Creating Microservices with Camel and Red Hat Fuse on OpenShift

Think 2019 | Session 7767

Richard Allred, Zach Gutterman

February 13, 2019

About Us

- Zach Gutterman
  - Curriculum Architect at Red Hat
  - Based in North Carolina
  - zgutterm@redhat.com
- Richard Allred
  - Curriculum Developer at Red Hat
  - Based in Virginia
  - rallred@redhat.com

Agenda

Objective: Introduce you to cloud-native Camel microservices that run on OpenShift.

- **Agile Integration**
  - What is Fuse?
  - What is Camel?
  - What are Enterprise Integration Patterns?
- **Camel Concepts**
- **Microservices with Camel**
  - Microservices
  - Rest DSL
- **Camel in the Cloud**
  - Cloud Best Practices
  - Deploying on OpenShift

Lab Prerequisites

https://github.com/zgutterm/IBMThink2019

You will need:

- Maven 3.5 or above
- JDK 1.8 or above
- Git
- Red Hat Developer Account
- OpenShift Online Instance

What is integration?

Traditional Integration

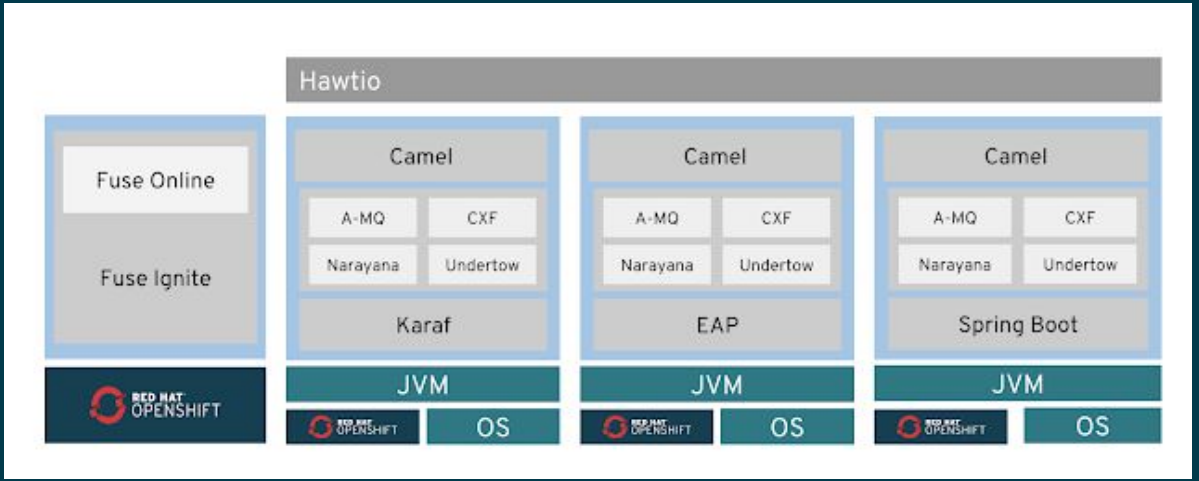
- Enterprise Service Bus (ESB)
  - Owned by the IT department
  - Each project interfaces with the ESB
  - Treated as part of the IT infrastructure.
- Works fine in an era of waterfall, but what about in an Agile era?
- ESBs create a bottleneck.

Red Hat Fuse and Agile Integration

- Red Hat Fuse is a distributed integration platform.
- Fuse embraces **agile integration**.
- Developing integration applications using agile development.
- Bridge **legacy** to the **new**
- **Pillars of Agile Integration**
  - Distributed Integration
  - Reusable, pattern based integration
  - API-first approach
  - Containers
- With Agile Integration, integration is a core framework of the application architecture.

Fuse 7 Distributions and Runtimes

- Fuse Standalone
  - Apache Karaf
  - EAP
  - Spring Boot
- Fuse on OpenShift
- Fuse Online



The core component that enables agile integration in Red Hat Fuse is...

