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, refering 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.

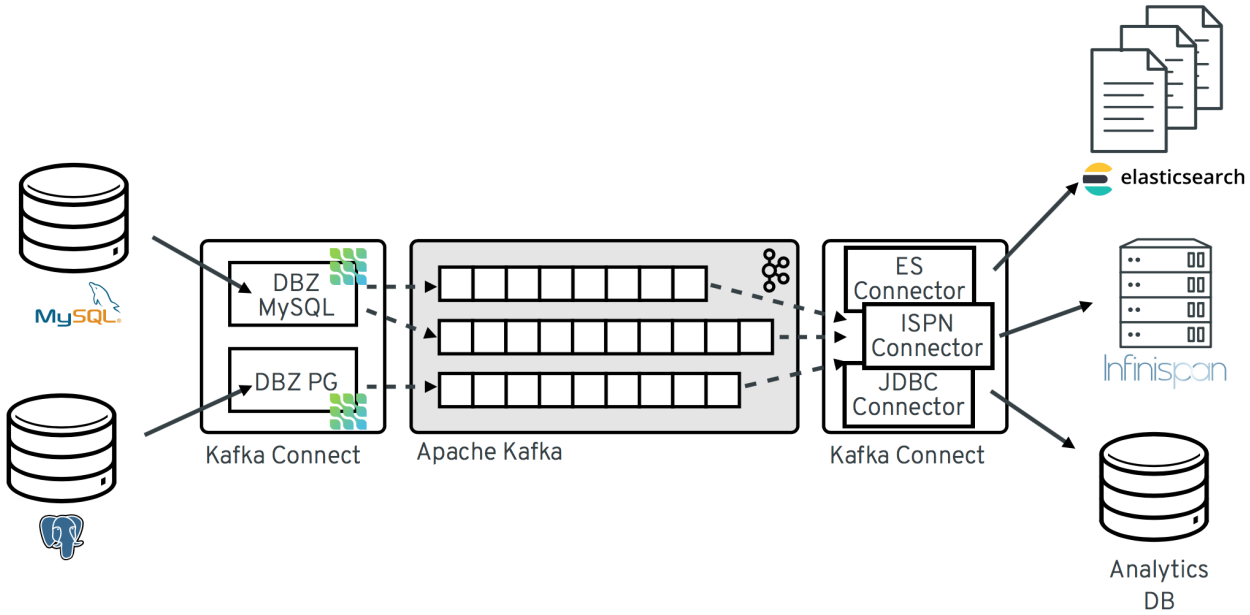


Figure 1. Debezium Arhitecture

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:

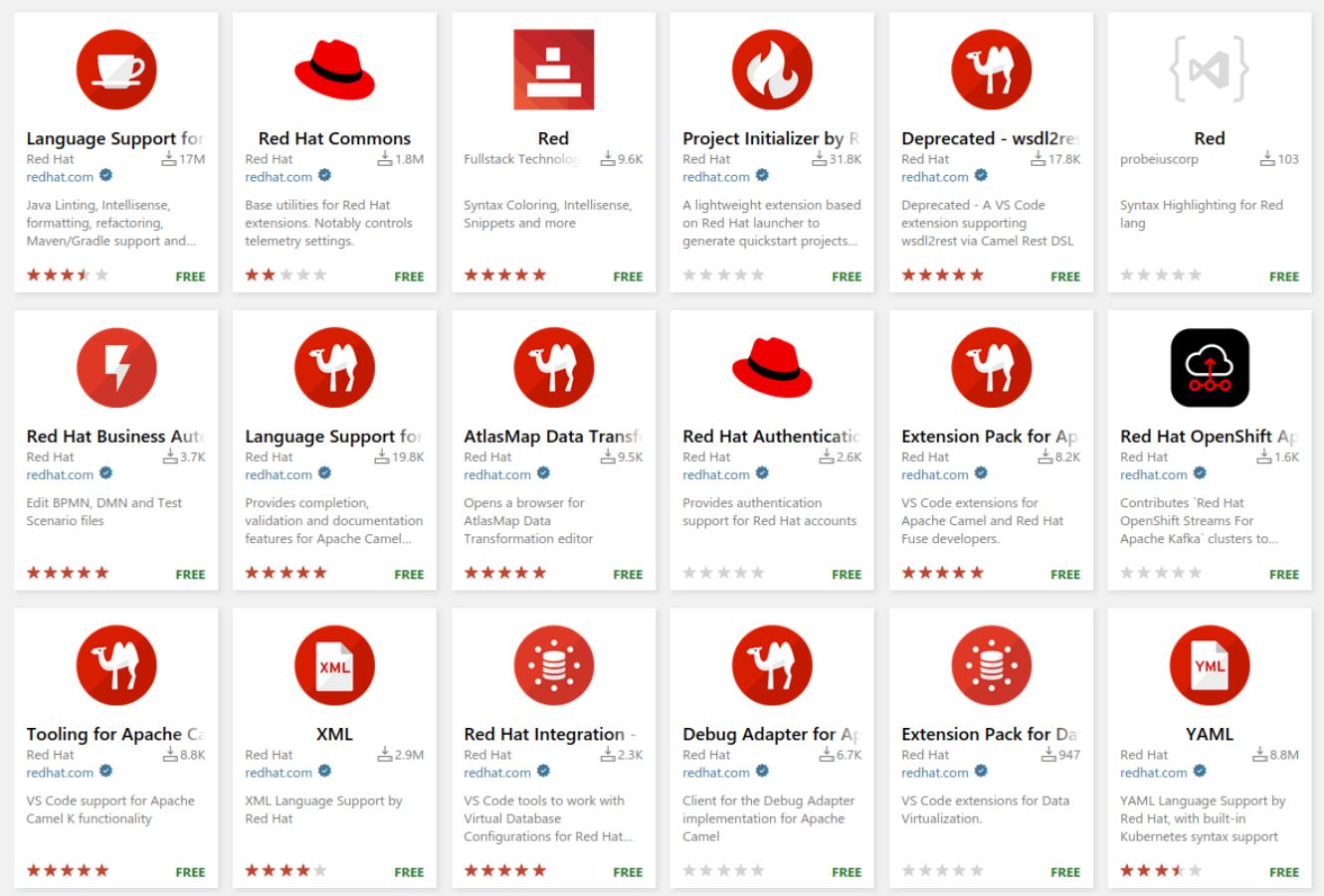


Figure 2. Red Hat supply a range of plugins for Microsoft Visual Studio Code

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. 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. 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

14.2. 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
```

14.3. 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.

14.4. 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.

14.5. 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.

14.6. 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.

