INTEGRATION PATTERNS IN A SERVERLESS WORLD: THE GOOD PARTS

Bilgin Ibryam
Principal Architect

Claus Ibsen
Senior Principal Software Engineer

May 7th 2019
About us

Claus Ibsen
- Senior Principal Software Engineer at Red Hat
- ASF Member & Java Champion
- Camel tech lead
- Author of Camel in Action books
- Based in Denmark

Bilgin Ibryam
- Principal Architect at Red Hat
- ASF Member
- Camel committer
- Author of Camel Design Patterns & Kubernetes Patterns books
- Based in the UK
Agenda

● Integration Evolution
● Workload Types
● What is Serverless
● What is Camel K
● Camel K & Knative
● Quarkus
● Summary
● Q & A
Integration Evolution
Integration architectures

Service-Oriented Architecture
(Smart pipes, dumb endpoints)

Microservices Architecture
(Smart endpoints, dumb pipes)

Cloud-Native Architecture
(Infrastructure focused platform, application focused services)

application concern  infrastructure concern

application concern  infrastructure concern

application concern  infrastructure concern
Service management unit - Pod

Deployment guarantees

Lifecycle guarantees
Stateful applications

StatefulSet guarantees:
- Dedicated storage
- Stable networking identity
- Persistent instance identity
- Instance ordinality
Singleton behaviour

In-application locking

Out-of-application locking

“At least” or “At most” semantics
Batch and recurring jobs

Characteristics
- Run until completion
- Idempotent operations
- Resource intensive
- Run as background tasks
- Typically executed on schedule
Stateless services

Characteristics
- The classic 12-factor applications
- Long running processes
- Horizontally (auto) scalable
- Healthcheck and recovery
Istio service mesh

High level capabilities

- **Traffic routing**: A/B tests, staged rollouts
- **Resilience**: retries, circuit breakers, connection limits, health checks
- **Security**: authentication, authorization, encryption (mTLS)
- **Observability**: metrics, tracing
- **Testing**: fault injection, traffic mirroring
- Platform-independent, polyglot, allows runtime configuration
Service mesh & Integration

- Traffic routing
- Resiliency
- Content routing
- Security
- Tracing
- Adapters
- Runtime configuration
- Polyglot
- Content transformation
Service mesh & API manager

- Infrastructure centric
- Traffic routing
- Resiliency
- Rate limiting
- Security
- Analytics
- Business centric
- Billing
- Developer & docs

#redhat #rhsummit
Serverless
What is serverless?

“What serverless computing refers to the concept of building and running applications that don’t require server management. It describes a finer-grained deployment model where applications, bundled as one or more functions, are uploaded to a platform and executed, scaled, billed in response to the exact demand needed”

https://github.com/cncf/wg-serverless

Serverless personas:

- **Developer**: writes code for, and benefits from the serverless platform.
- **Provider**: manages the serverless platform.
Serverless use cases

- **On-demand functionality**: batch processing, stream processing, extract-transform-load (ETL).
- **Task scheduling**: divisible work performed for a short time, such as batch jobs.
- **Event-driven architecture**: executing logic in response to data source changes.
- **Handling non-uniform traffic**: occasional inconsistent traffic that doesn’t happen often, traffic with unpredictable load.
- **General purpose “glue” code** in operations.
- **Continuous integration pipelines**: provision on-demand resources for build jobs.
- **Automate operational tasks**: when a known incident happens actions will be triggered to fix the problem or notify a person on call.
Serverless requirements

For the platform
- Scaling to zero
- Rapid scaling up and down
- Eventing mechanism
- Routing and networking

For the application runtime
- Fast startup
- Fast first response
- Low on memory and CPU
- Small size on disk
Camel K
What is Apache Camel K?

A lightweight integration platform, based on Apache Camel, born on Kubernetes, with serverless superpowers.

1. Runs on “vanilla” Kubernetes (1),
2. Openshift (2) and gives its best on a Knative-powered cluster (3)!
How?

1. Create an integration file (Java, Groovy, Kotlin, JS, XML...)

   from("telegram:bots/bot-id")
   .transform()
   .to("kafka:topic");

   from("kafka:topic")
   .to("http:my-host/api/path");

2. Run it

   $ kamel run integration.groovy

3. Runs on OpenShift or Kubernetes

Camel DSL, based on EIPs...
Architecture of Camel K

Dev Environment

Remote Cloud

“Integration” Custom Resource

Camel K Operator

Running Pod

Live updates!

