Objective:
Build a scalable, secure, and containerized AWS environment using Terraform. The infrastructure should include:

- Auto Scaling EC2 instances with Nginx, Docker, and Node.js 20
- RDS databases in private subnets (Publicly not accessible)
- Load Balancer with HTTPS support
- Multi-stage Dockerized web applications (Frontend + Backend)
- BI Tool deployment (Redash or Metabase)
- Domain and SSL setup for applications and BI Tool
- SSH tunneling for secure DB access
- Dashboard reflecting live DB updates

Infrastructure Requirements

1. EC2 Auto Scaling Group

- Launch **3 EC2 instances** using a Launch Template.
- Install the following via EC2 **User Data**:
  - Nginx
  - Docker
  - Node.js 20

Note: Use any OS (Amazon Linux 2 preferred for compatibility).

2. RDS Instances

- Launch **2 RDS instances**:
  - One for **MySQL**
  - One for **PostgreSQL**
- Place them in **Private Subnets** (No public IPs).
- Ensure proper **Subnet Grouping** and **Security Group rules** to allow access only from EC2 via SSH Tunnel.

3. Security Groups

Create and configure Security Groups for:

- **EC2 instances**: Allow internal access and application ports.
- **RDS MySQL and PostgreSQL**: Allow only internal access via EC2 Security Group.
- **Load Balancer**: Allow only **ports 80 and 443** (HTTP and HTTPS).

4. Load Balancer

- Use **Application Load Balancer (ALB)**.
- Attach EC2 Auto Scaling Group instances to it.

- Forward HTTP (80) and HTTPS (443) to application containers.

5. Dockerized Application Deployment

- Deploy **multi-stage Dockerized applications** on **2 EC2 instances**.
- Use the sample app GitHub Repo or your own custom Frontend + Backend.

Tasks:

- Create a **multi-stage Dockerfile** and store it in your own GitHub repository.
- Ensure the app runs using Docker containers.
- Expose the app via the Load Balancer.
- Secure the app with **Domain** and **SSL (HTTPS)**.

6. Database Access and Initialization

- Access the RDS instances securely using an **SSH tunnel** through the EC2 instance.
- Use **DBeaver or another DB client** for connecting.
- Populate the DB with **dummy data** to demonstrate functionality.

7. BI Tool Deployment

- On the **3rd EC2 instance**, deploy a **Business Intelligence Tool** using Docker:
    - Choose between **Redash** or **Metabase**.
    - Connect it to either MySQL or PostgreSQL RDS.
    - Create a **sample dashboard**.
    - Demonstrate **live updates** when new entries are added to the DB.

References:

- Redash Docker Setup
- Metabase Docker Setup

8. Domain & SSL Configuration

- Point a custom **domain name** to the Load Balancer and BI tool instance.
- Use **Let's Encrypt** or **AWS ACM** SSL with Load Balancer to configure **SSL certificates**.
- Ensure HTTPS is enforced.

9. Deliverables

1. **Terraform Code** (with Modular File Structure):
    - Separate .tf files:
        - main.tf
        - ec2.tf
        - rds.tf
        - alb.tf

        - route53.tf
        - target_group.tf
        - security_groups.tf
        - outputs.tf
        - variables.tf
    - Follow best practices for modular and reusable code.
2. **GitHub Repository**:
    - Push the Terraform code and multi-stage Dockerfile to your GitHub.
    - Include README with setup instructions.
3. **Demonstration Video**:
    - Create a **Loom video** showing:
        - Infrastructure provisioning
        - Application deployment
        - Domain and SSL setup
        - DB access via tunnel
        - BI dashboard with real-time data updates
4. **Report (PDF)**:
    - Add the **GitHub Repo link**
    - Describe each component and what was implemented
    - Highlight key configurations (e.g., security groups, Dockerfiles)
    - Screenshots from Terraform Apply, DB client, and BI dashboards

Summary

You are expected to automate the provisioning of AWS infrastructure using Terraform, deploy a full-stack Dockerized application, configure SSL-secured Load Balancer and domain, set up private RDS databases, and connect everything to a BI tool that reflects live data updates.