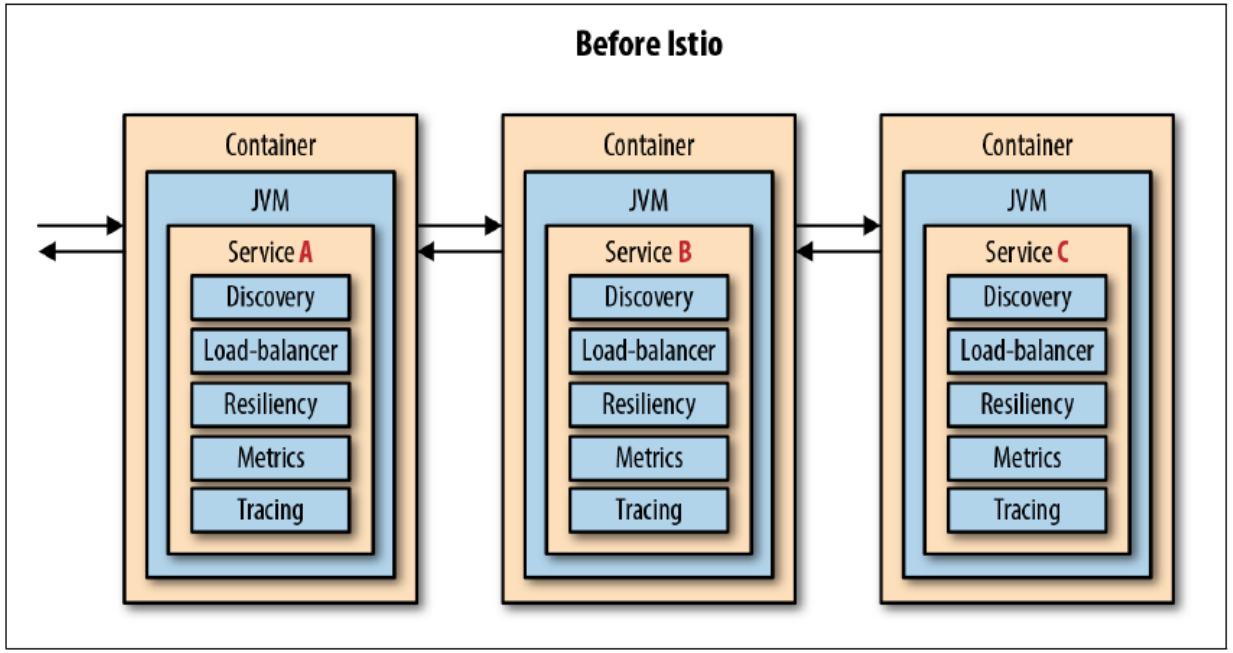
IBM MICROSERVICE I
SESSION 4

TEXAS SKILLS DEVELOPMENT FUND
TRAINING PARTNERSHIP

AUSTIN COMMUNITY COLLEGE DISTRICT

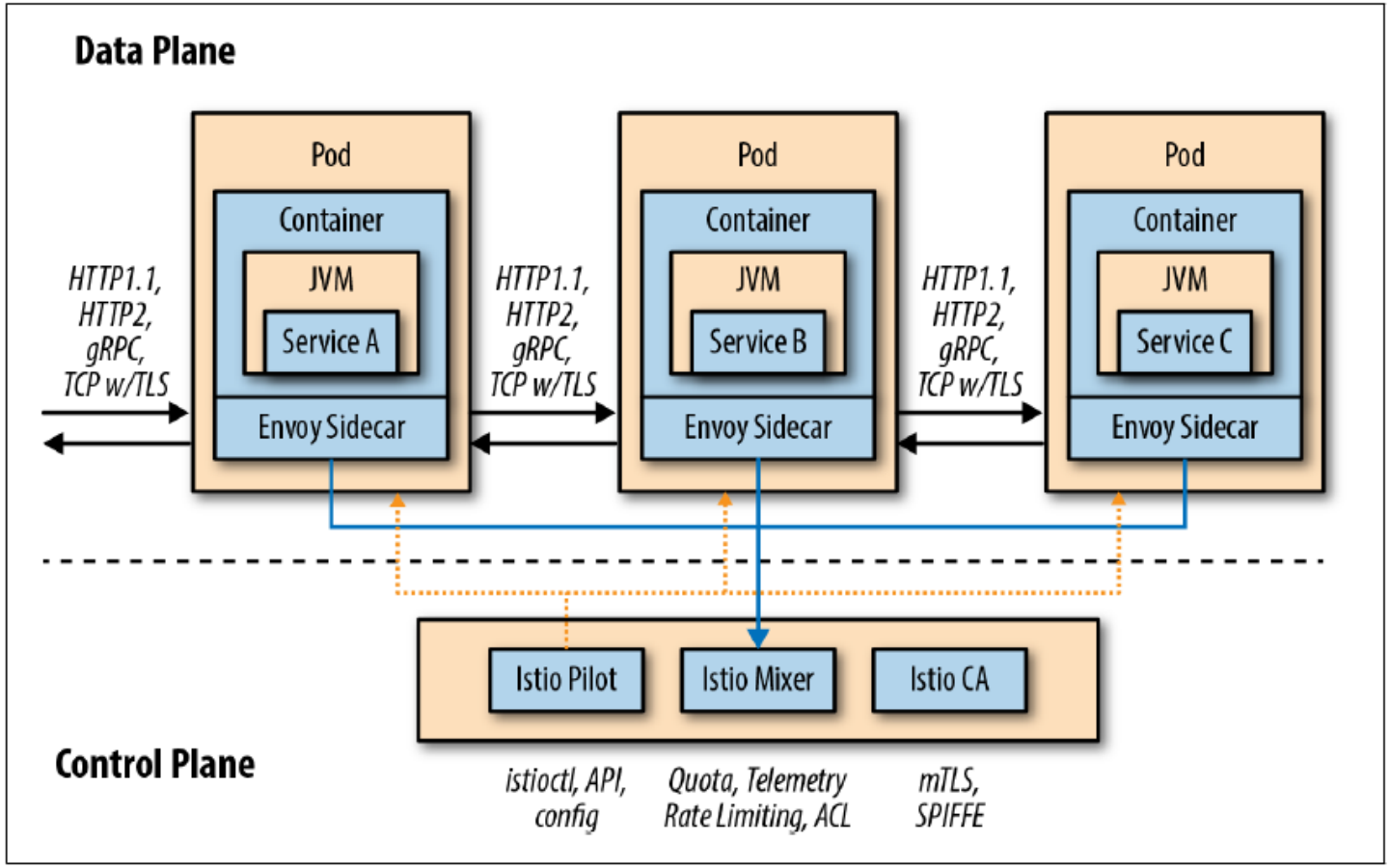
Why Istio ?

- Service mesh - connective tissue between services
  - Traffic control
  - Service discovery
  - Load balancing
  - Resilience
  - Observability
  - Security
  - …



[Introducing Istio Service Mesh for Microservices | Red Hat Developer]

Istio Components

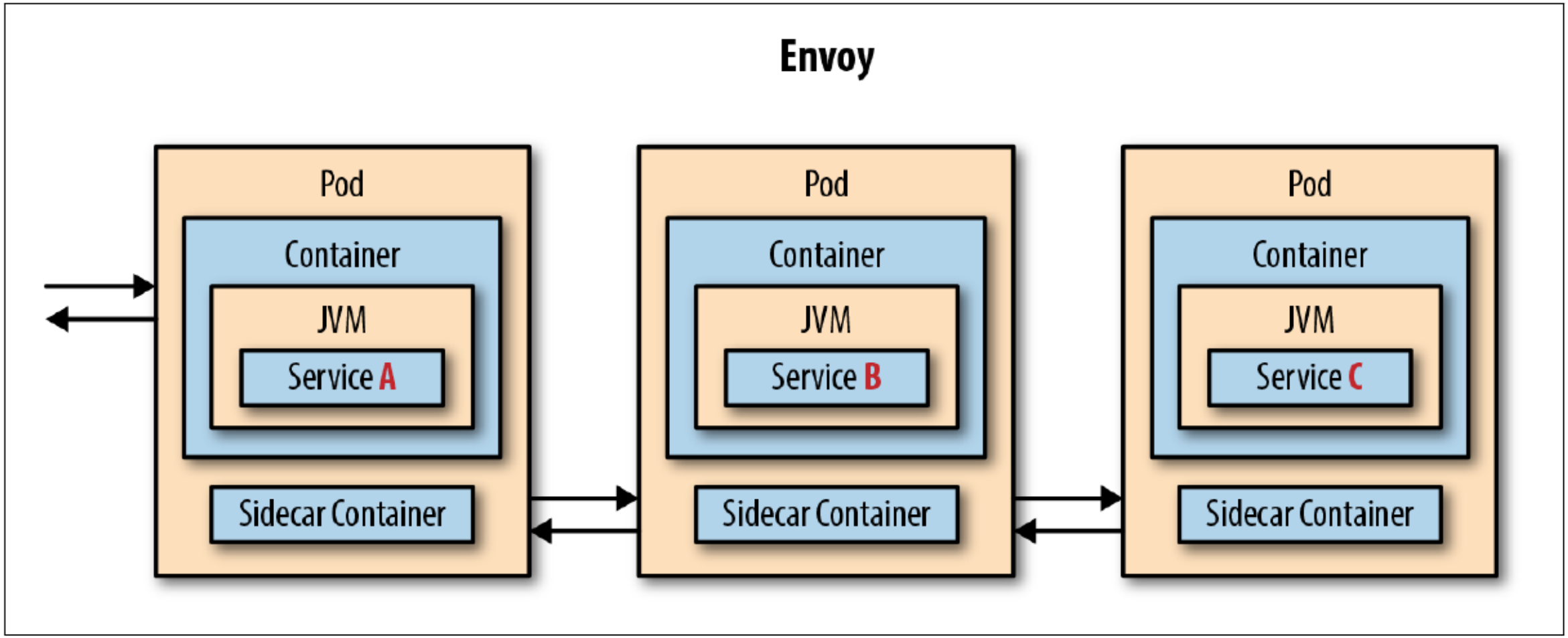


[Introducing Istio Service Mesh for Microservices | Red Hat Developer]

