Establishing an EDA Platform on Openshift

Pieter Malan, Robert van Tilburg

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.

1.3. Installation

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.

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

Serverless Quick Start - OpenShift console

Install the OpenShift Serverless Operator

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

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. Installation

A deployment of Kafka components to an OpenShift cluster using AMQ Streams is highly configurable through the application of custom resources. Custom resources are created as instances of APIs added by Custom resource definitions (CRDs) to extend OpenShift resources.

3.2.1. 1. Operator install

The AMQ Streams Operator can be install on a global

Chapter 4. Consclusion

Part II: Integration

Chapter 5. Camel K

5.1. Installation

Chapter 6. Debezium

Chapter 7. Quarkus

Chapter 8. Service Registry

Chapter 9. Conclusion

Part III: Monitoring

Chapter 10. x