

A group of riders on camels racing across a desert under a clear sky. The camels are in full gallop, kicking up dust. The riders are wearing helmets and traditional racing gear.

Camel microservices with Spring Boot and Kubernetes

Claus Ibsen
@davsclaus

Milan JUG
October 2018

First time in Milan

(Was in Novarelo earlier this year)



Riding the Camel

(Was in Novarello earlier this year)



About me

- Senior Principal Software Engineer at Red Hat
- 10 years as Apache Camel committer
- Author of Camel in Action books
- Based in Denmark



Blog:	http://www.davsclaus.com
Twitter:	@davsclaus
Linkedin:	davsclaus
Facebook:	Claus Ibsen

System Integration



Figure 1.1 Camel is the glue between disparate systems.

Apache Camel

is an

Integration Framework

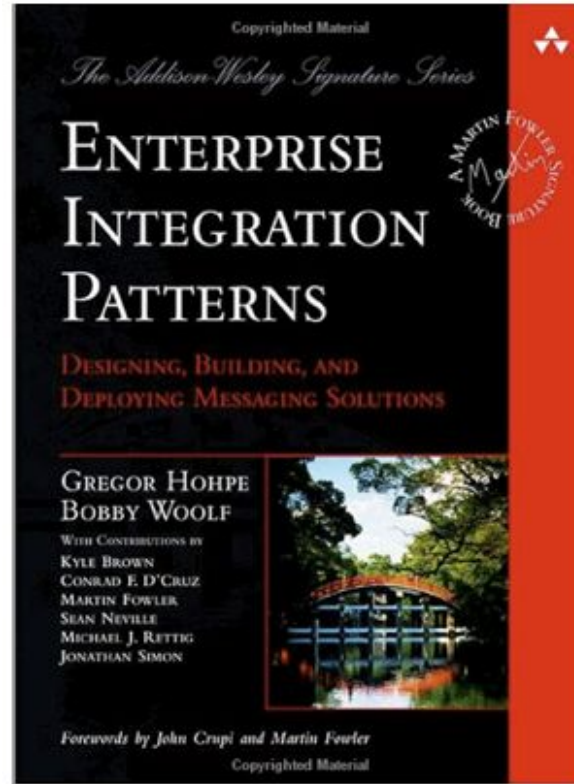
based on

Enterprise Integration Patterns

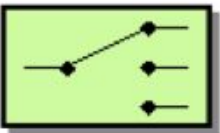
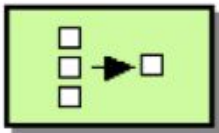

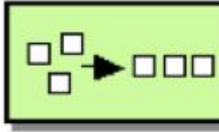
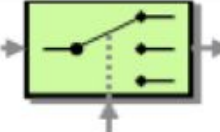
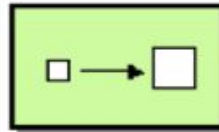
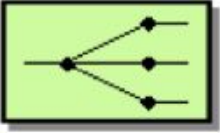
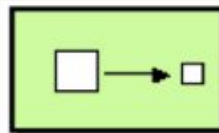
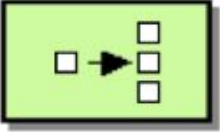
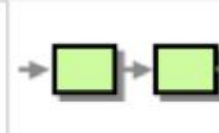
Integration Framework



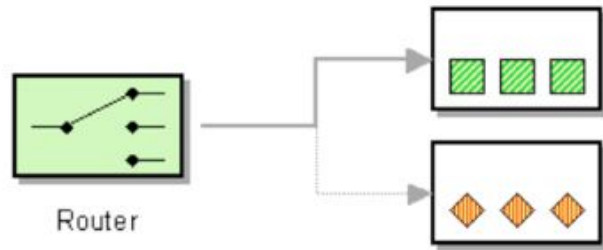
Enterprise Integration Patterns



Enterprise Integration Patterns

	Content Based Router		Aggregator
	Message Filter		Resequencer
	Dynamic Router		Content Enricher
	Recipient List		Content Filter
	Splitter		Pipes and Filters

Camel Routes



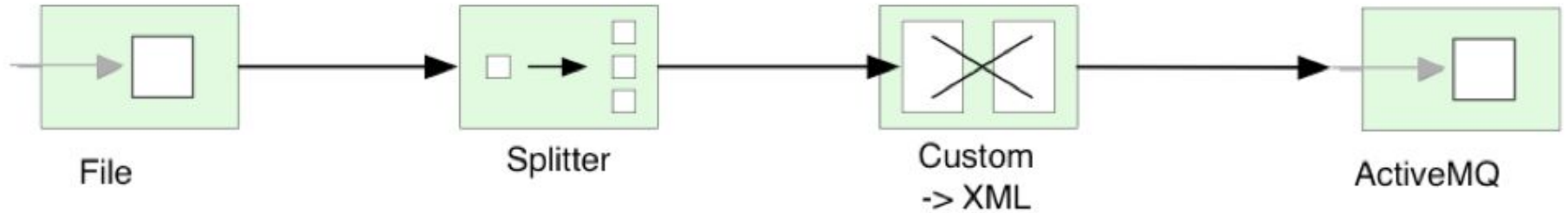
```
from("file:data/inbox")  
    .to("jms:queue:order");
```

Java DSL

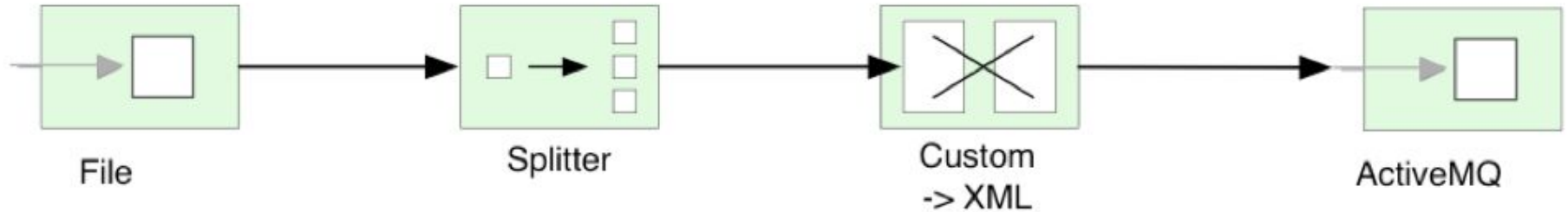
XML DSL

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

Camel Routes with Splitter

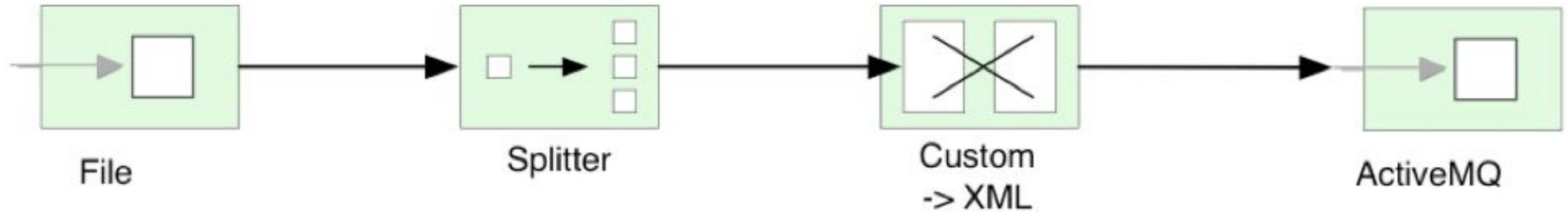


Camel Routes with Splitter



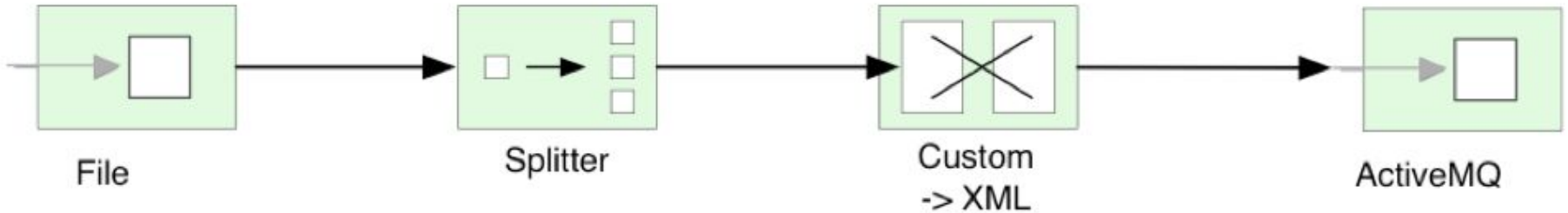
```
from("file:inbox")
```

Camel Routes with Splitter



```
from("file:inbox")  
    .split(body().tokenize("\n"))
```

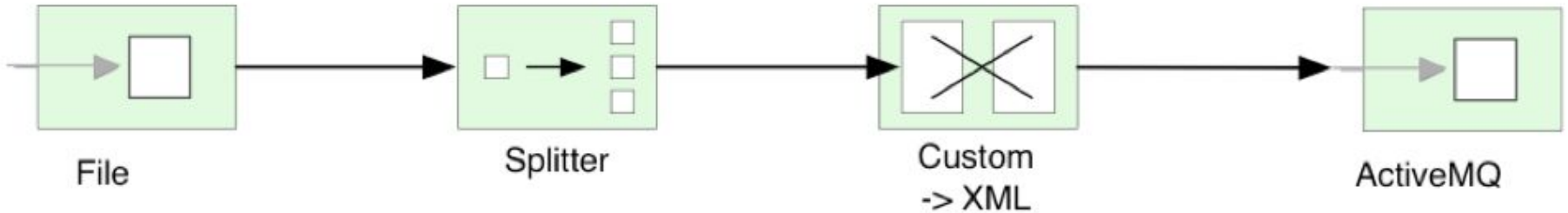
Camel Routes with Splitter



```
from("file:inbox")  
    .split(body().tokenize("\n"))  
    .marshal(customToXml)
```

