“Cloud Native My Camel”

Adopting Cloud Native Architecture with Apache Camel-K

APACHECON @HOME
Spt, 29th – Oct. 1st

2020

Quick Intro and Agenda

- Michael Costello
  - A programmer
  - Senior Architect Red Hat Enterprise Integration Practice
  - > 20 years of distributed software fun
  - check me out @ https://entropic.me

- David Gordon
  - Writes in YAML and Camel, sometimes both
  - Senior Architect Red Hat Integration Practice
  - check me out @aph3lio

Agenda

- **Intro: How we came to EIP's and Camel**
- **Prequel: SOA and the Enterprise Service Bus pattern**
- **Moving Integration Patterns to the Cloud**
- **What is *"Cloud Native"* Architecture and How do I apply it to Apache Camel?**
- **Live Demo: Camel-K, Strimzi, and Knative in action!**
- **Check out this demo and more at:**

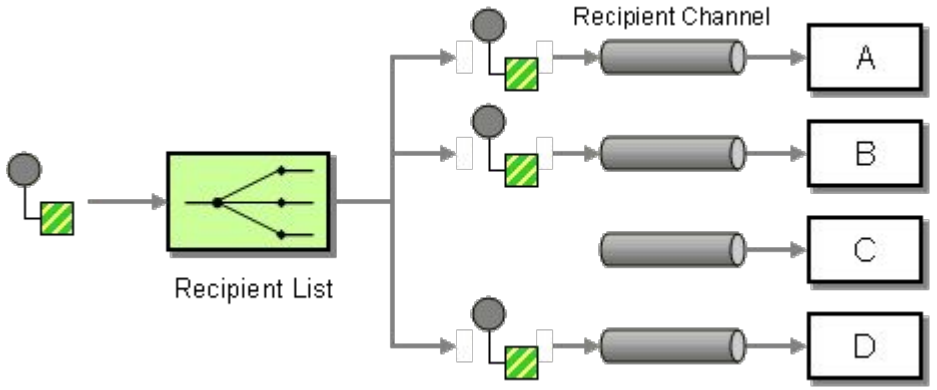
  https://github.com/rh-ei-stp/cloud-native-event-mesh

APACHECON @Home Sep 29 - Oct 1, 2020

APACHECON @Home Sep 29 - Oct 1, 2020

How we came to EIPs and Camel

- We transitioned away from mainframe to client/server to notice an explosion of endpoints and a **need for remote invocation**
- A variety of toolsets were published to support integration needs including **asynchronous messaging** (e.g. MQ Series) that modeled bus like patterns of prior mainframe systems
- "I had a nagging feeling that these tools share underlying concepts, which are obfuscated by different terminology."
  ~ Gregor Hohpe
- Enterprise Integration Patterns book was written to establish a common **vocabulary** and distill repeated techniques down to generic **patterns**
- Standards emerged and Camel was built around that common vocabulary, offering a useful, intuitive **DSL** for connecting systems with repeatable patterns




```
from("jms://queue:alerts")
    .recipientList(header("subscribers"))
    .parallelProcessing();
```

SOA and the Enterprise Service Bus Pattern

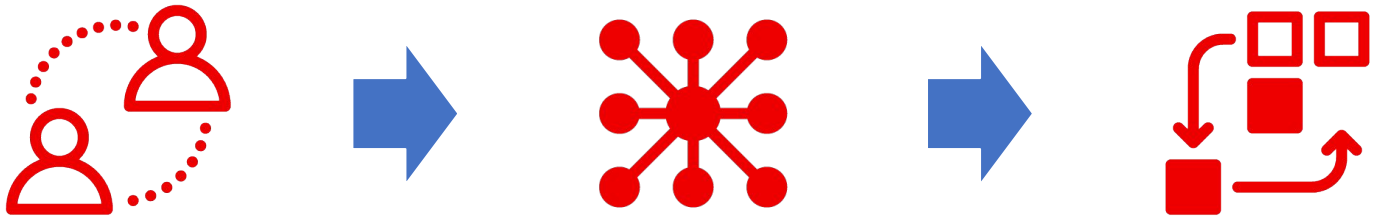
The Triumphs

- Exposing reusable service endpoints over **common communication standards** continues to be an effective architectural strategy
- **Loose coupling** between services reduces risks when introducing change
- The ESB pattern enables us to **adapt legacy services** that cannot natively conform to communication standards
- ESB's offer a toolset to **implement complex processes** that make use of multiple services

The Challenges: Where we left off...

