Introduction to KNATIVE

Application Integration via Knative Serving and Eventing

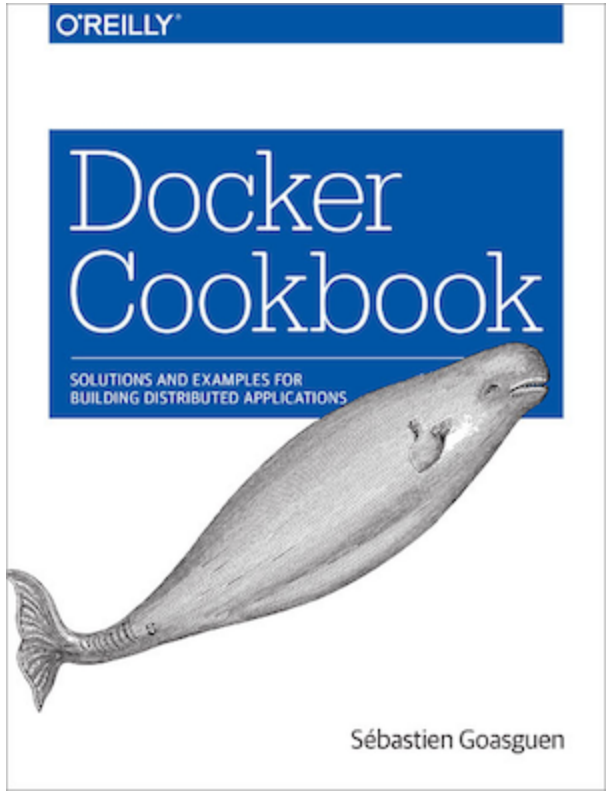
TRIGGERMESH
CLOUD NATIVE INTEGRATION PLATFORM

By sebgoa

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

@sebgoa

@triggermesh https://github.com/triggermesh



Pre-requisities

- `kubectl` , https://kubernetes.io/docs/user-guide/prereqs/
- Sign-in to https://cloud.triggermesh.io
- A fresh Kubernetes cluster

