Are your Apache Camel Routes Ready for Production?: How to test Camel applications

May 15, 2012

FuseSource
integration everywhere

About Me And This Presentation

- David Valeri
  - 2.5 years with FuseSource
  - 8 years in the SOA, Java, integration space
  - Apache Camel and Apache CXF committer
- Blog – http://davidvaleri.wordpress.com/
- Twitter – @DavidValeri
- Email – david@fusesource.com
- FuseSource – http://fusesource.com

FuseSource
integration everywhere

What is Apache Camel

- Apache Camel is an extensible framework for coordinating common enterprise integration patterns and communication protocols for solving business requirements.
  - Patterns: Poller, event driven consumer, content based routing, splitter/aggregator
    - Over **45** patterns out-of-the-box
  - Protocols and Libraries: FTP, JMS, JDBC, SMTP, Spring, CXF, Drools, etc.
    - Over **100** components out-of-the-box
- Simplifies integration
  - Abstracts the complexity of working with raw protocols and libraries
  - Provides a uniform language for describing integration logic

FuseSource
integration everywhere

Understanding Apache Camel

FuseSource
integration everywhere

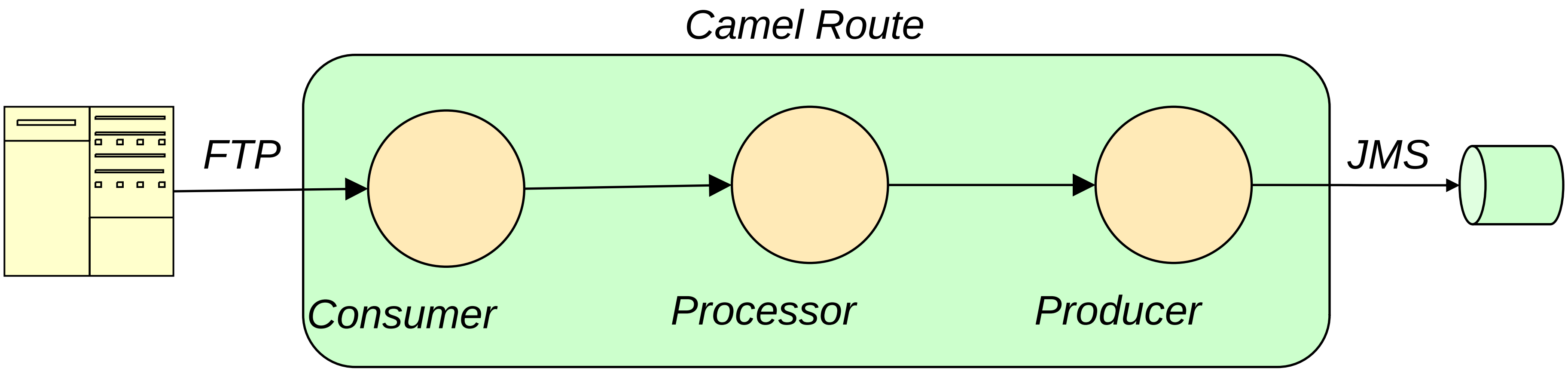
Camel Routes: The core building block

- A route is the step-by-step processing of a message:
  - From a consumer endpoint, which listens for the incoming message…
  - Through zero or more processors, which apply enterprise integration patterns / custom processing code / interceptor patterns / more
  - To zero or more producer endpoints, which send outgoing messages
- A route can be defined in Java or XML
  - Camel's Domain Specific Language is implemented in Java and you can program to the DSL in Java, XML, or Scala.

