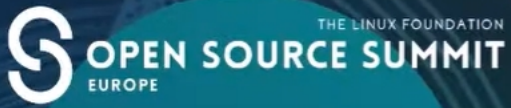


Knative: A Kubernetes Framework to Manage Serverless Workloads

Nikhil Barthwal / Google

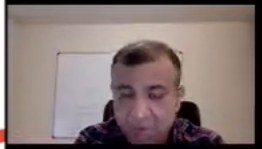
#ossummit

Nikhil Barthwal
Product Manager (Serverless), Google Cloud
@nikhilbarthwal
nikhilbarthwal@yahoo.com
www.nikhilbarthwal.com

Knative:

Kubernetes Framework to manage Serverless Workloads

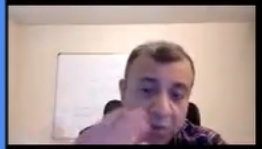
https://www.github.com/nikhilbarthwal/knative



Impostazioni

Introduction to Knative

01

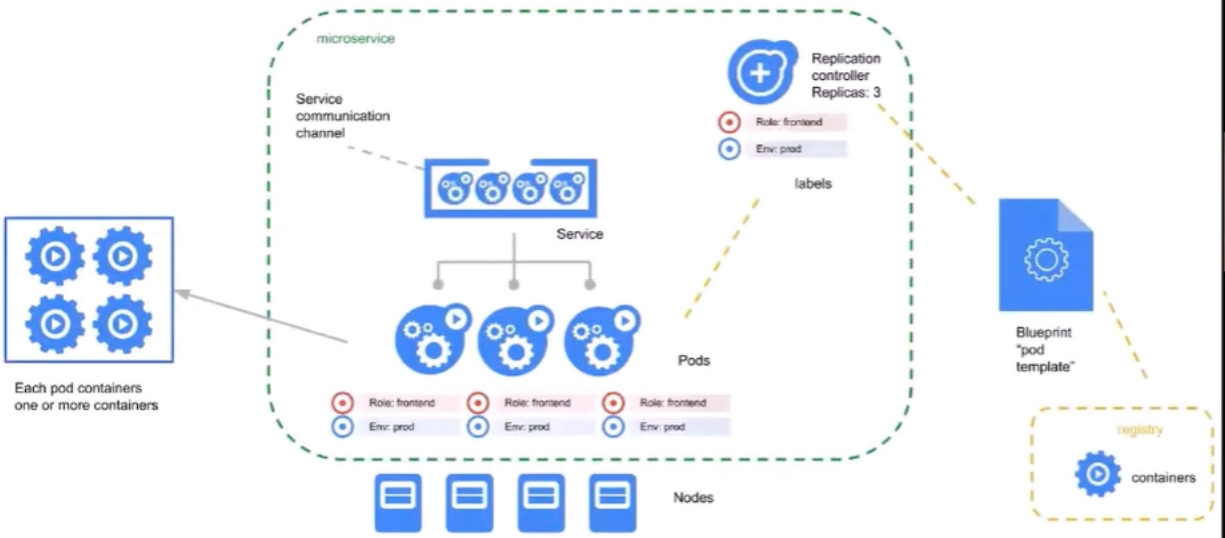


[kay-native]