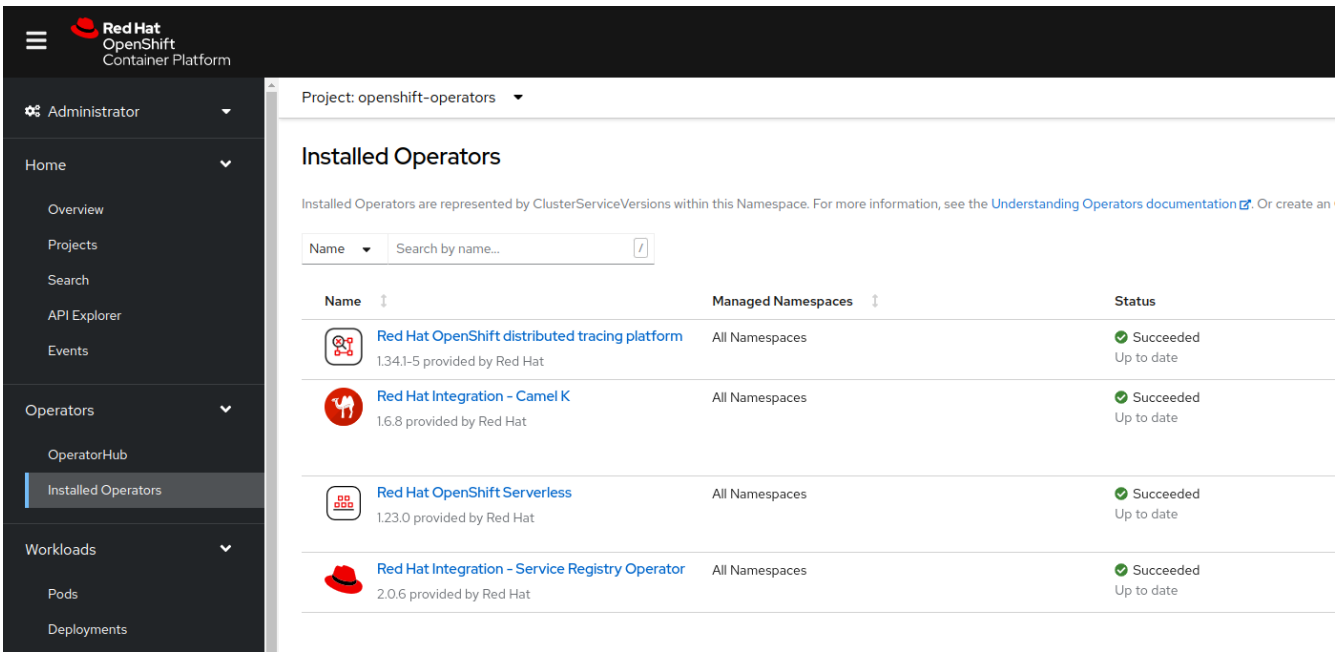


Figure 3. Installed Operators Summary

Chapter 15. 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.

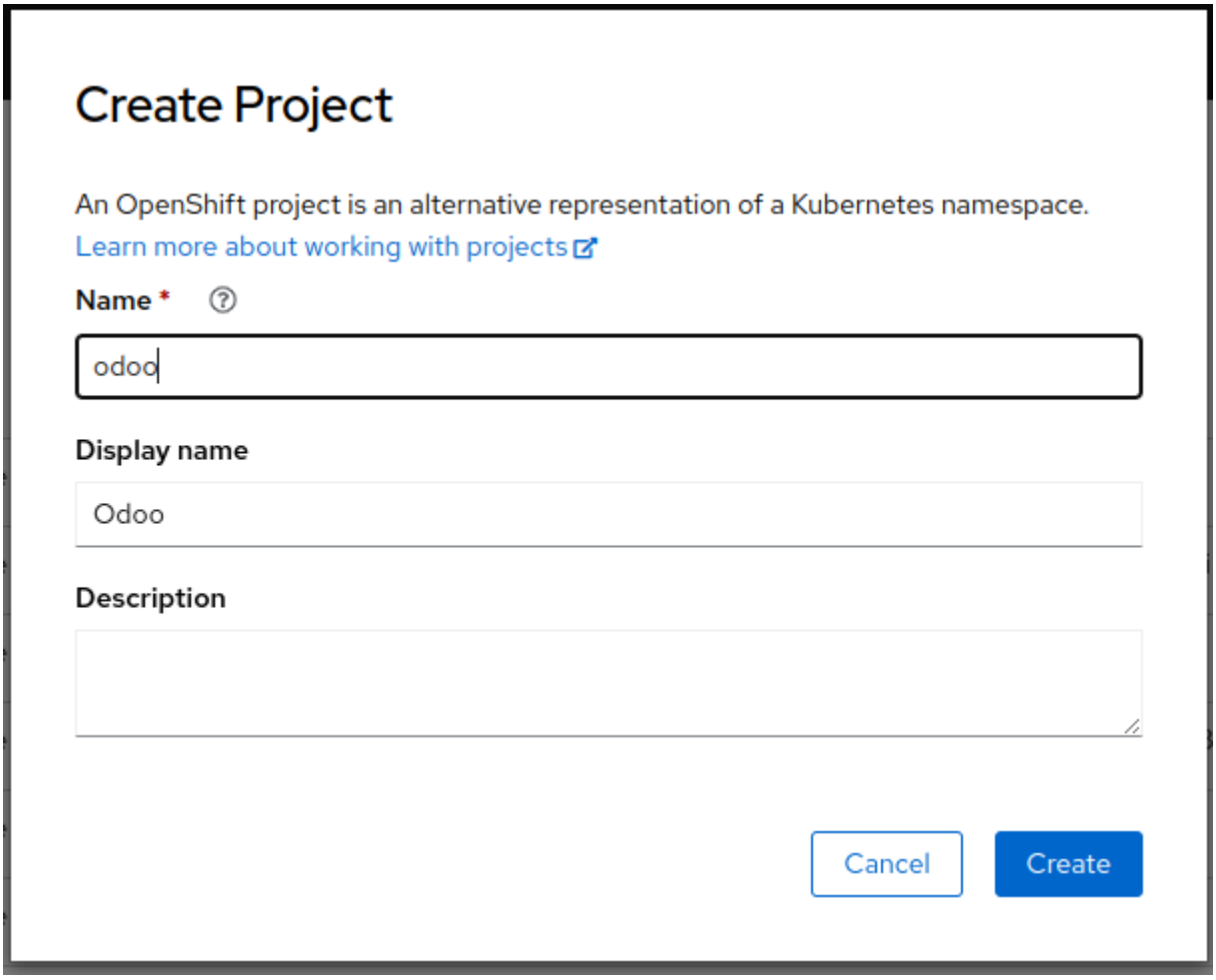
> Odoo is a suite of open source business apps that cover all your company needs: CRM, eCommerce, accounting, inventory, point of sale, project management, etc.
>
> Odoo's unique value proposition is to be at the same time very easy to use and fully integrated.
>
> — ODOO, https://odoo.com

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. 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



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.

