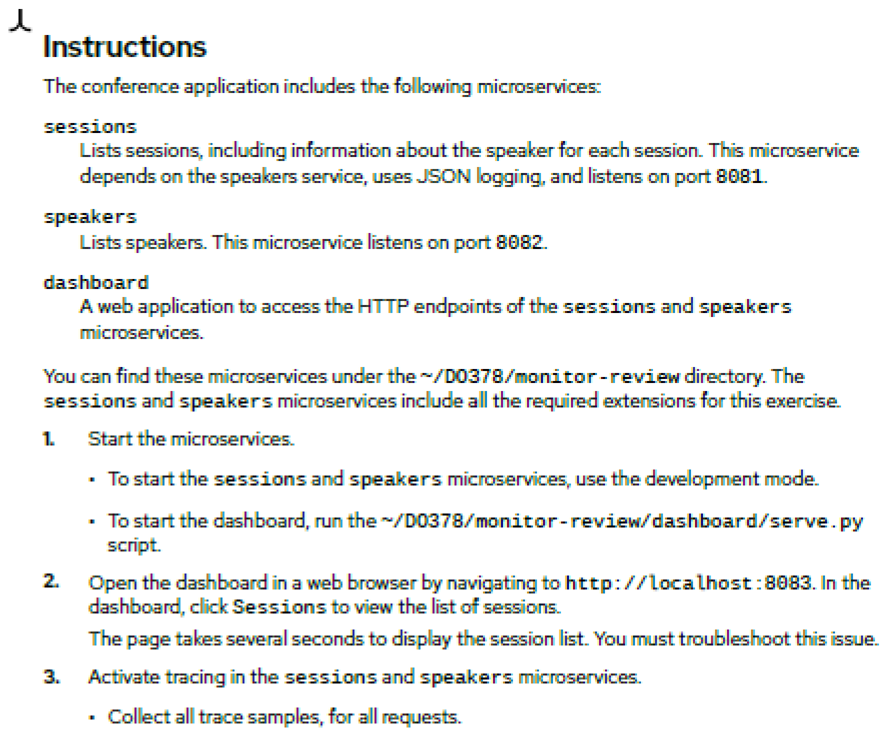
LAB 24: QUARKUS MONITOR REVIEW

Autor: José Díaz

Github Repo: https://github.com/joedayz/quarkus-bcp-2025.git

Abre el proyecto **monitor-review**



Instructions

The conference application includes the following microservices:

`sessions`
: Lists sessions, including information about the speaker for each session. This microservice depends on the speakers service, uses JSON logging, and listens on port `8081`.

