Developing Microservices with MicroProfile using Quarkus

Introduction

This tutorial builds two Java services from scratch that incrementally add Eclipse MicroProfile APIs. The first is a `student` service that returns a list of students, which is invoke by a `frontend` service. This domain model is simple so the focus is spent on the MicroProfile APIs themselves.

The documented steps in this tutorial follow a pattern:

1. Write code
2. Run a command to test code (curl)
3. Check output

This pattern flows quickly due to Quarkus' Live Coding feature, and makes checking code changes near-instantaneous so each change can be immediately evaluated.

NOTE This tutorial will require three terminal windows. Each command block will display the terminal where the command is to be run - *Terminal 1*, *Terminal 2*, or *Terminal 3*. As the tutorial progresses, it helps to organize the terminals on your screen so they can all be viewed at the same time. In addition, each code block will show the name of the file to be edited (ex: "StudentResource.java")

1. Clone the project to your local system

   Terminal 1

   ```
   $ git clone \
     https://github.com/jclingan/oreilly-microprofile-tutorial.git \
     tutorial
   ```

   Terminal 1 Output

   ```
   Cloning into 'tutorial'...
   remote: Enumerating objects: 130, done.
   remote: Counting objects: 100% (130/130), done.
   remote: Compressing objects: 100% (92/92), done.
   remote: Total 130 (delta 29), reused 120 (delta 22), pack-reused 0
   Receiving objects: 100% (130/130), 2.50 MiB | 644.00 KiB/s, done.
   Resolving deltas: 100% (29/29), done.
   ```

Create Student Service

Create a simple Quarkus project as a starting point - the "student" service. A default RESTful endpoint is generated. While this tutorial utilizes the mvn command line tooling, projects can also be generated at code.quarkus.io using a Web UI.

1. Go to the project’s base directory and create project using maven

   Terminal 1

   ```
   $ cd tutorial/working
   $ mvn io.quarkus:quarkus-maven-plugin:1.3.0.Final:create \
       -DprojectGroupId=org.acme \
       -DprojectArtifactId=student \
       -DclassName="org.acme.StudentResource" \
       -Dpath="/student" \
       -Dextensions="resteasy-jsonb,smallrye-health"
   ```

2. Optional: To see a list of available extensions, type:

   Terminal 1

   ```
   $ mvn quarkus:list-extensions
   ```

3. Open project in your favorite IDE

   a. Open StudentResource.java and peruse the JAX-RS source code

StudentResource.java

```
package org.acme;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/student")
public class StudentResource {

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String hello() {
        return "hello";
    }
}
```

NOTE Quarkus does not require a JAX-RS Application class, but will use it if provided. In addition, Quarkus RESTful resources are singletons (`@javax.inject.Singleton`). These are developer convenience features. These points are covered in the Quarkus Getting Started Guide.

4. Start Quarkus in developer mode

   Terminal 1

   ```
   $ mvn compile quarkus:dev
   ```

5. Check that the endpoint returns "hello"

   Terminal 2

   ```
   $ curl -i localhost:8080/student
   ```

   Output

   ```
   HTTP/1.1 200 OK
   Content-Length: 5
   Content-Type: text/plain;charset=UTF-8

   hello
   ```

6. Try out live reload. In the StudentResource.java hello() method, replace "Hello" with "Howdy" and save the file

   StudentResource.java

   ```
   public class StudentResource {

       @GET
       @Produces(MediaType.TEXT_PLAIN)
       public String hello() {
           return "Howdy"; ①
       }
   }
   ```

   ① Replace "Hello" with "Howdy"

   Terminal 2

   ```
   $ curl -i localhost:8080/student
   ```

   Terminal 2 Output

   ```
   HTTP/1.1 200 OK
   Content-Length: 5
   Content-Type: text/plain;charset=UTF-8

   Howdy
   ```

7. In StudentResource.java, create a list of Strings:

   StudentResource.java

   ```
   @Path("/student")
   public class StudentResource {
       List<String> students = new ArrayList<>(); ①
   ```

   ① Add this line

8. Add a method called `listStudents()` at the "/list" path that returns the students as a JSON array

StudentResource.java

```
@GET
@Path("/list")
@Produces(MediaType.APPLICATION_JSON)
public List<String> listStudents() {
    return students;
}
```

---

9. Check the output

   Terminal 2

   ```
   $ curl -i localhost:8080/student/list
   ```

   Terminal 2 Output

   ```
   HTTP/1.1 200 OK
   Content-Length: 2
   Content-Type: application/json

   []
   ```

MicroProfile Config

This section covers the MicroProfile CDI injection-based API for externalizing configuration. In these instructions, configuration parameters are stored in `src/main/resources/META-INF/microprofile-config.properties`.

NOTE While Quarkus supports MicroProfile APIs, it also supports much more than MicroProfile like Spring and Vert.x APIs. For that reason, the Quarkus guides refer to the more framework-agnostic `src/main/resources/application.properties`. Because this tutorial focuses on MicroProfile, it will use `src/main/resources/META-INF/microprofile-config.properties`. MicroProfile supports both, and property values defined in `application.properties` take precedence over values defined in `microprofile-config.properties`.

1. Create a `doDelay()` method to delay 3000 milliseconds and print "Waiting 3000 milliseconds" to stdout.

   StudentResource.java

```
void doDelay() {
    int delayTime;
    try {
        delayTime=3000;
        System.out.println("** Waiting " + delayTime + "ms **");
        TimeUnit.MILLISECONDS.sleep(delayTime);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

2. In `listStudents()`, call `doDelay()`

   StudentResource.java

```
@GET
@Path("/list")
@Produces(MediaType.APPLICATION_JSON)
public List<String> listStudents() {
    doDelay(); ①
    return students;
}
```

   ① Insert `doDelay()` call

3. Check the endpoint, which should take longer to complete

Terminal 2

```
$ curl -i http://localhost:8080/student/list
```

Terminal 2 Output (after 3 seconds)

```
HTTP/1.1 200 OK
Content-Length: 2
Content-Type: application/json

[]
```

Terminal 1 Output (after 3 seconds)

```
** Waiting 3000ms ** ①
```

① Output from `doDelay()`

---

4. Inject `delay` property value into variable `delay`

   StudentResource.java

   ```
   @Inject
   @ConfigProperty(name="delay")
   int delay;
   ```

---

5. In `doDelay()`, replace hard-coded "3000" with the `delay` variable

   StudentResource.java

   ```
   void doDelay() {
       int delayTime;
       try {
           delayTime=delay;  ①
           System.out.println("** Waiting " + delayTime + "ms **");
           TimeUnit.MILLISECONDS.sleep(delayTime);
       } catch (InterruptedException e) {
           e.printStackTrace();
       }
   }
   ```

   ① Replace `3000` with `delay`, as shown

---

6. Verify an error is generated because the `delay` property has not been defined.

Terminal 2

```
$ curl -i http://localhost:8080/student/list
```

Terminal 2 Output (Stack Trace)

```
Caused by: javax.enterprise.inject.spi.DeploymentException: No config value of type
[int] exists for: delay
```

7. Define `delay` property in `src/main/resources/META-INF/microprofile-config.properties`:

   microprofile-config.properties

   ```
   # Configuration file
   # key = value

   delay=2500 ①
   ```

   ① Add this line

8. Verify `delay` property is read.

   Terminal 2

   ```
   $ curl -i http://localhost:8080/student/list
   ```

   Terminal 2 Output (after 2.5 seconds)

   ```
   HTTP/1.1 200 OK
   Content-Length: 2
   Content-Type: application/json

   []
   ```

   Terminal 1 Output

   ```
   ** Waiting 2500ms **
   ```

9. Comment out `delay` in `microprofile-config.properties`

microprofile-config.properties

```
# Configuration file
# key = value

#delay=2500 ①
```

① Comment out `delay`

---

10. Update the `@ConfigProperty` annotation with a default value of 2000.

    StudentResource.java

    ```
    @Inject
    @ConfigProperty(name="delay", defaultValue="2000") ①
    int delay;
    ```

    ① Insert `defaultValue=2000`

---

11. Verify `defaultValue` is read.

    Terminal 2

    ```
    $ curl -i http://localhost:8080/student/list
    ```

    Terminal 2 Output (after 2 seconds)

    ```
    HTTP/1.1 200 OK
    Content-Length: 2
    Content-Type: application/json

    []
    ```

    Terminal 1 Output

    ```
    ** Waiting 2000ms **
    ```

---

12. Stop running Quarkus process.

    Terminal 1

    ```
    # Press CTRL-C to stop Quarkus
    ```

---

13. Define DELAY environmental variable

    Terminal 1

    ```
    export DELAY=4000
    ```

14. Restart Quarkus.

    Terminal 1

    ```
    $ mvn compile quarkus:dev
    ```

15. Verify the DELAY environment variable overrides the value in the property file.

    Terminal 2

    ```
    $ curl -i http://localhost:8080/student/list
    ```

    Terminal 2 Output (after 4 seconds)

    ```
    HTTP/1.1 200 OK
    Content-Length: 2
    Content-Type: application/json

    []
    ```

    Terminal 1 Output

    ```
    ** Waiting 4000ms **
    ```

16. Stop Quarkus

    Terminal 1

    ```
    # Press CTRL-C to stop Quarkus
    ```

17. Re-start Quarkus and define system property via CLI.

    Terminal 1

    ```
    $ mvn compile quarkus:dev -Ddelay=5000
    ```

18. Verify the `DELAY` system property overrides the value in the property file. In *Terminal 2*, type

    Terminal 2

    ```
    $ curl -i http://localhost:8080/student/list
    ```

    Terminal 2 Output (after 5 seconds)

    ```
    HTTP/1.1 200 OK
    Content-Length: 2
    Content-Type: application/json

    []
    ```

    Terminal 1 Output

    ```
    ** Waiting 5000ms **
    ```

19. Clean up by stopping Quarkus and unsetting DELAY environment variable

    Terminal 1

    ```
    # *** First, press CTRL-C to stop Quarkus ***
    # Next, remove DELAY environment variable
    unset DELAY
    ```

20. Change Quarkus HTTP port to 8082. Update microprofile-config.properties to look as follows:

    microprofile-config.properties

    ```
    #delay=2500
    quarkus.http.port=8082 ①
    ```

    ① Insert `quarkus.http.port` property

21. Restart Quarkus without defining `delay` system property and change debug port.

    Terminal 1

    ```
    $ mvn compile quarkus:dev -Ddebug=5006
    ```

22. Verify updated property

Terminal 2

```
# Note the port change to 8082!
$ curl -i http://localhost:8082/student/list
```

Terminal 2 Output (after 2 seconds)

