Web Application Deployment with Jenkins, Docker, and Ansible

A web application deployment project using Jenkins, Docker, and Ansible in a DevOps context involves automating the process of building, testing, and deploying a web application. This approach ensures consistency, scalability, and efficiency in the software development lifecycle. Let's break down the components and provide an introduction overview:

Project Overview:

1. Objective:

The primary goal of this project is to streamline the deployment pipeline for a web application using DevOps practices. This involves automating the build, test, and deployment phases to enhance the development workflow.

2. Technologies Used:

- **Jenkins:**
  - Acts as the automation server, orchestrating the entire deployment pipeline.
  - Manages the building, testing, and deployment of the application.
- **Docker:**
  - Containerization technology used to package the application and its dependencies into containers.
  - Ensures consistency across different environments and simplifies deployment.
- **Ansible:**
  - Configuration management tool used for provisioning, configuration, and application deployment.
  - Helps maintain consistency in infrastructure and automates tasks across multiple servers.

3. Workflow:

- **Source Code Management (SCM):**
  - Code is stored in a version control system (e.g., Git).
  - Jenkins monitors SCM for changes to trigger the pipeline.
- **Build:**
  - Jenkins pulls the source code and initiates the build process.
  - Docker is used to create container images with the application and its dependencies.
- **Test:**
  - Automated tests are executed to ensure the application's functionality and reliability.
  - Jenkins reports test results and triggers further actions based on the outcomes.
- **Deployment:**
  - Ansible is employed to deploy the application to various environments (development, staging, production).
  - Docker containers are orchestrated for scaling and managing the application.

4. Continuous Integration/Continuous Deployment (CI/CD):

- **Continuous Integration (CI):**
  - Ensures that changes are regularly integrated into the main codebase.
  - Automated builds and tests are triggered on every code commit.
- **Continuous Deployment (CD):**
  - Automates the deployment process, allowing for frequent and reliable releases.
  - Ensures that the application is deployable at any point in the development process.

5. Benefits:

- **Automation:**
  - Reduces manual intervention, minimizing errors and improving efficiency.
  - Enables frequent and reliable releases.
- **Consistency:**
  - Docker ensures consistent environments across different stages of deployment.
  - Ansible provides consistent configuration management.
- **Scalability:**
  - Docker containers allow for easy scaling of the application based on demand.
  - Ansible simplifies the management of infrastructure scaling.

6. Security Considerations:

- Secure configuration practices are implemented using Ansible roles.
- Docker images are scanned for vulnerabilities before deployment.

7. Monitoring and Logging:

- Monitoring tools are integrated to track the performance and health of the application.
- Logging mechanisms are in place to facilitate troubleshooting and debugging.

Conclusion:

This DevOps project aims to enhance the development and deployment processes by integrating Jenkins, Docker, and Ansible. The automation and containerization aspects contribute to a more efficient, scalable, and consistent deployment pipeline, ultimately improving the overall software delivery lifecycle.

Web Application Deployment with Jenkins, Docker, and Ansible with prectice

1. Introduction<a name="introduction"></a>

This documentation outlines the setup and configuration of a DevOps project using Jenkins, Git, Docker, and AWS. The project involves building, tagging, and pushing Docker images automatically using Jenkins and Ansible.

2. Infrastructure Setup<a name="infrastructure-setup"></a>

AWS EC2 Instances<a name="aws-ec2-instances"></a>

Create three public AWS EC2 instances named Jenkins, Docker, and Ansible.

Instance Naming<a name="instance-naming"></a>

- Jenkins Instance: `jenkins-instance`
- Docker Instance: `docker-instance`
- Ansible Instance: `ansible-instance`

Connecting Instances<a name="connecting-instances"></a>

Ensure that the instances can communicate with each other. Use SSH for connecting to each instance.

