Building Event-Driven DevOps Workflows for Kubernetes with Knative and Tekton

Evan Anderson and Omer Bensaadon
(Software Engineer) (Product Manager)

1 Sep 2021

vmware®

Agenda

10:00	Welcome and Lab Setup
10:05	What is Knative Serving?
10:15	(Lab) Hands on with Knative
10:25	Visualizing Kubernetes with Octant
10:30	What is Tekton?
10:40	(Lab) Building with Tekton
10:55	Q & A / Break
11:00	What is Knative Eventing?
11:10	(Lab) Knative Eventing
11:20	Wrapup / Final Q & A

Welcome and Lab Setup

Who we are

Evan Anderson

Software Engineer at VMware Tanzu

Knative Technical Oversight Committee member. Previously worked on Google App Engine and Compute Engine.

Omer Bensaadon

Product Manager at VMware Tanzu

Knative Docs & User Experience WG lead.

Lab Setup

Strigo provides a VM for running all the lab content. We've pre-loaded the lab VMs with all the following:

```
Kind:     Run Kubernetes on your local machine
Knative:  Serverless framework for Kubernetes
Octant:   Local UI for Kubernetes
Tekton:   Build pipelines for Kubernetes
```

It may take a minute or two for your VM to start up and get all the components running. You can check that Kubernetes is running with the following command:

```
$ kubectl get nodes
```

A quick Kubernetes note

The Kubernetes `kubectl` command is very handy for examining your cluster. Later on, we'll also introduce a web UI, Octant, but here are a few `kubectl` commands you may find useful:

List all resources of a type, or get pretty-printed details on a specific object of a type:

```
$ kubectl get <type>
$ kubectl describe <type> <name>
```

You can also get a very detailed view of the Kubernetes manifest for any object using the `kubectl get` command with YAML output:

```
$ kubectl get <type> <name> -o yaml
```

What is Knative Serving?

Serverless on Kubernetes

What is Serverless?

Serverless is a development model where the platform scales and manages itself automatically, so you can spend more time focusing on application code.

Why on Kubernetes?

Kubernetes provides a common infrastructure layer that works in many different environments… from cloud to datacenter to laptop to edge.

What does this mean for me?