```
HTTP/1.1 200 OK
Content-Length: 2
Content-Type: application/json

[]
```

Terminal 1 Output

```
** Waiting 2000ms **
```

23. In MicroProfile Config, comma-separated properties can be read as a `List`. Add the following to `microprofile-config.properties` to initialize the student list:

    microprofile-config.properties

    ```
    students=Duke,John,Jane,Arun,Christina
    ```

24. Inject students into student list. Change `List<String>` students to:

    StudentResource.java

    ```
    @Inject                                 ①
    @ConfigProperty(name = "students")
    List<String> students = new ArrayList<>();
    ```

    ① Add @Inject and @ConfigProperty annotations

25. Verify that the students have been injected.

    Terminal 1

    ```
    $ curl -i http://localhost:8082/student/list
    ```

Terminal 2 Output (after 2 seconds)

```
HTTP/1.1 200 OK
Content-Length: 41
Content-Type: application/json

["Duke","John","Jane","Arun","Christina"]
```

Terminal 1 Output

```
** Waiting 2000ms **
```

MicroProfile Rest Client

This section creates a "frontend" service that utilizes the type-safe MicroProfile Rest Client API to invoke the student service. Additional Quarkus extensions (aka maven dependencies) are also added to support upcoming sections well.

1. Create frontend project using mvn command line

   Terminal 2

   ```
   $ cd tutorial/working
   $ mvn io.quarkus:quarkus-maven-plugin:1.3.0.Final:create \
       -DprojectGroupId=org.acme \
       -DprojectArtifactId=frontend \
       -DclassName="org.acme.FrontendResource" \
       -Dpath="/frontend" \
       -Dextensions="resteasy-jsonb,metrics,rest-client,fault-tolerance"
   ```

2. Open frontend project in your IDE

3. Start frontend in Quarkus dev mode

   Terminal 2

   ```
   $ mvn compile quarkus:dev
   ```

4. Create `src/main/java/org/acme/StudentRestClient.java` and paste in the following content

frontend/src/main/java/org/acme/StudentRestClient.java

```
package org.acme;

import java.util.List;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

import org.eclipse.microprofile.rest.client.inject.RegisterRestClient;

@RegisterRestClient(baseUri = "http://localhost:8082")
@Path("/student")
public interface StudentRestClient {
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String hello();

    @GET
    @Path("/list")
    @Produces(MediaType.APPLICATION_JSON)
    public List<String> listStudents();
}
```

5. Inject `StudentRestClient` into `FrontendResource.java`

   FrontendResource.java

   ```
   @Inject
   @RestClient
   StudentRestClient student;
   ```

6. Change `hello()` method to invoke student service student service `hello` endpoint

   FrontendResource.java

   ```
   @GET
   @Produces(MediaType.TEXT_PLAIN)
   public String hello() {
       return student.hello(); ①
   }
   ```

   ① Replace `"hello"` with `student.hello()`, as shown

7. Check endpoint works properly

Terminal 3

```
$ curl -i localhost:8080/frontend
```

Terminal 3 Output

```
HTTP/1.1 200 OK
Content-Length: 5
Content-Type: text/plain;charset=UTF-8

Howdy
```

8. Remove `baseURI` parameter from `@RegisterRestClient` so it can be configured using a property

   StudentRestClient.java

   ```
   @RegisterRestClient ①
   ```

   ① Removed `(baseUri = "http://localhost:8082")`

9. Configure rest client `baseUri` in `microprofile-config.properties`

   frontend microprofile-config.properties

   ```
   org.acme.StudentRestClient/mp-rest/uri=http://localhost:8082
   ```

10. Check endpoint

    Terminal 3

    ```
    $ curl -i localhost:8080/frontend
    ```

    Terminal 3 Output

    ```
    HTTP/1.1 200 OK
    Content-Length: 5
    Content-Type: text/plain;charset=UTF-8

    Howdy
    ```

11. Update `@RegisterRestClient` annotation to specify `configKey` in `StudentRestClient.java`

StudentRestClient.java

```
@RegisterRestClient(configKey = "StudentService")
```

12. Update the frontend `src/main/resources/META-INF/microprofile-config.properties` to utilize the `configKey`

    frontend microprofile-config.properties

    ```
    StudentService/mp-rest/uri=http://localhost:8082
    ```

13. Check endpoint

    Terminal 3

    ```
    $ curl -i localhost:8080/frontend
    ```

    Terminal 3 Output

    ```
    HTTP/1.1 200 OK
    Content-Length: 5
    Content-Type: text/plain;charset=UTF-8

    Howdy
    ```

14. Add `listStudents()` method to `FrontendResource.java`.

    FrontendResource.java

    ```
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    @Path("/list")
    public List<String> listStudents() {
        List<String> students = student.listStudents();

        return students;
    }
    ```

15. Specify a `StudentRestClient` `readTimeout` in frontend `microprofile-config.properties` that will throw an exception if read time threshold is exceeded

frontend microprofile-config.properties

```
StudentService/mp-rest/readTimeout = 1000 ①
```

① Add this

---

16. Check endpoint, which should result in a "java.net.SocketTimeoutException: Read timed out" because Student doDelay() method is set at a 2000ms delay.

    Terminal 3

    ```
    $ curl -i localhost:8080/frontend/list
    ```

    Terminal 3 Output

    ```
    # Stack trace ...
    Unable to invoke request: java.net.SocketTimeoutException: Read timed out
    # Stack trace ...
    ```

    Terminal 2 Output

    ```
    # Stack trace ...
    Unable to invoke request: java.net.SocketTimeoutException: Read timed out
    # Stack trace ...
    ```

    Terminal 1 Output

    ```
    ** Waiting 2000ms **
    ```

---

17. Comment out the `readTimeout` property in `microprofile-config.properties` to avoid exception

    frontend microprofile-config.properties

    ```
    #StudentService/mp-rest/readTimeout = 1000 ①
    ```

    ① Comment this out

---

18. Check endpoint

    Terminal 3

    ```
    $ curl -i localhost:8080/frontend/list
    ```

Terminal 3 Output

```
HTTP/1.1 200 OK
Content-Length: 41
Content-Type: application/json

["Duke","John","Jane","Arun","Christina"]
```

Terminal 1 Output

```
** Waiting 2000ms **
```

MicroProfile Fault Tolerance

This section will utilize fault tolerance patterns in the frontend service to handle problematic conditions caused by the student service.

1. Add a timeout to `FrontendResource.listStudents()`

   FrontendResource.java

   ```
   @Timeout  ①
   @GET
   @Path("/list")
   @Produces(MediaType.APPLICATION_JSON)
   public List<String> listStudents() {
       List<String> students = student.listStudents();

       return students;
   }
   ```

   ① Add `@Timeout` annotation, which defaults to 1000ms

   ---

2. Check endpoint. Verify `org.eclipse.microprofile.faulttolerance.exceptions.TimeoutException` is thrown.

   Terminal 3

   ```
   $ curl -i localhost:8080/frontend/list
   ```

   Terminal 3 Output

   ```
   # Stack trace ...
   org.jboss.resteasy.spi.UnhandledException:
   org.eclipse.microprofile.faulttolerance.exceptions.TimeoutException:
   Timeout[org.acme.FrontendResource#listStudents] timed out
   # Stack trace ...
   ```

   Terminal 2 Output

   ```
   # Stack trace ...
   org.jboss.resteasy.spi.UnhandledException:
   org.eclipse.microprofile.faulttolerance.exceptions.TimeoutException:
   Timeout[org.acme.FrontendResource#listStudents] timed out
   # Stack trace ...
   ```

Terminal 1 Output

```
** Waiting 2000ms **
```

3. Add a src/main/java/org/acme/ListStudentsFallbackHandler.java fallback class to provide alternative logic when an exception is thrown

   ListStudentsFallbackHandler.java

```
package org.acme;

import java.util.Arrays;
import java.util.List;
import javax.inject.Inject;
import org.eclipse.microprofile.faulttolerance.ExecutionContext;
import org.eclipse.microprofile.faulttolerance.FallbackHandler;
import org.eclipse.microprofile.metrics.MetricRegistry;
import org.eclipse.microprofile.metrics.MetricRegistry.Type;
import org.eclipse.microprofile.metrics.annotation.RegistryType;

/*
A fallback handler has access to the ExecutionContext, which can
be used to determine where the failure originated (class and method)
and the cause of the failure
Print the cause of the failure to stdout, and return a list of "top students"
*/
public class ListStudentsFallbackHandler implements FallbackHandler<List<String>> {
    @Inject
    @RegistryType(type = Type.APPLICATION)
    MetricRegistry registry;

    @Override
    public List<String> handle(ExecutionContext ctx) {
        List<String> students = Arrays.asList("Smart Sam",
                                              "Genius Gabby",
                                              "AStudent Angie",
                                              "Intelligent Irene");
        String failure;
        registry.counter("listStudentsCounter").inc();
        if (ctx.getFailure() == null) {
            failure = "unknown";
        } else {
            failure = ctx.getFailure().getMessage();
        }
        System.out.println("Exception " + failure);
        System.out.println("listStudentsFallbackCounter: "
        + registry.counter(
            "ft.org.acme.FrontendResource.listStudents.fallback.calls.total")
            .getCount());
        return students;
    }
}
```

4. Call the fallback method to provide alternative logic when an exception is thrown

   FrontendResource.java

   ```
   @Fallback(value = ListStudentsFallbackHandler.class) ①
   @Timeout
   @GET
   @Path("/list")
   @Produces(MediaType.APPLICATION_JSON)
   public List<String> listStudents() {
       List<String> students;

       students = student.listStudents();

       return students;
   }
   ```

   ① Add `@Fallback` annotation

5. Check endpoint. Verify the fallback student list is retrieved

   Terminal 3

   ```
   $ curl -i localhost:8080/frontend/list
   ```

   Terminal 2 Output

   ```
   Exception Timeout[org.acme.FrontendResource#listStudents] timed out
   listStudentsFallbackCounter: 1
   ```

   Terminal 3 Output

   ```
   HTTP/1.1 200 OK
   Content-Length: 66
   Content-Type: application/json

   ["Smart Sam","Genius Gabby","A-Student Angie","Intelligent Irene"]
   ```

   Terminal 1 Output

   ```
   ** Waiting 2000ms **
   ```

6. Disable all fault tolerance annotations (except `@Fallback`). Useful for when running in a service mesh (e.g. Istio) environment. Commenting out any one of the timeout-disabling properties will disable the timeout.

frontend microprofile-config.properties

```
# Disable fault tolerance globally
MP_Fault_Tolerance_NonFallback_Enabled=false ①

