LINUXCON
containercon
CHINA 中国
CLOUDOPEN

THINK OPEN
开放性思维

Apache OpenWhisk + Kubernetes:

A Perfect Match for Your Serverless Platform

Ying Chun Guo
guoyingc@cn.ibm.com

Zhou Xing
xingzhou@qiyi.com

LF ASIA, LLC

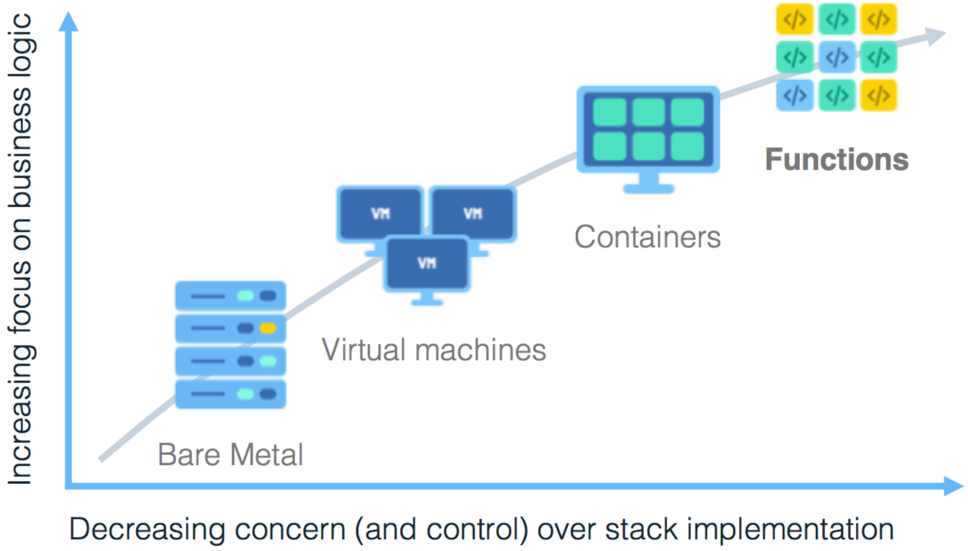
Agenda

- What is serverless?
- Kubernetes + Apache OpenWhisk
- Technical details
- Demo

What is serverless ?

What is serverless ?

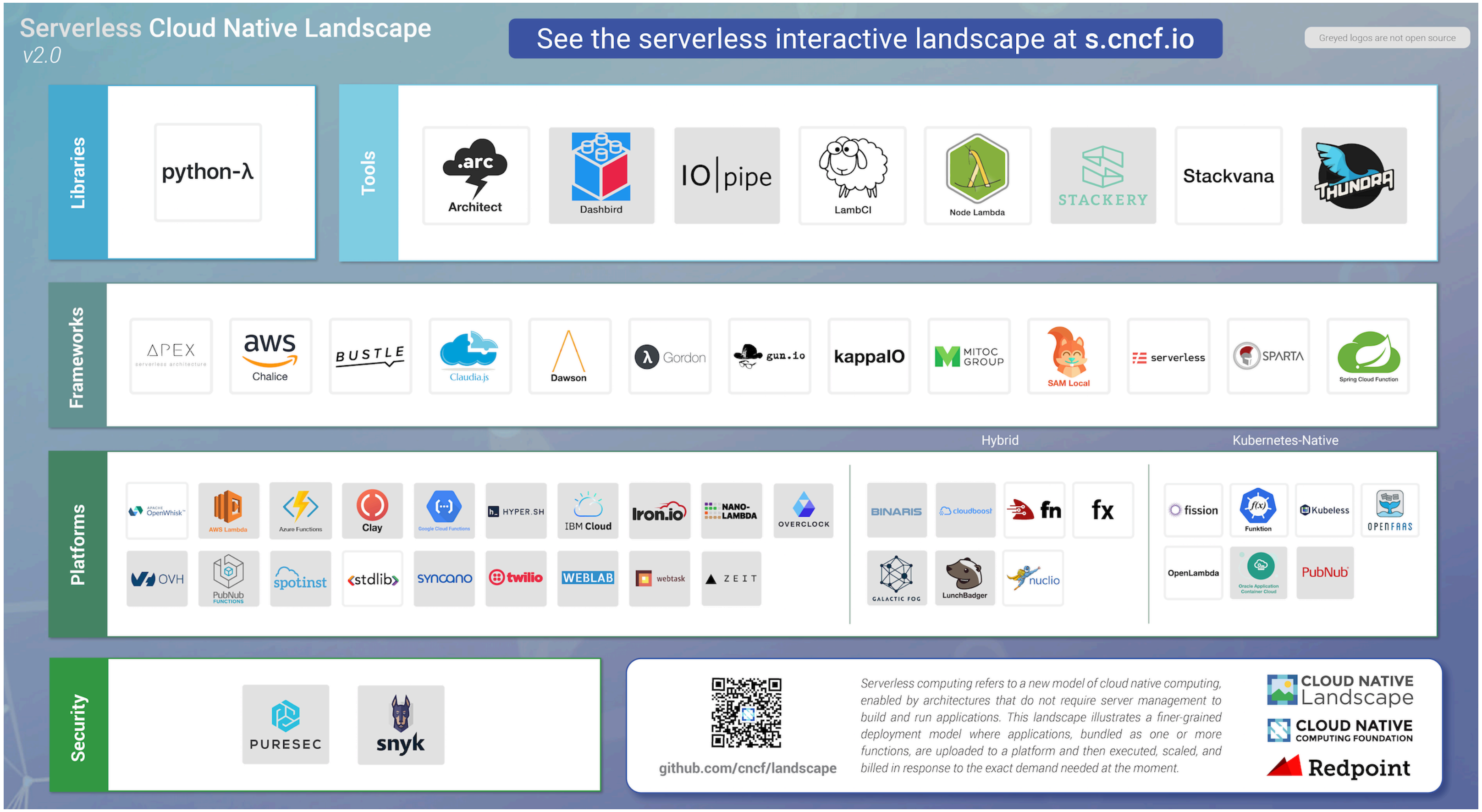
Serverless =
- **Functions as a Service**
- Backend as a Service


