1. Dockerize the Streamlit App (Frontend)

Create a Dockerfile for Streamlit: Once we have Streamlit app running locally, We need to containerize it.
Here's an example `Dockerfile` for the Streamlit app:
dockerfile

```
# Use an official Python runtime as the base image
FROM python:3.12-slim

# Set working directory
WORKDIR /app

# Copy requirements file
COPY requirements.txt .

# Install dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Copy the rest of the application files
COPY . .

# Expose the port that Streamlit will run on
EXPOSE 8501

# Run the Streamlit app
CMD ["streamlit", "run", "app.py"]
```

1. **Build the Docker Image for Streamlit**:

```
docker build -t streamlit-app .
```

   ○
2. **Run the Streamlit App Container**:

After the build is complete, We can run the container:

```
docker run -d -p 8501:8501 streamlit-app
```

   ○

---

2. Dockerize the Backend APIs (Agents)

1. **We Create Dockerfiles for Each Agent**:,We wrap each agent into a RESTful API using **FastAPI** or **Flask**, then containerize each one.
2. **We Create a Docker Compose File for All Services**: Now, We''ll integrate the Streamlit app and the backend agents with the PostgreSQL pgvector database using Docker Compose.

Docker Compose :

it'a yaml file (here's an example of how it should look like)

```
version: "3.8"

services:
  streamlit:
    build:
      context: ./streamlit-app
    ports:
      - "8501:8501"
    depends_on:
      - pgvector
      - agent1
      - agent2

  agent1:
    build:
      context: ./agents/agent1
    ports:
      - "8001:8000"
    depends_on:
      - pgvector

  agent2:
    build:
      context: ./agents/agent2
    ports:
      - "8002:8000"
    depends_on:
      - pgvector

  pgvector:
    image: phidata/pgvector:16
    environment:
      POSTGRES_DB: ai
      POSTGRES_USER: ai
```

```
      POSTGRES_PASSWORD: ai
      PGDATA: /var/lib/postgresql/data/pgdata
    volumes:
      - pgvolume:/var/lib/postgresql/data
    ports:
      - "5532:5432"

volumes:
  pgvolume:
```

3. Running the Application with Docker Compose:

1. **Start All Services**:
   `docker-compose up --build`
   -
2. **Verify that the containers are running**:
   - We should have :
     - Streamlit frontend running on port `8501`.
     - Agents (e.g., agent1 and agent2) running on their respective ports.
     - PostgreSQL (pgvector) running on port `5532`.
3. **Testing**:
   - Open the browser and go to `http://localhost:8501` to see your Streamlit app in action.
   - Ensure that the agents are callable via their respective API endpoints.

---

4. Deploying to Azure Kubernetes Service (AKS)

Once everything works fine locally with Docker Compose, we can deploy to **Azure Kubernetes Service (AKS)**.

Steps to Deploy Using AKS:

1. **Set Up AKS**:

We Create an AKS cluster :

```
az aks create --resource-group <your-resource-group> --name
<your-cluster-name> --node-count 3 --enable-addons monitoring
--generate-ssh-keys
```

   -
2. **Create Kubernetes Manifests** for each service:

- Write YAML files for the deployment and service of the frontend, agents, and PostgreSQL.

Example `streamlit-deployment.yaml`:

yaml
Copy code

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: streamlit-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: streamlit
  template:
    metadata:
      labels:
        app: streamlit
    spec:
      containers:
      - name: streamlit
        image: streamlit-app-image # Use your image from Docker Hub
or Azure Container Registry
        ports:
        - containerPort: 8501
---
apiVersion: v1
kind: Service
metadata:
  name: streamlit-service
spec:
  selector:
    app: streamlit
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8501
  type: LoadBalancer
```

We Repeat this for `agent1`, `agent2`, and `pgvector`, adjusting the ports and images accordingly.

3. **Push Docker Images to a Container Registry**:

We Push Ourimages to **Docker Hub** or **Azure Container Registry**:

```
docker tag streamlit-app yourregistry/streamlit-app:v1
docker push yourregistry/streamlit-app:v1
```

   -
4. **Deploy to AKS**:

Apply all the Kubernetes manifests:

```
kubectl apply -f streamlit-deployment.yaml
kubectl apply -f agent1-deployment.yaml
kubectl apply -f agent2-deployment.yaml
kubectl apply -f pgvector-deployment.yaml
```

   -

---

5. CI/CD Pipeline with Jenkins

1. **Set up Jenkins** to automate the deployment process:
   - **Build pipeline** to:
     - Pull the code from GitHub.
     - Build Docker images for the Streamlit app, agents, and pgvector.
     - Push the images to Docker Hub or Azure Container Registry.
     - Deploy to AKS using `kubectl` commands.

---

6. Monitoring with Prometheus and Grafana

1. **Install Prometheus and Grafana on Kubernetes**:

Use Helm to install Prometheus and Grafana for monitoring.
bash

```
helm install prometheus prometheus-community/kube-prometheus-stack
```

   -
2. **Set Up Dashboards** in Grafana:
   - Create or import custom dashboards to monitor the performance of your pods (Streamlit app, agents, and PostgreSQL).

Conclusion:

We Have Now a process to:

1. Dockerize the frontend (Streamlit app), backend (agents), and database (pgvector).
2. Use Docker Compose to spin up all services locally.
3. Deploy everything to Azure Kubernetes Service (AKS).
4. Set up a CI/CD pipeline using Jenkins.
5. Set up monitoring using Prometheus and Grafana.