FuseSource
integration everywhere

An Example Route

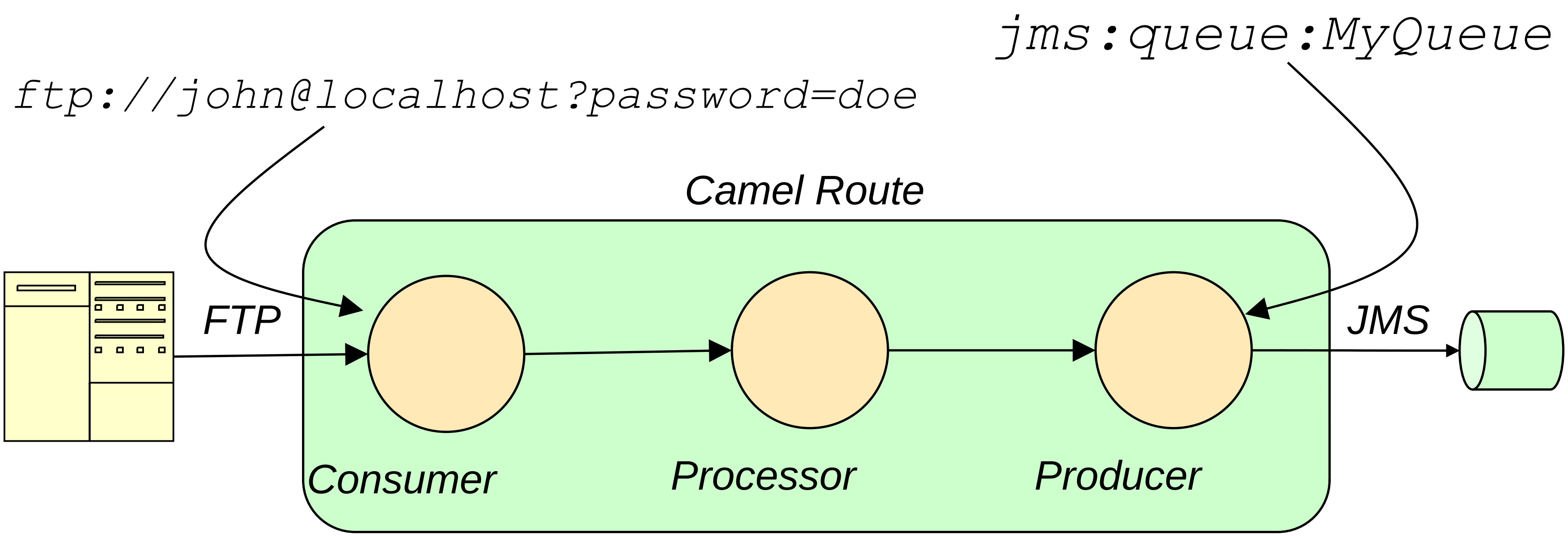
- A simple scenario:
  - <u>Consume</u> an XML file from an FTP server
  - <u>Process</u> it using XSLT
  - <u>Produce</u> a JMS message and place on a queue



FuseSource
integration everywhere

An Example Route: Endpoints and URIs

- An endpoint is something that can create or receive messages
  - Examples: an FTP server, a Web Service, or a JMS broker
- Camel allows you to define endpoints using simple URIs