https://cloud.triggermesh.io

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

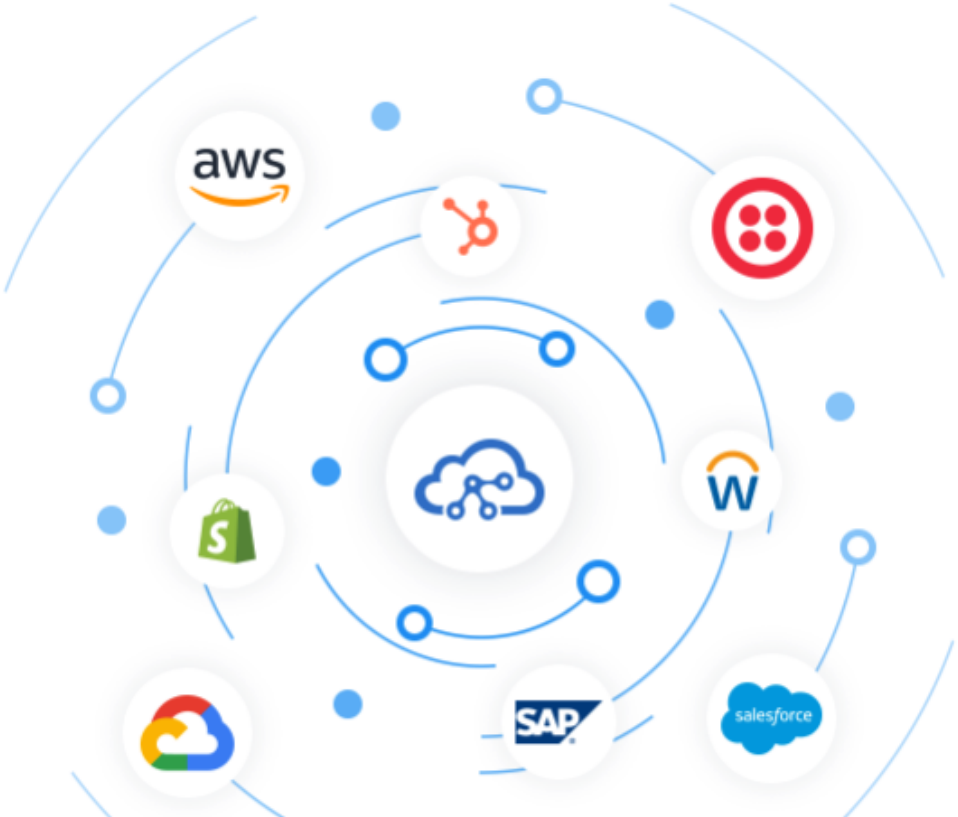
TRIGGERMESH

LOGIN SIGN UP

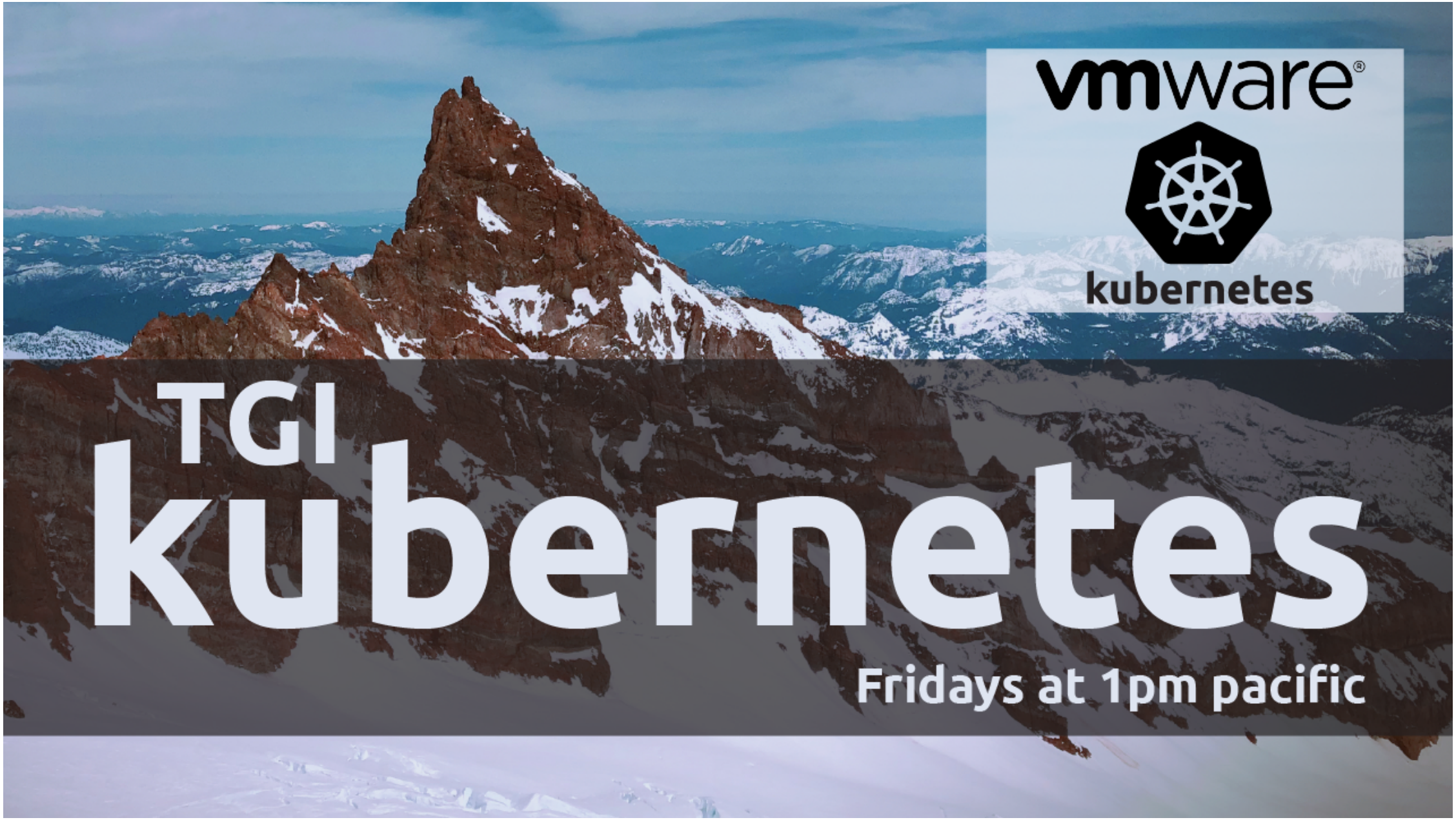
Welcome to TriggerMesh

Cloud native integration platform that provides a cloud native, scalable platform that integrates applications and automation workflows from data center to the cloud and back.

DOCUMENTATION SIGN UP



Training TGIK style



Agenda

Part I

- Serverless Intro
- A look at AWS Lambda and Google CloudRun
- Knative Installation

