**Proposed Documentation of Automated Infrastructure Deployment Using Terraform, AWS, Jenkins, and Ansible**

**1. Project Overview**

* **Project Title:** Automated Infrastructure Deployment Using Terraform, AWS, Jenkins, and Ansible
* **Objective:** The goal of this project is to automate the deployment of a secure, scalable, and reliable cloud infrastructure on AWS, utilizing modern DevOps tools such as Terraform, Jenkins, and Ansible. The infrastructure will support a web application hosted on EC2 instances, made accessible through a reverse proxy, and managed using Infrastructure as Code (IaC) principles.
* **Scope:** The project includes:

• Provisioning AWS resources using Terraform, such as Virtual Private Cloud (VPC), EC2 instances (public and private), security groups, and an S3 bucket.

• Setting up a bastion host (jumphost) to access private instances securely via SSH.

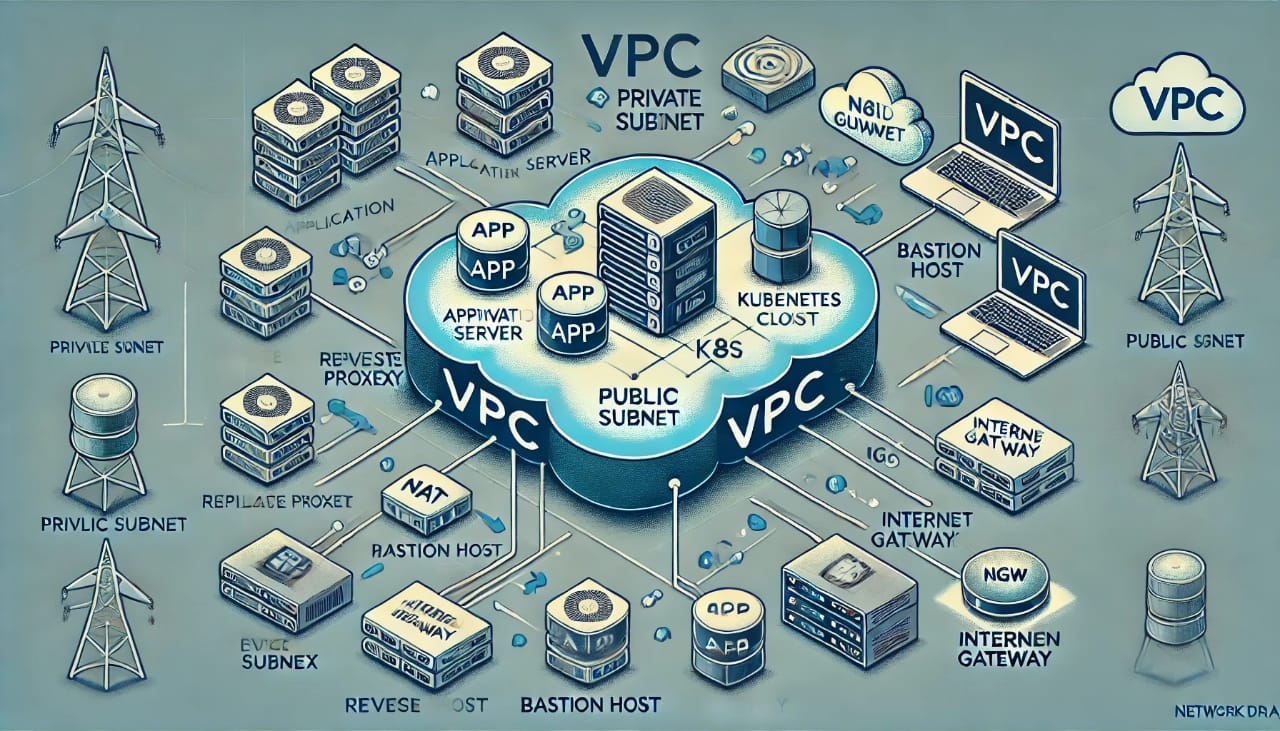
• Configuring a reverse proxy on the private EC2 instance to make the hosted application accessible to the public.

