SERVICE ORIENTED ARCHITECTURES

ACIT3855 – WINTER 2024

AGENDA

- Quick Review
- Quiz 4
- Overview of Processing Service
- Lab 5
  - Processing Service

REVIEW

- What would be examples of metadata in an OpenAPI specification?
- What do the paths represent?
- Within a path, what would we use the post and get keywords for? How are these different?
- How do associate the OpenAPI specification with our code?
- Why do we define component schemas in an OpenAPI specification?

QUIZ 4

- Quiz is on the Learning Hub
- Open book, you may refer to the reading materials
- You have <15 minutes to complete it

COURSE SCHEDULE

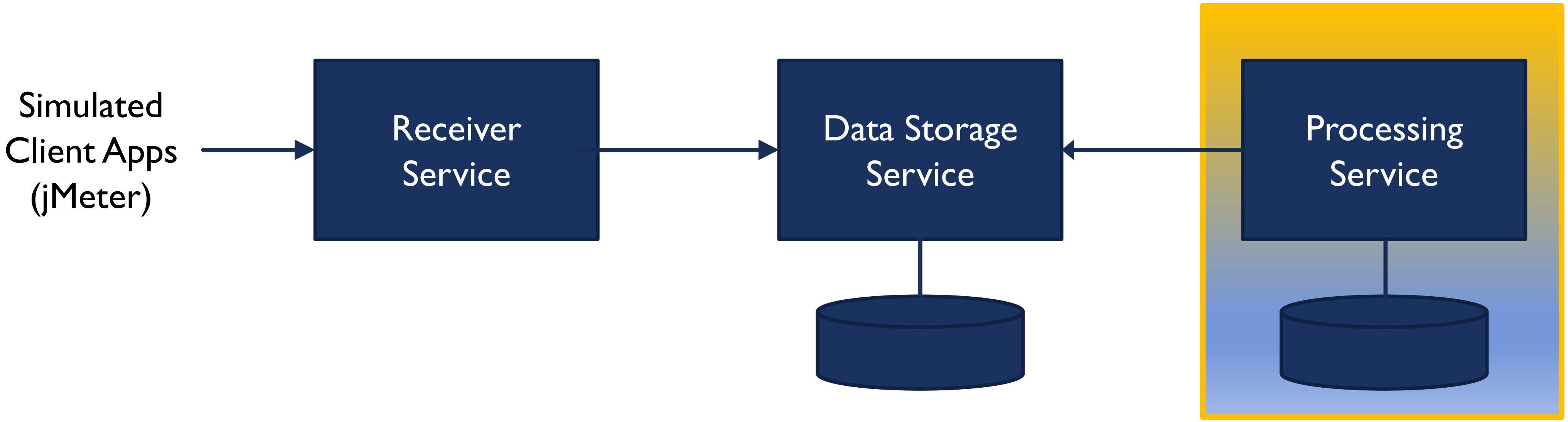
Week	Topics	Notes
1	• Services Based Architecture Overview • RESTful APIs Review	Lab 1
2	• Microservices Overview • Edge Service	Lab 2, Quiz 1
3	• Database Per Service • Storage Service (SQLite)	Lab 3, Quiz 2
4	• Logging, Debugging and Configuration • Storage Service (MySQL)	Lab 4, Quiz 3
5	• RESTful API Specification (OpenAPI) • Processing Service	Lab 5, Quiz 4
6	• Synchronous vs Asynchronous Communication • Message Broker Setup, Messaging and Event Sourcing	Lab 6, Quiz 5, Assignment 1 Due
7	• Deployment - Containerization of Services *Note: At home lab for Monday Set*	Lab 7, Quiz 6 (Sets A and B)
8	• Midterm Week	Midterm Review Quiz
9	• Dashboard UI and CORS	Lab 8, Quiz 6 (Set C), Quiz 7
10	• Spring Break	No Class
11	• Issues and Technical Debt	Lab 9, Quiz 8
12	• Deployment – Centralized Configuration and Logging	Lab 10, Quiz 9
13	• Deployment – Load Balancing and Scaling *Note: At home lab for Monday Set*	Lab 11, Quiz 10 (Sets A and B)
14	• Final Exam Preview	Quiz 10 (Set C), Assignment 2 Due
15	• Final Exam	

OUR SAMPLE APPLICATION

Our sample application will have three initial services:

- Receiver Service (Lab 2)
- Storage Service (Lab 3)
- Processing Service (Lab 5)

Add New Processing Service



Includes:

- RESTful API
- External Config
- Logging
- SQLite Database
- Periodic Processing

OPENAPI SPECIFICATION

What are the benefits?

