Knative Serverless to architect atomically decomposed components

Maarten Vandeperre
Specialist Solution Architect
Red Hat

Dieter De Moitié
Specialist Solution Architect
Red Hat

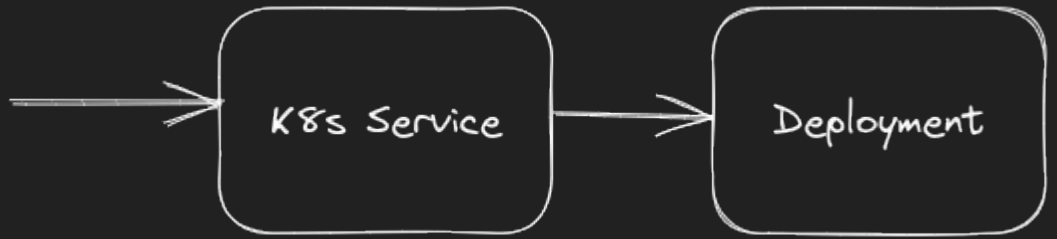
Red Hat Developer

Agenda

- Slides
- Demo

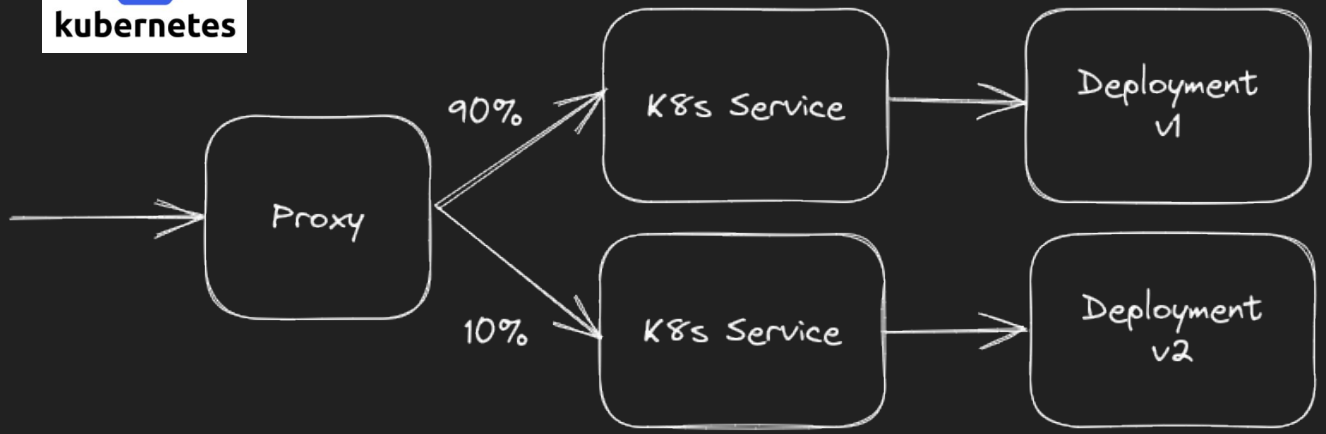

Red Hat
Developer
VOXXED DAYS
BRUSSELS
Knative overview

A simple application on Kubernetes



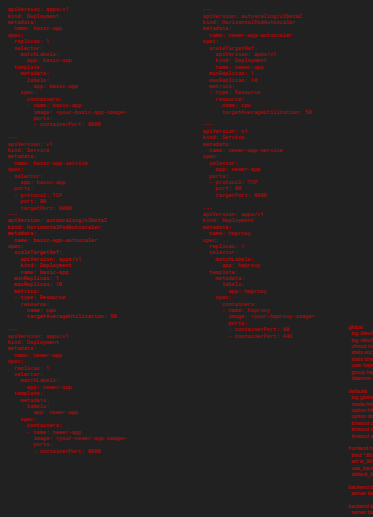

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: basic-app
spec:
  selector:
    matchLabels:
      app: basic-app
  template:
    metadata:
      labels:
        app: basic-app
    spec:
      containers:
      - name: basic-app
        image: <your-basic-app-image>
        ports:
        - containerPort: 8080

---
apiVersion: v1
kind: Service
metadata:
  name: basic-app-service
spec:
  selector:
    app: basic-app
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080

---
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: basic-app-autoscaler
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: basic-app
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      targetAverageUtilization: 50
```

A/B testing a simple application



Need to:

- Create k8s deployments
- Create k8s services
- Create proxy
- Configure proxy
- (Configure k8s ingress)



Serverless computing - What's in a name?

- Simplify deployment of applications
- Don't worry about underlying infrastructure (including kubernetes)
- Only use resources when needed

Picture from: https://pixabay.com/illustrations/rage-angry-frustrated-emotion-1015611/

What is Knative?

Knative is a Kubernetes Native implementation of Serverless technology

CNCF incubating project

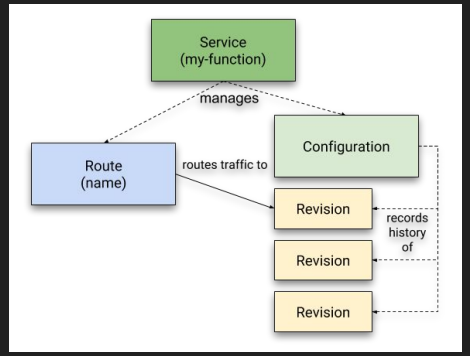
From the Open Source project site - **“Knative is an Open-Source Enterprise-level solution to build Serverless and Event Driven Applications”**

Knative provides a set of additional Objects (through CRD’s) that allow the management of Serverless components.

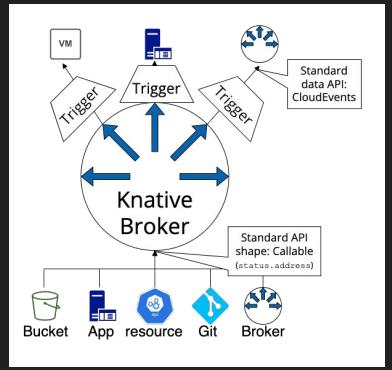
More yaml? => kn CLI

Knative overview

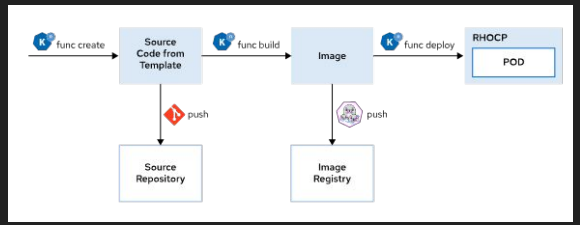
Serving



Eventing



Functions



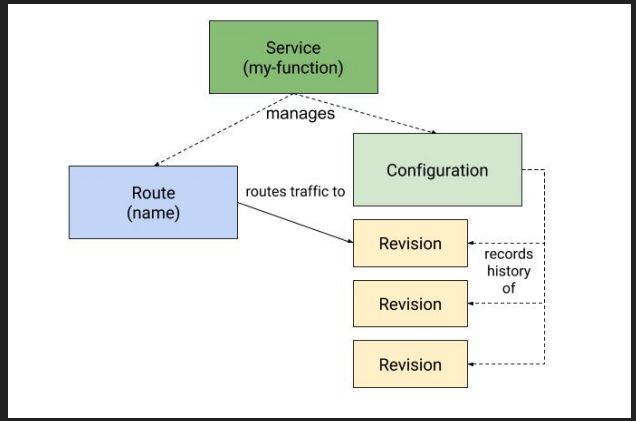
Builds


Red Hat
Developer
VOXXED DAYS
BRUSSELS
Knative Serving

Knative Serving Basics

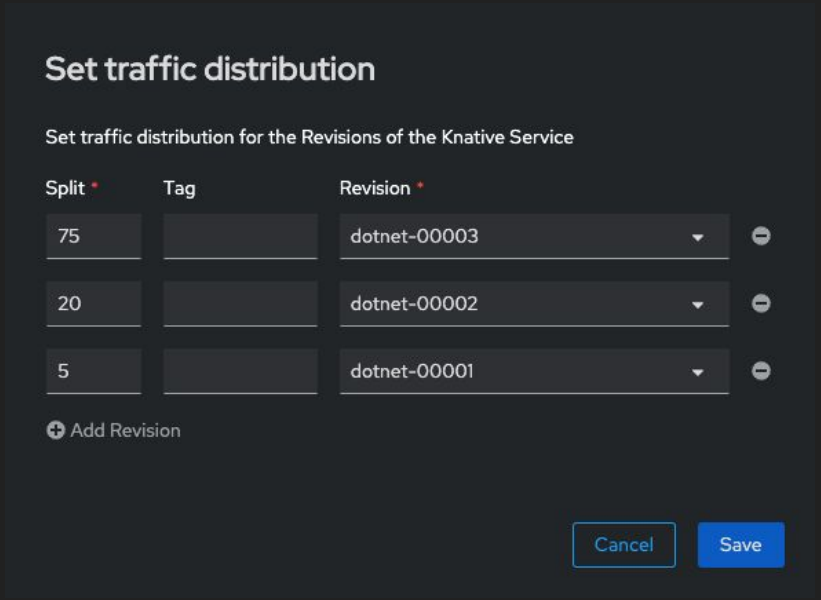
Knative Serving is realised using four CRD components:

1. **Services** - controller object that orchestrates the other components
2. **Configurations** - defines the state of the deployment. Editing the configuration generates a new **Revision**
3. **Revision** - immutable point-in-time snapshot of configuration
4. **Routes** - maps a network endpoint to one or more **Revisions**



Picture from https://knative.dev/docs/concepts/#what-is-knative

Revisions



A new **Revision** is created when the configuration of a knative Serving Application is changed

Revisions are immutable

Routes allow for changing the distribution of traffic arriving to push traffic to multiple **Revisions**

Allows for Blue/Green deployments, A/B testing, canary release

Revisions can have their own entry point

Knative Serving Autoscaling

Knative Serving provides auto-scaling to match incoming demand

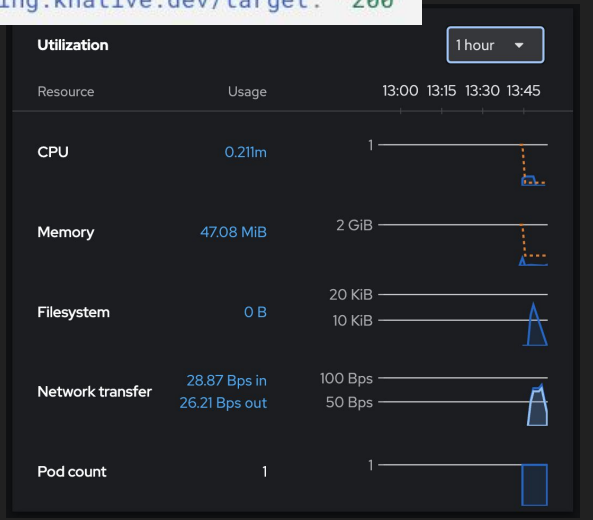
Scaling based on HTTP requests!

Scale-to-zero (can be disabled)
=> More efficient resource consumption

Configure scaling:

- **concurrency** (how many requests can be simultaneously processed by each replica) **, or**
- **requests-per-second** (per replica)
- configured per **Revision** (annotations) or globally (config-map or operator)

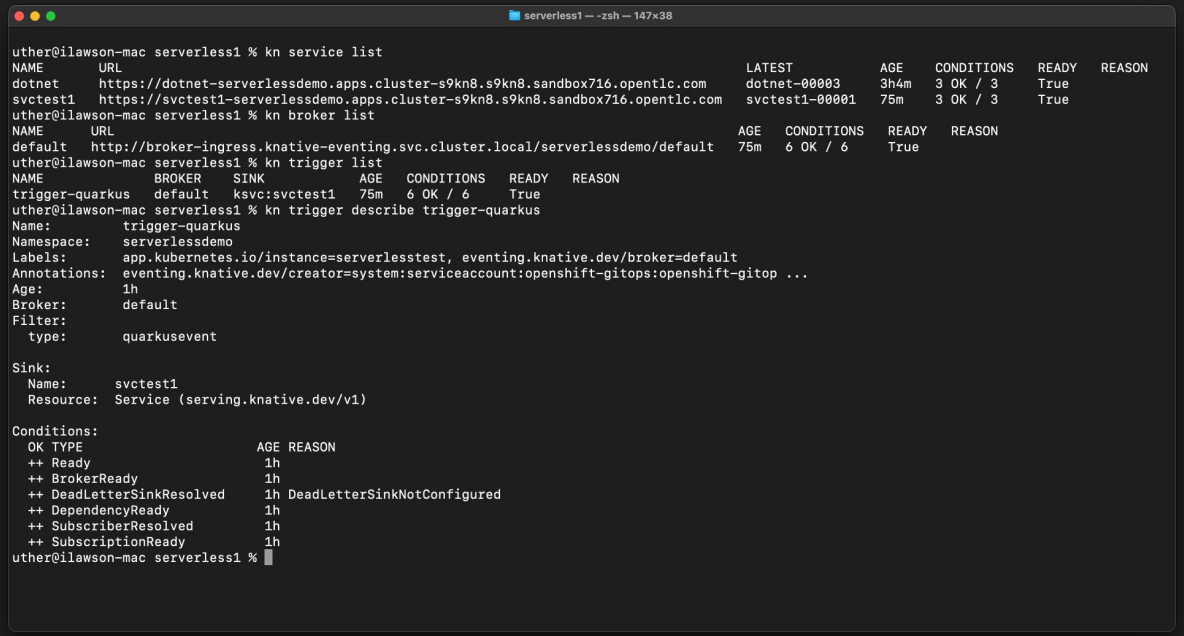
```yaml
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: helloworld-go
  namespace: default
spec:
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/target: "200"
```



Direct Control of Knative objects

Knative also provides a command-line utility called **kn**

kn allows you to create, modify and delete all knative objects directly on a target Kubernetes/OpenShift cluster



Knative Serving

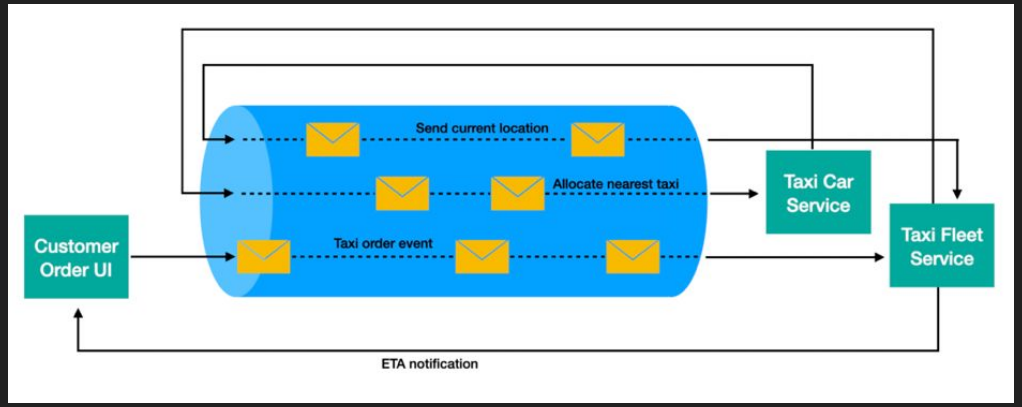

