ACIT 3855 – Lab 4 – MySQL, Logging and Configuration

Instructor	Mike Mulder (mmulder10@bcit.ca) – Sets A and C Tim Guicherd (tguicherd@bcit.ca) – Set B
Total Marks	10
Due Dates	Demo and Submission by End of the Lesson 3 Class: • Monday, Feb. 5th for Set C • Thursday, Feb. 8th for Sets A and B

Purpose

- To add external configuration and logging to your Receiver Service (Lab 2) and Storage Service (Lab 3)
- To convert your Storage Service (Lab3) from a SQLite DB to a MySQL DB (with minimal code changes)

Part 1 – Configuration

Create a new YAML file in your **Receiver** (Lab 2) project called *app_conf.yml*. It should look something like this:

```
version: 1
eventstore1:
  url: http://localhost:8090/<Your Path Here>
eventstore2:
  url: http://localhost:8090/<Your Path Here>
```

where eventstore1 holds the URL of your endpoint for your first event type and eventstore2 holds the URL of your endpoint for your second event type. You may give them more meaningful names if you wish.

Load the configuration into your *app.py* file as follows:

```
with open('app_conf.yml', 'r') as f:
    app_config = yaml.safe_load(f.read())
```

Your configuration is now available in app_config which is a Python dictionary.

Make sure you have imported the yaml module into your app.py file.

Now use the eventstore1 and eventstore2 configuration values in your request.post calls rather than the hard-code URL strings.

Part 2 – Logging

Create a new YAML file in your **Receiver** (Lab 2) project called *log_conf.yml*. It should look something like this:

```
version: 1
formatters:
  simple:
```

```
    format: '%(asctime)s - %(name)s - %(levelname)s - %(message)s'
handlers:
  console:
    class: logging.StreamHandler
    level: DEBUG
    formatter: simple
    stream: ext://sys.stdout
  file:
    class: logging.FileHandler
    level: DEBUG
    formatter: simple
    filename: app.log
loggers:
  basicLogger:
    level: DEBUG
    handlers: [console, file]
    propagate: no
root:
  level: DEBUG
  handlers: [console]
```

Load the configuration into your *app.py* file as follows:

```
with open('log_conf.yml', 'r') as f:
    log_config = yaml.safe_load(f.read())
    logging.config.dictConfig(log_config)
```

Create a logger from the basicLogger defined in the configuration file. Make sure to import the logging and logging.config modules.

```
logger = logging.getLogger('basicLogger')
```

Add the following INFO level log messages to each of the two functions in your Receiver Service (Lab 2):

- Log the receipt of the event request in a form similar to this:

  Received event <event name> request with a trace id of <trace_id>

  Where the event name is the name of that event and the trace_id is some unique value. You should generate the unique value yourself using functions in the uuid, datetime or random modules in Python. If you use a random number make sure it is large enough to prevent duplicate ids.

- Log the return of the event response in a form similar to this:

  Returned event <event name> response (Id: <trace_id>) with status <status code>

All log messages on logger will be written to both the console and the file app.log.

Make sure you have imported yaml, logging and logging.config in your app.py file.

The trace_id generated above should be passed on to other services so we can trace data across multiple services. So update your **Storage Service** as follows:

- Add the trace_id to the request messages of both events in the OpenAPI specification.
- Store the trace_id with the event in your database, i.e., add it to each table in the create_tables.py script and add it to the SQLAlchemy declarative for each event.

Part 3 – Convert SQLite to MySQL with Configuration and Logging

If you do not already have MySQL installed on your laptop or a VM with MySQL (from a previous class), do so now: https://dev.mysql.com/doc/refman/8.0/en/installing.html. Use the Developer Default setup to get MySQL Workbench.

You will be updating your **Storage** Service (Lab 3) code for this part of the lab.

Add the following Python modules to your project:

- mysql-connector-python
- pymysql module

Create the following two Python files based on the existing files:

- create_tables_mysql.py (based on create_tables.py)
- drop_tables_mysql.py (based on drop_tables.py)

Update the above files to create and drop tables in a MySQL database rather than SQLite. See the examples at the end of this lab as guidance. Note that MySQL has a different structure for autoincrementing ID columns than SQLite. The ID columns also require a constraint to make them the primary key.