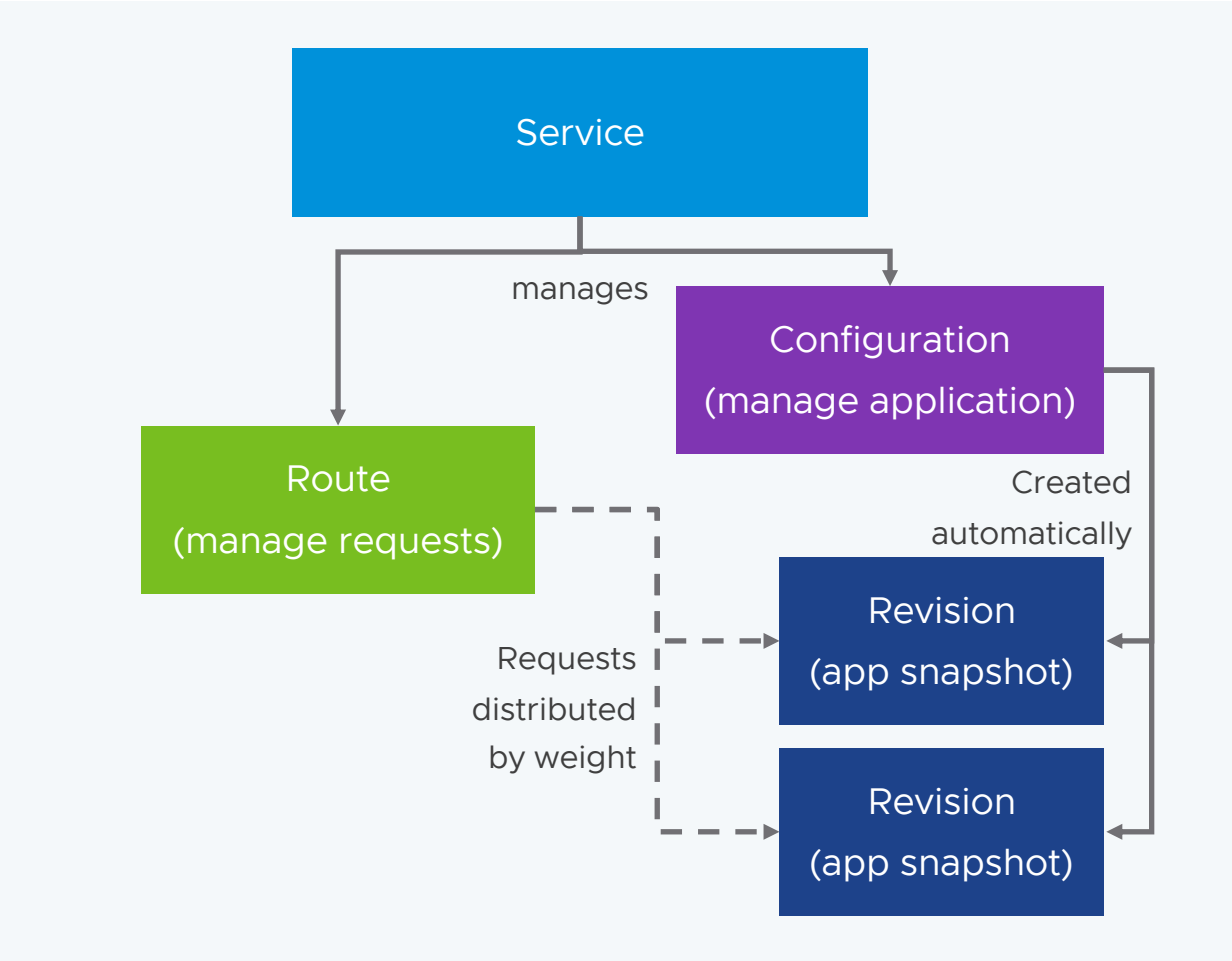
Serverless on Kubernetes means not needing to be a Kubernetes expert and less time spent managing builds, scaling, and resource provisioning.

Knative Serving: Abstractions and Value

Knative Serving provides a Service resource which enables the following features:

- Automatic request-based scaling
- Scale to zero when no requests
- Per-instance request concurrency limits
- Automatic DNS and TLS management
- Canarying and percentage traffic rollout
- Rollback and automatic Revision GC

