KubeCon | CloudNativeCon

North America 2019

KubeCon | CloudNativeCon

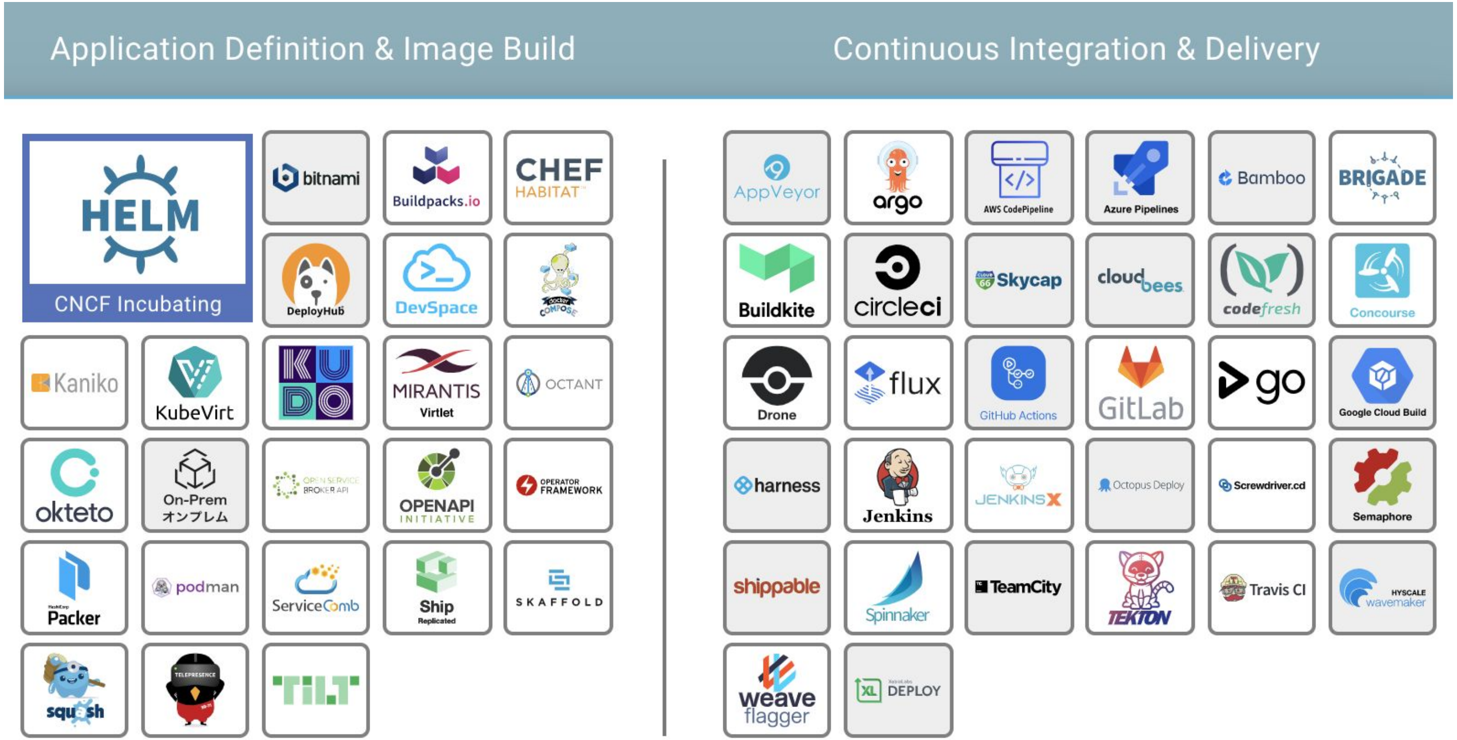
North America 2019

Defining Reference Model for Cloud-Native Application Delivery

A Deep Dive Session from CNCF App Delivery SIG

Alois Reitbauer, Lei Zhang

The App Mgmt & Delivery Ecosystem


Application Definition & Image Build

Continuous Integration & Delivery

source: https://landscape.cncf.io/

Take a closer look ...

Application Definition?

- YES!
    - Description for application
        - *templates/*
    - Metadata for application
        - *Chart.yaml*
            - Name
            - Description
            - Maintainers
            - Links to doc
            - …
    - Resources composed the application
        - E.g. chart.yaml, *dependencies* etc

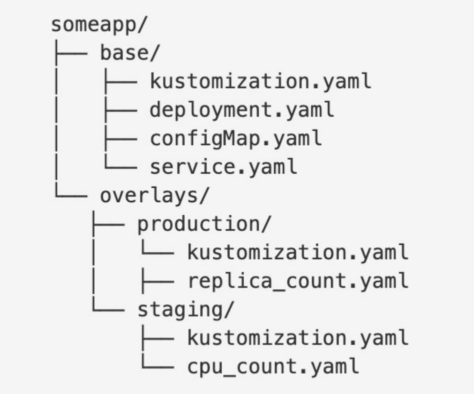
Take a closer closer look ...

- 🤔 Emm ...
  - Package management:
    - Search and browse chart repo, fetch charts
    - Parameterization & templating
      - *Values.yaml*
      - *gotpl, Lua (?)*
  - Release mgmt:
    - *helm upgrade, history, rollback*
  - App lifecycle mgmt hooks:
    - *"helm.sh/hook": post-install*
  - *and more…*

Are there part of “Application Definition”?

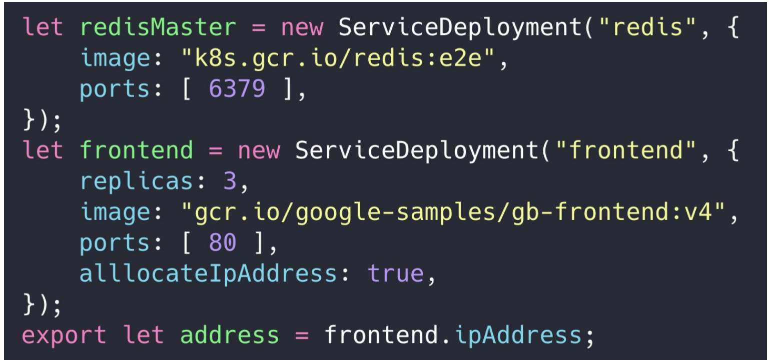
What is project “x” doing, really?

Helm



```
someapp/
├── base/
│   ├── kustomization.yaml
│   ├── deployment.yaml
│   ├── configMap.yaml
│   └── service.yaml
└── overlays/
    ├── production/
    │   └── kustomization.yaml
    │   ├── replica_count.yaml
    └── staging/
        ├── kustomization.yaml
        └── cpu_count.yaml
```

Kustomize

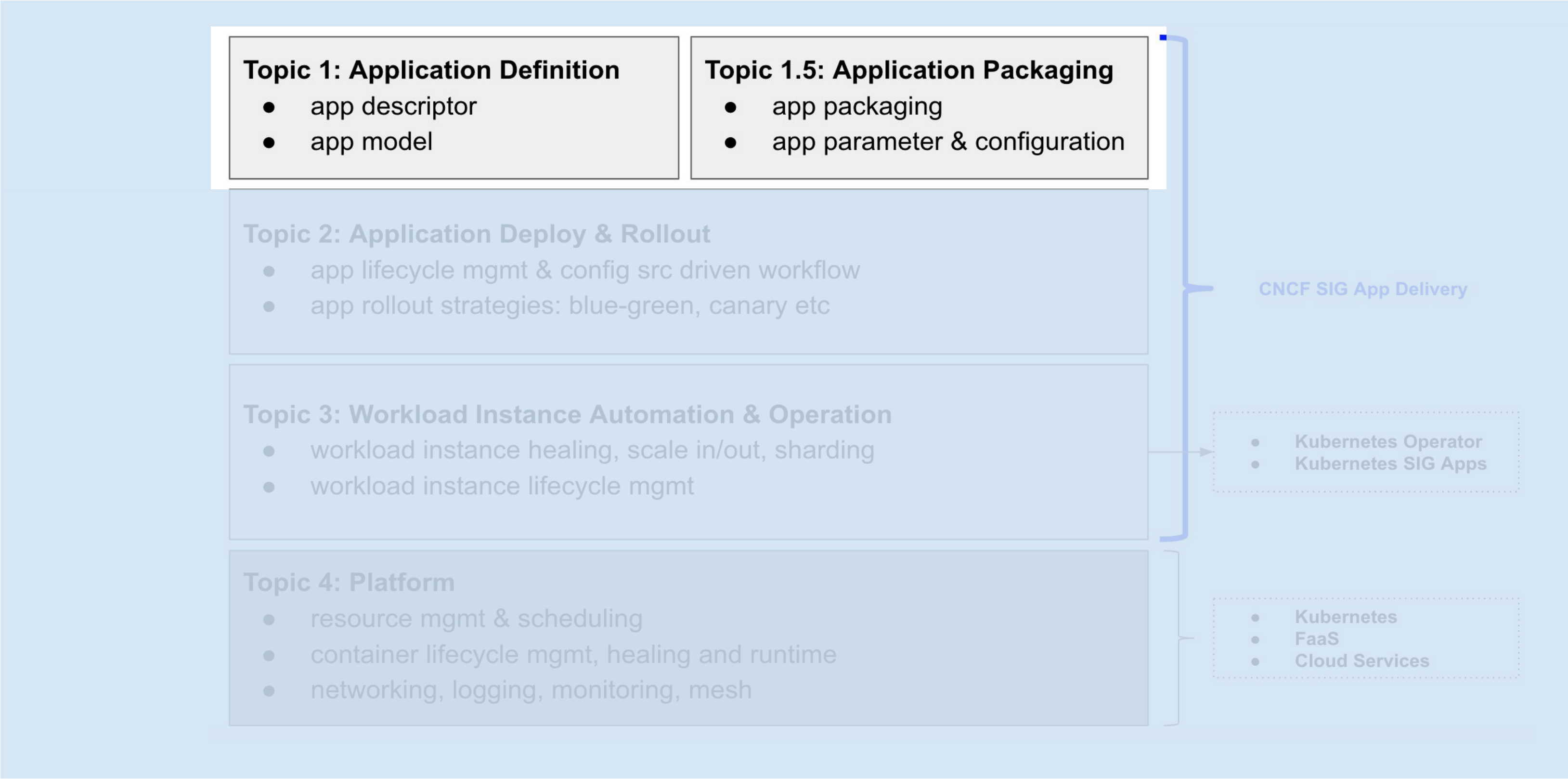


```
let redisMaster = new ServiceDeployment("redis", {
    image: "k8s.gcr.io/redis:e2e",
    ports: [ 6379 ],
});
let frontend = new ServiceDeployment("frontend", {
    replicas: 3,
    image: "gcr.io/google-samples/gb-frontend:v4",
    ports: [ 80 ],
    allocateIpAddress: true,
});
export let address = frontend.ipAddress;
```

Ksonnet

🧐 Are they “Application Definitions”?