Part II

- Knative Serving
- Auto-scaling

Part III

- Knative Eventing
- Integration with Kafka

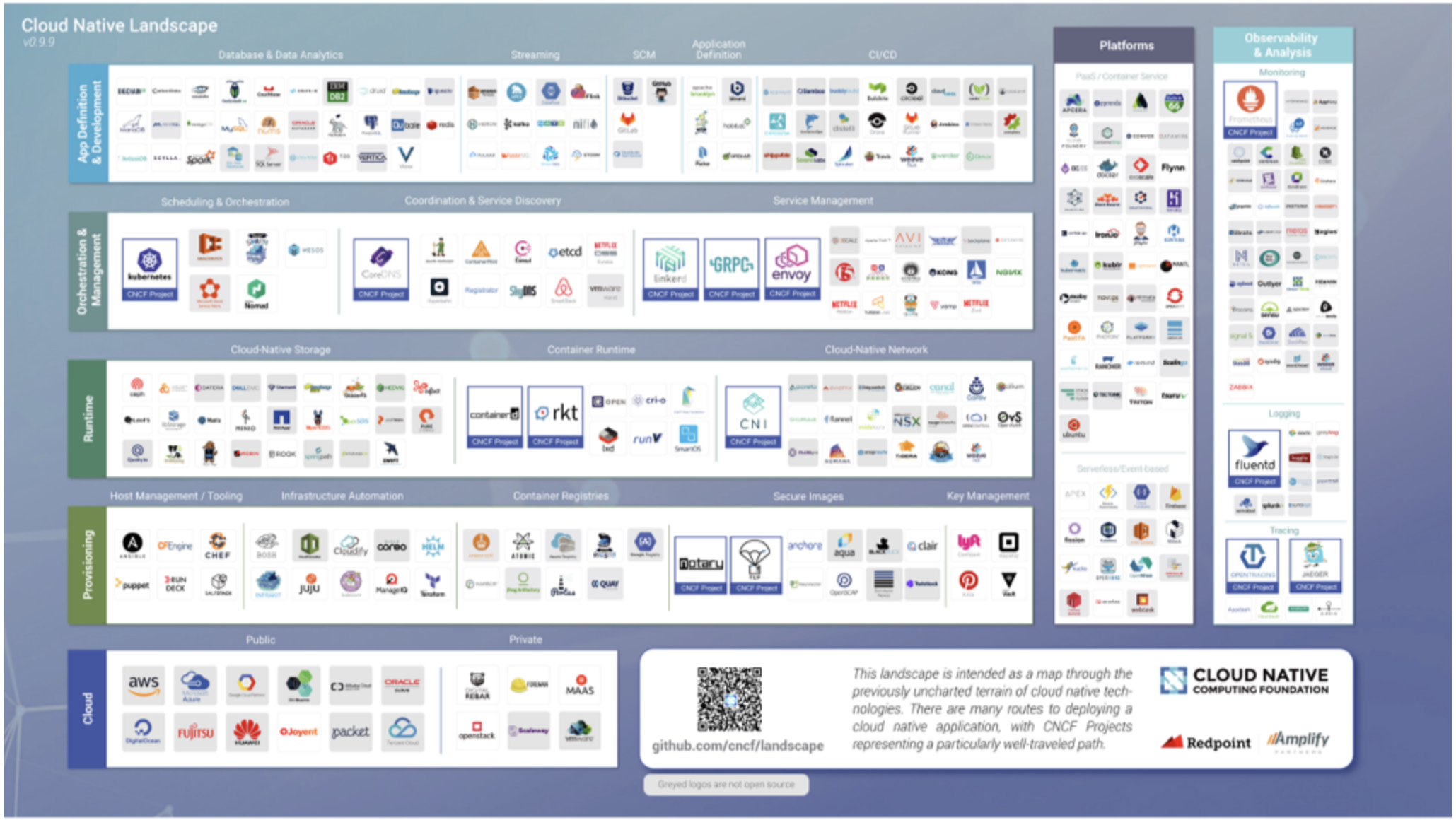
We break for 5 to 10 minutes twice !

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

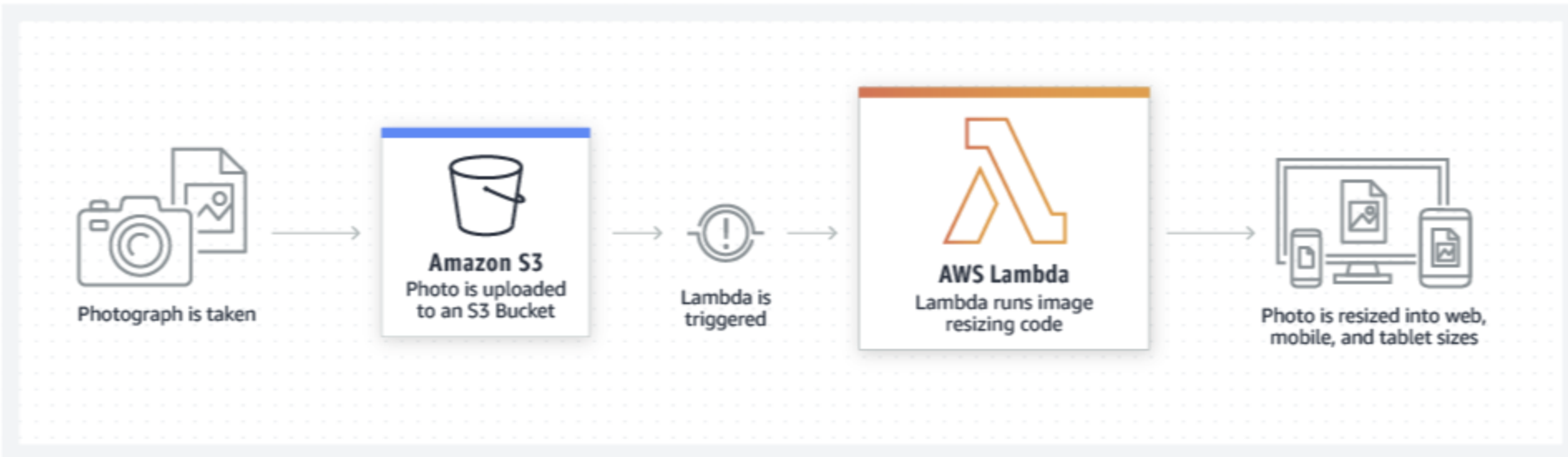
Kubernetes example !

