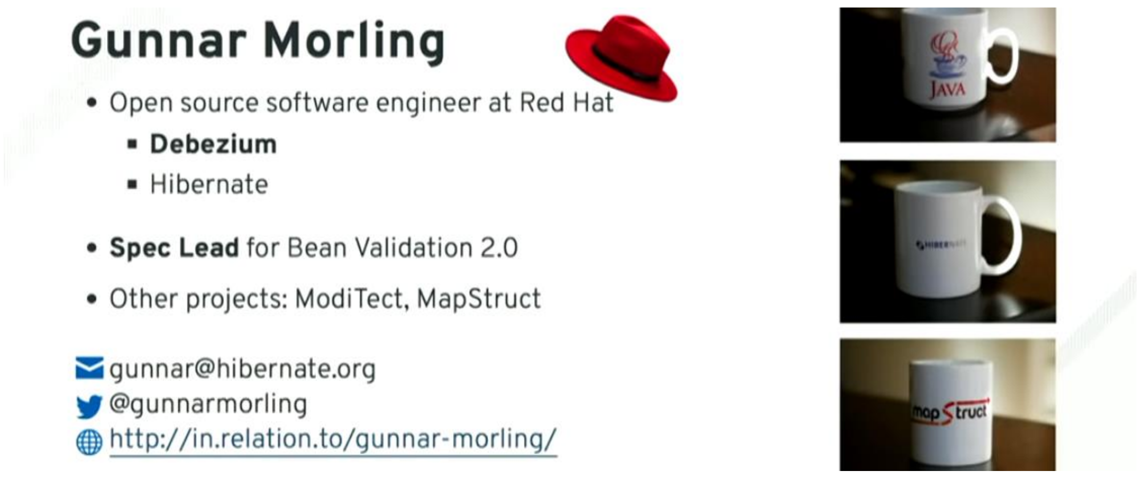


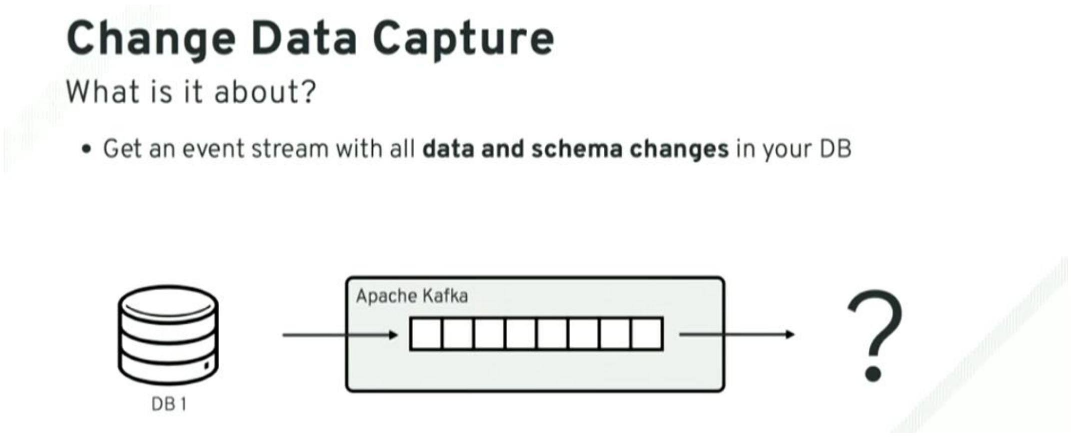
Gunnar Morling

- Open source software engineer at Red Hat
  - **Debezium**
  - Hibernate
- **Spec Lead** for Bean Validation 2.0
- Other projects: ModiTect, MapStruct

gunnar@hibernate.org
@gunnarmorling
http://in.relation.to/gunnar-morling/

JAVA
map struct

Streaming changes from your datastore enables you to solve multiple challenges: synchronizing data between microservices, maintaining different read models in a CQRS-style architecture, updating caches and full-text indexes, and feeding operational data to your analytics tools.

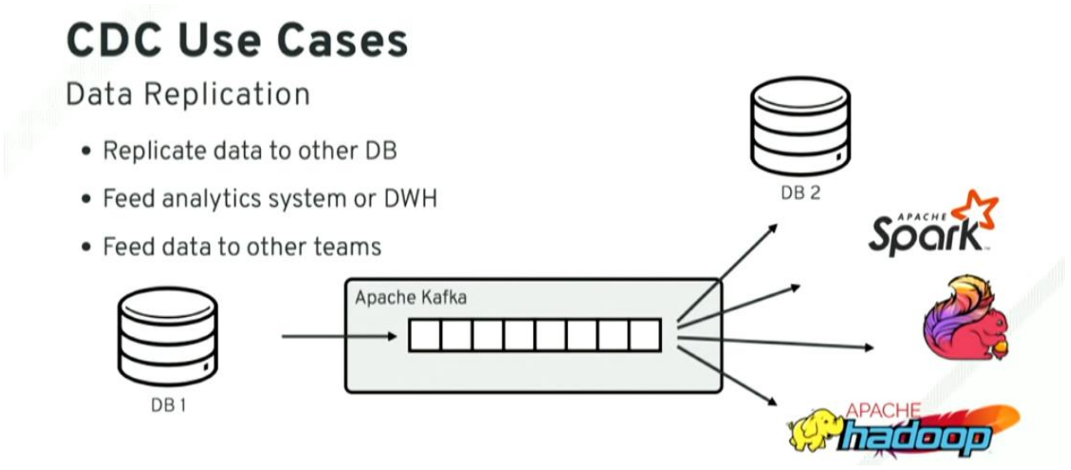


Change Data Capture

What is it about?

- Get an event stream with all **data and schema changes** in your DB

Join this session to learn what change data capture (CDC) is about and how it can be implemented using Debezium, an open-source CDC solution based on Apache Kafka. Find out how Debezium captures all the changes from datastores such as MySQL, PostgreSQL and MongoDB, how to react to the change events in near real-time, and how Debezium is designed to not compromise on data correctness and completeness also if things go wrong.



CDC Use Cases

Data Replication

- Replicate data to other DB
- Feed analytics system or DWH
- Feed data to other teams

In a live demo we'll show how to set up a change data stream out of your application's database, without any code changes needed. You'll see how to sink the change events into other databases and how to push data changes to your clients using WebSockets.

CDC Use Cases

Microservices

- **Microservice** Data Propagation
- **Extract microservices** out of monoliths

CDC Use Cases

Others

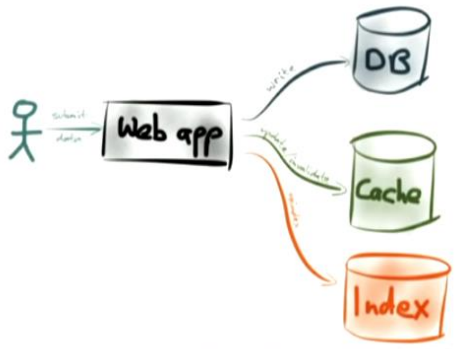
- **Auditing**/Historization
- Update or invalidate **caches**
- Enable **full-text search** via Elasticsearch, Solr etc.
- Update **CQRS** read models
- UI **live updates**
- Enable **streaming queries**

How to Capture Data Changes?

How to Capture Data Changes?

Possible approaches

- Dual writes
  - Failure handling?
  - Prone to **race conditions**
- Polling for changes
  - How to find changed rows?
  - How to handle deleted rows



https://www.confluent.io/blog/using-logs-to-build-a-solid-data-infrastructure-or-why-dual-writes-are-a-bad-idea/

How to Capture Data Changes!

Monitoring the DB

- Apps write to the DB -- changes recorded in log files, then tables updated
  - Used for TX recovery, replication etc.
- Let's **read the database log** for CDC!
  - MySQL: binlog; Postgres: write-ahead log; MongoDB op log
- **Guaranteed consistence**
  - All events, deletes
- **Transparent** to upstream applications

Apache Kafka

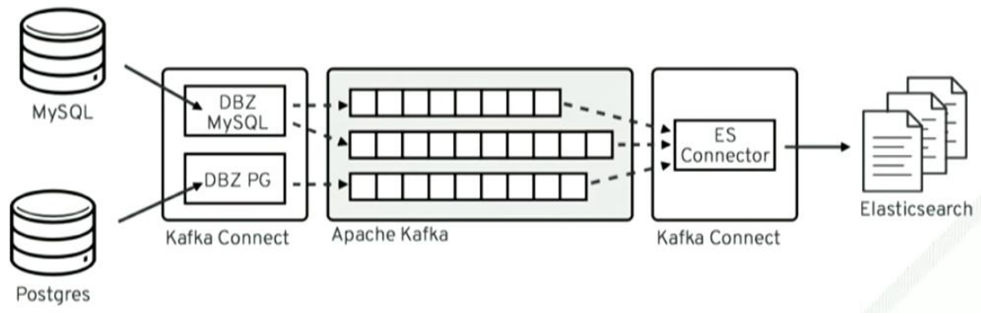
Perfect Fit for CDC

- Guaranteed ordering (per partition)
- Pull-based
- Scales horizontally
- Supports compaction

Kafka Connect

- A framework for **source** and **sink** connectors
- **Track offsets**
- Schema support
- Clustering
- Rich **eco-system** of connectors

CDC Topology with Kafka Connect



CDC Message Structure

- Key (PK of table) and Value
- Payload: **Before** state, **After** state, **Source** info
- Serialization format:
  - JSON
  - Avro (with Confluent Schema Registry)

```
{
  "schema": {
    ...
  },
  "payload": {
    "before": null,
    "after": {
      "id": 1004,
      "first_name": "Anne",
      "last_name": "Kretchmar",
      "email": "annek@noanswer.org"
    },
    "source": {
      "name": "dbserver1",
      "server_id": 0,
      "ts_sec": 0,
      "file": "mysql-bin.000003",
      "pos": 154,
      "row": 0,
      "snapshot": true,
      "db": "inventory",
      "table": "customers"
    },
    "op": "c",
    "ts_ms": 1486500577691
  }
}
```

Debezium Connectors

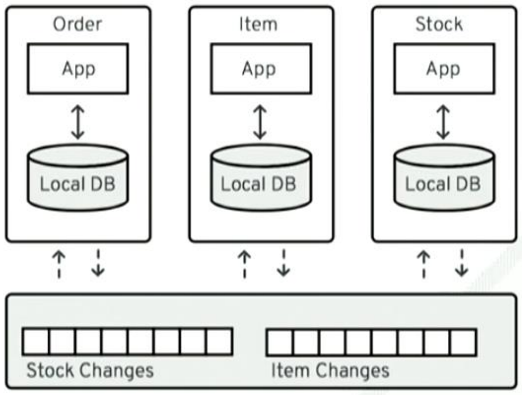
- **MySQL**
- **Postgres**
- **MongoDB**
- **Oracle** (Tech Preview, based on XStream)
- **SQL Server** (Tech Preview)
- Possible future additions
  - Cassandra?
  - MariaDB?

