Deploying a Django Application on AWS

Infrastructure Setup

1. **AWS EC2 Instance**:
   - Host the Django application.
   - Configure security groups to allow SSH (port 22), HTTP (port 80), and HTTPS (port 443).
2. **AWS RDS (PostgreSQL)**:
   - Host the database.
   - Secure it within a private subnet.
3. **AWS S3**:
   - Store user-uploaded documents and images.
4. **AWS CloudFront**:
   - Serve static and media files.
5. **AWS IAM**:
   - Manage access to AWS resources securely.

Application Structure

1. **Django Application**:
   - **Models**: Define the data schema using Django models.
   - **Views**: Handle requests and responses.
   - **Serializers**: Convert complex data types to and from JSON.
   - **URLs**: Define the application routes.
   - **Templates**: Render HTML pages.
   - **Static and Media Files**: Store static assets and user uploads.
2. **Authentication**:
   - Use Djoser for user management.
   - Use JWT for token-based authentication.
3. **Third-Party Integrations**:
   - **OpenAI API**: For text summarization and image generation.
   - **Langchain and Hugging Face**: For language model operations.

Initial Setup and Configuration

Task	**Details**
AWS Setup	- Launch an EC2 instance with appropriate configurations.
	- Ensure security groups are configured to allow necessary traffic (SSH, HTTP, HTTPS).
Server Preparation	- Update and upgrade the server.
	- Install necessary packages.
Database Setup	- Use PostgreSQL for the database.
	- Create a database and user.
Storage Setup	- Use AWS S3 for storing documents and images.
	- Configure Django to use S3 for media files.

Install and Configure Dependencies

Task	**Details**
Django Project Setup	- Clone your Django project repository.
	- Set up a virtual environment and install dependencies.
Django Settings Configuration	- Configure Django settings for production, including database, static files, and media files.
	- Set up JWT, Djoser, Langchain, OpenAI, and Hugging Face settings.

Deploying a Django Application on AWS

Continuous Integration/Continuous Deployment (CI/CD)

Task Details
CI/CD with GitHub Actions - Create GitHub Actions workflows to automate testing and deployment.
- Use Docker to containerize the Django application.
- Deploy the application to the EC2 instance using GitHub Actions.

Detailed Timeline for Deploying Django Application on AWS—

Week 1: Initial Setup and Configuration

Day(s)	Task	Details	Estimated Time
Day 1-2	**AWS Setup**	Launch EC2 instance with appropriate configurations.	2 days
		Ensure security groups are configured to allow SSH (port 22), HTTP (port 80), and HTTPS (port 443).	
Day 3-4	**Server Preparation**	Update and upgrade the server.	2 days
		Install necessary packages (e.g., Python, Nginx, PostgreSQL client, etc.).	
Day 5	**Database Setup**	Set up PostgreSQL on AWS RDS.	1 day
		Create a database and user.	
Day 6	**Storage Setup**	Set up AWS S3 for storing documents and images.	1 day
		Configure Django to use S3 for media files.	

Week 2: Install and Configure Dependencies

Day(s)	Task	Details	Estimated Time
Day 1-2	**Django Project Setup**	Clone your Django project repository.	2 days
		Set up a virtual environment and install dependencies from `requirements.txt`.	
Day 3-4	**Django Settings Configuration**	Configure Django settings for production, including database, static files, and media files.	2 days
		Set up JWT, Djoser, Langchain, OpenAI, and Hugging Face settings.	
Day 5-6	**Initial Testing and Debugging**	Test the Django application locally and fix any issues.	2 days

Week 3: Continuous Integration/Continuous Deployment (CI/CD)

Day(s)	Task	Details	Estimated Time
Day 1-2	**Set Up GitHub Actions**	Create GitHub Actions workflows to automate testing and deployment.	2 days
Day 3	**Dockerize the Django**	Create Dockerfile and docker-compose configurations.	1 day

Day(s)	Task	Details	Estimated Time
	Application		
Day 4-5	**Configure Deployment Pipeline**	Set up deployment pipeline in GitHub Actions to deploy to EC2 instance.	2 days
Day 6	**Final Testing and Debugging**	Perform final tests on the deployed application and fix any issues.	1 day