Custom data
transformation

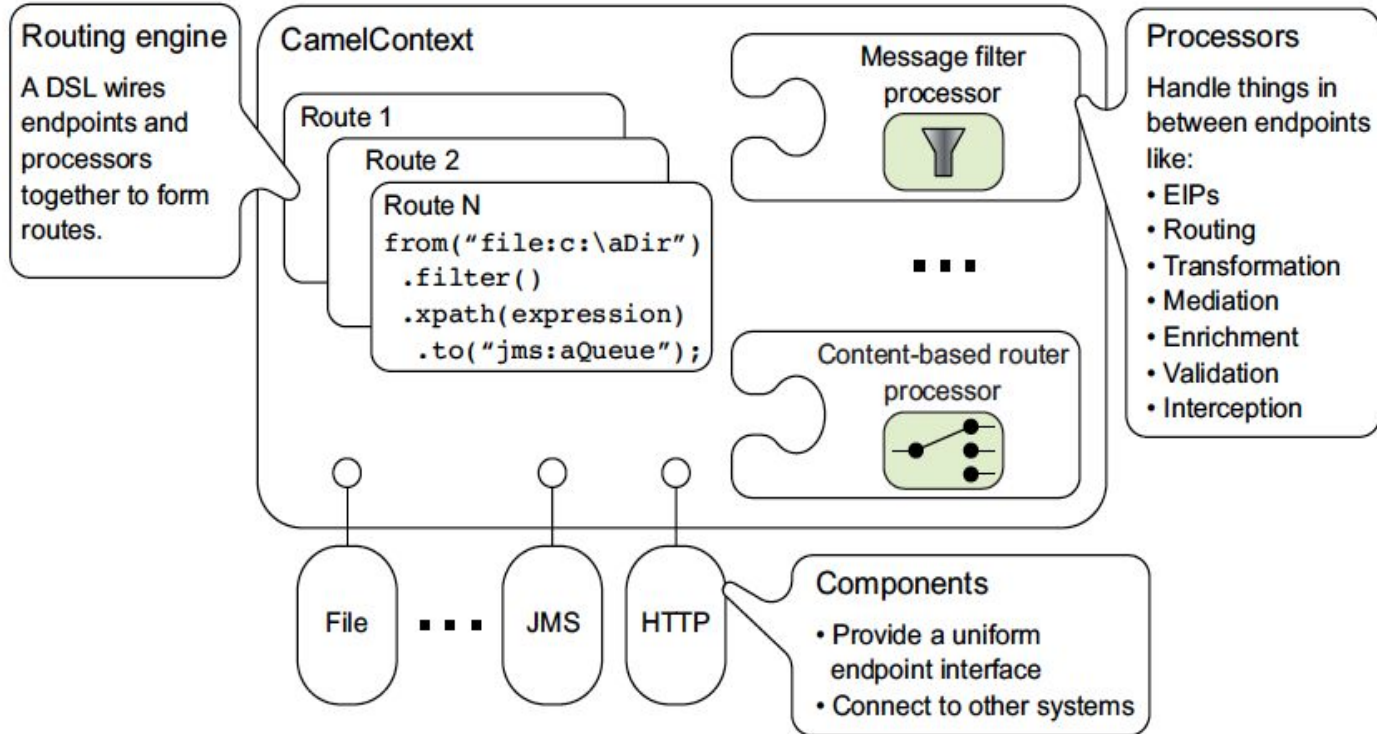
Camel Routes with Splitter



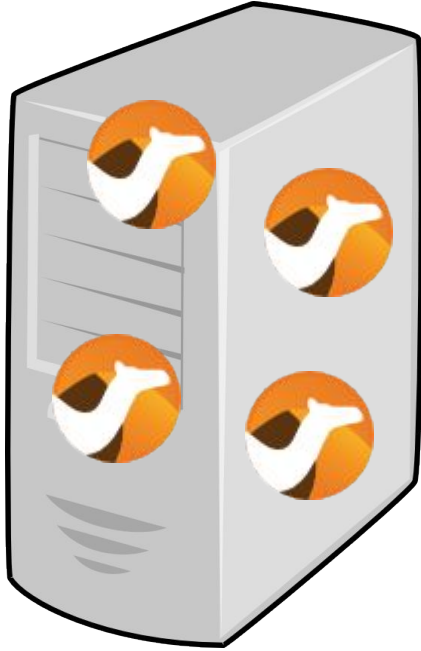
```
from("file:inbox")
    .split(body().tokenize("\n"))
    .marshal(customToXml)
    .to("activemq:line");
```

Custom data
transformation

Camel Architecture



Camel runs everywhere



Application
Servers

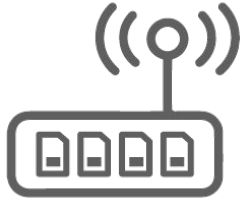


Linux
Containers

Camel connects everything



**Enterprise
Systems**



IoT

- File
- FTP
- JMS
- AMQP
- JDBC
- SQL
- TCP/UDP
- Mail
- HDFS
- JPA
- MongoDB
- Kafka
- ...

- CoAP
- MQTT
- PubNub

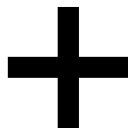
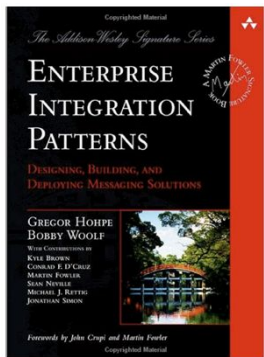


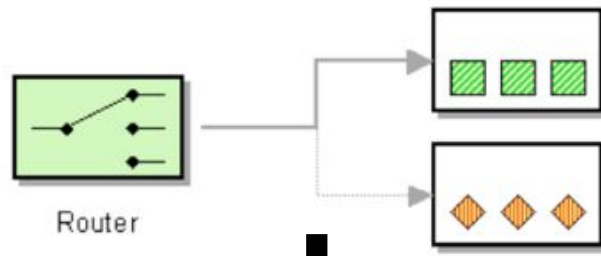
- AWS
 - S3
 - SQS
 - Kinesis
 - ...
- Google
 - BigQuery
 - PubSub
- Azure
 - Blob
 - Queue



- Box
- Dropbox
- Facebook
- LinkedIn
- Salesforce
- SAP
- ServiceNow

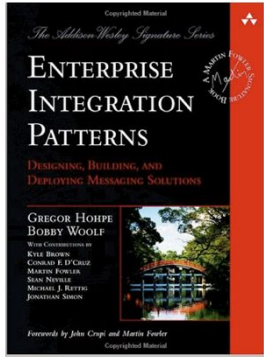


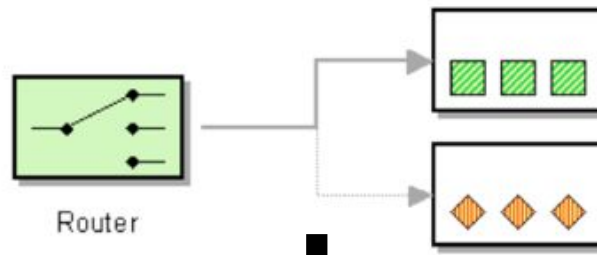




+

+



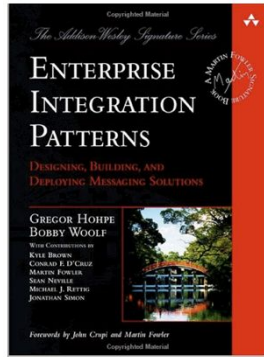
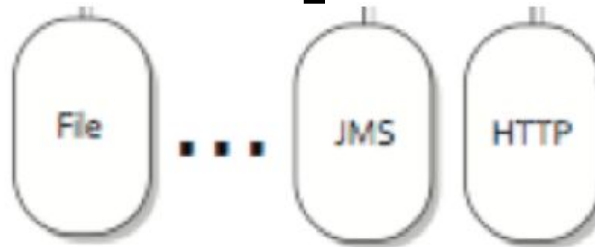


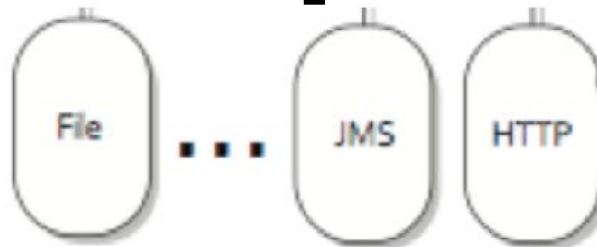
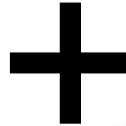
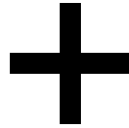
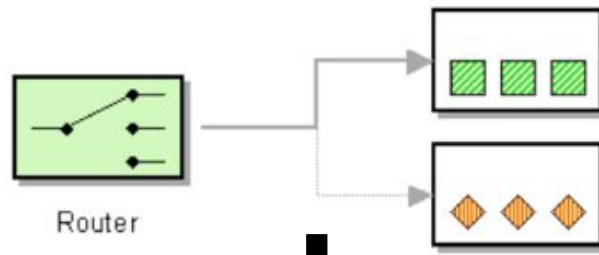
+

