Spring Cloud Stream Lab

Setting up a PAS environment - leveraging cloud services and app starters.

Setting up a local environment - manual mode

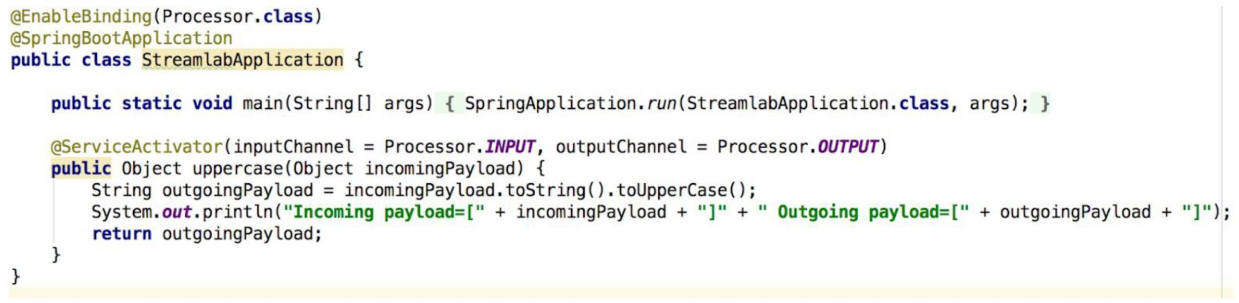
1. Install RabbitMQ following the instructions in Lab0 - Install RabbitMQ
2. Now let's setup the lab environment:
    a. Using git
        i. From a your local terminal or command prompt change directory to a clean working directory.
        ii. Now execute: git clone https://github.c/m/wxlund/DNDatafl/w
        iii. Now cd DNDataflow
    b. Using Thumb drive (Note: we haven't created a zip of the clone yet. Determine if lab will need this.
        i. Copy the DNDataflow directory from the thumb drive to a location on your laptop hard drive
        ii. Now from a terminal or command prompt cd to the DNDataflow directory you just created on your hard drive.

Creating your Processor

- CD into "DNDataflowolabs/lab1" folder
- First let's review the project in your favourite IDEotext-editor
- Start by opening the pom.xm and review the project dependencies
- You'll find the following dependency that brings the binder implementation we would like to use in this lab, which happens to be a RabbitMQ implementation in our case. *(You'll also find few other dependencies that are commented outª let's ignore them for now)*

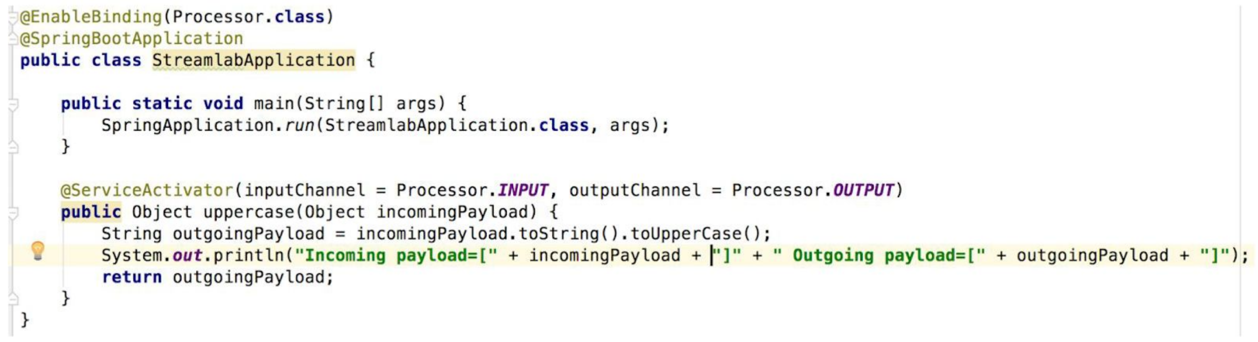
```
<dependency>
        <groupId> org.springframework.cloud </groupId>
        <artifactId>spring-cloud-starter-stream-rabbit</artifactId>
</dependency>
```

