Alfresco integrations

With

Spring-boot and Apache Camel

The aim of this blog is to introduce you to Enterprise Integration Patterns and to show you how to create an application to integrate Alfresco with another application…in this case we will be sending documents on request from Alfresco to Box. We will store both the content and the metadata in Box.

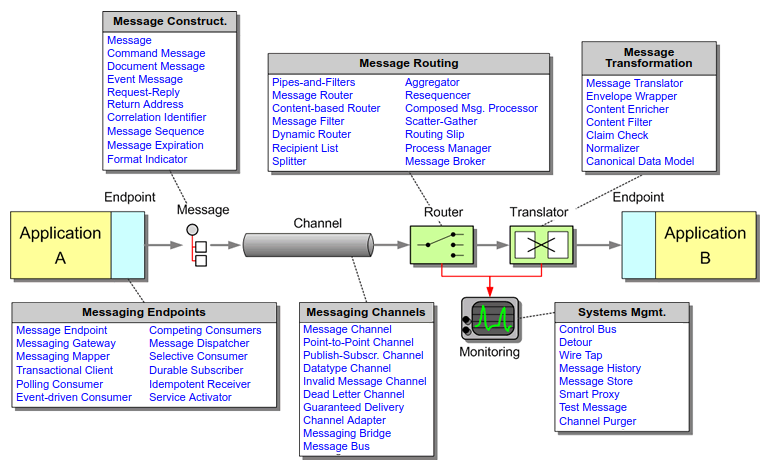
# Enterprise Integration Patterns

EIP (Enterprise Integration Patters) defines a language consisting of 65 integration patterns (<http://www.enterpriseintegrationpatterns.com/patterns/messaging/toc.html>) to establish a technology-independent vocabulary and a visual notation to design and document integration solutions.

Why EIP? Today's applications rarely live in isolation. Architecting integration solutions is a complex task.

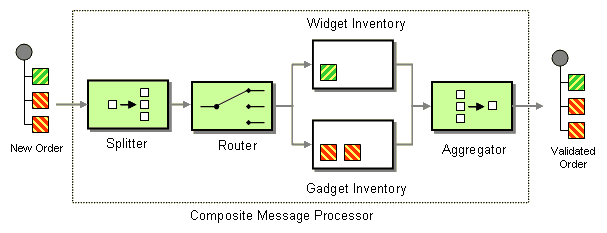
The lack of a common vocabulary and body of knowledge for asynchronous messaging architectures make it difficult to avoid common pitfalls.

For example the following diagram shows how content from one application is routed and transformed to be delivered to another application. Each step can be further detailed with specific annotations.



* **Channel Patterns** describe how messages are transported across a Message Channel. These patterns are implemented by most commercial and open source messaging systems.
* **Message Construction Patterns** describe the intent, form and content of the messages that travel across the messaging system.
* **Routing Patterns** discuss how messages are routed from a sender to the correct receiver. Message routing patterns consume a message from one channel and republish it message, usually without modification, to another channel based on a set of conditions.
* **Transformation Patterns** change the content of a message, for example to accommodate different data formats used by the sending and the receiving system. Data may have to be added, taken away or existing data may have to be rearranged.
* **Endpoint Patterns** describe how messaging system clients produce or consume messages.
* **System Management Patterns** describe the tools to keep a complex message-based system running, including dealing with error conditions, performance bottlenecks and changes in the participating systems.

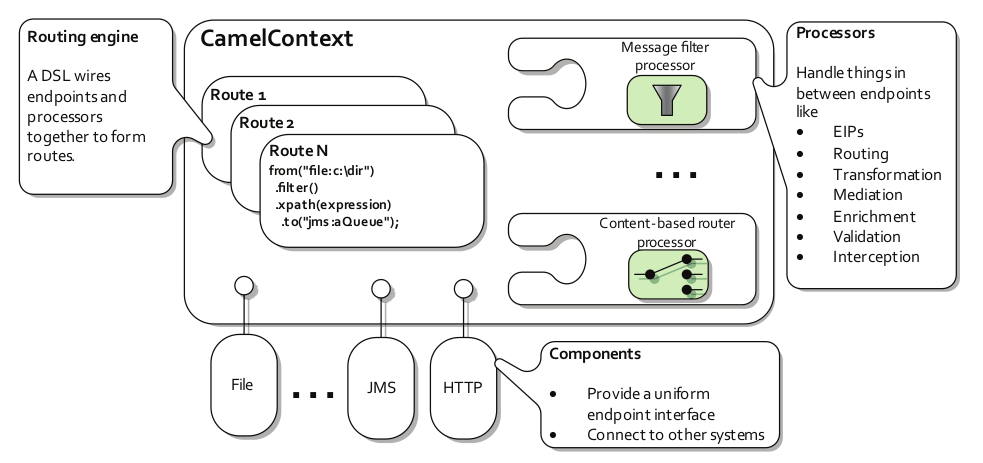
The following example shows how to maintain the overall message flow when processing a message consisting of multiple elements, each of which may require different processing.