# Disable group policy:
#Timeout/enabled=false

# Disable a specific fault tolerance policy. Ex:
#org.acme.FrontendResource/listStudents/Timeout/enabled=false
```

① All fault tolerance annotations disabled because this annotation is not commented out

7. Check that original list of students is returned

Terminal 3

```
$ curl -i localhost:8080/frontend/list
```

Terminal 3 Output

```
HTTP/1.1 200 OK
Content-Length: 41
Content-Type: application/json

["Duke","John","Jane","Arun","Christina"]
```

Terminal 1 Output

```
** Waiting 2000ms **
```

8. Comment out `MP_Fault_Tolerance_NonFallback_Enabled=false` in `microprofile-config.properties`

frontend microprofile-config.properties

```
# Disable fault tolerance globally
#MP_Fault_Tolerance_NonFallback_Enabled=false ①

# Disable group policy:
#Timeout/enabled=false

# Disable a specific fault tolerance policy. Ex:
#org.acme.FrontendResource/listStudents/Timeout/enabled=false
```

① Commented out

NOTE Feel free to uncomment the more specific approaches (all timeouts or just the timeout on `listStudents()`) and try them out. Just remember to comment them all out before continuing beyond this step.

9. External configuration of fault tolerance parameters.

   NOTE MicroProfile Fault Tolerance allows any fault tolerance annotation parameter to be configured in microprofile-config.properties, overriding the value in source code.

   frontend microprofile-config.properties

   ```
   ...
   ...
   #org.acme.FrontendResource/listStudents/Timeout/enabled=false
   org.acme.FrontendResource/listStudents/Timeout/value=3000 ①
   ```

   ① Add this, making the timeout longer than the wait time and preventing the fallback from being called.

   Terminal 3

   ```
   $ curl -i localhost:8080/frontend/list
   ```

   Terminal 3 Output

   ```
   HTTP/1.1 200 OK
   Content-Length: 41
   Content-Type: application/json

   ["Duke","John","Jane","Arun","Christina"]
   ```

   Terminal 1 Output

   ```
   ** Waiting 2000ms **
   ```

10. Comment out timeout value in `microprofile-config.properties` so annotation parameter values are used

frontend microprofile-config.properties

```
...
...
#org.acme.FrontendResource/listStudents/Timeout/enabled=false
#org.acme.FrontendResource/listStudents/Timeout/value=3000 ①
```

① Comment this out

---

11. Check endpoint. Verify fallback list of students is retrieved

    Terminal 3

    ```
    $ curl -i localhost:8080/frontend/list
    ```

    Terminal 2 Output

    ```
    Exception Timeout[org.acme.FrontendResource#listStudents] timed out
    listStudentsFallbackCounter: 2
    ```

    Terminal 3 Output

    ```
    HTTP/1.1 200 OK
    Content-Length: 66
    Content-Type: application/json

    ["Smart Sam","Genius Gabby","A-Student Angie","Intelligent Irene"]
    ```

    Terminal 1 Output

    ```
    ** Waiting 2000ms **
    ```

---

12. Update doDelay() in StudentResource.java to return a random delay.

StudentResource.java

```
void doDelay() {
    int delayTime;
    try {
        delayTime=(int)(Math.random()*delay); ①
        System.out.println("** Waiting " + delayTime + "ms **");
        TimeUnit.MILLISECONDS.sleep(delayTime);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

① Updated code to print random number: `delayTime=(int)(Math.random()*delay);`

---

13. Verify random sleep time.

Terminal 3

```
$ curl -i localhost:8080/frontend/list
```

Terminal 2 Output

```
# Depending on random timeout, may show:
Exception Timeout[org.acme.FrontendResource#listStudents] timed out
listStudentsFallbackCounter: 3
```

Terminal 3 Output

```
HTTP/1.1 200 OK
Content-Length: 66
Content-Type: application/json

["Smart Sam","Genius Gabby","A-Student Angie","Intelligent Irene"]
or
["Duke","John","Jane","Arun","Christina"]
```

NOTE Because the delay is random, a longer delay will return the fallback student list, and a shorter delay will return the original student list.

Terminal 1 Output

```
** Waiting 1-1000ms ** ①
```

① This will be a random number between 1 and 1000

NOTE Retry a few times to see random sleep times. Keep retrying until Timeout threshold is reached and fallback method is called.

14. Add a @Retry annotation, which by default will retry a request up to 3 times when exception is caught (e.g. TimeoutException)

    FrontendResource.java

    ```
    @Timeout
    @Retry  ①
    @Fallback(value = ListStudentsFallbackHandler.class)
    @GET
    @Path("/list")
    @Produces(MediaType.APPLICATION_JSON)
    public List<String> listStudents() {
        List<String> students;

        students = student.listStudents();

        return students;
    }
    ```

    ① Add this

15. Check retry logic

    Terminal 3

    ```
    $ curl -i localhost:8080/frontend/list
    ```

    Terminal 2 Output

    ```
    # Depending on random timeout, may show:
    Exception Timeout[org.acme.FrontendResource#listStudents] timed out
    listStudentsFallbackCounter: 4
    ```

    Terminal 3 Output

    ```
    HTTP/1.1 200 OK
    Content-Length: 66
    Content-Type: application/json

    ["Smart Sam","Genius Gabby","A-Student Angie","Intelligent Irene"]
    or
    ["Duke","John","Jane","Arun","Christina"]
    ```

Terminal 1 Output

```
** Waiting 1-1000ms ** ①
```

① One line will be displayed if less than 500ms, more than one line if more than 500ms due to retry

NOTE Re-run command until there are at least two output lines in Terminal 1 for a single `curl` command, at least one of which will be more than 500ms.

16. Create file src/main/java/org/acme/CircuitBreakerTracker.java

This class will be used to track the state of our upcoming circuitbreaker. It will print the state of the circuitbreaker to stdout.

CircuitBreakerTracker.java

```
package org.acme;

import java.time.Duration;
import java.util.HashMap;
import java.util.Map;

import javax.enterprise.context.Dependent;
import javax.inject.Inject;

import org.eclipse.microprofile.metrics.Gauge;
import org.eclipse.microprofile.metrics.MetricID;
import org.eclipse.microprofile.metrics.MetricRegistry;
import org.eclipse.microprofile.metrics.MetricRegistry.Type;
import org.eclipse.microprofile.metrics.annotation.RegistryType;

@Dependent
public class CircuitBreakerTracker {
    @Inject
    @RegistryType(type = Type.APPLICATION)
    MetricRegistry registry;

    public Map<String, String> track() {
        HashMap<String, String> map = new HashMap<>();
        MetricID id = new
MetricID("ft.org.acme.FrontendResource.listStudents.circuitbreaker.closed.total");
        Gauge gauge = registry.getGauges().get(id);

        if (gauge != null) {
            map.put("CBClosedTime", "" + Duration.ofNanos((long)
gauge.getValue()).toMillis() + "ms\n");
        }
        id = new
MetricID("ft.org.acme.FrontendResource.listStudents.circuitbreaker.halfOpen.total")
;
        gauge = registry.getGauges().get(id);
        if (gauge != null) {
            map.put("CBHalfOpenTime", "" + Duration.ofNanos((long)
gauge.getValue()).toMillis() + "ms\n");
        }

        id = new
MetricID("ft.org.acme.FrontendResource.listStudents.circuitbreaker.open.total");
        gauge = registry.getGauges().get(id);
        if (gauge != null) {
            map.put("CBOpenTime", "" + Duration.ofNanos((long)
gauge.getValue()).toMillis() + "ms\n");
        }

        map.put("CBSucceededCount",
                "" +
registry.counter("ft.org.acme.FrontendResource.listStudents.circuitbreaker.callsSuc
ceeded.total")
                        .getCount() + "\n");
        map.put("CBPreventedCount",
                "" +
registry.counter("ft.org.acme.FrontendResource.listStudents.circuitbreaker.callsPre
vented.total")
                        .getCount() + "\n");

        map.forEach((key, value) -> System.out.print(key + ": " + value));
        System.out.println();

        return map;

    }
}
```

17. Replace `@Timeout` logic with a `@CircuitBreaker`

FrontendResource.java

```
// @Timeout                          ①
// @Retry                            ②
@CircuitBreaker(                     ③
    requestVolumeThreshold = 4,      ④
    failureRatio = 0.5,              ⑤
    delay = 10000,                   ⑥
    successThreshold = 3             ⑦
    )
@Fallback(value = ListStudentsFallbackHandler.class)
@GET
@Path("/list")
@Produces(MediaType.APPLICATION_JSON)
public List<String> listStudents() {
    List<String> students;

    students = student.listStudents();

    return students;
}
```

① Comment out @Timeout

② Comment out @Retry

③ Add a circuit breaker. If circuit breaker throws a CircuitBreakerOpen exception, the @Retry annotation will retry the request.

④ Rolling window of 4 requests.

⑤ % of failures within the window that cause the circuit breaker to transition to "open"state

⑥ Wait 1000 milliseconds before allowing another request. Until then, each request will result in a CircuitBreakerOpen exception

⑦ Number of consecutive successful requests before circuit transitions from the half-open state to the closed state. The circuit breaker enters the half-open state upon the first successful request.

18. Inject the CircuitbreakerTracker instance in FrontendResource.java

FrontendResource.java

```
@Path("/frontend")
public class FrontendResource {
    @Inject                         ①
    CircuitBreakerTracker tracker;

    @Inject
    @RestClient
    StudentRestClient student;
```

① Inject an instance of the circuitbreaker tracker

---

19. Output circuit breaker stats

    FrontendResource.java

```
List<String> students;

tracker.track(); ①

students = student.listStudents();
```

① Display the circuitbreaker state

---

20. Inject the CircuitbreakerTracker instance in ListStudentsFallbackHandler.java

    ListStudentsFallbackHandler.java

```
@Inject
@RegistryType(type = Type.APPLICATION)
MetricRegistry registry;

@Inject                         ①
CircuitBreakerTracker tracker;
```

① Inject the tracker

---

21. Output circuit breaker stats in ListStudentsFallbackHandler.java

ListStudentsFallbackHandler.java

```
System.out.println("Exception " + failure);
System.out.println("listStudentsFallbackCounter: " +
registry.counter("ft.org.acme.FrontendResource.listStudents.fallback.calls.total").
getCount());
tracker.track();  ①
```

① Display the circuitbreaker state

---

22. Check CircuitBreaker logic

    Terminal 3

    ```
    $ curl -i localhost:8080/frontend/list
    ```

    Terminal 3 Output

    ```
    HTTP/1.1 200 OK
    Content-Length: 66
    Content-Type: application/json

    ["Duke","John","Jane","Arun","Christina"]
    ```

    Terminal 2

    ```
    CBClosedTime: 4ms         ①
    CBOpenTime: 0ms           ②
    CBSucceededCount: 0       ③
    CBPreventedCount: 0       ④
    CBHalfOpenTime: 0ms       ⑤
    ```

    ① Amount of time circuitbreaker is in the closed state

    ② Amount of time circuitbreaker is in the open state

    ③ Number of times a call is successfully completed

    ④ Number of times a call was prevented due to circuit breaker being open

    ⑤ Amount of time circuitbreaker is in the half-open state

    Terminal 1 Output

    ```
    ** Waiting 1-1000ms **
    ```

23. Stop student service

Terminal 1

```
CTRL-C
```

24. Check the circuit breaker

    This will result in `java.net.ConnectException`. When circuit breaker delay is exceeded, then the circuit breaker throws a CircuitBreakerOpenException. Both exceptions are caught by fallback logic to invoke fallback method. Try running this a few times, waiting 10-15 seconds occasionally.