For better answer to “what is project *X*”



Topic 1: Application Definition

- app descriptor
- app model

Topic 1.5: Application Packaging

- app packaging
- app parameter & configuration

Topic 2: Application Deploy & Rollout

- app lifecycle mgmt & config src driven workflow
- app rollout strategies: blue-green, canary etc

Topic 3: Workload Instance Automation & Operation

- workload instance healing, scale in/out, sharding
- workload instance lifecycle mgmt

Topic 4: Platform

- resource mgmt & scheduling
- container lifecycle mgmt, healing and runtime
- networking, logging, monitoring, mesh

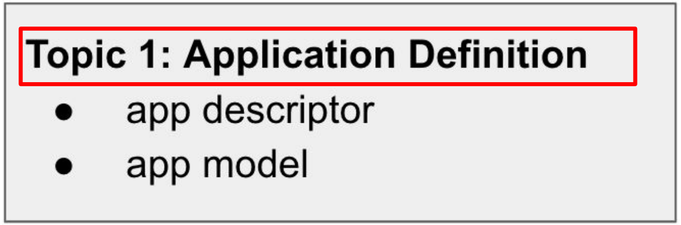
CNCF SIG App Delivery

- Kubernetes Operator
- Kubernetes SIG Apps

- Kubernetes
- FaaS
- Cloud Services

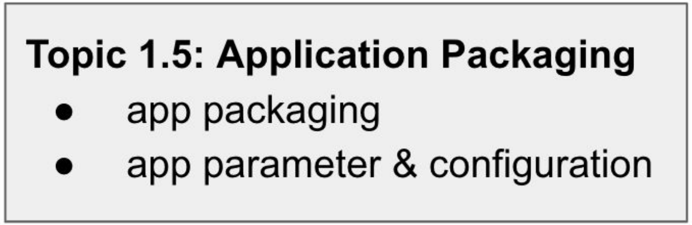
The Model of Application Delivery

Application Definition & Packaging



Topic 1: Application Definition

- app descriptor
- app model

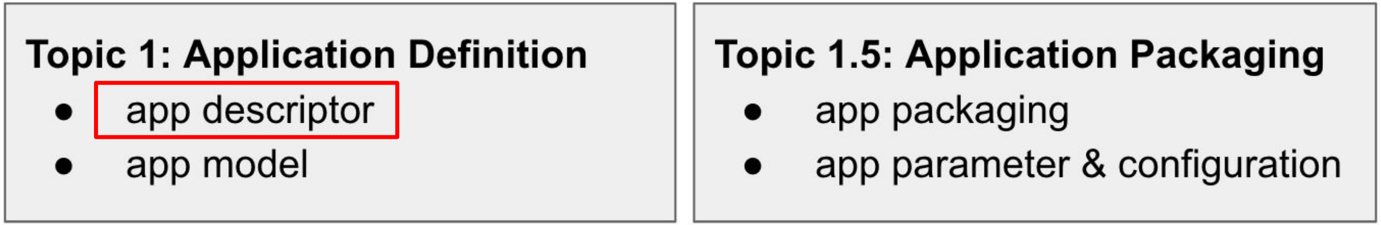


Topic 1.5: Application Packaging

- app packaging
- app parameter & configuration

- **Application Definition:**
  - The answer of “what to run”
  - The “start” of application delivery lifecycle
  - In real practice, mostly expressed as ***app descriptor*** or ***app model***

Application Definition & Packaging



Topic 1: Application Definition

- app descriptor
- app model

Topic 1.5: Application Packaging

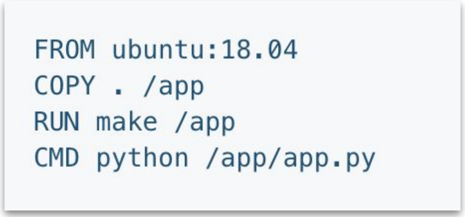
- app packaging
- app parameter & configuration

- **App descriptor:**
  a. Metadata for application as a whole, regardless of it’s instantiated or not
  b. Means for tracking resources composed the application

App descriptor could be in many forms (see next slides) ...

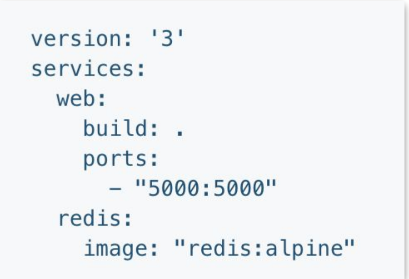
App descriptor could be simple

“Metadata for application as a whole”
😃



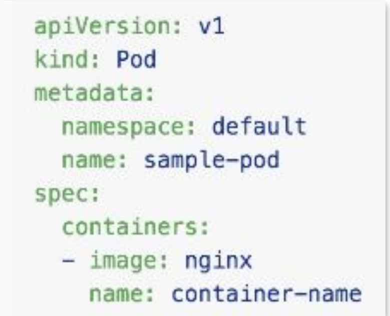
```
FROM ubuntu:18.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

single container



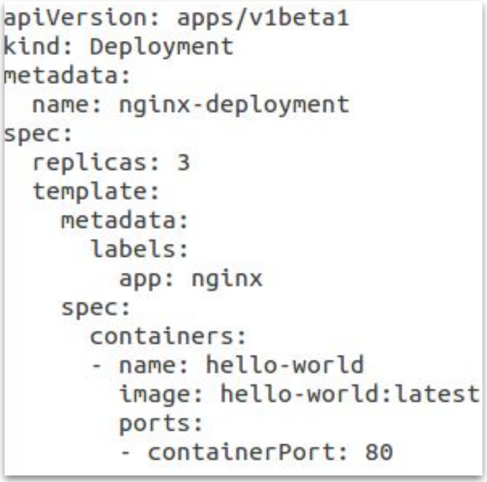
```
version: '3'
services:
  web:
    build: .
    ports:
      - "5000:5000"
  redis:
    image: "redis:alpine"
```

multiple containers



```
apiVersion: v1
kind: Pod
metadata:
  namespace: default
  name: sample-pod
spec:
  containers:
  - image: nginx
    name: container-name
```

multiple collaborative containers



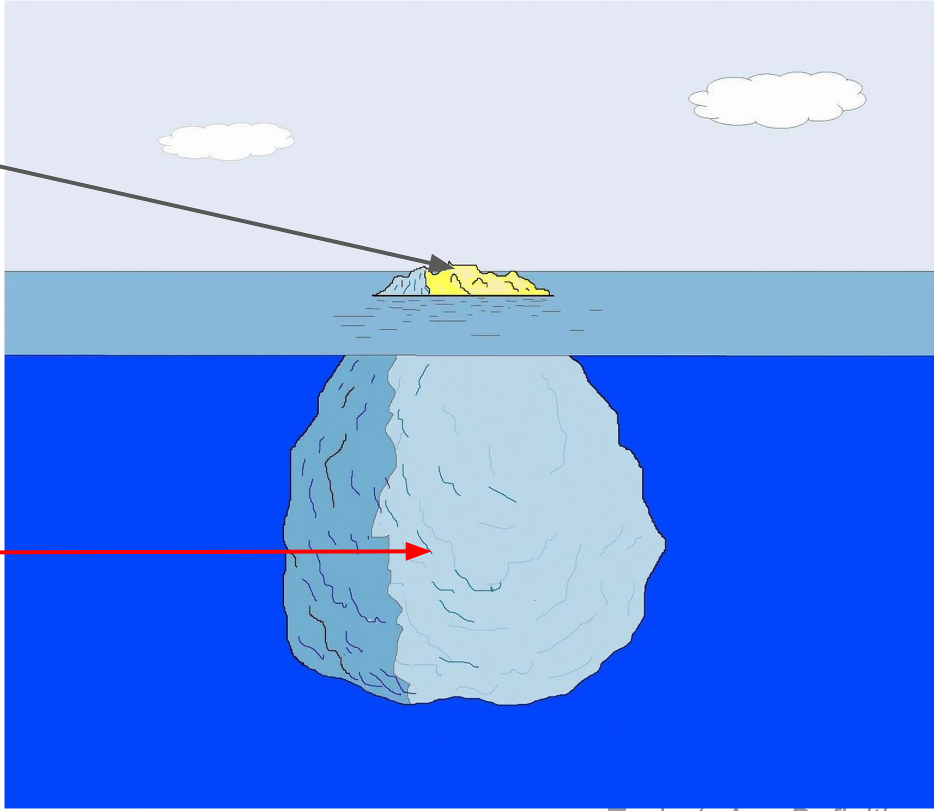
```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: hello-world
        image: hello-world:latest
        ports:
        - containerPort: 80
```

replicated collaborative containers group

“Means for tracking resources composed the application?”

Finding resources composed the app ...

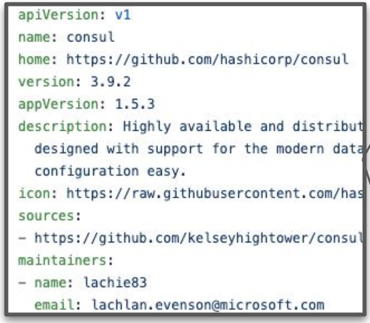
🥰 Your app description



😰 The resources composed your app

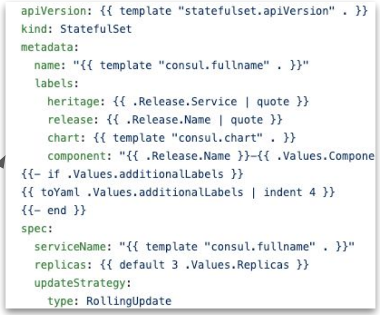
Service, CRD, Ingress, SLB, RBAC, Deployment
Image, PVC/PV, ConfigMap, Secret ...

Hence app descriptor could be sophisticated

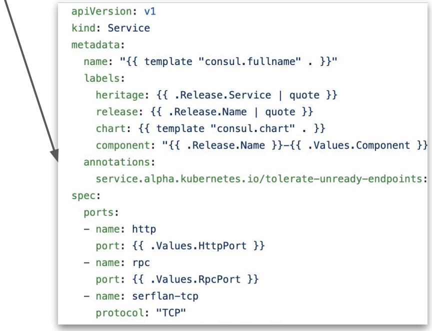


