**Assignment 4**

**Group 9**

**Darshan Wadekar (8934575)**

**Dhavalkumar Patel (8933139)**

**Cloud Development and Operations, Conestoga College**

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

**Professor: Andy Chow**

**Dec 04, 2024**

**Planning for a Project Solution (Part I)**

**Project Overview**

As part of DevOps solutioning for a project where technological specifications and are not known in advance, we are proposing a solution which could cover most use cases which may arise, this helps in reducing repetition and the framework can automate deployment for variety of technologies as per the use case.

**1. Objectives and Goals**

**Goal 1:** Create and deploy a solution for compiling a compiled binary using Docker containers.

Technological challenge : The solution proposed should support multiple tech stacks (for e.g. Node.js/Python/Java for the application or for front end and MySQL/PostGreSQL etc for the database).

Repeatable: It should be repeatable across technology stack to the point that it could be duplicated and executed by a user with little configuration, this is required to accommodate wide variety of technology it might need to cater to.

**Goal 2:** Post the compiled binary on GitHub.

To make sure the system copies the binary or artifact to a GitHub repository.

SSH key or GitHub API can be used to achieve this task.

**Goal 3:** Run the entire pipeline orchestrated using Docker Compose.

Let the solution scale, or swap technology stacks, with very little changes in configuration.

**2. Core Process Flow**

**Application Build:**

Containerized Build: Building the applications compiled binary in Docker containers (e.g., Node.js or Python).

Personal Docker Images: build Docker images based on the chosen tech stack (Node.js, Python, etc.) and compile the binary inside the container.

**Artifact Upload:**

After the binary has been built, the artifact is pushed to a GitHub repository.

Upload using Git CLI, GitHub API or SSH.

**Pipeline Execution:**

Docker Compose automates build and upload steps, making sure dependencies (such as databases) are handled before the app is executed.

**3. Supporting Environment Simulation**

Since we do not have real environments (e.g., CI/CD pipelines, cloud infrastructure) to work with, we will simulate an environment with Docker and Docker Compose.

**Simulated Environment Components:**

Application Container: A container which executes the application build cycle (Node.js, Python).

We will use environment variables to set stack (APP\_IMAGE=python:3.9, DB\_IMAGE=mysql:latest).

Database Container: The database container (MySQL, Redis)

You can easily replace the database by modifying the DB\_IMAGE value (for example from mysql to redis).

Data Persistence Volume: Allow the database to persist data between container restarts.

Technology Stack Flexibility:

Javascript Node.js: Choose the official node image for JavaScript applications.

Python: The official python image for Python applications.

MySQL/Redis: Embrace mysql or redis images, as the user desires.

**4. Docker Compose Configuration**

**Key Features:**

Portability: We can customize the docker-compose.yml file to support multiple tech stacks without changing the Docker Compose file.

Environment Variables: Env files specify which application stack (Node.js, Python) and database (MySQL, Redis) to use.

Volume Mounting: To save app and database data to Docker volumes to ensure long-term data storage.

Simplified network configuration: Since this a simulation before actual build, not having to consider complexities due to network configuration is helpful, if required network configuration can be added easily at a later stage.

**5. Potential Issues and Blockers**

**Authentication with GitHub:**

Problem: It can be difficult to replicate GitHub’s OAuth or SSH authentication locally.

Solution: Use personal keys to use the API or ssh keys for repository transactions. This could involve requiring users to pre-set their SSH keys or tokens in a file.

**Handling Technology Stack Changes:**

Problem: When migrating between tech stacks (Node.js/Python), it may require additional configuration changes.

Solution: Enable flexible configuration (Docker environment variables and volume mounts) to transition tech stacks without a major docker Compose file overhaul.

**Complexity in Container Dependencies:**

Problem: Some applications may require additional dependencies (e.g., compilers in a specific language).

Solution: Create separate Docker images for each stack with all the required tools and dependencies preinstalled.

**Database Initialization:**

Problem: The setup of databases (e.g., schema, seeding) could be difficult to simulate.

Solution: Create scripts in the Docker entrypoint to auto-register the database when the container starts up and eliminate the need for human ops.

**6. Principles of Simplicity and Usability**

**Minimal Setup:**

Users should only need to issue one docker-compose up command to initialize the environment.

Makefiles can be used to make builds, uploads, etc.

**Environment Configuration:**

Utilizing .env files to define environment variables, allowing administrators to easily update the system without affecting core files.

**Scalability:**

The solution should be extensible for more advanced applications.

**Conclusion**

The solution will be modular, technology independent, and easy to scale for future expansions. We can use Docker and Docker Compose to simulate what it would look like in real life to compile a compiled binary, upload it to GitHub, and run it in containerized mode.