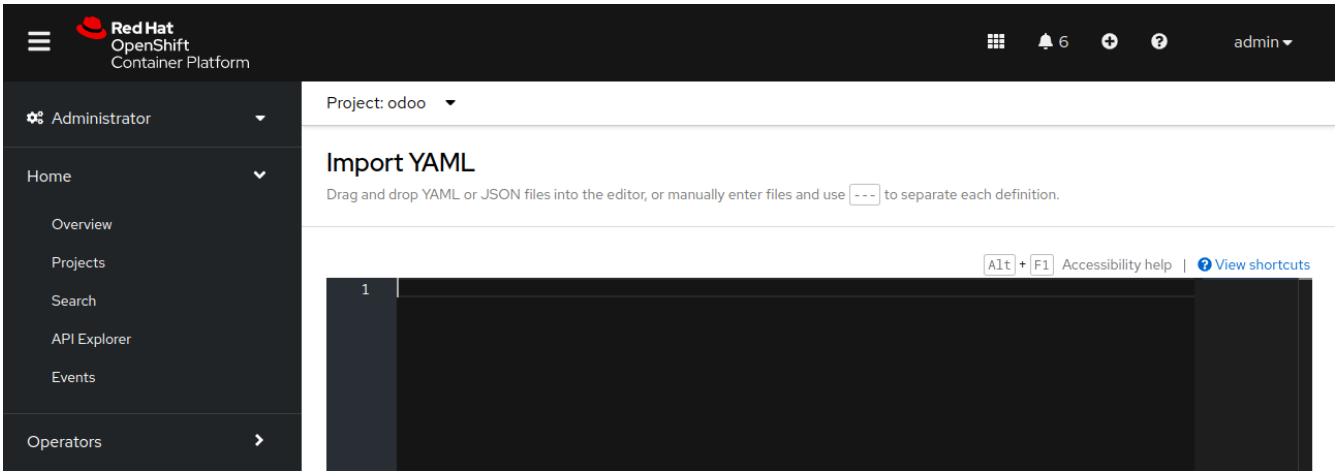
gitlab.paas.agesic.red.uy/franklin.gomez/odoo-ocp - Div. TE-Tecnologías, Agesic

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

15.2. 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:



- 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.

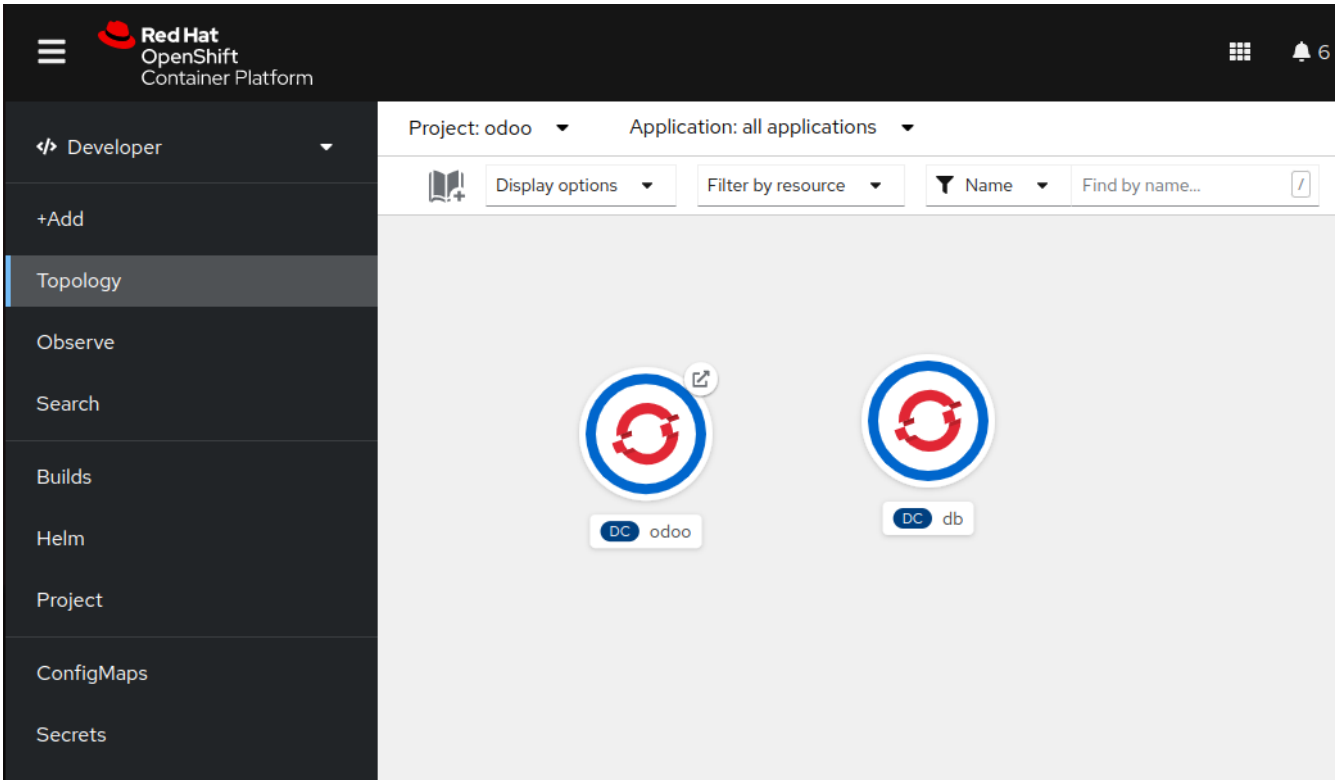
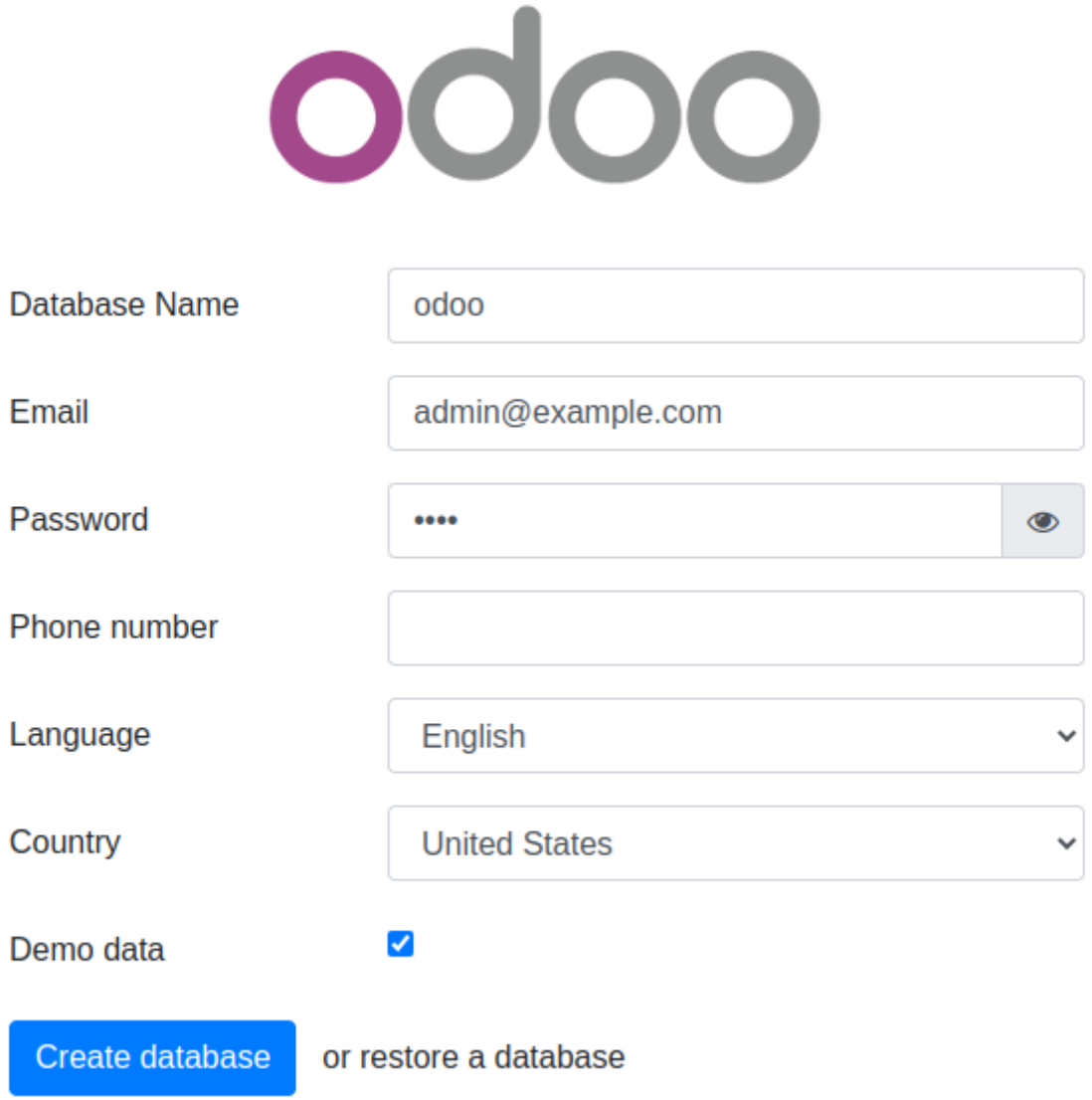


Figure 4. Odoo Deployment Completed

15.3. 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:

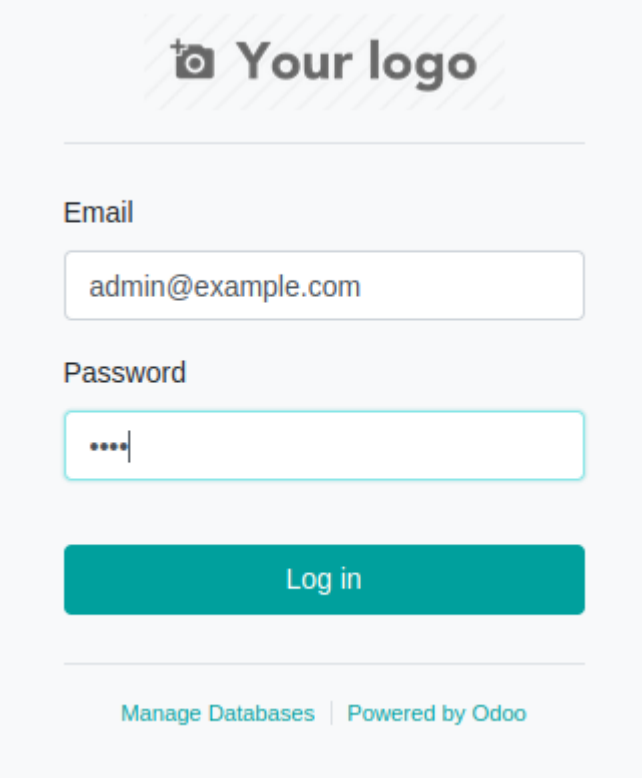


