# Cloud Native Development - Demo Guide

The following multi-part demo showcases Cloud Native Development (CND) capabilities of the OpenShift platform. This is the standard demo for the new CND Sales Play. The CND Sales Play will be published in October 2020 in the [NA Comm Solution Sales Plays](https://redhat.highspot.com/spots/57b75a51127958263ff9e1ef).

Red Hat has customers needing to modernize existing applications and/or building new applications and deploy them to OpenShift. OpenShift is more than just Kubernetes, and we can highlight the CND capabilities of the OpenShift platform. For example, customers can employ CodeReady Workspaces (CRW), to rapidly onboard developers into a web-based IDE that runs in OpenShift. Also, customers can build new applications in Quarkus to maximize the benefits of running Java in containers. There are many areas to explore for CND with OpenShift, and this demo can kickstart the customer conversation on this topic.

## Demo #1: DayTrader — Application Modernization

Demo #1 demonstrates how to modernize a legacy stock trader application (DayTrader) to event-driven microservices using the [Transactional Outbox pattern](https://debezium.io/blog/2019/02/19/reliable-microservices-data-exchange-with-the-outbox-pattern/), which is a prescriptive way to implement change data capture (CDC). The demo uses the following Red Hat software:

* OpenShift 4.x
* [CodeReady Workspaces](https://developers.redhat.com/products/codeready-workspaces/overview) Operator
  + [Image Puller](https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.1/html/administration_guide/caching-images-for-faster-workspace-start_crw) for caching images
* [OpenLiberty](https://access.redhat.com/products/open-liberty) (ex-WebSphere) runtime on OpenShift
* [Red Hat Build of Quarkus](https://access.redhat.com/products/quarkus) runtime on OpenShift
* [AMQ Streams](https://access.redhat.com/documentation/en-us/red_hat_amq/7.7/html/amq_streams_on_openshift_overview/index) (i.e. Kafka, Strimzi)
* [Debezium](https://access.redhat.com/documentation/en-us/red_hat_integration/2020-q2/html/getting_started_with_debezium/index)
* MySQL image for OpenShift
* PostgreSQL image for OpenShift

### Use Case

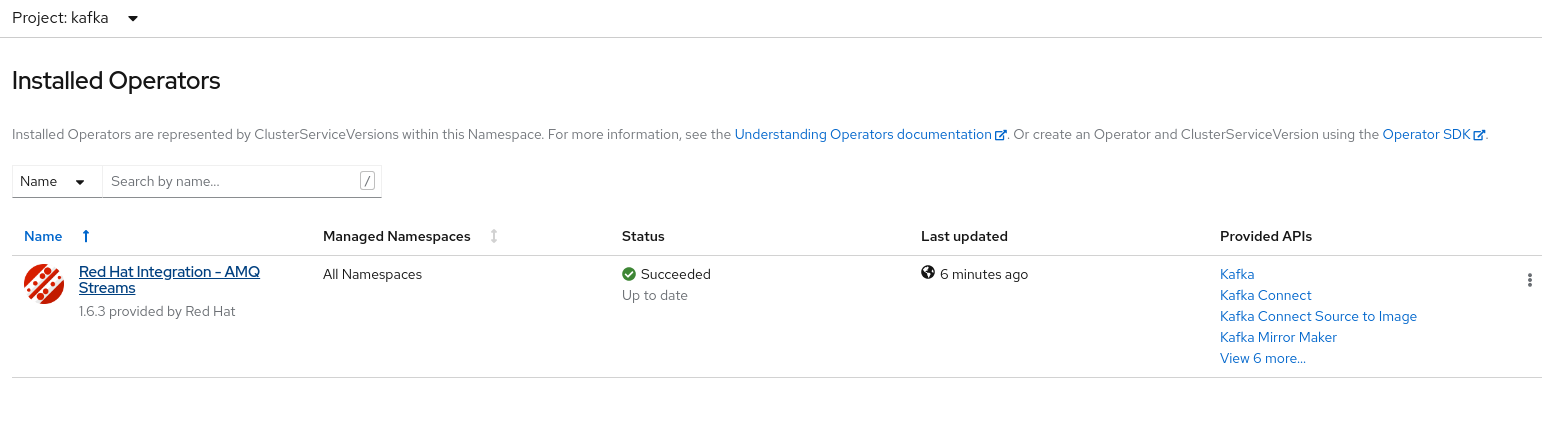
The customer has a legacy Java EE application, in this case WebSphere, that they need to modernize to a microservices that are running on OpenShift. The original code is old and unmaintainable, and a traditional “decompose the monolith” strategy is untenable. Additionally, as a mission-critical application, it needs to remain operational and maintain reliability and performance during this period of modernization.