3. Dependency Installation<a name="dependency-installation"></a>

Jenkins Instance<a name="jenkins-instance"></a>

Connect to the Jenkins instance and install the necessary dependencies:

```
bash
sudo apt update
sudo apt install openjdk-8-jdk -y
sudo apt install git -y
```

Docker Instance<a name="docker-instance"></a>

Connect to the Docker instance and install Docker:

```
bash
sudo apt update
sudo apt install docker.io -y
sudo usermod -aG docker ${USER}
sudo systemctl restart docker
```

Ansible Instance<a name="ansible-instance"></a>

Connect to the Ansible instance and install Ansible:

```
bash
sudo apt update
sudo apt install ansible -y
```

4. Jenkins Installation<a name="jenkins-installation"></a>

Install Jenkins on the Jenkins instance:

```
bash
wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key
add -
sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ >
/etc/apt/sources.list.d/jenkins.list'
sudo apt update
sudo apt install jenkins -y
sudo systemctl start jenkins
sudo systemctl enable jenkins
```

Configuring Jenkins<a name="configuring-jenkins"></a>

- Access Jenkins on your browser by navigating to `http://<jenkins-instance-public-ip>:8080`.
- Follow the on-screen instructions to complete the setup.
- Install necessary plugins, including the Git plugin.

Connecting Git Repository<a name="connecting-git-repository"></a>

- In Jenkins, navigate to "Manage Jenkins" > "Manage Plugins" > "Available" and install the Git plugin.
- Create a new Jenkins job and configure it to connect to your Git repository.

5. Automation Steps<a name="automation-steps"></a>

Jenkins Configuration<a name="jenkins-configuration"></a>

In the Jenkins job configuration, add the following build steps:

1. Execute Shell:

```
   bash
1. docker image build -t myimage:v1
2. docker image tag myimage:v1 mitul006/myimage:v1
3. docker image push mitul006/myimage:v1
```

4.

Ansible Configuration<a name="ansible-configuration"></a>

In the Ansible instance, create an Ansible playbook (`deploy.yml`) with the following content:

```
yaml
---
- hosts: docker-instance
  tasks:
    - name: Build and Tag Docker Image
      shell: |
        docker image build -t ..$JOB_NAME..$BUILD_ID
        docker image tag ..$JOB_NAME:.$BUILD_ID
mitul006/..$JOB_NAME:.$BUILD_ID
        docker image push mitul006/..$JOB_NAME:.$BUILD_ID
        docker image tag ..$JOB_NAME:.$BUILD_ID mitul006/..$JOB_NAME:latest
        docker image push mitul006/..$JOB_NAME:latest
        docker image rmi ..$JOB_NAME:.$BUILD_ID
mitul006/..$JOB_NAME:.$BUILD_ID mitul006/..$JOB_NAME:latest
      environment:
        JOB_NAME: "..$JOB_NAME"
        BUILD_ID: ".$BUILD_ID"
```

Docker Instance Configuration<a name="docker-instance-configuration"></a>

1. Create a `docker-compose.yml` file on the Docker instance:

```
yaml
---
version: '3'
services:
  docker-host:
    tasks:
      - name: stop container
        shell: docker container stop cloudknowledge
      - name: remove container
        shell: docker container rm cloudknowledge
      - name: remove docker image
        shell: docker image rmi mitul006/cloudknowledge
      - name: create new container
        shell: docker container run -itd --name cloudknowledge-container -p
9000:80 mitul006/cloudknowledge
```

2. Run the Ansible playbook:

```
bash
ansible-playbook -i localhost, -e "JOB_NAME=example_job BUILD_ID=1"
deploy.yml
```

6. Conclusion<a name="conclusion"></a>

Your DevOps project is now set up and configured. Jenkins is connected to your Git repository, and the automation steps for building and pushing Docker images are defined using Ansible. Monitor Jenkins for successful builds and deployments. Adjust configurations as needed for your specific project requirements.

This documentation provides a foundation for your DevOps workflow. Customize it based on your project's unique needs and scale as required.