- Fill in the values as supplied in above screenshot, make sure that the **Email** is admin@example.com, **Password** is odoo, and that checkbox Demo data is selected
- Click on **Create Database**
- It will take a bit to build the database, so be patient after clicking the Create Database button. The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

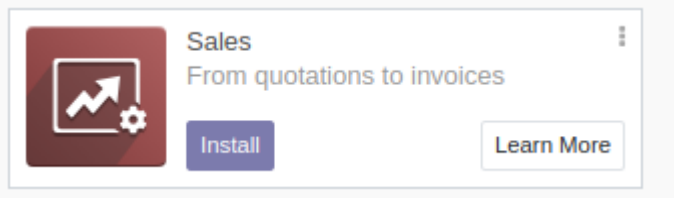
15.4. Customizing Odoo

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:

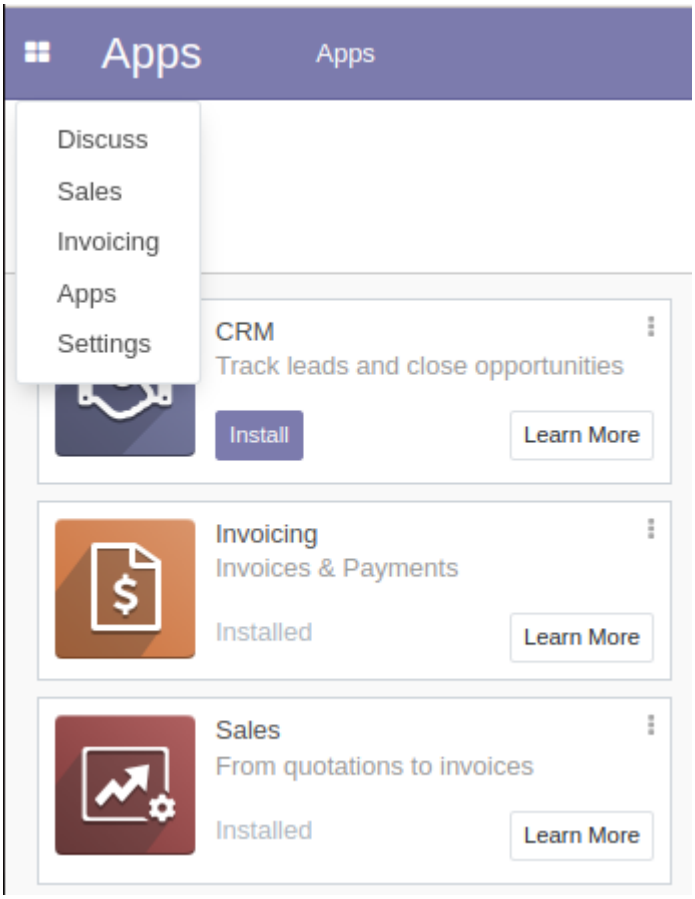


- Login with admin@example.com and the password odoo.
- Search for Sales, and click on **Install**



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.

