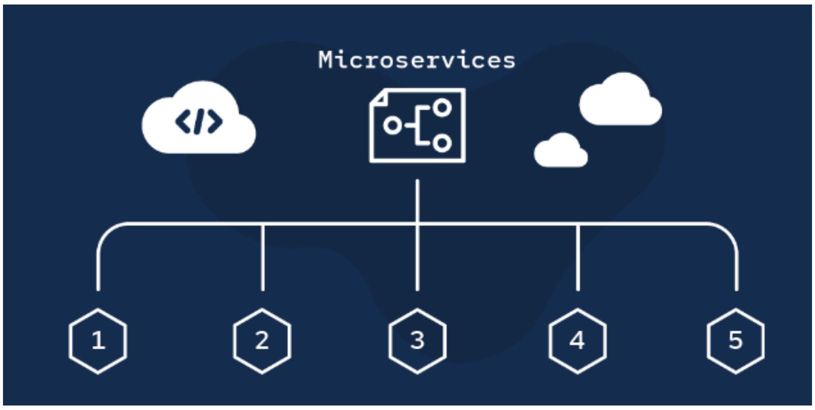
LAB 7

DISTRIBUTED SOFTWARE SYSTEMS

SCENARIO

Imagine you are a software developer tasked with building a basic microservices system for an e-commerce platform. The platform requires two essential microservices: a "User Service" responsible for managing user data, and an "Order Service" responsible for handling and managing orders.



Below is a step-by-step scenario based on the exercise requirements and how have been implemented:

STEP-BY-STEPS

Setup Development Environment:

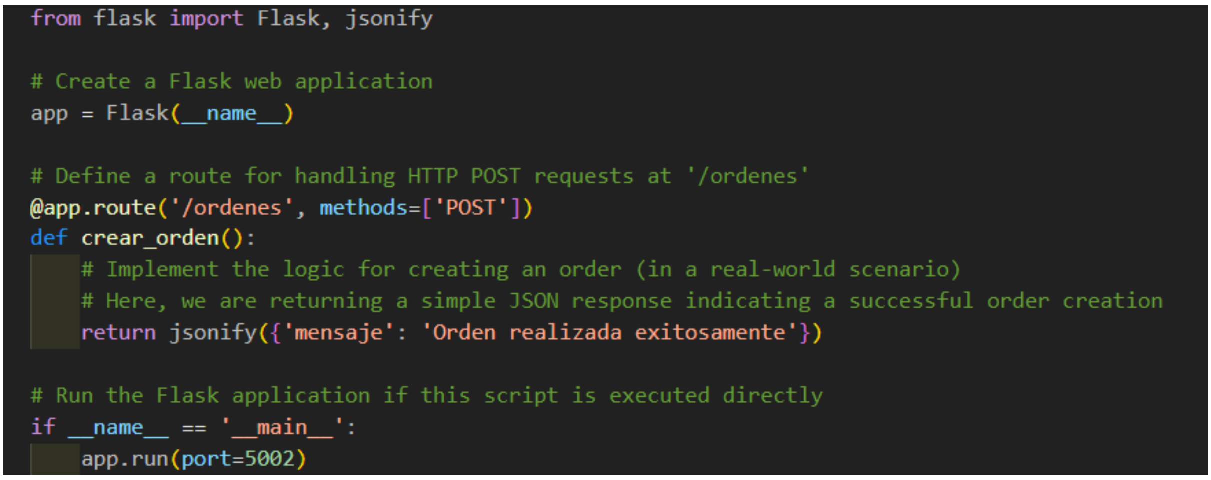
I have installed Docker and I will use Visual Studio Code as editor.

Define and implement Microservices:

The goal of this specific code is to create a web service (API endpoint) that listens for HTTP POST requests at the */ordenes* or */usuarios* endpoint. When a POST request is made to this endpoint, the *crear_orden* or *crear_usuario* function is executed, simulating the logic for creating an order. After processing, the server responds with a JSON message indicating that the order was successfully created.

