Exploring KNative

Video: https://youtu.be/DZQOgIWN1pE

Hello

Pah**Levi** Fikri Auliya (HIJUP, CTO)

- Uses Kubernetes since 2 years ago, in AWS & GCP
- 95%+ of our system runs on K8S
- Very recently exploring KNative
- This presentation is based on my experiments and various KNative presentations & docs

K8S challenges for developer

1. Write & test a code
2. Write Dockerfile (learn Docker)
3. Build Docker image (either locally/on cloud)
4. Run and test the Docker image
5. Upload Docker image to registry
6. Write deployment.yaml, service.yaml, config.yaml, secret.yaml (learn K8s)
7. kubectl apply -f *, if failed => repeat step 4
8. Setup logging & monitoring (learn Istio)
9. Scale

What developer wants

1. Write & test a code
2. Deploy

PaaS approach

- Deployment is simple**:** git push production master (e.g. Heroku)
- No Dockerfile & *.yamls needed
- Scaling is automatic
- Logging and monitoring are prebuilt

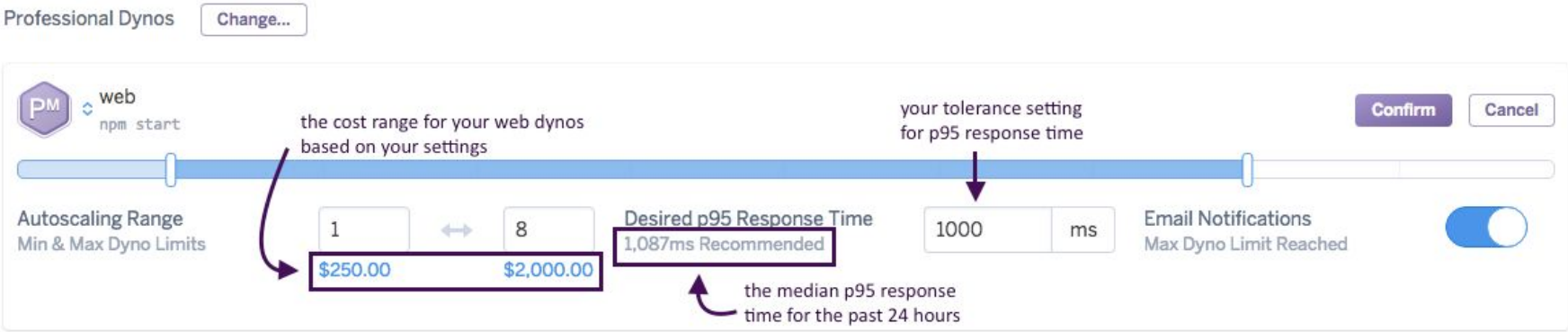


Image source https://blog.heroku.com/heroku-autoscaling

Hello KNative

Why KNative?

KNative offers a set of reusable components that focuses on **solving many mundane** but difficult tasks such as:

- Orchestrating source-to-container workflows
- Routing and managing traffic during deployment
- Auto-scaling your workloads
- Binding running services to eventing ecosystems.

What is KNative?

Kubernetes based building blocks for serverless workload

- **Build**: source-to-container build orchestration
- **Eventing**: management & delivery of event
- **Serving**: request-driven compute that can scale to 0

Architecture

Image source: https://github.com/knative/docs

1. KNative Build

- Build Docker image in Kubernetes cluster (Kaniko: https://github.com/GoogleContainerTools/kaniko)
- No need for Docker on dev machine
- Faster image push
- Building blocks for CI/CD

Build

```yaml
apiVersion: build.knative.dev/v1alpha1
kind: Build
metadata:
  name: example-build
spec:
  serviceAccountName: build-auth-example
  source:
    git:
      url: https://github.com/example/build-example.git
      revision: master
  steps:
  - name: ubuntu-example
    image: ubuntu
    args: ["ubuntu-build-example", "SECRETS-example.md"]
  steps:
  - image: gcr.io/example-builders/build-example
    args: ['echo', 'hello-example', 'build']
```