Istio Components – Data Plane

- Service Proxy
  - Auto retries, circuit breaker, service discovery, security
  - ***Envoy***
    - HTTP 1.1, HTTP2, gRPC load balancing
    - Collect request-level metrics, trace spans, provide for service discovery, inject faults …
- Side Car
  - Manually or automatically injected into pod
  - Intercepting all inbound (ingress) and outbound (egress) network traffic
  - Policies can be applied that reroute the traffic (in or out),
    - Access control lists (ACLs), rate limits, monitoring and tracing data (Mixer)
  - Chaos - network delays or HTTP errors

Introducing Istio Service Mesh for Microservices | Red Hat Developer


Envoy
Pod
Container
JVM
Service A
Sidecar Container
Pod
Container
JVM
Service B
Sidecar Container
Pod
Container
JVM
Service C
Sidecar Container

Istio Components – Control Plane

- Configuration and policy
- Usability of cluster with 1000s of services
- Primary Istio Control services
  - Pilot
  - Mixer
  - Citadel

Introducing Istio Service Mesh for Microservices | Red Hat Developer

Istio Components – Control Plane

- Pilot
  - Sidecar fleet management
  - Ensures independent microservices inside pods, have current view of overall topology and an up-to-date “routing table”
  - Service discovery as well as support for *VirtualService*
- Mixer
  - Service integration - Istio proxies deliver telemetry to Mixer
  - Maintains canonical model of usage & access policies for overall suite of microservices.
    - Create policies, apply rate-limiting rules, and capture custom metrics
  - Pluggable evolving backend architecture
- Citadel (formerly known as Istio CA or Auth)
  - Certificate signing, certificate issuance, and revocation/rotation
  - Issues X.509 certificates to all microservices using mutual Transport Layer Security (mTLS) between those services, encrypting all traffic
  - Identity built into the underlying deployment platform for policy enforcement

Introducing Istio Service Mesh for Microservices | Red Hat Developer

Traffic Control

Canary Deployment

- Targeted deployment
  - Platforms, Channels, User Persona, Data focus …
- Monitoring
  - Exceptions, errant behavior, SLA impact …
- Easy rollback
  - Fast deployment with minimal disruption
  - “Testing in production” – insurance against small QA gaps

Routing

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
        name: recommendation
        namespace: tutorial
spec:
        host: recommendation
        subsets:
        - labels:
                version: v1
        name: version-v1
        - labels:
                version: v2
        name: version-v2
```

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
        name: recommendation
        namespace: tutorial
spec:
        hosts:
        - recommendation
        http:
        - route:
                - destination:
                        host: recommendation
                        subset: version-v1
                weight: 100
```

Weighted Routing

- VirtualService
  - Configure a percentage of traffic and direct it to a specific version of the recommendation service
  - Selection of pods here similar to the Kubernetes selector model for matching based on labels.
- Applied to all inter-service communication within the mesh
  - Apply to services potentially deep within a service call graph.
  - Services in Kubernetes not part of the service mesh will not see rules
    - Will adhere to the default Kubernetes load-balancing rules

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
        name: recommendation
        namespace: tutorial
spec:
        hosts:
        - recommendation
        http:
        - route:
                - destination:
                host: recommendation
                        subset: version-v1
                                weight: 90
                                - destination:
                        host: recommendation
                        subset: version-v2
                weight: 10
```

Routing Based on Headers

- Routing based on request-level metadata
  - Matching predicates to set up specific route rules based on requests that match a specific set of criteria
  - Geography, mobile device, or browser

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  creationTimestamp: null
  name: recommendation
  namespace: tutorial
spec:
  hosts:
  - recommendation
  http:
  - match:
    - headers:
        baggage-user-agent:
          regex: .*Safari.*
    route:
    - destination:
        host: recommendation
        subset: version-v2
  - route:
    - destination:
        host: recommendation
        subset: version-v1
```

Dark Launch

- Invisible to customers
  - Mirror or duplicate current production to new version
  - Production quality measures and inputs into new service

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: recommendation
  namespace: tutorial
spec:
  hosts:
  - recommendation
  http:
  - route:
    - destination:
        host: recommendation
        subset: version-v1
    mirror:
      host: recommendation
      subset: version-v2
```

Egress

- Outbound traffic from cluster blocked by default
  - Zero-trust network architecture
  - Traditional perimeter-based security
  - Reverse shell threat blocked

```
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: httpbin-egress-rule
  namespace: tutorial
spec:
  hosts:
  - now.httpbin.org
  ports:
  - name: http-80
    number: 80
    protocol: http
```

Service Resiliency

Resiliency Components

- *Client-side load balancing*
  - Istio augments Kubernetes out-of-the-box load balancing
- *Timeout*
  - Wait only *N* seconds for a response and then give up
- *Retry*
  - If one pod returns an error (e.g., 503), retry for another pod.
- *Simple circuit breaker*
  - Instead of overwhelming the degraded service, open the circuit and reject further requests.
- *Pool ejection*
  - This provides auto removal of error-prone pods from the load balancing pool

Load Balancing

- *ROUND_ROBIN*
  - Distributes the load, in order, across the endpoints in the load-balancing pool.
- *RANDOM*
  - Distributes the load across the endpoints in the load-balancing pool but without any order.
- *LEAST_CONN*
  - Picks two random hosts from the load-balancing pool and determines which host has fewer outstanding requests

Timeout

- Avoid cascading failure
- Use with Retry

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
        creationTimestamp: null
        name: recommendation
        namespace: tutorial
spec:
        hosts:
                - recommendation
        http:
        - route:
                - destination:
                        host: recommendation
                timeout: 1.000s
```

Retry

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: recommendation
  namespace: tutorial
spec:
  hosts:
  - recommendation
  http:
  - route:
    - destination:
        host: recommendation
    retries:
      attempts: 3
      perTryTimeout: 2s
```

Circuit Breaker

- Load-balancing host level
  - Usual rules of load balancing

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  creationTimestamp: null
  name: recommendation
  namespace: tutorial
spec:
  host: recommendation
  subsets:
    - name: version-v1
      labels:
        version: v1
    - name: version-v2
      labels:
        version: v2
      trafficPolicy:
        connectionPool:
          http:
            http1MaxPendingRequests: 1
            maxRequestsPerConnection: 1
          tcp:
            maxConnections: 1
        outlierDetection:
          baseEjectionTime: 120.000s
          consecutiveErrors: 1
          interval: 1.000s
          maxEjectionPercent: 100
```

Circuit Breaker

- Connection-pool level
  - Delay traffic to erroring host in connection pool
  - Outlier detection

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  creationTimestamp: null
  name: recommendation
  namespace: tutorial
spec:
  host: recommendation
  subsets:
  - labels:
      version: v1
    name: version-v1
    trafficPolicy:
      connectionPool:
        http: {}
        tcp: {}
      loadBalancer:
        simple: RANDOM
      outlierDetection:
        baseEjectionTime: 15.000s
        consecutiveErrors: 1
        interval: 5.000s
        maxEjectionPercent: 100
  - labels:
      version: v2
    name: version-v2
    trafficPolicy:
      connectionPool:
        http: {}
        tcp: {}
      loadBalancer:
        simple: RANDOM
      outlierDetection:
        baseEjectionTime: 15.000s
        consecutiveErrors: 1
        interval: 5.000s
        maxEjectionPercent: 100
```

Combination Features

- Circuit Breaker + Pool Ejection + Retry

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  creationTimestamp: null
  name: recommendation
  namespace: tutorial
spec:
  hosts:
  - recommendation
  http:
  - retries:
      attempts: 3
      perTryTimeout: 4.000s
    route:
    - destination:
        host: recommendation
        subset: version-v1
      weight: 50
    - destination:
        host: recommendation
        subset: version-v2
      weight: 50
```

Chaos Testing

HTTP Errors

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: recommendation
  namespace: tutorial
spec:
  host: recommendation
  subsets:
  - labels:
      app: recommendation
    name: app-recommendation
```

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: recommendation
  namespace: tutorial
spec:
  hosts:
  - recommendation
  http:
  - fault:
      abort:
        httpStatus: 503
        percent: 50
   route:
   - destination:
       host: recommendation
       subset: app-recommendation
```