Increasing focus on business logic
Functions
Containers
VM
VM
VM
Virtual machines
Bare Metal
Decreasing concern (and control) over stack implementation

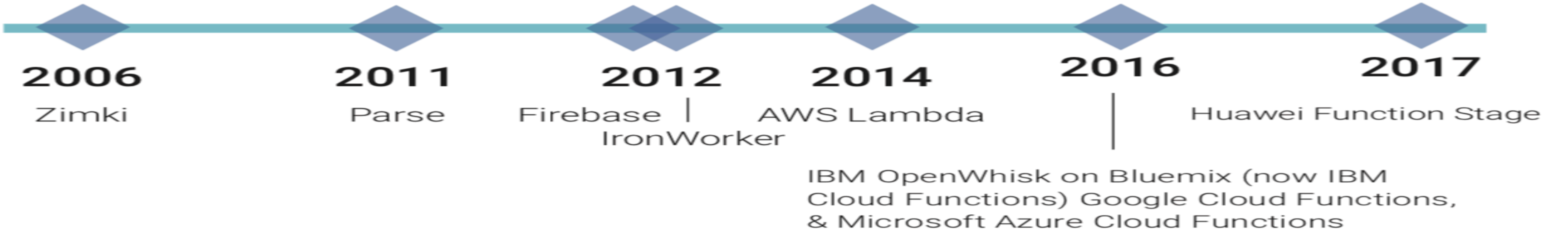
Benefits

- Zero server ops
  - No provisioning, updating, and managing server infrastructure.
  - Flexible Scalability
- No compute cost when idle

Serverless landscape defined in CNCF



Serverless computing refers to a new model of cloud native computing, enabled by architectures that do not require server management to build and run applications. This landscape illustrates a finer-grained deployment model where applications, bundled as one or more functions, are uploaded to a platform and then executed, scaled, and billed in response to the exact demand needed at the moment.



2006
Zimki

2011
Parse

2012
Firebase | IronWorker

2014
AWS Lambda

2016
IBM OpenWhisk on Bluemix (now IBM Cloud Functions) Google Cloud Functions, & Microsoft Azure Cloud Functions

2017
Huawei Function Stage

, LLC

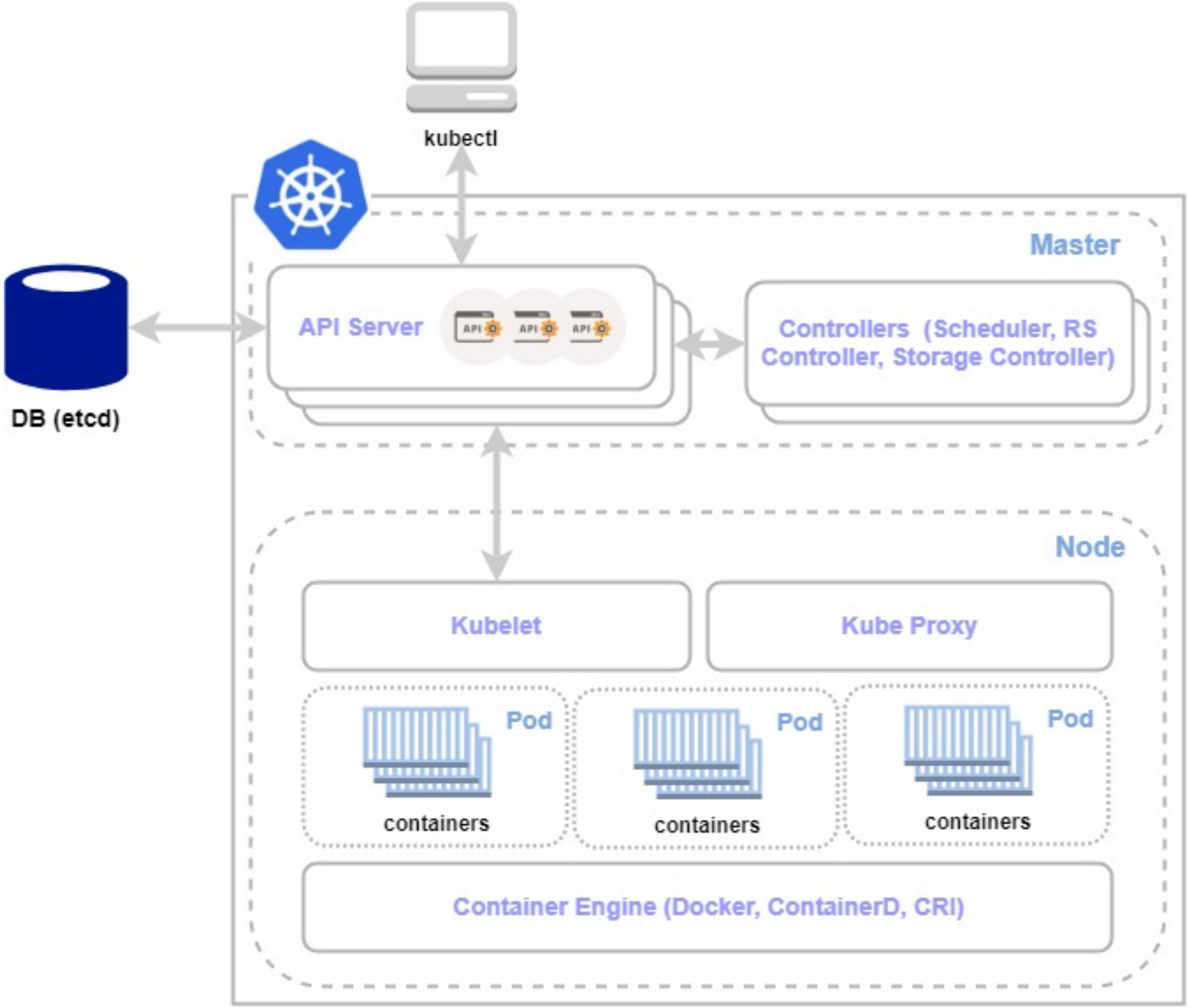
Kubernetes + Apache OpenWhisk

Kubernetes Introduction

- **K8s** is a production-grade container orchestration platform

- Declarative management of objects using configuration files.