```
apiVersion: v1
name: consul
home: https://github.com/hashicorp/consul
version: 3.9.2
appVersion: 1.5.3
description: Highly available and distribut
  designed with support for the modern data
  configuration easy.
icon: https://raw.githubusercontent.com/has
sources:
- https://github.com/kelseyhightower/consul
maintainers:
- name: lachie83
  email: lachlan.evenson@microsoft.com
```

app metadata

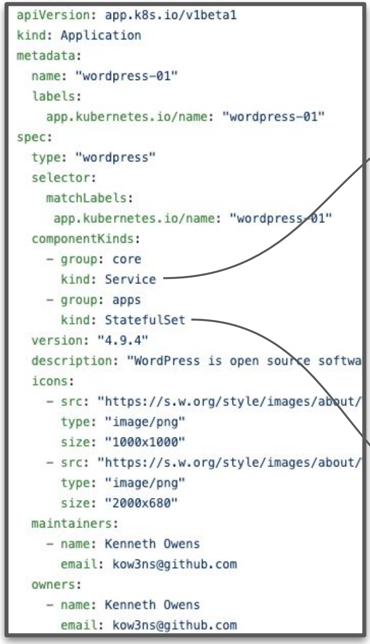


```
apiVersion: {{ template "statefulset.apiVersion" . }}
kind: StatefulSet
metadata:
  name: "{{ template "consul.fullname" . }}"
  labels:
    heritage: {{ .Release.Service | quote }}
    release: {{ .Release.Name | quote }}
    chart: {{ template "consul.chart" . }}
    component: "{{ .Release.Name }}-{{ .Values.Compone
{{- if .Values.additionalLabels }}
{{ toYaml .Values.additionalLabels | indent 4 }}
{{- end }}
spec:
  serviceName: "{{ template "consul.fullname" . }}"
  replicas: {{ default 3 .Values.Replicas }}
  updateStrategy:
    type: RollingUpdate
```



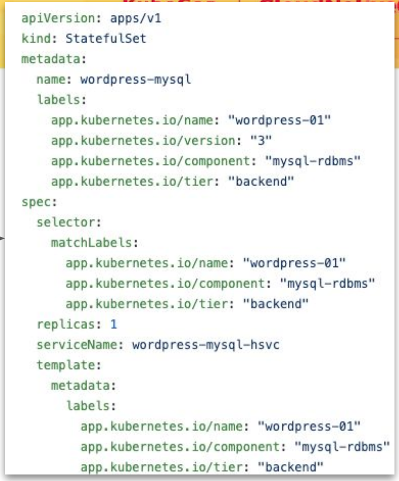
```
apiVersion: v1
kind: Service
metadata:
  name: "{{ template "consul.fullname" . }}"
  labels:
    heritage: {{ .Release.Service | quote }}
    release: {{ .Release.Name | quote }}
    chart: {{ template "consul.chart" . }}
    component: "{{ .Release.Name }}-{{ .Values.Component }}
  annotations:
    service.alpha.kubernetes.io/tolerate-unready-endpoints:
spec:
  ports:
  - name: http
    port: {{ .Values.HttpPort }}
  - name: rpc
    port: {{ .Values.RpcPort }}
  - name: serflan-tcp
    protocol: "TCP"
```

Helm (*Chart.yaml + templates/ + values.yaml*)

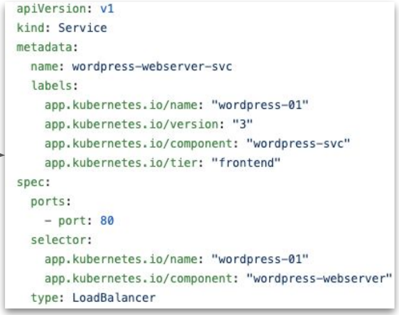


```
apiVersion: app.k8s.io/v1beta1
kind: Application
metadata:
  name: "wordpress-01"
  labels:
    app.kubernetes.io/name: "wordpress-01"
spec:
  type: "wordpress"
  selector:
    matchLabels:
      app.kubernetes.io/name: "wordpress-01"
  componentKinds:
    - group: core
      kind: Service
    - group: apps
      kind: StatefulSet
  version: "4.9.4"
  description: "WordPress is open source softwa
  icons:
    - src: "https://s.w.org/style/images/about/
      type: "image/png"
      size: "1000x1000"
    - src: "https://s.w.org/style/images/about/
      type: "image/png"
      size: "2000x680"
  maintainers:
    - name: Kenneth Owens
      email: kow3ns@github.com
  owners:
    - name: Kenneth Owens
      email: kow3ns@github.com
```

app metadata



```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: wordpress-mysql
  labels:
    app.kubernetes.io/name: "wordpress-01"
    app.kubernetes.io/version: "3"
    app.kubernetes.io/component: "mysql-rdbms"
    app.kubernetes.io/tier: "backend"
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: "wordpress-01"
      app.kubernetes.io/component: "mysql-rdbms"
      app.kubernetes.io/tier: "backend"
  replicas: 1
  serviceName: wordpress-mysql-hsvc
  template:
    metadata:
      labels:
        app.kubernetes.io/name: "wordpress-01"
        app.kubernetes.io/component: "mysql-rdbms"
        app.kubernetes.io/tier: "backend"
```



```
apiVersion: v1
kind: Service
metadata:
  name: wordpress-webserver-svc
  labels:
    app.kubernetes.io/name: "wordpress-01"
    app.kubernetes.io/version: "3"
    app.kubernetes.io/component: "wordpress-svc"
    app.kubernetes.io/tier: "frontend"
spec:
  ports:
    - port: 80
  selector:
    app.kubernetes.io/name: "wordpress-01"
    app.kubernetes.io/component: "wordpress-webserver"
  type: LoadBalancer
```

Application CRD (*app-crd.yaml + K8s YAMLs*)

Application Definition & Packaging

Topic 1: Application Definition

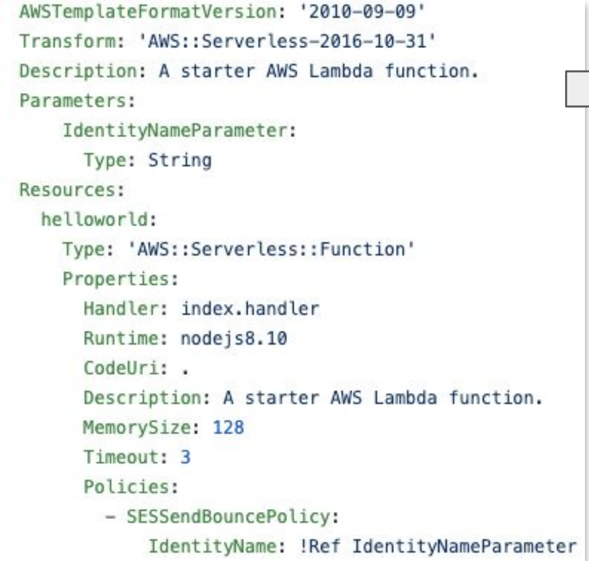
- app descriptor
- app model

Topic 1.5: Application Packaging

- app packaging
- app parameter & configuration

- **App model**: a "opinionated" form of app descriptor
  a. Metadata for application as a whole, regardless of it's instantiated or not
  b. Means for tracking resources composed the application
  c. A declarative **spec** for defining information above

E.g. AWS Serverless App Model (SAM)

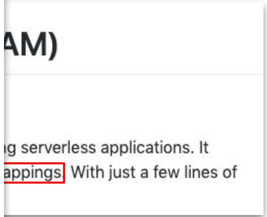


```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: A starter AWS Lambda function.
Parameters:
    IdentityNameParameter:
      Type: String
Resources:
  helloworld:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: index.handler
      Runtime: nodejs8.10
      CodeUri: .
      Description: A starter AWS Lambda function.
      MemorySize: 128
      Timeout: 3
      Policies:
        - SESSendBouncePolicy:
            IdentityName: !Ref IdentityNameParameter
```

AWS Serverless App Model (SAM), Spec
First announcement: Nov 18, 2016

nction
oda)

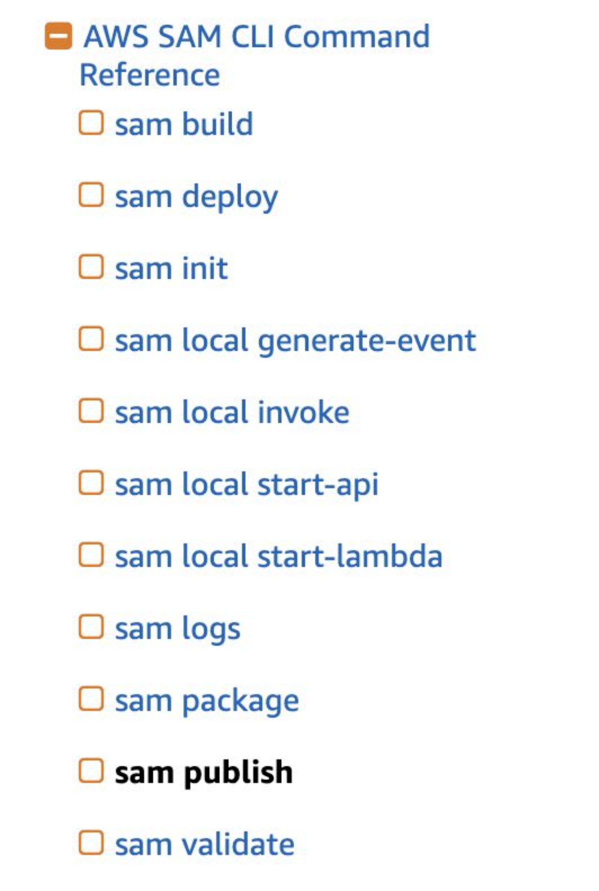
AV
licens
The A
provi
confi



AM)
g serverless applications. It
appings. With just a few lines of

A CloudF

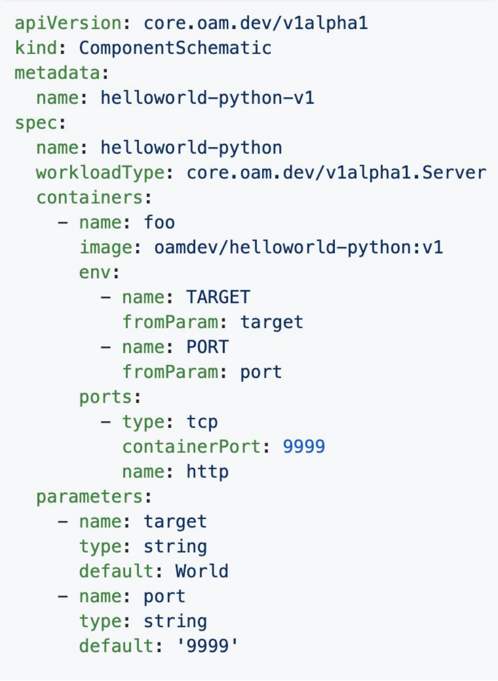
S serverless workloads



AWS SAM CLI Command Reference

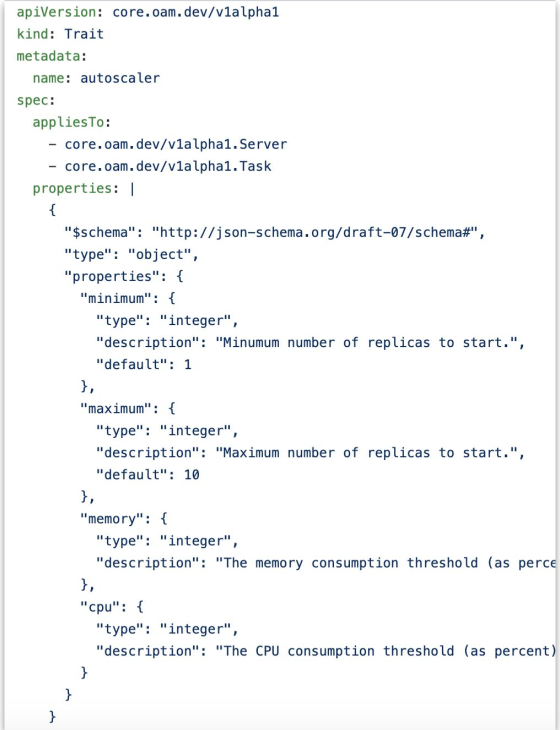
- □ sam build
- □ sam deploy
- □ sam init
- □ sam local generate-event
- □ sam local invoke
- □ sam local start-api
- □ sam local start-lambda
- □ sam logs
- □ sam package
- □ **sam publish**
- □ sam validate

E.g. Open Application Model (OAM)



