Project Report – Group 9

Giovanni Fincato de Loureiro (studentnumber 4926854
PUT YOURSELVES AS AUTHORS HERE

1 ABSTRACT

This report talks about the comprehensive process of developing a robust machine learning pipeline application. The project uses modern DevOps practices so that the CI/CD pipelines are seamless. We integrated Docker, Ansible, Poetry, etc. for an efficient and consistent deployment of the application. Each service is designed in a way that makes modularity, scalability, and maintainability better. Critical parts of the process (like building, deploying, testing) are automated to minimise human error during the process. Along with the documentation of the setup, this report reflects on the current state of the application as well as the strategies used for test coverage. The report also proposes enhancements to address the shortcomings of these strategies. These proposed improvements are meant to improve the quality and performance of the entire application as a whole.

2 INTRODUCTION

The purpose of this report is to give a comprehensive overview of the development process, architectural design, and operational strategies used to create our multi-service software project. This project has multiple interconnected services: app-service, app-frontend, model-service, lib-ml, etc., each playing a individually important role in delivering a cohesive and functional application.

Modern software development robust coding practices and also efficient and reliable deployment mechanisms. For these requirements to be met, our project integrated contemporary DevOps tools and practices, such as continuous integration and deployment (CI/CD), Docker for containerization, and Ansible for configuration management. These tools collectively ensure that the software can be developed, tested, and deployed easily across various environments.

The report begins by outlining the Release Pipeline Documentation. It then transitions into Deployment Documentation to document the final deployment of our application. We then reflect on the current state of our project and identify the points that we find the most critical or error prone. We then document how our pipeline is set up and the decisions we had to make to configure our project.

Furthermore, the report addresses the limitations of the current testing strategy, identifying areas that are prone to errors or insufficient coverage. Suggestions for enhancing the testing framework are provided, emphasizing the need for broader test coverage and the inclusion of performance testing.

This report aims to serve as a detailed guide for understanding the development and deployment processes of the project, so that we can offer insights into the decisions made and challenges we encountered. It also provides recommendations for future improvements in the project, ensuring that the project can improve to meet higher standards of quality and reliability. Our report aspires to contribute valuable knowledge to the field of software development, particularly in the realms of DevOps and continuous delivery.

3 PIPELINE

3.1 First pipeline (Software package release)

Release pipeline for the lib-ml package has several stages to ensure that the code is tested, built, and published to the PyPI. The first step is performing Code Checkout where the latest code is gotten from the latest version in the version control system. This is followed by setting up the environment, preparing it for subsequent operations by installing the dependencies. Once the environment is set the pipeline does the code quality checks and the static code analysis and linting is done using flake8 and pylint to ensure it is up to standard. Unit testing is then done using pytest to check whether the code functions as expected.

Passing all tests, the build package creates the actual Python package with setuptools, and the pipeline lastly publishes to PyPI with twine.

3.2 Second pipeline (Releasing container image)

The second pipeline builds and releases a docker dontainer image for model-service. It starts off similar to the first pipeline, with code checkout and setting up the environment. After the environment is set up, the code quality checks are carried out to ensure coding standards (flake8, pylint). Unit testing is lastly done.

The docker image is then built, creating an image with the dockerfile. Once the image is built, the docker image is pushed to a container registry. The pipeline can also have a deploy to kubernetes step where the docker image is deployed to a Kubernetes cluster. Data flow in this pipeline includes source code, dockerfile, configurations, liting reports, test reports, and docker image.

4 SHORTCOMING

An identified shortcoming is in the model-service project, in the process of managing dependencies and configurations across the different environments. This issue can be seen in the Dockerfile and the manual setup for installing dependencies. The absence of a streamlined and automated approach can lead to inconsistencies and environment-specific issues (as well as more work on the maintenance side) this problem is worse when deploying the application across multiple environments.

The proposed refactoring to address this would be to further the implementation of IaC tools like Ansible (or Terraform alternatively) along with a CI/CD pipeline.

5 PIPELINE SETUP DOCUMENTATION

The (CI/CD) pipeline for the project (including the app-service, app-frontend, model-service, etc.) automates the process of building and testing the components, as well as deploying them. It ensures consistency, reliability, and efficiency across the whole application. The main components of the pipeline are the Continuous Integraion for code verification and Continuous Deployment to automate deployment for all the different services involved.

We use Docker for creating the consistent and isolated environments, as well as ansible for managing configurations and Poetry is responsible for managing the python dependencies.

The design of he pipeline is to maintain modularity, and has the distinct stages throughout so that each service is responsible for particular tasks (including building, testing, deploying, cleaning). The modularity makes it easier to maintain the application as well as providing scalability. Ansible is used for automated configuration management with the Ansible playbooks, so there are reduced manual errors and consistency is better. Docker makes the apps run in consistent environment throughout all the different pipeline stages.

The steps of the pipeline itself beings with code checkout and the code is pulled from the repository (for all the services). Docker images are created using the Dockerfiles in the build for each service. The images have the application code and all dependencies. The test stage runs unit and integration tests inside Docker containers to check code correctness. The configure environment stage uses Ansible to set up all the dependencies.

6 TESTING STRATEGY REFLECTION

The current testing strategy for the entire project includes the unit tests and integration tests executed during the CI stage of the pipeline for all the services. Unit tests verify individual components while in isolation while the integration tests are responsible for ensuring the various components work together properly. End-to-end tests are performed on the application deployed to see the functionality in a production environment. The limitations of the strategy include insufficient coverage and the lack of automated performance testing.

The testing strategy should be improved to overcome these issues by expanding the test coverage and automating the performance testing. Expanding the test coverage is generally laborious but straightforward, we increase the number of unit test ot cover more edge cases and more different scenarios, and use tools to test the code coverage and identify which parts of the codebase remain untested. Automating the performance tests using a tool like JMeter or Locust would also help find the bottlenecks in application performance.

Baseline metrics for code coverage and performance should be collected to test the effectiveness of the improvements (i.e. compare before/after the improvements).

7 CITATIONS

You can cite papers, e.g., [1]. To make the references appear, make sure to compile the latex sources, then bibtex, and then latex twice.

REFERENCES

[1] First Author, Second Author, and Third Author. 2018. An Examplary Paper For The References. In *International Conference on Silly Walks.*