- More introductions, go to
  - K8s official document http://kubernetes.io
  - Open Tech Mini Academy @ IBM http://ibm.biz/opentech-ma



Kubernetes Resource Model

A common resource model can satisfy any deployment requirements

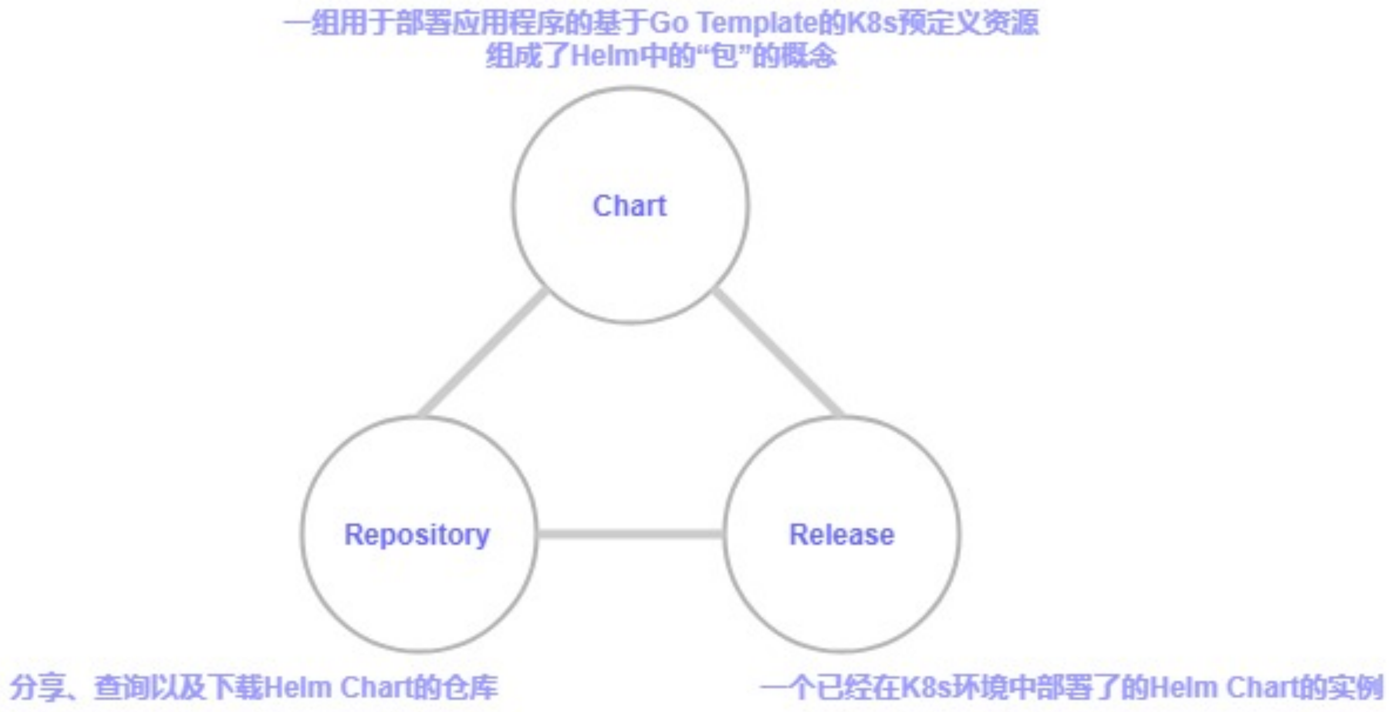
- **Config Maps**
- **Daemon Sets**
- **Deployments**
- **Events**
- **Endpoints**
- **Ingress**
- **Jobs**
- **Nodes**
- **Namespaces**
- **Pods**
- **Persistent Volumes**
- **Replica Sets**
- **Secrets**
- **Services**
- **Stateful Sets…**

- **K8s**通过这些资源模型构建应用程序
- 每一种资源都可以被用户所创建并存储在**K8s**数据库中
- 用户通过这些创建这些资源“描绘”应用程序在**K8s**平台上部署后的样子，**K8s**会根据这些资源的描述尽可能完成对应用程序和服务的部署
- 这其中，**Pod**包含了一组共享**Linux Namespace**的容器，是**K8s**平台所能调度的最小单元。其他多种资源，例如**Deployment**，**Job**等，都是构建在**Pod**的基础概念之上的。
- 用户可以通过**kubectl**配合描述资源的**yaml**文件创建这些资源

Helm

- The package manager for Kubernetes
- Easy to create, version, share, and publish — so start using Helm and stop the copy-and-paste madness.
- Help you define, install, and upgrade even the most complex Kubernetes application.
- Official community： https://helm.sh/

Core concepts in Helm



一组用于部署应用程序的基于Go Template的K8s预定义资源
组成了Helm中的“包”的概念

Chart

Repository

Release

分享、查询以及下载Helm Chart的仓库

一个已经在K8s环境中部署了的Helm Chart的实例

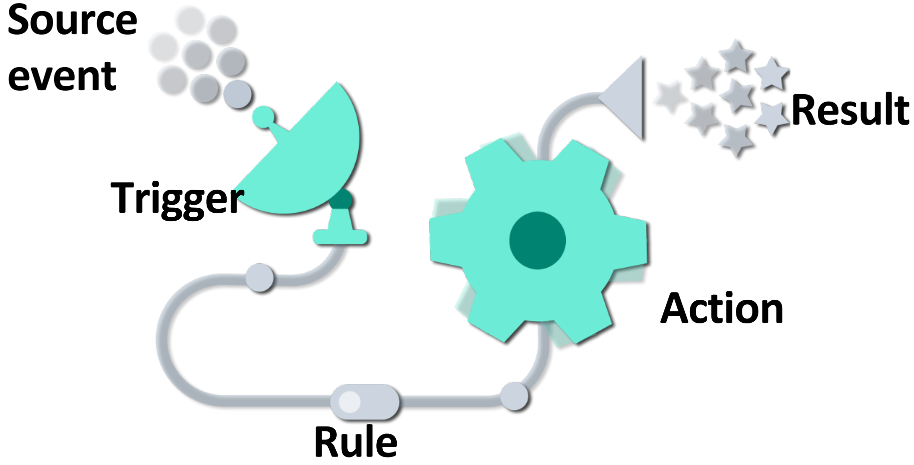
Helm installs *charts* into Kubernetes, creating a new *release* for each installation. And to find new charts, you can search Helm chart *repositories*.

