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: Openshift Container Overview

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, or CloudEvents, referring to the product.

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
- Portability

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 CloudEvents.

Serverless Documentation

OpenShift Documentation

```
https://access.redhat.com/documentation/en-
us/OpenShift_container_platform/4.10/html/serverless/index
```

CloudEvents

cloudevents

```
https://cloudevents.io/
```

Chapter 3. Distributed Tracing Platform

3.1. Capabilities

Chapter 4. Conclusion

Part II: Application Foundation Overview

Chapter 5. 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.

5.1. 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

5.2. 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 6. Red Hat Integration - Camel K

6.1. 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

6.2. 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 7. 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.

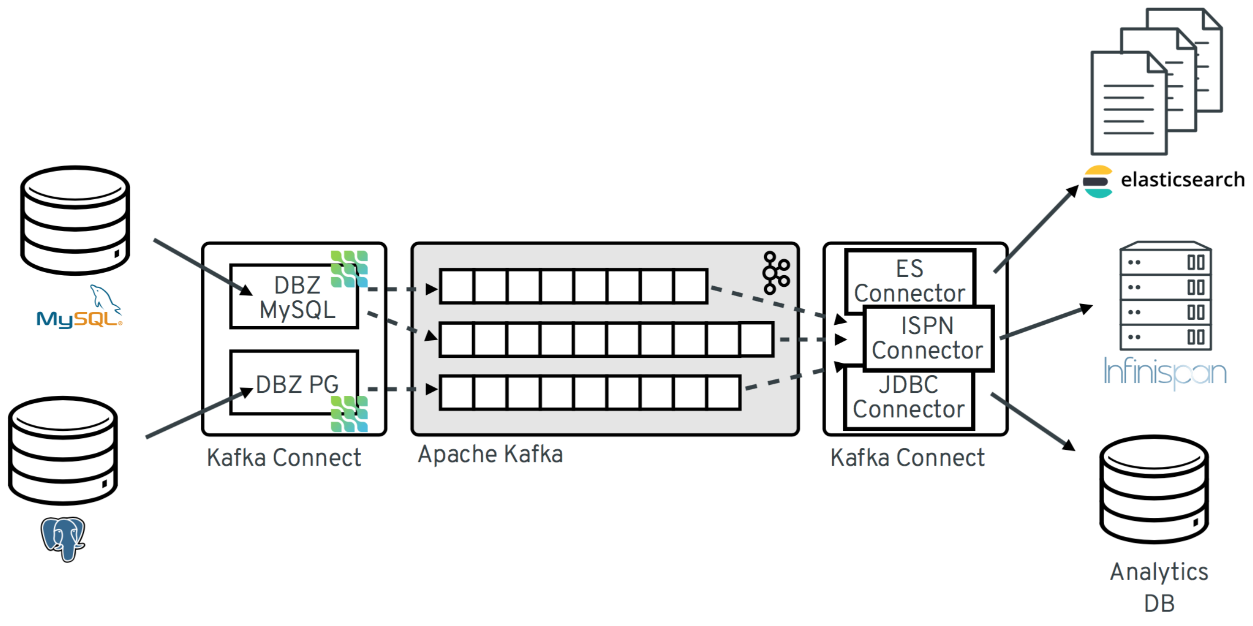


Figure 1. Debezium Arhitecture

7.1. Supported Databases

Debezium supports the following databases:

- DB2
- MongoDB
- Postgresql
- Oracle
- SQLServer

7.2. Implementation

7.2.1. 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.

7.2.2. 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

7.3. 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 8. Quarkus

Chapter 9. Service Registry

Chapter 10. Consclusion

Part III: Third Party Requirements

Chapter 11. Maven Repository

11.1. Mirrors

It is advised to use a local maven mirror repository to speed up deployments.

Camel K, Quarkus, supporting components of AMQ, all use maven artifacts.

The supported versions of Red Hat supplied artifacts all comes from the following repositories.

- maven.repository.redhat.com/ga
- maven.repository.redhat.com/earlyaccess/all

Other used repositories include:

- packages.confluent.io/maven/

Chapter 12. Visual Studio Code

Visual Studio Code is a downstream implementation of VSCodium.

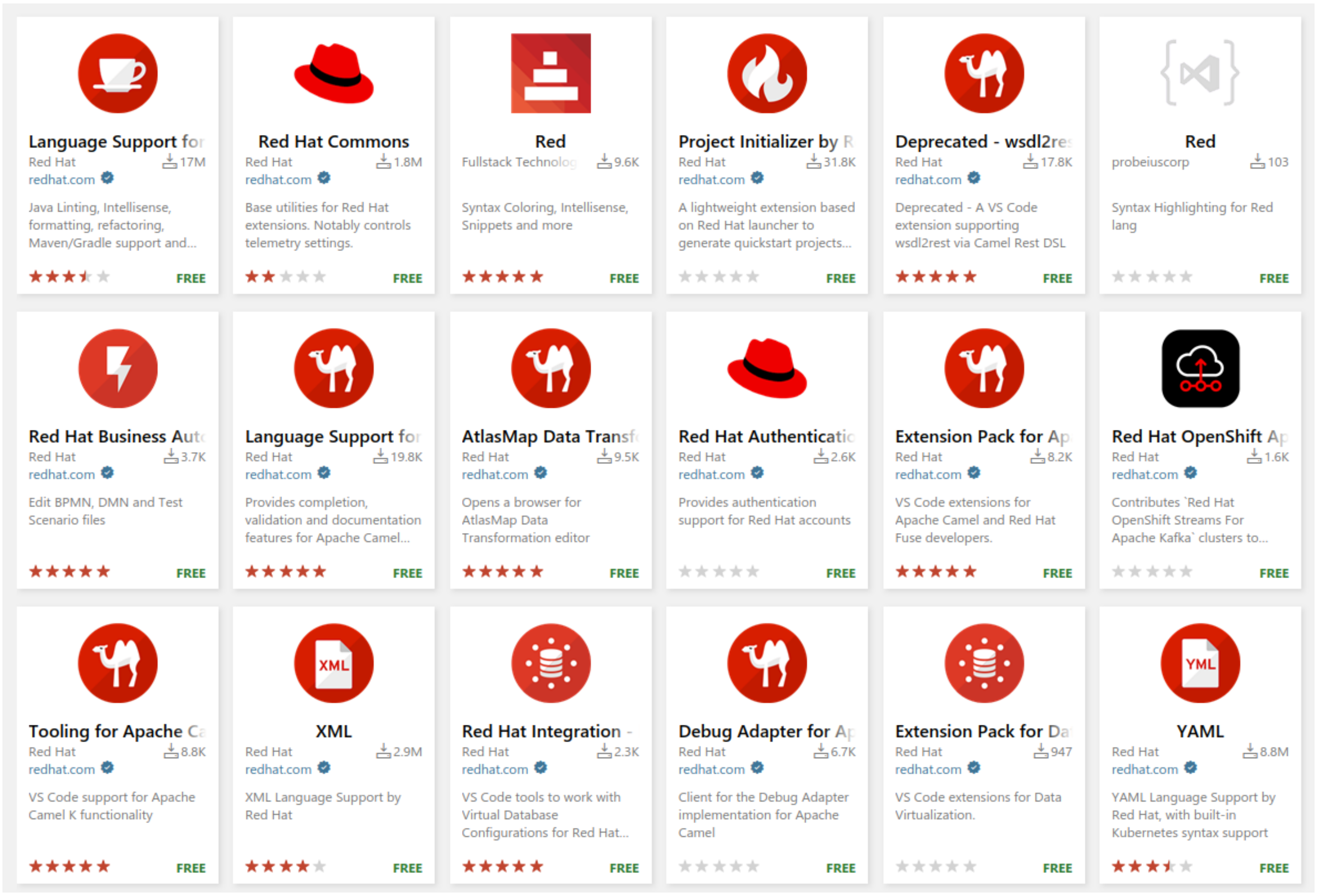
code.visualstudio.com/

vscodium.com/

12.1. Plugins

12.1.1. Red Hat Plugins

Lost of plugins are available:



Language Support for
Red Hat 17M
redhat.com
Java Linting, Intellisense, formatting, refactoring, Maven/Gradle support and...
FREE

Red Hat Commons
Red Hat 1.8M
redhat.com
Base utilities for Red Hat extensions. Notably controls telemetry settings.
FREE

Red
Fullstack Technolog 9.6K
Syntax Coloring, Intellisense, Snippets and more
FREE

Project Initializer by R
Red Hat 31.8K
redhat.com
A lightweight extension based on Red Hat launcher to generate quickstart projects...
FREE

Deprecated - wsdl2re
Red Hat 17.8K
redhat.com
Deprecated - A VS Code extension supporting wsdl2rest via Camel Rest DSL
FREE

Red
probeiuscorp 103
Syntax Highlighting for Red lang
FREE

Red Hat Business Aut
Red Hat 3.7K
redhat.com
Edit BPMN, DMN and Test Scenario files
FREE

Language Support for
Red Hat 19.8K
redhat.com
Provides completion, validation and documentation features for Apache Camel...
FREE

AtlasMap Data Transf
Red Hat 9.5K
redhat.com
Opens a browser for AtlasMap Data Transformation editor
FREE

Red Hat Authenticatic
Red Hat 2.6K
redhat.com
Provides authentication support for Red Hat accounts
FREE

Extension Pack for Ap
Red Hat 8.2K
redhat.com
VS Code extensions for Apache Camel and Red Hat Fuse developers.
FREE

Red Hat OpenShift Ap
Red Hat 1.6K
redhat.com
Contributes `Red Hat OpenShift Streams For Apache Kafka` clusters to...
FREE

Tooling for Apache C
Red Hat 8.8K
redhat.com
VS Code support for Apache Camel K functionality
FREE

XML
Red Hat 2.9M
redhat.com
XML Language Support by Red Hat
FREE

Red Hat Integration -
Red Hat 2.3K
redhat.com
VS Code tools to work with Virtual Database Configurations for Red Hat...
FREE

Debug Adapter for Ap
Red Hat 6.7K
redhat.com
Client for the Debug Adapter implementation for Apache Camel
FREE

Extension Pack for Da
Red Hat 947
redhat.com
VS Code extensions for Data Virtualization.
FREE

YAML
Red Hat 8.8M
redhat.com
YAML Language Support by Red Hat, with built-in Kubernetes syntax support
FREE

Figure 2. Red Hat supply a range of plugins for Microsoft Visual Studio Code

12.1.2. Other Plugins

12.1.2.1. Karavan - Camel K

marketplace.visualstudio.com/publishers/camel-karavan

Chapter 13. Conclusion

Part IV: Sample EDA Platform and Demo implementation

Chapter 14. Supporting Services

14.1. Installing Operators

Creation of the infrastructure, and Camel K, we are going to use OpenShift operators. Most of these operators provides a quick start, which we are going to use for installation.

To access the quick starts available, from the Openshift Consolse, in Administrator perspective, got to Home→Overview, and under "Getting Started resources", you will see the link "View all quick starts".

14.1.1. 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

14.1.2. Serverless Operator

Configuring the OpenShift Serverless environment requires the installation of the OpenShift Serverless Operator, and the configuration of the KNative and KServing deployments. For our needs we are going to use defaults, but keep in mind that customized configuration can be done.

Use the "Install the Openshift Serverless Operator" Quick start wizard to complete the installation.

Serverless Quick Start - OpenShift console

Install the OpenShift Serverless Operator using the guided wizard offered by a Quick Start

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

14.1.3. Camel K

To install the Red Hat Camel K Operator:

- From the **Administrator** perspective, got to the OperatorHub from the Operators section of the navigation pane.
- In the **All Items** filter, type in **Camel K**
- Click the Red Hat Integration - Camel K tile, to open the Operator details
- At the top of Red Hat Integration - Camel K Operator panel, click Install
- Leave all fields the defaults and click Install
- After a few minutes, you will get a confirmation that Operator is ready for use.

> If you running into an issue deploying Camel K Operator, where the deployment fails with a ImagePullBackOff error. You can view the progress on the issue at the following link: issues.redhat.com/browse/ENTESB-19495?focusedCommentId=20637854&page=com.atlassian.jira.plugin.system.issuetabpanels%3Acomment-tabpanel#comment-20637854
>
> As a workaround, we are going to patch the Camel K Cluster Service Version, to pull the operator image based on version, not hash:
>
> - Operators → Installed Operators → Red Hat Integration - Camel K → YAML Tab:
> - On Line: 423 - replace registry.redhat.io/integration/camel-k-rhel8-operator:1.6.8
> - Save the change
> - Delete the camel-k-operator deployment, under Deployments, project openshift-operators to force a redploy, with corrected iamge.

14.1.4. Red Hat Integration - Service Registry

To install the Red Hat Service Registry Operator:

- From the **Administrator** perspective, got to the OperatorHub from the Operators section of the navigation pane.
- In the **All Items** filter, type in **registry operator**
- Click the Red Hat Integration - Service Registry tile, to open the Operator details

- At the top of Red Hat Integration - Service Registry Operator panel, click Install
- Leave all fields the defaults and click Install
- After a few minutes, you will get a confirmation that Operator is ready for use.

14.1.5. Red Hat Openshift distributed tracing platform

To install the Red Hat distributed tracing platform:

- From the **Administrator** perspective, got to the OperatorHub from the Operators section of the navigation pane.
- In the **All Items** filter, type in **distributed tracing**
- Click the Red Hat OpenShift distributed tracing platform tile, to open the Operator details
- At the top of Red Hat OpenShift distributed tracing platform panel, click Install
- Leave all fields the defaults and click Install
- After a few minutes, you will get a confirmation that Operator is ready for use.

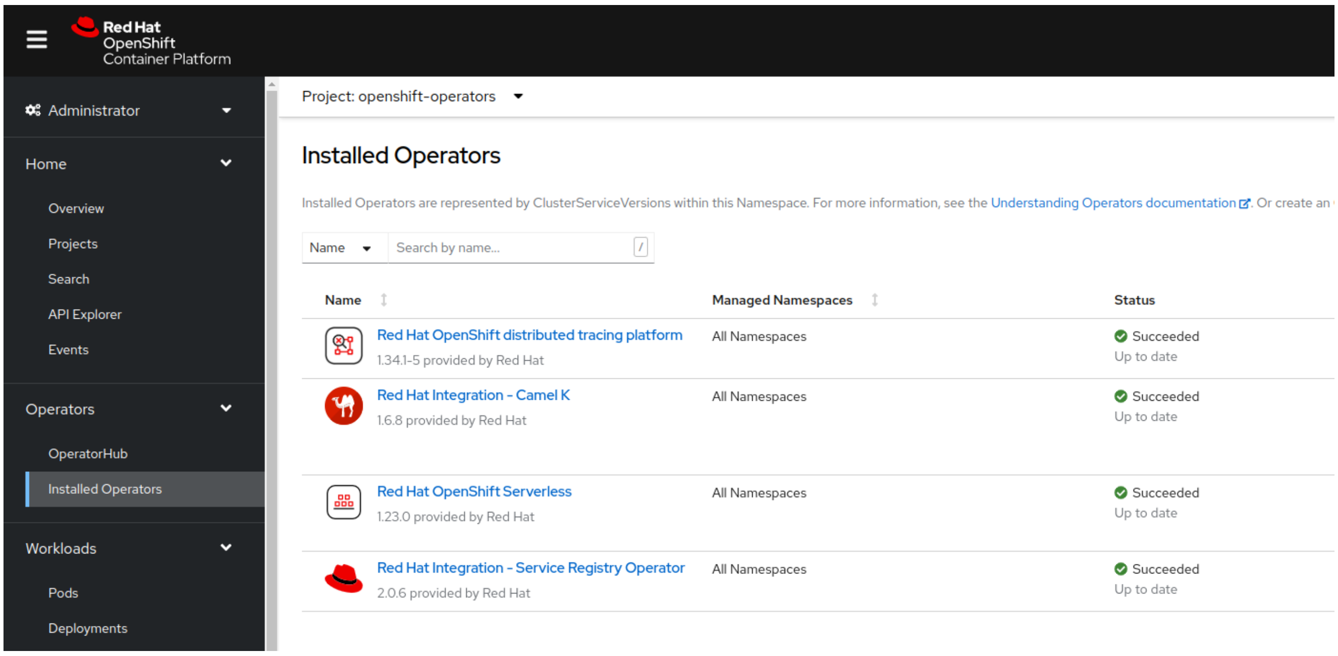
14.1.6. Red Hat Integration - AMQ Streams

To install the Red Hat Integration - AMQ Streams:

- From the **Administrator** perspective, got to the OperatorHub from the Operators section of the navigation pane.
- In the **All Items** filter, type in **AMQ**
- Click the Red Hat Integration - AMQ Streams tile, to open the Operator details
- At the top of Red Hat Integration - AMQ Streams panel, click Install
- Leave all fields the defaults and click Install
- After a few minutes, you will get a confirmation that Operator is ready for use.