APACHE®

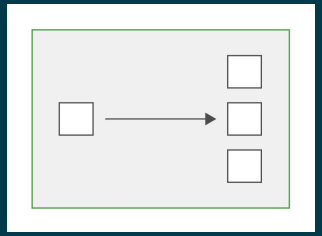
Camel

Apache Camel

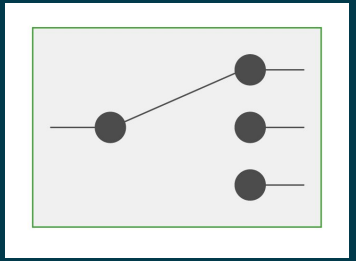
- Developed in 2007 to make simple and elegant integration solutions.
- An open source integration framework based on Enterprise Integration Patterns.
- Benefits:
  - Component library
  - Automatic type converters
  - Java and XML DSL
  - Lightweight
  - Modular (fully customizable)

Enterprise Integration Patterns

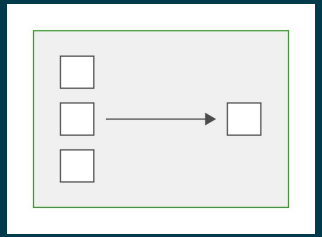
- While each integration problem is a unique challenge, the types of solutions tend to fall into some common patterns.
- These patterns are described in the 2003 book *Enterprise Integration Patterns* by Gregor Hohpe and Bobby Woolf.
- **Enterprise Integration Patterns (EIP) are proven solutions to recurring integration problems.**
- Camel provides ready-to-use implementations of most EIPs