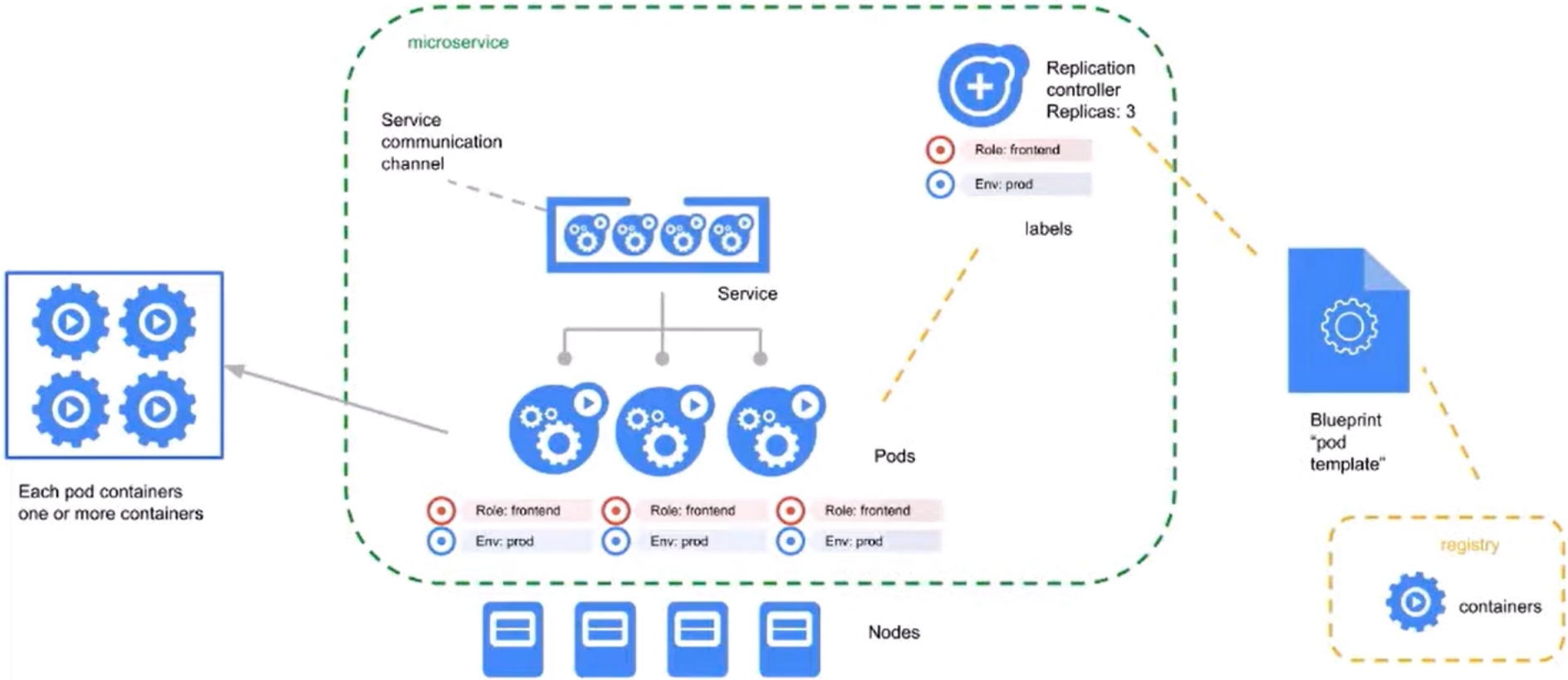
Knative

Kubernetes based open source
building blocks for serverless

Kubernetes: The de facto platform



Kubernetes: The de facto platform



Kubernetes: The defacto platform

Scheduling
Decide where my containers should run

Lifecycle and health
Keep my containers running despite failures

Scaling
Make sets of containers bigger or smaller

Naming and discovery
Find where my containers are now

Load balancing
Distribute traffic across a set of containers

Storage volumes
Provide data to containers

Logging and monitoring
Track what's happening with my containers

Debugging and introspection
Enter or attach to containers

Identity and authorization
Control who can do things to my containers

Kubernetes ecosystem

5,000+ Contributors **47k+** GitHub stars

*Sources: COCOMO Model, CNCF.io Certified Providers

Serverless model

Serverless model

Operational Model

No Infra Management

Managed Security

Pay only for usage

Serverless model

Operational Model

No Infra Management

Managed Security

Pay only for usage

Programming Model

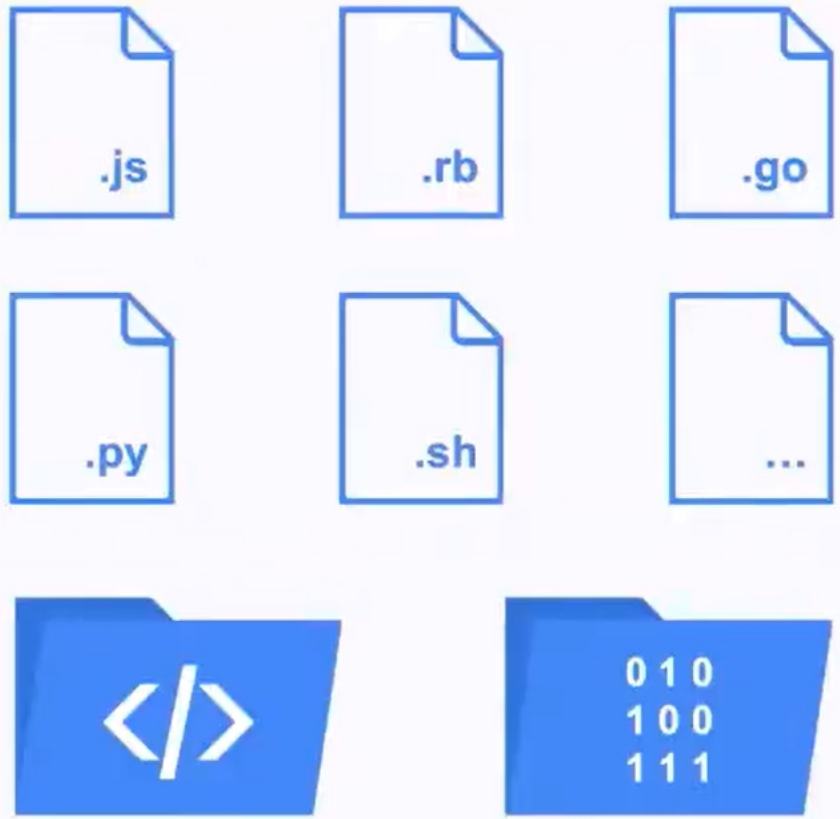
Service-based

Event-driven

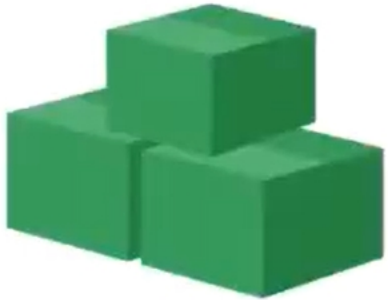
Portable

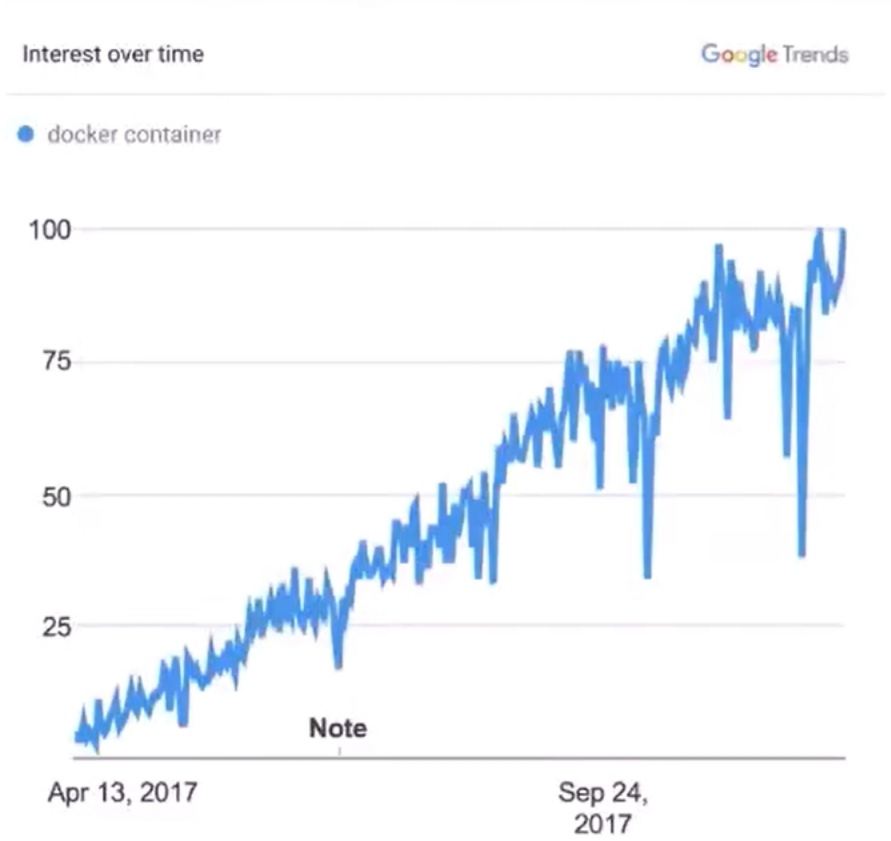
Containers

- Any Language
- Any Library
- Any Binary
- Ecosystem of base images



Containers:
An industry standard





Knative project

- Set of components (serving, eventing, ~~build~~)
- Ingredients for Serverless
- Solves for modern development patterns
- Implements learnings from Google, partners

