ACIT 3855 – Lab 4

MySQL, Logging and Configuration

Instructors

- Mike Mulder (mmulder10@bcit.ca)
- Tim Guicherd (tguicherd@bcit.ca)

Due date: demo and submission by end of next class.

Purpose

- To add external configuration and logging to your `receiver` and `storage` services
- To make your `storage` service use a MySQL database (with minimal code changes)

Part 1 – switch to MySQL

Install / set up a MySQL instance

You will need a MySQL instance up and running. Use whatever solution you are most comfortable with - but remember that we will start running our services on Docker in a few labs. It may be wise to run a MySQL docker container!

Option 1: run a Docker container (recommended)

Create a new Docker Compose file, and use the following example as inspiration (make sure you changes the user, password, and database name values!):

```
services:
  db:
    image: mysql
    environment:
      MYSQL_RANDOM_ROOT_PASSWORD: 1
      MYSQL_USER: skibidi
      MYSQL_PASSWORD: helpme
      MYSQL_DATABASE: did-you-copy-paste?
    ports:
      - 3306:3306
```

You can now run `docker-compose up -d`, and you will get access to a MySQL instance on port 3306.

Option 2: use the VM / setup you used in term 3

You may have an existing MySQL instance that you can reuse. Refer to your term 3 courses. You may also set up MySQL directly on your laptop using MySQL Workbench.

Make your `storage` use the MySQL instance

Install a MySQL connector

You will need a valid MySQL connector library in order to use SQLAlchemy with MySQL. There are several options available (`mysqlclient` - recommended, `mysql-connector-python`, and `pymysql`). Install it with `pip install mysqlclient`.

You may require development libraries in order to set up the MySQL connector. In Linux, `sudo apt-get install python3-dev default-libmysqlclient-dev build-essential` should be enough to make it work. Refer to your operating system guides to make it work.

Note: setting up `mysqlclient` on Windows may be complicated, if there is no wheel package available. You may have more luck with `pymysql`, or by using a Linux environment for your development tasks.

Change your `storage` service to use MySQL

In your database setup, change the URL used to create the engine to a MySQL resource string. For example: `mysql://skibidi:helpme@localhost/you-copy-pasted-again`.

 No other changes should be required!

Test your `storage` service and make sure the data is stored in the MySQL database.

Part 2 – Tracing

Generate a `trace_id` and add it to your event data

To make it easier to "track" events and communication between your microservices, we are going to add a unique identifier to all events received by the `receiver`. This identifier (`trace_id`) will be passed on to all other services.

In the `receiver` service, make changes so that:

- when an event is received, a `trace_id` is added to the JSON payload and sent to the `storage` service
- the `trace_id` is a unique identifier

Option 1: Use `time`

`time_ns()` returns the time since the epoch, with nanoseconds precision. That should be enough to make it unique for your events. The value returned is a **number**.

```
import time

trace_id = time.time_ns()
```

Option 2: Use `uuid`

```
import uuid

trace_id = str(uuid.uuid4())
```

`uuid4` generates a unique identifier - the value provided is a **string**.

Update the `storage` service to store the `trace_id`

- Make changes to the OpenAPI YAML for `storage`, and add the `trace_id` in the specifications.
- Update your SQLAlchemy models to add a column of the appropriate type.

> Do not forget to drop and recreate your tables with the updated schema!

Part 3 – Configuration

Create a new YAML file in your `receiver` folder called `app_conf.yml`. It should look something like this:

```
version: 1
eventstore1:
  url: http://localhost:8090/my-first-event
eventstore2:
  url: http://localhost:8090/my-second-event
```

- `eventstore1` holds the URL of your endpoint for your first event
- `eventstore2` holds the URL of your endpoint for your second event


You may give them more meaningful names if you wish, or use a different structure:

```
version: 1
events:
  snow:
    url: http://localhost:8090/my-first-event
  lift:
    url: http://localhost:8090/my-second-event
```

Then, in your `app.py`, load the configuration file:

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

Your configuration is now available in the dictionary `app_config`.

> Make sure you import `yaml` in your `app.py`.

You can now replace the harcoded URLs in your `POST` calls with the values from the configuration file.

Part 4 – Logging

Logging configuration file

Create a new YAML file in your `receiver` folder 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]
disable_existing_loggers: false
```

Load the configuration file in your `app.py` like you did for `app_conf.yml` - but this time, use it to configure the `logging` module:

```
with open("conf_log.yml", "r") as f:
  LOG_CONFIG = yaml.safe_load(f.read())
  logging.config.dictConfig(LOG_CONFIG)
```

Log messages

Create a logger from the `basicLogger` defined in the configuration file.

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

In your `receiver` project, make changes to log the following messages with `INFO` level :

- Log a message when an event is received, for example: `Received event snow_report with a trace id of 123456789`
- Log the response of the `storage` service, for example: `Response for event snow_report (id: 123456789) has status 201`

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

Part 4 – Configuration and logging for `storage`

Configuration

In your `storage` folder, create a YAML file `app_conf.yaml`. Use it to store the database credentials. For instance:

```
version: 1
datastore:
  user: skibidi
  password: helpme
  hostname: localhost
  port: 3306
  db: i-dont-know
```

Similar to what you have done with the `receiver`, load the configuration file. Replace the hardcoded values in your Python code with the values read from the configuration file.

Having the database settings in an external configuration file allows an administrator 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` to your `storage`.

- Similar to the `receiver`, load the `log_conf.yml` file and create a `logger` object.
- Log a message when an event is successfully stored (after the DB session is closed) using the `DEBUG` level. For example: `Stored event snow_report with a trace id of 123456789`

Testing

Once you've made all changes, test your services with Bruno or Postman, as well as your jMeter test plan. Make sure the MySQL database tables are being populated and the log messages are being written to the `app.log` file.

Grading and submission

Demo the following to your instructor, and answer any questions before the end of next class to receive your marks:

Code changes (6 marks)

- Changes in `receiver`:
  - External configuration loaded and used
  - Logging configuration loaded and used
- Changes in `storage`:
  - uses a MySQL database
  - Trace ID included in the OpenAPI specification and stored in the database
  - External configuration loaded and used
  - Logging configuration loaded and used

Load testing

Run your jMeter test plan and show the following:

- Log file from `receiver`
- Log file from `storage`
- Populated MySQL database

Submit the following to the Lab 4 Dropbox on D2L:

- A zipfile containing your updated `receiver` service
- A zipfile containing your updated `storage` service

Total: 10 marks