### Architecture

#### 

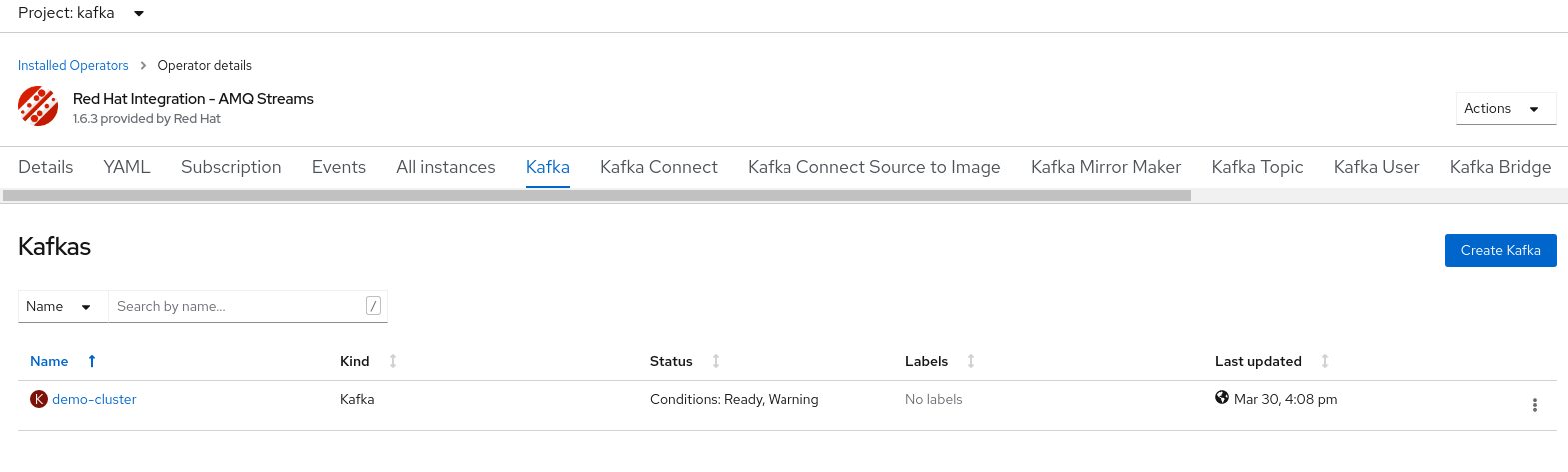
#### Walkthrough what's deployed in OpenShift

Login to OpenShift and navigate to the Kafka project, and view the "Installed Operators"



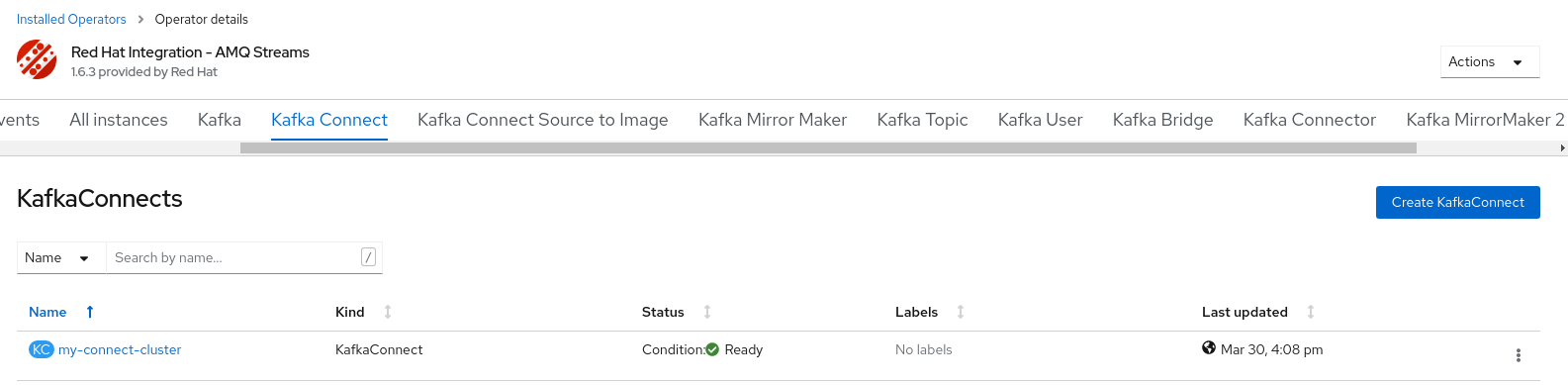
"This demonstration uses the Red Hat Integration AMQ Streams operator. When we look at this operator we will see the AMQ Streams cluster created and managed by the operator."