14.1.7. Summary

After the process of installing required operators, you should be able to confirm the Installed Operators, by looking at the project openshift-operators, and Installed Operators.



Project: openshift-operators

Installed Operators

Installed Operators are represented by ClusterServiceVersions within this Namespace. For more information, see the Understanding Operators documentation. Or create an

Name | Search by name...

Name	Managed Namespaces	Status
Red Hat OpenShift distributed tracing platform 1.34.1-5 provided by Red Hat	All Namespaces	Succeeded Up to date
Red Hat Integration - Camel K 1.6.8 provided by Red Hat	All Namespaces	Succeeded Up to date
Red Hat OpenShift Serverless 1.23.0 provided by Red Hat	All Namespaces	Succeeded Up to date
Red Hat Integration - Service Registry Operator 2.0.6 provided by Red Hat	All Namespaces	Succeeded Up to date

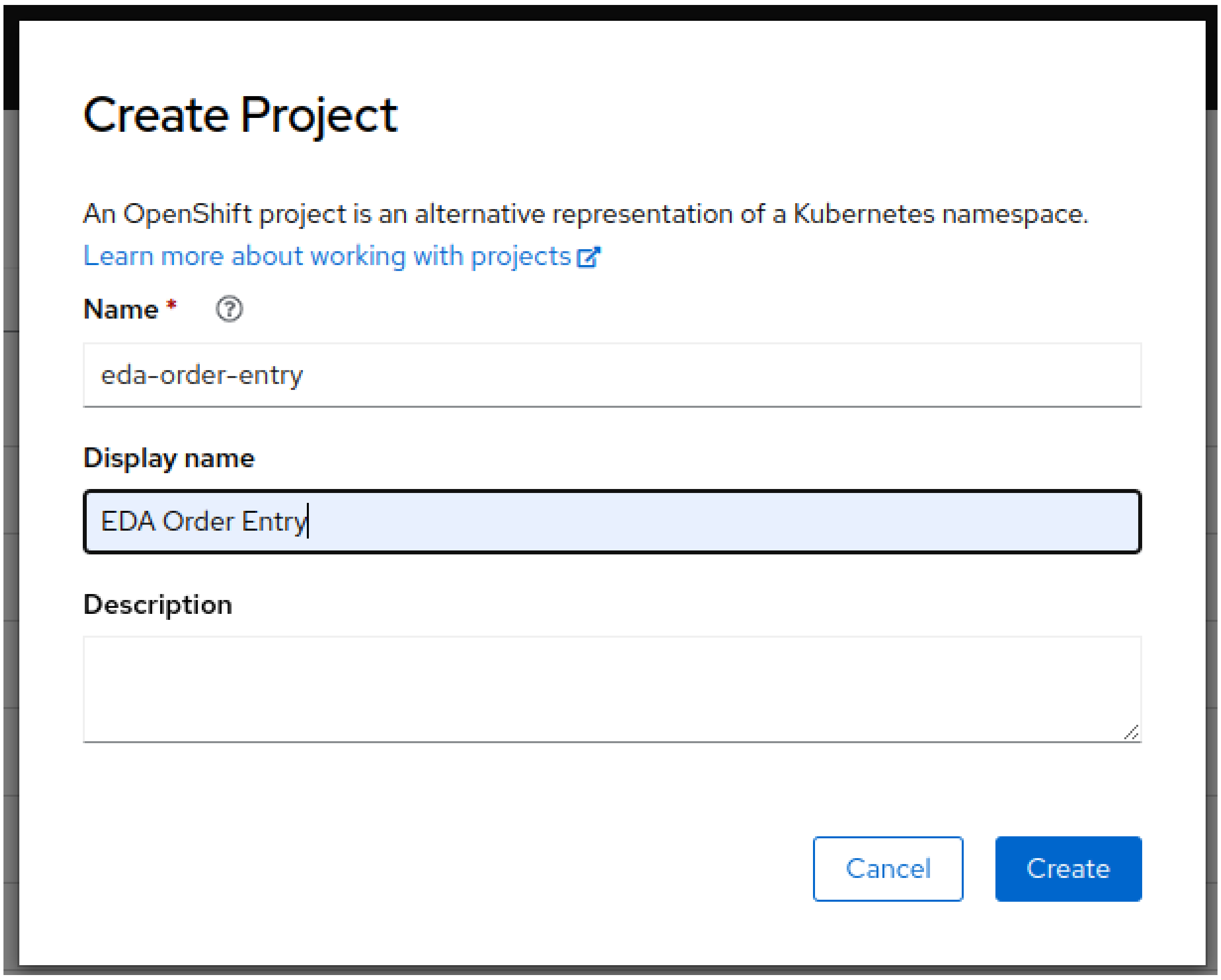
Figure 3. Installed Operators Summary

14.2. Serverless Configuration

14.2.1. Project

First we need to create a project to work in. Since the Serverless Operator is installed on a cluster level, we don't need any special configuration in a project to utilize Serverless Components.

- From the **Administrator** perspective, got to the Projects from the Home section of the navigation pane.
- **Create project**, called **eda-order-entry**



Create Project

An OpenShift project is an alternative representation of a Kubernetes namespace.
Learn more about working with projects

Name *: eda-order-entry

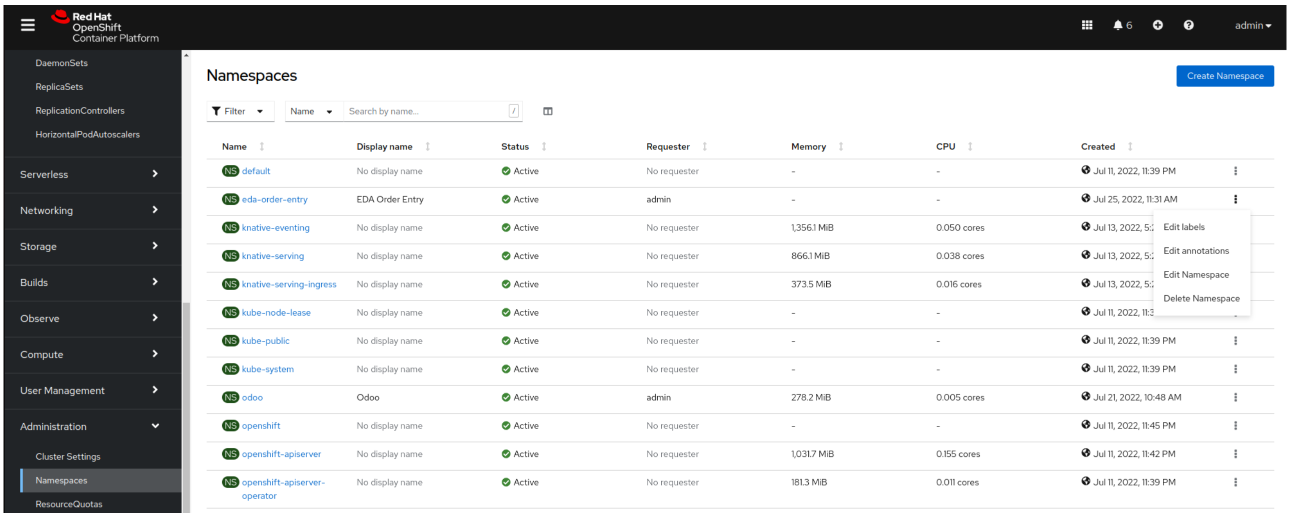
Display name: EDA Order Entry

Description:

14.2.2. Enable Knative Bindings

Next we have to make the project aware that it has the capability to bind to Knative services:

- From the **Administrator** perspective, got to the **Namespaces** from the **Administration** section of the navigation pane.
- Next to our eda-order-entry namespace, click the 3 dots, and click **Edit Labels**

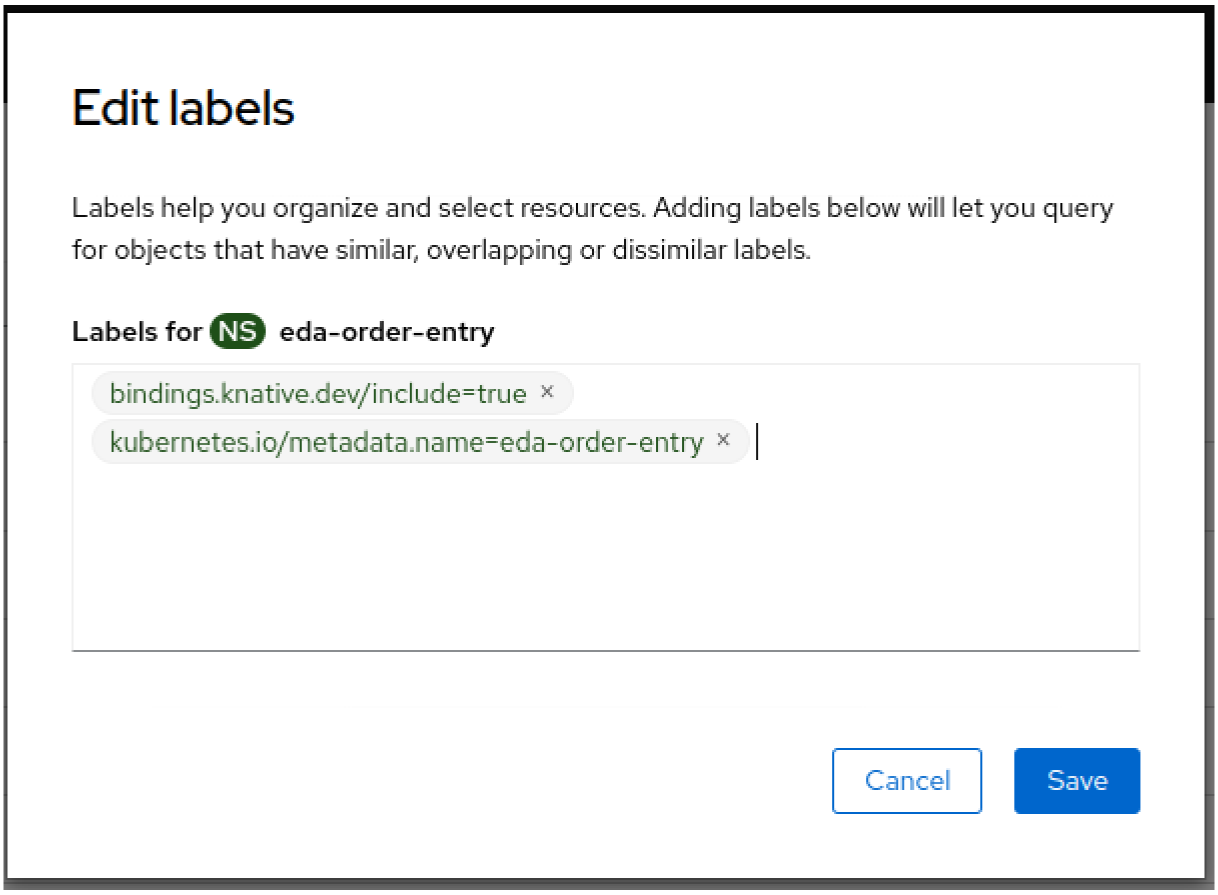


Namespaces

Name	Display name	Status	Requester	Memory	CPU	Created
NS default	No display name	Active	No requester	-	-	Jul 11, 2022, 11:39 PM
NS eda-order-entry	EDA Order Entry	Active	admin	-	-	Jul 25, 2022, 11:31 AM
NS knative-eventing	No display name	Active	No requester	1,356.1 MiB	0.050 cores	Jul 13, 2022, 5:
NS knative-serving	No display name	Active	No requester	866.1 MiB	0.038 cores	Jul 13, 2022, 5:
NS knative-serving-ingress	No display name	Active	No requester	373.5 MiB	0.016 cores	Jul 13, 2022, 5:
NS kube-node-lease	No display name	Active	No requester	-	-	Jul 11, 2022, 11:3
NS kube-public	No display name	Active	No requester	-	-	Jul 11, 2022, 11:39 PM
NS kube-system	No display name	Active	No requester	-	-	Jul 11, 2022, 11:39 PM
NS odoo	Odoo	Active	admin	278.2 MiB	0.005 cores	Jul 21, 2022, 10:48 AM
NS openshift	No display name	Active	No requester	-	-	Jul 11, 2022, 11:45 PM
NS openshift-apiserver	No display name	Active	No requester	1,031.7 MiB	0.155 cores	Jul 11, 2022, 11:42 PM
NS openshift-apiserver-operator	No display name	Active	No requester	181.3 MiB	0.011 cores	Jul 11, 2022, 11:39 PM

- Add the label:

```
bindings.knative.dev/include=true
```



Edit labels

Labels help you organize and select resources. Adding labels below will let you query for objects that have similar, overlapping or dissimilar labels.

Labels for NS eda-order-entry

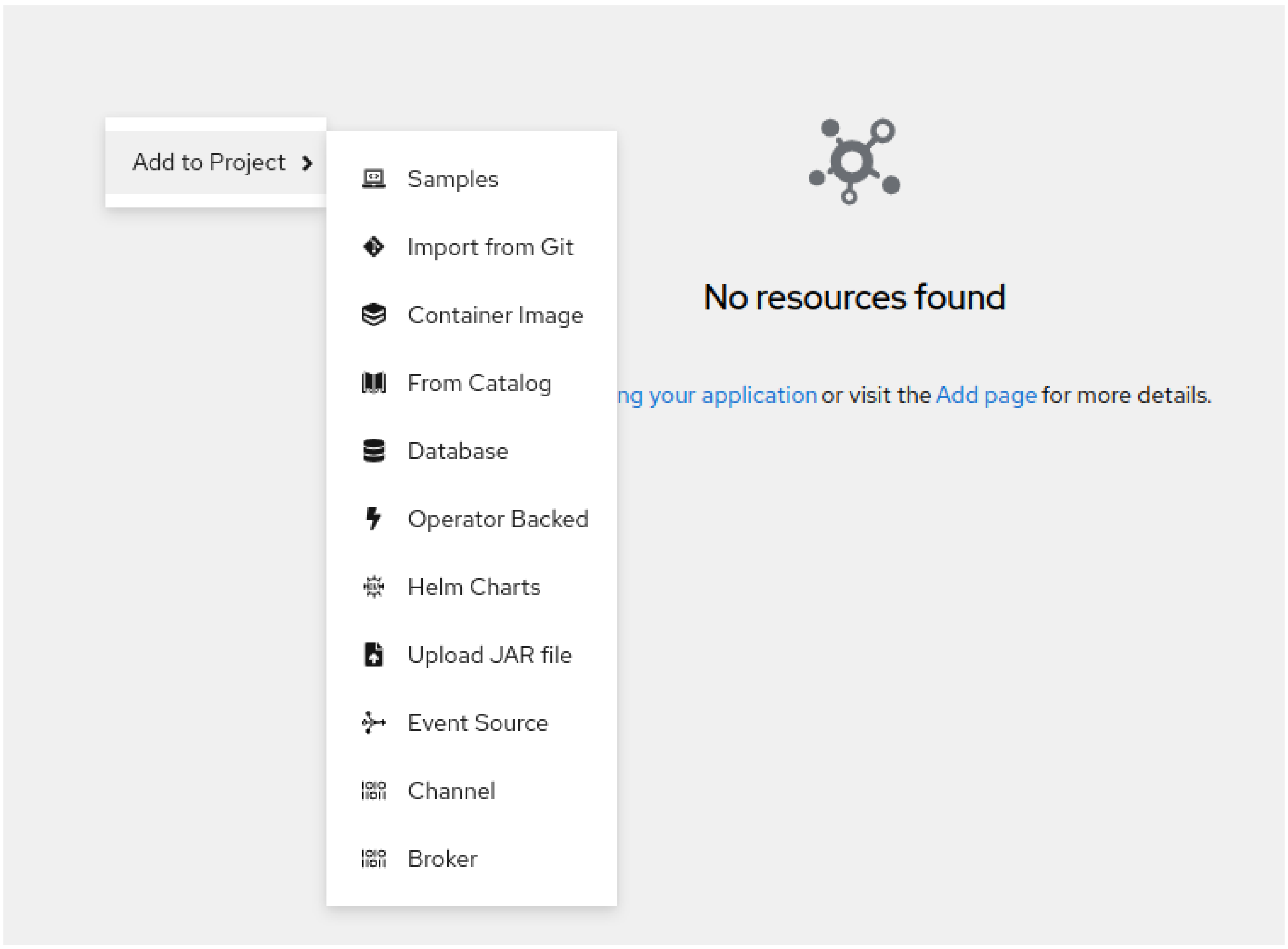
bindings.knative.dev/include=true ×
kubernetes.io/metadata.name=eda-order-entry ×

- Click **Save**

14.2.3. Creating a Channel and Broker

In order to add Knative components, you have the option to use the +Add and also to right click on the topology view.