Change Data Streaming Patterns

Pattern: Microservice Data Synchronization

Microservice Architectures

- Propagate data between different services **without coupling**
- Each service keeps **optimised views locally**



Pattern: Microservice Extraction

Migrating from Monoliths to Microservices

- Extract microservice for single component(s)
- Keep write requests against running monolith
- Stream changes to extracted microservice
- Test new functionality
- Switch over, evolve schema only afterwards

Pattern: Materialize Aggregate Views

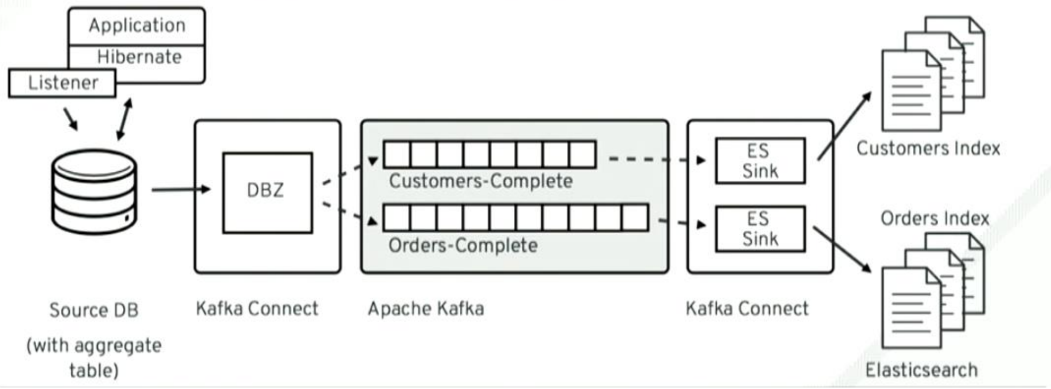
E.g. Order with Line Items and Shipping Address

- Distinct topics by default
- Often would like to have views onto **entire aggregates**
- Approaches
  - Use **KStreams** to join table topics
  - **Materialize views** in the source DB

```
{
  "id" : 1004,
  "firstName" : "Anne",
  "lastName" : "Kretchmar",
  "email" : "annek@noanswer.org",
  "tags" : [ "long-term", "vip" ],
  "addresses" : [ {
    "id" : 16,
    "street" : "1289 Lombard",
    "city" : "Canehill",
    "state" : "Arkansas",
    "zip" : "72717",
    "type" : "SHIPPING"
  }, ... ]
}
```

Pattern: Materialize Aggregate Views

Materialize Views in the Source DB



Pattern: Ensuring Data Quality

Detecting Missing or Wrong Data

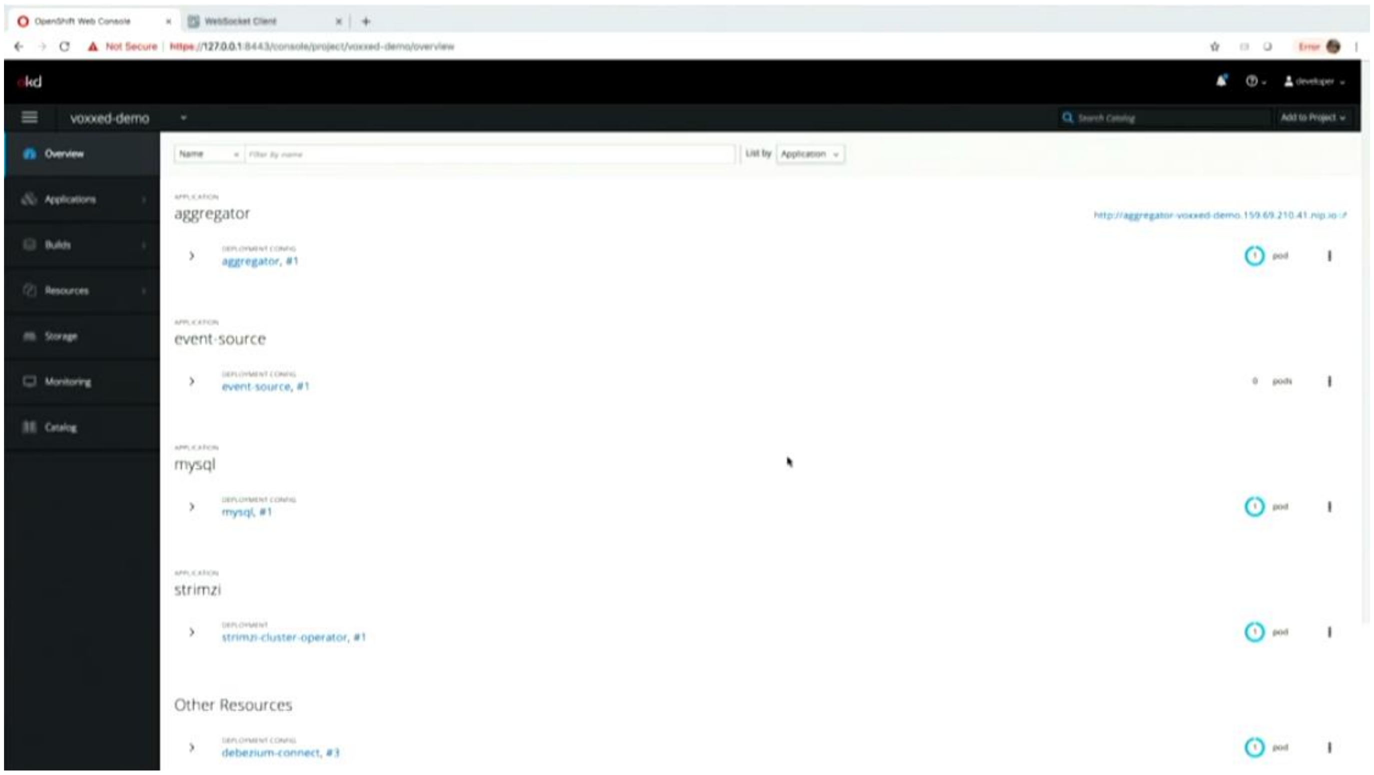
- Constantly **compare record counts** on source and sink side
  - Raise alert if threshold is reached
- Compare every n-th record **field by field**
  - E.g. have all records compared within one week

Pattern: Leverage the Powers of SMTs

Single Message Transformations

- **Aggregate** sharded tables to single topic
- **Keep compatibility** with existing consumers
- **Format conversions**, e.g. for dates
- Ensure compatibility with sink connectors
  - Extracting "after" state only
  - Expand MongoDB's JSON structures

Demo


OpenShift Web Console | WebSocket Client

Not Secure | https://127.0.0.1:8443/console/project/voxxed-demo/overview

okd | developer

voxxed-demo | Search Catalog | Add to Project

Overview | Applications | Builds | Resources | Storage | Monitoring | Catalog

Name | Filter by name | List by Application

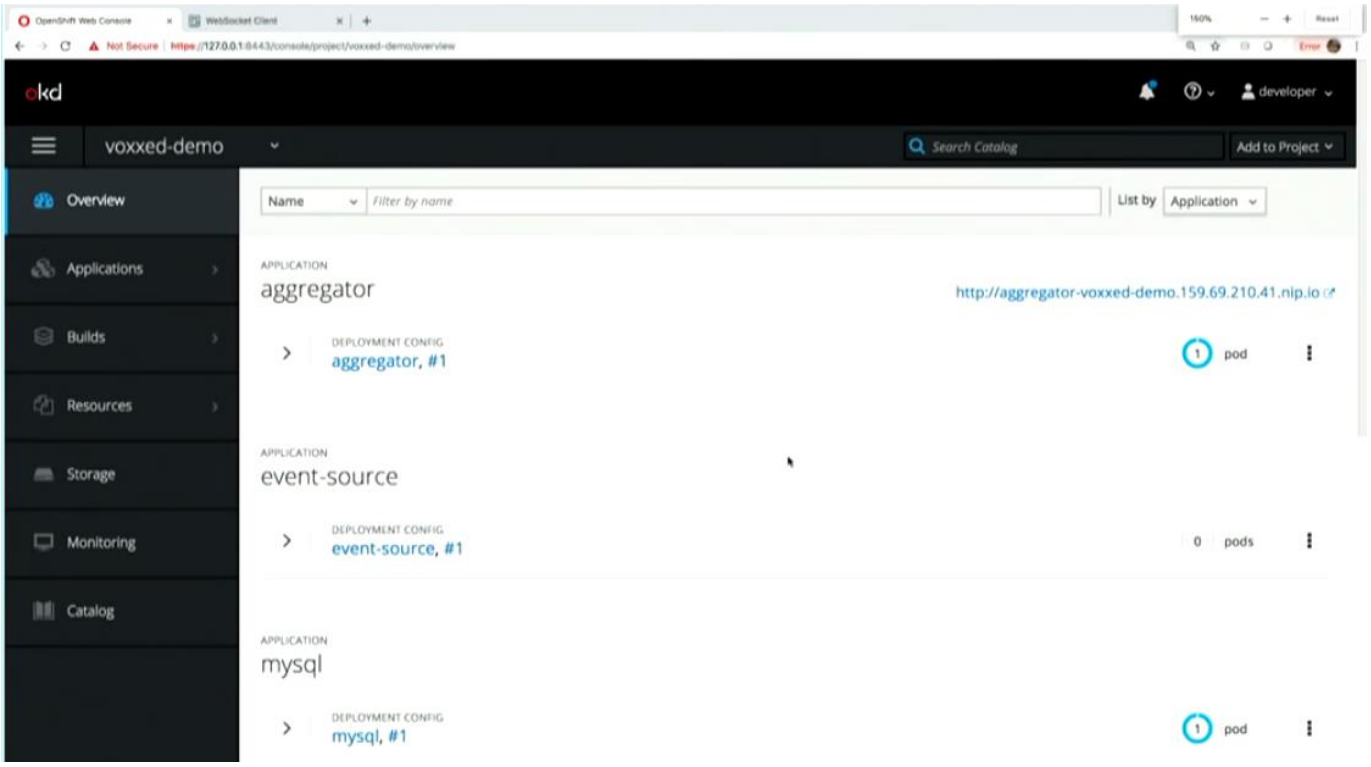
APPLICATION
aggregator
http://aggregator-voxxed-demo.159.69.210.41.nip.io
DEPLOYMENT CONFIG
aggregator, #1 — 1 pod