Delays

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  creationTimestamp: null
  name: recommendation
  namespace: tutorial
spec:
  hosts:
  - recommendation
  http:
  - fault:
      delay:
        fixedDelay: 7.000s
        percent: 50
    route:
    - destination:
        host: recommendation
        subset: app-recommendation
```

Observability

Tracing

- Istio *Jaeger* (orig. Uber)
  - Implements OpenTracing (vendor-neutral tracing)
  - **Span**
    - Logical unit of work – name, start-time, duration
    - Nested and ordered (causal relationships)
    - Example – RPC Call
  - **Trace**
    - Data/execution path through the system
    - Directed acrylic graph of spans

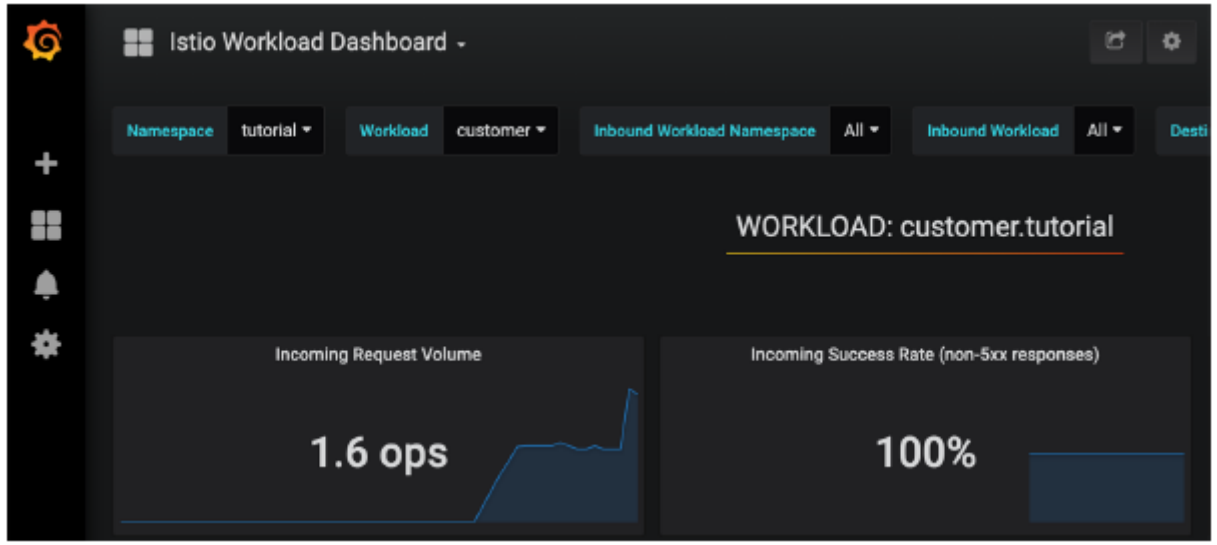

Jaeger UI
Search
Dependencies
About Jaeger
customer: customer.tutorial.svc.cluster.local:8080/*
View Options
Service & Operation
0ms
6.22ms
12.44ms
18.65ms
24.87ms
customer
customer
istio-policy
istio-mixer
customer
customer
customer
preference
preference
preference
preference
recommendation

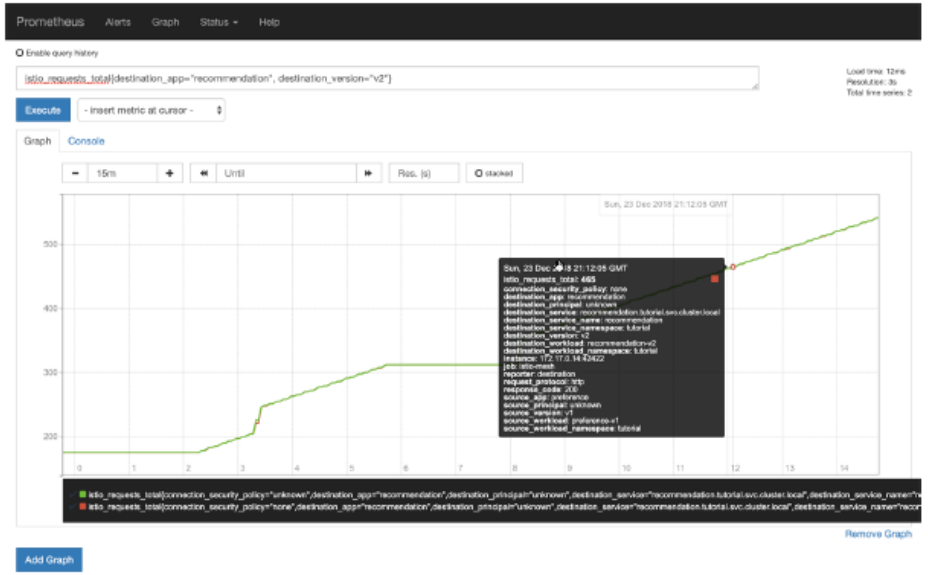
Tracing

- Programming logic must forward the OpenTracing headers from inbound to every outbound call
  - Microservice framework (e.g. Spring Boot) may have support for automatic header carry forward
- Jaeger traces can be configured via environment variables
- Istio captures or samples 100% of requests flowing through the mesh
  - PILOT_TRACE_SAMPLING
  - Can be set to desired value in production

Metrics

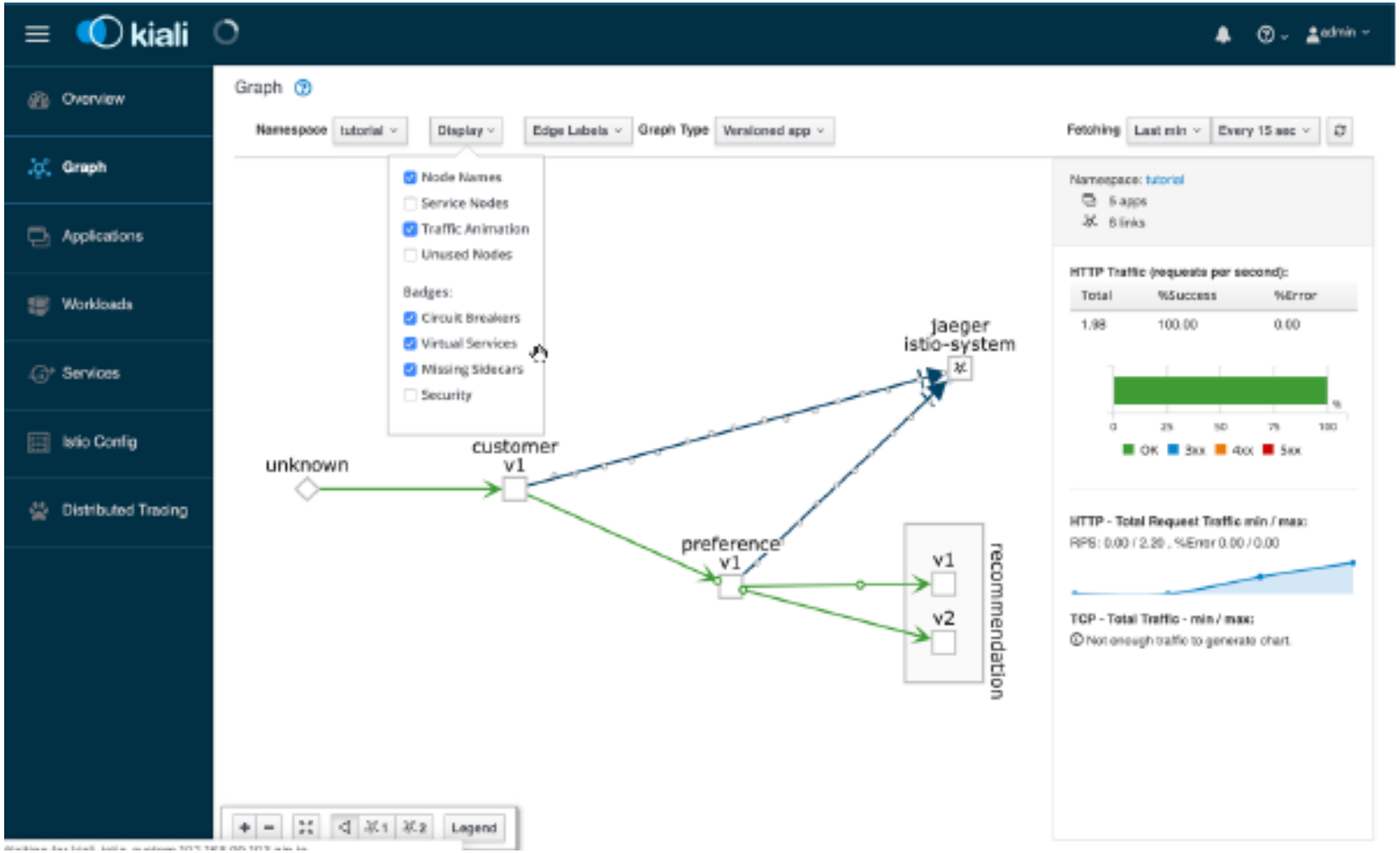
- Istio leverages Prometheus and Grafana





Service Graph

- Kiali service mesh management
  - Which microservices are part of a specific service mesh?
  - How are they connected?
  - How are they performing?
  - How can developers operate on them?
- Installed separately
  - Requires URLS for Jaeger and Grafana
  - Use *envsubst* tool from *gettext* package for Fedora


kiali
admin
Overview
Graph
Applications
Workloads
Services
Istio Config
Distributed Tracing
Graph
Namespace: tutorial
Display
Edge Labels
Graph Type: Versioned app
Fetching: Last min
Every 15 sec
Node Names
Service Nodes
Traffic Animation
Unused Nodes
Badges:
Circuit Breakers
Virtual Services
Missing Sidecars
Security
jaeger
istio-system
unknown
customer
v1
preference
v1
v1
v2
recommendation
Legend
Namespace: tutorial
6 apps
8 links
HTTP Traffic (requests per second):
Total
%Success
%Error
1.98
100.00
0.00
OK
3xx
4xx
5xx
HTTP - Total Request Traffic min / max:
RPS: 0.00 / 2.20 , %Error 0.00 / 0.00
TCP - Total Traffic - min / max:
Not enough traffic to generate chart

Security

mTLS

- Combination of Policy and DestinationRule objects
- Applies to all services in the namespace
  - Permissive vs. Strict
- Can use the command to verify:
  - istioctl authn tls-check

