**Project Guide for a DevOps-Driven Web Application**

**Project Overview**

This project guide walks you through creating a scalable web application with DevOps practices for CI/CD, monitoring, and automated infrastructure management. The tech stack includes a Node.js application deployed on AWS using Docker and Kubernetes, with Jenkins for CI/CD.

**Table of Contents**

1. **Project Requirements**
2. **Setting Up the Infrastructure**
3. **CI/CD Pipeline with Jenkins**
4. **Application Deployment Using Docker and Kubernetes**
5. **Monitoring and Logging**
6. **Scaling and Load Balancing**
7. **Security and Best Practices**

**1. Project Requirements**

* **Technologies**: AWS, Docker, Kubernetes, Jenkins, Node.js (or your web framework of choice)
* **Tools**: Terraform, Helm (for Kubernetes), Prometheus and Grafana (for monitoring), NGINX or ELB (for load balancing)
* **Basic knowledge**: Understanding of Linux, cloud services, containerization, and CI/CD principles

**2. Setting Up the Infrastructure**

**a. Provisioning with Terraform**

1. **Install Terraform** and configure it to work with your AWS account.
2. Create a **Terraform configuration file** (main.tf) to define the resources:
   * Define the VPC, subnets, security groups, EC2 instances, ECR (Elastic Container Registry), and RDS (if needed for databases).
3. Initialize and apply the configuration:

bash

Copy code

terraform init

terraform apply

**b. Configuring Kubernetes Cluster**

* Set up an EKS (Elastic Kubernetes Service) cluster on AWS via Terraform or directly from the AWS Management Console.
* Install **kubectl** and **aws-iam-authenticator** to manage the EKS cluster.
* Configure kubectl to connect to your cluster:

bash

Copy code

aws eks --region <region> update-kubeconfig --name <cluster\_name>

**3. CI/CD Pipeline with Jenkins**

**a. Set Up Jenkins**

* Deploy Jenkins on an EC2 instance or as a container in Kubernetes.
* Configure Jenkins with required plugins: Docker, Kubernetes, Git, and Slack (for notifications).

**b. Creating the Pipeline**

* Write a Jenkinsfile for your project to define stages:
  1. **Clone Code** from the Git repository.
  2. **Build Docker Image**: Use a Dockerfile to build the application image.
  3. **Push to ECR**: Push the Docker image to the Elastic Container Registry.
  4. **Deploy to Kubernetes**: Use kubectl commands or Helm to deploy the application.

**4. Application Deployment Using Docker and Kubernetes**

**a. Dockerize the Application**

* Create a Dockerfile for your application:

**b. Kubernetes Deployment and Service Files**

* Define a deployment file (deployment.yaml) and a service file (service.yaml):
* Deploy the application to Kubernetes:

**5. Monitoring and Logging**

**a. Set Up Prometheus and Grafana**

* Install Prometheus and Grafana on the Kubernetes cluster using Helm:
* Configure Prometheus to scrape metrics from your application.
* Use Grafana dashboards to visualize metrics like CPU usage, memory, and request counts.

**b. Logging with ELK Stack**

* Deploy Elasticsearch, Logstash, and Kibana (ELK) stack in Kubernetes.
* Configure your application to log to Logstash, which will forward logs to Elasticsearch.
* Visualize logs in Kibana for real-time monitoring.

**6. Scaling and Load Balancing**

**a. Horizontal Pod Autoscaling**

* Set up Horizontal Pod Autoscaling (HPA) based on CPU or memory usage:

**b. Load Balancing**

* Use AWS ELB or configure NGINX as a reverse proxy in Kubernetes.
* If using NGINX:
  + Deploy it as an Ingress controller in your cluster.
  + Configure an Ingress resource to handle incoming traffic routing.

**7. Security and Best Practices**

1. **Use IAM Roles and Policies** for EC2 instances, Jenkins, and Kubernetes to manage access control.
2. **Enable SSL/TLS** for secure communication by configuring NGINX or the AWS load balancer with SSL certificates.
3. **Environment Variables**: Use Kubernetes secrets to store sensitive information like API keys, database credentials, and passwords.
4. **Regular Backups**: Implement periodic database and configuration backups.
5. **Regular Security Audits**: Use tools like Aqua Security or Twistlock to scan containers for vulnerabilities.

**Conclusion**

Following this guide, you’ve built a robust infrastructure and CI/CD pipeline, deployed a containerized application, set up monitoring, and implemented scaling and security measures.