

Spring Cloud Stream Lab

Setting up the environment

1. Install RabbitMQ following the instructions in **Lab 0 - 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.com/cppwfs/DNDataflow.git`
 - iii. Now `cd DNDataflow`
 - b. Using Thumbdrive
 - i. Copy the **DNDataflow** directory from the thumbdrive 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 "DNDataflow/labs/lab1 " folder
- First let's review the project in your favourite IDE/text-editor
- Start with the `pom.xml` 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 out; 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(Processor.class)` annotation
- 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/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-SNAPSHOT.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 "/configprops" endpoint and verify the channel bindings loaded from the application.properties file

<http://localhost:9002/configprops>

```
- spring.cloud.stream-org.springframework.cloud.stream.config.BindingServiceProperties: {
  prefix: "spring.cloud.stream",
  - properties: {
    - bindings: {
      - input: {
        destination: "streamlabdest1"
      },
      - output: {
        destination: "streamlabdest2"
      }
    }
  }
},
```

- Now, let's build a streaming pipeline

(1) Source

```
java -jar labs/jars/http-source-rabbit-1.1.2.RELEASE.jar
--spring.cloud.stream.bindings.output.destination= streamlabdest1
--server.port=9001
```

(2) Processor

```
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

```
java -jar labs/jars/log-sink-rabbit-1.1.1.RELEASE.jar
--spring.cloud.stream.bindings.input.destination= streamlabdest2
--server.port=9003
```

- Monitor the logs of all the 3 applications in different terminal windows
- Post data to the http://localhost endpoint at port=9001

```
curl -target http://localhost:9001 -H "Content-Type:text/plain" -d
"hello world"
```

- Review the the logs in the log-sink console

```

2017-02-05 11:06:31.570 INFO 37295 --- [main] o.s.t.endpoint.EventDrivenConsumer : started inbound.streamlabdest2.default
2017-02-05 11:06:31.571 INFO 37295 --- [main] o.s.c.support.DefaultLifecycleProcessor : Starting beans in phase 2147483647
2017-02-05 11:06:31.635 INFO 37295 --- [main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on port(s): 9003 (http)
2017-02-05 11:06:31.638 INFO 37295 --- [main] o.s.c.s.a.l.s.r.LogSinkRabbitApplication : Started LogSinkRabbitApplication in 5.5
72 seconds (JVM running for 6.0)
2017-02-05 11:06:33.859 INFO 37295 --- [giesFHSRAwrrQ-1] log-sink : HELLO WORLD

```

Testing your Processor Code

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

- Unit Test
 - Uncomment "StreamlabApplicationTests"

```

@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);
        assertEquals(payload.toString(), not(""));
        assertEquals(payload.toString(), is("FOO"));
    }
}

```

- 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 ”

```
@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

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.

SPRING INITIALIZR bootstrap your application now

Generate a Maven Project with Spring Boot 1.5.1

Project Metadata

Artifact coordinates

Group

io.spring

Artifact

streamlab

Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

Web, Security, JPA, Actuator, Devtools...

Selected Dependencies

Stream Rabbit

Actuator

Generate Project