Implementation Report

EKS with Terraform and Github Actions CI/CD for Java Web App Deployment

This report outlines the approach taken to deploy a Java web application on Amazon EKS (Elastic Kubernetes Service) using Terraform for infrastructure provisioning and Github Actions for CI/CD automation.

Approach

1. **Infrastructure as Code (IaC):** Terraform scripts define the EKS cluster configuration, including worker node instances, security settings, and network configurations. These scripts ensure a repeatable and version-controlled infrastructure deployment.
2. **Containerization:** The Java web application is containerized using Docker, creating a portable and isolated environment for deployment.
3. **CI/CD Pipeline:** Github Actions workflows are triggered by code changes in the Git repository. These workflows perform the following tasks:
    - Performing Unit Testing, Code Coverage, Dependency Scanning, and Security Vulnerability Scanning.
    - Build the Java application and create a Docker image.
    - Push the Docker image to a container registry (Docker hub)
    - Deploy the application using Kubernetes manifests newly provisioned EKS Cluster.
4. **Deployment Strategy:** Kubernetes deployments manage the application lifecycle within the EKS cluster, ensuring a desired number of application pods are running at all times.

Choices made during EKS Terraform implementation:

- **Networking:** Defining the VPC configuration and Kubernetes cluster network for pod communication.
- **Node instance type:** Selection of EC2 instance type for worker nodes based on application resource requirements (CPU, memory). Selected minimum requirements for the POC.
- **Security groups:** Configuration of security groups to control inbound and outbound traffic for the EKS cluster and application pods.
- **Terraform:** Provides a declarative approach for managing EKS infrastructure, simplifying configuration and version control.
- **GitHub Actions:** Offers a built-in CI/CD solution within the GitHub ecosystem, promoting seamless integration with development workflows. This is a good choice for organizations using a full-fledged GitHub instance.
- **Docker:** Enables containerization of the application, facilitating portability and consistent deployment across environments.
- **AWS Load Balancer Controller:** In Kubernetes, Ingress efficiently manages external traffic. It directs requests based on factors like paths and domains, all while consolidating endpoint management into a single, internal resource.

Challenges Faced:

- AWS Load Balancer Controller setup**:** Installing the ALB Ingress Controller with Helm presented several version-related challenges. These were resolved through research.
- Intermittently, the application JAR file is not getting copied into the Docker image during the build process, hence the image is not generating correctly.
- Error Handling: Implementing robust error handling and rollback mechanisms within the CI/CD pipeline to gracefully handle deployment failures.

Conclusion

This approach leverages Terraform for automated EKS cluster provisioning and Github Actions for CI/CD automation, enabling efficient and reliable deployments of Java web applications on AWS. By addressing the potential challenges mentioned, a robust and scalable deployment pipeline can be established.