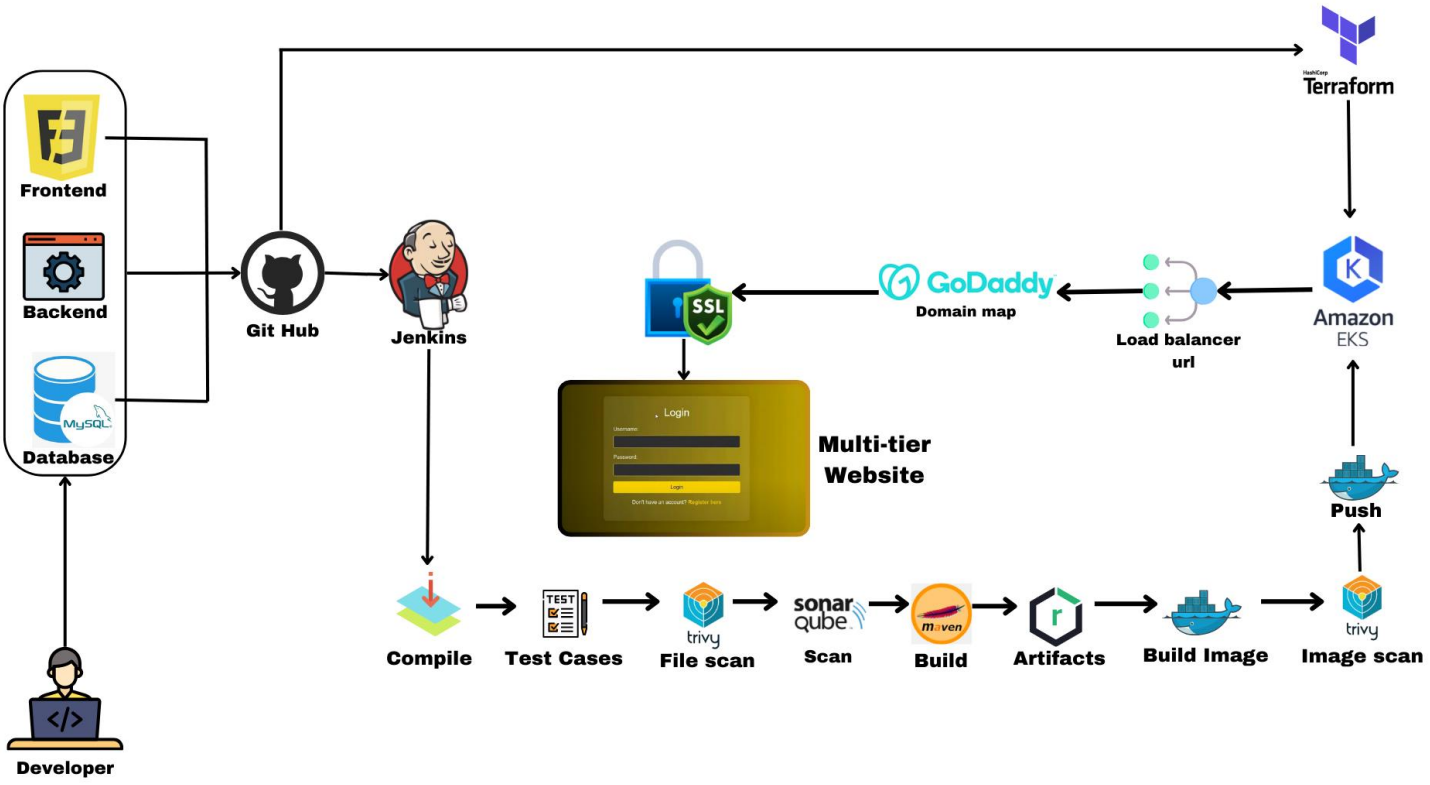
DevOps Shack

Multi-Tier CICD Pipeline with SSL Certificate

In today's fast-paced software development environment, building and deploying a three-tier application requires a streamlined and automated process to ensure efficiency and reliability.. The setup will involve configuring the necessary infrastructure, deploying the application on AWS EC2 instances, and ensuring that each tier (presentation, application, and database) is properly integrated and monitored. Tools such as Jenkins, SonarQube, Nexus, Docker, and Kubernetes will be utilized to manage the build, testing, deployment, and monitoring processes, ensuring a seamless and consistent workflow from development to production.



Key tools used in this pipeline include:

AWS: For creating and managing virtual machines.

Jenkins: For automating the build, test, and deployment processes.

