Red Hat Summit

Connect

Red Hat | intel.

Red Hat Summit

Connect

OpenShift Serverless

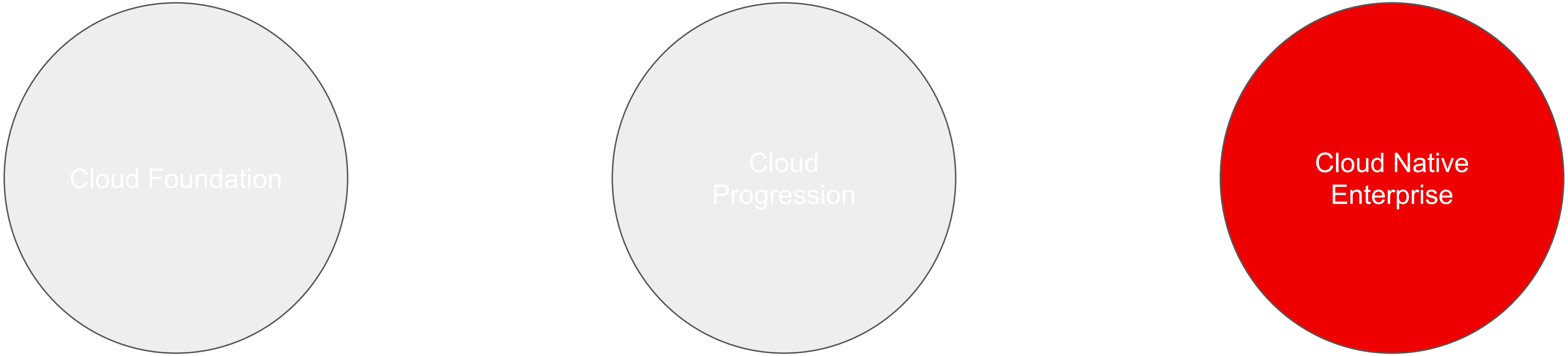
The Next Generation of Application Architectures

Red Hat | intel.

