CSC-519 Project Report

Andrew Shon (ashon), Isaac Taylor (itaylor), Corey Capooci (cvcapooc)

December 3rd, 2023

Introduction

Today, the term *DevOps* is used frequently in the software engineering community to refer to the combination of two originally separate practices: Development and Operations. This report will describe a real-world DevOps pipeline implementation created by Andrew Shon, Corey Capooci, and Isaac Taylor. We will use GitHub as our code repository and the primary driver for our pipeline. To begin our discussion, we will provide a definition of what the term DevOps means. In this report, DevOps is a set of software practices used to create and maintain software in an efficient way through the course of its lifecycle.

Problem Statement

When creating software, implementing a functioning product is only a minor step in creating production-ready tools and applications. Before a release, it is essential to ensure that all contributions to the software are well-tested and reviewed. Maintaining the integrity of an application is equally important and requires safeguards to prevent accidental or intentional code changes that may degrade quality, but this can be a major time investment. For this reason, it becomes apparent that a way to protect, build, and deploy applications quickly and efficiently should be in place. Fortunately, the solution to this issue is the creation of a DevOps Pipeline to automate this process.

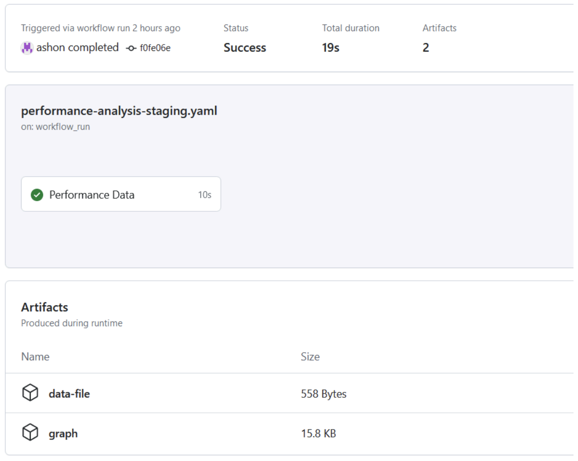
To better portray the issues mentioned, we use an example application called the *coffee project,* that takes coffee orders. The coffee project is a node application that is to be on a production server for users to access its functionality. Before deploying the application, we need to ensure that it runs and is built using best practices. We achieve this goal by implementing quality gates that require the approval of team members before any code can move on to protected branches. The project includes code and unit tests, and we need to make sure that the code is linted/formatted and that all tests execute successfully before they are accepted into our development branch and deployed to the staging server. However, this requires us to actually prepare a staging and production server in advance before our pipeline can function. Once our servers are prepared, they need to be maintained and regularly patched to remain reliable and secure. Another problem we intend to address is the need for rollback should anything go wrong with a release.

Motivation

Our primary goal is to prevent excessive human interaction needed to maintain and deploy applications in a seamless and efficient way. By architecting an event-driven Pipeline to accomplish this goal, we hope to demonstrate the effectiveness of having a workflow that enforces code reviews and the ability to continuously deploy an application using modern practices and containerization.

Summarized Accomplishments

- Feature 1: Playbooks to configure staging and production environments.
    - One playbook for updating the staging and production environments to remove security vulnerabilities.
    - One playbook to add packages and software to run the coffee-project releases.
- Feature 2: Building Quality Gates and Creating Branch Protection Rules
    - One Action to run the es-lint linter and report issues.
    - One Action to run the coffee-project tests.
    - One Action to check the coffee-project for common security pitfalls.
    - Branch protection rules to enforce the above actions in addition to other other best practices such as requiring at least one developer approval for every pull request.
- Feature 3: Deployment and Testing of Coffee-Project Releases
    - Actions to deploy releases to staging or productions depending on whether the releases have been approved by a Release Engineer.
    - Action to test the response times of staging and production deployments and provide the results. See figures below for more details.
    - Action to clean up the staging and production environments to prevent the environments from losing too much disk space.
- Feature 4: Coffee-Project Production Release Rollback
    - One GitHub Action to allow the engineer to change a broken release to a previous working release.



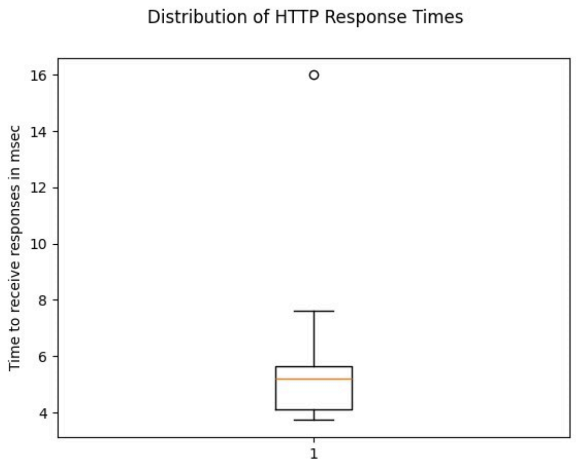
Triggered via workflow run 2 hours ago	Status	Total duration	Artifacts
ashon completed f0fe06e	Success	19s	2

