**Real-time Project (High-Level)**

**Three-Tier Application Deployment**

***with***

**EKS (Best Practices) & EFK Logging**

**Overview**

This project demonstrates the deployment of a scalable and secure three-tier application on Amazon Elastic Kubernetes Service (EKS) following best practices. Additionally, it incorporates centralized logging using the Elasticsearch, Fluentd, and Kibana (EFK) stack to ensure effective monitoring and troubleshooting.

**Project Objectives**

1. **Deploy a Three-Tier Architecture:**
   * Set up a frontend, backend, and database layer within the EKS environment.
2. **Implement EKS Best Practices:**
   * Secure configurations, IAM roles, namespaces, and resource quotas.
3. **Enable Centralized Logging:**
   * Deploy the EFK stack for application and system logs.
4. **Leverage Automation:**
   * Use Terraform and Helm charts to simplify the infrastructure and application deployment process.

**Project Roadmap**

**1. Prerequisites**

* AWS account with sufficient permissions.
* Pre-installed tools: AWS CLI, kubectl, Terraform, Helm, and Docker.
* Kubernetes knowledge, including pods, services, deployments, and namespaces.

**2. EKS Cluster Setup**

**Step 1:** Configure EKS Cluster using Terraform

* Create a VPC with public and private subnets.
* Deploy an EKS cluster using the Terraform AWS EKS module.
* Define IAM roles and policies for EKS.

**Step 2:** Configure Node Groups

* Create managed node groups with autoscaling enabled.
* Assign appropriate IAM roles for accessing AWS resources.

**Step 3:** Configure Networking

* Set up security groups, route tables, and internet gateways.
* Enable private communication for backend services.

**3. Deploy the Three-Tier Application**

**Step 1:** Define Namespaces and Resource Quotas

* Create separate namespaces for frontend, backend, and database tiers.
* Enforce resource quotas for CPU and memory usage.

**Step 2:** Deploy the Backend API

* Use a Dockerized Node.js/Java/Python backend API.
* Define Kubernetes Deployment and Service YAML files for the backend.

**Step 3:** Deploy the Frontend

* Use a React/Angular application.
* Deploy the frontend as a Kubernetes deployment.

**Step 4:** Configure the Database

* Use an AWS RDS instance (MySQL) for the database layer.
* Secure database credentials using AWS Secrets Manager.

**Step 5:** Create an Ingress Controller

* Deploy an NGINX Ingress Controller using Helm.
* Configure routing rules to direct traffic to the frontend and backend services.

**4. Implement EKS Best Practices**

**Step 1:** Enable Cluster Logging

* Activate EKS control plane logging for API server, scheduler, and authenticator logs.

**Step 2:** Restrict Access with IAM Roles for Service Accounts (IRSA)

* Assign service accounts specific IAM roles to limit access to AWS resources.

**Step 3:** Use Pod Security Policies (PSP)

* Enforce security best practices by restricting privileged containers and capabilities.

**Step 4:** Configure Auto-scaling

* Enable Kubernetes Horizontal Pod Autoscaler (HPA) for application services.
* Use Cluster Autoscaler to scale nodes dynamically.

**5. Integrate the EFK Stack**

**Step 1:** Deploy Elasticsearch

* Use Helm to install Elasticsearch in the logging namespace.
* Configure storage requirements and replicas for high availability.

**Step 2:** Deploy Fluentd

* Install Fluentd as a DaemonSet to collect application and system logs.
* Configure Fluentd to forward logs to Elasticsearch.

**Step 3:** Deploy Kibana

* Use Helm to install Kibana for log visualization.
* Expose Kibana via the Ingress Controller.

**6. Testing and Validation**

**Step 1:** Test Application Functionality

* Verify that the frontend connects to the backend and database.
* Test APIs and user workflows.

**Step 2:** Test Auto-scaling

* Perform load testing to validate HPA and Cluster Autoscaler functionality.

**Step 3:** Validate Logging

* Generate application logs and verify their availability in Kibana.
* Ensure log queries and visualizations are accurate.