APPLICATION
event-source
DEPLOYMENT CONFIG
event-source, #1 — 0 pods

APPLICATION
mysql
DEPLOYMENT CONFIG
mysql, #1 — 1 pod

APPLICATION
strimzi
DEPLOYMENT
strimzi-cluster-operator, #1 — 1 pod

Other Resources
DEPLOYMENT CONFIG
debezium-connect, #3 — 1 pod


OpenShift Web Console | WebSocket Client | 150% — + Reset

Not Secure | https://127.0.0.1:8443/console/project/voxxed-demo/overview

okd | developer

voxxed-demo | Search Catalog | Add to Project

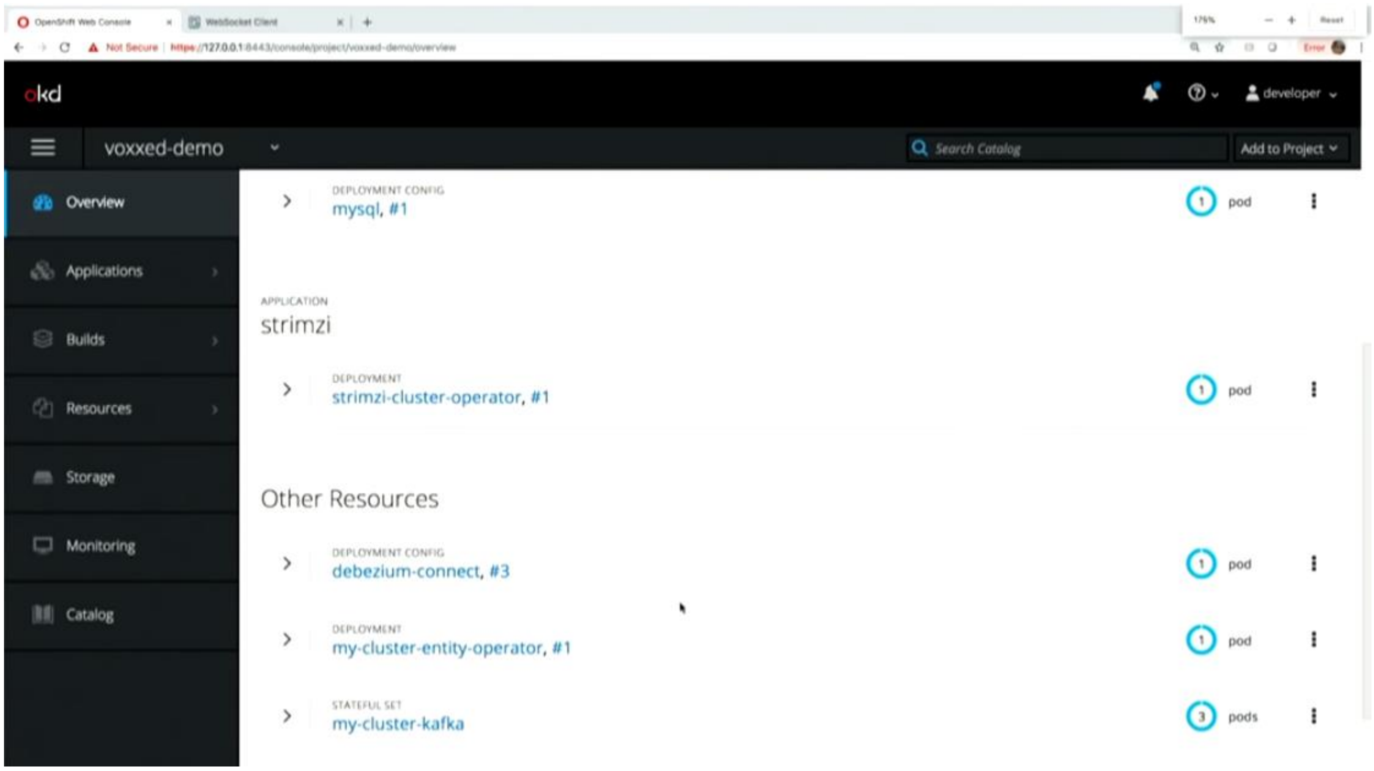
Overview | Applications | Builds | Resources | Storage | Monitoring | Catalog

Name | Filter by name | List by Application

APPLICATION
aggregator
http://aggregator-voxxed-demo.159.69.210.41.nip.io
DEPLOYMENT CONFIG
aggregator, #1 — 1 pod

APPLICATION
event-source
DEPLOYMENT CONFIG
event-source, #1 — 0 pods

APPLICATION
mysql
DEPLOYMENT CONFIG
mysql, #1 — 1 pod



DEPLOYMENT CONFIG
mysql, #1 — 1 pod

APPLICATION
strimzi

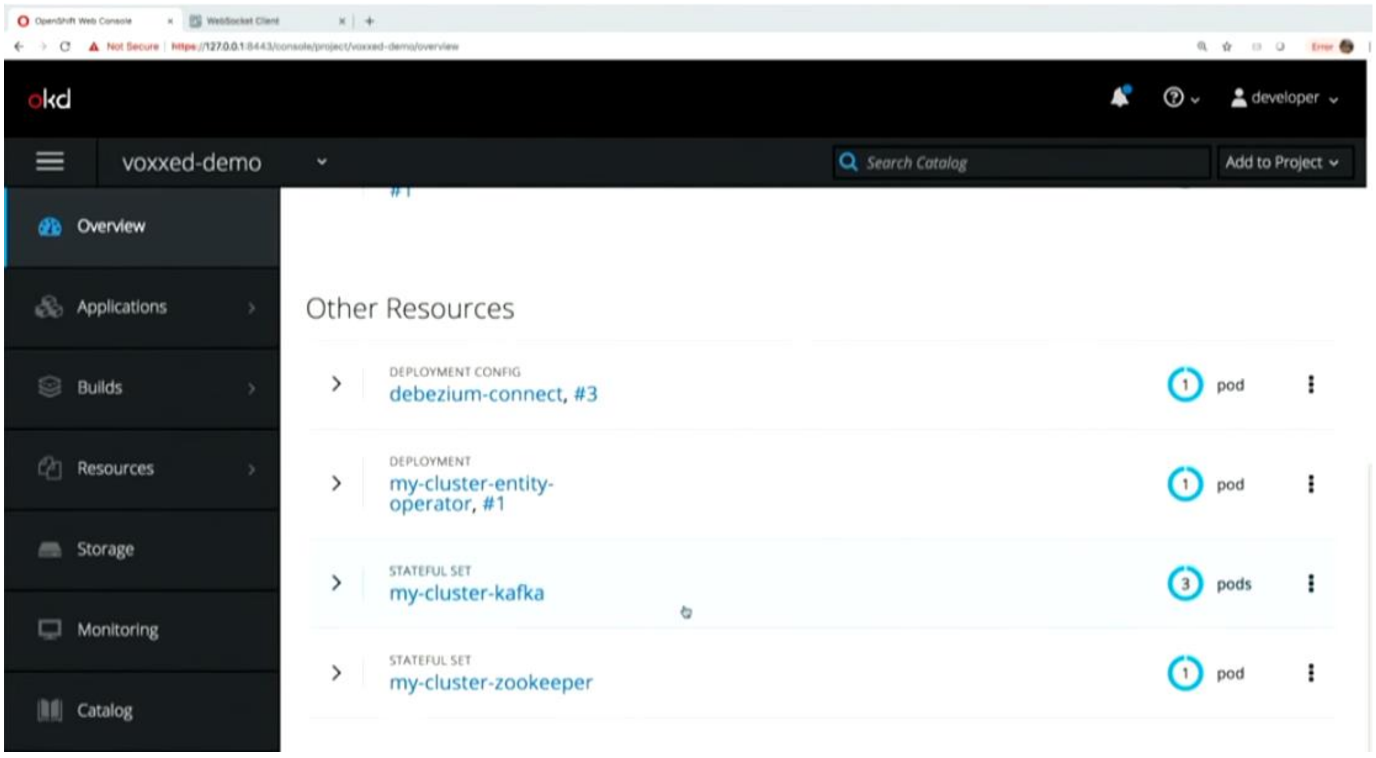
DEPLOYMENT
strimzi-cluster-operator, #1 — 1 pod

Other Resources

DEPLOYMENT CONFIG
debezium-connect, #3 — 1 pod

DEPLOYMENT
my-cluster-entity-operator, #1 — 1 pod

STATEFUL SET
my-cluster-kafka — 3 pods



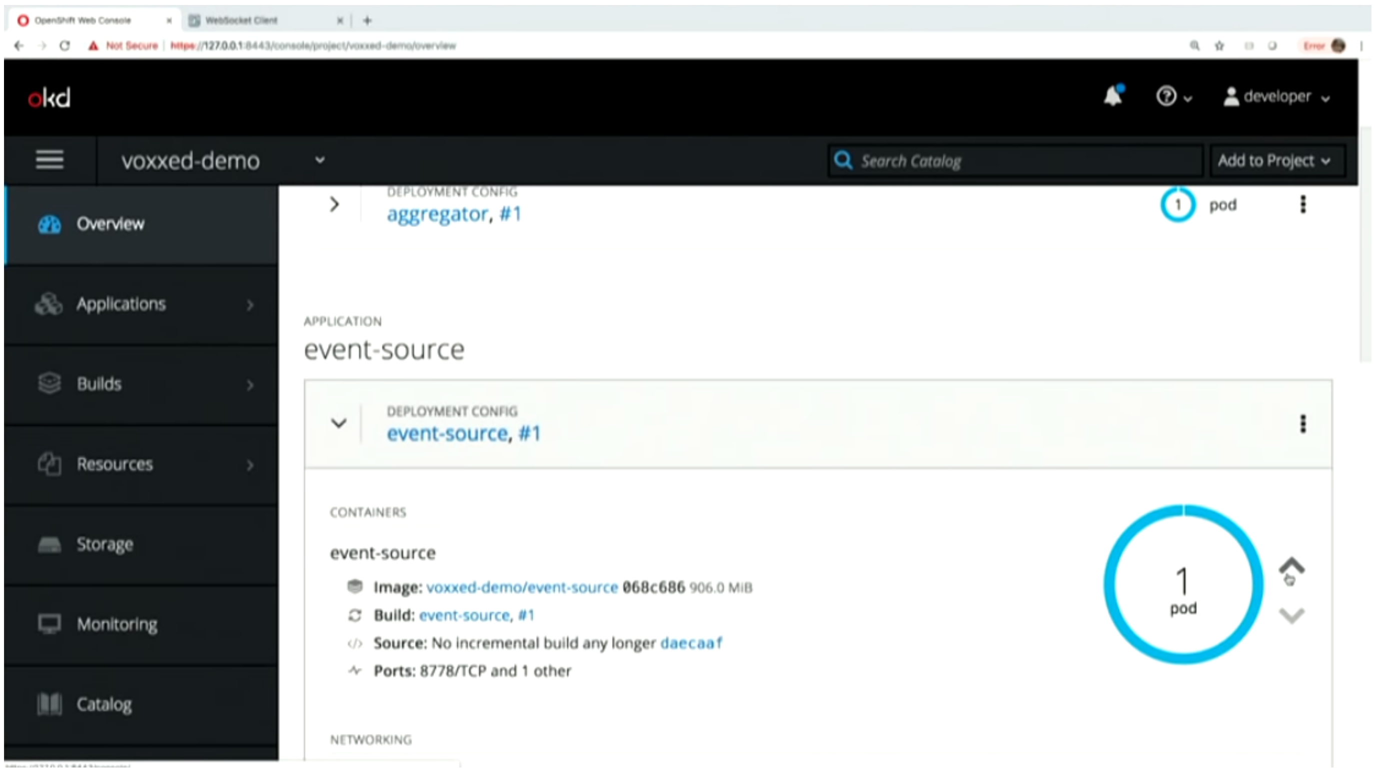
Other Resources

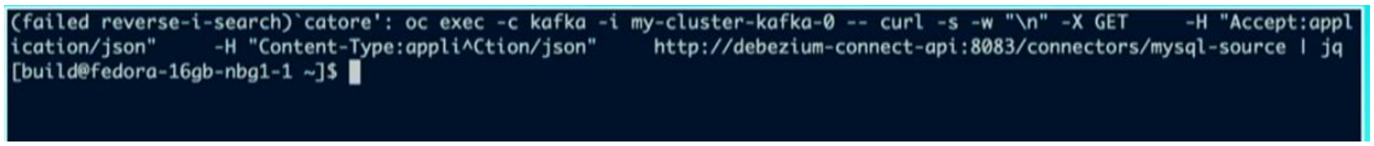
DEPLOYMENT CONFIG
debezium-connect, #3 — 1 pod