performance-analysis-staging.yaml
on: workflow_run

Performance Data 10s

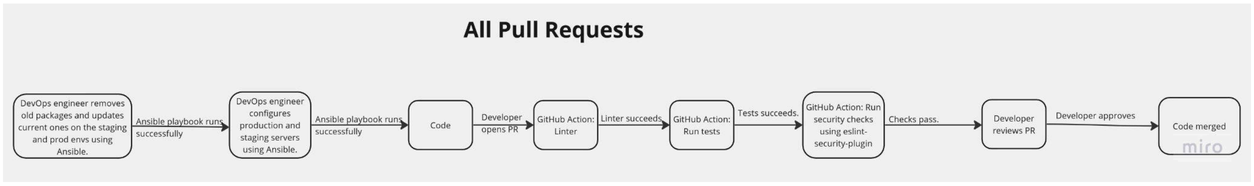
Artifacts
Produced during runtime

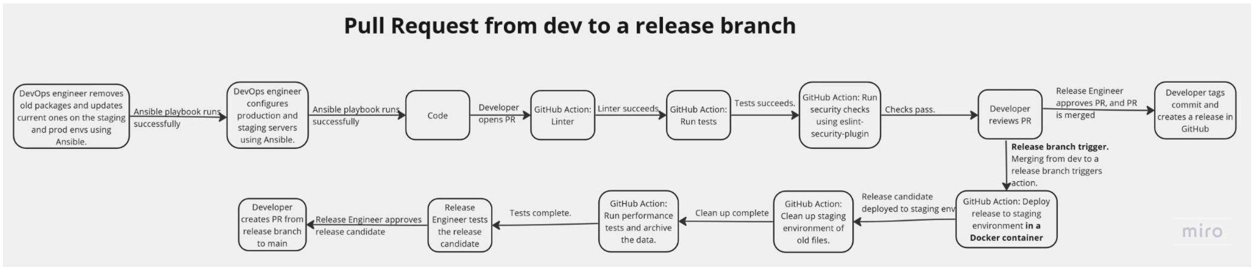
Name	Size
data-file	558 Bytes
graph	15.8 KB

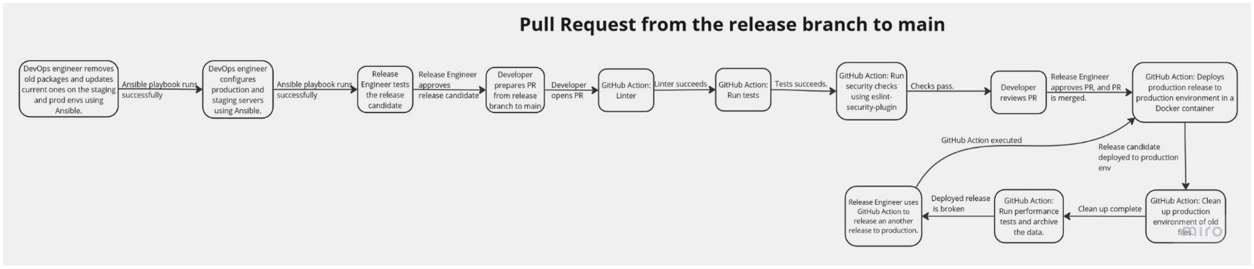


Technical Approach

Updated Pipeline Figures


All Pull Requests
DevOps engineer removes old packages and updates current ones on the staging and prod envs using Ansible.
Ansible playbook runs successfully
DevOps engineer configures production and staging servers using Ansible.
Ansible playbook runs successfully
Code
Developer opens PR
GitHub Action: Linter
Linter succeeds
GitHub Action: Run tests
Tests succeeds.
GitHub Action: Run security checks using eslint-security-plugin
Checks pass.
Developer reviews PR
Developer approves
Code merged
miro


Pull Request from dev to a release branch
DevOps engineer removes old packages and updates current ones on the staging and prod envs using Ansible.
Ansible playbook runs successfully
DevOps engineer configures production and staging servers using Ansible.
Ansible playbook runs successfully
Code
Developer opens PR
GitHub Action: Linter
Linter succeeds
GitHub Action: Run tests
Tests succeeds.
GitHub Action: Run security checks using eslint-security-plugin
Checks pass.
Developer reviews PR
Release Engineer approves PR, and PR is merged
Developer tags commit and creates a release in GitHub
Release branch trigger. Merging from dev to a release branch triggers action.
GitHub Action: Deploy release to staging environment in a Docker container
Release candidate deployed to staging env
GitHub Action: Clean up staging environment of old files.
Clean up complete
GitHub Action: Run performance tests and archive the data.
Tests complete.
Release Engineer tests the release candidate
Release Engineer approves release candidate
Developer creates PR from release branch to main
miro