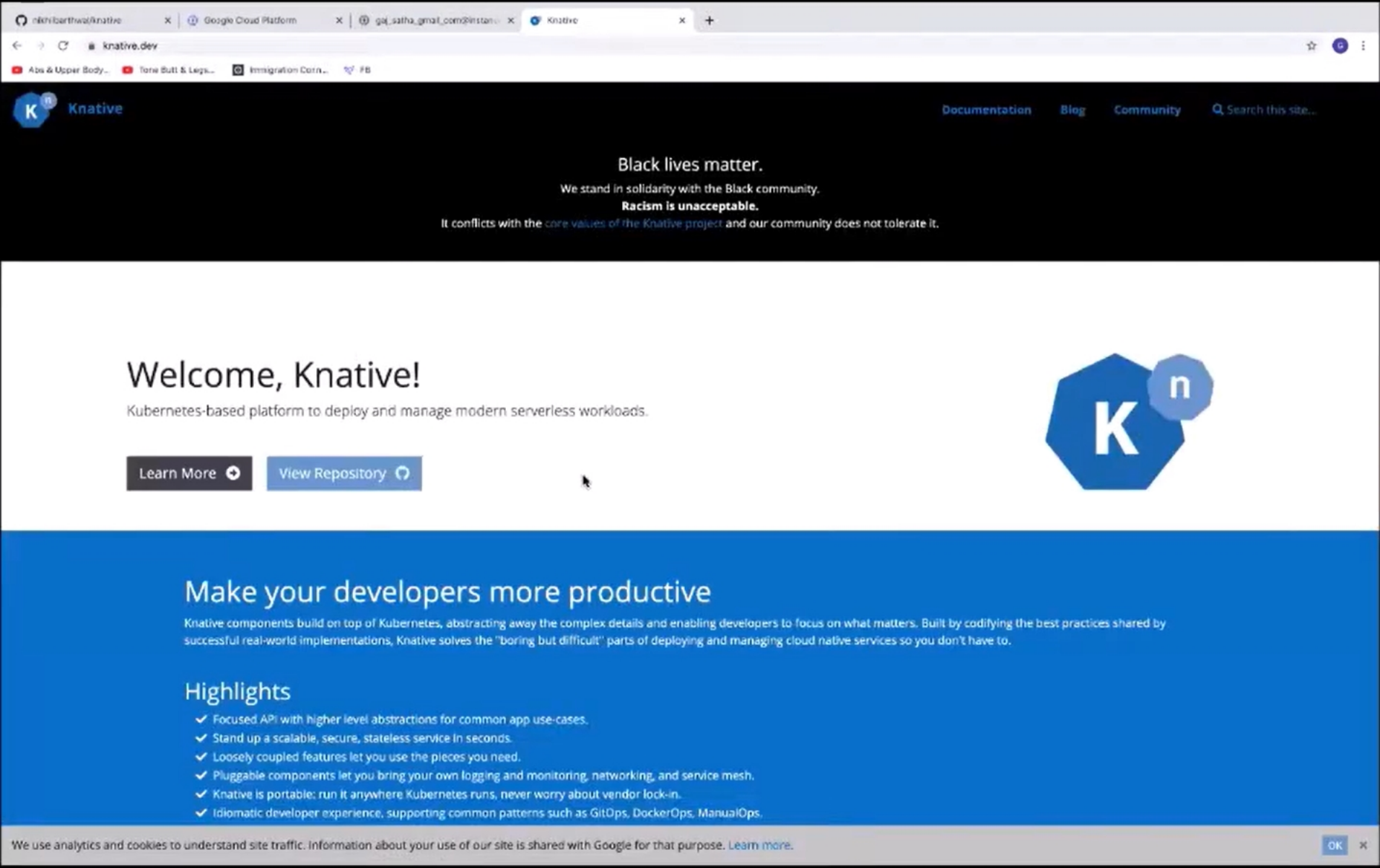
knative.dev

Pivotal

SAP

redhat

IBM


knative.dev
Knative
Documentation
Blog
Community
Search this site...
Black lives matter.
We stand in solidarity with the Black community.
Racism is unacceptable.
It conflicts with the core values of the Knative project and our community does not tolerate it.
Welcome, Knative!
Kubernetes-based platform to deploy and manage modern serverless workloads.
Learn More
View Repository
Make your developers more productive
Knative components build on top of Kubernetes, abstracting away the complex details and enabling developers to focus on what matters. Built by codifying the best practices shared by successful real-world implementations, Knative solves the "boring but difficult" parts of deploying and managing cloud native services so you don't have to.
Highlights
Focused API with higher level abstractions for common app use-cases.
Stand up a scalable, secure, stateless service in seconds.
Loosely coupled features let you use the pieces you need.
Pluggable components let you bring your own logging and monitoring, networking, and service mesh.
Knative is portable: run it anywhere Kubernetes runs, never worry about vendor lock-in.
Idiomatic developer experience, supporting common patterns such as GitOps, DockerOps, ManualOps.
We use analytics and cookies to understand site traffic. Information about your use of our site is shared with Google for that purpose. Learn more.
OK

Motivation

Developers want serverless

... just want to run their code.

... want to use their favorite languages and dependencies.

... don't want to manage the infrastructure.

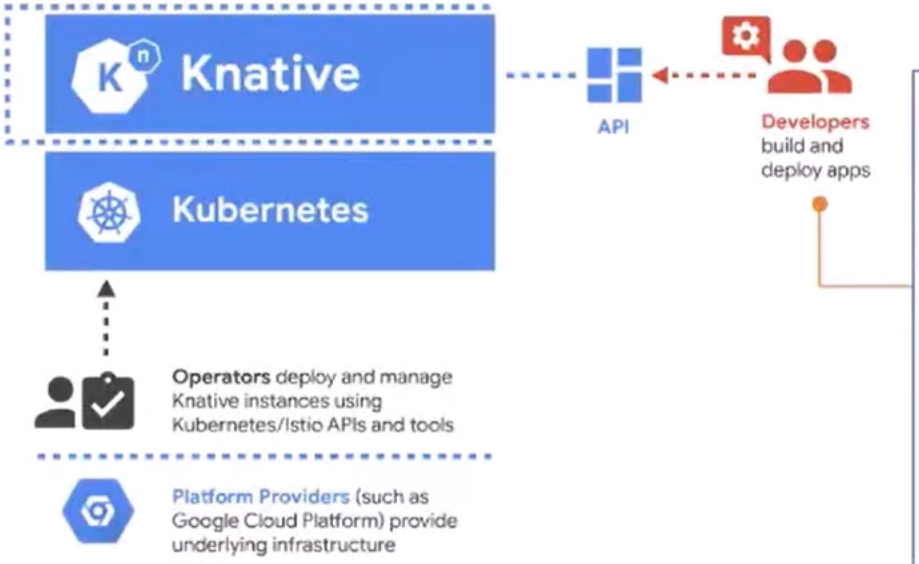
Operators want Kubernetes

Kubernetes is great orchestrating microservices

They love using GKE and not having to do operations for Kubernetes.

Kubernetes is not the right abstraction for their developers.

Knative for developer

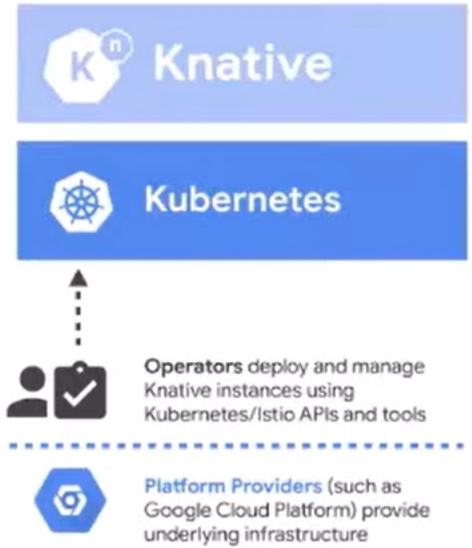


Want to: Write code

Don't have to

- Build docker image
- Upload image to registry
- Deploy service
- Expose to the internet
- Setup logging & monitoring
- Scale workload...

Knative for operator



Abstracts operational complexity, smooth infrastructure surface

Universal supported by all major Cloud providers, enables portability

Extendable platform with clear separation of concerns between operator and developer