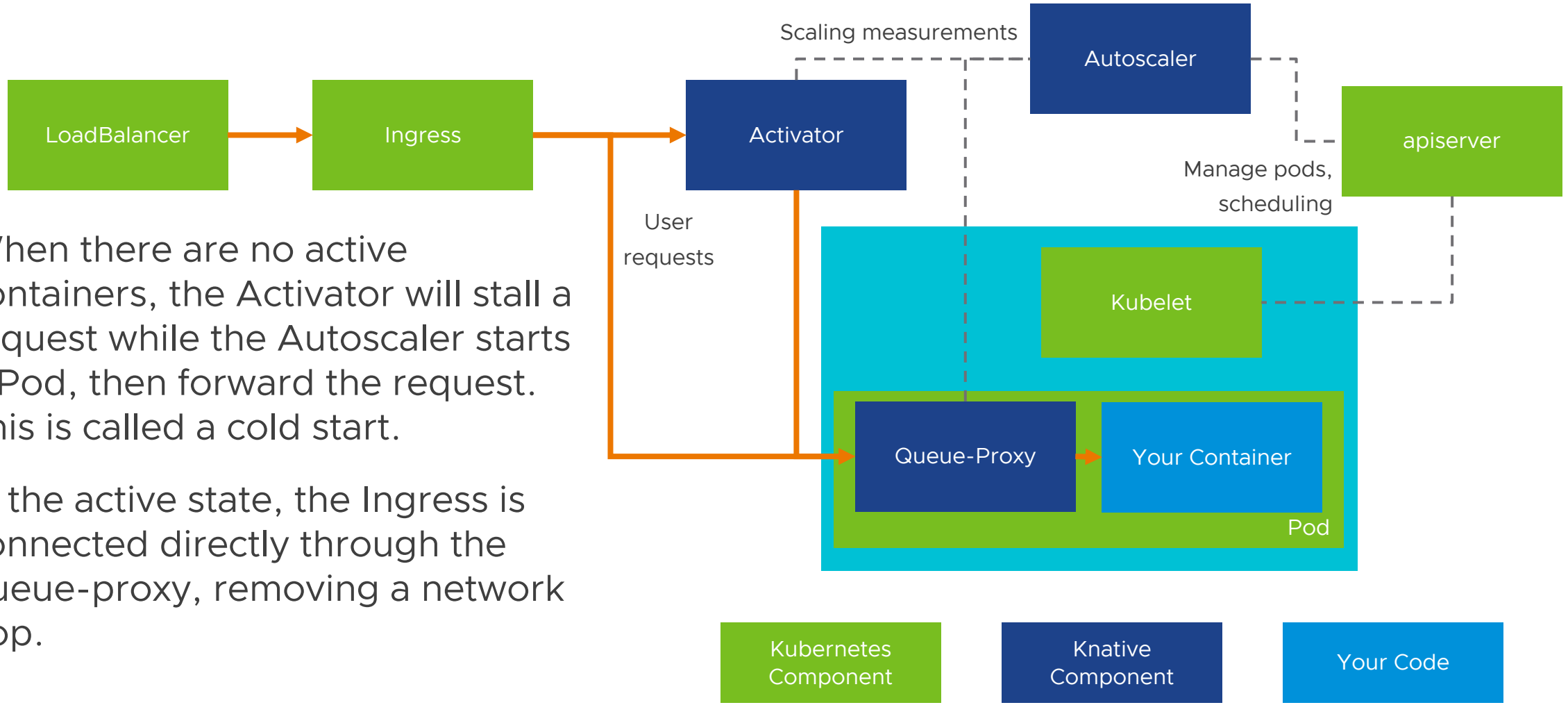
When starting, Service is the only thing you need. The other resources will be there when you need them later.



Knative Serving: The Life of a Request

When there are no active containers, the Activator will stall a request while the Autoscaler starts a Pod, then forward the request. This is called a cold start.

In the active state, the Ingress is connected directly through the queue-proxy, removing a network hop.



Deploying with Knative

We're going to start by deploying using the kn command-line tool.

```
$ kn service create hello --image gcr.io/knative-samples/helloworld-go
```

You can also use `kubectl` to deploy a manifest. We won't use that in the lab, but it's a good practice when managing production configurations using GitOps.

If you've got an existing pipeline using kn, you can export the configuration to a Kubernetes manifest using the `--target` flag:

```
$ kn service create hello --image gcr.io/knative-samples/helloworld-go \
    --target=manifest.yaml
```

Calling a Knative Service

When the deployment completes, you should see

```
...
  9.697s Ready to serve.

Service 'hello' updated to latest revision 'hello-00001' is available at URL:
http://hello.default.127.0.0.1.nip.io
```

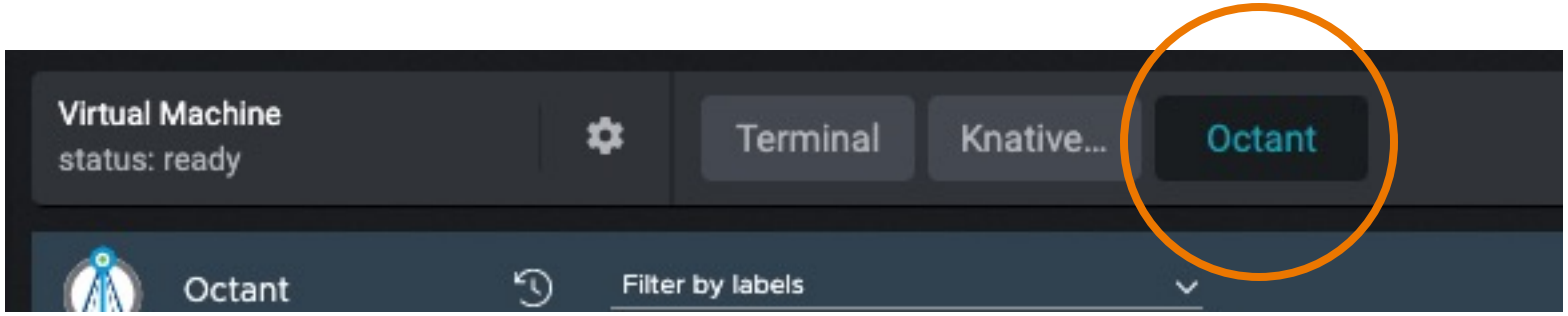
You can now use curl to make a request to the application:

```
$ curl http://hello.default.127.0.0.1.nip.io/
```

