Kubernetes 完全教程

Kubernetes 的 API Spec 以及安全机制

王渊命 @jolestar

Agenda

1. Kubernetes 的 API Spec
    - YAML
    - API Spec 概述
    - Label 和 Selector 机制
    - Pod 详解
    - Service/ReplicaSet/Deployment/DaemonSet/StatefulSet/Job/CronJob/ConfigMap
    - Kubernetes API Object 的连带删除机制
2. Kubernetes 的安全机制
    - 认证（Authentication）
    - 授权（Authorization）
    - Admission Control

YAML 简介

> YAML is a human-readable data serialization language. It is commonly used for configuration files, but could be used in many applications where data is being stored or transmitted.

- 空格缩进表示层级关系，不支持 TAB
- : 用来分割 key/value
- - 表示数组元素，每行一个，也可以用方括号（[] ）和 逗号（,）来区分。
- 字符串可以不用引号，也支持 " 和 '
- 可以用 --- 区隔文档，把多个文档合并到同一个文件中
- 支持多行字符输入，通过 | 保留换行符，或者 > 折叠换行
- 支持锚点标记（&）和参考标记（*）避免重复
- # 表示注释
- 是 json 格式的超集

YAML Example

```yaml
# Invoice example
receipt:     Oz-Ware Purchase Invoice
customer:
    given:   Dorothy
items:
    - part_no:   A4786
      descrip:   Water Bucket (Filled)
      price:     1.47
      quantity:  4
    - part_no:   E1628
      descrip:   High Heeled "Ruby" Slippers
      price:     133.7
      quantity:  1
bill-to:  &id001
    street: |
            123 Tornado Alley
            Suite 16
    city:   East Centerville
ship-to:  *id001
specialDelivery:  >
    Follow the Yellow Brick
    Road to the Emerald City.
```

Kubernets API Spec 概述

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    app: web
    env: prod
  annotations:
    mydomain.com/custom-extend-key: value
spec:
  containers:
  - name: nginx
    image: nginx:1.7.9
    ports:
    - containerPort: 80
```

Kubernetes API Spec

- apiVersion: Object Schema 版本 (v1,batch/v1,storage.k8s.io/v1)
- kind: Object Schema 类型（List,Pod,Node,Service)
- metadata:
  - namespace
  - name
  - uid
  - resourceVersion
  - labels
  - annotations

Kubernetes API Spec

- spec
- status

Label 和 Selector

equality/inequality-based

```
environment = production
tier != frontend
```

set-based

```
environment in (production, qa)
tier notin (frontend, backend)
partition
!partition
```