```
apiVersion: "authentication.istio.io/v1alpha1"
kind: "Policy"
metadata:
  name: "default"
  namespace: "tutorial"
spec:
  peers:
  - mtls: {}


 apiVersion: "networking.istio.io/v1alpha3"
 kind: "DestinationRule"
 metadata:
   name: "default"
   namespace: "tutorial"
 spec:
   host: "*.tutorial.svc.cluster.local"
   trafficPolicy:
     tls:
       mode: ISTIO_MUTUAL
```

mTLS

- OpenShift Routes
- DNS resolution nip.io
  - Exposes URL external to cluster
  - Supports monitoring, traffic management, policy

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: customer-gateway
  namespace: tutorial
spec:
  selector:
    istio: ingressgateway # use istio default controller
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: customer
  namespace: tutorial
spec:
  hosts:
  - "*"
  gateways:
  - customer-gateway
  http:
  - match:
    - uri:
        exact: /
    route:
    - destination:
        host: customer
        port:
          number: 8080
```

Access Control Mixer Policy

- Construct a series of rules that ensure the various microservices that make up your application follow an approved invocation path
- Istio has some additional objects or Kinds involved in this sort of access control:
  - denier, checknothing, and rule
  - Whitelist, blacklist

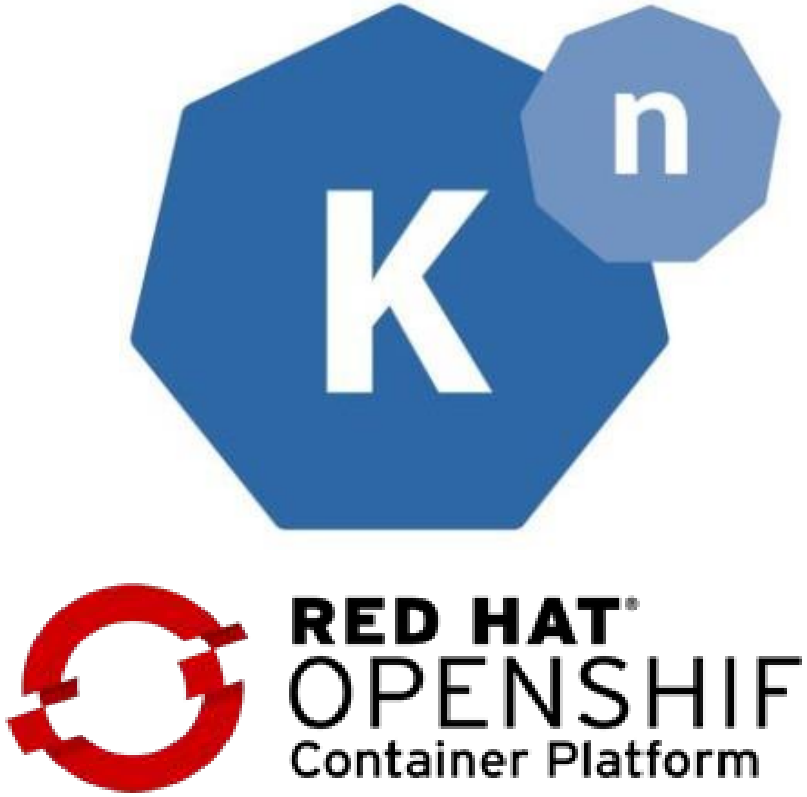
```
oc get rules
NAME                                AGE
no-customer-to-recommendation       3m
no-preference-to-customer           3m
no-recommendation-to-customer       3m
no-recommendation-to-preference     3m

oc describe rule no-preference-to-customer
...
Spec:
  Actions:
    Handler:  do-not-pass-go.denier
    Instances:
      just-stop.checknothing
  Match:  source.labels["app"]=="preference" &&
   destination.labels["app"] == "customer"
Events:   <none>
```

RBAC

- Where *mode* can be:
- OFF: Istio authorization is disabled.
- ON: Istio authorization is enabled for all services in the mesh.
- ON_WITH_INCLUSION: Enabled only for services and namespaces specified in the inclusion field.
- ON_WITH_EXCLUSION: Enabled for all services in the mesh except the services and namespaces specified in the exclusion field.

```
apiVersion: "rbac.istio.io/v1alpha1"
kind: RbacConfig
metadata:
  name: default
spec:
  mode: 'ON_WITH_INCLUSION'
  inclusion:
    namespaces: ["tutorial"]
```

```
apiVersion: "rbac.istio.io/v1alpha1"
kind: ServiceRole
metadata:
  name: service-viewer
  namespace: tutorial
spec:
  rules:
  - services: ["*"]
    methods: ["GET"]
    constraints:
    - key: "destination.labels[app]"
      values: ["customer", "recommendation", "preference"]
---
apiVersion: "rbac.istio.io/v1alpha1"
kind: ServiceRoleBinding
metadata:
  name: bind-service-viewer
  namespace: tutorial
spec:
  subjects:
  - user: "*"
  roleRef:
    kind: ServiceRole
    name: "service-viewer"
```



Knative on OpenShift

Presenter: Veer Muchandi
Title: Principal Architect - Container Solutions
Social Handle: @VeerMuchandi
Blogs: https://blog.openshift.com/author/veermuchandi/

Serverless Model

Serverless computing is a cloud-computing execution model in which the cloud provider acts as the server, dynamically managing the allocation of machine resources. Pricing is based on the actual amount of resources consumed by an application, rather than on pre-purchased units of capacity.[1] It is a form of utility computing.

Knative

Kubernetes-based platform to build, deploy, and manage modern container-based serverless workloads

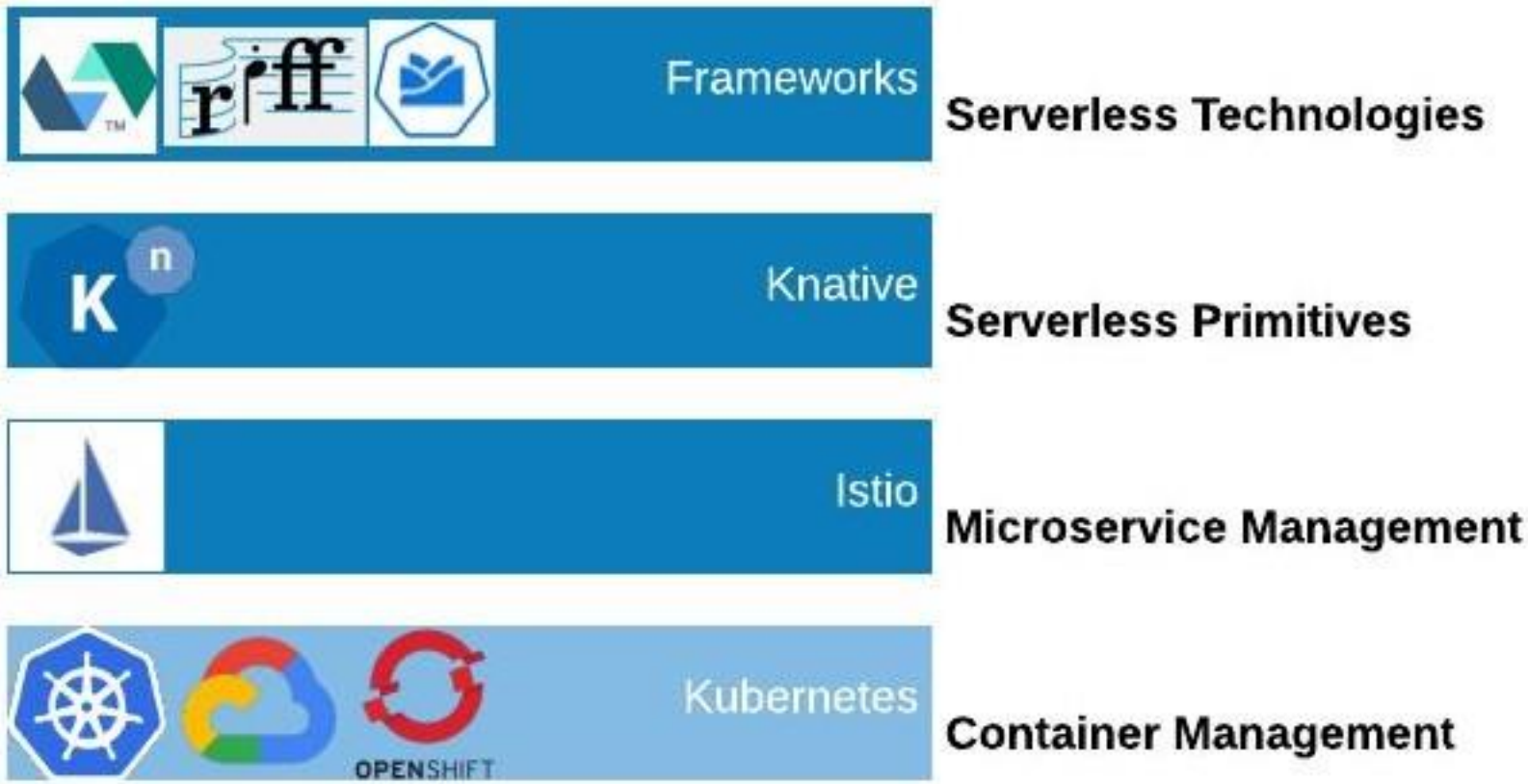
Builds on Kubernetes and Istio

Announced by Google at GoogleNext in July 2018

Contributions from Red Hat, IBM, SAP, Pivotal

Collaboration with open-source Function-as-a-Service framework communities like OpenWhisk, Riff, and Kyma

