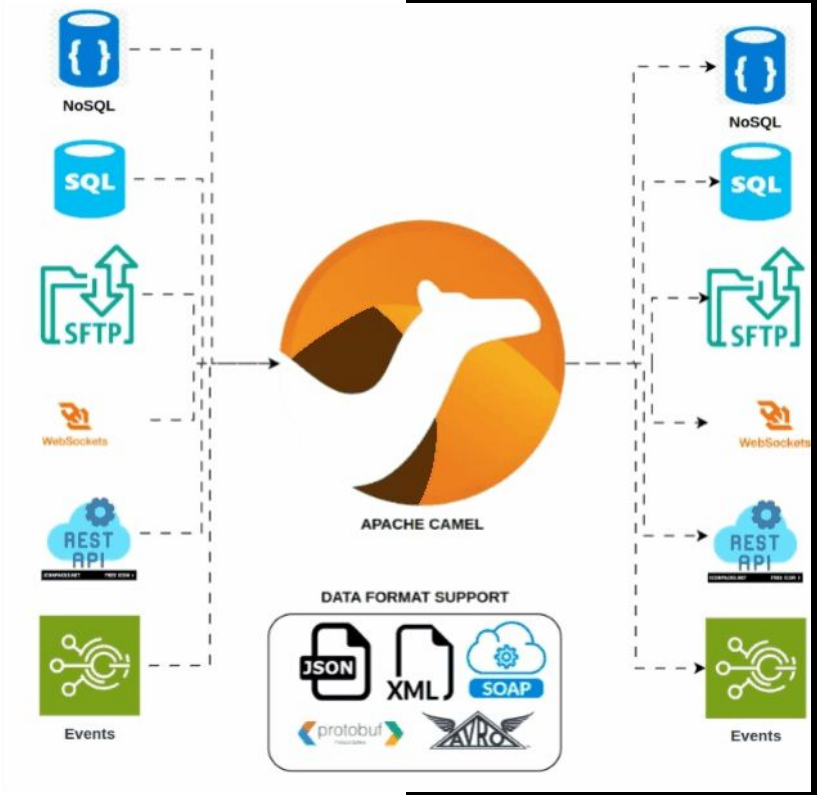
INTEGRACIONES EMPRESARIALES CON APACHE CAMEL

Introducción

Luis Espinel Fuentes

Ingeniero de Sistemas
Universidad de Pamplona - Colombia

APACHE CAMEL



307

Componentes No Core

27

Componentes Core

Apache Camel

Es framework de integración de código abierto que facilita la implementación de soluciones de integración empresarial. Proporciona un conjunto de patrones de integración empresarial (EIP) que permiten a los desarrolladores resolver problemas comunes de integración de forma estandarizada y reutilizable.

Principales ventajas de Apache Camel

01

Patrones de Integración Empresarial (EIP) Documentados "Enterprise Integration Patterns".

02

Permite definir rutas de integración utilizando un lenguaje de dominio específico (DSL)

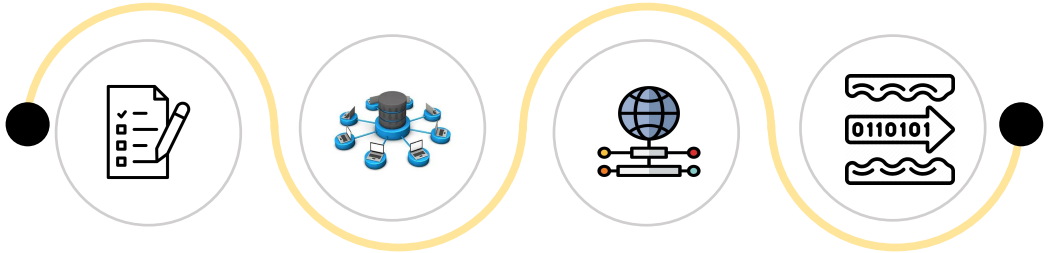
03

Soporte para Múltiples Protocolos y Formatos de Datos

04

Fácil Despliegue: aplicaciones independientes, servidores de aplicaciones, contenedores y en la nube

FLUJOS PARA IMPLEMENTACIÓN DE INTEGRACIONES



Identificar Requerimientos

Identificar Sistemas

Identificar Protocolos

Definir flujo de datos

Implementar Rutas de Integración

Probar Rutas de Integración

Despliegue de Aplicación

Mantenimiento

Terminología

Middleware

Software que conecta diferentes aplicaciones o servicios

Conectores y Adaptadores

Componentes que permiten la conexión e interacción con sistemas externos

Enrutamiento de Mensaje

La capacidad de dirigir los mensajes a diferentes destinos basándose en criterios específicos

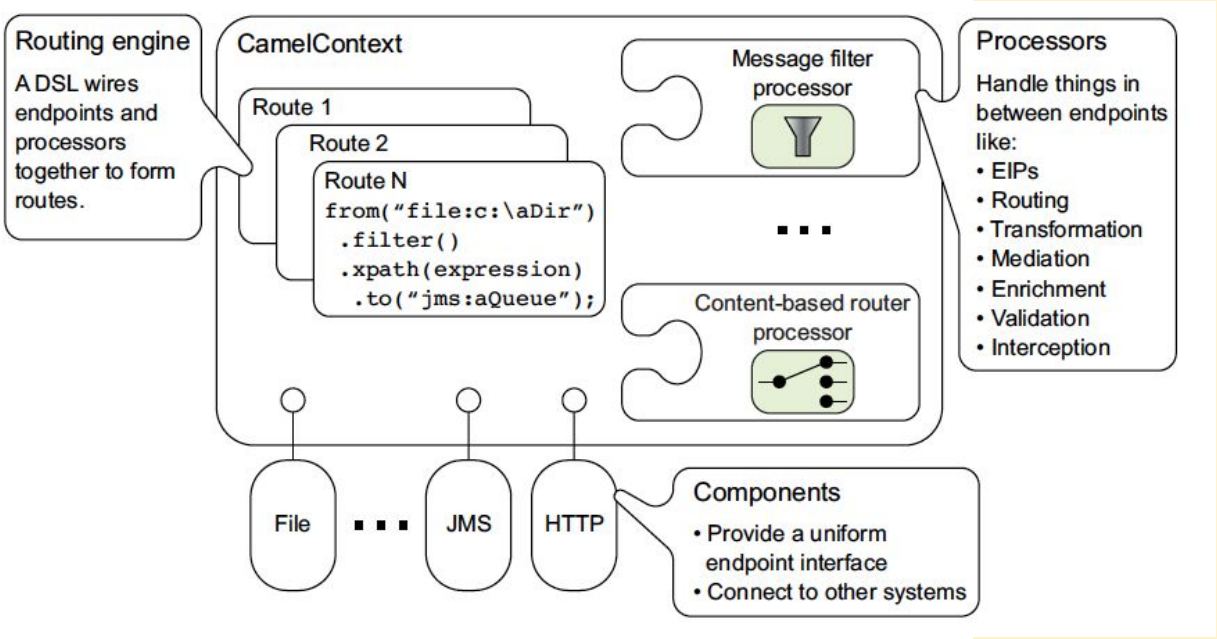
Orquestación

La coordinación de múltiples servicios y componentes para ejecutar un flujo de trabajo complejo

Transformación de Datos

convertir datos de un formato o estructura a otro para que sean compatibles con diferentes sistemas

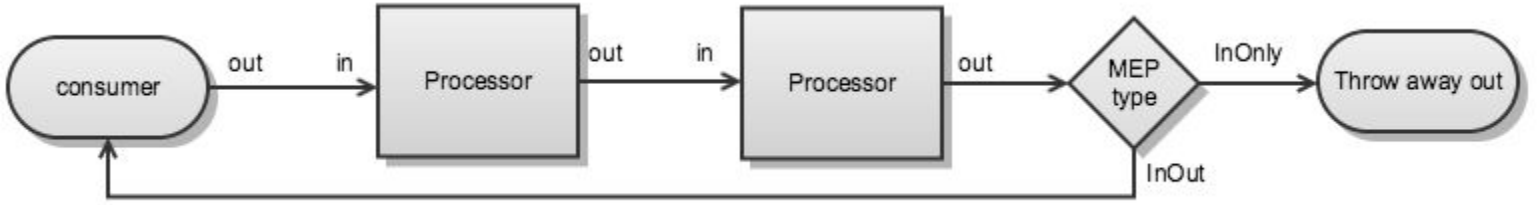
ARQUITECTURA APACHE CAMEL



Review Arquitectura Camel

ARQUITECTURA APACHE CAMEL

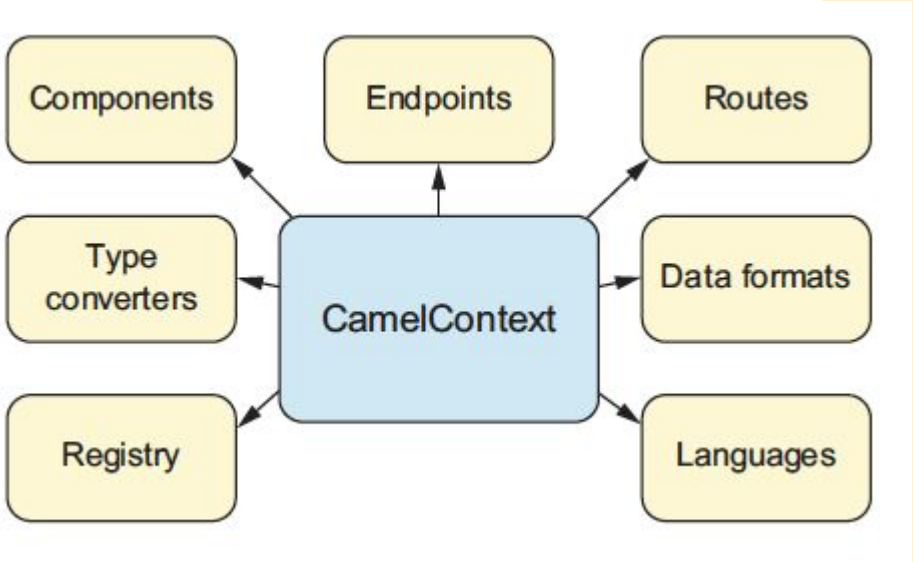
Consumidores esperan datos de entrada que son encapsulados en un objeto Exchange, un Processor tiene la posibilidad de usar, crear o modificar datos de un Exchange que servirá como datos de entradas para el siguiente punto de procesamiento.



Review Arquitectura Camel

CONTEXTO CAMEL

Sistema en tiempo de ejecución que mantiene unido los componentes principales como rutas, endpoints, componentes, etc.