```
kubectl get pods -l environment=production,tier=frontend
```

Pod

1. Pod 的实现原理（参看预备课）
2. Pod Spec
3. Pod 生命周期

Container

- command/args
- image/imagePullPolicy: Always, Never, IfNotPresent
- livenessProbe/readinessProbe: exec,httpGet,tcp
- resources
- volumeMounts

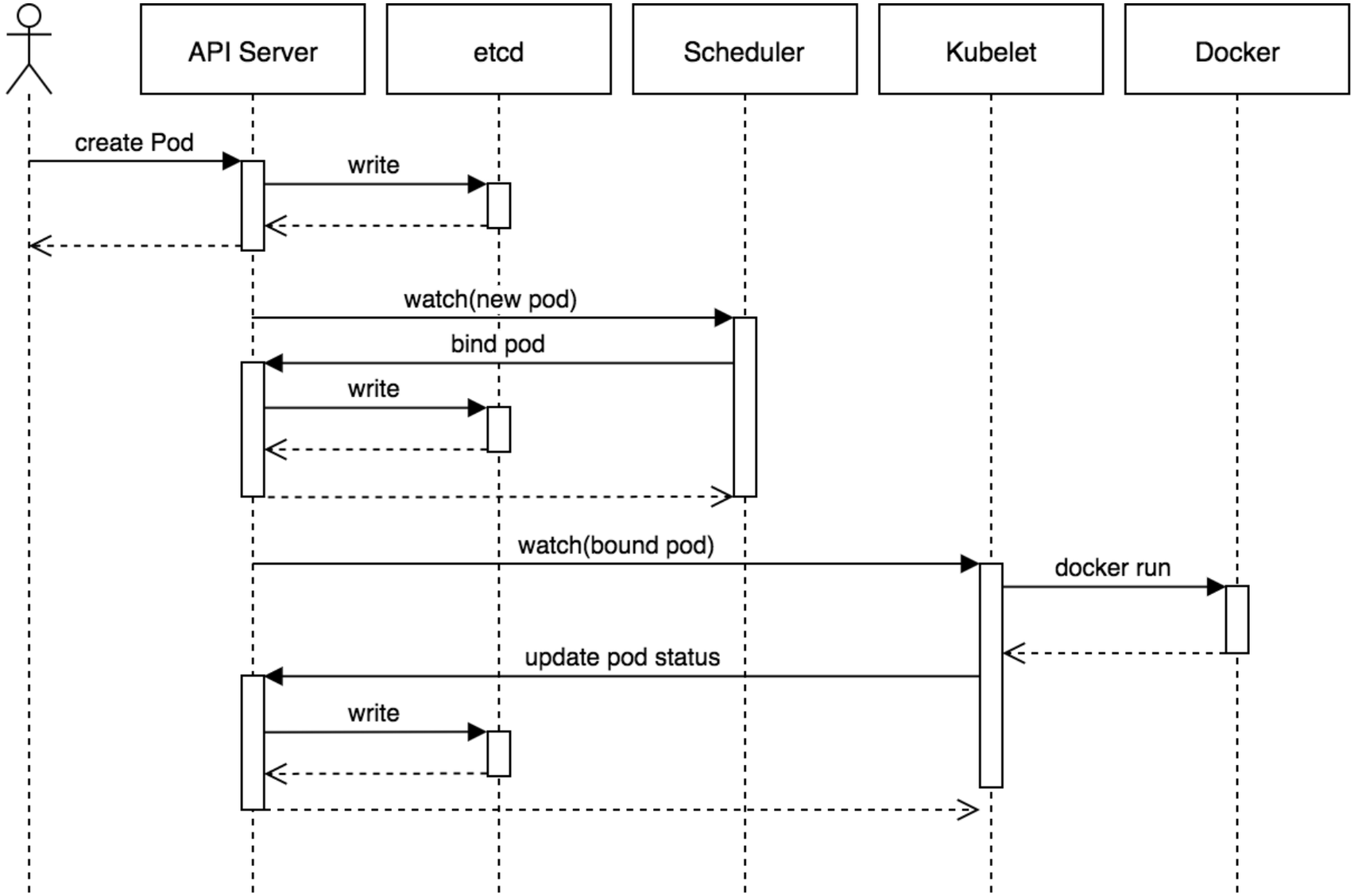
Pod Spec

- apiVersion: core/v1
- activeDeadlineSeconds/terminationGracePeriodSeconds
- dnsPolicy: ClusterFirst,ClusterFirstWithHostNet,Default
- hostNetwork/hostPID/hostIPC
- initContainers/containers
- imagePullSecrets
- serviceAccountName
- lifecycle: Hooks: postStart/preStop
- restartPolicy: Always, OnFailure, Never
- nodeName/nodeSelector
- tolerations
- affinity
- volumes

Pod 生命周期

- PodStatus
  - phase Pending/Running/Succeeded/Failed/Unknown
  - conditions PodScheduled,Initialized,Ready
  - containerStatuses

Pod 生命周期-创建



https://dzone.com/articles/kubernetes-lifecycle-of-a-pod

PodTemplateSpec

- metadata
- spec（PodSpec）

PodDisruptionBudget

```
apiVersion: policy/v1beta1
kind: PodDisruptionBudget
metadata:
  name: zk-pdb
spec:
  selector:
    matchLabels:
      app: zk
  maxUnavailable: 1
```

Service

- apiVersion: v1
- spec
  - clusterIP: None/IP
  - externalIPs
  - ports
  - selector
  - externalName
  - type: ExternalName, ClusterIP, NodePort, LoadBalancer

Service

```yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    k8s-app: kube-dns
  name: kube-dns
  namespace: kube-system
spec:
  clusterIP: 10.96.0.10
  ports:
  - name: dns
    port: 53
    protocol: UDP
    targetPort: 53
  selector:
    k8s-app: kube-dns
  type: ClusterIP
```

selector 只支持 equality-based

Endpoint

- addresses
- notReadyAddresses
- ports

