a2-micro-25fa

Start Assignment

- Due Oct 7 by 11:59pm
- Points 15
- Submitting a website url
- Available after Sep 15 at 12am

What is microservices?

Microservices architecture (often shortened to microservices) refers to an architectural style for developing applications. Microservices allow a large application to be separated into smaller independent parts, with each part having its own realm of responsibility. To serve a single user request, a microservices-based application can call on many internal microservices to compose its response. [**reference: Google, "What is Microservices Architecture?" (https://cloud.google.com/learn/what-is-microservices-architecture?hl=en)**]

Objective

- You are going to develop and deploy a toy example of microservices.
- You will learn how to deploy services on cloud, using Docker containers, an immensely popular software containerization technology

High-level Architecture

- Aggregator: single public entrypoint. Users sends a productID to this aggregator service through API.
- Item Info Service: owns product metadata (productID and the corresponding name string).
- Stock Service: owns inventory counts (productID and the corresponding count integer).

```
Client
  |
  | 1. POST /lookup { productID }
  v
Aggregator
  |\
  | \-- 2a. GET /item-info/items/{productID}
  |       -> { id, name string }
  |
```

```
  | \-- 2b. GET /stock-info/items/{productID}
  |       -> { id, count integer }
  |
  | 3. Merge + check errors
  v
Client
```

API Definitions

Aggregator

```
POST /lookup
Body:
{ "productID": "XYZ-12345" }

200
{
    "productID": "XYZ-12345",
    "name": "Amayon Basics Wipes",
    "available": 210
}

404
{ "error": "Product not found" }
```

2a: Item Info Service

```
GET /stock-info/items/{productID}

200
{
    "productID": "XYZ-12345",
    "name": "Amayon Basics Wipes"
}

404
{ "error": "Product not found" }
```

2b: Stock Service

```
GET /stock-info/items/{productID}

200
```

```
{
    "productID": "XYZ-12345",
    "available": "210"
}

404
{ "error": "Product not found" }
```

Final Aggregation

- If Item Info = 404, return 404 to client. (No product.)
- If Item Info = 200 and Stock = 200, return merged information.
- If Item Info = 200 and Stock = 404, return 200 with `available: 0`.

Pseudo Database

- You can prepare a simple CSV file containing dummy entries of a few productIDs and names for the Item Info Service.
- When you run the program, it should read the CSV file and use the data as a database. (In reality, the program sends a query to an actual database.)
- Do the same for the Stock Service. Its CSV file should have a few product IDs and counts.
- In both tables, you must include:
  - `"productID": "XYZ-12345", "name": "Amayon Basics Wipes"`
  - `"productID": "XYZ-12345", "available": "210"`

Tasks

- Implement four python files
  - client.py: sending a productID to an aggregator. (Look at the user.py in the demo in class)
  - aggregator.py
  - iteminfo.py
  - stockinfo.py
- Prepare three Docker files and main.py files that define APIs.
  - aggregator.py
  - iteminfo.py
  - stockinfo.py

- Deploy the three containers on Cloud Run, each for the python code

Submission Guide

- Use the GitHub classroom assignment: **https://classroom.github.com/a/W4tgy1bp (https://classroom.github.com/a/W4tgy1bp)** (See the repo structure below)
- Write a sample URL to test your system on cloud. (e.g., `https://pc-611571974386.us-west2.run.app/lookup`) in a file named `myURL.txt`
- Using your `client.py`, cover all scenarios listed in the "Final Aggregation" Section.
- Copy-paste the outputs from the trials to `results.txt`. (Make sure to annotate the file for all scenarios.)
- We will have an evaluation demo on Oct 8 during the class.

Repo Structure

```
repo-root/
├── clientd/
│   └── client.py
│
├── aggregatord/
│   ├── Dockerfile
│   └── app/
│       ├── main.py
│       └── model/
│           └── aggregator.py
│
├── stockinfod/
│   ├── Dockerfile
│   └── app/
│       ├── main.py
│       └── model/
│           └── stockinfo.py
│
├── iteminfod/
│   ├── Dockerfile
│   └── app/
│       ├── main.py
│       └── model/
│           └── iteminfo.py
│
├── results.txt
└── myURL.txt
```