**7. Monitoring and Alerts**

* Enable Kubernetes Metrics Server.
* Configure Prometheus and Grafana for resource monitoring.
* Set up alerts for pod failures, resource usage, and application downtime.

**8. Cleanup**

* Remove unused resources to avoid unnecessary costs.
* Backup important logs and data.

**Expected Outcomes**

1. A scalable and secure three-tier application deployed on EKS.
2. Centralized log management using the EFK stack.
3. Automated infrastructure provisioning with Terraform and Helm.
4. Enhanced monitoring and alerting for efficient operations.

**Tools and Technologies Used**

* **Cloud Provider:** AWS (EKS, RDS, Secrets Manager, IAM, VPC)
* **Orchestration:** Kubernetes (kubectl, namespaces, deployments, services)
* **Infrastructure as Code:** Terraform
* **Package Management:** Helm
* **Logging and Monitoring:** Elasticsearch, Fluentd, Kibana, Prometheus, Grafana
* **Languages:** YAML, HCL (Terraform)

**Conclusion**

This project not only demonstrates the deployment of a robust three-tier application but also emphasizes best practices in scalability, security, and centralized logging. By integrating the EFK stack, teams can effectively monitor and troubleshoot their application, ensuring reliability and performance in a production-grade environment.

**Important yaml files are here:**

**sudo vim cluster.yaml**

Insert the following configuration.

**apiVersion: eksctl.io/v1alpha5**

**kind: ClusterConfig**

**metadata:**

**name: dec-24-eks-19**

**region: us-east-1**

**nodeGroups:**

**- name: ng-1**

**instanceType: t2.medium**

**desiredCapacity: 3**

**volumeSize: 20**

**iam:**

**withOIDC: true**

**addons:**

**- name: aws-ebs-csi-driver**

**wellKnownPolicies:**

**ebsCSIController: true**

…

**vim backend.yaml**

insert this…

**env:**

**- name: MYSQL\_HOST**

**value: "mysql-svc.mysql.svc.cluster.local"**

**- name: MYSQL\_USER**

**value: "ffmuser"**

**- name: MYSQL\_PASSWORD**

**value: "my-secret-pw"**

**- name: MYSQL\_DB**

**value: "twotier"**

Install **Helm and Run MySql**

Copy the Helm installation commands from

[*https://helm.sh/docs/intro/install/*](https://helm.sh/docs/intro/install/)

**curl https://baltocdn.com/helm/signing.asc | gpg --dearmor | sudo tee /usr/share/keyrings/helm.gpg > /dev/null**

**sudo apt-get install apt-transport-https --yes**

**echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/helm.gpg] https://baltocdn.com/helm/stable/debian/ all main" | sudo tee /etc/apt/sources.list.d/helm-stable-debian.list**

**sudo apt-get update**

**sudo apt-get install helm**

**vim values.yaml**

Write configurations as

**auth:**

**rootPassword: "hello123"**

**createDatabase: true**

**database: "twotierbackenddb"**

**username: "secureuser"**

**password: "very-secure-password"**

**vim storage.yaml**

apiVersion: storage.k8s.io/v1

kind: StorageClass

metadata:

name: ebs-sc

namespace: mysql

annotations:

storageclass.kubernetes.io/is-default-class: "true" # Marking it as default class

provisioner: ebs.csi.aws.com

parameters:

type: gp2

reclaimPolicy: Retain

volumeBindingMode: WaitForFirstConsumer

**helm repo add bitnami** [**https://charts.bitnami.com/bitnami**](https://charts.bitnami.com/bitnami)

**helm repo update**

Install mysql with the values file

**helm install mysql oci://registry-1.docker.io/bitnamicharts/mysql -f values.yaml**

**vim backend.yaml**

Insert the following environment variables under the **containers**section

**env:**

**- name: MYSQL\_HOST**

**value: "mysql.default.svc.cluster.local"**

**- name: MYSQL\_USER**

**value: "secureuser"**

**- name: MYSQL\_PASSWORD**

**value: "very-secure-password"**

**- name: MYSQL\_DB**

**value: "twotierbackenddb"**

MySQL part

…….