+



+

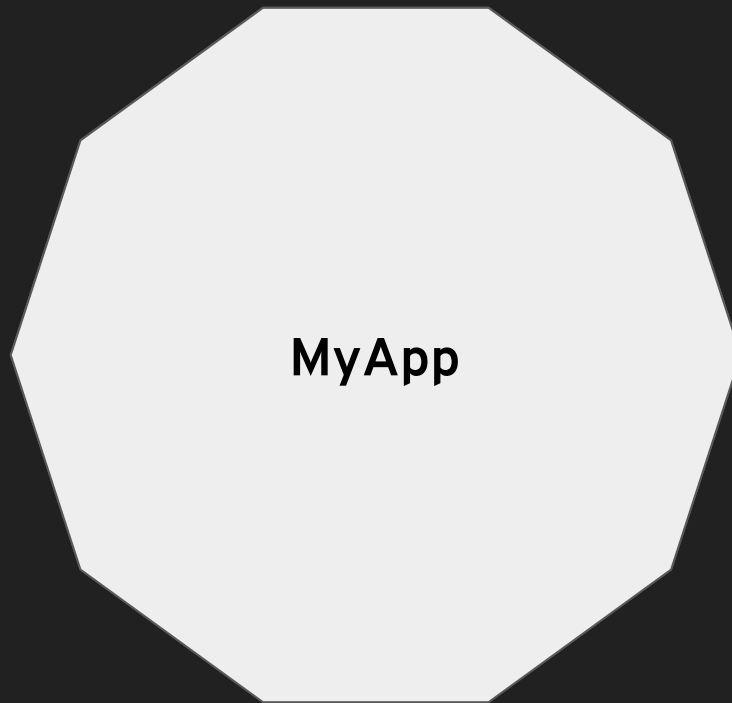




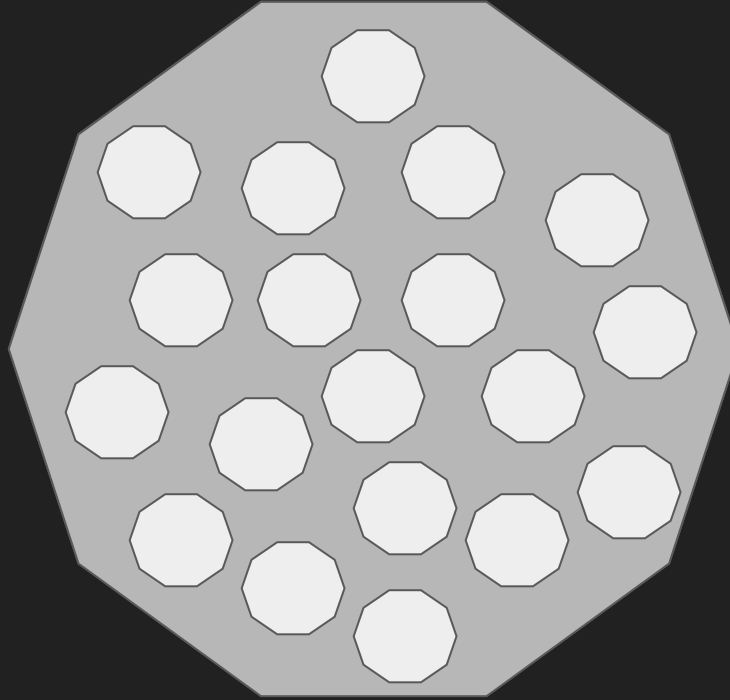


**What about Camel
in the Cloud?**

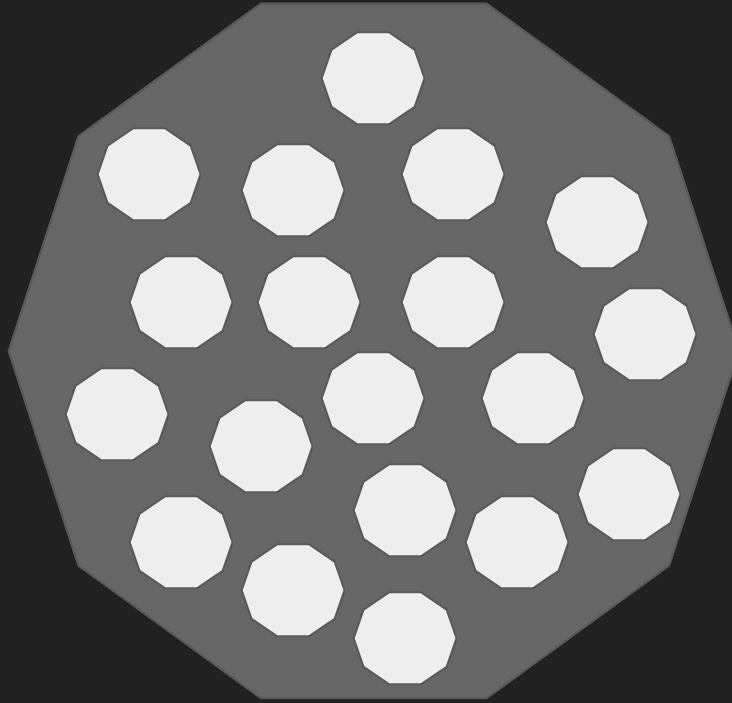
Monolith



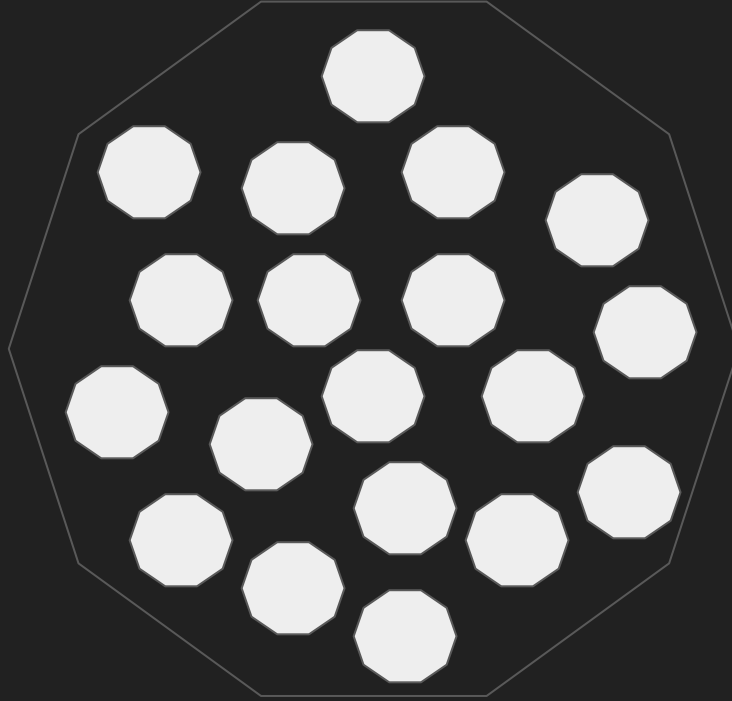
Microservices



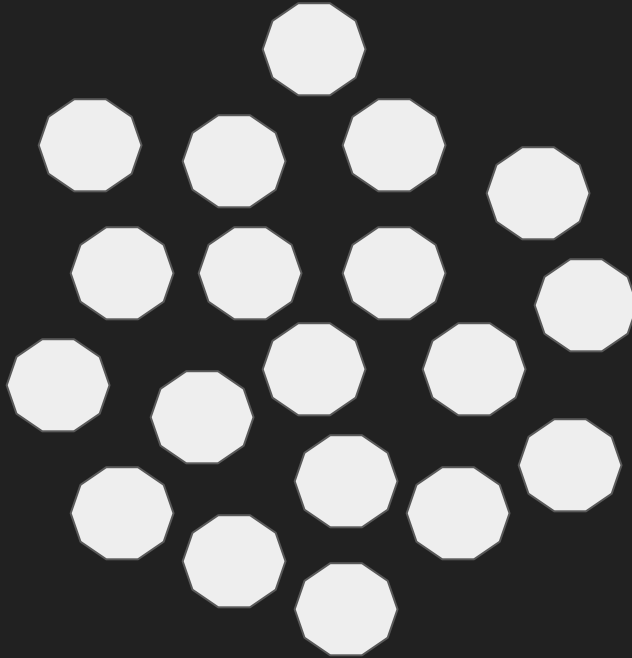
Microservices



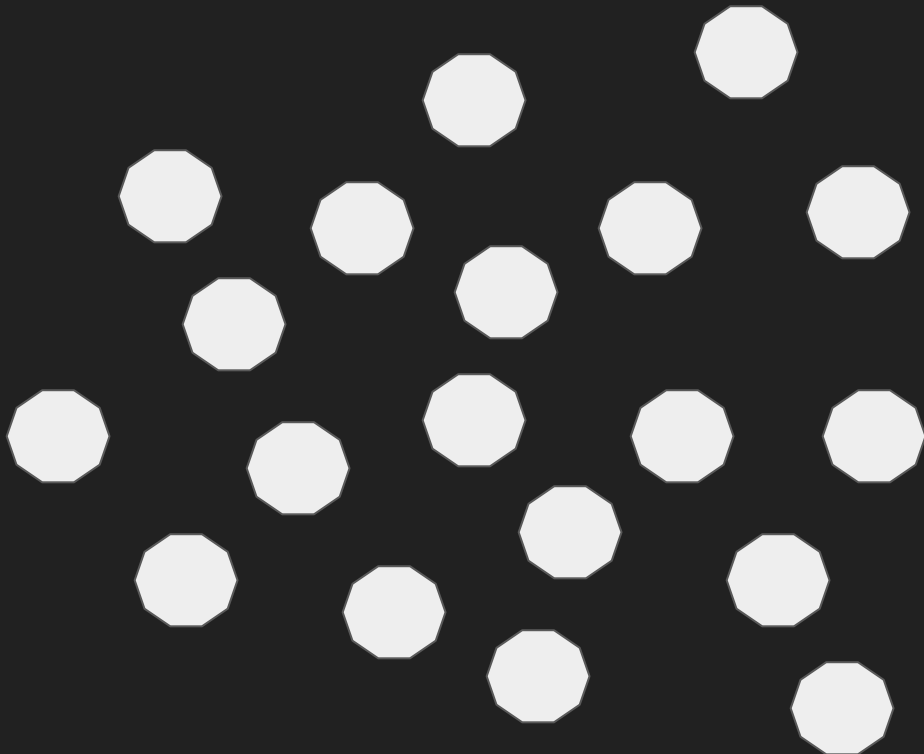
Microservices



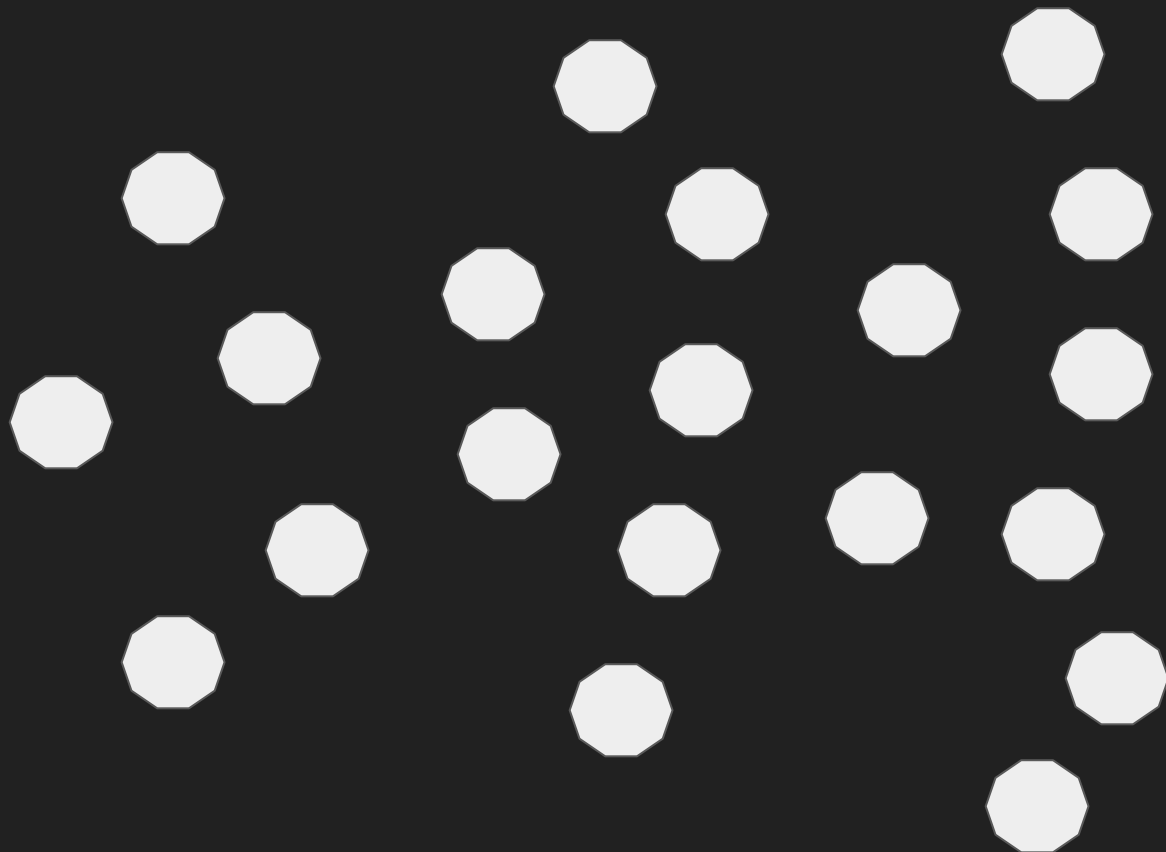
