Here we would cover the components used in building the application and the infrastructure. We'll also include the rationale behind the design and the components used. Finally we'll wrap it up with possible improvements that could be made to make the system more robust and secure.

Components

- VPC, public and private Subnets
- API Gateway
- Application load balancer (ALB)
- EKS (Fargate)
- Aurora RDS (MySQL)
- ECR

VPC, public and private Subnets

VPC provides isolation of resources from the public cloud. While private vpn enforces secure access to resources within the VPC, reducing the blast radius in times of system compromise.
Public subnets provide outside access to our application via ALB and NAT

API Gateway

This helps in API management, rate limiting, authentication (JWT validation).

Application load balancer (ALB)

ALB is responsible to distribute the incoming requests to the desired targets (in our case EKS). It also helps in scaling up/down depending on request traffic. With WAF (Web App Firewall) integration we could have another layer of security filter.

EKS (Fargate)

Reduces operational overhead, simplifies scaling based on resources usage, cost optimized especially in test environment.

Aurora RDS (MySQL)

Managed RDS service again reducing operational overhead. Very much scalable with almost realtime read replicas.

ECR

Docker registry service to store our application image with simple integration with EKS.

Other Components

Above being the major chunk of the infrastructure, other supporting elements such as security groups, ACLs, IAM roles, secret manager and CloudWatch play a pivotal role in ensuring proper and secure communication among the various resources

The Application

The application was built using Java 21 and the Spring Framework. Gradle is used for build automation and dependency management, while Liquibase manages database versioning and schema changes.

Deployment process

GitHub is used as the source code repository, with the `main` branch serving as the deployable branch. GitHub Actions automates the CI/CD pipeline by testing the build, containerizing the application with Docker, pushing the image to Amazon ECR, and applying the Kubernetes configuration.

Improvements

Though the application is not fully production ready, I would like to suggest a few possible improvements which would make the application more robust, reliable and secure:

- The API endpoints need to be protected with role-based access control. This could be achieved by adding the role to JWT, parsing it and using Spring security to authorise the incoming request.
- In the current implementation, the account-id is passed as params in the request. This could be overcome by using account identifiers inside the JWT.
- CloudWatch integration with EKS should be enabled by side-carring log agent to pull the application logs.
- Alerts to be set up to monitor resource usage, frequent request failures, etc.