DEPLOYMENT
my-cluster-entity-operator, #1 — 1 pod

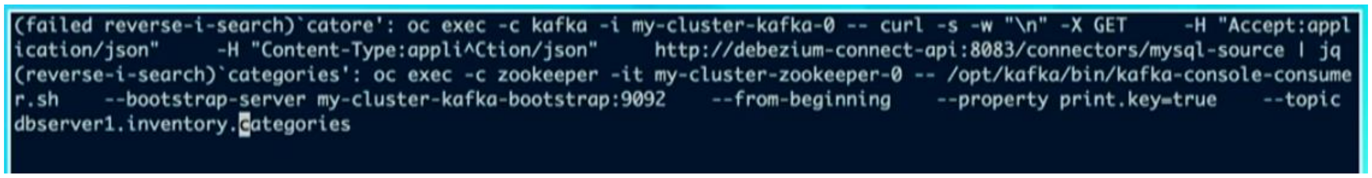
STATEFUL SET
my-cluster-kafka — 3 pods

STATEFUL SET
my-cluster-zookeeper — 1 pod

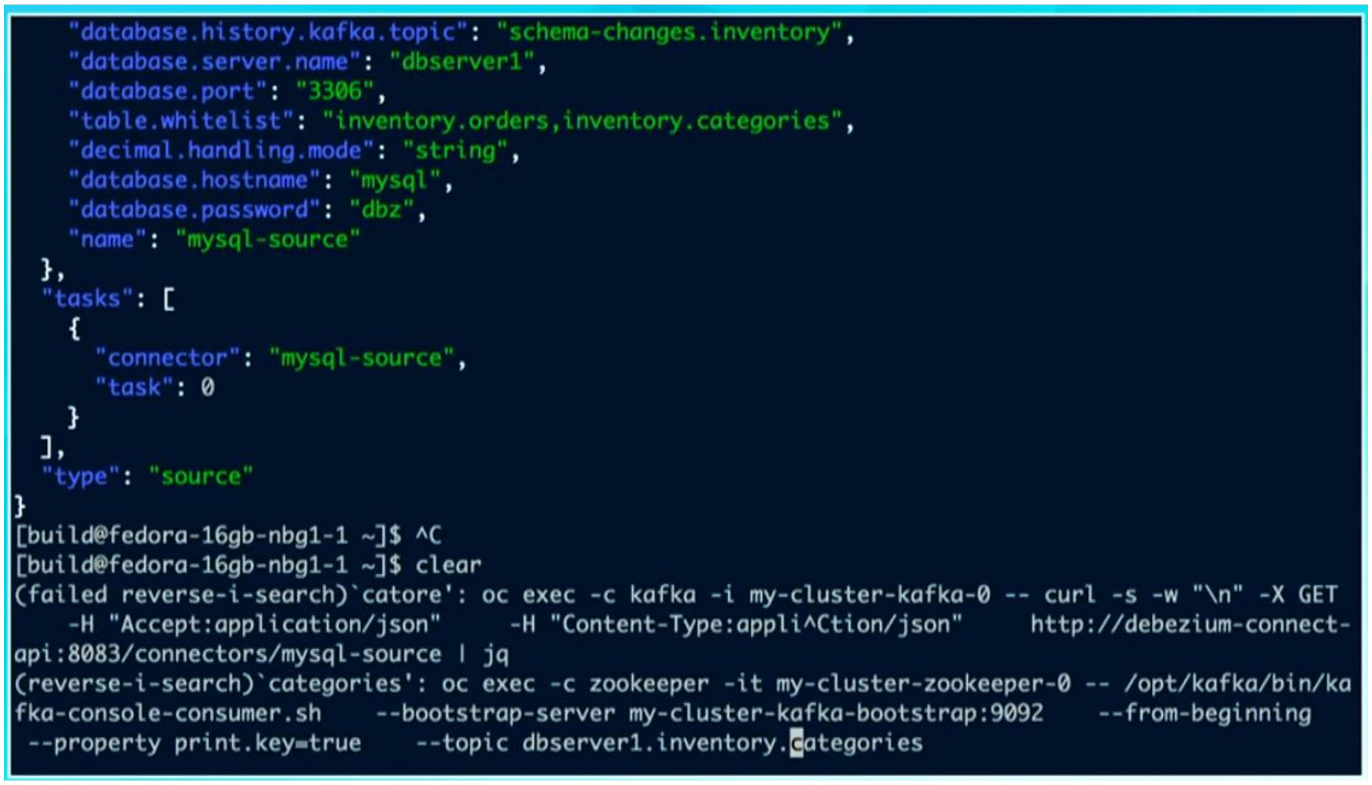




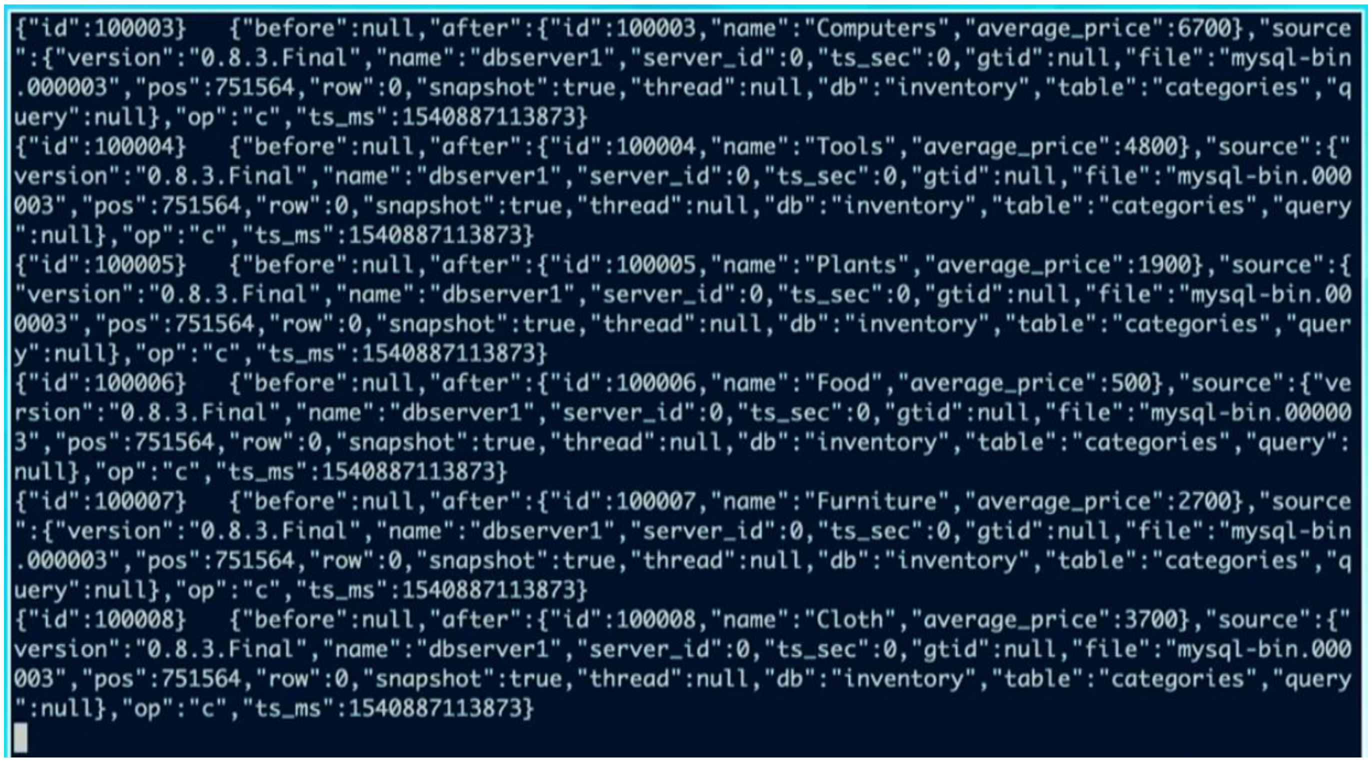
```
(failed reverse-i-search)`catore': oc exec -c kafka -i my-cluster-kafka-0 -- curl -s -w "\n" -X GET    -H "Accept:appl
ication/json"     -H "Content-Type:appli^Ction/json"     http://debezium-connect-api:8083/connectors/mysql-source | jq
[build@fedora-16gb-nbg1-1 ~]$ 
```



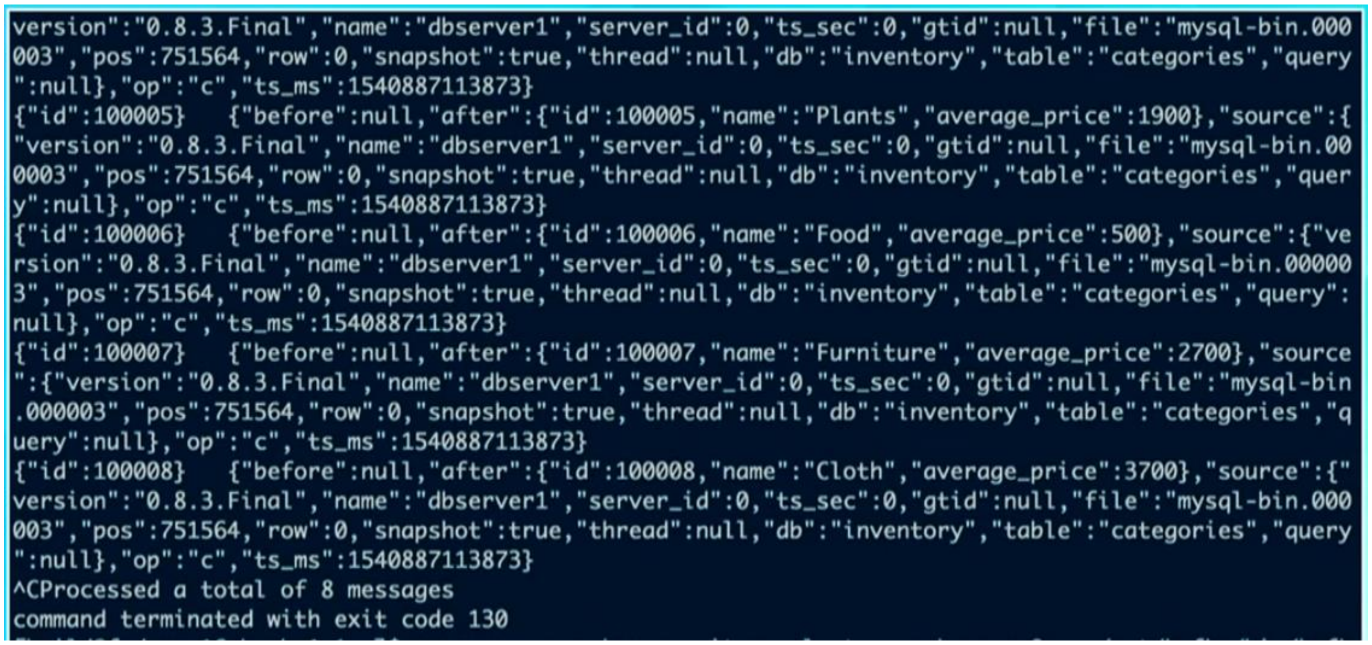
```
(failed reverse-i-search)`catore': oc exec -c kafka -i my-cluster-kafka-0 -- curl -s -w "\n" -X GET    -H "Accept:appl
ication/json"     -H "Content-Type:appli^Ction/json"     http://debezium-connect-api:8083/connectors/mysql-source | jq
(reverse-i-search)`categories': oc exec -c zookeeper -it my-cluster-zookeeper-0 -- /opt/kafka/bin/kafka-console-consume
r.sh    --bootstrap-server my-cluster-kafka-bootstrap:9092    --from-beginning    --property print.key=true    --topic
dbserver1.inventory.categories
```



```
    "database.history.kafka.topic": "schema-changes.inventory",
    "database.server.name": "dbserver1",
    "database.port": "3306",
    "table.whitelist": "inventory.orders,inventory.categories",
    "decimal.handling.mode": "string",
    "database.hostname": "mysql",
    "database.password": "dbz",
    "name": "mysql-source"
  },
  "tasks": [
    {
      "connector": "mysql-source",
      "task": 0
    }
  ],
  "type": "source"
}
[build@fedora-16gb-nbg1-1 ~]$ ^C
[build@fedora-16gb-nbg1-1 ~]$ clear
(failed reverse-i-search)`catore': oc exec -c kafka -i my-cluster-kafka-0 -- curl -s -w "\n" -X GET
   -H "Accept:application/json"     -H "Content-Type:appli^Ction/json"     http://debezium-connect-
api:8083/connectors/mysql-source | jq
(reverse-i-search)`categories': oc exec -c zookeeper -it my-cluster-zookeeper-0 -- /opt/kafka/bin/ka
fka-console-consumer.sh    --bootstrap-server my-cluster-kafka-bootstrap:9092    --from-beginning
 --property print.key=true    --topic dbserver1.inventory.categories
```

