name: **Ali Mehraj**

Student Number: **50508879**

1. Instructions for the teaching assistant

To run the application:

Clone the repo: git clone -b project https://course-gitlab.tuni.fi/compse140-2022-2023/ntalme.git

Change directory to project folder and run: docker-compose build –-no-cache

Run: docker-compose up –d

Wait for about 30 seconds. Services will be up and running by that time.

Implemented optional features

- *GET /node-statistic*
- *GET /queue-statistic*
- *Testing of individual components*

Instructions for examiner to test the system.

- The system can be tested using curl commands.

  For example: curl localhost:8083**/state** -X **PUT** -d ”PAUSED” -**H** ”Content-Type:text/plain” -**H** ”Accept:text/plain”, this will pause the messaging.
  curl localhost:8083**/state** -X **PUT** -d ”RUNNING” -**H** ”Content-Type:text/plain” -**H** ”Accept: text/plain”, this will resume the messaging.

- To test the node-statistic: curl localhost:8083**/node-statistic** -X **GET** -**H** ”Content-Type:text/plain” -**H** ”Accept: text/plain”, this will fetch the node statistics.
- To test the node-statistic: curl localhost:8083**/queue-statistic** -X **GET** -**H** ”Content-Type:text/plain” -**H** ”Accept: text/plain”, this will fetch the queue statistics.

- To test the system locally using the created tests, change directory to 'tests' folder. Run 'npm install' and then run 'npm run test'. This will run the tests locally that are also run in the pipeline. (This will also run the node-statistic and queue-statistic tests)

2. Description of the CI/CD pipeline

The pipeline is created using gitlab-ci.yml file. It was tested both locally and on gitlab.com pipeline. The gitlab.com link: https://gitlab.com/pingusta/devops-project/-/pipelines

Both in course-gitlab and gitlab.com the application is in 'project' branch.

For building the application, the choice of programming language was Node.js. Express.js library was used for the ease of use of routing. Axios was used for fetching data using APIs. For message broker, RabbitMQ was used. For containerization, Docker and Docker-compose were used.

Mocha and Chai were libraries used for testing. These were chosen for ease of use, and I was familiar with the libraries as well. Test cases are mostly based on unit testing and some integration testing. The tests can be found in the tests folder. The test cases are:

- Get running state (Check if the system is running and messages are being sent)
- Pause ORIG service (Pause the service)
- Get paused state (Check the state during the pause)
- Resume ORIG service (Resume the service again)
- Get running state after resume (Check the state after resuming)
- Get messages log (Get message logs for the application)
- Get RabbitMQ node statistic (Get node statistics, top 5 stats)
- Get RabbitMQ queue statistic (Get queue statistic for each queue)
- Get messages (Get all the messages what were written in the file)

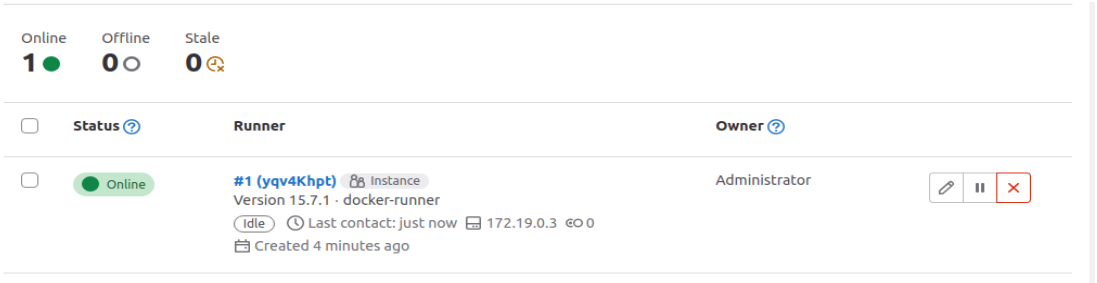
The Gitlab-ci.yml file was designed in such a way that the whole application is built and then the tests are run separately.

.gitlab-ci.yml

```yaml
default:
  image: docker/compose:latest

DockerSetup:

  variables:
    DOCKER_HOST: tcp://docker:2375/
    DOCKER_DRIVER: overlay2

  image:
    name: docker/compose:latest

  services:
    - docker:dind
  before_script:
    - docker version
    - docker-compose version
    - apk add --update nodejs npm

  stage: build
  script:
    - docker-compose down
    - docker-compose build
    - docker-compose up -d
    - cd tests/ && npm install
    - npm run test-docker
```

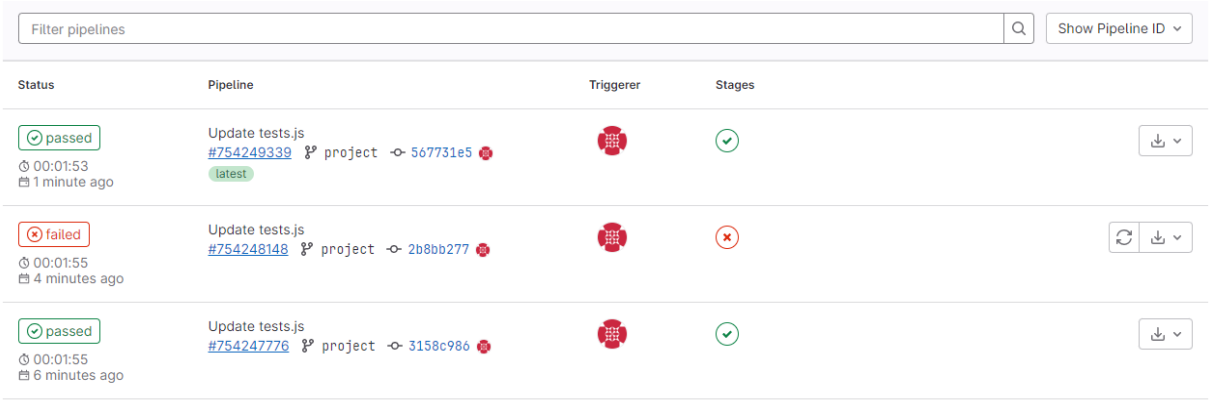
As previously mentioned, the pipeline was tested both locally and on gitlab.com.