```
kn service create myservice --image docker.io/openshift/hello-openshift --env RESPONSE="hello1"
```

```
kn service update myservice --env RESPONSE="hello2"
```

```
kn service update myservice --traffic myservice-00001=50,myservice-00002=50
```


Red Hat
Developer
VOXXED DAYS
BRUSSELS
Knative Eventing

Event-driven architecture



Microservices that exchange info through events

Asynchronous (contrast to REST architectures)

Loosely coupled

Resilient

Scalable

Picture from: https://developer.ibm.com/articles/advantages-of-an-event-driven-architecture/

CloudEvents

CNCF incubating project - https://cloudevents.io/

Provides **a common event schema**
=> Interoperability, portability

Extensible through extension attributes

SDKs for different programming languages

Protocol-agnostic (HTTP, AMQP, MQTT, ...)

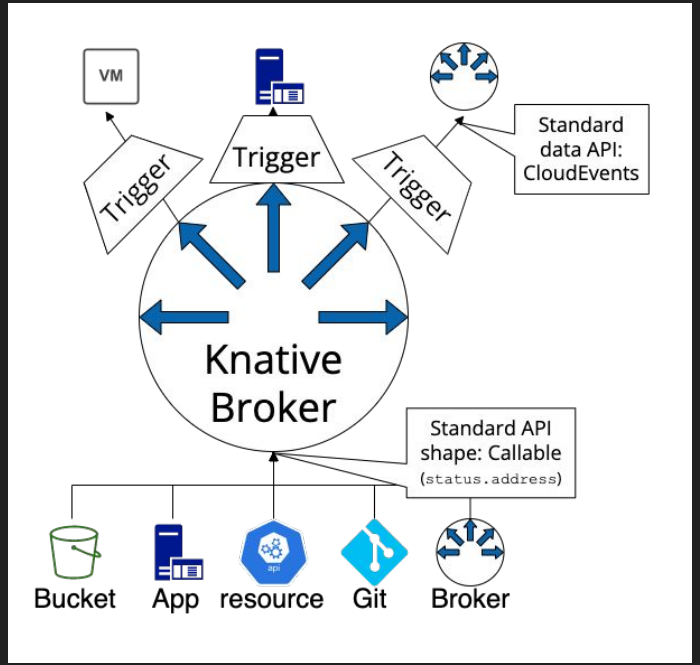
Wide adoption

Knative Eventing

Eventing is a set of APIs for routing events from **Producers** to **Consumers** (known as **Sinks**)

CloudEvent specification

allows for the creation of Serverless components that are driven by Event rather than Traffic

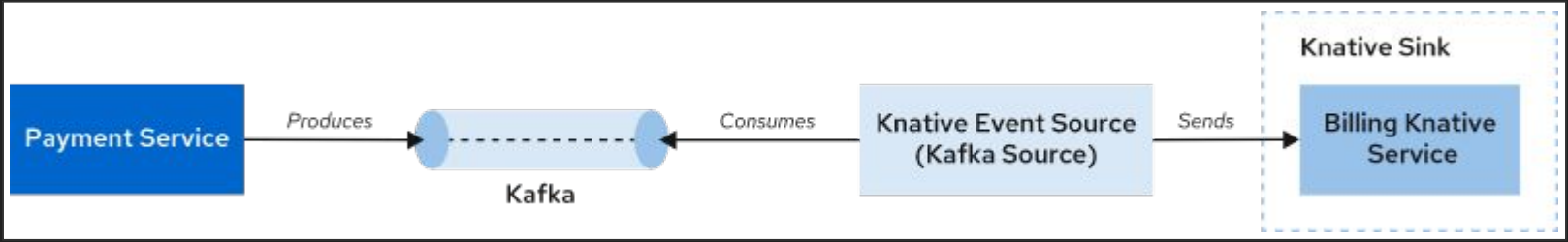


Knative Source

Knative Source:

- PingSource
- APIServerSource
- KafkaSource
- SinkBinding
- …

