FROM CODE CHAOS TO DEPLOYMENT SYMPHONY

Ameya Vaichalkar
Department of Computer Science
North Carolina State University
Raleigh, NC 27606
agvaicha@ncsu.edu

Deep Mehta
Department of Computer Science
North Carolina State University
Raleigh, NC 27606
dmmehta2@ncsu.edu

Subodh Gujar
Department of Computer Science
North Carolina State University
Raleigh, NC 27606
sgujar@ncsu.edu

1 Introduction

Software products rely on two main pillars: Development and Operations. In the past, these pillars had separate roles and worked in isolation. DevOps is the key to bridging this gap by making the developer who creates the feature also responsible for deploying it to production. This approach has proven to be very effective and popular in the industry and has been adopted by many large software companies.

The aim of this project is to design and implement DevOps automation using tools like GitHub Actions and Ansible, which enable developers to deliver their features with confidence and efficiency.

2 Problem Statement

In the realm of DevOps, a recurring issue is the manual intervention required for deploying, testing, and maintaining software systems. The problem is the cumbersome and error-prone process of deploying applications in a production environment, particularly when dealing with complex, multi-tiered applications and infrastructure configurations. Manual deployment procedures are time-consuming and prone to human errors, leading to service downtime, performance issues, and increased operational costs, often leading to problematic situations requiring urgent fixes. This inefficiency can result in frustrated users, loss of revenue, and decreased developer productivity.

3 Motivation

Our motivation for the project stems from the problem stated above because it impacts both developers and end-users. Developers face delays in delivering new features and bug fixes due to these deployment complexities, while users may experience service disruptions or reduced performance during updates. Automation is the key to mitigating these challenges. Automating the deployment process is critical to reducing these challenges and ensuring a seamless, error-free deployment experience. Through automation, we aim to streamline processes, reduce time-to-production, and enhance the overall efficiency of the development life cycle.

In particular, the coffee app may seem simple now, but it won't take long before it has multiple developers working on it every day. This might include adding new features, fixing existing bugs, and scaling up/down the infrastructure to ensure the users have a good experience. As the app grows and the number of developers working on it increases, the following problems arise:

- Every developer has different code styles, making it difficult for other developers to follow through or ramp up.
- A dedicated team of infrastructure and release engineers is needed to ensure the infrastructure is up to date with the latest code, run multiple test suits to ensure the application works as expected, and then plan out the release to end users.
- All the above points indicate a major problem, i.e., longer time to production. A team's efficiency is measured by the average time it takes for a code to be written, tested, verified, and deployed to customers. Automating with DevOps helps reduce this time and makes the development team efficient.

4 Summarized Accomplishments

- **Build & Test:** This workflow is the first step to ensure the quality and reliability of the code. It runs several checks, such as build, unit tests and coverage, ESlint checks, and Ansible lint checks, before allowing the docker image to be published and released.
- **Publish:** This workflow builds and publishes the docker image to GitHub packages, where it can be accessed by the deployment pipeline. The image tags are important here, as they indicate the branch and the version of the code. The master branch is always tagged as 'latest,' while other branches are tagged as 'dev.'
- **Deployment Environments:** This workflow defines five different environments for testing and deploying the code, namely DEV, QA, UAT, Baking, and PRD. Each environment has its own purpose and role, and we avoid any duplication of efforts or resources.
- **Tests:** This workflow runs various tests across the deployment pipeline, such as Integration, UI, Security, Load, and Performance tests. These tests help the developers and reviewers verify the code's functionality, usability, security, and scalability, and confidently deploy it to production.
- **Baking:** This workflow is a special stage that allows us to deploy the new features in a production-like environment and monitor their performance and behavior for a certain time. Then, we compare the results of the Baking environment, which has the new features, with the current Production environment, which has the older version of the code, to ensure that there are no unexpected or negative impacts on the resources or the user experience.
- **Load Balancer:** This workflow simulates a real-world production environment where there are multiple servers running the code and serving the users. We use a load balancer and two production servers to achieve this. This also enables us to deploy new features with zero downtime, using the blue-green deployment model, where we switch the traffic from one server to another without affecting the users.
- **Rollback:** This workflow is a safety measure that allows us to roll back the production code to a previous stable version in case of any major issue or bug detected in the latest release. We can do this with just a single click and restore functionality and user satisfaction.

