CSC 519 Project Report

Jae Jimmy Wong
Hsueh-Yang Yu
Li-Ling Ku

1 Introduction

The project is to design an efficient and reliable DevOps infrastructure that can help the future development of the application of the coffee project. Our project was built on a 3-tier layered architecture using HTML, NodeJS, and PostgreSQL. To be more specific, the staging and production environment was set up on two separate virtual machines via the Ansible playbook. We also use several trustworthy tools to help us develop the whole pipelines such as Docker, EsLint, Selenium, etc. Our infrastructure includes several useful features including linting, unit testing, end-to-end testing, log and metric collection, and performance monitoring. We will illustrate each feature in the section 5.2.

2 Problem Statement

Developing an e-commerce application involves constantly adding new features. However, manual testing is time-consuming and prone to errors. In addition, the testing process is not visible to other collaborators while executed on a local machine. There may also be differences between a local machine and the production environment, leading to undetected errors slipping through. Therefore, testing, logging, and performance monitoring are necessary to ensure that new changes do not affect the other parts of the pipeline

Our pipeline utilizes Ansible to provision the same environment on both staging and production environments. When a developer creates a pull request for new features, GitHub Action automatically runs linting and unit tests on the code to reduce the time needed for manual testing. Furthermore, when the release engineer creates a release branch, GitHub Action will trigger end-to-end testing and show the status on the GitHub repository, with its result being accessible to all repository collaborators. Our newly added log and metric collection system which is built on Fluentd can help improve the efficiency of analysis and debugging. The error tracking and performance monitoring system can also help track the internal server errors or timeout.

3 Motivation

The motivation of our project is that we want to solve several severe problems of the coffee project such as time-consuming and error-prone manual testing, inconsistent environment,

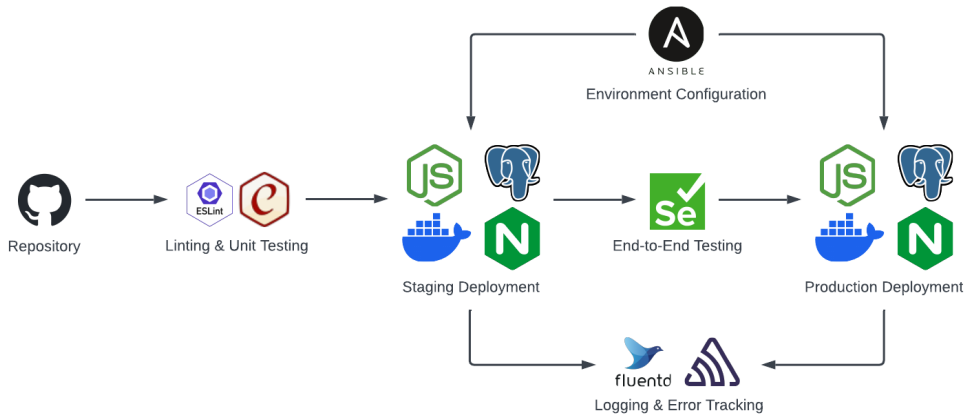
non-transparency of testing status, etc. If those problems are not solved properly, the application of the coffee project will be crushed frequently as the number of new features will increase explosively in the future. To prevent the development of the coffee project from being chaos in the future, we decided to design several automated DevOps pipelines for testing before producing any new feature. Furthermore, our design of a log and metric collection system and performance monitoring system can improve the reliability of the application since developers can detect potential flaws at an early stage. We believe that adding our system to the original coffee project can make the application meet the standard of a modern infrastructure.

4 Summary of Accomplishments

- Set up staging and production environment by using Ansible
- Rewrite the coffee project to separate the application layer and data layer.
- Add branch protection rule to prevent `main` and `develop` branch from being deleted
- Add branch protection rule to make sure PR is reviewed by the development leader before merging
- Add GitHub Action task to automatically run linting and unit testing
- Add GitHub Action task to build docker images for deployments
- Add GitHub Action task to create issues to notify developers when the e2e tests failed
- Add GitHub Action task to create tag while merging release branches to `main`
- Add GitHub Action task to do end-to-end testing with Selenium
- Use GitHub Action secrets to store database passwords and other confidential data
- Add timeout to GitHub Action workflow to prevent worker from stucking
- Build Log and metric collection system by using Fluentd
- Apply error tracking and performance monitoring with Sentry

5 Technical Approach

5.1 Pipeline Diagram



5.2 Description of Pipeline

- Environment Configuration

  We reserved two Ubuntu 22.04 machines on the VCL platform as our staging machine and production machine and used Ansible to automatically apply security patches, install required packages, and start the background services including Docker, Nginx, and PostgreSQL. The staging machine's environment is set to be identical to the production machine's to mimic the real-world situation.

