**API Users (Clients)**

* Users or applications (e.g., front-end apps, external clients) send **requests to the API**.
* The request contains the **desired fields and number of dummy records** they need.
* The API should **only allow authenticated and rate-limited users**.

**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.