5 Technical Approach

5.1 Pipeline Diagram

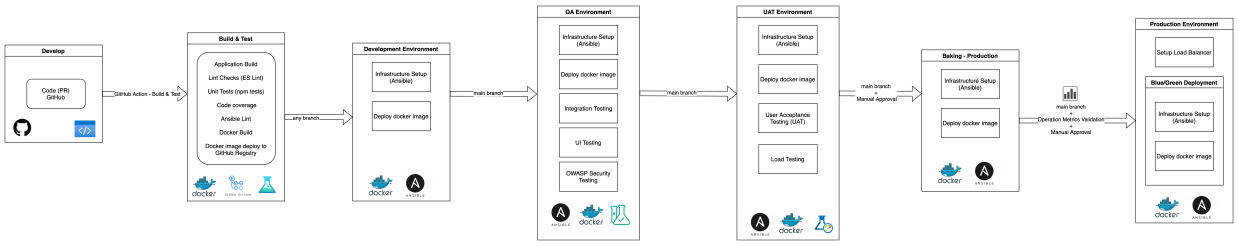


Figure 1: Pipeline Design

5.2 Description of Pipeline

5.2.1 Build, Test & Publish

Workflow: [build-test-publish.yml]

This workflow has two stages:

1. [build-test]: This installs the required dependencies and runs the following checks:

- Application Lint: We added ESlint configuration to the existing coffee project to run the ESlint checks on the code base. The existing application has many errors that are printed in [deployment logs], but it currently has

continue-on-error parameter set as true which lets the step pass even with errors as we currently do not intend to fix these errors.

- Unit Tests and Coverage: We run the existing unit tests in the coffee-project app and ensure the test coverage is greater than 90% to pass the step.
- Ansible Lint: We run ansible linting checks for the ansible playbooks in the repository. We currently have some errors, but we haven't fixed them just to showcase how the errors show up in the summary of workflow as [annotations].

2. [publish]: This builds the docker image of the coffee-project and publishes it to GitHub Packages.

- Here, we have two identical steps of build and publish; this is because we run the [Master publish] with tag latest only when the pipeline is run from master branch, but when run from any other branch it deploys with tag dev. This tag is used during the deployment stage, i.e., the docker image with tag dev is deployed in the development environment, while the latest tag is used to deploy in all other environments. This provides an opportunity for developers to see their changes in a deployment environment similar to production.
- This step doesn't run when this pipeline is run as part of a PR, as we don't want the docker image to build and publish unnecessarily for every PR. We have ensured this by following the conditions in the publish step: if: github.event_name != 'pull_request'

5.2.2 Deploy

Figure 2: Deployment Pipeline

Workflows:

1. deploy.yml: This is the main deployment workflow that is responsible for invoking templates and deploying the app to different environments.
2. deploy-template.yml: This is a template workflow that plays an important role in ensuring consistent deployment across environments.
3. rollback.yml: This workflow allows to deploy a previous stable docker image directly to the production environment in case of emergencies when the new features might have caused issues in the production environment.

Ansible Playbooks:

1. setup-docker.yml - This is responsible for setting up the infrastructure in the vcl, i.e. required packages and docker
2. deploy-application.yml - This is responsible for pulling the docker image from GitHub packages and deploying it onto the host
3. open-port.yml - This is responsible for opening up port 3000 so that the website is accessible over the internet.
4. comment_nginx.yml - The playbook updates the NGINX configuration on the load balancer used by the blue-green deployment as well as rollback by commenting out a specific server line with the target IP and then restarting the NGINX service.
5. deploy-nginx.yml - This configures the vcl as an NGINX web server, exposes the port 80, restarts the server.
6. deploy-production.yml - This orchestrates the deployment process for a production server, first removing it from the NGINX configuration, setting up Docker, deploying an application, opening a port, and then adding the production server back to the NGINX configuration.

7. hosts.yml - Inventory file with a list of IP addresses of all environments.
8. uncomment_nginx.yml - This reverts the NGINX configuration by uncommenting the server line in the NGINX default file.

Stages:

1. [DEV]: In this stage, we only set up infra and deploy the app in the development environment.

2. [QA]: Once the dev succeeds, we deploy the app in the QA environment and thus we run the following tests:

- Integration Tests: We wrote integration tests for the coffee-project which invokes the deployed APIs and ensured the orders are placed successfully.
- UI Tests: We wrote UI tests that open up the webpage, clicks on the order button and then ensures a popup is visible with ordered text.
- Security Tests: We perform an OWASP ZAP scan on the deployed application and publish the output as an [artifact] with results. We currently let it continue even if security vulnerabilities are detected as we don't have a plan to fix all issues currently.

3. [UAT]: Once QA and all its tests succeed we deploy to the UAT environment and run load tests as a [python script], by invoking the website 1000 times simultaneously and getting the average response time and ensure it is below the acceptable threshold which for now we have set as 3 seconds.

4. [Baking]: To initiate deployment to this environment we have added a manual approval check-in [environment settings], i.e., we want someone to approve the deployment from UAT to Baking manually. In baking, we measure the performance metrics, i.e., monitor its CPU and Memory Baking and Production environment and compare them to make a decision if the newly introduced features utilize the compute resources within the threshold.

5. [PRD]: As part of production deployment, we first run load balancer Ansible script to setup the load balancer infrastructure. Next, we deploy the new docker image to the production environment in a blue-green fashion.

For the blue-green deployment strategy, we are removing one VCL server out of two from the load balancer so that no user will be redirected to the ongoing deployment machine and get an error response to ensure the high availability of our application. After that, we set up Docker and deployed a newer version of the application. We add this server to the load balancer once the application is healthy and ready to serve requests.

Similar approach is followed for the deployment on the second server.