Objetos Contexto Camel

CONTEXTO CAMEL - RUTAS

Abstracción usada por Apache Camel que nos permite definir los flujos de integración de una manera muy fácil y entendible, estas se pueden definir usando los lenguajes específicos de dominio (DSL).

```xml
<camelContext id="demo-routes" xmlns="http://camel.apache.org/schema/blueprint">
        <route>
                <from uri="timer:myTimer?period=1000"/>
                <setBody>
                    <constant>Body Content</constant>
                </setBody>
                <to uri="direct:showBody"/>
        </route>

        <route>
                <from uri="direct:showBody"/>
                <log message="El cuerpo del mensaje es: ${body}"/>
        </route>
</camelContext>
```

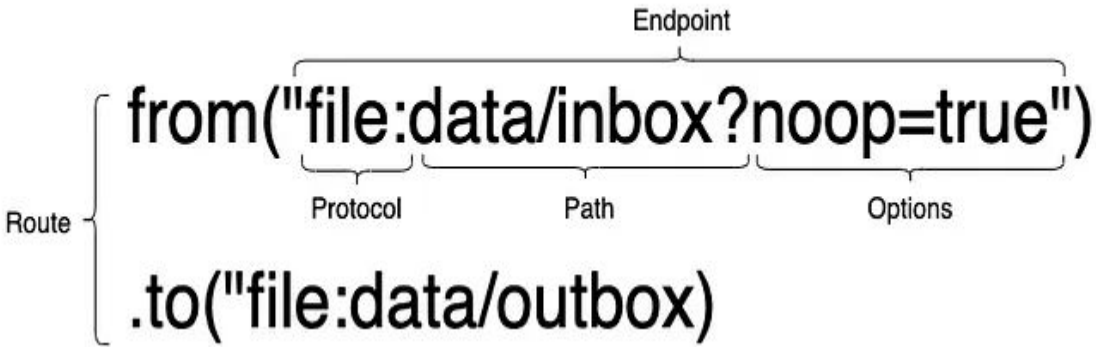
```java
from("direct:orderValidator")
        .routeId("route-order-validator")
        .log("Validating orderId ${body.id} ")
        .choice()
            .when(simple("${body.total} > 500.0"))
                .log("OrderId ${body.id} greater than 500")
                .to("direct:orderFilter")
            .otherwise()
                .log("OrderId ${body.id} less than 500")
        .end()// IMPORTANT: This end finish the choice flow
        .log("Choice finish")
        .to("direct:orderFilter");

from("direct:orderFilter")
        .routeId("route-order-filter")
        .log("Filtering orders with total greater than 500")
        .filter(simple("${body.total} > 800"))
            .log("Filtered OrderId ${body.id} with total ${body.total}")
            .to("direct:confirmOrder")
            .stop() // IMPORTANT: Stop the filtered exchange
        .end() // IMPORTANT:  this 'end' finish the filter flow
        .log("After filter ${body.id} with total ${body.total}");
```

CONTEXTO CAMEL - RUTAS

Lenguajes específicos de dominio (DSL) soportados por Apache Camel

- Java DSL - A Java based DSL using the fluent builder style.
- XML DSL A XML based DSL in Camel XML files only.
- Spring XML - A XML based DSL in classic Spring XML files.
- Yaml DSL for creating routes using YAML format.
- Rest DSL - A DSL to define REST services using REST verbs.
    - Rest DSL contract first - Rest DSL using *contract first* when OpenAPI specs.
- Groovy DSL - A Groovy-based DSL to create routes leveraging closures and a specific Groovy extension module.
- Kotlin DSL - A Kotlin-based DSL.
- Annotation DSL - Use annotations in Java beans.

APACHE CAMEL - RUTAS



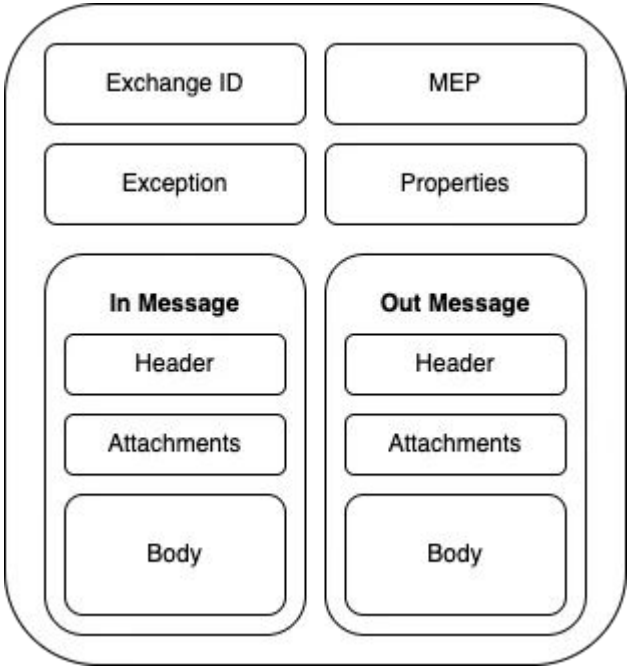
Endpoint

URI que define el uso de un protocolo/componente con definiciones y propiedades (opciones) que definen el comportamiento del procesamiento

Protocolo/Componente

Pieza interna fundamental construida para habilitar la integración con ciertas tecnologías. Ejemplo: REST, Files, SFTP, KAFKA, etc.

APACHE CAMEL - EXCHANGE OBJECT



Exchange

Objeto contenedor de los mensajes de entrada y salida además de encapsular las excepciones y propiedades definidas dentro de la ejecución de las rutas

Message

Objetos que contienen los datos transferidos entre rutas

APACHE CAMEL - MANIPULANDO EL EXCHANGE

```
from("direct:a").recipientList(header("myHeader"));
```

```
from("direct:start")
    .filter(body().contains("aceptado"))
    .to("log:aceptados");
```

- **Headers**

  Podemos acceder a los headers de nuestros mensajes a través del método **header(“NOMBRE”)**

- **Body**

  Para acceder directamente al body de nuestro mensaje podemos usar el método **body()**

APACHE CAMEL - PATRONES DE INTEGRACIÓN MÁS USADOS

Dentro de los patrones de integración más usados en flujos de integración en Apache Camel tenemos:

- Envío y recepción de Mensajes a través de canales estándar
- Split - División de un Mensaje en N Mensajes
- Procesadores
- Agregación de Mensajes
- Enrutamiento basado en contenido
- Filtrado de Mensajes
- Enriquecimiento de contenido
- Wire Tap - Copia del mensaje para procesamiento paralelo
- Recipient List - Enrutado de mensaje a múltiples destinos dinámicamente
- Multicasting
- Dead Letter Channel - Manejo de errores con cola de mensajes muertos

APACHE CAMEL - SIMPLE LANGUAGE

Simple Language en Apache Camel es un lenguaje de expresión sencillo para evaluar y manipular mensajes dentro de las rutas.

Uso Principal:

- Evaluar condiciones en rutas.
- Obtener y modificar encabezados, cuerpo y propiedades de mensajes.
- Configurar rutas dinámicas y acceder a expresiones complejas de forma sencilla.

https://camel.apache.org/components/4.8.x/languages/simple-language.html

APACHE CAMEL - SIMPLE LANGUAGE

Ejemplo de uso de simple dentro de la definición de rutas en Java

```
from("direct:start")
    .choice()
        .when(simple("${header.tipo} == 'pedido'"))
            .to("direct:pedidos")
        .when(simple("${header.tipo} == 'factura'"))
            .to("direct:facturas")
        .otherwise()
            .to("direct:otros");
```

APACHE CAMEL - SIMPLE LANGUAGE

Simple facilita el acceso y modificación de encabezados, cuerpo y propiedades dentro de las rutas.

```
from("direct:procesar")
    .setHeader("id", simple("${body.id}"))
    .setBody(simple("El mensaje tiene ID ${header.id} y contenido ${body.contenido}"))
    .to("log:resultado");
```

Permite enriquecer mensajes de manera fácil definiendo esto dentro de nuestras rutas

```
from("direct:enriquecer")
        .setHeader("timestamp", simple("${date:now:yyyy-MM-dd HH:mm:ss}"))
        .setBody(simple("Mensaje recibido en ${header.timestamp}"));
```

APACHE CAMEL - SIMPLE LANGUAGE

Simple soporta el uso de operadores y cuenta con funciones predefinidas

Operator	Description
==	equals
=~	equals ignore case (will ignore case when comparing String values)
>	greater than
>=	greater than or equals
<	less than
<=	less than or equals
!=	not equals
!=~	not equals ignore case (will ignore case when comparing String values)
contains	For testing if contains in a string-based value
!contains	For testing if it does not contain in a string-based value
~~	For testing if contains by ignoring case sensitivity in a string-based value
!~~	For testing if it does not contain by ignoring case sensitivity in a string-based value

MANEJO DE ERRORES

El manejo de errores en Camel es el conjunto de prácticas y configuraciones que permiten gestionar fallos en el flujo de integración de datos, esto permite que las integraciones sean más resilientes, que puedan manejar fallos de manera controlada, y que eviten interrupciones en los servicios críticos.

Dentro de Camel podemos definir el manejo de excepciones de distintas maneras

- Manejo de errores de manera general por tipo excepción
- De manera global para todas las excepciones
- De manera individual en cada ruta

MANEJO DE ERRORES - MANEJO POR TIPO DE EXCEPCIÓN

Ejemplo de cómo definir el manejo de excepciones a partir del tipo de excepción

```
onException(IllegalArgumentException.class)
        .handled(true)
        .log("Error de argumento no válido: ${exception.message}")
        .to("direct:procesarError");
```

MANEJO DE ERRORES - MANEJO GENERAL

Ejemplo de cómo definir el manejo de excepciones de manera general dentro de un Camel Context

Esto lo podemos hacer aplicando el patrón de DLQ o DLC donde irán los mensajes que al procesar han dado algún tipo de Excepción.

```
errorHandler(deadLetterChannel("jms:queue:dead")
        .maximumRedeliveries(3));
```

MANEJO DE ERRORES - MANEJO EN CADA RUTA

De manera individual en cada ruta podemos hacer uso de doTry…doCatch y doFinally para manejar excepciones presentadas dentro de la ruta

```
from("direct:start")
    .doTry()
        .process(new ProcessorFail())
        .to("mock:result")
    .doCatch(IOException.class, IllegalStateException.class)
        .to("mock:catch")
    .doFinally()
        .to("mock:finally")
    .end();
```

MANEJO DE ERRORES - RECOMENDACIONES

- **Configurar manejadores de errores según el tipo de excepción**

  Usar `onException` para manejar casos específicos y customizados.
- **Utilizar DeadLetterChannel para errores graves**

  Asegura que los mensajes fallidos sean gestionados sin detener el flujo de la aplicación.
- **Definir políticas de reintento adecuadas**

  Configurar reintentos para errores transitorios y limitar reintentos para errores irreversibles.

CASO INTEGRACIÓN

Tienes un sistema que recibe órdenes de compra. Dependiendo del **método de pago** (`credito` o `paypal`), la orden debe ser procesada por una ruta diferente.

Casos:

1. **Si el método de pago es "credito"**, envía la orden a `"direct:credito"`.
2. **Si el método de pago es "paypal"**, envía la orden a `"direct:paypal"`.
3. **Si el método de pago es desconocido**, registra un mensaje de error.

CASO INTEGRACIÓN

Un sistema recibe correos electrónicos, pero solo quiere procesar aquellos que contengan la palabra `"URGENTE"`.

Requisitos:

1. Los correos llegan a `"direct:correos"`.
2. Filtra los mensajes y solo procesa los que contienen `"URGENTE"`.
3. Los correos válidos se envían a `"direct:procesarCorreo"`.

CASO INTEGRACIÓN

Un sistema de notificaciones envía alertas a diferentes canales (email, SMS, Slack), según las preferencias del usuario.

Objetivo:

- Usa `.recipientList()` para enviar la notificación a múltiples destinos dinámicamente.