Apache OpenWhisk

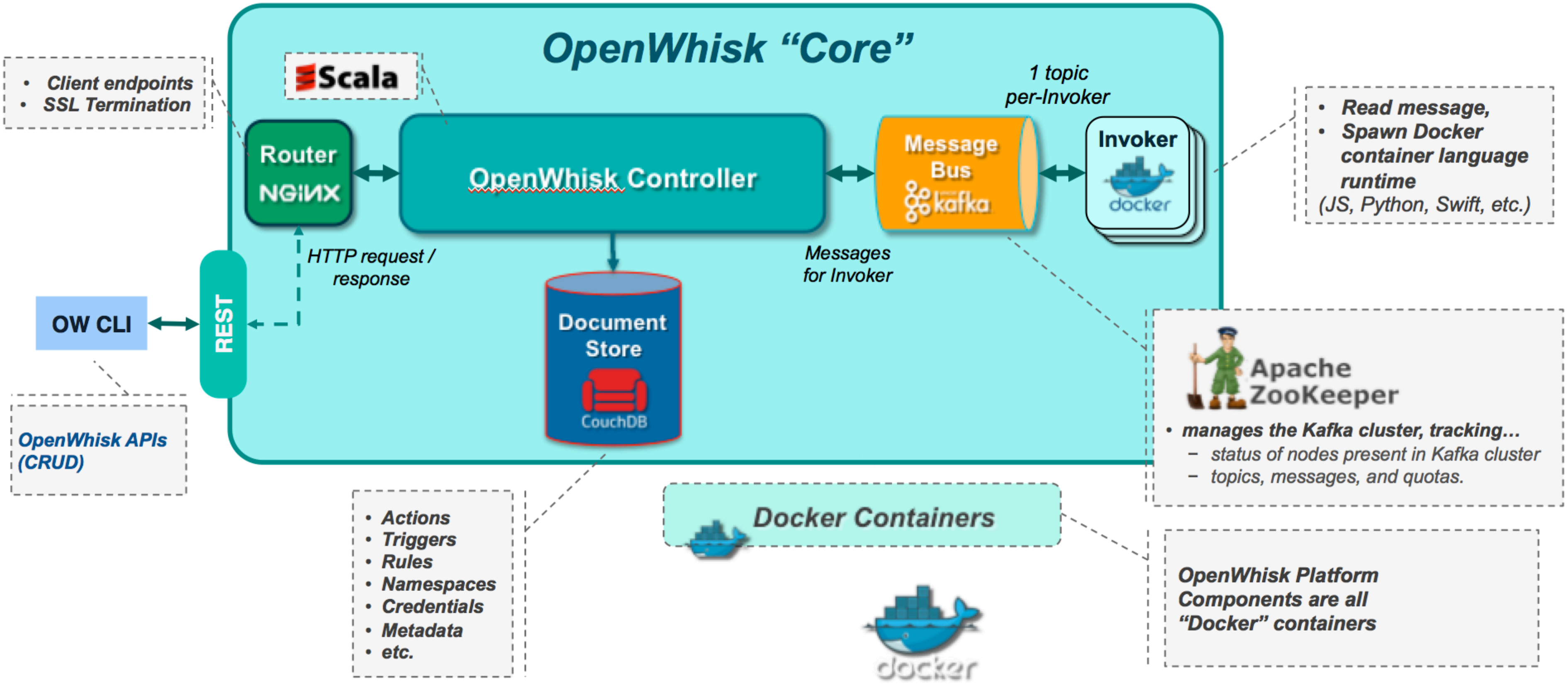
A serverless, open source cloud platform that executes functions in response to events at any scale.

Apache OpenWhisk *offers:*

- **Apache Software Foundation** (ASF)
  - *True, community-driven open source (Apache 2 License)*
- **Proven on IBM Cloud**
  - *Exact, same code in open source*



Architecture of Apache OpenWhisk



OpenWhisk "Core"

- Client endpoints
- SSL Termination

Scala

Router
NGINX

OpenWhisk Controller

1 topic
per-Invoker

Message
Bus
kafka

Invoker
docker

- Read message,
- Spawn Docker container language runtime

(JS, Python, Swift, etc.)

HTTP request / response

Messages
for Invoker

OW CLI

REST

Document
Store
CouchDB

OpenWhisk APIs
(CRUD)

Apache
ZooKeeper

- manages the Kafka cluster, tracking…
  - status of nodes present in Kafka cluster
  - topics, messages, and quotas.