```
kn source ping create <pingsource-name> \
  --namespace <namespace> \
  --schedule "<cron-schedule>" \
  --data '<data>' \
  --sink <sink-name>
```

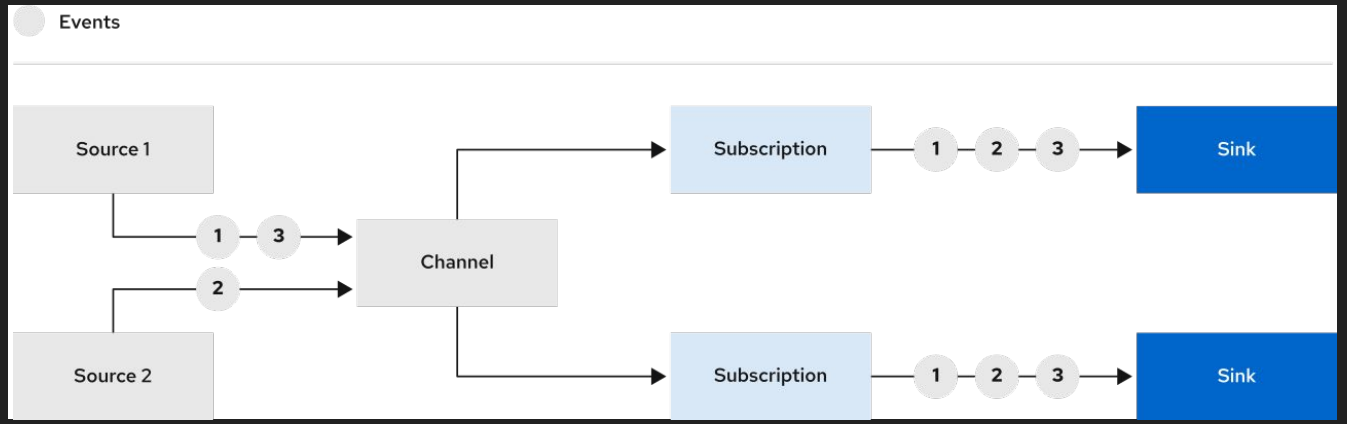


Channels and Subscriptions

provides a delivery mechanism for Cloudevents, through **Subscriptions**, to multiple destinations or **Sinks**

Sinks are consumers of events and can include **Brokers** and **Knative Services**

Subscriptions are used to hook together the **Channel** and the **Sink**

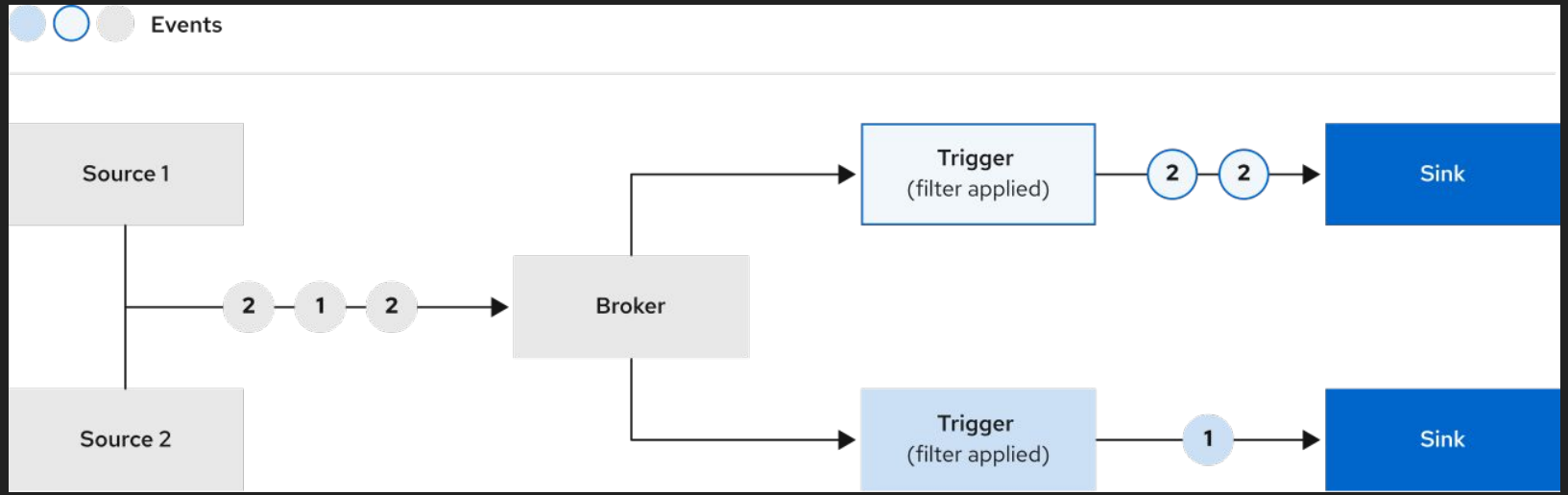


Brokers and Triggers

Brokers are namespace bound Cloudevent receivers/forwarders

Triggers are forwarders that push events to **Consumers**

Triggers use the cloudevent metadata (attributes) to filter cloudevents



Knative Eventing

kn broker create <broker-name>

kn trigger create <name> --sink <sink> --broker <broker-name> --filter <attribute>=<value>

kn source binding <name> --subject "Service:serving.knative.dev/v1:<service-name>" --sink broker:<broker-name>

Ephemeral or Persistent

Channel options (*):

- InMemoryChannel
- KafkaChannel
- NatssChannel

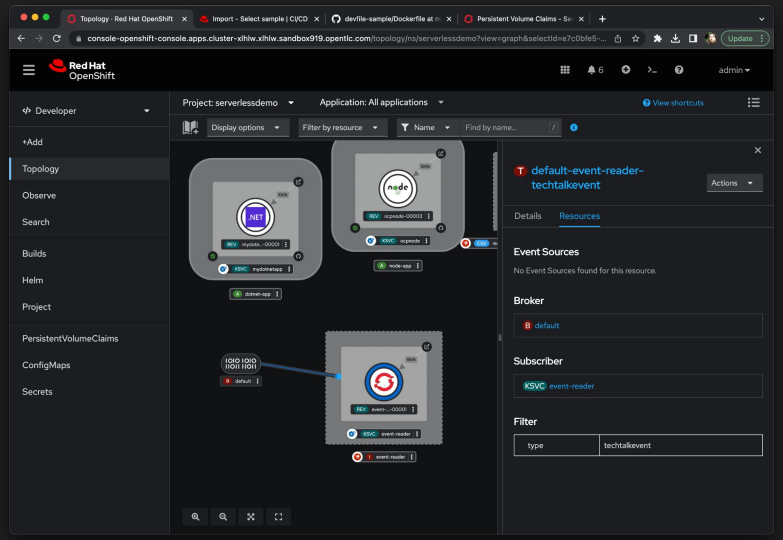
Broker options (*):

- Channel based broker
  - default
  - backed by InMemoryChannel
  - ephemeral
- Knative Broker for Apache Kafka
- RabbitMQ broker

* Brokers/channels lists on https://knative.dev/docs/eventing/

