

Cloud Native CI/CD with Knative and Tekton Pipelines

Andrea Frittoli

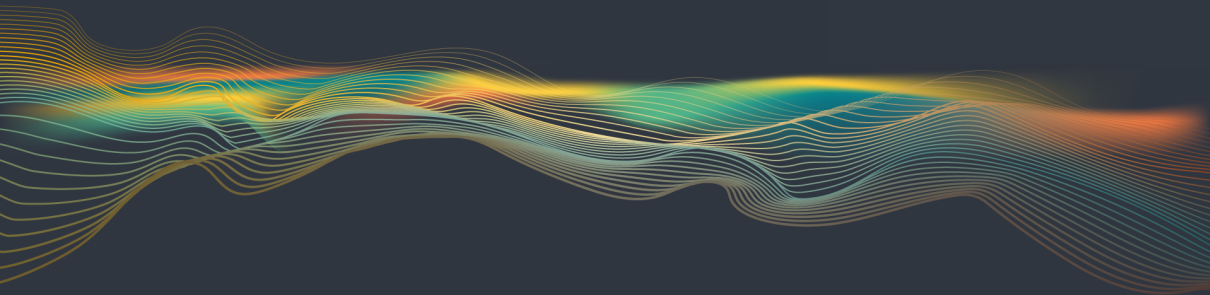
Developer Advocate

andrea.frittoli@uk.ibm.com

@blackchip76

Devoxx France, 8ème édition

A Bit of History



Knative

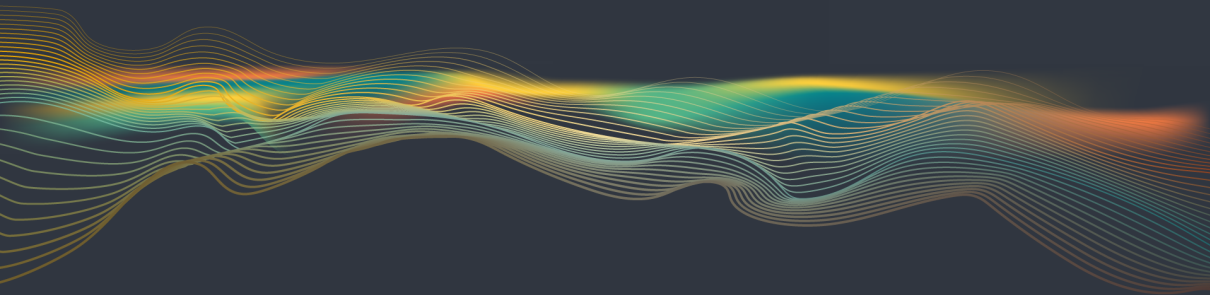
- Beginning of 2018...
- Knative:
 - Build
 - Eventing
 - Serving
- OpenSource
- Contributors:
 - Google
 - Pivotal
 - IBM
 - RedHat
 - Cloudbees
 - ...and others



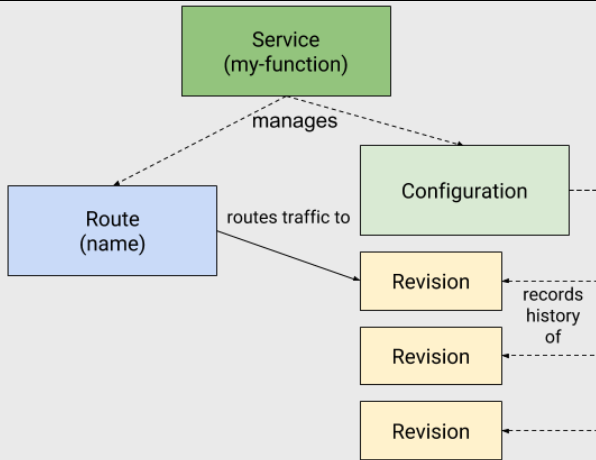
Community

- *Valid for Knative. Tekton TBD.*
- Steering Committee (SC)
- Technical Oversight Committee (TOC)
- Various Contribution profiles
- Design, issues: on GitHub
- Communication:
 - Weekly video meetings, recorded, Build WG
 - Asynch: Knative Users / Developers ML
 - Sync: slack.knative.dev

