Assignment for Implementation Engineer @OnFinance AI

The objective of this assignment is to implement a sample full-stack application on AWS EKS with a professional approach to logging, monitoring, and scalability. You are required to design, deploy, and demonstrate a fully functional system that follows best practices for high availability and security. The implementation should include:

- **High-Level Architecture Design**: Structuring a scalable, highly available, and secure cloud solution.
- **Infrastructure as Code (IaC)**: Using Terraform to automate AWS resource provisioning.
- **Kubernetes Deployment**: Deploying a sample application to EKS with best practices.
- **Logging and Monitoring**: Setting up centralized logging, monitoring, and alerting.
- **Bonus Points**: Implementing a CI/CD pipeline using GitHub Actions and ECR for container management.

Stage 1: High-Level Architecture Design (AWS Deployment)

Task: Design a high-level cloud architecture for deploying OnFinance AI's solution on AWS. Describe how the system will be structured to be scalable, highly available, and secure. Include all major components and AWS services you plan to use. For example, consider:

- **Compute Platform**: EKS Kubernetes cluster on AWS, or ECS/Lambda if justified.
- **Networking**: VPC, subnets across multiple AZs.
- **Load Balancing**: Elastic Load Balancer or API Gateway.
- **Data Storage**: Databases or storage services if needed.
- **Additional Services**: IAM roles, logging, monitoring, and auto-scaling.

The architecture should accommodate auto-scaling, failover across at least two Availability Zones, and incorporate basic logging and monitoring.

Deliverables:

- Architecture diagram and/or a written description explaining the flow of data and the role of each component.
- Justification for service choices.

Evaluation Criteria:

- Completeness of architecture.
- High Availability and Scalability considerations.
- Appropriate use of AWS services.
- Clarity of documentation.
- Security and reliability considerations.

Stage 2: Infrastructure as Code (Terraform Templates)

Task: Implement the architecture from Stage 1 using Terraform. Write Terraform configuration files to provision the necessary AWS infrastructure. This should include:

- **Networking stack**: VPC, subnets, route tables, security groups.
- **EKS cluster**: Worker nodes via Auto Scaling Group or AWS Fargate.
- **Additional services**: RDS database, S3 bucket, IAM roles/policies.

Ensure the Terraform code is modular and parameterized with variables. Provide a README with instructions on how to initialize and apply the Terraform configuration.

Evaluation Criteria:

- Correctness of Terraform manifests.
- Infrastructure best practices (high availability, security).
- Code quality (modularization, reusability).
- Documentation clarity.

Stage 3: Kubernetes Deployment (Application & Services on EKS)

Task: Deploy a sample application on the Kubernetes cluster. The app should have:

- **Deployments**: Backend API service and frontend web service.
- **Service objects**: ClusterIP for internal communication, LoadBalancer for public access.
- **Configuration management**: ConfigMaps/Secrets.
- **Health checks**: Liveness and readiness probes.
- **Scaling**: Horizontal Pod Autoscaler (HPA) for auto-scaling.

Provide deployment manifests (YAML) or Helm charts, along with deployment instructions.

Evaluation Criteria:

- Kubernetes proficiency (correct use of Deployments, Services, etc.).
- High availability (multiple replicas, self-healing, probes).
- Scaling configuration.
- Best practices (ConfigMaps, Secrets, resource limits).

Stage 4: Logging and Monitoring Implementation

Task: Implement logging, monitoring, and alerting:

- **Logging**: Centralized logs via CloudWatch, Fluent Bit, or ELK stack.
- **Monitoring**: CloudWatch metrics, Kubernetes metrics server.

- **Alerts**: CloudWatch Alarms with SNS notifications.
- **Reliability**: Ensuring uptime with Kubernetes self-healing, multi-AZ deployment.

Evaluation Criteria:

- Logging implementation (centralized logs, CloudWatch/Fluent Bit).
- Monitoring depth (key metrics tracking, dashboards).
- Alerting (at least one automated alarm setup).
- Documentation clarity.

Stage 5: External Data Integration (API/ETL Pipeline)

Task: Implement a custom data integration pipeline:

- Fetch data from a public API (e.g., stock prices, weather, etc.).
- Store the retrieved data in AWS (S3, DynamoDB, etc.).
- Automate execution via AWS Lambda (triggered by EventBridge) or a Kubernetes CronJob.
- Ensure error handling and security (API keys stored in Secrets Manager).
- Provide a way to verify integration (sample output, logs, API endpoint).

Evaluation Criteria:

- Successful API integration.
- Correct AWS service usage.
- Code quality and documentation.

Bonus Tasks (Optional Advanced Challenges)

For candidates who want to go above and beyond, the following optional tasks can showcase additional expertise. These are **not required** but will be considered impressively if completed:

- **Continuous Deployment Pipeline:** Set up a CI/CD pipeline that automates the Terraform and Kubernetes deployment. For example, use **GitHub Actions** or **AWS CodePipeline** to lint/test your code, apply Terraform to provision infrastructure, build the application container, and deploy to EKS. Show that a commit can trigger a seamless deployment process.

- **Enhanced Security Hardening:** Implement security best practices such as using AWS Secrets Manager for all secrets and integrating it with the application, enforcing least-privilege IAM roles for your resources, setting up network policies in Kubernetes, or adding a Web Application Firewall (e.g., AWS WAF) in front of the load

balancer for the web service.

- **Advanced Monitoring & Observability:** Integrate more advanced observability tools. For instance, deploy a **Prometheus & Grafana** stack on the cluster for detailed metrics, or use AWS X-Ray (or OpenTelemetry) to demonstrate tracing of requests through the system. You could also set up log analytics by aggregating logs into an ELK stack or Amazon OpenSearch Service and writing a sample query/visualization.

- **Disaster Recovery and Scaling:** Document or implement a strategy for **disaster recovery** (e.g., backups of data, multi-region failover design) or test the auto-scaling by simulating load. For example, you might run a load test to trigger your HPA and show it working, or outline how you would scale the system to handle 10x traffic.

- **Code Quality and Testing:** Include automated tests for your infrastructure or scripts. For example, use Terraform compliance tools or linters, or a tool like Terratest to write a basic test that validates an AWS resource. Similarly, you could have unit tests for your data integration lambda function.

Submission Guidelines

Provide all code, configuration files, and documentation in a structured repository with clear instructions for setup, deployment, and cleanup.

Deadline: 6/4/2025 (Sunday EoD)

The assignment will be evaluated based on correctness, best practices and code quality.