Knative for Serverless



Knative

Uses K8S CRDs to define a new set of APIs

Provides a set of building blocks that enable modern, source-centric and container-based development workloads on Kubernetes:

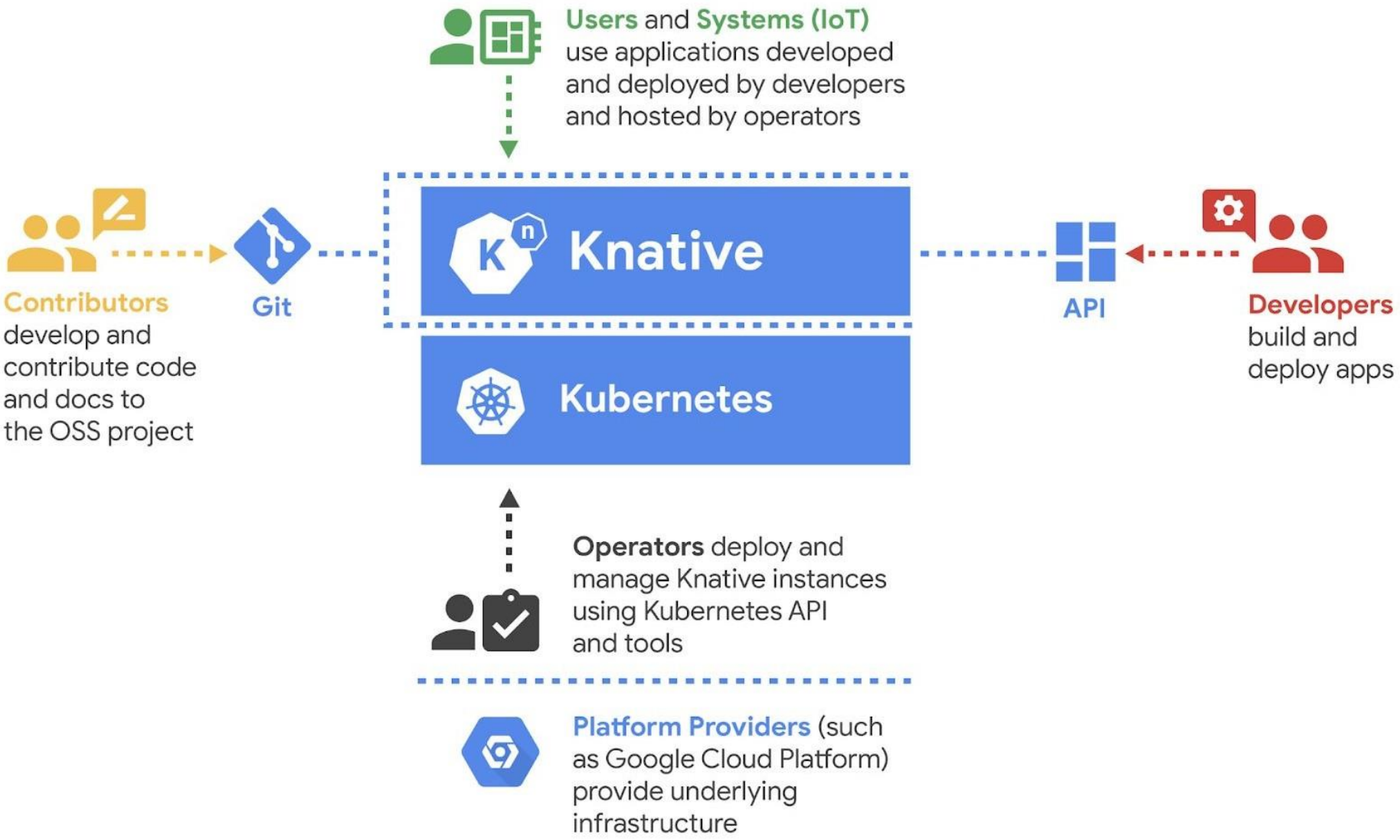
- Build—Source-to-container build orchestration
- Serving—Request-driven compute model, scale to zero, autoscaling, routing and managing traffic
- Eventing—Universal subscription, binding services to event ecosystems, delivery and management of events

KNative

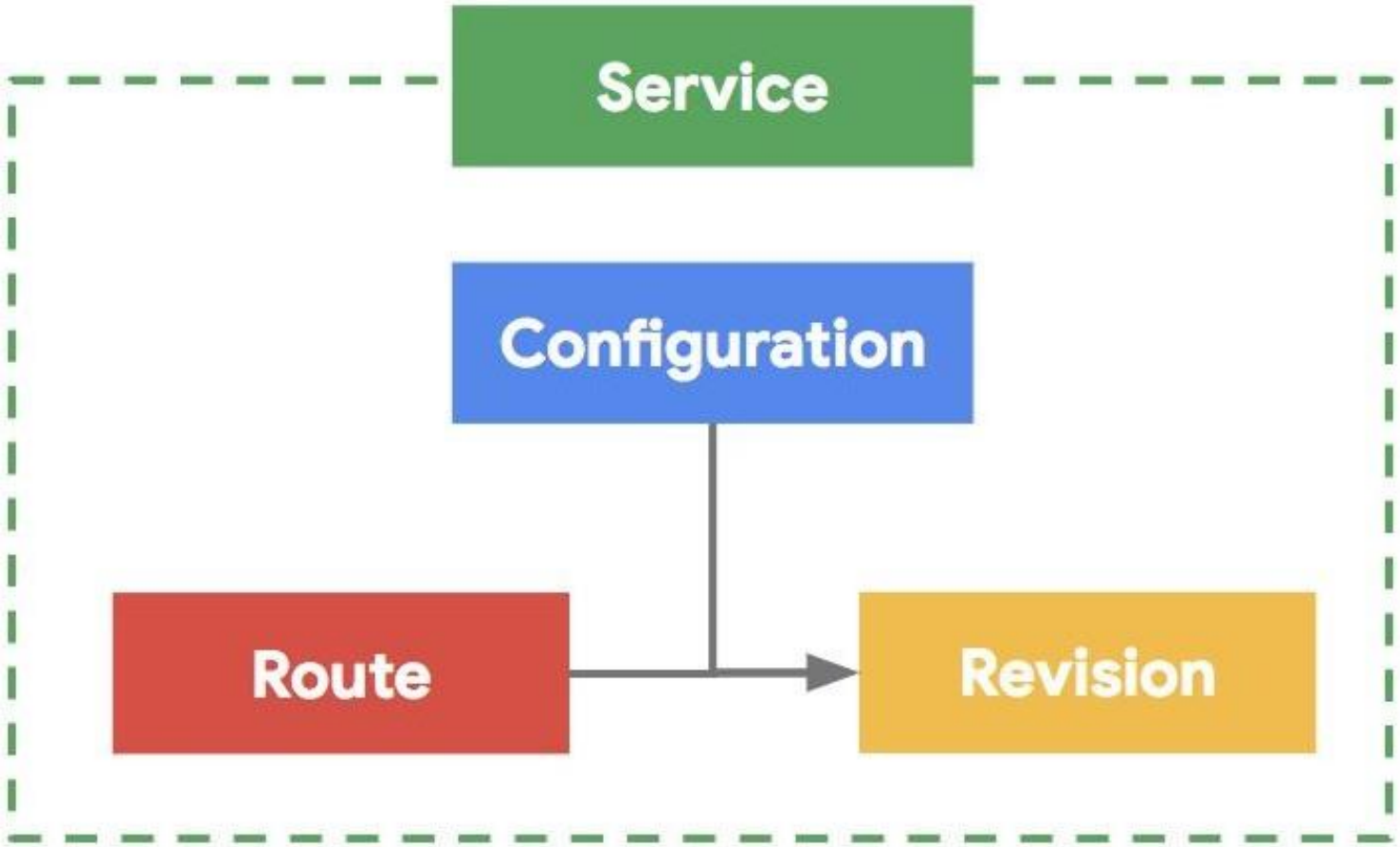
Knative components focus on solving many mundane tasks such as:

- Deploying a container
- Orchestrating source-to-URL workflows on Kubernetes
- Routing and managing traffic with blue/green deployment
- Automatic scaling and sizing workloads based on demand
- Binding running services to eventing ecosystems

Knative Actors



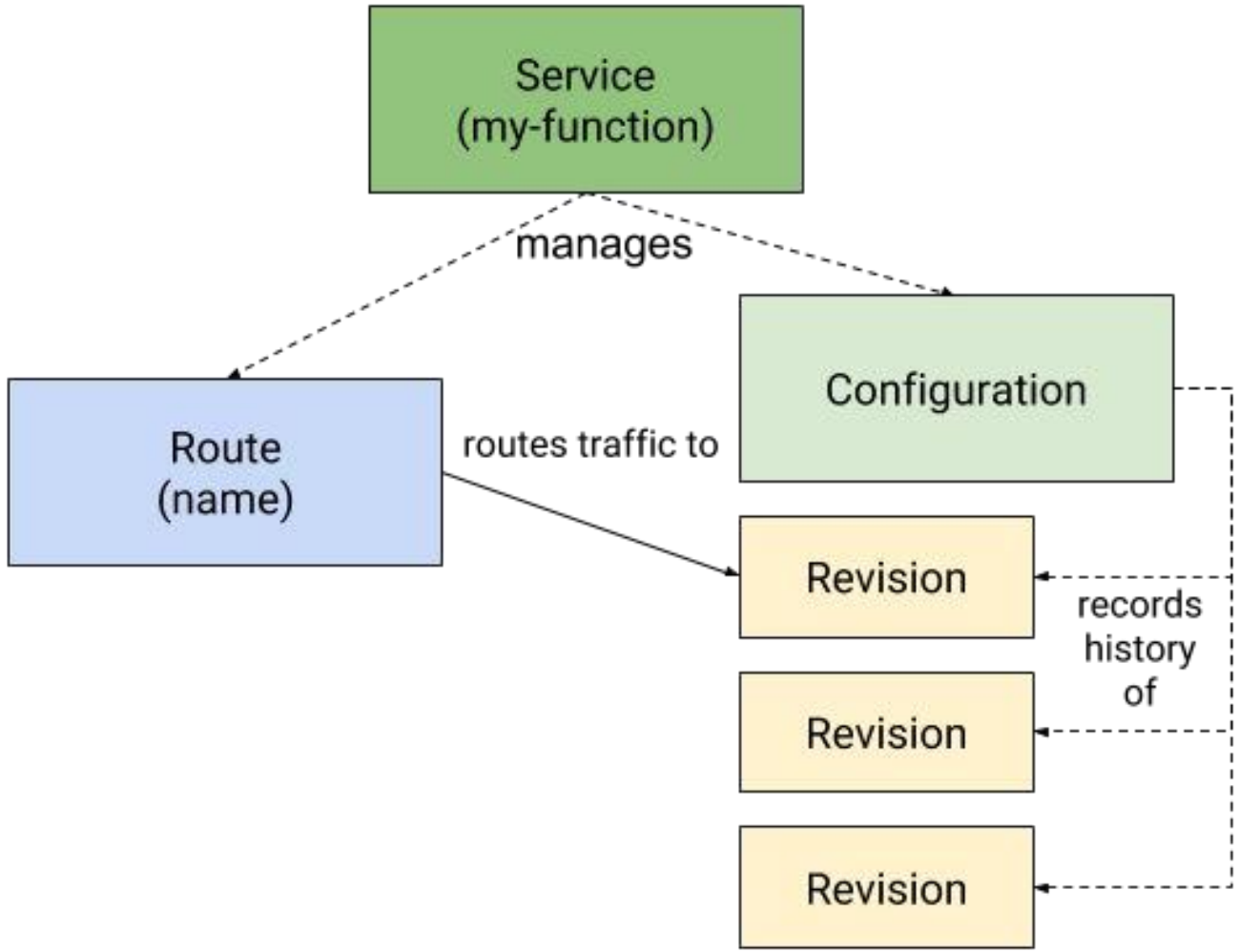
KNative Serving Primitives

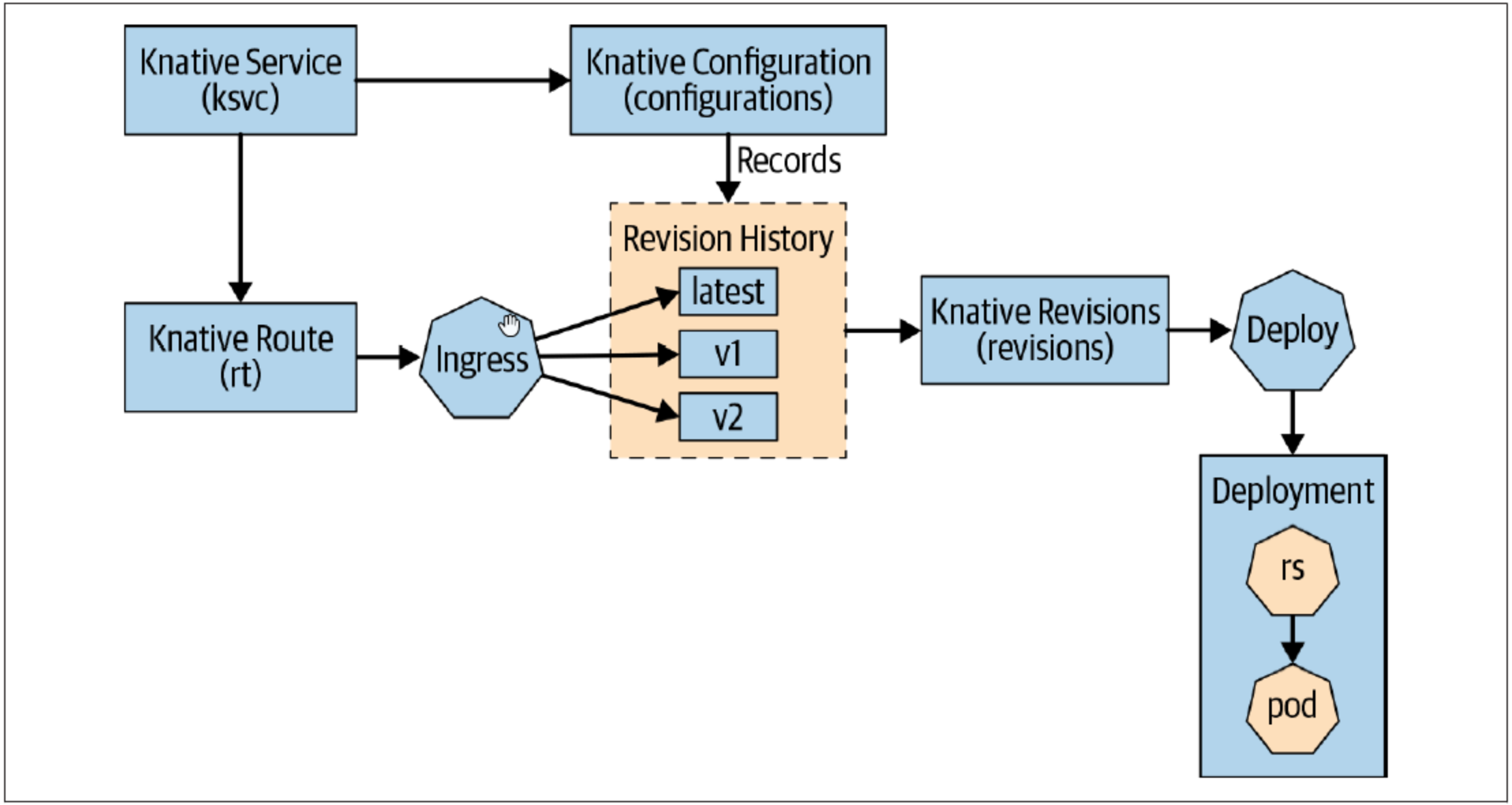


Service—is the combined lite version of all the objects below to enable simple use cases

- **Configuration**—is the desired state for your service, both code and configuration
- **Revision**—represents an immutable point-in-time snapshot of your code and configuration
- **Route**—assigns traffic to a revision or revisions of your service

Serving Resources





Knative Cookbook | Red Hat Developer

[Knative Cookbook | Red Hat Developer]

```yaml
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: greeter ❶
spec:
  template:
    metadata:
      name: greeter-v1 ❷
    spec:
      containers:
      - image: quay.io/rhdevelopers/knative-tutorial-greeter:quarkus
        livenessProbe:
          httpGet:
            path: /healthz
        readinessProbe:
          httpGet:
            path: /healthz
```

```yaml
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: foo
spec:
  template:
    # removed for brevity
  traffic: ❶
  - tag: v1 ❷
    revisionName: foo-v1 ❸
    percent: 50 ❹
  - tag: v2
    revisionName: foo-v2
    percent: 50
```

Route

`CRD: route.serving.knative.dev`

Network endpoint for Service (combo of software and configuration revisions)

Long lived, stable, named HTTP Endpoint backed by 1..* revisions

Namespace can have multiple routes

Default to route traffic to latest **Revision** created by a **Configuration**

Supports splitting traffic by percentage to different revisions, multiple configurations per route, n-way traffic split etc

Optionally assign addressable subdomains to any or all revisions.