• Using Jenkins for CI/CD to automate the infrastructure deployment and ensure consistent environments.

• Leveraging Ansible to automate software installation and configuration on the EC2 instances.

• Storing Terraform state files in an S3 bucket for collaborative work and version control.

**2. Architecture Diagram:**



**3**. **Introduction**

The shift to cloud-based infrastructure has transformed how modern applications are built and managed. To keep up with these advancements, automating infrastructure deployment and management has become a crucial part of cloud architecture. This project aims to demonstrate a fully automated infrastructure setup using Terraform for provisioning, Jenkins for continuous integration and deployment (CI/CD), and Ansible for configuration management.

**4. Architecture Overview**

**4.1 AWS Resources**

The infrastructure is hosted on AWS and includes:

• VPC: Custom VPC with public and private subnets.

• EC2 Instances: One public (bastion) and one private instance.

• NAT Gateway: Ensures private instances can access the internet securely for software updates.

• S3 Bucket: Stores Terraform state files for centralized access.

**4.2. Security**

• Jumphost (Bastion): Acts as a secure gateway to access private instances.

• Security Groups: Enforce restricted access to instances based on ports and IP ranges.

**5. Toolchain Overview**

**5.1. Terraform**

• Used for Infrastructure as Code (IaC) to provision AWS resources.

• Key Benefits:

• Modular and Scalable: Facilitates infrastructure reuse across environments.

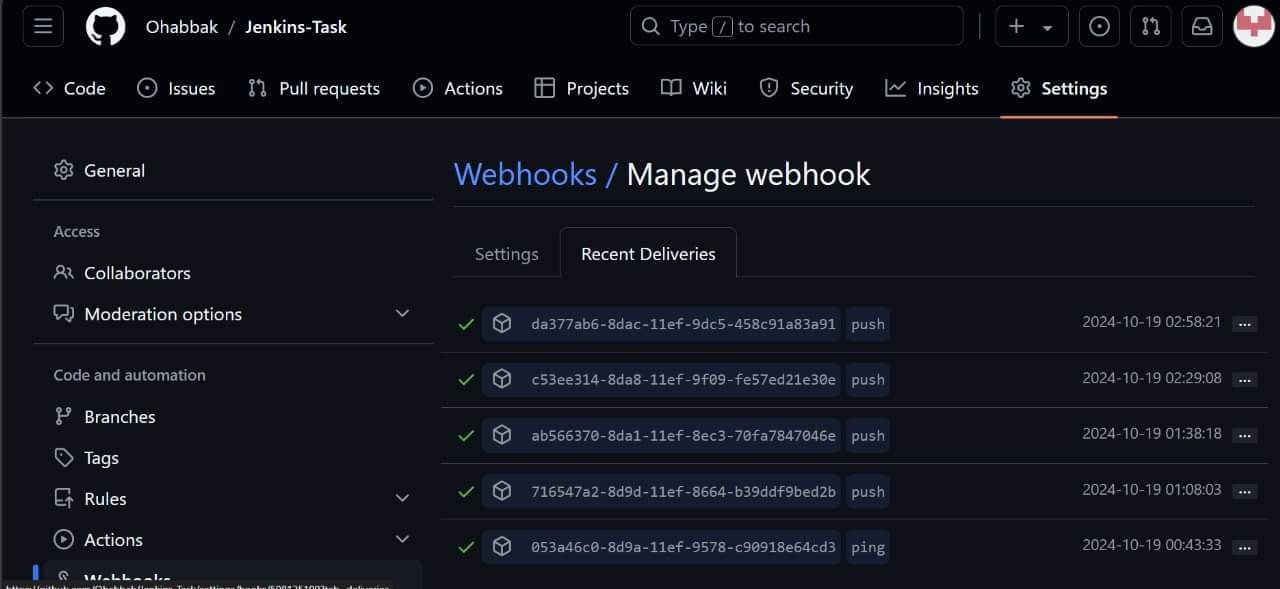
• State Management: State files are stored in S3 for team collaboration.

**5.2. Jenkins**

• Integrates with GitHub to trigger pipelines that automate Terraform deployment.