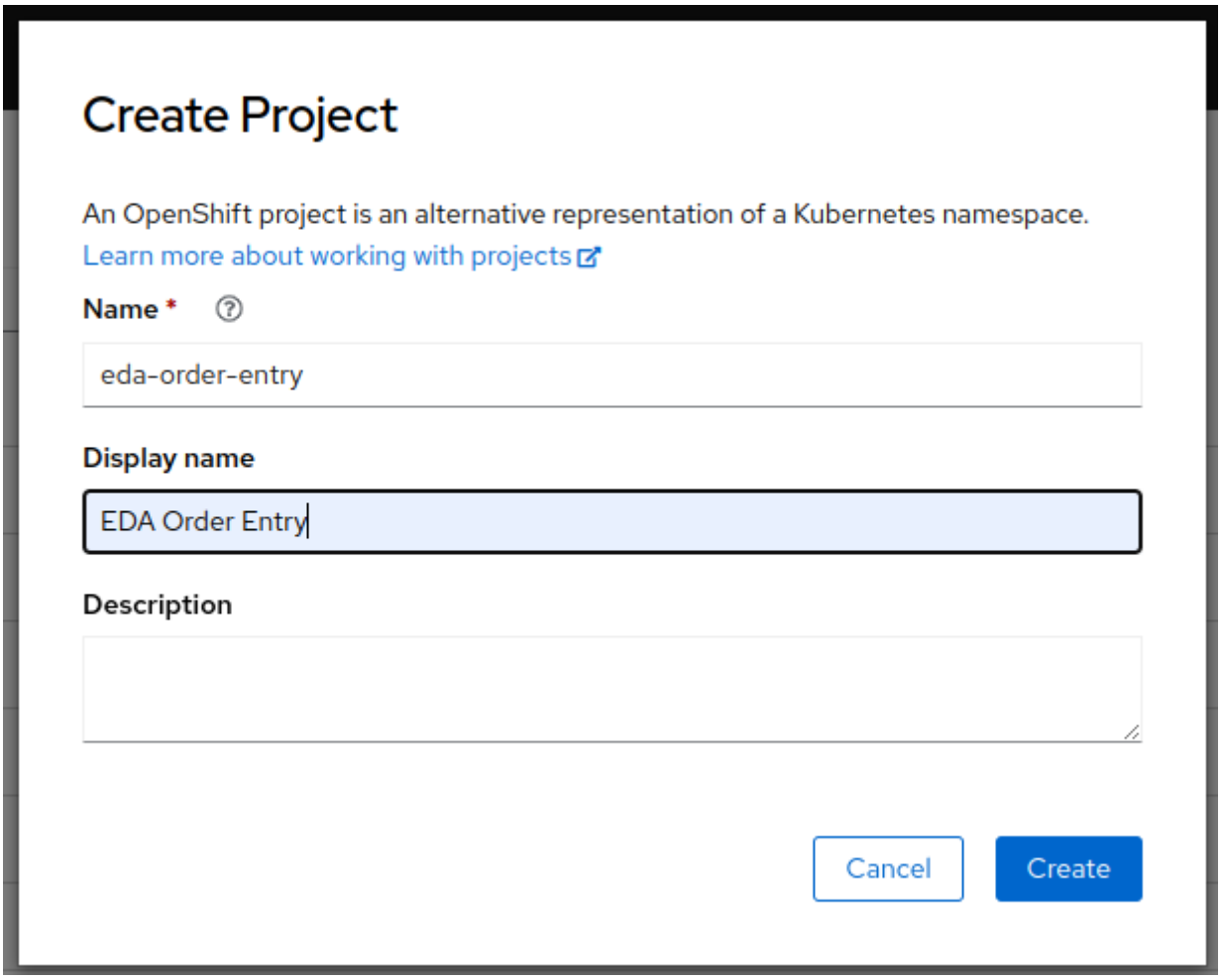

Apps
Apps
Discuss
Sales
Invoicing
Apps
Settings
CRM
Track leads and close opportunities
Install
Learn More
Invoicing
Invoices & Payments
Installed
Learn More
Sales
From quotations to invoices
Installed
Learn More

Chapter 16. Serverless Configuration

16.1. 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.

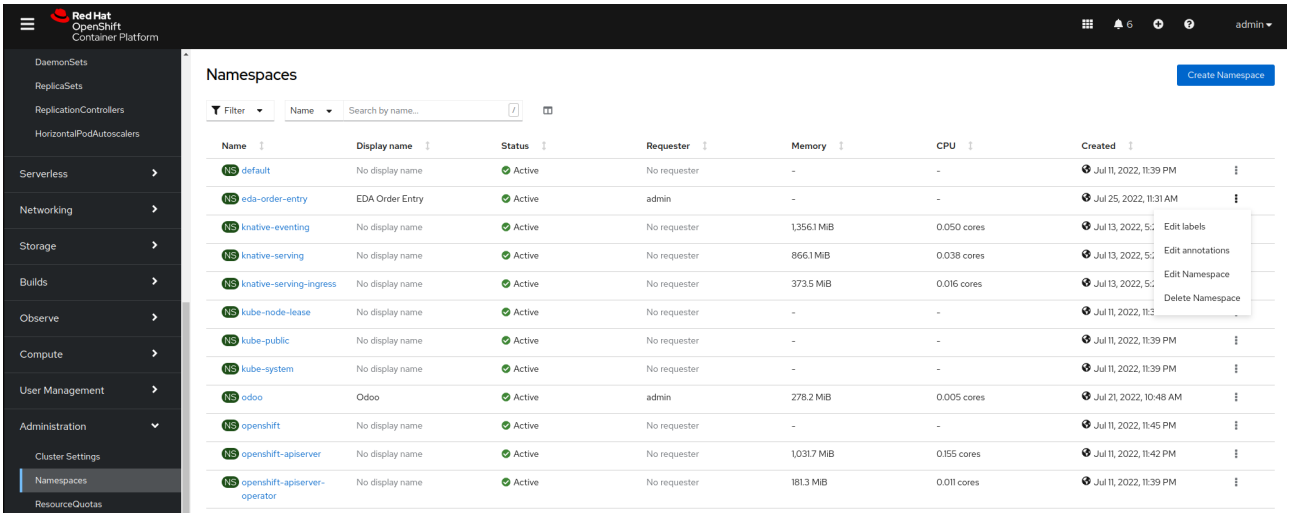
- From the **Administrator** perspective, got to the Projects from the Home section of the navigation pane.