**kgp**

**k exec -it** <mysql\_pod\_name> **bash**

Go to the MySQL console using the newly create secureuser user and its password

**mysql -u secureuser -p**

Enter the password: **very-secure-password**

See mysql>

Configure the Database

See that the **twotierbackenddb**database has been created.  
Run the following commands to create the tables in the database

**show databases;**

**use twotierbackenddb;**

**CREATE TABLE users (id int NOT NULL AUTO\_INCREMENT, email varchar(100) NOT NULL, password varchar(100) NOT NULL, primary key(id));**

**CREATE TABLE history (id int NOT NULL AUTO\_INCREMENT, number varchar(100) NOT NULL, response varchar(100) NOT NULL, type varchar(30) NOT NULL, time timestamp default current\_timestamp, primary key(id));**

**show tables;**

Create a user in the users table and view the table

**insert into users (email, password) values ("user1@example.com", "hello123");**

**SELECT \* FROM users;**

Exit from MySQL and the pod.

**exit**

**exit**

Run the Backend Application with the Help of **ConfigMap** to Serve the Environment Variables

Create a db-config.yaml file for key-value pairs in the ConfigMap

**vim db-config.yaml**

Write configuration

**apiVersion: v1**

**kind: ConfigMap**

**metadata:**

**name: mysql-config**

**data:**

**MYSQL\_HOST: "mysql.default.svc.cluster.local"**

**MYSQL\_USER: "secureuser"**

**MYSQL\_PASSWORD: "very-secure-password"**

**MYSQL\_DB: "twotierbackenddb"**

Apply the ConfigMap and see it in action

**k apply -f db-config.yaml**

**k get configmap**

**k get cm mysql-config -o yaml**

Edit the **backend.yaml**file and link to the newly created ConfigMap so that it receives the environment variables from the ConfigMap

**vim backend.yaml**

Under the **containers** section add the following configuration

**envFrom:**

**- configMapRef:**

**name: mysql-config**

**Incorporate EFK stack for logging**

Create logging namespace and change the context

**k create ns logging**

**k config set-context --current --namespace=logging**

**helm repo add elastic**[**https://helm.elastic.co**](https://helm.elastic.co/)

**helm repo update**

helm install elasticsearch elastic/elasticsearch --set imageTag=8.5.1 --set replicas=1 -n logging

*Add the****Fluentd****repo and install it*

**helm repo add fluent**[**https://fluent.github.io/helm-charts**](https://fluent.github.io/helm-charts)

**helm repo update**

**helm install my-fluentd fluent/fluentd --version 0.5.2**

**kgp**

See that a set of Fluend are running but they keep restarting

See that **Fluentd** DaemonSet is failing to be ready

**k get ds**

**k edit ds fluentd**

Remove the **livenessProbe**and **readinessProbe**configurations

See that **Fluentd** daemonset has been edited and the pods and daemonsets are ready now

**kgp**

**k get ds**

[***https://artifacthub.io/packages/helm/elastic/kibana***](https://artifacthub.io/packages/helm/elastic/kibana)

Install kibana with the type as **NodePort**

**helm install kibana --version 8.5.1 elastic/kibana --set service.type=NodePort**

**kgs**

**kgp**

Go to one of the cluster node's <**IP\_Address>:kibana\_NodePort** and see EFK stack in action. User: elastic

But, password is at here:

**kubectl get secrets --namespace=logging elasticsearch-master-credentials -ojsonpath='{.data.password}' | base64 -d**

Dismiss the messages and choose to **Explore on my own**

Click on the three-bar icon and go to **Stack Management**

Select **Index Patterns**

Click on the **Create Index pattern**

Insert the Name **fluentd\***then proceed to **Create index pattern**

Click on the three-bar icon and go to **Index Management**

See the **fluentd** index and its health is yellow, which is good

Click on the three-bar icon and go to **Discover**

You can see all the logs of the entire **EKS**cluster

You can query to view custom logs. View the backend logs

**kubernetes.labels.app: backend**

Select the field **log** for better view

**Thank You**