Requisitos:

1. Los mensajes llegan a `"direct:notificaciones"`.
2. El encabezado `"destinos"` contiene las rutas separadas por comas (ej. `"direct:email,direct:sms"`).
3. Enruta el mensaje a cada destino en la lista.

Evaluemos lo Aprendido

¿Qué es Apache Camel y cuál es su propósito principal?

a) Un framework para crear aplicaciones web

b) Un motor de mensajería empresarial (EIP) para integrar sistemas

c) Un servidor de bases de datos

¿Qué es una “ruta” en Apache Camel?

a) Una dirección IP

b) Una secuencia de instrucciones para procesar un mensaje

c) Un componente para monitorear el tráfico de red

En el contexto de Camel, ¿qué es un “Exchange”?

a) Un formato de archivo

b) Un objeto que representa el mensaje y su estado durante el procesamiento

c) Un servicio de intercambio de divisas

Evaluemos lo Aprendido

¿Qué es un “Message” en Apache Camel?

a) Un comando para iniciar un servidor

b) El contenedor de los datos que se transportan a través de un “Exchange”

c) Un archivo de configuración de red

¿Cómo defines un “Endpoint” en Apache Camel?

a) Una función en Java

b) Una URL que representa el destino o origen de un mensaje en una ruta

c) Un componente gráfico para dibujar diagramas

¿Cuál es la estructura básica de una ruta en Camel?

a) Una ruta debe comenzar con un `from()` y luego definir un conjunto de transformaciones o procesamientos de mensajes

b) Las rutas en Camel se construyen sin un punto de inicio

c) No hay estructura básica, todas las rutas son diferentes

Evaluemos lo Aprendido

¿Cuál es la principal ventaja de usar Simple Language en Apache Camel?

A) Permite acceder y manipular datos en el cuerpo, encabezados y propiedades de los mensajes de manera sencilla.

B) Ofrece soporte para trabajar con múltiples tipos de bases de datos.

C) Simplifica la administración de servicios SOAP.

D) Reemplaza la necesidad de definir beans en Camel.

¿Cuál es la sintaxis correcta para acceder al valor de un encabezado llamado tipo en una ruta Camel usando Simple? (Varias Opciones son correctas)

A) header(tipo)

B) ${headers.tipo}

C) ${header.tipo}

D) header["tipo"]

Evaluemos lo Aprendido

¿Qué componente se utiliza para definir la lógica de procesamiento en una ruta en Java?

a) `RouteManager`

b) `Processor`

c) `Component`

¿Qué rol juega `RouteBuilder` en la definición de rutas en Camel?

a) Proporciona una interfaz gráfica para diseñar rutas

b) Define y configura rutas de integración utilizando código Java

c) Define el esquema de bases de datos relacionales

¿Qué función cumple el método `from()` en una ruta de Camel?

a) Especificar el origen de los datos en la ruta

b) Establecer las credenciales de seguridad

c) Conectar varios componentes de forma paralela

COMUNICACIÓN ASÍNCRONA ENTRE RUTAS

La comunicación asíncrona entre rutas permite que las rutas interactúen sin esperar a que se completen las operaciones de una ruta antes de iniciar otra.
Esto es útil para desacoplar rutas, mejorar la eficiencia y manejar grandes volúmenes de datos o tareas que pueden procesarse en paralelo.

```
// Ruta de origen que envía mensajes a la cola `seda`
from("direct:start")
    .log("Enviando mensaje a la cola asíncrona SEDA")
    .to("seda:asyncQueue");

// Ruta que consume los mensajes en paralelo
from("seda:asyncQueue?concurrentConsumers=5")
    .log("Mensaje procesado asíncronamente: ${body}")
    .process(exchange -> {
        // Lógica de procesamiento
    });
```

COMUNICACIÓN ASÍNCRONA ENTRE RUTAS

Los componentes que habilitan por default el comportamiento asíncrono al momento de comunicar rutas son los siguientes

- SEDA
- DIRECT-VM
- VM
- JMS

También contamos con algunos Patrones de Integración que nos habilitan el procesamiento en paralelo como:

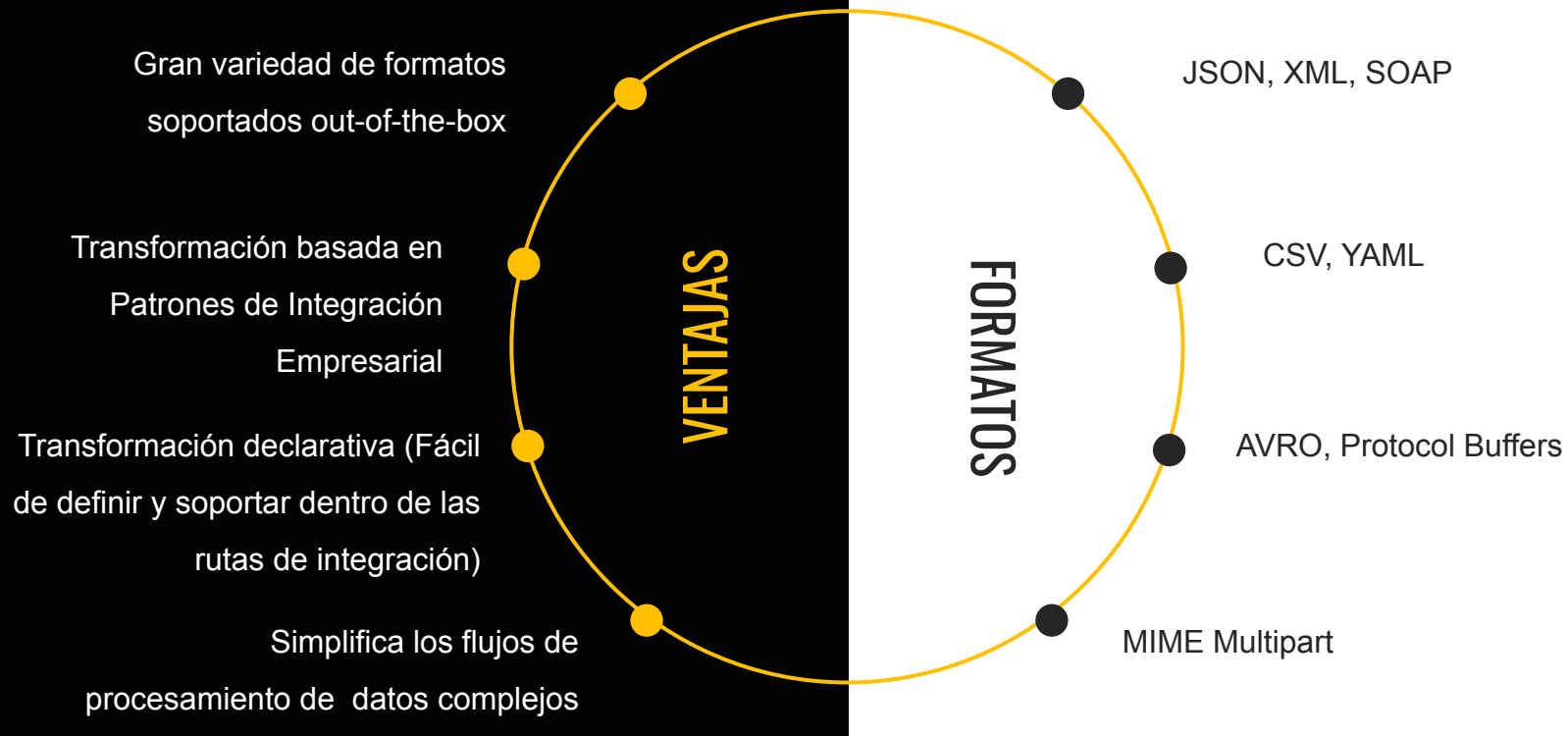
- Multicast con Parallel Processing
- Split con parallel processing
- WireTap

COMUNICACIÓN ASÍNCRONA ENTRE RUTAS

```
from("direct:start")
    .log("Mensaje principal recibido: ${body}")
    .wireTap("direct:asyncProcess");

from("direct:asyncProcess")
    .log("Procesando mensaje en paralelo: ${body}")
    .process(exchange -> {
        // Lógica de procesamiento
    });
```

```
from("direct:start")
    .multicast().parallelProcessing()
        .to("direct:route1", "direct:route2", "direct:route3");
```

TRANSFORMACIÓN DE DATOS



DATA FORMATS

Lista de formatos soportados directamente por Apache Camel

Lista completa: https://camel.apache.org/components/4.4.x/dataformats/index.html

Data Format	Artifact	Support Level	Since	Description
ASN.1 File	camel-asn1	Stable	2.20	Encode and decode data structures using Abstract Syntax Notation One (ASN.1).
Avro	camel-avro	Stable	2.14	Serialize and deserialize messages using Apache Avro binary data format.
Avro Jackson	camel-jackson-avro	Stable	3.10	Marshal POJOs to Avro and back using Jackson.
Barcode	camel-barcode	Stable	2.14	Transform strings to various 1D/2D barcode bitmap formats and back.
Base64	camel-base64	Stable	2.11	Encode and decode data using Base64.
BeanIO	camel-beanio	Stable	2.10	Marshal and unmarshal Java beans to and from flat files (such as CSV, delimited, or fixed length formats).
Bindy	camel-bindy	Stable	2.0	Marshal and unmarshal between POJOs and key-value pair (KVP) format using Camel Bindy
CBOR	camel-cbor	Stable	3.0	Unmarshal a CBOR payload to POJO and back.
Crypto (Java Cryptographic Extension)	camel-crypto	Stable	2.3	Encrypt and decrypt messages using Java Cryptography Extension (JCE).
CSV	camel-csv	Stable	1.3	Handle CSV (Comma Separated Values) payloads.
FHIR JSon	camel-fhir	Stable	2.21	Marshall and unmarshall FHIR objects to/from JSON.
FHIR XML	camel-fhir	Stable	2.21	Marshall and unmarshall FHIR objects to/from XML.
Flatpack	camel-flatpack	Stable	2.1	Marshal and unmarshal Java lists and maps to/from flat files (such as CSV, delimited, or fixed length formats) using Flatpack library.
Grok	camel-grok	Stable	3.0	Unmarshal unstructured data to objects using Logstash based Grok patterns.
GZip Deflater	camel-zip-deflater	Stable	2.0	Compress and decompress messages using java.util.zip.GZIPStream.
HL7	camel-hl7	Stable	2.0	Marshal and unmarshal HL7 (Health Care) model objects using the HL7 MLLP codec.
iCal	camel-ical	Stable	2.12	Marshal and unmarshal iCal (.ics) documents to/from model objects.
Jackson XML	camel-jacksonxml	Stable	2.16	Unmarshal an XML payloads to POJOs and back using XMLMapper extension of Jackson.
JAXB	camel-jaxb	Stable	1.0	Unmarshal XML payloads to POJOs and back using JAXB2 XML marshalling standard.
JSON Fastjson	camel-fastjson	Stable	2.20	Marshal POJOs to JSON and back using Fastjson
JSON Gson	camel-gson	Stable	2.10	Marshal POJOs to JSON and back using Gson
JSON Jackson	camel-jackson	Stable	2.0	Marshal POJOs to JSON and back using Jackson.

DATA FORMATS - MARSHAL

El proceso de Marshal convierte un objeto o estructura de datos en un formato específico (por ejemplo, JSON, XML, etc.) para que pueda ser transmitido o almacenado de manera uniforme.