    Terminal 3

    ```
    $ curl -i localhost:8080/frontend/list
    ```

    Terminal 2

    ```
    Exception RESTEASY004655: Unable to invoke request: java.net.ConnectException:
    Connection refused (Connection refused)
    listStudentsFallbackCounter: 1
    CBClosedTime: 6770ms
    CBOpenTime: 0ms
    CBSucceededCount: 1
    CBPreventedCount: 0
    CBHalfOpenTime: 0ms
    ```

    Terminal 1

    ```
    ** Waiting 844ms **
    ```

25. Re-run student service

    Terminal 1

    ```
    mvn compile quarkus:dev -Ddebug=5006
    ```

26. Retry until circuit breaker closes and the normal student list is displayed.

    Terminal 3

    ```
    $ curl -i localhost:8080/frontend/list
    ```

Terminal 3 Output

```
HTTP/1.1 200 OK
Content-Length: 66
Content-Type: application/json

["Smart Sam","Genius Gabby","A-Student Angie","Intelligent Irene"]
```

Retry the command until the primary student list is displayed. During this time, you will see changes to the stats output to the CLI.

Example:

Terminal 2

```
CBClosedTime: 25202ms
CBOpenTime: 28069ms
CBSucceededCount: 4
CBPreventedCount: 3
CBHalfOpenTime: 19369ms
```

MicroProfile Metrics

This section will cover business and performance metrics that will be graphed in Prometheus and Grafana in the packaging and deployment section.

1. Enable MicroProfile 2.3 REST metrics

   While REST metrics in MicroProfile Metrics are optional to support, Quarkus does support them. However, in Quarkus, metrics must be enabled on an extension-by-extension basis. See https://quarkus.io/guides/all-config .

   FrontendService microprofile-config.properties

   ```
   # ...
   # org.acme.FrontendResource/listStudents/Timeout/value=3000
   quarkus.resteasy.metrics.enabled=true ①
   ```

   ① Enable rest metrics

2. View all default metrics (in Prometheus/OpenMetrics format)

   Terminal 3

   ```
   $ curl -i http://localhost:8080/metrics
   ```

   Terminal 3 Example Output

   ```
   ...
   ...
   # TYPE
   application_ft_org_acme_FrontendResource_listStudents_circuitbreaker_callsSucceeded
   _total counter
   application_ft_org_acme_FrontendResource_listStudents_circuitbreaker_callsSucceeded
   _total 2.0
   # HELP base_gc_time_total Displays the approximate accumulated collection elapsed
   time in milliseconds. This attribute displays -1 if the collection elapsed time is
   undefined for this collector. The Java virtual machine implementation may use a
   high resolution timer to measure the elapsed time. This attribute may display the
   same value even if the collection count has been incremented if the collection
   elapsed time is very short.
   # TYPE base_gc_time_total counter
   base_gc_time_total_seconds{name="PS MarkSweep"} 0.156
   ...
   ...
   ```

   NOTE OpenMetrics format provides metadata like metrics type (ex: gauge) and description

3. View base metrics (in JSON this time)

   Terminal 3

   ```
   $ curl -i -H "Accept: application/json" \
   http://localhost:8080/metrics/base
   ```

   Terminal 3 Example Output

   ```
   HTTP/1.1 200 OK
   Access-Control-Allow-Origin: *
   Access-Control-Allow-Credentials: true
   Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS, HEAD
   Access-Control-Max-Age: 1209600
   Access-Control-Allow-Headers: origin, content-type, accept, authorization
   Content-Type: application/json
   content-length: 630


   {
       "gc.total;name=PS MarkSweep": 2,
       "cpu.systemLoadAverage": 2.1572265625,
       "thread.count": 78,
       "classloader.loadedClasses.count": 8145,
       "classloader.unloadedClasses.total": 26,
       "gc.total;name=PS Scavenge": 7,
       "gc.time;name=PS MarkSweep": 75,
       "jvm.uptime": 6725918,
       "thread.max.count": 158,
       "memory.committedHeap": 879230976,
       "classloader.loadedClasses.total": 8171,
       "cpu.availableProcessors": 12,
       "gc.time;name=PS Scavenge": 72,
       "thread.daemon.count": 12,
       "memory.maxHeap": 7635730432,
       "cpu.processCpuLoad": 0.00015370844246171116,
       "memory.usedHeap": 102588008
   }
   ```

4. View vendor-specific (Quarkus) metrics (in JSON)

   Terminal 3

   ```
   $ curl -i -H "Accept: application/json" \
   http://localhost:8080/metrics/vendor
   ```

Terminal 3 Example Output

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS, HEAD
Access-Control-Max-Age: 1209600
Access-Control-Allow-Headers: origin, content-type, accept, authorization
Content-Type: application/json
content-length: 933


{
    "memory.freePhysicalSize": 185147392,
    "memoryPool.usage;name=Metaspace": 41917128,
    "memoryPool.usage.max;name=PS Eden Space": 534773760,
    "memoryPool.usage;name=PS Eden Space": 0,
    "memoryPool.usage.max;name=PS Old Gen": 26178520,
    "memoryPool.usage;name=PS Old Gen": 26162136,
    "cpu.processCpuTime": 23883246000,
    "memory.committedNonHeap": 62717952,
    "memoryPool.usage.max;name=PS Survivor Space": 22014064,
    "memoryPool.usage.max;name=Compressed Class Space": 5191952,
    "memoryPool.usage;name=Code Cache": 12367808,
    "memory.freeSwapSize": 185192448,
    "memoryPool.usage.max;name=Metaspace": 41909544,
    "cpu.systemCpuLoad": 0.059001660401582626,
    "memoryPool.usage.max;name=Code Cache": 12367808,
    "memory.usedNonHeap": 59479424,
    "memoryPool.usage;name=PS Survivor Space": 20868400,
    "memoryPool.usage;name=Compressed Class Space": 5193208,
    "memory.maxNonHeap": -1
}
```

5. View application metrics (in JSON)

   Terminal 3

   ```
   $ curl -i -H "Accept: application/json" \
   http://localhost:8080/metrics/application
   ```

Terminal 3 Example Output

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS, HEAD
Access-Control-Max-Age: 1209600
Access-Control-Allow-Headers: origin, content-type, accept, authorization
Content-Type: application/json
content-length: 1162


{
    "ft.org.acme.FrontendResource.listStudents.circuitbreaker.closed.total":
229004662188,
    "ft.org.acme.FrontendResource.listStudents.circuitbreaker.callsFailed.total":
4,

"ft.org.acme.FrontendResource.listStudents.retry.callsSucceededNotRetried.total":
5,
    "ft.org.acme.FrontendResource.listStudents.invocations.total": 9,
    "ft.org.acme.FrontendResource.listStudents.circuitbreaker.open.total":
138015497877,
    "ft.org.acme.FrontendResource.listStudents.retry.callsFailed.total": 4,
    "ft.org.acme.FrontendResource.listStudents.retry.retries.total": 16,
...
...
...
```

6. Add `@Counted` to `FrontendResource`, counting invocations for each method

FrontendResource.java

```
@Counted(                           ①
    absolute = true,
    name = "FrontendCounter")
@Path("/frontend")
public class FrontendResource {

    @Inject
    @RestClient
    StudentRestClient student;
// ...
```

① Add `@Counted` annotation

7. Time `listStudents()` method duration

FrontendResource.java

```
@SimplyTimed(absolute = true,                                    ①
       name = "listStudentsTime",                                ②
       displayName = "FrontendResource.listStudents()")          ③
@Retry(maxRetries = 4,delay = 1000)
@CircuitBreaker(
    requestVolumeThreshold = 4,
    failureRatio = 0.5,
    delay = 10000,
    successThreshold = 2)
@Fallback(value = ListStudentsFallbackHandler.class)
@GET
@Path("/list")
@Produces(MediaType.APPLICATION_JSON)
public List<String> listStudents() {
    List<String> students = student.listStudents();

    return students;
}
```

① **absolute** Remove package name. Metric uses name parameter if it exists, if not it uses the name of the class or method.

② **name** Metric name (custom name)

③ **displayName** Human-readable name

8. View `@SimplyTimed` metrics

Terminal 3

```
$ curl -i -s localhost:8080/metrics | grep -i "elapsedTime" | grep -v TYPE
```

Terminal 3 Example Output

```
application_listStudentsTime_elapsedTime_seconds 1.009317591
base_REST_request_elapsedTime_seconds{class="org.acme.FrontendResource",method="lis
tStudents"} 1.011809823
```

NOTE Notice some metrics have curly braces around them "{}". These are metric tags that subset a metric. See the metrics-tags example to see metric tags in action.