- **Create project**, called **eda-order-entry**



16.2. Enable Knative Bindings

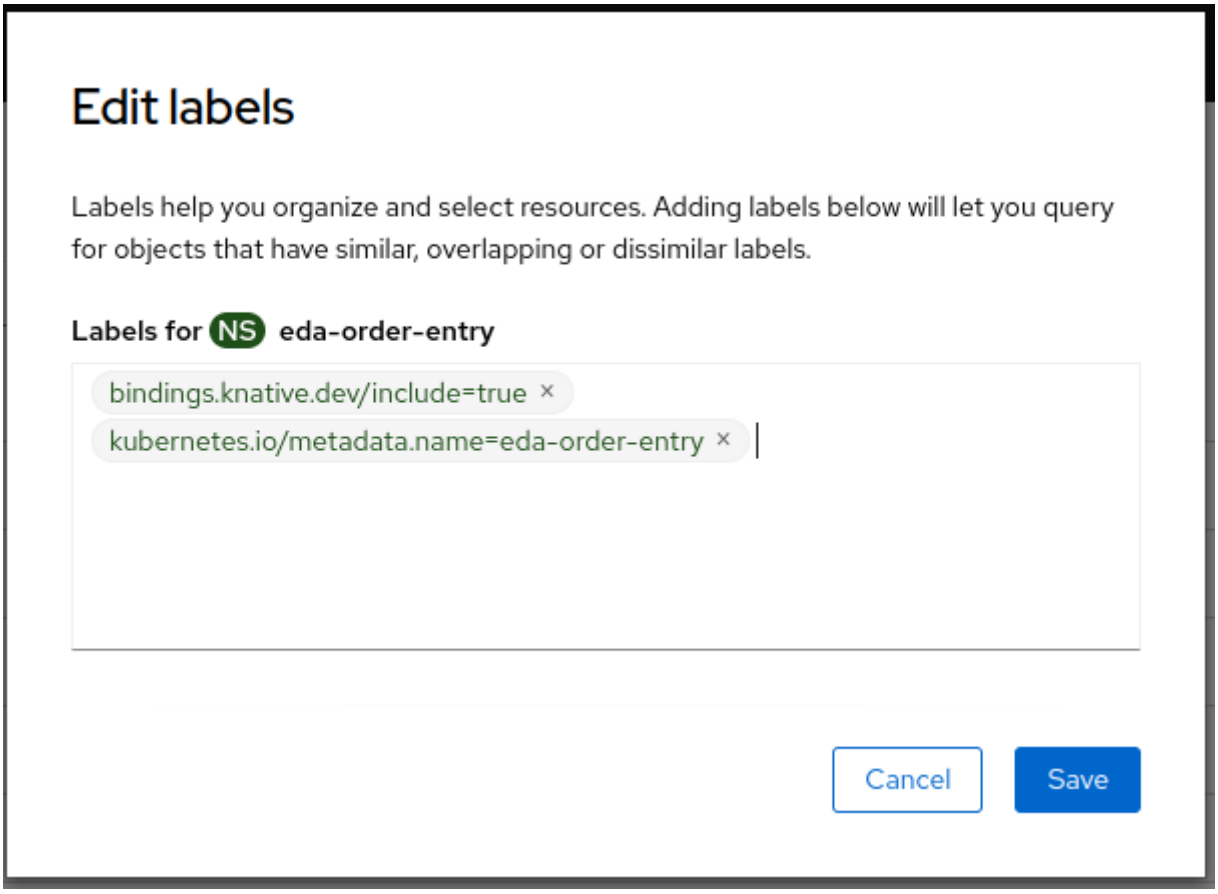
Next we have to make the project aware that it has the capability to bind to Knative services:

- 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**



- Add the label:

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

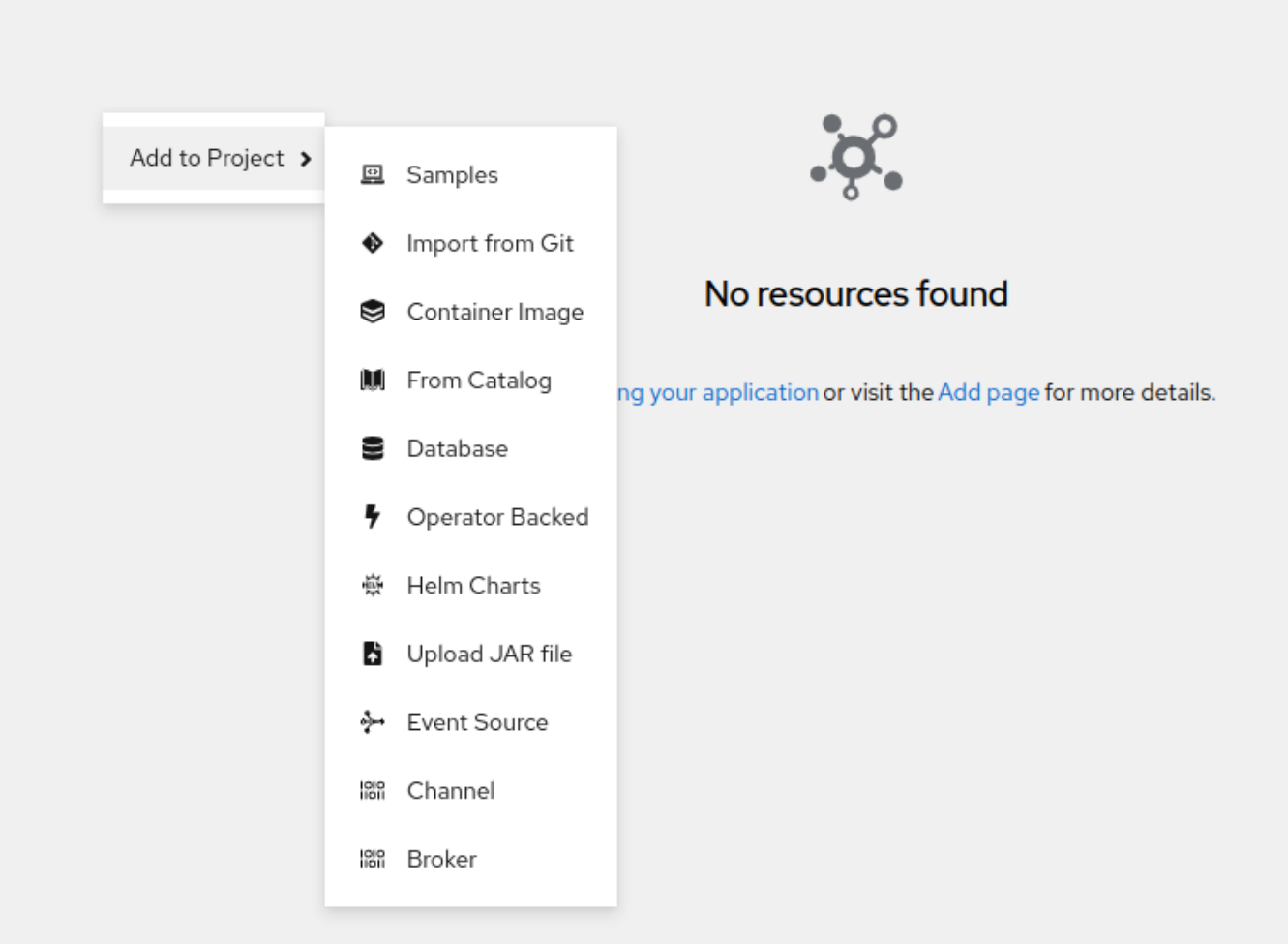


- Click **Save**

16.3. 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**

16.4. 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**, for exmple "Test Data", and specify the **Schedule** as */1 * * * *
- Make sure the **Input Target**'s **Resource** is pointing to the channel.
