Task 2

1. **Write a brief overview of how you would redesign the blog system using a microservices architecture.**

    - Firstly, I would decouple the frontend from the system into a separate structure;
    - I would transform this web application into a REST application;
    - The next step would be to split the User management and Blog Post structure into two services;
    - I would containerize the entire project (Django services, databases, etc.), preparing this application for the next step, which would be the clustering stage with Kubernetes, enabling scalable deployment of the application.

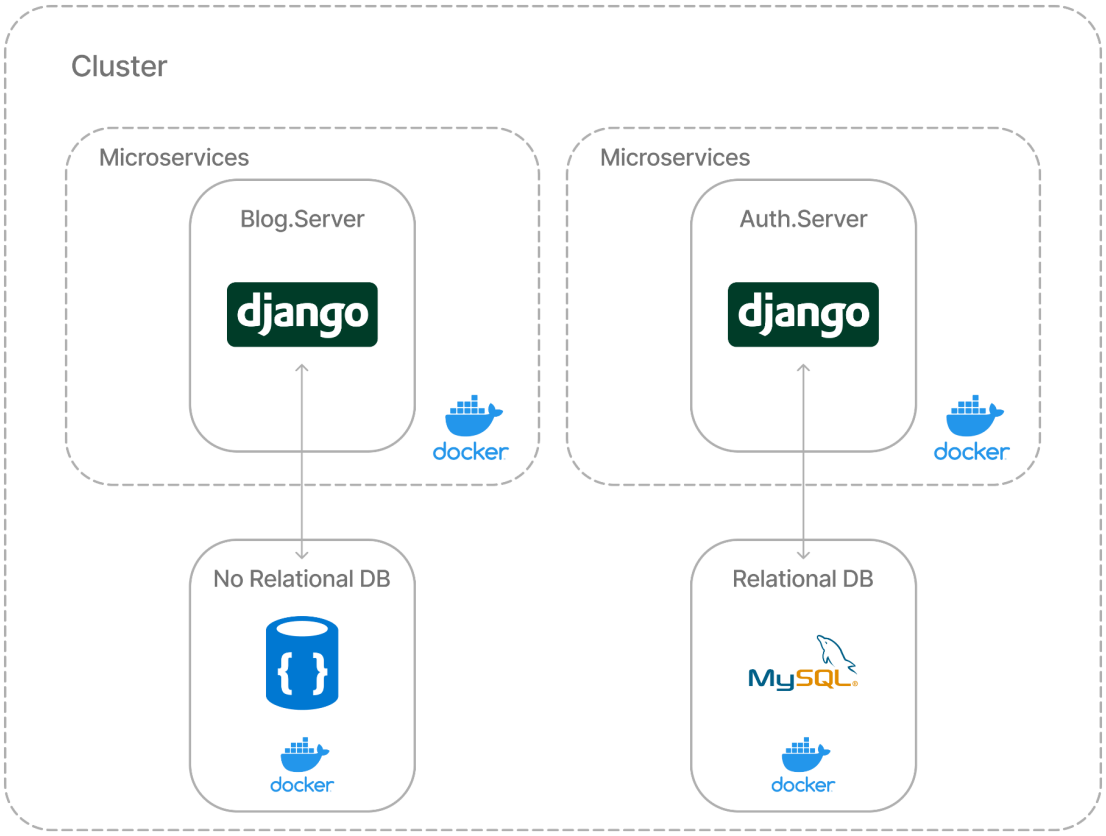
2. **How would you incorporate non-relational databases (e.g MongoDB/DynamoDB) into this architecture, if necessary? Provide a simple schema and the motivation to implement it.**

    User Structure

    - As the architecture is divided into microservices, the user service would be maintained with a relational database structure to leverage Django's ORM advantage in handling user management.

    Blog Structure

    - It would be migrated to the non-relational database, which focuses on flexibility, scalability, and performance.



- As the blog structure doesn't require much rigidity due to the nature of the data itself, opting for a non-relational database might be a wise choice. This is because this data has a significant volume and is subject to frequent changes, highlighting the advantages of using a non-relational database in this context.

Task 3

Provide some high-level documentation on how to deploy the Django application from Task 1 or Task 2 (the container) on an AWS service of your choice (e.g., EC2, EKS): - Steps taken to deploy the application.

Configuration settings

Step 1: Install AWS Copilot

Make sure you have AWS Copilot installed on your local machine. You can follow the installation guide provided by AWS Copilot

Step 2: Configure AWS Credentials

Ensure you have AWS credentials configured on your machine. You can do this by running `aws configure` and providing your AWS access key, secret key, region, and other required information.

Step 3: Initialize Copilot App

Navigate to the root directory of your project and run the following command to initialize a Copilot application:

copilot init

Follow the prompts to configure your Copilot application.

Step 4: Create Environments

Create the environments you want to deploy your services to. For example, you might have `test` and `prod` environments:

copilot env init --name test --profile <your-aws-profile>
copilot env init --name prod --profile <your-aws-profile>

Step 5: Deploy Services

Deploy your services to the environments you created. For each service (e.g., `django-auth`, `django-web-app`, `nginx`), navigate to its respective directory and run:

copilot svc deploy --name <service-name> --env <environment-name>

Step 6: Access Your Services

Once deployed, Copilot will provide you with the URLs to access your services. You can access the services through the provided URLs for each environment.

Additional Considerations:

- Copilot automatically handles tasks like load balancing, scaling, and service discovery.
- Monitor your services using AWS CloudWatch.

How to access the deployed application

After deploying your application using AWS Copilot, you can access your services through the URLs provided by Copilot. Here's how you can access your services:

1. View Copilot Service Information:

After deploying a service, Copilot will display information about the deployed service, including its URL. You can view this information using the following command:

copilot svc show --name <service-name> --env <environment-name>

Look for the "URL" field to find the endpoint where your service is accessible.

2. Access Services via Web Browser:

Open your web browser and navigate to the provided URL. For example, if your service is named `django-auth`, and you deployed it to the `test` environment, the URL might look like `http://django-auth.test.yourcopiloturl.com`.

Replace `<service-name>` and `<environment-name>` with the actual names you used during deployment.

3. Load Balancer URL (Nginx):

If you deployed an Nginx service, you might be using a load balancer. In this case, check the Copilot service information for the load balancer URL. Access your Nginx service through the load balancer URL.

Remember that the URLs depend on the names you provided during deployment and the environment you deployed to.