Microservices



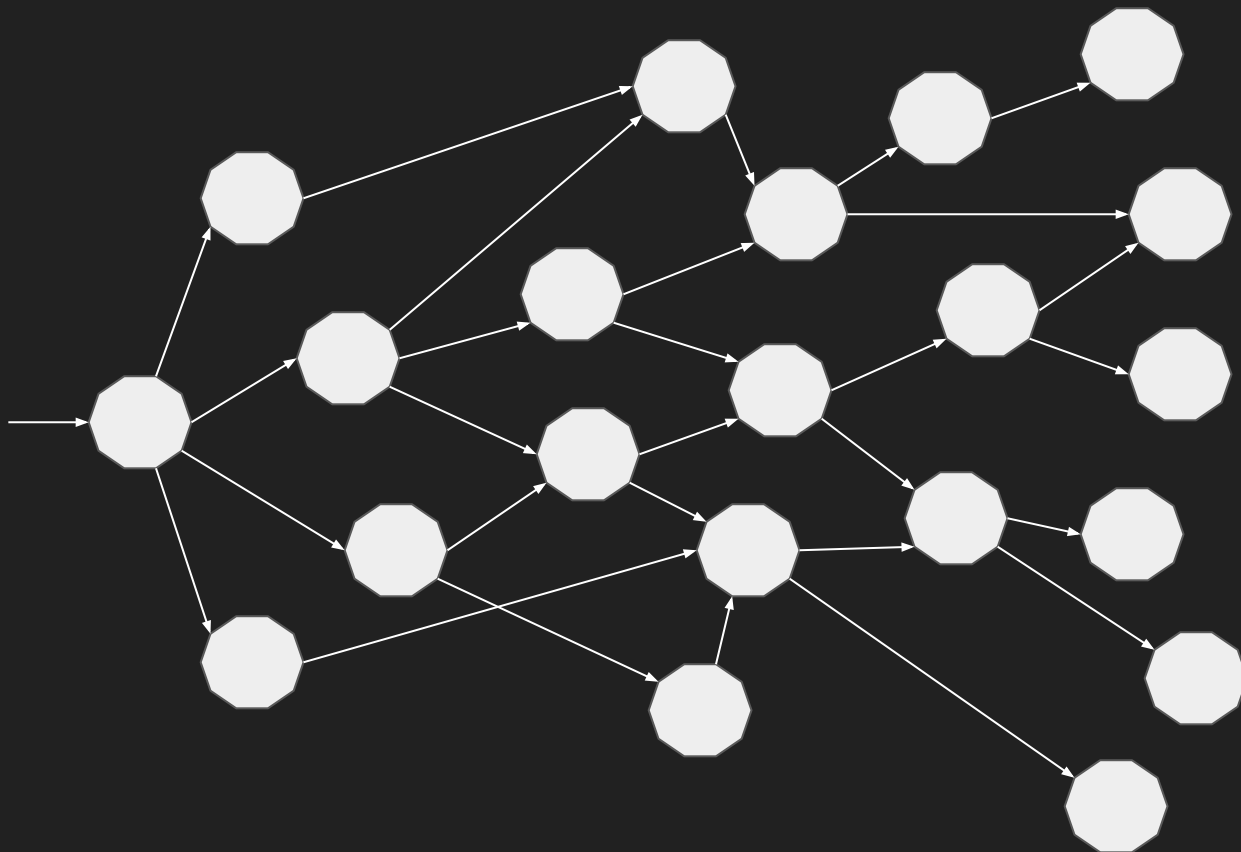
Microservices



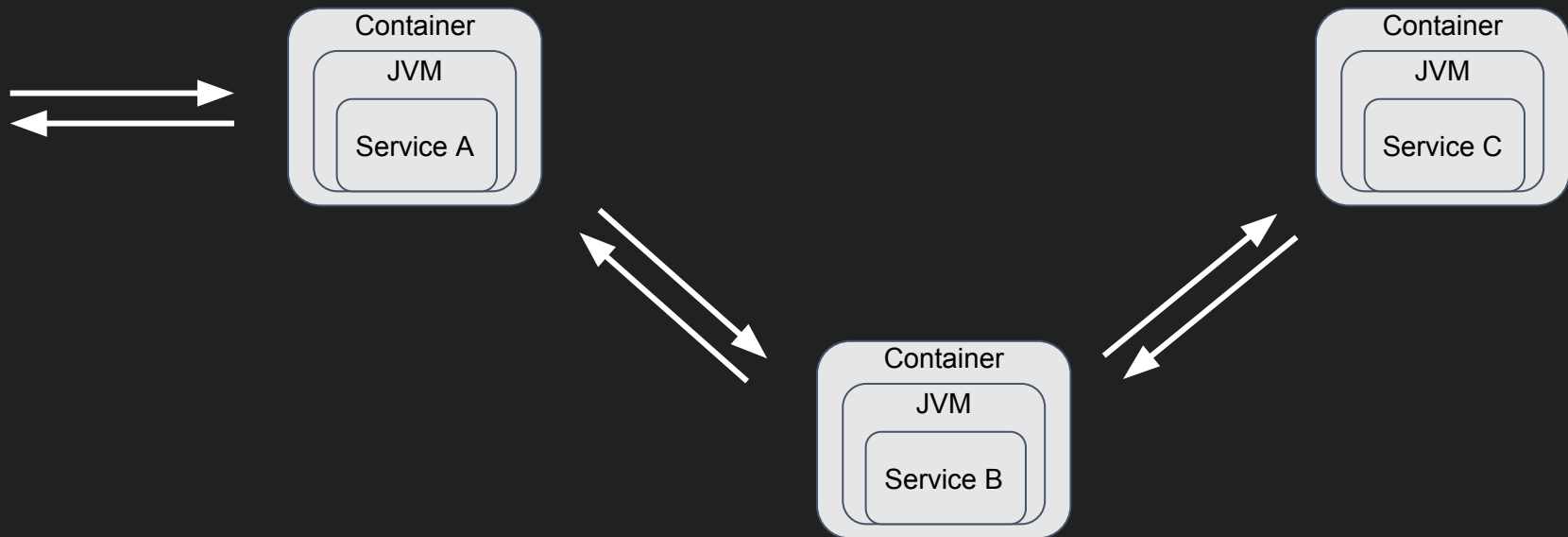
Microservices



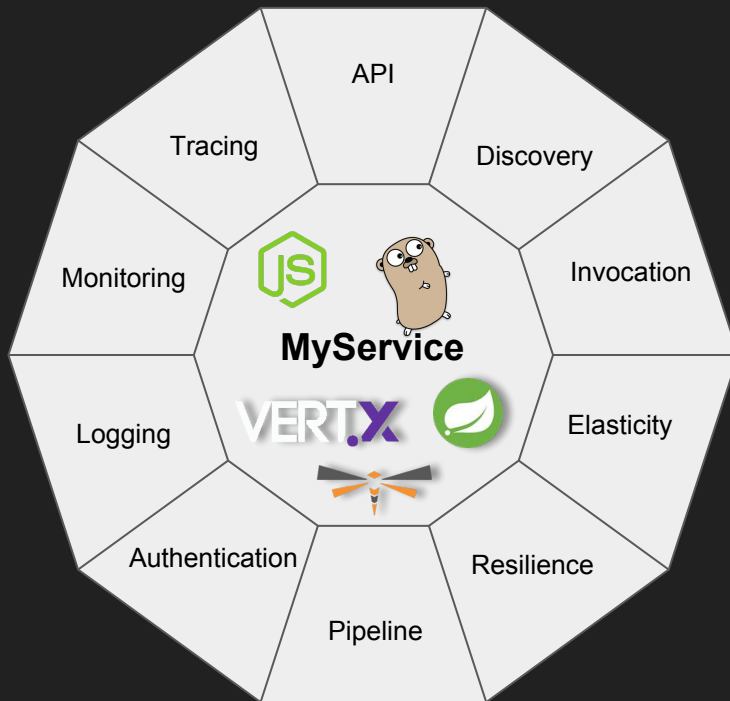
Network of Services



Microservices == Distributed Computing

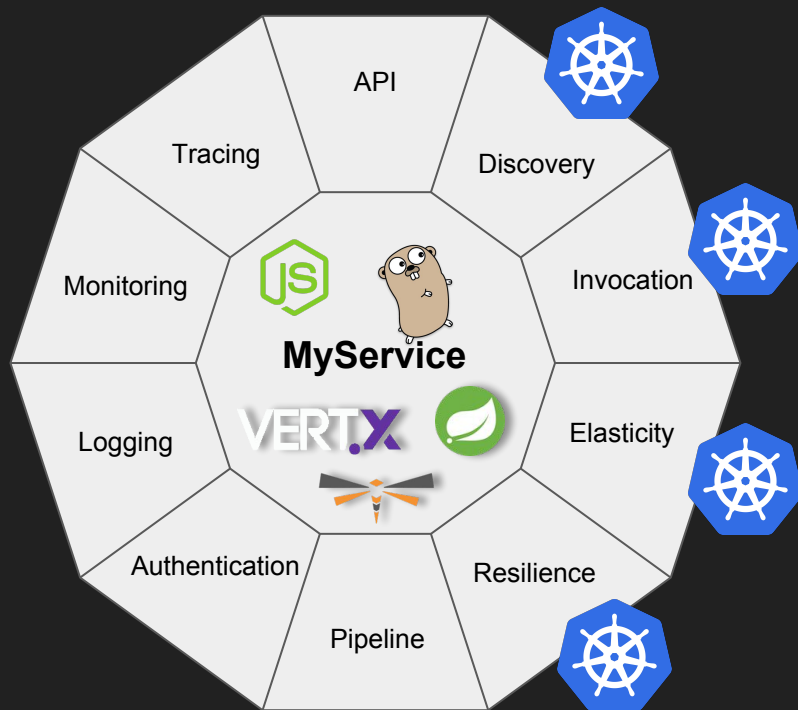


Microservices'ilities

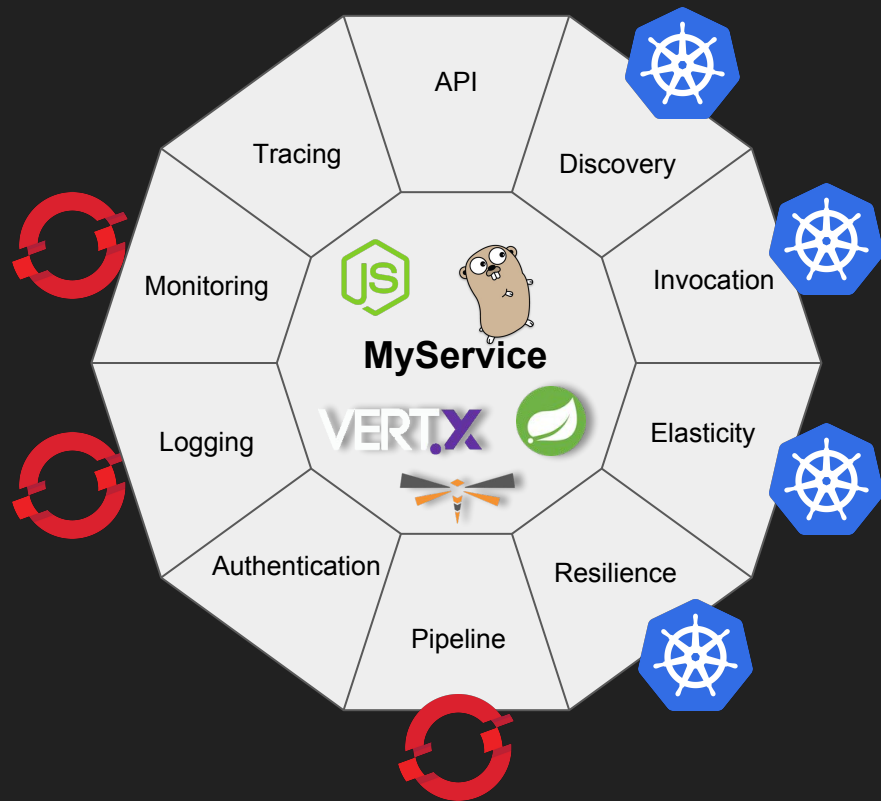




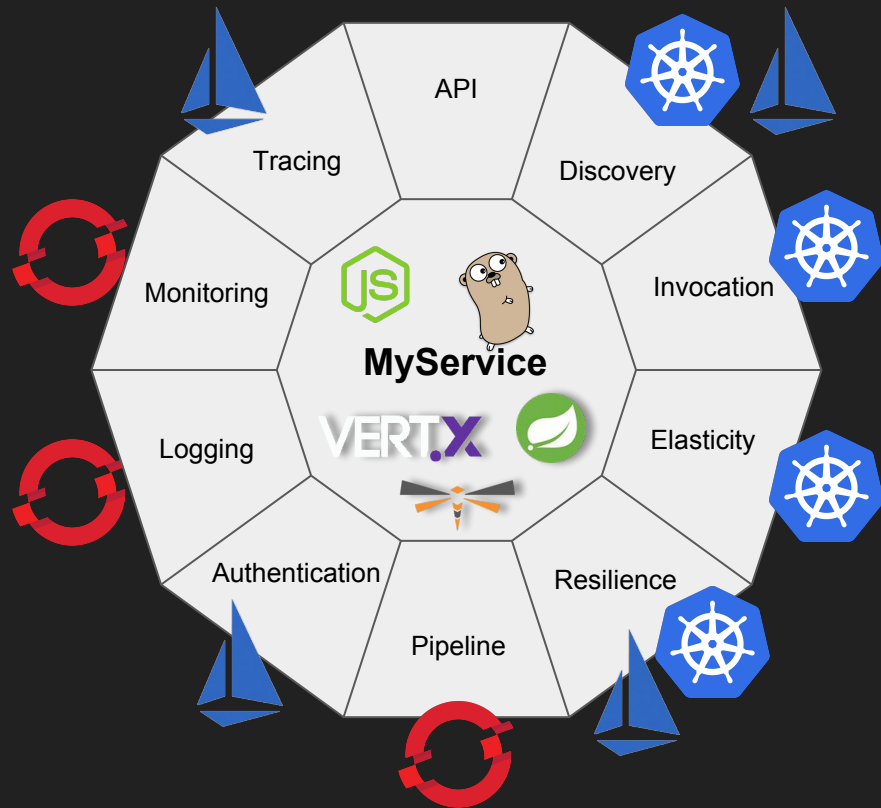
