Kubernetes Workloads

> A workload is an application running on Kubernetes.
>
> Single container or Several that work together.

Kubernetes中最小调度单元Pod。Kubernetes pods有明确的**生命周期**，为了维护Pod的生命周期，Kubernetes提供Workloads resource实现该需求。

期望的状态-->Kubernetes-->维护状态

Kuberentes提供几种内建的工作负载资源

- Deployment and ReplicaSet，无状态副本集
- StatefulSet
- DaemonSet
- Job or CronJob

Pod详解

- 最小部署单元
- Container组合
  - 共享存储
  - 网络
  - 运行容器的声明
  - 共同调度
  - init container
  - ephemeral container

What is a POD ?

Kubernetes中Pod的两种用法：

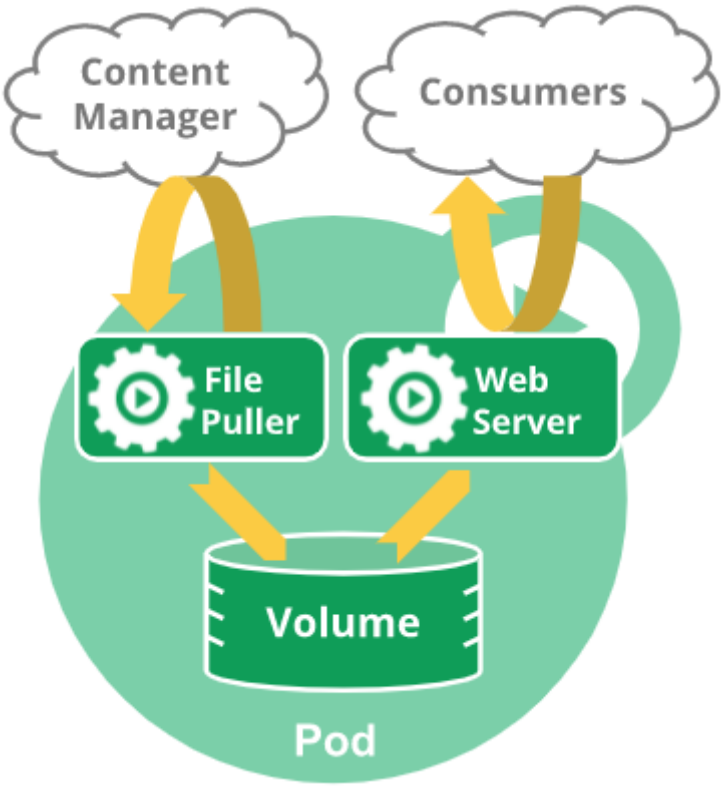
- 运行单个容器的POD
- 运行多个容器的POD

每个POD都是完成一个服务的单个实例，当希望横向扩展应用程序时，则应使用多个POD。我们称具有多个实例POD的服务称之为**多副本**服务

Workloads + Controller实现对一组POD副本的控制

管理多个容器

高耦合业务容器需求，适用于单个POD管理多个容器



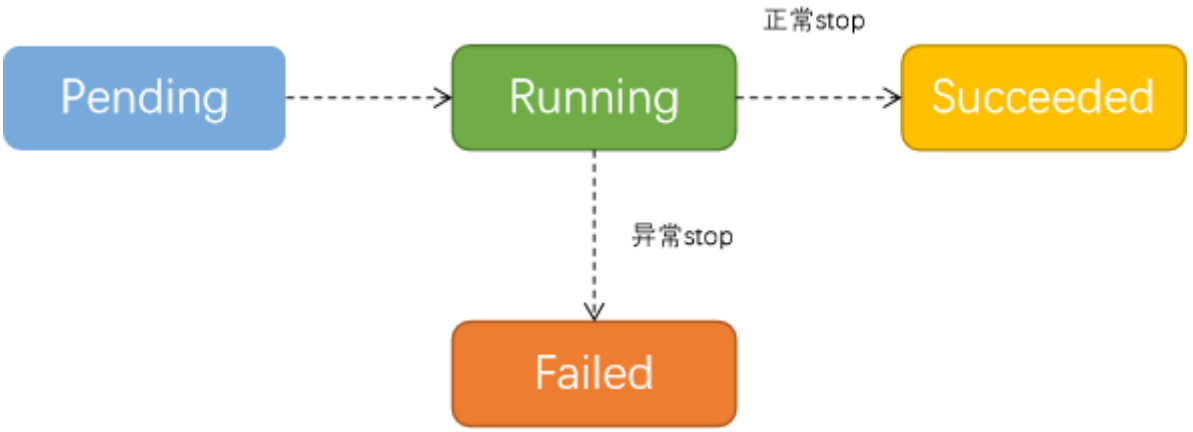
具有init container的Pod，init container会在启动应用容器前运行。 Pod内container拥有独立网络空间并可共享存储。

