**DevOps Approach**

**Infrastructure as Code (IaC) Practice and Automation:**

The infrastructure will be defined as code using Terraform, enabling consistent and repeatable provisioning of AWS resources. This approach ensures the entire infrastructure is version-controlled, allows for easy tracking of changes, and simplifies collaboration.

**CI/CD Approach:**

For continuous integration and continuous deployment (CI/CD), we will utilize the following tools and practices:

**Code repository**: GitHub or Bitbucket for version control and collaboration.

**Build and dependency management**: Apache Maven or Gradle for building the Spring Boot microservices and React JS application.

**Containerization**: Docker will be used to containerize the microservices.

**Container Registry**: AWS Elastic Container Registry (ECR) will be used to store and manage Docker images.

**Continuous Integration and Deployment**: Jenkins will be used to establish a robust CI/CD pipeline. It will integrate with the version control system (e.g., Git) to trigger automated builds, perform code quality checks, execute tests, and deploy the applications to the EKS cluster. Jenkins pipelines will be defined using declarative syntax for easy maintenance and scalability.

**Infrastructure Updates**: Infrastructure updates will be handled using Terraform, triggered by the Jenkins CI/CD pipeline.

**Monitoring and Logging**:

To monitor the deployed infrastructure and applications, we will implement the following:

**Prometheus and Grafana**: Prometheus and Grafana are widely used open-source tools for monitoring, logging, and alerting in Kubernetes environments, including AWS EKS.

Prometheus will send the metrics data it collects from the AWS EKS Cluster, and Grafana uses it to create the visualizations.

**Application Logging**: We can integrate Grafana Loki, a log aggregation and search system, with your EKS cluster. Grafana Loki helps you collect and store logs from various sources within your EKS cluster, including application logs, container logs, and system logs.

Configure Grafana Loki to collect and index logs and use Grafana to search, filter, and visualize log data.

**Alerting with Prometheus Alertmanager:**

Configure Prometheus Alertmanager to receive and manage alerts generated by Prometheus. Define alerting rules in Prometheus based on your monitoring requirements. These rules define the conditions that trigger alerts. Configure Alertmanager to send alerts via various notification channels such as email, Slack or any other custom integrations.

**Disaster Recovery**:

To ensure high availability and disaster recovery, we will implement the following measures:

**EKS Multi-AZ Deployment**: The EKS cluster will be deployed across multiple Availability Zones (AZs) to achieve redundancy and fault tolerance.

**Automated Database Backups**: AWS RDS with MySQL supports automated backups, allowing us to easily restore data in case of a failure.

**Cross-Region Replication**: For critical workloads, we can set up cross-region replication of the RDS database to a different AWS region, ensuring data redundancy and disaster recovery.

**Cost-Effectiveness, High Availability, and Scalability:**

To achieve cost-effectiveness, high availability, and scalability, we will employ the following strategies:

**AWS Auto Scaling**: EKS clusters can be configured with AWS Auto Scaling, allowing the cluster to scale horizontally based on demand and workload metrics.

**Load Balancing**: We will utilize AWS Elastic Load Balancer (ELB) or AWS Application Load Balancer (ALB) to distribute traffic across multiple instances of the microservices and React JS application, ensuring high availability and load distribution.

**Cost Analysis:** Cost analysis will be performed to optimize expenses while maintaining a highly available and scalable architecture. This includes right-sizing EC2 instances, using spot instances for non-critical workloads, leveraging reserved instances for cost savings, and utilizing auto-scaling features to match resource usage to demand.

**Implementation Plan:**

The implementation plan will include the following steps:

1. Setting up the AWS environment (VPC, subnets, security groups)
2. Defining Terraform scripts for provisioning infrastructure components.
3. Configuring EKS cluster with auto-scaling and worker nodes
4. Creating the RDS instance and connecting it to the EKS cluster
5. Developing React JS-based UI and deploying it to S3.
6. Configuring API Gateway to manage API endpoints.
7. Setting up Jenkins pipelines for CI/CD
8. Implementing monitoring, logging, alerting, and disaster recovery mechanisms