IAN 'UTHER' LAWSON

Developer Advocate, OpenShift Evangelist
Red Hat

Journey to the Cloud

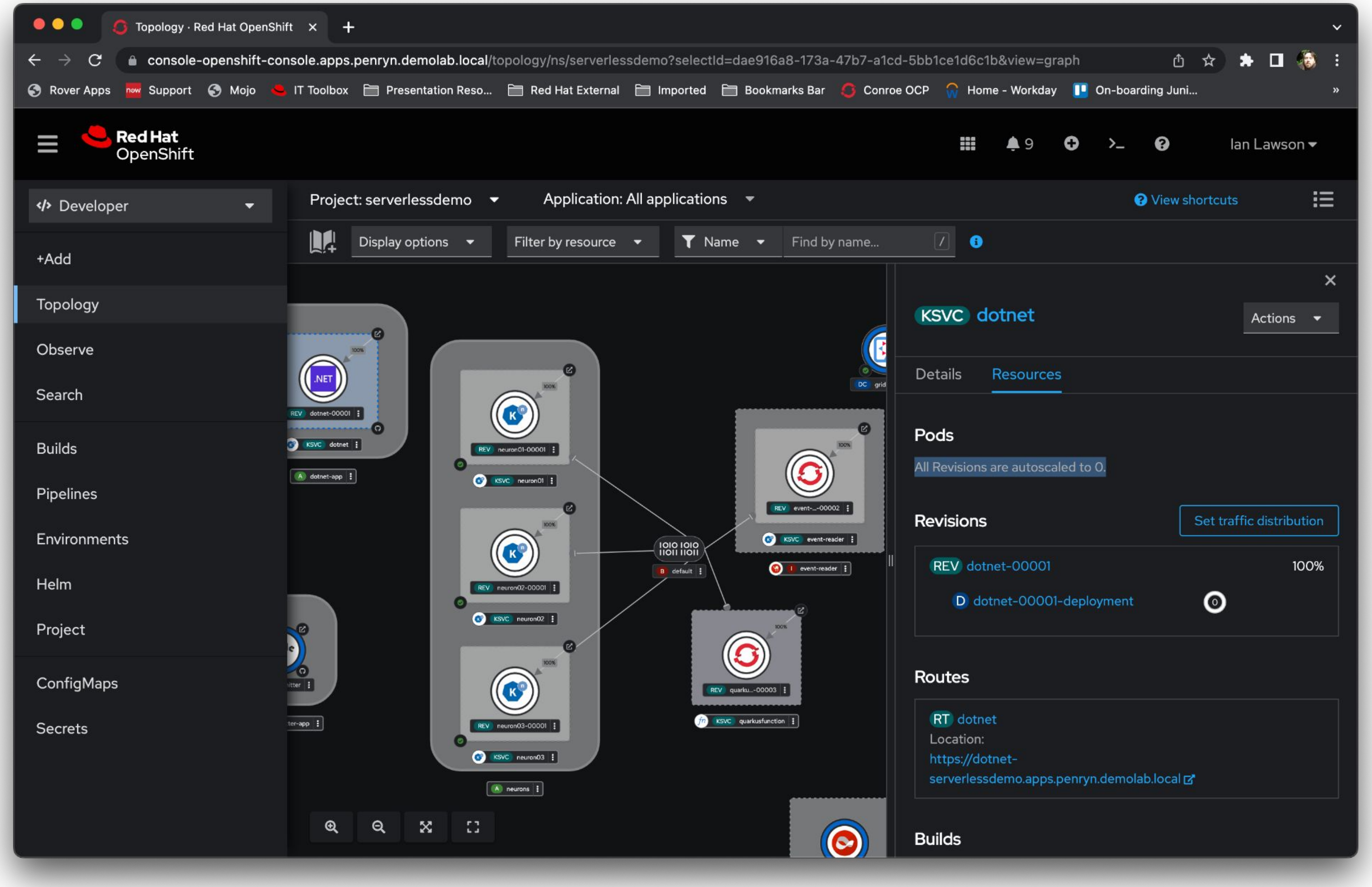


Cloud Foundation

Cloud Progression

Cloud Native Enterprise

Serverless – good idea, bad name...



Pods

All Revisions are autoscaled to 0.

Revisions

REV dotnet-00001 100%

D dotnet-00001-deployment 0

Routes

RT dotnet

Location:

https://dotnet-serverlessdemo.apps.penryn.demolab.local

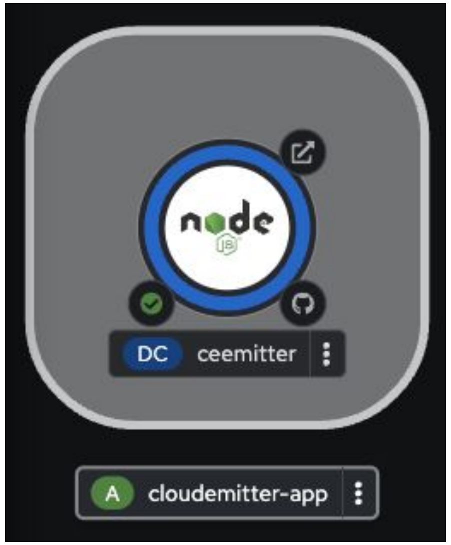
Builds

Persistent Application Model

Kubernetes maintains a ‘persistent application model’

In English - Containers orchestrated in Pods remain resident and reconciled

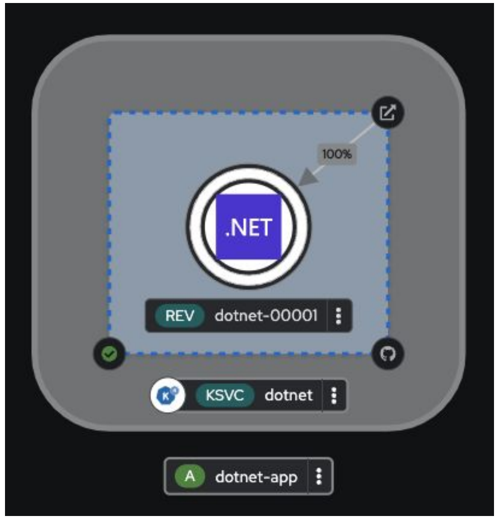
“Up all the time”



On-Demand Application Model

Knative Serverless introduces the concept of 'on-demand orchestration'

In English - the Application is only resident in memory and active for the duration of an interaction