Click on the Red Hat Integration AMQ Streams Operator, then click on "Kafka"

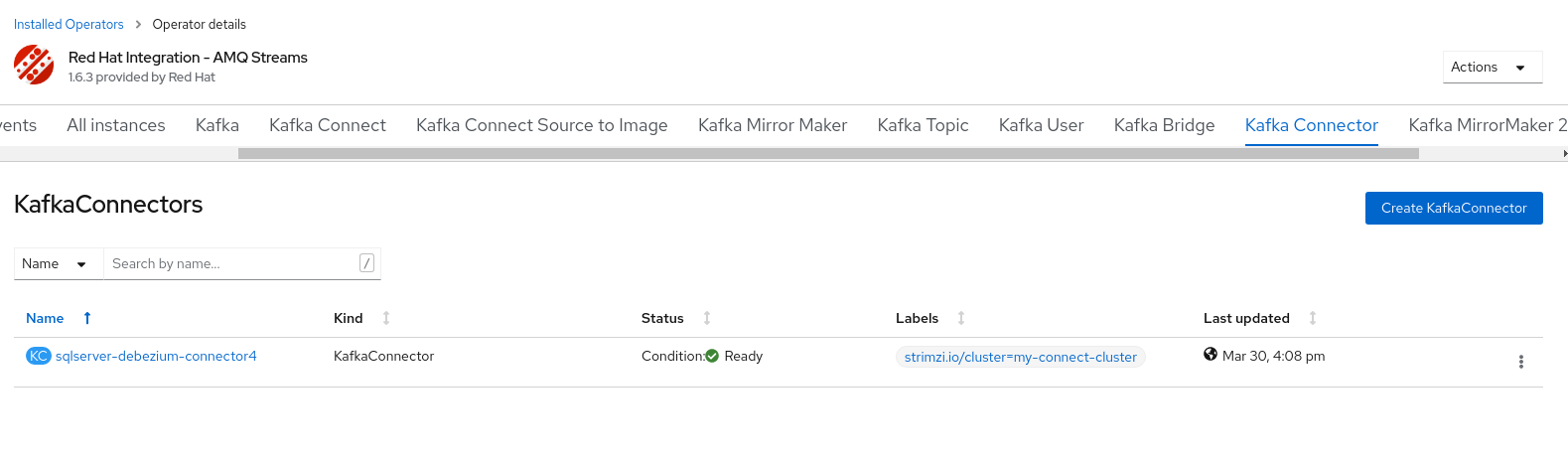


"We've deployed a Kafka cluster called 'demo-cluster', as we go through the other tabs in this operator view we can also see an instance of Kafka Connect"