• Supports automated tests and validation for infrastructure integrity.





**5.3. Ansible**

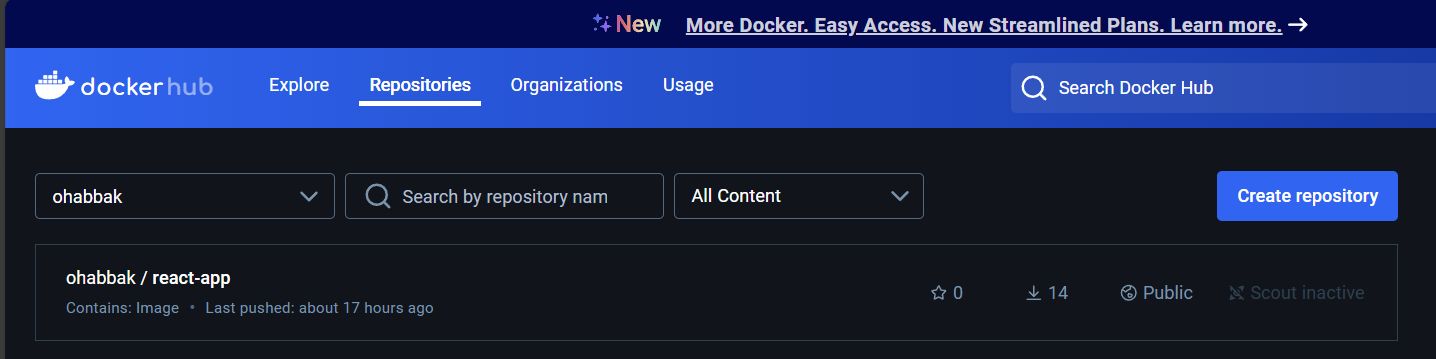
• Handles the configuration of EC2 instances, including:

• Installing necessary software (e.g., Nginx for reverse proxy).

• Configuring security settings and environment variables.

#### **5.4 Docker Hub :**

The Docker image of the application is built and pushed to Docker Hub, allowing for easy deployment and integration with other tools like Jenkins and Ansible.



**6. Workflow**

**1. Terraform Deployment:**

Terraform scripts are executed through Jenkins to create the VPC, EC2 instances, security groups, NAT gateway, and S3 bucket.

**2. Jenkins Pipeline:**

Jenkins is configured to monitor changes in the repository. On push, the pipeline initiates Terraform to provision or update infrastructure.

**3. Ansible Configuration:**

Once the infrastructure is up, Ansible is used to configure the EC2 instances. This includes setting up the necessary software, dependencies, and security configurations.

**4. Reverse Proxy Setup:**

The private EC2 instance is configured as a web server, with Nginx acting as a reverse proxy to handle requests from the public.

**5. Webhook Integration:**

To automate the continuous integration and deployment (CI/CD) process, a webhook service was implemented. The webhook is a crucial part of the project as it enables automatic triggering of the Jenkins pipeline whenever changes are made to the code repository.

**7. Slack Integration:**

To enhance communication and provide real-time updates on the CI/CD pipeline, Slack integration was implemented in the project. This integration ensures that the development team is promptly notified about the status of the deployment process, including successful builds, failures, and other relevant events.

**7. Key Challenges and Solutions**

1.  **Challenge:** Securing private EC2 instances.

* **Solution:** Implemented a bastion host to restrict access and used security groups for strict ingress/egress rules

2.  **Challenge:** Managing Terraform state files.

* **Solution:** Stored state files in an S3 bucket with versioning for collaboration and state integrity.

3. **Challenge:** Automating configuration on EC2 instances.

* **Solution:** Integrated Ansible to automatically configure and manage software on the instances.

**8. Project Outcomes**

By the end of this project, we will have:

• A fully automated and scalable cloud infrastructure that supports a web application.

• An efficient CI/CD pipeline that ensures rapid deployment and updates.

• A secure, accessible setup with a reverse proxy for external access.

**9. Future Enhancements**

• Auto-scaling: Enable auto-scaling groups to handle traffic spikes automatically.