Knative Serving



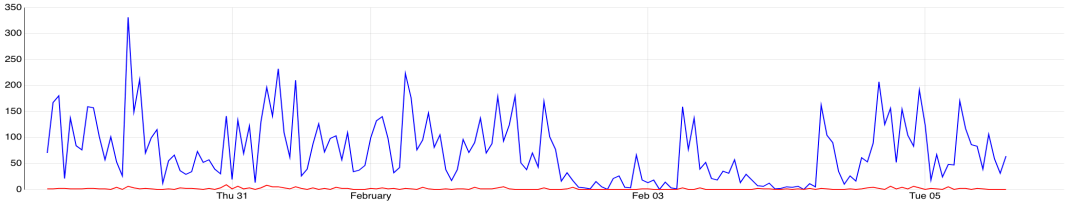
Knative Serving



OpenStack Health

is a dashboard for visualizing test results of OpenStack CI jobs.

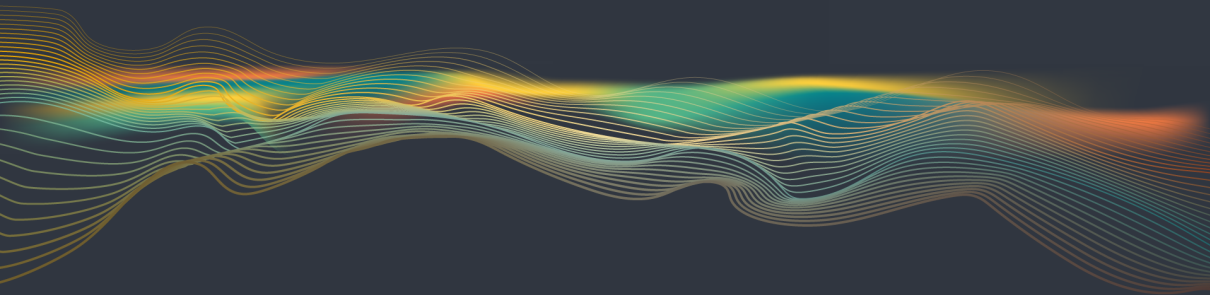
Total Jobs



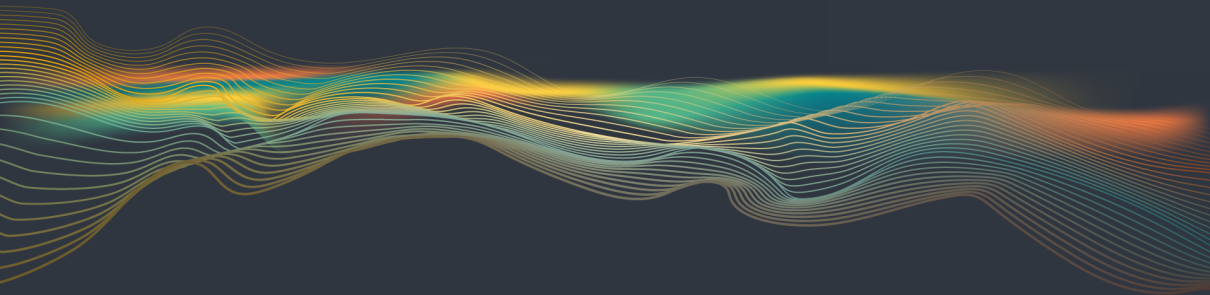
Job Failure Rate



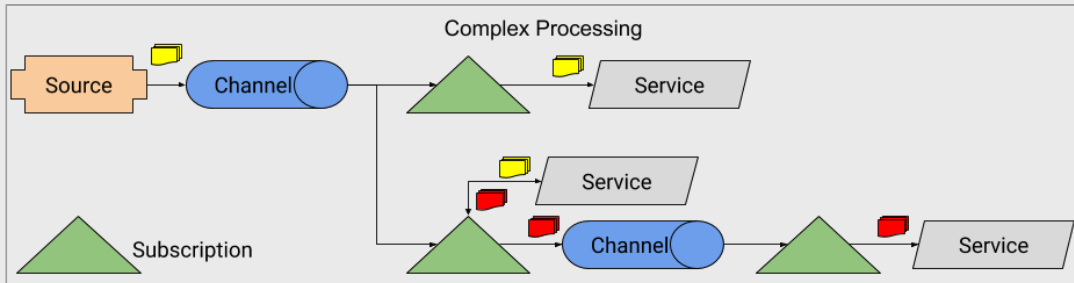
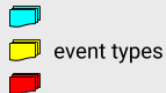
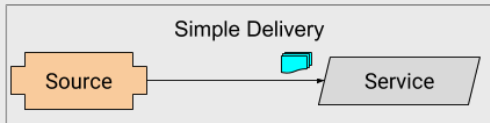
Knative Serving Demo