Portability

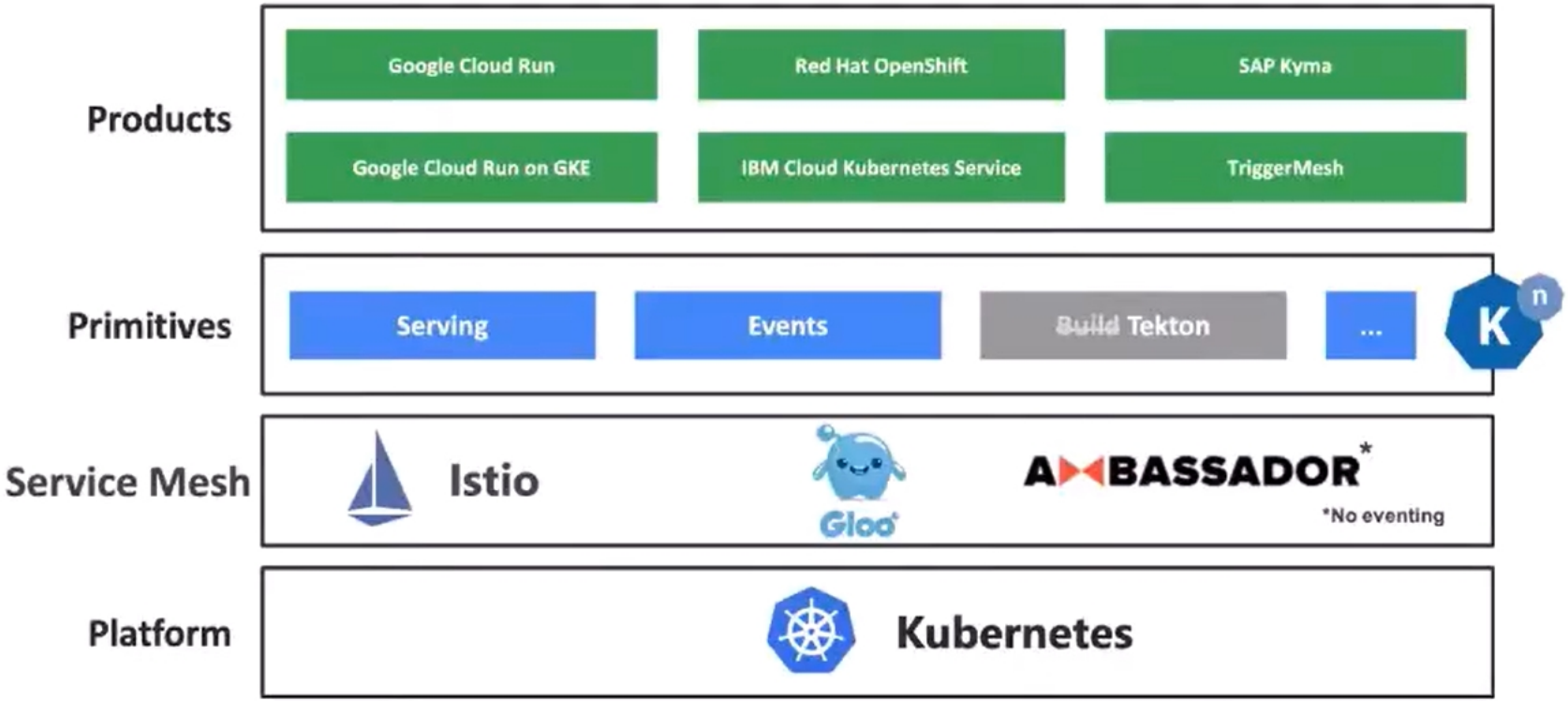
Kubernetes

Offered by virtually all
Cloud Service Providers

Knative

Codifies serverless, broad
contributor/user community

Knative stack



Serverless on Google Cloud

Cloud Run

Fully managed, deploy your workloads and don't see the cluster.

Cloud Run on GKE

Deploy into your GKE cluster, run serverless side-by-side with your existing workloads.

Knative Everywhere

Use the same APIs and tooling anywhere you run Kubernetes with Knative.

Knative Serving

02

Knative Serving

What is it?

- Rapid deployment of serverless containers
- Automatic (0-n) scaling
- Configuration and revision management
- Traffic splitting between revisions

Pluggable

- Connect to your own logging & monitoring platform, or use the built-in system
- Auto-scaler can be tuned or swapped out for custom code

Knative Serving Primitives

Knative Service

High level abstraction for the application

Configuration

Current/desired state of an application

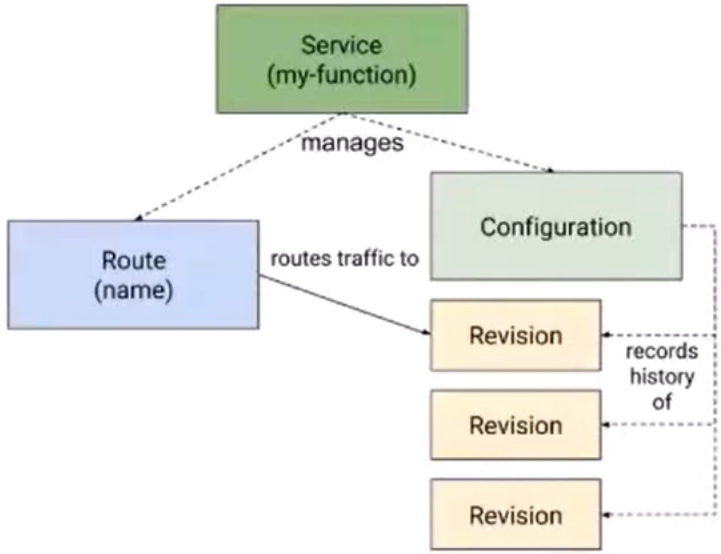
Code & configuration separated (a la 12-factor)

