Introduction to KNATIVE - O'REILLY

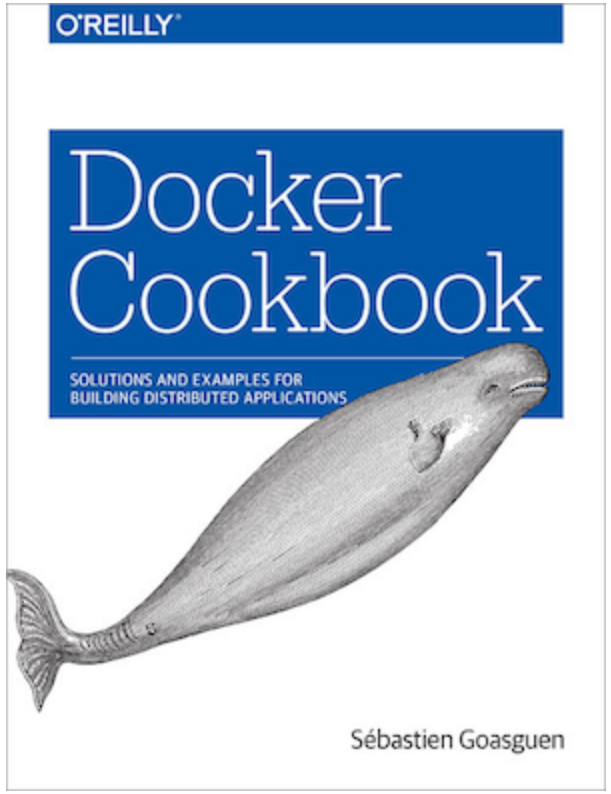
Going Serverless on Kubernetes

By sebgoa

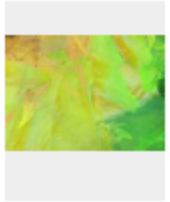
By Sebastien Goasguen, author of the Docker Cookbook and co-author of Kubernetes cookbook.

@sebgoa

@triggermesh https://github.com/triggermesh



O'Reilly Courses



LIVE ONLINE TRAINING

Introduction to Knative

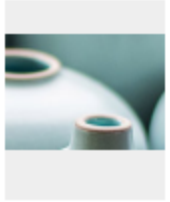
By Sébastien Goasguen

Session on May 14, 2019

Wait list available

Going serverless on Kubernetes As Kubernetes matures the focus is nov
to provide compute primitives that make applications building and man

---



LIVE ONLINE TRAINING

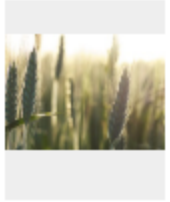
Introduction to Kubernetes

By Sébastien Goasguen

Session occurred on April 3, 2019

Managing distributed applications If your company is about to embrace
hands-on tour of Kubernetes core concepts covering installation, confi

---



LIVE ONLINE TRAINING

Building and Managing Kubernetes Applications

By Sébastien Goasguen

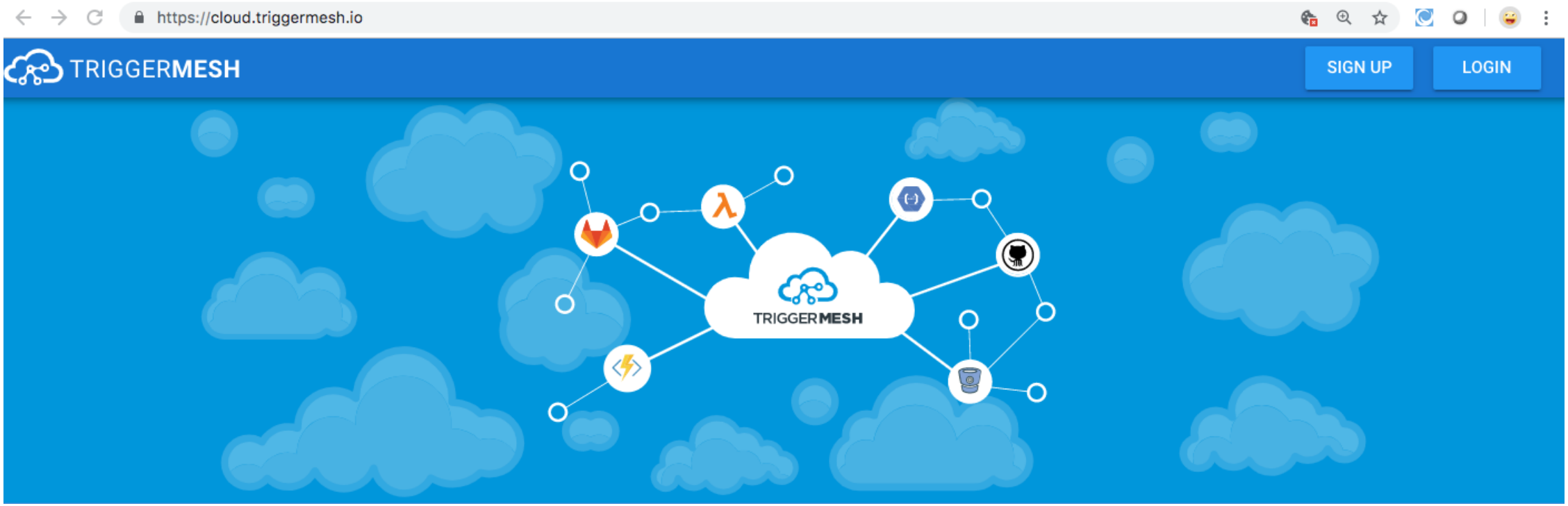
Pre-requisities

- `kubectl` , https://kubernetes.io/docs/user-guide/prereqs/
- `tm` https://github.com/triggermesh/tm
- Sign-in to https://cloud.triggermesh.io