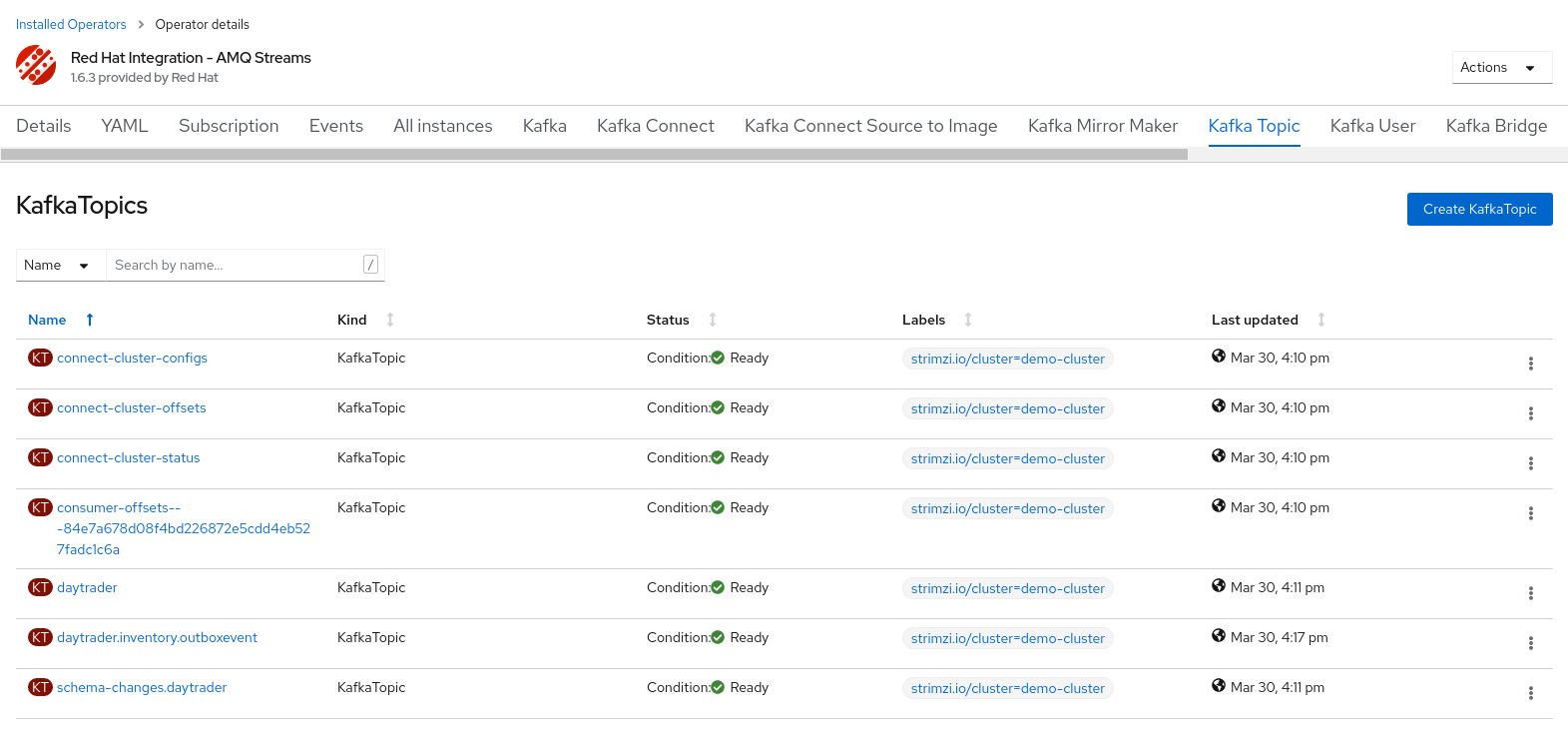
Click on the "Kafka Connect" tab



"We can see an instance of Kafka Connector"



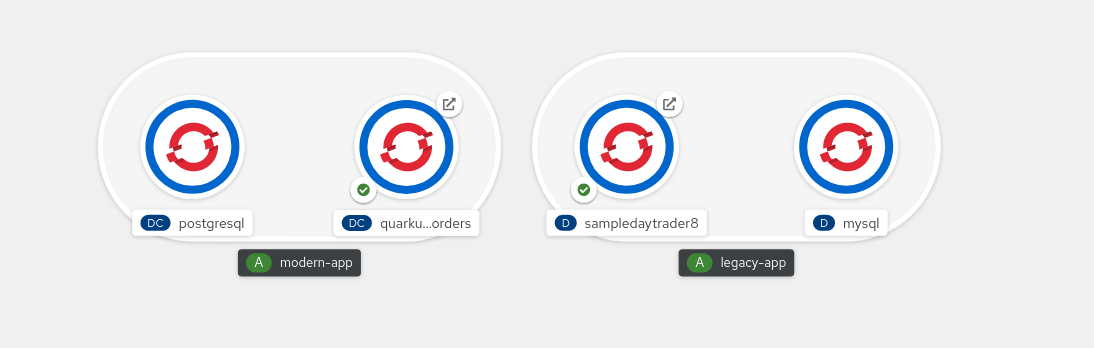
"This is the connection between Kafka and Debezium and MySql. The final thing deployed relating to Kafka are the Kafka topics, we can see these from the Kafka Topic tab"



"All of these objects are deployed and managed using the Red Hat AMQ Streams operator, this makes it extremely easy to integrate the management of Kafka with Infrastructure as Code"

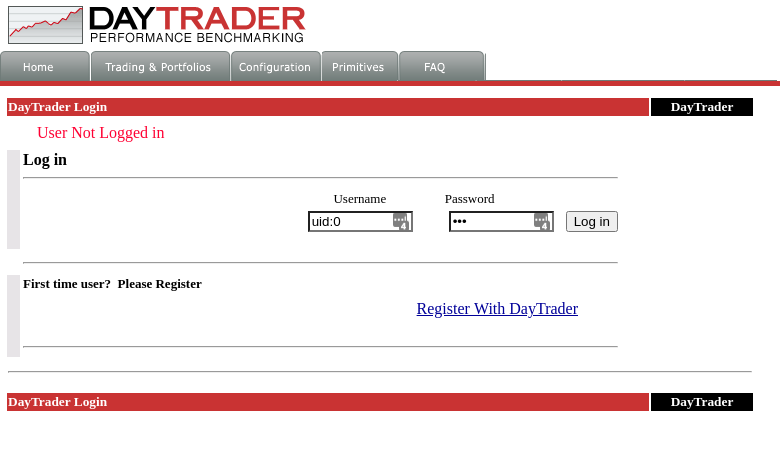
"Moving on from Kafka to the application related workloads, we'll now take a look at the daytrader project"