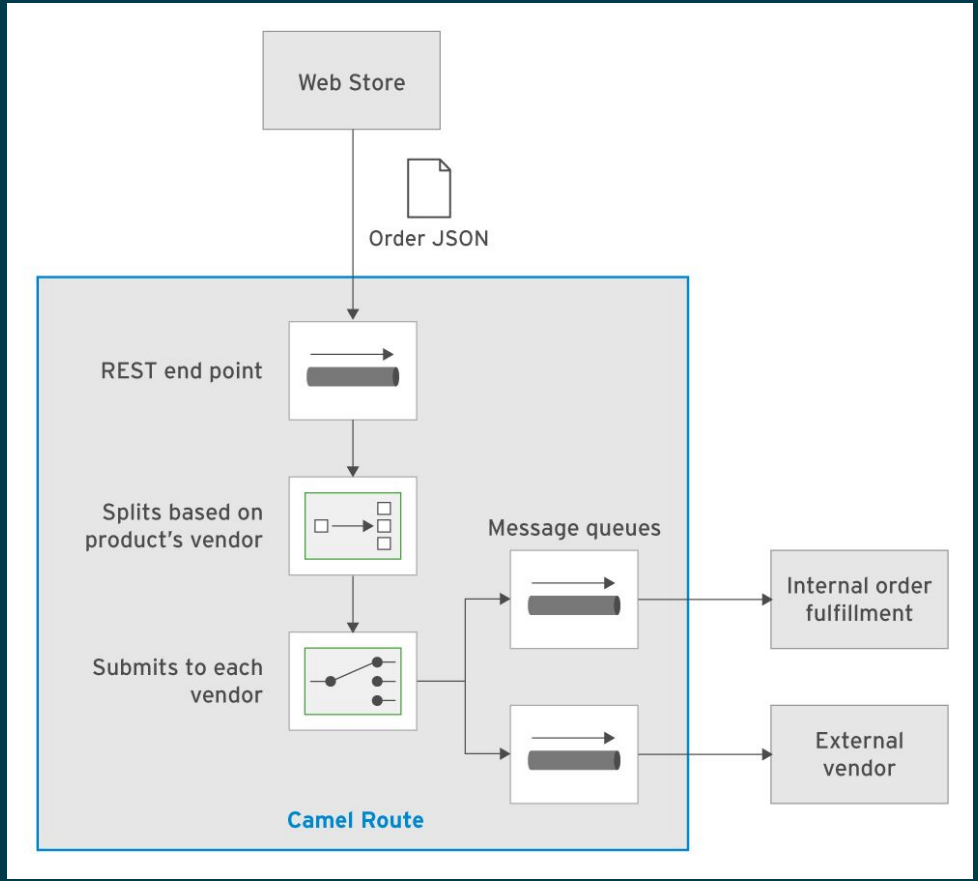
Splitter EIP



Content Based Router EIP



Aggregator EIP

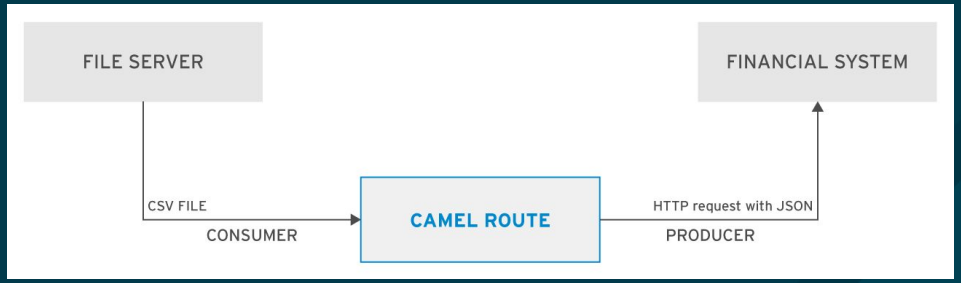

Web Store
Order JSON
REST end point
Splits based on product's vendor
Submits to each vendor
Message queues
Internal order fulfillment
External vendor
Camel Route

Camel Routes

- A **route** connects two **endpoints**, known as the **consumer** and **producer.**
- Fuse supports a Java DSL and XML DSL.
- Routes are created in Java by extending the RouteBuilder class.

```
from("file:data/inbox")
   .to("jms:queue:order");
```

```
<route>
    <from uri="file:data/inbox:/>
    <to uri="http4://service.com/endpoint"/>
<route>
```



Camel Components

- A **component** provides configuration for endpoints and encapsulates the logic required to work with different sources.
- Camel supports custom components in addition to Fuse supporting hundreds of useful components.

```
from("file:data/inbox").
    to("jms:queue:order");
```

```
<route>
    <from uri="file:data/inbox:/>
    <to uri="jms:queue:order"/>
<route>
```

CamelContext

- To implement a Camel route, the route must be attached to a CamelContext instance.
- The CamelContext loads all resources like **components** and **processors (things between the endpoints)** in the runtime system to execute the routes.
- To run routes with the CamelContext:
  - Instantiate a new context
  - Add routes to the context
  - Start the context

```
<camelContext id="example" … >
  <route>
    <from uri="file:orders/incoming"/>
    <to uri="file:orders/outgoing"/>
  </route>
</camelContext>
```

Processors

- The processor interface has a single method: **`public void process(Exchange exchange)`**
- This gives you full access to the message exchange and is a clean way of keeping routes simple and readable.
- Custom processors are often used to make changes to the headers or the payloads.

```
public class VendorProcessor implements Processor {
    @Override
    public void process(Exchange exchange) throws Exception {

        String vendorJSON = exchange.getIn().getBody(String.class);

        ...

        String vendorName = vendorVo.get("name").toString();

        exchange.getIn().setHeader("vendor_name", vendorName);
    }
}
```

Microservices

MICROSERVICES

What are they?

- Small
  - Modeled around a single business problem or “domain”
- Self-contained
  - Implement their own business logic, often has their own persistent storage
- Loosely coupled
  - Have an individually published contract or API which they use for intra-service communication
- Independently managed and deployed
  - Are easily replaced or upgraded