- open “ StreamlabApplication ” and add @EnableBinding(Processoroclass) annotation



```
@EnableBinding(Processor.class)
@SpringBootApplication
public class StreamlabApplication {

    public static void main(String[] args) { SpringApplication.run(StreamlabApplication.class, args); }

    @ServiceActivator(inputChannel = Processor.INPUT, outputChannel = Processor.OUTPUT)
    public Object uppercase(Object incomingPayload) {
        String outgoingPayload = incomingPayload.toString().toUpperCase();
        System.out.println("Incoming payload=[" + incomingPayload + "]" + " Outgoing payload=[" + outgoingPayload + "]");
        return outgoingPayload;
    }
}
```

- Uncomment the uppercase () business logic and review the INPUT and OUTPUT channel configurations activated via @ServiceActivator



```
@EnableBinding(Processor.class)
@SpringBootApplication
public class StreamlabApplication {

    public static void main(String[] args) {
        SpringApplication.run(StreamlabApplication.class, args);
    }

    @ServiceActivator(inputChannel = Processor.INPUT, outputChannel = Processor.OUTPUT)
    public Object uppercase(Object incomingPayload) {
        String outgoingPayload = incomingPayload.toString().toUpperCase();
        System.out.println("Incoming payload=[" + incomingPayload + "]" + " Outgoing payload=[" + outgoingPayload + "]");
        return outgoingPayload;
    }
}
```

- From the terminal or shell command-prompt, let’s build the application

```
mvnw clean package
```

- Run the application and verify it starts normally

```
java  -jar target/streamlab-0.0.1-SNAPSH/T.jar
```

- Uncomment Spring Boot’s Actuator dependency

```
<dependency>
        <groupId> org.springframework.boot</groupId>
        <artifactId> spring-boot-starter-actuator</artifactId>
</dependency>
```

- Rebuild and run the application to review the actuator endpoints; specifically, we would want to confirm whether the channel bindings are setup correctly

```
      java  -jar target/streamlab-0-0-1-SNAPSHOT.jar
```

- Launch “http://<HOST>:<PORT>/ configprops ” endpoint and verify the channel bindings loaded from the application.properties

  Example:

```
file http://localhost:9002/configprops
```

● Now, let’s build a streaming pipeline
(1) Source

This app is a simple spring boot application and is available from the app starters. For convenience a copy of the jar has been moved into the labs/jars directory. The app-starters are found at https://repo.spring.io/libs-release/org/springframework/cloud/stream/app/spring-cloud-stream-app-descriptor/Einstein.RELEASE/spring-cloud-stream-app-descriptor-Einstein.RELEASE.rabbit-apps-maven-repo-url.properties in our configuration of using rabbit.

```
java -jar labs/jars/http-source-rabbit-2.1.0.RELEASE.jar
-spring.cloud.stream.bindings.output.destination=streamlabdest1
--server.port=9001
```

(2) Processor

This is the app that you built, which is why the jar is found from the labs /target directory

```
java -jar labs/lab1/target/streamlab-0.0.1-SNAPSHOT.jar
--spring.cloud.stream.bindings.input.destination=streamlabdest1
--spring.cloud.stream.bindings.output.destination=streamlabdest2
--server.port =9002
```

(3) Sink

This app is a simple spring boot application and is available from the app starters. See the note from (1) Source.

```
java -jar labs/jars/log-sink-rabbit-2.1.0.RELEASE.jar
--spring.cloud.stream.bindings.input.destination=streamlabdest2
--server.port=9003
```

- Monitor the logs of all the 3 applications in the terminal window where the app was launched.
- Post data to the http://localhost endpoint at port=9001