- Complex interactions often require **state management** in order to offer guarantees, so familiar tradeoffs between consistency and availability exist, especially when transaction management is involved
- Integration implementations are often coupled to a **platform-specific interfaces** such as an ESB's message broker API
- ESB popularized a **central management** model for integrations viewed in many cases as a bottleneck for feature delivery and a philosophical clash with Microservices

On the Integration Highway



Point to Point

Direct connection between systems, application both internally and with external services

Enterprise Service Bus

Placing a centralized bus that integrate between loosely coupled services.

Microservices

Fine grained distributed services, allowing faster turnover rate, more agile and flexible deployment model.

Moving Integration Patterns to the Cloud

New concerns

- Expect infrastructure failure and tolerate it
- Remain cloud vendor-neutral
- Scale down application components when demand is low to save cost
- Distribute architecture across cloud infrastructure availability zones

These new concerns lead many software organizations to consider **container platform adoption**. As part of the transition to containers, organizations often break down monolithic implementations into independently deployable components (**Microservices**) to achieve finer scale points and a smaller failure blast-radius.

Pain-points

- Instrumenting, observing, and responding to application component metrics and health requires a new set of tools
- K8s helps to abstract away cloud-specific infrastructure APIs, but adoption is a journey for developers
- Decomposing monoliths into microservices could result in an increased resource footprint due to the number of components times the overhead of the each service's baseline resource requirement
- There's still a need to manage state reliably and in a cloud context, we should expect to lose persistent storage occasionally

But, what does “Cloud Native” Mean?

https://github.com/rh-ei-stp/cloud-native-integration

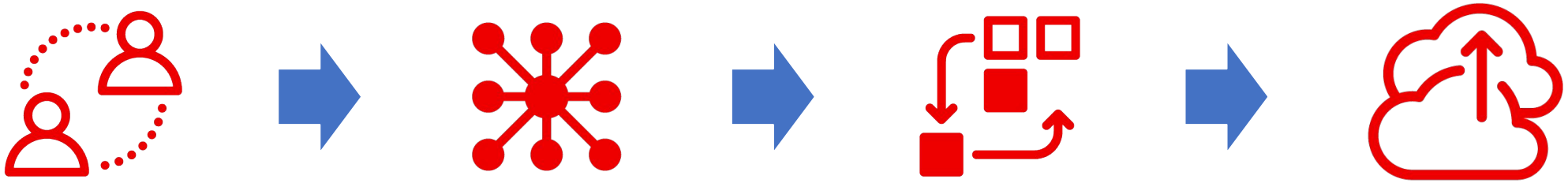
Cloud Native Characteristics

- Elastic
- Scalable on-demand
- Resilient (able to survive the loss of an AZ)
- Observable/Manageable
- Location Agnostic
- API-Centric
- Event Driven

More Than Just a “Move to the Cloud”

- Relying solely on a Cloud API makes us only native to that cloud
- Abstraction from proprietary cloud API’s via Kubernetes
- Kubernetes and containers alone aren’t enough, we need something to care and feed for deployments (such as the Operator SDK)

The Integration Destination



Point to Point

Direct connection between systems, application both internally and with external services

Enterprise Service Bus

Placing a centralized bus that integrate between loosely coupled services.

Microservices

Fine grained distributed services, allowing faster turnover rate, more agile and flexible deployment model.