Visualizing Kubernetes with Octant

Octant: a dashboard for Kubernetes

We've started Octant running on the VM; you should be able to use the "Octant" tab to browse your cluster.



We've automatically loaded the Knative Plugin for Octant, and started Octant with the following command:

```
$ octant --disable-open-browser --listener-addr 0.0.0.0:8081
```

Navigating Octant

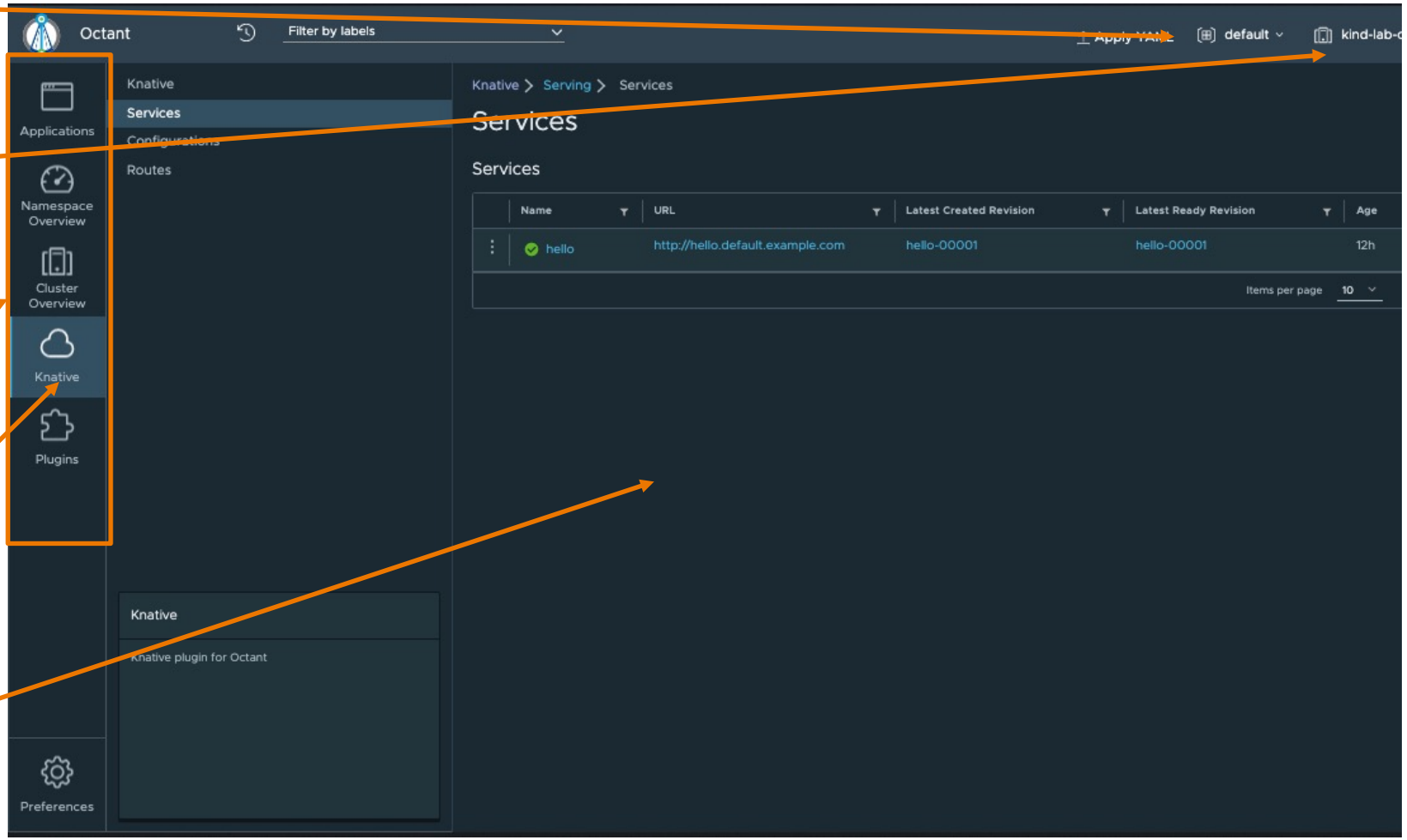
Namespace selector

Cluster / context selector

Application Views (pluggable)

Knative plugin (navigate here)

View content (explore this area)



What Makes Octant Different?

Unlike many cloud dashboards, Octant is a local tool that runs on your desktop:

- It can read your local KUBECONFIG (just like the `kubectl` binary)
- It uses your credentials, since it runs on your desktop
- It can reach things your desktop can reach (including local Kind clusters)