```
apiVersion: v1
kind: Endpoints
metadata:
  name: helloworld
  namespace: default
subsets:
- addresses:
  - ip: 192.168.100.244
    nodeName: i-0elqbqzu
  - ip: 192.168.101.247
    nodeName: i-7d7zqw6x
  ports:
  - port: 80
    protocol: TCP
```

ReplicaSet (ReplicationController)

- apiVersion: apps/v1beta2 (1.8 以前是 extensions/v1beta1)
- spec
  - minReadySeconds
  - replicas
  - selector
  - template: PodTemplateSpec

Deployment

- apiVersion: apps/v1beta2 （1.8以前是 apps/v1beta1）
- spec
  - minReadySeconds
  - paused
  - replicas
  - selector
  - strategy: Recreate/RollingUpdate
    - rollingUpdate.maxSurge
    - rollingUpdate.maxUnavailable
  - revisionHistoryLimit
  - template: PodTemplateSpec

Deployment

```yaml
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

Deployment Rolling updates

```
kubectl set image deployment <deployment> <container>=<image>
kubectl rollout status deployment <deployment>
kubectl rollout pause deployment <deployment>
kubectl rollout resume deployment <deployment>
kubectl rollout undo deployment <deployment>
```

https://kubernetes.io/docs/tutorials/kubernetes-basics/update-intro/

DaemonSet

- apiVersion: apps/v1beta2 (1.8以前是 extensions/v1beta1)
- spec
  - minReadySeconds
  - revisionHistoryLimit
  - selector
  - template
  - updateStrategy: RollingUpdate,OnDelete

StatefulSet

- apiVersion: apps/v1beta2 (1.8以前是 apps/v1beta1)
- spec
  - podManagementPolicy: OrderedReady,Parallel
  - serviceName
  - replicas
  - revisionHistoryLimit
  - selector
  - template
  - updateStrategy: OnDelete,RollingUpdate
    - rollingUpdate.partition
  - volumeClaimTemplates: PersistentVolumeClaim

StatefulSet

- Pod Identity
  - ordinal: [0,N)
  - pod name: $(statefulset name)-$(ordinal)
  - pod dns: $(pod name).$(service name).$(namespace).svc.cluster.local
- Storage
  - PersistentVolume Provisioner
  - volumeClaimTemplates 的定义优先级高于 PodTemplate 中的定义
  - PVC 的生命周期独立于 StatefulSet

Job

- apiVersion: batch/v1
- spec
  - activeDeadlineSeconds
  - backoffLimit: default 6
  - completions
  - parallelism
  - selector
  - manualSelector
  - template: PodTemplateSpec
    - RestartPolicy: Never,OnFailure

Job

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    metadata:
      name: pi
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
      backoffLimit: 4
```

CronJob

- apiVersion: batch/v1beta1
- spec
  - concurrencyPolicy: Allow, Forbid, Replace
  - startingDeadlineSeconds
  - successfulJobsHistoryLimit: default 3
  - failedJobsHistoryLimit: default 1
  - jobTemplate: JobTemplateSpec
  - schedule: Cron format
  - suspend

CronJob

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hello
            image: busybox
            args:
            - /bin/sh
            - -c
            - date; echo Hello from the Kubernetes cluster
          restartPolicy: OnFailure
```