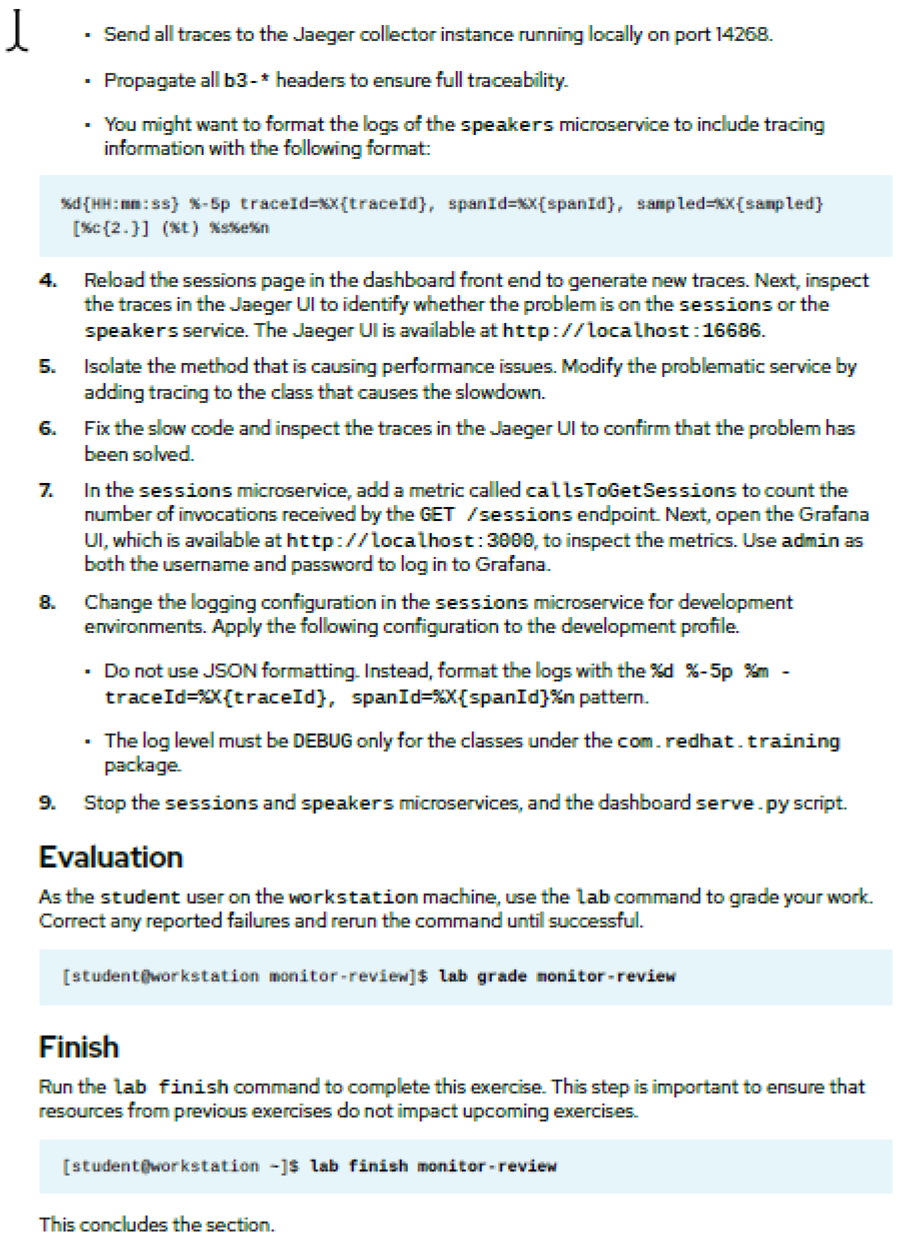
`speakers`
: Lists speakers. This microservice listens on port `8082`.

`dashboard`
: A web application to access the HTTP endpoints of the `sessions` and `speakers` microservices.

You can find these microservices under the `~/DO378/monitor-review` directory. The `sessions` and `speakers` microservices include all the required extensions for this exercise.

1. Start the microservices.
   - To start the `sessions` and `speakers` microservices, use the development mode.
   - To start the dashboard, run the `~/DO378/monitor-review/dashboard/serve.py` script.
2. Open the dashboard in a web browser by navigating to `http://localhost:8083`. In the dashboard, click `Sessions` to view the list of sessions.

   The page takes several seconds to display the session list. You must troubleshoot this issue.
3. Activate tracing in the `sessions` and `speakers` microservices.
   - Collect all trace samples, for all requests.



- Send all traces to the Jaeger collector instance running locally on port 14268.
- Propagate all `b3-*` headers to ensure full traceability.
- You might want to format the logs of the `speakers` microservice to include tracing information with the following format:

```
%d{HH:mm:ss} %-5p traceId=%X{traceId}, spanId=%X{spanId}, sampled=%X{sampled}
 [%c{2.}] (%t) %s%e%n
```

4. Reload the sessions page in the dashboard front end to generate new traces. Next, inspect the traces in the Jaeger UI to identify whether the problem is on the `sessions` or the `speakers` service. The Jaeger UI is available at `http://localhost:16686`.
5. Isolate the method that is causing performance issues. Modify the problematic service by adding tracing to the class that causes the slowdown.
6. Fix the slow code and inspect the traces in the Jaeger UI to confirm that the problem has been solved.
7. In the `sessions` microservice, add a metric called `callsToGetSessions` to count the number of invocations received by the `GET /sessions` endpoint. Next, open the Grafana UI, which is available at `http://localhost:3000`, to inspect the metrics. Use `admin` as both the username and password to log in to Grafana.
8. Change the logging configuration in the `sessions` microservice for development environments. Apply the following configuration to the development profile.
   - Do not use JSON formatting. Instead, format the logs with the `%d %-5p %m - traceId=%X{traceId}, spanId=%X{spanId}%n` pattern.
   - The log level must be `DEBUG` only for the classes under the `com.redhat.training` package.
9. Stop the `sessions` and `speakers` microservices, and the dashboard `serve.py` script.