- Allows you to do upfront design that can easily be reviewed by others
- It can be committed to a source code repository for collaboration and audit
- Generate documentation for users of the API
- Generate code (clients and/or server)
- Validate requests and responses against the specification

Generally each Microservice will have its own API specification (independent of other services).

For Edge Services, like our Receiver Service, that API specification (and documentation) may be provided to external clients of that API.

WALKTHROUGH OF AN OPENAPI SPECIFICATION

- Let’s walkthrough the OpenAPI Specification of a sample Data Storage Service
    - Metadata
    - Resources and HTTP Methods (POST and GET)
        - Request Messages and Parameters
        - Response Messages
    - Component Schemas

PROCESSING SERVICE

- Sometimes we need a service that processes data or events (for example: anomaly detection, reporting)
- The service can be designed to do its processing upon receipt of new data or events
- However, sometimes it's more efficient to process the data or events in batches, especially if there are no requirements for "real-time" processing
- Our **Processing Service** will periodically process "batches" of recently received events
  - Its processing will be to calculate statistics on those events (i.e., number received, min/max value received, etc.)
  - It will store those statistics and provide an API to retrieve them

PERIODIC PROCESSING

- Sometimes we want to perform the same task on a scheduled basis (i.e., every few seconds, minutes, days, etc)
- Python has a module, apscheduler, that let's you create background processes
- This will not block your RESTful APIs

```
from apscheduler.schedulers.background import BackgroundScheduler


def background_job():
    """ Periodically do something """
    print("Doing something")

def init_scheduler():
    sched = BackgroundScheduler(daemon=True)
    sched.add_job(background_job, 'interval',seconds=30)
    sched.start()
```

This is somewhat similar to a cron job you may be familiar with where you can schedule a task to run on a schedule.

We'll briefly demo apscheduler now

OUR PROCESSING SERVICE

Data Store Updates:

- Add two new GET endpoints, one per event type, that will query for all events with a date_created timestamp greater than or equal to a given timestamp provided as a parameter on the endpoint.

Processing Service:

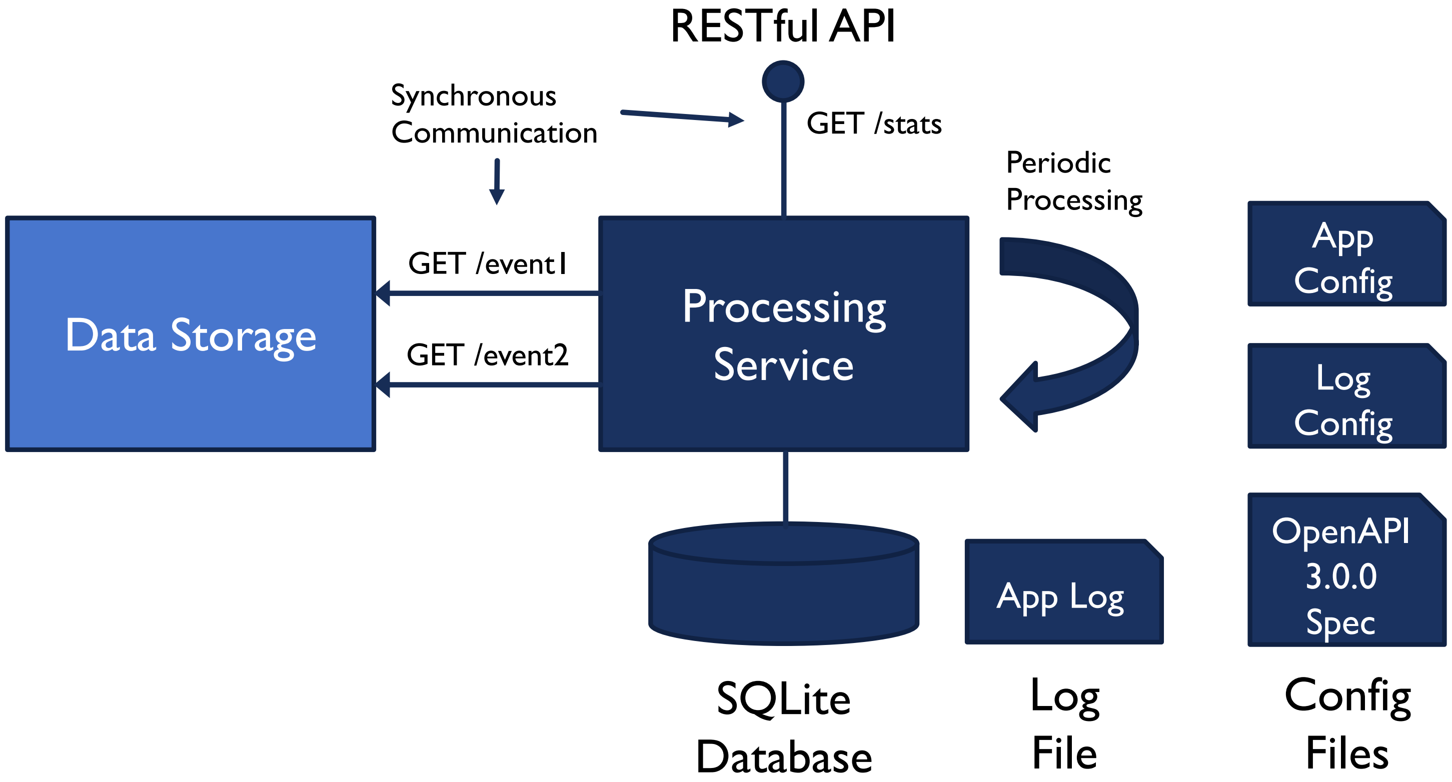
- Includes a RESTful API (with an OpenAPI Spec), logging, external configurations (app and log) and makes API requests to another service (the Data Storage Service). It adds periodic processing as a background scheduled task.
- Periodically polls the Data Store, using the two GET endpoints for retrieving new events in the datastore by passing in a timestamp parameter of the last time it requested those events
- It uses those events to gather some statistics (i.e., number of event1, number of event2, max event1, max event2)
  - The statistics should be cumulative
- It writes those statistics in a SQLite database, along with the timestamp of the last time it retrieved new events
- It provides a GET endpoint to retrieve the current values of the statistics

