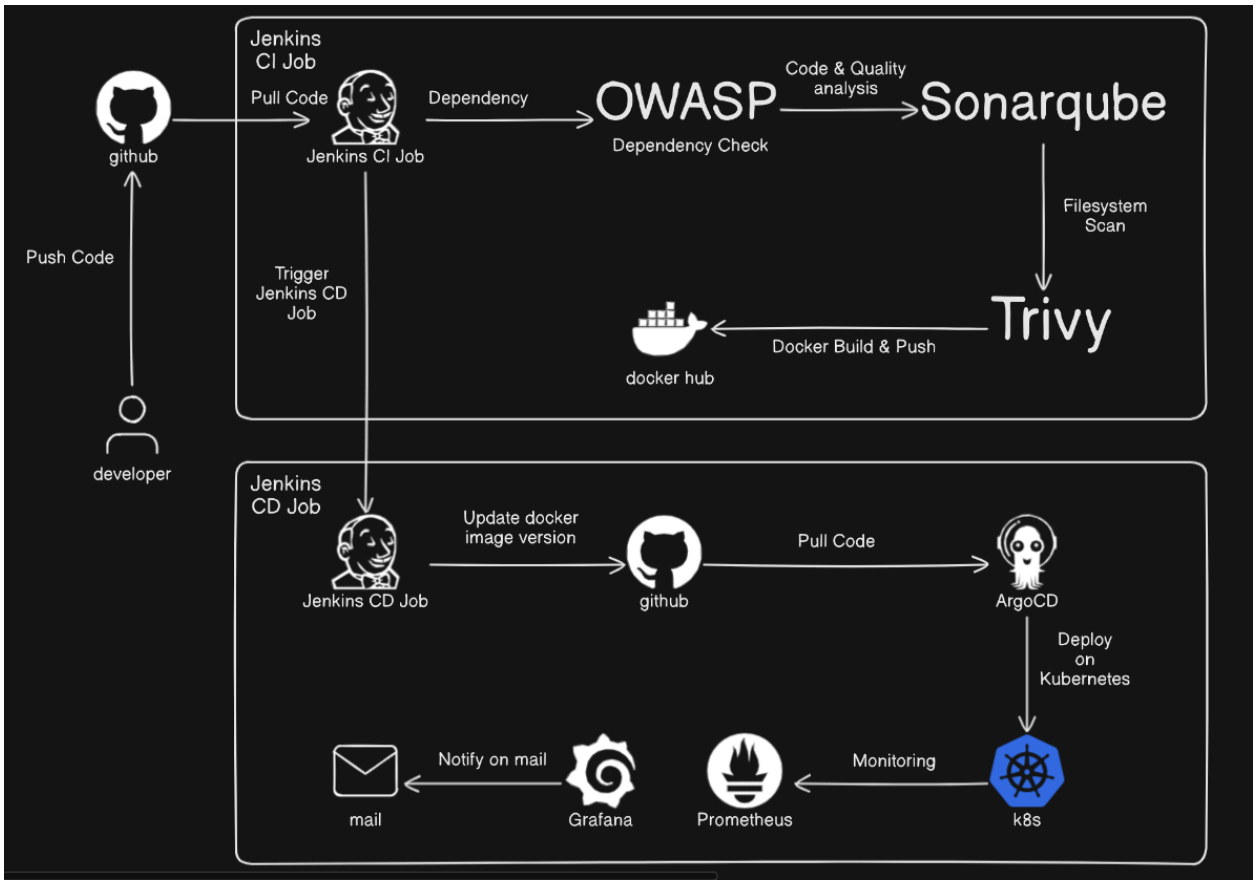
BulipeTech Demo Project

Summery

In this project, we are using **React** for the frontend, **Node.js** for the backend, and **MongoDB** for the database and the following services.

- ➔ **Cloud Provider:** AWS
- ➔ **Infrastructure as Code:** Terraform
- ➔ **Containerization:** Docker
- ➔ **Orchestration:** K8s
- ➔ **CI/CD Pipeline:** jenkins
- ➔ **K8s Cluster Deployment:** ArgoCD
- ➔ **Monitoring:** Prometheus & Grafana

Project Deployment flow:



Tech stack used in this project:

- GitHub (Code)
- Docker (Containerization)
- Jenkins (CI)
- OWASP (Dependency check)
- SonarQube (Quality)
- Trivy (Filesystem Scan)
- ArgoCD (CD)
- Redis (Caching)
- AWS EKS (Kubernetes)
- Helm (Monitoring using grafana and prometheus)

Infrastructure setup (Terraform):

Use Terraform to create a Jenkins EC2 server on AWS. The Terraform files are located in the terraform directory.