SonarQube: For static code analysis to ensure code quality.

Trivy: For file scanning and vulnerability scanning of Docker images.

Nexus Repository Manager: For managing artifacts.

Terraform: As infrastructure as code to create the EKS Cluster.

Docker: For containerization to ensure consistency and portability.

Kubernetes: For container orchestration and deployment.

SSL Certification: To secure communication between clients and the server.

By following this guide, you'll be able to set up a fully functional CI/CD pipeline that supports continuous delivery, helps maintain high standards for code quality, and ensures secure application deployment.

Workflow

The workflow begins with setting up the necessary infrastructure on AWS, where EC2 instances are provisioned to host Jenkins, SonarQube, and Nexus. Terraform is used to automate the creation of an Amazon EKS cluster for deploying the application. Jenkins automates the build process, pulling code from the repository, running tests, and using SonarQube for static code analysis to ensure quality. Docker containers are created for the application and database, with Trivy scanning the Docker images for vulnerabilities. Nexus manages the application’s artifacts, and Kubernetes orchestrates the deployment on the EKS cluster. Finally, SSL certification is applied to secure communication between clients and the server, ensuring a secure and reliable deployment.

Step 1: Set Up GitHub Repository and Push Local Code

Create a New GitHub Repository:

1. Login to GitHub:

- Visit GitHub and log in to your account.

2. Create a New Repository:

- Click on the "+" icon in the top-right corner and select "New repository."

- Repository Name: Enter a name for your repository (e.g., `my-ci-cd-project`).

- Description: (Optional) Add a brief description of your project.

- Visibility: Choose between Public or Private.

- Initialize Repository:

- You can choose to add a README file, `.gitignore` file, and select a license, or leave these options unchecked if you're pushing an existing project.

3. Click on "Create repository."

2.2 Push Existing Local Code to GitHub:

1. Initialize Git in Your Local Project:

- Open your terminal or command prompt.

- Navigate to your project directory:

```
cd /path/to/your/project
```

- Initialize Git:

```
git init
```

2. Add Remote Repository:

- Add your GitHub repository as a remote:

```
git remote add origin https://github.com/your-username/my-ci-cd-project.git
```

- Replace `your-username` and `my-ci-cd-project` with your GitHub username and repository name.

3. Add and Commit Your Code:

- Stage all changes:

```
git add .
```

- Commit the changes:

```
git commit -m "Initial commit"
```

4. Push to GitHub:

- Push your code to the `main` branch:

```
git push -u origin main
```

Repository Link

Repository URL Add your GitHub repository link here:

https://github.com/jaiswaladi246/Multi-Tier-With-Database.git

Step 2. Launch Virtual Machine for Jenkins, Sonarqube and Nexus

Here is a detailed list of the basic requirements and setup for the EC2 instance i have used for running Jenkins, including the specifics of the instance type, AMI, and security groups.

EC2 Instance Requirements and Setup:

1. Instance Type

- Instance Type: `t2.large`

  - vCPUs: 2

  - Memory: 8 GB

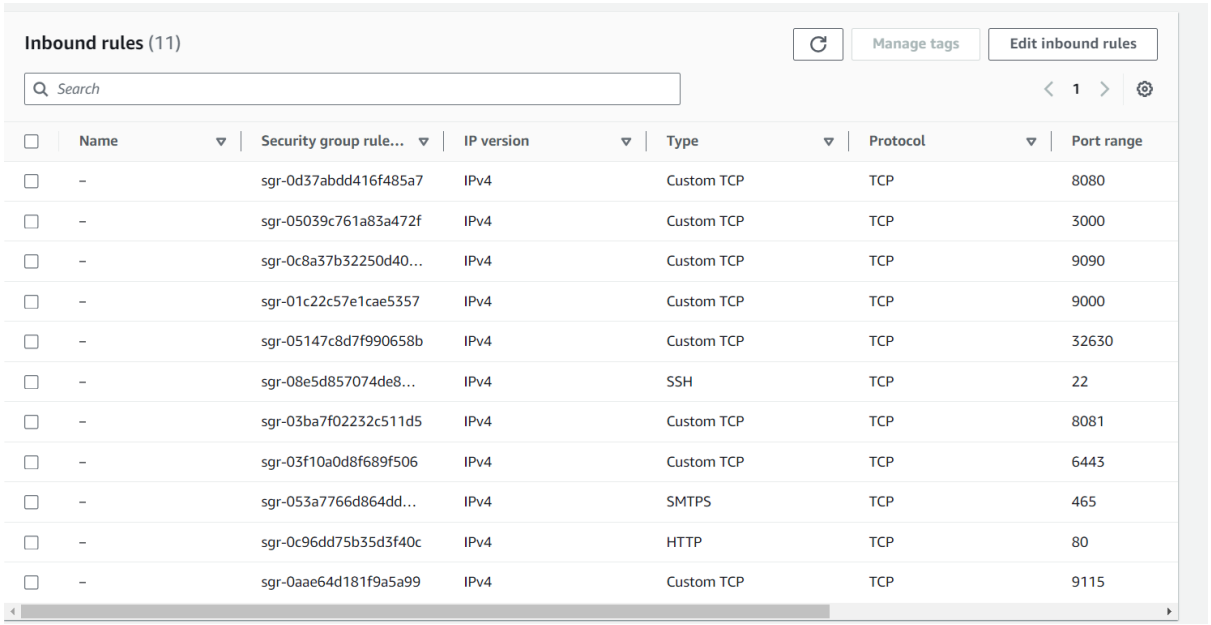
  - Network Performance: Moderate

2. Amazon Machine Image (AMI)

- AMI: Ubuntu Server 20.04 LTS (Focal Fossa)

3. Security Groups

Security groups act as a virtual firewall for your instance to control inbound and outbound traffic.



Inbound rules (11)

Manage tags | Edit inbound rules

Search

1

Name	Security group rule...	IP version	Type	Protocol	Port range
–	sgr-0d37abdd416f485a7	IPv4	Custom TCP	TCP	8080
–	sgr-05039c761a83a472f	IPv4	Custom TCP	TCP	3000
–	sgr-0c8a37b32250d40...	IPv4	Custom TCP	TCP	9090
–	sgr-01c22c57e1cae5357	IPv4	Custom TCP	TCP	9000
–	sgr-05147c8d7f990658b	IPv4	Custom TCP	TCP	32630
–	sgr-08e5d857074de8...	IPv4	SSH	TCP	22
–	sgr-03ba7f02232c511d5	IPv4	Custom TCP	TCP	8081
–	sgr-03f10a0d8f689f506	IPv4	Custom TCP	TCP	6443
–	sgr-053a7766d864dd...	IPv4	SMTPS	TCP	465
–	sgr-0c96dd75b35d3f40c	IPv4	HTTP	TCP	80
–	sgr-0aae64d181f9a5a99	IPv4	Custom TCP	TCP	9115

After Launching your Virtual machine, SSH into the Server.

Step 3 .Setup EKS Cluster Using terraform

Create a Virtual Machine on AWS

SSH into the VM and Run the command to install Terraform
`sudo snap install terraform --classic`

AWSCLI

Download AWS CLI on VM

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"

sudo apt install unzip

unzip awscliv2.zip

sudo ./aws/install

aws configure
```

KUBECTL

```
curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.19.6/2021-01-05/bin/linux/amd64/kubectl

chmod +x ./kubectl

sudo mv ./kubectl /usr/local/bin

kubectl version --short --client
```

EKSCTL

```
curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp

sudo mv /tmp/eksctl /usr/local/bin

eksctl version
```

Save all the script in a file, for example, ctl.sh, and make it executable

using:

```
chmod +x ctl.sh
```

Create Terraform files

1.main.tf

2.output.tf

3.variable.tf

Terraform Code -> Click here

Run the command

```
terraform init
```

```
terraform plan
```

```
terraform apply -auto approve
```

Create EKS Cluster

```
eksctl create cluster --name=EKS-1 \
                --region=ap-south-1 \
                --zones=ap-south-1a,ap-south-1b \
                --without-nodegroup
```

Open ID Connect

```
eksctl utils associate-iam-oidc-provider \
  --region ap-south-1 \
  --cluster EKS-1 \
  --approve
```

Create node Group

```
eksctl create nodegroup --cluster=EKS-1 \
                --region=ap-south-1 \
                --name=node2 \
                --node-type=t3.medium \
                --nodes=3 \
                --nodes-min=2 \
                --nodes-max=4 \
                --node-volume-size=20 \
                --ssh-access \
                --ssh-public-key=DevOps \
                --managed \
                --asg-access \
                --external-dns-access \
                --full-ecr-access \
                --appmesh-access \
                --alb-ingress-access
```

Make sure to change the name of ssh-public-Key with your SSH key.

Step 4. Installing Jenkins on Ubuntu

Execute these commands on Jenkins Server