TriggerMesh Cloud

https://cloud.triggermesh.io

- **Runs Knative so you don't have to**
- Exposes some of the Kubernetes API
- Free + gain time



Get `tm`

Head to the Github Release page https://github.com/triggermesh/tm/releases

- Download the `tm` version `0.0.12` and put it in your Path.
- Download the `config.json` file from the Dashboard and put it in `~/.tm/config.json`
- Verify with `tm get services`

Agenda

Part I

- Serverless Intro
- Knative Installation

Part II

- Knative Serving
- Knative Build and Tekton

Part III

- Knative Eventing

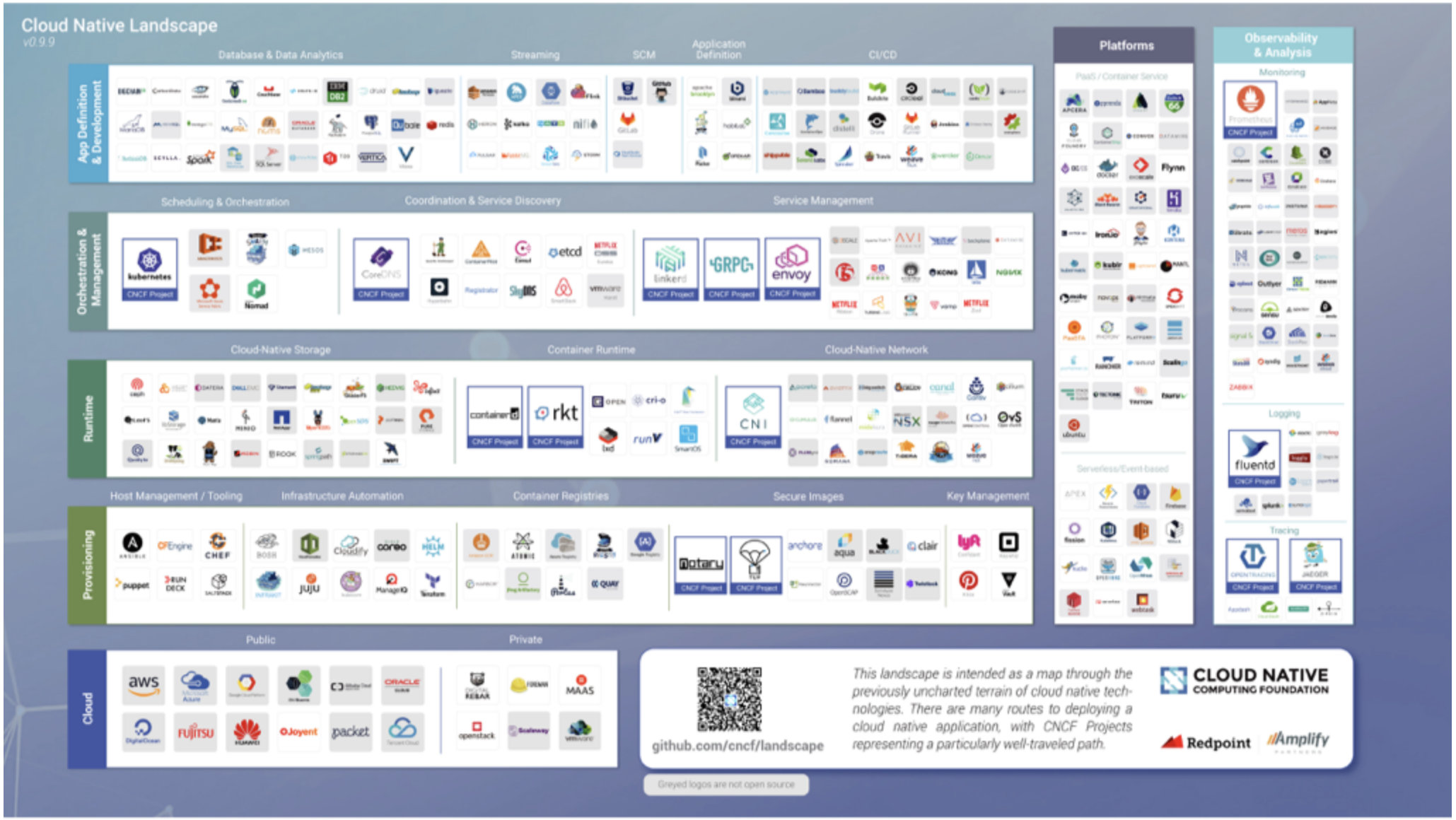
We break for 5 to 10 minutes twice !

