**as**

**Cloud Development and Operations, Conestoga College**

**INFO8995-24F-Sec1: Container and Orchestration**

**Rahul Chandna**

**Astha Suthar**

**Professor: Andy Chow**

**Assignment 4**

**Group 9**

**Dec 04, 2024**

**Project Planning for Binary Build and Upload Pipeline**

**Objective:**

Build a PoC DevOps pipeline to generate a binary and push it to a Github repository in a simulated, real-world environment.

**Core Process Overview**

1. Build Binary

* Source code will be compiled to binary using an emulated dev environment.
* Build will be executed inside a Docker container to scale to job-runner functionality in future.

2. Upload to GitHub

* The binary will be uploaded to a Github repo and an auth token will be issued.
* GitHub Actions or a CI/CD pipeline tool (such as Jenkins or GitLab CI/CD) will be modelled to automate this action.

**Proposed Steps for Execution**

1. Setup Source Code Repository

o Create a demo repository in Github with the mock source code (a C/C++ or Go script that produces a binary).

o The repository will contain:

A Dockerfile specifying the build environment.

build.sh to build the binary.

o A basic .gitignore file.

2. Design Docker Container

o Install lightweight base image (alpine or ubuntu) with built-in build tools.

o Define a private docker image to model the environment:

Install dependencies like gcc for C/C++ or go for Go binaries.

Add scripts to automate the build.

3. Simulate GitHub Actions Workflow

o Model GitHub Actions with a local CI/CD simulation using Docker Compose:

Execute build and upload jobs through docker run.

Upload binary from container using GitHub CLI (gh).

o Or write an elementary shell script that automates all these steps to show you how pipeline works.

4. Setup Authentication

o Create a Personal Access Token (PAT) on Github.

o Save it in a safe place (equivalent to environment variables inside the docker container).

5. Document and Simplify

o Give clear, simple instructions on how to run the simulation:

Clone the repository instructions.

How to compile the binary from the Docker container.

Paths for copying the binary to GitHub.

6. Identify and Mitigate Gaps

o Simulate real-world constraints like:

1. Limited environment access.
2. Network connectivity issues.

o Comment and report where the simulation is different from production.

**Supporting Environment Simulation**

1. Development Environment

o Slapped on a docker container.

o Download prebuilt build packages depending on the language.

2. CI/CD Environment

o Simulated using shell scripts or Docker Compose as a local job runner.

3. GitHub

o Import binaries from a test repository.

**Potential Challenges and Mitigation**

|  |  |
| --- | --- |
| **Challenge** | **Mitigation** |
| Accurate simulation of CI/CD environment | Use Docker Compose to replicate job runners and workflows locally. |
| Lack of official infrastructure access | Rely on lightweight Docker containers to simulate infrastructure. |
| Authentication and token management | Store tokens securely as environment variables and use .env files to simulate secure storage. |
| Build environment discrepancies | Document and provide a detailed list of tools and libraries included in the Docker container. |

**Demonstration Execution Plan**

1. Build the Environment

o Copy the demo repo: git clone https://github.com/demo-repo.git.

o Create the Docker image: docker build-t binary-builder.

2. Build the Binary

o Launch the container to compile the binary:

bash

Copy code

docker run --rm -v $(pwd):/app binary-builder sh /app/build.sh

3. Upload the Binary

o Upload GitHub CLI in a Docker container mimic:

bash

Copy code

docker run -rm -e GITHUB\_TOKEN=your\_token -v $(pwd):/app binary-uploader gh release upload release-name> /app/output-binary

4. Document Steps

o Add step by step instructions in the README.md file on reproducing the PoC.

**Best Practices and Simplicity**

• Automation: Automation with scripts and Docker reduces human involvement.

• Modularity: Separate build and upload processes for debugging purposes.

• Security: Never hardcode your passwords; use environment variables.

• documentation: Post a single README.md that explains the entire process.