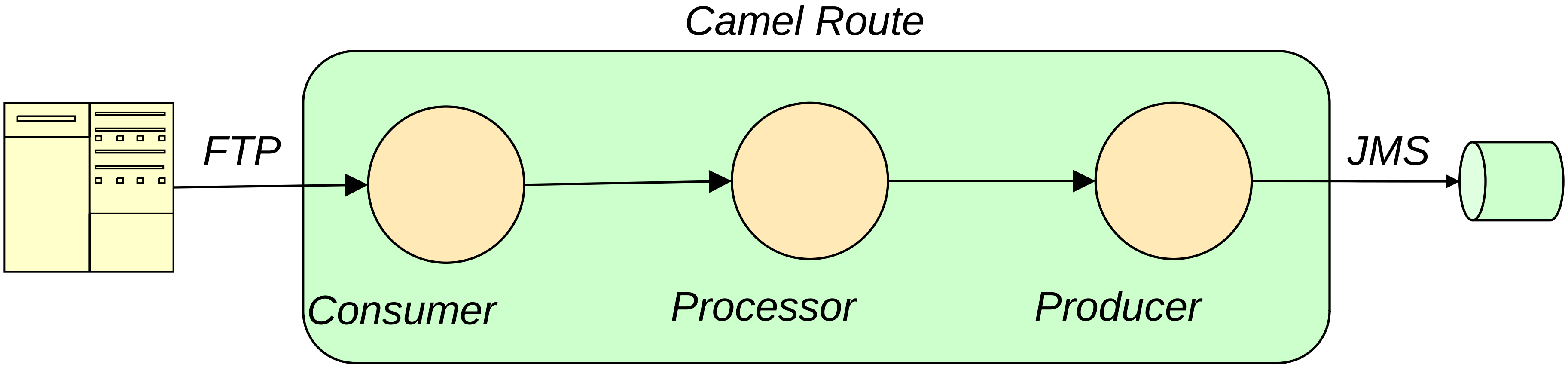


FuseSource
integration everywhere

An Example Route: Java DSL

- A Java DSL based route definition

```
from("ftp://john@localhost?password=doe")
  .to("xslt:MyTransform.xslt")
  .to(“jms:queue:MyQueue");
```



FuseSource integration everywhere

How do these concepts relate to testing?

- Processors/Endpoints
  - We need to simulate business logic and integration with external systems
  - We need to capture inputs/outputs for comparison with expected state
- Error handling
  - We need to simulate error conditions to test transactional and compensating behavior
  - We need to ensure that retry logic is working properly

Testing Camel Routes

FuseSource
integration everywhere

Integration with popular testing frameworks

- Common test initialization and cleanup
  - Create a CamelContext
  - Prepare any resources that the Camel route(s) will need
  - Load some route(s) into the CamelContext
  - Setup a means to send messages to the route(s)
- Integration with:
  - JUnit3
    - org.apache.camel.test.CamelTestSupport
  - JUnit4
    - org.apache.camel.test.junit4.CamelTestSupport
  - TestNG
    - org.apache.camel.testng.CamelTestSupport
  - Spring Test

FuseSource
integration everywhere

Your first unit test with Apache Camel

```
…
import org.apache.camel.test.junit4.CamelTestSupport;

public class ExampleTest extends CamelTestSupport {

  @Override
  protected RouteBuilder
        createRouteBuilder() throws
        Exception {
     …
  }

  @Test
  public void testMethod() {
     …
  }
}
```

FuseSource
integration everywhere

Approaches to testing a Camel route

- How do we test a Camel route?
  - Producer templates – Send messages into a Camel route
  - Mock endpoints – Capture messages and validate assertions against those messages
  - AdviceWith – Intercept messages in a route and perform some action
  - Embedded ActiveMQ – Run an in-memory instance of ActiveMQ solely for the test
  - Live integration test environment – Sometimes you just have to interact with an external system

FuseSource
integration everywhere

Producer Template

- Can send arbitrary messages to a Camel endpoint
- Use *ProducerTemplate* instances to drive a test case by triggering your route(s)
  - Can send a body and headers with the message exchange pattern of your choosing
  - Can trigger routes synchronously and asynchronously
- Easily retrieved within a *CamelTestSupport* based test case
  - Inject using Camel annotations
  - Retrieve from the Camel context
  - Use the default instance created for you by Camel test support

FuseSource
integration everywhere

Producer Template – Inject

- Inject using Camel annotations
  - Declare a member variable of type *ProducerTemplate* in your test class
  - Annotate with the *Produce* annotation
  - Use the producer in your test

```
@Produce
private ProducerTemplate producer;

...

producer.sendBody("activemq:myQueue", "Hello");
```


FuseSource
integration everywhere

Producer Template – Retrieve

- Retrieve from the Camel context
  - Declare a member variable of type *ProducerTemplate* in your test class
  - Implement a setup method that initializes the variable
  - Use the producer in your test

```
private ProducerTemplate producer;
...
@Before
public void setup() throws Exception {
    producer = context.createProducerTemplate();
    producer.start()
}
...
producer.sendBody("activemq:myQueue", "Hello");
```

FuseSource
integration everywhere

Producer Template – Default

- Use the default instance created for you by Camel test support
  - Retrieve the default instance by using the *template()* method from *CamelTestSupport*
  - Use the producer in your test

```
template().sendBody("activemq:myQueue", "Hello");
```