Week 4: Finalization and Monitoring

Day(s)	Task	Details	Estimated Time
Day 1-2	**Set Up Monitoring and Logging**	Configure monitoring and logging for the Django application (e.g., AWS CloudWatch).	2 days
Day 3	**Security and Performance Optimization**	Optimize security settings (e.g., update security groups, ensure HTTPS) and performance tuning.	1 day
Day 4-5	**Documentation and Backup Plan**	Document the setup process, configurations, and create a backup plan.	2 days

Summary timeline: -

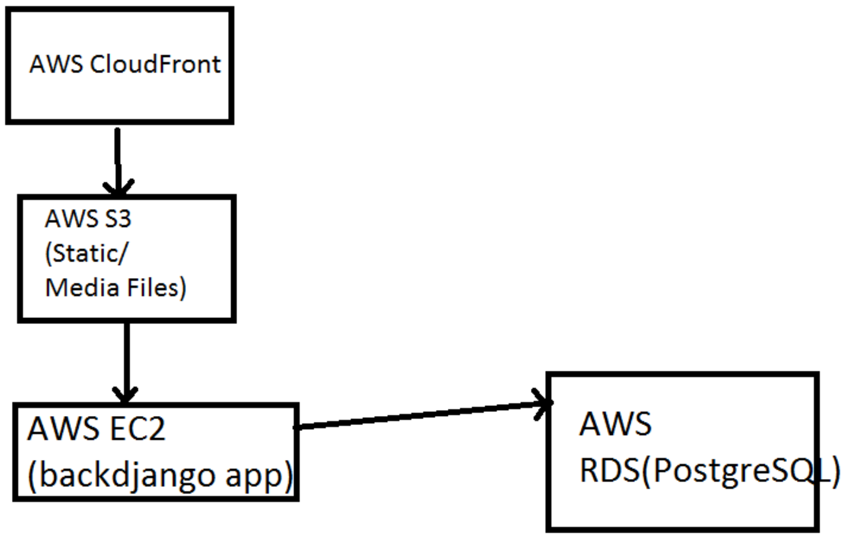
Week	Day(s)	Task	Estimated Time
Week 1	Day 1-2	AWS Setup: Launch EC2 instance, configure security groups	2 days
	Day 3-4	Server Preparation: Update/upgrade server, install necessary packages	2 days
	Day 5	Database Setup: Set up PostgreSQL on AWS RDS	1 day
	Day 6	Storage Setup: Set up AWS S3 for documents and images	1 day
Week 2	Day 1-2	Django Project Setup: Clone repo, set up virtual env, install dependencies	2 days
	Day 3-4	Django Settings Configuration: Configure for production	2 days
	Day 5-6	Initial Testing and Debugging: Local testing and issue fixing	2 days
Week 3	Day 1-2	Set Up GitHub Actions: CI/CD workflows	2 days
	Day 3	Dockerize Django Application: Dockerfile and docker-compose	1 day
	Day 4-5	Configure Deployment Pipeline: Deploy to EC2 instance	2 days
	Day 6	Final Testing and Debugging: On deployed application	1 day
Week 4	Day 1-2	Set Up Monitoring and Logging: AWS CloudWatch	2 days
	Day 3	Security and Performance Optimization	1 day
	Day 4-5	Documentation and Backup Plan: Setup process, configurations, backup plan	2 days