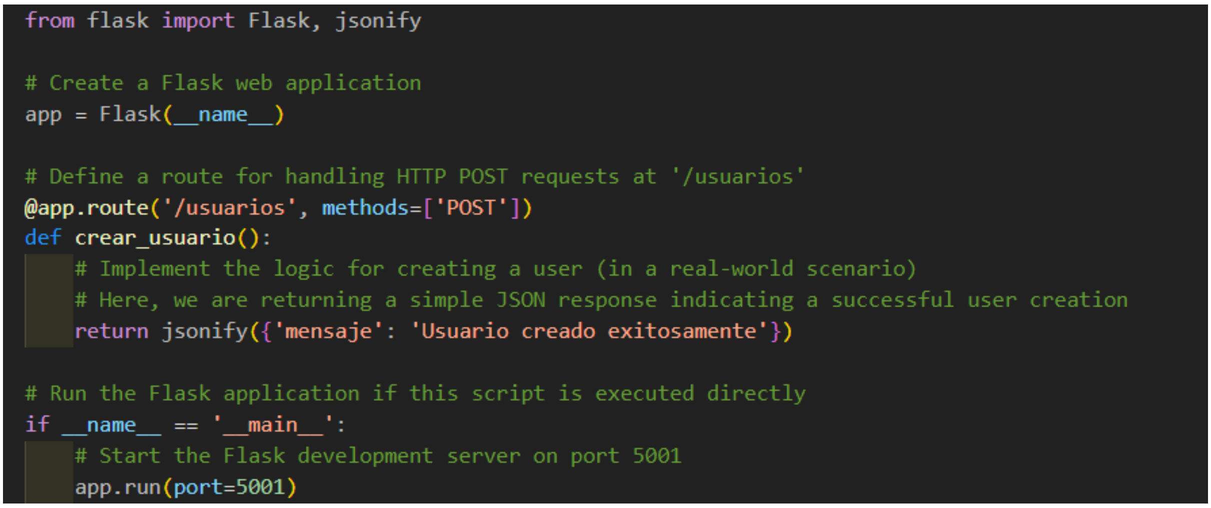
This code is a simple example of a Flask web application in Python:

"Order Service"



```python
from flask import Flask, jsonify

# Create a Flask web application
app = Flask(__name__)

# Define a route for handling HTTP POST requests at '/ordenes'
@app.route('/ordenes', methods=['POST'])
def crear_orden():
    # Implement the logic for creating an order (in a real-world scenario)
    # Here, we are returning a simple JSON response indicating a successful order creation
    return jsonify({'mensaje': 'Orden realizada exitosamente'})

# Run the Flask application if this script is executed directly
if __name__ == '__main__':
    app.run(port=5002)
```

"User Service"