- You can write and install your own plugins locally
- It knows how to proxy logs and ports for debugging
- It's open source

What is Tekton?

CI/CD on Kubernetes

Tekton is a Kubernetes-based CI/CD system supporting build, test, and deploy pipelines.

Why run CI/CD on Kubernetes?

A few good reasons:

- Secure your supply chain with reproducible builds not tied to your desktop environment
- Normalize your credential management for build and test (docker credentials, etc)
- Standardize and share build configurations (GitOps for pipelines)
- Utilize build resources more effectively across teams – share a cluster without sharing individual build configs
- Get more resources for your build without overloading your laptop

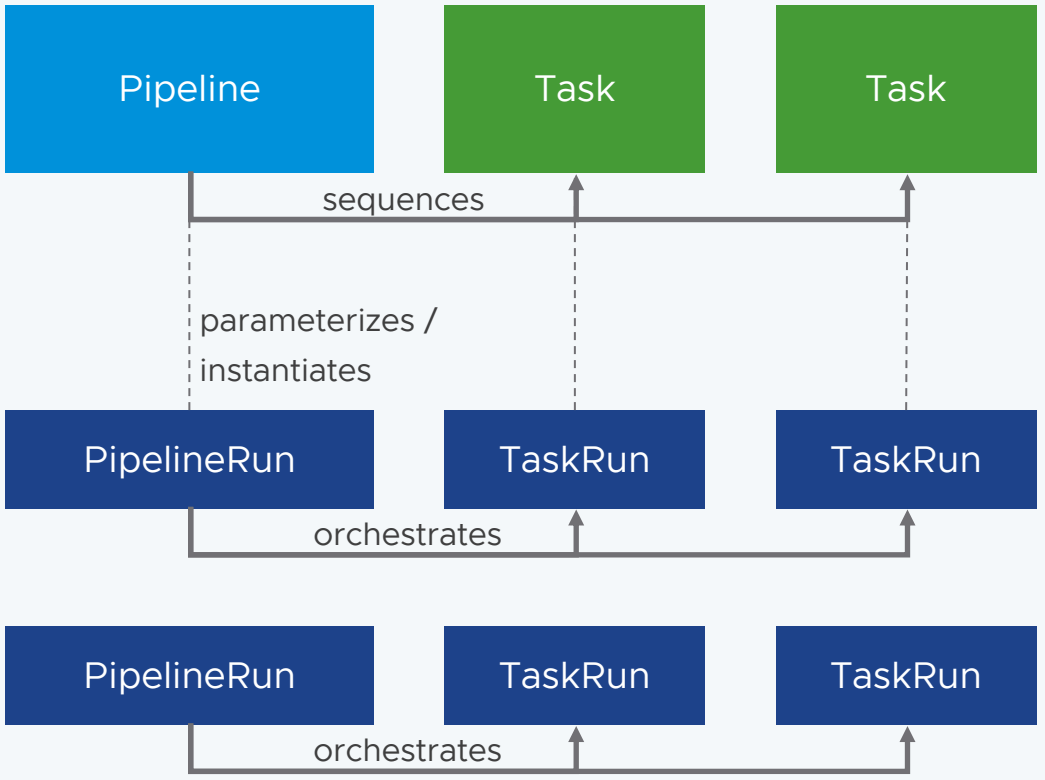
Tekton: Abstractions and Value

Tasks are a sequence of commands (containers) run in a single Pod with a shared environment.

