Assignment 1 and 2

Title:

1. Design and Provision of a Multi-Server Django–PostgreSQL Application on AWS using Terraform and Ansible
2. Automation of Container Orchestration and CI/CD Deployment using Docker Swarm, Docker Compose, and GitHub Actions/Jenkins

Deadline:

1 week (20, October 2025)

Objective

Design, provision and fully automate deployment of a simple Django login/register web application backed by PostgreSQL. Provision AWS infrastructure (free-tier eligible), create Elastic IPs via Terraform, configure servers using Ansible, orchestrate containers with Docker Swarm + Docker Compose, and automate the full flow with a CI/CD pipeline (GitHub Actions or Jenkins).

Required functionality (application)

- **Django frontend**
  - Login page: inputs **username** (Roll No, e.g., **ITA700**) and **password** (Admission no, e.g., **2022PE0000**).
  - Register page: allow creating a row in DB (Roll no and Admission no).
  - Home page (after successful login) displays:
    Hello ITA700 How are you
    where **ITA700** comes from the **login.username** column in the database.
  - Logout button returns user to login page.
- **Database**
  - PostgreSQL database name: **postgres**.
  - Table: **login** with two columns: **username** and **password**.
  - **The initial table should be empty** (students must implement registration to add rows).
  - The app must authenticate against this **login** table.

Infrastructure (students must implement, Terraform-driven)

All instances must be free-tier eligible: Use **t2.micro** (or AWS free tier recommended AMI) and Ubuntu images (like Ubuntu 20.04 LTS used in lab manual examples). Use Terraform to create EC2 instances **and** attach Elastic IPs (EIPs) so servers have static public IPs (see example in the lab manual).

Servers:

1. **Controller** — Terraform + Ansible + CI runner (GitHub Actions runner or Jenkins agent). (t2.micro)
2. **Swarm Manager** — Docker, Docker Swarm manager node. (t2.micro + EIP)
3. **Swarm Worker A** — runs Django container replicas and/or PostgreSQL container replicas as needed. (t2.micro + EIP)
4. **Swarm Worker B** — same as Worker A (t2.micro + EIP)

Controller separates tooling (Terraform/Ansible/CI) from the cluster nodes and gives students a clean workflow to run Ansible against other servers. Two worker nodes + one manager give a realistic Swarm cluster where you can demonstrate replication, scaling and node drains using the patterns taught in the lab manual (e.g., **--replicas** flags).

Terraform requirements

- Create the EC2 instances (t2.micro) and associated keypair (auto-generated and saved to repo), security group that allows required ports (HTTP/HTTPS/SSH and any custom ports) — lab manual uses an open SG for labs; adopt at-least minimal restrictions but permit ports for testability.
- Create **Elastic IPs** and associate them to each EC2 instance (Terraform **aws_eip** example in manual). Output the public IPs after applying.
- Save the private key (PEM) locally via Terraform (example **local_file** saving key is in the manual).

Configuration & orchestration

Ansible

- Use the controller to run Ansible playbooks against manager & worker nodes. Playbooks must:

    - Install Docker, Docker Compose, Docker Swarm prerequisites (per manual).
    - Initialize the Swarm on the Manager node and join Worker A & B.
    - Deploy stacks/services **docker-compose.yml** adapted for Swarm.
- Use inventory files that reference the Elastic IPs (examples exist in manual).

Docker / Docker Swarm

- Build Docker images for web (Django) and **db** (Postgres image + init scripts). Provide Dockerfiles and a **docker-compose.yml** for stack deployment; for Swarm use **docker stack deploy -c docker-compose.yml myapp.**
- Run PostgreSQL as a service in Swarm. For this lab, simple single-write Postgres service replicated as a Swarm service is acceptable — students must handle DB persistence via Docker volumes.
- **Replicas:** the web service must run with **at least 2 replicas**. The manual already demonstrates replication commands.
- Ensure overlay network so the **web** can reach db by service name.

CI/CD

- **Jenkins:**
    - Create a **Jenkins pipeline job** (freestyle pipeline).
    - Configure a **GitHub Webhook** that triggers Jenkins automatically on every **push**.
    - Store credentials (e.g., AWS keys, SSH private key) securely in Jenkins credentials manager.
    - Use Jenkinsfile (declarative syntax) in the repository to define build stages.
- Students must push everything to a new branch named with their Roll No (e.g. **ITA700**) on the **public GitHub repo** named **DevOps_Assignment**
- Provide a single top-level script **bootstrap.sh** that:
    - Runs Terraform (**terraform init && terraform apply -auto-approve**) to create infra.
    - Uploads the generated PEM to local workspace (as Terraform saved).

    - Runs Ansible playbooks to configure servers and deploy the stack.
    - Triggers the initial GitHub Action run.

Submission (what to push to your RollNo branch)

- **terraform/** — all Terraform HCL files (including EIP resources and outputs) and **terraform-key.pem** creation logic.
- **ansible/** — inventory, playbooks, roles.
- **docker/** — Dockerfiles for **web** and **db**, **docker-compose.yml** / stack file.
- **django_app/** — Django project code with login/register/home/logout implemented using the **login** table in **postgres** DB.
- **scripts/** — **bootstrap.sh** (single-run script), helper scripts.
- **ci/** — GitHub Actions YAML or Jenkinsfile.
- **selenium/** — Selenium test script(s) (provided below).

Evaluation (100)

Terraform 20, Ansible 20, Docker Swarm 20, Django app 20, CI/CD 20