FuseSource
integration everywhere

Mock Endpoints

- Can assert that a set of expected conditions are met
- Can provide arbitrary responses to incoming requests
- Use *MockEndpoint* to assert the correctness of your routing logic
  - Listen for, collect, and validate the output of a route or a step within a route
  - Optionally produce a mock response
- Easily retrieved within a *CamelTestSupport* based test case
  - Inject using Camel annotations
  - Retrieve from the Camel context

FuseSource
integration everywhere

Mock Endpoint – Inject

- Inject using Camel annotations
  - Declare a member variable of type *MockEndpoint* in your test class
  - Annotate with the *EndpointInject* annotation
  - Use the endpoint in your test

```
@EndpointInject(uri = "mock:output")
private MockEndpoint output;
…
output.setExpectedMessageCount(10);
output.expectedBodiesReceived(…);
output.message(1).header("myHeader").isEqualTo(
        "myHeaderValue");
…
output.assertIsSatisfied();
```


FuseSource
integration everywhere

Mock Endpoint – Retrieve

- Retrieve from the Camel context
  - Declare a member variable of type *MockEndpoint* in your test class
  - Implement a setup method that initializes the variable
  - Use the endpoint in your test

```
private MockEndpoint output;
...
@Before
public void setup() throws Exception {
    output = resolveMandatoryEndpoint("mock:output",
            MockEndpoint.class);
}
...
```

FuseSource
integration everywhere

Demonstration 1 – Using Mock Endpoints and Producer Templates

FuseSource
integration everywhere

Helpful hints for success with ProducerTemplate and MockEndpoint

- Remember that a *MockEndpoint* is only a mock
  - You still have not tested that the endpoint you are mocking actually works (this is a case where you need integration testing)
- Sometimes it is easier to simply collect messages with the *MockEndpoint* and validate them yourself
  - `output.getReceivedExchanges()` returns all of the messages that the endpoint has collected
- Remember that *MockEndpoint* is stateful
  - If you use them in more than one scenario per test method, you need to reset them by calling `resetMocks()`
- *MockEndpoint* will block while waiting to be satisfied
  - *You can change the timeout period if you need to*

FuseSource
integration everywhere

Helpful hints for success with ProducerTemplate and MockEndpoint

- Design your routes with the use of *MockEndpoint* in mind
  - *Allow producer endpoints to be easily altered or substituted from outside of the route builder to enable the selective use of mock endpoints in tests*

```
…
from("direct:persistRecord")
   .routeId(PERSIST_RECORD_ROUTE_ID)
   .transacted("JDBC_PROPAGATION_REQUIRES_NEW")
   .to(getPersistEndpointUri());
…
protected String getPersistEndpointUri() {
   if (alternatePersistEndpointUri != null)
   {
      return alternatePersistEndpointUri;
   } else {
      return "ibatis:example.insertRecord?statementType=Insert";
   }
}
```


FuseSource integration everywhere

AdviceWith

- AdviceWith allows the mutation and decoration of a Camel route without altering the code that defines the route
- Much like Aspect Oriented Programming
  - Add advice to an endpoint/processor
- Also has some more powerful capabilities to alter the route
  - Remove, replace, and add endpoint/processor
- Testing primarily uses a subset of these capabilities

FuseSource
integration everywhere

AdviceWith – Replacing a consumer endpoint

- How do you test an externally triggered route such as an HTTP consumer, polling consumer, or other endpoint that you do not wish to actually instantiate in a test?

- Example polling route

```
from("timer://poll?delay=10000")
    .routeId("pollingRoute")
    .to("ibatis:example.query?statementType=QueryForList")
    …
```

- Replace the from endpoint with an alternate endpoint and then trigger the route with a *ProducerTemplate*

FuseSource
integration everywhere

AdviceWith – Replacing a consumer endpoint

- Before executing the test logic, alter the route under test

```
RouteDefinition routeDef = context
        .getRouteDefinition("pollingRoute");

routeDef.adviceWith(context, new AdviceWithRouteBuilder() {
    @Override
    public void configure() throws Exception {
        replaceFromWith("direct:input");
    }
});
```

- Drive the test using a *ProducerTemplate*

```
template().sendBody("direct:input", null);
```