```
#!/bin/bash
# Install OpenJDK 17 JRE Headless
sudo apt install openjdk-17-jre-headless -y
# Download Jenkins GPG key
sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
# Add Jenkins repository to package manager sources
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
/etc/apt/sources.list.d/jenkins.list > /dev/null
# Update package manager repositories
sudo apt-get update
# Install Jenkins
sudo apt-get install jenkins -y
```

Save this script in a file, for example, install_jenkins.sh, and make it executable using:

```
chmod +x install_jenkins.sh
```

Then, you can run the script using:

```
./install_jenkins.sh
```

This script will automate the installation process of OpenJDK 17 JRE Headless and Jenkins.

Install docker for future use

Execute these commands on Jenkins, SonarQube and Nexus Servers

```
#!/bin/bash
# Update package manager repositories
sudo apt-get update
# Install necessary dependencies
sudo apt-get install -y ca-certificates curl
# Create directory for Docker GPG key
sudo install -m 0755 -d /etc/apt/keyrings
# Download Docker's GPG key
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o
/etc/apt/keyrings/docker.asc
# Ensure proper permissions for the key
sudo chmod a+r /etc/apt/keyrings/docker.asc
# Add Docker repository to Apt sources
echo "deb [arch=$(dpkg --print-architecture) signed
by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu
\
$(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
# Update package manager repositories
sudo apt-get update
sudo apt-get install -y docker-ce docker-ce-cli containerd.io docker-buildx-
plugin
docker-compose-plugin
```

Save this script in a file, for example, install_docker.sh, and make it executable using:

```
chmod +x install_docker.sh
```

Then, you can run the script using:

```
./install_docker.sh
```

SetUp Nexus

Execute these commands on Nexues VM

```
#!/bin/bash
# Update package manager repositories
sudo apt-get update
# Install necessary dependencies
sudo apt-get install -y ca-certificates curl
# Create directory for Docker GPG key
sudo install -m 0755 -d /etc/apt/keyrings
# Download Docker's GPG key
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o
/etc/apt/keyrings/docker.asc
# Ensure proper permissions for the key
sudo chmod a+r /etc/apt/keyrings/docker.asc
# Add Docker repository to Apt sources
echo "deb [arch=$(dpkg --print-architecture) signed
by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu
\
$(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
# Update package manager repositories
sudo apt-get update
sudo apt-get install -y docker-ce docker-ce-cli containerd.io docker-buildx-
plugin
docker-compose-plugin
```

Save this script in a file, for example, install_docker.sh, and make it executable using:

```
chmod +x install_docker.sh
```

Then, you can run the script using:

```
./install_docker.sh
```

Create Nexus using docker container

To create a Docker container running Nexus 3 and exposing it on port 8081, you can
use the following command:

```
docker run -d --name nexus -p 8081:8081 sonatype/nexus3:latest
```

This command does the following:

• -d: Detaches the container and runs it in the background.
• --name nexus: Specifies the name of the container as "nexus".
• -p 8081:8081: Maps port 8081 on the host to port 8081 on the container, allowing
access to Nexus through port 8081.
• sonatype/nexus3:latest: Specifies the Docker image to use for the container, in this
case, the latest version of Nexus 3 from the Sonatype repository.

After running this command, Nexus will be accessible on your host machine at http://IP:8081.

Get Nexus initial password

Your provided commands are correct for accessing the Nexus password stored in the
container. Here's a breakdown of the steps:

1. **Get Container ID**: You need to find out the ID of the Nexus container. You can do this by running:

```
docker ps
```

This command lists all running containers along with their IDs, among other information.

2. **Access Container's Bash Shell**: Once you have the container ID, you can execute the docker exec command to access the container's bash shell:

```
docker exec -it <container_ID> /bin/bash
```

Replace **<container_ID>** with the actual ID of the Nexus container.

3. **Navigate to Nexus Directory**: Inside the container's bash shell, navigate to the directory where Nexus stores its configuration:

```
cd sonatype-work/nexus3
```

4. **View Admin Password**: Finally, you can view the admin password by displaying the contents of the admin.password file:

```
cat admin.password
```

5. **Exit the Container Shell**: Once you have retrieved the password, you can exit the container's bash shell:

```
exit
```

This process allows you to access the Nexus admin password stored within the container.

SetUp SonarQube

Execute these commands on SonarQube VM