Serverless

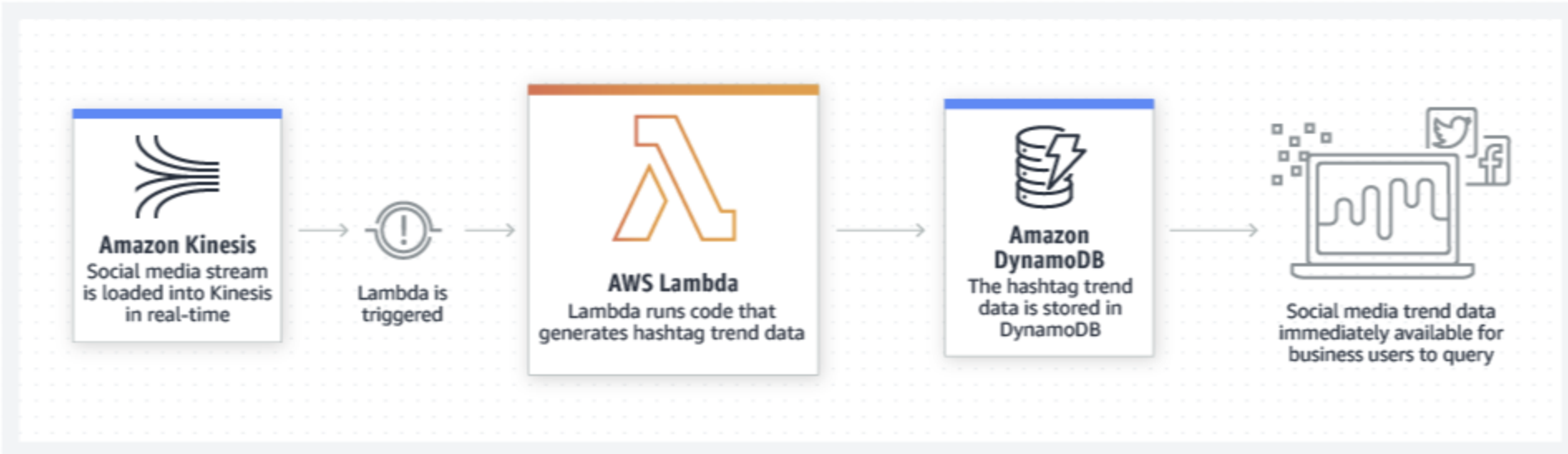
IT MAKES SENSE

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

File-processing

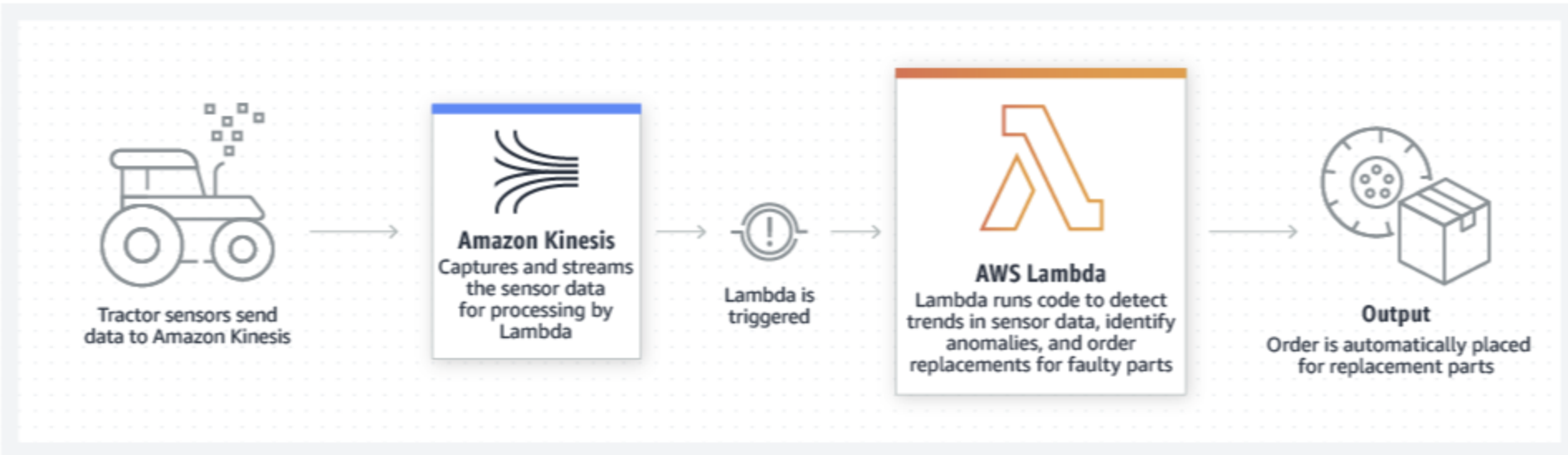


Stream Processing



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**
- **Integrations of Services thanks to Events**

Challenge

How can you build these applications:

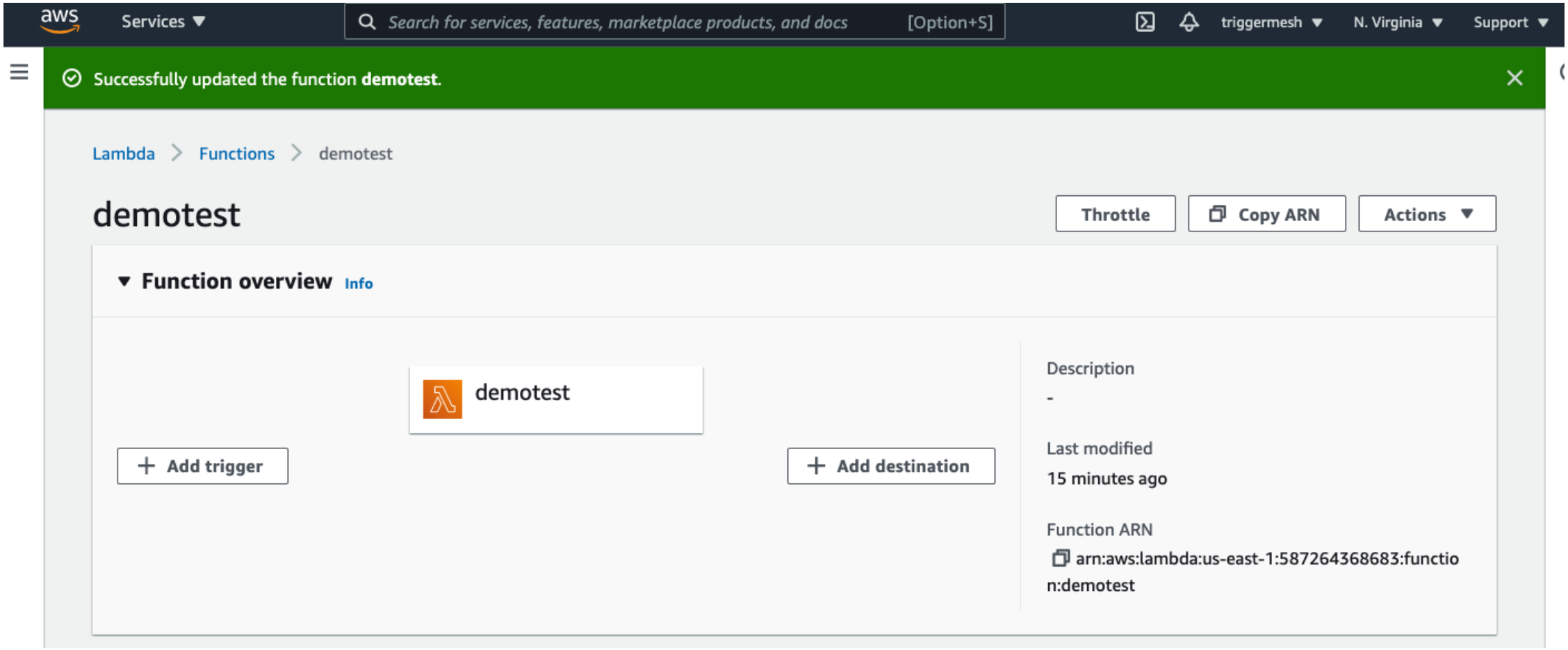
- On your own or just using the services
- Without Lockin
- Using services that may only be available on-prem
- But with limited operational cost while having scale and resilience

DEMOS

- AWS Lambda
- Google CloudRun

TGIK style ...

AWS Lambda


aws
Services
Search for services, features, marketplace products, and docs
[Option+S]
triggermesh
N. Virginia
Support
Successfully updated the function demotest.
Lambda
Functions
demotest
demotest
Throttle
Copy ARN
Actions
Function overview
Info
demotest
Add trigger
Add destination
Description
-
Last modified
15 minutes ago
Function ARN
arn:aws:lambda:us-east-1:587264368683:function:demotest

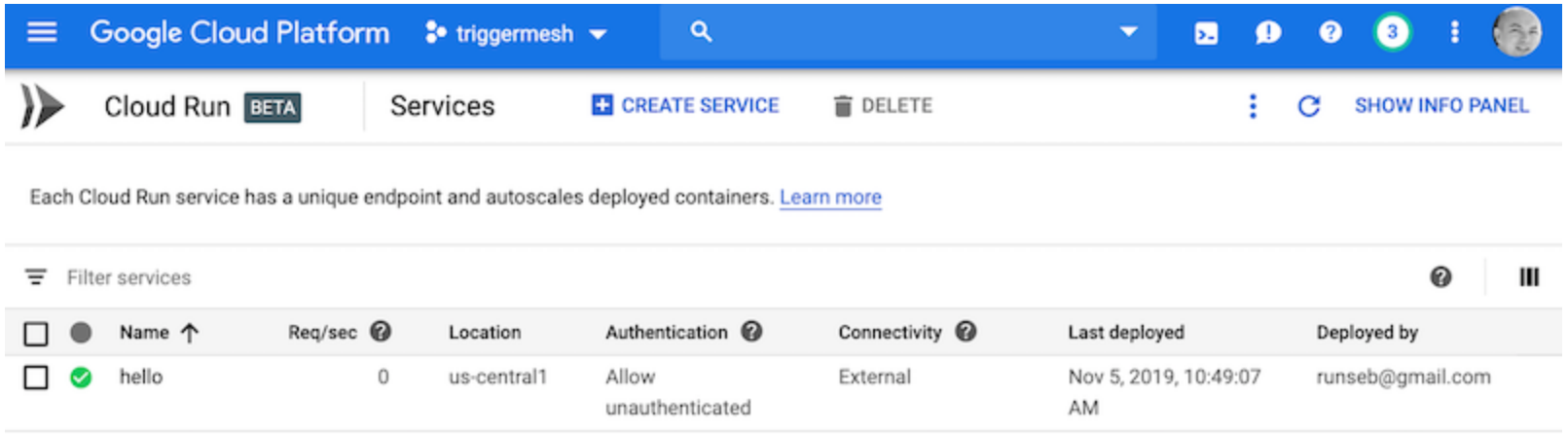
Google CloudRun

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`



See the Web application

CloudRun

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

Find the CloudRun manifest

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

Deploy to TriggerMesh

Two methods:

Copy Paste by Creating a Service using the Icon.