- Make sure you are in the correct Project, and the **Developer Perspective**
- Select **Topology** in the navigation pane
- Right click on the blank canvase, stating **No resources found**



- Select **Add to Project**
- Add Channel, keep the defaults and click **Create**
- Again, right click on the canvas, and add a Broker
- Keep the defaults and click **Create**

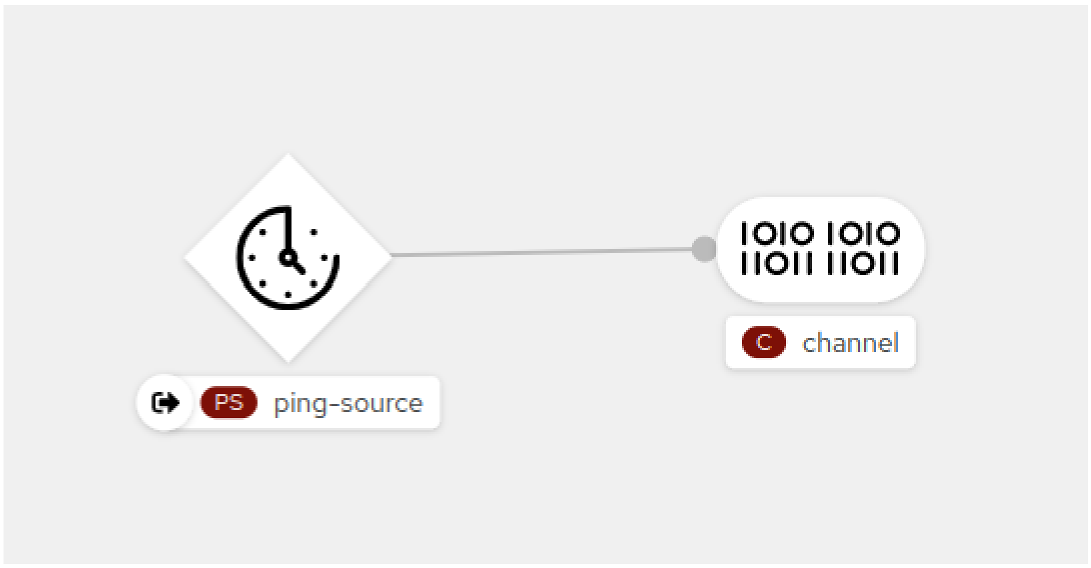
14.2.4. Source

To define a source, we follow very similar steps as we did for adding a channel and broker.

- Right click on the canvas, **Add to Project**, and **Event Source**
- Click on the **PingSource** Panel
- In the PingSource pop-up window click **Create Event Source**
- In the **Create Event Source** form, keep all the defaults, add your **Data**, {"message": "Hello

world!"}, and specify the **Schedule** as */1 * * * *

- Make sure the **Input Target**'s **Resource** is pointing to the channel.
- Click on **Create**



Feel free to move the components around.

14.2.5. Target

For a target we are going to utilize a container to log the CloudEvent, and for demonstration purposes, we are alo going to make the container serverless.

- Again, from the **Developer** perspective, click +**Add**
- Click on the **Import YAML**
- Copy the following YAML

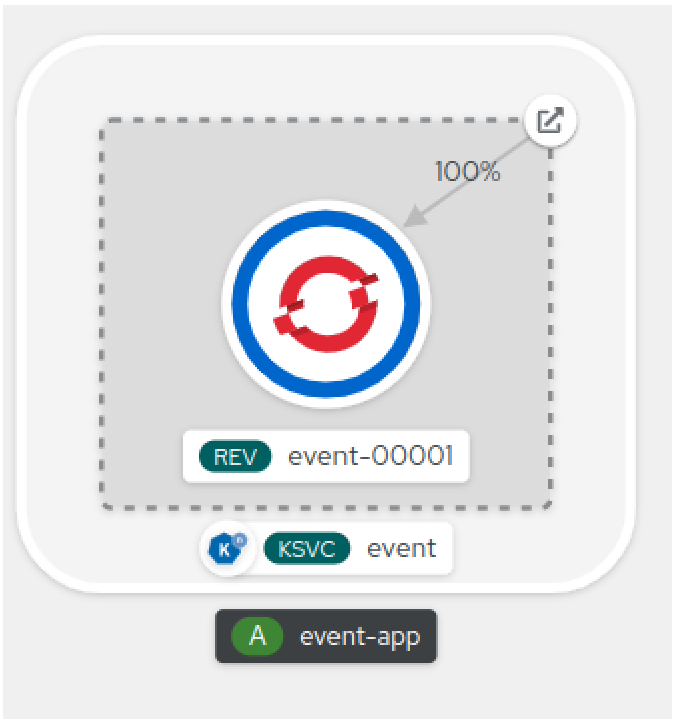
Event Display Service

raw.githubusercontent.com/pmalan-rh/EDAOpenShift-code/main/servlerless/event-display.yaml

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: event-display
spec:
  template:
    spec:
      containers:
        - image: quay.io/openshift-knative/knative-eventing-sources-event-
display:latest
```

- Click **Create**

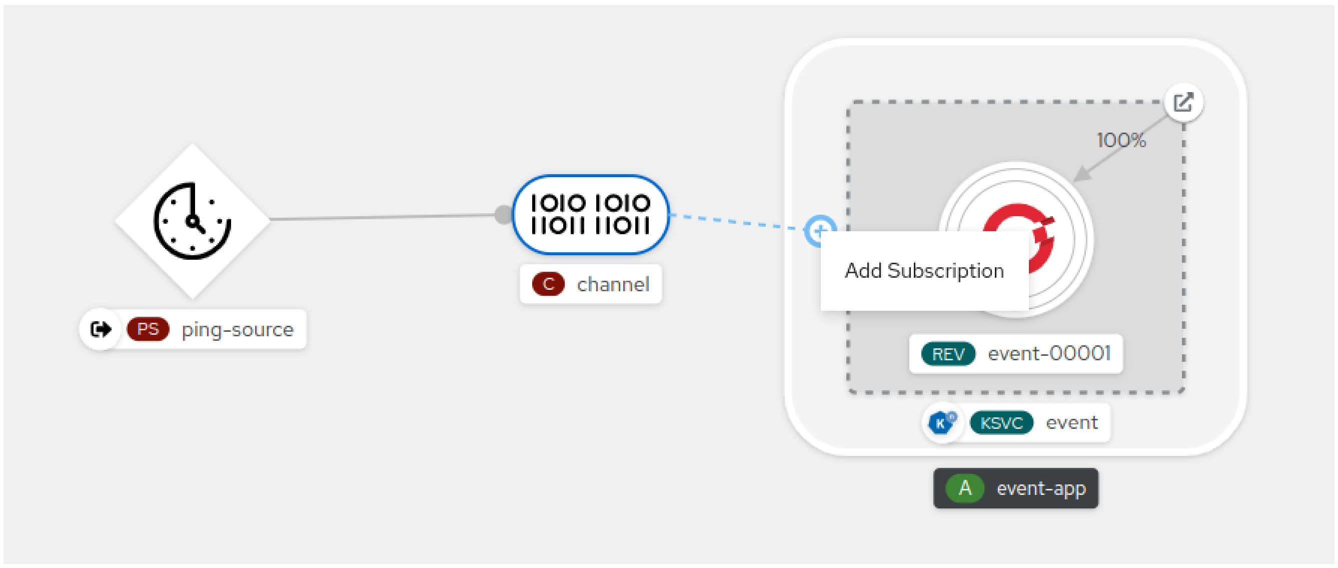
It will take a while to deploy, and the following image will appear when deployment is completed:



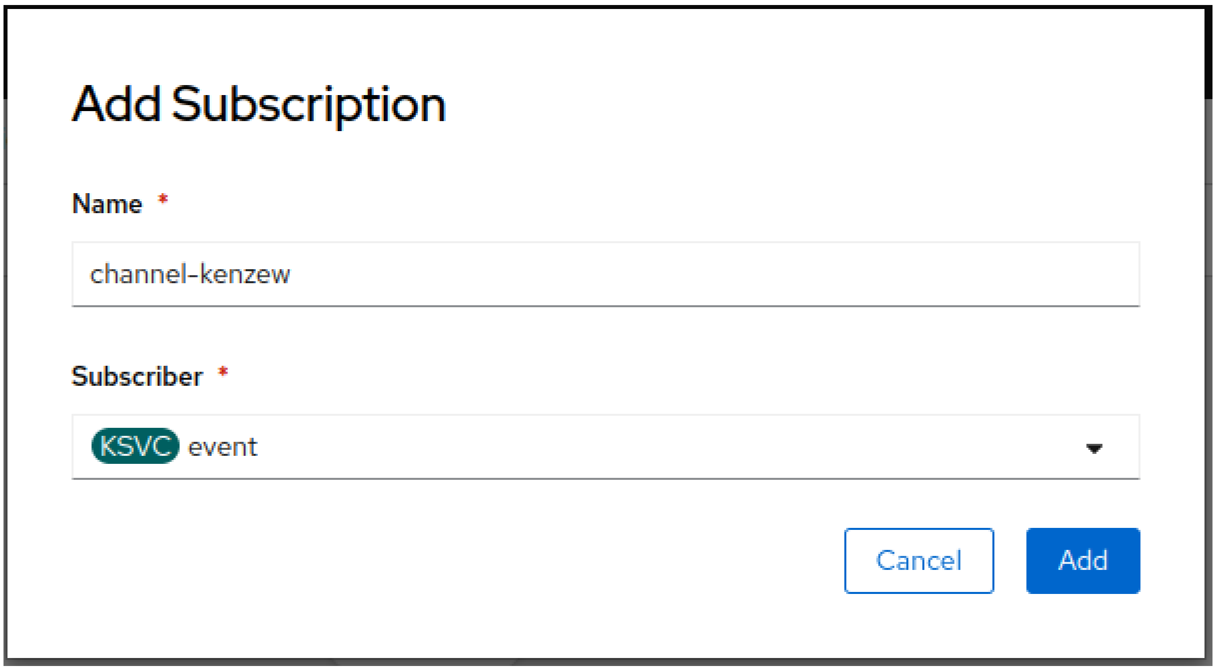
14.2.6. Connecting Knative Events

The last step is to hook up the event sink, in our case our serverless container, to the channel.

- Hover your mouse icon over the channel. As soon as the blue arrow appears, grab the arrow and point it to the outer box (containing the **KSVC event** label)



- The following pop-up will appear:


Add Subscription

Name *

channel-kenzew

Subscriber *

KSVC event

- Leave the defaults and click on **Add**

14.2.7. Verification

To verify that the events are reaching the event logger:

- In the **Developer** perspective, navigate to **Topology**
- Click on the **event-display** Knative service
- On the pop-up window, on the right hand, select the **Resources** tab, under **event-display** heading
- Under the **Pods**, click on **View logs**

The pod's log console will be displayed, and you should be able to see the cloudevents being generated.

Example CloudEvent in Log

```
☁️ cloudevents.Event
Validation: valid
Context Attributes,
specversion: 1.0
type: dev.knative.sources.ping
source: /apis/v1/namespaces/eda-order-entry/pingsources/ping-source
id: d998378c-9973-4c06-a4e6-d0a534a1be01
time: 2022-07-26T17:15:00.454760422Z
Data,
{"name":"Hello World!"}
```

14.2.8. Experimentation

Since our events fires too close to each other, the Knative service never scales down to zero.

To see the effect of serverless deployments, you need to change the frequency of events generated by the ping-source. The default time out for a serverless deployment is 600 seconds. Changing the interval to 10 minutes would cause our event-display to time out, and scale to zero.

Example schedule change for ping-source

```
spec:
  data: '{"name":"Hello World!"}'
  schedule: '*/10 * * * *'
```

14.3. AMQ Streams Configuration

14.3.1. Project

Let us create a project to work in. Since the AMQ Streams Operator is installed on a cluster level, we don’t need any special configuration in a project to utilize ANQ Streams Components.

- From the **Administrator** perspective, got to the **Projects** from the Home section of the navigation pane.
- **Create project**, called **amq-streams**

14.3.2. AMQ Streams Instance

Now to instantiate a Kafka cluster, using AMQ Streams Operator:

- Navigate to **Installed Operators** under **Operators**
- Confirm that the selected project is **amq-streams**
- Click on the **Red Hat Integration - AMQ Streams Operator**
- Click on the **Kafka** panel, **Create instance**
- Switch to **YAML** view and paste the following YAML:

Raw yaml

raw.githubusercontent.com/pmalan-rh/EDAOpenShift-code/main/amq-streams/amq-streams.yaml

```
kind: Kafka
apiVersion: kafka.strimzi.io/v1beta2
metadata:
  name: my-cluster
  namespace: amq-streams
spec:
  kafka:
    version: 3.1.0
    replicas: 3
    listeners:
      - name: plain
        port: 9092
        type: internal
        tls: false
      - name: tls
        port: 9093
        type: internal
        tls: true
    config:
      offsets.topic.replication.factor: 3
      transaction.state.log.replication.factor: 3
      transaction.state.log.min.isr: 2
      default.replication.factor: 3
      min.insync.replicas: 2
      inter.broker.protocol.version: '3.1'
    storage:
      deleteClaim: true
      size: 5Gi
      type: persistent-claim
  zookeeper:
    replicas: 3
    storage:
      deleteClaim: true
      size: 5Gi
      type: persistent-claim
  entityOperator:
    topicOperator: {}
    userOperator: {}
```

- Click on **Create**

14.4. Camel K

14.4.1. Integration Platform

The integration platform configuration controls the building and implementation of creating integrations.

For our demo, we are going to utilize a maven repository configured with mirrors, and we have to reference the maven's settings.xml in the integration platform.

We also going to change a couple of integration runtime traits. One setting, the logging color, cause a really bad output on an Openshift console, so we are going to change that, and the other setting is to enable tracing across all integrations and deployments, within the project.

14.4.1.1. Create Maven settings.xml and configmap

It is highly recommended that you setup a local maven repository for use by Camel K.

Several third party tools are available for setting up a mirror repository server, even with OpenShift Operators available like Nexus and Artifactory.

Here is a sample configuration to start with, containing all the required repositories for Camel K.

Sample settings.xml

raw.githubusercontent.com/pmalan-rh/EDAOpenShift-code/main/camel-k/settings.xml

From terminal, login into OpenShift and create the ConfigMap:

```
oc login -u _user_
cd ~/.m2
oc create configmap maven-settings --from-file=settings.xml -n eda-order-entry
```

14.4.1.2. Create Integration Platform

- Navigate to **Installed Operators** under **Operators**
- Confirm that the selected project is **eda-order-entry**
- Click on the **Red Hat Integration - Camel K**
- Click on the **Integration Platform** panel, **Create instance**
- Paste the following YAML:

Raw yaml

raw.githubusercontent.com/pmalan-rh/EDAOpenShift-code/main/camel-k/integrationplatform.yaml

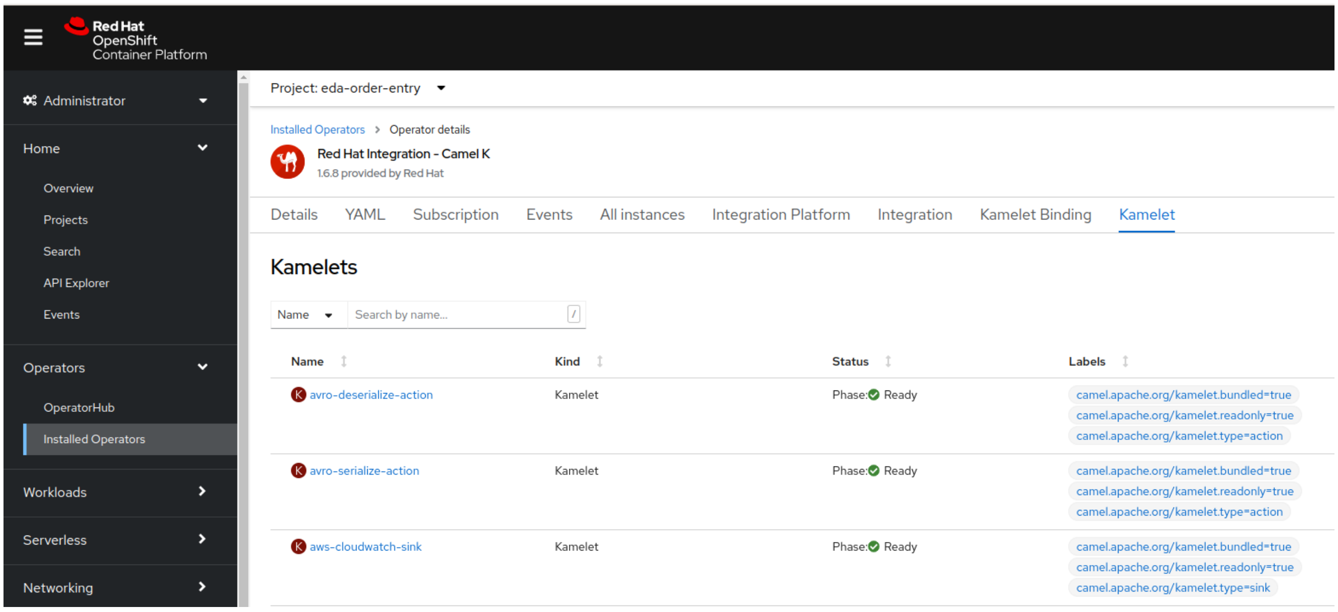
```
apiVersion: camel.apache.org/v1
kind: IntegrationPlatform
metadata:
  labels:
    app: camel-k
  name: camel-k
  namespace: eda-order-entry
spec:
  build:
    maven:
      localRepository: /tmp/artifacts/m2
      settings:
        configMapKeyRef:
          key: settings.xml
          name: maven-settings
    registry: {}
  kamelet: {}
  resources: {}
  traits:
    logging:
      configuration:
        color: false
    tracing:
      configuration:
        property:
          enabled: true
```

- Click on **Create**

14.4.2. Verification

To verify the installation of Camel K, is to look at the list Kamelets:

- Form **Administrator** perspective, go to **Installed Operators**, and select **Red Hat Integration - Camel K**
- Select the **Kamelet** tab
- You will see a list of kamelets, if install worked.



Name	Kind	Status	Labels
avro-deserialize-action	Kamelet	Phase: Ready	camel.apache.org/kamelet.bundled=true camel.apache.org/kamelet.readonly=true camel.apache.org/kamelet.type=action
avro-serialize-action	Kamelet	Phase: Ready	camel.apache.org/kamelet.bundled=true camel.apache.org/kamelet.readonly=true camel.apache.org/kamelet.type=action
aws-cloudwatch-sink	Kamelet	Phase: Ready	camel.apache.org/kamelet.bundled=true camel.apache.org/kamelet.readonly=true camel.apache.org/kamelet.type=sink

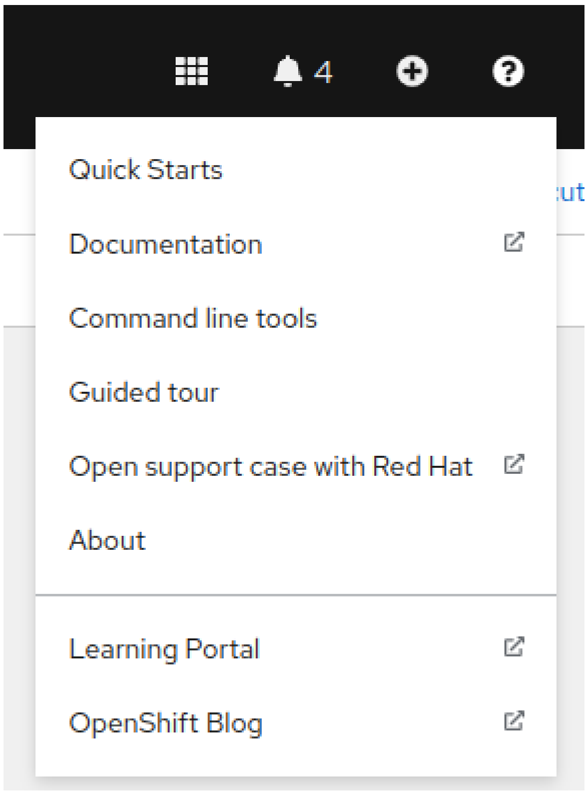
Figure 4. List of Kamelets

14.4.3. Development Environment - Visual Studio

14.4.3.1. Perquisites

Visual Studio use the OpenShift client's context to interact with remote OpenShift deployment.

To install required command line (cli) tools, click on the ? icon next to your user name on the OpenShift Console.



There are multiple Visual Studio Code extensions available for Camel K, we are going to use the one available from Red Hat, at marketplace.visualstudio.com/items?itemName=redhat.vscode-camelk

14.4.3.2. Hello World Camel K Style

Open Visual Studio Code in a newly created folder.

To start Visual Studio Code, in a folder, you start from the cmd line using **code .**.

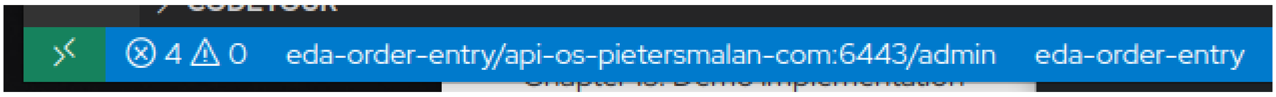
Building our Hello World Camel K:

- In Visual Studio Code, press **Ctrl**+**Shift**+**A**, or open **View** → **Command Palette** menu item
- Search and select **Create a new Apache Camel K Integration file**
- Select **Java**
- Select Workspace
- Name it **HelloWorld**

The generated code will show a sample Java Camel K route.

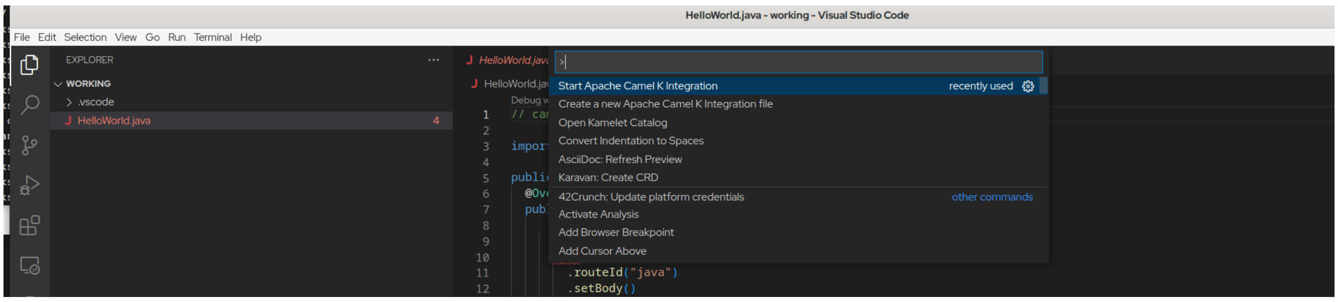
Before we deploy the code to our eda-order-entry project, we have to select the OpenShift context, and the project:

- Select at the bottom of editor, the OpenShift context, and the namespace,**eda-order-entry**



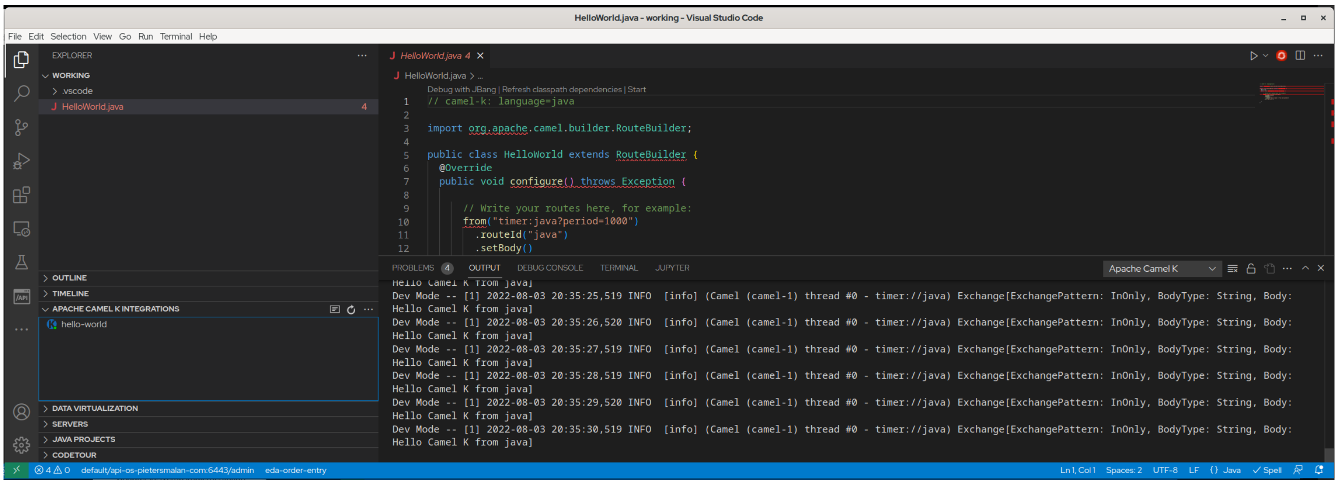
If the context is not populated, then you have to login into OpenShift, either from a different terminal, or by pressing **CTRL+SHIFT+**, then do a **oc login ….**

For deployment, press **Ctrl+Shift+A**, Select **Start Apache Camel K Integration**



HelloWorld.java - working - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
WORKING
.vscode
HelloWorld.java
Start Apache Camel K Integration
recently used
Create a new Apache Camel K Integration file
Open Kamelet Catalog
Convert Indentation to Spaces
AsciiDoc: Refresh Preview
Karavan: Create CRD
42Crunch: Update platform credentials
other commands
Activate Analysis
Add Browser Breakpoint
Add Cursor Above

After initial deployment, you can check the status, and output of the log files within Visual Studio Code.



```java
// camel-k: language=java

import org.apache.camel.builder.RouteBuilder;

public class HelloWorld extends RouteBuilder {
  @Override
  public void configure() throws Exception {

      // Write your routes here, for example:
      from("timer:java?period=1000")
        .routeId("java")
        .setBody()
```