使用POD

作为最小调度单元的POD很少独立使用，而是通过Workload + Controller管理POD的生命周期。Workload控制全生命周期状态，controller负责满足我们对于POD运行状态的预期

各类资源定义中的PodTemplate提供Pod的定义。

POD的生命周期



在Pod运行期间：kubelet处理Pod内的container（可重启），以处理一些服务失效的场景，注意这里说的是重启Container

生命周期内仅被调度一次，POD一直驻留在被分配的节点，直到POD被STOP或TERMINATED

POD运行会被赋予一个UID，节点Down掉之后，会在给定的超时期限后删除POD

POD不能自恢复，Kubernetes使用Controller来控制POD的恢复过程

POD不能被重调度（rescheduled），只可以重建

POD PHASE

Pod的`status`字段是一个PodStatus对象，其中包含一个`phase`字段，显示POD的状态

VALUE	概述
Pending	Pod 已被Kubernetes系统接受，但有一个或者多个容器尚未创建亦未运行。此阶段包括等待Pod被调度的时间和通过网络下载镜像的时间
Running	Pod 已经绑定到了某个节点，Pod中所有的container都已被创建。至少有一个容器仍在运行，或者正处于启动或重启状态。
Succeeded	Pod中的所有容器都已成功终止，并且不会再重启。
Failed	Pod中的所有容器都已终止，并且至少有一个容器是因为失败终止。也就是说，容器以非 0 状态退出或者被系统终止
Unknown	因为某些原因无法取得Pod的状态。这种情况通常是因为与Pod所在主机通信失败。

如果某节点Down掉或者与集群中其他节点失联，Kubernetes会实施某种策略，将节点上运行的所有Pod的phase设置为Failed

Container Status

Kubernetes跟踪POD中所有container的状态

container状态机：

- Waiting
- Running
- Terminated

容器重启策略

POD的spec中有`restartPolicy`属性，可能取值包含Always、OnFailure和Never。策略适用于POD中的所有Container。

重启延时（指数回退10s, 20s, 40s, ...），最长5分钟，运行10分钟无异常，重置重启延时

POD Conditions

PodConditions是PodStatus对象的一个数组属性

- `PodScheduled`，已完成调度
- `ContainersReady`，所有container就绪
- `Initialized`，init containers成功启动
- `Ready`，可以提供服务，并join到endpoints中

Container Probes

Probe由kubelet定期对Container进行检测，kubelet可调用三种处理机制：

- ExecAction，容器内可执行的shell命令
- TCPSocketAction，运行一个针对容器内指定端口的TCP检测
- HTTPGetAction，使用`GET`方法发起一个HTTP请求，以状态码为状态依据