MICROSERVICES

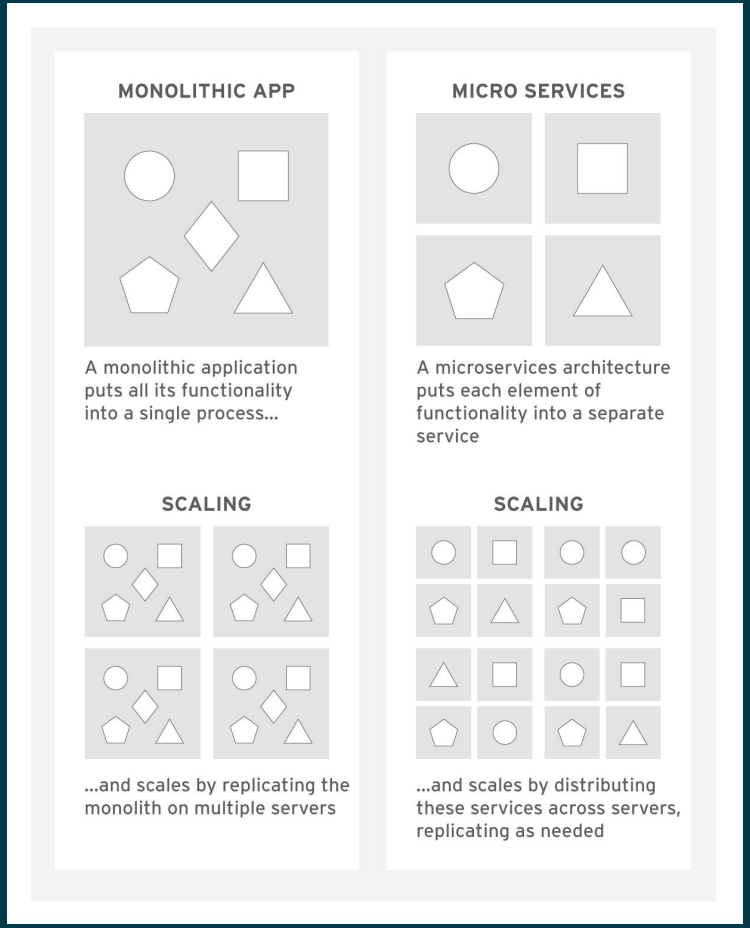
Why are they beneficial?

- Faster development time
- Easier to fix and maintain
- Developed by a small team
- Easier to scale

MICROSERVICES

What are the principles of good microservice development?

- Reorgazine to DevOps
- Packaging the Microservice as a Container
- Use an elastic infrastructure
- Infrastructure automation via CI/CD
- Deploy Independently on a Lightweight Runtime
- Design for Failure


MONOLITHIC APP
A monolithic application puts all its functionality into a single process...
SCALING
...and scales by replicating the monolith on multiple servers
MICRO SERVICES
A microservices architecture puts each element of functionality into a separate service
SCALING
...and scales by distributing these services across servers, replicating as needed

Camel REST DSL

Creating REST Services

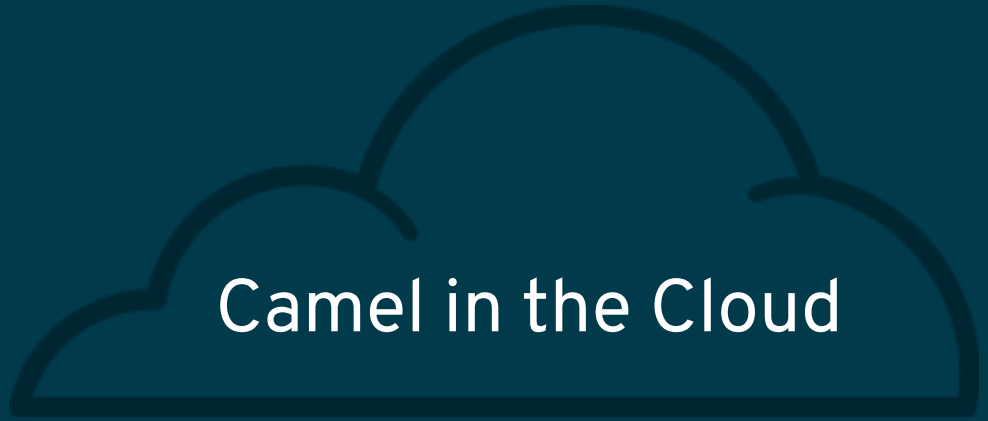
- Create REST Services as Camel routes.
- Supports multiple REST DSL capable components, such as:
  - camel-spark-rest
  - camel-jetty
  - camel-restlet
  - camel-undertow

```
public class OrderRoute extends RouteBuilder {

    public void configure() throws Exception {

      restConfiguration()
         .component("spark-rest").port(8080);

      rest("/orders")
         .get("{id}")
            .to("bean:orderService?method=get(${header.id})")
         .post()
            .to("bean:orderService?method=create")
         .put()
            .to("bean:orderService?method=update")
         .delete("{id}")
            .to("bean:orderService?method=cancel(${header.id})");
}
```