Fast redeploy!
Less than 1 second!

Tailored for cloud-native development experience
Fast Deployment of Camel K

Time to run a integration using different strategies (in seconds)

-F-m-p is the “fabric8-maven-plugin” (http://maven.fabric8.io/) deploying a average spring-boot based integration on Minishift vs. a remote OpenShift cluster (accounting time to upload the fat Jar). Source S2I build has been measured in Red Hat Fuse Online.

Lower is better 😃
Quick Camel K Demo

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

public class Sample extends RouteBuilder {
    @Override
    public void configure() throws Exception
    {
        from("timer:tick")
            .log("Hello Camel K!");
    }
}
```
import org.apache.camel.builder.RouteBuilder

public class Sample extends RouteBuilder {
    @Override
    public void configure() throws Exception
    from("timer://tick")
    .log("Hello Boston");
}

2019-05-03 08:50:08.082 INFO [Camel (camel-k) thread #2 - timer://tick] route1 - Hello Camel K!
2019-05-03 08:50:08.082 INFO [Camel (camel-k) thread #2 - timer://tick] route1 - Hello Camel K!
integration "sample" updated
Camel K & Knative
What is Knative?
Knative Overview - Components

Kubernetes-based platform to build, deploy, and manage modern serverless workloads

**Build**
A pluggable model for building artifacts, like jar files, zips or containers from source code.

**Serving**
An event-driven model that serves the container with your application and can "scale to zero".

**Eventing**
Common infrastructure for consuming and producing events that will stimulate applications.
What about Camel K & Knative?
Camel K and Knative

Knative components relevance to Camel

**Knative Build**
- Standardize building container images

**Knative Serving**
- Auto-scaling and scale-to-zero

**Knative Eventing**
- Messaging for event-based applications
Camel K and Knative

Knative components relevance to Camel

**Knative Build**
- Standardize building container images

**Knative Serving**
- Auto-scaling and scale-to-zero

**Knative Eventing**
- Messaging for event-based applications
There's no container if no one needs it!
Camel K and Knative Serving

```
rest().post("/path")
  .to("xx:system1")
  .to("xx:system2")
```

Kubernetes Namespace

Pod

Knative Service

A container is created only when needed!

300+ components!

System 1
System 2

Request
Camel K and Knative Serving

