Establishing an EDA Platform on OpenShift

Pieter Malan

What is Event-Driven Architecture ?

To understand what is the meaning of Event-Driven Architecture, let see what the industry describe it as, by looking at different definitions given by vendors and analysts:

> An event-driven architecture is a software design pattern in which microservices react to changes in state, called events. Events can either carry a state (such as the price of an item or a delivery address) or events can be identifiers (a notification that an order was received or shipped, for example).
>
> — Google, https://cloud.google.com/eventarc/docs/event-driven-architectures

> Event-driven architecture (EDA) is a design paradigm in which a software component executes in response to receiving one or more event notifications. EDA is more loosely coupled than the client/server paradigm because the component that sends the notification doesn't know the identity of the receiving components at the time of compiling.
>
> — Gartner, https://www.gartner.com/en/information-technology/glossary/eda-event-driven-architecture#:~:text=Event%2Ddriven%20architecture%20(EDA)%20is%20a%20design%20paradigm%20in

> An event-driven architecture uses events to trigger and communicate between decoupled services and is common in modern applications built with microservices. An event is a change in state, or an update, like an item being placed in a shopping cart on an e-commerce website. Events can either carry the state (the item purchased, its price, and a delivery address) or events can be identifiers (a notification that an order was shipped).
>
> — AWS, https://aws.amazon.com/event-driven-architecture/

And for comparison, Red Hat's definition :-

> Event-driven architecture is a **software architecture** and model for **application design**. With an event-driven system, the **capture, communication, processing, and persistence** of events are the core structure of the solution. This differs from a traditional request-driven model.
>
> Many modern application designs are event-driven, such as customer engagement frameworks that must utilize customer data in real time. Event-driven apps can be created in **any programming language** because event-driven is a programming approach, not a language. Event-driven architecture enables minimal coupling, which makes it a good option for modern, distributed application architectures.
>
> An event-driven architecture is loosely coupled because event producers don't know which event consumers are listening for an event, and the event doesn't know what the consequences are of its occurrence.
>
> — Red Hat, https://www.redhat.com/en/topics/integration/what-is-event-driven-architecture

Part I: Infrastructure

Chapter 1. Storage Subsystem (Optional)

1.1. OpenShift Data Foundation Capabilities

OpenShift Data Foundation (ODF) is an integrated collection of storage and data services for OpenShift.

1.2. Role in EDA Platform

From an EDA Platform point of view, ODF gives us the capability to store information in a highly available replicated clustered environment on different types of storage types, block, file system and even Simple Storage System (S3), using native devices, or virtual devices offered by the underlying compute/cloud provider.

ODF Documentation

OpenShift ODF Documentation access.redhat.com/documentation/en-us/red_hat_OpenShift_data_foundation

YouTube Video

Installing ODF on Red Hat Virtualization using builtin Ovirt storage class - youtu.be/5kqyFDlyv54

Chapter 2. Serverless

2.1. Capabilities

OpenShift Serverless gives our EDA Platform the infrastructure to create Cloud Native Eventing and Serving capabilities.

On the serving side, it gives us access to automatic scaling and rapid deployment of applications.

Scaling includes traffic-splitting across different versions, flexible routing and scale to zero, which saves resources if deployment is not in use.

Eventing opens a host of features directly related to EDA processing, channels (publish/subscribe), broker (filter based subscription) and Cloud Events.

CloudEvents forms an important part of EDA, and acts as the internal payload definition, with typically HTTP as the communication protocol.

A sample CloudEvent:

```
{
    "specversion":"1.0",
    "type":"dev.knative.samples.helloworld",
    "source":"dev.knative.samples/helloworldsource",
    "id":"536808d3-88be-4077-9d7a-a3f162705f79",
    "data":{"msg":"Hello Knative2!"}
}
```

CloudEvents caters for the following features:

- Consistency
- Accesibility
- Protability

2.2. Role in EDA Platform

Serverless handles the internal eventing in a Cloud Native environment. Not only does it allow to scale on demand (or lack of), but it also provides a canonical message model, based on Cloud Events.

