CSC 519 Project Proposal

Jae Jimmy Wong
Hsueh-Yang Yu
Li-Ling Ku

1 Problem Statement

Developing an e-commerce application involves constantly adding new features, so testing is necessary to ensure that new changes do not break the other parts. However, manual testing is time-consuming and prone to errors. Besides, the testing process is not visible to other collaborators while executed on a local machine. In addition, there may be differences between a local machine and the production environment, leading to undetected errors slipping through.

Our pipeline utilizes Ansible to provision the same environment on both staging and production environments. While 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. While 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.

2 Use Cases

2.1 Add New Feature

Precondition

- The development branch exists.
- A self-hosted Github Action runner is provisioned.

Main Flow

Developers create a feature branch from the development branch. Push commits to the feature branch [S1]. Create pull requests to the development branch [S2]. Pull Requests have been approved by development leaders [S3].

Sub Flows

[S1] Push commits with new features from the local repository to the feature branch.

[S2] Developers provide PR messages and request appropriate reviewers (development leaders must be included).

[S3] PRs approved by development leaders.

Alternative Flows

[E1] Add fixing commits to the feature branch.

2.2 Create New Release

Precondition

- Features for the new release are merged into the development branch.
- A self-hosted GitHub Action runner is provisioned.
- The staging server is provisioned.

Main Flow

Release Engineer create a release branch[S1], which triggers automatic end-to-end testing[S2]. If the testing fails, the development team is notified[E1], and then fixes will be merged into the release branch with pull requests[E2].

Sub Flows

[S1] The release branch is created from the development branch.

[S2] GitHub Action deploys the application to the staging server and triggers the E2E testing.

Alternative Flows

[E1] A GitHub issue is created to notify the development team.

[E2] Pull requests will also trigger deployments and tests in [S2].

2.3 Publish New Release

Precondition

- The release branch has passed all of the tests and is ready to be published.
- The main branch exists.

- The production server is provisioned.

Main Flow

The release engineer will create a pull request to merge the release branch to the main branch[S1][S2].

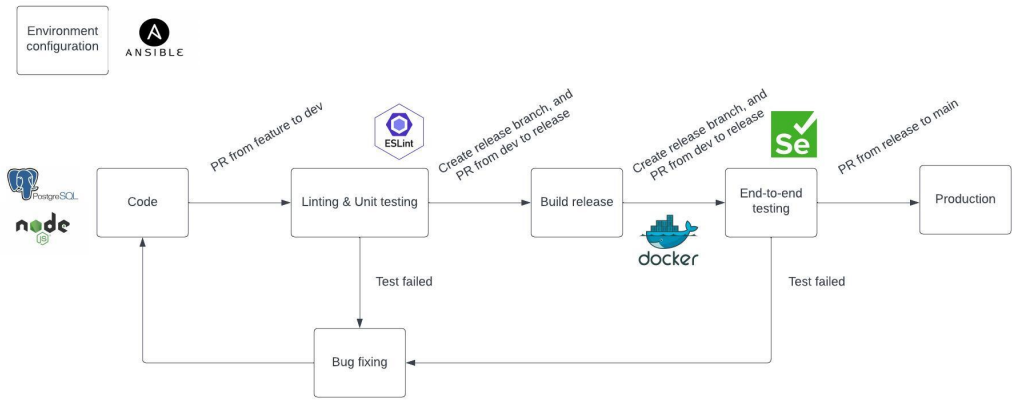
Sub Flows

[S1] GitHub action will deploy new code to the production environment.

[S2] When the merge is successfully done, the release tag will be automatically added.

3 Pipeline Design

3.1 Pipeline Diagram



3.2 Architecture Components

- Environment configuration
  - Description: We will provision two remote machines which can be connected through SSH as our staging and production environment, and use these remote machines as the inventory for Ansible. We will create an Ansible playbook that will apply security patches to our inventories, and install required dependencies such as PostgreSQL database and Docker. The Ansible playbook will be executed from the release engineer's machine (the control node) to configure the inventories.
  - Guideline:
    * PostgreSQL password will be stored in a YAML file that only the release engineer can access.

- Code:
  - Description: Codes related to developing new features or modifying existing features would be completed in the “Code” component. All actions in this component would be done in the feature branch.
  - Guideline:
    * Each developer would create a feature branch from the development branch on GitHub.
    * Each developer could git clone the repository of the “Coffee-project” on the feature branch.
    * Each developer would develop new features or modify existing features of the “Coffee-project” via Node.js on their local repository.
    * Each developer would create multiple tables and insert related data on the PostgreSQL database. This step would also be processed on the local repository.
    * After completing the development of a feature, each developer would commit their codes and push them back to the feature branch on GitHub.
- Linting & Unit testing:
  - Description: The “Linting & Unit testing” component would be automatically triggered by the GitHub action “pull-request”. In this component, ESLint would first check some common problems in your JS code such as syntax errors, formatting issues, or code style violations. Subsequently, unit testing for JS codes would be executed to check the correctness of each function in your code.
  - Guideline:
    * When each developer sends a PR to the development branch on GitHub, the GitHub action “pull-request” would trigger both linting and unit testing to run automatically.
    * Linting would be run on ESLint, and unit testing would be run on Chai.
    * If there are some issues reported after linting and unit testing, the current step would be pushed to the “Bug fixing” component.
- Build Release:
  - Description: The “Build release” component would be automatically triggered by the GitHub Action event “create”. In this component, code will be built into a Docker image.
  - Guideline:
    * The result Docker image will be uploaded to the staging server and used to create a container.

    - If a previous release is still running on the staging server, it will be removed before running the latest release.

- End-to-end Testing:
  - Description: The "End-to-end testing" component would be automatically triggered by the github action "pull-request". In this component, Selenium WebDriver would automate the cross-browser and cross-platform testing of web applications based on your JS code.
  - Guideline:
    - Developers should create a Selenium script for the new feature and put it in the project's testing folder.

- Bug Fixing:
  - Description: Either the "Linting & Unit testing" component or the "End-to-end testing" component fail, the current step would be directed to the "Bug fixing" component.
  - Guideline:
    - When there are some issues reported after linting and unit testing, developers who own the specific features would be notified to fix the issues. After the issues have been solved, developers have to re-send PRs to the development branch on github.
    - When there are some issues reported after end-to-end testing, developers who own the specific features would be notified to fix the issues. Developers would need to create a new feature branch on github to solve these issues.

- Production
  - Description: When the code has passed the end-to-end testing in staging environment, the release engineer will deploy new code to the production environment when needed.
  - Guideline:
    - The release engineer will create a pull request to merge from the release to main branch.
    - The merge operation will trigger GitHub action to deploy the new code to the production environment.
    - When the merge is successfully done, it will trigger GitHub action to create a tag with the release branch's name.

3.3 Pipeline Feature List

Core features that we will implement first:

- GitHub action (Linting and unit testing)
- Branch protection rule (main and the dev branch can't be deleted)
- Branch protection rule (PRs have to be reviewed by the development leader)
- Ansible (set up staging and production environment)
- GitHub action (create docker image)
- End-to-end testing (Selenium)
- Add database to the coffee project
- Use GitHub action secret to save database password

Additional features to our pipeline:

- GitHub action (add tag when merging release to main)
- GitHub action (create an issue to notify developers that the E2E test has failed)
- Github action (set timeout for workflows)