Marshall se usa cuando tenemos un Objeto Java dentro de nuestros flujos Camel y requerimos transformarlos a formatos específicos para transmitirlos a través de la red.

```java
from("direct:start")
    .marshal().json()
    .to("log:jsonOutput");
```

DATA FORMATS - UNMARSHAL

El proceso de Unmarshal toma datos en un formato específico y los convierte en un objeto o estructura utilizable en el código de aplicación.

Unmarshall se usa cuando tenemos dentro de nuestro Exchange datos en formato JSON, XML, etc y requerimos mapearlos a objetos Java para usarlos dentro de nuestra lógica de integración

```java
from("direct:start")
    .unmarshal().json()
    .process(exchange -> {
        MyObject obj = exchange.getIn().getBody(MyObject.class);
        System.out.println("Nombre: " + obj.getName());
    });
```

MARSHAL & UNMARSHAL

Apache Camel facilita el manejo de estos dos modos de transformación de datos dentro de las rutas de integración

A continuación se muestra un ejemplo de esto

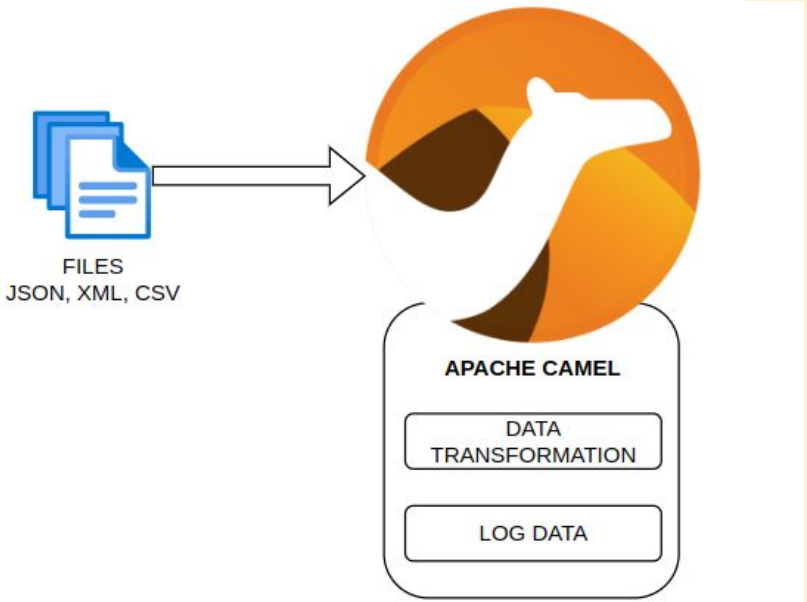
```java
from("direct:start")
    .marshal().json()          // Java a JSON
    .unmarshal().json()        // JSON a Java
    .marshal().jaxb("com.example.model") // Java a XML
    .to("log:output");
```

CASO INTEGRACIÓN

Desarrolle un proyecto Apache Camel que sea capaz de leer archivos entrantes en formatos JSON, XML y CSV de una carpeta del sistema.

El formato debe evaluarse a partir de la extensión del archivo entrante (.json, .xml ó .csv)

En caso de que el archivo no contenga ninguna de estas extensiones debe omitirse el procesamiento.



APACHE CAMEL - BEANS

Camel soporta la inyección de Beans (Clases Java con lógica de negocio específica) para ser usados dentro de la definición de las rutas de integración

```
from( uri: "direct:deleteUser") RouteDefinition
        .log("Processing request Delete User by User Id ${header.userId}")
        .bean( bean: "fakeUsersRepository",  method: "delete")
        .choice() ChoiceDefinition
```

```
@BindToRegistry("fakeUsersRepository")  no usages  Luis Espinel
public FakeUsersRepository usersRepository(){
    return new FakeUsersRepository();
}
```

APACHE CAMEL - BEANS

Inyección y uso de Beans en rutas Camel DSL - XML

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

        <bean id="poDataGenerator" class="com.lhespinel.camel.beans.PoDataGenerator"></bean>

        <camelContext xmlns="http://camel.apache.org/schema/blueprint">

                <route id="route-main-timer">
                        <from uri="timer:mainTimer?period=5000"/>
                        <to uri="bean:poDataGenerator?method=generateOrders" />
                        <log message="Total Body Size (Orders): ${body.size}"/>
                        <to uri="direct:processOrders" />
                </route>
```

BEANS - ENLACE DE PARÁMETROS

Podemos con el uso de anotaciones o nombramiento especial inyectar de manera directa los datos existentes dentro de un Exchange

Annotation	Meaning	Parameter
`org.apache.camel.Body`	To bind to an inbound message body	
`org.apache.camel.Header`	To bind to a message header	String name of the header
`org.apache.camel.Headers`	To bind to the Map of the message headers	
`org.apache.camel.Variable`	To bind to a named variable	String name of the variable
`org.apache.camel.Variables`	To bind to the variables map	
`org.apache.camel.ExchangeProperty`	To bind to a named property on the exchange	String name of the property
`org.apache.camel.ExchangeProperties`	To bind to the exchange property map on the exchange	
`org.apache.camel.ExchangeException`	To bind to an Exception set on the exchange	

BEANS - ENLACE DE PARÁMETROS

Ejemplo de uso de anotaciones como @Header, @Headers, @Body

```
@
    public User save(@Body User user){  no usages   Luis Espinel
        user.setId(UUID.randomUUID().toString());
        users.add(user);
        return user;
    }

    public boolean delete(@Header("userId") String userId){  no usages   Luis Espinel
        return users.remove(User.builder().id(userId).build());
    }
```

LECTURA DE PROPERTIES

Los **properties** en Apache Camel permiten **parametrizar** configuraciones, facilitando la separación entre código y valores externos (como URLs, credenciales, límites de reintentos, etc.).

properties

```
mi.ruta.origen=direct:entrada
mi.ruta.destino=log:salida
reintentos=3
timeout=5000
```

java

```
from("{{mi.ruta.origen}}")
    .routeId("MiRuta")
    .log("Mensaje recibido: ${body}")
    .to("{{mi.ruta.destino}}");
```

```
public class MiServicio {

    @PropertyInject("timeout")
    private int timeout;
```

EXPOSICIÓN DE SERVICIOS REST - REST DSL

Camel REST: Soporte de configuración declarativa

Soporte de Open API e integración fácil con Swagger

```
@Override
public void configure() throws Exception {
    restConfiguration()
            .contextPath(pathBase)
            .component( componentId: "jetty") // App server to use for REST APIs exposition
            .host("0.0.0.0")
            .port(serverPort)
            .bindingMode(RestBindingMode.json)
            .apiContextPath("/api-docs")
                .apiProperty("api.title", "Camel REST API")
                .apiProperty("api.version", "1.0")
                .apiProperty("cors", "true")
                .apiProperty("api.specification.contentType.json", "application/vnd.oai.openapi+json;version=2.0")
                .apiProperty("api.specification.contentType.yaml", "application/vnd.oai.openapi;version=2.0");
```

EXPOSICIÓN DE SERVICIOS REST - REST DSL

Camel REST: Soporte de configuración declarativa

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
        <camelContext xmlns="http://camel.apache.org/schema/blueprint">

                <restConfiguration component="jetty" port="8080">
                </restConfiguration>

                <rest path="/v1/examples">
                        <get path="/hello">
                                <to uri="direct:hello"/>
                        </get>
                        <get path="/bye" consumes="application/json">
                                <to uri="direct:bye"/>
                        </get>
                        <post path="/bye">
                                <to uri="mock:update"/>
                        </post>
                </rest>

                <route>
                        <from uri="direct:hello"/>
                        <transform>
                                <constant>Hello World</constant>
                        </transform>
                </route>
```

CAMEL REST OPEN API - CONTRACT FIRST

Se soporta la aproximación de Contract First donde se provee una definición OpenAPI y Camel automáticamente expondrá los servicios definidos allí. [Documentación]

```
@Override
public void configure() throws Exception {
    rest().openApi("petstore-v3.json");
}
```

```
http://0.0.0.0:8080/api/v3/user                    (POST)   (accept:application/json,application/x-www-form-urlencoded,application/xml produce:a
http://0.0.0.0:8080/api/v3/user/createWithList     (POST)   (accept:application/json produce:application/json,application/xml)
http://0.0.0.0:8080/api/v3/user/login              (GET)    (produce:application/json,application/xml)
http://0.0.0.0:8080/api/v3/user/logout             (GET)
http://0.0.0.0:8080/api/v3/user/{username}         (DELETE,GET,PUT)
```

```
http://0.0.0.0:8080/api/v3/user                    direct:createUser
http://0.0.0.0:8080/api/v3/user/createWithList     direct:createUsersWithListInput
http://0.0.0.0:8080/api/v3/user/login              direct:loginUser
http://0.0.0.0:8080/api/v3/user/logout             direct:logoutUser
http://0.0.0.0:8080/api/v3/user/{username}         direct:getUserByName
```

SERVICIOS REST USANDO APACHE CXF

Camel soporta el enlace con Controladores REST Y SOAP implementados usando CXF

Ejemplo Implementación

```
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;

@Path("/")
public class EmployeeServiceResource {

        public EmployeeServiceResource() {
        }

        @GET
        @Path("/employees/{name}/")
        public String getCustomer(@PathParam("name") String name) {
                return null;
        }
```

SERVICIOS REST USANDO APACHE CXF

Camel soporta el enlace con Controladores REST Y SOAP implementados usando CXF

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:cxf="http://camel.apach
        xmlns:jaxrs="http://cxf.apache.org/jaxrs"
        xsi:schemaLocation="
       http://www.springframework.org/schema/beans http://www.springframework.org/schema/be
       http://camel.apache.org/schema/cxf http://camel.apache.org/schema/cxf/camel-cxf.xsd
       http://cxf.apache.org/jaxrs http://cxf.apache.org/schemas/jaxrs.xsd
       http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-sp

        <cxf:rsServer id="restService" address="http://localhost:9000/employeeservice"
                serviceClass="com.javainuse.beans.EmployeeServiceResource">
        </cxf:rsServer>

        <bean id="processor" class="com.javainuse.beans.CamelProcessor" />

        <camelContext id="camelId" xmlns="http://camel.apache.org/schema/spring">
                <route>
                        <from uri="cxfrs://bean://restService" />
                        <process ref="processor" />
                </route>
        </camelContext>

</beans>
```

APACHE CAMEL - CUESTIONARIO APIS REST

¿Qué componente en Apache Camel se utiliza para exponer servicios REST?

- A) jms
- B) cxf
- C) rest
- D) ftp

¿Cuál de los siguientes métodos se utiliza en Camel para definir la configuración general de servicios REST?

- A) `restEndpoints()`
- B) `restConfiguration()`
- C) `configureRest()`
- D) `setupRest()`

¿Cuál es el propósito del método `rest()` en Apache Camel?

- A) Ejecutar un método de autenticación.
- B) Configurar una ruta para manejar servicios REST y definir sus endpoints.
- C) Configurar los headers de las solicitudes HTTP.
- D) Definir solo las credenciales del servicio REST.

APACHE CAMEL - CUESTIONARIO APIS REST

¿Qué función cumple `bindingMode` en la configuración de servicios REST en Camel?

- A) Define la URL base del servicio REST.
- B) Especifica el número de reintentos.
- C) Determina si la respuesta se serializa a JSON, XML ó automáticamente.
- D) Establece la conexión con la base de datos.

¿Cómo especificas la URL base para un servicio REST en Camel?

- A) En el método `restBaseUrl()`
- B) En el método `restConfiguration().host()`
- C) Con el método `restConfiguration().contextPath()`
- D) Con el método `baseUri()`

¿Qué anotación o configuración adicional se necesita para transformar automáticamente el cuerpo del mensaje a JSON?

- A) `.setBody()`
- B) `bindingMode(RestBindingMode.JSON)`
- C) `.toJson()`
- D) `.serializeAsJSON()`

APACHE CAMEL - CUESTIONARIO APIS REST

¿Cuál de las siguientes configuraciones es necesaria para que Camel funcione con un servidor Jetty embebido para servicios REST?

- A) `.restConfiguration().component("jetty")`
- B) `.restServer().jettyServer(true)`
- C) `.setupComponent("jetty")`
- D) `.setJettyMode(true)`

¿Qué hace el siguiente código en una ruta Camel?

