Deploy Java microservices On Amazon ECS

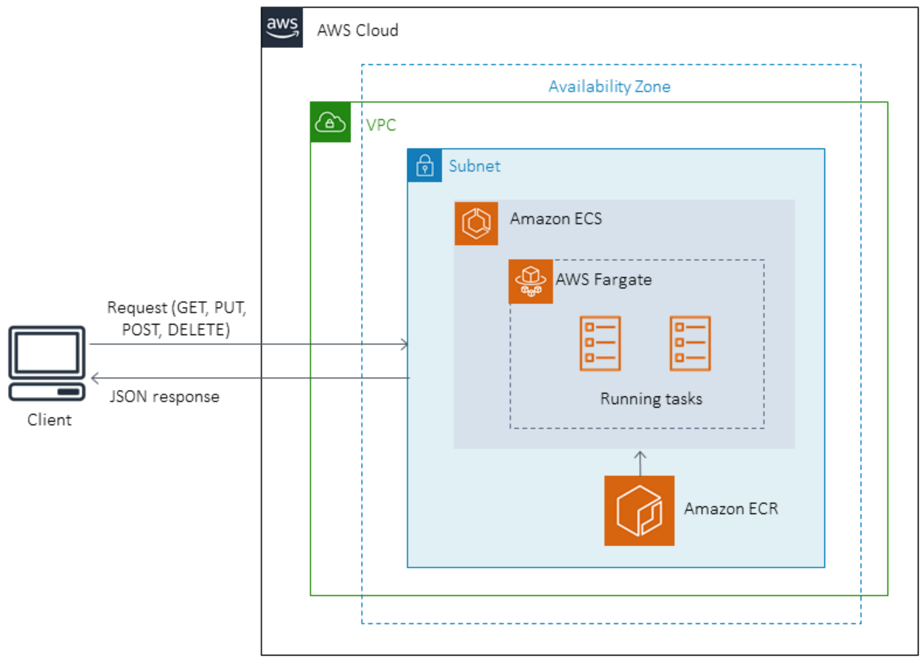
This pattern guides you through the steps for deploying Java microservices as containerized applications in Amazon ECS.

Git repo : link

Type of the cloud Architecture : Public Cloud architecture

Target technology stack :

- Amazon Elastic containers Registry : fully managed registry that makes it easy for developers to store , manage and deploy Docker container images.
  → Amazon ECR hosts your images in a highly available and scalable architecture so you can reliably deploy containers for your applications
- Amazon Elastic Containers Service : is a highly scalable , high-performance container orchestration service that supports Docker containers and allows you to easily run and scale containerized apps on AWS.
- Aws Fargate : a compute engine that you can use with Amazon ECS to run containers without having to manage servers or clusters of Amazon EC2 instances. With AWS Fargate, you no longer have to provision, configure, or scale clusters of virtual machines to run containers.
- Docker is a platform that lets you build, test, and deliver applications in packages called containers.



Non-Functional Requirements :

- Scalability :
  - Auto-scaling : Configure ECS services to automatically scale based on demand , ensuring that the system can handle varying loads efficiently.
- High availability :
  - Deploy ECS services across multiple Availability Zones to ensure high availability and fault tolerance.
- Security :
  - Encryption: Ensure that data at rest in ECR and in transit between ECR and ECS is encrypted using AWS-managed encryption keys or customer-managed keys for enhanced security.

Epics :

Terraform Epics:

Epic 1: Containerization and Image Push to Amazon ECR

- Task 1.1: Write a Dockerfile for each microservice. For Spring Boot applications, use a base image that includes Java, and for the Python microservice, use a base image that includes Python.
- Task 1.2: Build Docker images for each microservice using the Dockerfiles created.
- Task 1.3: Create an Amazon Elastic Container Registry (ECR) repository for each microservice using Terraform.
- Task 1.4: Push the Docker images to their corresponding ECR repositories using Terraform.

Epic 2: Setting Up the Amazon ECS Cluster

- Task 2.1: Define a Terraform configuration for creating an Amazon ECS cluster using AWS Fargate as the launch type.
- Task 2.2: Configure a Virtual Private Cloud (VPC) network and associated security groups using Terraform.
- Task 2.3: Set up an Internet Gateway and route tables to allow internet access to the ECS cluster.

Epic 3: Defining Task Definitions and Deploying Services

- Task 3.1: Create Terraform configurations for ECS task definitions for each microservice. Specify the Docker image, CPU, memory, and other required parameters.
- Task 3.2: Deploy the Spring Boot microservices as ECS services using Terraform. Configure the desired number of tasks and other service settings.
- Task 3.3: Deploy the Python microservice as an ECS service using Terraform. Configure the desired number of tasks and other service settings.
- Task 3.4: Configure an Amazon Elastic Load Balancing (ELB) to distribute traffic among the microservices using Terraform.

Ansible Epics:

Epic 4: Configuring Auto Scaling and Monitoring

- Task 4.1: Use Ansible to configure auto-scaling for the ECS services. Define scaling policies based on CPU utilization or other metrics.
- Task 4.2: Set up AWS CloudWatch for monitoring the ECS services. Use Ansible to create CloudWatch alarms and notifications.

Epic 5: Database Integration and Continuous Integration/Continuous Deployment (CI/CD)

- Task 5.1: If required, use Ansible to set up a database using Amazon RDS. Configure the database and ensure it is accessible to the microservices.
- Task 5.2: Automate the deployment process using AWS CodePipeline and AWS CodeBuild. Use Ansible to configure the CI/CD pipeline, including build, test, and deployment stages.

Epic 6: Finalizing and Testing

- Task 6.1: Use Ansible to perform final configuration tasks, such as setting environment variables or configuring network settings.
- Task 6.2: Conduct thorough testing of the microservices in the ECS environment. Use Ansible to automate testing scripts and validate the deployment.
- Task 6.3: Document the deployment process, including Terraform configurations, Ansible playbooks, and any lessons learned. Use Ansible to automate the documentation process.

Other type of applications that can benefit from this architecture:

- Web Applications : including simple websites and complex web applications

→ ELB (Elastic Load Balancing) and Amazon EBS (Amazon Elastic Block Store) ensures high availability and scalability

- Batch Processing Applications : Apps that require data processing , analytics and ML workloads .

→ This Architecture Allows for the efficient scheduling and execution of batch jobs.