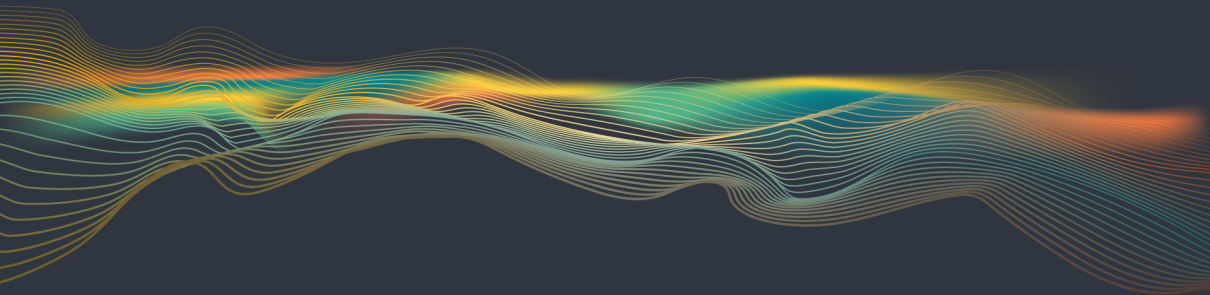
Knative Eventing



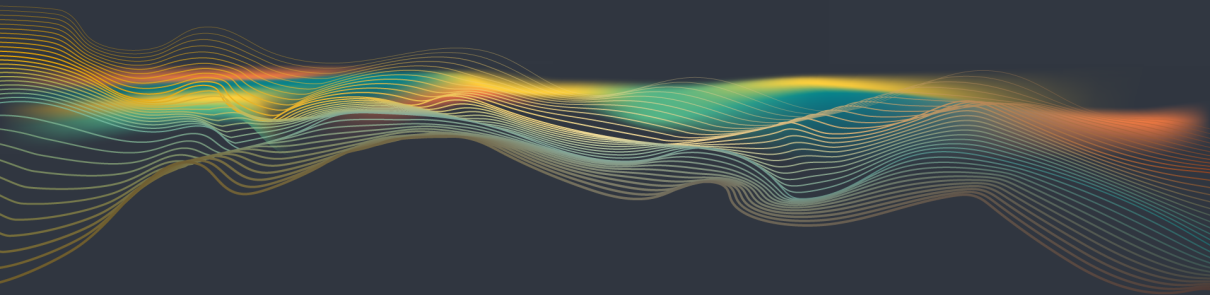
Knative Eventing



Knative Eventing Demo



Knative Build and Pipelines



Knative Build

TBD

Going further

TBD Diagram of complex pipeline

~Sept 2018: Knative Pipelines

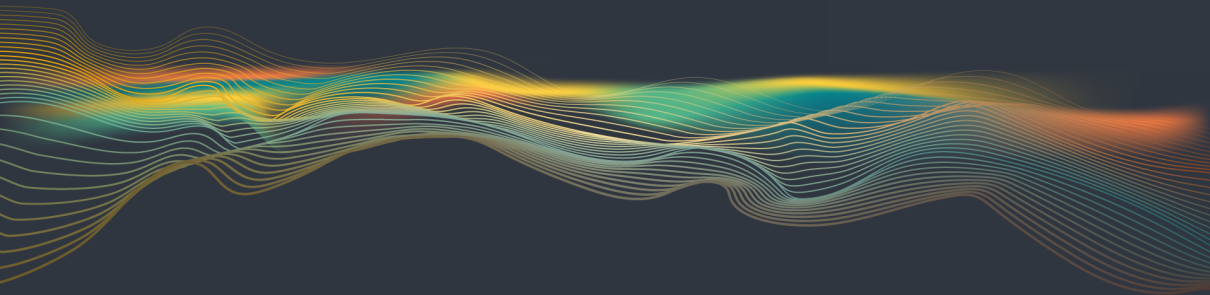


Latest news!

