|  |
| --- |
| **Devops-Technical Assignment**  This technical assignment’s main goal is to design, provision, and manage a highly available Kubernetes environment and  deploying a EKS cluster using modern automation and GitOps and FluxCD practices.  **1. High-Level Infrastructure Overview**   * Provision infrastructure using Terraform (AWS EKS) and deploy and manage a Redis cluster. * Implement FluxCD using Argo CD to ensure continuous delivery with zero downtime. * Integrate monitoring using Prometheus and Grafana for visibility into cluster and application performance. * The project reflects real-world SRE workflow covering infrastructure such as code, Kubernetes management, observability, and automation.   **Infrastructure Provisioning Objective**  Provision a fully functional Kubernetes (EKS) cluster using Terraform on AWS.  **Overview:**  This section demonstrates how the infrastructure for a scalable Redis deployment is provisioned using Infrastructure as Code (IaC) with Terraform.  **Objectives**   1. **Infrastructure Provisioning:**  * Provision an AWS EKS cluster using Terraform. * Create VPC, public subnets, IGW, and worker nodes.  1. **Kubernetes Deployment:**  * Deploy a Redis cluster with Master-Replica setup. * Ensure high availability, scalability, and persistence.  1. **Continuous Integration / Deployment (CI/CD):**  * Use Argo CD for GitOps-based deployments. * Enable zero-downtime updates for Redis.  1. **Monitoring:**  * Deploy Prometheus and Grafana via Helm. * Create custom dashboards for Redis and Kubernetes metrics.   **Setup involves:**   * AWS components: VPC, Subnets, IAM Roles, Node Groups, etc. * AWS EKS cluster * FluxCd for GitOps deployment * Prometheus + Grafana for monitoring * Cluster deployed inside EKS   Terraform folder structure.    **Terraform Highlights**   * Modules: VPC, EKS, IAM roles * Remote backend: to store tf state file in S3 bucket. * Cluster Name: Complredict-eks-cluster * Region: eu-central-1   **Terraform commands:**  Initialization  terraform init  terraform plan  terraform apply -auto-Approve  Once you deploy terraform code to create a network and eks cluster on aws cloud.  **Please follow the command below**.  1.Connect to EKS Cluster:   * This command configures your local kubectl to connect to your AWS EKS cluster named Compredict-eks-clusterin region eu-central-1. * It updates (or creates) your kubeconfig file with the cluster's API endpoint and authentication token. * After running this, you can use kubectl to manage resources in that EKS cluster.   aws eks --region eu-central-1 update-kubeconfig --name Compredict-eks-cluster # please update your region  **2. Check Existing Namespaces**   * Lists all namespaces in your Kubernetes cluster. * Namespaces logically separate resources (eg, default, kube-system, argocd, etc.)   kubectl get ns    3. Check System Pods:  Shows all system pods running in the kube-systemnamespace.  This includes core components like:   * coredns– handles cluster DNS. * aws-node– VPC CNI networking pods. * kube-proxy– manages network routing inside cluster. * metrics-serveror other controllers.   **Used to confirm your EKS cluster system services are healthy**.  kubectl get pods -n kube-system    **Summary:** aws eks update-kubeconfig-Connect kubectl to your EKS cluster.  kubectl get ns - View existing namespaces. kubectl get pods -n kube-system -- Check system components’ health  **Output:**      **kubectl get pods -n nginx -o wide (Lists pods with extra details like node, IP, and image.) Explanation:**   * This command does the same as above but with more details (that's what -o widemeans). * The -flag means “output format”, and wide gives you extended information such as:   + Node each pod is running on   + Pod IP address   + Container image   + Host IP   + More columns than the default view   Output  You already have a nginx master Stateful Set running. Since this is just the master,  We can test basic connectivity and functionality first. Here’s a step-by-step guide:    **2.Check Services (you created one)** If you defined a headless service verify it:  **kubectl get svc -n nginx**  **Step-by-Step Installation of Prometheus & Grafana via Helm** To complete, step-by-step guide to install Prometheus and Grafana via Helm on your Redis cluster (or any Kubernetes cluster).  Step 1: Add Helm repositories:  helm repo add prometheus-community https://prometheus-community.github.io/helm-charts  helm repo add grafana https://grafana.github.io/helm-charts  helm repo update  Step 2: Create a monitoring namespace  kubectl create namespace monitoring    Step 3: Install the kube-prometheus-stack chart please run all below commands. helm install kube-prometheus prometheus-community/kube-prometheus-stack -n monitoring  helm repo add prometheus-community <https://prometheus-community.github.io/helm-charts>  helm repo add grafana <https://grafana.github.io/helm-charts>  helm repo add grafana <https://grafana.github.io/helm-charts>  helm repo update  helm install prometheus-stack prometheus-community/kube-prometheus-stack --namespace monitoring --create-namespace --  set grafana.enabled=true --set prometheus.prometheusSpec.serviceMonitorSelectorNilUsesHelmValues=false  A computer screen with text  AI-generated content may be incorrect.  