```
Hello Camel K from java]
Dev Mode -- [1] 2022-08-03 20:35:25,519 INFO  [info] (Camel (camel-1) thread #0 - timer://java) Exchange[ExchangePattern: InOnly, BodyType: String, Body:
Hello Camel K from java]
Dev Mode -- [1] 2022-08-03 20:35:26,520 INFO  [info] (Camel (camel-1) thread #0 - timer://java) Exchange[ExchangePattern: InOnly, BodyType: String, Body:
Hello Camel K from java]
Dev Mode -- [1] 2022-08-03 20:35:27,519 INFO  [info] (Camel (camel-1) thread #0 - timer://java) Exchange[ExchangePattern: InOnly, BodyType: String, Body:
Hello Camel K from java]
Dev Mode -- [1] 2022-08-03 20:35:28,519 INFO  [info] (Camel (camel-1) thread #0 - timer://java) Exchange[ExchangePattern: InOnly, BodyType: String, Body:
Hello Camel K from java]
Dev Mode -- [1] 2022-08-03 20:35:29,520 INFO  [info] (Camel (camel-1) thread #0 - timer://java) Exchange[ExchangePattern: InOnly, BodyType: String, Body:
Hello Camel K from java]
Dev Mode -- [1] 2022-08-03 20:35:30,519 INFO  [info] (Camel (camel-1) thread #0 - timer://java) Exchange[ExchangePattern: InOnly, BodyType: String, Body:
Hello Camel K from java]
```

If you expand the **APACHE CAMEL K INTEGRATIONS**, you will a list of deployed integrations, in our case **hello-world** with a green dot indicating it is running. If the dot is red, an error occurred.

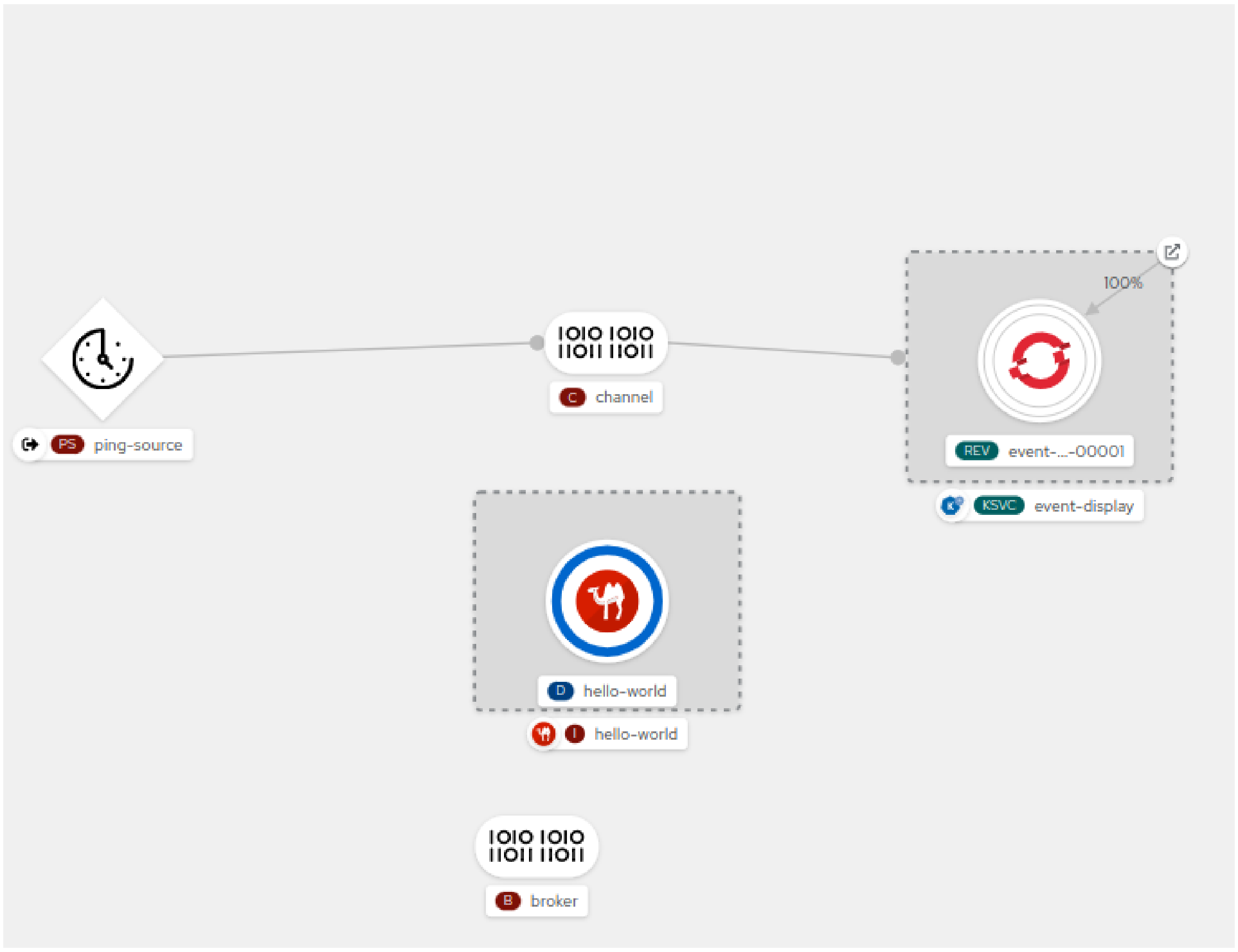
In the OUTPUT windows, you will see the log output of our deployed integration.

14.4.3.3. Looking under the covers

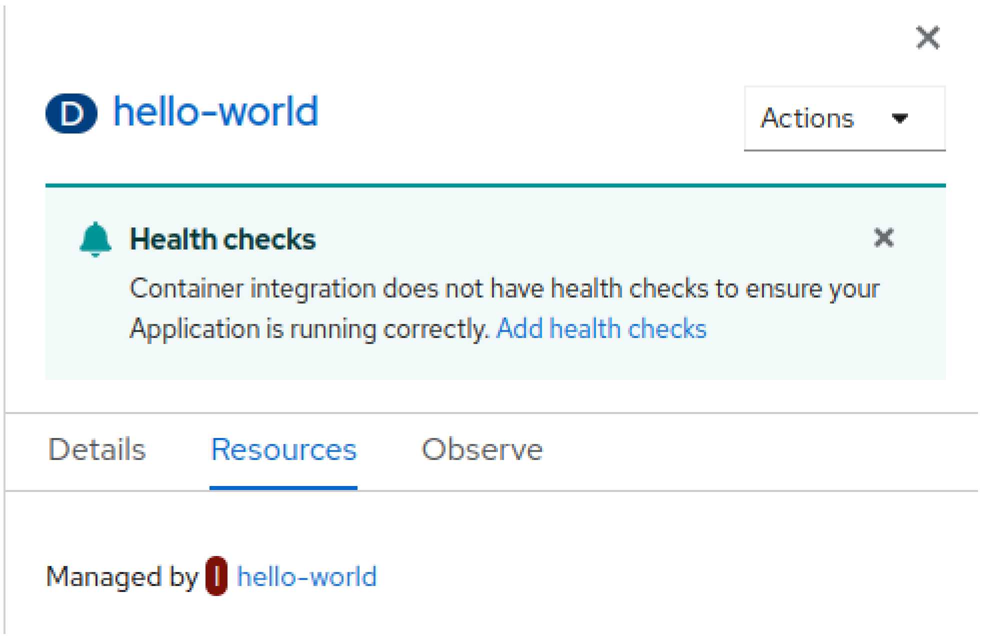
Open the OpenShift Console to see what happened to within our project.

- From the **Administrator** perspective, go to **Projects**, under **Home**
- On the project list, select the **eda-order-entry**
- Switch perspective to **Developer**

Notice the new **hello-world** with the Red Camel icon deployment.



- Click on **hello-world**
- On the Information panel, click on the **Resources** tab
- You will see the **hello-world-..** pod; FYI by clicking on **View Logs**. This log output of the pod, is what you see in the Visual Studio Code OUTPUT


D hello-world

Actions

Health checks

Container integration does not have health checks to ensure your Application is running correctly. Add health checks

Details Resources Observe

Managed by I hello-world

- Click on the **Managed by hello-world** link

- Select the **YAML** tab, and hopefully you will see something familiar. Our source code !

Integration Custom Resource

When you do a deployment from Visual Studio Code, we have created an Integration Custom Resource. The creation of the resource, triggered a full build of an Camel K S2I build, which resulted in an image deployed to our environment.

Camel K has the smarts to utilize different technologies, or traits, depending on the functionality of the code. A good example might be if we deployed a REST service, our resulting deployment might result in a KNative serverless deployment.

These traits can be configured within the code, or as we did for the logging on an Integration Platform level.

For promoting our integration from Dev to Test, we can use the Integration Resource to recreate the deployment, with the advantage, that we might have configured different settings for our Integration Platform. Off course there might be additional resources involved, like properties, secrets which you have to deploy beforehand if in use by your code.

14.5. Service Registry

14.5.1. Installing Service Registry

The service registry requires persistence storage, in the form of an RDBMS or KafkaSQL.

Since we have our kafka instance, we are going to use that.

14.5.1.1. Getting Kafka Bootstrap Server Endpoint

- From **Administrator** perspective, click on **Projects** under **Home**
- Select **amq-streaams** from the projects
- On the **Inventory** panel, click on **Services**
- Click on **my-cluster-kafka-bootstrap** service
- On the right hand-side, under **Service Routing**, take note of the **Hostname**
- Also note the ports, under **Service port mapping**, the one we are interested in is the tcp-clients 9092.

Bootstrap Server Endpoint for my-cluster-kafka-bootstrap

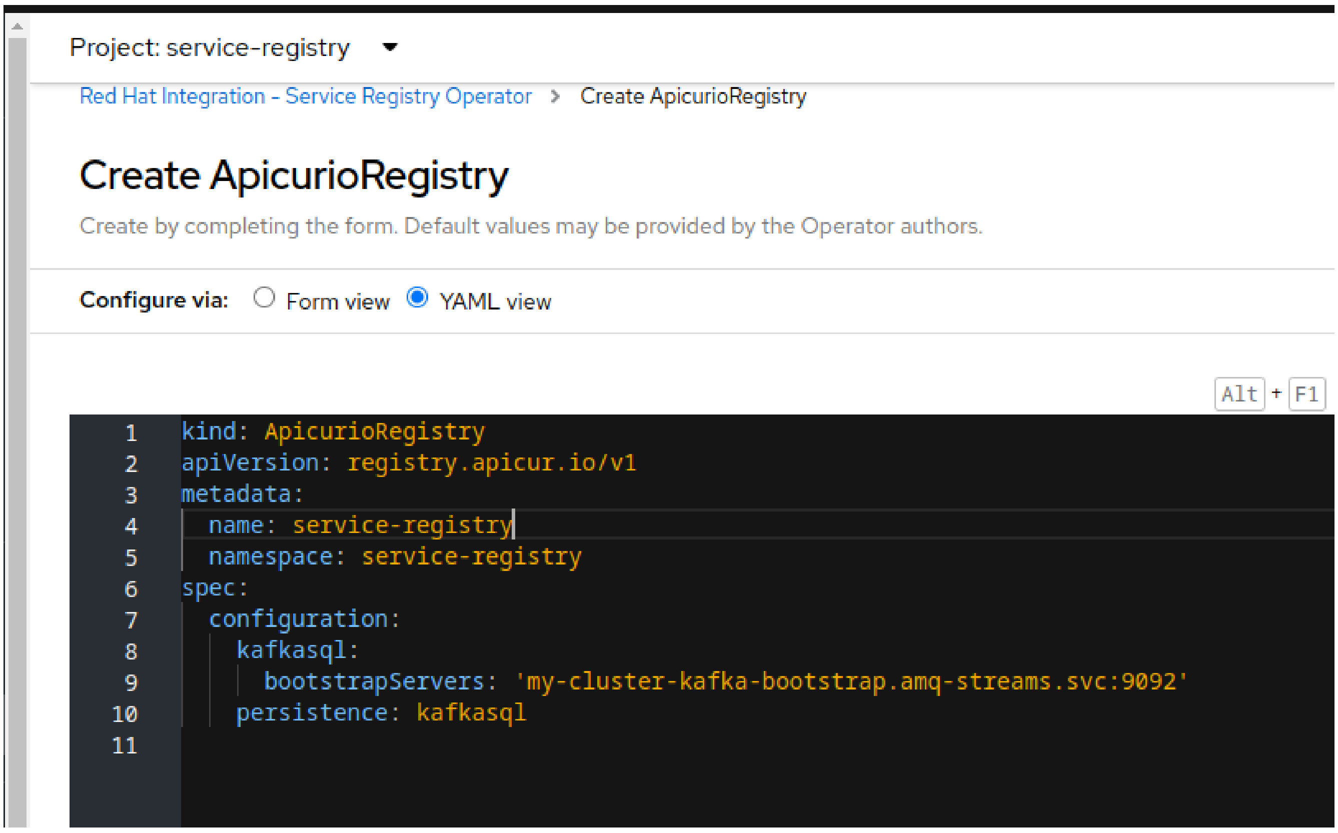
```
my-cluster-kafka-bootstrap.amq-streams.svc:9092
```

The service hostname is determined by service and project (namespace), appended with svc.cluster.local.

The 'cluser.local' is added automatically,as it is the default local domain name, so typically it is dropped from service endpoints, if used internal to OpenShift.

Next up, we have to create a namespace for Service Registry.

- From the **Administrator** perspective, go to **Projects** from the **Home** section of the navigation pane.
- On the list of projects, click on **Create Project**
- Call the namespace **service-registry** and give a description, and click on **Create** namespace
- Go to the **Installed Operators**, under **Operators** heading
- Select **Red Hat Integration - Service Registry Operator** from the operator list
- On the **Apicurio Registry** pane, click **Create Instance**
- Select the **YAML view** option in the **Configure via:** choice
- Change the **name:** to service-registry
- Replace <service name> with **my-cluster-kafka-bootstrap**
- Replace <namespace> with **amq-streams**
- Clicnk on **Create**



Project: service-registry

Red Hat Integration - Service Registry Operator > Create ApicurioRegistry

Create ApicurioRegistry

Create by completing the form. Default values may be provided by the Operator authors.

Configure via: Form view [✓ YAML view]