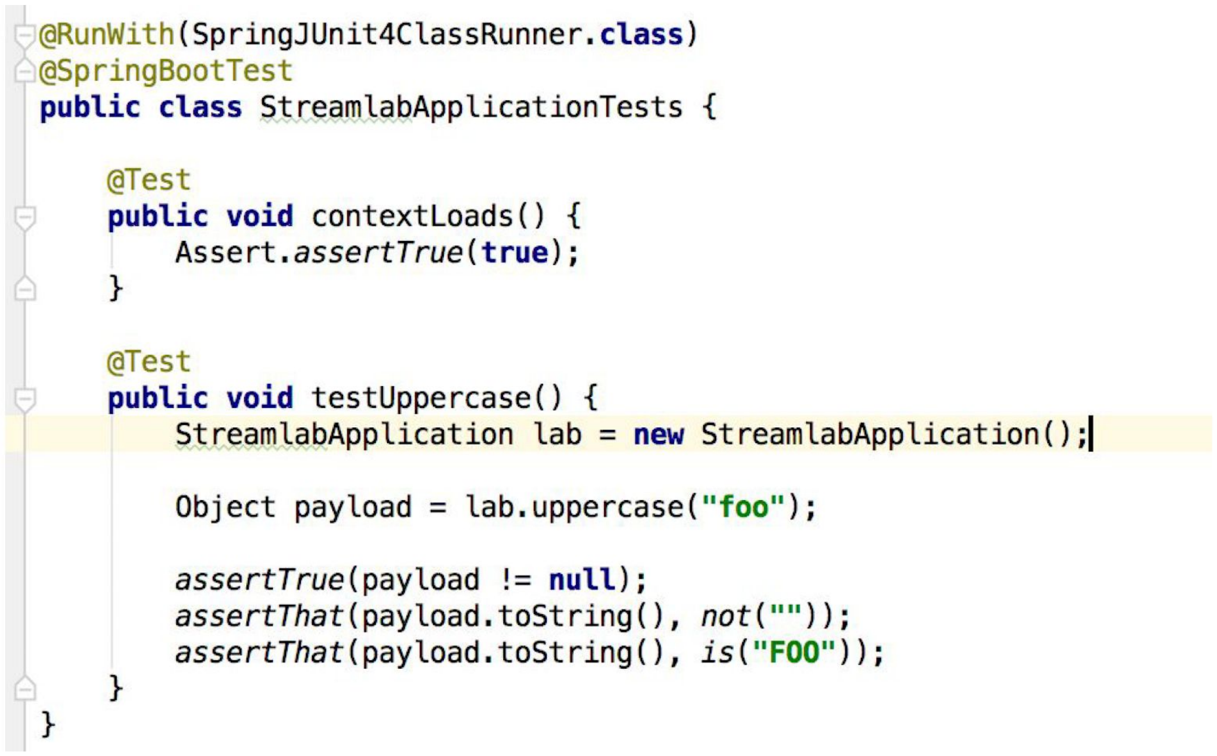
```
curl  -target https://localhost:9001 -H “Content-Type:text/plain”  -d
“hello world”
```

● Review the the logs in the log-sink console (terminal)