CI/CD Pipeline

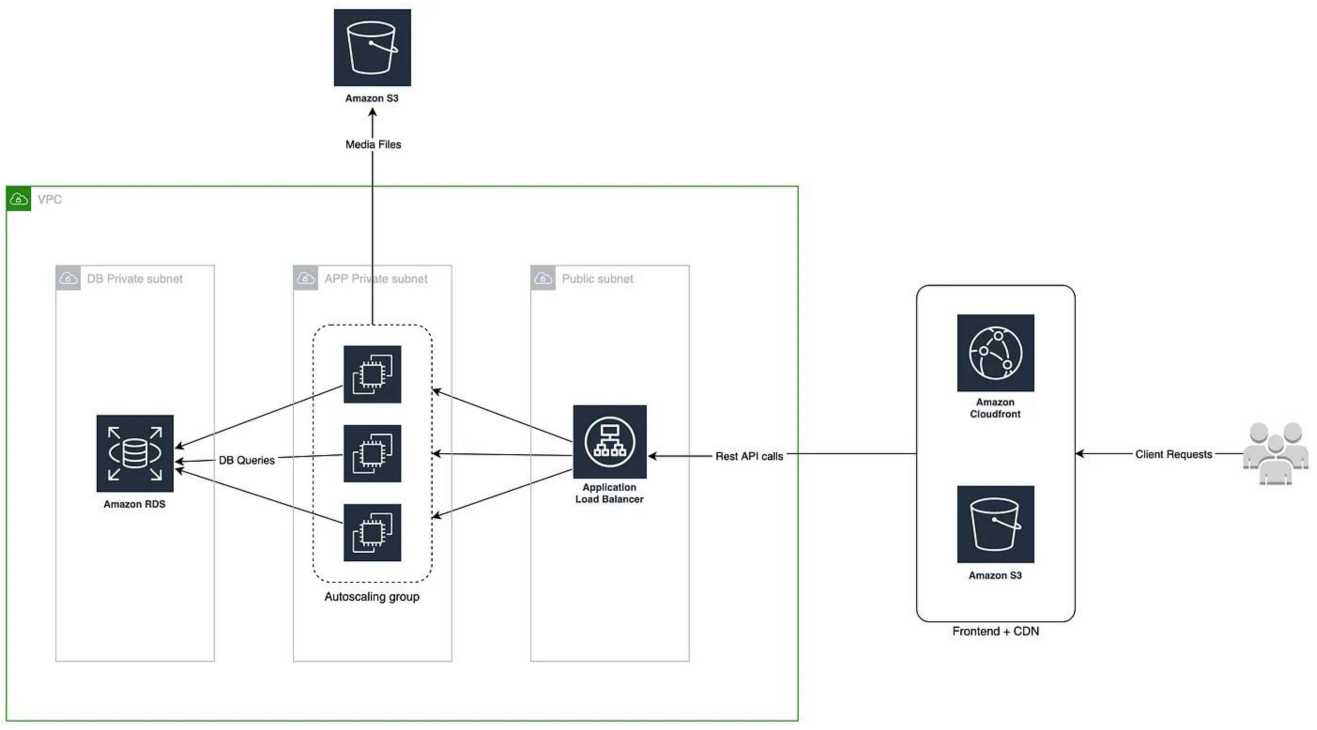
GitHub Actions:

- **Continuous Integration**: Automated testing on push to the main branch.
- **Continuous Deployment**: Automated deployment to AWS EC2 on successful tests.