Serverless Documentation

OpenShift Documentation access.redhat.com/documentation/en-us/OpenShift_container_platform/4.10/html/serverless/index

CloudEvents

cloudevents cloudevents.io/

YouTube Video

Installing Serverless on OpenShift youtu.be/JQd4aqeBtc8

Chapter 3. AMQ Streams (Kafka)

Apache Kafka is an open-source distributed publish-subscribe messaging system for fault-tolerant real-time data feeds.AMQ Streams is based Apache Kafka.

3.1. AMQ Streams Capabilities

AMQ Streams is a containerized deployment managed by OpenShift supplied AMQ Streams operator. The AMQ Streams Operator are purpose-built with specialist operational knowledge to ease the management of Kafka.

- Optmized for Microservices and other streaming/eventing applications
- Messaging order guaranteed
- Horizontal scalability using OpenShift infrastructure
- Fault tolerant
- Quick access to high volumes of data

3.2. Role in EDA Platform

Red Hat AMQ Streams is a massively scalable, distributed, and high-performance data streaming platform based on the Apache Kafka project. AMQ Streams provides an event streaming backbone that allows microservices and other application components to exchange data with extremely high throughput and low latency.

Chapter 4. Conclusion

Part II: Integration

Chapter 5. Red Hat Integration - Camel K

5.1. Camel K Capabilities

Red Hat Integration's Camel K is a light weight integration framework, built from the upstream Apache Camel K runtime.

Camel K runs natively on top of OpenShift and takes advantage of Quarkus, as a runtime, and can be used in conjunction with Serverless to allow for autoscaling of deployments.

Camel K gives developers the following advantages:

- Predefined integration templates, called **Kamelets**, which allows for quick configuration of integrations. Kamelets are pure Camel DSL and can embody all the logic that allows to consume or produce data from public SaaS or enterprise systems and only expose to the final users a clean interface that describes the expected parameters, input and output.
- Code in the form of Java, Groovy, Kotlin, JavaScript and XML or Yaml DSL deployment for quick deployment.
- Container creation is automated by a code compilation optimized subsystem, called the integration platform.
- Multiple development tool support, for example Karavan a plugin for Visual Studio Code, and also a full CLI implementation

Supports multiple traits, which can be enabled globally on integration platform level, or per deployment. Traits are automated and usually only requires a couple configuration parameters to be enabled. An example of a trait is tracing,

Traits includes technologies like:

- 3scale (API Management), Swagger, OpenAPI
- Ingress Control
- Service Mesh, Serverless
- Jolokia, JVM
- Health endpoints, Logging, tracing, Prometheus
- Deployment configuration, mounts, pull secrets, route, tolerations, resources

5.2. Kamelets

Preconfigured Kamelets are shipped with Camel K, which includes a wide range of technologies.

Kamelet Catalog

To see the current list of available Kamelets, see the Apache Camel Kamelets catalog: camel.apache.org/camel-kamelets/

On top of all the features like Kamelets and traits, Camel K also support standard Camel Components.

Camel Components

To see the current list of available components, see the Apache Camel Component Documentation: camel.apache.org/components/

Chapter 6. Debezium

Debezium enables you to capture events from a database, when there is any changes on data.

Debezium is built on top of Kafka, and records the history of database changes (Change Data Capture aka. CDC) in Kafka logs. Even if your Debezium configuration is not active, events will be captured as soon as the configuration is restarted.

Debezium is based on log mining, and requires no changes to your data models.