9. View `@Counted` metrics

Terminal 3

```
$ curl -i localhost:8080/metrics/application | grep -i count | grep -v TYPE
```

Terminal 3 Example Output

```
application_FrontendCounter_listStudentsFallback_total 0.0
application_FrontendCounter_hello_total 0.0
application_FrontendCounter_FrontendResource_total 1.0
application_FrontendCounter_listStudents_total 2.0
```

10. Add a field to hold the size of the student list

    FrontendResource.java

    ```
    public class FrontendResource {
        int numStudents; ①
    ```

    ① Add this

11. Assign list size to `numStudents` in `listStudents()`

    FrontendResource.java

    ```
    public List<String> listStudents() {
        List<String> students = student.listStudents();
        numStudents = students.size(); ①

        return students;
    }
    ```

    ① Add this

12. Replace Fallback class with Fallback method

    FrontendResource.java

    ```
        @CircuitBreaker(
            requestVolumeThreshold = 4,
            failureRatio = 0.5,
            delay = 10000,
            successThreshold = 2)
        // @Fallback(value = ListStudentsFallbackHandler.class) ①
        @Fallback(fallbackMethod = "listStudentsFallback")      ②
    ```

① Comment out @Fallback that falls back to a handler class

② Create a new @Fallback that falls back to method

13. Create a Fallback method

    FrontendResource.java

    ```
    public List<String> listStudentsFallback() {
        List<String> students = Arrays.asList(
            "Smart Sam",
            "Genius Gabby",
            "A-Student Angie",
            "Intelligent Irene");

        numStudents = students.size();①

        return students;
    }
    ```

    ① This method also stores the student size

14. Create a gauge to display number of students

    FrontendResource.java

    ```
    @Gauge(unit = MetricUnits.NONE, name = "numberOfStudents",
        absolute = true)
    public int getNumberOfStudents() {
        return numStudents;
    }
    ```

15. "Prime" numStudents by calling listStudents()

    Terminal 3

    ```
    $ curl -i localhost:8080/frontend/list
    ```

    Terminal 3 Output

    ```
    HTTP/1.1 200 OK
    Content-Length: 41
    Content-Type: application/json

    ["Duke","John","Jane","Arun","Christina"]
    ```

16. Get number of students using the gauge

    Terminal 3

    ```
    $ curl -i -s localhost:8080/metrics/application  | \
      grep -i application_numberOfStudents
    ```

    Terminal 3 Output

    ```
    # TYPE application_numberOfStudents gauge
    application_numberOfStudents 5.0
    ```

Another useful example, beyond this tutorial, is the health-metrics example, which demonstrates the use of the metrics repository API.

MicroProfile Health

This section will create an endpoint that exposes the health of the student service. The logic will result in the student service being healthy 50% of the time. This will be checked using a CLI, but in the packaging section will be checked using a docker-compose healthcheck.

1. Verify default health check endpoint

   Terminal 3

   ```
   $ curl -i localhost:8082/health
   ```

   Terminal 3 Output

   ```
   HTTP/1.1 200 OK
   content-type: application/json; charset=UTF-8
   content-length: 46


   {
       "status": "UP",
       "checks": [
       ]
   }
   ```

---

2. Create a MicroProfile Health Endpoint

student/src/main/java/org/acme/StudentHealth.java

```
package org.acme;

import org.eclipse.microprofile.health.HealthCheck;
import org.eclipse.microprofile.health.HealthCheckResponse;
import org.eclipse.microprofile.health.Liveness;
import org.eclipse.microprofile.health.Readiness;

@Liveness  ①
@Readiness ②
public class StudentHealth implements HealthCheck {
    @Override
    public HealthCheckResponse call() {
        double random = Math.random();
        return HealthCheckResponse
            .named("StudentLivenessReadiness")         ③
            .state(random < .50 ? true : false)    ④
            .withData("randomNumber", "" + random) ⑤
            .build();
    }
}
```

① Restart unrecoverable service

② Pause traffic until ready

③ A healthcheck can be named

④ State is UP (true) or DOWN (false)

⑤ Data can be added to provide state context

NOTE Retry a few times until both UP and DOWN have been displayed across subsequent requests. If there is more than one health check class in an application, then all must be UP for overall state to be UP.

NOTE Typically there would be a separate health check class for readiness and liveness, but shown here in a single class for "conciseness" under time constraints.

3. Check health liveness endpoint specifically

Terminal 3

```
$ curl -i localhost:8080/health/live
```

Terminal 3 Output

```
HTTP/1.1 503 Service Unavailable  ①
content-type: application/json; charset=UTF-8
content-length: 231


{
    "status": "DOWN",
    "checks": [
        {
            "name": "StudentLivenessReadiness",
            "status": "DOWN",
            "data": {
                "randomNumber": "0.60806403626233085"
            }
        }
    ]
}
```

① The HTTP Reponse code will be 503 when a service is down

NOTE There is a /health/ready endpoint as well

MicroProfile JWT RBAC

This section will secure the student service and frontend service endpoints, and propagate a bearer token across services.

NOTE A token has already been generated using a supplied build of Adam Bien's jwtenizr. In addition, to facilitate these instructions, the token will last until 2070 :-)

Because tokens are base64 encoded, they can be easily decoded. jwt.io can display jwt tokens and verify them using the issuer's public key. (Click here) to see the token used in this course.

1. Add the MicroProfile dependency to **both the student service and frontend service** `pom.xml` files.

   pom.xml

   ```
   <dependency> ①
       <groupId>io.quarkus</groupId>
       <artifactId>quarkus-smallrye-jwt</artifactId>
   </dependency>
   ```

   ① Add this

2. Add required MicroProfile JWT RBAC properties to **both the student service and frontend service** `microprofile-config.properties` files.

   microprofile-config.properties

   ```
   mp.jwt.verify.issuer=airhacks
   mp.jwt.verify.publickey=MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAtBR6TwVxolT5E2e
   mnQEwqJztmeWRThU4ZA3V9+4vjOXoNmSKWrLfqLaKuMric9opYQi86yO1o0qChkAnlRY7ZytcaFqcehYOSA
   hcghYNn4Wzi70D2lJHj/YflFKdssySyNzqMIBMxNWZWx8kIVDRrVamsmF2Fo4Dg72ce8KiMSlqkWrHiSbfW
   pa2aQru9dEhErJPf05fGzQWwtvOvtLCp/tLXq7GmTE2XJJdiCk3CdE3OP/FQRWyeRtHk6Uq4hjzXTX6Wnrb
   7xDZCjQubfWYq9yoINet1eMFWFUXRsAJQbMJKIstcCvwmO35iPjFrftWTADOh3pzIARVqWwupDN7fwIDAQA
   B
   ```

Securing Frontend Service

1. Create a new endpoint in `FrontendResource.java` to display the Principal

   FrontendResource.java

   ```
   @Inject
   Principal principal;

   @GET
   @Path("/tokeninfo")
   @Produces(MediaType.TEXT_PLAIN)
   public String tokeninfo() {
       String string = "Principal: " + principal.getName();
       return string;
   }
   ```

2. Check endpoint, which should return 'null' since no principal has been supplied

   Terminal 3

   ```
   $ curl -i localhost:8080/frontend/tokeninfo
   ```

   Terminal 3 Output

   ```
   HTTP/1.1 200 OK
   Content-Length: 15
   Content-Type: text/plain;charset=UTF-8

   Principal: null
   ```

3. Assign token to a `TOKEN` environmental variable

   Terminal 3

   ```
   $ export
   TOKEN="eyJraWQiOiJqd3Qua2V5IiwidHlwIjoiSldUIiwiYWxnIjoiUlMyNTYifQ.eyJzdWIiOiJ1c2VyX
   C80Mzk3MSIsInVwbiI6ImRlbW9AYWNtZS5vcmciLCJteWMiOiJNeSBDdXN0b20gQ2xhaW0iLCJhdXRoX3Rp
   bWUiOjE1Nzg2NTEyODMsImlzcyI6ImFpcmhhY2tzIiwiZ3JvdXBzIjpbInVzZXIiLCJhZG1pbiJdLCJleHA
   iOjMxNTU4ODI4OTgsImlhdCI6MTU3ODY1MTI4MywianRpIjoiYWlyaGFja3Mtand0LXVuaXF1ZS1pZC0xMj
   M0MjE0MiJ9.Eaqe3sTH64doIVW3on25EA_uD9XrfppndiweUNLVbFK3KxaIfXaAdQ4N9IkQG6Iw0A7I7kng
   jeSHwb2DzH8rQE8yp7sCtey6kmC689eQC0j2k-YbyGZ68xnsMj5taOBVGH_ZSWC6E1L-Gk-
   GgcTvX6I3SaBC8pwZ267q6psknqlAtfD2JoE7ezEb7LrLVwP1vaGqKzC2X6pv5J-
   07DNBqe75uBWQyqX_WE856ug3uqWcHtNck8nqU6VhwXqxHZ6vkRlx9VoMgFUF851D-
   WuKMCUdfXJHekDyKmjYuyLiw7jtQSdliY3ONOXgFm_uzjKGuZ1VKPdQXyx7GQ9NsNTYfw"
   ```

4. Re-run the command, this time supplying the token:

   Terminal 3

   ```
   $ curl -i -H"Authorization: Bearer ${TOKEN}"
   http://localhost:8080/frontend/tokeninfo
   ```

   Terminal 3 Output

   ```
   HTTP/1.1 200 OK
   Content-Length: 15
   Content-Type: text/plain;charset=UTF-8

   Principal: demo@acme.org
   ```

5. Update "/tokeninfo" endpoint to display all claims

   FrontendResource.java

   ```
   @Inject
   JsonWebToken token; ①

   @GET
   @Path("/tokeninfo")
   @Produces(MediaType.TEXT_PLAIN)
   public String tokenInfo() { ②
       String string = "Principal: " + principal.getName();

       string += ",\n";

       string += token.getClaimNames()
           .stream()
           .map(claim -> "\n " + claim + ": " + token.getClaim(claim))
           .collect(Collectors.toList())
           .toString();

       return string;
   }
   ```

   ① Inject the token

   ② Replace the contents of tokenInfo

6. Check the token output

Terminal 3

```
$ curl -i -H"Authorization: Bearer ${TOKEN}"
http://localhost:8080/frontend/tokeninfo
```

Terminal 3 Output

```
HTTP/1.1 200 OK
Content-Length: 929
Content-Type: text/plain;charset=UTF-8

Principal: demo@acme.org,
[
 sub: user/43971,
 upn: demo@acme.org,
 myc: My Custom Claim,
 raw_token:
eyJraWQiOiJqd3Qua2V5IiwidHlwIjoiSldUIiwiYWxnIjoiUlMyNTYifQ.eyJzdWIiOiJ1c2VyXC80Mzk3
MSIsInVwbiI6ImRlbW9AYWNtZS5vcmciLCJteWMiOiJNeSBDdXN0b20gQ2xhaW0iLCJhdXRoX3RpbWUiOjE
1Nzg2NTEyODMsImlzcyI6ImFpcmhhY2tzIiwiZ3JvdXBzIjpbImFkbWluIiwidXNlciJdLCJleHAiOjMxNT
U4ODI4OTgsImlhdCI6MTU3ODY1MTI4MywianRpIjoiYWlyaGFja3Mtand0LXVuaXF1ZS1pZC0xMjM0MjE0M
iJ9.Eaqe3sTH64doIVW3on25EA_uD9XrfppndiweUNLVbFK3KxaIfXaAdQ4N9IkQG6Iw0A7I7kngjeSHwb2
DzH8rQE8yp7sCtey6kmC689eQC0j2k-YbyGZ68xnsMj5taOBVGH_ZSWC6E1L-Gk-
GgcTvX6I3SaBC8pwZ267q6psknqlAtfD2JoE7ezEb7LrLVwP1vaGqKzC2X6pv5J-
07DNBqe75uBWQyqX_WE856ug3uqWcHtNck8nqU6VhwXqxHZ6vkRlx9VoMgFUF851D-
WuKMCUdfXJHekDyKmjYuyLiw7jtQSdliY3ONOXgFm_uzjKGuZ1VKPdQXyx7GQ9NsNTYfw,
 auth_time: 1578651283,
 iss: airhacks,
 groups: [admin, user],
 exp: 3155882898,
 iat: 1578651283,
 jti: airhacks-jwt-unique-id-12342142]
```