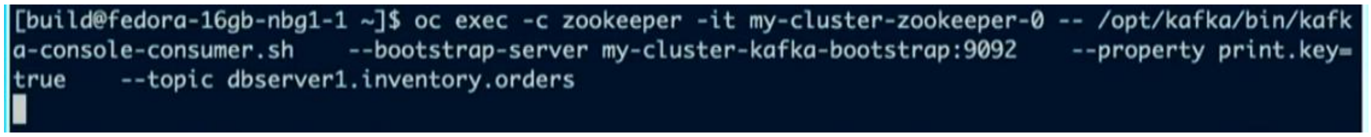


```
{"id":100003}   {"before":null,"after":{"id":100003,"name":"Computers","average_price":6700},"source
":{"version":"0.8.3.Final","name":"dbserver1","server_id":0,"ts_sec":0,"gtid":null,"file":"mysql-bin
.000003","pos":751564,"row":0,"snapshot":true,"thread":null,"db":"inventory","table":"categories","q
uery":null},"op":"c","ts_ms":1540887113873}
{"id":100004}   {"before":null,"after":{"id":100004,"name":"Tools","average_price":4800},"source":{"
version":"0.8.3.Final","name":"dbserver1","server_id":0,"ts_sec":0,"gtid":null,"file":"mysql-bin.000
003","pos":751564,"row":0,"snapshot":true,"thread":null,"db":"inventory","table":"categories","query
":null},"op":"c","ts_ms":1540887113873}
{"id":100005}   {"before":null,"after":{"id":100005,"name":"Plants","average_price":1900},"source":{
"version":"0.8.3.Final","name":"dbserver1","server_id":0,"ts_sec":0,"gtid":null,"file":"mysql-bin.00
0003","pos":751564,"row":0,"snapshot":true,"thread":null,"db":"inventory","table":"categories","quer
y":null},"op":"c","ts_ms":1540887113873}
{"id":100006}   {"before":null,"after":{"id":100006,"name":"Food","average_price":500},"source":{"ve
rsion":"0.8.3.Final","name":"dbserver1","server_id":0,"ts_sec":0,"gtid":null,"file":"mysql-bin.00000
3","pos":751564,"row":0,"snapshot":true,"thread":null,"db":"inventory","table":"categories","query":
null},"op":"c","ts_ms":1540887113873}
{"id":100007}   {"before":null,"after":{"id":100007,"name":"Furniture","average_price":2700},"source
":{"version":"0.8.3.Final","name":"dbserver1","server_id":0,"ts_sec":0,"gtid":null,"file":"mysql-bin
.000003","pos":751564,"row":0,"snapshot":true,"thread":null,"db":"inventory","table":"categories","q
uery":null},"op":"c","ts_ms":1540887113873}
{"id":100008}   {"before":null,"after":{"id":100008,"name":"Cloth","average_price":3700},"source":{"
version":"0.8.3.Final","name":"dbserver1","server_id":0,"ts_sec":0,"gtid":null,"file":"mysql-bin.000
003","pos":751564,"row":0,"snapshot":true,"thread":null,"db":"inventory","table":"categories","query
":null},"op":"c","ts_ms":1540887113873}
```

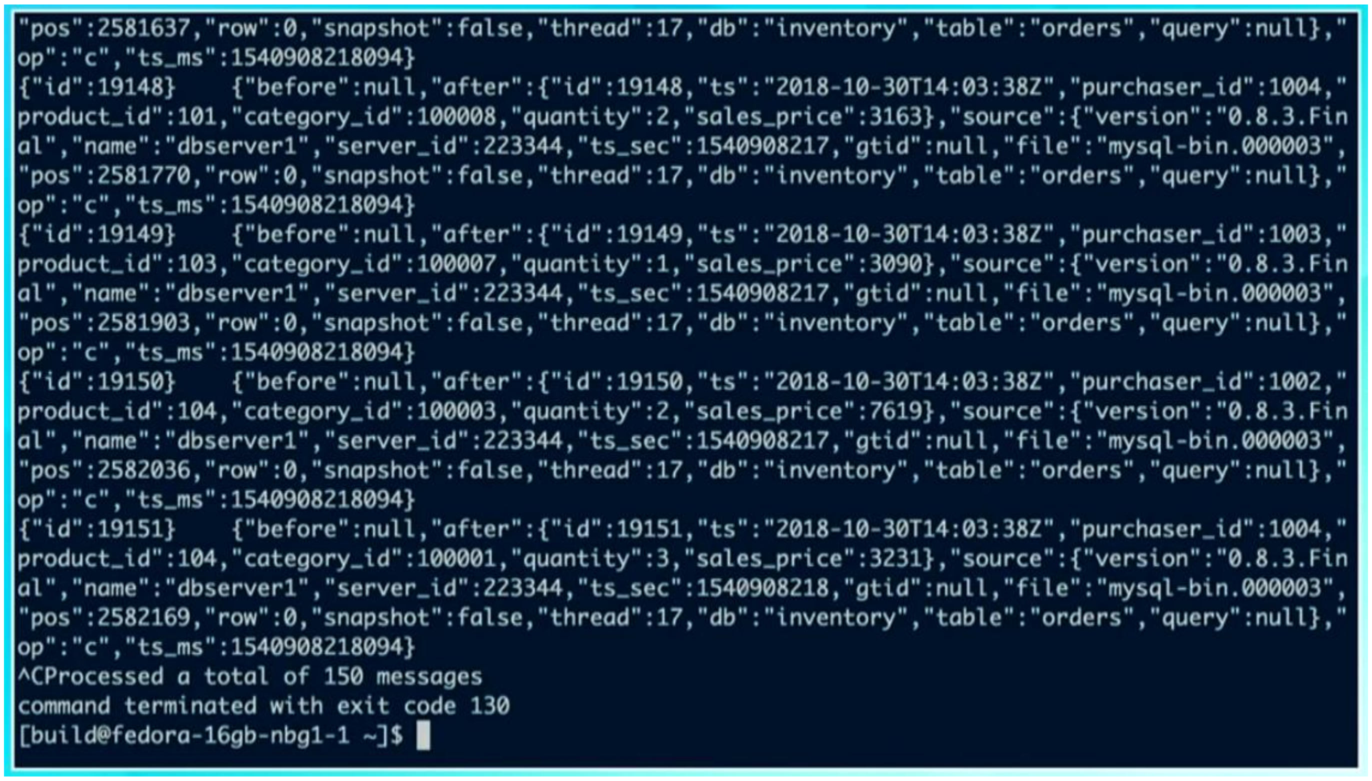


```
version":"0.8.3.Final","name":"dbserver1","server_id":0,"ts_sec":0,"gtid":null,"file":"mysql-bin.000
003","pos":751564,"row":0,"snapshot":true,"thread":null,"db":"inventory","table":"categories","query
":null},"op":"c","ts_ms":1540887113873}
{"id":100005}   {"before":null,"after":{"id":100005,"name":"Plants","average_price":1900},"source":{
"version":"0.8.3.Final","name":"dbserver1","server_id":0,"ts_sec":0,"gtid":null,"file":"mysql-bin.00
0003","pos":751564,"row":0,"snapshot":true,"thread":null,"db":"inventory","table":"categories","quer
y":null},"op":"c","ts_ms":1540887113873}
{"id":100006}   {"before":null,"after":{"id":100006,"name":"Food","average_price":500},"source":{"ve
rsion":"0.8.3.Final","name":"dbserver1","server_id":0,"ts_sec":0,"gtid":null,"file":"mysql-bin.00000
3","pos":751564,"row":0,"snapshot":true,"thread":null,"db":"inventory","table":"categories","query":
null},"op":"c","ts_ms":1540887113873}
{"id":100007}   {"before":null,"after":{"id":100007,"name":"Furniture","average_price":2700},"source
":{"version":"0.8.3.Final","name":"dbserver1","server_id":0,"ts_sec":0,"gtid":null,"file":"mysql-bin
.000003","pos":751564,"row":0,"snapshot":true,"thread":null,"db":"inventory","table":"categories","q
uery":null},"op":"c","ts_ms":1540887113873}
{"id":100008}   {"before":null,"after":{"id":100008,"name":"Cloth","average_price":3700},"source":{"
version":"0.8.3.Final","name":"dbserver1","server_id":0,"ts_sec":0,"gtid":null,"file":"mysql-bin.000
003","pos":751564,"row":0,"snapshot":true,"thread":null,"db":"inventory","table":"categories","query
":null},"op":"c","ts_ms":1540887113873}
```

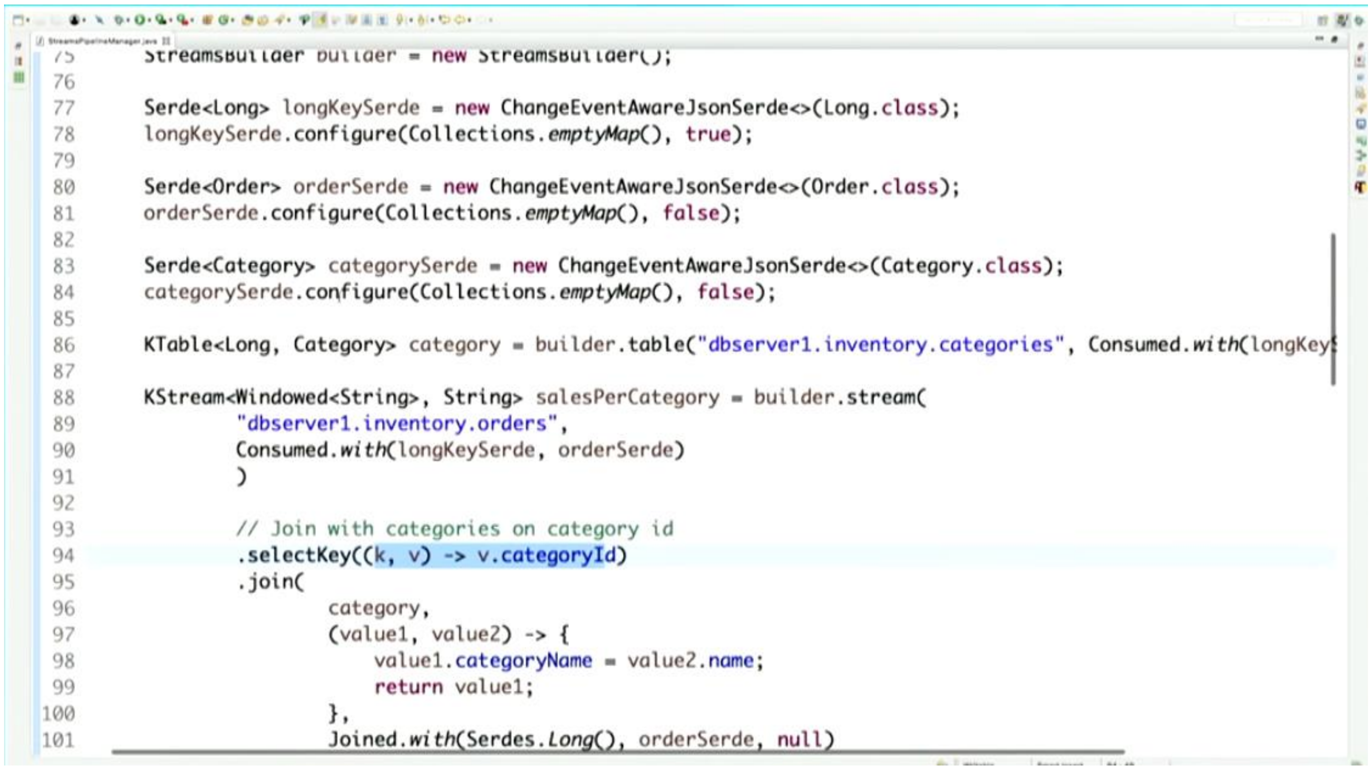
```
^CProcessed a total of 8 messages
command terminated with exit code 130
```