IT Landscape

We are being bombarded with new tech every day.

Our landscapes of tools and solutions is increasingly hard to understand

EASY TO DISMISS NEW TECH AND BE JADED


Cloud Native Landscape
v0.9.9
App Definition & Development
Database & Data Analytics
Streaming
SCM
Application Definition
CI/CD
Orchestration & Management
Scheduling & Orchestration
Coordination & Service Discovery
Service Management
kubernetes
CNCF Project
CoreDNS
etcd
linkerd
GRPC
envoy
Runtime
Cloud-Native Storage
Container Runtime
Cloud-Native Network
containerd
rkt
CNI
Provisioning
Host Management / Tooling
Infrastructure Automation
Container Registries
Secure Images
Key Management
Notary
TUF
Cloud
Public
Private
aws
openstack
github.com/cncf/landscape
This landscape is intended as a map through the previously uncharted terrain of cloud native technologies. There are many routes to deploying a cloud native application, with CNCF Projects representing a particularly well-traveled path.
CLOUD NATIVE COMPUTING FOUNDATION
Redpoint
Amplify
Greyed logos are not open source
Platforms
PaaS / Container Service
Serverless/Event based
Observability & Analysis
Monitoring
Prometheus
Logging
fluentd
Tracing
OPENTRACING
JAEGER

How do you Choose and Keep up

- Software does not come out of thin air (i.e foundation)
- Evaluate technology and historical context
- Do not dismiss new paradigm

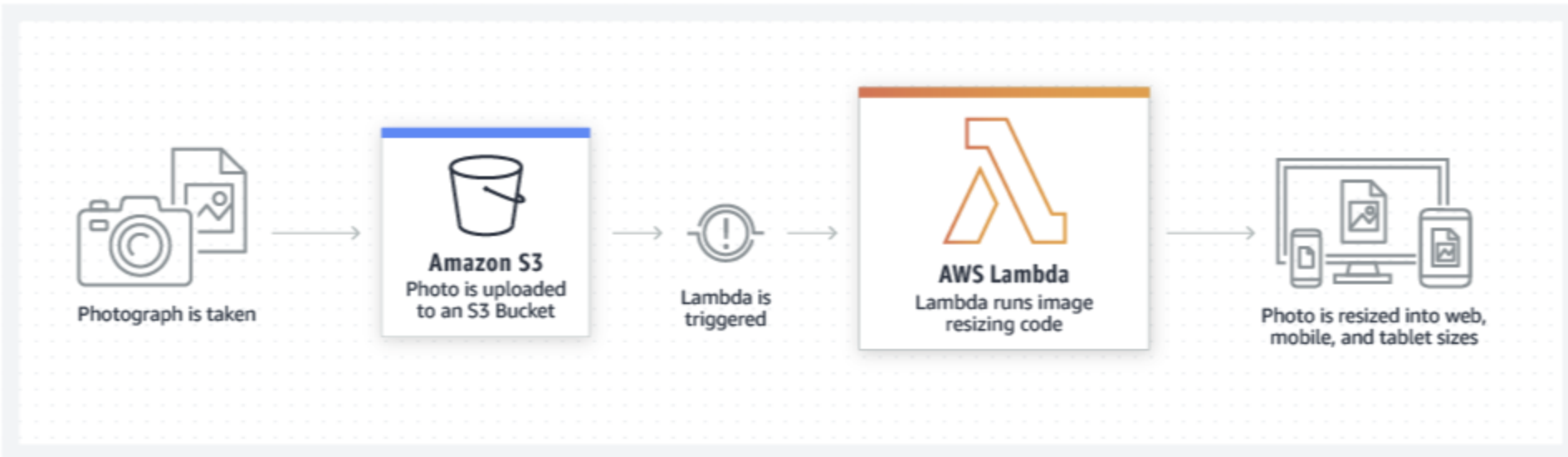
Kubernetes example !

