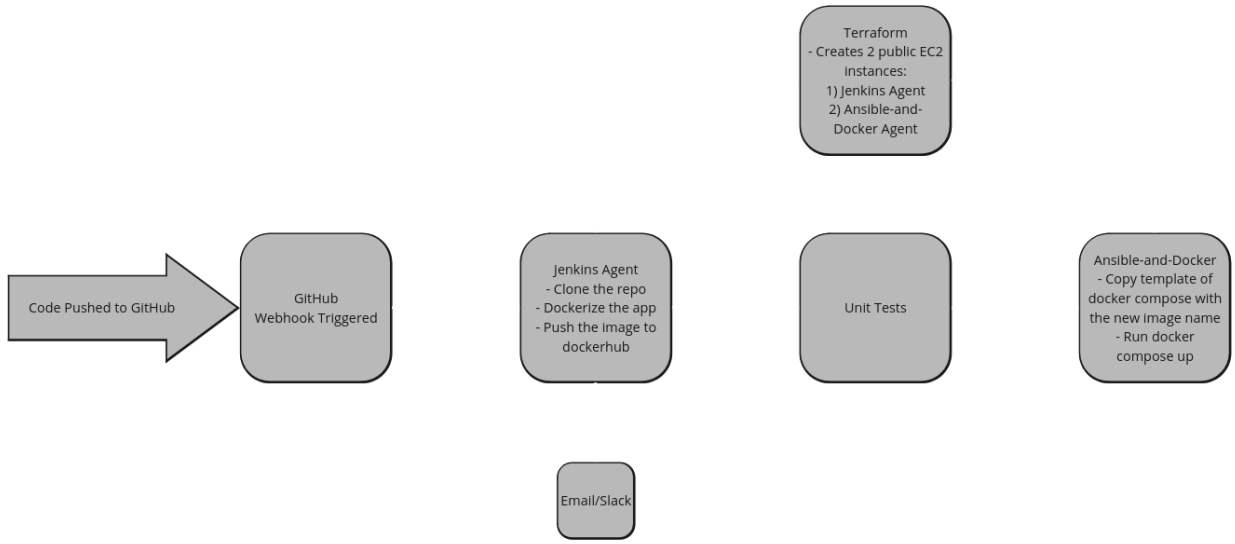
Automated Deployment Pipeline with Jenkins and Docker

BY: Ahmed sayed, yousef mohamed, mohamed ramadan, David nabil, Shehab mustafa

Overview

This project demonstrates an end-to-end Continuous Integration/Continuous Deployment (CI/CD) pipeline using Jenkins, Docker, Ansible, and Terraform. The pipeline automates the process from code push to deployment, utilizing infrastructure as code (IAC) and containerization principles for consistent and automated deployments.



Pipeline Steps Explanation

Step 1: Infrastructure Setup with Terraform

Terraform is used to provision two EC2 instances in AWS:

- Jenkins Agent EC2: Hosts the Jenkins agent that runs the pipeline.
- Ansible and Docker Agent EC2: Used to deploy the application.

Terraform scripts define and manage the infrastructure-as-code, ensuring reproducibility and consistency in environment setups.

Step 2: Code Push to GitHub

When the developer pushes code to the GitHub repository, a webhook is triggered, notifying Jenkins of the update. This initiates the pipeline process automatically.

Step 3: Dockerization of the Application

Upon receiving the webhook, Jenkins:

1. Clones the repository: Jenkins pulls the latest code from GitHub.
2. Dockerizes the application: Jenkins uses a Dockerfile to package the application into a Docker image.

Step 4: Unit Testing

Jenkins runs unit tests to ensure that the new code does not break the application. If the unit tests pass, the deployment process proceeds. If they fail, Jenkins stops the process and notifies the team via email or Slack. After the image is tested, it is pushed to a Docker Hub repository for storage and future deployments.

Step 5: Deployment with Ansible and Docker

- Docker Compose Template Update: Ansible updates the Docker Compose file template, injecting the new Docker image name.
- Deployment:
    - If there are changes in the image or configuration, Ansible runs docker-compose up on the Ansible-and-Docker EC2 instance.
    - This deploys the latest version of the application.