6 Use of Generative AI in the project

- GitHub Copilot: The team primarily used this tool to get boilerplate tasks done faster and get the syntax of GitHub workflows quickly rather than having to search the internet. This was also used to write Python scripts that were used for load testing and performance testing.
- Chat GPT: This was used to debug errors that we encountered while running workflows.

7 Retrospective

7.1 What worked

- Status reports and equal distribution of tasks helped the team stay on track and work in a timely manner.
- Setting up three git action runners saved waiting time as integration tests, ui-tests, and security tests run in parallel.

7.2 What didn't work

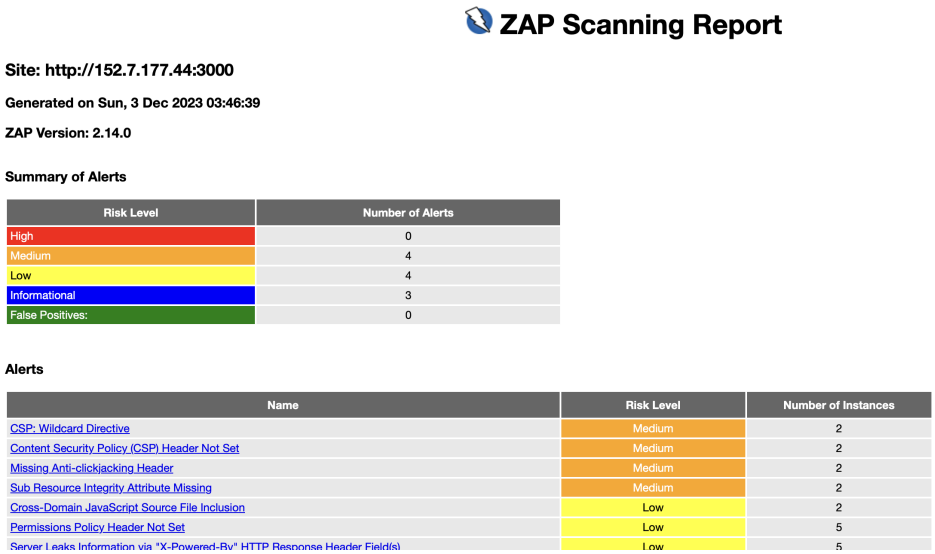
- Setting up three git action runners for parallel execution of jobs was time-consuming, as every environment required password-less SSH setup for the git action runner.
- Adding the user to the docker group was required(for every environment and the three git action runners) to provide that user with the necessary permissions to interact with the Docker daemon without sudo.

7.3 What you would do differently

- Currently, rollback is configured as a manual rollback, but it could also be automated by analyzing the production environment after deployment, and if the metrics aren't as expected, the version should rollback by default to the previous stable version.

8 Security Testing

In our QA environment, we have implemented a security scan using OWASP ZAP (Zed Attack Proxy) through a GitHub Actions workflow. The scan is configured to target the application deployed on the QA environment. Notably, the workflow is set to continue even if ZAP identifies security alerts during the full scan. The scan publishes the output as an artifact with results as shown in figure 3. We let it continue even if security vulnerabilities are detected, as we don't have a plan to fix all issues. This workflow provides a comprehensive security assessment in our QA environment while allowing flexibility in handling identified security issues.



ZAP Scanning Report

Site: http://152.7.177.44:3000

Generated on Sun, 3 Dec 2023 03:46:39

ZAP Version: 2.14.0

Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	4
Low	4
Informational	3
False Positives:	0

Alerts

Name	Risk Level	Number of Instances
CSP: Wildcard Directive	Medium	2
Content Security Policy (CSP) Header Not Set	Medium	2
Missing Anti-clickjacking Header	Medium	2
Sub Resource Integrity Attribute Missing	Medium	2
Cross-Domain JavaScript Source File Inclusion	Low	2
Permissions Policy Header Not Set	Low	5
Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)	Low	5

Figure 3: ZAP Scan Artifact

9 Team Contributions

1. **Ameya Vaichalkar** - VCL Setup, Ansible Playbooks, Deployment workflow, Integration Test, Load Balancer
   - Ansible Playbook and Deployment Workflow (Commit)
   - Deployment Workflow, Integration Tests (Commit)
   - Load Balancer Setup (Commit)
2. **Deep Mehta** - Lint Checks, Unit Test Coverage, Workflow Templates and Variables, UI Test, OWASP Security Test, Rollback
   - Linting, Code Coverage Checks and Deployment Templates (Commit)
   - Variable Group Action, UI & OWASP Tests, Environment Setup (Commit)
   - Fix Pipeline Issues, Integrate new components into main deploy pipeline, rollback workflow (Commit)
3. **Subodh Gujar** - Build, Test & Publish Setup, Ansible Playbooks, Load Testing, Performance Testing, Blue-Green Deployment
   - Build, Test, Publish Workflow and Deployment Workflow (Commit)
   - Ansible Playbooks, Load Testing Script (Commit)
   - Blue-Green Deployment playbook and workflow setup (Commit)