Serverless

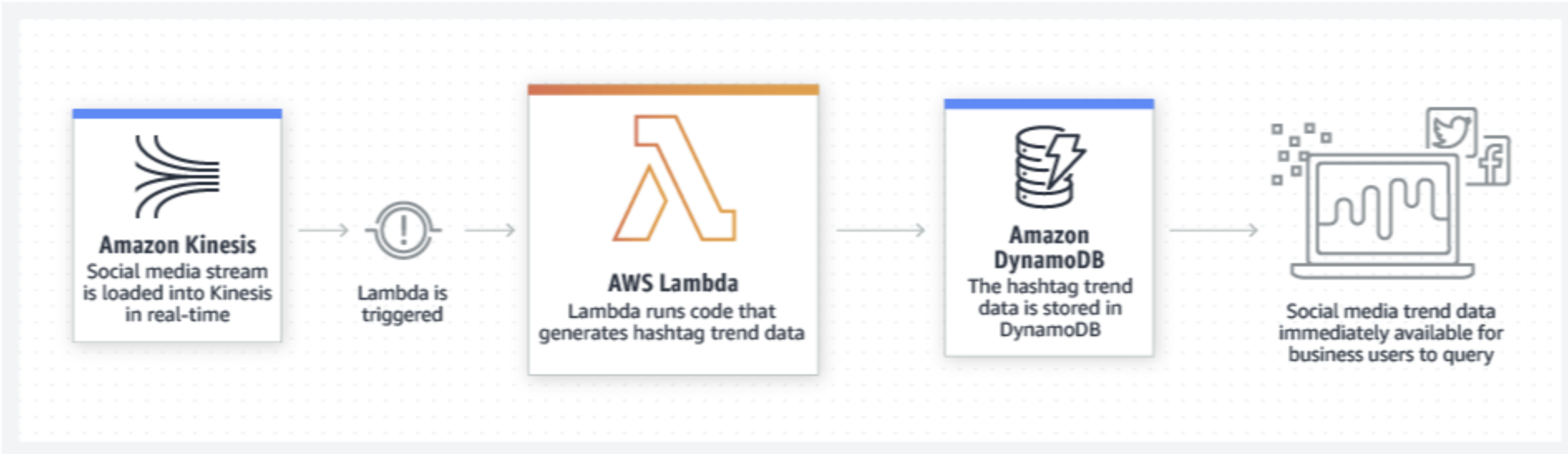
IT MAKES SENSE

- Less worry about infrastructure
- Less code
- Less wait
- More resilience
- More security
- More scalability
- More applications

File-processing

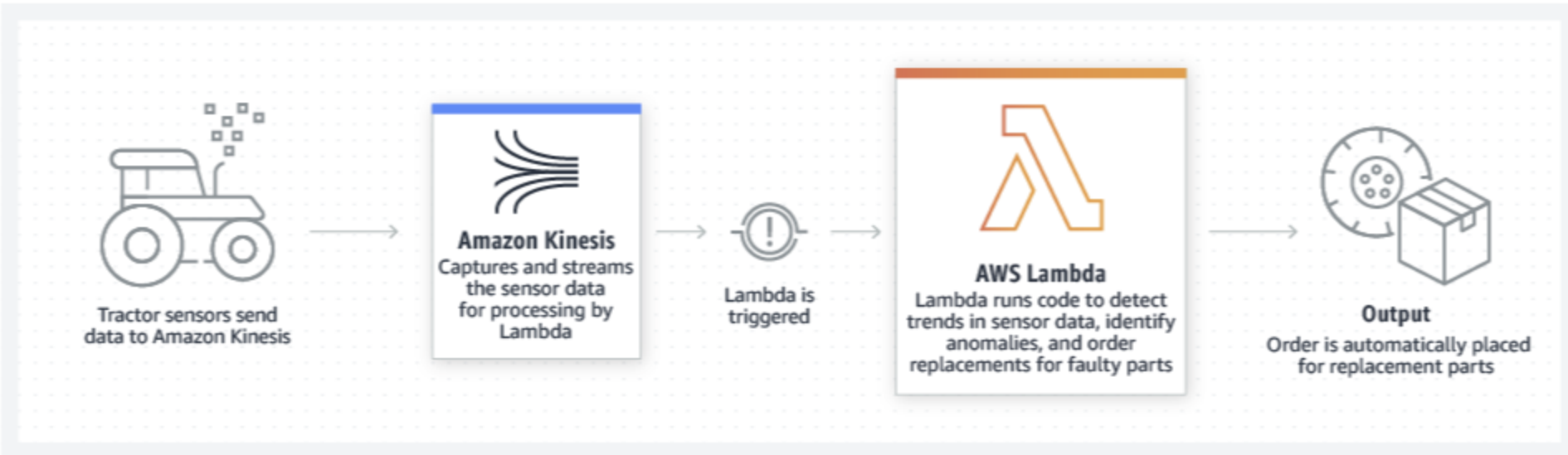


Stream Processing



Extract, Transform, Load (ETL)

IoT


Tractor sensors send data to Amazon Kinesis
Amazon Kinesis
Captures and streams the sensor data for processing by Lambda
Lambda is triggered
AWS Lambda
Lambda runs code to detect trends in sensor data, identify anomalies, and order replacements for faulty parts
Output
Order is automatically placed for replacement parts

Observations

- AWS is the leader
- "Simple" Pipeline but that can scale
- Serverless but also **ServiceFull**

Challenge

How can you build these applications:

- On your own or just using the services
- Without Lockin
- Using services that may only be available on-prem
- But with limited operational cost while having scale and resilience