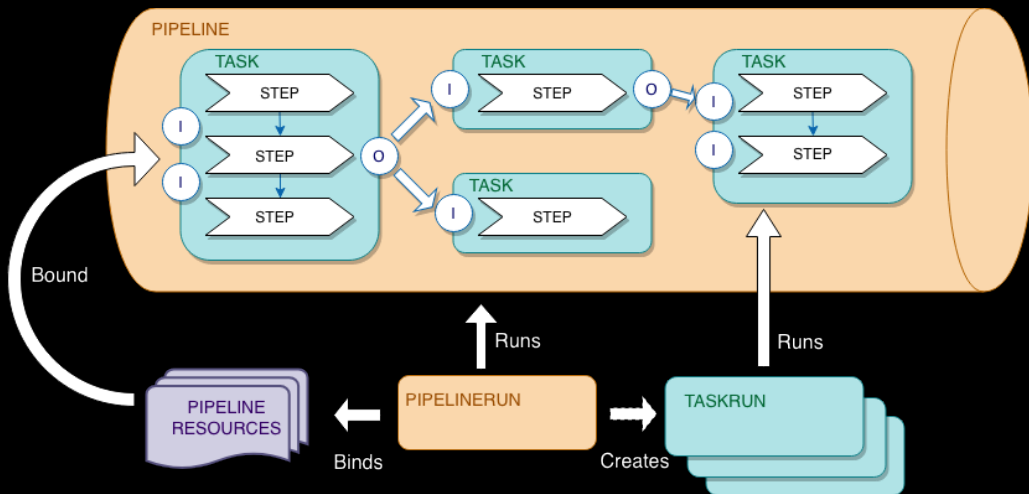
- Tekton pipelines
- Focus on CI/CD
- @CD Foundation
- Deploy “anywhere”
- Compatible with Knative Build



Tekton Pipelines



Cloud Native Pipelines



Using Kaniko

- Features:
 - Build from Context and Dockerfile
 - Unprivileged
 - Reproducible
 - Remote caching of layers
 - Base images caching (warmer)

- Dockefile?
 - Most common changes last
 - Careful with COPY/ADD
 - Remove what you don't need



Using Kaniko

Source to Image (spec only):

```
inputs:
  resources:
    - name: workspace
      type: git
  params:
    - name: pathToDockerFile
      default: Dockerfile
    - name: pathToContext
      default: .
    - name: useImageCache
      default: "true"
    - name: imageTag
      default: "default"
outputs:
  resources:
    - name: builtImage
      type: image
volumes:
  - name: kaniko-base-image-cache
    persistentVolumeClaim:
      claimName: kaniko-base-image-cache
steps:
  - name: build-and-push
    image: gcr.io/kaniko-project/executor
    command:
      - /kaniko/executor
```

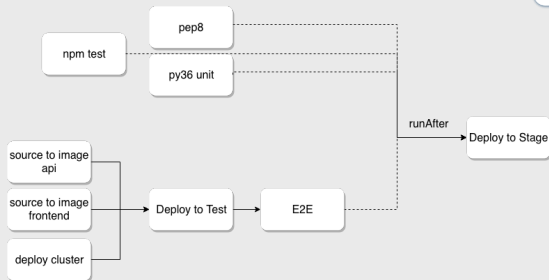
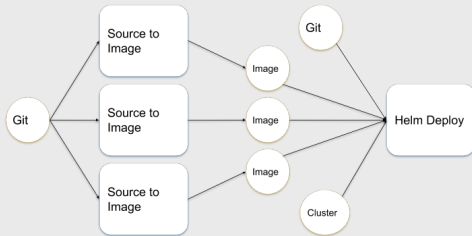
```
args:
  - --cache=${inputs.params.useImageCache}
  - --cache-dir=/cache
  - --dockerfile=${inputs.params.pathToDockerFile}
  - --reproducible
  - --destination=${outputs.resources.builtImage.url}:
    ${inputs.params.imageTag}
  - --context=/workspace/workspace/${inputs.params.
    pathToContext}
volumeMounts:
  - name: kaniko-base-image-cache
    mountPath: /cache
```