```
apiVersion: eventing.knative.dev/v1
kind: Broker
metadata:
  annotations:
    # case-sensitive
    eventing.knative.dev/broker.class: Kafka
    # Optional annotation to point to an externally
managed kafka topic:
    # kafka.eventing.knative.dev/external.topic:
<topic-name>
  name: default
  namespace: default
spec:
  # Configuration specific to this broker.
  config:
    apiVersion: v1
    kind: ConfigMap
    name: kafka-broker-config
    namespace: knative-eventing
```

Knative Eventing vs Serving



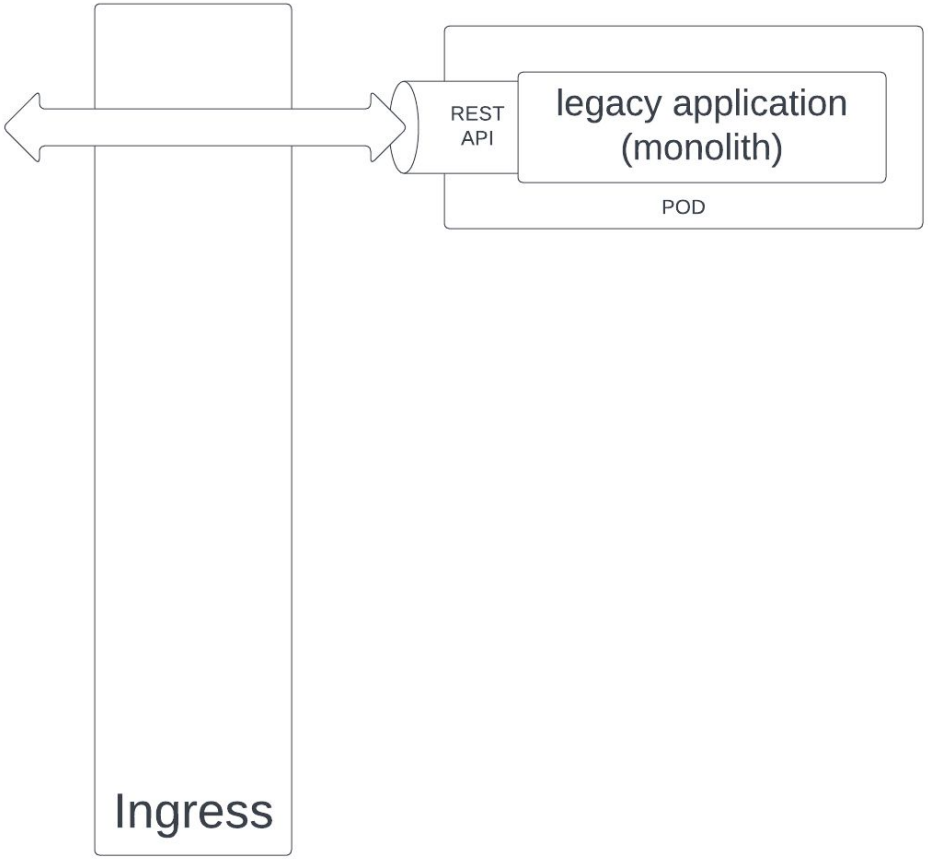
Serving allows us to create services that can scale dynamically

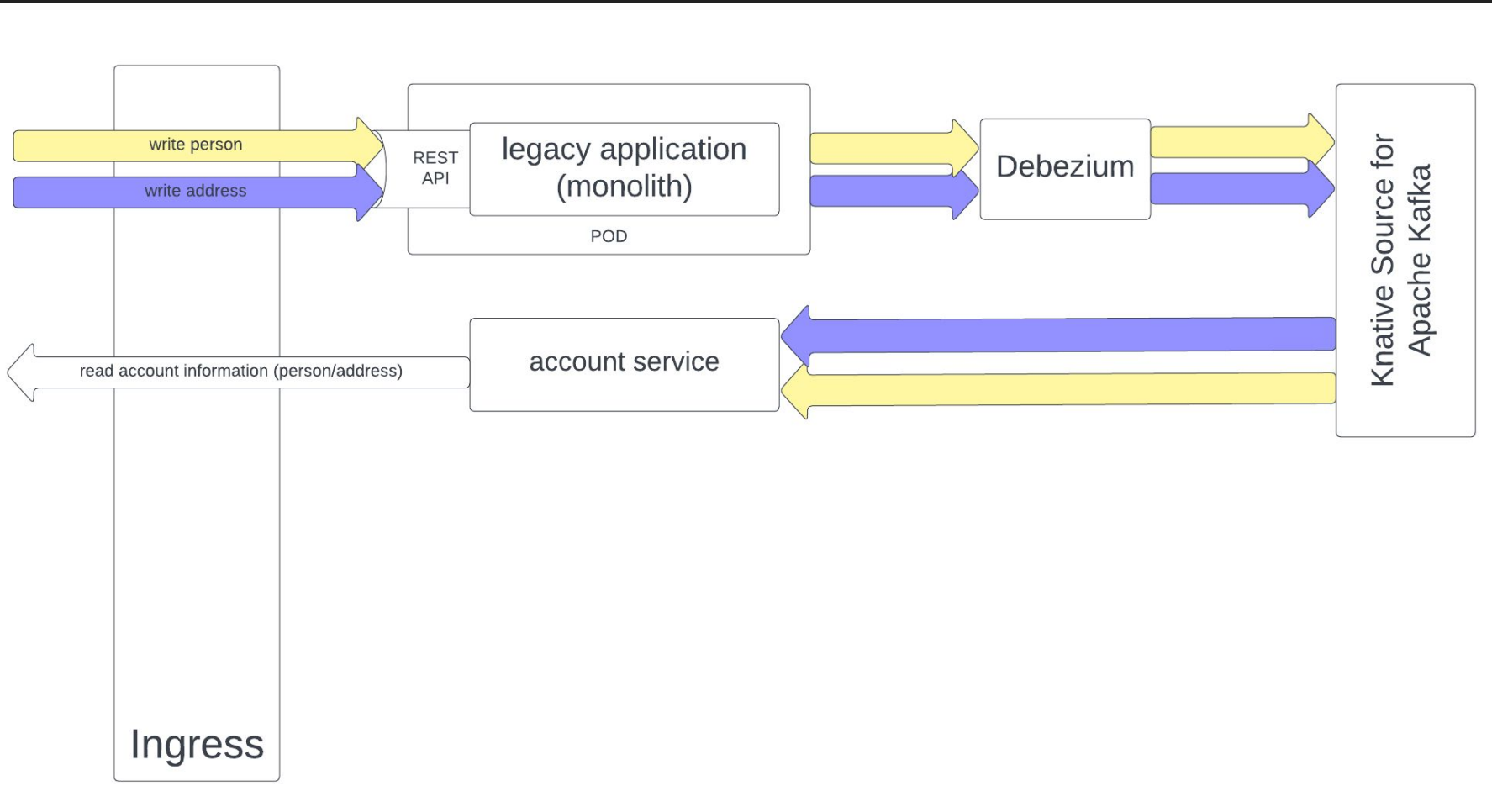
Eventing allows for the creation of Event-driven architectures

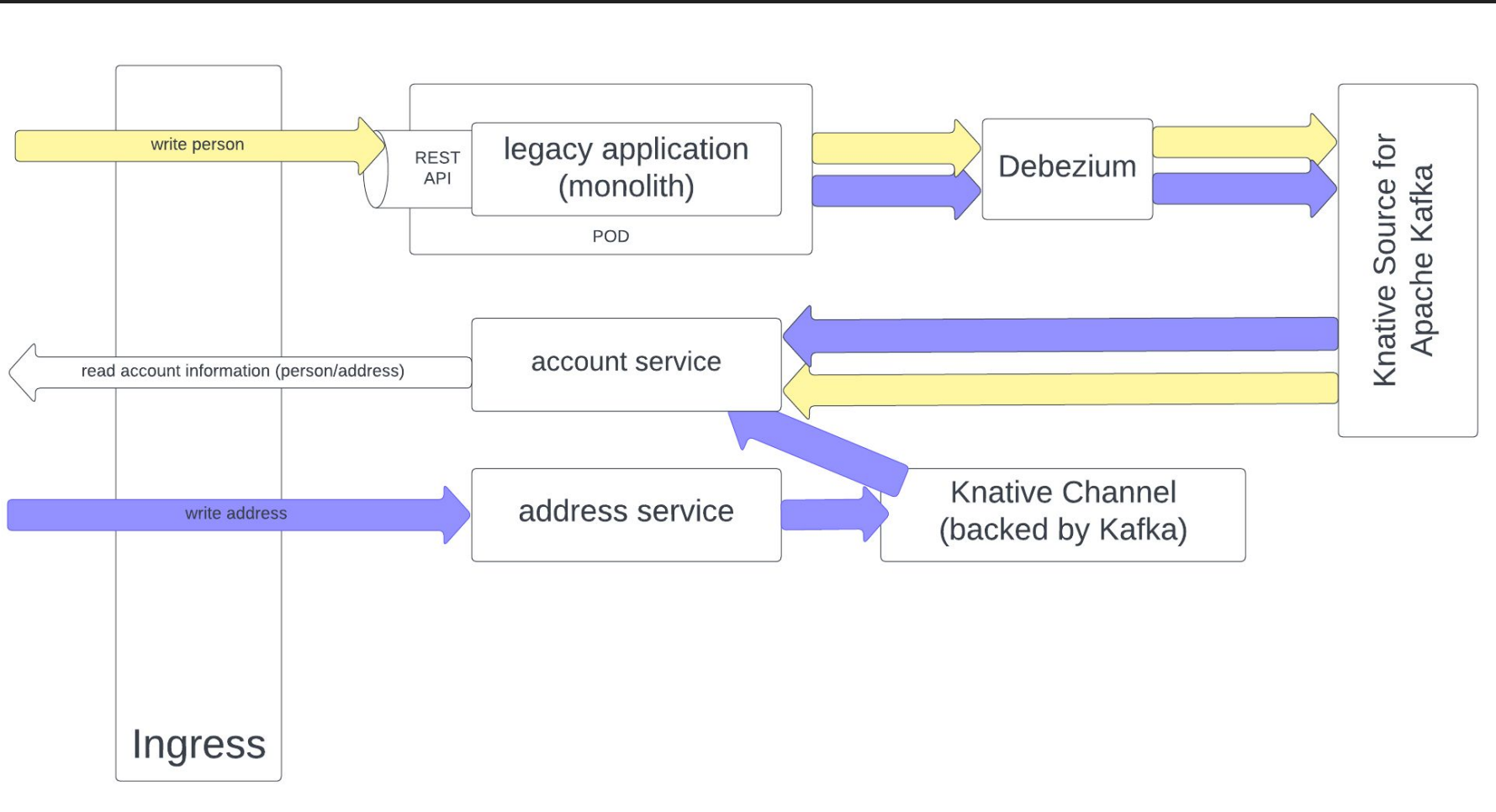
Knative Serving applications can act as **sinks** for Eventing => a combination of EDA and efficient resource Applications

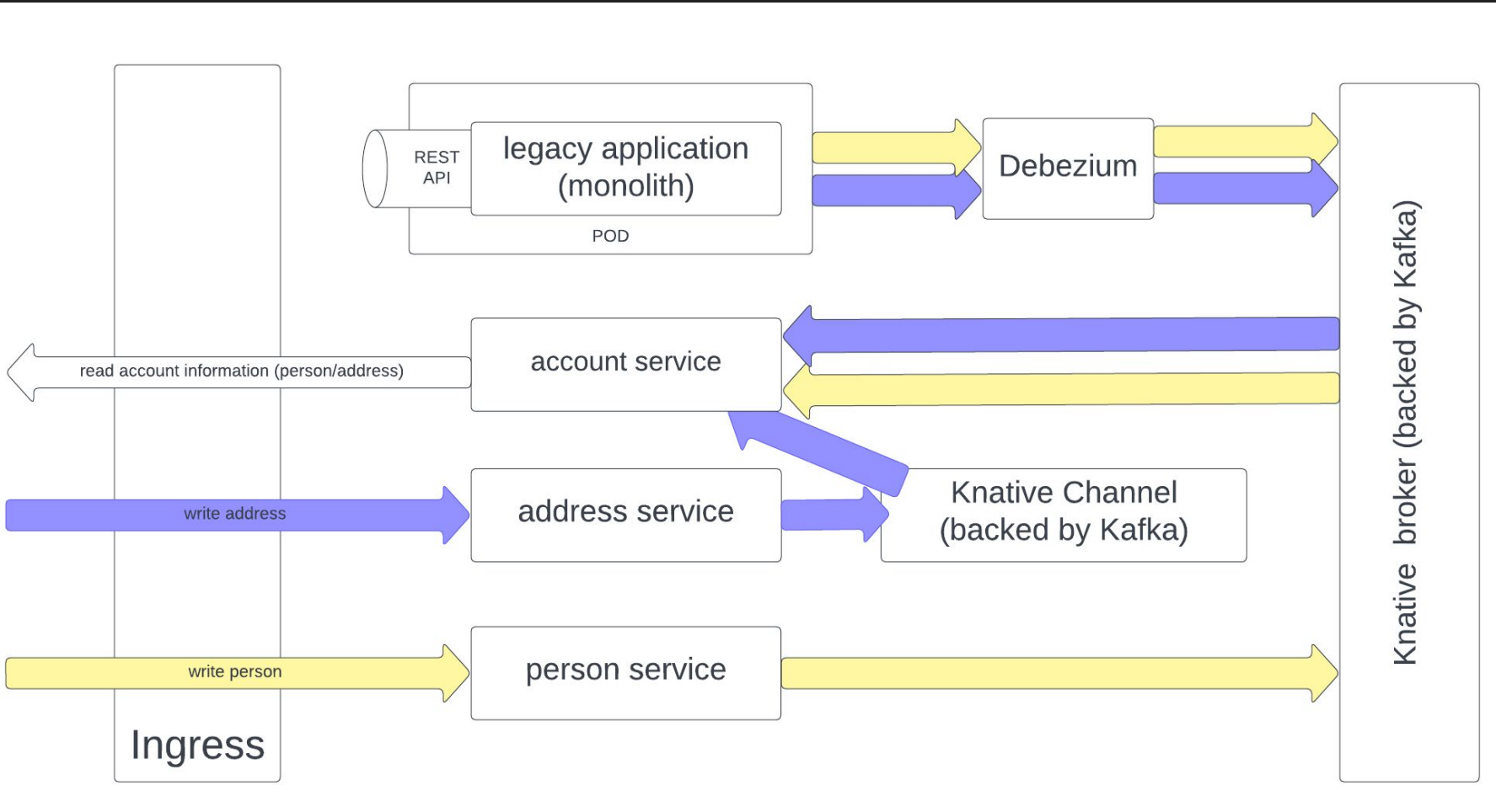
Use Case

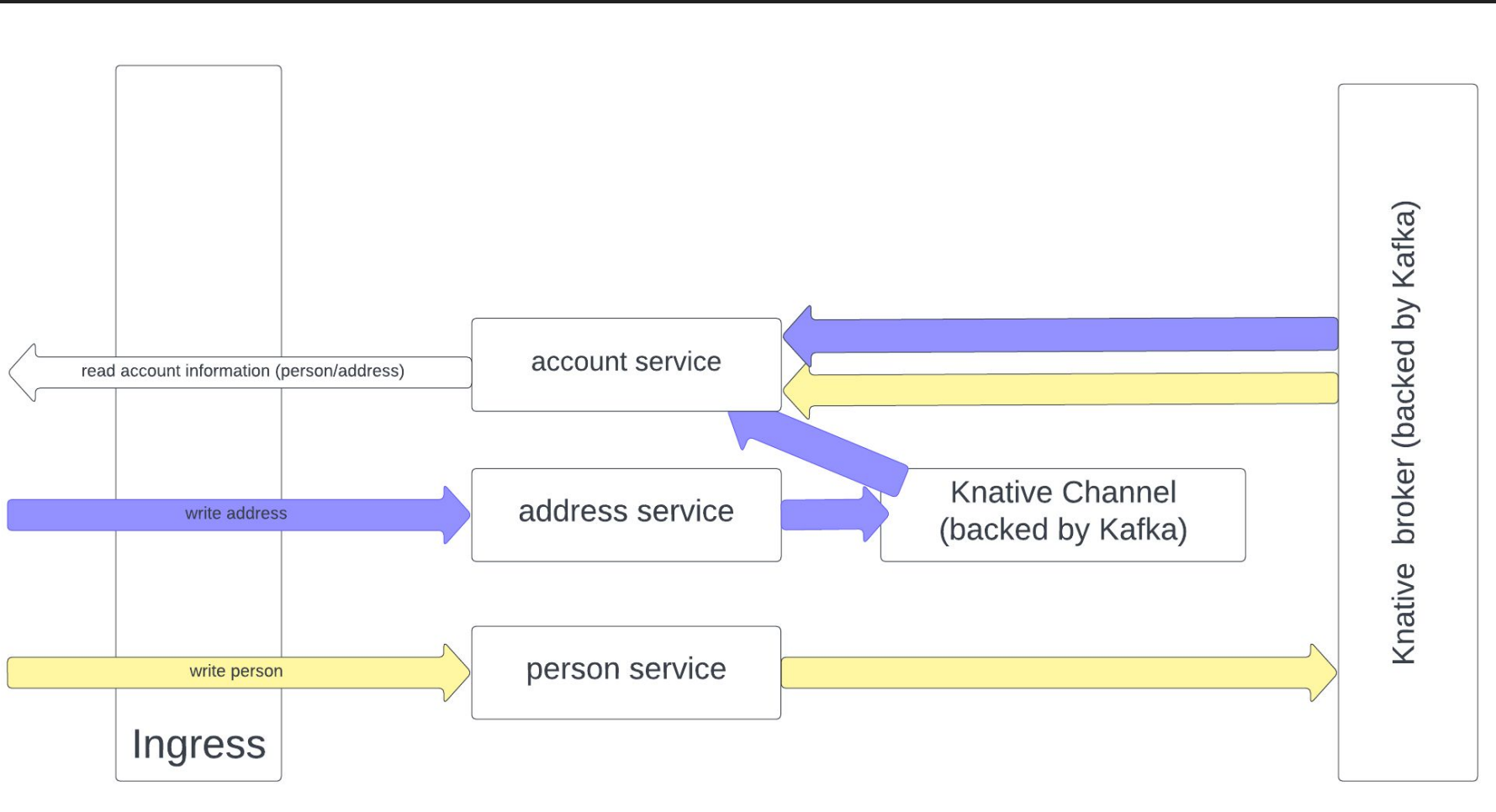

REST
API
legacy application
(monolith)


REST
API