Revision

Point in time snapshots for your code and configuration

Route

Maps traffic to revisions



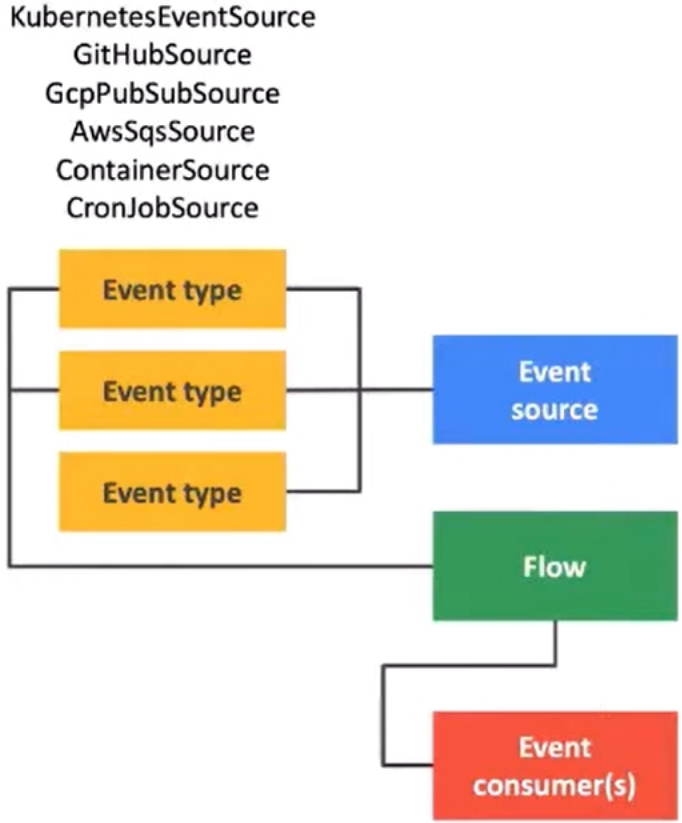
Knative Eventing

03

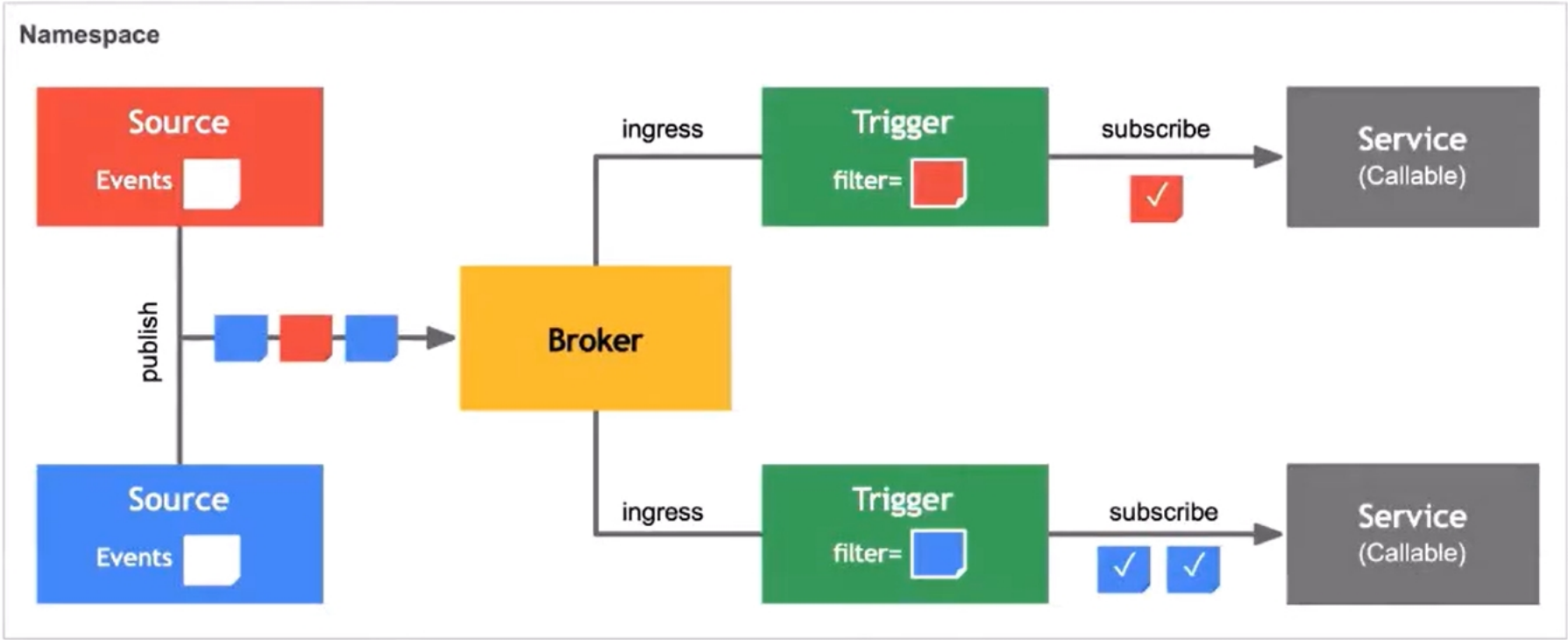
Knative Eventing

What is it?

- For loosely coupled, event-driven services
- Declaratively bind between event producers and Knative services
- Scales from just few events to live streams
- Custom event pipelines to connect with your own existing systems



Knative Eventing



Knative Event Sources

Name	Description
Apache Camel	Allows to use Apache Camel components for pushing events into Knative
Apache Kafka	Brings Apache Kafka messages into Knative
AWS SQS	Brings AWS Simple Queue Service messages into Knative
Cron Job	Uses an in-memory timer to produce events on the specified Cron schedule.
GCP PubSub	Brings GCP PubSub messages into Knative
GitHub	Brings GitHub organization/repository events into Knative
GitLab	Brings GitLab repository events into Knative.
Google Cloud Scheduler	Google Cloud Scheduler events in Knative when jobs are triggered
Google Cloud Storage	Brings Google Cloud Storage bucket/object events into Knative
Kubernetes	Brings Kubernetes cluster/infrastructure events into Knative

https://github.com/knative/docs/tree/master/docs/eventing/sources

Knative Eventing use-cases

- Cron job importer to wire up weekly report
- Process IoT Core events
- GCP PubSub (connector to many other GCP event sources)
- Actuate on Kubernetes events (beyond webhook)
- Declarative GitHub webhook processing

github.com/knative/docs/tree/master/docs/eventing/samples

Building

04

Knative Build (Pre 0.8)

Tekton Pipelines (Post 0.8)

Confidential & Proprietary

Knative build (Deprecated)

What is it?

- Go from source code to container images on repositories
- Build pipelines can have multiple steps and can push to different registries
- Builds run in containers in the cluster. No need for Docker locally

---

Primitives

- **Build:** Represents a single build job with 1 or more steps.
- **BuildTemplate:** A set of ordered and parameterized build steps.
- **ServiceAccount:** For auth with DockerHub etc

Tekton Pipelines

Primitives

- Task: Represents the work to be executed with 1 or more steps
- TaskRun: Runs the Task with supplied parameters
- Pipeline: A list of Tasks to execute in order
- ServiceAccount: For authentication with DockerHub etc.

What is it?

Kubernetes style resources for declaring CI/CD-style pipelines

Go from source code to container images on repositories

Build pipelines can have multiple steps and can push to different registries

Builds run in containers in the cluster. No need for Docker locally

github.com/tektoncd/pipeline

Knative Community

v0.8
Predictable Releases

55+
Contributing Companies

>6K
Pull Requests

~450
Individual Contributors

9
Working Groups

knative.teststats.cncf.io/d/8/dashboards

Knative value, today

1-step deploy

Build & deploy with less config/code

Source to container safely in your cluster

Auto-scale

Auto-scale your stateless container based workloads

Scale down to zero

Manage workloads

Automatically deploys containers and provision ingress

Go from source to URL

knative/changeconfig.md at m | Google Cloud Platform | ga_satha_gmail_com@instanc | Knative | Knative | Install Docker Desktop on Mac
github.com/nikhilbarthwal/knative/blob/master/docs/changeconfig.md
Abs & Upper Body.. | Tone Butt & Legs.. | Immigration Conse.. | FB

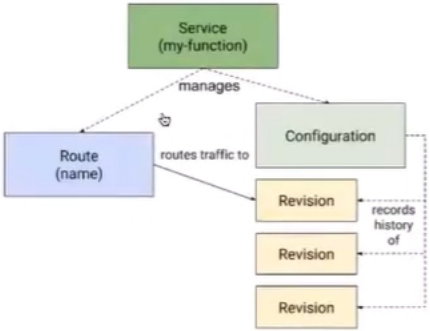
nikhilbarthwal Upgrade — Latest commit f3d635c on Jul 4 — History

1 contributor

85 lines (54 sloc) 2.96 KB — Raw | Blame

Change Configuration

In Knative Serving whenever you change Configuration of the Service, it creates a new Revision which is a point-in-time snapshot of code. It also creates a new Route and the new Revision will start receiving traffic.



Change environment variable

https://github.com/knative/serving/raw/master/docs/spec/images/object_model.png

```
  name: helloworld
  namespace: default
spec:
  template:
    metadata:
      name: helloworld-v1
    spec:
      containers:
        # Replace {username} with your actual DockerHub
        - image: docker.io/{username}/helloworld:v1
          env:
            - name: TARGET
              value: "v1"
  traffic:
  - tag: current
    revisionName: helloworld-v1
    percent: 100
  - tag: latest
    latestRevision: true
    percent: 0
```

Notice a couple of things:

1. The revision of the Service has now a specific name: `helloworld-v1`
2. There's a `traffic` section where we pin 100% of the traffic to the named revision.