Pipelines are graph of Tasks to be completed with explicit inputs and outputs. Pipelines allow composing smaller Tasks in different ways to achieve an outcome.

__Run resources represent a single run of a Task or a Pipeline.

Triggers (not covered here) allow automatically creating Run resources when certain events are received.



What to build?

I've written a couple Spring Java applications which should be checked out in your home directory in the `springone-2021-knative-tekton` directory.

> `apitizer` is a simple app which responds to requests with a dad joke. It also emits a CloudEvent, which we'll get to in part 3.
>
> `dessert` is an even simpler app which consumes CloudEvents and prints them out as JSON, so we can see the events.

Note: the lab machines do not have `java` or `javac` installed! We're going to be building using Tekton on the Kubernetes cluster!

Since this is a local cluster, I've also set up a local Docker registry so we don't have to worry about credentials. It's accessible at `localhost:5000`, so we'll see that a few times.

Building and Deploying with Tekton

Run these commands in the `apitizer` directory!

For this lab, we're going to use `tar` and ConfigMaps to copy data into our cluster. In a real environment, you'd use hosted Git repositories with credentials.

If you make changes, you'll need to delete the ConfigMap before running create again.

```
$ kubectl create configmap apitizer-src --from-file src.tar=<(tar -cvf - .)
```

The parameters to start a Tekton pipeline are just complicated enough that we're going to use a manifest to deploy them:

```
$ kubectl apply -f pipelinerun.yaml
```

Watching a Tekton Build

Once your PipelineRun has been started, you can use the tkn command to track the progress. Here are a few popular commands:

To see the state of your running builds:

```
$ tkn pipelinerun list
$ tkn taskrun list
```

To fetch logs from a build:

```
$ tkn pipeline logs -f buildpack-java
```

Break / Q & A

What is Knative Eventing?

Event Delivery on Kubernetes

What is event delivery, and why do I want it?

Events are asynchronous notifications of state changes sent from inside a process to outside observers. You can think of it as publishing the app's internal monologue.

Event delivery enables routing events from publishers to interested observers, where the observers don't need to coordinate with the publisher. Designing systems in this way is part of event-driven architecture, where work is done asynchronously based on reactions to external events.

Why use Kubernetes?