7. Secure endpoints by limiting access to specified roles

FrontendResource.java

```
@RolesAllowed("user")        ①
@GET
@Path("/tokeninfo")
@Produces(MediaType.TEXT_PLAIN)
public String tokeninfo() {
    String string = "Principal: " + principal.getName();
    string += ",\n";

    string += token.getClaimNames().stream().map(tok -> "\n " + tok + ": " +
token.getClaim(tok))
           .collect(Collectors.toList()).toString();

    return string;
}

@RolesAllowed("superuser")    ②
@SimplyTimed(
    absolute = true,
    name = "listStudentsTime",
    displayName = "FrontendResource.listStudents()")
// @Fallback(value = ListStudentsFallbackHandler.class)
@Fallback(fallbackMethod = "listStudentsFallback")
// @Timeout
// @Retry
@CircuitBreaker(
    requestVolumeThreshold = 4,
    failureRatio = .5,
    delay = 10000,
    successThreshold = 3)
@GET
@Produces(MediaType.APPLICATION_JSON)
@Path("/list")
public List<String> listStudents() {
    return student.listStudents();
}
```

① Apply `@RolesAllowed("user")` to the getToken() method

② Apply `@RolesAllowed("superuser")` to the listStudents() method

8. Check the endpoints to validate access

Terminal 3

```
$ curl -i http://localhost:8080/frontend/list
```

Output

```
HTTP/1.1 401 Unauthorized
www-authenticate: Bearer {token}
Content-Length: 0
```

NOTE Access is denied because the user is anonymous and there are no roles tied to the anonymous user. Note the HTTP response code is `401 Unauthorized`

9. Retry the request using a token.

Terminal 3

```
$ curl -i -H"Authorization: Bearer ${TOKEN}" http://localhost:8080/frontend/list
```

Terminal 3 Output

```
HTTP/1.1 403 Forbidden
Content-Length: 9
Content-Type: application/json

Forbidden
```

NOTE This time access is denied because the demo user does not belong to the "superuser" group. The demo user belongs to the "user" and "admin" groups. Note the HTTP response code is `403 Forbidden`

---

10. Change the "superuser" role to the "admin" role, which the "demo" user belongs to

FrontendResource.java

```
@RolesAllowed("admin")    ①
@SimplyTimed(
    absolute = true,
    name = "listStudentsTime",
    displayName = "FrontendResource.listStudents()")
// @Fallback(value = ListStudentsFallbackHandler.class)
@Fallback(fallbackMethod = "listStudentsFallback")
// @Timeout
// @Retry
@CircuitBreaker(
    requestVolumeThreshold = 4,
    failureRatio = .5,
    delay = 10000,
    successThreshold = 3)
@GET
@Produces(MediaType.APPLICATION_JSON)
@Path("/list")
public List<String> listStudents() {
    return student.listStudents();
}
```

① Change "superuser" to "admin"

11. Check access with newly supplied "admin" role

Terminal 3

```
$ curl -i -H"Authorization: Bearer ${TOKEN}" http://localhost:8080/frontend/list
```

Terminal 3 Output

```
HTTP/1.1 200 OK
Content-Length: 41
Content-Type: application/json

["Duke","John","Jane","Arun","Christina"]
```

Securing Student Service

1. Secure `StudentResource.listStudents()`, requiring the admin role

   StudentResource.java

   ```
   @RolesAllowed("admin")  ①
   @GET
   @Path("/list")
   @Produces(MediaType.APPLICATION_JSON)
   public List<String> listStudents() {
       doDelay();
       return students;
   }
   ```

   ① Secure with "admin" role

   Terminal 3

   ```
   $ curl -i -H"Authorization: Bearer ${TOKEN}" http://localhost:8080/frontend/list
   ```

   Terminal 3 Output

   ```
   HTTP/1.1 200 OK
   Content-Length: 66
   Content-Type: application/json

   ["Smart Sam","Genius Gabby","A-Student Angie","Intelligent Irene"]
   ```

   This implies that the request to the student service is not being managed properly because the fallback output is returned.

2. The token needs to be forwarded to the student service. This requires annotating StudentRestClient with `@RegisterClientHeaders` and defining the headers to propagate (Authorization header) using the `org.eclipse.microprofile.rest.client.propagateHeaders` property.

   StudentRestClient.java

   ```
   @RegisterClientHeaders    ①
   @RegisterRestClient(configKey = "StudentService")
   @Path("/student")
   public interface StudentRestClient {
   ```

   ① Add `@RegisterClientHeaders` to frontend microprofile-config.properties *frontend/src/main/resources/META-INF/microprofile-config.properties*

```
org.eclipse.microprofile.rest.client.propagateHeaders=Authorization①

mp.jwt.verify.issuer=airhacks
mp.jwt.verify.publickey=MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAtBR6TwVxolT5E2e
mnQEwqJztmeWRThU4ZA3V9+4vjOXoNmSKWrLfqLaKuMric9opYQi86yO1o0qChkAnlRY7ZytcaFqcehYOSA
hcghYNn4Wzi70D2lJHj/YflFKdssySyNzqMIBMxNWZWx8kIVDRrVamsmF2Fo4Dg72ce8KiMSlqkWrHiSbfW
pa2aQru9dEhErJPf05fGzQWwtvOvtLCp/tLXq7GmTE2XJJdiCk3CdE3OP/FQRWyeRtHk6Uq4hjzXTX6Wnrb
7xDZCjQubfWYq9yoINet1eMFWFUXRsAJQbMJKIstcCvwmO35iPjFrftWTADOh3pzIARVqWwupDN7fwIDAQA
B
```

① Add this line to propagate the Authorization header. Additional headers can be propagated as well, separated by commas.

Terminal 3 Output

```
HTTP/1.1 200 OK
Content-Length: 41
Content-Type: application/json

["Duke"," John"," Jane"," Arun"," Christina"]
```

The token (Authorization header) has been successfully propagated.

Packaging, deploying, and monitoring

Build applications as a thin jar files, package them as docker files and start them. Once running, view health endpoint state and view application metrics in grafana.

1. Update student URI for docker deployment.

   frontend/src/main/resources/META-INF/microprofile-config.properties

   ```
   #StudentService/mp-rest/uri=http://localhost:8082 ①
   %dev.StudentService/mp-rest/uri=http://localhost:8082 ②
   %prod.StudentService/mp-rest/uri=http://student:8082 ③
   ```

   ① Comment out current uri

   ② Properties prefixed with `%dev.` will be used while in development mode only (`mvn compile quarkus:dev`).

   ③ Properties prefixed with `%prod.` will be used when running in native mode or with `java -jar` only. When running via the docker-compose, the host will be "student".

   NOTE Quarkus supports multiple configuration profiles. There is also a `%test.` profile when running tests. Properties with no profile defined are always utilized, as has been the case so far in this lab.

2. Package applications as docker files

   WARNING Make sure docker daemon is running and is accessible (ex: `docker info` shows proper results)

   Package student as thin jar and create a docker image

   ```
   $ cd tutorial/working/student
   $ mvn clean package -DskipTests
   $ docker build -t acme/student:1.0 -f src/main/docker/Dockerfile.jvm .
   ```

   Package frontend as thin jar and create a docker image

   ```
   $ cd tutorial/working/frontend
   $ mvn clean package -DskipTests
   $ docker build -t acme/frontend:1.0 -f src/main/docker/Dockerfile.jvm .
   ```

3. Stop student and frontend apps running in development mode to avoid port conflicts

Terminal 1

```
# Press CTRL-C to stop Quarkus (student)
```

Terminal 2

```
# Press CTRL-C to stop Quarkus (frontend)
```

4. Start student, frontend, prometheus, and grafana.

   Terminal 1

   ```
   $ cd tutorial/working/docker
   $ docker-compose up
   ```

5. View student service health

   Terminal 2

   ```
   $ docker-compose ps
   ```

   Terminal 2 Output

   ```
       Name           Command           State           Ports
   ------------------------------------------------------------
   docker_fronte   /deployments/   Up              0.0.0.0:8080-
   nd_1            run-java.sh                     >8080/tcp,
                                                   8778/tcp,
                                                   9779/tcp
   docker_grafan   /run.sh         Up              0.0.0.0:3000-
   a_1                                             >3000/tcp
   docker_prom_1   /bin/promethe   Up              0.0.0.0:9090-
                   us --config.f                   >9090/tcp
   docker_studen   /deployments/   Up              8080/tcp, 0.0
   t_1             run-java.sh     (unhealthy)     .0.0:8082->80 ①
                                                   81/tcp,
                                                   8778/tcp,
                                                   9779/tcp
   ```

   ① Run `docker-compose ps` until both `(healthy)` and `(unhealthy)` are displayed. In container orchestration environment, these pods containers would be restarted.

6. Get Prometheus IP address

Terminal 2

```
$ docker inspect docker_prom_1
```

Terminal 2 Output

```
...
...
        "NetworkSettings": {
            "Bridge": "",
            "SandboxID":
"acf7dc6b9591f7992fb3053639a38cc98f91281e98582b8a8a420026506d88b8",
            "HairpinMode": false,
            "LinkLocalIPv6Address": "",
            "LinkLocalIPv6PrefixLen": 0,
            "Ports": {
                "9090/tcp": [
                    {
                        "HostIp": "0.0.0.0",
                        "HostPort": "9090"
                    }
                ]
            },
            "SandboxKey": "/var/run/docker/netns/acf7dc6b9591",
            "SecondaryIPAddresses": null,
            "SecondaryIPv6Addresses": null,
            "EndpointID": "",
            "Gateway": "",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "IPAddress": "",
            "IPPrefixLen": 0,
            "IPv6Gateway": "",
            "MacAddress": "",
            "Networks": {
                "docker_default": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": [
                        "prom",
                        "ea79e602a5e5"
                    ],
                    "NetworkID":
"b4e117d0c94abe2a49a94164883405a8140acca16a1bc376f865b0dca5b839cc",
                    "EndpointID":
"bd1307800d42bb303d79d2ac8cf7bf856d84c8b84d7d4077b112822c8fb711e6",
                    "Gateway": "172.18.0.1",
                    "IPAddress": "172.18.0.4", ①
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:ac:12:00:04",
                    "DriverOpts": null
                }
            }
        }
    }
]
...
...
```

① IP address iis 172.18.0.4. IP address may vary.