In this setup, we can still deploy a new revision but that revision (`latest`) is not going to get the traffic

```
kubectl apply -f service-v1-pinned.yaml
```

You should see the new named revision created:

```
kubectl get revision

NAME                SERVICE NAME        GENERATION
helloworld-z9clz    helloworld-z9clz    1
helloworld-f4xvr    helloworld-f4xvr    2
helloworld-ln8rv    helloworld-ln8rv    3
helloworld-v1       helloworld-v1       4
```

And `helloworld-v1` is the one getting the traffic:

```
curl http://helloworld.default.$ISTIO_INGRESS.xip.io

Hello v1
```

Deploy a new version

Let's create a new revision. Create a service-v4.yaml file that has `TARGET` value of `v4` :

```
NAME                SERVICE NAME        GENERATION
helloworld-z9clz    helloworld-z9clz    1
helloworld-f4xvr    helloworld-f4xvr    2
helloworld-ln8rv    helloworld-ln8rv    3
helloworld-v1       helloworld-v1       4
helloworld-v4       helloworld-v4       5
```

But `helloworld-v1` is still one getting the traffic:

```
curl http://helloworld.default.$ISTIO_INGRESS.xip.io

Hello v1
```

You can verify that the new version is deployed by accessing the `latest` endpoint:

```
curl http://latest-helloworld.default.$ISTIO_INGRESS.xip.io

Hello v4
```

But the `current` one is `helloworld-v1` :

```
curl http://current-helloworld.default.$ISTIO_INGRESS.xip.io
```

knative/trafficsplitting.md at m | Google Cloud Platform | gaj_satha_gmail_com@instanc | Knative | Knative | Install Docker Desktop on Mac
github.com/n/khilbarthwal/knative/blob/master/docs/trafficsplitting.md
Abs & Upper Body... | Tone Butt & Legs... | Immigration Corra... | FB

```
  template:
    metadata:
      name: helloworld-v4
    spec:
      containers:
        # Replace {username} with your actual DockerHub
        - image: docker.io/{username}/helloworld:v4
          env:
            - name: TARGET
              value: "v4"
  traffic:
  - tag: current
    revisionName: helloworld-v1
    percent: 50
  - tag: candidate
    revisionName: helloworld-v4
    percent: 50
  - tag: latest
    latestRevision: true
    percent: 0
```

Apply the change:

```
kubectl apply -f service-v1v4-split.yaml
```

You should see roughly 50% of the requests split between revisions:

```
  - tag: current
    revisionName: helloworld-v1
    percent: 50
  - tag: candidate
    revisionName: helloworld-v4
    percent: 50
  - tag: latest
    latestRevision: true
    percent: 0
```

Apply the change:

```
kubectl apply -f service-v1v4-split.yaml
```

You should see roughly 50% of the requests split between revisions:

```
for i in {1..10}; do curl http://helloworld.default.$ISTIO_INGRESS.xip.io; sleep 1; done
Hello v1
Hello v4
Hello v1
Hello v4
```

Configure Autoscaling

You might have realized that autoscaler in Knative scales down pods to zero after some time. This is actually configurable through annotations. The type of autoscaler itself is also configurable. Autoscale Sample in Knative docs explains the details of autoscaler but let's recap the main points here as well.

There are two autoscaler classes built into Knative:

1. The default concurrency-based autoscaler which is based on the average number of in-flight requests per pod.
2. Kubernetes CPU-based autoscaler which autoscales on CPU usage.

The autoscaling can be bounded with `minScale` and `maxScale` annotations.

Create a 'Sleeping' service

Let's deploy a service to showcase autoscaling, by essentially sleeping for 4,000ms before responding to requests:

- C# Startup.cs

```
app.Run(async (context) =>
{
    Thread.Sleep(4000);
```

```
sleepingservice-00001-deployment-5865bc498c-w7qc7
```

And this pod will continue to run, even if there's no traffic.

Test autoscaling

Let's now send some traffic to our service to see that it scales up. Download and install Fortio if you don't have it.

Send some requests to our sleeping service:

```
fortio load -t 0 http://sleepingservice.default.$ISTIO_INGRESS.xip.io
```

After a while, you should see pods scaling up to 5:

```
kubectl get pods

sleepingservice-cphdq-deployment-5bf8ddb477-787sq
sleepingservice-cphdq-deployment-5bf8ddb477-b6ms9
sleepingservice-cphdq-deployment-5bf8ddb477-bmrds
sleepingservice-cphdq-deployment-5bf8ddb477-g2ssv
sleepingservice-cphdq-deployment-5bf8ddb477-kzt5t
```

Once you kill Fortio, you should also see the sleeping service scale down to 1!

```
  template:
    metadata:
      annotations:
        # Default: Knative concurrency-based autoscaling with
        # 100 requests in-flight per pod.
        autoscaling.knative.dev/class:  kpa.autoscaling.knative.dev
        autoscaling.knative.dev/metric: concurrency
        # Changed target to 1 to showcase autoscaling
        autoscaling.knative.dev/target: "1"

        # Alternative: Kubernetes CPU-based autoscaling.
        # autoscaling.knative.dev/class:  hpa.autoscaling.knative.dev
        # autoscaling.knative.dev/metric: cpu

        # Disable scale to zero with a minScale of 1.
        autoscaling.knative.dev/minScale: "1"
        # Limit max scaling to 5 pods.
        autoscaling.knative.dev/maxScale: "5"
    spec:
      containers:
        - image: gcr.io/nikhilbarthwal-knative/sleepingservice:v1
```

© 2020 GitHub, Inc. Terms Privacy Security Status Help Contact GitHub Pricing API Training Blog About

command line and user interface to quickly deploy and manage your serverless containers.

In this lab, we will see what it takes to deploy our Hello World Knative serving sample from the previous lab and deploy to Cloud Run. You'll be surprised how easy it is!

Get a project in a Cloud Run region

Cloud Run is currently available in `us-central1`, `us-east1`, `europe-west1`, and `asia-northeast1` regions. Make sure your project is in one of them:

```
gcloud config list

[compute]
region = us-central1
zone = us-central1a
```

If not, you can set the region with `gcloud config set` command or switch to a configuration with a new project in that region:

```
gcloud config configurations activate cloudrun-nikhilbarthwal
```

You also want to make sure that the Cloud Build and Cloud Run APIs are enabled:

Push container image to Google Container Registry

Cloud Run currently deploys images from Google Container Registry (GCR) only. In Hello World Knative serving sample, we built and pushed the container image to Docker Hub. We need to push the same image to GCR.

First, set an environment variable for your project:

```
export PROJECT_ID=nikhilbarthwal-knative
```

In helloworld folder, go to the folder for the language of your choice (csharp, python). Run the following command:

```
gcloud builds submit --tag gcr.io/${PROJECT_ID}/helloworld:v1
```

This builds and pushes the image to GCR using Cloud Build.