```yaml
apiVersion: core.oam.dev/v1alpha1
kind: ComponentSchematic
metadata:
  name: helloworld-python-v1
spec:
  name: helloworld-python
  workloadType: core.oam.dev/v1alpha1.Server
  containers:
    - name: foo
      image: oamdev/helloworld-python:v1
      env:
        - name: TARGET
          fromParam: target
        - name: PORT
          fromParam: port
      ports:
        - type: tcp
          containerPort: 9999
          name: http
  parameters:
    - name: target
      type: string
      default: World
    - name: port
      type: string
      default: '9999'
```

Component: metadata for app component

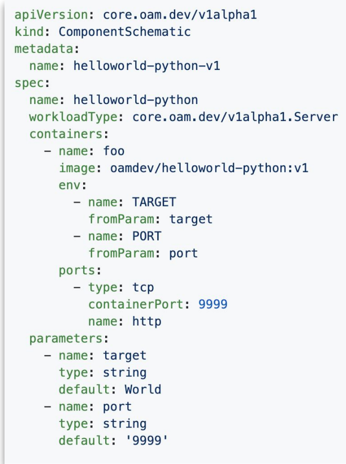


```yaml
apiVersion: core.oam.dev/v1alpha1
kind: Trait
metadata:
  name: autoscaler
spec:
  appliesTo:
    - core.oam.dev/v1alpha1.Server
    - core.oam.dev/v1alpha1.Task
  properties: |
    {
      "$schema": "http://json-schema.org/draft-07/schema#",
      "type": "object",
      "properties": {
        "minimum": {
          "type": "integer",
          "description": "Minumum number of replicas to start.",
          "default": 1
        },
        "maximum": {
          "type": "integer",
          "description": "Maximum number of replicas to start.",
          "default": 10
        },
        "memory": {
          "type": "integer",
          "description": "The memory consumption threshold (as perce
        },
        "cpu": {
          "type": "integer",
          "description": "The CPU consumption threshold (as percent)
        }
      }
    }
```

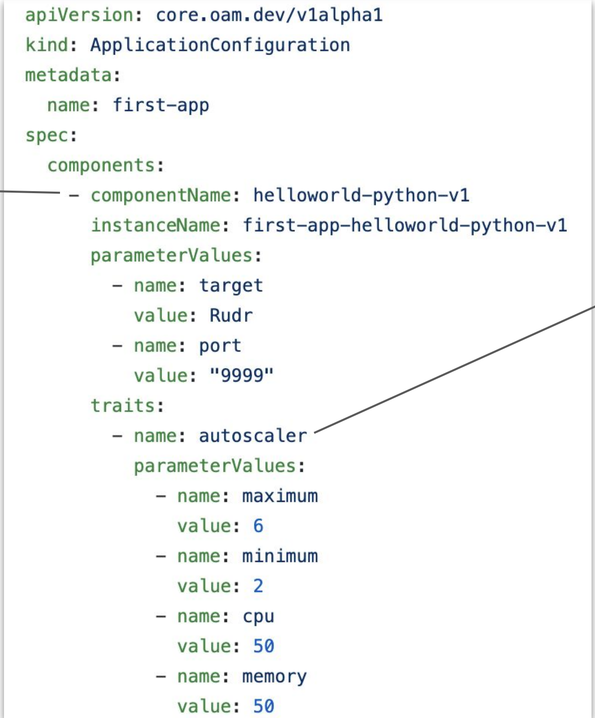
Trait: metadata for platform capability

Source: https://github.com/oam-dev/rudr/blob/master/docs/README.md

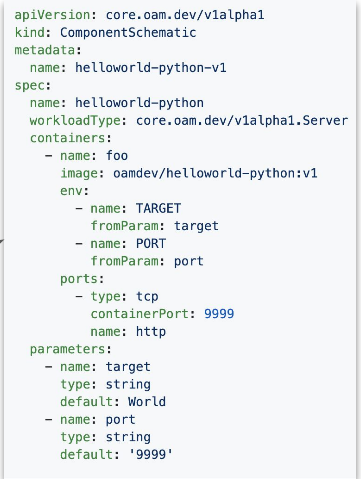
E.g. Open Application Model (OAM)



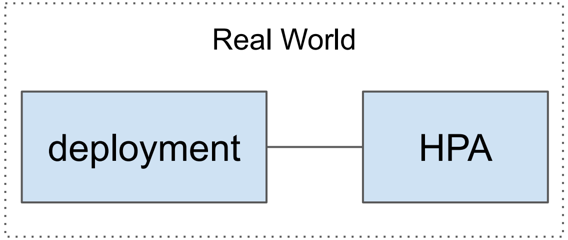
```yaml
apiVersion: core.oam.dev/v1alpha1
kind: ComponentSchematic
metadata:
  name: helloworld-python-v1
spec:
  name: helloworld-python
  workloadType: core.oam.dev/v1alpha1.Server
  containers:
    - name: foo
      image: oamdev/helloworld-python:v1
      env:
        - name: TARGET
          fromParam: target
        - name: PORT
          fromParam: port
      ports:
        - type: tcp
          containerPort: 9999
          name: http
  parameters:
    - name: target
      type: string
      default: World
    - name: port
      type: string
      default: '9999'
```



```yaml
apiVersion: core.oam.dev/v1alpha1
kind: ApplicationConfiguration
metadata:
  name: first-app
spec:
  components:
    - componentName: helloworld-python-v1
      instanceName: first-app-helloworld-python-v1
      parameterValues:
        - name: target
          value: Rudr
        - name: port
          value: "9999"
      traits:
        - name: autoscaler
          parameterValues:
            - name: maximum
              value: 6
            - name: minimum
              value: 2
            - name: cpu
              value: 50
            - name: memory
              value: 50
```



```yaml
apiVersion: core.oam.dev/v1alpha1
kind: ComponentSchematic
metadata:
  name: helloworld-python-v1
spec:
  name: helloworld-python
  workloadType: core.oam.dev/v1alpha1.Server
  containers:
    - name: foo
      image: oamdev/helloworld-python:v1
      env:
        - name: TARGET
          fromParam: target
        - name: PORT
          fromParam: port
      ports:
        - type: tcp
          containerPort: 9999
          name: http
  parameters:
    - name: target
      type: string
      default: World
    - name: port
      type: string
      default: '9999'
```



Components + Traits = Application

Application Definition & Packaging

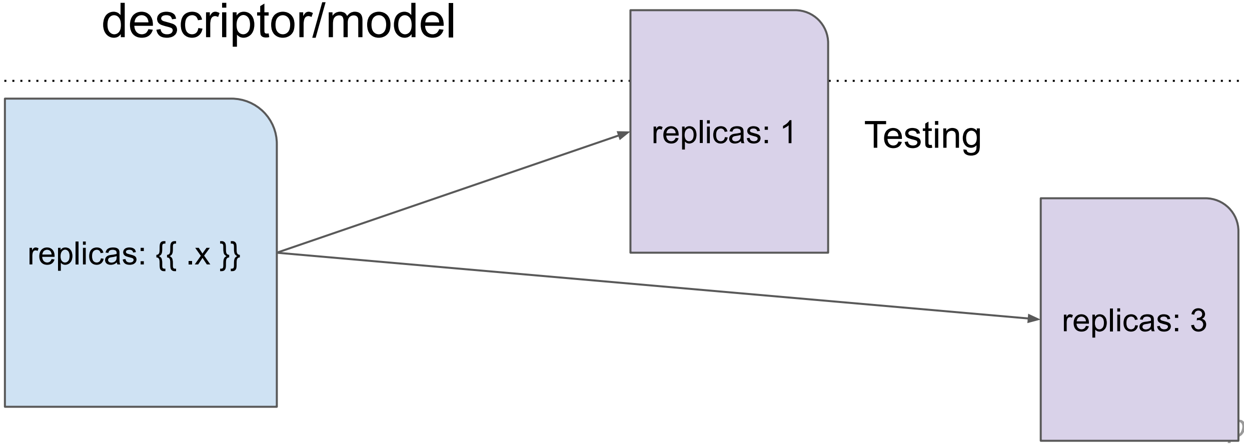
Topic 1: Application Definition

- app descriptor
- app model

Topic 1.5: Application Packaging

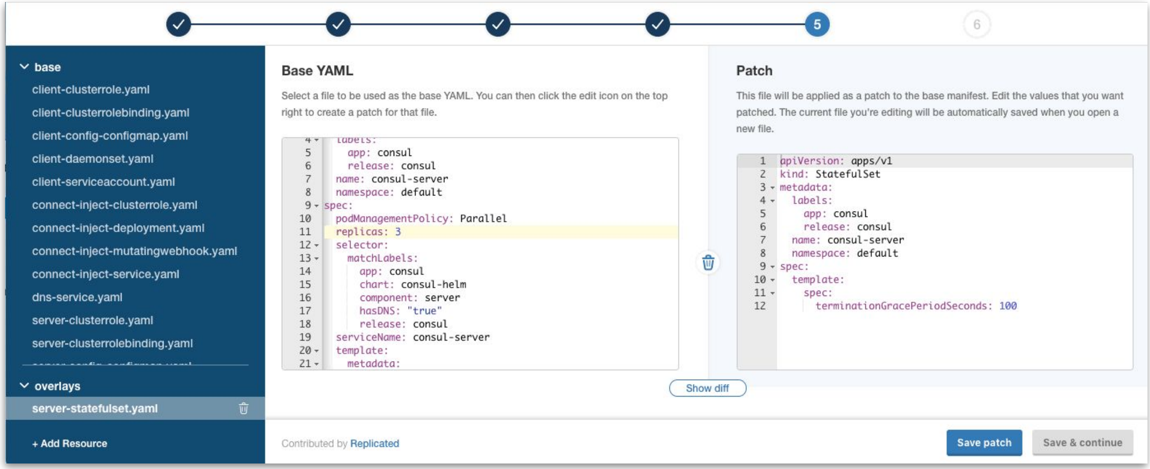
- app packaging
- app parameter & configuration

- **Application parameter & configuration**
  - The way to customize fields and parameters in app descriptor/model



Application Definition & Packaging

1. **Templating**
   a. Helm: *easy to use, while break integrity of YAML*
2. **Overlay**
   a. Kustomize: *keep integrity of YAML, GitOps friendly layout, while higher learning curve*
3. **DSL**
   a. jsonnet/ksonnet/isopod: *powerful, no YAML, highest learning curve*


5
6
base
client-clusterrole.yaml
client-clusterrolebinding.yaml
client-config-configmap.yaml
client-daemonset.yaml
client-serviceaccount.yaml
connect-inject-clusterrole.yaml
connect-inject-deployment.yaml
connect-inject-mutatingwebhook.yaml
connect-inject-service.yaml
dns-service.yaml
server-clusterrole.yaml
server-clusterrolebinding.yaml
overlays
server-statefulset.yaml
+ Add Resource
Show diff
Save patch
Save & continue

Base YAML

Select a file to be used as the base YAML. You can then click the edit icon on the top right to create a patch for that file.