Microservices'ilities + Kubernetes



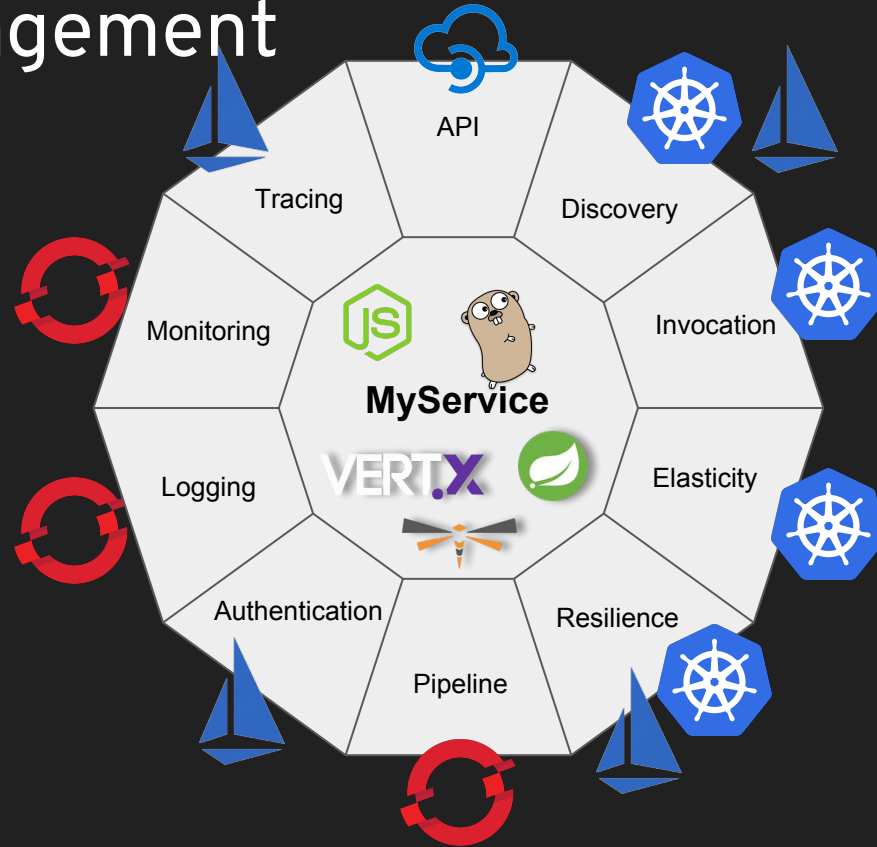
Microservices'ilities + PaaS



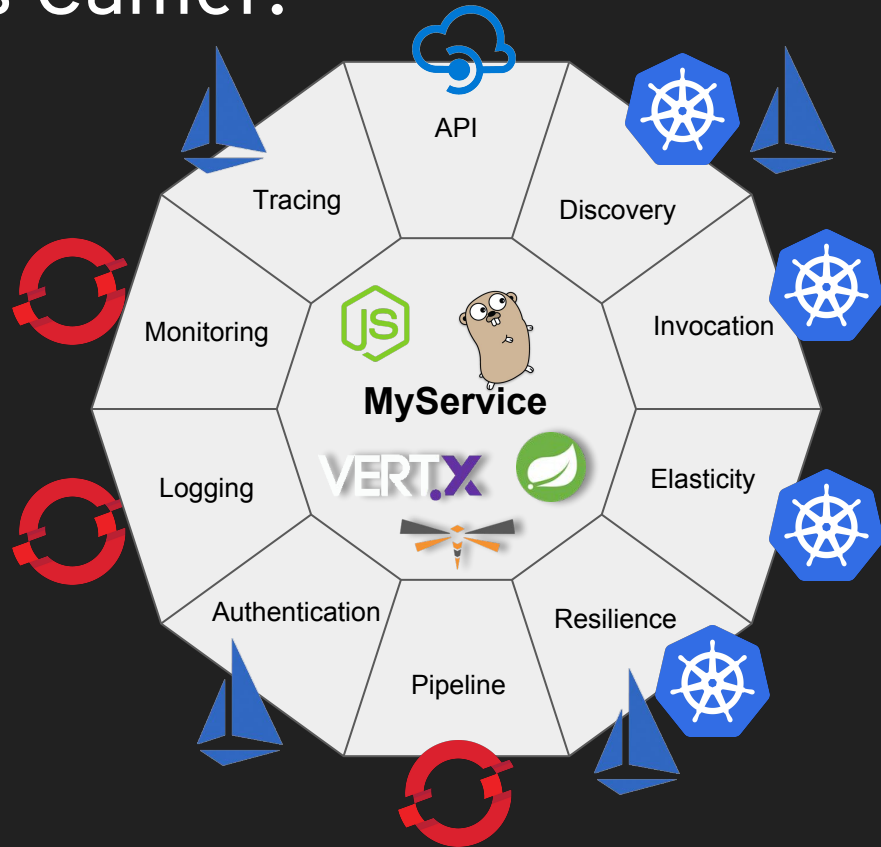
Microservices'ilities + Istio



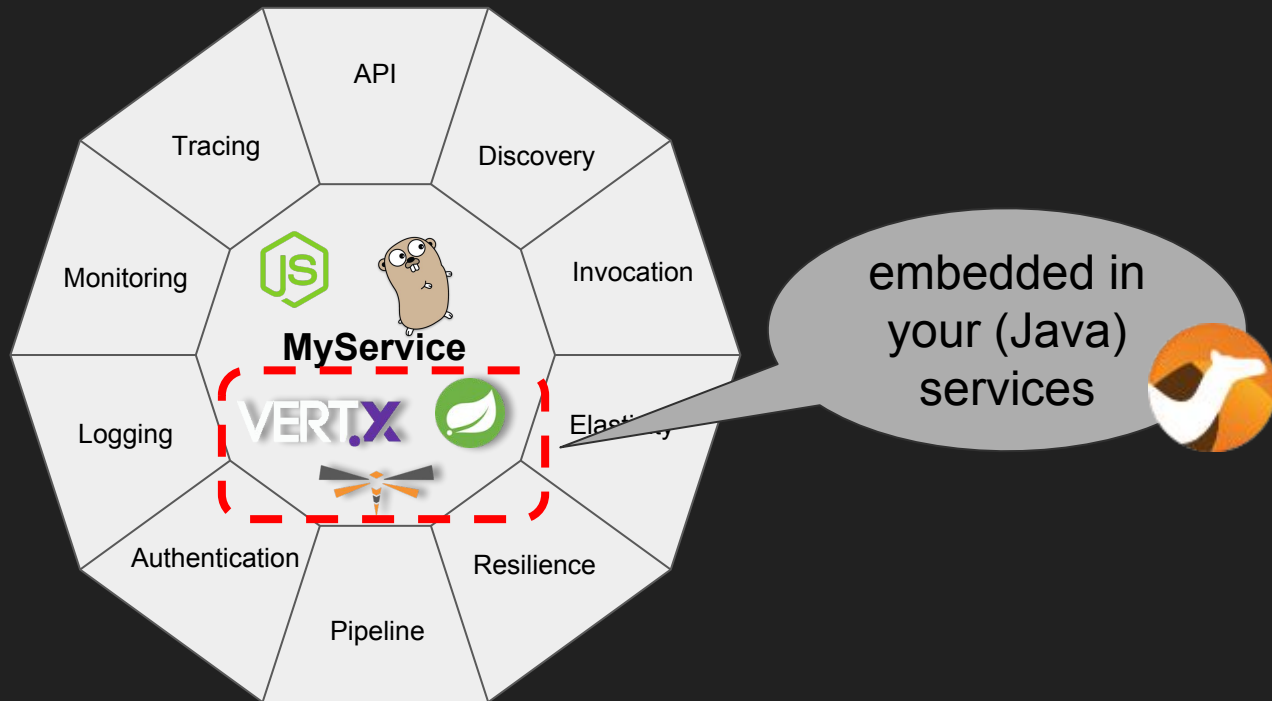
Microservices'ilities + API management



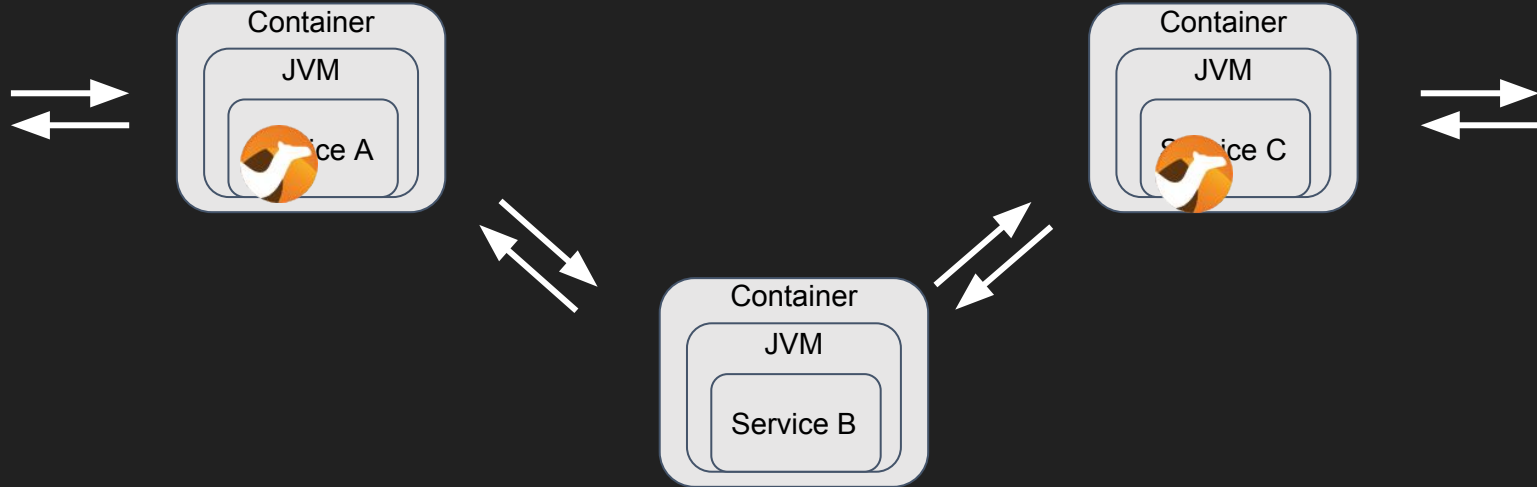
But where is Camel?



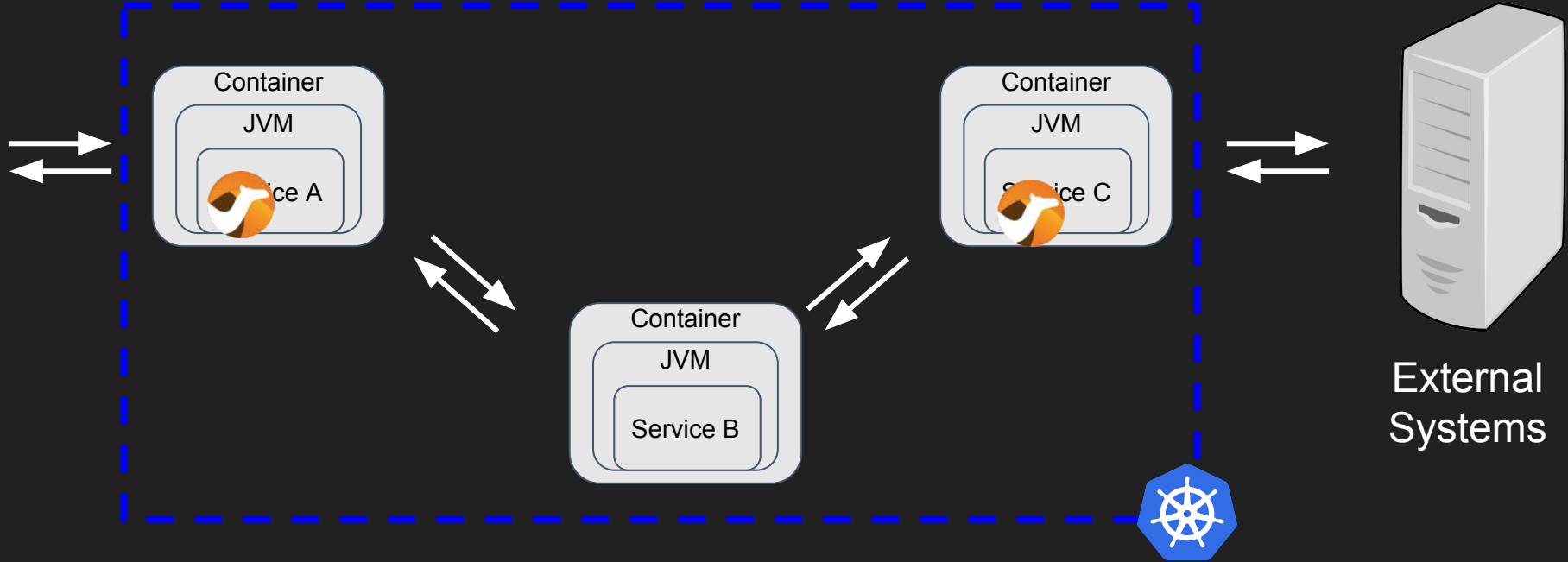
But where is Camel?