# Apache Camel

Apache Camel (<http://camel.apache.org/>) is an integration framework whose main goal is to make integration easier. It implements many of the EIP patterns and allows you to focus on solving business problems, freeing you from the burden of plumbing.

At a high level, Camel is composed of components, routes and processors. All of these are contained within the CamelContext .



The CamelContext provides access to many useful services, the most notable being components, type converters, a registry, endpoints, routes, data formats, and languages.

|  |  |
| --- | --- |
| **Service** | **Description** |
| Components | A Component is essentially a factory of Endpoint instances. To date, there are over 80 components in the Camel ecosystem that range in function from data transports, to DSL s, data formats, and so on i.e. cmis, http, box, salesforce, ftp, smtp, etc |
| Endpoints | An endpoint is the Camel abstraction that models the end of a channel through which a system can send or receive messages. Endpoints are usually created by a Component and Endpoints are usually referred to in the DSL via their URIs i.e. cmis://cmisServerUrl[?options] |
| Routes | The steps taken to send a message from one end point to another end point. |
| Type Converters | Camel provides a built-in type-converter system that automatically converts between well-known types. This system allows Camel components to easily work together without having type mismatches. |
| Data Formats | Allow messages to be marshalled to and from binary or text formats to support a kind of Message Translator i.e. gzip, json, csv, crypto, etc |
| Registry | Contains a registry that allows you to look up beans i.e. use a bean that defines the jdbc data source |
| Languages | To wire processors and endpoints together to form routes, Camel defines a DSL. DSL include among other Java, Groovy, Scala, Spring XML. |

# Building an Integration Application

The aim of the application is to send documents on request from Alfresco to Box. We will store both the content and the metadata in Box.

To build an EIP Application we are going to use:

* Maven to build the application
* Spring-boot to run the application
* Apache Camel to integrate Alfresco and Box

The full source code is available on GitHub: <https://github.com/miguel-rodriguez/Alfresco-Camel>

## Maven

Apache Maven (<https://maven.apache.org/>) is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.

## Maven Pom.xml

For our project the pom.xml file looks like this:

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>support.alfresco</groupId>

<artifactId>camel</artifactId>

<name>Spring Boot + Camel</name>

<version>0.0.1-SNAPSHOT</version>

<description>Project Example.</description>

<!-- Using Spring-boot 1.4.3 -->

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.4.3.RELEASE</version>

</parent>

<!-- Using Camel version 2.18.1 -->

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<camel-version>2.18.1</camel-version>

<app.version>1.0-SNAPSHOT</app.version>

</properties>

<!-- Spring -->

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- The Core Camel Java DSL based router -->