```
rest("/api").get("/users").route().to("bean:userService?method=getAllUsers");
```

- A) Crea un servicio SOAP para la ruta `/api/users`.
- B) Crea un servicio REST para la ruta `/api/users` que ejecuta el método `getAllUsers` del bean `userService`.
- C) Agrega autenticación para la ruta `/api/users`.
- D) Redirige las solicitudes a otro servidor.

Camel REST DSL

Ventajas:

1. **Simplicidad y Facilidad de Uso**:
   - **Configuración Sencilla**: La configuración es más directa y concisa. Ideal para casos de uso simples y medianos.
   - **Integración con Rutas Camel**: Permite definir y manipular rutas Camel directamente junto con las definiciones REST.
2. **Desempeño**:
   - **Menos Dependencias**: Generalmente tiene menos dependencias adicionales en comparación con CXF, lo que puede resultar en un mejor rendimiento.
3. **Flexibilidad**:
   - **Opciones de Serialización**: Soporte integrado para serialización/deserialización de JSON, XML, etc., con bibliotecas populares como Jackson.
   - **Integración Simplificada**: Se integra fácilmente con otros componentes Camel sin necesidad de configuraciones adicionales.

COMPARATIVA OPCIONES EXPOSICIÓN DE SERVICIOS REST

Camel REST DSL

Desventajas:

1. **Funcionalidad Limitada**:
   - **Características Avanzadas**: Puede carecer de algunas de las características avanzadas de WS-* que ofrece CXF.
   - **Seguridad y Políticas**: Implementar políticas de seguridad y WS-Security puede ser más limitado en comparación con CXF.
2. **Menos Estandarización**:
   - *No WS- Compliant**: No cumple con todas las especificaciones de servicios web SOAP/WS-*.

Apache CXF

Ventajas:

1. **Características Avanzadas**:
   - *WS- Support*: Soporte completo para especificaciones WS-*, como WS-Security, WS-RM, WS-Policy, etc.
   - **SOAP y REST**: Puede manejar tanto servicios SOAP como REST, proporcionando una solución más completa para servicios web.
2. **Estandarización**:
   - **Cumplimiento de Especificaciones**: Cumple con diversas especificaciones y estándares de servicios web, lo que puede ser crucial para integraciones empresariales complejas.
3. **Seguridad**:
   - **WS-Security**: Soporte completo para WS-Security y otras políticas de seguridad, lo que permite una implementación más robusta de seguridad a nivel de mensaje.

Apache CXF

Desventajas:

1. **Complejidad**:
   - **Configuración Compleja**: Requiere más configuración y puede ser más complejo de implementar, especialmente para casos de uso simples.
   - **Curva de Aprendizaje**: Mayor curva de aprendizaje debido a la necesidad de comprender las especificaciones WS-* y la configuración de CXF.
2. **Desempeño**:
   - **Mayor Sobrecarga**: La implementación de todas las características avanzadas puede introducir una mayor sobrecarga en comparación con el REST DSL.

APACHE CAMEL - INTEGRACIÓN CON BASES DE DATOS

Apache Camel provee distintas capacidades para integrarse con distintos tipos de bases de datos.

Para acceso a bases de datos relacionales podemos hacer uso de componentes como

- Componente SQL: Provee la capacidad de interactuar directamente con bases de datos relacionales definiendo Query nativas sobre los motores de bases de datos
- Componente JPA: Habilita todas las funcionalidades provistas por JPA para poder interactuar con nuestras bases de datos a través de objetos Java

Para acceso a bases de datos NoSQL

- Componente MongoDB
- Componente JPA: Con soporte para bases de datos no relacionales como Mongo
- Componentes especializados para motores específicos como Athena, CouchDB, ArangoDB, AWS Datasources, GCP Datasources, etc.

APACHE CAMEL - DATASOURCE CONFIGURATION

Para poder acceder a las bases de datos debemos configurar un Datasource dentro de nuestro proyecto Camel, para esto podemos hacer a través de generación de Beans (Modo Standalone) ó configurando los Datasource en Spring Boot

```java
@Configuration
public class DatasourceConfig {

    @BindToRegistry
    public DataSource mysqlDatasource(@PropertyInject("database.host") String host,
                                      @PropertyInject(value = "database.port", defaultValue = "3306") Integer port,
                                      @PropertyInject("database.name") String databaseName,
                                      @PropertyInject("database.user") String user,
                                      @PropertyInject("database.pass") String password)
    {
        org.apache.commons.dbcp2.BasicDataSource dataSource = new org.apache.commons.dbcp2.BasicDataSource();
        dataSource.setDriverClassName("com.mysql.cj.jdbc.Driver");
        dataSource.setUrl("jdbc:mysql://"+host +":"+port+"/"+databaseName+"?useSSL=true&serverTimezone=UTC");
        dataSource.setUsername(user);
        dataSource.setPassword(password);
        return dataSource;
    }

}
```

```
// Example for a mysql datasource
spring.datasource.url=jdbc:mysql://localhost/test
spring.datasource.username=dbuser
spring.datasource.password=dbpass
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

APACHE CAMEL - COMPONENTE SQL

Luego de agregar las dependencias requeridas dentro de nuestro proyecto camel-sql en modo Standalone ó camel-sql-starter dentro de Spring Boot vamos a poder definir nuestras consultas SQL

```
from("direct:selectOrders")
    .to("sql:select * from orders where status = 'NEW'")
    .to("log:orders");
```

APACHE CAMEL - JPA INTEGRATION

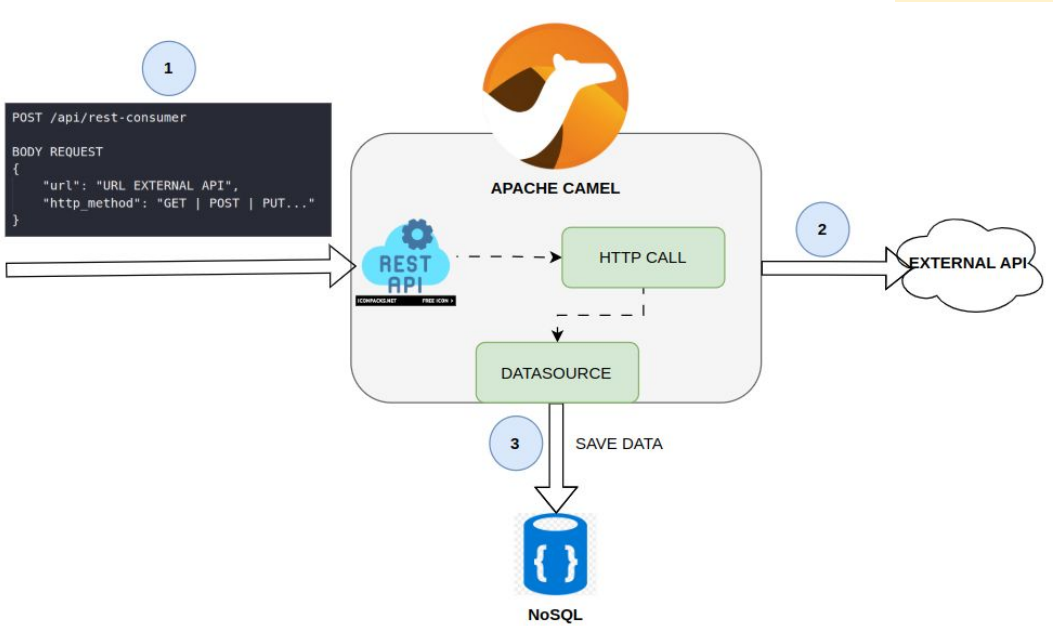
Luego de agregar las dependencias requeridas dentro de nuestro proyecto camel-jpa en modo Standalone ó camel-jpa-starter dentro de Spring Boot vamos a poder definir nuestras entidades

```java
@Entity
public class Order {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String product;
    private Integer quantity;
    private Double price;
    // Getters y setters
}
```

```java
from("direct:start")
    .process(exchange -> {
        Order order = new Order();
        order.setProduct("Laptop");
        order.setQuantity(2);
        order.setPrice(1200.0);
        exchange.getIn().setBody(order);
    })
    .to("jpa:com.example.Order");
```

CASO INTEGRACIÓN

Desarrolle un proyecto Camel en donde se exponga un servicio REST que reciba como body de entrada una URL y un método HTTP para posteriormente con estos datos consumir dicha URL a través del método HTTP y los datos de respuesta deberán ser persistidos en una base de datos MongoDB



PRINCIPALES COMPONENTES USADOS EN CAMEL

A continuación se listan los componentes Camel más usados para implementar integraciones de Software a nivel Empresarial

Componentes de Transporte y Mensajería

- **File**: Para leer y escribir archivos en el sistema de archivos.
- **FTP/SFTP**: Para transferir archivos mediante FTP/SFTP.
- **JMS (Java Message Service)**: Para integrar con colas y temas de mensajería JMS.
- **AMQP**: Para trabajar con protocolos de mensajería avanzada como RabbitMQ.
- **MQTT**: Para integraciones basadas en el protocolo MQTT.
- **HTTP/HTTPS**: Para realizar y manejar solicitudes HTTP/HTTPS.

Componentes de Bases de Datos y Almacenamiento

- **JDBC**: Para interactuar con bases de datos relacionales usando JDBC.
- **JPA**: Para integrar con JPA (Java Persistence API).
- **MongoDB**: Para trabajar con bases de datos NoSQL MongoDB.
- **Cassandra**: Para integrarse con la base de datos NoSQL Cassandra.
- **ElasticSearch**: Para interactuar con el motor de búsqueda ElasticSearch.

PRINCIPALES COMPONENTES USADOS EN CAMEL

Componentes de Transformación de Datos

- **XSLT**: Para transformaciones XML usando XSLT.
- **JSON**: Para trabajar con datos JSON.
- **CSV**: Para leer y escribir archivos CSV.
- **Bean**: Para invocar métodos de Java y manipular datos.
- **Dozer**: Para mapeo de objetos complejos.

Componentes de Web Services y APIs

- **CXF**: Para exponer y consumir servicios web SOAP y RESTful.
- **REST**: Para construir y consumir servicios RESTful.
- **Swagger**: Para documentar y probar APIs RESTful.