DEMOS

- AWS Lambda
- CloudRun

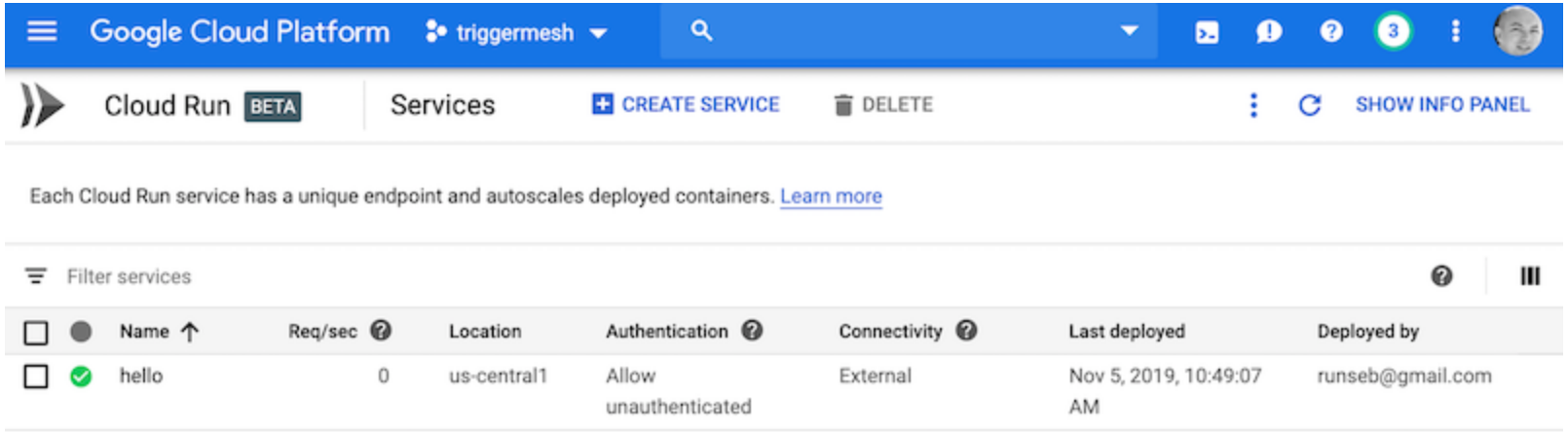
LAB 1

If you have an account on Google cloud you can try it on your own, if not please follow the live demo (creating an account on GCP is not mandatory)

- Log In to Google Cloud
- Launch a Hello example on Cloud Run
- Get its file manifest using `gcloud`

Cloud run interface

- Click on `Create Service`
- Use the `gcr.io/cloudrun/hello` container image
- Make sure to select `Allow Unauthenticated Invocations`



CloudRun

```
gcloud beta run services list
gcloud beta run services describe \
                          hello \
                          --region us-central1 \
                          --format yaml > hello.yaml
```

Let's clean the manifest

- Remove the `status` section
- Remove the `serviceaccount`
- Remove the `namespace`
- Just keep the `name` in the metadata

Deploy to TriggerMesh

Two methods:

```
kubectl -n <your_id> apply -f foo.yaml
```

Or:

Copy Paste by Creating a Service using the Icon.

Deploy a Service with `tm`

```
tm deploy service oreilly \
                  --from-image=gcr.io/cloudrun/hello \
                  --wait
tm get services
tm delete services oreilly
```

Go !

Extending Kubernetes

What if you need additional objects ...

Custom Resource Definitions.

Kubernetes lets you add your own API objects. Kubernetes can create a new custom API endpoint and provide CRUD operations as well as watch method.

This is great to extend the k8s API server with your own API.

Check the Custom Resource Definition documentation

CRD Example

```yaml
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: databases.foo.bar
spec:
  group: foo.bar
  version: v1
  scope: Namespaced
  names:
    plural: databases
    singular: database
    kind: DataBase
    shortNames:
    - db
```

Let's create this new resource and check that it was indeed created.

```
$ kubectl create -f database.yml
$ kubectl get customresourcedefinition
NAME                          KIND
databases.foo.bar             CustomResourceDefinition.v1beta1.apiextensions.k8s.io
```

Custom Resources

You are now free to create a *customresource*.

```
$ cat db.yml
apiVersion: foo.bar/v1
kind: DataBase
metadata:
  name: my-new-db
spec:
  type: mysql
$ kubectl create -f foobar.yml
```

And dynamically `kubectl` is now aware of the *customresource* you created.

```
$ kubectl get databases
NAME          KIND
my-new-db     DataBase.v1.foo.bar
```

Operator Framework(s)

- Kubebuilder: https://github.com/kubernetes-sigs/kubebuilder
- Operator Framework: https://github.com/operator-framework/operator-sdk
- Metaontroller: https://github.com/GoogleCloudPlatform/metacontroller

... Write your own

Knative CRDs

Knative components are a set of Kubernetes controllers. There are Knative CRDs and associated controllers

```
$ kubectl get crd | grep knative
brokers.eventing.knative.dev                                  39d
builds.build.knative.dev                                      160d
buildtemplates.build.knative.dev                              160d
channels.eventing.knative.dev                                 160d
clusterchannelprovisioners.eventing.knative.dev               160d
configurations.serving.knative.dev                            160d
containersources.sources.eventing.knative.dev                 160d
revisions.serving.knative.dev                                 160d
routes.serving.knative.dev                                    160d
services.serving.knative.dev                                  160d
subscriptions.eventing.knative.dev                            160d
triggers.eventing.knative.dev                                 39d
...
```

Knative Installation

At a high level we will:

- Create some CRDs
- Create some namespaces
- Launch controllers in those namespaces

Then we will be able to create the Knative API objects.

```
$ kubectl get ns | grep knative
knative-build          Active    160d
knative-eventing       Active    160d
knative-monitoring     Active    160d
knative-serving        Active    160d
knative-sources        Active    160d
```