Cache Warmer (spec only):

```
volumes:
  - name: kaniko-base-image-cache
    persistentVolumeClaim:
      claimName: kaniko-base-image-cache
steps:
  - name: prepare-cache
    image: gcr.io/kaniko-project/warmer
    args:
      - --cache-dir=/cache
      - --image=python:3.6-slim-stretch
      - --image=postgres:alpine
      - --image=nginx:latest
    volumeMounts:
      - name: kaniko-base
        mountPath: /cache
```

Inputs, Outputs & DAG

- Steps are sequential
- Tasks are a Directed Acyclic Graph
- Order defined by:
 - *from*: input from another task's output
 - *runAfter*: enforced task ordering



CD Pipeline as code

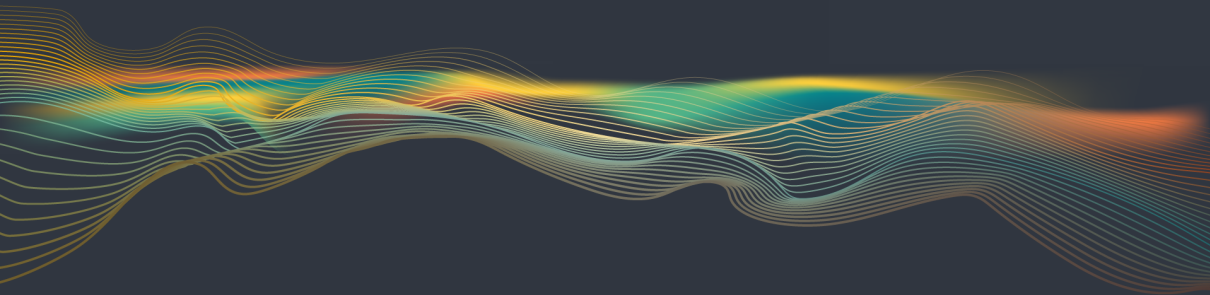
- Pipeline and Tasks in git (YAML)
- Parameters for env/run specific
- Security?

```
apiVersion: tekton.dev/v1alpha1
kind: PipelineResource
metadata:
  name: health-helm-git-knative
  labels:
    tag: agreatrelease
spec:
  type: git
  params:
    - name: revision
      value: knative
    - name: url
      value: https://github.com/afrittoli/health-helm
```

```
metadata:
  name: mycluster
spec:
  type: cluster
  params:
    - name: name
      value: mycluster
    - name: url
      value: https://mycluster.containers.cloud.ibm.com
    - name: username
      value: admin
  secrets:
    - fieldName: token
      secretKey: tokenKey
      secretName: cluster-secrets
    - fieldName: cadata
      secretKey: cadataKey
      secretName: cluster-secrets
```

```
metadata:
  name: health-api-image
spec:
  type: image
  params:
    - name: url
      value: registry.ng.bluemix.net/andrea/health-api
```

Under the Hood



Custom Resources

CRDs: Task(Run), Pipeline(Run), PipelineResource

Services in the *tekton-pipelines* namespace:

- Webhook Service: resource validation
- Controller Service:
 - Handles inputs and outputs
 - Calculates the DAG
 - Provisions pods and containers

Custom Resource Provisioning:

- Via YAML
- Via Go API
- Labels!

Pods, Entrypoints & Volumes

Steps (of a Task):

- Containers in one POD (single node)
- Any container image
- Entrypoint re-written
- Serial execution
- Resource allocation?

TaskRun:

- Provisions a POD
- Deploys entrypoint tool
- Input/output containers
- User containers (steps)

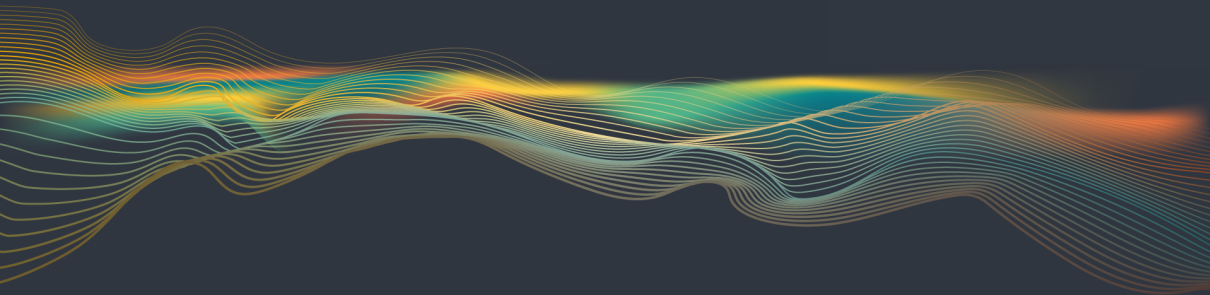
Volumes:

- EmptyDir for workspace/home
- Tools (entrypoint)
- Secrets
- Any user ConfigMap / Volume
- (Optionally) Pipeline Share

PipelineRun:

- Several PODs, different nodes
- Shared storage: PVC or GCS

Tekton and Knative Serving



Pipelines and Knative Build



CI with Tekton Pipelines

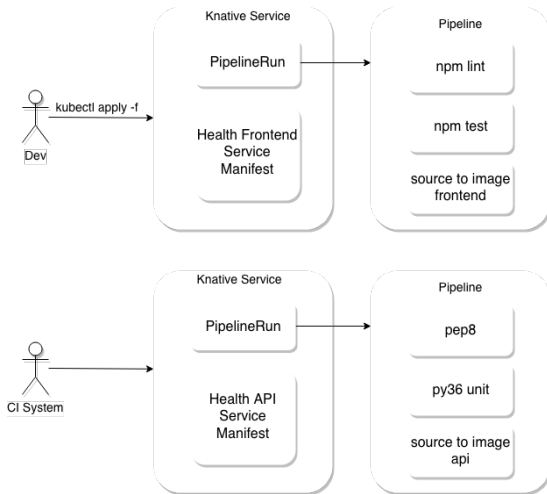
You need a “CI” application:

- Prow, JenkinsX, Zuul...
- Pipelines triggered by a CI app
- ...or by a developer

Tekton to CI for Tekton (AKA Dogfooding) \o/

What about security?