NOTE `docker inspect docker_prom_1 | grep IPAddress` should also make the IP address quickly apparent.

7. Log in to Grafana

   a. Point browser to http://localhost:3000/login.

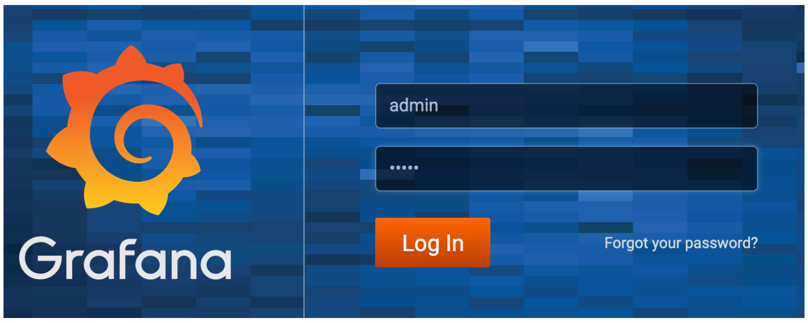

Figure 1. user:admin, password:admin

8. Add a data source

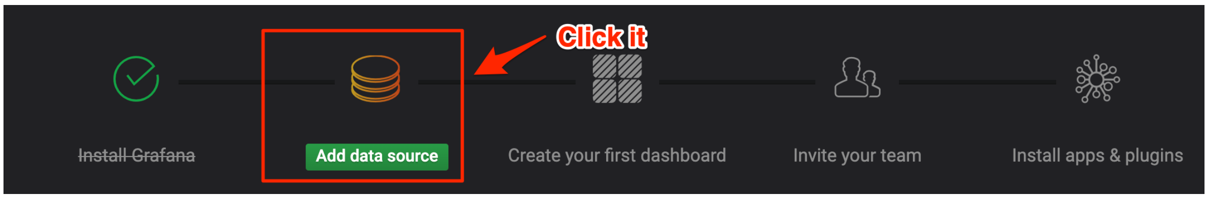

Figure 2. Click "Add datasource`

9. Filter and select Prometheus

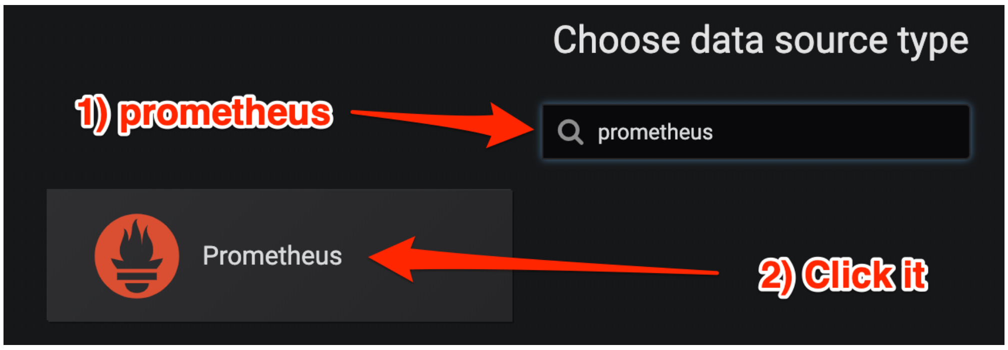

Figure 3. Filter by Prometheus and click Prometheus

10. Configure Prometheus Data Source

    Use the IP address retrieved with `docker inspect` command above

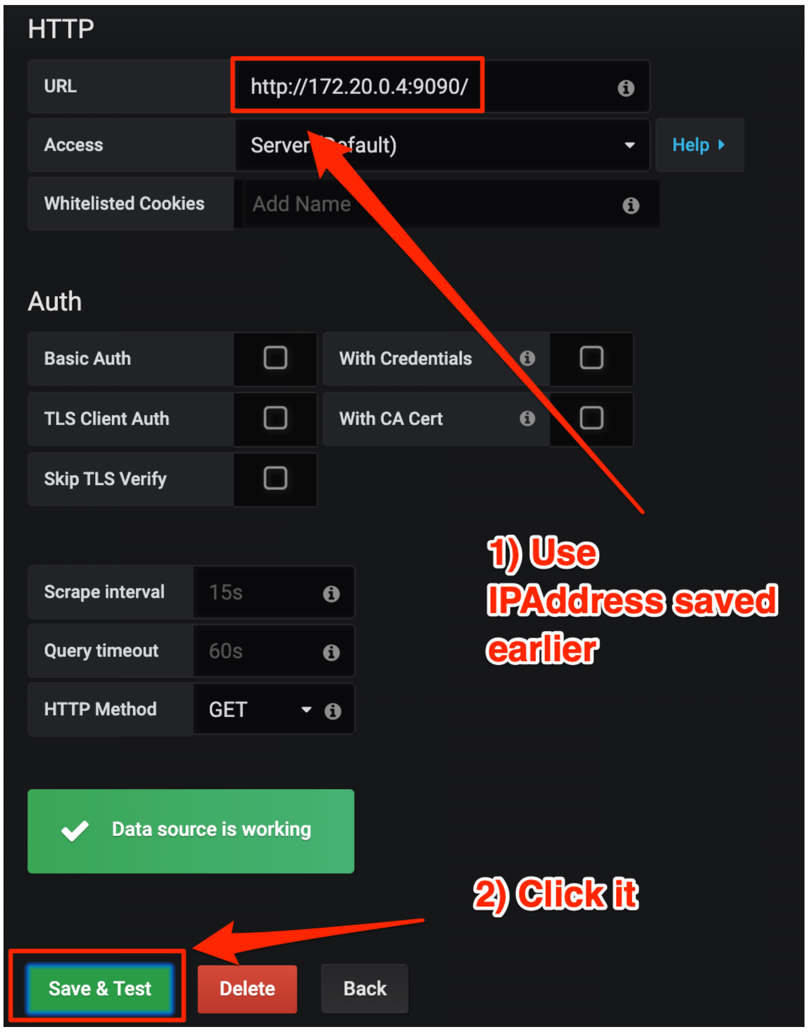

Figure 4. Configure URL using IP Address and save & test it

11. Import JSON File

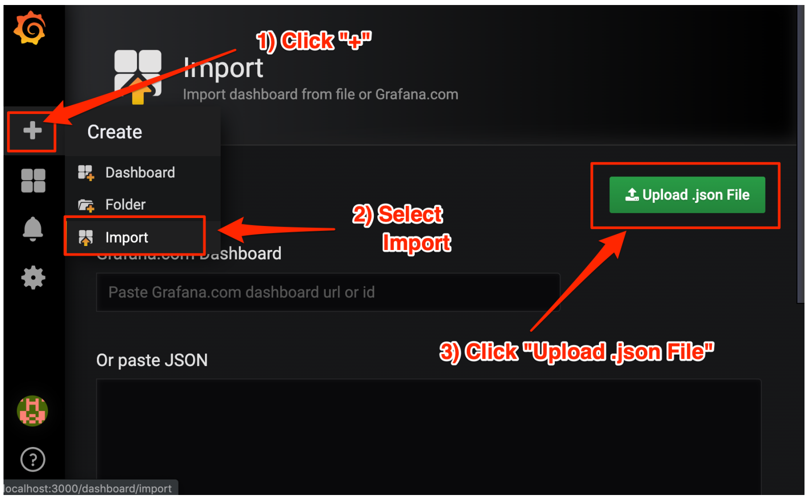
1) Click "+"
Import
Import dashboard from file or Grafana.com
Create
Dashboard
Folder
Import
2) Select Import
Upload .json File
Grafana.com Dashboard
Paste Grafana.com dashboard url or id
Or paste JSON
3) Click "Upload .json File"
localhost:3000/dashboard/import

Figure 5. Import JSON File

12. Select Dashboard - tutorial/working/docker/grafana-frontend-dashboard.json

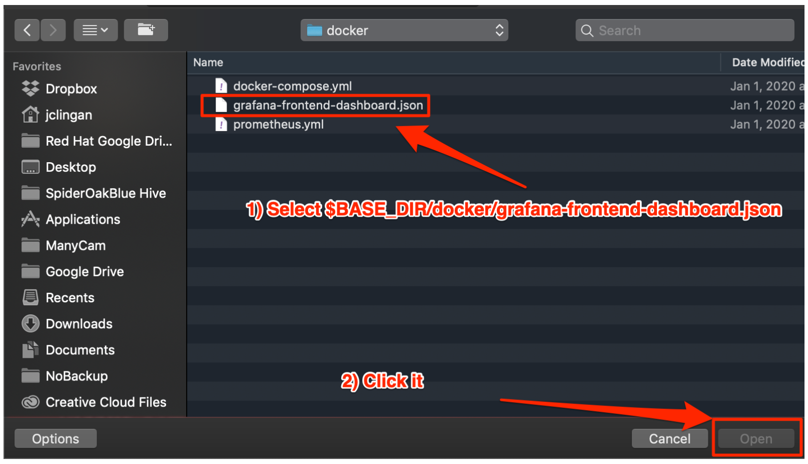

Figure 6. Select

13. Generate load by running curl a random number of times

Terminal 2

```
$ curl -i -H"Authorization: Bearer ${TOKEN}" http://localhost:8080/frontend/list
$ curl -i -H"Authorization: Bearer ${TOKEN}" http://localhost:8080/frontend/list
$ curl -i -H"Authorization: Bearer ${TOKEN}" http://localhost:8080/frontend/list
$ curl -i -H"Authorization: Bearer ${TOKEN}" http://localhost:8080/frontend/list
$ curl -i -H"Authorization: Bearer ${TOKEN}" http://localhost:8080/frontend/list
```

--image::http://localhost:3000/public/img/grafana_icon.svg[]
image::http://localhost:3000/public/img/grafana_typelogo.svg[]

+ '''

1. Stop the student service

```
$ docker-compose stop student
```

+ '''

1. Generate load by running curl a random number of times with the circuit breaker in an open state.

Terminal 2

```
$ curl -i localhost:8080/student/list
$ curl -i localhost:8080/student/list
```

+ '''