Step 6: Notifications

Throughout the pipeline, JenkAnsible and Docker Agent EC2: For application deployment.

    - Security Groups: Ensure SSH (port 22) and HTTP/HTTPS (port 80/443) are open for server access.

2. Tools Installed
    - Jenkins: Installed on the Jenkins Agent EC2 instance.
    - Terraform: Installed on your local machine or build server.
    - Docker: Installed on both EC2 instances.
    - Ansible: Installed on the Ansible and Docker Agent EC2 instance.
3. GitHub Repository
    - Code Repository: Application code with a Dockerfile and unit tests.
    - Webhook Setup: GitHub webhook to trigger Jenkins builds.
4. Docker Hub Account Docker Hub Repository: For storing Docker images created during the pipeline. Jenkins Plugins

Install the following plugins in Jenkins to enable the full CI/CD pipeline:

- Git Plugin: Allows Jenkins to pull code from GitHub.
- Docker Pipeline Plugin: Provides support for Docker operations in Jenkins Pipeline.
- Ansible Plugin: Runs Ansible playbooks for automated deployments.
- Mailer Plugin: Sends custom email notifications for build and deployment statuses.

ins is configured to send notifications about the pipeline status (successful build, failed tests, deployment status) via email.

Prerequisites

Ensure you have the following prerequisites set up before running the pipeline:

1. Infrastructure
    - AWS Account
    - Two EC2 Instances:
        - Jenkins Agent EC2: Where Jenkins will run.
        - Ansible and Docker Agent EC2: For application deployment.
    - Security Groups: Ensure SSH (port 22) and HTTP/HTTPS (port 80/443) are open for server access.
2. Tools Installed
    - Jenkins: Installed on the Jenkins Agent EC2 instance.
    - Terraform: Installed on your local machine or build server.
    - Docker: Installed on both EC2 instances.
    - Ansible: Installed on the Ansible and Docker Agent EC2 instance.
3. GitHub Repository
    - Code Repository: Application code with a Dockerfile and unit tests.
    - Webhook Setup: GitHub webhook to trigger Jenkins builds.
4. Docker Hub Account Docker Hub Repository: For storing Docker images created during the pipeline. Jenkins Plugins

Install the following plugins in Jenkins to enable the full CI/CD pipeline:

- Git Plugin: Allows Jenkins to pull code from GitHub.
- Docker Pipeline Plugin: Provides support for Docker operations in Jenkins Pipeline.
- Ansible Plugin: Runs Ansible playbooks for automated deployments.
- Mailer Plugin: Sends custom email notifications for build and deployment statuses.