Provider Agnostic Installation

https://knative.dev/docs/install/knative-with-any-k8s/

Install the Knative CRDs:

```
kubectl apply --filename https://github.com/knative/serving/\
                          releases/download/v0.12.0/serving-crds.yaml
```

Then the Knative controllers:

```
kubectl apply --filename https://github.com/knative/serving/\
                          releases/download/v0.12.0/serving-core.yaml
```

Need an Ingress Gateway

- Istio
- Ambassador
- Solo
- Contour

  ...

```
kubectl apply --filename https://raw.githubusercontent.com/knative/\
              serving/v0.12.0/third_party/contour-latest/contour.yaml
```

See https://knative.dev/docs/install/knative-with-contour/

Live Screencast

Knative Installation

TGIM

BREAK TIME

Part II

- Serving
- A bit on Tekton

Knative Serving

> Knative Serving builds on Kubernetes to support deploying and serving of serverless applications and functions.

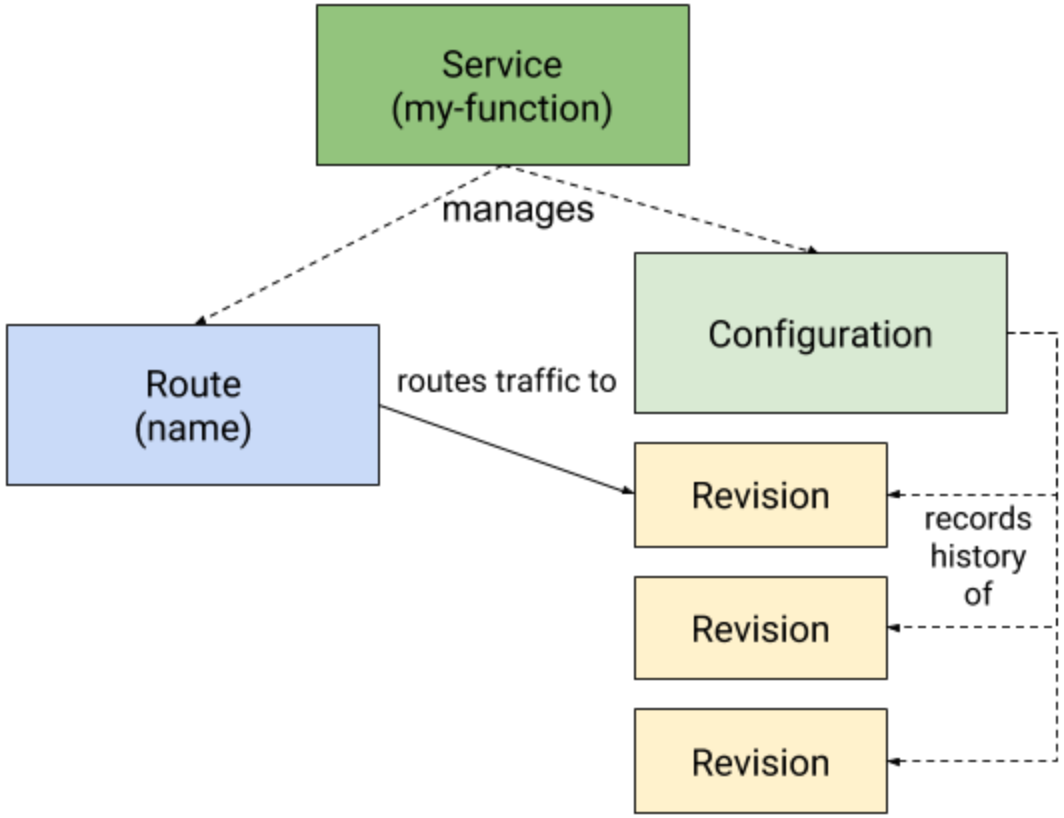
```
$ kubectl get pods -n knative-serving
NAME                                READY     STATUS
activator-6f55c97c6d-tsm5w          2/2       Running
autoscaler-84cc7b78c4-ng96p         2/2       Running
controller-db5bbf4b9-6vdq9          1/1       Running
webhook-85ddccf9c6-gfcjh            1/1       Running
```

Under the hood still a Deployment and a Pod ...

Knative Serving API Objects

- **Service**: The `service.serving.knative.dev` resource automatically manages the whole lifecycle of your workload.
- **Route**: The `route.serving.knative.dev` resource maps a network endpoint to a one or more revisions.
- **Configuration**: The `configuration.serving.knative.dev` resource maintains the desired state for your deployment.
- **Revision**: The `revision.serving.knative.dev` resource is a point-in-time snapshot of the code and configuration for each modification made to the workload.

Knative Serving Objects Diagram



Serving Specification

```yaml
apiVersion: serving.knative.dev/v1alpha1
kind: Service
metadata:
  name: helloworld-go
spec:
  template:
    spec:
      containers:
        - image: gcr.io/knative-samples/helloworld-go
          env:
            - name: TARGET
              value: "Go Sample v1"
```

`kubectl apply -f hello.yaml` or paste it in the TriggerMesh UI

Deploy a Service with `tm`

```
tm deploy service oreilly --from-image=gcr.io/cloudrun/hello \
                          --wait
tm get services
tm delete services oreilly
```

Traffic Splitting

See Blue/Green with Knative sample

Route can split traffic between Revisions:

```yaml
apiVersion: serving.knative.dev/v1alpha1
kind: Route
metadata:
  name: blue-green-demo # Updating our existing route
spec:
  traffic:
    - revisionName: blue-green-demo-00001
      percent: 50 # Updating the percentage from 100 to 50
    - revisionName: blue-green-demo-00002
      percent: 50 # Updating the percentage from 0 to 50
      name: v2
```

Let's try it ?