Revision

`CRD: revision.serving.knative.dev`

Immutable snapshot of code and configuration

Created by updates to **Configuration**

References a container image and a build that created it from source

Revisions addressable via **Route** utilize resources as per the load they take

Revisions not addressable via Route are retired and underlying K8S resources cleaned up

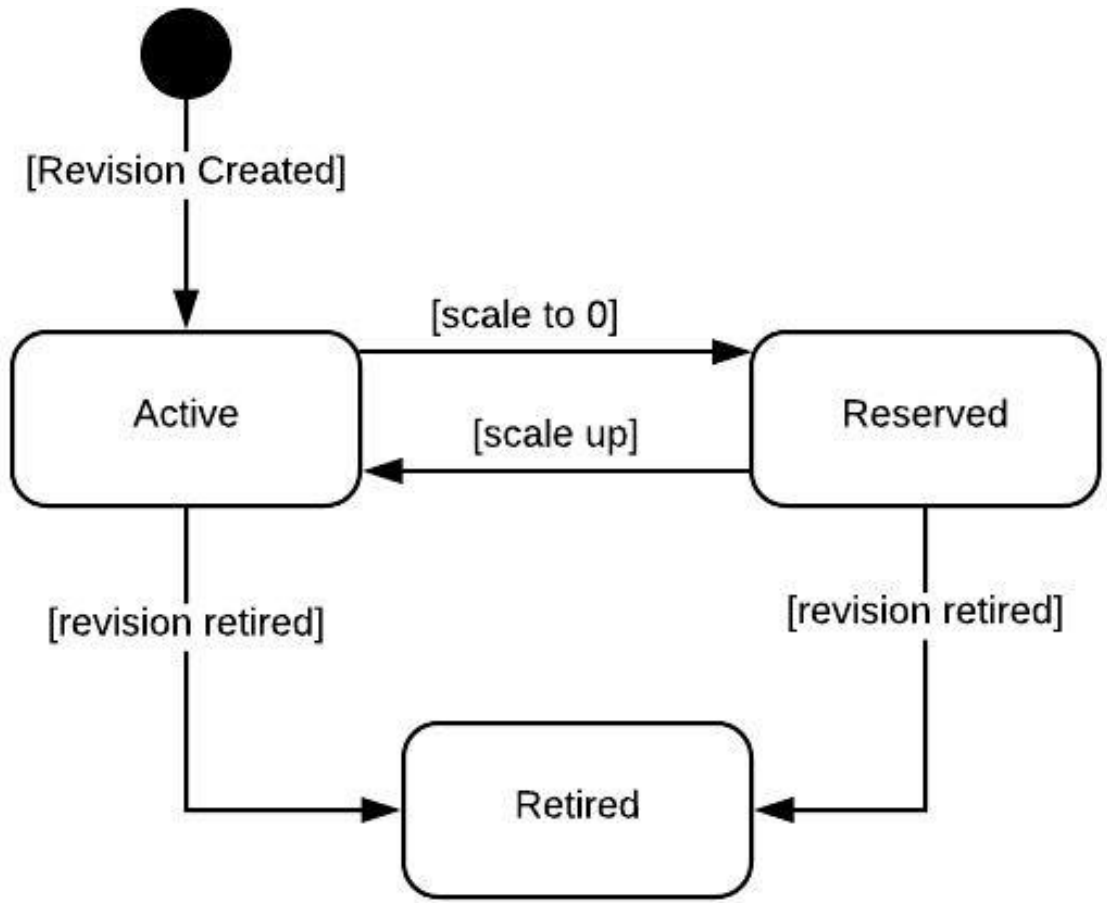
Configuration

`CRD: configuration.serving.knative.dev`

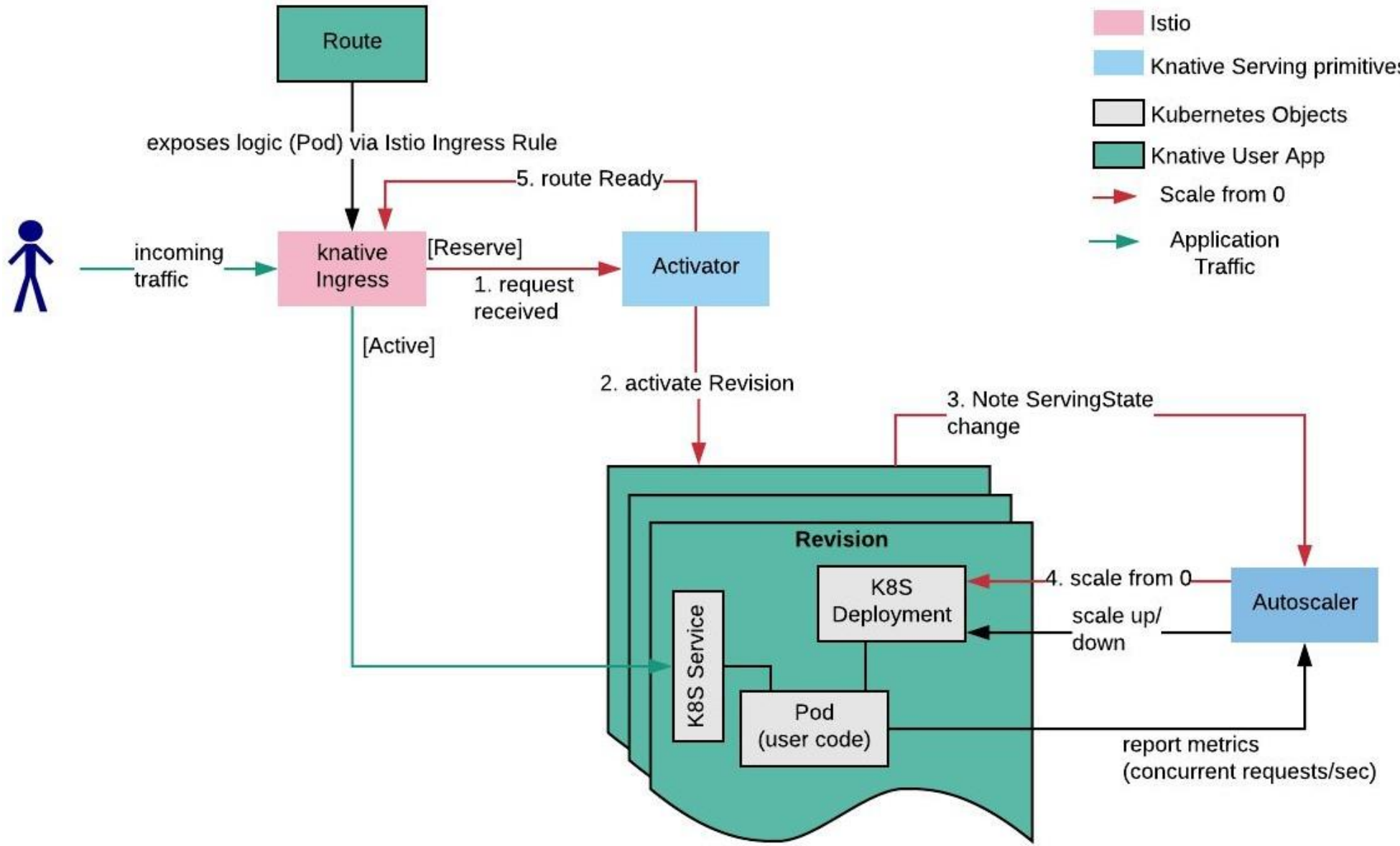
Describes desired latest **Revision** state. Tracks the status of revisions as the desired state is updated

Might reference a **Build** (instructions for transforming source to container) or reference a container image,

