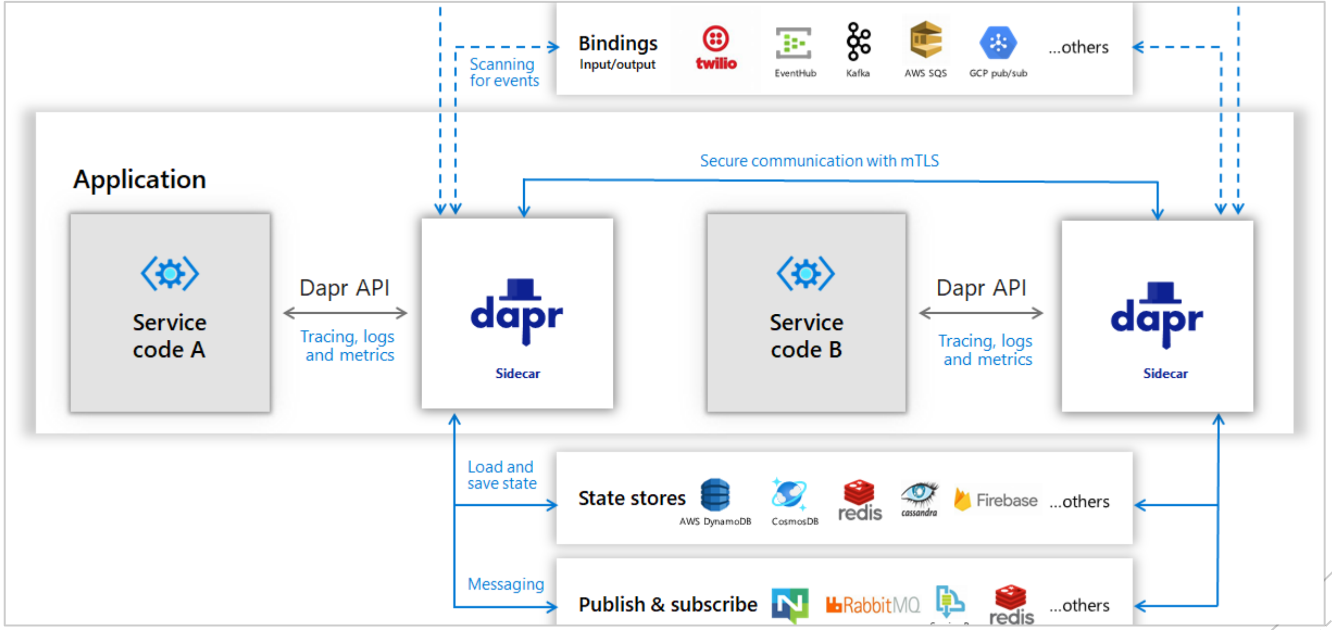
LAB-5

DISTRIBUTED SOFWTARE SYSTEMS

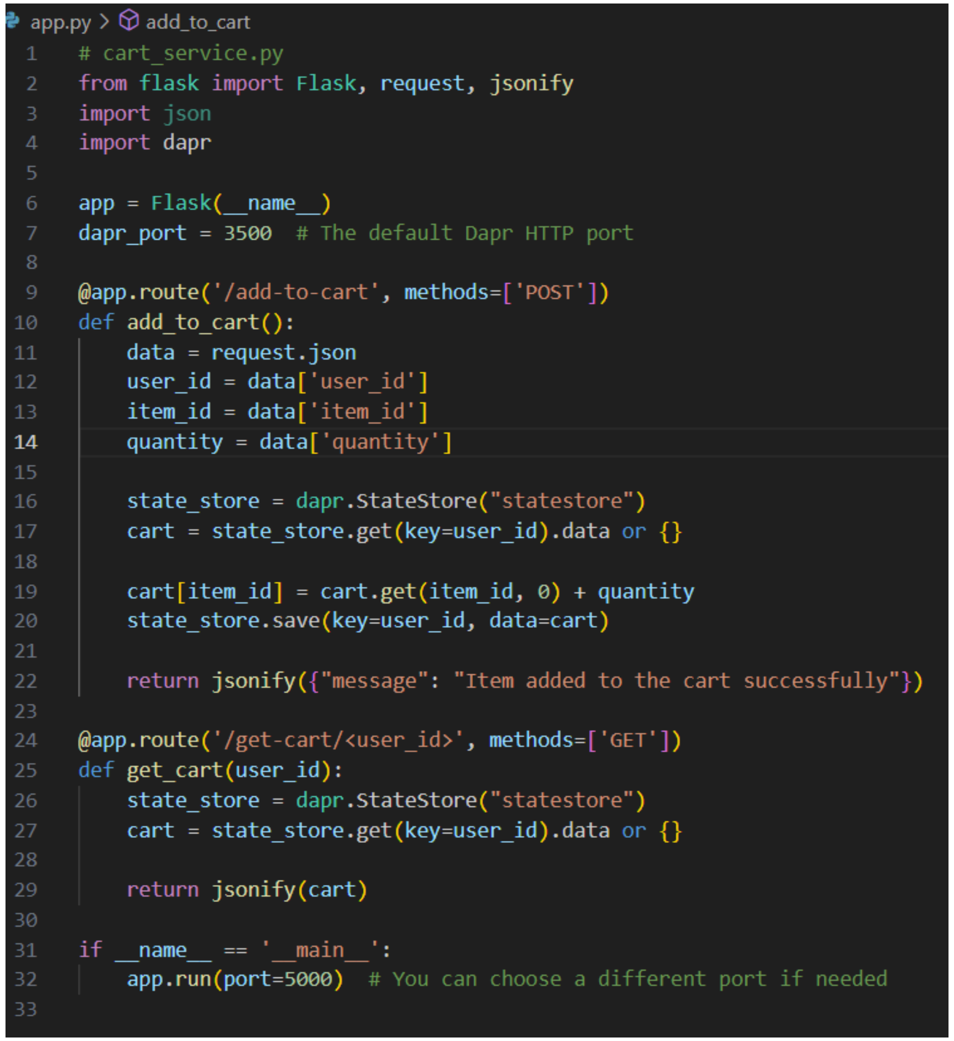
Scenario

- We will use DAPR to create a microservice architecture. This architecture will be a stateful service or an event driven system.
- In our case, we chose to build a microservice based on e-commerce application with a stateful service for managing user chopping carts.



STEP 1:

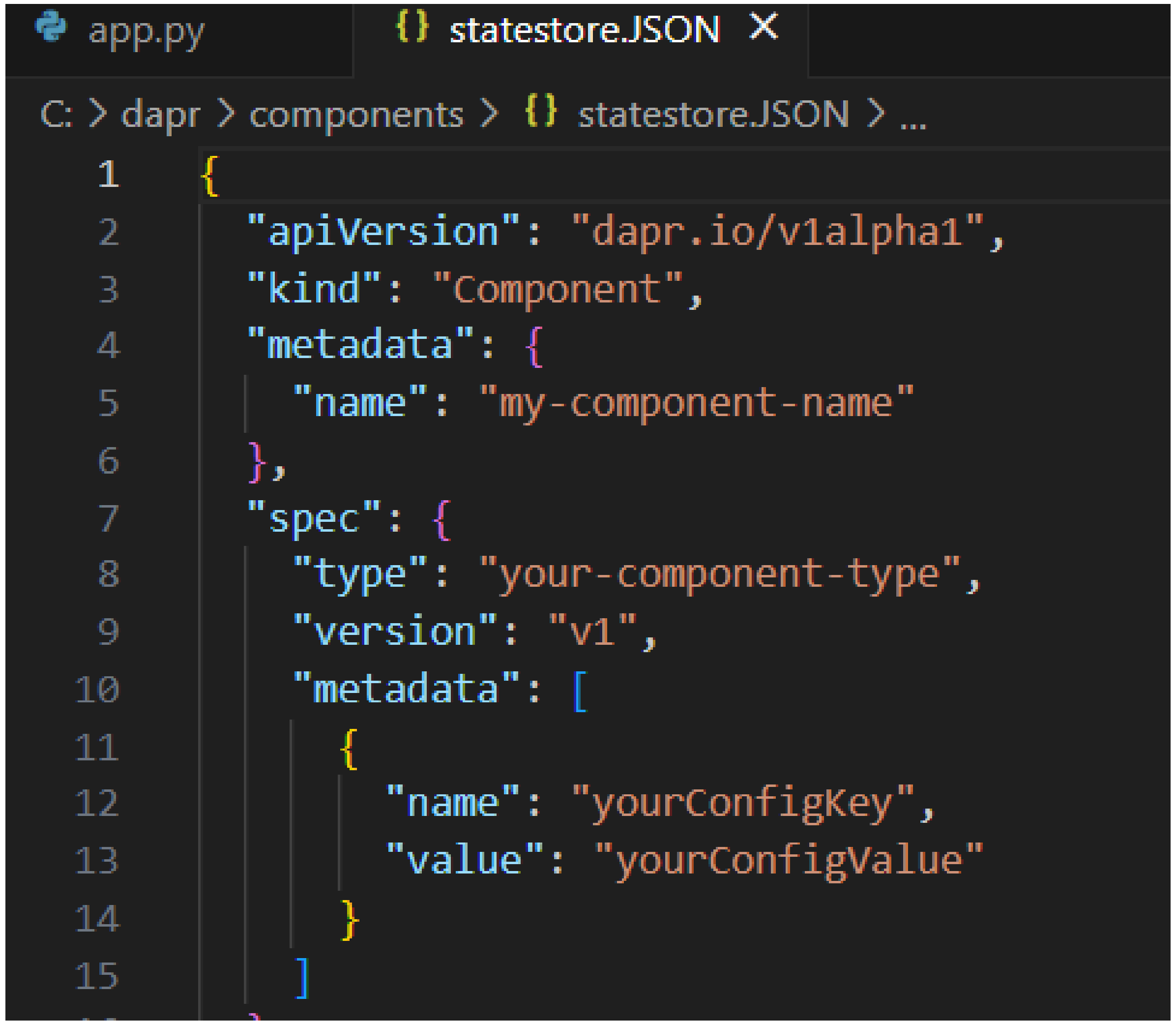
- Firstly we had to install DAPR from the command prompt using the following command: powershell -Command "iwr –useb https://raw.githubusercontent.com/dapr/cli/master/install/install.ps1 | iex“
- Then we needed to run the init CLI command by using “dapr init” and we had to verify the containers were running after this. To check this we had to use the next command: “docker ps”