**Proposed Documentation of Saay Application Deployment Pipeline :**

**1. Project Overview:**

* **Project Title:**  Saay Application Deployment Pipeline
* **Objective:** The main objective of this project is to automate the deployment of the Saay Application by setting up a CI/CD pipeline using Jenkins, Docker, and Kubernetes. The pipeline is designed to fetch the latest code from GitHub, build and push Docker images, and deploy the application in an Amazon EKS cluster.
* **Project Scope:** Set up an Amazon EKS cluster with one node group.
* Create a Jenkins instance for automating deployment tasks.
* Implement a pipeline to build, push Docker images, and deploy services.
* Automate the deployment of frontend (with LoadBalancer), backend (with ClusterIP), and database (with ClusterIP) services using Kubernetes.

**2. Introduction:**

This project focuses on building an automated pipeline for deploying a microservices-based application into a Kubernetes cluster (Amazon EKS). The setup leverages AWS infrastructure for scalability, Jenkins for continuous integration, Docker for containerization, and Kubernetes for orchestration. This ensures the efficient deployment of the frontend, backend, and database components of the Saay application.

**3. Architecture Overview:**

**3.1 AWS Resources**

EKS Cluster: A single Amazon EKS cluster with one node group to run the application services.

Jenkins Instance: A separate EC2 instance configured with Docker and Jenkins to execute the CI/CD pipeline.

Jenkins Instance Configuration:

#!/bin/bash

sudo apt-get update -y

sudo apt-get install -y openjdk-11-jdk

# Install Jenkins

curl -fsSL <https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key> | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null

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 -y jenkins

# install docker

sudo apt install -y apt-transport-https ca-certificates curl software-properties-common

curl -fsSL <https://download.docker.com/linux/ubuntu/gpg> | sudo apt-key add -

sudo add-apt-repository "deb [arch=amd64] <https://download.docker.com/linux/ubuntu> $(lsb\_release -cs) stable"

sudo apt update -y

sudo apt install -y docker-ce

sudo usermod -aG docker ubuntu

sudo usermod -aG docker jenkins

# Start and enable docker

sudo systemctl start docker

sudo systemctl enable docker

# Start and enable Jenkins

sudo systemctl start jenkins

sudo systemctl enable jenkins

#install kubectl

sudo curl -LO "<https://dl.k8s.io/release/$(curl> -L -s <https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl>"

sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

#install awsctl

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

sudo apt-get install unzip -y

unzip awscliv2.zip

sudo ./aws/install

VPC: A VPC created for the EKS cluster with subnets and proper networking configuration.

**3.2 Security**

Jenkins and Ubuntu users have been granted necessary IAM permissions to access the EKS cluster.

EKS security group ensures secure communication between frontend, backend, and database services within the cluster.

Secure authentication mechanisms are implemented for Docker Hub access and AWS resources.

**4. Toolchain Overview:**

4.1 Terraform

Terraform was used to provision the necessary AWS infrastructure, including the VPC, EKS cluster, and node groups.

4.2 Jenkins

Jenkins is responsible for the continuous integration and deployment of the application.

A GitHub webhook triggers the pipeline, which fetches code, builds Docker images, and deploys services to the Kubernetes cluster.

4.3 EKS

Amazon EKS is used for managing the containerized application services.

Frontend is deployed with a LoadBalancer service, while backend and database components use ClusterIP services for internal communication.

**5. Workflow:**

Code Fetching: Jenkins pipeline is triggered by a GitHub webhook upon every push to the repository.

Build Stage: Dockerfiles are built for the frontend, backend, and database components.

Push to Docker Hub: Jenkins logs into Docker Hub and pushes the newly built images.

Deploy Stage: Jenkins applies Kubernetes configurations to deploy the frontend (LoadBalancer), backend (ClusterIP), and database (ClusterIP) services.

6. **Project Outcomes:**

Automated deployment of the Saay application, ensuring that every code update from GitHub is built, pushed, and deployed seamlessly.

Efficient separation of frontend, backend, and database services within a Kubernetes cluster.

Improved development workflow with automated CI/CD practices using Jenkins.