PRINCIPALES COMPONENTES USADOS EN CAMEL

Componentes de Integración Empresarial (EIP)

- **Splitter**: Para dividir un mensaje en múltiples partes.
- **Aggregator**: Para combinar varios mensajes en uno solo.
- **Content-based Router**: Para enrutar mensajes según su contenido.
- **Recipient List**: Para enviar mensajes a múltiples destinatarios.
- **Dynamic Router**: Para enrutar mensajes a destinos dinámicos.
- **Message Filter**: Para filtrar mensajes según ciertos criterios.

Componentes de Sistemas de Gestión de Procesos

- **Quartz**: Para la planificación y ejecución de tareas cron.
- **Scheduler**: Para la programación simple de tareas.

PRODUCER TEMPLATE

Podemos desde nuestro código de programación llamar directamente a rutas Camel que existan sobre un contexto Camel determinado, esto teniendo acceso a nuestro objeto Context Camel. Documentación

```
// Crear un ProducerTemplate para enviar mensajes a las rutas
ProducerTemplate template = context.createProducerTemplate();

// Enviar mensaje a la ruta "direct:start"
String response1 = template.requestBody("direct:start", "Hello Camel!", String.class);
System.out.println("Response from direct:start: " + response1);

// Enviar mensaje a la ruta "direct:process" y recibir la respuesta
String response2 = template.requestBody("direct:process", "Hello World!", String.class);
System.out.println("Response from direct:process: " + response2);
```

ALGUNOS COMPONENTES CAMEL

QUARTZ

Apache Camel permite la integración con Quartz para la planificación y ejecución de tareas de manera cron

```
@Override
public void configure() throws Exception {
    from("quartz://myTimer?cron=0+0/1+*+*+*+?")
        .routeId("quartzRoute")
        .log("Ejecutando tarea programada cada minuto");
}
```

ALGUNOS COMPONENTES CAMEL

SFTP

Apache Camel proporciona un componente SFTP que permite transferir archivos a través de SSH (Secure Shell) o usando también el protocolo FTP

```
@Override
public void configure() throws Exception {
    from("sftp://user@remote-server.com:22/download?password=secret&binary=true")
        .to("file://local-directory/downloaded")
        .log("Archivo descargado exitosamente: ${file:name}");
}
```

ALGUNOS COMPONENTES CAMEL

MAIL

Permite enviar de manera fácil emails haciendo uso de Spring Mails y el sistema interno de mails de Java

```
from("direct:startEmailRoute")
        .setHeader("subject", constant("Aviso Importante"))
        .setHeader("to", constant("destinatario@example.com"))
        .setHeader("from", constant("tu-email@example.com"))
        .setBody(constant("Este es el cuerpo del mensaje de correo electrónico."))
       .to("smtp://smtp.example.com:587?username=tu-email@example.com&password=tu_contraseña")
        .log("Correo electrónico enviado exitosamente a ${header.to}");
```

ALGUNOS COMPONENTES CAMEL

MONGODB

Permite interactuar con bases de datos MongoDB

```
from("direct:startMongoInsert")
        .routeId("mongoInsertRoute")
        .process(exchange -> {
            Map<String, Object> documento = new HashMap<>();
            documento.put("nombre", "Juan");
            documento.put("edad", 30);
            documento.put("correo", "juan@example.com");
            exchange.getIn().setBody(documento);
        })
        .to("mongodb:myMongo?database=miBaseDeDatos&collection=usuarios&operation=insert")
        .log("Documento insertado en MongoDB: ${body}");
```

ALGUNOS COMPONENTES CAMEL

MyBatis

Permite interactuar con bases de datos Relacionales

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="userMapper">

    <select id="selectUser" parameterType="int" resultType="User">
        SELECT * FROM usuarios WHERE id = #{id}
    </select>

    <insert id="insertUser" parameterType="User">
        INSERT INTO usuarios (nombre, edad, correo) VALUES (#{nombre}, #{edad}, #{
    </insert>

</mapper>
```

```java
public class User {
    private Integer id;
    private String nombre;
    private Integer edad;
    private String correo;

    // Getters y setters
}
```

```java
from("direct:startInsertUser")
    .routeId("myBatisInsertRoute")
    .process(exchange -> {
        User user = new User();
        user.setNombre("Juan");
        user.setEdad(30);
        user.setCorreo("juan@example.com");
        exchange.getIn().setBody(user);
    })
    .to("mybatis:insertUser?statementType=Insert")
    .log("Usuario insertado exitosamente: ${body}");
```

ALGUNOS COMPONENTES CAMEL

REDIS

Para el uso de cachés dentro de nuestros desarrollos

```
from("direct:startSetRedis")
    .routeId("redisSetRoute")
    .process(exchange -> {
        exchange.getIn().setHeader("CamelRedis.Key", "user:1001");
        exchange.getIn().setBody("Juan");
    })
    .to("redis://localhost:6379?command=SET")
    .log("Valor guardado en Redis: Clave=user:1001, Valor=Juan");
```

OTROS EIP CON CAMEL

VALIDATE

Otra forma para validar el contenido de los mensajes en nuestras rutas para poder validar que sean válidos para procesarse por determinado flujo. Se arroja una excepción de tipo **PredicateValidationException** en caso de que no se pase la validación

```
<route>
  <from uri="file:inbox"/>
  <validate>
    <simple>${header.bar} &gt; 100</simple>
  </validate>
  <to uri="bean:myServiceBean" method="processLine"/>
</route>
```

OTROS EIP CON CAMEL

INTERCEPT

Permite interceptar cada ruta de procesamiento antes de su inicio (se intercepta cada From llamado con To), sin interferir directamente en el flujo definido

```
interceptFrom()
  .to("log:incoming");

from("jms:queue:order")
  .to("bean:validateOrder")
  .to("bean:processOrder");
```

```
<camelContext>

  <intercept>
    <to uri="log:incoming"/>
  </intercept>

  <route>
    <from uri="jms:queue:order"/>
    <to uri="bean:validateOrder"/>
    <to uri="bean:processOrder"/>
  </route>

</camelContext>
```

OTROS EIP CON CAMEL

LOAD BALANCING

Permite distribuir la carga de mensajes entrantes a una ruta a una lista de rutas destino. Soporta diferentes métodos de distribución de carga
https://camel.apache.org/components/next/eips/loadBalance-eip.html

```
from("direct:start")
    .loadBalance().roundRobin()
        .to("seda:x")
        .to("seda:y")
        .to("seda:z")
    .end();
```

```
<route>
    <from uri="direct:start"/>
    <loadBalance>
        <roundRobinLoadBalancer/>
        <to uri="seda:x"/>
        <to uri="seda:y"/>
        <to uri="seda:z"/>
    </loadBalance>
</route>
```

BUILT-IN LOAD BALANCING POLICIES

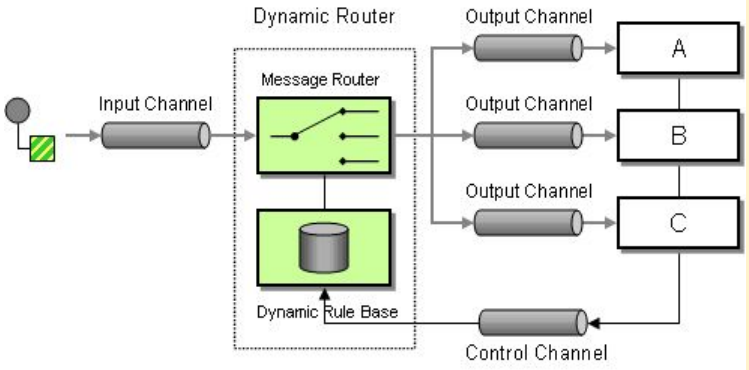
Camel provides the following policies out-of-the-box:

Policy	Description
Custom Load Balancer	To use a custom load balancer implementation.
Fail-over Load Balancer	In case of failures, the exchange will be tried on the next endpoint.
Round Robin Load Balancer	The destination endpoints are selected in a round-robin fashion. This is a well-known and classic policy, which spreads the load evenly.
Random Load Balancer	The destination endpoints are selected randomly.
Sticky Load Balancer	Sticky load balancing using an Expression to calculate a correlation key to perform the sticky load balancing.
Topic Load Balancer	Topic which sends to all destinations.
Weighted Loader Balancer	Use a weighted load distribution ratio for each server with respect to others.

OTROS EIP CON CAMEL

DYNAMIC ROUTING

Permite enrutar mensajes de manera dinámica aplicando la lógica que se requiera



OTROS EIP CON CAMEL

DYNAMIC ROUTING

Permite enrutar mensajes de manera dinámica aplicando la lógica que se requiera

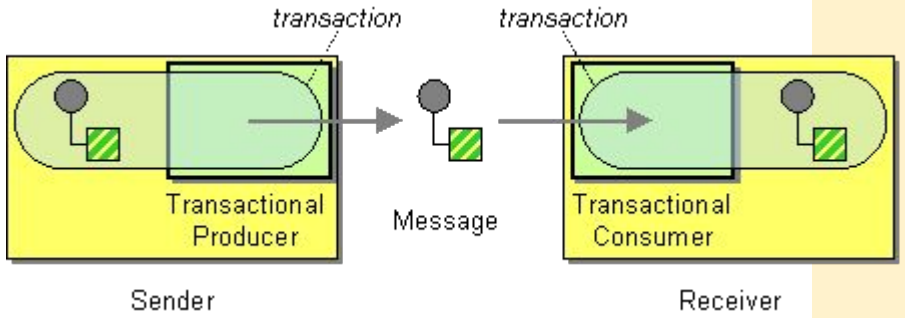
```
from("direct:start")
    .dynamicRouter(method(DynamicRouterExample.class, "nextRoute"));

// Rutas
from("direct:routeA")
    .log("Processing message in route A")
    .to("mock:resultA");

from("direct:routeB")
    .log("Processing message in route B")
    .to("mock:resultB");

from("direct:routeC")
    .log("Processing message in route C")
    .to("mock:resultC");
```