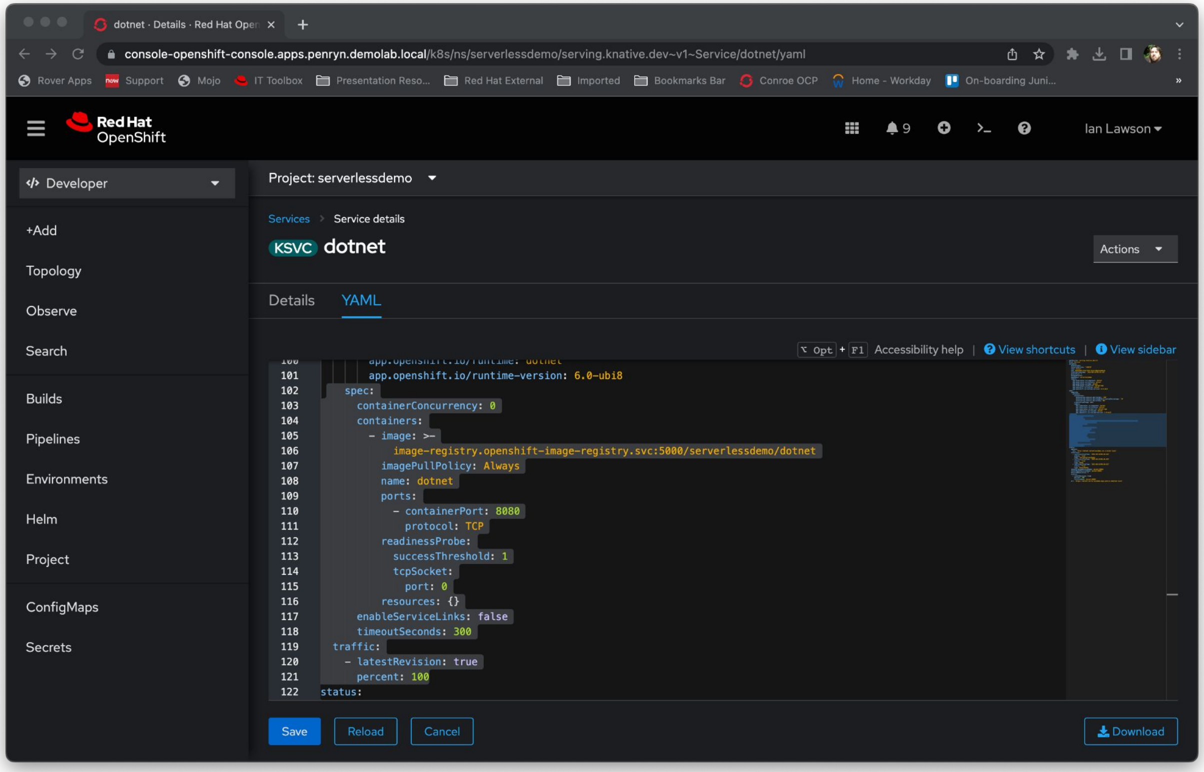


The Mechanics of Serverless Orchestration

Applications defined by a ‘Knative Service object’.