Evaluation

As the `student` user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation monitor-review]$ lab grade monitor-review
```

Finish

Run the `lab finish` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish monitor-review
```

This concludes the section.

SOLUCIÒN:

Instructions

The conference application includes the following microservices:

`sessions`
: Lists sessions, including information about the speaker for each session. This microservice depends on the speakers service, uses JSON logging, and listens on port `8081`.

`speakers`
: Lists speakers. This microservice listens on port `8082`.

`dashboard`
: A web application to access the HTTP endpoints of the `sessions` and `speakers` microservices.

You can find these microservices under the `~/DO378/monitor-review` directory. The `sessions` and `speakers` microservices include all the required extensions for this exercise.

1. Start the microservices.

   - To start the `sessions` and `speakers` microservices, use the development mode.
   - To start the dashboard, run the `~/DO378/monitor-review/dashboard/serve.py` script.

   1.1. Navigate to the `~/DO378/monitor-review/sessions` directory.

```
[student@workstation DO378]$ cd ~/DO378/monitor-review/sessions
```

   1.2. Start the `sessions` microservice in development mode.

```
[student@workstation sessions]$ mvn quarkus:dev
...output omitted...
{ ... "message":"... Listening on: http://localhost:8081" ... }
...output omitted...
```

1.3. In a new terminal, navigate to the `~/DO378/monitor-review/speakers` directory.

```
[student@workstation ~]$ cd ~/DO378/monitor-review/speakers
```

1.4. Start the `speakers` microservice in development mode.

```
[student@workstation speakers]$ mvn quarkus:dev
...output omitted...
INFO  [io.quarkus] ... Listening on: http://localhost:8082
...output omitted...
```

1.5. In a new terminal, start the dashboard by running the `~/DO378/monitor-review/dashboard/serve.py` script.

```
[student@workstation ~]$ ~/DO378/monitor-review/dashboard/serve.py
Serving '/home/student/DO378/monitor-review/dashboard/build' at http://
localhost:8083/
```

2. Open the dashboard in a web browser by navigating to `http://localhost:8083`. In the dashboard, click **Sessions** to view the list of sessions.

The page takes several seconds to display the session list. You must troubleshoot this issue.

3. Activate tracing in the `sessions` and `speakers` microservices.

- Collect all trace samples, for all requests.
- Send all traces to the Jaeger collector instance running locally on port 14268.
- Propagate all `b3-*` headers to ensure full traceability.
- You might want to format the logs of the `speakers` microservice to include tracing information with the following format:

```
%d{HH:mm:ss} %-5p traceId=%X{traceId}, spanId=%X{spanId}, sampled=%X{sampled}
 [%c{2.}] (%t) %s%e%n
```

3.1. Add the following properties to the `speakers/src/main/resources/application.properties` file.

```
quarkus.jaeger.service-name = speakers
quarkus.jaeger.sampler-type = const
quarkus.jaeger.sampler-param = 1
quarkus.log.console.format = %d{HH:mm:ss} %-5p traceId=%X{traceId}, spanId=
%X{spanId}, sampled=%X{sampled} [%c{2.}] (%t) %s%e%n
quarkus.jaeger.endpoint = http://localhost:14268/api/traces
quarkus.jaeger.propagation= b3
quarkus.jaeger.reporter-log-spans = true
```

3.2. Add the following properties to the `sessions/src/main/resources/application.properties` file.

```
quarkus.jaeger.service-name = sessions
quarkus.jaeger.sampler-type = const
quarkus.jaeger.sampler-param = 1
quarkus.jaeger.endpoint = http://localhost:14268/api/traces
quarkus.jaeger.propagation= b3
```

4. Reload the sessions page in the dashboard front end to generate new traces. Next, inspect the traces in the Jaeger UI to identify whether the problem is on the `sessions` or the `speakers` service. The Jaeger UI is available at `http://localhost:16686`.