Microservices == Distributed Integration



Microservices == Distributed Integration



Camel in the Cloud



Best Practice - Small in Size

- Camel is light-weight
 - (camel-core 4mb)
 - + what you need
- Single fat-jar via:



Best Practice - Stateless

- Favour stateless applications
- If state is needed:
 - Data-grid
 - camel-infinispan
 - camel-hazelcast
 - camel-ignite
 - ...
 - Storage
 - camel-sql
 - camel-jpa
 - camel-kafka
 - ...
 - Kubernetes
 - Stateful-set

Best Practice - Configuration Management

- Kubernetes ConfigMap
 - Inject via ENV
 - Inject via files
- Kubernetes Secrets
 - inject via ENV
 - Inject via files



```
// inject configuration via spring-style @Value  
@Value("${fallback}")  
private String fallback;
```



```
.simple( text: "{{fallback}}")
```

```
$ kubectl get cm -o yaml my-configmap  
apiVersion: v1  
data:  
  fallback: I still got no response  
kind: ConfigMap
```

Best Practice - Fault Tolerant

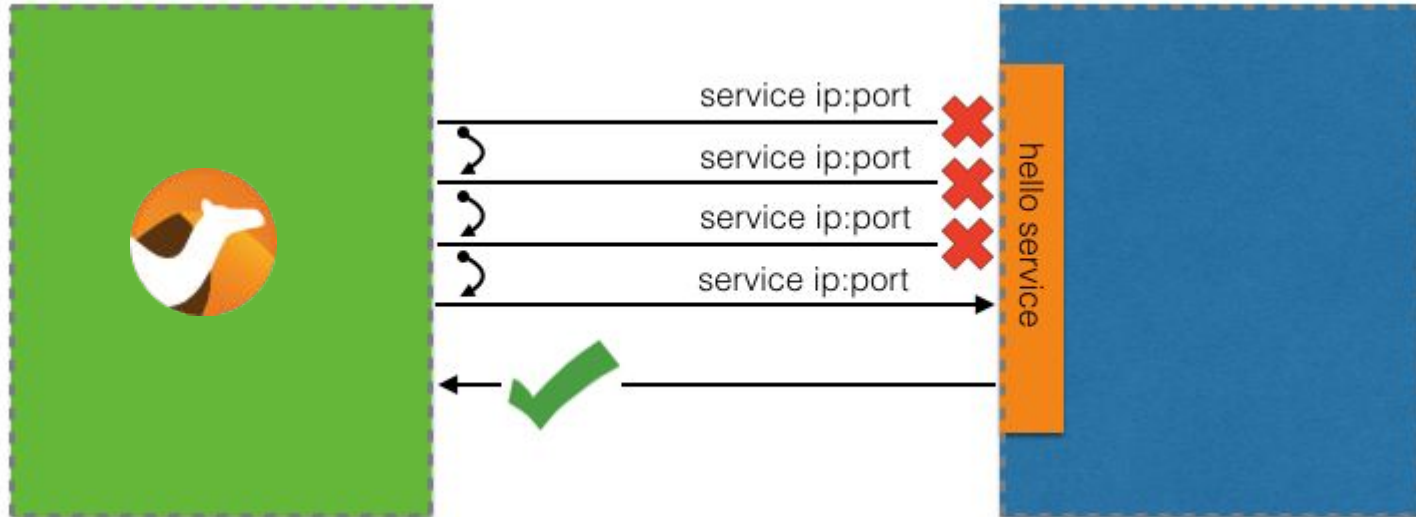
- Camel Retry
 - `onException`
 - `errorHandler`
- Circuit Breaker
 - `camel-hystrix`



Best Practice - Fault Tolerant

- Camel Retry
 - `onException`
 - `errorHandler`

```
onException(Exception.class)  
    .maximumRedeliveries(10)  
    .redeliveryDelay(1000);
```



Best Practice - Fault Tolerant

- Camel Retry
 - `onException`
 - `errorHandler`

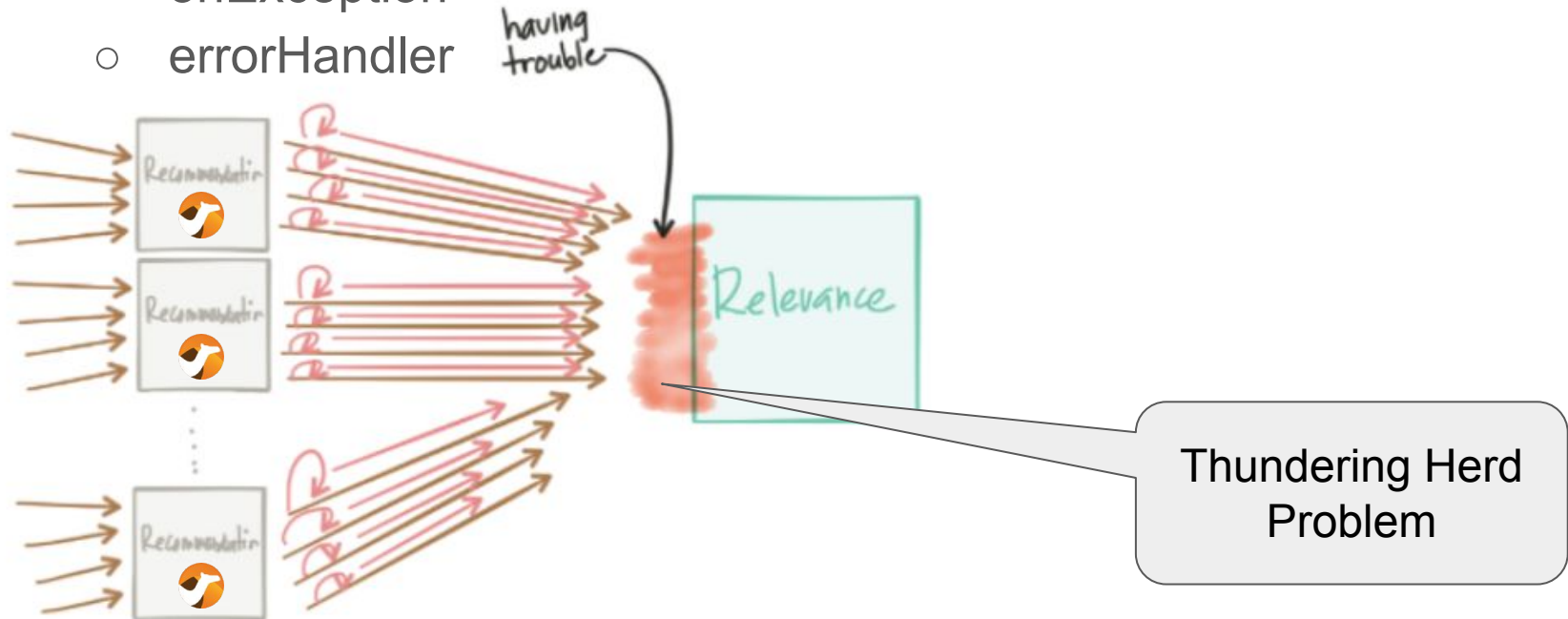


Figure by Christian Posta

Best Practice - Health Checks

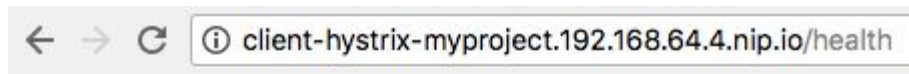
- Health Checks
 - camel-spring-boot actuator
 - wildfly-swarm monitor



- Readiness Probe
 - Kubernetes



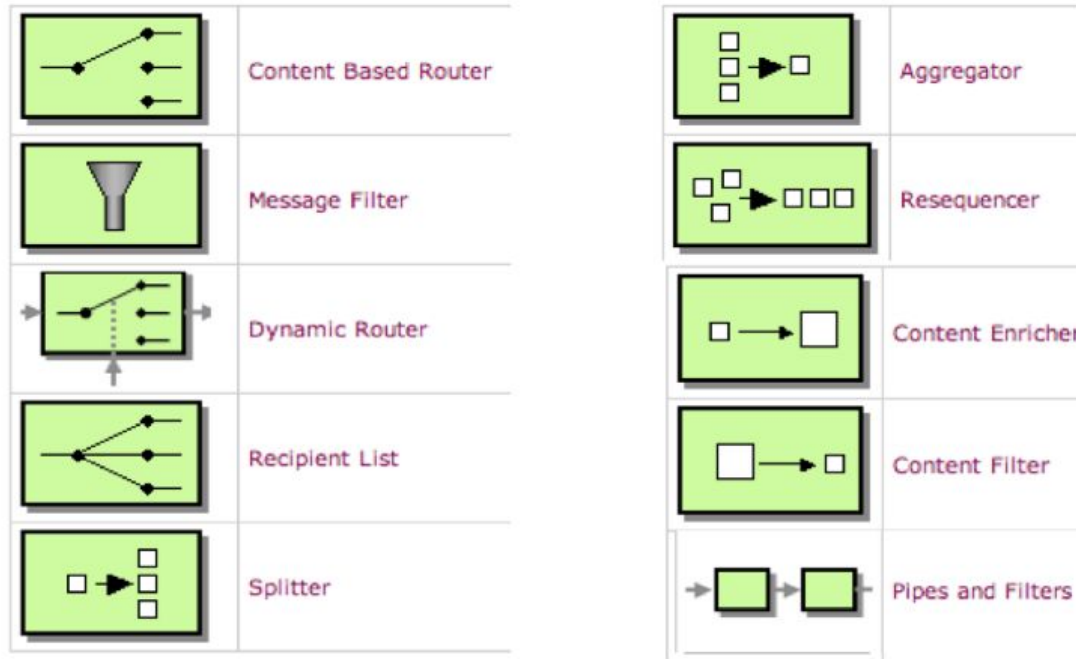
- Liveness Probe
 - Kubernetes



```
{
  status: "UP",
  - camel: {
    status: "UP",
    name: "camel-1",
    version: "2.20.2",
    contextStatus: "Started",
  },
  - camel-health-checks: {
    status: "UP",
    route:routel: "UP",
  },
  - diskSpace: {
    status: "UP",
    total: 19195224064,
    free: 5747757056,
    threshold: 10485760,
  },
}
```

Best Practice - EIP Patterns

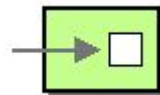
- Works anywhere



Plugins

- Consul
- Etcd
- Kubernetes
- Ribbon
- Zookeeper

EIP Cloud Patterns



Service Call

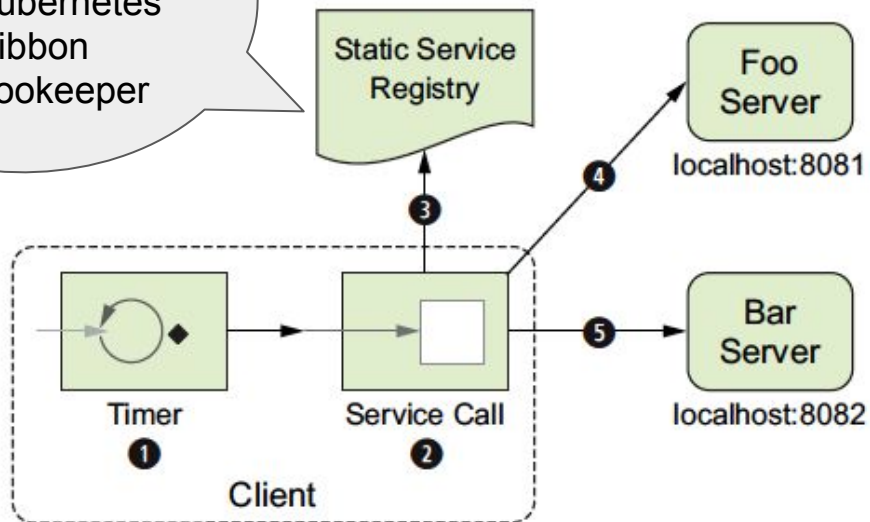
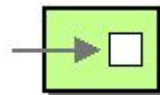


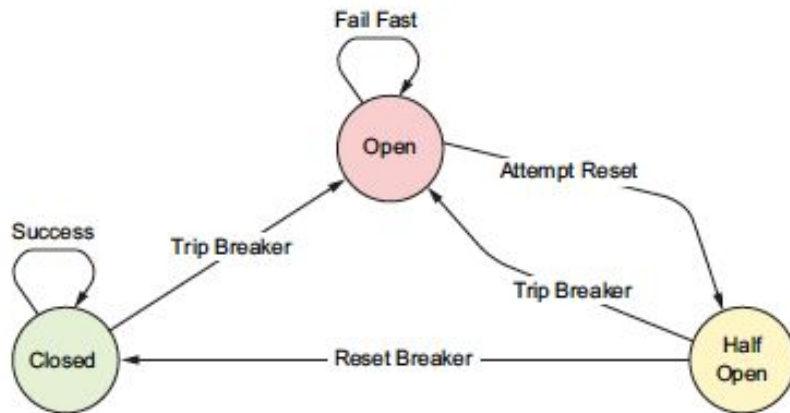
Figure 17.9 A timer ① triggers the Service Call EIP ② to call a clustered service. The physical locations of the service are looked up in the service registry ③. The service is then called in a round-robin fashion by calling either Foo server ④ or Bar server ⑤.

```
from("timer")
    .serviceCall("hello-service");
```