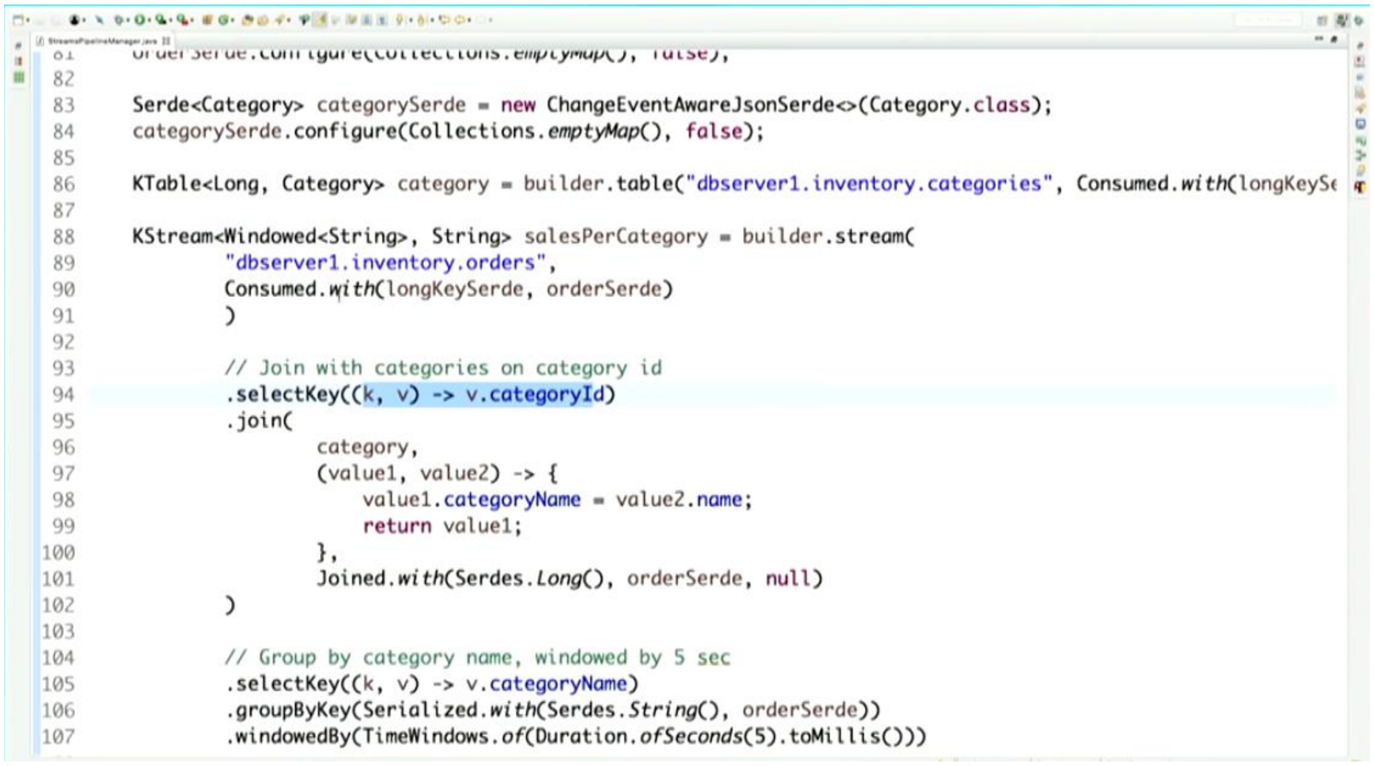
```
[build@fedora-16gb-nbg1-1 ~]$ oc exec -c zookeeper -it my-cluster-zookeeper-0 -- /opt/kafka/bin/kafk
a-console-consumer.sh    --bootstrap-server my-cluster-kafka-bootstrap:9092    --property print.key=
true    --topic dbserver1.inventory.orders
```



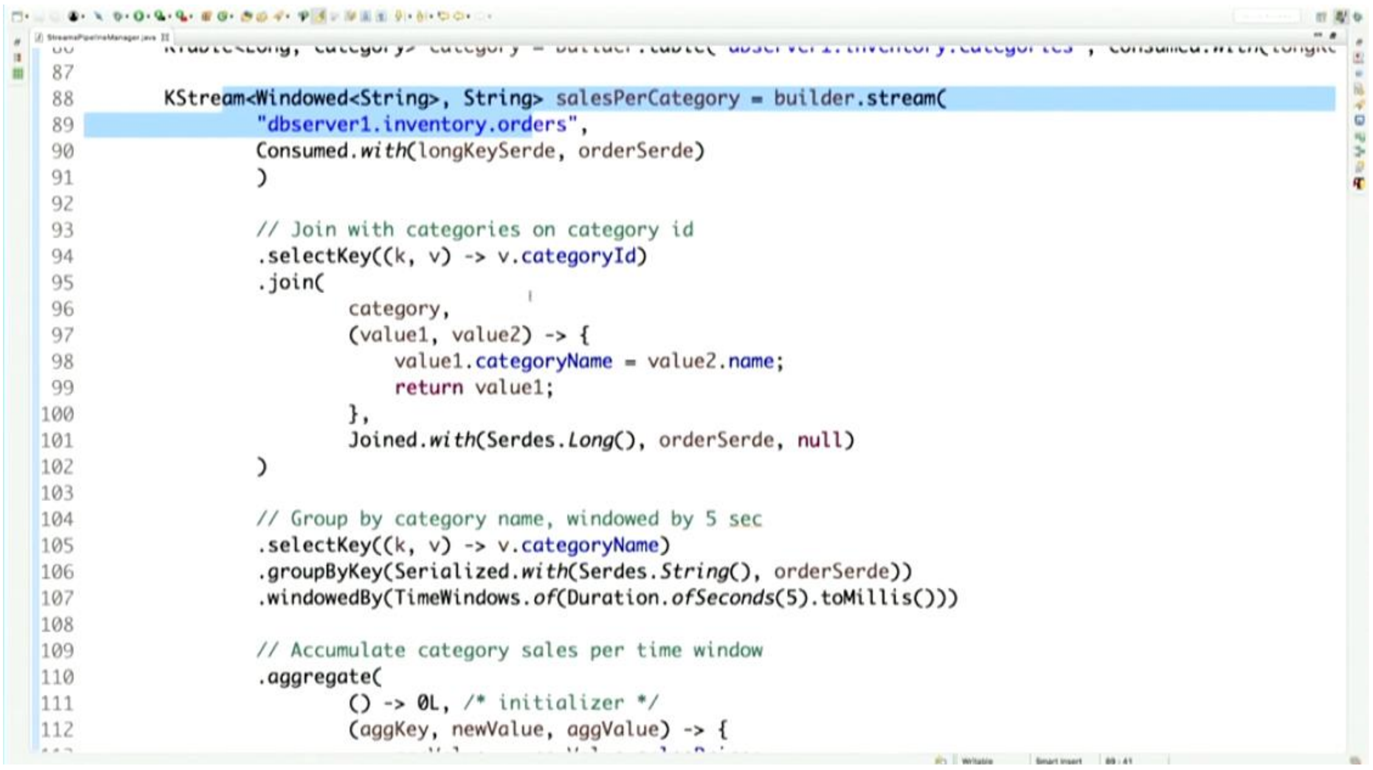
```
"pos":2581637,"row":0,"snapshot":false,"thread":17,"db":"inventory","table":"orders","query":null},"
op":"c","ts_ms":1540908218094}
{"id":19148}    {"before":null,"after":{"id":19148,"ts":"2018-10-30T14:03:38Z","purchaser_id":1004,"
product_id":101,"category_id":100008,"quantity":2,"sales_price":3163},"source":{"version":"0.8.3.Fin
al","name":"dbserver1","server_id":223344,"ts_sec":1540908217,"gtid":null,"file":"mysql-bin.000003",
"pos":2581770,"row":0,"snapshot":false,"thread":17,"db":"inventory","table":"orders","query":null},"
op":"c","ts_ms":1540908218094}
{"id":19149}    {"before":null,"after":{"id":19149,"ts":"2018-10-30T14:03:38Z","purchaser_id":1003,"
product_id":103,"category_id":100007,"quantity":1,"sales_price":3090},"source":{"version":"0.8.3.Fin
al","name":"dbserver1","server_id":223344,"ts_sec":1540908217,"gtid":null,"file":"mysql-bin.000003",
"pos":2581903,"row":0,"snapshot":false,"thread":17,"db":"inventory","table":"orders","query":null},"
op":"c","ts_ms":1540908218094}
{"id":19150}    {"before":null,"after":{"id":19150,"ts":"2018-10-30T14:03:38Z","purchaser_id":1002,"
product_id":104,"category_id":100003,"quantity":2,"sales_price":7619},"source":{"version":"0.8.3.Fin
al","name":"dbserver1","server_id":223344,"ts_sec":1540908217,"gtid":null,"file":"mysql-bin.000003",
"pos":2582036,"row":0,"snapshot":false,"thread":17,"db":"inventory","table":"orders","query":null},"
op":"c","ts_ms":1540908218094}
{"id":19151}    {"before":null,"after":{"id":19151,"ts":"2018-10-30T14:03:38Z","purchaser_id":1004,"
product_id":104,"category_id":100001,"quantity":3,"sales_price":3231},"source":{"version":"0.8.3.Fin
al","name":"dbserver1","server_id":223344,"ts_sec":1540908218,"gtid":null,"file":"mysql-bin.000003",
"pos":2582169,"row":0,"snapshot":false,"thread":17,"db":"inventory","table":"orders","query":null},"
op":"c","ts_ms":1540908218094}
^CProcessed a total of 150 messages
command terminated with exit code 130
[build@fedora-16gb-nbg1-1 ~]$ 
```



```java
    StreamsBuilder builder = new StreamsBuilder();

    Serde<Long> longKeySerde = new ChangeEventAwareJsonSerde<>(Long.class);
    longKeySerde.configure(Collections.emptyMap(), true);

    Serde<Order> orderSerde = new ChangeEventAwareJsonSerde<>(Order.class);
    orderSerde.configure(Collections.emptyMap(), false);

    Serde<Category> categorySerde = new ChangeEventAwareJsonSerde<>(Category.class);
    categorySerde.configure(Collections.emptyMap(), false);

    KTable<Long, Category> category = builder.table("dbserver1.inventory.categories", Consumed.with(longKey

    KStream<Windowed<String>, String> salesPerCategory = builder.stream(
            "dbserver1.inventory.orders",
            Consumed.with(longKeySerde, orderSerde)
            )

            // Join with categories on category id
            .selectKey((k, v) -> v.categoryId)
            .join(
                    category,
                    (value1, value2) -> {
                        value1.categoryName = value2.name;
                        return value1;
                    },
                    Joined.with(Serdes.Long(), orderSerde, null)
```