Docker Containers

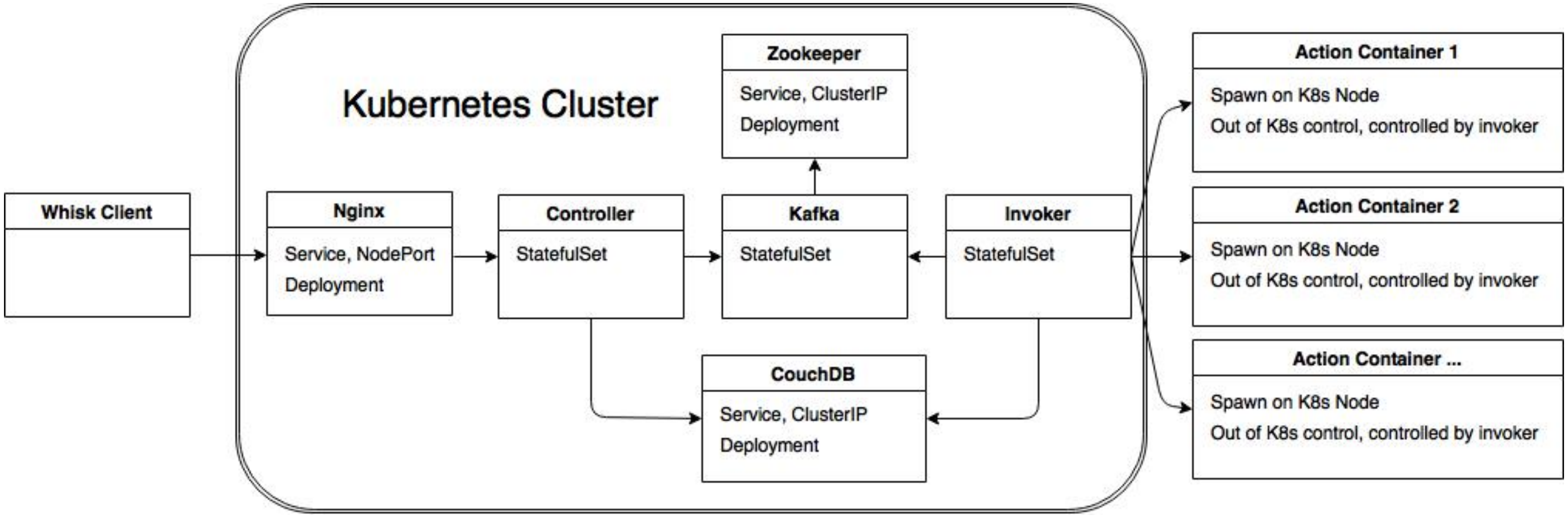
docker

- Actions
- Triggers
- Rules
- Namespaces
- Credentials
- Metadata
- etc.

OpenWhisk Platform
Components are all
"Docker" containers

Deploy Apache OpenWhisk on Kubernetes

- The architecture diagram of OpenWhisk components on Kubernetes, e.g.



- https://github.com/apache/incubator-openwhisk-deploy-kube

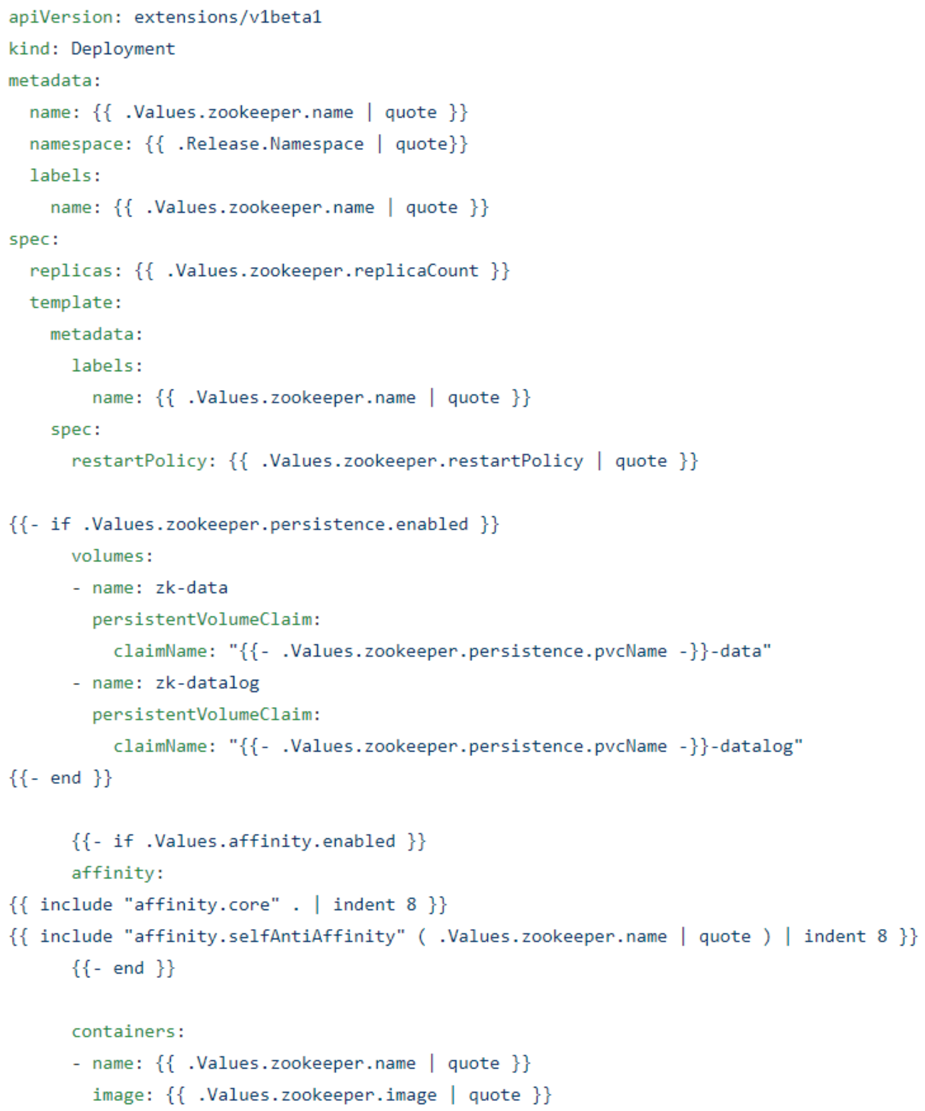
LINUXCON
containercon
CLOUDOPEN
CHINA 中国

THINK OPEN
开放性思维

Technical details

Deployment

- A ***Deployment*** controller provides declarative updates for **Pods** and **ReplicaSets**.
- Stands for a long running task, can be exposed as K8s services
- In OpenWhisk, usually, we deploy those core components' dependencies lib or tools as Deployment:
  - CouchDB
  - Redis
  - Zookeeper
  - Nginx