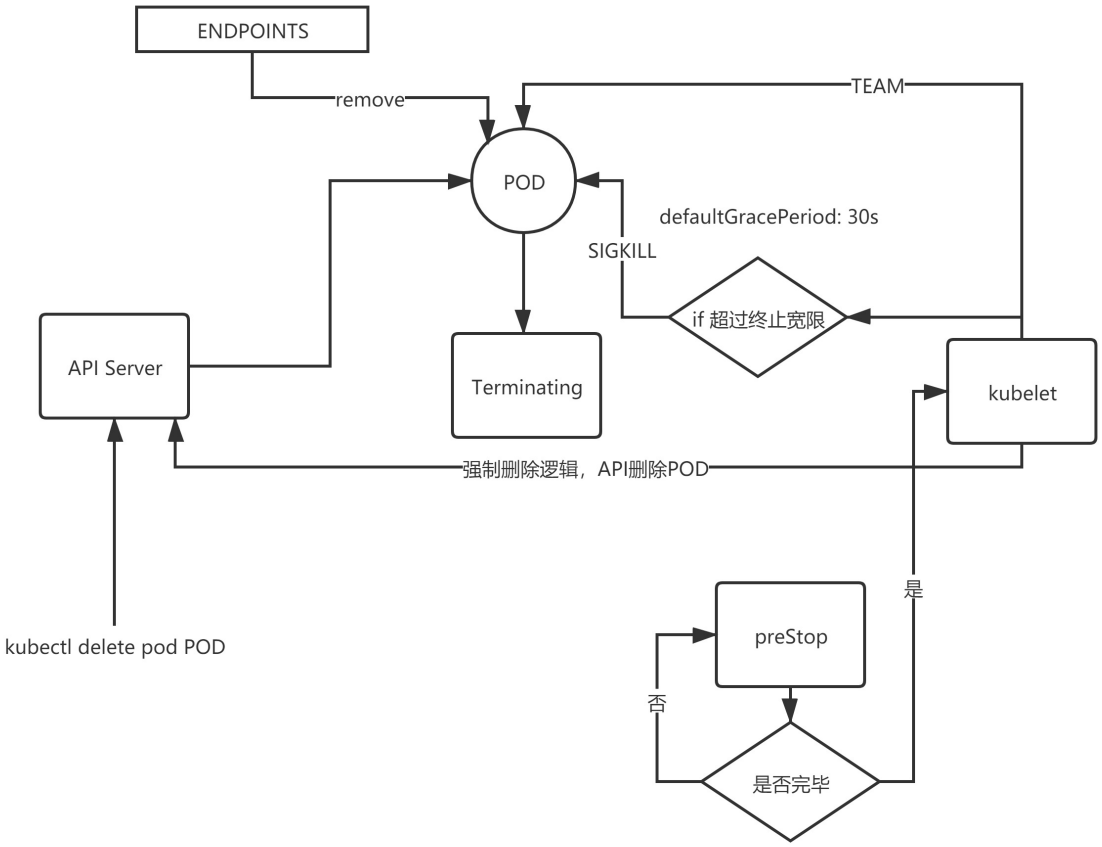
Probe状态机：

- Success
- Failure
- Unknown

Kubelet提供三种探针：

- livenessProbe：不设置默认通过，设置探针未通过将会根据重启策略重启指定container
- readinessProbe：不设置默认通过，设置探针未通过endpoints controller将会从endpoints的地址表中将失败POD的IP地址删除掉。在初始化延迟前该探针的状态为Failure
- startupProbe：该探针Success前其他探针失效，失败会根据重启策略重启指定container

Termination of Pods



INIT Container

Init Container是一种特殊容器，在 Pod 内的应用容器启动之前运行。Init 容器可以包括一些应用镜像中不存在的实用工具和安装脚本。

Init Container数量可以0-多个，为POD设置初始化容器，在pod的`spec`中的`initContainers`字段

initContainer支持除了probe外的所有container属性，多个initContainer会依据定义的顺序，顺序执行

initContainer的优势

- 可引入应用镜像中没有的实用工具
- 避免POD引入实用工具带来的安全风险
- 应用镜像的创建者和部署者可以解耦
- initContainer能以不同于 Pod 内应用容器的文件系统视图运行
- 延迟应用容器启动的一种机制

实例

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
```

```
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox:1.28
    command: ['sh', '-c', 'echo The app is running! && sleep 3600']
  initContainers:
  - name: init-myservice
    image: busybox:1.28
    command: ['sh', '-c', "until nslookup myservice.$(cat
/var/run/secrets/kubernetes.io/serviceaccount/namespace).svc.cluster.local; do echo
waiting for myservice; sleep 2; done"]
  - name: init-mydb
    image: busybox:1.28
    command: ['sh', '-c', "until nslookup mydb.$(cat
/var/run/secrets/kubernetes.io/serviceaccount/namespace).svc.cluster.local; do echo
waiting for mydb; sleep 2; done"]
```

initContainer行为

- 如果POD的`restartPolicy`设置为`Always`，initContainer失败时会使用`OnFailure`
- Pod重启，所有initContainer必须重新执行
- 修改initContainer `image`字段，POD将重启，initContainer内的代码应该是**幂等**的
- init Container的name必须唯一

POD

```
apiVersion: v1
kind: Pod
metadata: ObjectMeta
spec: PodSpec
status: PodStatus
```

ObjectMeta