Consuming REST Services

- Consume REST services or other HTTP-based resources using the **http4** component.

```
from("direct:start")
    .to("http4://example.com?order=123&detail=short");
```



Camel in the Cloud

Health Checks

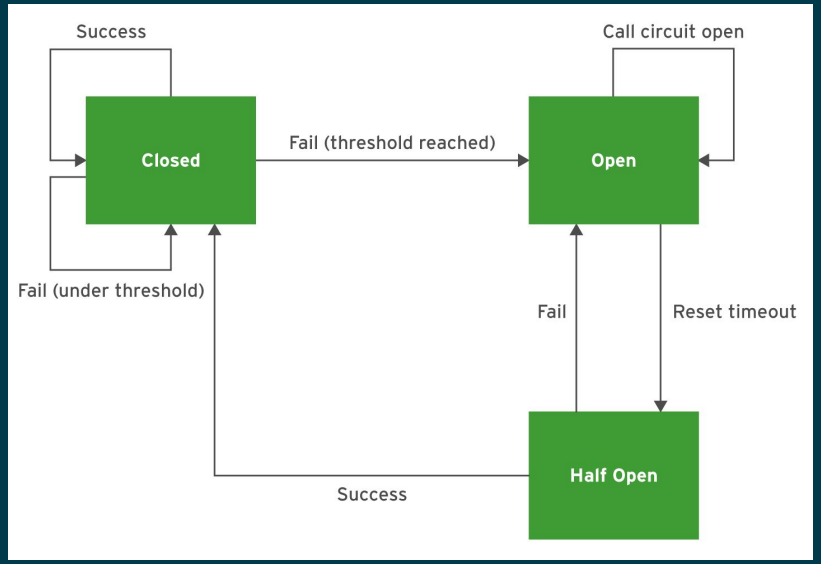
- Distributed architectures complicate service monitoring because you must keep track of **many more deployments** than a traditional architecture.
- Health checks are critical for microservice architectures, especially in platforms like OpenShift, to ensure that the service (and container) is functioning properly.
- The Spring Boot actuator is a simple add-on package for automatically configuring health checks.
- You can also use the actuator to create your own health checks.

Resilient Microservices

- Distributed systems often encounter unexpected failures of various components.
- You cannot predict every problem in your microservices.
- Therefore, you must design your microservices as resilient and fault-tolerant.
- Fault tolerant applications are able to operate at a certain degree of satisfaction despite failures that appear.
- Common fault tolerance techniques:
  - Retry pattern
  - Circuit breaker pattern
  - Fallback pattern

Circuit Breaker Pattern

- Hystrix is a latency and fault tolerance library designed to isolate points of access to remote systems and stop cascading failure in complex distributed systems.
- Any failure that occurs inside the Hystrix block will be monitored and acted on by the circuit breaker.




```
public void configure() throws Exception {
  from("direct:start")
    .hystrix()
      .to("bean:counter")
    .end()
    .log("After calling counter service: ${body}");
}
```

Circuit Breaker Fallback

- You can use `onFallback()` to dictate the route behavior when the circuit is open.