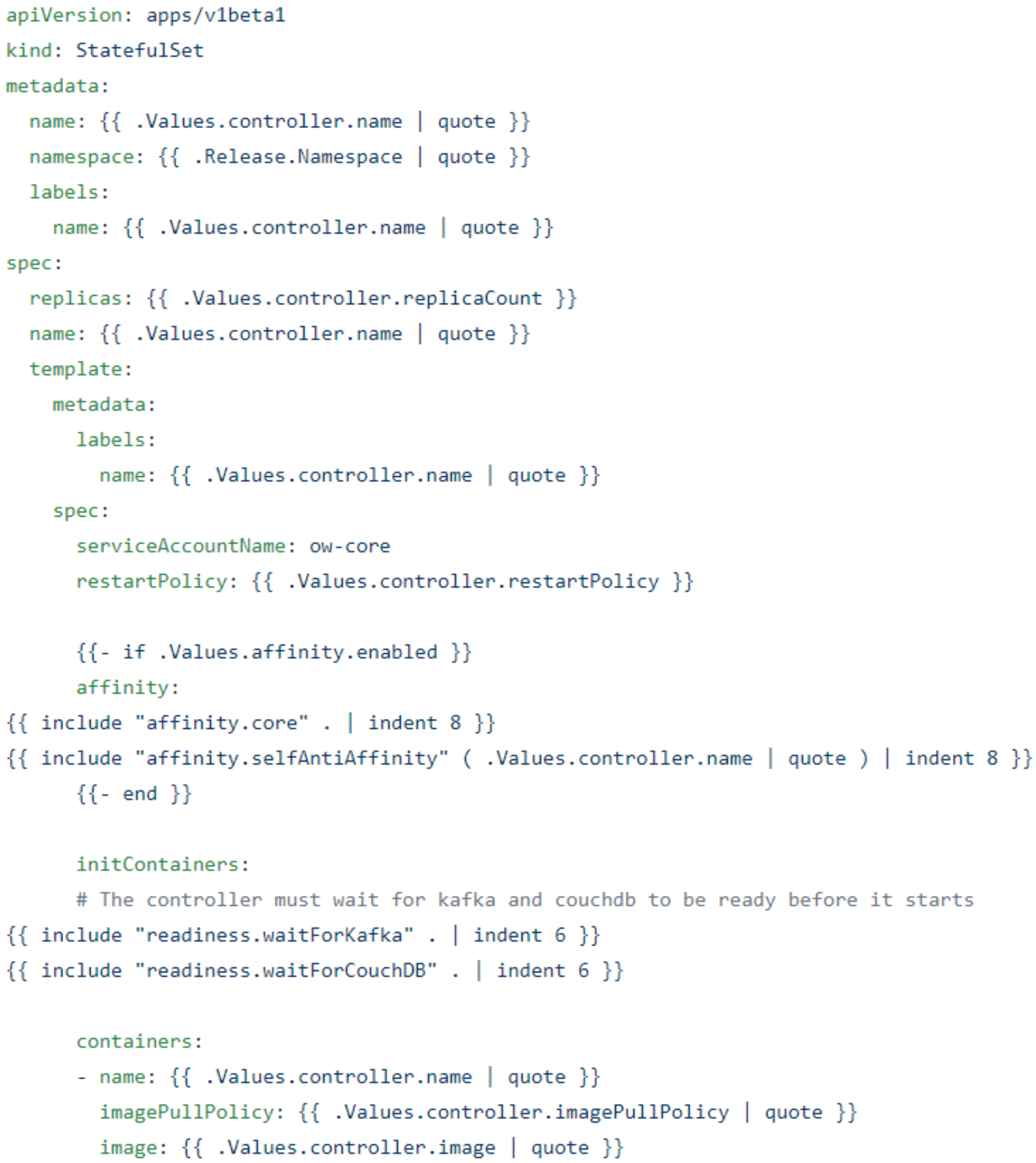
```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: {{ .Values.zookeeper.name | quote }}
  namespace: {{ .Release.Namespace | quote}}
  labels:
    name: {{ .Values.zookeeper.name | quote }}
spec:
  replicas: {{ .Values.zookeeper.replicaCount }}
  template:
    metadata:
      labels:
        name: {{ .Values.zookeeper.name | quote }}
    spec:
      restartPolicy: {{ .Values.zookeeper.restartPolicy | quote }}

{{- if .Values.zookeeper.persistence.enabled }}
      volumes:
      - name: zk-data
        persistentVolumeClaim:
          claimName: "{{- .Values.zookeeper.persistence.pvcName -}}-data"
      - name: zk-datalog
        persistentVolumeClaim:
          claimName: "{{- .Values.zookeeper.persistence.pvcName -}}-datalog"
{{- end }}

      {{- if .Values.affinity.enabled }}
      affinity:
{{ include "affinity.core" . | indent 8 }}
{{ include "affinity.selfAntiAffinity" ( .Values.zookeeper.name | quote ) | indent 8 }}
      {{- end }}

      containers:
      - name: {{ .Values.zookeeper.name | quote }}
        image: {{ .Values.zookeeper.image | quote }}
```

StatefulSet/DaemonSet

- ***StatefulSet*** is the workload API object used to manage stateful applications. Manages the deployment and scaling of a set of pods, *and provides guarantees about the ordering and uniqueness* of these Pods

- A ***DaemonSet*** ensures that all (or some) Nodes run a copy of a Pod

- In OpenWhisk, we deploy strictly mangaged objects as **StatefulSet** or **DaemonSet**:
  - Controller
  - Invoker
  - Kafka