FuseSource
integration everywhere

AdviceWith – Intercepting messages

- How do you simulate responses from remote systems, intermittent errors, and other behaviors?
- Example transaction route

```
from("direct:persistRecord")
   .routeId("persistRecord")
   .transacted("JDBC_PROPAGATION_REQUIRES_NEW")
   .onException(IOException.class)
      .maximumRedeliveries(2)
      .redeliveryDelay(1000l)
      .logRetryAttempted(true)
      .logRetryStackTrace(true)
      .retryAttemptedLogLevel(LoggingLevel.WARN)
   .end()
   .to("ibatis:example.insertRecord?statementType=Insert");
```

- Add advice to the iBatis endpoint to simulate various error conditions

FuseSource
integration everywhere

AdviceWith – Intercepting messages

- Before executing the test logic, alter the route under test

```
RouteDefinition routeDef = context
      .getRouteDefinition("persistRecord");

routeDef.adviceWith(context, new AdviceWithRouteBuilder() {

   private AtomicInteger count = new AtomicInteger(0);

   @Override
   public void configure() throws Exception {
      interceptSendToEndpoint(
            "ibatis:example.insertRecord?statementType=Insert").process(
         new Processor() {
            @Override
            public void process(Exchange exchange)
               throws Exception {
               if (count.getAndIncrement() < 2) {
                  throw new IOException(
                     "Simulated JDBC Error!");
               }
            }
         });
   }
});
```


FuseSource
integration everywhere

Demonstration 2 – Using AdviceWith

FuseSource
integration everywhere

Embedded ActiveMQ

- Apache ActiveMQ is a highly scalable and flexible message broker
- JMS is a common theme with integration projects

```
from("activemq:myQueue")
    .to("bean:myBean?method=process")
    .to("activemq:myOtherQueue");
```

- You can either unit and integration test, or you can use an embedded broker and only test once

FuseSource
integration everywhere

Embedded ActiveMQ – Starting an embedded broker

- Start an embedded ActiveMQ broker and configure Camel to connect to it

```
@Override
protected CamelContext createCamelContext() throws Exception {
   CamelContext newContext = super.createCamelContext();

   // Start the embedded broker.
   broker = BrokerFactory.createBroker("broker:(vm://localhost)?persistent=false");
   broker.start();

   // Simple AMQ component configuration using defaults for all settings
   // and using the VM transport to connect to the embedded AMQ broker.
   ActiveMQComponent amqComponent =
         newContext.getComponent("activemq", ActiveMQComponent.class);
   amqComponent.setBrokerURL("vm://localhost?waitForStart=20000&create=false");

   return newContext;
}
```


FuseSource integration everywhere

Embedded ActiveMQ – Using helper routes

- Use a helper route to forward messages from the output queue of the route under test to a *MockEndpoint*

```
RouteBuilder testHelperRouteBuilder = new RouteBuilder() {
    @Override
    public void configure() throws Exception {
        from("activemq:myOtherQueue")
            .routeId(Demonstration3.class.getName() + ".helper")
            .to("mock:output");
    }
};
```


FuseSource
integration everywhere

Demonstration 3 – Using embedded ActvieMQ and helper routes

FuseSource
integration everywhere

Helpful hints for success with AdviceWith, ActiveMQ, and helper routes

- Name all of your routes
- Use AdviceWith's capabilities judiciously
  - replaceFromWith
  - interceptSendtoEndpoint
    - skipSendToOriginalEndpoint
- Consider using an embedded ActiveMQ broker for "unit" testing JMS based route
- Helper routes and *ProducerTemplate* enable you to unit/integration test Camel routes with external event triggers
  - HTTP, JMS, etc.

FuseSource
integration everywhere

Further Reading

- Camel Test Component - http://camel.apache.org/test.html
  - Test data from external resources
- Camel DataSet Component - http://camel.apache.org/dataset.html
  - Load / Soak testing
- Notify Builder - http://camel.apache.org/notifybuilder.html
  - An alternative or supplement to mocks and AdviceWith
- *isMockEndpoints*
  - Decorates endpoints/processors in a route with *MockEndpoint*
- Spring Test and *CamelSpringTestSupport*

FuseSource
integration everywhere

Q&A

FuseSource
integration everywhere