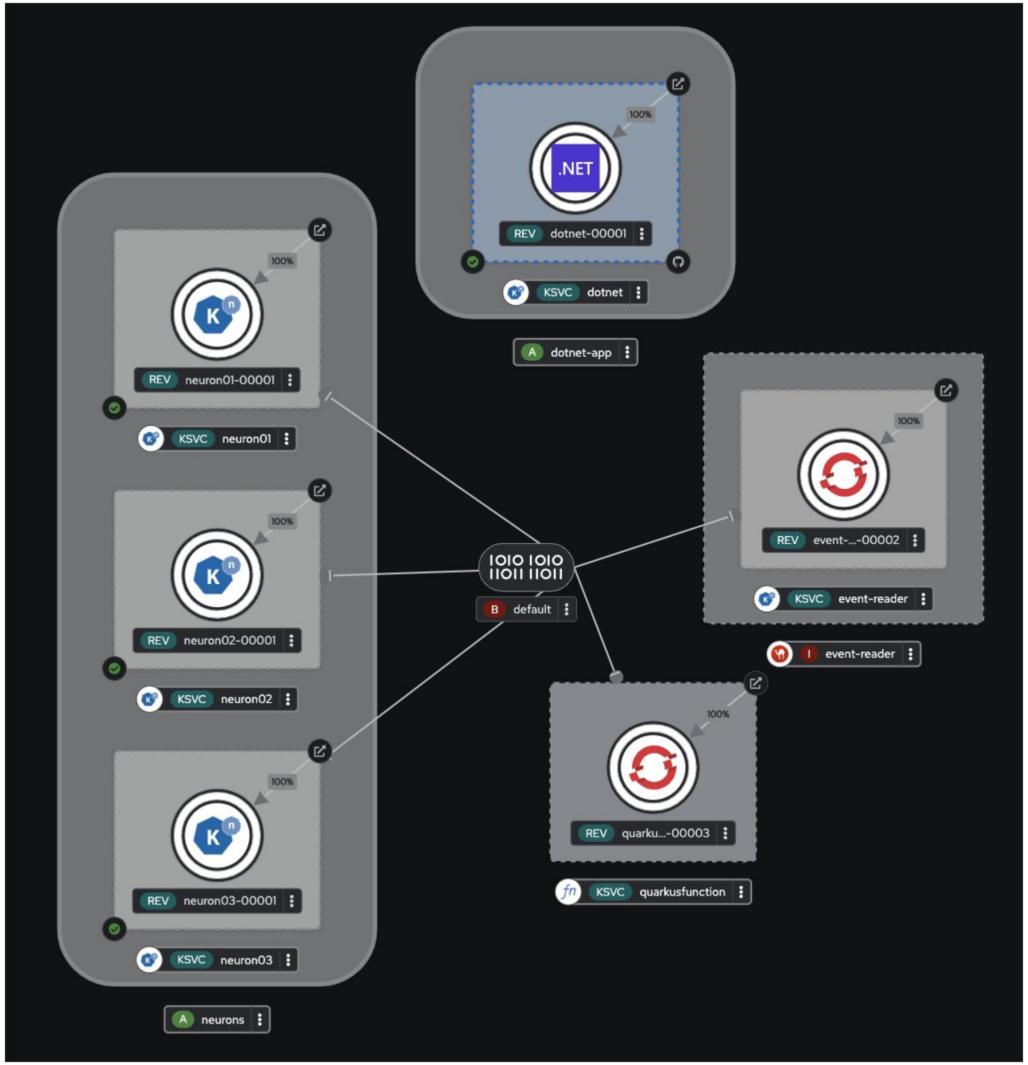
This provides a consistent traffic endpoint in the system, *regardless* of whether the target Pod(s) are resident

This traffic endpoint handles the recreation of the Pod(s) if required (when a call occurs to a down-scaled Application)



```yaml
        app.openshift.io/runtime: dotnet
        app.openshift.io/runtime-version: 6.0-ubi8
    spec:
      containerConcurrency: 0
      containers:
        - image: >-
            image-registry.openshift-image-registry.svc:5000/serverlessdemo/dotnet
          imagePullPolicy: Always
          name: dotnet
          ports:
            - containerPort: 8080
              protocol: TCP
          readinessProbe:
            successThreshold: 1
            tcpSocket:
              port: 0
          resources: {}
      enableServiceLinks: false
      timeoutSeconds: 300
  traffic:
    - latestRevision: true
      percent: 100
status:
```

The Types of Serverless Orchestration



There are two mechanisms for triggering the call/scale-up of a Serverless Application

1. "Serving" - in which the trigger method is traffic arriving at a service endpoint

2. "Eventing" - in which the trigger method is a native "Cloud Event" arriving at the service endpoint

Eventing - Triggers and Cloud Events

Eventing works using a Project-bound event hub called a "Broker"