BuildTemplate

```
spec:
  source:
    git:
      url: https://github.com/mchmarny/rester-tester.git
      revision: master
  template:
    name: dockerfile-build-and-push
    arguments:
      - name: IMAGE
        value: gcr.io/my-project/rester-tester
```

Potential: Buildpack

```
spec:
  source:
    git:
      url: https://github.com/fikriauliya/my-ruby-project.git
      revision: master
  template:
    name: ruby-build
    arguments:
      - name: VERSION
        value: 2.6
      - name: IMAGE
        value: gcr.io/my-project/my-ruby-project
```

2. KNative Eventing

- Composable primitive for late-binding event sources + consumers
- Currently available sources:
  - KubernetesEventSource
  - GitHubSource
  - GcpPubSubSource
  - AwsSqsSource
  - ContainerSource

3. KNative Serving

Built on top of K8s + Istio to support deploying and serving serverless apps/functions

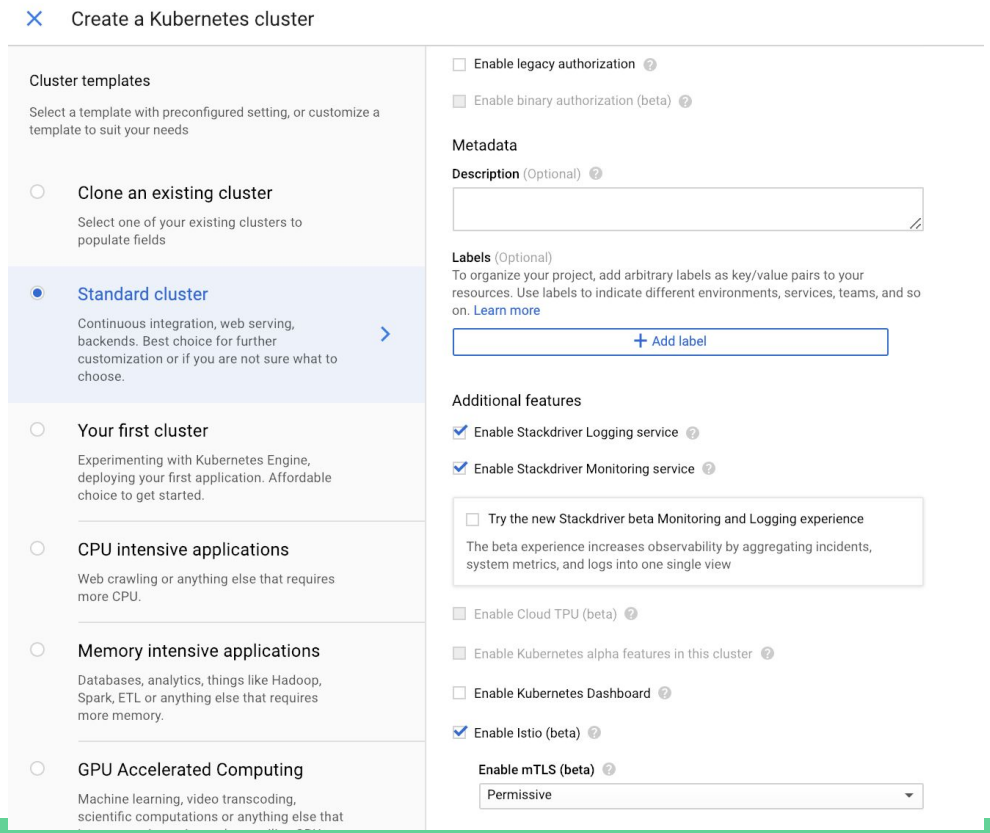
- Rapid deployment of serverless containers
- Automatic scaling up and down to **0**
- Routing and network programming for Istio components
- Point-in-time snapshots of deployed code and configurations

Istio

Istio is an open source independent service mesh that provides the fundamentals you need to successfully run a distributed microservice architecture.