- Malicious users
- Running a pipeline from a PR
- Access to secrets

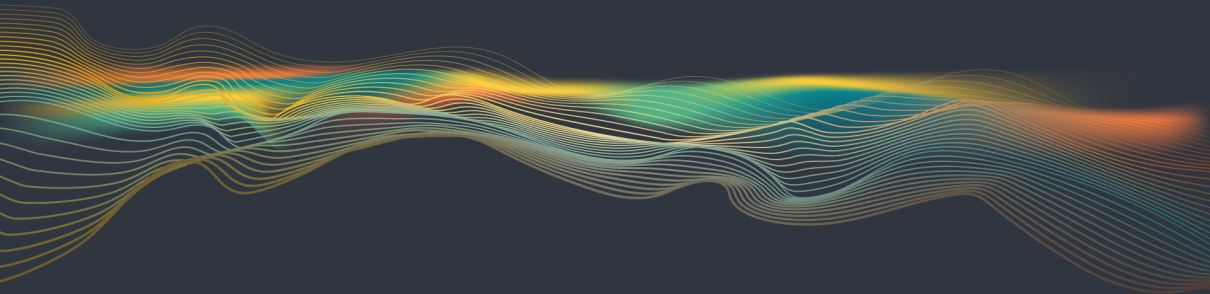


KService for Health Frontend

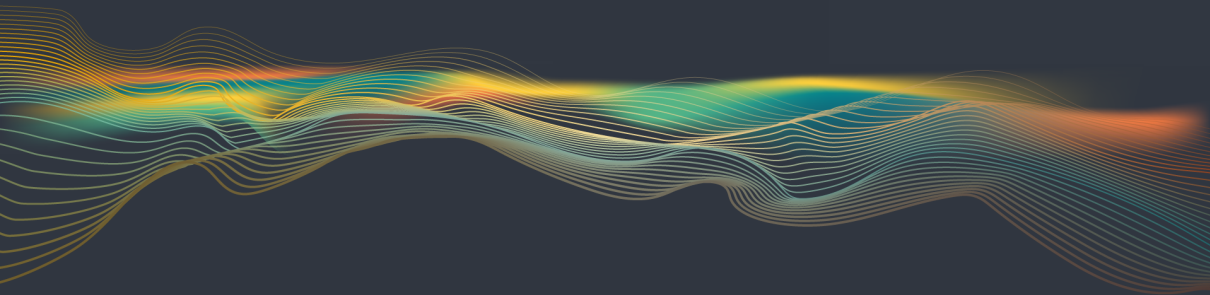
```
apiVersion: serving.knative.dev/v1alpha1
kind: Service
metadata:
  name: health-frontend
  labels:
    app: health
    component: frontend
    tag: "__TAG__"
spec:
  runLatest:
    configuration:
      build:
        apiVersion: tekton.dev/v1alpha1
        kind: PipelineRun
        metadata:
          labels:
            app: health
            component: frontend
            tag: "__TAG__"
        spec:
          pipelineRef:
            name: dev-test-build-frontend
          params:
            - name: imageTag
              value: "__TAG__"
            - name: nodeTestImage
              value: __NODE_IMAGE_NAME__
```

```
trigger:
  type: manual
resources:
  - name: src
    resourceRef:
      name: __GIT_RESOURCE_NAME__
  - name: builtImage
    resourceRef:
      name: __IMAGE_RESOURCE_NAME__
revisionTemplate: # template for building Revision
spec:
  container:
    image: us.icr.io/andreafr/health-frontend:__TAG__
    imagePullPolicy: Always
    env:
      - name: API_URL
        value: http://health-api.containers.domain
    ports:
      - name: http1
        containerPort: 80
        protocol: TCP
    livenessProbe:
      httpGet:
        path: /
    readinessProbe:
      httpGet:
        path: /
```

Demo Tekton and Knative Serving

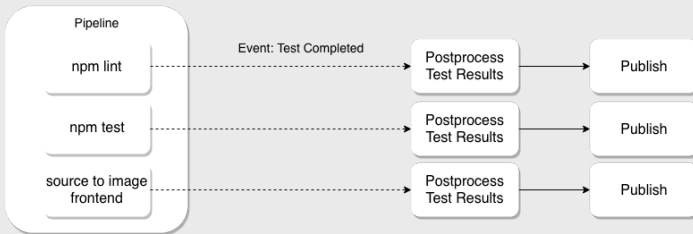


Asynchronous Pipelines

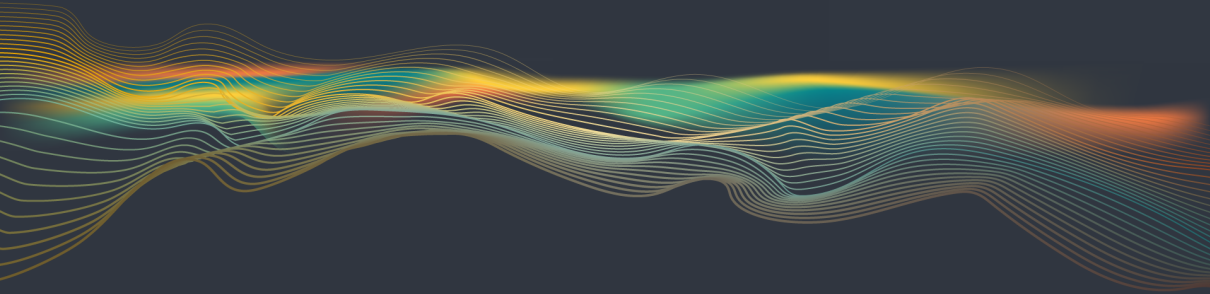


Triggering and Knative Eventing

- Manual trigger for *PipelineRun*
- Native Eventing triggers TBD
- What about async pipelines?
 - GitHub Source
 - Container Source
 - Pipeline Output as a Source



Conclusions



Roadmap

- Conditional Execution
- Build Results and Logs
- Pluggable Tasks
- Triggering
- Community Library

References

- This Talk: https://github.com/afrittoli/tekton_pipelines_knative_intro
- <https://tekton.dev/>, <https://cd.foundation/>
- <https://github.com/tektoncd/pipeline>
- <https://github.com/knative/docs/tree/master/community>
- <https://github.com/tektoncd/pipeline>
- https://github.com/tektoncd/pipeline/blob/master/api_compatibility_policy.md
- <https://github.com/tektoncd/pipeline/blob/master/roadmap-2019.md>
- <https://github.com/GoogleContainerTools/kaniko>
- <https://github.com/afrittoli/health-helm/tree/knative>
- <https://github.com/afrittoli/openstack-health/tree/knative-eventing>
- <https://andreafrittoli.me>
- <https://cloud.ibm.com>

Q&A