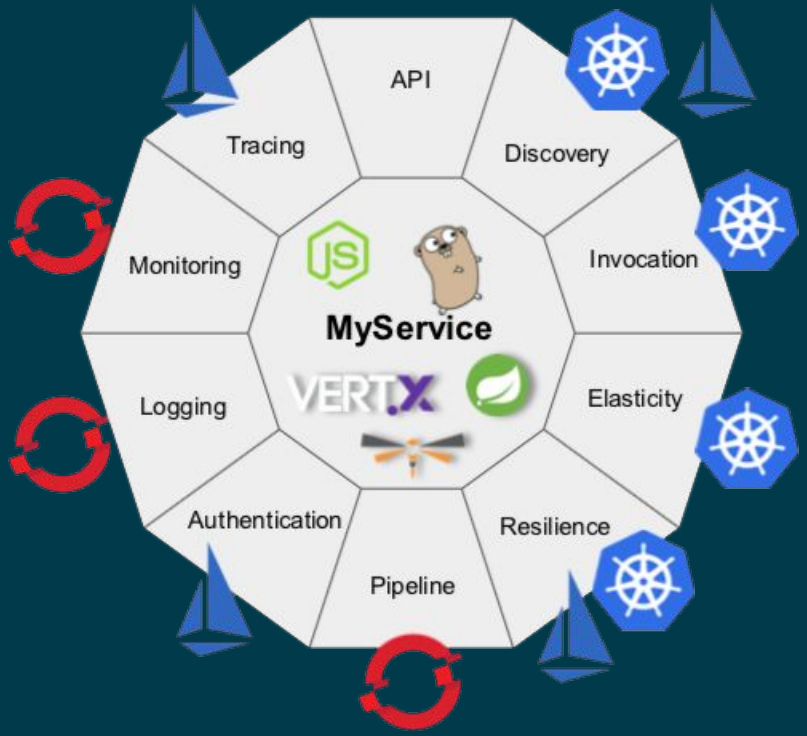
```
public void configure() throws
Exception {
  from("direct:start")
    .hystrix()
      .to("bean:counter")
    .onFallback()
      .log(“Circuit is open”)
    .end()
    .log("After calling counter
service: ${body}");
}
```

OpenShift Container Platform

Introduction

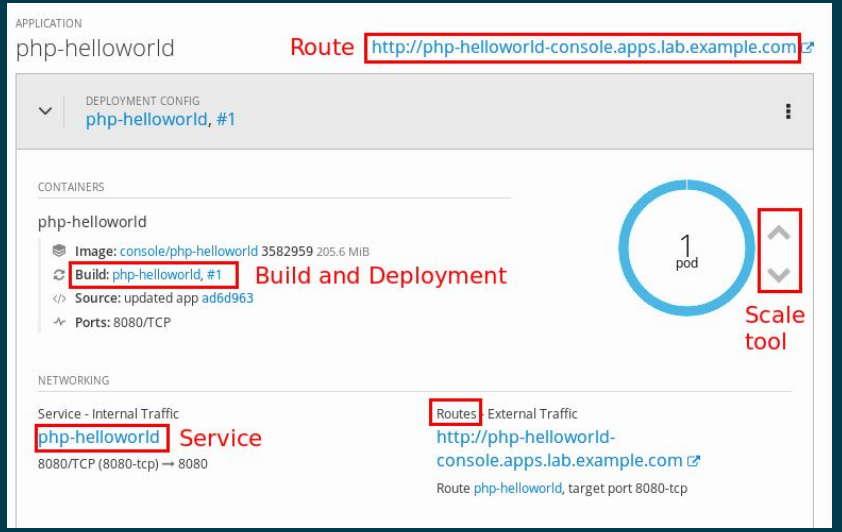
- OpenShift Container Platform allows you to create and manage containers.
- OpenShift Container Platform brings together Docker and Kubernetes, and provides an API to manage these services.
- Containers are standalone processes that run within their own environment, independent of operating system and the underlying infrastructure. OpenShift helps you to develop, deploy, and manage container-based applications.
- OpenShift provides you with a self-service platform to create, modify, and deploy applications on demand, thus enabling faster development and release life cycles.
- Think of images as cookie cutters and containers as the actual cookies.
- Think of OpenShift as an operating system, images as applications that you run on them, and the containers as the actual running instances of those applications.

OpenShift Container Platform



Fuse on OpenShift

- Fuse on OpenShift provides the necessary runtimes for running Camel applications in OpenShift.
- Applications can be deployed using the **fabric8** plugin for quick and easy deployment to an OpenShift cluster.
- **Fabric8** also simplifies creating simple health checks (liveness/readiness probes).
- Once deployed, you can use the OpenShift UI to scale, monitor, and manage your pods.



Exercises

https://github.com/zgutterm/IBMThink2019

redhat.

THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHat

youtube.com/user/RedHatVideos