```
  labels:
    app: consul
    release: consul
  name: consul-server
  namespace: default
spec:
  podManagementPolicy: Parallel
  replicas: 3
  selector:
    matchLabels:
      app: consul
      chart: consul-helm
      component: server
      hasDNS: "true"
      release: consul
  serviceName: consul-server
  template:
    metadata:
```

Patch

This file will be applied as a patch to the base manifest. Edit the values that you want patched. The current file you're editing will be automatically saved when you open a new file.

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  labels:
    app: consul
    release: consul
  name: consul-server
  namespace: default
spec:
  template:
    spec:
      terminationGracePeriodSeconds: 100
```

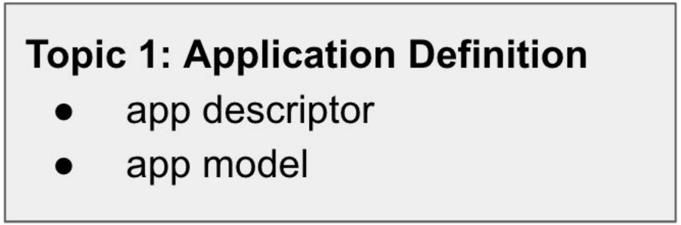
Contributed by Replicated

More interesting attempts:

replicatedhq/ship: i.e. render charts w/ default values beforehand, and then use kustomize to patch them

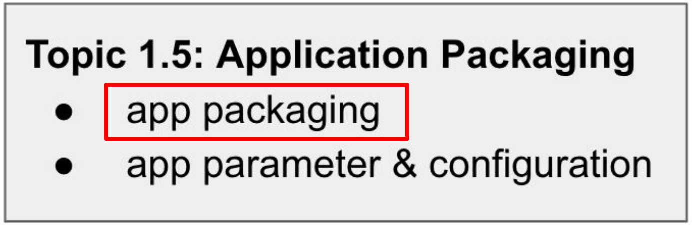
a.k.a “kustomize” helm charts instead of templating

Application Definition & Packaging



Topic 1: Application Definition

- app descriptor
- app model



Topic 1.5: Application Packaging

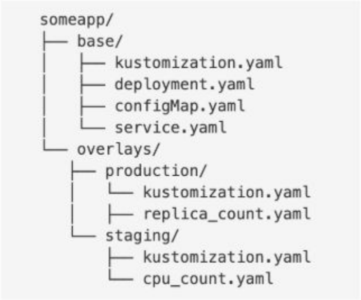
- app packaging
- app parameter & configuration

- **Application packaging:**
  - The way to bundle app descriptors/models into a deployable unit so for easier searching and distribution

1. **Any compression form**
   a. *.tar.gz, *.zip
2. **OCI artifacts**
   a. CNAB, docker image, Helm charts
3. **Helm ecosystem**
   a. Helm charts + Helm Hub

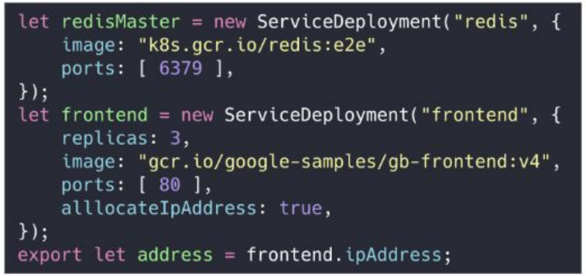
Checkpoint

Helm



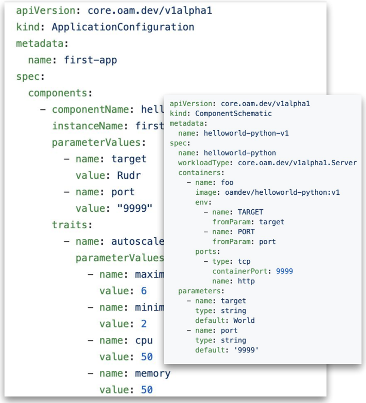
```
someapp/
├── base/
│   ├── kustomization.yaml
│   ├── deployment.yaml
│   ├── configMap.yaml
│   └── service.yaml
└── overlays/
    ├── production/
    │   └── kustomization.yaml
    │   ├── replica_count.yaml
    └── staging/
        ├── kustomization.yaml
        └── cpu_count.yaml
```

Kustomize



```
let redisMaster = new ServiceDeployment("redis", {
    image: "k8s.gcr.io/redis:e2e",
    ports: [ 6379 ],
});
let frontend = new ServiceDeployment("frontend", {
    replicas: 3,
    image: "gcr.io/google-samples/gb-frontend:v4",
    ports: [ 80 ],
    alllocateIpAddress: true,
});
export let address = frontend.ipAddress;
```

Ksonnet



```
apiVersion: core.oam.dev/v1alpha1
kind: ApplicationConfiguration
metadata:
  name: first-app
spec:
  components:
    - componentName: hell
      instanceName: first
      parameterValues:
        - name: target
          value: Rudr
        - name: port
          value: "9999"
      traits:
        - name: autoscale
          parameterValues
            - name: maxim
              value: 6
            - name: minim
              value: 2
            - name: cpu
              value: 50
            - name: memory
              value: 50
```

```
apiVersion: core.oam.dev/v1alpha1
kind: ComponentSchematic
metadata:
  name: helloworld-python-v1
spec:
  name: helloworld-python
  workloadType: core.oam.dev/v1alpha1.Server
  containers:
    - name: foo
      image: oamdev/helloworld-python:v1
      env:
        - name: TARGET
          fromParam: target
        - name: PORT
          fromParam: port
      ports:
        - type: tcp
          containerPort: 9999
          name: http
  parameters:
    - name: target
      type: string
      default: World
    - name: port
      type: string
      default: '9999'
```

OAM/AWS SAM

Topic 1

app descriptor

app model

app packaging

Topic 1.5

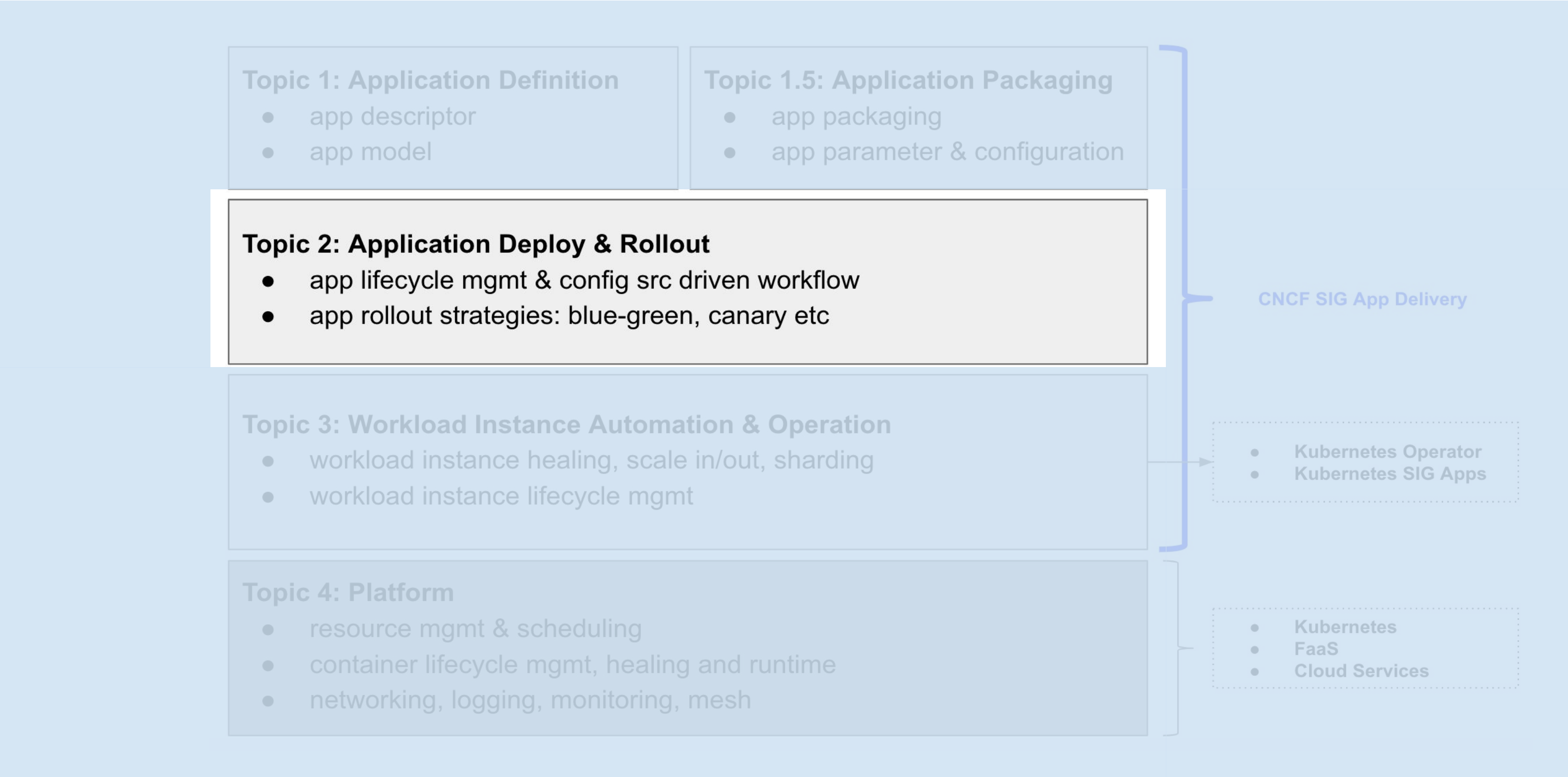
app parameter & configuration

app parameter & configuration*

app parameter & configuration*

* its *app descriptor is raw K8s API resource*

Application Deploy & Rollout



Topic 1: Application Definition
- app descriptor
- app model

Topic 1.5: Application Packaging
- app packaging
- app parameter & configuration

Topic 2: Application Deploy & Rollout
- app lifecycle mgmt & config src driven workflow
- app rollout strategies: blue-green, canary etc