Serverless

Scale down to zero. Optimize Resource Usage. Avoid random, arbitrary workload prediction

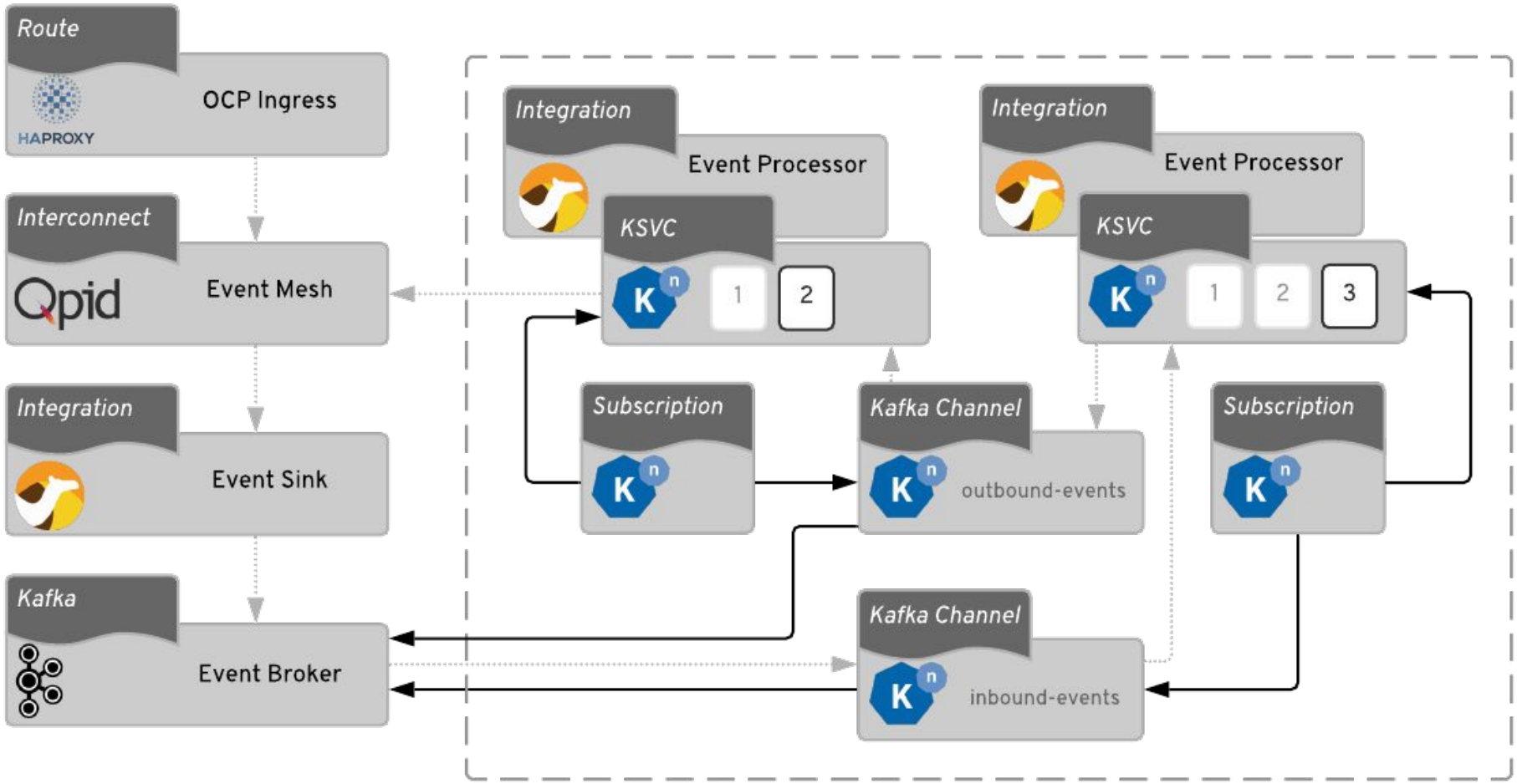
Camel-K: Same Camel, New Kontext

- Write **Camel DSL in multiple languages**: Java, XML, YAML, Groovy, JavaScript, Kotlin
- Uses the **Kubernetes Operator pattern** to manage application lifecycle
- Offers a **convenient CLI** that abstracts K8s details from developers
- Subsecond deployment and startup using the **Quarkus** runtime
- Run integrations in **serverless** mode; scale from zero to n replicas according to demand
- Integrate with Knative **event channels** and implement EDA with the CloudEvents spec

APACHE Camel


```
$ kamel run integration.js \
    -t service.enabled=true \
    -t knative.enabled=true \
    -t quarkus.enabled=true \
    -t quarkus.native=true
```

Cloud Native Integration Demo



DEMO

https://github.com/rh-ei-stp/cloud-native-event-mesh

Get on the Bus

Q&A