```yaml
kind: ApicurioRegistry
apiVersion: registry.apicur.io/v1
metadata:
  name: service-registry
  namespace: service-registry
spec:
  configuration:
    kafkasql:
      bootstrapServers: 'my-cluster-kafka-bootstrap.amq-streams.svc:9092'
    persistence: kafkasql
```

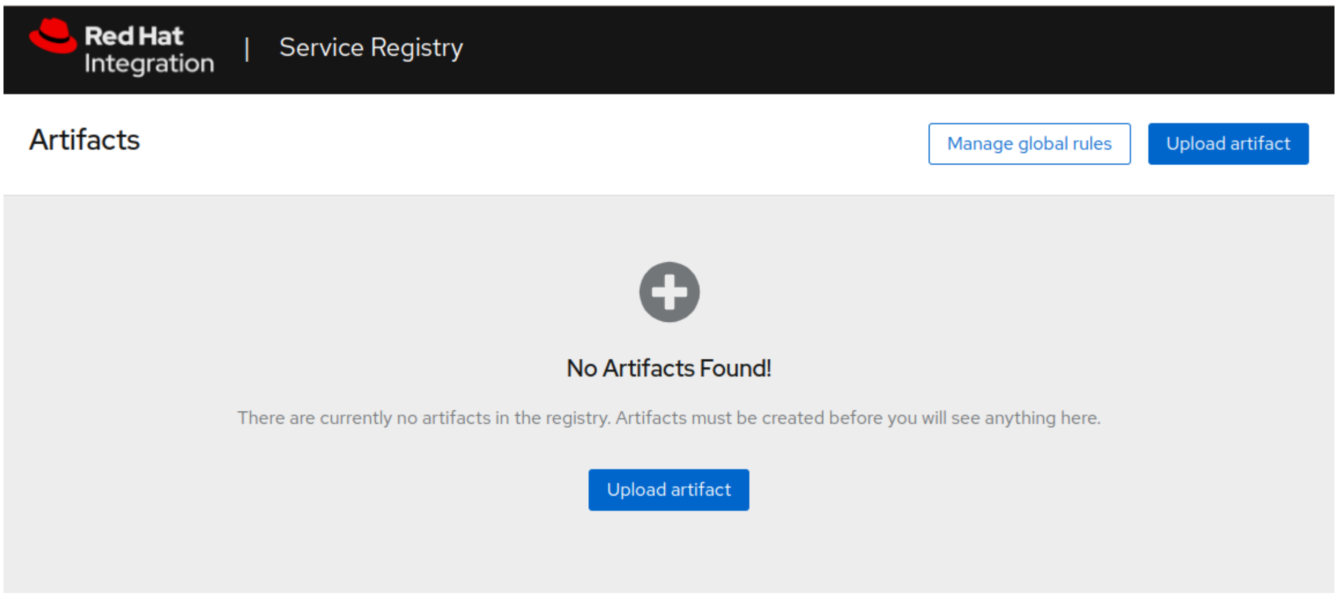
Figure 5. Service Registry YAML

14.5.2. Testing the Registry

By default, there would be a route created to expose the registry to the outside world.

In order to get the external facing registry UI:

- From **Administrator** perspective, go to **Routes**, under **Networking** heading
- Click on the link displayed, next to the service-registry-ingress.



Red Hat Integration | Service Registry

Artifacts

No Artifacts Found!

There are currently no artifacts in the registry. Artifacts must be created before you will see anything here.

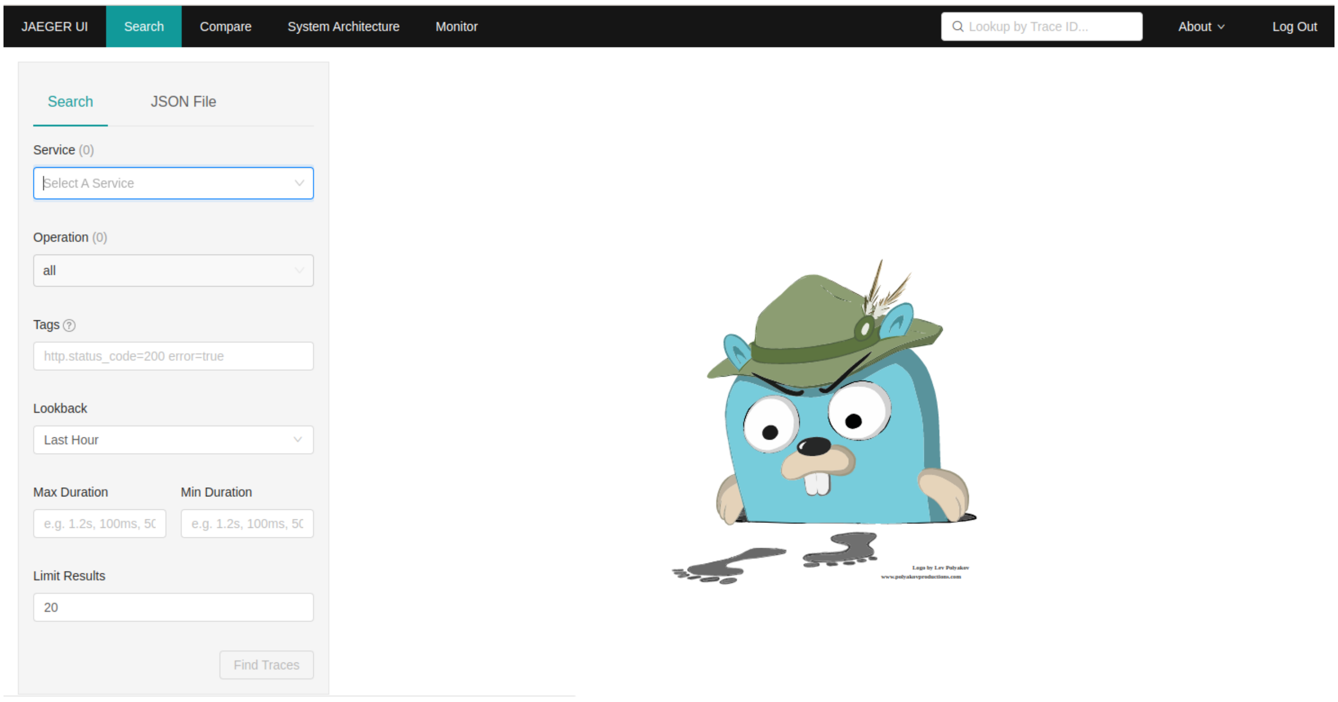
Figure 6. Service Registry UI

It is always possible to remove the service, or add additional security on top of the Registry's web console.

14.6. Distributed Tracing Platform

To see what is happening within our components, we can utilize the distributed tracing platform.

- From the **Administrator** perspective, got to the Projects from the Home section of the navigation pane.
- Click on **eda-order-entry**, under the project list
- Go to **Installed Operators**, under the **Operators** section
- Click on the **Red Hat OpenShift distributed tracing platform** Operator
- On the **Jaeger** panel, click **Create Instance**
- Leave the defaults, but confirm the **Strategy** is set to **allInOne**
- Click on **Create**
- Click on **Routes**, under the section **Networking**
- Click on the link displayed under **Location** to access Jaeger
- Login to gain access to Jaeger



Service (0)
Select A Service
Operation (0)
all
Tags
http.status_code=200 error=true
Lookback
Last Hour
Max Duration
e.g. 1.2s, 100ms, 5C
Min Duration
e.g. 1.2s, 100ms, 5C
Limit Results
20

Figure 7. Jaeger UI

Chapter 15. Demo Implementation

15.1. Demo ERP - ODOO

Our third party application is going to be ODOO.

The reason for choosing Odoo is the fact that it is free to use, and does not, for our purposes, expose any API's, hence we have to use the database to integrate.

The database schema is straight forward and easy to understand, which gives us another reason to Odoo.

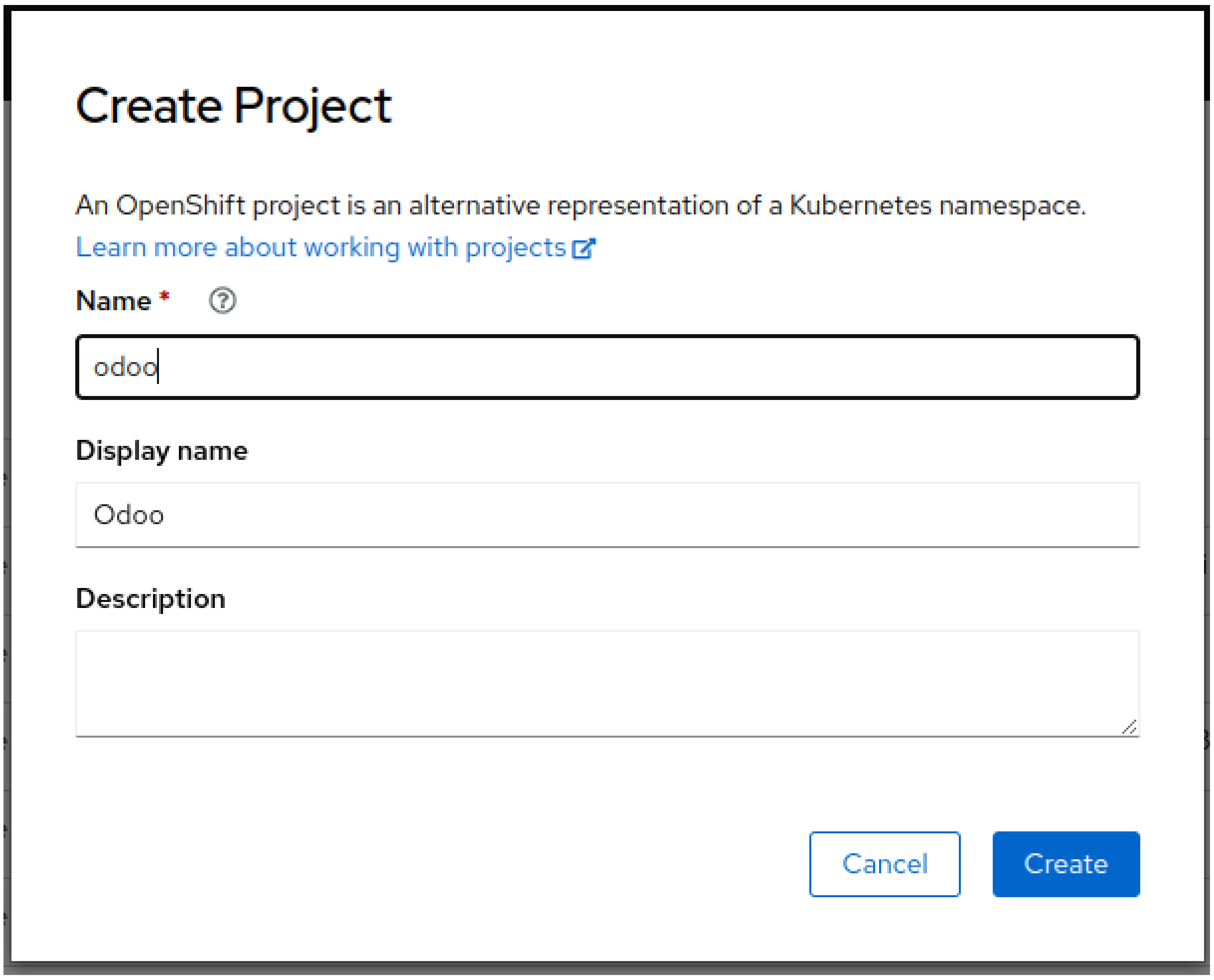
> Odoo is a suite of open source business apps that cover all your company needs: CRM, eCommerce, accounting, inventory, point of sale, project management, etc.
>
> Odoo's unique value proposition is to be at the same time very easy to use and fully integrated.
>
> — ODOO, https://odoo.com

Keep in mind, however, that even if ODOO is open source, that it can be replaced with Oracle eBusiness Suite, SAP, Workday and a multitude of other ERP, CRM, Supply Chain and other off-the-shelf applications.

15.1.1. Installing Odoo

First we have to create a namespace for Odoo.

- From the **Administrator** perspective, go to **Projects** from the **Home** section of the navigation pane.
- On the list of projects, click on **Create Project**
- Call the namespace odoo and give a description, and click on Create namespace


Create Project

An OpenShift project is an alternative representation of a Kubernetes namespace.
Learn more about working with projects

Name * : odoo

Display name: Odoo

Description:

For the installation we are going to use a predefined template to build the application.

The template was translated and modified from the original template which developed by Franklin Gomez.

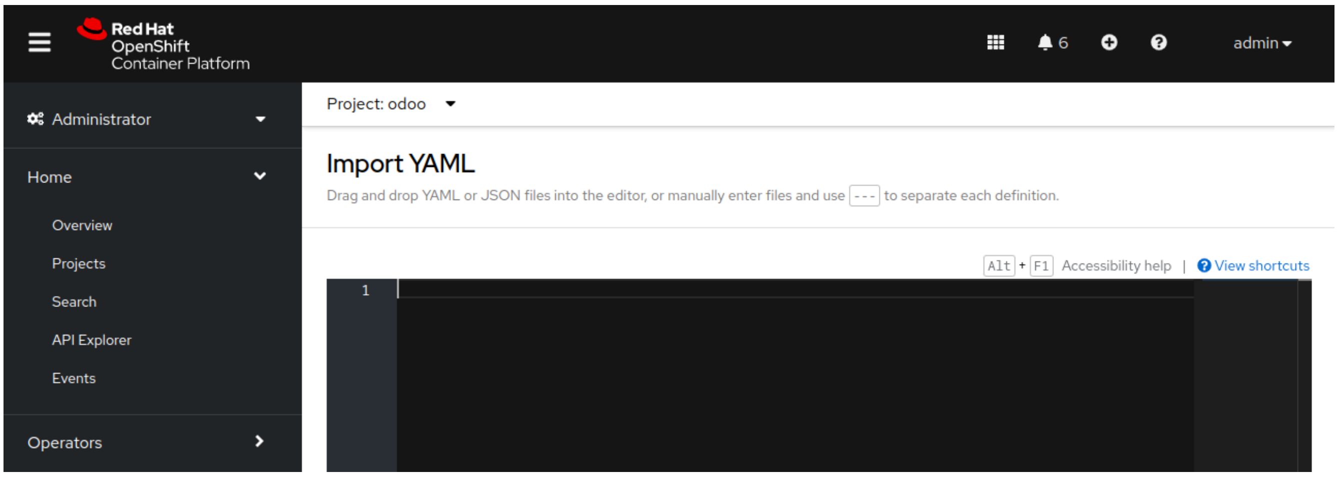
gitlab.paas.agesic.red.uy/franklin.gomez/odoo-ocp - Div. TE-Tecnologías, Agesic

There is also an Odoo Operator, but it requires a lot of pre-configuration.

15.1.2. Creating Template and Instantiating Template

To install the template:

- From the **Administrator** perspective, got to **Projects** from the **Home** section of the navigation pane.
- In the **Projects** panel, click on the project **odoo**
- On the top, next to your user name, and the **?**, there is a + sign. Click on the + sign:


Red Hat OpenShift Container Platform
6
admin
Administrator
Home
Overview
Projects
Search
API Explorer
Events
Operators
Project: odoo
Import YAML
Drag and drop YAML or JSON files into the editor, or manually enter files and use --- to separate each definition.
Alt + F1 Accessibility help | View shortcuts

- Paste the contents of the Odoo Temlpate in the Import YAML, and click create.

Odoo Template

raw.githubusercontent.com/pmalan-rh/EDAOpenShift-code/main/odoo/odoo-template.yaml

- Switch perspective to **Developer**
- Click on +**Add**
- Under **Developer Catalog**, click on **All Services**
- In the search box, type in **odoo**
- Select **Odoo**
- In the popup window, click on **Instantiate Template**
- In the template, specify **odoo** for **Project Name**, leaving the others as defaults
- Click **Create**

After a few minutes, you will see the deployments in ready state as indicated by the blue rings around it.

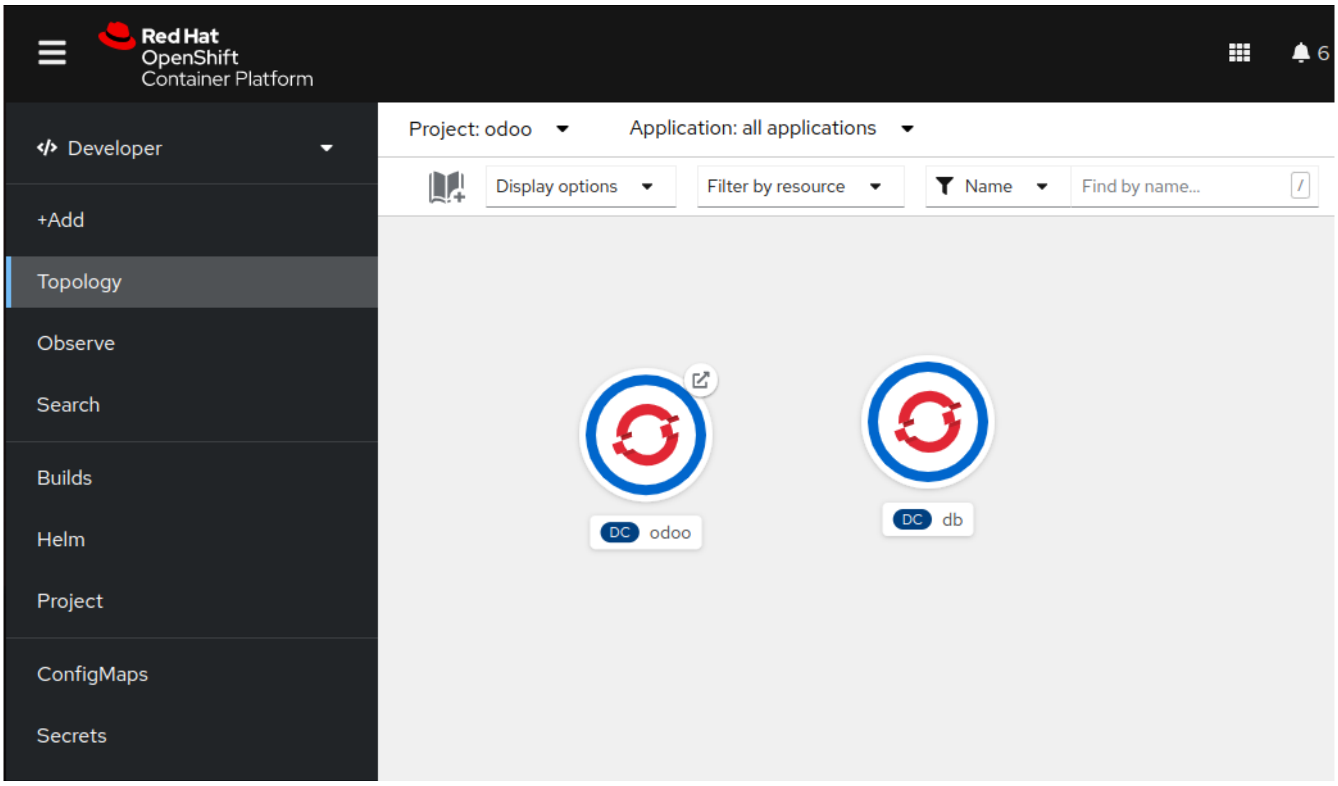
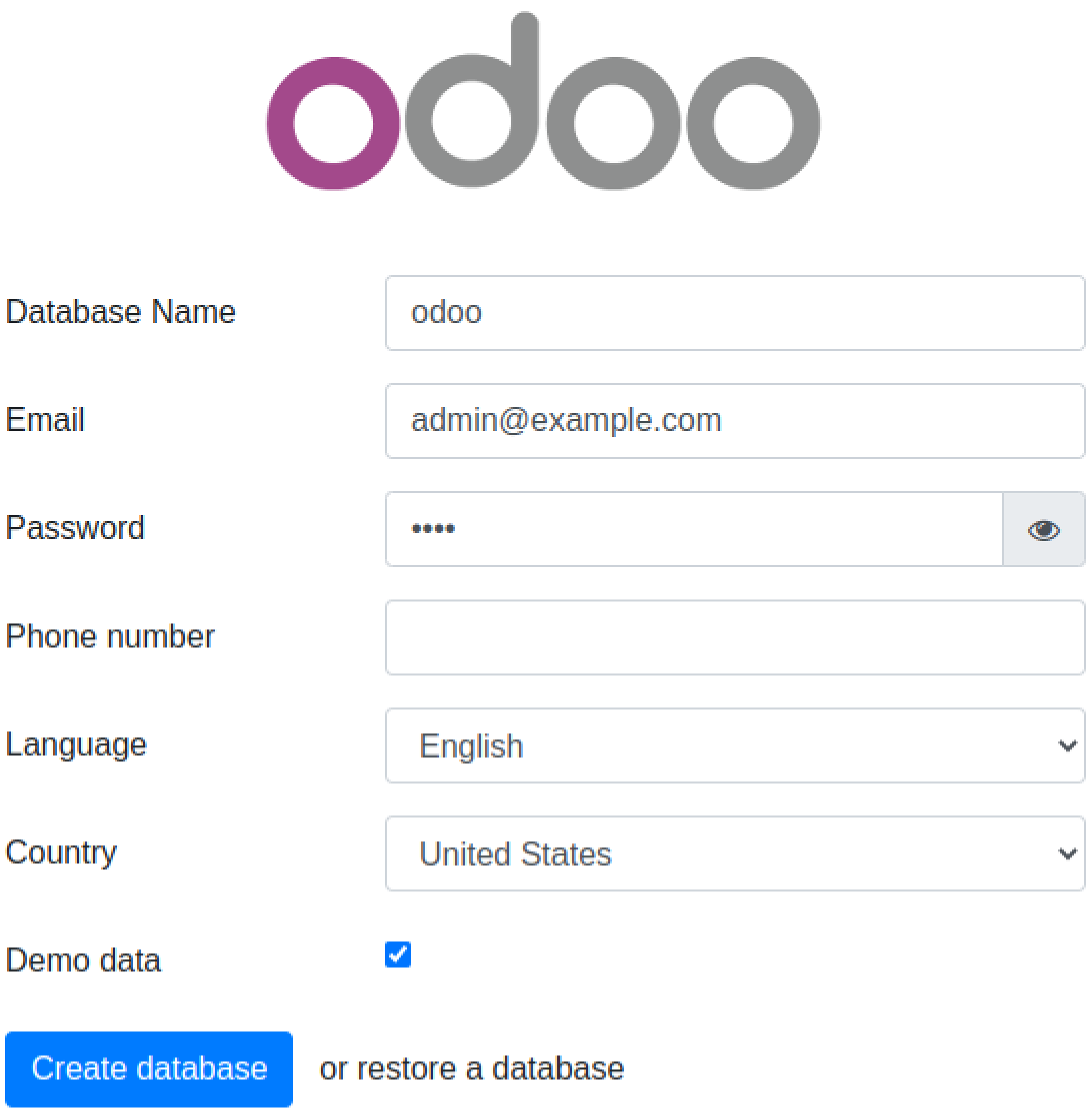


Figure 8. Odoo Deployment Completed

15.1.3. Configuring odoo application

Next step is to configuring the application, and enable the sales module.

- Switch perspective to **Developer**, making sure that the selected Project is still odoo.
- If the Topology is not shown, click on **Topology**
- Click on ↗ icon to open the application in browser.
- You will be greeted with a configuration wizard as follow:



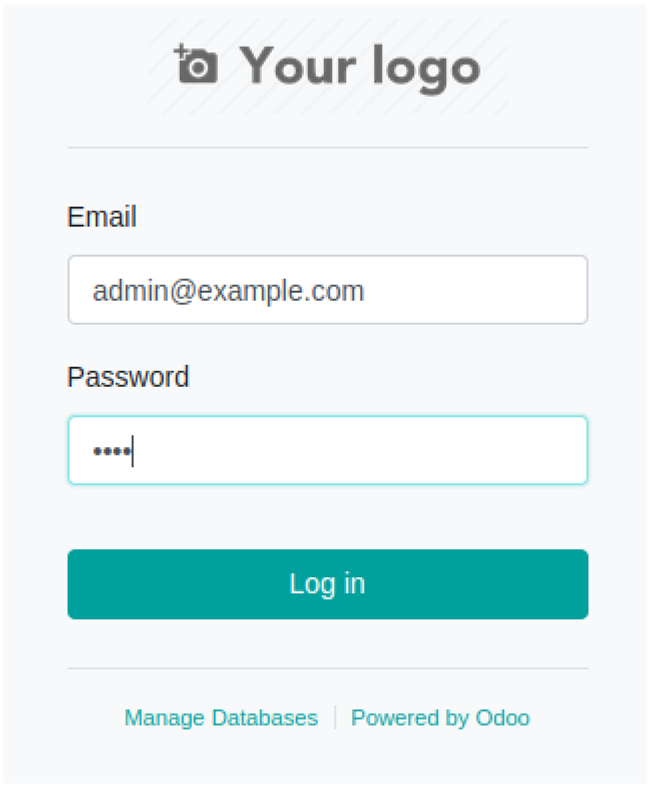
Database Name	odoo
Email	admin@example.com
Password	••••
Phone number	
Language	English
Country	United States
Demo data	✓

- Fill in the values as supplied in above screenshot, make sure that the **Email** is admin@example.com, **Password** is odoo, and that checkbox Demo data is selected
- Click on **Create Database**
- It will take a bit to build the database, so be patient after clicking the Create Database button. The only indication that something is happing is the loading indicator in the browser.

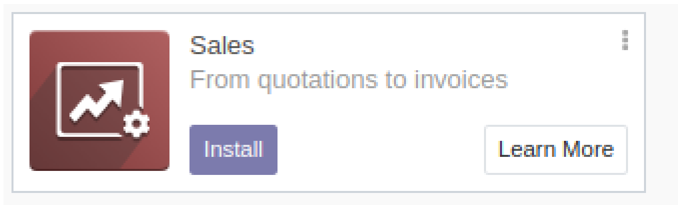
15.1.4. Customizing Odoo

Last step is to install required functionality within Odoo, in our case, we are going to use the Sales module.

- Switch perspective to **Developer**, making sure that the selected Project is still odoo.
- If the Topology is not shown, click on **Topology**
- Click on icon to open the application in browser.
- This time round, you will be greeted with a login page:

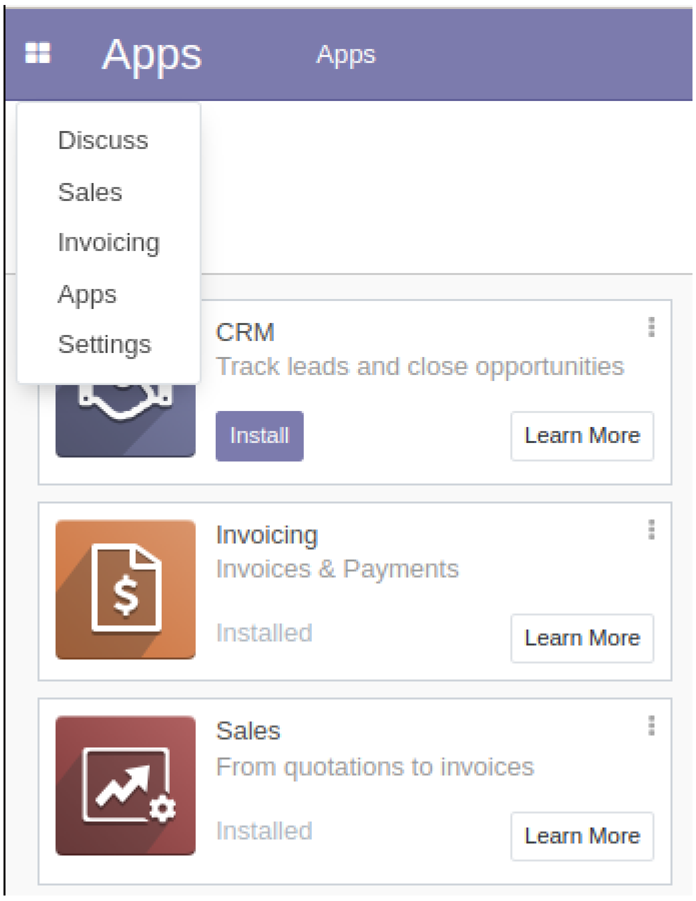


- Login with admin@example.com and the password odoo.
- Search for Sales, and click on **Install**



The installation of the Sales module will take a while, if you get an error message in the browser, don't worry, the installation is still happening in the background, just refresh the browser page.

You would be able to use the Sales module after installation, using the four square icon next to the Apps menu.


Apps
Apps
Discuss
Sales
Invoicing
Apps
Settings
CRM
Track leads and close opportunities
Install
Learn More
Invoicing
Invoices & Payments
Installed
Learn More
Sales
From quotations to invoices
Installed
Learn More

15.2. Demo Debezium - Change Data Capture

For our demo, we are going to listen for changes on a sales order. The specific change we are interested in, is when an order is approved.

Since Odoo does not emit any events, we have to capture the change on the database level.

We are going to setup Debezium to capture the changes on the table sale_order.

Debezium configuration require two distinct areas.

First we have to prepare the database to publish, or capture logs, depending on the RDBMS in question. For our instance, we are going to use the PogresSQL under Odoo.

Secondly, we have to configure the Kafka Connector to enable the events to be published to a topic in Kafka.

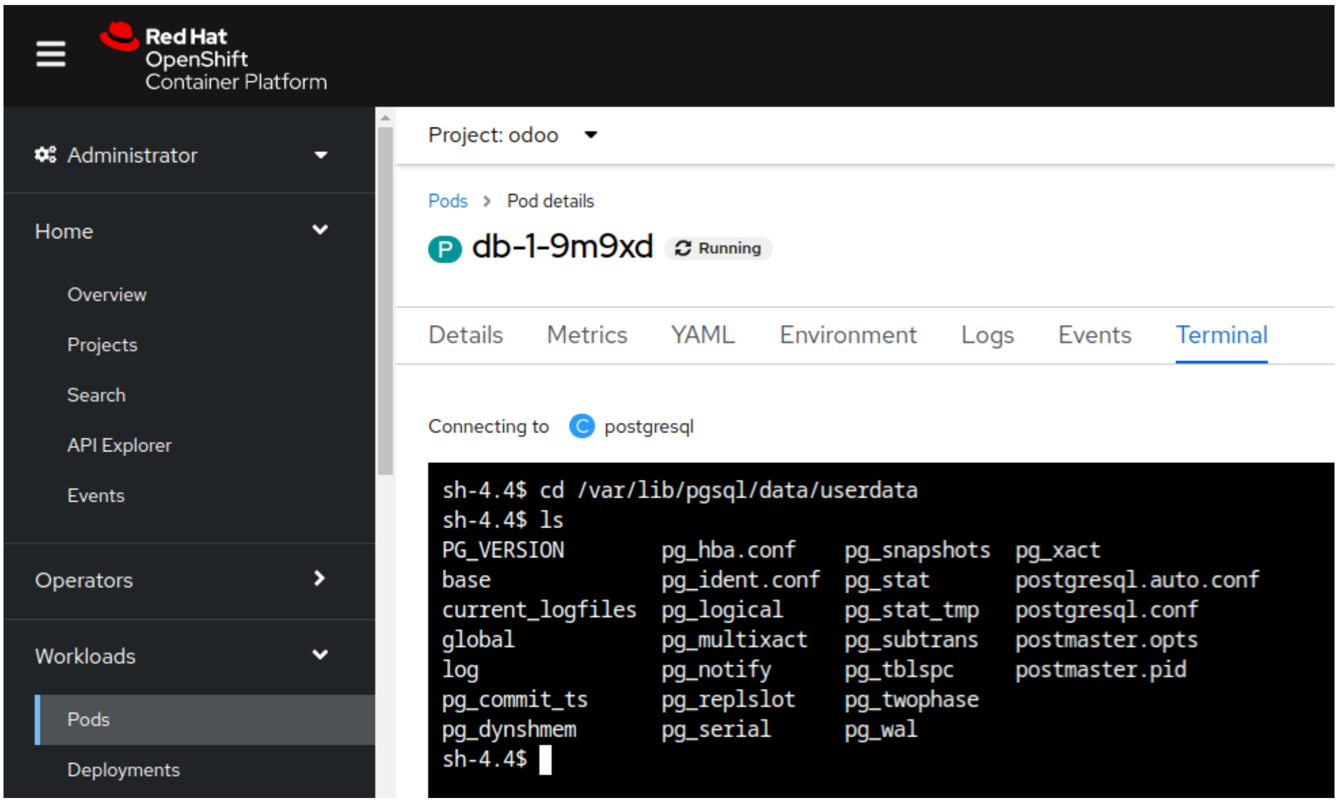
15.2.1. Postgres Configuration

We have to configure the database engine configuration, and also apply some changes on a schema level, through sql interaction.

For the database configuration:

- From **Admin** perspective, select **Projects**, from the **Home**
- Click on the **odoo** project
- In the navigation panel, select **Pods**, under **Workloads**
- On the list of pods, click on the pod named **db-..**
- With pod information open, click on the **Terminal** tab, to open a terminal session

Now that we have access to the database host pod, we can start doing the configuration.