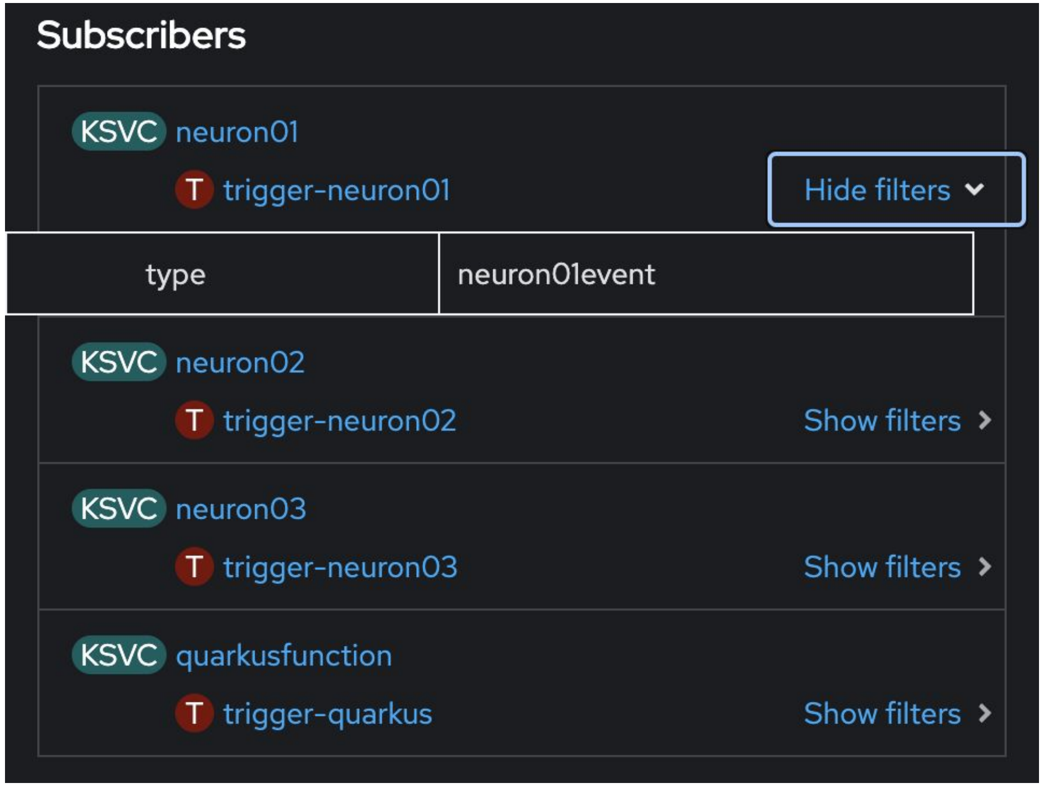
You can have many Brokers in a project uniquely identified by a name

These Brokers have "subscribers", indicated by the use of "Trigger" object

Cloud Events are basically a packet with an ID, an Origin and a Type

The ID and Origin act as a unique identifier for single send

The Type is matched against Triggers and if a match occurs the Event (labels and payload) is sent to the appropriate Knative service point, which does the magic



Subscribers

KSVC neuron01
T trigger-neuron01 — Hide filters

type	neuron01event

KSVC neuron02
T trigger-neuron02 — Show filters

KSVC neuron03
T trigger-neuron03 — Show filters

KSVC quarkusfunction
T trigger-quarkus — Show filters

Architectural Considerations

Serverless provides a highly efficient way of hosting fragmented Applications

You get much more *bang* for your *buck*; you can host hundreds of Application components in a much smaller footprint

By doing a form of atomic decomposition on the functionality of your Applications and then implementing each micro-service as either a Serving or Eventing Knative service you get agility and flexibility in developing and hosting complex applications

Currently there are caveats - Knative Applications do not support the use of PVs (because the spin-up, spin-down time is radically affected by the mounting and unmounting of external file systems), but this can be engineered around

Ease of Development

Kubernetes is hard and complex (although elegant and simple in design)

Knative Functions provide a simple programming model for creating functions on Knative without having to have in-depth knowledge of Knative, Kubernetes, containers or dockerfiles

This provides a CLI extension for kn (the Knative CLI) called “func”

Using a yaml definition, this CLI will build and run, including adding all the wiring and scaffolding, Knative services/functions

```
name: quarkusfunction
namespace: ""
runtime: quarkus
registry: “”
image: quay.io/ilawson/techtalkfunction
trigger: events
builder: default
builders:
  default: quay.io/boson/faas-quarkus-jvm-builder
  jvm: quay.io/boson/faas-quarkus-jvm-builder
  native: quay.io/boson/faas-quarkus-native-builder
buildpacks: []
buildEnvs: []
envVars:
  TESTENV: test_env_value
```

Demo Time....

Red Hat Summit

Connect

Thank you

in linkedin.com/company/red-hat

f facebook.com/redhatinc

youtube.com/user/RedHatVideos

twitter.com/RedHat

Red Hat | intel.