The tables should still match those defined in your SQLAlchemy declaratives.

Create the database in your MySQL installation using your mysql client. Run your create_tables_mysql.py to create your tables in the database (and drop_tables_mysql.py to remove them). Use your mysql client or MySQL Workbench to check that your tables are created.

Configuration

Update the database connection in your app.py script. The new connection should look something like this:

```
DB_ENGINE = create_engine
   ('mysql+pymysql://<user>:<password>@<hostname>:<port>/<db>')
```

The <user>, <password>, <hostname>, <port> and <db> values should come from an app_conf.yml file similar to what you did for your Receiver Service (Lab 2). The contents of file should look something like this:

```
version: 1
datastore:
  user: <user>
  password: <password>
```

```
hostname: <hostname>
port: <port>
db: <db>
```

Having the database settings in an external configuration file allows an adminstrator to change the settings based on the environment the app is deployed to without having to change the code itself.

Logging

Copy over the log_conf.yml file from your Receiver Service (Lab 2) into your Storage Service.

Load the log_conf.yml file into your app.py and create a logger object (same as for the Receiver Service).

Add the following DEBUG level log message to each of the <u>two</u> functions in your **Storage** Service (Lab 3):

- Log the successful storing of the received event (i.e., after the DB session is closed) but before the return statement. It should look something like:

  Stored event <event name> request with a trace id of <trace_id>

  Where the event name is the name of that event and the trace_id is the trace_id from the contents of the request message (same value as output in the log messages in the Receiver Service).

Testing

Once you've made the above updates (connection string, config file and logging), test your services with Postman and your jMeter script to make sure the MySQL database tables are being populated and the log messages are being written to the app.log file.

It should be "just that easy" to switch from SQLite to MySQL when using an Object Relational Mapping framework like SQLAlchemy.

Grading and Submission

Submit the following to the Lab 4 Dropbox on D2L:

- A zipfile containing your updated Receiver Service
- A zipfile containing your updated Storage Service

Demo the following to your instructor before the end of next class to receive your marks:

Your Receiver Service (Lab 2) Code Changes: • External Configuration loaded and used • Logging Configuration loaded and used Your Storage Service (Lab 3) Code Changes: • MySQL Database Connection	6 marks

• Trace ID included in the OpenAPI specification and stored in both database tables • External Configuration loaded and used • Logging Configuration loaded and used	
Run your Receiver and Storage Services against jMeter and show the following: • Log file from Receiver Service • Log file from Storage Service • Populated MySQL Database	4 marks
Total	**10 marks**

Example MySQL Table Creation/Drop Scripts

create_tables_mysql.py

```
import mysql.connector

db_conn = mysql.connector.connect(host="localhost", user="user",
password="password", database="events")

db_cursor = db_conn.cursor()

db_cursor.execute('''
          CREATE TABLE blood_pressure
          (id INT NOT NULL AUTO_INCREMENT,
           patient_id VARCHAR(250) NOT NULL,
           device_id VARCHAR(250) NOT NULL,
           systolic INTEGER NOT NULL,
           diastolic INTEGER NOT NULL,
           timestamp VARCHAR(100) NOT NULL,
           trace_id VARCHAR(100) NOT NULL,
           date_created VARCHAR(100) NOT NULL,
           CONSTRAINT blood_pressure_pk PRIMARY KEY (id))
          ''')

db_cursor.execute('''
          CREATE TABLE heart_rate
          (id INT NOT NULL AUTO_INCREMENT,
           patient_id VARCHAR(250) NOT NULL,
           device_id VARCHAR(250) NOT NULL,
           heart_rate INTEGER NOT NULL,
           timestamp VARCHAR(100) NOT NULL,
           trace_id VARCHAR(100) NOT NULL,
           date_created VARCHAR(100) NOT NULL,
           CONSTRAINT heart_rate_pk PRIMARY KEY (id))
```

```
          ''')

db_conn.commit()
db_conn.close()
```

drop_tables_mysql.py

```
import mysql.connector

db_conn = mysql.connector.connect(host="localhost", user="user",
password="password", database="events")

db_cursor = db_conn.cursor()

db_cursor.execute('''
                  DROP TABLE blood_pressure, heart_rate
                  ''')

db_conn.commit()
db_conn.close()
```