```
name: string
generateName: string
namespace: string
lables: map[string]string
annotations: map[string]string
```

PodSpec

```
containers: []Container,required
initContainers: []Container
```

```
imagePullSecrets: []LocalObjectReference
enableServiceLinks: boolean
volumes: []Volume
nodeSelector: map[string]string
nodeName: string
affinity:
  nodeAffinity:
    preferredDuringSchedulingIgnoredDuringExecution:
    - preference:
        matchExpressions: []NodeSelectorRequirement
        matchFields: []NodeSelectorRequirement
      weight: int32
    requiredDuringSchedulingIgnoredDuringExecution:
    - matchExpressions: []NodeSelectorRequirement
      matchFields: []NodeSelectorRequirement
  podAffinity: PodAffinity
  podAntiAffinity: PodAntiAffinity
tolerations:
- key: string
  operator: string  # Exists,Equal
  value: string
  effect: string
  tolerationSeconds: int64
schedulerName: string
runtimeClassName: string
priorityClassName: string
priority: int32
topologySpreadConstraints:
  maxSkew: int32    # Pod不均匀分布的程度
  topologyKey: string   # 拓扑结构lable（On Node），具有相同key，value视为同一图拓扑
  whenUnsatisfiable: string #指出如何处理不满足约束的情况DoNotSchedule,ScheduleAnyway
  labelSelector: LabelSelector
restartPolicy: string   # Always,OnFailure,Never
terminationGracePeriodSeconds: int64    # TERM racefully持续时间
activeDeadlineSeconds:  int64
readinessGates:
  conditionType: string
hostnme: string
setHostnameAsFQDN: booleansubdomain
subdomain: string
hostAliases:
- hostnames: []string
  ip: string
dnsConfig:
  nameservers: []string
  options:
  - name: string
    value: string
  searches: []string
dnsPolicy: string   # ClusterFirst,ClusterFirstWithHostNet,ClusterFirst,Default,None
hostNetwork: boolean
hostPID: boolean
hostIPC: boolean
shareProcessNamespace: boolean
serviceAccountName: string
automountServiceAccountToken: boolean
securityContext:
```

```
    runAsUser: int64
    runAsNonRoot: boolean
    runAsGroup: int64
    supplementalGroups: []int64
    fsGroup: int64
    fsGroupChangePolicy: string
    seccompProfile:
      type: string
      localhostProfile: string
    seLinuxOptions:
      level: string
      role: string
      type: string
      user: string
    sysctls:
      name: string
      value: value
    windowsOptions:
      gmsaCredentialSpec: string
      gmsaCredentialSpecName: string
      runAsUserName: string
  ephemeralContainers: []EphemeralContainer   # Alpha功能
```

Container

```
image: string
imagePullPolicy: string # Always,Never,IfNotPresent
command: []string
args: []string
workingDir: string
ports:
  containerPort: int32
  hostIP: string
  hostPort: int32
  name: string
  protocol: string
env:
- name
  value
  valueFrom:
    configMapKeyRef:
      key: string
      name: string
      optional: string
    fieldRef:
      fieldPath: string
      apiVersion: string
    resourceFieldRef:
      resource: string
      containerName: string
      divisor: Quantity
    secretKeyRef:
      key: string
      name: string
      optional: boolean
```

```
envFrom:
  configMapRef:
    name: string
    optional: boolean
  prefix: string
  secretRef:
    name: string
    optional: boolean
volumeMounts:
  mountPath: string
  name: string
  mountPropagation: string
  readOnly: boolean
  subPath: string
  subPathExpr: string
volumeDevices:
  devicePath: string
  name: string
resources:
  limits: map[string]Quantity
  requests: map[string]Quantity
lifecycle:
  postStart:
    exec:
      command: []string
    httpGet:
      port: IntOrString
      host: string
      httpHeaders:
      - name: string
        value: string
      path: string
      scheme: HTTP
    tcpSocket:
      port: IntOrString
      host: string
  preStop: Handler
terminationMessagePath: string
terminationMessagePolicy: string
livenessProbe: Probe
readinessProbe: Probe
startupProbe:
  exec:
    command: []string
  httpGet:
    port: IntOrString
    host: string
    httpHeaders:
      name: string
      value: string
    path: string
    scheme: string
  tcpSocket:
    port: IntOrString
    host: string
  initialDelaySeconds: int32
  terminationGracePeriodSeconds: int64
  periodSeconds: int32
```