legacy application
(monolith)
POD
Ingress


write person
write address
REST API
legacy application (monolith)
POD
Debezium
Knative Source for Apache Kafka
read account information (person/address)
account service
Ingress

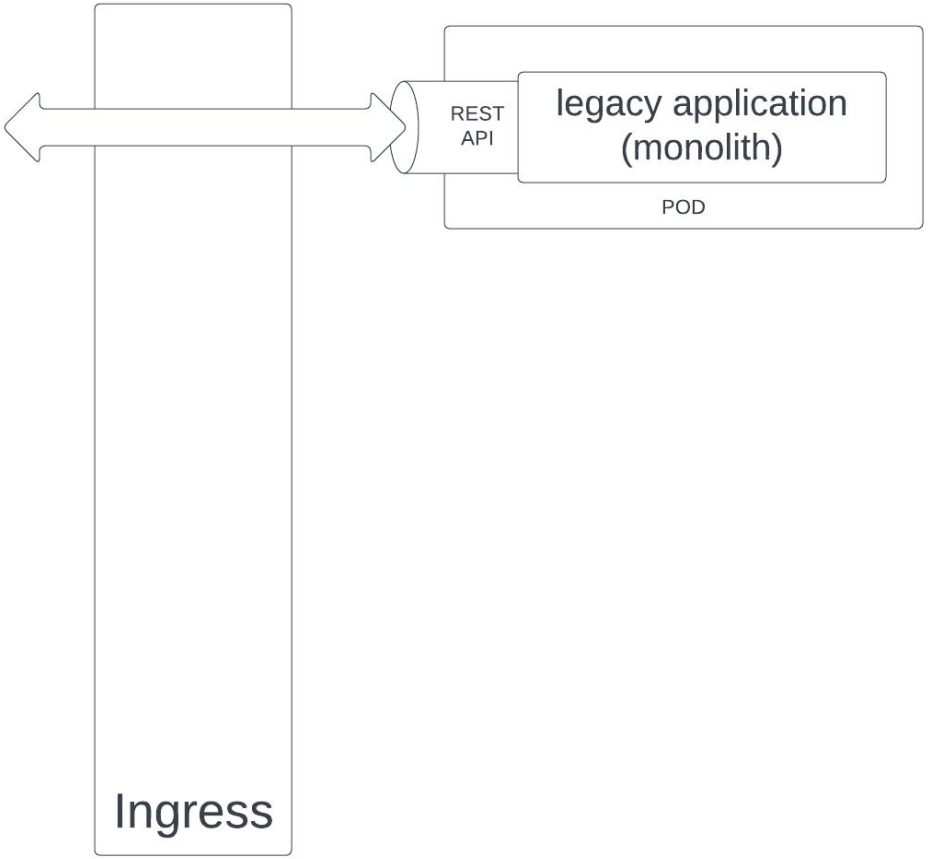

write person
REST API
legacy application (monolith)
POD
Debezium
Knative Source for Apache Kafka
read account information (person/address)
account service
write address
address service
Knative Channel (backed by Kafka)
Ingress

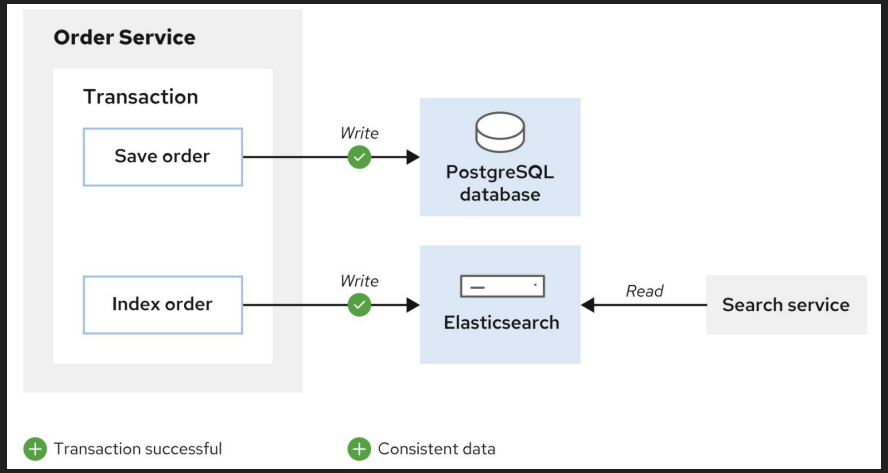

REST
API
legacy application
(monolith)
POD
Debezium
Knative broker (backed by Kafka)
read account information (person/address)
account service
write address
address service
Knative Channel
(backed by Kafka)
write person
person service
Ingress

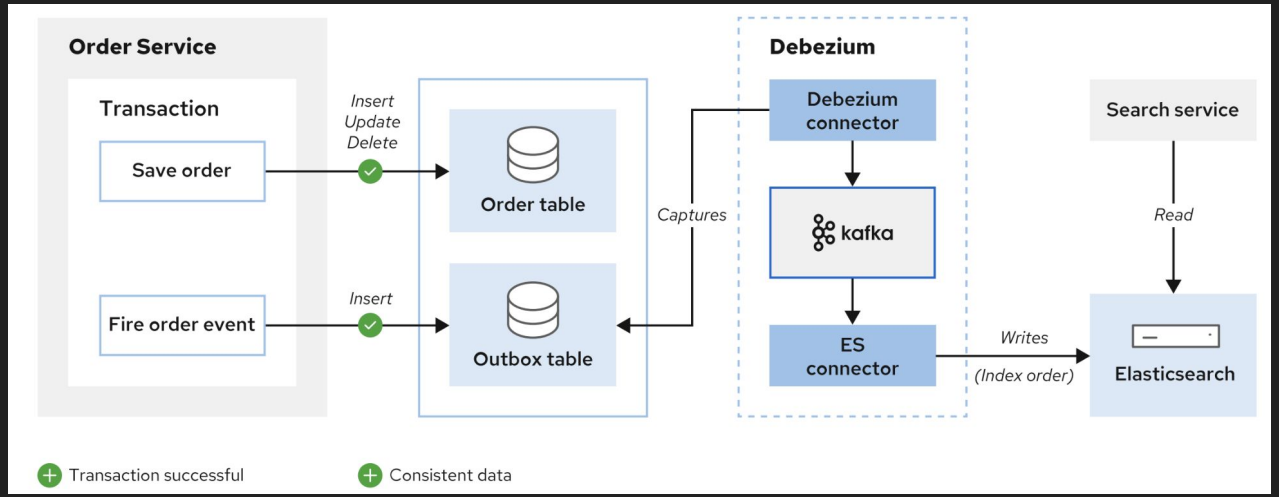

read account information (person/address)
account service
write address
address service
Knative Channel
(backed by Kafka)
write person
person service
Ingress
Knative broker (backed by Kafka)