Or:

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

How can we get CloudRun on Kubernetes ?

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
# kubectl get crd | grep knative
images.caching.internal.knative.dev                    2020-02-10T14:27:05Z
certificates.networking.internal.knative.dev           2020-02-10T14:27:06Z
configurations.serving.knative.dev                     2020-02-10T14:27:06Z
ingresses.networking.internal.knative.dev              2020-02-10T14:27:06Z
metrics.autoscaling.internal.knative.dev               2020-02-10T14:27:06Z
podautoscalers.autoscaling.internal.knative.dev        2020-02-10T14:27:06Z
revisions.serving.knative.dev                          2020-02-10T14:27:06Z
routes.serving.knative.dev                             2020-02-10T14:27:06Z
...
```

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-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/

Install the `kn` client

https://github.com/knative/client/blob/main/docs/README.md

```
$ kn
kn is the command line interface for managing Knative Serving and Eventing resources

 Find more information about Knative at: https://knative.dev

Serving Commands:
  service      Manage Knative services
  revision     Manage service revisions
  route        List and describe service routes
  domain       Manage domain mappings
```

Live Screencast

Knative Installation

BREAK TIME

Part II

- Serving

Knative Serving

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

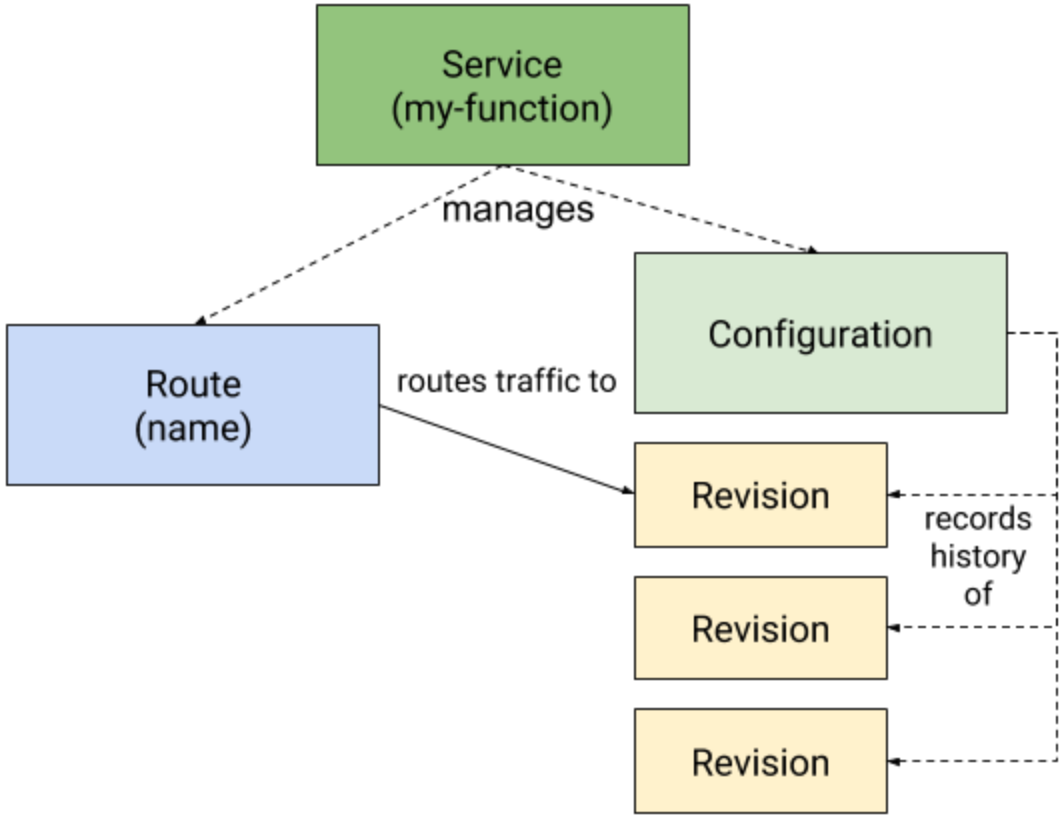
```
$ kubectl get pods -n knative-serving
NAME                                      READY   STATUS    RESTARTS   AGE
webhook-9cd7878cd-nbwkc                   1/1     Running   0          93m
controller-6569b6687d-btzhs               1/1     Running   0          93m
autoscaler-77dd48cfdf-5fw49               1/1     Running   0          93m
activator-67889464d8-zqskq                1/1     Running   0          93m
contour-ingress-controller-6588cf5fdd-zrxvz   1/1     Running   0          91m
```

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/v1
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

Checking all "Children"

Let's check it out via some additional bonus

- Let's install Krew
- Let's install tree

Then

```
kubectl tree ksvc hello
```

Rollout and Traffic Control

```yaml
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: helloworld-go
  namespace: default
  annotations:
    serving.knative.dev/rolloutDuration: "380s"
```

And

