OnFinance AI Assignment - Full Documentation

This document captures the complete workflow, commands, errors, and configurations used to complete the OnFinance AI assignment, from infrastructure setup to application deployment and monitoring using Fluent Bit.

Before getting started remember to install AWS CLI, Kubectl, EKSCTL, Docker,

After aws cli do aws configure and give access-key, secret-access-key, region and then this command: **aws eks --region ap-south-1 update-kubeconfig --name terraform-cluster**

1. Infrastructure Setup Using Terraform

The infrastructure was built on AWS using Terraform. It includes VPC, subnets, internet gateway, NAT gateway, route tables, security groups, an EKS cluster, and an RDS instance.

Key resources created:

- VPC with public and private subnets

- Internet Gateway and NAT Gateway

- Security Groups for EKS and RDS

- EKS Cluster

- MySQL RDS instance

2. Application Setup

The application was created using Python Flask for the backend with an HTML frontend.

3. Docker Image Creation

A Dockerfile was created to containerize the Flask app.

Commands used:  
# docker build -t onfinance-app .  
# docker run -p 5000:5000 onfinance-app

4. Amazon ECR Setup

# aws ecr create-repository --repository-name onfinance

# aws ecr get-login-password --region ap-south-1 | docker login --username AWS --password-stdin <account\_id>.dkr.ecr.ap-south-1.amazonaws.com  
docker tag onfinance-app <account\_id>.dkr.ecr.ap-south 1.amazonaws.com/onfinance

# docker push <account\_id>.dkr.ecr.ap-south-1.amazonaws.com/onfinance

5. Deploy to EKS

Kubernetes manifests were created to deploy the application. This includes deployment and service YAMLs.  
# kubectl apply -f deployment.yaml  
# kubectl apply -f service.yaml

6. RDS Database Connection

The Flask app was configured to connect to the RDS instance using environment variables.

7. Monitoring with Fluent Bit

Fluent Bit was configured to send logs to Amazon CloudWatch. Three files were used:  
# fluent-bit.yaml  
# fluent-bit-cluster-info.yaml  
# fluent-bit-daemonset.yaml

8. Errors and Debugging

Error encountered during Fluent Bit deployment:

STATUS: CreateContainerConfigError

This error typically indicates missing ConfigMap, invalid configuration, or incorrect image. Troubleshooting involved checking logs with:

kubectl describe pod <pod-name> -n amazon-cloudwatch  
kubectl logs <pod-name> -n amazon-cloudwatch

9. Final Notes

Key points to remember:

- Always validate Terraform with `terraform validate` and `terraform plan`

- Test the Docker image locally before pushing to ECR

- Ensure IAM permissions for CloudWatch and EKS nodes

- Use proper ConfigMap and Fluent Bit settings for CloudWatch integration

- Use `kubectl get all -n <namespace>` and logs to troubleshoot Kubernetes issues