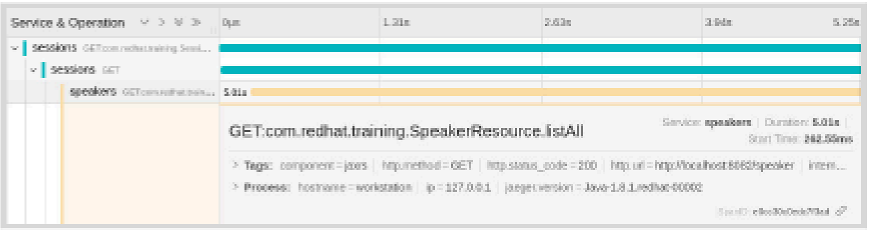
4.1. Return to the dashboard front end in the browser and reload the `Sessions` page to generate traces.

4.2. In a new browser tab, navigate to `http://localhost:16686` to open the Jaeger UI.

4.3. In the Jaeger web console, select the `sessions` service from the `Service` field in the `Search` panel. Click Find Traces.

4.4. Click the first `sessions: GET:com.redhat.training.SessionResource.getAllSessions` trace to view the trace details.

4.5. In the trace, verify that the `GET:com.redhat.training.SpeakerResource.listAll` span is taking several seconds to serve the request.



GET:com.redhat.training.SpeakerResource.listAll

Service: speakers | Duration: 5.01s | Start Time: 262.55ms

Tags: component = jaxrs | http.method = GET | http.status_code = 200 | http.url = http://localhost:8082/speaker | intern...

Process: hostname = workstation | ip = 127.0.0.1 | jaeger.version = Java-1.8.1.redhat-00002

5. Isolate the method that is causing performance issues. Modify the problematic service by adding tracing to the class that causes the slowdown.

5.1. The `SpeakerResource#listAll` method handles requests to the `GET /speaker` endpoint of the `speaker` service. This method uses the `com.redhat.training.SpeakerFinder` bean, which gathers speakers from a database. Add the `@Traced` annotation to this class.

```
@Traced
@ApplicationScoped
public class SpeakerFinder {
    ...code omitted...
}
```

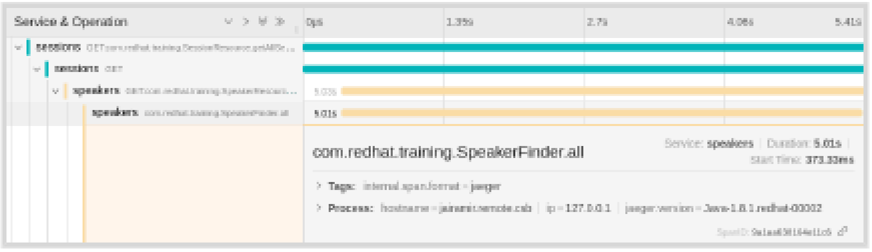
5.2. Return to the dashboard front end in the browser and reload the `Sessions` page to generate new traces.

5.3. Return to the Jaeger console and click Search at the top navigation bar.

5.4. In the `Search` pane, click Find Traces to refresh the traces list.

5.5. Click the most recent trace of the list.

5.6. Verify that the `SpeakerFinder#all` method is causing the bottleneck. This method is taking several seconds to load the list of speakers.



com.redhat.training.SpeakerFinder.all

Service: speakers | Duration: 5.01s | Start Time: 373.33ms

Tags: internal.span.format = jaeger

Process: hostname = [illegible] | ip = 127.0.0.1 | jaeger.version = Java-1.8.1.redhat-00002

6. Fix the slow code and inspect the traces in the Jaeger UI to confirm that the problem has been solved.

6.1. In the `SpeakerFinder#all` method, remove or comment the code that causes the bottleneck.

```
public List<Speaker> all() {
    Log.info("Retrieving all speakers from database");

    // runSlowAndRedundantOperation();

    return Speaker.listAll();
}
```

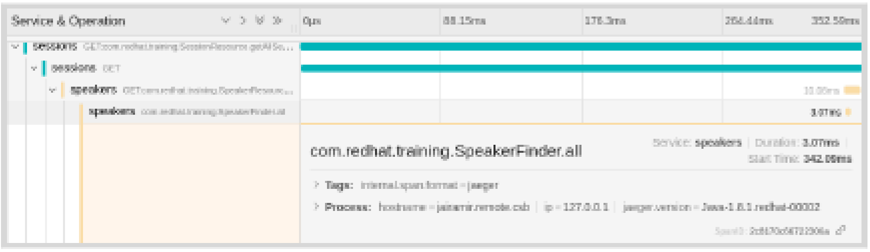
6.2. Return to the dashboard front end in the browser and reload the `Sessions` page to generate new traces.