<dependency>

<groupId>org.apache.camel</groupId>

<artifactId>camel-core</artifactId>

<version>${camel-version}</version>

</dependency>

<!-- Camel Spring support -->

<dependency>

<groupId>org.apache.camel</groupId>

<artifactId>camel-spring</artifactId>

<version>${camel-version}</version>

</dependency>

<!-- Camel Metrics based monitoring component -->

<dependency>

<groupId>org.apache.camel</groupId>

<artifactId>camel-metrics</artifactId>

<version>${camel-version}</version>

</dependency>

<!-- Camel JMS support -->

<dependency>

<groupId>org.apache.camel</groupId>

<artifactId>camel-jms</artifactId>

<version>${camel-version}</version>

</dependency>

<!-- ActiveMQ component for Camel -->

<dependency>

<groupId>org.apache.activemq</groupId>

<artifactId>activemq-camel</artifactId>

</dependency>

<!-- Camel CMIS which is based on Apache Chemistry support -->

<dependency>

<groupId>org.apache.camel</groupId>

<artifactId>camel-cmis</artifactId>

<version>2.14.1</version>

</dependency>

<!-- Camel Stream (System.in, System.out, System.err) support -->

<dependency>

<groupId>org.apache.camel</groupId>

<artifactId>camel-stream</artifactId>

<version>${camel-version}</version>

</dependency>

<!-- Camel JSON Path Language -->

<dependency>

<groupId>org.apache.camel</groupId>

<artifactId>camel-jsonpath</artifactId>

<version>${camel-version}</version>

</dependency>

<!-- Apache HttpComponents HttpClient - MIME coded entities -->

<dependency>

<groupId>org.apache.httpcomponents</groupId>

<artifactId>httpmime</artifactId>

</dependency>

<!-- Camel HTTP (Apache HttpClient 4.x) support -->

<dependency>

<groupId>org.apache.camel</groupId>

<artifactId>camel-http4</artifactId>

<version>${camel-version}</version>

</dependency>

<!-- Camel SQL support -->

<dependency>

<groupId>org.apache.camel</groupId>

<artifactId>camel-sql</artifactId>

<version>${camel-version}</version>

</dependency>

<!-- Camel Zip file support -->

<dependency>

<groupId>org.apache.camel</groupId>

<artifactId>camel-zipfile</artifactId>

<version>${camel-version}</version>

</dependency>

<!-- Support for PostgreSQL database -->

<dependency>

<groupId>org.postgresql</groupId>

<artifactId>postgresql</artifactId>

<exclusions>

<exclusion>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-simple</artifactId>

</exclusion>

</exclusions>

</dependency>

<!-- Camel Component for Box.com -->

<dependency>

<groupId>org.apache.camel</groupId>

<artifactId>camel-box</artifactId>

<version>${camel-version}</version>

</dependency>

<!-- Camel script support -->

<dependency>

<groupId>org.apache.camel</groupId>

<artifactId>camel-script</artifactId>

<version>${camel-version}</version>

</dependency>

<!-- A simple Java toolkit for JSON -->

<dependency>

<groupId>com.googlecode.json-simple</groupId>

<artifactId>json-simple</artifactId>

<version>1.1.1</version>

<!--$NO-MVN-MAN-VER$-->

</dependency>

<!-- XStream is a Data Format which to marshal and unmarshal Java objects to and from XML -->

<dependency>

<groupId>org.apache.camel</groupId>

<artifactId>camel-xstream</artifactId>

<version>2.9.2</version>

</dependency>

<!-- Jackson XML is a Data Format to unmarshal an XML payload into Java objects or to marshal Java objects into an XML payload -->

<dependency>

<groupId>org.apache.camel</groupId>

<artifactId>camel-jackson</artifactId>

<version>2.9.2</version>

</dependency>

<!-- test -->

<dependency>

<groupId>org.apache.camel</groupId>

<artifactId>camel-test</artifactId>

<version>${camel-version}</version>

<scope>test</scope>

</dependency>

<!-- logging -->

<dependency>

<groupId>commons-logging</groupId>

<artifactId>commons-logging</artifactId>

<version>1.1.1</version>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-api</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-core</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-slf4j-impl</artifactId>

<scope>test</scope>

</dependency>

<!-- monitoring -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-remote-shell</artifactId>

</dependency>

<dependency>

<groupId>org.jolokia</groupId>

<artifactId>jolokia-core</artifactId>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

## Spring Boot

Spring Boot (<https://projects.spring.io/spring-boot/>) makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run". Most Spring Boot applications need very little Spring configuration.

Features

* Create stand-alone Spring applications
* Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)
* Provide opinionated 'starter' POMs to simplify your Maven configuration
* Automatically configure Spring whenever possible
* Provide production-ready features such as metrics, health checks and externalized configuration

### Spring Boot applicationContext.xml

The applicationContext.xml defines the java beans used by our application. Here we define the beans for connecting to Box, Database connectivity, ActiveMQ and Camel.