```yaml
traffic:
- percent: 99
  revisionName: config-00008
- percent: 1
  revisionName: config-00009
```

About Building Containers as Functions

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

TriggerMesh KLR

TRIGGERMESH KLR
KNATIVE LAMBDA RUNTIME

Get AWS Lambda compatibility https://github.com/triggermesh/knative-training/tree/main/python

And also

Build your Own CloudEvent Function

AutoScaling

Documentation / Knative Serving / Autoscaling / Autoscale Sample App - Go

Autoscale Sample App - Go

A demonstration of the autoscaling capabilities of a Knative Serving Revision.

Prerequisites

1. A Kubernetes cluster with Knative Serving installed.
2. The `hey` load generator installed (`go get -u github.com/rakyll/hey`).
3. Clone this repository, and move into the sample directory:

BREAK TIME

Part III

Knative Eventing

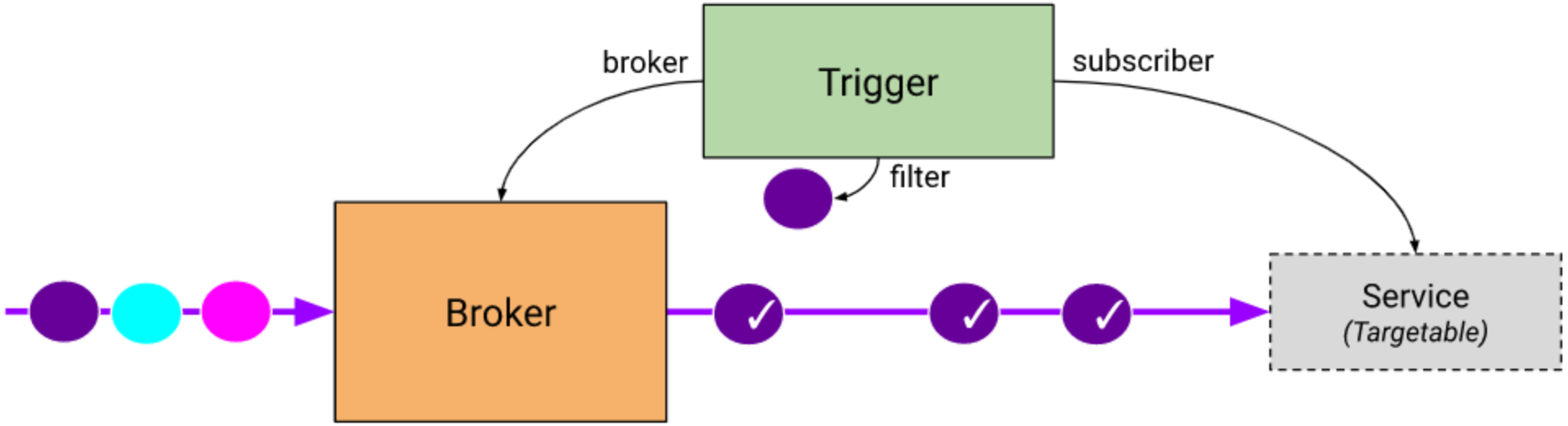
> Knative Eventing is a system that is designed to address a common need for cloud native development and provides composable primitives to enable late-binding event sources and event consumers.

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

Knative eventing Objects

``v1.0` is coming SOOOONNNN !

- 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
sources-controller-6bf9f6d958-h9996     1/1     Running   0          15s
imc-controller-675dd47677-w9kjr         1/1     Running   0          15s
eventing-controller-6f4bbb779b-frslp    1/1     Running   0          15s
imc-dispatcher-6c9875f557-md4k2         1/1     Running   0          15s
eventing-webhook-9c697c59-z95xs         1/1     Running   0          15s
```

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

Knative Eventing Objects

Sources, Channels, Triggers, Brokers ...

```yaml
apiVersion: sources.knative.dev/v1
kind: PingSource
metadata:
  name: test-pingsource
spec:
  schedule: "*/1 * * * *"
  data
  sink:
    ref:
      apiVersion: v1
      kind: Service
      name: sockeye
```

Demo Eventing

- 101 Ping Source
- 102 GitHub Source
- 201 AWS Sources
- 301 Kafka Sources and Sinks
- 401 Writing your Own Sink

Basic Ping

1. Run a `sockeye` service
2. Run a `Pingsource` source

Check the objects with `kubectl` or `tm`

See https://knative.dev/docs/eventing/sources/ping-source/

Let's do a GitHub Source ...

AWS Sources

release v1.6.0 | downloads 321 | circleci passing | go report A+ | license Apache-2.0