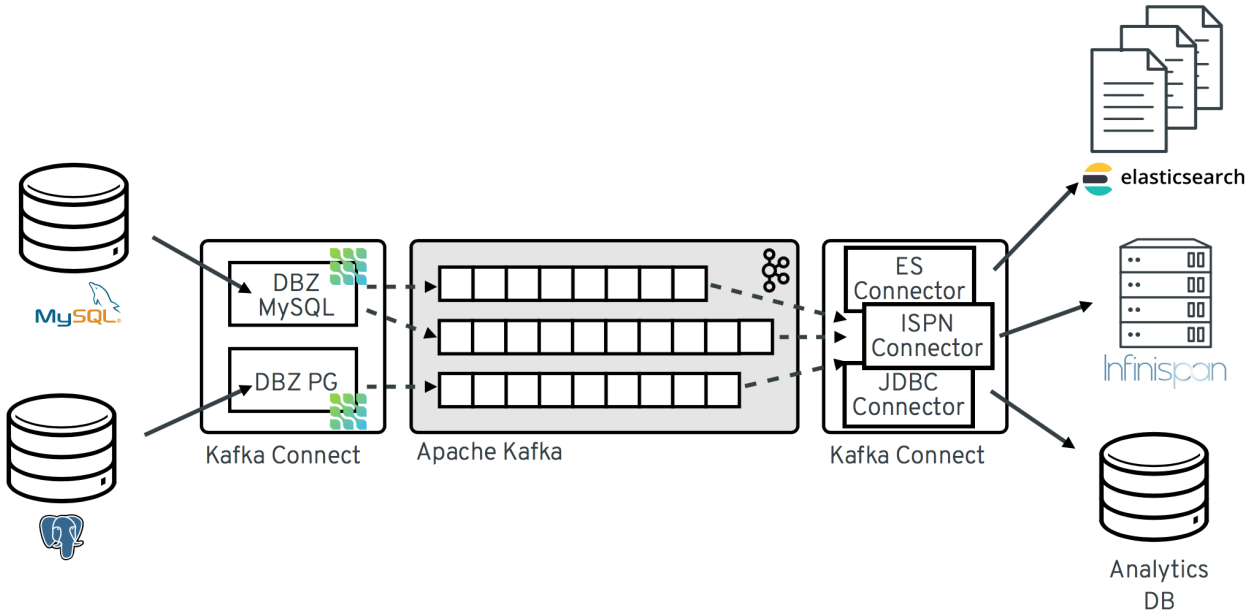

Figure 1. Debezium Arhitecture

6.1. Supported Databases

Debezium supports the following databases:

- DB2
- MongoDB
- Postgresql
- Oracle
- SQLServer

6.2. Implementation

6.2.1. Database Configration

Database configuration is required, and depends on the flavor of the database. Debezium utilizes built in functionality of a specific database.

For example, for Oracle, you have to configure snapshots.

6.2.2. Debezium Connector

For the connector configuration, you specify:

- Database connection information
- Kafka connection information
- Filters, a set of schemas, tables and columns to include/exclude
- Masking, optionally hide sensitive data
- Optionally converters, how the information is pusblished to Kafka, with support of JSON, Protobuf and AVRO.
- Optionally value converters stored in the Service Registry

6.3. Role in EDA Platform

Most third party applications provides some sort of method to subscribe/publish events, but it is not always true in all cases. In these special cases, you can tap into the power of the database to capture changes on a database level using Debezium, without any code changes to the application, or the underlying database schema.

Debezium can also be used to enable events, in applications where it is difficult to retrofit event publishers.

A bit more work is required, as a typical database event requires some logic to recreate an event. For example, a new order event, consists of the order information and associated order lines. The new order event can be based on a table order, but the Debezium event needs the logic the extract the full event, including querying the order lines from the database.

Chapter 7. Quarkus

Chapter 8. Service Registry

Chapter 9. Consclusion

Part III: Sample EDA Platform and Demo implementation

Chapter 10. Operators

Creation of the infrastructure, and Camel K, we are going to use OpenShift operators.

10.1. Openshift Data Foundation (Optional)

Installation is straight forward using the OpenShift Data Foundation Operator, which includes a wizard to create a storage subsystem.

During the wizard you are presented with a choice of using an existing storage class, local storage, or connecting to an existing ceph cluster. You also have the option to taint the nodes, to be dedicated storage nodes.

For in depth information on installing ODF see the documentation.

Quick Start for OpenShift Data Foundation

console-openshift-console.apps.clustername.domain.com/quickstart?quickstart=odf-install-tour

10.2. Serverless Operator

Serverless Quick Start - OpenShift console

Install the OpenShift Serverless Operator

```
https://console-OpenShift-
console.apps.*clustername*.*yourdomain*.com/quickstart?quickstart=install-
serverless
```

Chapter 11. x

Part IV: Index

Index

C

Cloud Events, 5
CloudEvent, 5
CloudEvents, 5

O

OpenShift Data Foundation Operator, 18
OpenShift Serverless, 5

Chapter 12. YouTube Play List

YouTube Play List

Establish an EDA Platform on OpenShift - www.youtube.com/watch?v=5kqyFDlyv54&list=PLJhX_m4u3PRGoSjHpHJrUHWGEzVHz5G6P