```
sh-4.4$ cd /var/lib/pgsql/data/userdata
sh-4.4$ ls
PG_VERSION        pg_hba.conf    pg_snapshots  pg_xact
base              pg_ident.conf  pg_stat       postgresql.auto.conf
current_logfiles  pg_logical     pg_stat_tmp   postgresql.conf
global            pg_multixact   pg_subtrans   postmaster.opts
log               pg_notify      pg_tblspc     postmaster.pid
pg_commit_ts      pg_replslot    pg_twophase
pg_dynshmem       pg_serial      pg_wal
sh-4.4$ 
```

Figure 9. Postgresql Terminal

On the terminal - Click on **Expand** to open a full window - cd /var/lib/pgsql/data/userdata - vi postgress.conf

Apply the changes as follow at the the bottom of the file, **#WAL** / **Replication** paragraph should be added:

Changes to postgresql.conf

```
#------------------------------------------------------------------------------
# CUSTOMIZED OPTIONS
#------------------------------------------------------------------------------

# Add settings for extensions here

# WAL / replication
wal_level = logical
max_wal_senders = 3

# Custom OpenShift configuration:
include '/var/lib/pgsql/openshift-custom-postgresql.conf'
```

- **Collapse** to get back to pod information panel
- Restart pod, using **Actions**, **Delete Pod**, to force the Postgresql to restart with new configuration.

After the pod has restarted, we have to enable the odoo database with replication.

- Again, following steps above, to get the terminal of the **db-..** pod.

- In the terminal, execute the following:

```
psql -U postgres
psql (10.21)
Type "help" for help.

postgres=# create role debezium replication login;
CREATE ROLE
postgres=# grant debezium to odoo;
GRANT ROLE
postgres=# ALTER USER odoo WITH SUPERUSER;
ALTER ROLE
postgres=#/q
```

15.2.2. Configuring Kafka Connect

Kafka Connect is the service to which a Kafka Connector is deployed to. The connector is build by OpenShift’s S2I with all the relevant libraries required, as specified in the KafkaConnect resource.

In order to use Debezium, we have to include the relevant libraries into the Kafka Connect. We are also going to deploy the service registry libraries.

To create a Kafka Connect configuration:

We have to create a target ImageStream for the output of the build of our connector, before we start the build.

- Click on the + at the top next to your username.
- Paste the following YAML, to create ImageStream

YAML for ImageStream

raw.githubusercontent.com/pmalan-rh/EDAOpenShift-code/main/debezium/debezium-streams-connect.yaml

```
apiVersion: image.openshift.io/v1
kind: ImageStream
metadata:
  name: debezium-streams-connect
  namespace: amq-streams
```

- From the **Administrator** perspective, navigate to **Projects**, under **Home**
- Select the **amq-streams** from the project list
- Click on **Installed Operators**, and click on **Red Hat Integration - AQM Streams**
- Click on the **Create Instance** on the **Kafka Connect** panel
- Click on the **YAML view** on the **Configure via** option

- Change the YAML to reflect the following:

YAML for Kafka Connect

raw.githubusercontent.com/pmalan-rh/EDAOpenShift-code/main/debezium/my-connect-cluster.yaml

Kafka Connect YAML

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
  namespace: amq-streams
  annotations:
    strimzi.io/use-connector-resources: "true"
spec:
  config:
    group.id: connect-cluster
    offset.storage.topic: connect-cluster-offsets
    config.storage.topic: connect-cluster-configs
    status.storage.topic: connect-cluster-status
    config.storage.replication.factor: -1
    offset.storage.replication.factor: -1
    status.storage.replication.factor: -1
    key.converter: io.apicurio.registry.utils.converter.AvroConverter
    key.converter.schema.registry.url: http://service-registry-service.service-
registry.svc.cluster.local:8080/api
    key.converter.apicurio.registry.global-id:
io.apicurio.registry.utils.serde.strategy.AutoRegisterIdStrategy
    key.converter.apicurio.registry.as-confluent: true
    key.converter.apicurio.registry.auto-register: true
    value.converter: io.apicurio.registry.utils.converter.AvroConverter
    key.converter.schema.registry.url: http://service-registry-service.service-
registry.svc.cluster.local:8080/api
    value.converter.apicurio.registry.global-id:
io.apicurio.registry.utils.serde.strategy.AutoRegisterIdStrategy
  tls:
    trustedCertificates:
      - secretName: my-cluster-cluster-ca-cert
        certificate: ca.crt
  version: 3.1.0
  build:
    output:
      type: imagestream
      image: debezium-streams-connect:lastest
    plugins:
      - name: debezium-connector-postgresql
        artifacts:
          - type: zip
            url: https://maven.repository.redhat.com/ga/io/debezium/debezium-
connector-postgres/1.9.5.Final-redhat-00001/debezium-connector-postgres-1.9.5.Final-
```

```
redhat-00001-plugin.zip
          - type: zip
            url: https://maven.repository.redhat.com/ga/io/apicurio/apicurio-registry-
distro-connect-converter/1.2.2.Final-redhat-00005/apicurio-registry-distro-connect-
converter-1.2.2.Final-redhat-00005-converter.zip
          - type: jar
            url: https://maven.repository.redhat.com/ga/io/apicurio/apicurio-registry-
serdes-jsonschema-serde/2.0.0.Final-redhat-00005/apicurio-registry-serdes-jsonschema-
serde-2.0.0.Final-redhat-00005.jar

  replicas: 1
  bootstrapServers: 'my-cluster-kafka-bootstrap:9093'
```

Latest Releases for Debezium

The get the latest Debezium plugins, you can browse to the following URL, and search for specific plugins.

