To design an AWS infrastructure solution diagram that meets the provided requirements. I’ll break down the components and services, and then provide a comprehensive diagram.

**1. Backend Components:**

* **Two API Servers**:
  + These can be implemented using **Amazon EC2** instances.
* **Autoscaling for API Servers**:
  + We’ll use **Amazon EC2 Auto Scaling** to automatically adjust the number of instances based on demand.
* **RDS (Relational Database Service)**:
  + For the database, we’ll use **Amazon RDS** (e.g., MySQL, PostgreSQL, or Aurora).
* **Redis Server**:
  + We’ll set up a **Redis cluster** using **Amazon ElastiCache**.
* **Lambda Function**:
  + We’ll set up a **Lambda Function** to serve all the API services.
* **API Gateway**:
  + To authenticate and authorize the API request we will use an **API Gateway**.
* **SSL Certificates and Domains**:
  + We’ll use **Amazon Certificate Manager (ACM)** for SSL certificates.
  + For domains, we can use **Amazon Route 53**.

**2. Frontend Components:**

* **Amazon CloudFront**:
  + We’ll implement CloudFront as a **content delivery network (CDN)** for the frontend.
  + CloudFront will cache static assets (e.g., HTML, CSS, JS) and distribute them globally.
* **Access Restriction**:
  + To restrict access to the frontend:
    - We can use **Amazon CloudFront signed URLs or cookies**.
    - Configure **Origin Access Identity (OAI)** to allow only CloudFront to access the S3 bucket (where frontend assets are stored).

**3. Continuous Integration/Deployment (CI/CD):**

* We’ll set up CI/CD using **AWS CodePipeline**, **AWS CodeBuild & AWS CodeDeploy**
  + Source: **AWS CodePipeline** connect to your code repository (GitHub).
  + Build: Use **AWS CodeBuild** to build and package your application.
  + Deploy: **AWS CodeDeploy** to the appropriate environment
  + **S3** bucketto store **the Code and fronted assets**

**4. Networking Components:**

* **Amazon VPC (Virtual Private Cloud)**:
  + We will create a custom VPC with public and private subnets.
  + Use **Network ACLs (NACLs)** and **Security Groups** for network security.
* **Subnet Configuration**:
  + Public Subnets: For API servers, RDS, and ElastiCache.
  + Private Subnets: For backend instances (EC2).
* **Route Tables**:
  + Route traffic between subnets using route tables.
* **Internet Gateway (IGW)**:
  + Attach to the VPC for public internet access.
* **NAT Gateway**:
  + In public subnets, route outbound traffic from private subnets to the internet.

**Terraform Configuration (VPC):**

resource "aws\_vpc" "my\_vpc" {

cidr\_block = "10.0.0.0/16"

}

resource "aws\_subnet" "public\_subnet" {

vpc\_id = aws\_vpc.my\_vpc.id

cidr\_block = "10.0.1.0/24"

availability\_zone = "us-east-1a"

}

resource "aws\_subnet" "private\_subnet" {

vpc\_id = aws\_vpc.my\_vpc.id

cidr\_block = "10.0.2.0/24"

availability\_zone = "us-east-1b"

}

resource "aws\_internet\_gateway" "my\_igw" {

vpc\_id = aws\_vpc.my\_vpc.id

}

resource "aws\_route\_table" "public\_route" {

vpc\_id = aws\_vpc.my\_vpc.id

route {

cidr\_block = "0.0.0.0/0"

gateway\_id = aws\_internet\_gateway.my\_igw.id

}

}

resource "aws\_route\_table\_association" "public\_subnet\_association" {

subnet\_id = aws\_subnet.public\_subnet.id

route\_table\_id = aws\_route\_table.public\_route.id

}

**Diagram:**