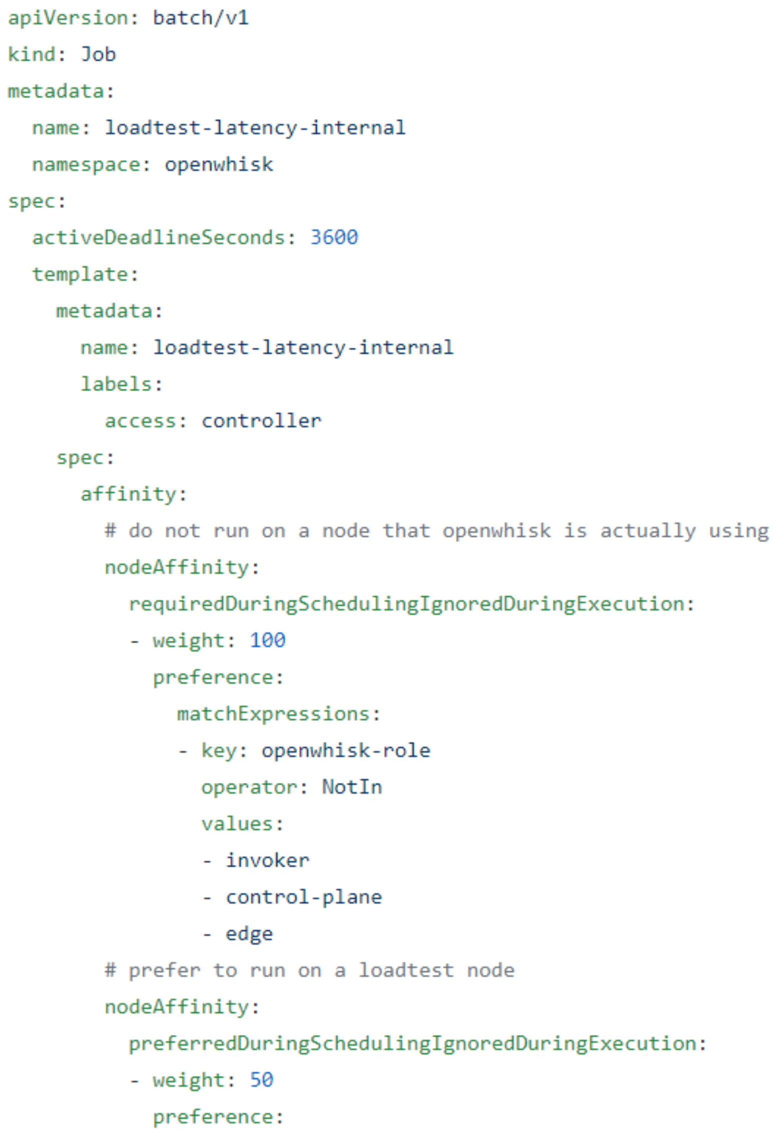
```
apiVersion: apps/v1beta1
kind: StatefulSet
metadata:
  name: {{ .Values.controller.name | quote }}
  namespace: {{ .Release.Namespace | quote }}
  labels:
    name: {{ .Values.controller.name | quote }}
spec:
  replicas: {{ .Values.controller.replicaCount }}
  name: {{ .Values.controller.name | quote }}
  template:
    metadata:
      labels:
        name: {{ .Values.controller.name | quote }}
    spec:
      serviceAccountName: ow-core
      restartPolicy: {{ .Values.controller.restartPolicy }}

      {{- if .Values.affinity.enabled }}
      affinity:
{{ include "affinity.core" . | indent 8 }}
{{ include "affinity.selfAntiAffinity" ( .Values.controller.name | quote ) | indent 8 }}
      {{- end }}

      initContainers:
      # The controller must wait for kafka and couchdb to be ready before it starts
{{ include "readiness.waitForKafka" . | indent 6 }}
{{ include "readiness.waitForCouchDB" . | indent 6 }}

      containers:
      - name: {{ .Values.controller.name | quote }}
        imagePullPolicy: {{ .Values.controller.imagePullPolicy | quote }}
        image: {{ .Values.controller.image | quote }}
```

Jobs

- A ***job*** creates one or more pods and ensures that a specified number of them successfully terminate.

- Job stands for a short running task

- In OpenWhisk, we used to deploy package installation and tasks like catalog installation as Job:
  - Package installation
  - Catalog installation



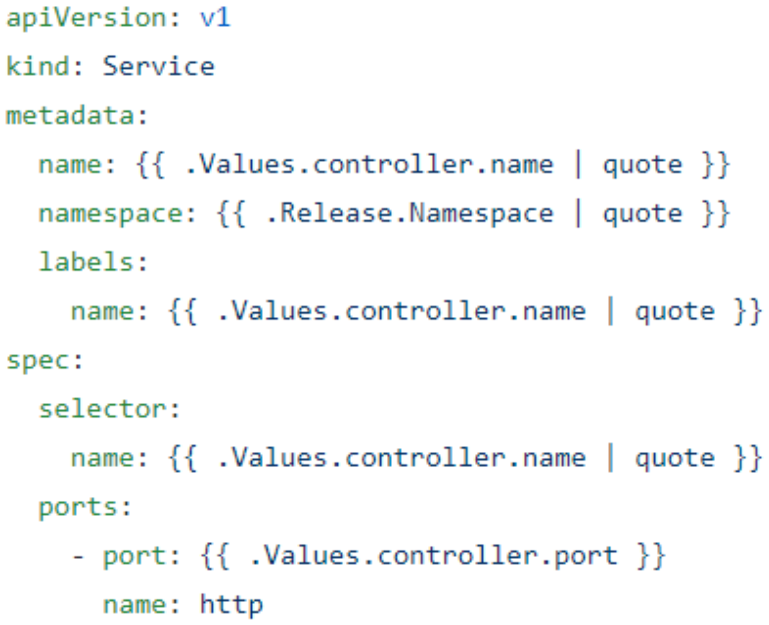
```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: loadtest-latency-internal
  namespace: openwhisk
spec:
  activeDeadlineSeconds: 3600
  template:
    metadata:
      name: loadtest-latency-internal
      labels:
        access: controller
    spec:
      affinity:
        # do not run on a node that openwhisk is actually using
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - weight: 100
            preference:
              matchExpressions:
              - key: openwhisk-role
                operator: NotIn
                values:
                - invoker
                - control-plane
                - edge
        # prefer to run on a loadtest node
        nodeAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 50
            preference:
```

Service

- A Kuberentes ***Servcie*** is an abstraction which defines a logical set of Pods and a policy by which to access them.

- *Service* provides a way for applications to communicate with each other on K8s platform

- In OpenWhisk, we deploy all the dependencies as service, usually, they are deployed as ClusterIP service:
  - Controller
  - Invoker
  - Nginx
  - Kafka
  - Zookeeper
  - Redis