Deploy

Let's finally deploy our service to Cloud Run, it's a single gcloud command:

```
gcloud run deploy --image gcr.io/${PROJECT_ID}/helloworld:v1
```

Let's finally deploy our service to Cloud Run, it's a single gcloud command:

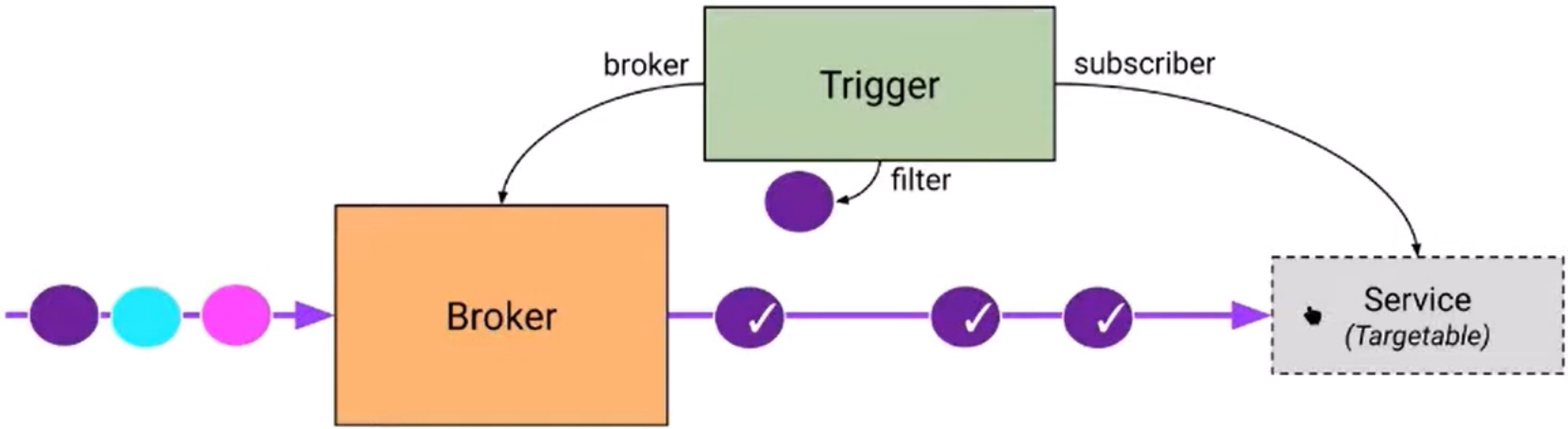
```
gcloud run deploy --image gcr.io/${PROJECT_ID}/helloworld:v1

Please choose a target platform:
 [1] Cloud Run (fully managed)
 [2] Cloud Run on GKE
 [3] a Kubernetes cluster
 [4] cancel
Please enter your numeric choice:  1

To specify the platform yourself, pass `--platform managed`. Or, to make this the default target plat

Service name (helloworld):
Deploying container to Cloud Run service [helloworld] in project [nikhilbarthwal-knative] region [eur
✓ Deploying... Done.
  ✓ Creating Revision...
  ✓ Routing traffic...
Done.
Service [helloworld] revision [helloworld-00011] has been deployed and is serving 100 percent of traf
```

This creates a Cloud Run service and a revision for the current configuration. In the end, you get a url that you can browse to.

You can also see the service in Cloud Run console:

Hello World Eventing

In Knative Eventing, you'd typically use Broker and Trigger to receive and filter messages. This is explained in more detail on Knative Eventing page:



Knative Eventing has a few different types of event sources (Kubernetes, GitHub, GCP Pub/Sub etc.) that it can listen.

Knative Eventing has a few different types of event sources (Kubernetes, GitHub, GCP Pub/Sub etc.) that it can listen.

In this tutorial, we will create our own events using an event producer and listen and log the messages in an event consumer. This tutorial is based on Getting Started with Knative Eventing with slight modifications to make it easier.

Knative Eventing

First, make sure Knative Eventing is installed:

```
kubectl get pods -n knative-eventing

NAME                                    READY   STATUS    RESTARTS   AGE
broker-controller-b85986f7d-xqj2k       1/1     Running   0          4m30s
eventing-controller-58b889c4b4-dnf62    1/1     Running   2          8m55s
eventing-webhook-5549c4b664-jc6x6       1/1     Running   2          8m53s
imc-controller-64cfbf485d-7h2k7         1/1     Running   0          4m32s
imc-dispatcher-5fc7ccf7d8-729ds         1/1     Running   0          4m32s
```

If not, you can follow the instructions on Knative Eventing Installation page.

Broker

github.com/nikhilbarthwal/knative/blob/master/docs/helloworldeventing.md

```
kubectl get broker

NAME      READY   REASON   URL
default   True             http://broker-ingress.knative-eventing.svc.cluster.local/default/default
```

Consumer

Event Display

For event consumer, we'll use an Event Display service that simply logs out received messages. Follow the instructions for your preferred language to create a service to log out messages:

- Create Event Display - C#
- Create Event Display - Python

Docker image

Build and push the Docker image (Replace `{project-id}` with your GCP project name):

```
docker build -t gcr.io/{project-id}/event-display:v1 .
```

```
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: event-display
spec:
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/minScale: "1"
    spec:
      containers:
        - image: gcr.io/nikhilbarthwal-knative/event-display:v1
```

You can only access the Broker from within your Eventing cluster. Normally, you would create a Pod within that cluster to act as your event producer. In this case, we'll simply create a Pod with curl installed and use curl to manually send messages.

Curl Pod

Create a curl-pod.yaml file.

Create the pod:

```
kubectl apply -f curl-pod.yaml

pod/curl created
```

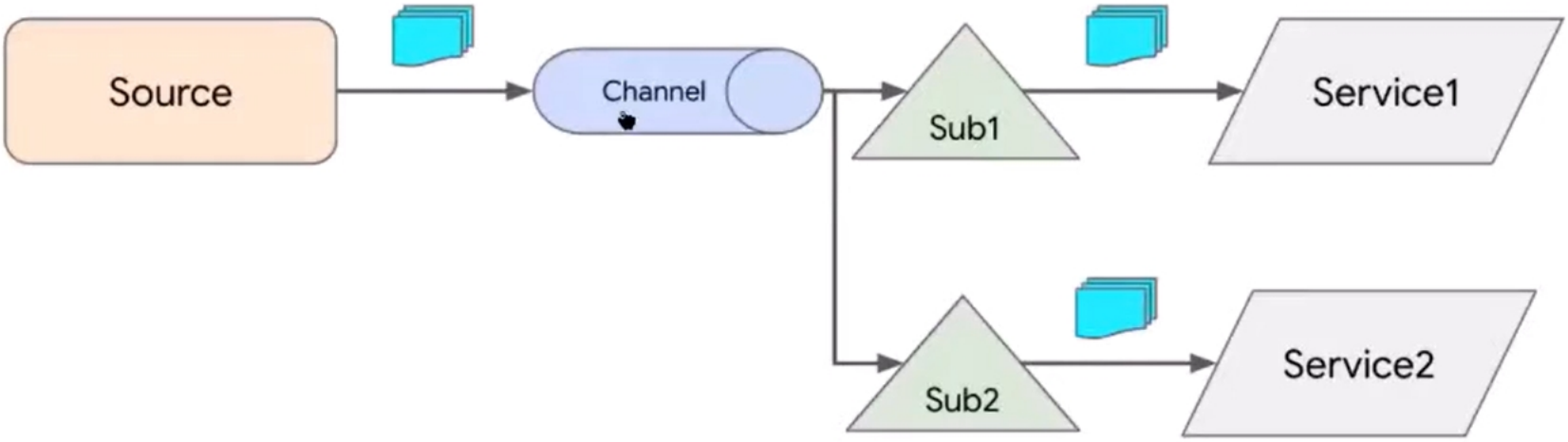
Send events to Broker

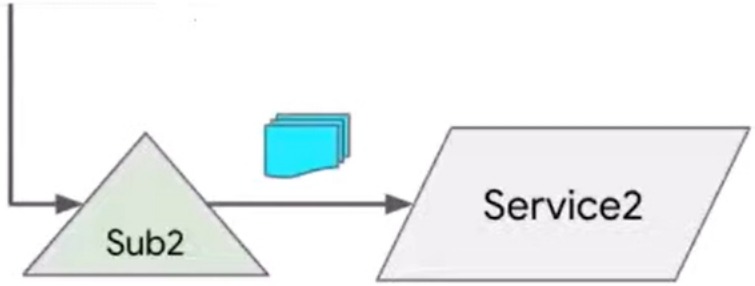
SSH into the pod:

```
kubectl attach curl -it

Defaulting container name to curl.
Use 'kubectl describe pod/' to see all of the containers in this pod.
If you don't see a command prompt, try pressing enter.
```

Complex Delivery

In Simple Delivery example, we see how an Event Source can send a message directly to a Service. This works in 1-1 case but if you need to fan out from an Event Source to multiple Services, you need to use Channels and Subscriptions.