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans" xmlns:jdbc="http://www.springframework.org/schema/jdbc" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/jdbc http://www.springframework.org/schema/jdbc/spring-jdbc-3.0.xsd

http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-spring.xsd">

**<!-- Define configuration file application.properties -->**

<bean id="placeholder" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">

<property name="locations">

<list>

<value>classpath:application.properties</value>

</list>

</property>

<property name="ignoreResourceNotFound" value="false" />

<property name="searchSystemEnvironment" value="true" />

<property name="systemPropertiesModeName" value="SYSTEM\_PROPERTIES\_MODE\_OVERRIDE" />

</bean>

**<!-- Bean for Box authentication. Please note you need a Box developer account -->**

<bean id="box" class="org.apache.camel.component.box.BoxComponent">

<property name="configuration">

<bean class="org.apache.camel.component.box.BoxConfiguration">

<property name="userName" value="${box.userName}" />

<property name="userPassword" value="${box.userPassword}" />

<property name="clientId" value="${box.clientId}" />

<property name="clientSecret" value="${box.clientSecret}" />

</bean>

</property>

</bean>

**<!-- Define database connectivity -->**

<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">

<property name="driverClassName" value="org.postgresql.Driver" />

<property name="url" value="jdbc:postgresql://localhost:5432/alfresco" />

<property name="username" value="alfresco" />

<property name="password" value="admin" />

</bean>

**<!-- Configure the Camel SQL component to use the JDBC data source -->**

<bean id="sql" class="org.apache.camel.component.sql.SqlComponent">

<property name="dataSource" ref="dataSource" />

</bean>

**<!-- Create a connection to ActiveMQ -->**

<bean id="jmsConnectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory">

<property name="brokerURL" value="tcp://localhost:61616" />

</bean>

**<!-- Create Camel context -->**

<camelContext id="camelContext" xmlns="http://camel.apache.org/schema/spring" autoStartup="true">

<routeBuilder ref="myRouteBuilder" />

</camelContext>

**<!-- Bean defining Camel routes -->**

<bean id="myRouteBuilder" class="support.alfresco.Route" />

</beans>

### Application.java

The Application class is used to run our Spring application.

package support.alfresco;

import org.springframework.boot.SpringApplication;

import org.springframework.context.annotation.ImportResource;

@ImportResource("applicationContext.xml")

public class Application {

**public static void main(String[] args) {**

**SpringApplication.run(Application.class, args);**

**}**

}

### Route.java

In the Route.java file we define the Camel routes to send traffic from Alfresco to Box.

* Routes to Execute cmis query, download content and properties, compress it and upload it to Box

//////////////////////////////////////

// Download Alfresco documents //

//////////////////////////////////////

from("jms:alfresco.downloadNodes")

.log("Running query: ${body}")

.setHeader("CamelCMISRetrieveContent", constant(true))

.to(alfrescoSender + "&queryMode=true")

// Class FileContentProcessor is used to store the files in the filesystem together with the metadata

.process(new FileContentProcessor());

///////////////////////////////////////////////

// Move documents and metadata to Box //

//////////////////////////////////////////////

from("file:/tmp/downloads?antInclude=\*")

.marshal().zipFile()

.to("file:/tmp/box");

from("file:/tmp/metadata?antInclude=\*")

.marshal().zipFile()

.to("file:/tmp/box");

from("file:/tmp/box?noop=false&recursive=true&delete=true")

.to("box://files/uploadFile?inBody=fileUploadRequest");

Let’s break it down…

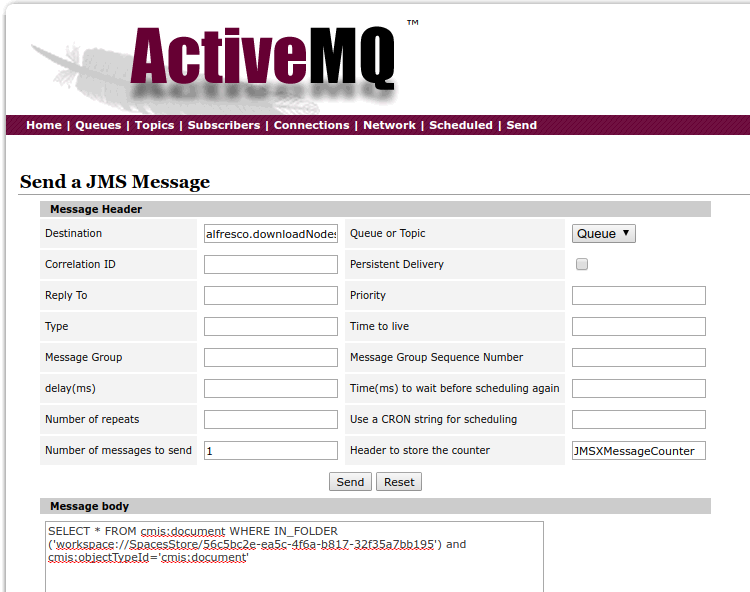
1. We read requests message with a CMIS query from an ActiveMQ queue

from("jms:alfresco.downloadNodes")