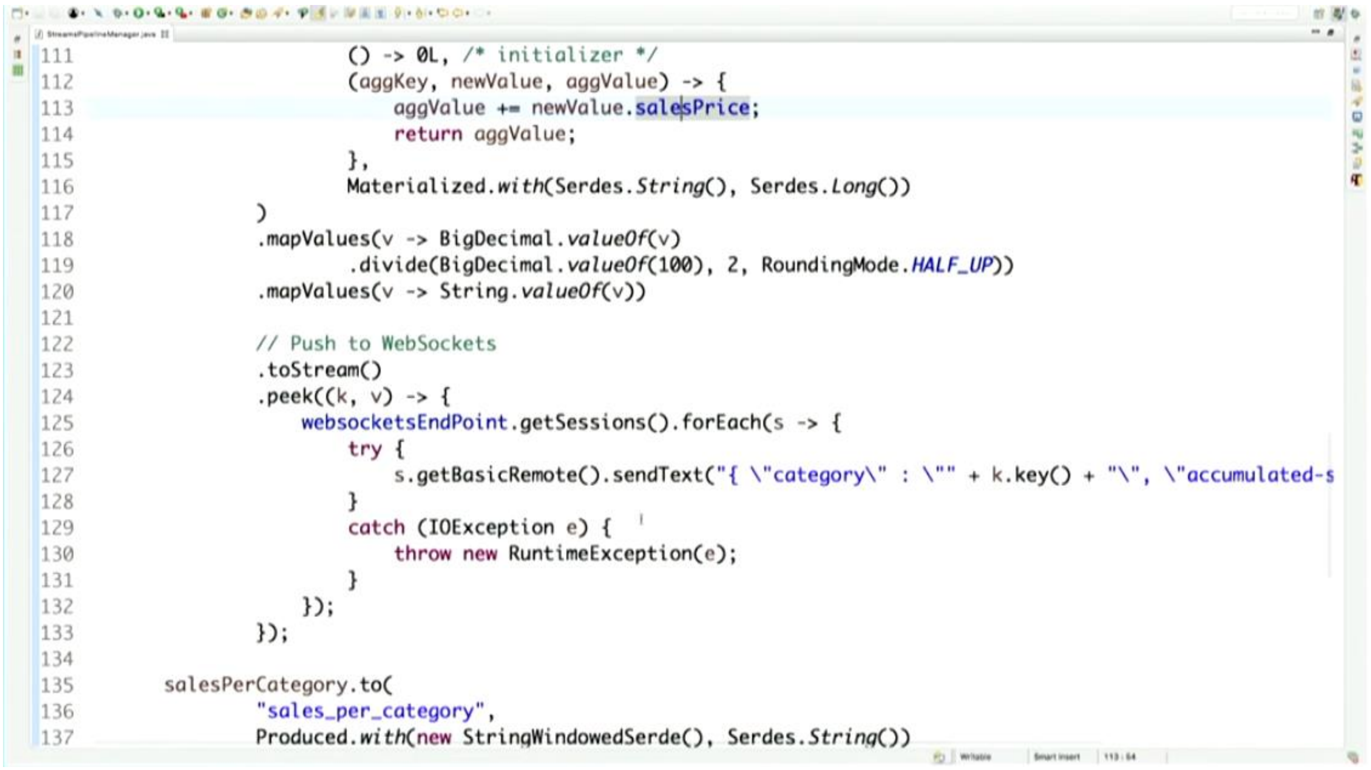


```java
    orderSerde.configure(Collections.emptyMap(), false);

    Serde<Category> categorySerde = new ChangeEventAwareJsonSerde<>(Category.class);
    categorySerde.configure(Collections.emptyMap(), false);

    KTable<Long, Category> category = builder.table("dbserver1.inventory.categories", Consumed.with(longKeySe

    KStream<Windowed<String>, String> salesPerCategory = builder.stream(
            "dbserver1.inventory.orders",
            Consumed.with(longKeySerde, orderSerde)
            )

            // Join with categories on category id
            .selectKey((k, v) -> v.categoryId)
            .join(
                    category,
                    (value1, value2) -> {
                        value1.categoryName = value2.name;
                        return value1;
                    },
                    Joined.with(Serdes.Long(), orderSerde, null)
            )

            // Group by category name, windowed by 5 sec
            .selectKey((k, v) -> v.categoryName)
            .groupByKey(Serialized.with(Serdes.String(), orderSerde))
            .windowedBy(TimeWindows.of(Duration.ofSeconds(5).toMillis()))
```

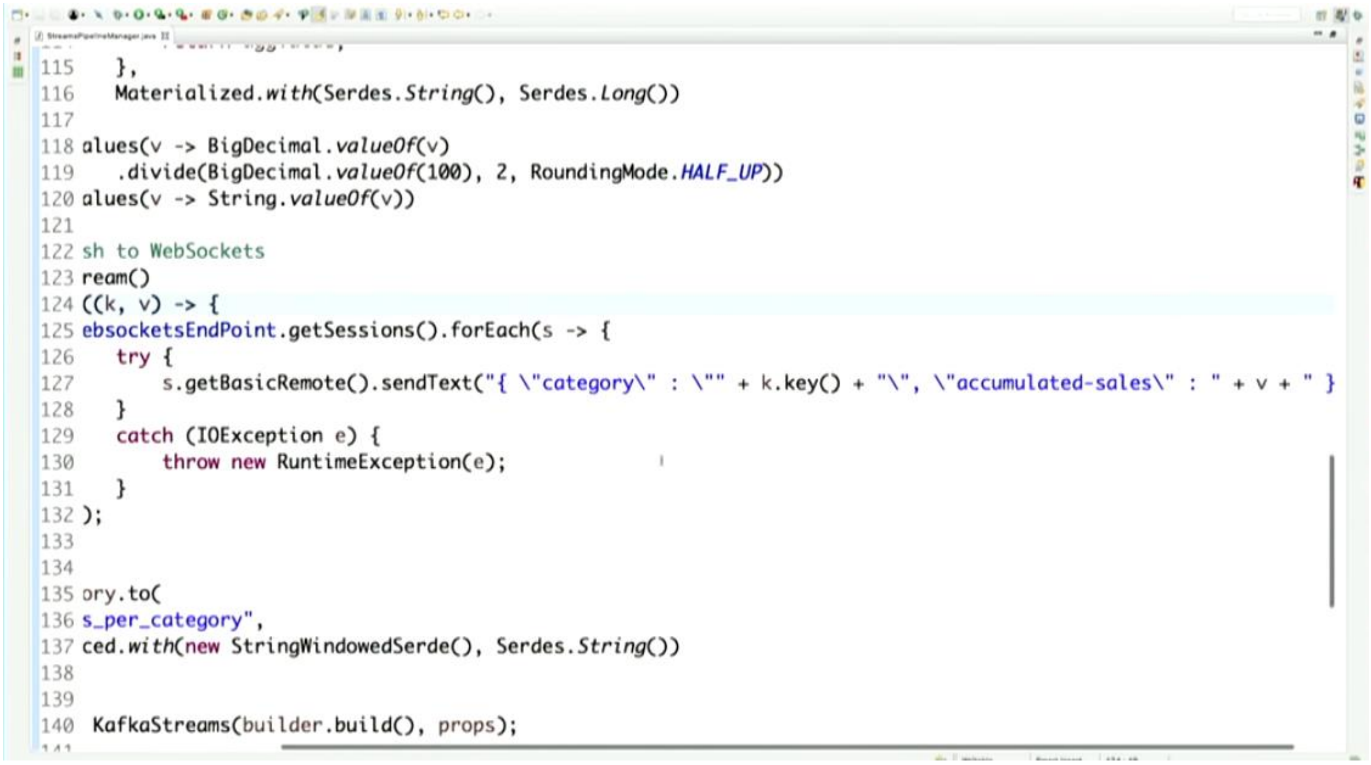


```java
    KStream<Windowed<String>, String> salesPerCategory = builder.stream(
            "dbserver1.inventory.orders",
            Consumed.with(longKeySerde, orderSerde)
            )

            // Join with categories on category id
            .selectKey((k, v) -> v.categoryId)
            .join(
                    category,
                    (value1, value2) -> {
                        value1.categoryName = value2.name;
                        return value1;
                    },
                    Joined.with(Serdes.Long(), orderSerde, null)
            )

            // Group by category name, windowed by 5 sec
            .selectKey((k, v) -> v.categoryName)
            .groupByKey(Serialized.with(Serdes.String(), orderSerde))
            .windowedBy(TimeWindows.of(Duration.ofSeconds(5).toMillis()))
```

