**SHRIKANT SHERKAR: TERRAFORM ASSIGNMENT**

**1. Architecture Overview**

We will deploy a **Node.js API** on **AWS ECS (Fargate)** for serverless, scalable container management. The API will be secured and rate-limited at the infrastructure level. The architecture includes:

* **AWS API Gateway** → For rate limiting and authentication.
* **AWS Cognito** → To manage API authentication.
* **AWS ECS (Fargate)** → To run the API in containers.
* **AWS ALB (Application Load Balancer)** → To handle blue-green deployments.
* **AWS Route 53** → For DNS management.
* **AWS RDS (PostgreSQL)** → To store API usage data if needed.
* **AWS CloudWatch** → For monitoring and logging.

**2. Blue-Green Deployment**

We will deploy two separate ECS services (Blue & Green) behind an ALB. Traffic switching will be handled via **Route 53 and ALB Target Groups**, ensuring zero downtime.

**3. CI/CD Pipeline**

* **GitHub Actions/GitLab CI/CD** will automate:
  + Code linting & formatting (ESLint, Prettier)
  + Code quality checks (SonarQube)
  + SAST (GitHub CodeQL or Snyk)
  + Container build & push to AWS ECR
  + Terraform infrastructure provisioning
  + Deployment to AWS ECS using blue-green strategy

**4. Infrastructure as Code (Terraform)**

* **Modularized Terraform Setup:**
  + networking (VPC, Subnets, Security Groups)
  + ecs (Cluster, Services, Task Definitions)
  + alb (Load Balancer, Target Groups, Listeners)
  + api\_gateway (Rate limiting, Authentication)
  + cicd (IAM Roles, Pipelines)
  + monitoring (CloudWatch Alarms & Logs)

**5. Deliverables**

* **GitHub/GitLab Repository** with:
  + Terraform modules for infrastructure.
  + CI/CD pipeline configuration (GitHub Actions/GitLab CI).
  + Architecture diagram.

**API Gateway**

**Why API Gateway?**

* It acts as the entry point for all API requests.
* It ensures rate limiting to prevent abuse.
* It integrates AWS Cognito for authentication, enforcing only authorized users can access the API.

**Application Load Balancer (ALB)**

**Why ALB?**

* Handles incoming traffic and distributes it to the ECS cluster.
* Ensures high availability by routing traffic only to healthy containers.
* Supports HTTPS termination to secure API requests.

**ECS Fargate Cluster (Node.js API)**

**Why ECS Fargate?**

* Serverless container management, so there is no need to manage EC2 instances.
* Scales automatically based on traffic demand.
* Supports blue-green deployments for zero-downtime updates.

**How It Works:**

1. The Node.js API container runs inside ECS Fargate.
2. It receives requests from ALB and processes them.
3. The API fetches dummy data based on request parameters.
4. If the API needs persistent storage, it interacts with the RDS database.

**RDS PostgreSQL Database**

**Why RDS?**

* A fully managed relational database optimized for performance and security.
* Supports automatic backups, multi-AZ failovers, and encryption.

**CI/CD Pipeline (GitHub Actions)**

**Why CI/CD?**

* Automates testing, security scans, and deployments.
* Ensures high code quality before deploying to production.
* Supports automatic rollback if something goes wrong.

**How It Works:**

1. **Developers push code to GitHub.**
2. **GitHub Actions pipeline triggers:**
   * Linting & Code Quality Checks (ESLint, SonarQube)
   * Static Application Security Testing (SAST) (Check for vulnerabilities)
   * Build & Push Docker Image to Amazon ECR.
   * Deploy to ECS Fargate using Terraform.
3. **If tests pass, it automatically updates the ECS service (Blue-Green Deployment).**

**Monitoring & Logging (CloudWatch)**

**Why CloudWatch?**

* Provides real-time monitoring for API and database health.
* Sends alerts for high CPU, memory, or errors.

**How It Works:**

1. ECS API logs are stored in CloudWatch Log Groups.
2. API Gateway logs requests for security audits.
3. CloudWatch Alarms trigger alerts if:
   * ECS CPU > 75%
   * RDS CPU > 80%
4. SNS Notifications send alerts to DevOps teams.

**End-to-End Request Flow**

* **Client → Sends request to API Gateway**
* **API Gateway → Authenticates via Cognito, applies rate limits**
* **ALB → Routes request to ECS API service**
* **ECS API (Node.js) → Processes request, queries RDS if needed**
* **RDS (PostgreSQL) → Returns data to ECS API**
* **ECS API → Sends response back to client**
* **CloudWatch → Monitors logs, sends alerts**

**Final Thoughts**

* This architecture ensures scalability, security, and automation.
* It's designed to handle real-world production workloads efficiently.
* The CI/CD pipeline automates deployment while CloudWatch ensures reliability.