6.3. Return to the Jaeger console and click Search at the top navigation bar.

6.4. In the `Search` pane, click Find Traces to refresh the traces list.

6.5. Click the most recent trace of the list.

6.6. Verify that the `SpeakerFinder#all` method takes a few milliseconds to complete the request.


Service & Operation
0µs
88.15ms
176.3ms
264.44ms
352.59ms
sessions
sessions GET
speakers
speakers com.redhat.training.SpeakerFinder.all
com.redhat.training.SpeakerFinder.all
Service: speakers
Duration: 3.07ms
Start Time: 342.05ms
Tags: internal.span.format = jaeger
Process: ip = 127.0.0.1 jaeger.version = Java-1.8.1.redhat-00002

7. In the `sessions` microservice, add a metric called `callsToGetSessions` to count the number of invocations received by the `GET /sessions` endpoint. Next, open the Grafana UI, which is available at **`http://localhost:3000`**, to inspect the metrics. Use **`admin`** as both the username and password to log in to Grafana.

   7.1. Open the `com.redhat.training.SessionResource` class and add the **`@Counted`** annotation to the `getAllSessions` method.

```
@GET
@Counted( value = "callsToGetSessions" )
public Collection<Session> getAllSessions() throws Exception {
    ...code omitted...
}
```

   7.2. Return to the dashboard front end in the browser and reload the `Sessions` page to generate new metric values.

   7.3. Open the web browser and navigate to **`http://localhost:3000/dashboards`**.

   7.4. Enter **`admin`** as the username and **`admin`** as the password, and then click Log in.

   7.5. Click Skip to omit changing the account password.

   7.6. Click the conference directory, and then click DO378 Conference Metrics Dashboard.

   7.7. Observe the dashboard that collects the metric added to the application.

8. Change the logging configuration in the `sessions` microservice for development environments. Apply the following configuration to the development profile.

   - Do not use JSON formatting. Instead, format the logs with the `%d %-5p %m - traceId=%X{traceId}, spanId=%X{spanId}%n` pattern.
   - The log level must be `DEBUG` only for the classes under the `com.redhat.training` package.

   8.1. Inspect the logs of the `session` microservice and verify that the logs are in JSON format.

   8.2. Deactivate JSON formatting for the `dev` profile. Add the following line to the `application.properties` file:

```
%dev.quarkus.log.console.json = false
```

8.3. Set the format for the development profile. Add the following line to the `application.properties` file:

```
%dev.quarkus.log.console.format = %d %-5p %m - traceId=%X{traceId}, spanId=
%X{spanId}%n
```

8.4. Set the log level for the development profile. Add the following line to the `application.properties` file:

```
%dev.quarkus.log.category."com.redhat.training".level = DEBUG
```

8.5. Return to the dashboard front end in the browser and reload the `Sessions` page.

8.6. Inspect the logs of the `sessions` microservice. The console displays debug messages and the logs use the custom format.

```
...output omitted...
2023-02-01 14:27:48,405 INFO  Finding all sessions - traceId=..., spanId=...
2023-02-01 14:27:48,406 DEBUG Gathering extended speaker information ... -
 traceId=..., spanId=...
2023-02-01 14:27:48,419 DEBUG Added speakers information to session list -
 traceId=..., spanId=...
```

9. Stop the `sessions` and `speakers` microservices, and the dashboard `serve.py` script.

Evaluation

As the `student` user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation monitor-review]$ lab grade monitor-review
```

Finish

Run the `lab finish` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish monitor-review
```

This concludes the section.

enjoy!

Jose