1. View the Grafana dashboard

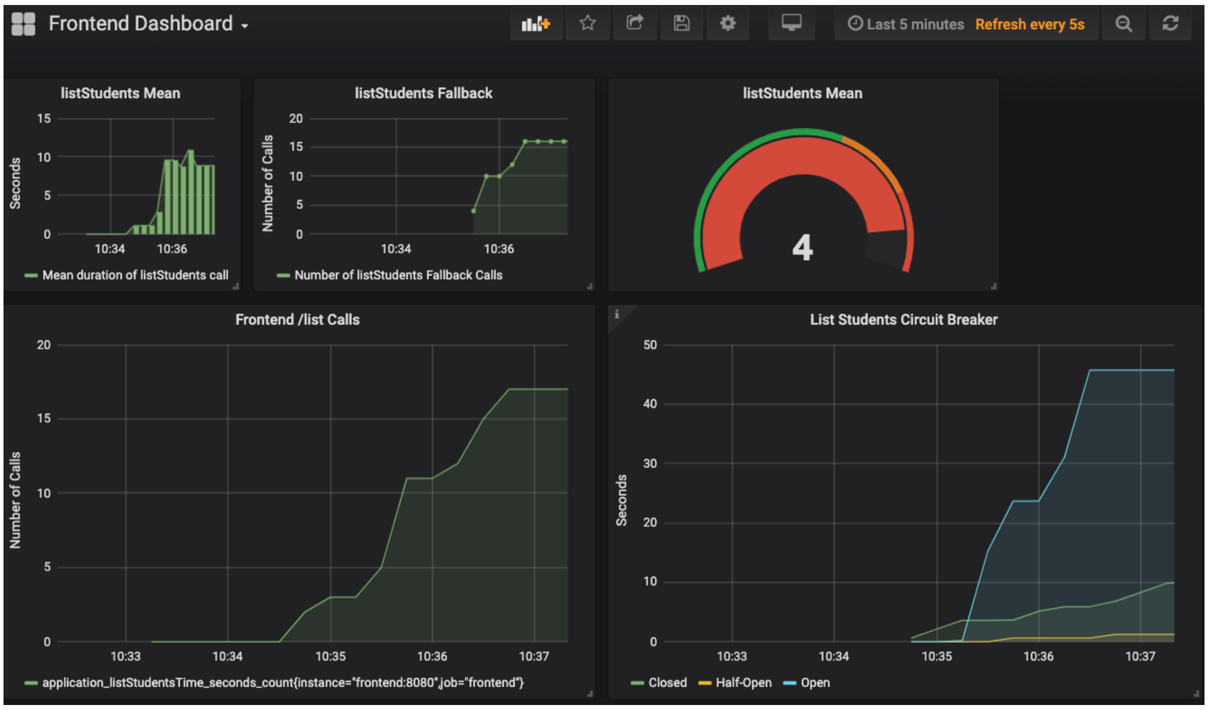

Figure 7. View Grafana Dashboard

Some interesting notes on the dashboard:

- During metrics gathering, the goal was to stop and start the student service to force some circuit breaker time in the half-open state (yellow line in lower-right hand graph). Relative to the other states, a small amount of time is spent in half-open state (due to small window [`requestVolumeThreshold`] and small `successThreshold`).
- Because of time spent with the student service stopped, there is growth in fallback calls
- The lower-left hand graph uses the MicroProfile Metrics default metric name being graphed. The other graphs uses custom names defined in the dashboard itself
- The proportionally large mean time spent in `listStudents()` (roughly 10 seconds) is due to the number or retries combined with the delay between requests - `@Retry(maxRetries = 4, delay = 1000)`

- While not implemented in this tutorial, these metrics could easily be business-oriented metrics, like 'show the average number of students retrieved per course' to display a live statistic related to class size.

== Appendix

=== Deploy to docker using native builds of `student` and `frontend`.

1. Stop student and frontend apps running in development mode to avoid port conflicts

Terminal 1

```
# Press CTRL-C to stop Quarkus (student)
```

Terminal 2

```
# Press CTRL-C to stop Quarkus (frontend)
```

1. Compile `student` native binary

Terminal 1

```
$ cd tutorial/working/student
$ mvn clean package -Dnative -Dquarkus.native.container-build=true -DskipTests
```

NOTE Native compilation is resource intensive. It may take up to 2+ minutes and utilize a lot of RAM. The Docker VM may need to be set to 4GB+ (Docker icon in menu bar or system tray → Preferences → Advanced).

Terminal 1 Output

```
...
...
[INFO] [io.quarkus.deployment.pkg.steps.NativeImageBuildStep] docker run -v
/Users/jclingan/Documents/Personal/OReilly/MicroProfile/solution/solution/student/targ
et/student-1.0-SNAPSHOT-native-image-source-jar:/project:z --rm quay.io/quarkus/ubi-
quarkus-native-image:19.2.1 -J-Dsun.nio.ch.maxUpdateArraySize=100 -J
-Djava.util.logging.manager=org.jboss.logmanager.LogManager -J-Dvertx.logger-delegate
-factory-class-name=io.quarkus.vertx.core.runtime.VertxLogDelegateFactory -J
-Dvertx.disableDnsResolver=true -J-Dio.netty.leakDetection.level=DISABLED -J
-Dio.netty.allocator.maxOrder=1 --initialize-at-build-time=
-H:InitialCollectionPolicy=com.oracle.svm.core.genscavenge.CollectionPolicy$BySpaceAnd
Time -jar student-1.0-SNAPSHOT-runner.jar -J
-Djava.util.concurrent.ForkJoinPool.common.parallelism=1 -H:FallbackThreshold=0
-H:+ReportExceptionStackTraces -H:+AddAllCharsets -H:EnableURLProtocols=http -H:-JNI
--no-server -H:-UseServiceLoaderFeature -H:+StackTrace student-1.0-SNAPSHOT-runner
[student-1.0-SNAPSHOT-runner:23]    classlist:  11,461.31 ms
[student-1.0-SNAPSHOT-runner:23]        (cap):   1,627.85 ms
[student-1.0-SNAPSHOT-runner:23]        setup:   3,978.77 ms
03:35:11,103 INFO  [org.jbo.threads] JBoss Threads version 3.0.0.Final
[student-1.0-SNAPSHOT-runner:23]   (typeflow):  21,123.43 ms
[student-1.0-SNAPSHOT-runner:23]    (objects):  17,071.98 ms
[student-1.0-SNAPSHOT-runner:23]   (features):     472.98 ms
[student-1.0-SNAPSHOT-runner:23]     analysis:  40,365.68 ms
[student-1.0-SNAPSHOT-runner:23]     (clinit):     719.13 ms
[student-1.0-SNAPSHOT-runner:23]     universe:   1,880.23 ms
[student-1.0-SNAPSHOT-runner:23]      (parse):   3,293.33 ms
[student-1.0-SNAPSHOT-runner:23]     (inline):   7,098.10 ms
[student-1.0-SNAPSHOT-runner:23]    (compile):  26,555.96 ms
[student-1.0-SNAPSHOT-runner:23]      compile:  38,699.39 ms
[student-1.0-SNAPSHOT-runner:23]        image:   2,898.33 ms
[student-1.0-SNAPSHOT-runner:23]        write:     635.92 ms
[student-1.0-SNAPSHOT-runner:23]      [total]: 100,194.70 ms
[INFO] [io.quarkus.deployment.QuarkusAugmentor] Quarkus augmentation completed in
106387ms
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  01:48 min
[INFO] Finished at: 2020-01-10T19:36:34-08:00
[INFO] ------------------------------------------------------------------------
```

1. Compile `frontend` service

Terminal 1

```
$ cd tutorial/working/frontend
$ mvn clean package -Dnative -Dquarkus.native.container-build=true -DskipTests
```

+

1. Create docker containers

Terminal 1

```
$ cd tutorial/working/student
$ docker build -t acme/student:1.0 -f src/main/docker/Dockerfile.native . ①
$ cd tutorial/working/frontend
$ docker build -t acme/frontend:1.0 -f src/main/docker/Dockerfile.native . ①
```

① Note the change to Dockerfile.native

+

1. Start student, frontend, prometheus, and grafana.

Terminal 1

```
$ cd tutorial/working/docker
$ docker-compose up
```

=== Using jwtenizr

jwtenizr is a handy tool created by Adam Bien to create MicroProfile-ready tokens. This tutorial includes a `jwt` subdirectory that includes:

- **jwtenizr.jar**: A conveniently pre-compiled jar file. Feel free to download and compile your own jar file.
- **jwtenizr.sh**: A convient shell script that invokes the jar file
- **Pre-generated artifacts**: Makes explaining this tutorial consistent over time. Includes:
  - **jwt-token.json**: Contains pe-defined token contents. An important aspect to call out is that the `exp` (expiration) claim was added to ensure this token does not expire until Jan 2070. Without this addition, jwtenizr will use an expiration time of 1000 seconds (~16 minutes) from when the token was created.
  - The remainder of the artifacts is documented in the jwtenizr github.

To generate a new token with the default 1000 second expiration time, remove the `exp` line from jwt-token.json, which would then look as follows:

jwt-token.json

```
{
  "iss":"airhacks",
  "jti":"airhacks-jwt-unique-id-12342142",
  "sub":"user/43971",
  "upn":"demo@acme.org",
  "myc":"My Custom Claim",
  "groups":[
    "user",
    "admin"
  ]
}
```

==== Generate a new token with a customized expiration (Mac/Linux)

1. Get the current date

Terminal Window

```
# Get the number of seconds since the Epoch (Jan 1, 1970) and add to it (200 seconds
in the following example):

$ $((`date +%s` + 200)) ①
1579412832 ②
```

① The first `$` is the command prompt :-)

② The output from the evaluated expression. This is the current time plus 200 seconds. Replace `200` with whatever value you want.

1. Place that number as the expiration claim in `jwt-token.json`:

jwt-token.json

```
{
  "iss":"airhacks",
  "jti":"airhacks-jwt-unique-id-12342142",
  "sub":"user/43971",
  "exp":1579412832, ①
  "upn":"demo@acme.org",
  "myc":"My Custom Claim",
  "groups":[
    "user",
    "admin"
  ]
}
```

① Add the expiration claim with the expiration date acquired from the prior step.

1. Re-run the jwtenizer to generate an updated token

Terminal

```
$ cd tutorial/jwt
$ ./jwtenizr.sh
```

NOTE While running `jwtenizr.sh` will not add the `exp` claim to `jwt-token.json`, copying the generated token from `token.jwt` and pasting it into jwt.io will show the token does include the `exp` claim.

1. Use the new token in requests to the `frontend`