Switch to the Developer perspective, and navigate to the daytrader project. Look at the Topology:



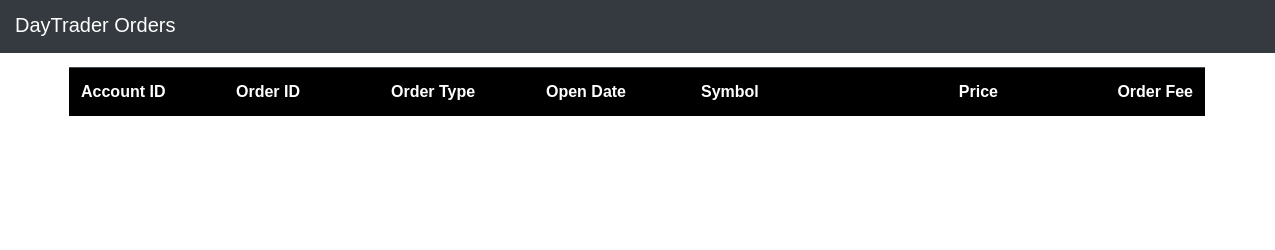
"What we can see here are 4 pods grouped into two applications; modern-app and legacy-app. In the legacy app group we have the legacy websphere application and MySQL database. In the modern app group we have the quarkus application and a postgres database. "

Click on the external route icon for the sample day trader application.



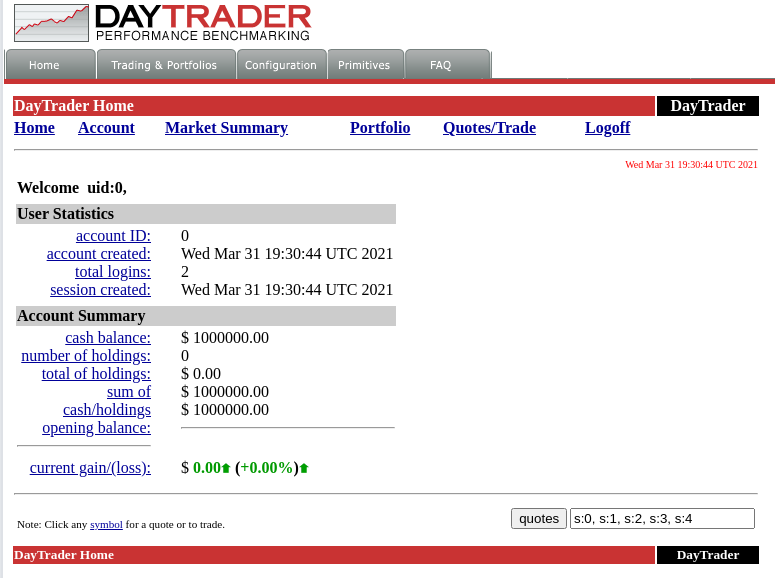
"This is the login page for the legacy application, day trader. "

Click on the external route icon for the Quarkus application.