```
#!/bin/bash
# Update package manager repositories
sudo apt-get update
# Install necessary dependencies
sudo apt-get install -y ca-certificates curl
# Create directory for Docker GPG key
sudo install -m 0755 -d /etc/apt/keyrings
# Download Docker's GPG key
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o
/etc/apt/keyrings/docker.asc
# Ensure proper permissions for the key
sudo chmod a+r /etc/apt/keyrings/docker.asc
# Add Docker repository to Apt sources
echo "deb [arch=$(dpkg --print-architecture) signed
by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu
\
$(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
# Update package manager repositories
sudo apt-get update
sudo apt-get install -y docker-ce docker-ce-cli containerd.io docker-buildx-
plugin
docker-compose-plugin
```

Save this script in a file, for example, install_docker.sh, and make it executable using:

```
chmod +x install_docker.sh
```

Then, you can run the script using:

```
./install_docker.sh
```

Create Sonarqube Docker container

To run SonarQube in a Docker container with the provided command, you can follow
these steps:
1. Open your terminal or command prompt.
2. Run the following command:

```
docker run -d --name sonar -p 9000:9000 sonarqube:lts-community
```

This command will download the sonarqube:lts-community Docker image from Docker
Hub if it's not already available locally. Then, it will create a container named "sonar"
from this image, running it in detached mode (-d flag) and mapping port 9000 on the
host machine to port 9000 in the container (-p 9000:9000 flag).
3. Access SonarQube by opening a web browser and navigating to http://VmIP:9000.
This will start the SonarQube server, and you should be able to access it using the
provided URL. If you're running Docker on a remote server or a different port, replace localhost with the appropriate hostname or IP address and adjust the port

Step 5. Install and Configur Plugins on Jenkins

Plugins:

1. SonarQube Scanner
2. Config file provider
3. Maven Integration
4. Pipeline maven integration.
5. Kubernetes
6. Kubernetes Client API
7. Kubernetes Credentials
8. Kubernetes CLI
9. Docker
10. Docker Pipeline
11. Pipeline Stage View

Configuration part Click her- > Video

Create Docker Credentials:

-Go to Manage Jenkins > Manage Credentials > (global) > Add Credentials.

-Choose Username with password as the kind.

-ID: docker-cred

-Username: Your Docker Hub username.

-Password: Your Docker Hub password.

-Click OK.

Create GitHub Credentials:

-Go to Manage Jenkins > Manage Credentials > (global) > Add Credentials.

-Choose Secret text as the kind.

-ID: git-cred

-Secret: Your GitHub Personal Access Token.

-Click OK.

Step 6. Creating Pipeline

Highly Recommended to not copy & write instead after watching the YT video at https://www.youtube.com/watch?v=pFzzWH3w03U

This will help you learn how to write the Jenkinsfile.

Continuous Deployment

Create Service Account, Role & Assign that role, And create a secret for Service Account and generate a Token. We will Deploy our Application on the main branch .

Create a file : Vim svc.yml

Creating Service Account

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: jenkins
  namespace: webapps
```

To run the svc.yml : kubectl apply -f svc.yaml

Similarly create a role.yml file

Create Role

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: app-role
  namespace: webapps
rules:
  - apiGroups:
      - ""
      - apps
```

```
    - autoscaling
    - batch
    - extensions
    - policy
    - rbac.authorization.k8s.io
  resources:
    - pods
    - componentstatuses
    - configmaps
    - daemonsets
    - deployments
    - events
    - endpoints
    - horizontalpodautoscalers
    - ingress
    - jobs
    - limitranges
    - namespaces
    - nodes
    - pods
    - persistentvolumes
    - persistentvolumeclaims
    - resourcequotas
    - replicasets
    - replicationcontrollers
    - serviceaccounts
```

```
    - services
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
```

To run the role.yaml file: kubectl apply -f role.yaml

Create Bind

Similarly create a bind.yml file

Bind the role to service account

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: app-rolebinding
  namespace: webapps
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: app-role
subjects:
- namespace: webapps
  kind: ServiceAccount
  name: jenkins
```

To run the bind.yaml file: kubectl apply -f bind.yaml

Create Token

Similarly create a secret.yml file

```
apiVersion: v1
kind: Secret
```

```
type: kubernetes.io/service-account-token
metadata:
 name: mysecretname
 annotations:
   kubernetes.io/service-account.name: Jenkins
```

Kubernates Secret Docker

```
kubectl create secret docker-registry regcred \
  --docker-server=https://index.docker.io/v1/ \
  --docker-username=adijaiswal \
  --docker-password=XYZ@123 \
        --namespace=webapps
```

Now Run – `kubectl describe secret mysecretname -n webapps`

Save the Token.