Get Grafana 'admin' user password by running:  kubectl --namespace monitoring get secrets prometheus-stack-grafana -o jsonpath="{.data.admin-password}" | base64 -d ; echo    This installs:   * Prometheus * Grafana * Alertmanager * Exporters for nodes, pods, and cluster metrics   Step 4: Verify the installation  kubectl get pods -n monitoring  kubectl get svc -n monitoring  Step 5: Access Grafana Dashboard  Grafana is deployed as a service named kube-prometheus-grafana  Port-forward (quick access)  kubectl port-forward svc/kube-prometheus-grafana -n monitoring 3000:80  <http://localhost:3000/>  **Step 6: Get Grafana admin password**  The default username is admin, and the password is stored in a Kubernetes secret:  kubectl get secret -n monitoring kube-prometheus-grafana -o jsonpath="{.data.admin-password}" | base64 –decode  **Step 7: Login to Grafana**  Open Grafana (via port-forward or external IP) and login:  Username: admin  Password: <decoded password>  **Step 8: View dashboards**  Go to:   * Dashboards → Browse * You’ll find prebuilt dashboards for:   + Kubernetes / Nodes   + Pods   + Deployments   + Cluster / API server metrics   + Kubernetes / Nodes     Computer resource pods details:    Kubernetes/Networking/Workload |

==========

|  |  |
| --- | --- |
| Absolutely! Let’s go step by step to **install Fluentd on your EKS cluster using Helm**, configure it, and collect logs from your applications. I’ll make it complete and practical.  **Prerequisites**   * You have **kubectl configured** for your EKS cluster:   kubectl get nodes   * Helm is installed:   helm version   * A namespace for logging (optional, we’ll create one):   **Add Fluentd Helm Chart Repository**  There are multiple charts; we’ll use the official **fluent Helm chart**:  helm repo add fluent https://fluent.github.io/helm-charts  helm repo update  **Create a Logging Namespace**  kubectl create namespace logging  **Install Fluentd via Helm**  Basic installation (logs go to stdout):  helm install fluentd fluent/fluentd \  --namespace logging \  --set daemonset.enabled=true \  --set rbac.create=true   |  | | --- | | NAME: fluentd  LAST DEPLOYED: Tue Oct 28 19:56:58 2025  NAMESPACE: logging  STATUS: deployed  REVISION: 1  NOTES:  Get Fluentd build information by running these commands:  export POD\_NAME=$(kubectl get pods --namespace logging -l "app.kubernetes.io/name=fluentd,app.kubernetes.io/instance=fluentd" -o jsonpath="{.items[0].metadata.name}")  kubectl --namespace logging port-forward $POD\_NAME 24231:24231  curl http://127.0.0.1:24231/metrics |     **✅ Explanation:**   * daemonset.enabled=true → runs Fluentd on all cluster nodes * rbac.create=true → creates roles and permissions for Fluentd to read pod logs   **Optional: Send Logs to a Backend (e.g., Elasticsearch or Loki)**  For example, to send logs to Loki:  helm install fluentd fluent/fluentd \  --namespace logging \  --set daemonset.enabled=true \  --set rbac.create=true \  --set backend.type=loki \  --set backend.loki.url=http://loki:3100/loki/api/v1/push  **Verify Fluentd is Running**  kubectl get pods -n logging  kubectl logs -n logging -l app.kubernetes.io/name=fluentd  You should see **Fluentd pods running on each node** and logs being collected from applications.  **Configure Application Logging**  Fluentd automatically collects logs from:   * /var/log/containers * /var/log/pods   Your applications just need to write logs to **stdout/stderr**, which is standard for Kubernetes containers. For example:  apiVersion: apps/v1  kind: Deployment  metadata:  name: myapp  namespace: default  spec:  replicas: 2  selector:  matchLabels:  app: myapp  template:  metadata:  labels:  app: myapp  spec:  containers:  - name: myapp  image: nginx:latest  ports:  - containerPort: 80  # All logs from nginx go to stdout/stderr  Fluentd will automatically collect these logs.  **View Application Logs via Fluentd**  If you installed **stdout backend**, you can check logs from Fluentd pods:  kubectl logs -n logging -l app.kubernetes.io/name=fluentd  If you installed **backend like Loki**, use Grafana or Loki UI to view logs.  **Optional: Forward Logs to Centralized System**  You can configure values.yaml for Fluentd to send logs to:   * Elasticsearch * Loki * AWS CloudWatch Logs   For example, **CloudWatch Logs**:  backend:  type: cloudwatch  cloudwatch:  region: <AWS\_REGION>  logGroupName: /eks/fluentd  Then install:  helm install fluentd fluent/fluentd -n logging -f values.yaml  **Summary Commands for Quick Setup**  # 1. Add repo and update  helm repo add fluent https://fluent.github.io/helm-charts  helm repo update  # 2. Create logging namespace  kubectl create namespace logging  # 3. Install Fluentd  helm install fluentd fluent/fluentd -n logging --set daemonset.enabled=true --set rbac.create=true  # 4. Check pods  kubectl get pods -n logging  # 5. Check logs  kubectl logs -n logging -l app.kubernetes.io/name=fluentd  This setup ensures:   * Fluentd runs on all nodes (DaemonSet) * Collects logs from all pods and nodes * You can send logs to stdout, CloudWatch, Loki, or Elasticsearch   If you want, I can give a **ready-to-use Helm values.yaml** that configures **Fluentd for EKS + CloudWatch** so you can instantly see all application logs in CloudWatch.  Do you want me to do that? |