app.py > add_to_cart

```python
# cart_service.py
from flask import Flask, request, jsonify
import json
import dapr

app = Flask(__name__)
dapr_port = 3500  # The default Dapr HTTP port

@app.route('/add-to-cart', methods=['POST'])
def add_to_cart():
    data = request.json
    user_id = data['user_id']
    item_id = data['item_id']
    quantity = data['quantity']

    state_store = dapr.StateStore("statestore")
    cart = state_store.get(key=user_id).data or {}

    cart[item_id] = cart.get(item_id, 0) + quantity
    state_store.save(key=user_id, data=cart)

    return jsonify({"message": "Item added to the cart successfully"})

@app.route('/get-cart/<user_id>', methods=['GET'])
def get_cart(user_id):
    state_store = dapr.StateStore("statestore")
    cart = state_store.get(key=user_id).data or {}

    return jsonify(cart)

if __name__ == '__main__':
    app.run(port=5000)  # You can choose a different port if needed
```

STEP 2:

- **Create a Stateful Service:** we will create a shopping cart service that stores user's shopping cart data in a state store.
- At the left you can see the python code for the service.



```json
{
  "apiVersion": "dapr.io/v1alpha1",
  "kind": "Component",
  "metadata": {
    "name": "my-component-name"
  },
  "spec": {
    "type": "your-component-type",
    "version": "v1",
    "metadata": [
      {
        "name": "yourConfigKey",
        "value": "yourConfigValue"
      }
    ]
```

STEP 3:

- **We also had to create DAPR sidecar configuration:** We had to configure the DAPR sidecar for our microservice in the “components” folder of our DAPR application. This folder was previously created by us.
- To do this, we created a JSON file that defined the state store component for storing shopping cart data.

STEP 4:

- **Testing and deployment:** To test and deploy our microserve into our environment, we tested the functionality of our shopping cart service.
- **To do so, we used the next command:** curl -X POST -H "Content-Type: application/json" -d '{"user_id": "123", "item_id": "item123", "quantity": 2}' http://localhost:5000/add-to-cart
- By using this command, our local server processes the request and responds correctly to it, by having a great shopping cart service funcionality .

STEP 5:

- **Stengths and limitations of DAPR:**
  - STRENGTHS:
    - **Abstraction of Infrastructure**: DAPR abstracts the underlying infrastructure, making it easier to develop and deploy microservices on various platforms.
    - **State Management**: DAPR simplifies state management by providing a consistent API for storing and retrieving data.
    - **Integration**: It offers built-in support for various communication patterns like pub/sub, service invocation, and state management.
  - LIMITATIONS:
    - **Learning Curve**: DAPR introduces a learning curve for developers who are new to it, and it may take time to fully understand its features.
    - **Performance Overhead**: While DAPR is designed for scalability, the sidecar pattern can introduce some performance overhead due to additional network hops.
    - **Ecosystem Maturity**: The ecosystem around DAPR, including available components and tools, may not be as mature as other technologies like Kubernetes.