- Securing service communications
- Logging, monitoring, and keeping services operational
- Traffic management and policy control

Istio status



Istio diagram

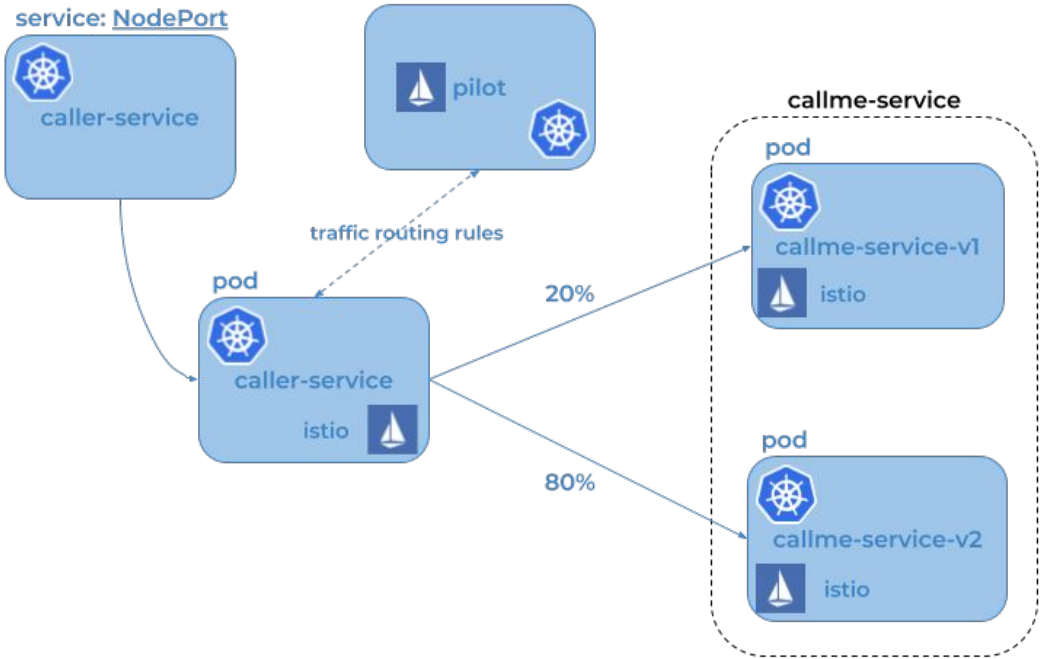


Image source:
https://dzone.com/articles/service-mesh-with-istio-on-kubernetes-in-5-steps

Istio sample

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
 name: percentage-frontend-gateway
spec:
 hosts:
 - "percentage-playground.levifikri.com"
 http:
 - route:
   - destination:
       host: frontend.playground.svc.cluster.local
       subset: v1-canary
     weight: 10
   - destination:
       host: frontend.playground.svc.cluster.local
       subset: v1
     weight: 90
```

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
 name: header-frontend-gateway
spec:
 hosts:
 - "header-playground.hijup.com"
 http:
 - match:
   - headers:
       canary:
         exact: "true"
   route:
   - destination:
       host: frontend.playground.svc.cluster.local
       subset: v1-canary
 - route:
   - destination:
       host: frontend.playground.svc.cluster.local
       subset: v1
```

Demo: helloworld

Canary Deployment

Intro to Canary Release

- “Canary release is a technique to **reduce the risk** of introducing a new software version in production by **slowly rolling out** the change to a **small subset of users** before rolling it out to the entire infrastructure and making it available to everybody” - Danilo Sato
- Related: https://medium.com/pujanggateknologi/canary-deployment-dengan-istio-d6ef55db155e