- Repository

  We used Git to manage our code and created a remote repository on GitHub. Both the source code of the coffee project and the configuration files are stored inside the single repository to get the benefits of version control.

- Linting & Unit Testing

  We created a GitHub Action workflow that executes linting and unit testing when a pull request toward the `develop` branch is created. The linting is backed by ESLint which analyzes if the code contains any syntax errors or does not follow the common code style. The unit testing runs all the testing scripts with Mocha. The testing assertion that is written with Chai includes all of the POST and GET requests to ensure our each function in our Node.js application is working as intended. Completing the linting and unit testing is required before merging to ensure that codes on the `develop` branch is properly tested.

- Staging Deployment

  We created a GitHub Action workflow that automatically deploys the coffee project into the staging environment when a new `release` branch is created or a new commit is added to a `release` branch. While triggered, a GitHub Action runner will download

the repository into the staging environment, build the Docker image of the coffee project, and create a Docker container with the image. The staging version of the coffee project runs inside the container and uses the staging PostgreSQL server to store its data.

- End-to-End Testing

  Description: We use Selenium to simulate real-world user scenarios on a headless Firefox browser. For example, it clicks every button on our page to order coffees, and checks if any button is not functioning and if the database is updated or not. The Selenium test is triggered by user upon creating a new `release` branch or adding new commit on the `release` branch, and runs on the staging machine.

- Production Deployment

  We created a GitHub Action workflow that automatically triggers the deployment of the coffee project to the production environment when the pull request from `release` branch to `main` branch is merged. While triggered, a GitHub Action runner will download the repository into the production environment, build the Docker image of the coffee project, and create a Docker container with the image. The production version of the coffee project runs inside the container and uses the production PostgreSQL server to store its data.

- Logging & Error Tracking

  We use Fluentd to collect server logs and Sentry to track errors and performance metrics of the staging machine and the production machine. Fluentd is a log-collecting daemon that acts as the unified collection service in our application. We hosted the service in the staging machine and connected each instance of the Node.js application to it. Sentry automatically generates an organized dashboard and shows some critical metrics, such as transactions per minute, failure rate, and average duration of each SQL query.

6 Retrospectives

- **What worked?**

  1. The repository has a continuous integration pipeline that ensures the quality and integrity of codes on the specified branches.
  2. The repository has a continuous delivery pipeline that deploys the application to the staging and production server while reaching deployment and production branches respectively.
  3. The unit tests can run with their exclusive databases by using GitHub Action service container.
  4. The Node.js environment of the coffee project can be reproduced with the `package-lock.json` file.

5. The code owner feature of GitHub allows us to define the development leader as the reviewer of each PR.

- **What did not work?**

1. Some marketplace features in GitHub cannot be used.
2. Yarn failed to install packages for coffee-project.
3. Initially, failed all unit testing with error message (TypeError: app.address is not a function) since some Node.js packages require the application to be written as ES module. However, the format is incompatible with the original CommonJS testing script.
4. `gh` command which was used to create new issue did not work since it needed two necessary environment variables, GITHUB_ENTERPRISE_TOKEN and GH_HOST.
5. VCL has a built-in firewall that prevents Fluentd service from being connected.

- **What we would do differently?**

1. We will do some research on the NCSU Github environment before implementing new Github Actions.
2. We will choose to use more stable tool, e.g. npm, rather than newer tool with more features, e.g. Yarn.
3. We will plan how to access database for automated testing on the workflow.

7 Contribution

- Li-Ling Ku (lku): Develop Ansible playbooks (commit), add E2E testing (Selenium) (commit), set up performance monitor (Sentry) (commit), write project documents, edit demo video
- Jae Jimmy Wong (jwong23): migrate Node.js application to PostgreSQL (commit), add Dockerfile (commit), add staging deployment workflow (commit), add database initialization script (commit), add Fluentd logging middleware (commit), add production deployment workflow (commit)
- Hsueh-Yang Yu (hyu25): Add unit testing (Mocha) (commit), add linting (ESLint) (commit), add tag when merging release to main (commit), create issues to notify developers when E2E test fails (commit), set time-out for each workflow (commit) and write project documents