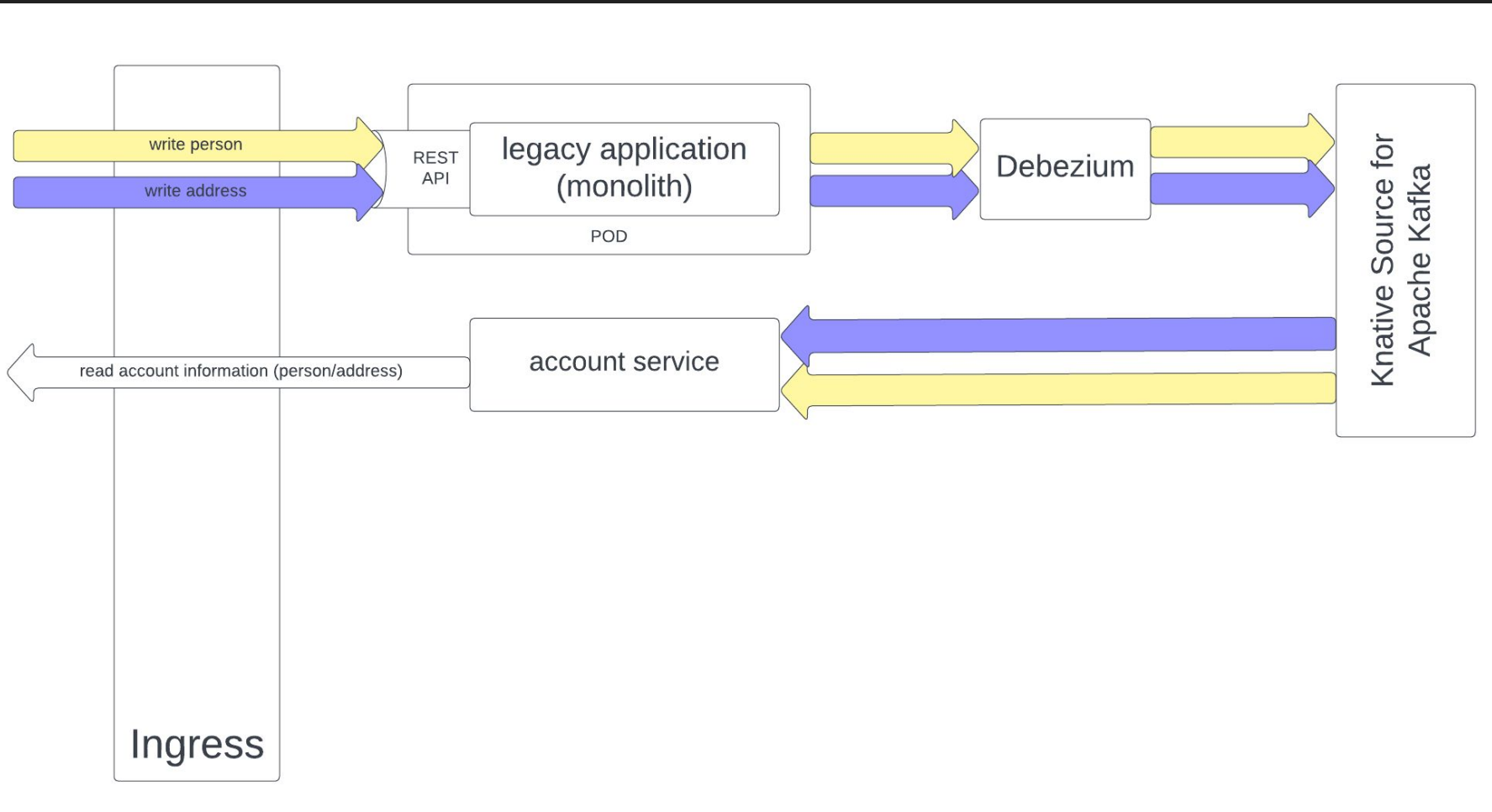

Red Hat
Developer
VOXXED DAYS
BRUSSELS
Demo Time

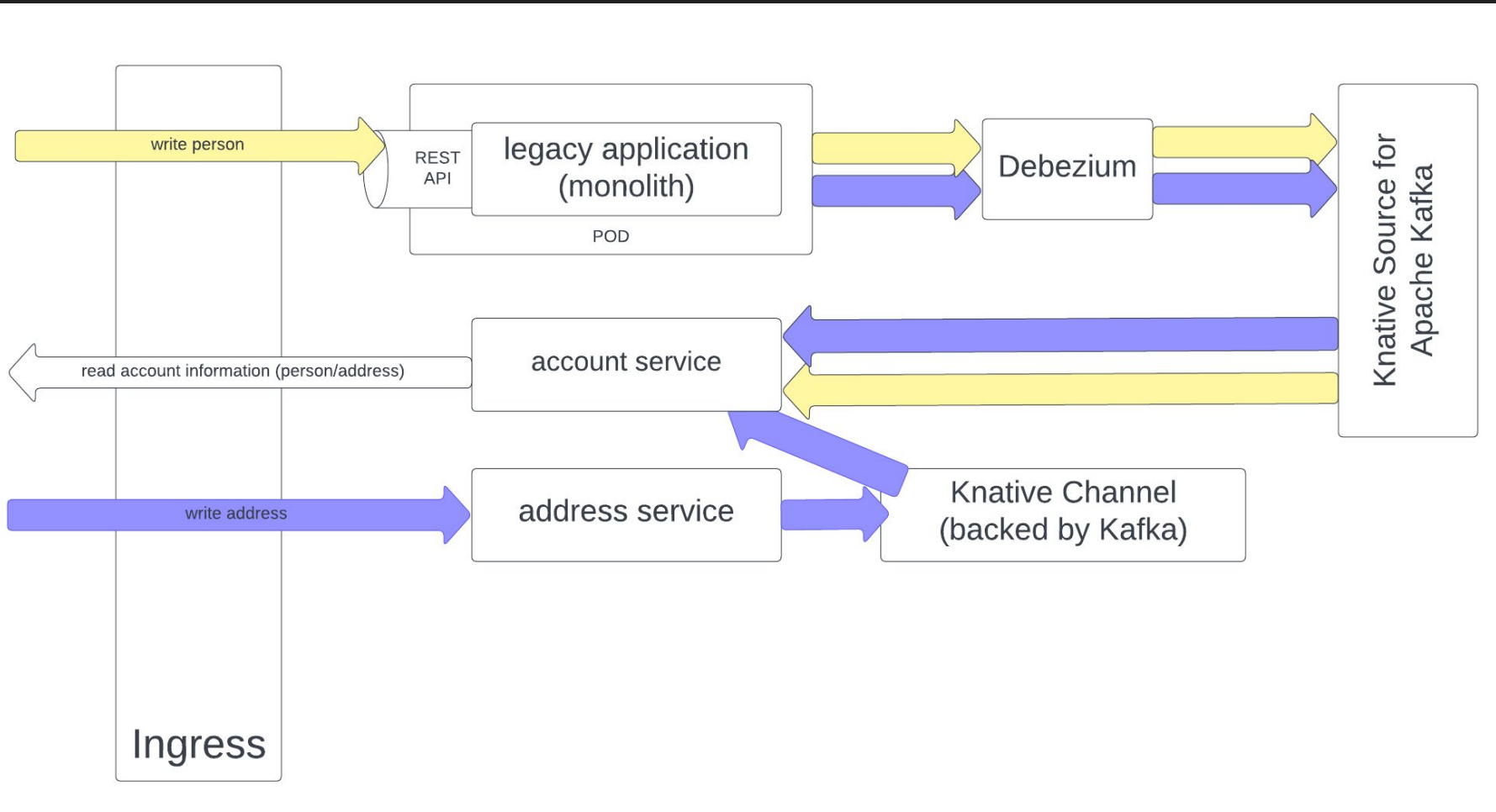

REST
API
legacy application
(monolith)

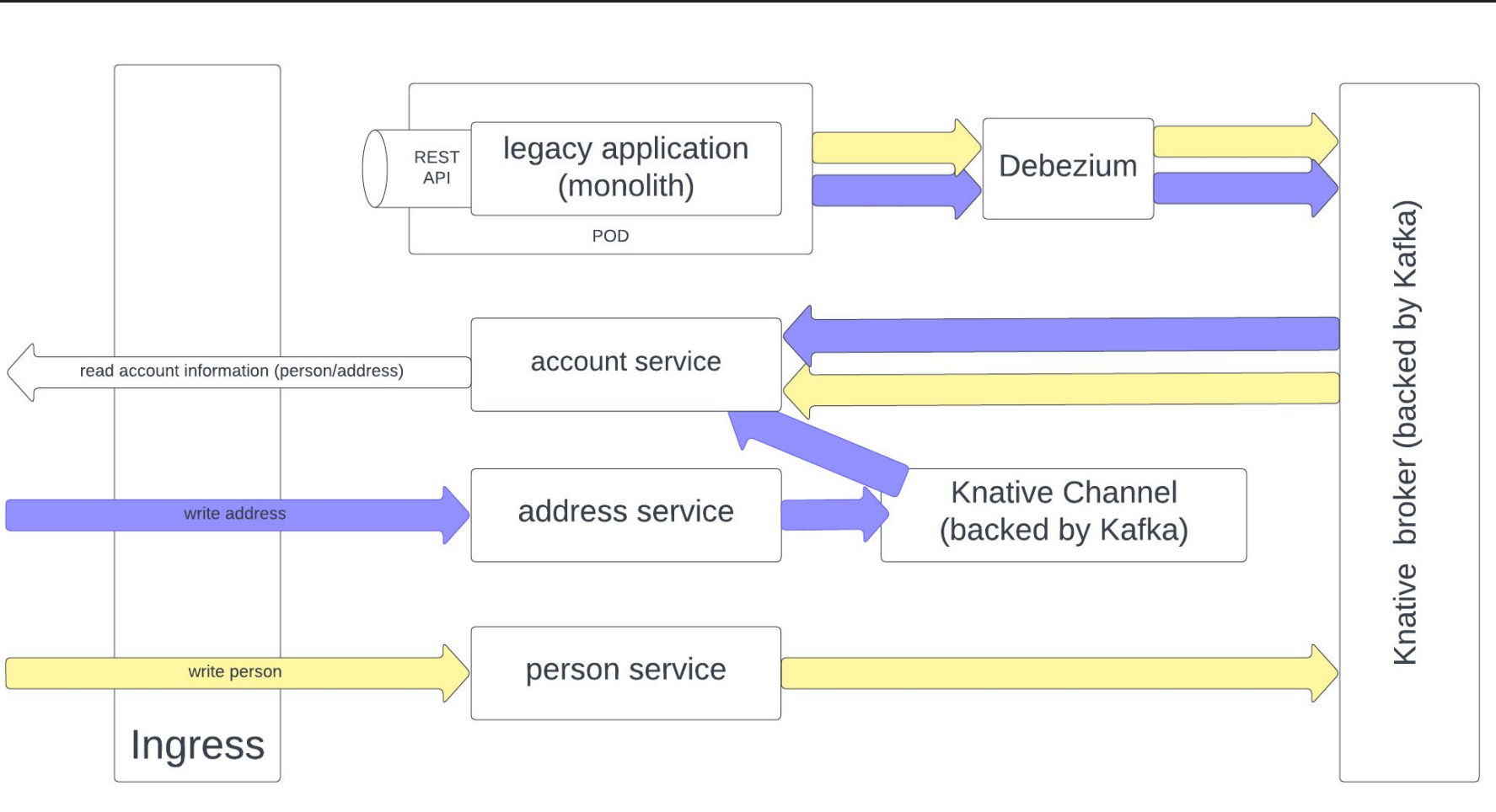

REST
API
legacy application
(monolith)
POD
Ingress

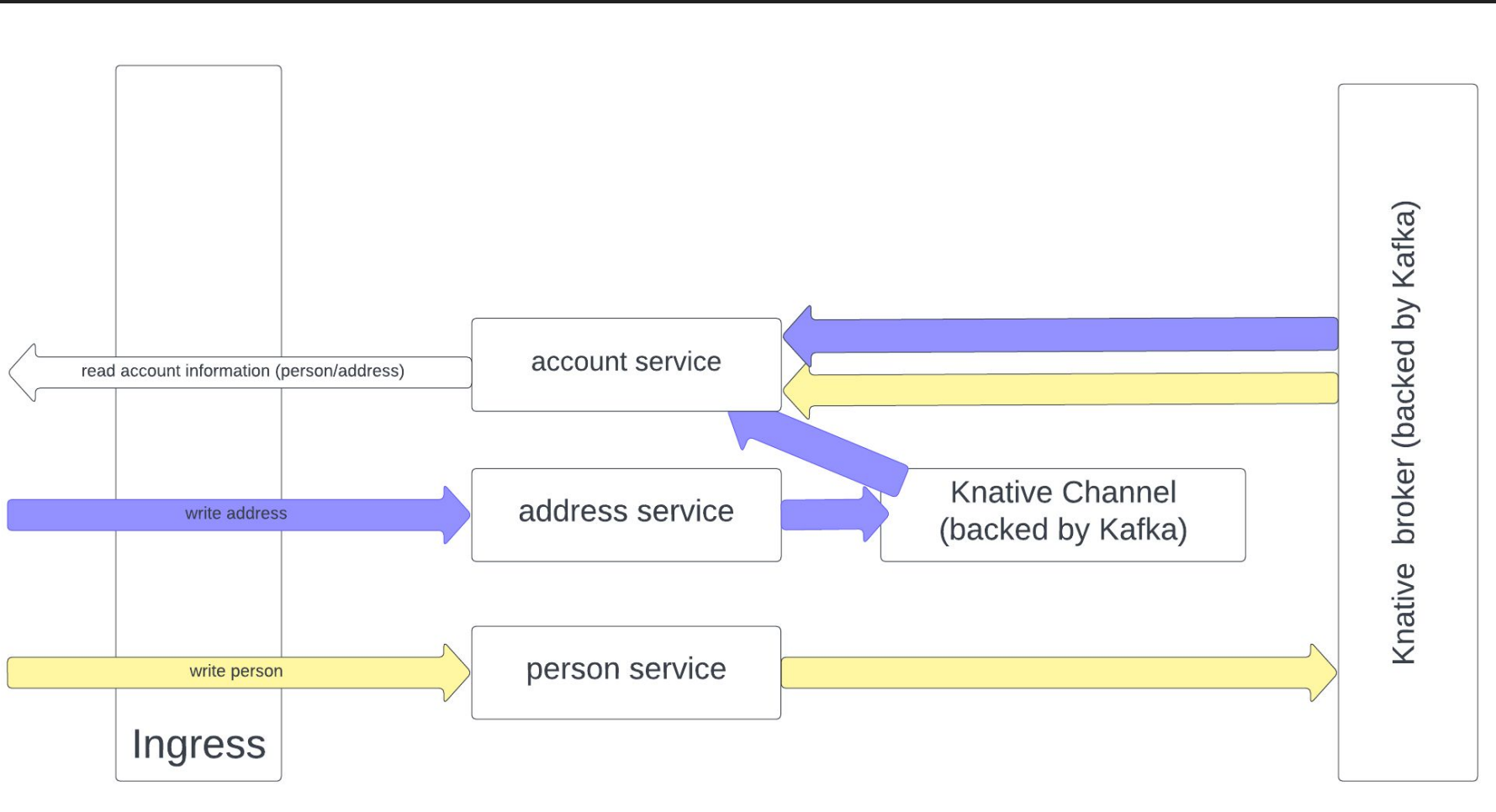

Order Service
Transaction
Save order
Write
PostgreSQL database
Index order
Write
Elasticsearch
Read
Search service
Transaction successful
Consistent data


Order Service
Transaction
Save order
Insert
Update
Delete
Order table
Fire order event
Insert
Outbox table
Captures
Debezium
Debezium connector
kafka
ES connector
Writes
(Index order)
Search service
Read
Elasticsearch
Transaction successful
Consistent data


write person
write address
REST API
legacy application (monolith)
POD
Debezium
Knative Source for Apache Kafka
read account information (person/address)
account service
Ingress


write person
REST API
legacy application (monolith)
POD
Debezium
Knative Source for Apache Kafka
read account information (person/address)
account service