ConfigMap

- apiVersion: v1
- data: object

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: example-config
  namespace: default
data:
  examplekey: hello
  example.property: |-
    property.1=value-1
    property.2=value-2
    property.3=value-3
```

- 使用： env,volume

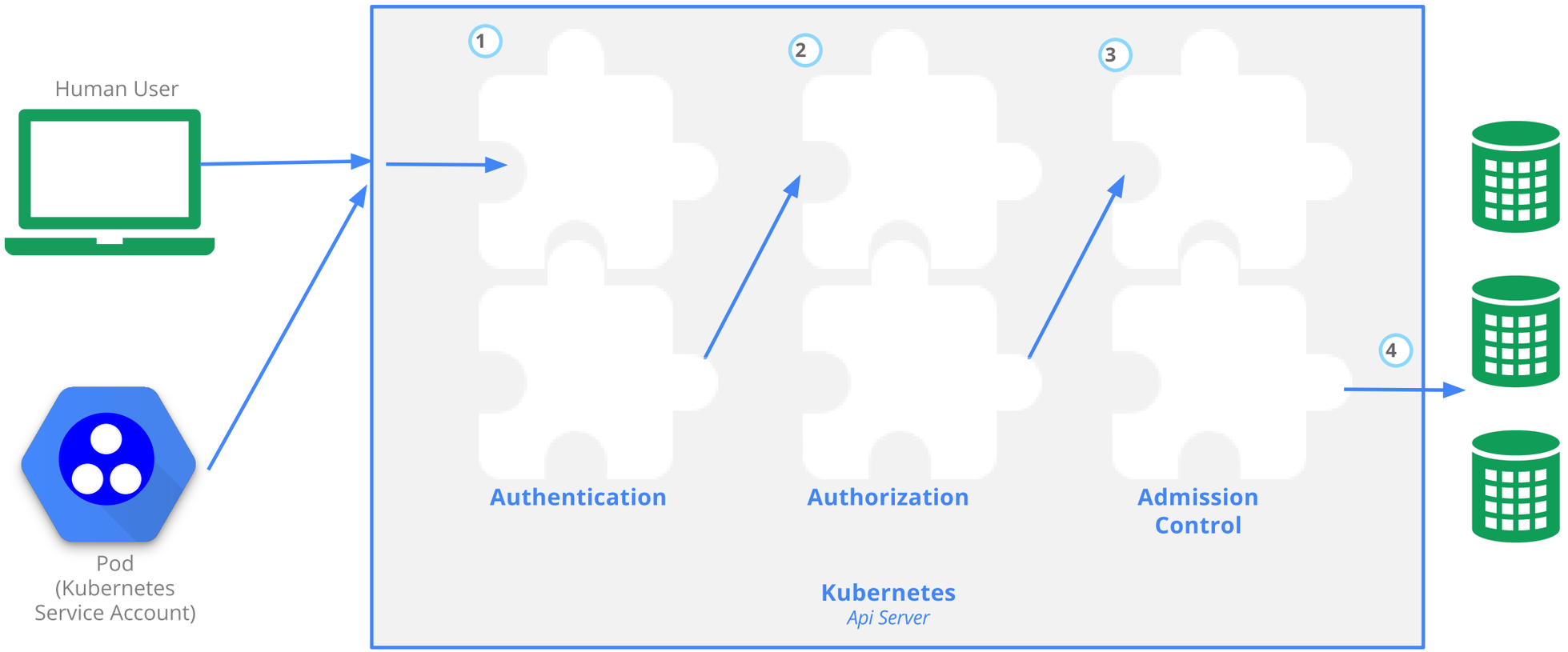
Kubernetes API Object 的连带删除机制

- Label/Selector
- metadata.ownerReferences
- kubectl delete --cascade=false

Kubernetes 安全机制

1. 认证（Authentication）： Client Certificates, Password, and Plain Tokens, Bootstrap Tokens, JWT Tokens
2. 授权（Authorization）： ABAC, RBAC, Webhook
3. Admission Control

Kubernetes 安全机制



Kubernetes 的认证（Authentication）

1. user (normal users, service accounts)
2. group

Kubernetes 的认证（Authentication）

- X509 Client Certs

```
openssl req -new -key jbeda.pem -out jbeda-csr.pem -subj "/CN=jbeda/O=app1/O=app2"
```

https://kubernetes.io/docs/tasks/tls/managing-tls-in-a-cluster/

- Static Token File
  - apiserver --token-auth-file=SOMEFILE

```
token,user,uid,"group1,group2,group3"
```

```
Authorization: Bearer token-xxxx
```

- Static Password File
  - apiserver --basic-auth-file=SOMEFILE

```
password,user,uid,"group1,group2,group3"
```

```
Authorization: Basic BASE64ENCODED(USER:PASSWORD)
```

Kubernetes 的认证（Authentication）

- OpenID Connect Tokens
  - OAuth2 + id_token(JSON Web Token)
- Webhook Token Authentication
  - apiserver
    - --authentication-token-webhook-config-file
    - --authentication-token-webhook-cache-ttl
- Authenticating Proxy
  - --requestheader-username-headers
  - --requestheader-group-headers

ServiceAcccount

- 一种将身份和资源绑定的机制（AWS Role, GCE ServiceAccount）
- 通过以下机制实现
  - ServiceAccount admission
    - 自动检查或者设置 pod 的 serviceAccountName
    - 增加 token volume 并且挂载（/var/run/secrets/kubernetes.io/serviceaccount）
  - Token controller
  - Service account controller
  - client sdk 自动读取固定路径下的 token
- 支持关联 imagePullSecret

Kubernetes 的授权（Authorization）

授权的主要要素

- User
- Group
- Verb: get, list, create, update, patch, watch, proxy, redirect, delete, deletecollection
- Resource
- Namespace
- API
- API Group

Kubernetes Authorization Modules

- Node
- ABAC