Topic 3: Workload Instance Automation & Operation
- workload instance healing, scale in/out, sharding
- workload instance lifecycle mgmt

Topic 4: Platform
- resource mgmt & scheduling
- container lifecycle mgmt, healing and runtime
- networking, logging, monitoring, mesh

CNCF SIG App Delivery

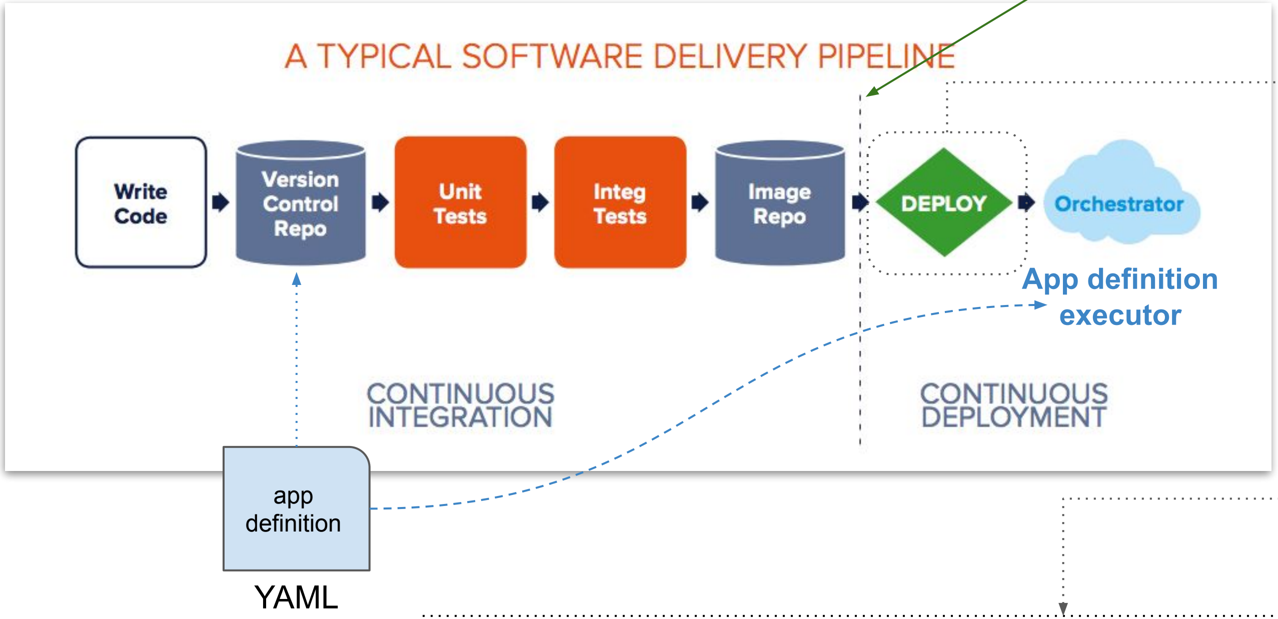
- Kubernetes Operator
- Kubernetes SIG Apps

- Kubernetes
- FaaS
- Cloud Services

Application Deploy & Rollout

copyright: weaveworks blog

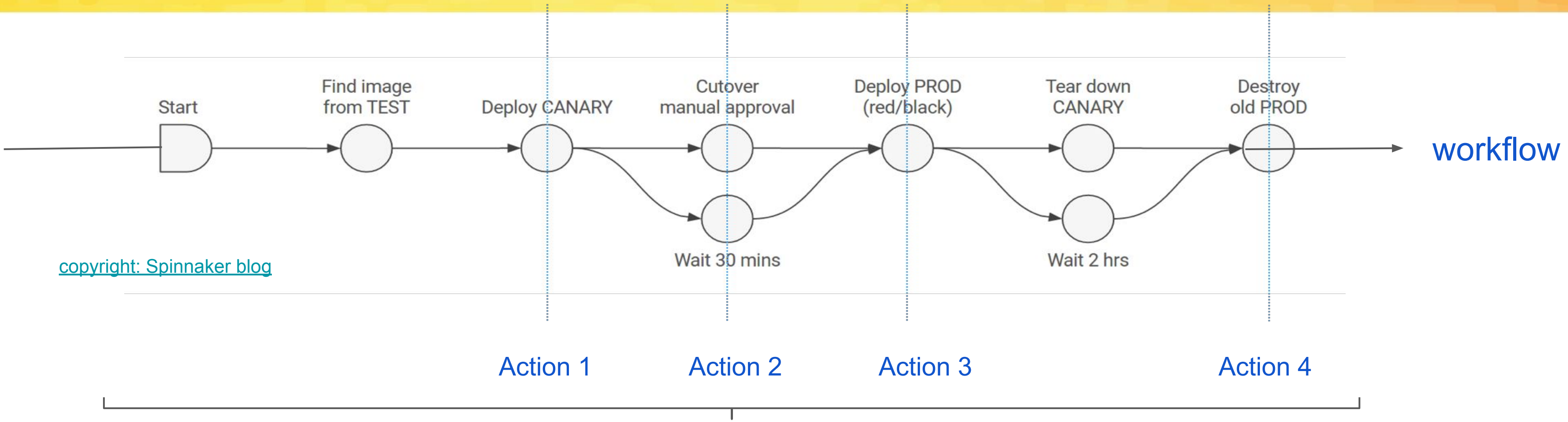
A clear **boundary** between CI & CD: **artifacts** ready



From **deployable application artifacts** to **running instances**, and keep them running until been terminated.

Normally, achieved by a **workflow** composed by app delivery **actions**.

App Delivery Workflow and Actions



copyright: Spinnaker blog

Driven by Git as source of truth (or other version control)

- Well-known *git review, approve, merge, rollback* actions as the triggers of this workflow -- GitOps

Pipeline style workflow:

Tekton, Argo Workflow, Spinnaker

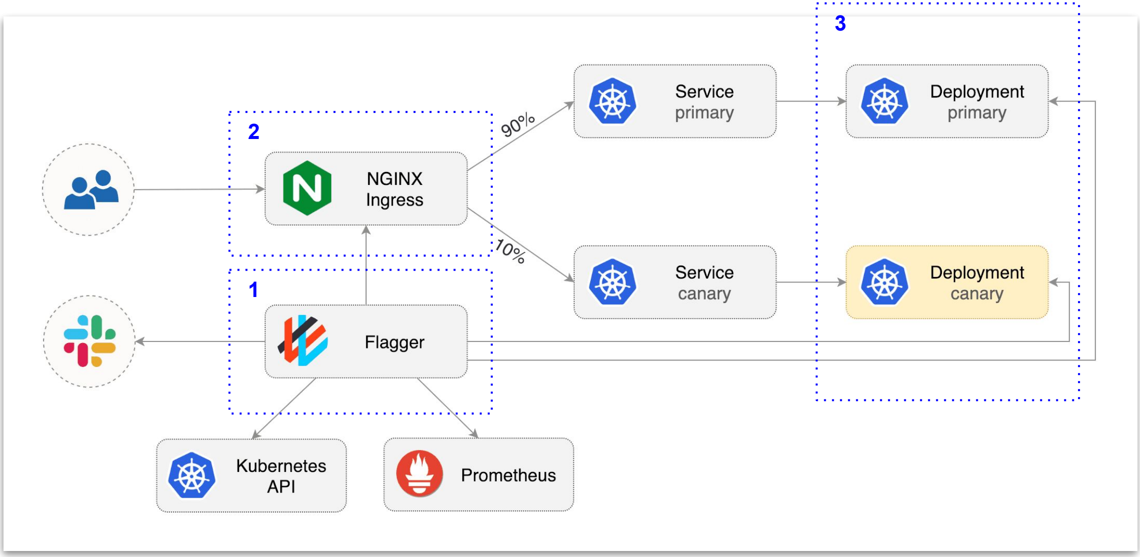
Event style workflow:

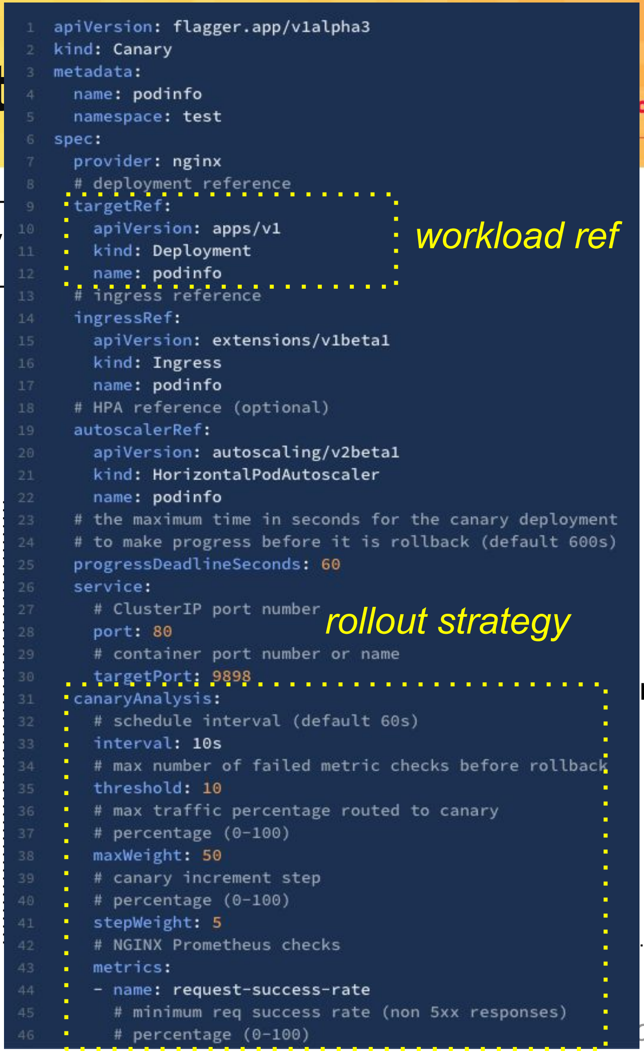
Keptn (knative eventing based)

