**Enterprise Integration using Apache Camel with Spring Boot**

Apache Camel is a powerful enterprise integration framework which when combined with Spring Boot becomes faster, easier and flexible to implement. It is also a great fit for microservice architecture where the data subscribed from upstream is transformed and sent to downstream systems.

**Introduction**

Apache Camel is a powerful enterprise integration framework which when combined with Spring Boot becomes faster, easier and flexible to implement.

It is also a great fit for microservice architecture where the data subscribed from upstream is transformed and sent to downstream systems.

**Code example:**

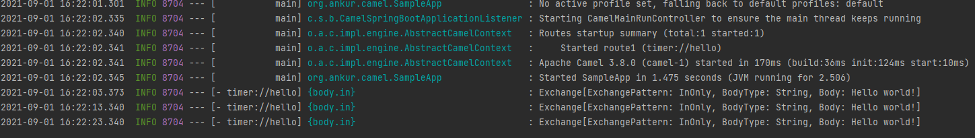
Below are the snippets of a sample camel springboot application that:

1. Fires an event every 10 seconds (input),
2. Appends a text in the event body (process).
3. And then logs it (output).

Steps:

1. Create a new Java Gradle project and add "camel-springboot" dependencies in Java project:
2. plugins {
3. id 'java'
4. id 'org.springframework.boot' version '2.5.0'
5. }
6. dependencies {
7. implementation 'org.apache.camel.springboot:camel-spring-boot-starter:3.8.0'
8. Create the main Spring Boot application class:
9. @SpringBootApplication
10. public class SampleApp {
11. public static void main(String[] args) {
12. SpringApplication.run(SampleApp.class, args);
13. }
14. }
15. Add camel routes. A Route is a basic construct in camel that defines the path the message should take while moving from source to destination.
16. @Component
17. public class FirstRoute extends RouteBuilder {
18. @Override
19. public void configure() {
20. from("{{route.from}}") // source: read from application properties.
21. .setBody(simple("Hello ")) // added the message body.
22. .bean(FirstProcessor.class, "appender") // sent to bean class for processing
23. .to("{{route.to}}"); // destination: read from application properties.
24. }
25. }
26. Create the processor bean that appends "world!" to incoming message body (Hello):
27. @Component
28. public class FirstProcessor {
29. public String appender(String source) {
30. return source + " World!";
31. }
32. }
33. In application.properties file, add below properties:
34. # to keep the application running
35. camel.springboot.main-run-controller = true
36. # source: A timer event that triggers every 10 secs for 5 times
37. route.from=timer:hello?repeatCount=5&period=10000
38. # destination: only logging the message body here.
39. route.to=log:{body.in}
40. Your first camel app using springboot is ready! We can run the spring boot app using below gradle command:

./gradlew bootRun

1. You would see the following output if everything goes fine:  The application is up, routes are started and complete "Hello World" getting printed every 10 secs.

**Summary:**

The above is a simple Camel application that triggers an event periodically, changes the message body and logs it to console.

In real world, Camel framework used to implement various Enterprise Integration Patterns (EIP) like aggregator, splitter, router, publisher-subscriber etc.

Camel offers various components to integrate with multiple sources using different protocols like AWS S3, RabbitMQ, REST API, SFTP, Email etc.

Apache camel when used with SpringBoot is easier and faster to implement. It provides the flexibility to control various aspects of the application using spring properties.