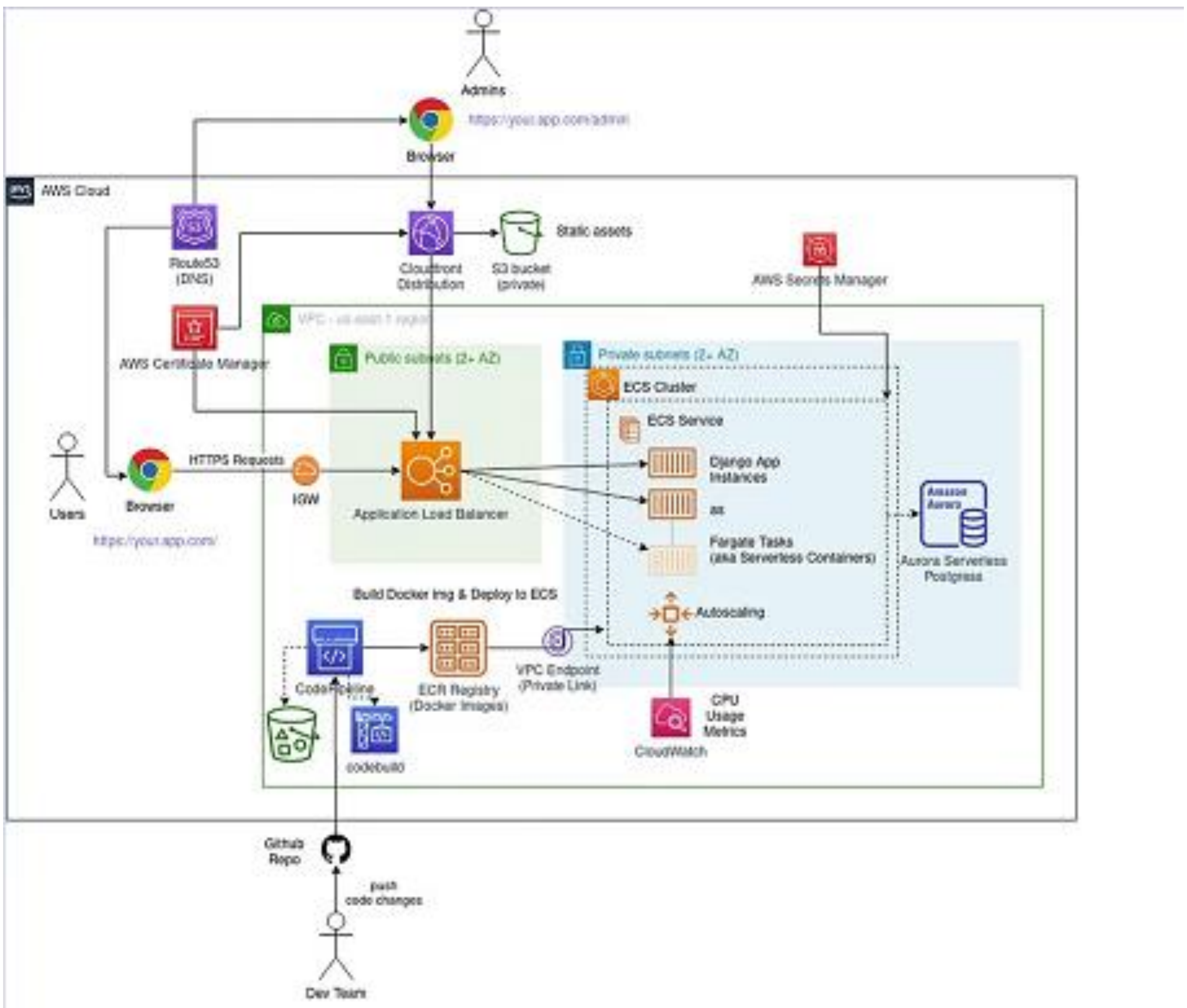
Deploying a Django Application on AWS



High level design :



Deploying a Django Application on AWS



= Django Settings Configuration

```
# settings.py
import os
from pathlib import Path
import dj_database_url
from dotenv import load_dotenv
load_dotenv()
BASE_DIR = Path(__file__).resolve().parent.parent

SECRET_KEY = os.getenv('SECRET_KEY')
DEBUG = os.getenv('DEBUG') == 'True'
ALLOWED_HOSTS = ['your-ec2-public-ip', 'your-domain.com']
DATABASES = {
    'default': dj_database_url.parse(os.getenv('DATABASE_URL'))
}
```

```
# Static and media files
STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'static')
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')

# AWS S3 settings
AWS_ACCESS_KEY_ID = os.getenv('AWS_ACCESS_KEY_ID')
AWS_SECRET_ACCESS_KEY= os.getenv('AWS_SECRET_ACCESS_KEY')
AWS_STORAGE_BUCKET_NAME = os.getenv('AWS_STORAGE_BUCKET_NAME')
AWS_S3_REGION_NAME = os.getenv('AWS_S3_REGION_NAME')
AWS_S3_SIGNATURE_VERSION = 's3v4'
AWS_DEFAULT_ACL = None

DEFAULT_FILE_STORAGE = 'storages.backends.s3boto3.S3Boto3Storage'

# Additional settings for Djoser, JWT, Langchain, OpenAI, Hugging Face
```

= GitHub Actions Workflow

```
name: Django CI/CD
on:
  push:
    branches:
      - main
jobs:
  build:
    runs-on: ubuntu-latest
    services:
      postgres:
        image: postgres:latest
        env:
          POSTGRES_DB: myproject
          POSTGRES_USER: myprojectuser
          POSTGRES_PASSWORD: password
        ports:
          - 5432:5432
    steps:
      - uses: actions/checkout@v2
      - name: Set up Python
        uses: actions/setup-python@v2
        with:
          python-version: '3.8'
      - name: Install dependencies
        run: |
          python -m venv venv
          source venv/bin/activate
          pip install -r requirements.txt

      - name: Run tests
        run: |
          source venv/bin/activate
          python manage.py test

  deploy:
    runs-on: ubuntu-latest
    needs: build
    steps:
      - uses: actions/checkout@v2
      - name: Set up SSH
        uses: webfactory/ssh-agent@v0.5.3
        with:
          ssh-private-key: ${{ secrets.SSH_PRIVATE_KEY }}
      - name: Deploy to EC2
        run: |
          ssh -o StrictHostKeyChecking=no ubuntu@your-ec2-public-ip 'cd /home/ubuntu/your-django-app && git pull origin main && source myprojectenv/bin/activate && pip install -r requirements.txt && python manage.py migrate && sudo systemctl restart gunicorn && sudo systemctl restart nginx'
```