EIP Cloud Patterns

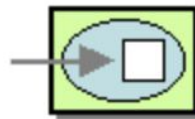


Hystrix EIP

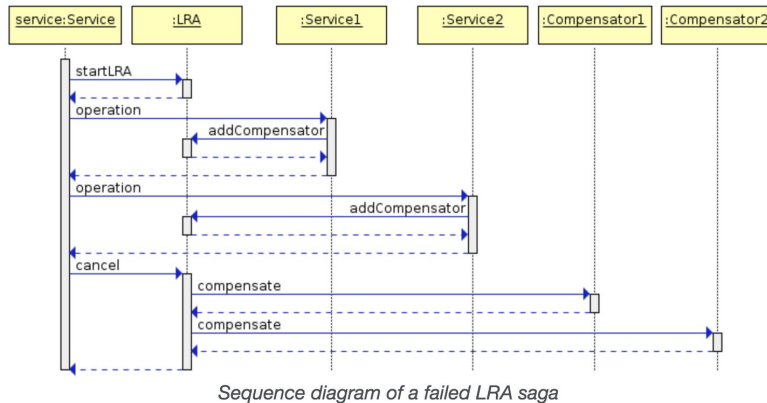


```
from("timer:foo")
  .hystrix()
    .to("http:myservice")
  .onFallback()
    .to("bean:myfallback")
  .end()
```

EIP Cloud Patterns



Saga EIP



```
rest().post("train/buy/seat")
    .saga()
        .compensation("direct:cancel")
        ...
    .to("http:trainservice/buy")
```

EIP Cloud Patterns



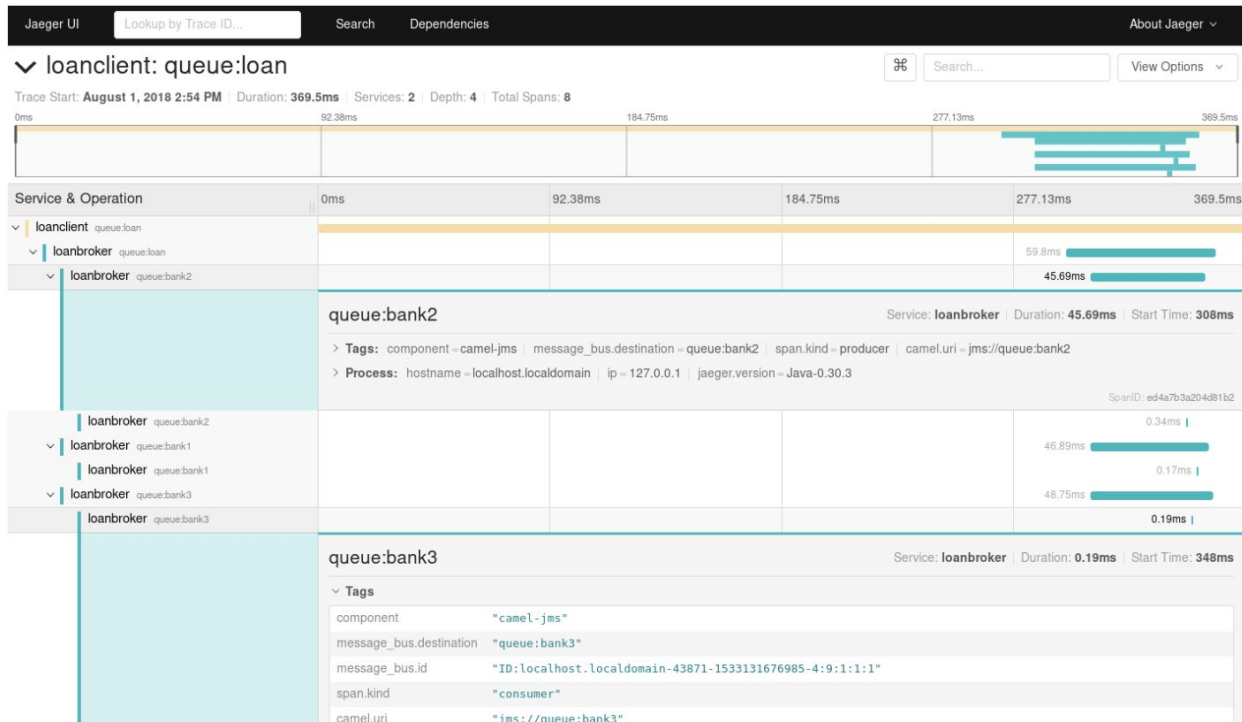
Distributed
Tracing



EIP Cloud Patterns



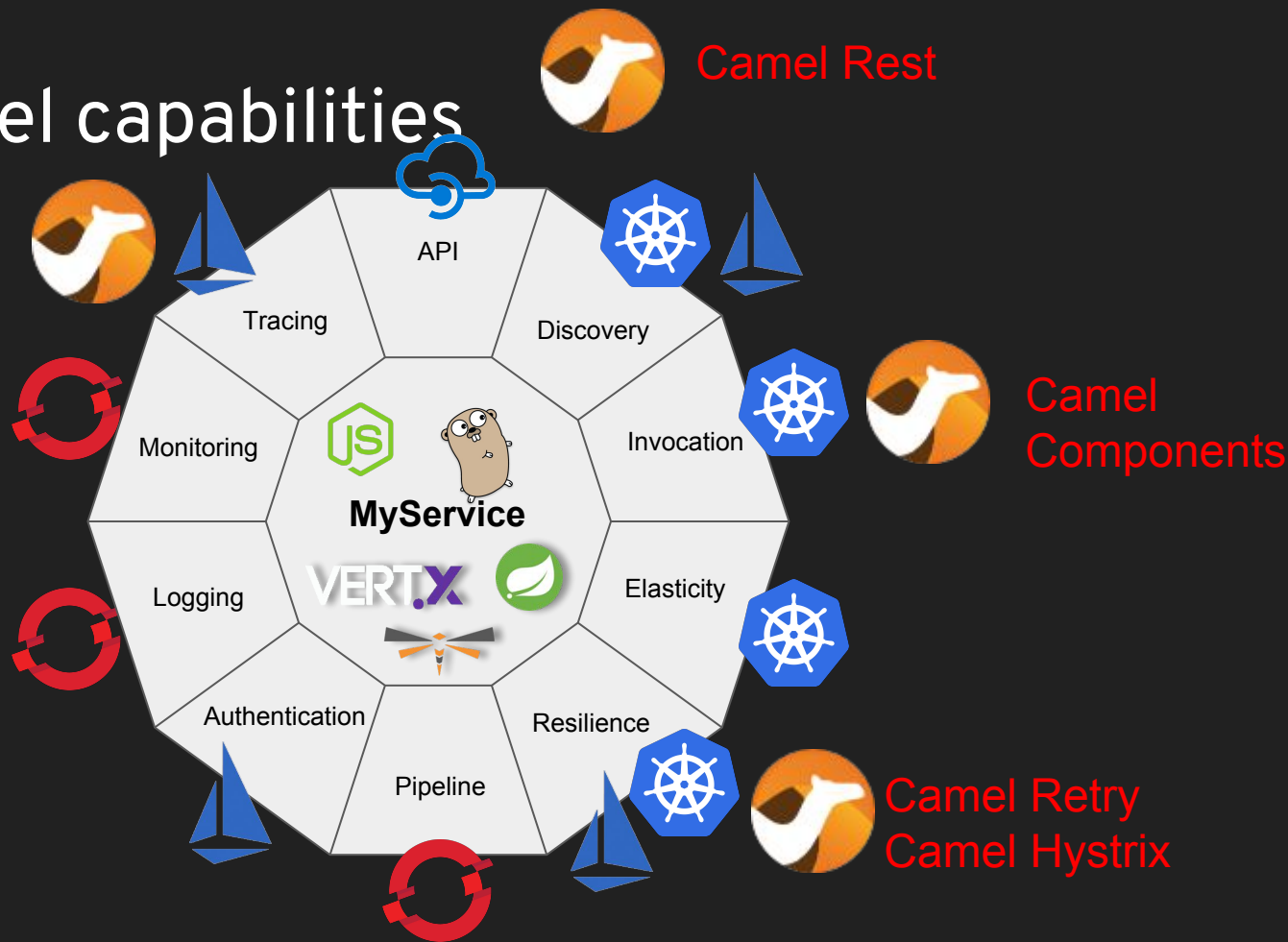
Distributed
Tracing



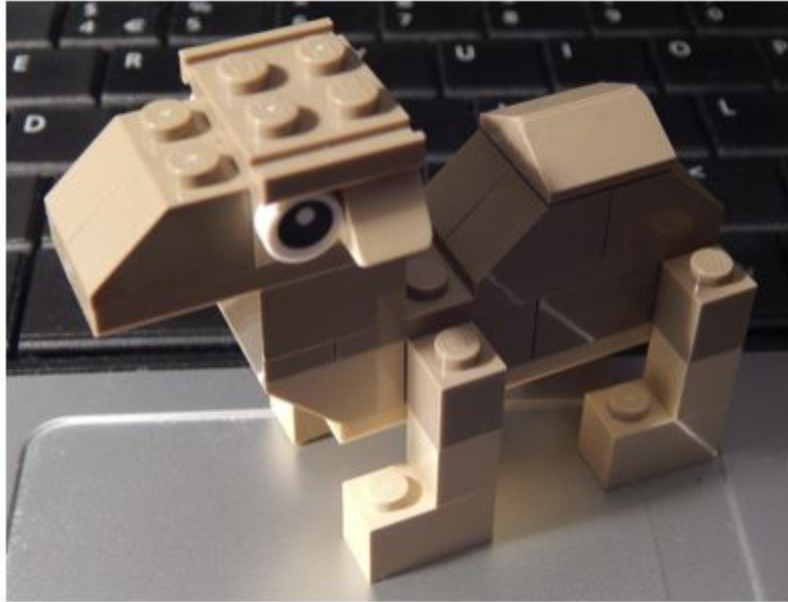
Jaeger UI

Usable Camel capabilities

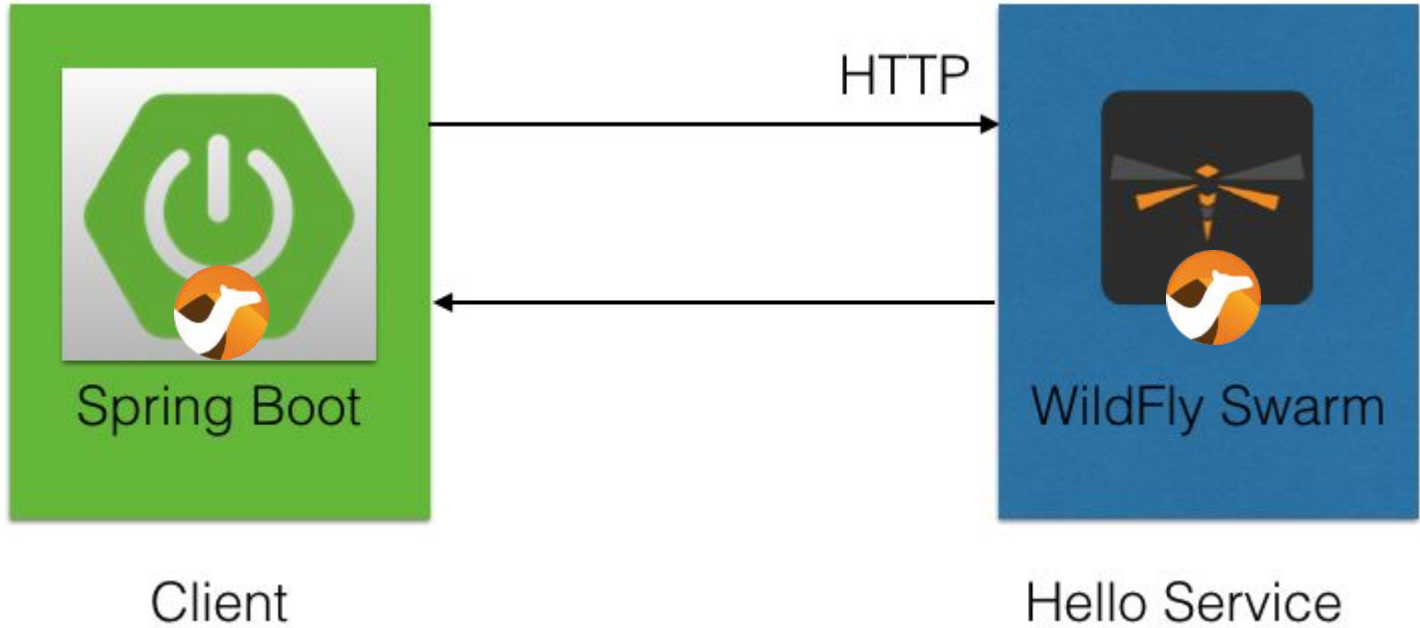
Camel Zipkin
Camel OpenTracing



Demo Time



Basic Demo





Client Camel Route

```
from("timer:foo?period=1000")  
  .hystrix()  
    .to("http:helloswarm:8080/hello")  
  .onFallback()  
    .setBody()  
      .constant("Nobody want to talk to me")  
  .end()  
  .log("${body}");
```