Configure Jenkins Server:

Install & Configure Docker:

```
sudo apt-get update
sudo apt-get install docker.io -y
sudo usermod -aG docker ubuntu && newgrp docker
```

Install and configure Jenkins:

```
sudo apt update -y
sudo apt install fontconfig openjdk-17-jre -y

sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key

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

sudo apt-get update -y
sudo apt-get install jenkins -y
```

Create EKS Cluster on AWS:

IAM user with access keys and secret access keys
AWSCLI should be configured (Setup AWSCLI)

```
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
```

Install 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
```

Install 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
```

Create EKS Cluster

```
eksctl create cluster --name=wanderlust \
                      --region=us-east-2 \
                      --version=1.30 \
                      --without-nodegroup
```

Associate IAM OIDC Provider

```
eksctl utils associate-iam-oidc-provider \
  --region us-east-2 \
  --cluster wanderlust \
  --approve
```

Create Nodegroup

```
eksctl create nodegroup --cluster=wanderlust \
                       --region=us-east-2 \
                       --name=wanderlust \
                       --node-type=t2.large \
                       --nodes=2 \
                       --nodes-min=2 \
                       --nodes-max=2 \
                       --node-volume-size=29 \
```

```
                        --ssh-access \
                        --ssh-public-key=eks-nodegroup-key
```

Install and configure SonarQube

```
docker run -itd --name SonarQube-Server -p 9000:9000
sonarqube:lts-community
```

Install Trivy

```
sudo apt-get install wget apt-transport-https gnupg lsb-release -y
wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key | sudo
apt-key add -
echo deb https://aquasecurity.github.io/trivy-repo/deb $(lsb_release -sc)
main | sudo tee -a /etc/apt/sources.list.d/trivy.list
sudo apt-get update -y
sudo apt-get install trivy -y
```

Install and Configure ArgoCD

```
kubectl create namespace argocd
kubectl apply -n argocd -f
https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install
.yaml
```

- Make sure all pods are running in argocd namespace

```
watch kubectl get pods -n argocd
```

- Access ArgoCD on browser

```
<public-ip>:31000
```

Monitor EKS cluster, kubernetes components and workloads using prometheus and grafana via HELM

- Install Helm Chart

```
curl -fsSL -o get_helm.sh
https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
chmod 700 get_helm.sh
./get_helm.sh
```

- Add Helm Stable Charts for Your Local Client

```
helm repo add stable https://charts.helm.sh/stable
```

- Add Prometheus Helm Repository

```
helm repo add prometheus-community
https://prometheus-community.github.io/helm-charts
```

- Create Prometheus Namespace

```
kubectl create namespace prometheus
kubectl get ns
```

- Install Prometheus using Helm

```
helm install stable prometheus-community/kube-prometheus-stack -n prometheus
```

- Verify prometheus installation

```
kubectl get pods -n prometheus
```

- Check the services file (svc) of the Prometheus

```
kubectl get svc -n prometheus
```

- Access Prometheus & Grafana on browser

```
Prometheus: <public-ip>:31001
Grafana: <public-ip>:31002
```

Deployment Process:

CI/CD Workflow for Frontend & Backend Releases

1. **Code Update & CI Pipeline Trigger**
    - When a new version of the **frontend** or **backend** is released (pushed to Git), the **CI pipeline** (Jenkins) is triggered automatically.
2. **CI Pipeline Steps (Jenkins)**
    - Pull the latest code from the repository.
    - Build the Docker image with the **updated version**.
    - Tag the image with the new version (e.g., `v2.3`).
    - Push the updated image to **Docker Hub**.
3. **Triggering the CD Pipeline**
    - Once the CI pipeline completes successfully, the **CD pipeline** (GitOps Jenkinsfile) starts.
    - It updates the **Kubernetes manifest files** in the GitOps repository with the **new image version** for frontend or backend.
4. **ArgoCD Deployment**
    - ArgoCD detects changes in the GitOps repository (updated manifest files).
    - It automatically syncs the **new image version** to the **EKS cluster**.
    - The frontend or backend pods are updated with the **latest version**.

Git Repository Structure:

1. **Backend**
   - Contains the backend source code.

2. **Frontend**
   - Contains the frontend source code.

3. **Database**
   - Contains the Dockerfile for building the database.

4. **Kubernetes**
   - Contains Kubernetes manifest files for backend, frontend, and MongoDB.

5. **Terraform**
   - Contains Terraform configurations for automating the creation of an AWS EC2 instance for the Jenkins server.

6. **Jenkinsfile**
   - Defines the CI pipeline for the project.

7. **GitOps**
   - Contains the Jenkinsfile used for the CD pipeline.

Project repo: https://github.com/mazibinfo/Bulipetech-Project.git
Jenkins Shared Library repo: https://github.com/mazibinfo/Jenkins_SharedLib.git