```
// Método para determinar la próxima ruta dinámicamente
public String nextRoute(String body) {
    // Lógica para determinar la próxima ruta basada en el contenido del
    if (body.contains("A")) {
        return "direct:routeA";
    } else if (body.contains("B")) {
        return "direct:routeB";
    } else if (body.contains("C")) {
        return "direct:routeC";
    }
    // Si no hay una ruta específica, finalizamos el enrutamiento
    return null;
}
```

PATRONES DE MICROSERVICIOS CON CAMEL

TRANSACTIONAL CLIENT

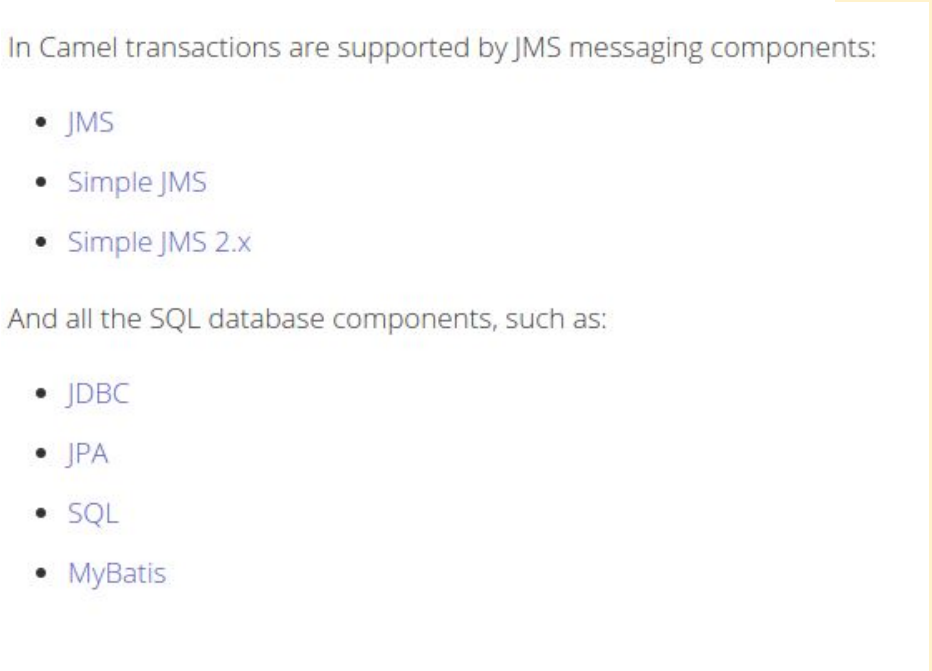
Soporte de especificación de transacciones que envuelven rutas de procesamiento con varios endpoints de procesamiento. Camel soportada las transacciones de Spring y también transacciones JTA.



PATRONES DE MICROSERVICIOS CON CAMEL

TRANSACTIONAL CLIENT

Soporte de especificación de transacciones que envuelven rutas de procesamiento con varios endpoints de procesamiento. Camel soportada las transacciones de Spring y también transacciones JTA.


In Camel transactions are supported by JMS messaging components:

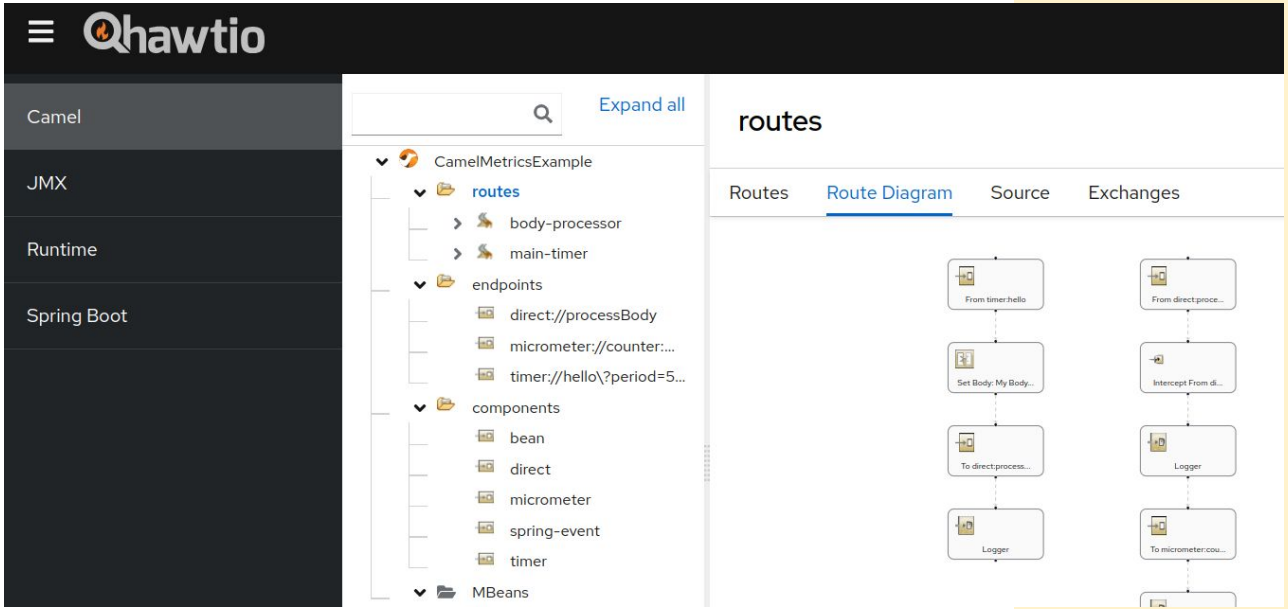
- JMS
- Simple JMS
- Simple JMS 2.x

And all the SQL database components, such as:

- JDBC
- JPA
- SQL
- MyBatis

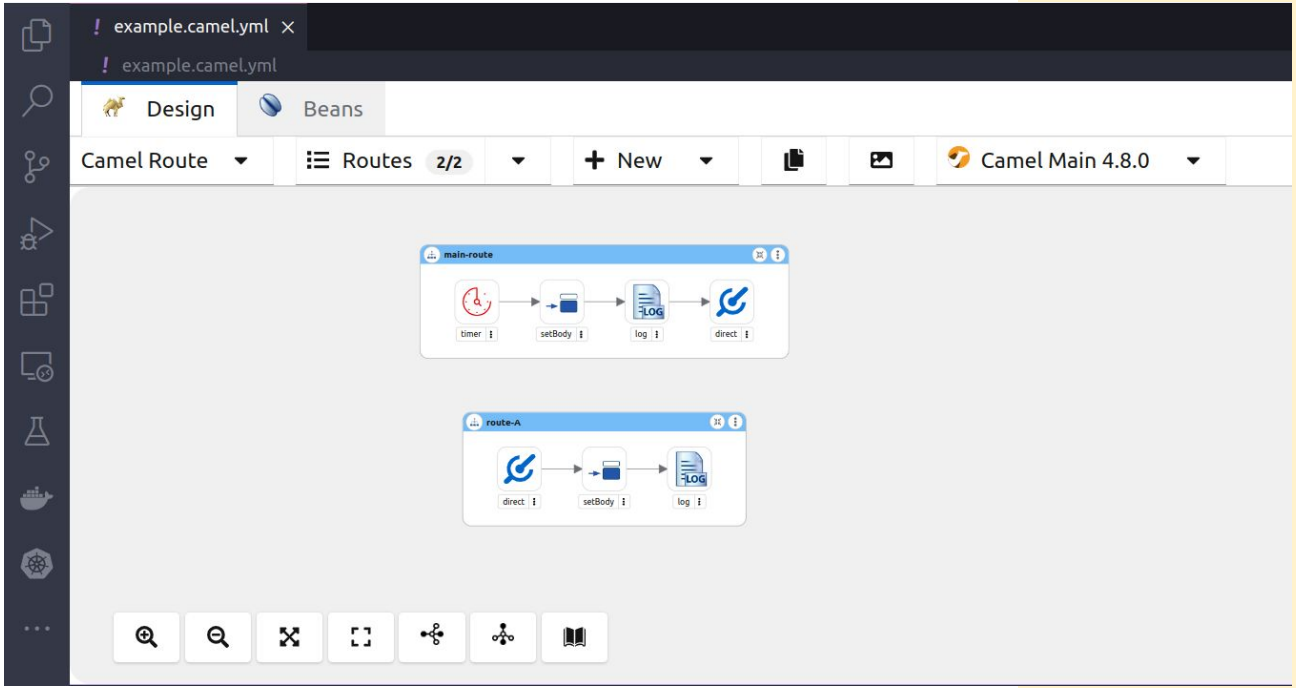
APACHE CAMEL - TOOLS

Hawtio es una interfaz web de administración y monitoreo para aplicaciones basadas en **Apache Camel**. Permite a los desarrolladores y administradores de sistemas supervisar y gestionar las rutas Camel de manera visual y sencilla. Hawtio es especialmente útil para monitorizar el estado de las rutas, revisar el rendimiento, y obtener información detallada sobre las integraciones, como los endpoints y los mensajes procesados.



APACHE CAMEL - KAOTO

Kaoto es una herramienta visual de integración diseñada para crear, diseñar y gestionar rutas **Apache Camel** de manera interactiva. Es una interfaz gráfica que permite a los desarrolladores construir flujos de integración sin necesidad de escribir código de forma manual, lo que facilita el proceso de creación de rutas complejas, especialmente para quienes no tienen experiencia en programación. Ver Kaoto



APACHE CAMEL - KAMELETS

Un **Kamelet** es una abstracción en **Apache Camel K** y **Apache Camel Kamelets** que permite crear y reutilizar bloques de procesamiento como componentes modulares, configurables y reutilizables, especialmente en entornos de Kubernetes y OpenShift.

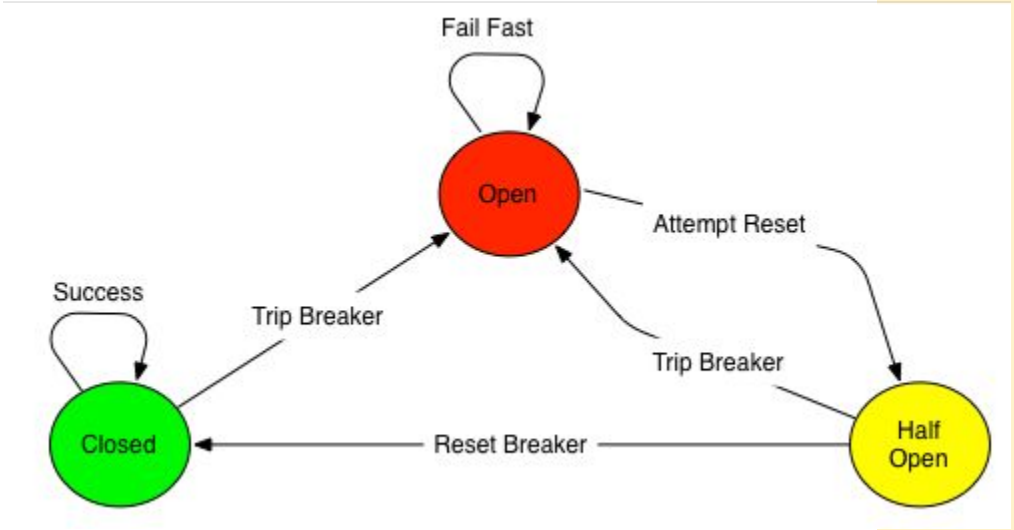
```yaml
apiVersion: camel.apache.org/v1alpha1
kind: Kamelet
metadata:
  name: active-mq-sender
spec:
  definition:
    title: "ActiveMQ Sender"
    description: "Este Kamelet envía un mensaje a una cola de ActiveMQ"
    required:
      - name
      - queue
    properties:
      name:
        title: "Mensaje"
        description: "El mensaje que se enviará"
        type: string
      queue:
        title: "Cola de destino"
        description: "La cola JMS a la que se enviará el mensaje"
        type: string
  flow:
    from:
      uri: "direct:start"
      steps:
        - to: "jms:queue:${property.queue}?username=${property.username}&password=
```