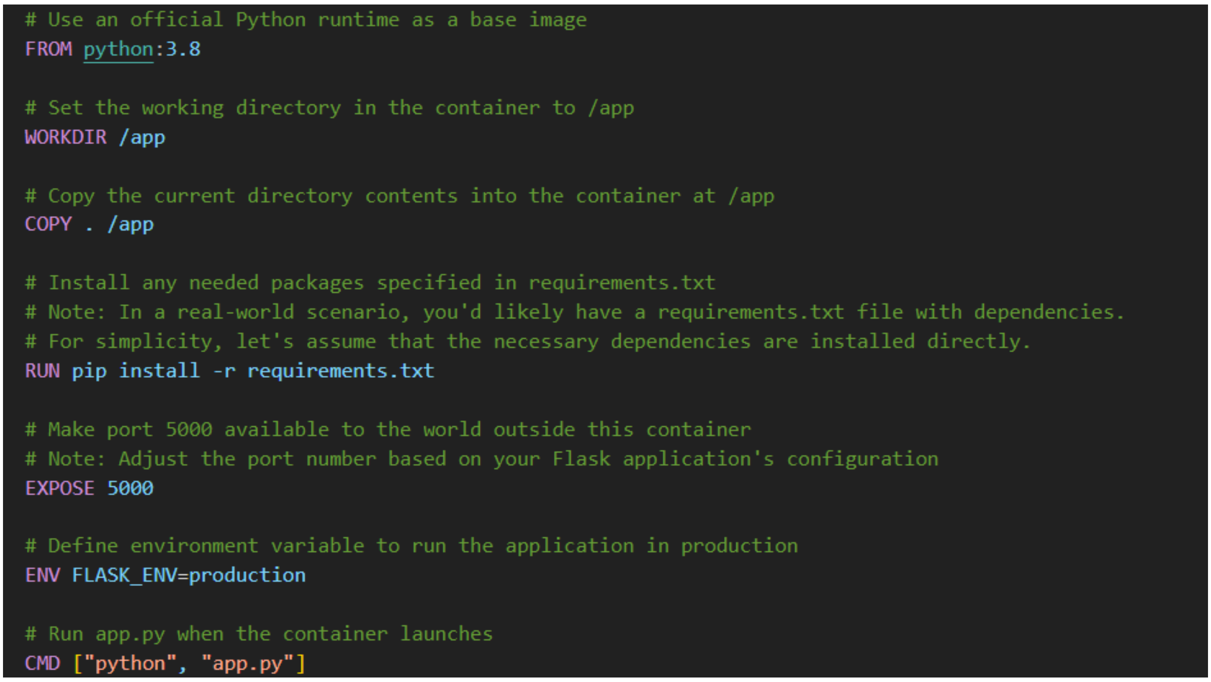


```python
from flask import Flask, jsonify

# Create a Flask web application
app = Flask(__name__)

# Define a route for handling HTTP POST requests at '/usuarios'
@app.route('/usuarios', methods=['POST'])
def crear_usuario():
    # Implement the logic for creating a user (in a real-world scenario)
    # Here, we are returning a simple JSON response indicating a successful user creation
    return jsonify({'mensaje': 'Usuario creado exitosamente'})

# Run the Flask application if this script is executed directly
if __name__ == '__main__':
    # Start the Flask development server on port 5001
    app.run(port=5001)
```

Containerize Services:

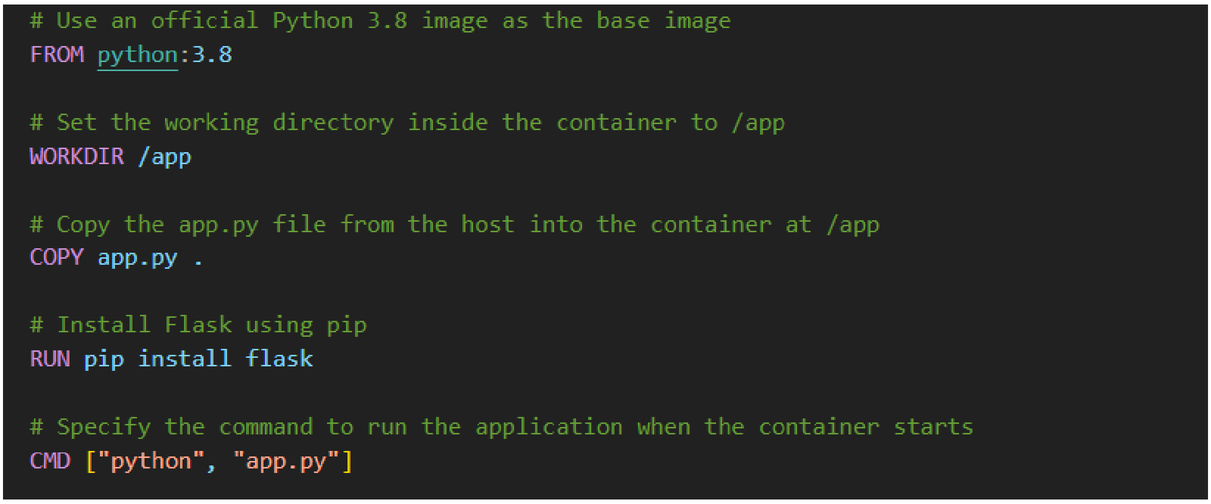
Now the overall goal is to create a Docker image that encapsulates the Flask application along with its dependencies, making it portable and easy to deploy across different environments using Docker.