- Click on **Create**

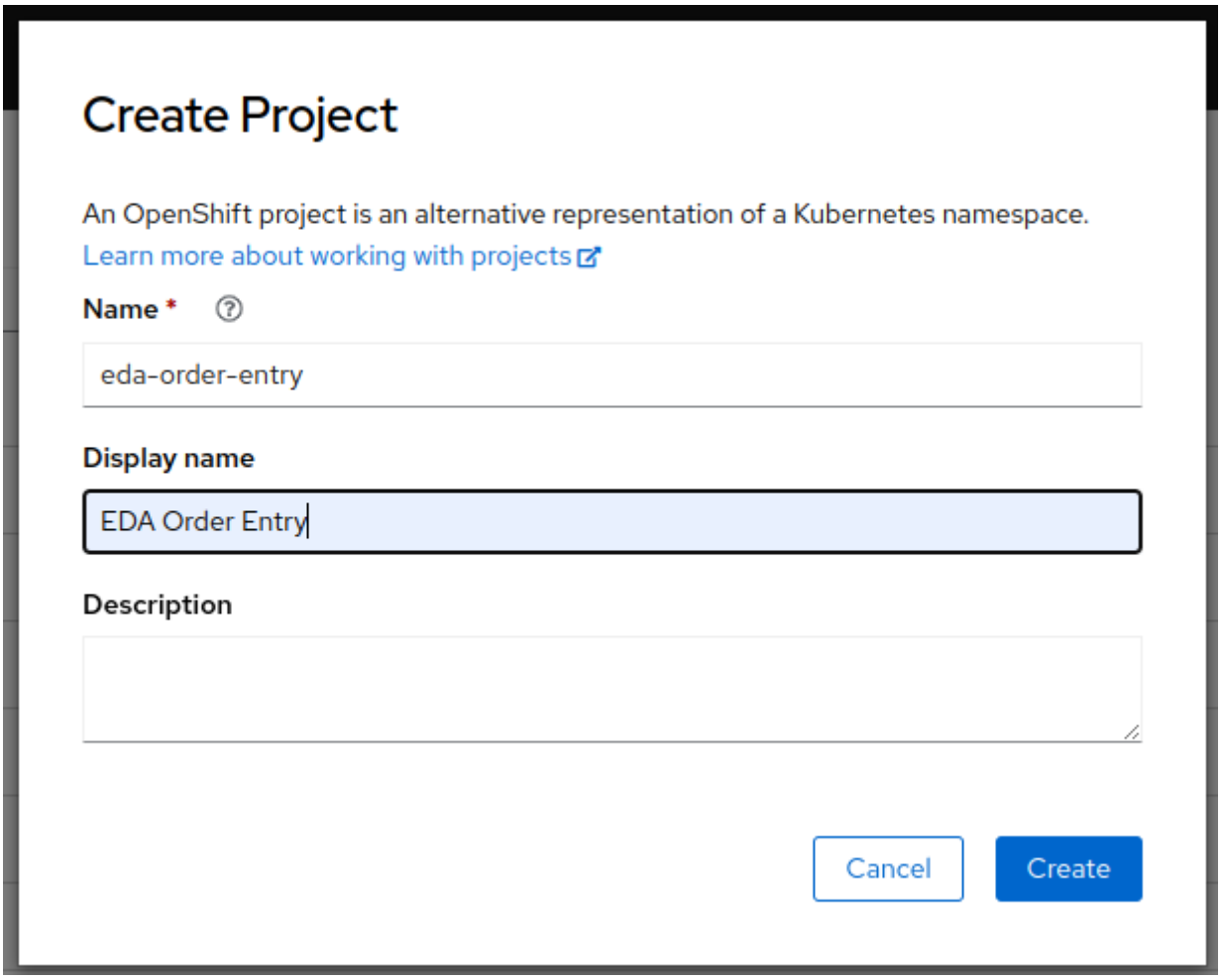
Feel free to move the components around.

Chapter 17. Serverless Configuration

17.1. 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.

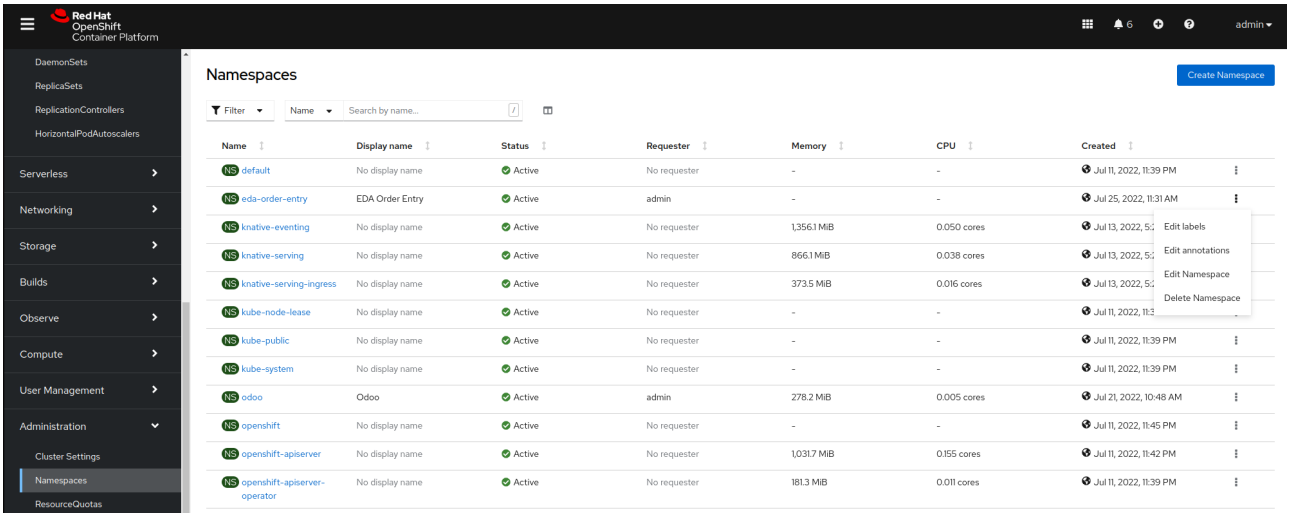
- From the **Administrator** perspective, got to the Projects from the Home section of the navigation pane.