Action Executor: App Rollout Project

Rollout with strategies: the way to **upgrade/rollback** the application seamlessly

Flagger as example:





```yaml
apiVersion: flagger.app/v1alpha3
kind: Canary
metadata:
  name: podinfo
  namespace: test
spec:
  provider: nginx
  # deployment reference
  targetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: podinfo
  # ingress reference
  ingressRef:
    apiVersion: extensions/v1beta1
    kind: Ingress
    name: podinfo
  # HPA reference (optional)
  autoscalerRef:
    apiVersion: autoscaling/v2beta1
    kind: HorizontalPodAutoscaler
    name: podinfo
  # the maximum time in seconds for the canary deployment
  # to make progress before it is rollback (default 600s)
  progressDeadlineSeconds: 60
  service:
    # ClusterIP port number
    port: 80
    # container port number or name
    targetPort: 9898
  canaryAnalysis:
    # schedule interval (default 60s)
    interval: 10s
    # max number of failed metric checks before rollback
    threshold: 10
    # max traffic percentage routed to canary
    # percentage (0-100)
    maxWeight: 50
    # canary increment step
    # percentage (0-100)
    stepWeight: 5
    # NGINX Prometheus checks
    metrics:
    - name: request-success-rate
      # minimum req success rate (non 5xx responses)
      # percentage (0-100)
```

workload ref

rollout strategy

on

rd,

ollout

Fun Fact

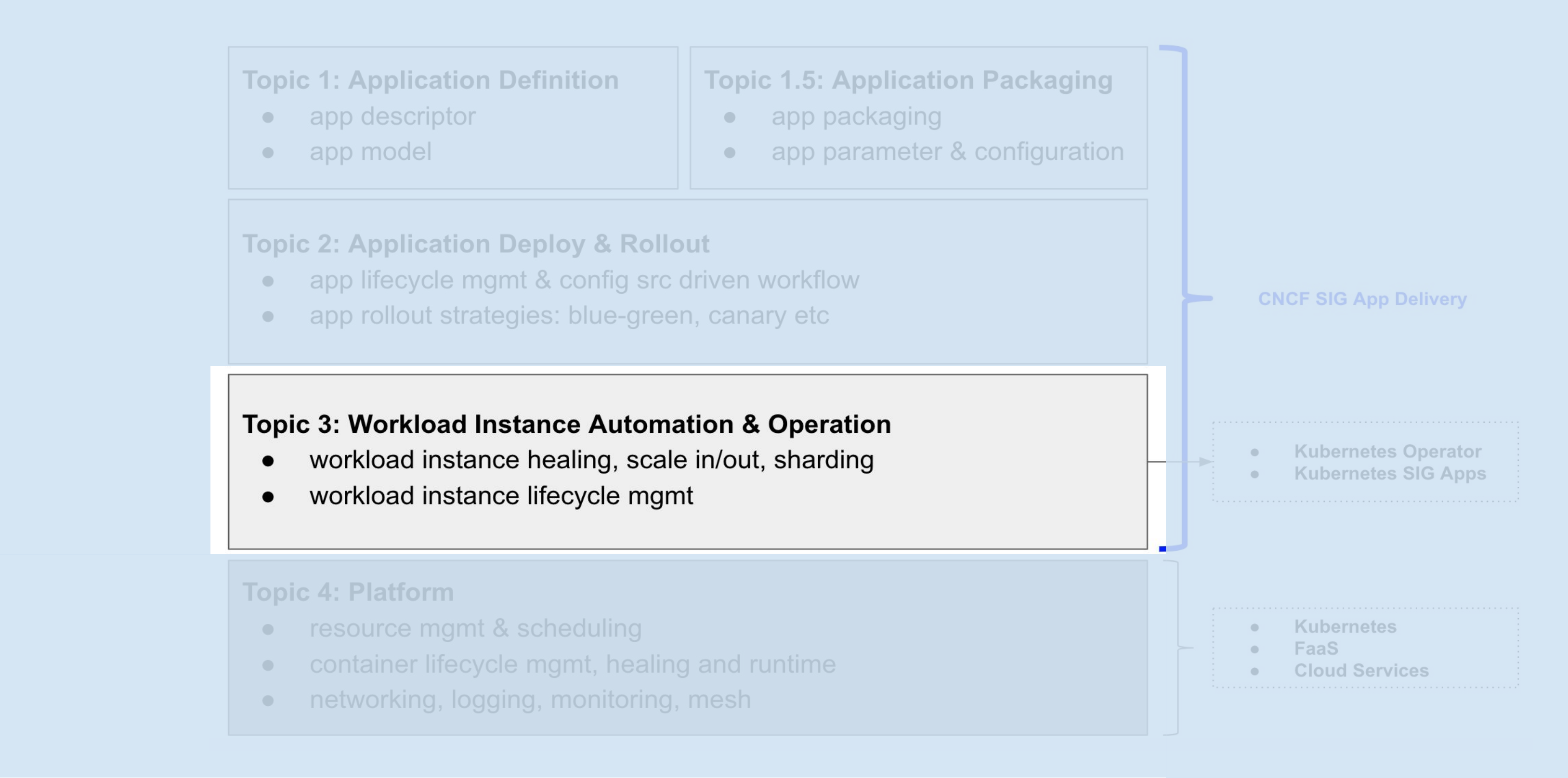
Helm actually has several functionalities sit in Topic 2:

helm upgrade
helm history
helm rollback

"helm.sh/hook": post-install

While with Helm 3 released, seems Helm now focus more on **Topic 1 & 1.5**.

Workload Instance Automation & Operation



Topic 1: Application Definition

- app descriptor
- app model

Topic 1.5: Application Packaging

- app packaging
- app parameter & configuration

Topic 2: Application Deploy & Rollout

- app lifecycle mgmt & config src driven workflow
- app rollout strategies: blue-green, canary etc

Topic 3: Workload Instance Automation & Operation

- workload instance healing, scale in/out, sharding
- workload instance lifecycle mgmt

Topic 4: Platform

- resource mgmt & scheduling
- container lifecycle mgmt, healing and runtime
- networking, logging, monitoring, mesh

CNCF SIG App Delivery

- Kubernetes Operator
- Kubernetes SIG Apps

- Kubernetes
- FaaS
- Cloud Services

Workload Instance Automation & Operation

Workloads in K8s world:

- Deployment, StatefulSet, DaemonSet, Job … (K8s SIG-APP)
- Operator
- OpenKruise
- …

Emm, what is workload instances?

- **Pods** managed by workload controllers
  - *but could be function or VM in other context.*

- *A K8s Deployment with `replicas=3`: has **3 workload instances** which are identical to each other*
- *A MySQL Cluster managed by MySQL Operator with `size=5`: has **5 workload instances** which are **not identical to each other.***

What’s the difference?

The outstanding difference:

- Topic 2 focuses on “**application level**” operations
  - Blue-green, Canary, A/B test, traffic split, app rollout, progressive deploy, GitOps ...
- Topic 3 focuses on “**workload instance level**” operations
  - Scale in/out, maxUnavailable/maxSurge, partition, Pod rolling update ...

Fun fact:

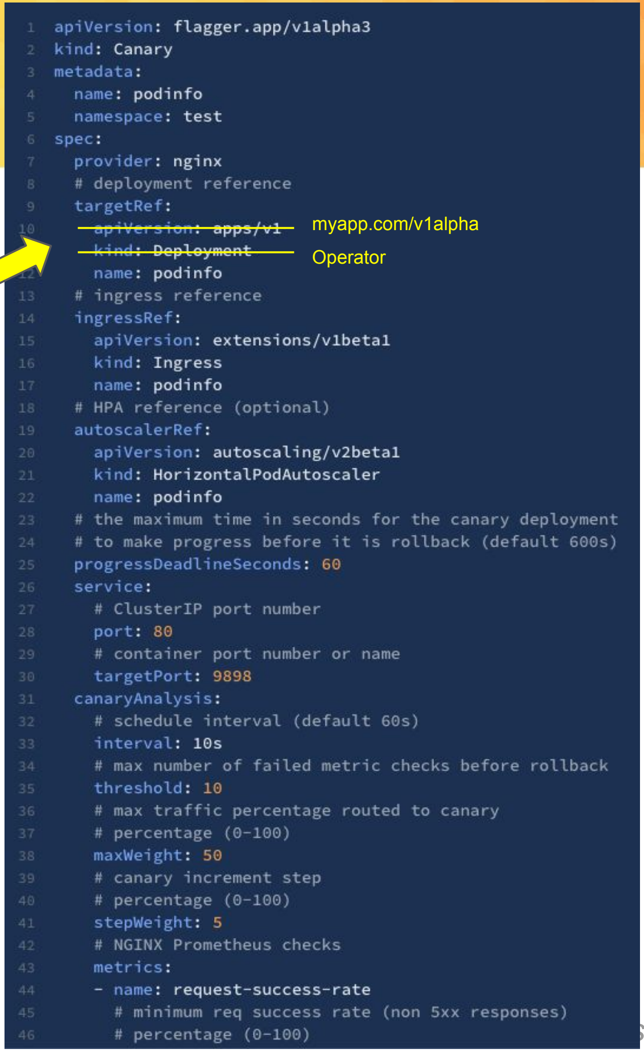
1. Though with the name of *Advanced Deployment*, Argo Rollout is a Topic 2 project:
   a. It focuses on performing Blue-green/Canary deployment at application level (Topic 2)
   b. By leveraging *ReplicaSet* as workload instance level controller (Topic 3)

Why decouple *Topic 2* and *Topic 3*?

Q: Can I do Flagger *Canary deployment* for **Operator based applications** with **this spec**?

An application could be managed by Deployment controller, but also by **StatefulSet** and **Operator** etc. Can we apply same *Canary deployment* strategy to them as well?

Do not "vendor lock" developers by your rollout capabilities! (Topic 2)
And let them choose their own workloads (Topic 3) freely!