write address
address service
Knative Channel (backed by Kafka)
Ingress

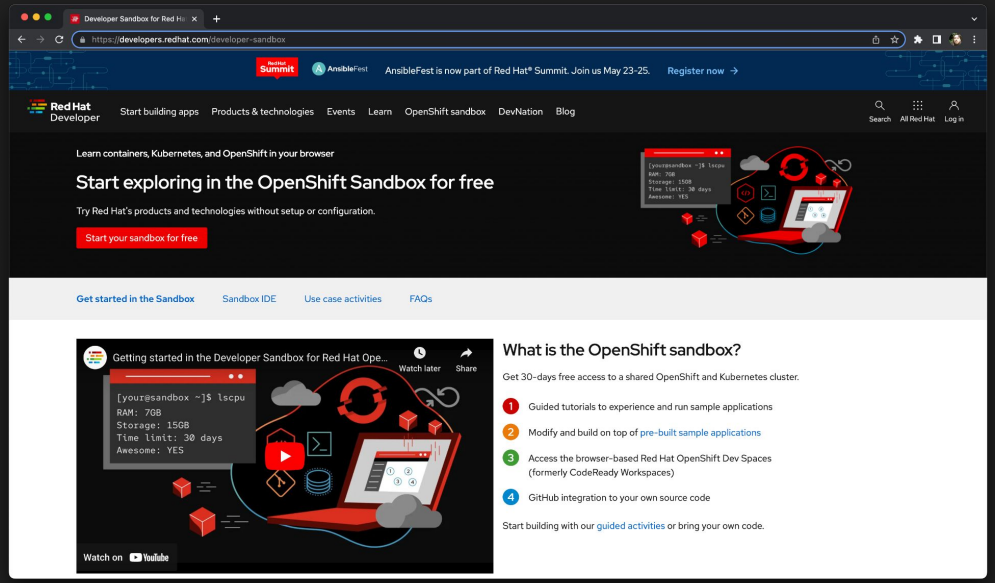

REST
API
legacy application
(monolith)
POD
Debezium
Knative broker (backed by Kafka)
read account information (person/address)
account service
write address
address service
Knative Channel
(backed by Kafka)
write person
person service
Ingress


read account information (person/address)
account service
write address
address service
Knative Channel
(backed by Kafka)
write person
person service
Ingress
Knative broker (backed by Kafka)

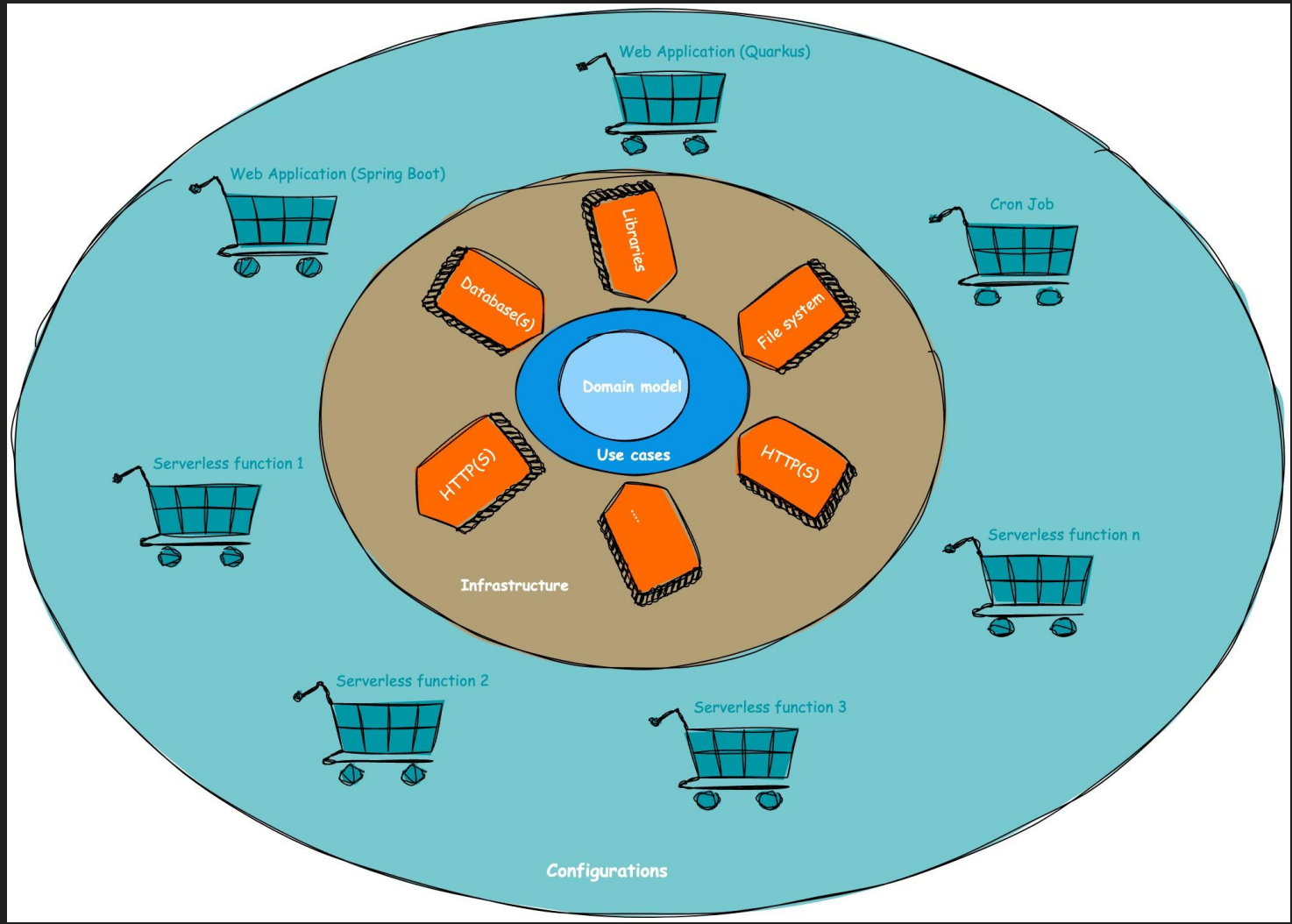
Want a quick tinker?

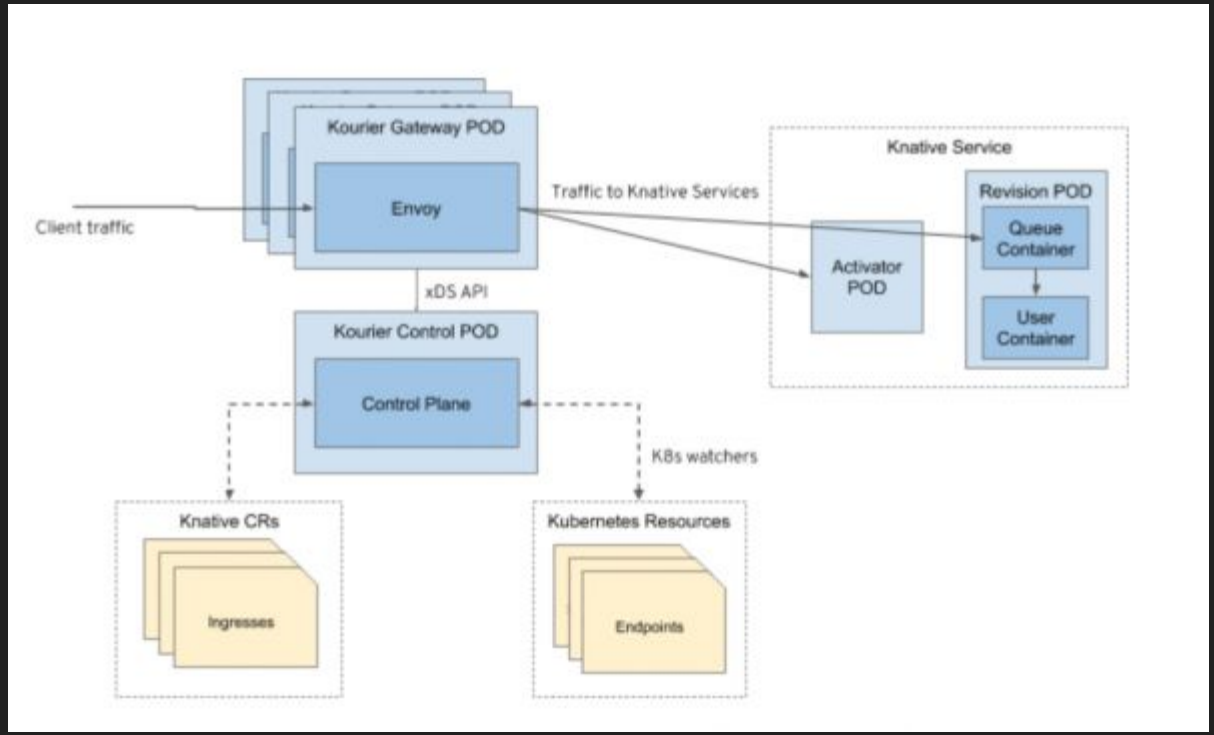
Red Hat provides a free-to-use OpenShift experience at:

https://developers.redhat.com/developer-sandbox




Thank you!


Web Application (Quarkus)
Web Application (Spring Boot)
Cron Job
Libraries
Database(s)
File system
Domain model
Use cases
HTTP(S)
HTTP(S)
....
Infrastructure
Serverless function 1
Serverless function n
Serverless function 2
Serverless function 3