"Order Service"



```dockerfile
# Use an official Python runtime as a base image
FROM python:3.8

# Set the working directory in the container to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install any needed packages specified in requirements.txt
# Note: In a real-world scenario, you'd likely have a requirements.txt file with dependencies.
# For simplicity, let's assume that the necessary dependencies are installed directly.
RUN pip install -r requirements.txt

# Make port 5000 available to the world outside this container
# Note: Adjust the port number based on your Flask application's configuration
EXPOSE 5000

# Define environment variable to run the application in production
ENV FLASK_ENV=production

# Run app.py when the container launches
CMD ["python", "app.py"]
```

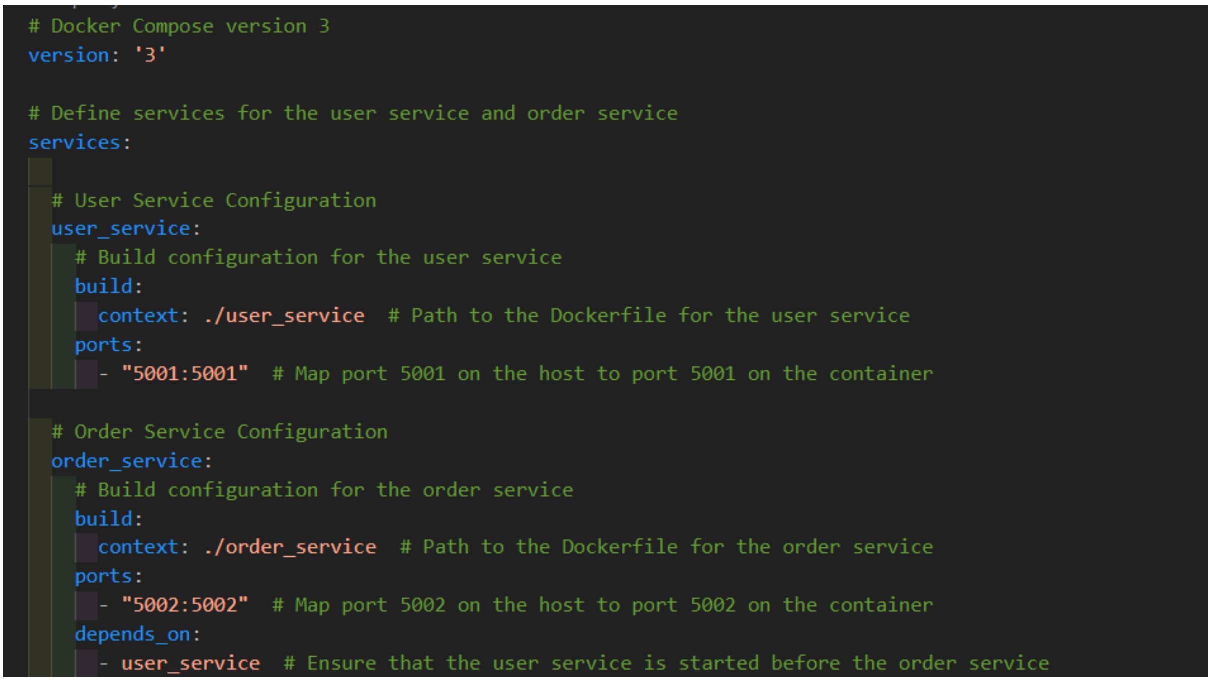
"User Service"



```dockerfile
# Use an official Python 3.8 image as the base image
FROM python:3.8

# Set the working directory inside the container to /app
WORKDIR /app

# Copy the app.py file from the host into the container at /app
COPY app.py .

# Install Flask using pip
RUN pip install flask

# Specify the command to run the application when the container starts
CMD ["python", "app.py"]
```

Deploy Services:

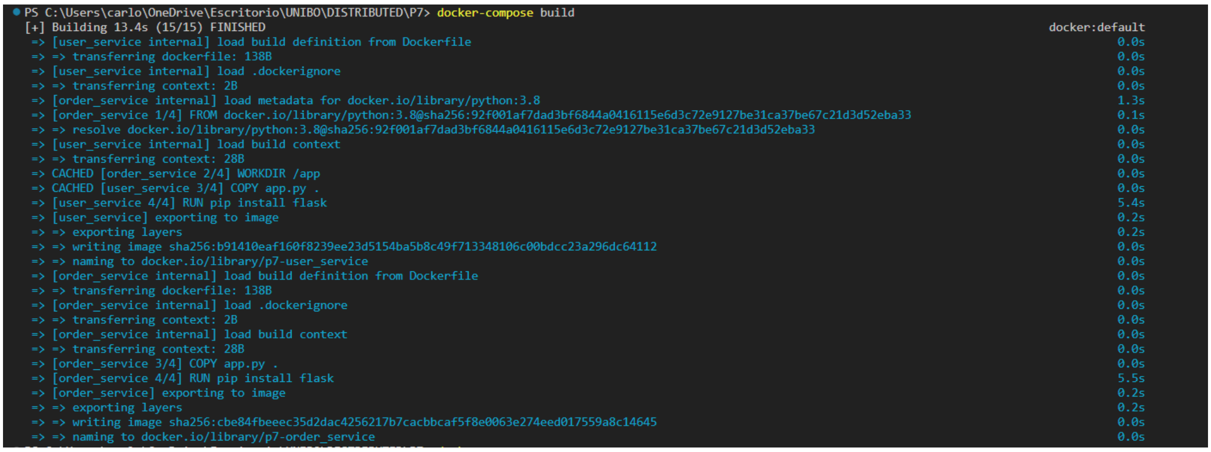
Now we will use Docker Compose to define a deployment configuration for both microservices and ensure communication between microservices is established.



```yaml
# Docker Compose version 3
version: '3'

# Define services for the user service and order service
services:

  # User Service Configuration
  user_service:
    # Build configuration for the user service
    build:
      context: ./user_service  # Path to the Dockerfile for the user service
    ports:
      - "5001:5001"  # Map port 5001 on the host to port 5001 on the container

  # Order Service Configuration
  order_service:
    # Build configuration for the order service
    build:
      context: ./order_service  # Path to the Dockerfile for the order service
    ports:
      - "5002:5002"  # Map port 5002 on the host to port 5002 on the container
    depends_on:
      - user_service  # Ensure that the user service is started before the order service
```

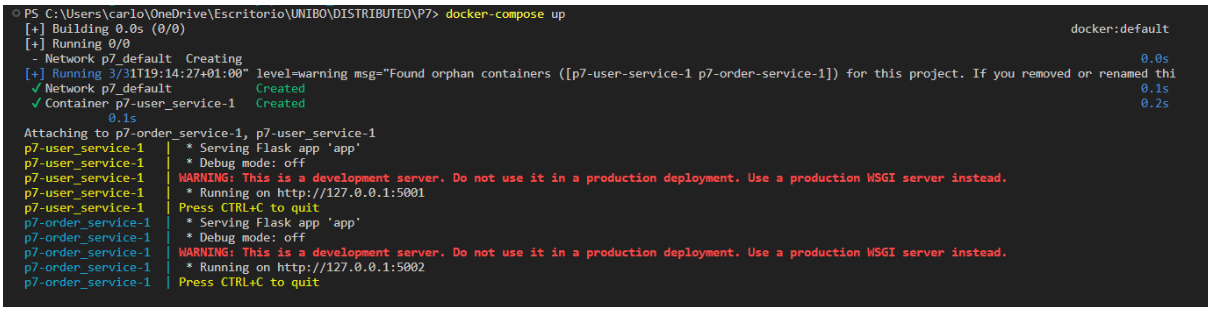
Test the System:

In the context of Docker we will use the build command to build a Docker image from the specified Dockerfile, which are "user_service" and "order_service" microservices.



```
PS C:\Users\carlo\OneDrive\Escritorio\UNIBO\DISTRIBUTED\P7> docker-compose build
 [+] Building 13.4s (15/15) FINISHED                                                                          docker:default
  => [user_service internal] load build definition from Dockerfile                                                      0.0s
  => => transferring dockerfile: 138B                                                                                   0.0s
  => [user_service internal] load .dockerignore                                                                         0.0s
  => => transferring context: 2B                                                                                        0.0s
  => [order_service internal] load metadata for docker.io/library/python:3.8                                            1.3s
  => [order_service 1/4] FROM docker.io/library/python:3.8@sha256:92f001af7dad3bf6844a0416115e6d3c72e9127be31ca37be67c21d3d52eba33   0.1s
  => => resolve docker.io/library/python:3.8@sha256:92f001af7dad3bf6844a0416115e6d3c72e9127be31ca37be67c21d3d52eba33    0.0s
  => [user_service internal] load build context                                                                         0.0s
  => => transferring context: 28B                                                                                       0.0s
  => CACHED [order_service 2/4] WORKDIR /app                                                                            0.0s
  => CACHED [user_service 3/4] COPY app.py .                                                                            0.0s
  => [user_service 4/4] RUN pip install flask                                                                           5.4s
  => [user_service] exporting to image                                                                                  0.2s
  => => exporting layers                                                                                                0.2s
  => => writing image sha256:b91410eaf160f8239ee23d5154ba5b8c49f713348106c00bdcc23a296dc64112                           0.0s
  => => naming to docker.io/library/p7-user_service                                                                     0.0s
  => [order_service internal] load build definition from Dockerfile                                                     0.0s
  => => transferring dockerfile: 138B                                                                                   0.0s
  => [order_service internal] load .dockerignore                                                                        0.0s
  => => transferring context: 2B                                                                                        0.0s
  => [order_service internal] load build context                                                                        0.0s
  => => transferring context: 28B                                                                                       0.0s
  => [order_service 3/4] COPY app.py .                                                                                  0.0s
  => [order_service 4/4] RUN pip install flask                                                                          5.5s
  => [order_service] exporting to image                                                                                 0.2s
  => => exporting layers                                                                                                0.2s
  => => writing image sha256:cbe84fbeeec35d2dac4256217b7cacbbcaf5f8e0063e274eed017559a8c14645                           0.0s
  => => naming to docker.io/library/p7-order_service                                                                    0.0s
```

Then we will use the docker-compose up command is to start and initialize the services defined in the docker-compose.yml



```
PS C:\Users\carlo\OneDrive\Escritorio\UNIBO\DISTRIBUTED\P7> docker-compose up
 [+] Building 0.0s (0/0)                                                                                                                  docker:default
 [+] Running 0/0
  - Network p7_default  Creating                                                                                                                    0.0s
 [+] Running 3/31T19:14:27+01:00" level=warning msg="Found orphan containers ([p7-user-service-1 p7-order-service-1]) for this project. If you removed or renamed thi
  ✓ Network p7_default           Created                                                                                                            0.1s
  ✓ Container p7-user_service-1   Created                                                                                                           0.2s
             0.1s
 Attaching to p7-order_service-1, p7-user_service-1
 p7-user_service-1    |  * Serving Flask app 'app'
 p7-user_service-1    |  * Debug mode: off
 p7-user_service-1    | WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 p7-user_service-1    |  * Running on http://127.0.0.1:5001
 p7-user_service-1    | Press CTRL+C to quit
 p7-order_service-1   |  * Serving Flask app 'app'
 p7-order_service-1   |  * Debug mode: off
 p7-order_service-1   | WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 p7-order_service-1   |  * Running on http://127.0.0.1:5002
 p7-order_service-1   | Press CTRL+C to quit
```

To test the interaction between the two services we will use the curl command, it can be tested by putting this ‘curl http.//127.0.0.1:5001/usuarios’ on the terminal.

Discuss Production Environment:

Monitoring:

- Implement robust logging and metrics collection within each microservice.
- Utilize centralized logging solutions and monitoring tools for tracking performance metrics.
- Set up alerting based on predefined thresholds to proactively detect and address issues.

Scaling:

- Consider horizontal scaling by running multiple instances of each microservice.
- Implement auto-scaling mechanisms to adjust the number of instances based on demand.
- Use load balancing strategies to evenly distribute incoming traffic across instances.

These considerations encompass monitoring for observability, scaling for handling varying workloads, and security for protecting against potential threats in a production environment.