Server Camel Route



```
from("undertow:http://0.0.0.0:8080/hello")  
    .bean(helloBean);
```



```
public String sayHello() throws Exception {  
    return "Swarm says hello from " +  
        InetAddressUtil.getLocalHostName();  
}
```

Tip of the iceberg



Figure by Bilgin Ibryam

Future of Apache Camel

- New website and documentation
 - End of 2018 / Start of 2019
- Apache Camel 2.23
 - November
 - ... likely last 2.x release
- Camel 3 currently being planned
 - To be released in 2019 (summer or before)

Future of Apache Camel

- Camel on GraalVM

- <https://lburgazzoli.github.io/2018/09/04/Adventures-in-GraalVM-polyglot-Camel-routes-with-native-image.html>

Now, let's write a simple JavaScript route:

```
from('timer:js')  
  .setBody('test')  
  .to('log:js')
```

And finally, let's run it:

```
sources/route.js  
t - Apache Camel 2.23.0-SNAPSHOT (CamelContext: camel-1) is starting  
t - StreamCaching is not in use. If using streams then its recommended to enable  
t - Route: route1 started and consuming from: timer://js  
t - Total 1 routes, of which 1 are started  
t - Apache Camel 2.23.0-SNAPSHOT (CamelContext: camel-1) started in 0.001 seconds  
xchange[ExchangePattern: InOnly, BodyType: String, Body: test]  
xchange[ExchangePattern: InOnly, BodyType: String, Body: test]  
xchange[ExchangePattern: InOnly, BodyType: String, Body: test]
```

Future of Apache Camel

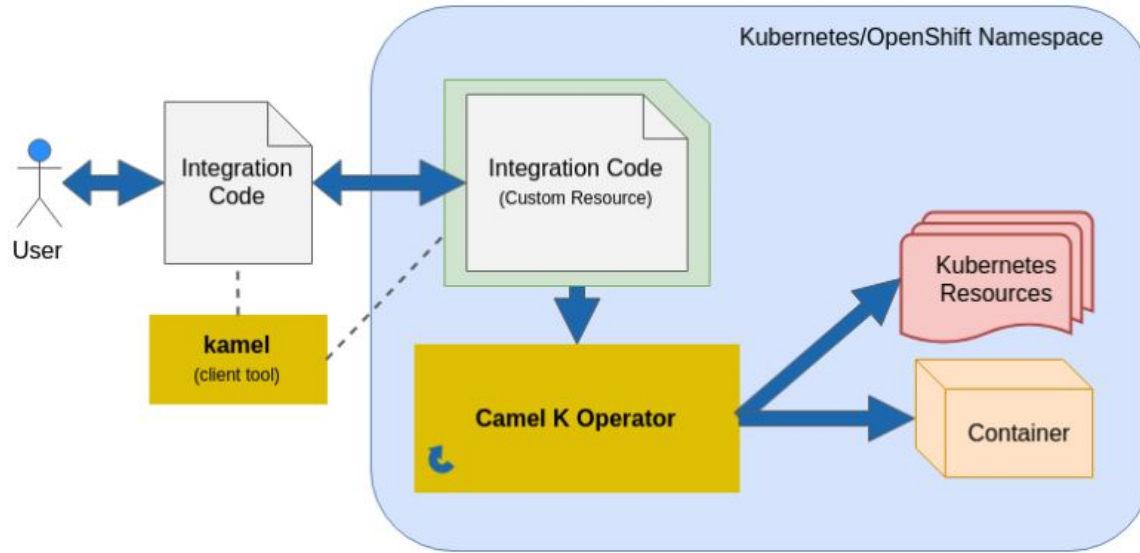
- Apache Camel K
 - <https://github.com/apache/camel-k>
 - Blog with details: <https://www.nicolaferraro.me/2018/10/15/introducing-camel-k>

Apache Camel K (a.k.a. Kamel) is a lightweight integration framework built from Apache Camel that runs natively on Kubernetes and is specifically designed for serverless and microservice architectures.

Future of Apache Camel

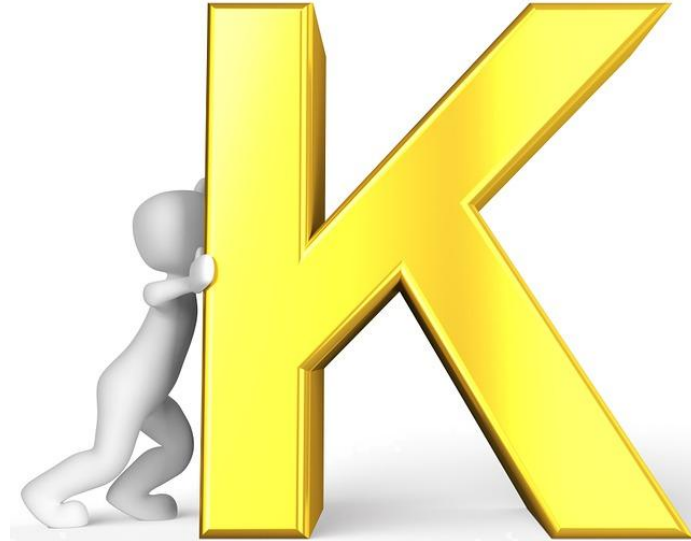
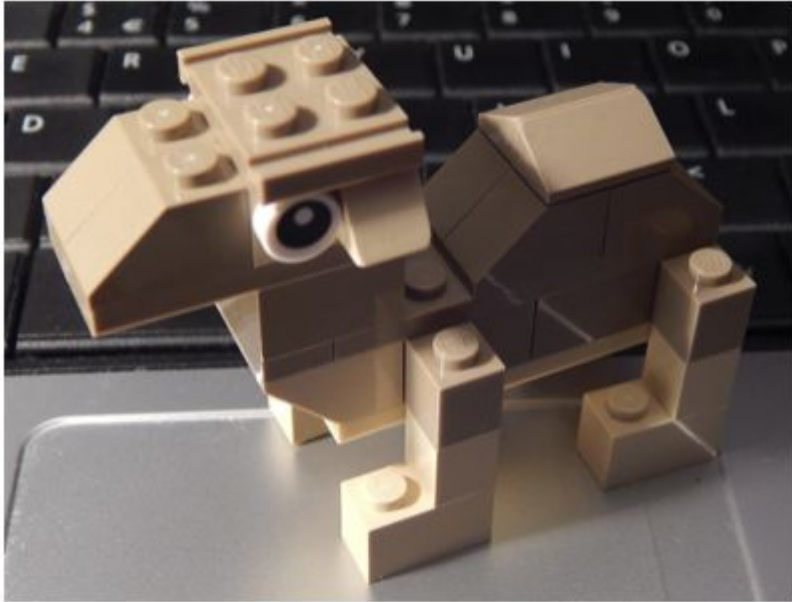
- Apache Camel K

This is what happens under the hood:

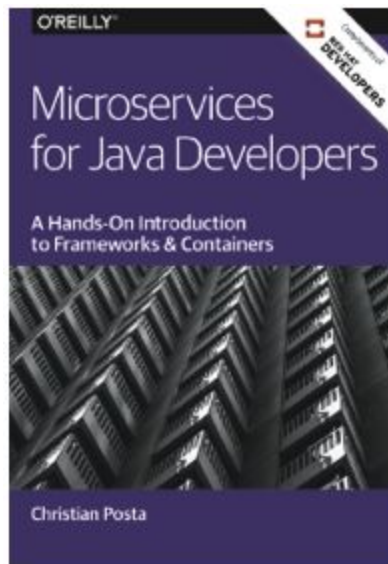


<https://www.nicolaferraro.me/2018/10/15/introducing-camel-k>

Demo Time

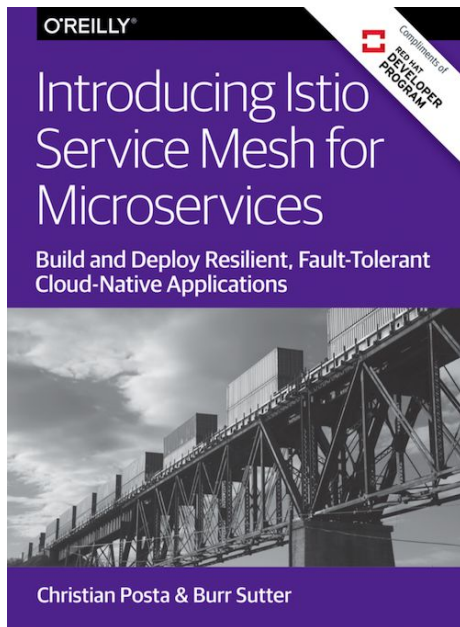


Free book



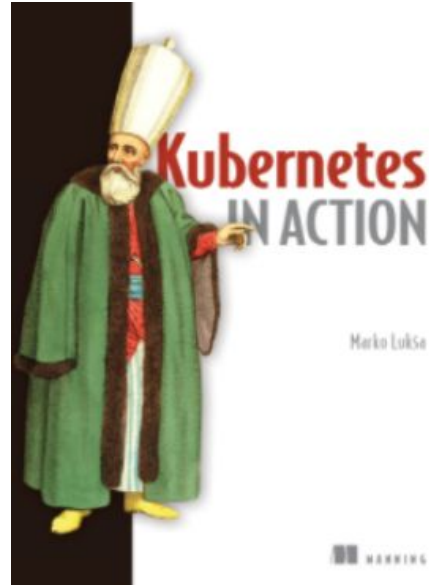
<http://developers.redhat.com/promotions/microservices-for-java-developers>

Free book



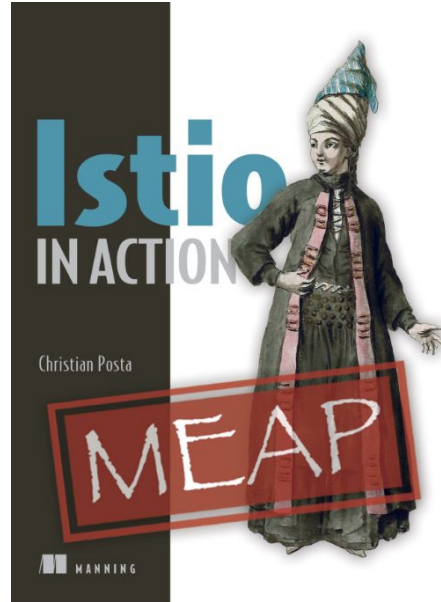
<https://developers.redhat.com/books/introducing-istio-service-mesh-microservices/>

Not so free book



<https://www.manning.com/books/kubernetes-in-action>

Not so free book



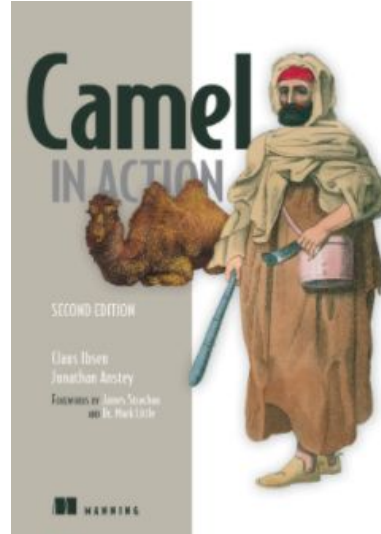
<https://www.manning.com/books/istio-in-action>

Not so free book

- Discount code (39%):

came139

(ordering from Manning)



<https://www.manning.com/books/camel-in-action-second-edition>

Buon compleanno Filippo da Milano



More Information

- Slides and Demo source code:
<https://github.com/davsclaus/camel-riders-in-the-cloud>
- Apache Camel website:
<http://camel.apache.org>
- Best "What is Apache Camel" article:
<https://dzone.com/articles/open-source-integration-apache>
- My blog:
<http://www.davsclaus.com>
- Camel K:
<https://github.com/apache/camel-k>

Q & A