maven.repository.redhat.com/ga/io/debezium/

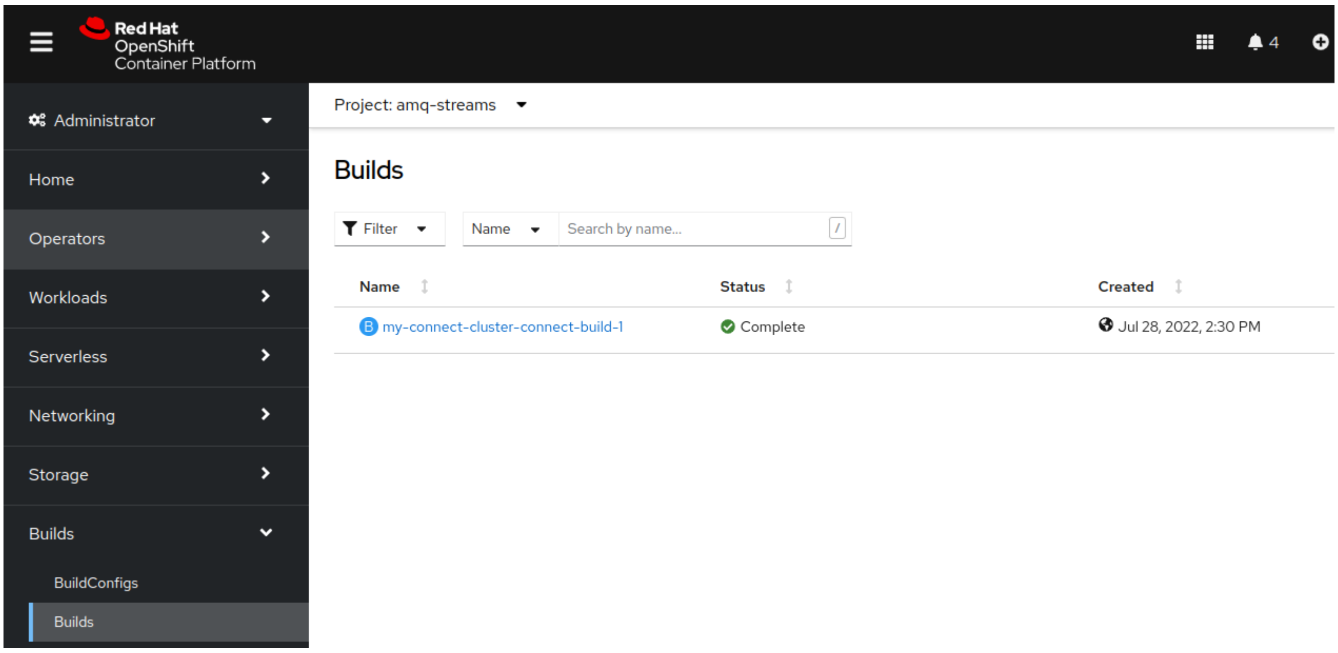
For the APICurio Registry converter:

maven.repository.redhat.com/ga/io/apicurio/apicurio-registry-distro-connect-converter/

15.2.2.1. Verify Build

To verify that the build was successful, from **Administrator** perspective, go to **Builds** under heading **Builds**.

You should see a **complete** build if configuration applied correctly.



Name	Status	Created
B my-connect-cluster-connect-build-1	Complete	Jul 28, 2022, 2:30 PM

Figure 10. Completed Build

15.2.3. Configuring Kafka Connector

The Kafka Connector is responsible to define the database connection and relevant schema elements we are interested in capturing change events from. This configuration is fed into the Kafka Connect to start capturing of events.

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnector
metadata:
  labels:
    strimzi.io/cluster: my-connect-cluster
  name: sales-connector-postgresql
spec:
  class: io.debezium.connector.postgresql.PostgresConnector
  tasksMax: 1
  config:
    database.history.kafka.bootstrap.servers: 'my-cluster-kafka-bootstrap.amq-
streams.svc:9092'
    database.history.kafka.topic: schema-changes.sales
    database.hostname: db.odoo.svc.cluster.local
    database.port: 5432
    database.user: odoo
    database.password: odoo
    database.dbname: odoo
    database.server.name: sales_connector_postgresql
    database.include.list: public.sale_order
    plugin.name: pgoutput
```

15.2.4. Testing Debezium

15.2.5. Peek into Kafka

UI for Apache Kafka is a third=party tool to see events in Kafka.

Steps for eploying of UI for Apache Kafka:

- From **Administrator** perspective, click **Projects** under **Home** section
- Click **amq_streams** on the list of projects
- Change the perspective to **Developer**
- Right click on **Topology Map**, **Add to Project**, and click on **Container Image**
- Specify **provectuslabs/kafka-ui:latest** for **Image name from external registry**
- Leave the rest defaults, and click on **Deployment** at the bottom in the **Click on the names to access ..** sentence
- Under *Environment Variables, add the following values:

Name: **KAFKA_CLUSTERS_0_NAME** value **amq-streams**

Name: **KAFKA_CLUSTERS_0_BOOTSTRAPSERVERS** value **my-cluster-kafka-bootstrap.amq-streams.svc:9092**

- Click on **Create**

Next we have to get the URL to application:

- Click on **Project**
- Click on **Route** on the **Inventory** panel
- Click on the **Location**, next to **kafka-ui**

UI for Apache Kafka, will display a list of options, select **Topics**, and **odoo.public.sale_order**

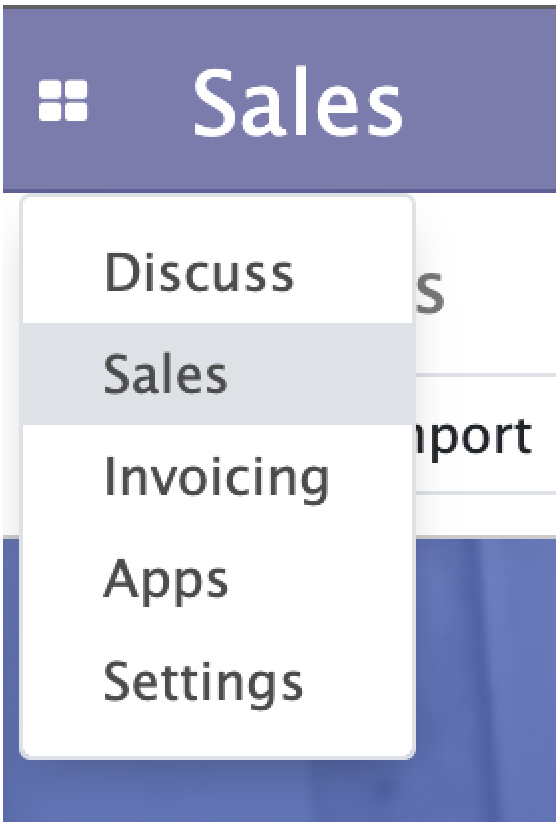
- select the **Messages** tab
- On the far right hand side, change the dropdown **Oldest First** to **Live Mode**

In the next section, we are going to create a Kafka message, by update a sales order.

15.2.5.1. Odoo Update

Login into Odoo, refer to Odoo configuration to get login link.

After login go to **Sales**

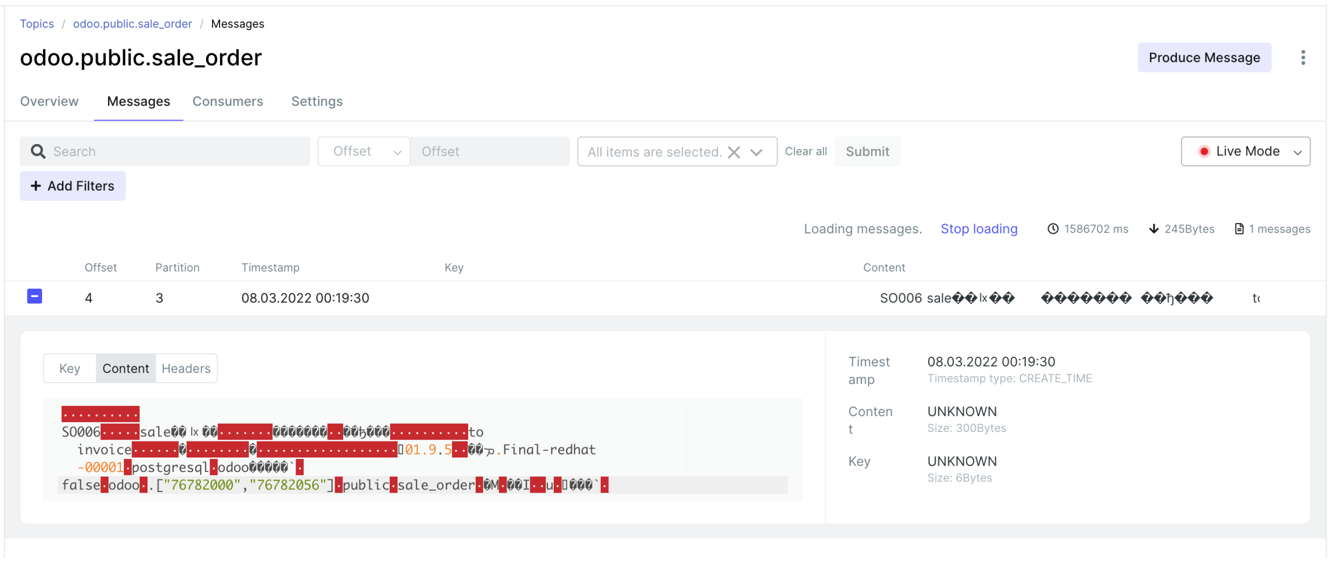


To update a quote:

- Click on the first **quote** in the table
- Click on **Edit**

- Update the **Payment Terms**, to a new value
- Click on **Save**

Switching back to UI for Apache Kafka, you will a message similar to the following, after clicking on the + sign next to new message:


Topics / odoo.public.sale_order / Messages

odoo.public.sale_order

Produce Message

Overview · Messages · Consumers · Settings

Search · Offset · Offset · All items are selected. · Clear all · Submit · Live Mode

+ Add Filters

Loading messages. Stop loading · 1586702 ms · 245Bytes · 1 messages

Offset	Partition	Timestamp	Key	Content
4	3	08.03.2022 00:19:30		SO006 sale��Ix�� ������� ��ђ��� t

Key · Content · Headers

```
..........
SO006.....sale�� lx ��.......�������..��ђ���..........to
  invoice......�........�..................01.9.5..��p.Final-redhat
  -00001.postgresql.odoo�����`.
false.odoo..["76782000","76782056"].public.sale_order.�M.��I..u.���`.
```

Timestamp: 08.03.2022 00:19:30 (Timestamp type: CREATE_TIME)

Content: UNKNOWN (Size: 300Bytes)

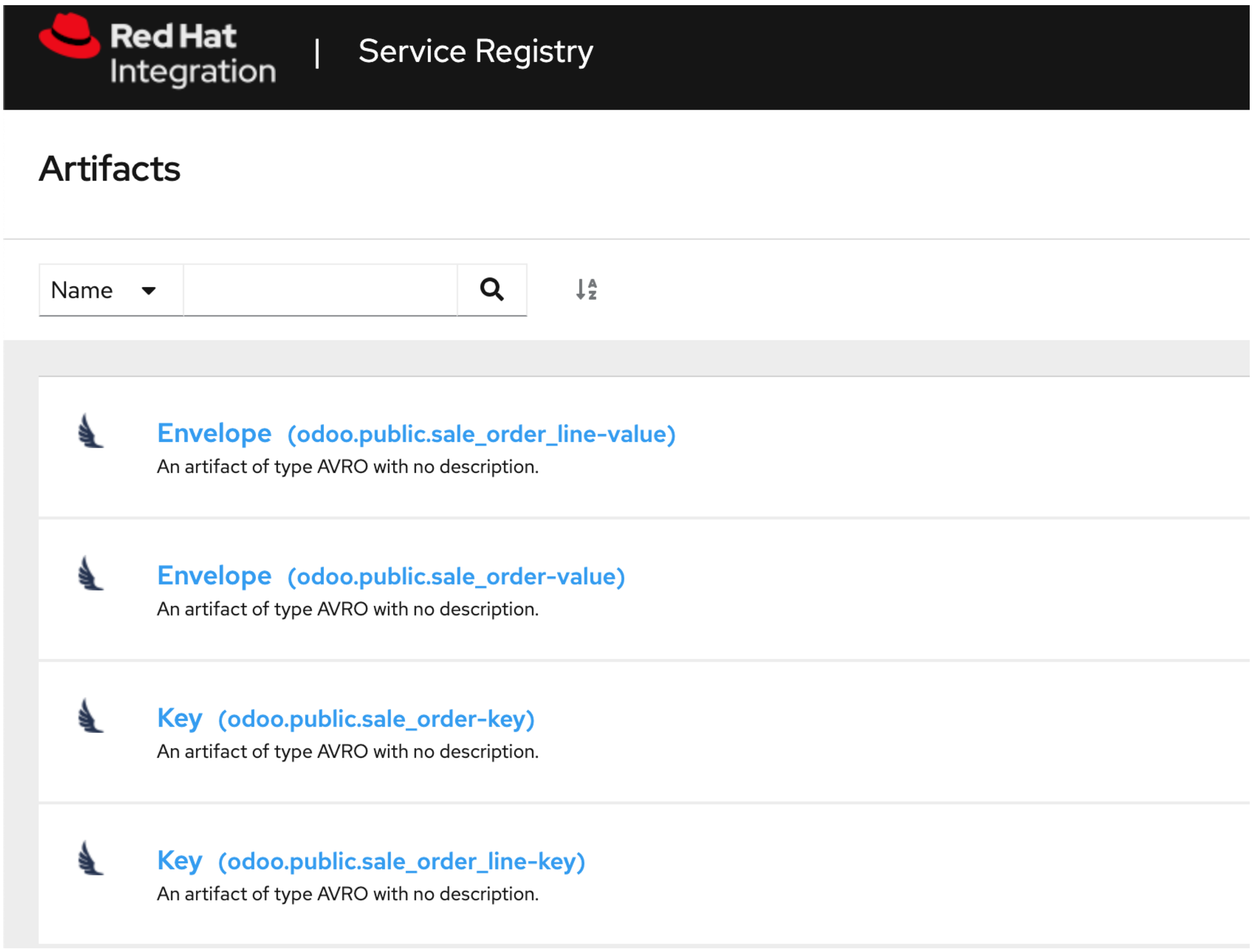
Key: UNKNOWN (Size: 6Bytes)

UI for Apache Kafka does not decode the message correctly, since it does not understand the associated AVRO schema, but at least we can confirm that Debezium is doing it's job as expected.

Last check is to see if the schema was registered in the service registry.

Go to the Service Registry home page:

- You would be able to see the auto registered AVRO schemas for our defined Debezium events, as illustrated in the follwong image.



Artifacts

Envelope (odoo.public.sale_order_line-value)
An artifact of type AVRO with no description.

Envelope (odoo.public.sale_order-value)
An artifact of type AVRO with no description.

Key (odoo.public.sale_order-key)
An artifact of type AVRO with no description.

Key (odoo.public.sale_order_line-key)
An artifact of type AVRO with no description.

Figure 11. Registered Schemas in Service Registry

15.3. Demo Camel K

For this demo, we are going to capture the Debezium CDC event, sales order changed, and publish it to the Knative broker as a CloudEvent.

15.3.1. Camel K Kafka Subscriber

To build our Camel K Kafka subscriber:

- In Visual Studio Code, press **Ctrl**+**Shift**+**A**, or open **View** → **Command Palette** menu item
- Search and select **Create a new Apache Camel K Integration file**
- Select **Java**
- Select the default Workspace
- Name it **SaleOrderCDCKafka**

We are going to include a property file (which will result in a configmap deployment) in our integration.

- Add the following line after the // **camel-k: language**=**java** line.
- Create a new file, call it **kafka.properties**
- Add the following content:

```
camel.component.kafka.brokers=localhost:9092
camel.component.kafka.groupId=mygroup
camel.component.kafka.valueDeserializer=io.apicurio.registry.utils.serde.AvroKafkaDese
rializer
registryurl=http://localhost:8080/api
datumprovider=io.apicurio.registry.utils.serde.avro.ReflectAvroDatumProvider
globalid=io.apicurio.registry.utils.serde.strategy.GetOrCreateIdStrategy
```

Part V: Index

Index