Configurations

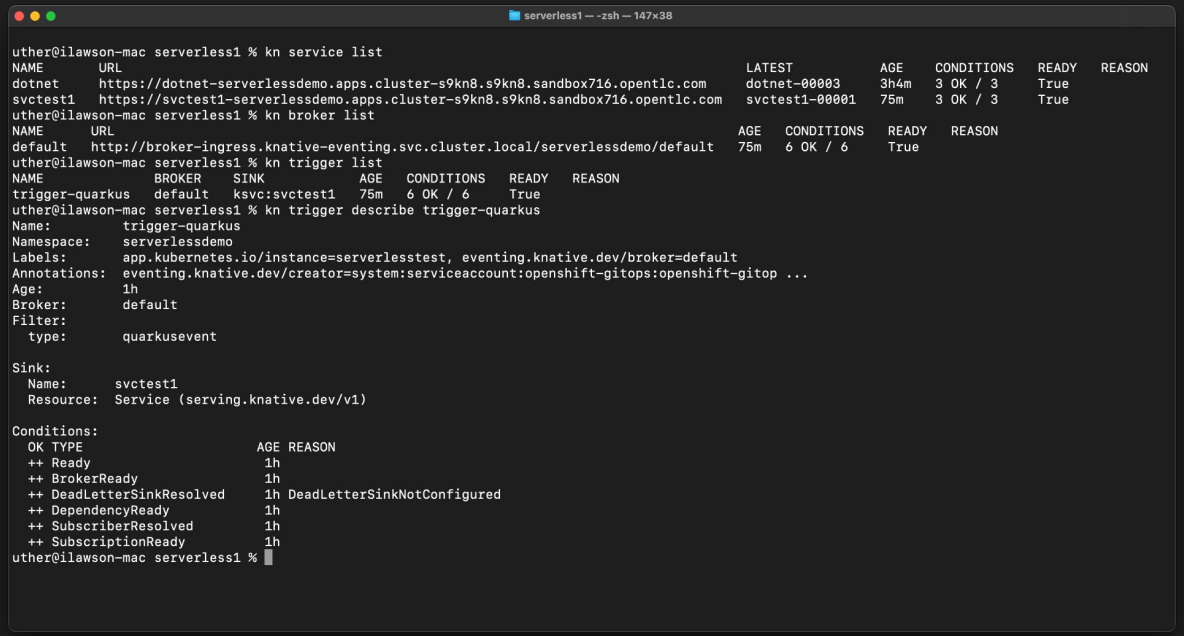

Kourier Gateway POD
Envoy
Client traffic
Traffic to Knative Services
Knative Service
Revision POD
Queue Container
User Container
Activator POD
xDS API
Kourier Control POD
Control Plane
K8s watchers
Knative CRs
Ingresses
Kubernetes Resources
Endpoints

Using kn

Direct Control of Knative objects

Knative also provides a command-line utility called **kn**

kn allows you to create, modify and delete all knative objects directly on a target Kubernetes/OpenShift cluster



kn examples

You can use kn to easily setup, for example, test event sources

```
kn source ping create testpingsource --schedule "* * * * */1"
--data "{\"payload\":\"cron test\"}" --sink ksvc:svctest1
```

You can create serverless components quickly and easily as well

```
kn service create scaleablesvc --image=quay.io/ilawson/devex4
--scale-min=2 --scale-max=10
```

And update the traffic distribution for services

```
kn service update myService --traffic myService-rev1=50,myService-rev2=50
```