- **Create project**, called **eda-order-entry**



17.2. Enable Knative Bindings

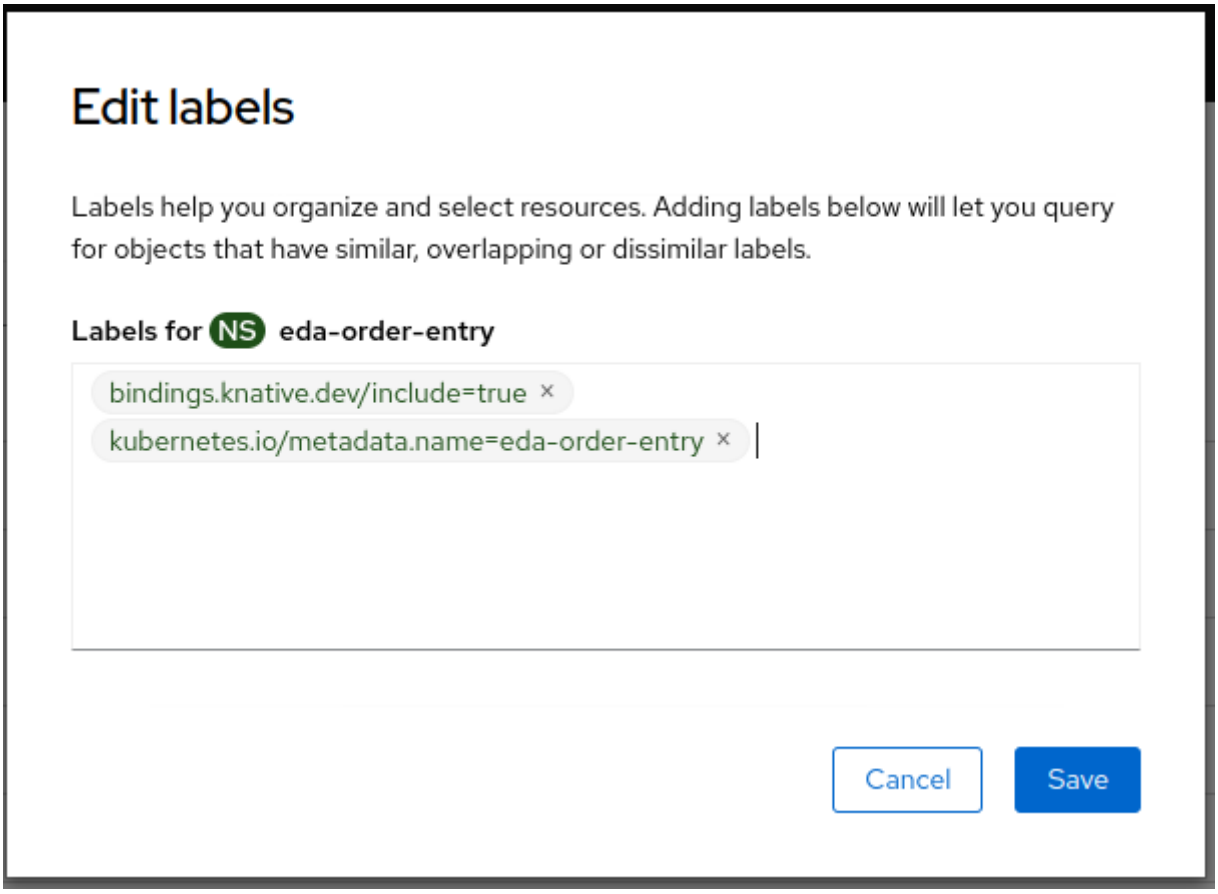
Next we have to make the project aware that it has the capability to bind to Knative services:

- 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**



- Add the label:

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

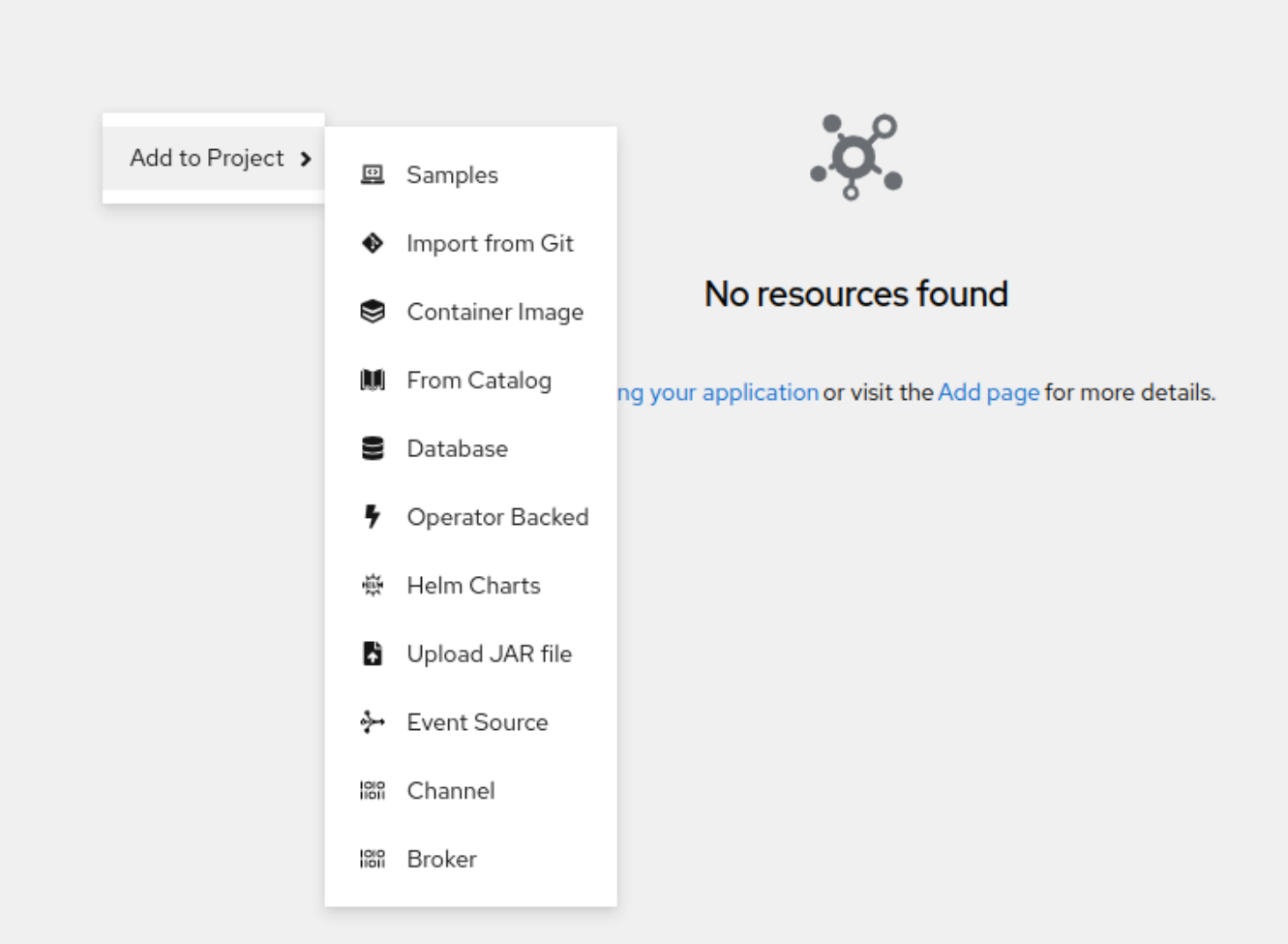


- Click **Save**

17.3. 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**

17.4. 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**, for exmple "Test Data", 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.

Chapter 18. x

Part V: Index

Index

C

cloud events, 5
CloudEvent, 5
CloudEvents, 5

O

OpenShift Data Foundation Operator, 23
OpenShift Serverless, 5
OpenShift Serverless Operator, 24

R

Red Hat Camel K Operator, 25
Red Hat distributed tracing platform, 26
Red Hat Service Registry Operator, 25