For example a CMIS query to get the nodes on a specific folder looks like…

SELECT \* FROM cmis:document WHERE IN\_FOLDER ('workspace://SpacesStore/56c5bc2e-ea5c-4f6a-b817-32f35a7bb195') and cmis:objectTypeId='cmis:document'

For testing purposes we can fire the message requests directly from the ActiveMQ admin UI (<http://127.0.0.1:8161/admin/>)



1. We send the CMIS query to Alfresco defined as “alfrescoSender”

.to(alfrescoSender + "&queryMode=true")

1. Alfresco sender is defined in application.properties as

alfresco.sender=cmis://http://alfresco:8080/alfresco/cmisatom?username=admin&password=admin

and mapped to “alfrescoSender” variable in Route.java

public static String alfrescoSender;

@Value("${alfresco.sender}")

public void setAlfrescoSender(String inSender) {

alfrescoSender = inSender;

}

1. Store the file in the filesystem using class FileContentProcessor for that job

.process(new FileContentProcessor());

1. Zip the content file and the metadata file

from("file:/tmp/downloads?antInclude=\*")

.marshal().zipFile()

.to("file:/tmp/box");

from("file:/tmp/metadata?antInclude=\*")

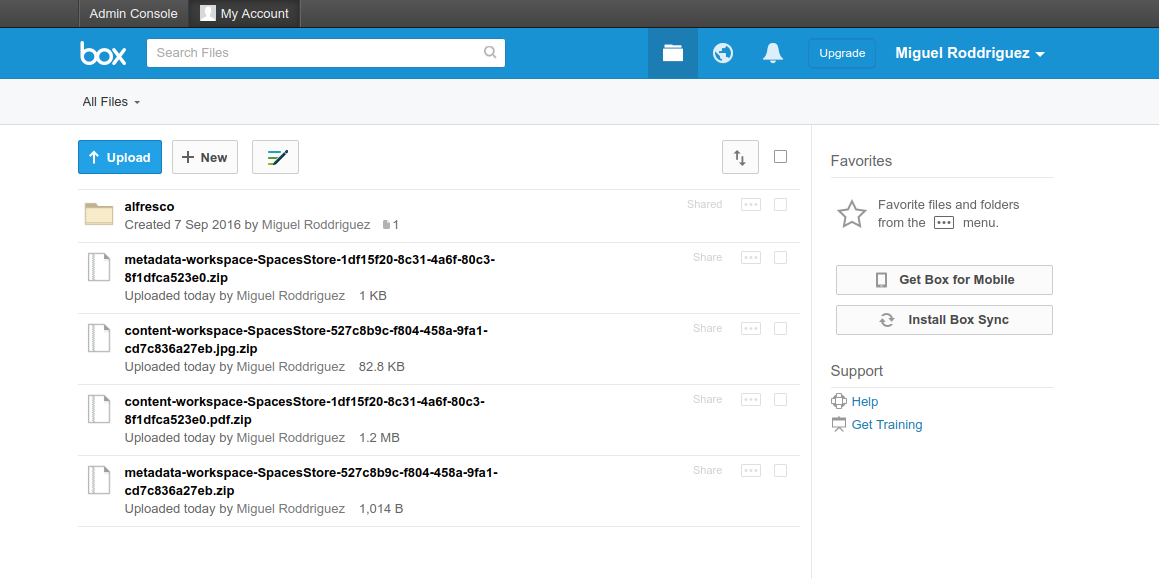
.marshal().zipFile()

.to("file:/tmp/box");

1. And finally upload the content to Box

from("file:/tmp/box?noop=false&recursive=true&delete=true")

.to("box://files/uploadFile?inBody=fileUploadRequest");