Traffic Splitting Sample

- Create the Service object with the UI (paste the yaml and remove the namespace)

```yaml
apiVersion: serving.knative.dev/v1alpha1
kind: Service
metadata:
  name: demo
spec:
  template:
    metadata:
      name: demo-blue
    spec:
      containers:
        - image: gcr.io/knative-samples/knative-route-demo:blue
          env:
            - name: T_VERSION
              value: "blue"
  traffic:
  - tag: current
    revisionName: demo-blue
    percent: 100
```

Update the YAML via the UI

Switch to **green**

```yaml
apiVersion: serving.knative.dev/v1alpha1
kind: Service
metadata:
  name: demo
spec:
  template:
    metadata:
      name: demo-green
    spec:
      containers:
        - image: gcr.io/knative-samples/knative-route-demo:green
          env:
            - name: T_VERSION
              value: "green"
  traffic:
  - tag: green
    revisionName: demo-green
    percent: 50
  - tag: blue
    revisionName: demo-blue
    percent: 50
```

Traffic Splitting

Building Containers

But but...

I thought Serverless had nothing to do with Containers, can't I just run my code ?

Sure but it will need to run somewhere and be packaged. Containers are a great packaging artefcats. If you give me your code, I still need to package it, aka. Build.

Hence we need a way to create Containers within a Kubernetes cluster

Originally `Pipeline` project within the Knative github organization. Donated to CNCF at creation of the CDF foundation.

Tekton Pipelines

BREAK TIME

Part III

Knative Eventing

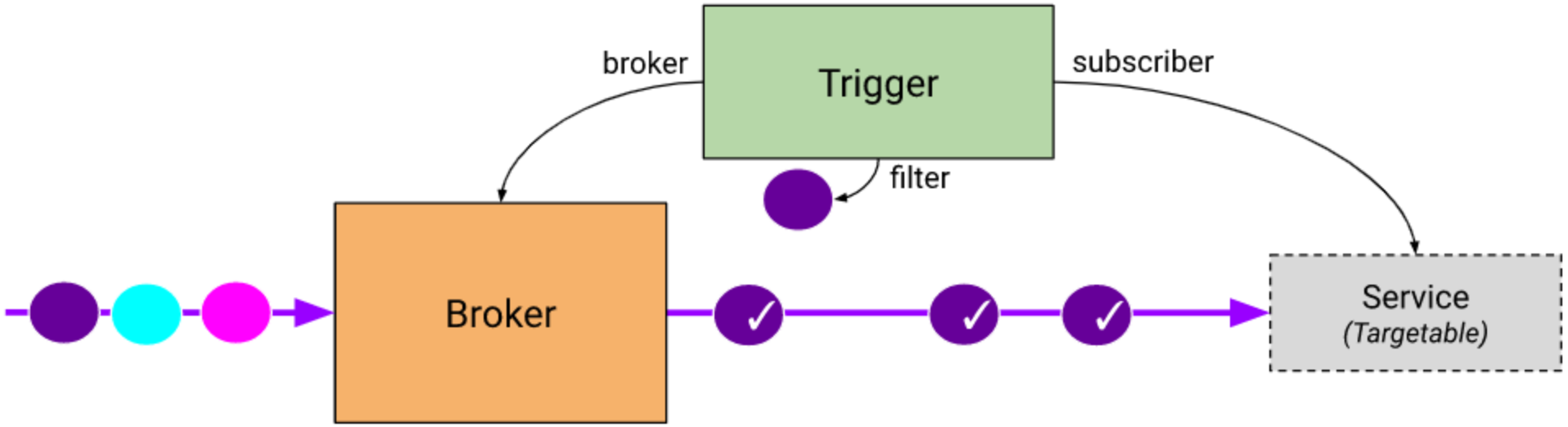
> Knative Eventing is a system that is designed to address a common need for cloud native development and provides composable primitives to enable late-binding event sources and event consumers.

Consume events from *Sources*, use those events to *Trigger* execution of *functions*.

Knative eventing Objects

Architecture still in flux (`v0.7`) trying to find the right abstractions to decouple eventing from messaging and provide easy to use objects.

- Broker
- Trigger
- Channel
- Subscription



Knative Eventing

When running properly, check the `knative-eventing` namespace

```
$ kubectl get pods -n knative-eventing
NAME                                        READY     STATUS    RESTARTS   AGE
eventing-controller-774f79f989-xp2kc        1/1       Running   0          10d
in-memory-channel-controller-5c686c86c7-5kvgr   1/1       Running   0          10d
in-memory-channel-dispatcher-7bcd7f556-q25qb    2/2       Running   3          10d
eventing-webhook-5b689bfcc4-78772           1/1       Running   0          10d
```

You may see other channel controllers (e.g Kafka, NATS, GCP PubSub ...)