OUR PROCESSING SERVICE

Putting Everything Together in the **Processing Service**:

- Synchonous Communication – Providing and Using RESTful APIs using OpenAPI 3.0.0, connexion and requests
- Separate Database (SQLite)
- Configuration and Logging

And adds periodic processing (i.e., a scheduled task) to our application.



OUR PROCESSING SERVICE

- The statistics will be stored as rows in a stats table in a SQLite DB
- Every time the stats are calculated, a new row is stored with a last_updated timstamp

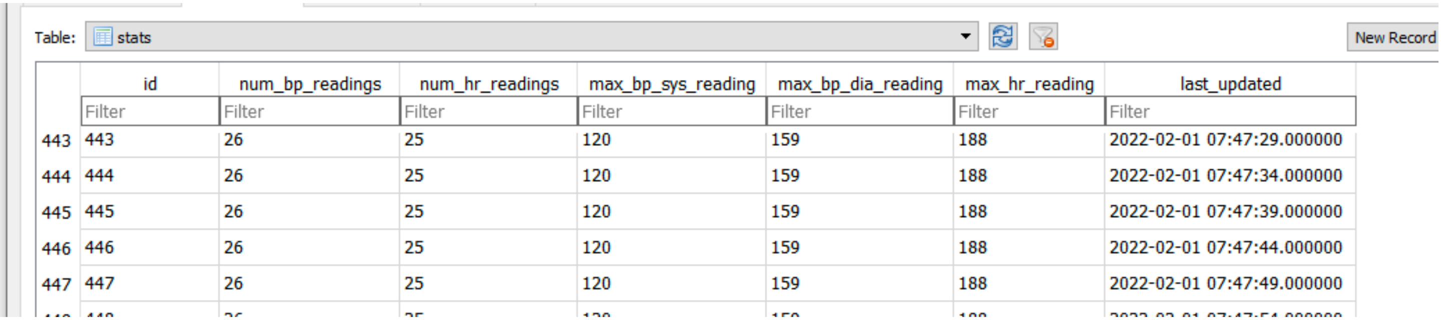


Table: stats

	id	num_bp_readings	num_hr_readings	max_bp_sys_reading	max_bp_dia_reading	max_hr_reading	last_updated
	Filter	Filter	Filter	Filter	Filter	Filter	Filter
443	443	26	25	120	159	188	2022-02-01 07:47:29.000000
444	444	26	25	120	159	188	2022-02-01 07:47:34.000000
445	445	26	25	120	159	188	2022-02-01 07:47:39.000000
446	446	26	25	120	159	188	2022-02-01 07:47:44.000000
447	447	26	25	120	159	188	2022-02-01 07:47:49.000000
448	448	26	25	120	159	188	2022-02-01 07:47:54.000000

It must have the timestamp of the last time you calculated the statistics (last_updated).