```
rest().post("/path")
  .to("xx:system1")
  .to("xx:system2")
```

Kubernetes Namespace

Multiple containers under high load!

300+ components!
Camel K and Knative

Knative components relevance to Camel

- **Knative Build**
  - Standardize building container images
  - <<custom-resource>> Build
  - ... 

- **Knative Serving**
  - Auto-scaling and scale-to-zero
  - <<custom-resource>> Service
  - ... 

- **Knative Eventing**
  - Messaging for event-based applications
  - <<custom-resource>> Channel
  - ... 

Standardize building container images
Auto-scaling and scale-to-zero
Messaging for event-based applications
Kubernetes Namespace

Camel K and Knative Eventing

from("knative:channel/a")
  .to("xx:system1")
  .to("xx:system2")

Kubernetes Namespace

Cloud Event

Persisted by Kafka, in-memory (pluggable)

https://cloudevents.io/

300+ components!
Camel K and Knative Eventing

Camel K: same model for different purposes

300+ components!

65+ patterns
Knative Eventing includes Camel

Standard Knative Sources that can be used without any knowledge of Camel K!

... but Camel K is under the hood ;)

#redhat #rhsummit
Quarkus
SAY JAVA IS SLOW
ONE MORE TIME
## Java Density Problem

![Diagram of Java Density Problem]

<table>
<thead>
<tr>
<th>Node</th>
<th>Node</th>
<th>Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional Cloud-Native Java Stack</td>
<td>NodeJS</td>
<td>NodeJS</td>
</tr>
<tr>
<td>Traditional Cloud-Native Java Stack</td>
<td>NodeJS</td>
<td>NodeJS</td>
</tr>
<tr>
<td>Traditional Cloud-Native Java Stack</td>
<td>NodeJS</td>
<td>NodeJS</td>
</tr>
<tr>
<td>Traditional Cloud-Native Java Stack</td>
<td>NodeJS</td>
<td>NodeJS</td>
</tr>
</tbody>
</table>

**CONTAINER ORCHESTRATION**

[https://developers.redhat.com/blog/2017/03/14/java-inside-docker/]
Java Fast Scaling Problem

SCALE-UP  SCALE-DOWN  SCALE-TO-ZERO
What is Quarkus?

A Kubernetes Native Java stack tailored for GraalVM & OpenJDK HotSpot, crafted from the best of breed Java libraries and standards.

https://quarkus.io
Minimal Footprint

- Minimal footprint Java applications
  - Native vs Quarkus+OpenJDK JVM vs Traditional JVM

[Memory and Boot + First Response Time Diagram]

- Memory (RSS) in Megabytes
  - REST
    - Quarkus + GraalVM: 13 MB
    - Quarkus + OpenJDK: 74 MB
    - Traditional Cloud-Native Stack: 140 MB
  - REST + JPA
    - Quarkus + GraalVM: 35 MB
    - Quarkus + OpenJDK: 130 MB
    - Traditional Cloud-Native Stack: 218 MB

- Boot + First Response Time in Seconds
  - REST
    - Quarkus + GraalVM: 0.014 sec
    - Quarkus + OpenJDK: 0.75 sec
    - Traditional Cloud-Native Stack: 4.3 sec
  - REST + JPA
    - Quarkus + GraalVM: 0.55 sec
    - Quarkus + OpenJDK: 2.5 sec
    - Traditional Cloud-Native Stack: 9.5 sec
Apache Camel and Quarkus runs blazingly fast 🚀 with native compiled binaries. Startup is 7 msec, 28mb binary file, and 15mb RSS memory used.
Summary
# Application trends (Java)

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Monolithic/SOA</th>
<th>Microservices/CNA</th>
<th>Functions/Serverless</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packaging</td>
<td>Ear/War/Jar</td>
<td>Fat-jar</td>
<td>Native executable</td>
</tr>
<tr>
<td>Startup procedure</td>
<td>XML scanning</td>
<td>Annotation scanning</td>
<td>Instant (AOT compilation)</td>
</tr>
<tr>
<td>Memory/binary size</td>
<td>1000s MBs</td>
<td>100s MBs</td>
<td>10s of MBs</td>
</tr>
<tr>
<td>Startup/first response</td>
<td>Minutes</td>
<td>Seconds</td>
<td>Milliseconds</td>
</tr>
<tr>
<td>Process lifecycle</td>
<td>Long Running</td>
<td>Managed (w/ restarts)</td>
<td>Executes when invoked</td>
</tr>
<tr>
<td>Data ownership</td>
<td>Shared data</td>
<td>Owns data</td>
<td>Does not own data</td>
</tr>
<tr>
<td>Operations</td>
<td>Multiple unrelated operations</td>
<td>Domain related operations</td>
<td>Single operation</td>
</tr>
<tr>
<td>Focus area</td>
<td>Multiple business domains</td>
<td>Single business domain</td>
<td>Single task</td>
</tr>
</tbody>
</table>
Platform trends (Kubernetes)

Application concerns:
- Routing & transformation
- Technology adapters
- Error handling
- Source-to-service
- Scale-to-zero
- Eventing
- Traffic routing
- Network resilience
- Security
- Automated placement
- Declarative deployment
- (Auto) scaling

Workload types:
- Stateful
- Singleton
- Batch Job
- Cron Job
- Stateless
- Serverless

High-level primitives:
- Build
- Serving
- Eventing

Low-level primitives:
- Gateway
- VirtualService
- DestinationRule
- ServiceEntry
- ReplicaSet
- CronJob
- StatefulSet
- Container
- Deployment
- ConfigMap

#redhat #rhsummit
Book Signing
Red Hat Dev Zone

Thursday May 9th
1:30pm - 2:30pm

https://k8spatterns.io
More Material

- Apache Camel K
  https://github.com/apache/camel-k
- Quick Camel K demo
  https://www.youtube.com/watch?v=I-g2xt-Qcb8
- Camel K Introduction Blog
  https://www.nicolaferraro.me/2018/10/15/introducing-camel-k
- Kubernetes enterprise integration patterns with Camel K video
  https://www.youtube.com/watch?v=51x9BewGCYA
- Camel K and Knative video
  https://www.youtube.com/watch?v=btf_e2GniXM
- Quarkus Camel Demo
  https://github.com/quarkusio/quarkus-quickstarts/tree/master/camel-java
Any Questions?

https://github.com/apache/camel-k
A ⭐ star on github is appreciated

Follow us on Twitter
@bibryam
@davsclaus
@ApacheCamel
THANK YOU

linkedin.com/company/Red-Hat
youtube.com/user/RedHatVideos
facebook.com/RedHatinc
twitter.com/RedHat