Knative Eventing Objects

Sources, Channels, Triggers, Brokers ...

```yaml
apiVersion: sources.eventing.knative.dev/v1alpha1
kind: CronJobSource
metadata:
  name: test-cronjob-source
spec:
  schedule: "*/2 * * * *"
  data: '{"message": "Hello world!"}'
  sink:
    apiVersion: serving.knative.dev/v1alpha1
    kind: Service
    name: event-display
```

Demo Eventing

1. Run a `message-dumper` service
2. Run a `CronJob` source

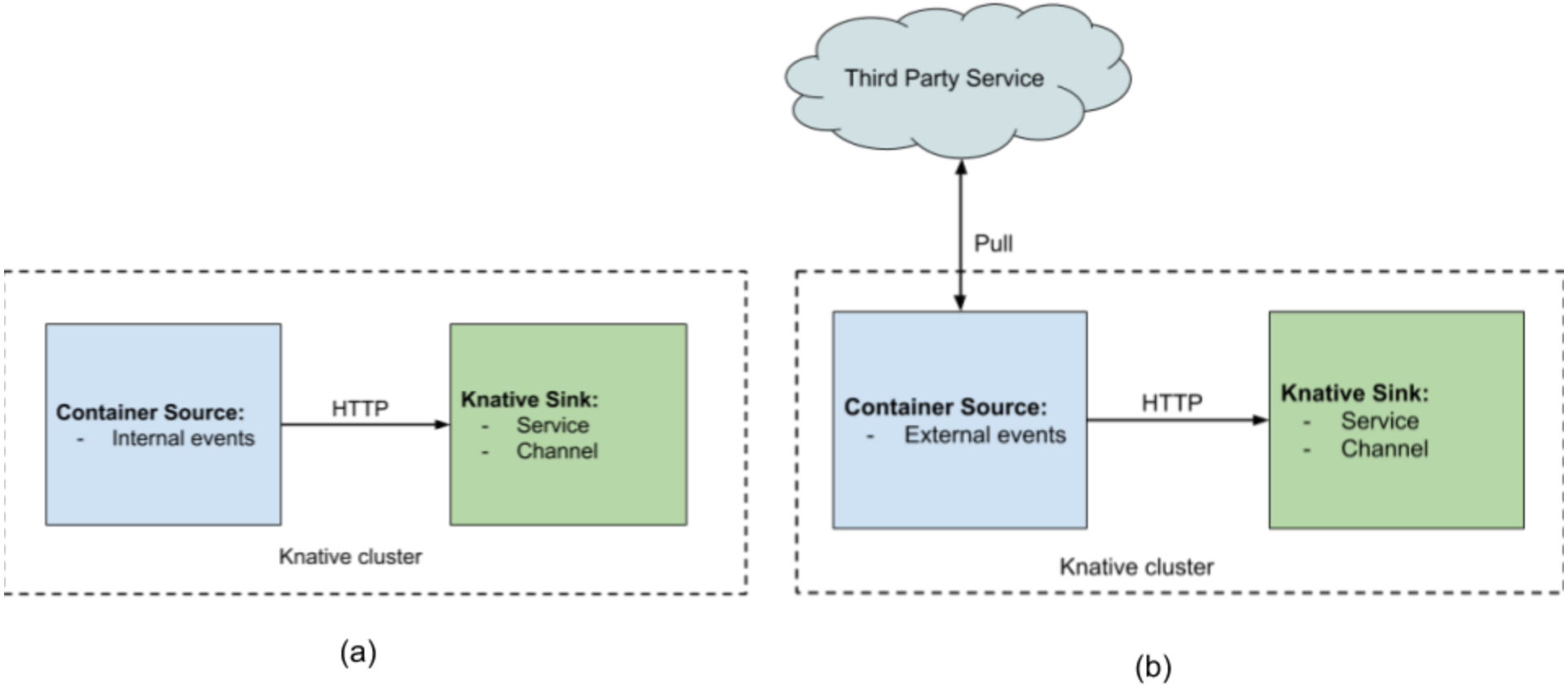
Check the objects with `kubectl` or `tm`

See https://github.com/knative/docs/tree/master/docs/eventing/samples/cronjob-source

If time permits, let's do a GitHub Source ...

Writing your Own Knative Event Source

Check ksources



Writing your Own Knative Event Source

Can be as simple as packaging a Bash script in a container:

```yaml
apiVersion: sources.eventing.knative.dev/v1alpha1
kind: ContainerSource
metadata:
  name: bashsample
spec:
  image: gcr.io/triggermesh/bash
  sink:
    apiVersion: eventing.knative.dev/v1alpha1
    kind: Channel
    name: default
```

Wrap-Up

- Knative is an extension of the Kubernetes API
- It provides APIs to build serverless workloads
- Serving gives you scale to zero
- Eventing allows you to trigger function when events happen

Serverless is more than FaaS, it blends Event Driven Architecture (EDA) with new containerized workloads.

Thank You

@sebgoa

Feedback, contributions to TriggerMesh would be lovely !!!