This is needed to query for new events from the Storage Service and so we can find the latest stats in the DB

OUR PROCESSING SERVICE

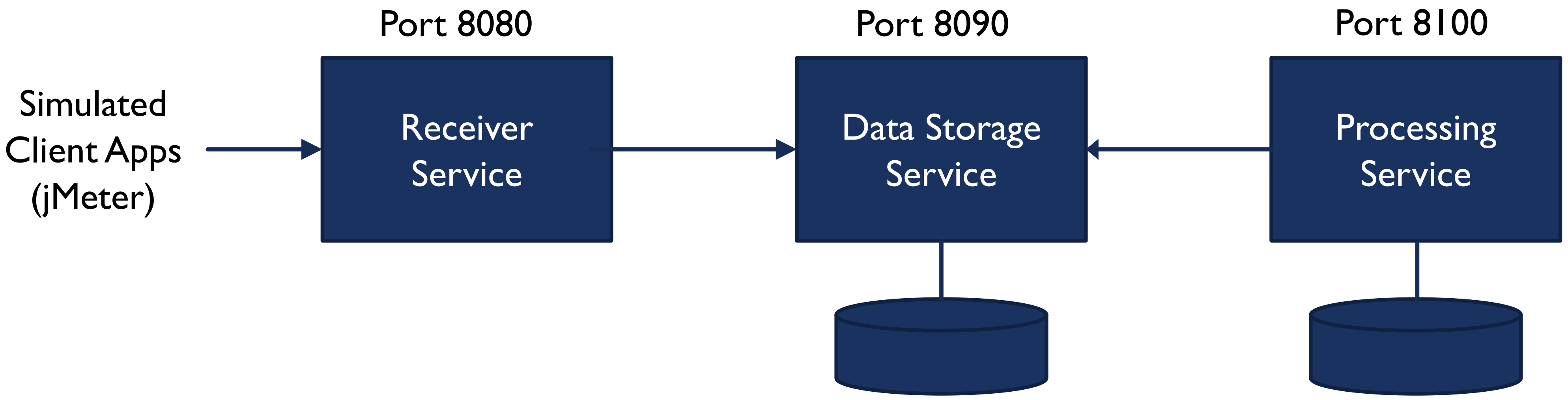
- The API allows other services to retrieve the latest statistics
- Here is an example response for our GET /stats endpoint

```
{
    "num_bp_readings": 203,
    "max_bp_dia_reading": 160,
    "max_bp_sys_reading": 100,
    "num_hr_readings": 200,
    "max_hr_reading": 197
}
```

Your JSON response should consist of a single object with the latest statistics

THREE SERVICES RUNNING TOGETHER

- When demoing this lab, you will be running all three services at the same time (on different ports).
- When you run your jMeter test, you should see the Processing Service periodically generate new stats over the duration of the test.
- You should be able to call the GET /stats endpoint on the Processing Service to see the latest statistics.



TODAY'S TOOLS

RESTful API Specification: SwaggerHub and OpenAPI

- Define a RESTful API in a yaml format

RESTful API Implementation: Python connexion

- Built on top of Flask but allows integration with an OpenAPI specification

Configuration and Logging:

- Yaml for configuration
- Python logging module for tracing

Periodic Processing: apscheduler

- Allows scheduled calls to functions to be defined

Database: SQLAlchemy and MySQL, SQLite

- SQLAlchemy and MySQL for the Storage Service
- SQLite file for the Processing Service

RESTful API Testing: PostMan and Apache jMeter

- Postman – same as ACIT 2515
- Apache jMeter – for load testing

You will be using these in your Lab today.

TODAY'S LAB

The lab is to be submitted individually. Today you will:

- Demo your Lab 4 results
- Create a new Processing Service that collects and stores some statistics on your Events
  - The metrics should be cumulative and stored in a SQLite database
  - You need to design the process_stats function which will calculate and update those statistics
- Test out all three of your services under load using your jMeter script