Local runner:



The examiner can visit https://gitlab.com/pingusta/devops-project/-/pipelines and see the pipeline in action as well.

3. Example runs of the pipeline



Successful run of pipeline:

```
Pulling rabbitmq (rabbitmq:3-management)...
3-management: Pulling from library/rabbitmq
Digest: sha256:c8f3d8efed804fb1b93d046ef9bc65f124eda1517f3047fbe813dd9512e1682e
Status: Downloaded newer image for rabbitmq:3-management
Creating devops-project_rabbitmq_1 ...
Creating devops-project_httpserv_1 ...
Creating devops-project_imed_1     ...
Creating devops-project_obse_1     ...
Creating devops-project_apigw_1    ...
Creating devops-project_orig_1     ...
$ cd tests/ && npm install
npm WARN read-shrinkwrap This version of npm is compatible with lockfileVersion@1, but package-lock.json was generated for lockfileVersion@2. I'll try to do my best with it!
npm WARN tests@1.0.0 No description
npm WARN tests@1.0.0 No repository field.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.3.2 (node_modules/fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})
added 95 packages from 92 contributors and audited 96 packages in 7.456s
21 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
$ npm run test-docker
> tests@1.0.0 test-docker /builds/pingusta/devops-project/tests
> mocha tests.js
  APIs
    ✓ Get running state (59ms)
    ✓ Pause ORIG service (53ms)
    ✓ Get paused state
    ✓ Resume ORIG service
    ✓ Get running state after resume
    ✓ Get messages log
    ✓ Get rabbitmq node statistic
    ✓ Get rabbitmq queue statistic
  8 passing (221ms)
Cleaning up project directory and file based variables
00:02
Job succeeded
```

Failed run of pipeline:

```
$ npm run test-docker
> tests@1.0.0 test-docker /builds/pingusta/devops-project/tests
> mocha tests.js
  APIs
RUNNING
    ✓ Get running state (63ms)
PAUSED
    ✓ Pause ORIG service (52ms)
PAUSED
    ✓ Get paused state
RUNNING
    ✓ Resume ORIG service
RUNNING
    ✓ Get running state
2023-01-21T19:24:51.298Z INIT
2023-01-21T19:24:51.298Z RUNNING
    ✓ Get messages log
    ✓ Get rabbitmq node statistic
    ✓ Get rabbitmq queue statistic
    1) Get messages
  8 passing (242ms)
  1 failing
  1) APIs
       Get messages:
     Error: socket hang up
      at Function.AxiosError.from (file:///builds/pingusta/devops-project/tests/node_modules/axios/lib/core/AxiosError.js:89:14)
      at RedirectableRequest.handleRequestError (file:///builds/pingusta/devops-project/tests/node_modules/axios/lib/adapters/http.js:533:25)
      at ClientRequest.eventHandlers.<computed> (node_modules/follow-redirects/index.js:14:24)
      at Socket.socketOnEnd (_http_client.js:458:9)
      at endReadableNT (_stream_readable.js:1241:12)
      at processTicksAndRejections (internal/process/task_queues.js:84:21)
npm ERR! code ELIFECYCLE
npm ERR! errno 1
npm ERR! tests@1.0.0 test-docker: `mocha tests.js`
npm ERR! Exit status 1
npm ERR!
npm ERR! Failed at the tests@1.0.0 test-docker script.
```

4. Reflections

Main learnings and worst difficulties

I think setting up the GitLab and GitLab-runner locally was the hardest part of this project. I spent around 3 days trying to go back and forth searching for relevant content to figure out how to do it. There is also vague and scattered information on the web about this topic. I had to reset 2 Linux virtual machines during the whole setup where I lost a lot of time as well. But on a positive note, I was able to learn how GitLab setup works. I learned a lot about docker and docker-compose as well. I did not know that node-statistics and queue-statistics were available for RabbitMQ and could be fetched using APIs which was also very insightful. Overall, I learned a lot about the new technologies that I was not familiar with and setting up each tool turned out to be an obstacle with GitLab runner being the hardest obstacle of them all.

The first improvement of my solution would be to keep only the APIGW service as the server. In my solution, I created another server for ORIG service to process the API requests, which serves the purpose but can be improved. I tried to go that route, but I was unable to make the

application work in that way. One other thing is handling of the shutdown of containers. My application simply halts operation and does not allow any API request but truly does not stop the containers. From my searches, I found out that that can be done with docker.sock but I was unable to implement that after too many failed attempts. There are most likely other options available to implement which would have made the operation better. Another concept that would have been particularly useful is docker in docker. Where the APIGW is the main docker container that has the other containers in it. It would have been easier to control the other containers that way. Similarly, to the previous improvement I was unable to do it that way.

Amount effort (hours) used

Around 65-70 hours