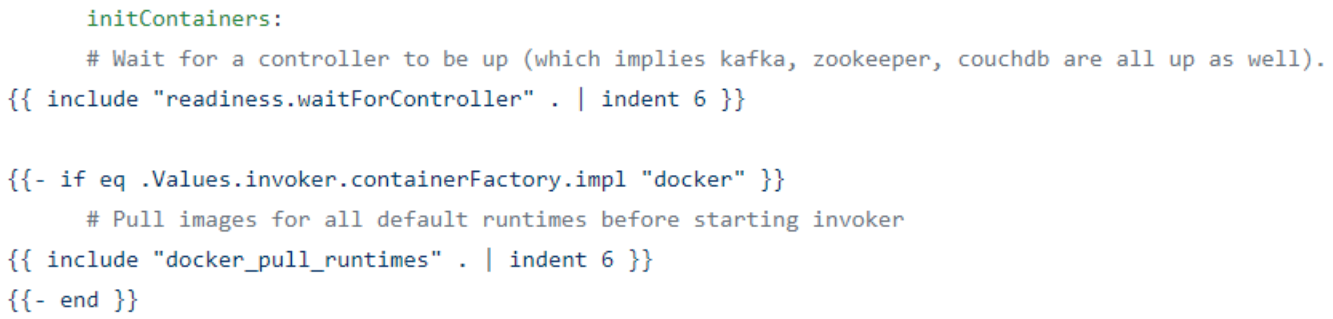
```
apiVersion: v1
kind: Service
metadata:
  name: {{ .Values.controller.name | quote }}
  namespace: {{ .Release.Namespace | quote }}
  labels:
    name: {{ .Values.controller.name | quote }}
spec:
  selector:
    name: {{ .Values.controller.name | quote }}
  ports:
    - port: {{ .Values.controller.port }}
      name: http
```

Other objects used in OW charts

- ConfigMap: like nginx deployment configuration
- Secrets: like DB access credentials
- Ingress

Component Launch Sequence

- In Kubernetes, we can use the following mechanisms to handle the component launch sequence:
  - Init Container: a pre-handling container to process staff which need to be done before the major costainer starts
  - Probe: readiness probe and liveness probe

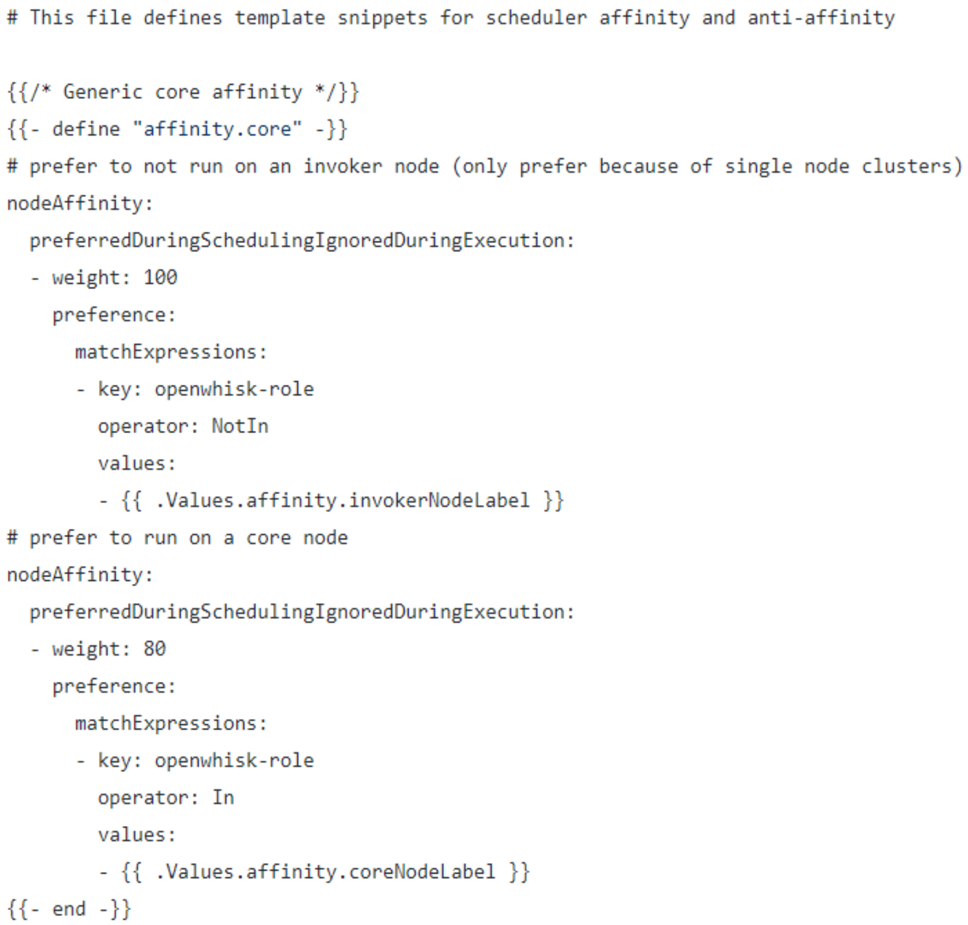


```
      initContainers:
      # Wait for a controller to be up (which implies kafka, zookeeper, couchdb are all up as well).
{{ include "readiness.waitForController" . | indent 6 }}

{{- if eq .Values.invoker.containerFactory.impl "docker" }}
      # Pull images for all default runtimes before starting invoker
{{ include "docker_pull_runtimes" . | indent 6 }}
{{- end }}
```

Component Deployment Topology

- Use affinity to make deployment topology policies for different component. E.g. controller node and DB node may not be assigned to the same K8s node

- Affinity type
  - Node Affinity
  - Pod Affinity



```
# This file defines template snippets for scheduler affinity and anti-affinity

{{/* Generic core affinity */}}
{{- define "affinity.core" -}}
# prefer to not run on an invoker node (only prefer because of single node clusters)
nodeAffinity:
  preferredDuringSchedulingIgnoredDuringExecution:
  - weight: 100
    preference:
      matchExpressions:
      - key: openwhisk-role
        operator: NotIn
        values:
        - {{ .Values.affinity.invokerNodeLabel }}
# prefer to run on a core node
nodeAffinity:
  preferredDuringSchedulingIgnoredDuringExecution:
  - weight: 80
    preference:
      matchExpressions:
      - key: openwhisk-role
        operator: In
        values:
        - {{ .Values.affinity.coreNodeLabel }}
{{- end -}}
```

LINUXCON

containercon

CLOUDOPEN

CHINA 中国

THINK OPEN
开放性思维

Demo

LF ASIA, LLC

Steps

1. Create a namespace
2. Label worker nodes to execute user actions
3. Create a mycluster.yaml file to customize the deployment
4. Deploy with Helm
5. Wait…and done

LINUXCON

containercon

CLOUDOPEN

CHINA 中国

THINK OPEN

开放性思维

LF ASIA, LLC