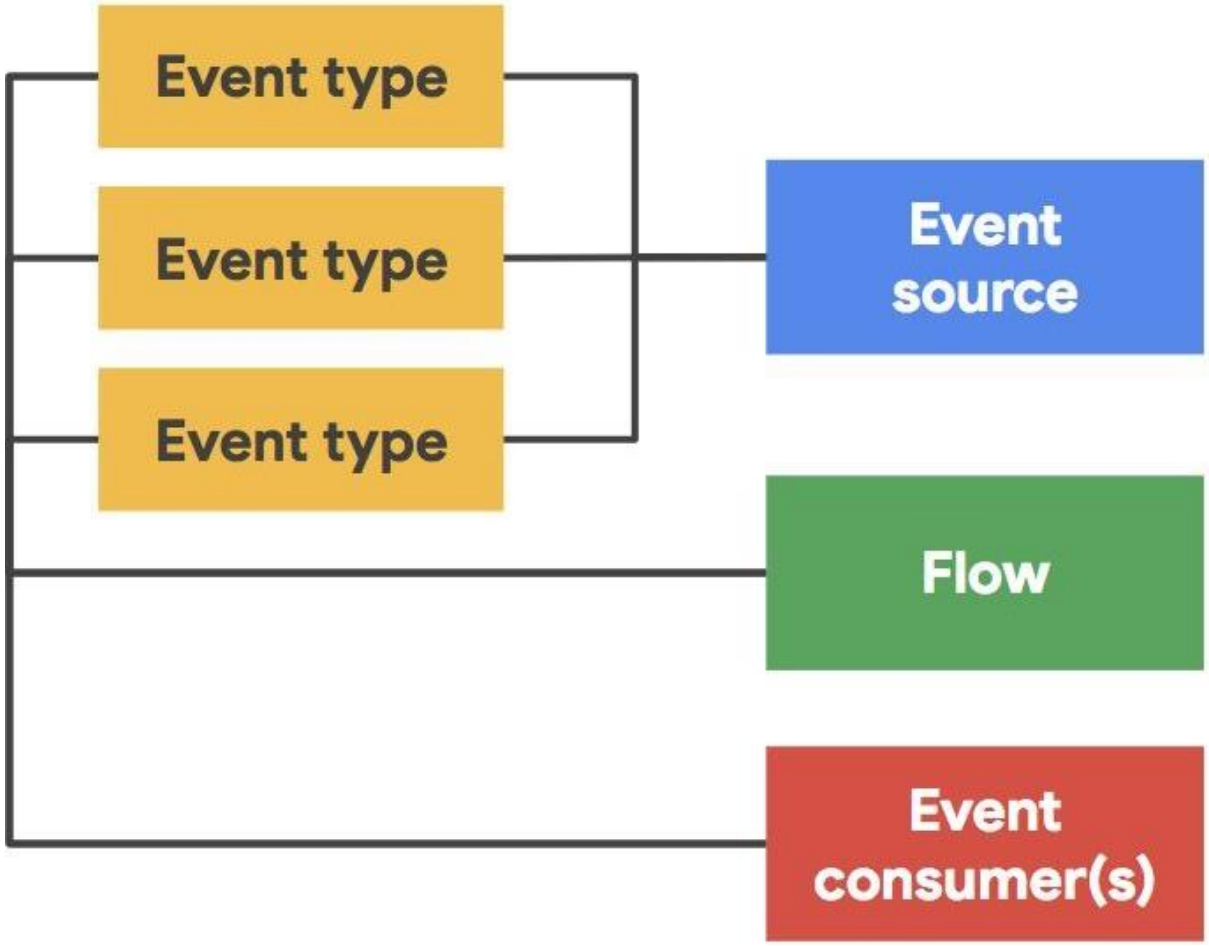
Revision Status



Knative Serving



KNative Eventing Objects

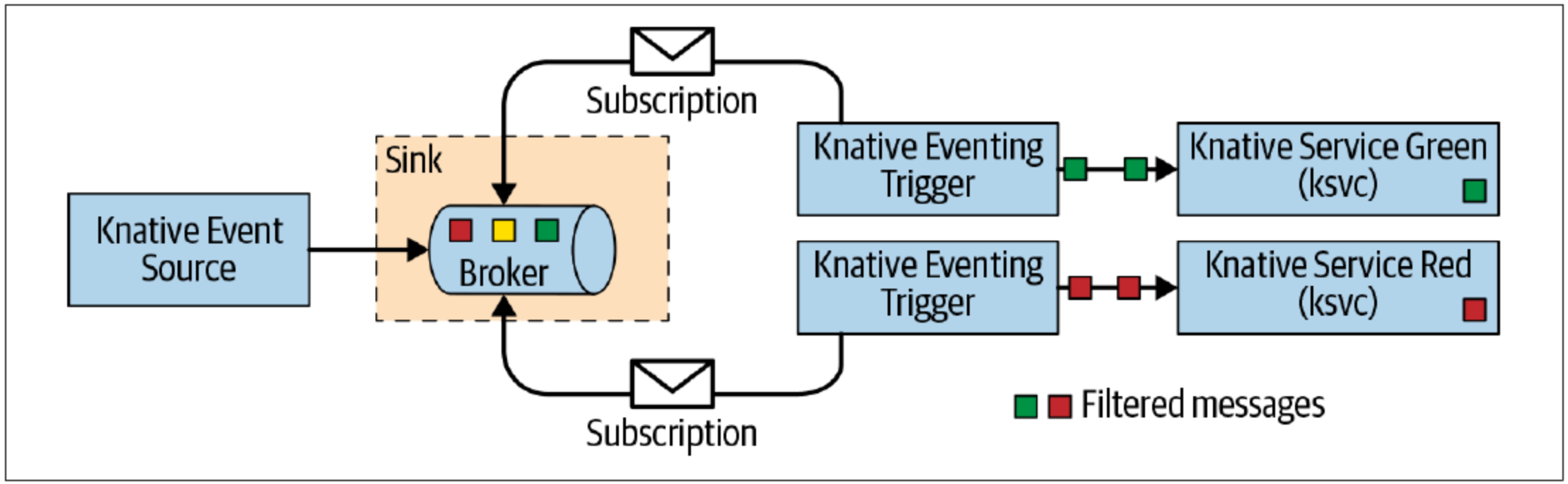


Event Sources—represents the producer of events (e.g. GitHub)

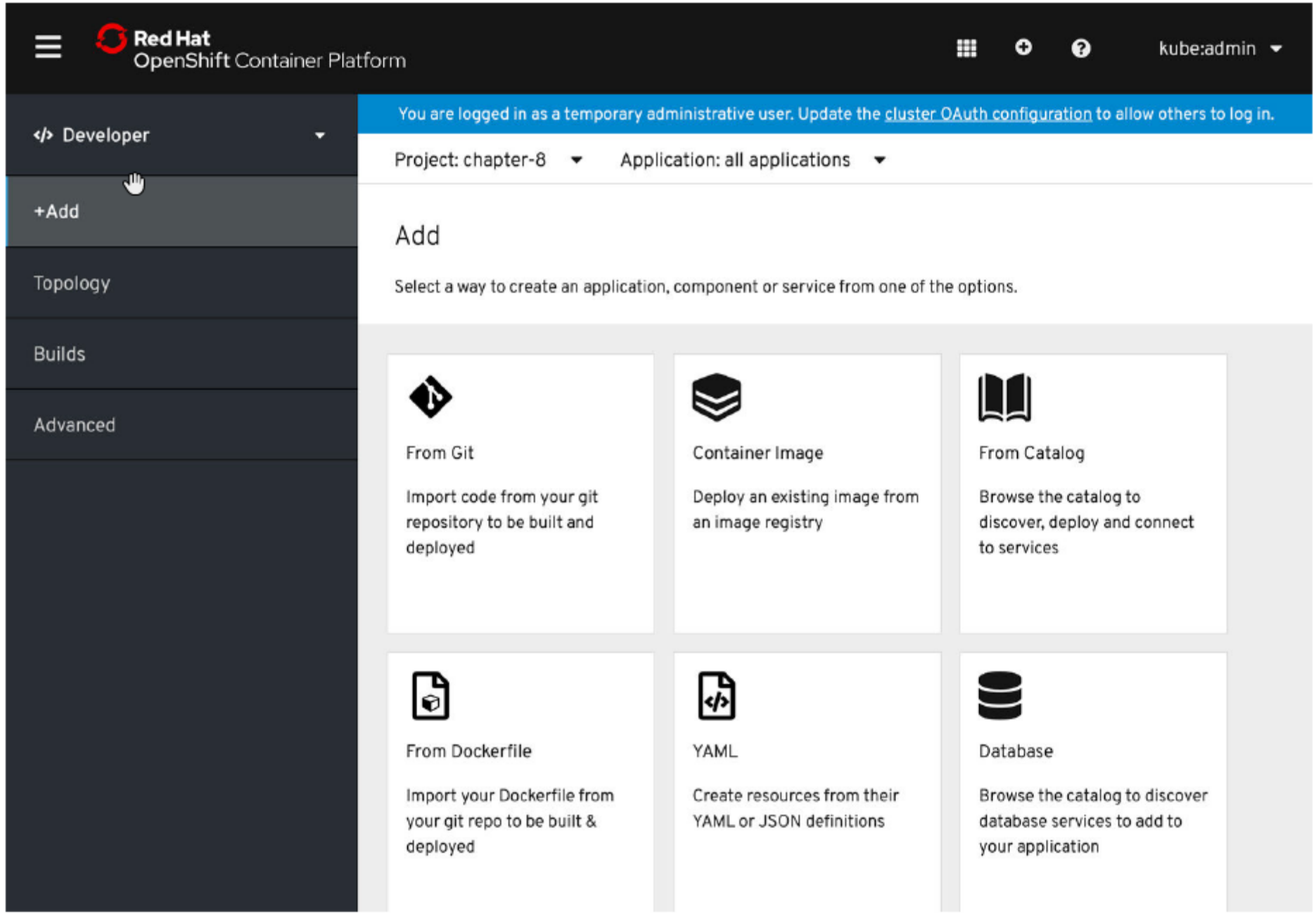
Event Types—describes the types of events supported by the different event sources (e.g. Webhook for the above mentioned GitHub source)

Event Consumers—represents the target of your action (i.e. any route defined by Knative)

Event Feeds—is the binding or configuration connecting the event types to actions



[Knative Cookbook | Red Hat Developer]


Red Hat
OpenShift Container Platform
kube:admin
Developer
You are logged in as a temporary administrative user. Update the cluster OAuth configuration to allow others to log in.
Project: chapter-8
Application: all applications
+Add
Topology
Builds
Advanced
Add
Select a way to create an application, component or service from one of the options.
From Git
Import code from your git repository to be built and deployed
Container Image
Deploy an existing image from an image registry
From Catalog
Browse the catalog to discover, deploy and connect to services
From Dockerfile
Import your Dockerfile from your git repo to be built & deployed
YAML
Create resources from their YAML or JSON definitions
Database
Browse the catalog to discover database services to add to your application