-Create a dummy job in your Jenkins with Pipeline job and go to the **pipeline syntax** and select **With Kubernetes:Configure Kubernetes**

1. **Credentials** – Provide the Token that you have saved .
2. **Kubernates Endpoint API**- You can find it in your AWS EKS cluster.
3. **Cluster name**- Provide any name.
4. **NameSpace** – webapps

Click on Generate Syntax.

You will get pipeline syntax :-

```
withKubeCredentials(kubectlCredentials: [[caCertificate: '', clusterName: 'EKS-1', contextName: '', credentialsId: 'k8-token', namespace: 'webapps', serverUrl: 'https://B7C7C20487B2624AAB0AD54DF1469566.yl4.ap-south-1.eks.amazonaws.com']]) {
//block of code
```

```
}
```

Step 7. DNS Mapping with GoDaddy Domain

To map a load balancer domain with a GoDaddy domain, you'll need to follow these steps:

1. Get the Load Balancer DNS Name

- If you're using AWS, go to the EC2 Dashboard > Load Balancers.
- Select your load balancer and find the DNS name under the Description tab.
- Copy this DNS name as you'll need it when configuring your GoDaddy domain.

2. Log In to GoDaddy

- Go to the GoDaddy website and log in to your account.

3. Access Domain Settings

- Once logged in, go to the Domains section and select the domain name you want to map to your load balancer.
- Click on the domain name to access its settings.

4. Manage DNS Settings

- Scroll down to the DNS Settings section.
- Click on Manage DNS to open the DNS management page.

5. Add a CNAME Record

- In the DNS management page, under the Records section, click on Add to create a new DNS record.
- Choose CNAME as the type of record.
- In the Host field, enter the subdomain you want to use (e.g., www or app).
- In the Points to field, paste the DNS name of your load balancer that you copied earlier.
- Set the TTL (Time To Live) value (the default is usually fine).
- Click Save to apply the changes.

6. Update the A Record (Optional)

- If you want to map the root domain (e.g., example.com), you may need to update the A Record.
- Instead of a CNAME, add or modify the existing A Record to point to the IP address of your load balancer (if available) or use a service like AWS Route 53 to map the root domain to the load balancer.

7. Wait for DNS Propagation

- DNS changes can take some time to propagate, usually within a few minutes to 24 hours.
- You can use tools like What's My DNS to check if the changes have propagated globally.

8. Test the Setup

- Once DNS propagation is complete, you should be able to access your application using your GoDaddy domain name mapped to the load balancer.

Step 8. Add SSL Certificate

1. Sign Up for Cloudflare and Add Your Domain

- Sign up at cloudflare.com if you don't have an account.
- After logging in, click Add a Site and enter your domain name (e.g., example.com).
- Cloudflare will scan your DNS records. Review them for accuracy, then click Continue.

2. Update Your DNS Nameservers

- Cloudflare will provide new nameservers. Go to your domain registrar (e.g., GoDaddy) and update your domain's nameservers to the ones provided by Cloudflare.
- The nameserver change may take a few minutes to 24 hours to propagate.

3. Configure SSL/TLS Settings in Cloudflare

- Once your domain is active on Cloudflare, navigate to the SSL/TLS tab in your Cloudflare dashboard.

Choose the SSL/TLS mode:

Flexible: Encrypts traffic between Cloudflare and the browser, but not between Cloudflare and your server.

Full: Encrypts traffic between both the browser and Cloudflare, and Cloudflare and your server. Your server must have an SSL certificate.

Full (Strict): Same as Full, but requires a valid SSL certificate on your server.

4. Test the SSL Setup

Visit your site using https:// to ensure the SSL certificate is working correctly.

5. Force HTTPS (Optional)

In the SSL/TLS tab on Cloudflare, enable Always Use HTTPS and Automatic HTTPS Rewrites to ensure all traffic to your site is securely encrypted.

Conclusion

By following the steps outlined in this guide, you've successfully set up a robust CI/CD pipeline for your three-tier Java application with a MySQL database, utilizing AWS, Jenkins, Docker, Kubernetes, and other essential tools. The implementation of SSL certificates via Cloudflare ensures secure communication, enhancing the overall security of your application. This setup not only automates the deployment process but also integrates quality assurance measures, artifact management, and secure deployment practices. With this pipeline in place, you can confidently push code changes, knowing they will be automatically tested, deployed, and monitored, ensuring consistent and reliable application performance in production.