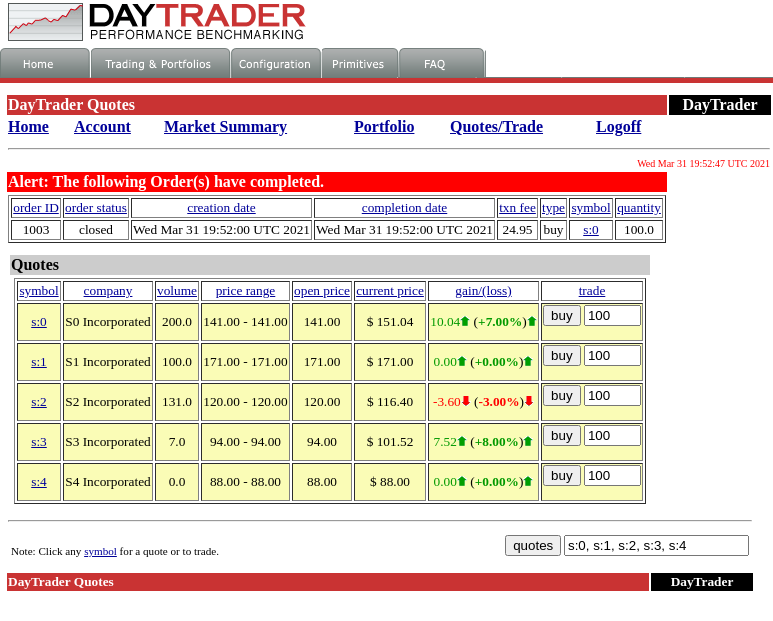


"This is the modern application, a Quarkus based event driven application which subscribes to a kafka topic to receive messages"

Login to the day trader application by switching back to the daytrader application and clicking on login.

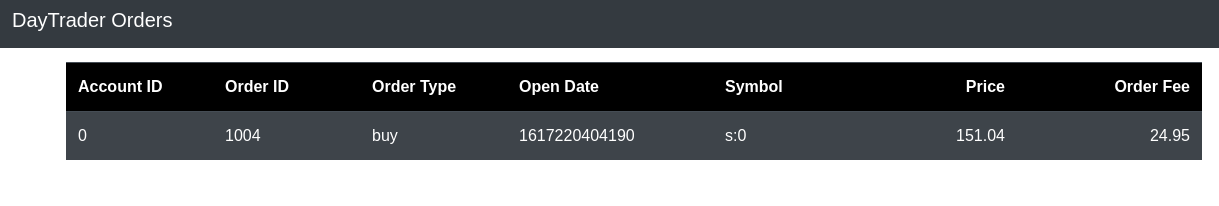


Click on "Quotes/Trade"



"From this view we can see some mock quotes and prices, we are now going to simulate a trade. We're going to buy 100 stocks for company S0. Once we place this trade we will immediately see this record appearing in our modern Quarkus application."

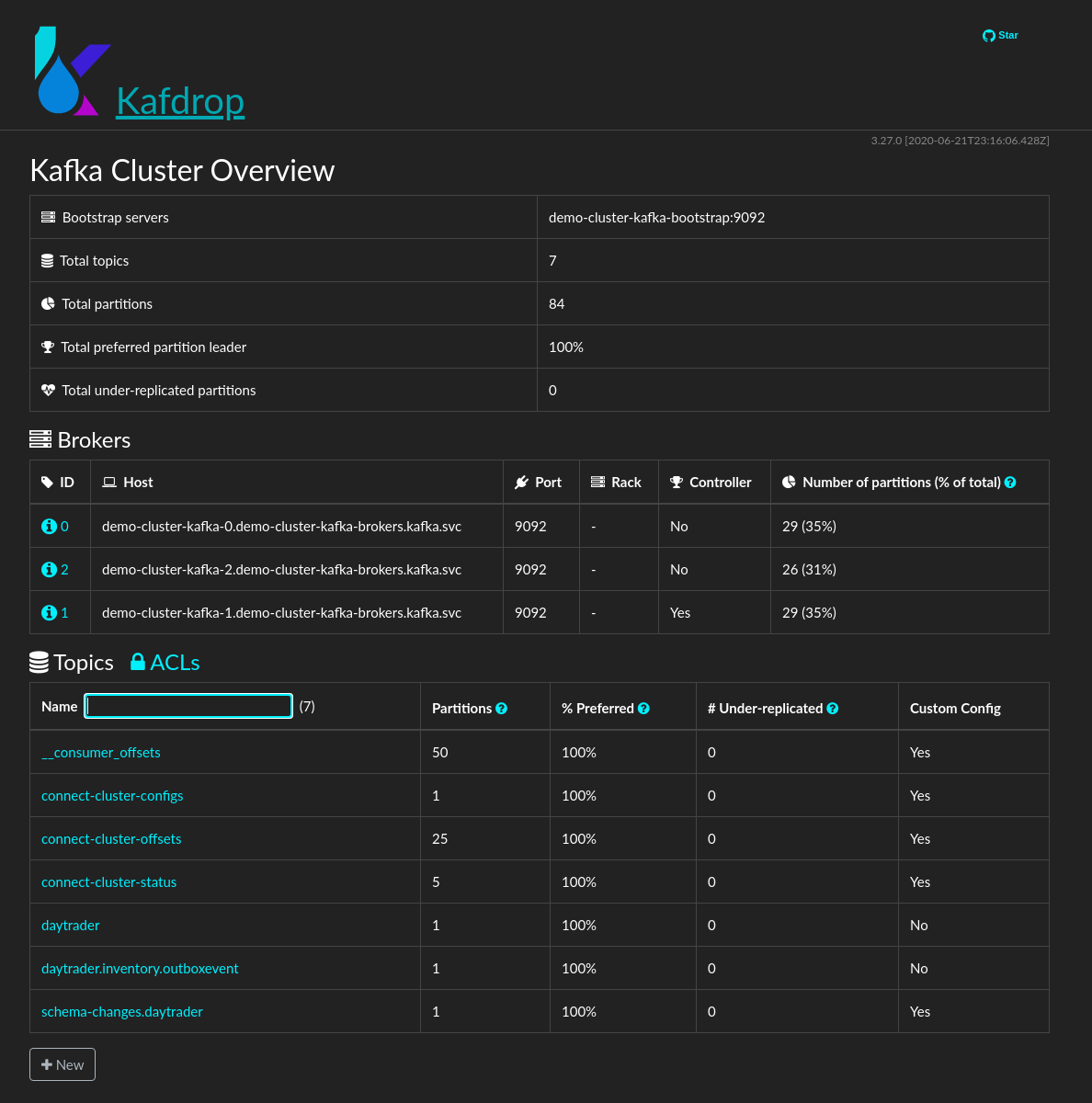
Click on "buy" on the first row of the table. Switch to the Quarkus app.