Channel

Channel is the persistence and forwarding layer. Knative supports a number of Channels with different persistence guarantees.

For this example, we'll use `InMemoryChannel`. It's not meant for production but ok for testing.

Define channel.yaml.

Create the channel:

```
kubectl apply -f channel.yaml

inmemorychannel.messaging.knative.dev/channel created
```

View/Edit this page
Create documentation issue
Create project issue

Available Channels

This is a non-exhaustive list of the available Channels for Knative Eventing.

Notes:

- Inclusion in this list is not an endorsement, nor does it imply any level of support.

Name	Status	Support	Description
GCP PubSub	Proof of Concept	None	Channels are backed by GCP PubSub.
InMemoryChannel	Proof of Concept	None	In-memory channels are a best effort Channel. They should NOT be used in Production. They are useful for development.
KafkaChannel	Proof of Concept	None	Channels are backed by Apache Kafka topics.
NatssChannel	Proof of Concept	None	Channels are backed by NATS Streaming.

Feedback

Was this page helpful?

Yes No

We use analytics and cookies to understand site traffic. Information about your use of our site is shared with Google for that purpose. Learn more.
OK

Connect services to the channel with subscriptions.

Define subscription1.yaml.

Define another subscription2.yaml for the second subscription.

Create the subscriptions:

```
kubectl apply -f subscription1.yaml -f subscription2.yaml

subscription.messaging.knative.dev/subscription1 created
subscription.messaging.knative.dev/subscription2 created
```

Verify

Check the running pods:

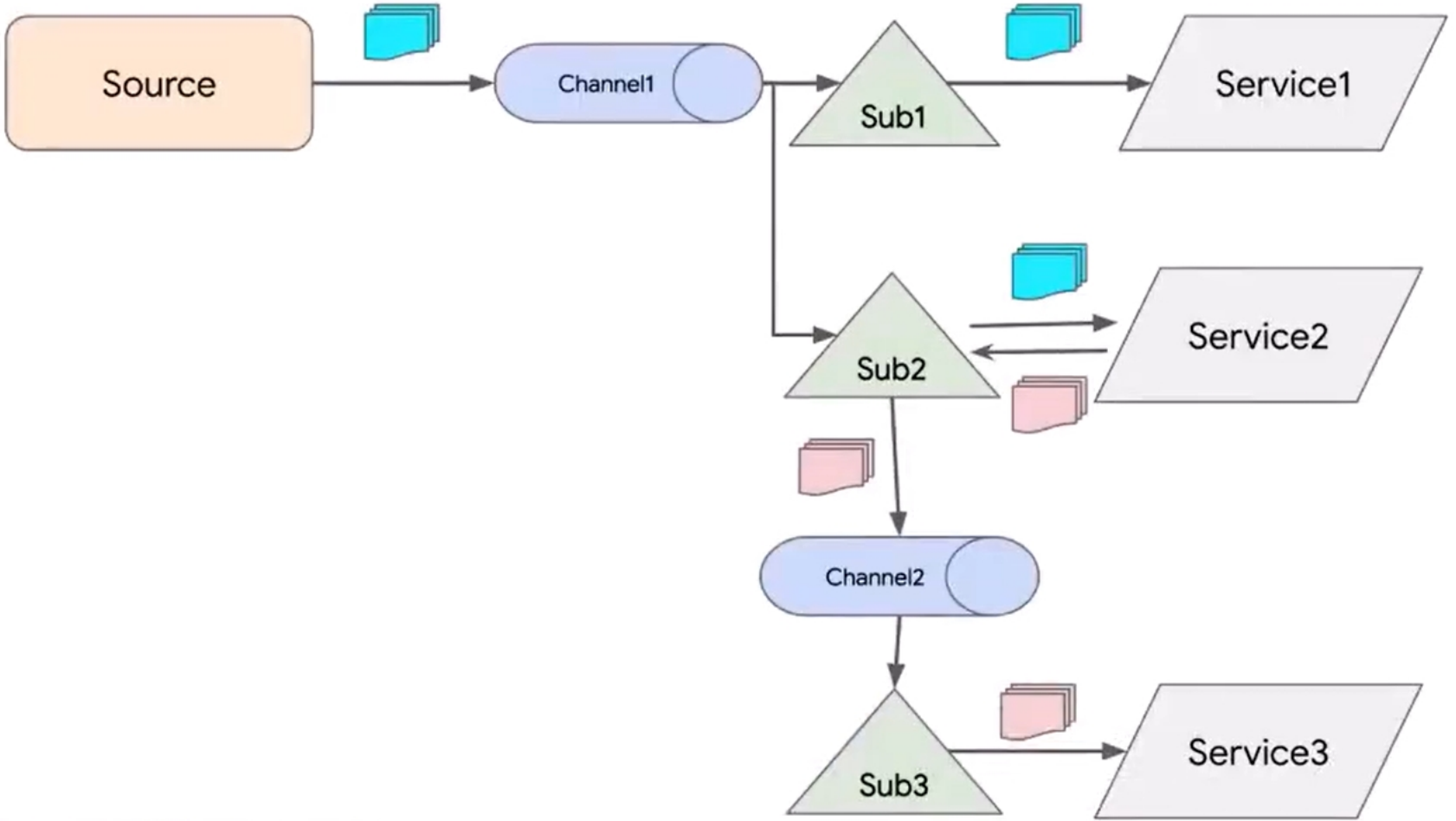
```
kubectl get pods

NAME                                                    READY STATUS    RESTARTS   AGE
cronjobsource-source-3c465b71-4bbe-4611-a816-a73a75cf8681-66ps5   1/1     Running   0          2m54s
service1-x25xg-deployment-55f957448-rtljx               2/2     Running   0          2m57s
```

```
kubectl get pods
```

Check the logs of the services. You should see messages from the CronJobSource in each service:

```
kubectl logs {container-name} -c user-container

[2020-10-12 04:11:02 +0000] [1] [INFO] Starting gunicorn 20.0.4
[2020-10-12 04:11:02 +0000] [1] [INFO] Listening at: http://0.0.0.0:8080 (1)
[2020-10-12 04:11:02 +0000] [1] [INFO] Using worker: threads
[2020-10-12 04:11:02 +0000] [8] [INFO] Booting worker with pid: 8
[2020-10-12 04:11:02 +0000] [8] [INFO] Event Display starting
[2020-10-12 04:11:03 +0000] [8] [INFO] Event Display received event: {"message":"Hello world from ping
[2020-10-12 04:12:00 +0000] [8] [INFO] Event Display received event: {"message":"Hello world from ping
```

© 2020 GitHub, Inc. Terms Privacy Security Status Help Contact GitHub Pricing API Training Blog About


Source
Channel1
Sub1
Service1
Sub2
Service2
Channel2
Sub3
Service3