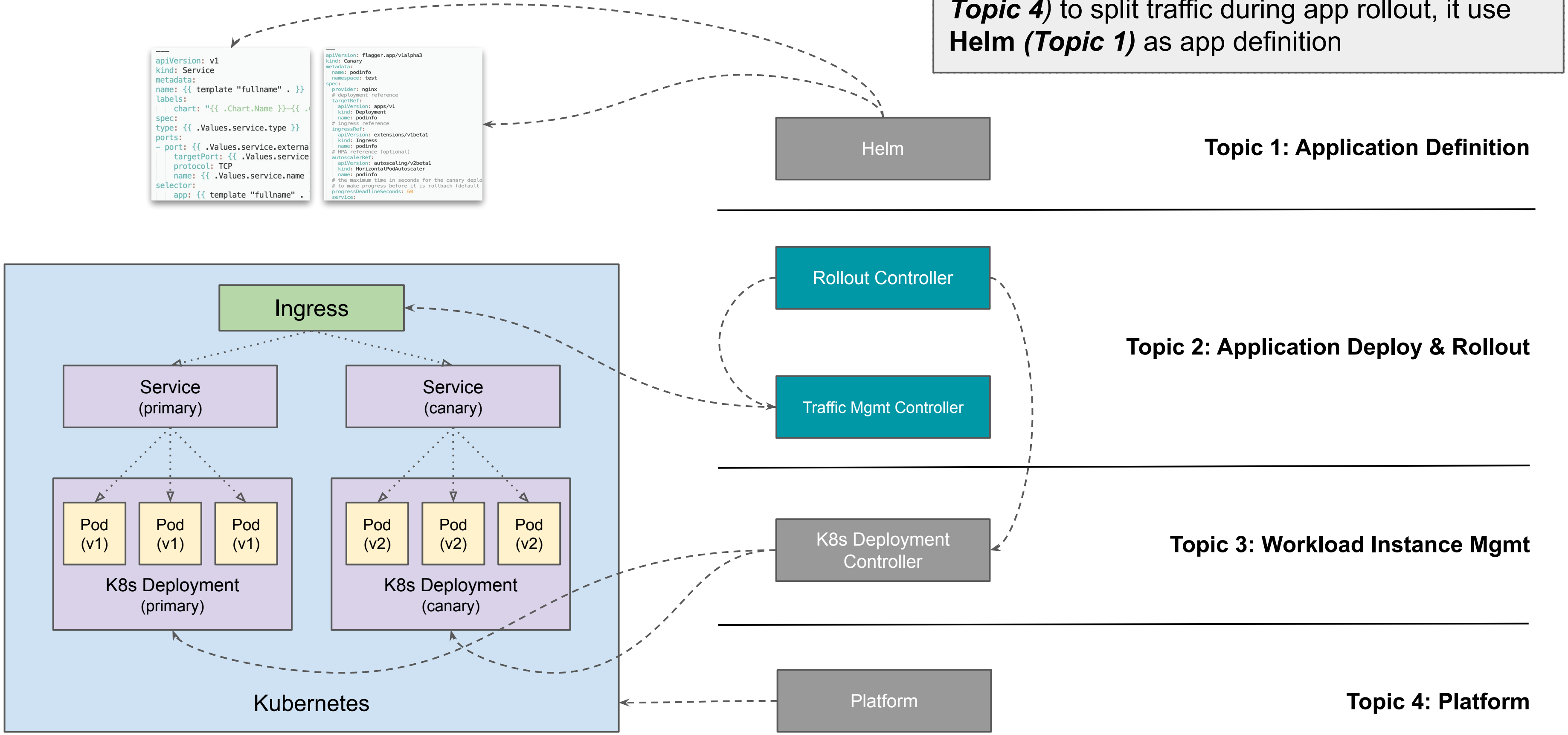
```yaml
apiVersion: flagger.app/v1alpha3
kind: Canary
metadata:
  name: podinfo
  namespace: test
spec:
  provider: nginx
  # deployment reference
  targetRef:
    ~~apiVersion: apps/v1~~
    ~~kind: Deployment~~
    name: podinfo
  # ingress reference
  ingressRef:
    apiVersion: extensions/v1beta1
    kind: Ingress
    name: podinfo
  # HPA reference (optional)
  autoscalerRef:
    apiVersion: autoscaling/v2beta1
    kind: HorizontalPodAutoscaler
    name: podinfo
  # the maximum time in seconds for the canary deployment
  # to make progress before it is rollback (default 600s)
  progressDeadlineSeconds: 60
  service:
    # ClusterIP port number
    port: 80
    # container port number or name
    targetPort: 9898
  canaryAnalysis:
    # schedule interval (default 60s)
    interval: 10s
    # max number of failed metric checks before rollback
    threshold: 10
    # max traffic percentage routed to canary
    # percentage (0-100)
    maxWeight: 50
    # canary increment step
    # percentage (0-100)
    stepWeight: 5
    # NGINX Prometheus checks
    metrics:
    - name: request-success-rate
      # minimum req success rate (non 5xx responses)
      # percentage (0-100)
```

myapp.com/v1alpha

Operator

Summary: let’s practice the model!

A: **Project X** mainly focuses on **Topic 2,** i.e. progressive app rollout. It uses **Deployment controller *(Topic 3)*** to manage workload instances and use Ingress *(provided by K8s in **Topic 4**)* to split traffic during app rollout, it use **Helm *(Topic 1)*** as app definition



```yaml
---
apiVersion: v1
kind: Service
metadata:
name: {{ template "fullname" . }}
labels:
    chart: "{{ .Chart.Name }}-{{ .(
spec:
type: {{ .Values.service.type }}
ports:
- port: {{ .Values.service.externa
    targetPort: {{ .Values.service
    protocol: TCP
    name: {{ .Values.service.name
selector:
    app: {{ template "fullname" .
```

```yaml
---
apiVersion: flagger.app/v1alpha3
kind: Canary
metadata:
  name: podinfo
  namespace: test
spec:
  provider: nginx
  # deployment reference
  targetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: podinfo
  # ingress reference
  ingressRef:
    apiVersion: extensions/v1beta1
    kind: Ingress
    name: podinfo
  # HPA reference (optional)
  autoscalerRef:
    apiVersion: autoscaling/v2beta1
    kind: HorizontalPodAutoscaler
    name: podinfo
  # the maximum time in seconds for the canary deplo
  # to make progress before it is rollback (default
  progressDeadlineSeconds: 60
  service:
```

Helm

Topic 1: Application Definition

Rollout Controller

Traffic Mgmt Controller

Topic 2: Application Deploy & Rollout

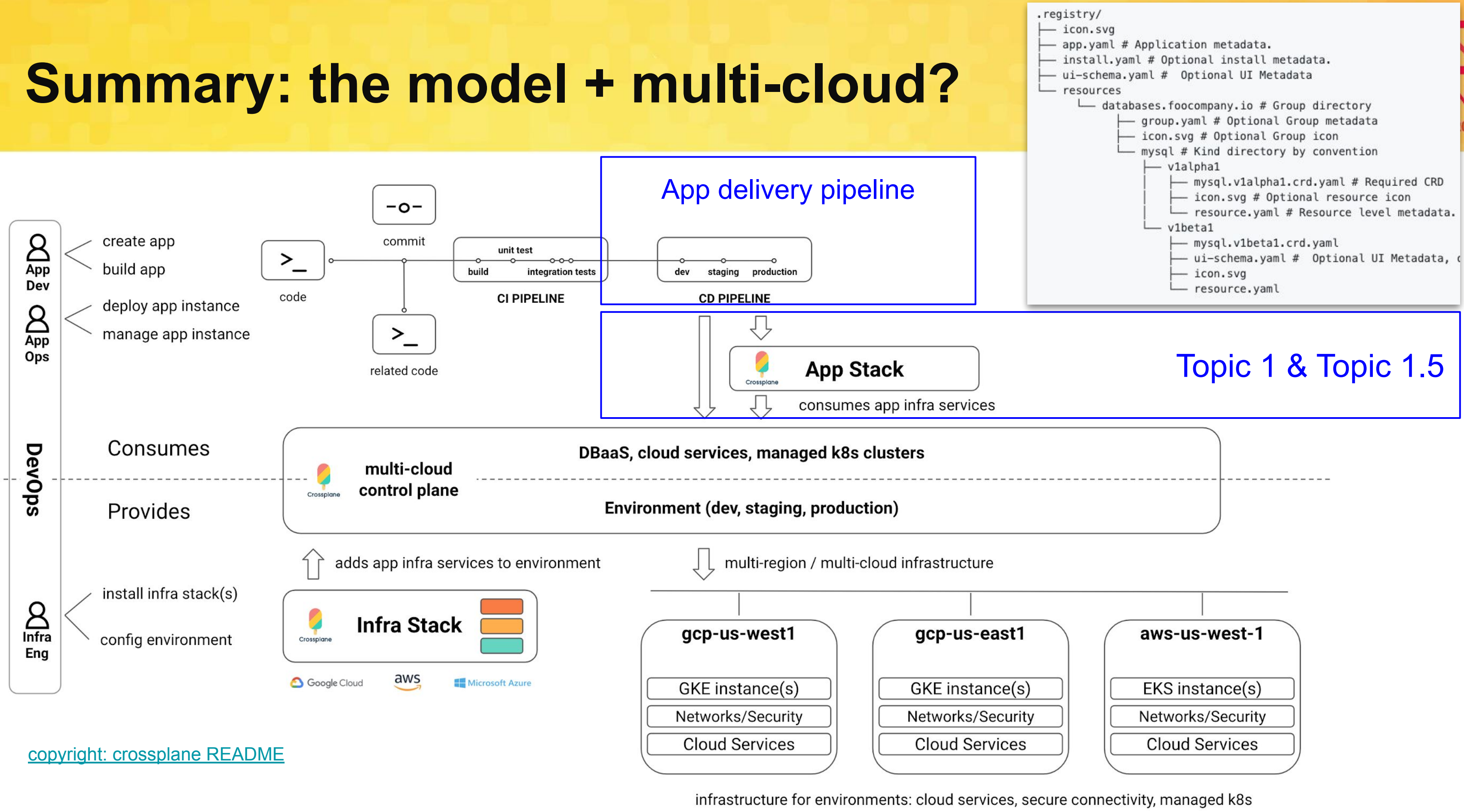
K8s Deployment Controller

Topic 3: Workload Instance Mgmt

Platform

Topic 4: Platform

Summary: the model + multi-cloud?



```
.registry/
├── icon.svg
├── app.yaml # Application metadata.
├── install.yaml # Optional install metadata.
├── ui-schema.yaml #  Optional UI Metadata
└── resources
    └── databases.foocompany.io # Group directory
        ├── group.yaml # Optional Group metadata
        ├── icon.svg # Optional Group icon
        └── mysql # Kind directory by convention
            ├── v1alpha1
            │   ├── mysql.v1alpha1.crd.yaml # Required CRD
            │   ├── icon.svg # Optional resource icon
            │   └── resource.yaml # Resource level metadata.
            └── v1beta1
                ├── mysql.v1beta1.crd.yaml
                ├── ui-schema.yaml #  Optional UI Metadata,
                ├── icon.svg
                └── resource.yaml
```

copyright: crossplane README

The multi-cloud control plane defines **app descriptors**, **packaging**, and sits in front of the delivery targets.

For more information, please check out: The Dictionary of Cloud-Native App Delivery

Join the Community:

SIG Home: https://github.com/cncf/sig-app-delivery

Mailing List: cncf-sig-app-delivery

Bi-Weekly Meeting:

- 1st and 3rd Wednesdays at 8am Pacific, 11am Eastern - starting November 6
- Zoom: https://zoom.us/j/7276783015