# Building and Running the application

To build the application using maven execute the following command:

mvn clean install

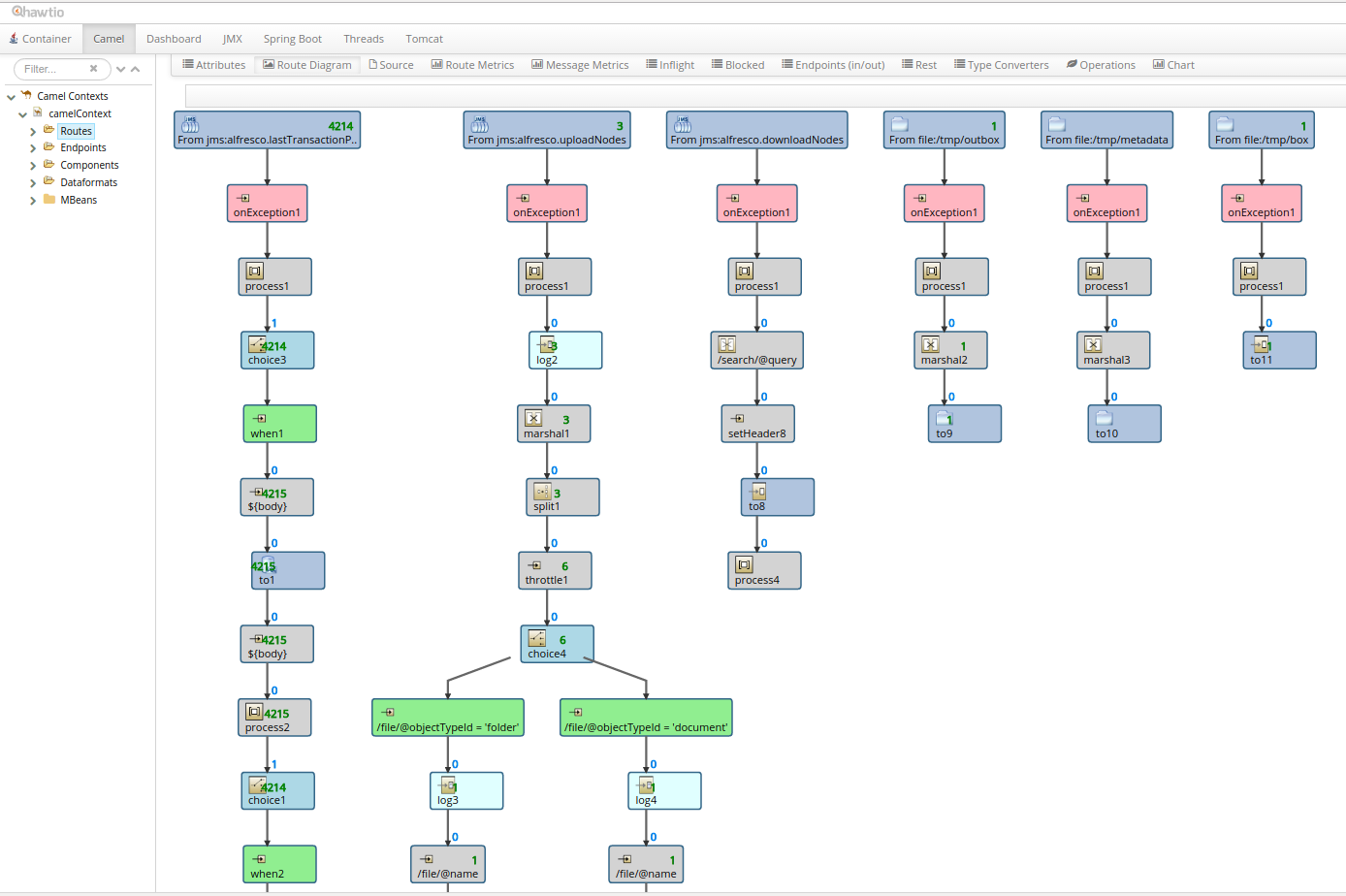
To run the application execute the following command:

mvn spring-boot:run

# Monitoring with Hawtio

Hawtio (http://hawt.io) is a pluggable management console for Java stuff which supports any kind of JVM, any kind of container (Tomcat, Jetty, Karaf, JBoss, Fuse Fabric, etc), and any kind of Java technology and middleware.

Hawtion can help you to visualize Routes with real-time updates on messages metrics



You can get statistical data for each individual route.