Pull Request from the release branch to main
DevOps engineer removes old packages and updates current ones on the staging and prod envs using Ansible.
Ansible playbook runs successfully
DevOps engineer configures production and staging servers using Ansible.
Ansible playbook runs successfully
Release Engineer tests the release candidate
Release Engineer approves release candidate
Developer prepares PR from release branch to main
Developer opens PR
GitHub Action: Linter
Linter succeeds
GitHub Action: Run tests
Tests succeeds.
GitHub Action: Run security checks using eslint-security-plugin
Checks pass.
Developer reviews PR
Release Engineer approves PR, and PR is merged.
GitHub Action: Deploys production release to production environment in a Docker container
Release candidate deployed to production env
GitHub Action: Clean up production environment of old files.
Clean up complete
GitHub Action: Run performance tests and archive the data.
Deployed release is broken
Release Engineer uses GitHub Action to release an another release to production.
GitHub Action executed
miro

Description of Pipeline

How We Used Generative AI

We used generative AI to create example and starter files, to construct Linux commands, and to configure some of the static analysis tools. Generative AI helped us create the starter YAML files for GitHub Actions and the Ansible playbooks. From there, the starter files were modified to fix bugs, add missing steps, and include additional functionality. Generative AI also helped us by generating Linux regex commands that were used during debugging our Actions. The regex helped us find files in our file system. Lastly, generative AI provided the configuration needed to run the eslint-plugin-security executable that we used for our security analysis.

Retrospective

What Worked

Our team liked using the GitHub Project Board and using it to track issues. We met frequently enough to align on our goals and assign tasks. Then, we used asynchronous to provide advice and progress. We emphasized creating multiple pull requests or creating one big one. We thought that splitting the pipeline design into multiple components made it easier to understand. The team encouraged pair programming to remove blockers early and share knowledge. The team encouraged completing issues earlier especially when it is required for other issues to complete. The team encouraged each to make videos to help explain updates. The team encouraged linking the pull requests with the issues in the project to allow for better issue tracking.

What Didn't Work

Our team learned that we needed to host our own runner since the NCSU runners do not allow basic, universal runners. We also learned that GitHub Actions are strongly limited due to NCSU GitHub restrictions. The team recommended that everyone update their chat settings to allow for quicker communication when a team member needs assistance. The team avoided time consuming testing procedures such only testing workflows on complicated triggers when they could be tested with simpler to execute triggers.

What You Would Do Differently

Our team recommended creating videos for pull requests that are not intuitive to understand. We encouraged the team to share what they have learned early and often. Whenever, the team may need another developer for assistance alert the developer ahead of time so that he or she will watch for communication. For writing assignments, developers should look to complete writing more gradually to allow for more review and commentary earlier.

Who Did What

Isaac Taylor worked on the features for quality gates, deployment, and rollback. For the quality gates, Isaac created a workflow to lint and format the coffee-project, and worked with Corey to implement the security gate using eslint-plugin-security. For the deployment, Isaac worked with Andrew to create a workflow to deploy the coffee project to the environments and he created a workflow to clean up the deployment environment after the deployment. For rollback, Isaac worked with Andrew to implement the GitHub Action that performs rollback. Isaac wrote the introduction and the description of the pipeline.

Andrew Shon worked on the features for quality gates, deployment, and rollback. For all of the GitHub Actions, Andrew configured the self-hosted runner. For the quality gates, Andrew created a workflow for automated testing, and set up the branch protection rules to enforce the quality gates. For deployment, Andrew worked with Isaac to create a workflow that deploys the coffee-project to the environments. For rollback, Andrew worked with Isaac to implement the GitHub Action that performs rollback. Andrew completed the video for the project report.

Corey Capooci worked on the features for initial configuration, quality gates, and deployment. For initial configuration, Corey created the Ansible playbooks to configure the environments, and to remove and update packages on the environments to improve their security. For quality gates, Corey created a workflow to check for security pitfalls in Node code. For deployment, Corey created a workflow to run performance tests on staging and production deployments. Corey wrote the summarized accomplishments, generative AI, security extra credit, and who did what sections of the report. In addition, he updated the pipeline figure.

The entire team worked together on the rest of the project report.

Security Extra Credit

For our security extra credit, we decided to leverage the ESLint tool that we use for the pipeline's linting step. We did this to take advantage of a toolset that we are familiar with. According to the eslint-plugin-security's website, https://github.com/eslint-community/eslint-plugin-security, this project implements rules for Node Security and "will help identify potential security hotspots."

This fulfills the security extra credit because we use the eslint-security-plugin to identify security issues within Node software. The plugin is run every time that a pull request is created as part of the checks that must be fulfilled to complete the pull request. The elint-security-plugin checks for 14 different security hotspots in the Node software. Any hotspots are reported during the check and would prevent the user from merging the potentially malicious code into the develop or main branches.

Links to Commits

Isaac Taylor
https://github.ncsu.edu/ashon/CSC-519-DevOps-Pipeline/pull/94/commits/9a0ce0e0505b2977cb741b175527b59373ad92bc

Andrew Shon
https://github.ncsu.edu/ashon/CSC-519-DevOps-Pipeline/pull/89/commits/4e242a8816b0f4c03c08ecf78d72782a58b20be9

Corey Capooci
https://github.ncsu.edu/ashon/CSC-519-DevOps-Pipeline/pull/95/commits/d87dab9f27b6eafa4816de09cceade78d54835a0