"We've made a purchase on the legacy site, and we can now see this reflected on the modern app.

So what's happened here? The interface and usage of the legacy application hasn't changed. A small code change was made to the legacy app to write the details of the transaction to a new table using the outbox pattern. Debezium picks it up from here, without adding any additional load to MySQL, it reads the data from the transaction logs. Debezium writes the message to a Kafka topic, the Quarkus application subscribes to this topic and processes the message."

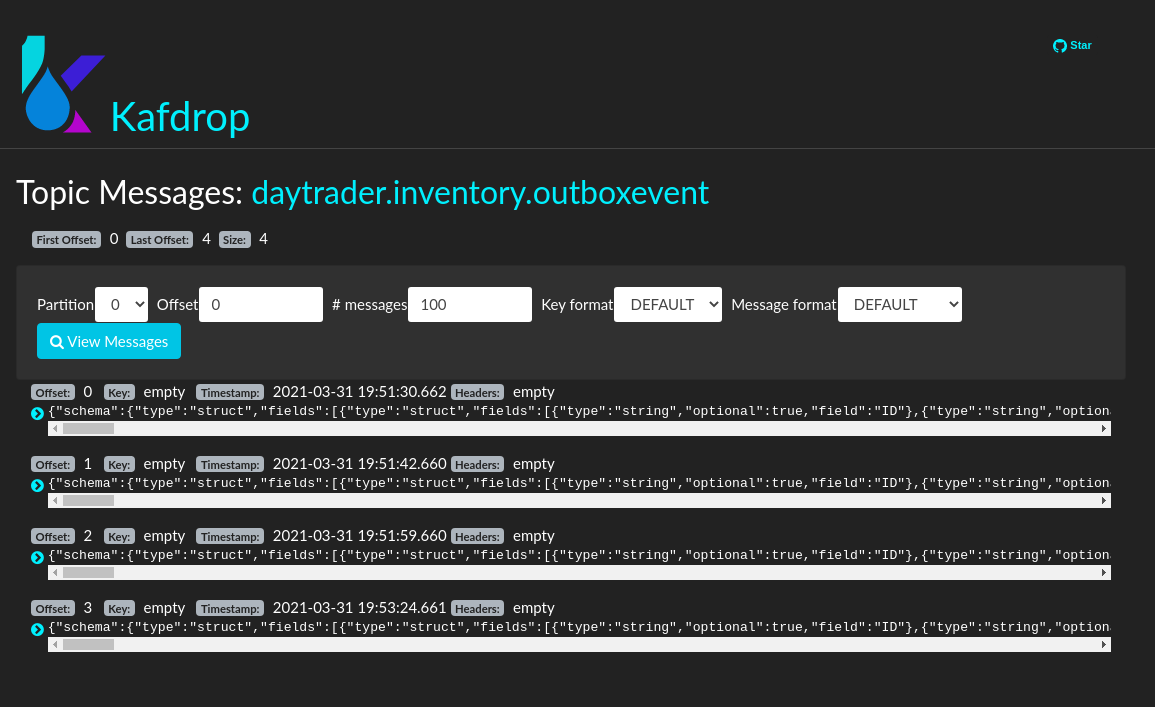
From OpenShift, navigate to the Kafka project and click on the "Kafdrop" route.