TRIGGERMESH SAWS

SOURCES FROM AMAZON WEB SERVICES

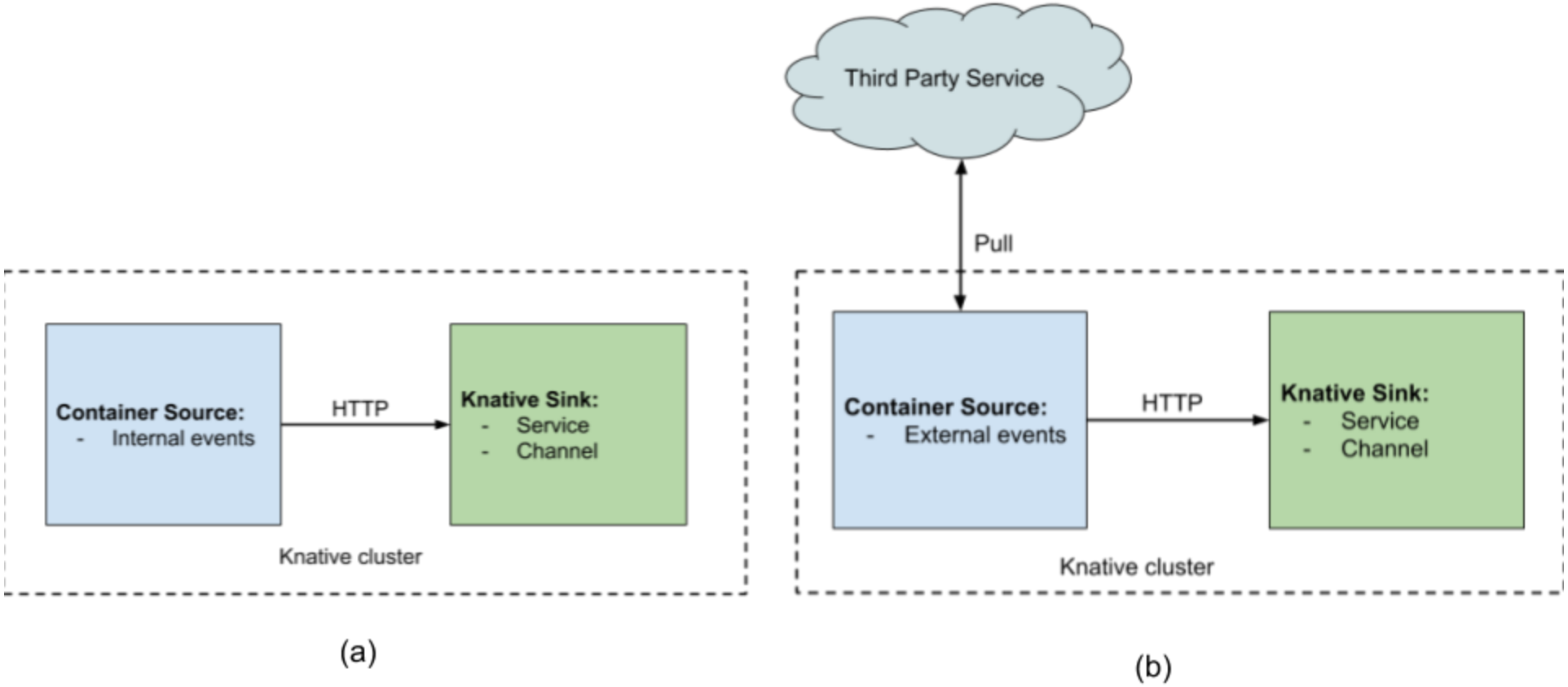
TriggerMesh Sources for Amazon Web Services (SAWS) allow you to quickly and easily consume events from your AWS services and send them to workloads running in your cluster.

Other Knative Sources maintained by TriggerMesh are available in the following repositories:

- Knative Sources
- GitLab Source (Knative sandbox project)

Writing your Own Knative Event Source

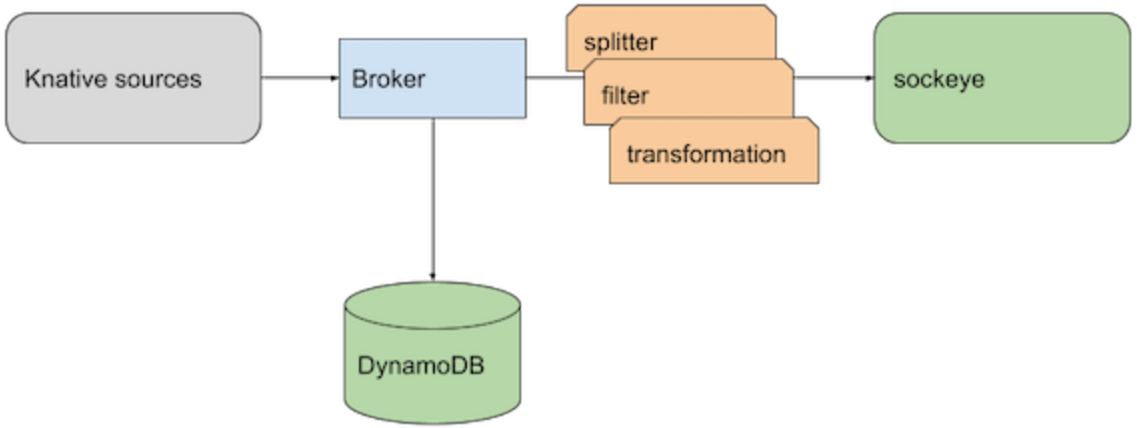
Check https://github.com/triggermesh/bringyourown



Python CloudEvent Handling

```
@app.route('/', methods=['POST'])
def target():

    # create a CloudEvent
    event = from_http(request.headers, request.get_data())
    cejson = json.loads(to_json(event).decode('utf-8'))

    # Do your Transformation or Target work based on the eventype
    if event['type'] == "io.triggermesh.target.dynamodb.insert":
        logging.info("Store event in dynamodb")
        table.put_item(Item=cejson)
    ...

    return "", 204

if __name__ == '__main__':
        app.run(host='0.0.0.0', port=8080)
```

"Wire Tap" to DynamoDB



Check the code at https://github.com/triggermesh/knative-training/tree/main/sink

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
- Multicloud service integration is possible with Knative
- TriggerMesh gives you a Platform to do it simply

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

Thank You

@sebgoa

Contact TriggerMesh for product demos, knative training and services

sebgoa@triggermesh.com

Feedback, contributions to TriggerMesh would be lovely !!!