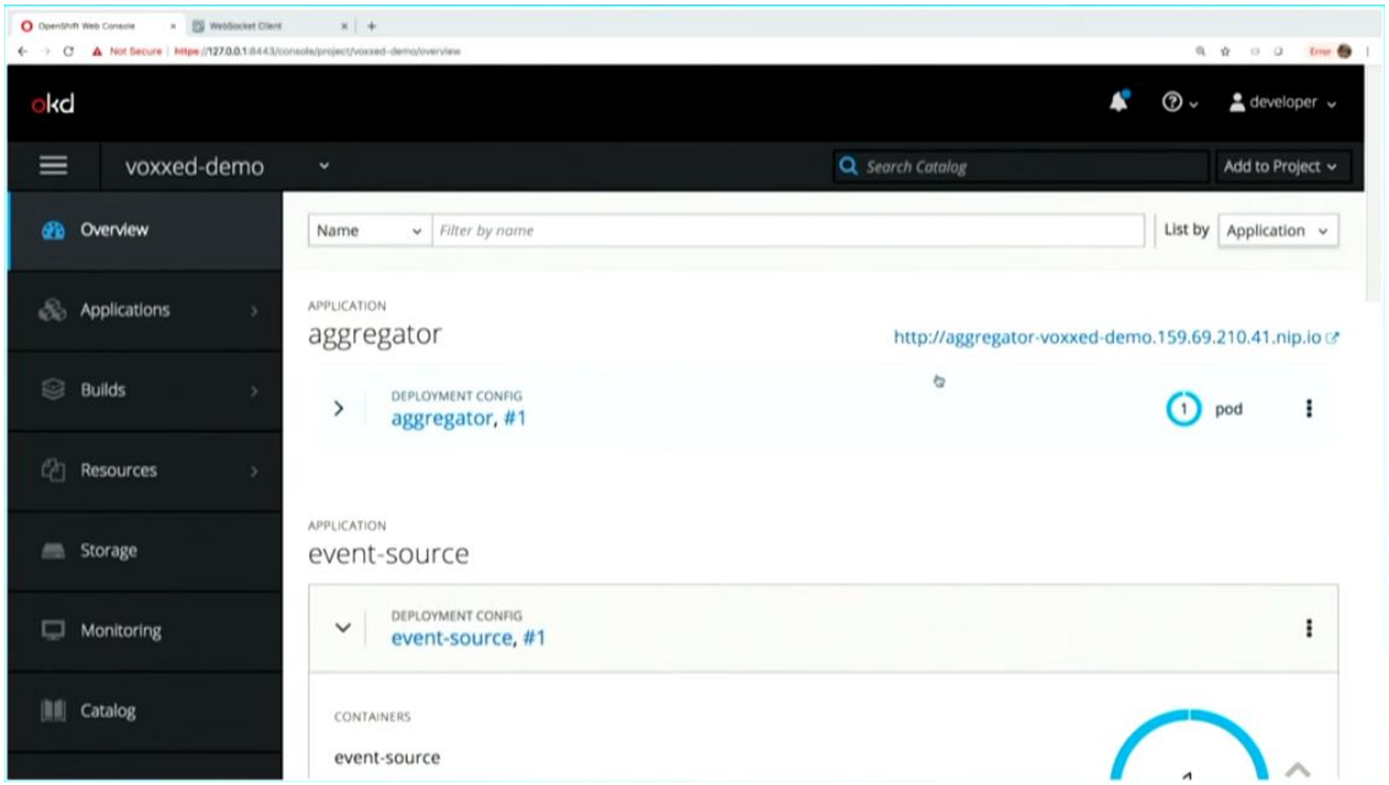
```java

            // Accumulate category sales per time window
            .aggregate(
                    () -> 0L, /* initializer */
                    (aggKey, newValue, aggValue) -> {
```



```
                        () -> 0L, /* initializer */
                        (aggKey, newValue, aggValue) -> {
                            aggValue += newValue.salesPrice;
                            return aggValue;
                        },
                        Materialized.with(Serdes.String(), Serdes.Long())
                )
                .mapValues(v -> BigDecimal.valueOf(v)
                        .divide(BigDecimal.valueOf(100), 2, RoundingMode.HALF_UP))
                .mapValues(v -> String.valueOf(v))

                // Push to WebSockets
                .toStream()
                .peek((k, v) -> {
                    websocketsEndPoint.getSessions().forEach(s -> {
                        try {
                            s.getBasicRemote().sendText("{ \"category\" : \"" + k.key() + "\", \"accumulated-s
                        }
                        catch (IOException e) {
                            throw new RuntimeException(e);
                        }
                    });
                });

        salesPerCategory.to(
                "sales_per_category",
                Produced.with(new StringWindowedSerde(), Serdes.String())
```



```
    },
    Materialized.with(Serdes.String(), Serdes.Long())

alues(v -> BigDecimal.valueOf(v)
    .divide(BigDecimal.valueOf(100), 2, RoundingMode.HALF_UP))
alues(v -> String.valueOf(v))

sh to WebSockets
ream()
((k, v) -> {
ebsocketsEndPoint.getSessions().forEach(s -> {
    try {
        s.getBasicRemote().sendText("{ \"category\" : \"" + k.key() + "\", \"accumulated-sales\" : " + v + " }
    }
    catch (IOException e) {
        throw new RuntimeException(e);
    }
);


ory.to(
s_per_category",
ced.with(new StringWindowedSerde(), Serdes.String())


 KafkaStreams(builder.build(), props);
```



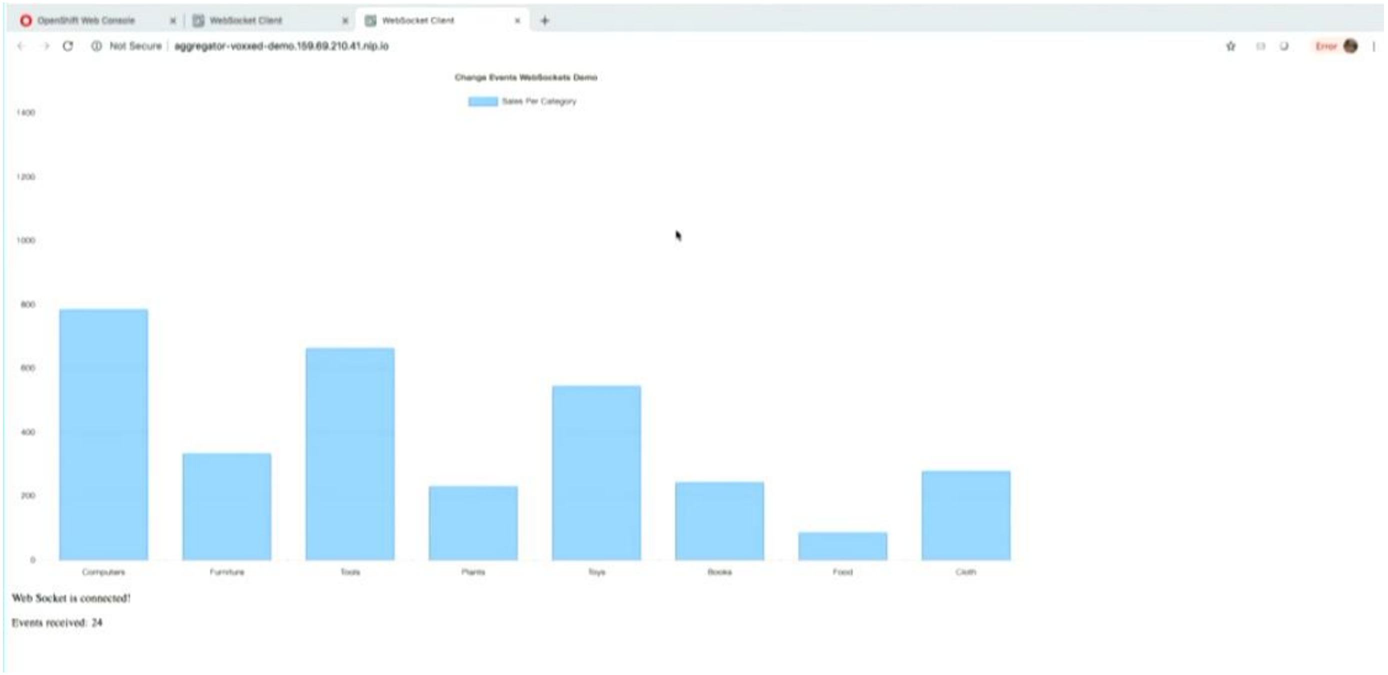
APPLICATION
aggregator
http://aggregator-voxxed-demo.159.69.210.41.nip.io

DEPLOYMENT CONFIG
aggregator, #1
1 pod

APPLICATION
event-source

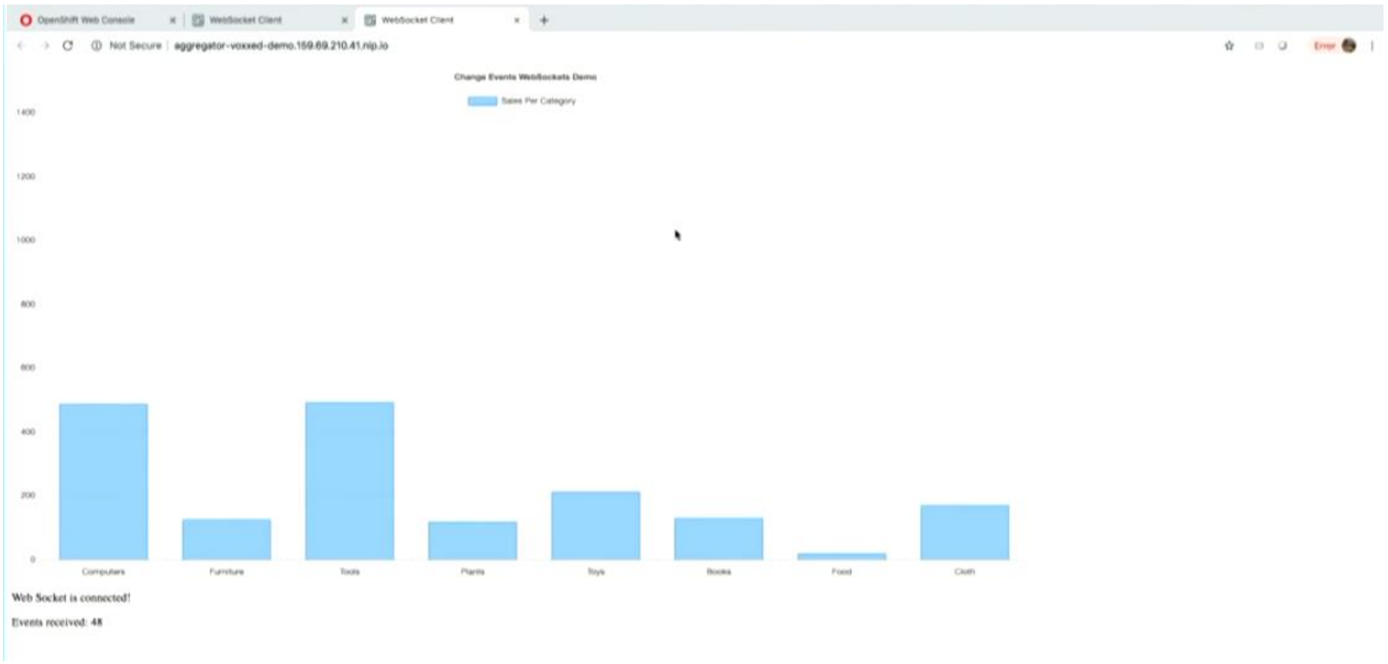
DEPLOYMENT CONFIG
event-source, #1

CONTAINERS
event-source



Web Socket is connected!

Events received: 24



Web Socket is connected!

Events received: 48

Running on Kubernetes

AMQ Streams: Enterprise Distribution of Apache Kafka

- Provides
  - **Container images** for Apache Kafka, Connect, Zookeeper and MirrorMaker
  - **Operators** for managing/configuring Apache Kafka clusters, topics and users
  - Kafka Consumer, Producer and Admin clients, Kafka Streams
- Supported by Red Hat
- Upstream Community: **Strimzi**

Debezium

Current Status

- Current version: 0.8/0.9 (based on **Kafka 2.0**)
  - **Snapshotting**, Filtering etc.
  - Comprehensive **type support** (PostGIS etc.)
  - Common event format as far as possible
  - Usable on **Amazon RDS**
- Production deployments at multiple companies (e.g. WePay, BlaBlaCar etc.)
- Very **active community**
- Everything is **open source** (Apache License v2)

Outlook

- **Debezium 0.9**
  - Expand Support for Oracle and SQL Server
- **Debezium 0.x**
  - Reactive Streams support
  - Infinispan as a sink
  - Installation via OpenShift service catalogue
- **Debezium 1.x**
  - Event aggregation, declarative CQRS support
  - Roadmap: http://debezium.io/docs/roadmap/

Summary

- Use **CDC** to Propagate Data Between Services
- **Debezium** brings CDC for a growing number of databases
- Transparently set up change data event streams
- **Works reliably** also in case of failures
- **Contributions welcome!**

Resources

- **Website:** http://debezium.io/
- **Source code**, examples, Compose files etc. https://github.com/debezium
- **Discussion group** https://groups.google.com/forum/#!forum/debezium
- **Strimzi** (Kafka on Kubernetes/OpenShift) http://strimzi.io/
- **Latest news:** @debezium

Debezium

Stream changes from your databases.

Debezium is an open source distributed platform for change data capture. Start it up, point it at your databases, and your apps can start responding to all of the inserts, updates, and deletes that other apps commit to your databases. Debezium is durable and fast, so your apps can respond quickly and never miss an event, even when things go wrong.

Do more with your data

Your data is always changing. Debezium lets your apps react every time your data changes, and you don't have to change your apps that modify the data. Debezium continuously monitors your databases and lets any of your applications stream every row level change in the same order they were committed to the