```
{"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy",
"spec": {"user": "alice", "namespace": "*", "resource": "*", "apiGroup": "*"}}
```

- RBAC
- Webhook

Kubernetes RBAC Authorization

Role/ClusterRole

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: secret-reader
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```

Kubernetes RBAC Authorization

RoleBinding/ClusterRoleBinding

```
kind: RoleBinding/ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: read-pods
  namespace: default
subjects:
- kind: User/Group/ServiceAccount
  name: jane
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role/ClusterRole
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

Admission Control

- 插件设置: apiserver --admission-control
- 主要插件:
  - AlwaysPullImages
  - DefaultStorageClass
  - GenericAdmissionWebhook
  - ImagePolicyWebhook
  - Initializers: https://github.com/kelseyhightower/kubernetes-initializer-tutorial
  - LimitRanger
  - PodNodeSelector
  - PodPreset
  - ResourceQuota: https://kubernetes.io/docs/concepts/policy/resource-quotas/
  - ServiceAccount
  - PodSecurityPolicy: https://kubernetes.io/docs/concepts/policy/pod-security-policy/

演示

1. DaemonSet

   https://github.com/QingCloudAppcenter/kubernetes/blob/master/k8s/addons/kube-proxy

2. Service 以及 Deployment 滚动升级

```
wget https://raw.githubusercontent.com/jolestar/kubernetes-complete-course/master/ex
kubectl apply -f helloworld.yaml
kubectl scale --replicas=6 deployment/helloworld
curl -H "accept: application/yaml" helloworld/status
kubectl get replicaset
kubectl set image deployment/helloworld web=jolestar/go-probe:v0.2
kubectl rollout status deployment/helloworld
kubectl rollout pause deployment/helloworld
kubectl get replicaset
kubectl rollout resume deployment/helloworld
kubectl rollout undo deployment/helloworld
curl -H "accept: application/yaml" helloworld/status
```

3. 将 pod 从 Deployment 摘除

4. RBAC 以及 ServiceAccount 的使用

https://github.com/QingCloudAppcenter/kubernetes/tree/master/k8s/addons/dashboard

演示

- 通过 StatefulSet 部署 zookeeper 集群

  https://kubernetes.io/docs/tutorials/stateful-application/zookeeper/

```
wget https://raw.githubusercontent.com/jolestar/kubernetes-complete-course/master/ex
kubectl apply -f zookeeper_mini.yaml
kubectl exec zk-0 -- cat /opt/zookeeper/conf/zoo.cfg
kubectl exec zk-0 -- /opt/zookeeper/bin/zkCli.sh create /test "test"
kubectl exec zk-0 -- cat /opt/zookeeper/bin/start-zookeeper|less
kubectl exec zk-0 -- /opt/zookeeper/bin/zkCli.sh create ls /
kubectl exec zk-1 -- /opt/zookeeper/bin/zkCli.sh create ls /
kubectl delete -f zookeeper_mini.yaml
kubectl get pvc
kubectl get pv
kubectl apply -f zookeeper_mini.yaml
kubectl exec zk-0 -- /opt/zookeeper/bin/zkCli.sh create ls /
```

总结

对 Kubernetes 的 Spec 以及安全机制有一个整体的理解

1. Kubernetes 的描述文件能做什么，如何将当前系统组件通过 Kubernetes 中的对象描述出来
2. Kubernetes 提供了哪些安全机制，如何和当前内部权限系统整合
3. Kubernetes 提供的扩展插件能做什么，如何通过插件在集群全局配置策略，约束和规范应用的部署行为

关于 API Spec 中的版本以及支持的属性，参看 Api Reference 1.7 1.8

下节课：通过应用案例解析 kubernetes 中的 pod 放置策略，autoscale 等。

作业

1. 通过 StatefulSet 部署 kafka，参看 https://github.com/kow3ns/kubernetes-kafka, 并理解如何利用 Kubernetes 的能力部署分布式系统。
2. 通过证书生成工具，生成一个普通用户证书并让 Kubernetes 批准，然后使用该证书操作集群，而不是默认的 admin 账号。
3. 思考并尝试通过 Kubernetes 搭建一个日志收集分析系统。

关于我

个人博客: http://jolestar.com

课程 Github： https://github.com/jolestar/kubernetes-complete-course

关注我们

QingCloud-IaaS

青云QingCloud

www.qingcloud.com

QINGCLOUD青云