"We've also deployed a Kafka message viewer, kafdrop. If we navigate through this ui we can see the json messages created by debezium based on MySQL records."

Navigate daytrader.inventory.outboxevent topic.

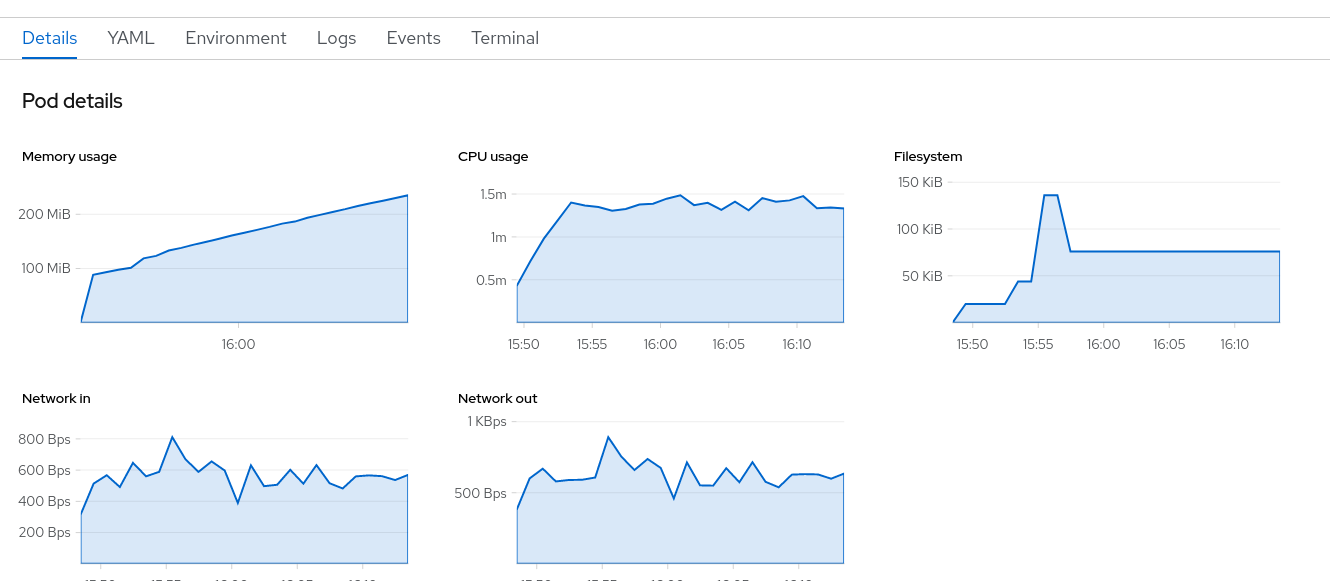
Click on "View Messages" and then "View Messages" again.



"You can see here the records created in Kafka by debezium. "

"Final thing we're going to show is the extremely low resource utilization of our quarkus application"

Navigate to the daytrader project, click on the quarkus application pod, and show the details tab:



"CPU usage and memory utilization are extremely low, that's one of the key benefits of Quarkus allowing for much improved container density resulting in lower infrastructure costs."

"So what we've seen here is pretty cool. We've interfaced with a legacy application, which in real world terms could be problematic to modernize for a variety of reasons. The integration we've performed has not introduced any new dependencies to this legacy application, simply writing to an existing database. These records are then captured using CDC and written to Kafka. Once this is in place we're now able to utilize modern application development techniques."

## Demo #2: Currie — New Application Development (Coming Soon)

Demo #2 demonstrates how to build a brand new application that allows a user to request a driver (courier) to pick up and deliver a package. The demo uses the following Red Hat software:

* OpenShift 4.x
* [CodeReady Workspaces](https://developers.redhat.com/products/codeready-workspaces/overview) Operator
  + [Image Puller](https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.1/html/administration_guide/caching-images-for-faster-workspace-start_crw) for caching images
* [Red Hat Build of Quarkus](https://access.redhat.com/products/quarkus) runtime on OpenShift
* [AMQ Streams](https://access.redhat.com/documentation/en-us/red_hat_amq/7.7/html/amq_streams_on_openshift_overview/index) (i.e. Kafka, Strimzi)
* [Data Grid](https://access.redhat.com/documentation/en-us/red_hat_data_grid/8.1/)

***Currie is under development and will be included in a future release of this demo guide.***

|  |
| --- |
| [Click to See Video](http://todo) |