Environment Setup

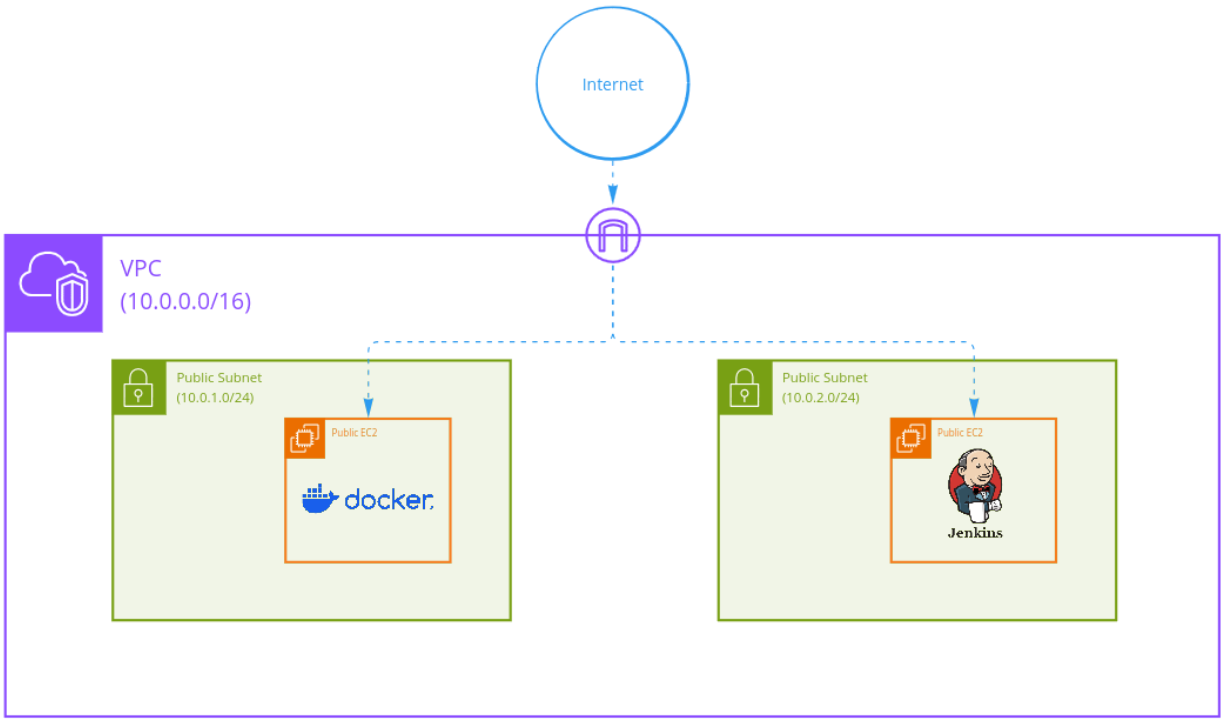
This environment setup leverages Terraform for provisioning AWS infrastructure and Ansible for configuring Docker and Jenkins on EC2 instances. The setup includes creating two public

subnets, one for a Docker agent and another for a Jenkins agent. This section of the repository is focused on setting up the infrastructure required for an automated deployment pipeline.

Overview

This project automates the deployment of two EC2 instances in AWS, each running a different service:

- Docker Agent: Installed on one EC2 instance to run Docker.
- Jenkins Agent: Installed on a separate EC2 instance to run Jenkins. Terraform is used to provision the necessary AWS infrastructure, including EC2 instances, security groups, and key pairs. Ansible is used to install and configure Docker and Jenkins on the respective EC2 instances.



Prerequisites

Before running this setup, ensure the following tools are installed:

- Terraform (version 0.12 or above)
- Ansible (version 2.9 or above)
- AWS CLI (for managing AWS resources)
- SSH Access (for connecting to the EC2 instances) You should also have an AWS account configured with appropriate permissions to create EC2 instances, security groups, and VPCs.

Infrastructure Setup

Terraform Configuration

The infrastructure is defined using Terraform scripts, which include:

- AWS VPC: A Virtual Private Cloud with two public subnets.
- EC2 Instances: Two EC2 instances, one for the Docker agent and one for the Jenkins agent.
- Security Groups: A security group allowing SSH, HTTP, and Jenkins port access.
- Key Pair: An RSA private key is generated and used for EC2 access.

Key components of the Terraform configuration:

- Public EC2 Instances: The instances are created with public IPs and are assigned to the public subnets.
- Provisioner: A local provisioner is used to dynamically update an inventory file for Ansible after the EC2 instances are created.

Variables

Key values used in the Terraform configuration:

VPC Configuration:

- VPC Name: test-vpc
- VPC CIDR Block: 10.0.0.0/16
- Public Subnets CIDR: ["10.0.1.0/24", "10.0.2.0/24"]
- Availability Zones: ["us-east-1a", "us-east-1b"]

EC2 Configuration:

- Instance Names: ["docker-agent", "jenkins-agent"]
- AMI ID: ami-0e86e20dae9224db8
- Key Name: private_key
- Instance Type: t2.micro

Configuration Setup

Ansible Playbooks

Once the EC2 instances are provisioned, Ansible is used to configure them.

Docker Agent Configuration

The docker-agent EC2 instance is configured with Docker. The following tasks are executed:

- Update the apt cache and install necessary prerequisites.