```java
from("direct:start")
    .setHeader("queue", constant("ordersQueue"))
    .setHeader("username", constant("admin"))
    .setHeader("password", constant("adminpass"))
    .to("kamelet:active-mq-sender");
```

PATRONES DE MICROSERVICIOS CON CAMEL

CIRCUIT BREAKER

Utilizado en la arquitectura de software para mejorar la resiliencia y estabilidad de un sistema distribuidor. Su principal objetivo es prevenir que fallos en una parte del sistema se propaguen y afecten a otros componentes, lo cual podría llevar a una cascada de fallos o a una degradación general del sistema. Este patrón es particularmente útil en sistemas de microservicios, donde múltiples servicios se comunican entre sí a través de redes que pueden ser poco fiables.



PATRONES DE MICROSERVICIOS CON CAMEL

CIRCUIT BREAKER

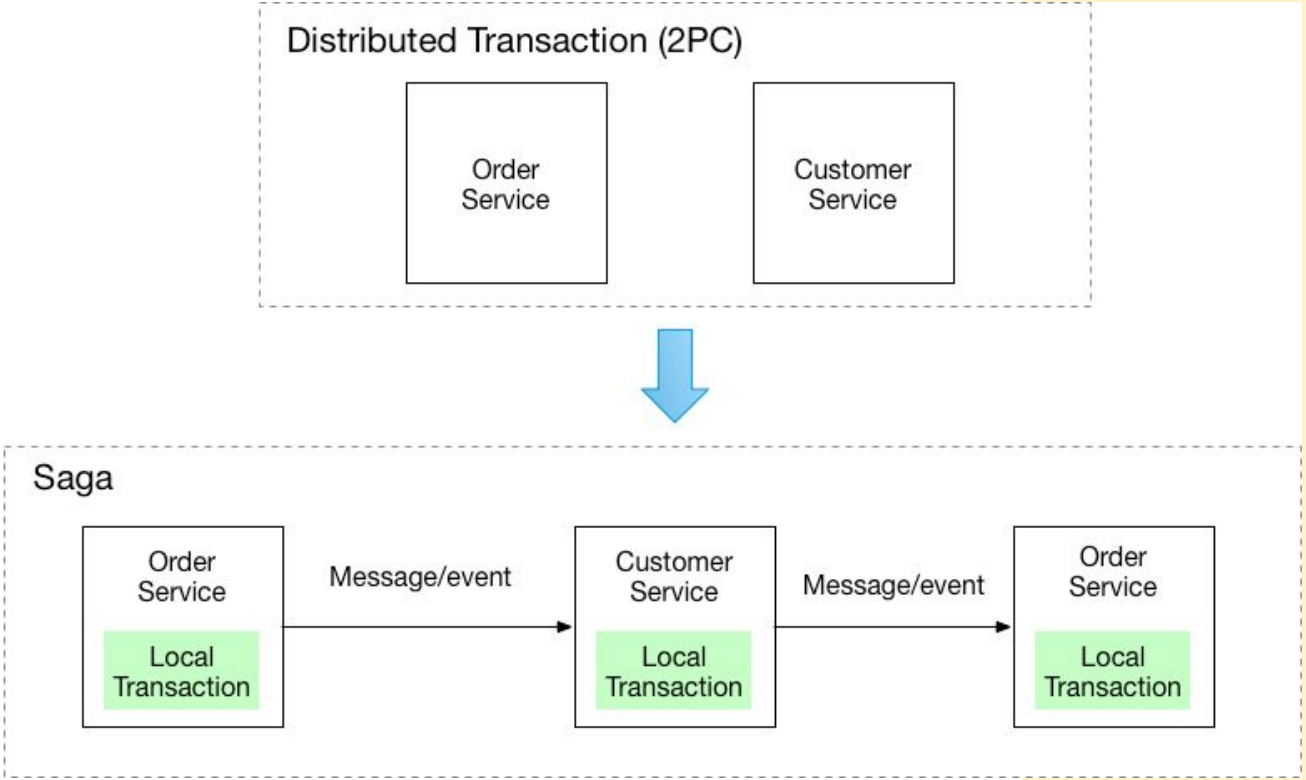
```
from("direct:start")
    .circuitBreaker()
        .to("http://fooservice.com/faulty")
    .onFallback()
        .transform().constant("Fallback message")
    .end()
    .to("mock:result");
```

```
<route>
  <from uri="direct:start"/>
  <circuitBreaker>
    <to uri="http://fooservice.com/slow"/>
    <onFallback>
      <transform>
        <constant>Fallback message</constant>
      </transform>
    </onFallback>
  </circuitBreaker>
  <to uri="mock:result"/>
</route>
```

PATRONES DE MICROSERVICIOS CON CAMEL

SAGA

Patrón usado para gestionar transacciones distribuidas



PATRONES DE MICROSERVICIOS CON CAMEL

SAGA

Patrón usado para gestionar transacciones distribuidas

```
from("direct:newOrder")
  .saga()
  .propagation(SagaPropagation.MANDATORY)
  .compensation("direct:cancelOrder")
  .completion("direct:completeOrder") // completion endpoint
    .transform().header(Exchange.SAGA_LONG_RUNNING_ACTION)
    .bean(orderManagerService, "newOrder")
    .log("Order ${body} created");

// direct:cancelOrder is the same as in the previous example

// called on successful completion
from("direct:completeOrder")
  .transform().header(Exchange.SAGA_LONG_RUNNING_ACTION)
  .bean(orderManagerService, "findExternalId")
  .to("jms:prepareOrder")
  .log("Order ${body} sent for preparation");
```

PRUEBAS UNITARIAS

Podemos usar JUnit y Camel Test para escribir nuestras pruebas unitarias
Documentación

Component	Description
camel-test-junit5	**JUnit 5**: Is an older standalone Java library letting you easily create Camel test cases using a single Java class for all your configuration and routing without.
camel-test-main-junit5	**JUnit 5**: Used for testing Camel in Camel Main mode
camel-test-spring-junit5	**JUnit 5**: Used for testing Camel with Spring / Spring Boot
camel-test-infra	**Camel Test Infra**: Camel Test Infra is a set of modules that leverage modern JUnit 5 features to abstract the provisioning and execution of test infrastructure. Among other things, it provides abstraction of the infrastructure (based on Test Containers - being the de-facto successor of the camel-testcontainers components) as well as JUnit 5 extensions for the Camel Context itself.

PRUEBAS UNITARIAS

Podemos usar JUnit y Camel Test para escribir nuestras pruebas unitarias

```java
import org.apache.camel.builder.RouteBuilder;

public class MyRouteBuilder extends RouteBuilder {
    @Override
    public void configure() throws Exception {
        from("direct:start")
            .marshal().json() // Marshalling del mensaje a JSON
            .to("rabbitmq:myExchange?routingKey=myQueue"); // Envío a RabbitMQ

        from("rabbitmq:myExchange?queue=myQueue")
            .unmarshal().json() // Unmarshalling del mensaje JSON
            .log("Received message: ${body}")
            .to("mock:end");
    }
}
```

```java
import org.apache.camel.CamelContext;
import org.apache.camel.EndpointInject;
import org.apache.camel.ProducerTemplate;
import org.apache.camel.component.mock.MockEndpoint;
import org.apache.camel.test.junit5.CamelTestSupport;
import org.junit.jupiter.api.Test;

public class MyRouteTest extends CamelTestSupport {

    // Inyectamos los endpoints necesarios para las pruebas
    @EndpointInject("mock:end")
    protected MockEndpoint mockEnd;

    @Override
    protected CamelContext createCamelContext() throws Exception {
        // Creamos el contexto de Camel y agregamos las rutas definidas
        CamelContext context = super.createCamelContext();
        context.addRoutes(new MyRouteBuilder());
        return context;
    }

    @Test
    public void testRoute() throws Exception {
        // Configuramos las expectativas en el endpoint mock
        mockEnd.expectedMessageCount(1);
        mockEnd.expectedBodiesReceived("{\"name\":\"John\",\"age\":30}");

        // Enviamos un mensaje directamente a la ruta
        template.sendBody("direct:start", "{\"name\":\"John\",\"age\":30}");

        // Verificamos que las expectativas se cumplan
        mockEnd.assertIsSatisfied();
    }
}
```

Anexo - GENERACIÓN PROYECTOS CAMEL SPRING BOOT

Haciendo uso de Maven podemos generar fácilmente un proyecto base que integre Spring Boot y Apache Camel

```
mvn archetype:generate \
  -DarchetypeGroupId=org.apache.camel.archetypes \
  -DarchetypeArtifactId=camel-archetype-spring-boot \
  -DarchetypeVersion=4.8.0
```

Pregunta 1

¿Qué es Apache Camel?

a) Un servidor de aplicaciones
b) Una herramienta de desarrollo web
c) Un framework de integración de software
d) Un sistema de gestión de bases de datos

Pregunta 2

¿Qué es un "RouteBuilder" en Apache Camel?

a) Una clase utilizada para construir rutas en Apache Camel
b) Un tipo de componente de Camel
c) Un archivo de configuración XML
d) Un patrón de diseño de software

Pregunta 3

¿Cuál de los siguientes lenguajes se puede usar para definir rutas en Apache Camel?

a) Java
b) XML
c) Groovy
d) Todos los anteriores

CUESTIONARIO

Pregunta 4

¿Qué es un "Endpoint" en Apache Camel?

a) Un punto de entrada para mensajes en una ruta Camel
b) Un tipo de componente de Camel
c) Una clase base para todos los componentes de Camel
d) Un tipo de archivo de configuración

Pregunta 5

¿Qué es un "Component" en Apache Camel?

a) Una librería de interfaz gráfica
b) Una unidad de procesamiento de datos
c) Una pieza de software que crea y configura endpoints
d) Un archivo de configuración de rutas

CUESTIONARIO

Pregunta 6

¿Cuál de los siguientes es un patrón de integración soportado por Apache Camel?

a) Message Router
b) Content Enricher
c) Splitter
d) Todos los anteriores

Pregunta 7

¿Qué función tiene el "ProducerTemplate" en Apache Camel?

a) Configurar rutas de integración
b) Enviar mensajes a endpoints desde código Java
c) Definir componentes de Camel
d) Transformar mensajes entre diferentes formatos

CUESTIONARIO

Pregunta 8

¿Qué hace el componente "camel-rabbitmq"?

a) Permite la integración con bases de datos relacionales
b) Permite la integración con servicios RESTful
c) Permite la integración con RabbitMQ para enviar y recibir mensajes
d) Permite la transformación de mensajes a formato JSON

Pregunta 9

¿Cuál de los siguientes módulos de Camel se utiliza para la transformación de mensajes a JSON?

a) camel-csv
b) camel-xmljson
c) camel-jackson
d) camel-http

CUESTIONARIO

Pregunta 10

¿Qué hace el método "marshal()" en una ruta Camel?

a) Envía mensajes a un endpoint
b) Convierte un objeto en una representación específica de datos
c) Deserializa un mensaje en un objeto
d) Filtra mensajes basados en una condición

Pregunta 11

¿Qué comando se utiliza para instalar características adicionales en Red Hat Fuse?

a) `install`
b) `features:install`
c) `add-feature`
d) `deploy`

CUESTIONARIO

Pregunta 12

¿Qué significa el término "Unmarshal" en el contexto de Apache Camel?

a) Enviar un mensaje a múltiples endpoints
b) Convertir datos de una representación específica en un objeto
c) Dividir un mensaje en partes más pequeñas
d) Fusionar varios mensajes en uno solo

Pregunta 13

¿Qué componente se utiliza para programar tareas en Apache Camel?

a) camel-sql
b) camel-jms
c) camel-quartz
d) camel-sftp

GENERACIÓN DE PROYECTOS CAMEL

Para generar proyectos base de Apache Camel nos podemos apoyar de herramientas como

1. Apache Camel Kameleon Dev: Plataforma Web que provee de manera sencilla las opciones para inicializar proyectos nuevos usando Camel. kameleon.dev

2. Maven: Haciendo uso de los Archetypes proveídos por el proyecto de Apache Camel
   Documentación Archetypes

Links de Referencia

- Apache Camel
- Patrones de Integración Empresarial
- Repo GitHub Capacitación
- GitHub Camel Examples

Gracias