Illustration

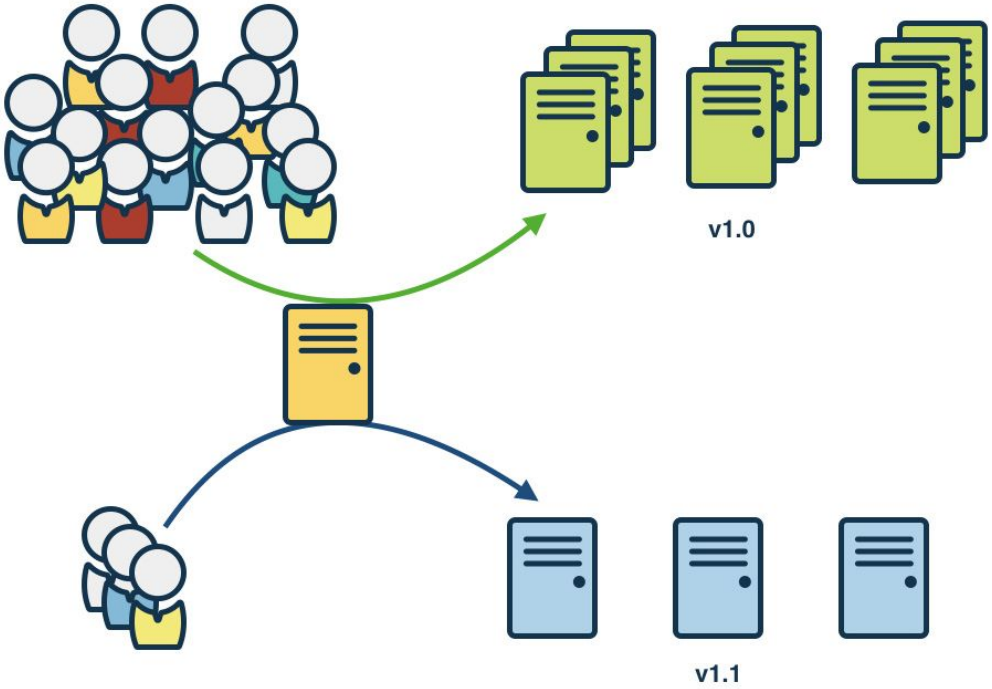


Image source: https://www.gocd.org/2017/08/15/canary-releases/

Demo: canary

Autoscaling

Autoscaling config (optional)

```
spec:
  runLatest:
    configuration:
      revisionTemplate:
        metadata:
          annotations:
            # Knative concurrency-based autoscaling (default).
            autoscaling.knative.dev/class:  kpa.autoscaling.knative.dev
            autoscaling.knative.dev/metric: concurrency
            # Target 10 requests in-flight per pod.
            autoscaling.knative.dev/target: "10"
            # Disable scale to zero with a minScale of 1.
            autoscaling.knative.dev/minScale: "1"
            # Limit scaling to 100 pods.
            autoscaling.knative.dev/maxScale: "100"
```

Let’s attack

Attack

```
GET http://35.187.241.194?sleep=1000&prime=10000&bloat=5
Host: autoscale.default.example.com
```

```
cat attack | ~/go/bin/vegeta attack -rate=300 -duration=5s \
   | tee results.bin | ~/go/bin/vegeta report
```

Demo: autoscaling

Conclusion

Conclusion

1. Write & test a code
2. ~~Write Dockerfile (learn Docker)~~
3. ~~Build Docker image (either locally/on cloud)~~
4. ~~Run and test the Docker image~~
5. ~~Upload Docker image to registry~~
6. ~~Write deployment.yaml, service.yaml, config.yaml, secret.yaml (learn K8s)~~
7. Use buildpack
8. Write service.yaml
9. kubectl apply -f *, if failed => repeat step 7
10. ~~Setup logging & monitoring (learn Istio)~~
11. ~~Scale~~

Related Projects

Related projects

- Kaniko
- Prow
- Jenkins-X (Prow + Skaffold + Nexus + Helm + ChartMuseum)
- Skaffold
- Draft
- https://github.com/fikriauliya/knative-demo

Thank you

https://medium.com/@pahlevifikriauliya

pahlevi@hijup.com

This presentation is available here: http://bit.ly/knative