# **Project Setup and Deployment Guide**

1. Building and running the project locally using Docker Compose.
2. Deploying the project to **Vercel**.

# **1. Build and Run the Project Locally Using Docker Compose**

### **Prerequisites**

Ensure you have the following installed on your local machine:

* **Docker** and **Docker Compose**: Install from [Docker's official website](https://www.docker.com/).
* **Git**: Install Git from [Git's official website](https://git-scm.com/).

### **Step 1: Clone the Repository**

Clone the repository containing the project code:

**git clone <repository-url>**

**cd <repository-directory>**

### **Step 2: Set Up the Folder Structure**

### **Step 3: Update the Environment Variables**

For local development, create .env files for each service to store environment variables.

#### **Example .env for User Service:**

**DATABASE\_URL**=postgres://postgres:password@user\_db:5432/user\_service\_db

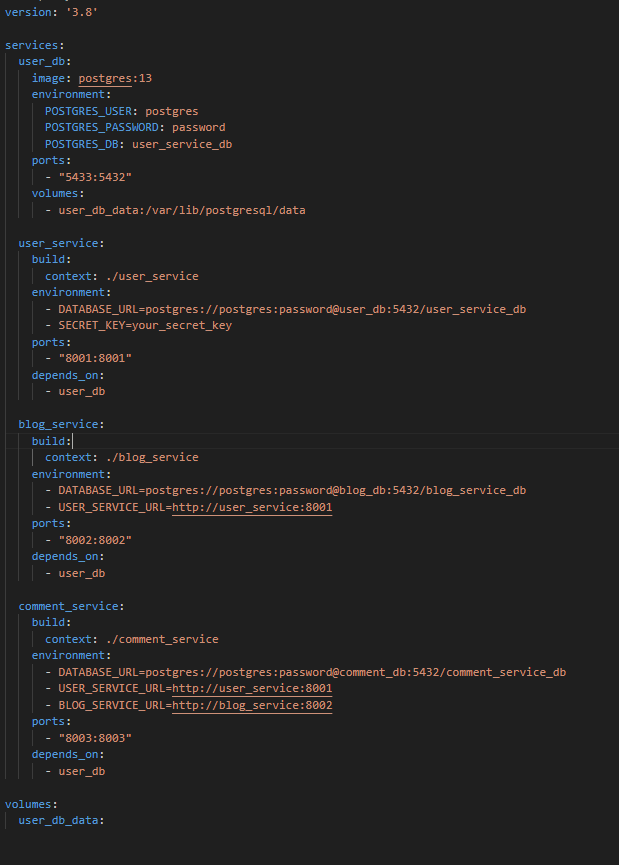
**SECRET\_KEY**=your\_secret\_key

### **Step 4: Write the Docker Compose File**

Ensure docker-compose.yml is configured to:

1. Include separate PostgreSQL instances for each service.
2. Define dependencies for each service.
3. Use environment variables for configurations.

#### **Example docker-compose.yml:**



### **Step 5: Build and Run the Project Locally**

Run the following commands to start the project locally:

docker-compose up --build

This will:

* Start PostgreSQL for each service.
* Build and run the services (user\_service, blog\_service, and comment\_service).

#### **Access the Services Locally:**

* User Service: [http://localhost:8001](http://localhost:8001/)
* Blog Service: [http://localhost:8002](http://localhost:8002/)
* Comment Service: [http://localhost:8003](http://localhost:8003/)

### **Step 6: Stop the Services**

To stop the services and free up resources, run:

docker-compose down

# **2. Deploy the Project to Vercel**

### **Step 1: Create a Vercel Account**

1. Sign up or log in to [Vercel](https://vercel.com/).
2. Connect your GitHub repository to Vercel.

### **Step 2: Prepare Each Service for Vercel Deployment**

Vercel works best for **serverless** setups. Since Django backends aren't serverless by default, you'll need to configure each service:

1. **Install the Vercel Python Runtime**: Add a vercel.json file for each service to define the deployment settings.

#### **Example vercel.json:**

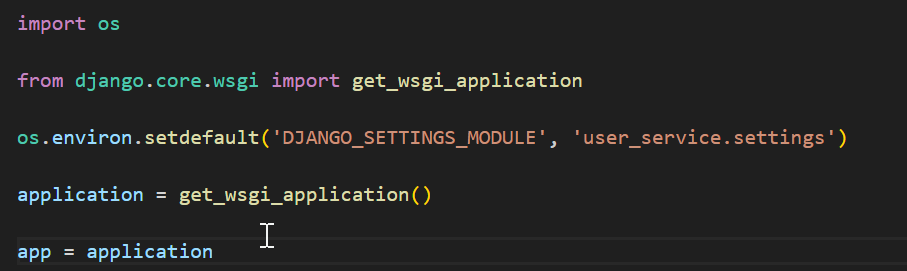


1. **Update settings.py for Deployment**:

Allow Vercel to access your services by updating ALLOWED\_HOSTS:  
 ALLOWED\_HOSTS = ['your-service-name.vercel.app']

* + Ensure CORS is configured correctly.

3.Change in wspi.py:

**Add the last line** 

**4.Push Changes to GitHub**: Commit and push all changes to the repository connected to Vercel.

### **Step 3: Deploy Each Service to Vercel**

1. Go to your Vercel dashboard.
2. Deploy each service separately by connecting its respective folder.
3. Vercel will automatically build and deploy the service.

#### **Example Service URLs:**

* User Service: https://user-service.vercel.app
* Blog Service: https://blog-service.vercel.app
* Comment Service: https://comment-service.vercel.app

### **Step 4: Test Inter-Service Communication**

Update the environment variables for each service to point to the deployed URLs.

#### **Example:**

For the **Comment Service**:

USER\_SERVICE\_URL=https://user-service.vercel.app

BLOG\_SERVICE\_URL=https://blog-service.vercel.app

Test workflows like:

1. User registration via the User Service.
2. Blog creation via the Blog Service.
3. Commenting via the Comment Service.

**Decisions made and Trade-offs:**

The original assignment required using AWS EC2 for service deployment and PostgreSQL databases. However, due to AWS not being free, I chose **Vercel** for deploying services and **Railway** for hosting PostgreSQL databases, leveraging their free tiers. This decision prioritized cost-effectiveness while maintaining reliable deployment and database management.

#### **Trade Offs:**

1. **Cost vs. Scalability**: AWS offers more scalability, but Vercel and Railway's free tiers suffice for development.
2. **Ease of Use vs. Control**: Vercel and Railway are simpler but offer less customization compared to AWS.
3. **Latency**: Distributed platforms may introduce slight delays compared to AWS's unified setup.

This approach ensures a balance between cost and functionality, suitable for initial development phases.