Testing your Processor

Both unit and integration tests are included in the `/lab1` project

- Unit Test



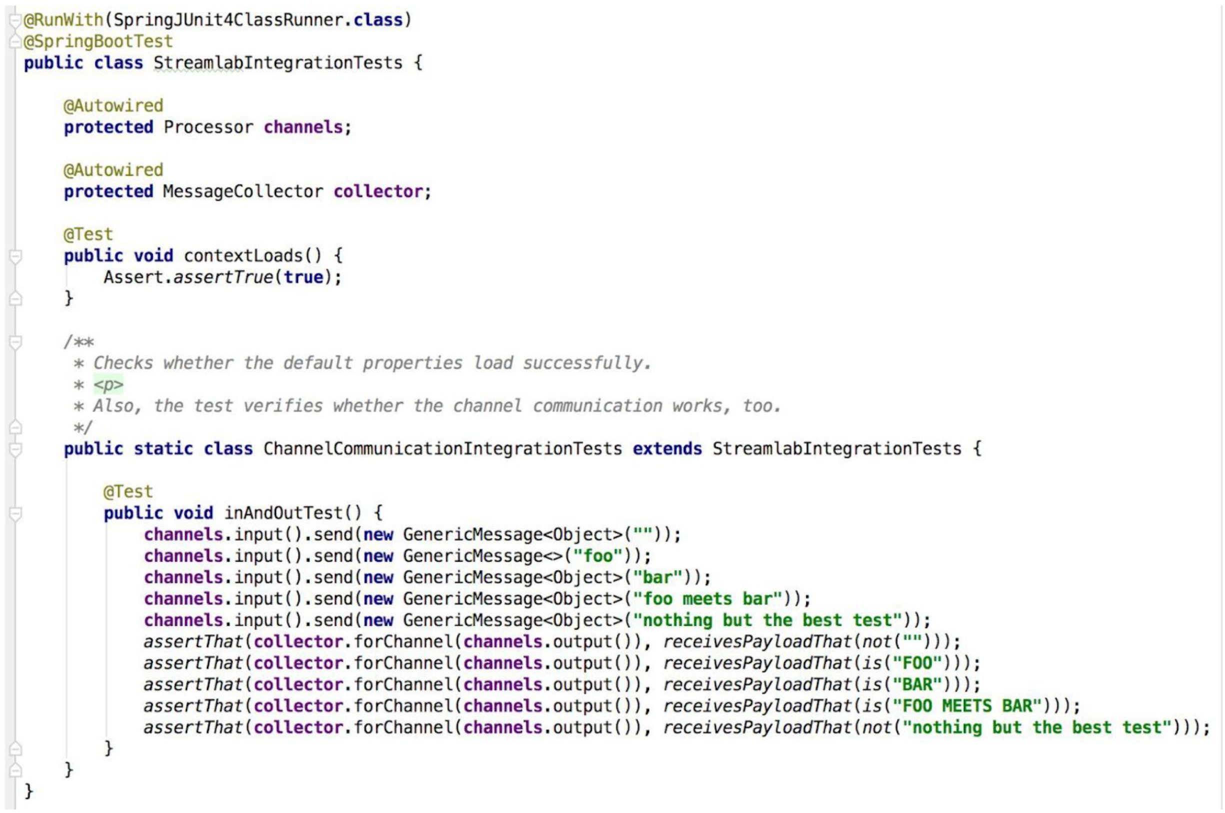
```
@RunWith(SpringJUnit4ClassRunner.class)
@SpringBootTest
public class StreamlabApplicationTests {

    @Test
    public void contextLoads() {
        Assert.assertTrue(true);
    }

    @Test
    public void testUppercase() {
        StreamlabApplication lab = new StreamlabApplication();

        Object payload = lab.uppercase("foo");

        assertTrue(payload != null);
        assertThat(payload.toString(), not(""));
        assertThat(payload.toString(), is("FOO"));
    }
}
```

  - Uncomment “ StreamlabApplicationTests ”
  - Change the expectations and let the test fail
  - Run the tests
  - Verify the assertions fail

    - Revert to original state
    - Run the tests
    - Verify the assertions work correctly
- Integration Test
    - Uncomment "spring-cloud-stream-test-support" from pom.xml

```
<!--<dependency>-->
        <!--<groupId>org.springframework.cloud</groupId>-->
        <!--<artifactId>spring-cloud-stream-test-support</artifactId>-->
<!--</dependency>-->
```

    - Re-import/refresh maven dependencies at the project level
    - Uncomment "StreamlabIntegrationTests"



```java
@RunWith(SpringJUnit4ClassRunner.class)
@SpringBootTest
public class StreamlabIntegrationTests {

    @Autowired
    protected Processor channels;

    @Autowired
    protected MessageCollector collector;

    @Test
    public void contextLoads() {
        Assert.assertTrue(true);
    }

    /**
     * Checks whether the default properties load successfully.
     * <p>
     * Also, the test verifies whether the channel communication works, too.
     */
    public static class ChannelCommunicationIntegrationTests extends StreamlabIntegrationTests {

        @Test
        public void inAndOutTest() {
            channels.input().send(new GenericMessage<Object>(""));
            channels.input().send(new GenericMessage<>("foo"));
            channels.input().send(new GenericMessage<Object>("bar"));
            channels.input().send(new GenericMessage<Object>("foo meets bar"));
            channels.input().send(new GenericMessage<Object>("nothing but the best test"));
            assertThat(collector.forChannel(channels.output()), receivesPayloadThat(not("")));
            assertThat(collector.forChannel(channels.output()), receivesPayloadThat(is("FOO")));
            assertThat(collector.forChannel(channels.output()), receivesPayloadThat(is("BAR")));
            assertThat(collector.forChannel(channels.output()), receivesPayloadThat(is("FOO MEETS BAR")));
            assertThat(collector.forChannel(channels.output()), receivesPayloadThat(not("nothing but the best test")));
        }
    }
}
```

    - Change the expectations and let the test fail
    - Run the tests
    - Verify the assertions fail
    - Revert to original state
    - Run the tests
    - Verify the assertions work correctly

Appendix

For the curious, who would like to start from Spring Initializr experience, please select the dependencies as listed below and then generate a new project.