- Add the official Docker GPG key and repository.
- Install Docker and ensure the service is running.

Jenkins Agent Configuration

The jenkins-agent EC2 instance is configured with Jenkins. The tasks include:

- Install Java, a prerequisite for Jenkins.
- Add the Jenkins GPG key and repository.
- Install Jenkins and start the service.
- Configure the firewall to allow traffic on port 8080 (Jenkins' default port).

Running the Setup

Follow these steps to run the environment setup:

Step 1: Clone the Repository

git clone <repository-url>
cd Environment-setup

Step 2: Initialize and Apply Terraform

Ensure that your AWS credentials are set up, and then run the following Terraform commands:
cd terraform
terraform init
terraform apply
This will provision the infrastructure, including EC2 instances, security groups, and key pairs.

Step 3: Run Ansible Playbooks

After Terraform completes, the dynamic inventory file will be updated with the IP addresses of the EC2 instances. Run the following Ansible playbook to configure the instances:
ansible-playbook setup_project.yml

Web Application for Automated Deployment

This app is a Python-based web application that has been Dockerized as part of an automated deployment setup. The app increments a counter stored in Redis and displays the environment in which it is running.

Application Structure

The web app is built using Tornado and Redis to store and manage a simple counter. It serves an HTML page that displays the environment and the counter value.

Key Components:

- Tornado Framework: Used to handle web requests and serve the application.
- Redis: An in-memory data store used to manage a counter.
- Templates and Static Files: The HTML templates are in the templates/ folder, and static assets are in the static/ folder.

Updates

Changes Made:

- Dockerization: A Dockerfile was created to containerize both the app and Redis, simplifying deployment.
- Docker Compose: A docker-compose.yml file was added to manage and run the app and Redis containers together.
- Environment Variables: The app configuration no longer relies on a .env file. All necessary environment variables are directly set within the Dockerfile. These changes make the application easier to deploy and manage in a containerized environment.

Dockerfile Explanation

The Dockerfile is a script that contains a series of instructions to build a Docker image for your web application. Here's a breakdown of its components:

- Base Image: The FROM python:latest instruction sets the base image to the latest version of Python. This provides a pre-configured environment with Python installed.
- Working Directory: The WORKDIR /myapp instruction sets the working directory inside the container to /myapp. This is where all subsequent commands will be executed.
- Installing Dependencies: The RUN pip install tornado redis command installs the necessary Python packages, Tornado and Redis, which are required for the application to run.
- Copying Files: The COPY . . instruction copies all files from the current directory on your host machine into the /myapp directory in the container. This includes your application code and any necessary files.
- Environment Variables: The ENV instructions set various environment variables needed by the application. These include:
    - ENVIRONMENT: Specifies the running environment (e.g., DEV).
    - HOST: Defines the hostname for the application.
    - REDIS_HOST: The hostname for the Redis server.
    - REDIS_PORT: The port for Redis communication.

    - PORT: The port on which the application will run.
    - REDIS_DB: Specifies which Redis database to use.
- Exposing Ports: The EXPOSE 8000 instruction informs Docker that the application will listen on port 8000, allowing external access to this port when the container is running.
- Command to Run the Application: The CMD ["python" , "hello.py"] instruction specifies the command that will be executed when the container starts. In this case, it runs the hello.py script, starting the Tornado web server.

Docker Compose File Explanation

The Docker Compose file (docker-compose.yml) is used to define and run multi-container Docker applications. Here's what it includes:

1. Version: The version: "3" line specifies the version of the Docker Compose file format.
2. Services: This section defines the different services that make up the application.
    - my_app: This is the main application service. It includes:
        - container_name: Sets a custom name for the container.
        - build: Specifies the context and Dockerfile to use for building the image.
        - ports: Maps port 8000 on the host machine to port 8000 in the container, allowing external access to the application.
        - depends_on: Indicates that this service depends on the database service (Redis), ensuring that Redis starts before the application.
    - database: This service represents the Redis database.
        - image: Specifies the Docker image to use for Redis. By default, it pulls the official Redis image from Docker Hub.

License