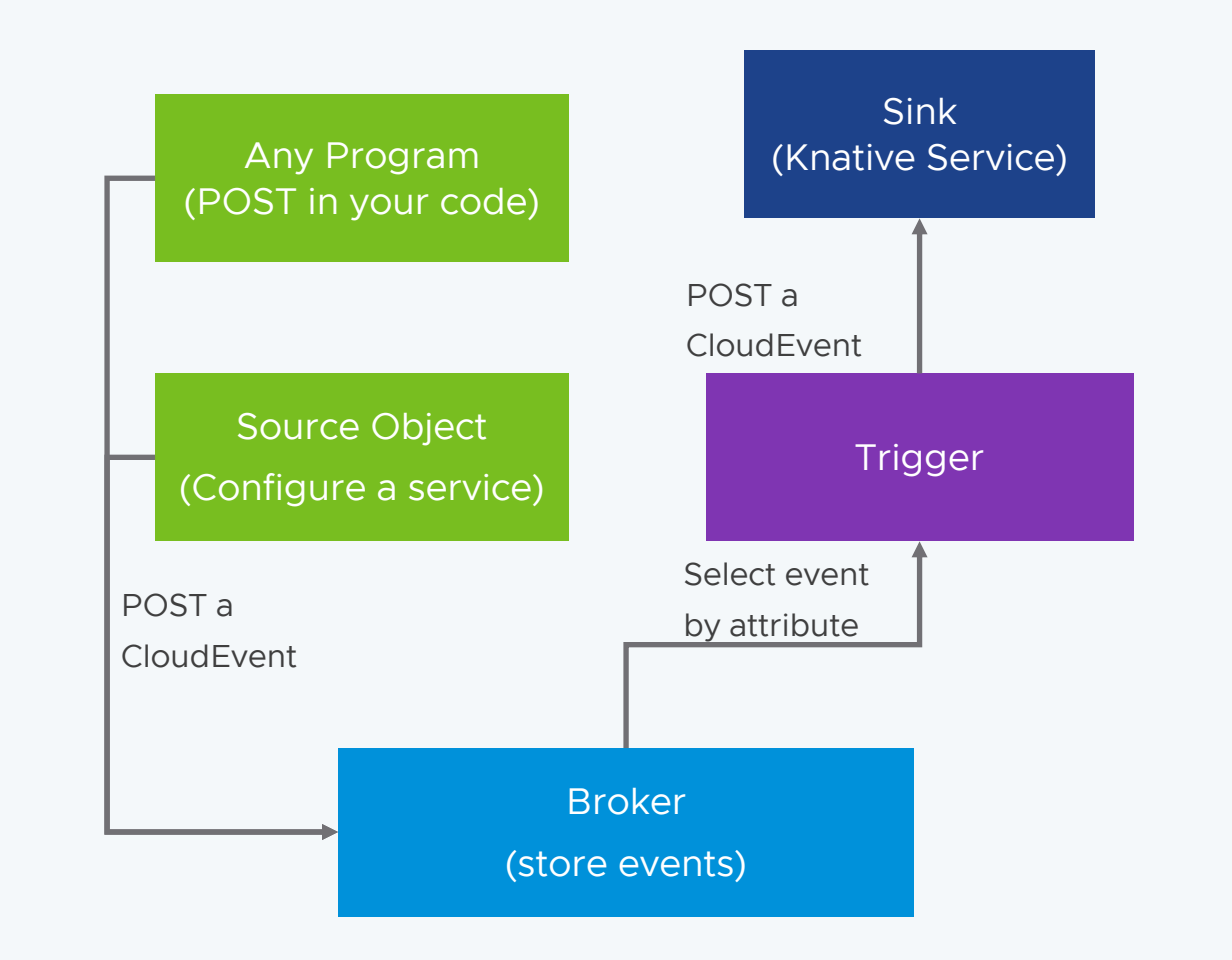
Kubernetes provides an extensible platform (using Custom Resource Definitions) for building APIs. Using Kubernetes allows you to use the same tools to manage your application code (and build) as well as your service configuration.

Knative Eventing: Abstractions and Value

Brokers provides reliable asynchronous event routing for applications. Events sent through a broker will be retried at the Sink until they are accepted.

Triggers provide a filter on events sent to the Broker. Triggers allow consumers to receive events without knowing the configuration of producers.

Sources provide an easy way to connect events from third-party services to a Broker. It's also possible to send events directly to a Broker using POST (`apitizer` does this).



Knative Eventing: What is an Event?

Knative Eventing uses the CloudEvents standard to send and store events.

CloudEvents provides a standard way to format and translate events in several different formats and transports, including:

- JSON
- HTTP
- AVRO
- AMQP

CloudEvents provides a simple envelope (like HTTP headers) with some standardized fields, and an opaque payload identified by a MIME type. The format is designed to allow intermediate routers to operate on the attributes without needing to process the payload.

Listening to Events

Our first application, `apitizer`, publishes a CloudEvent to a Broker named default.

There is a second application, `dessert`, which consumes the events and logs them (in a real-world example, the second application might track users for loyalty rewards or for abuse detection). `dessert` is checked out on the VM, but I've also published the image so you don't need to build it.

The `dessert` directory also contains a manifest which defines both a Knative Service definition for `dessert` and a Trigger to select messages from the Broker.

```
$ kubectl apply -f dessert.yaml
```

Verifying it works

You should be able to call the `apitizer` application and then look at the logs of the dessert application and see the CloudEvent sent by `apitizer`:

```
$ curl http://hello.default.127.0.0.1.nip.io
```

To read the logs of the `dessert` container:

```
$ kubectl logs -l serving.knative.dev/service=dessert -c user-container
```

Thank You