```
    timeoutSeconds: int32
    failureThreshold: int32
    successThreshold: int32
  stdin: boolean
  stdinOnce: boolean
  tty: boolean
```

DeploymentSpec

```
selector: LabelSelector
replicas: int32
template: PodTemplateSpec
minReadySeconds: int32
strategy:
  type: string
  rollingUpdate:
    maxSurge: IntOrString
    maxUnavailable: IntOrString
revisionHistoryLimit: int32
progressDeadlineSeconds: int32
paused: boolean
```

StatefulSetSpec

```
serviceName: string
selector: LabelSelector
template: PodTemplateSpec
replicas: int32
updateStrategy:
  type: string
  rollingUpdate:
    partition: int32
podManagementPolicy: OrderedReady   # Parallel
revisionHistoryLimit: 10
volumeClaimTemplates:
- accessModes: []string
  selector: LabelSelector
  resources:
    limits: map[string]Quantity
    requests: map[string]Quantity
  volumeName: string
  storageClassName: string
  volumeMode: string
```

DaemonSetSpec

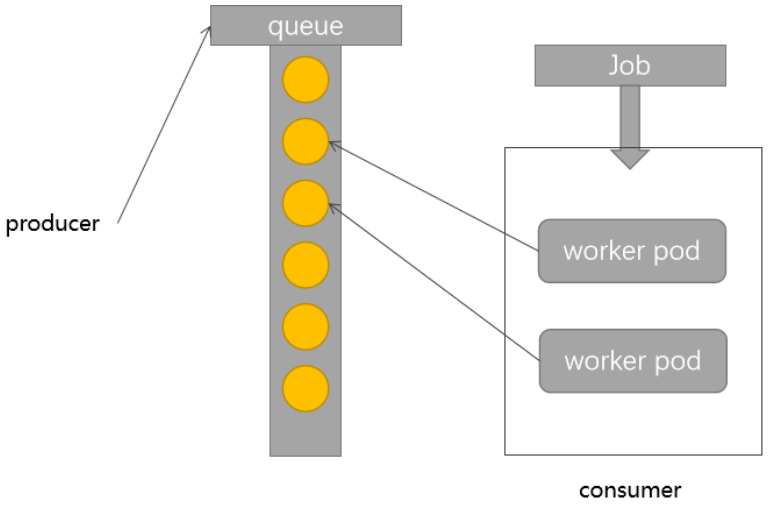
```
selecor: LabelSelector
template: PodTemplateSpec
minReadySeconds : int32
updateStrategy:
  type: string
  rollingUpdate:
    maxSurge: IntOrString
    maxUnavailable: IntOrString
revisionHistoryLimit: int32
```

JobSpec

```
template: PodTemplateSpec
parallelism: int32
completions: int32
completionMode: string  #Indexed, NonIndexed
```

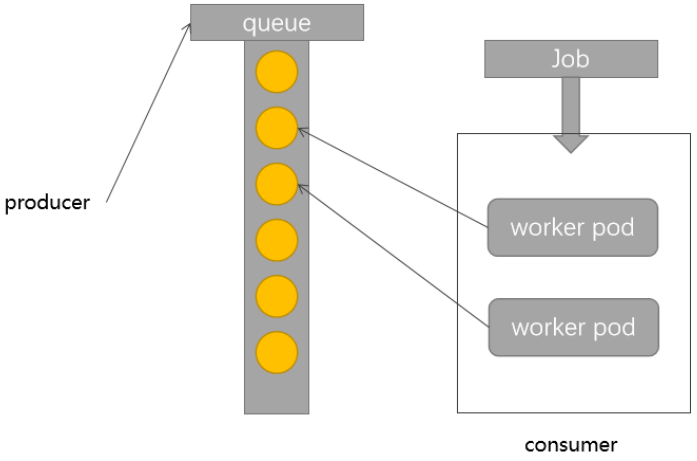
适合以 Job 形式来运行的任务主要有三种：

1. 非并行 Job：
   - 通常只启动一个 Pod，除非该 Pod 失败。
   - 当 Pod 成功终止时，立即视 Job 为完成状态。
2. 具有 确定完成计数 的并行 Job：
   - .spec.completions 字段设置为非 0 的正数值。
   - Job 用来代表整个任务，当成功的 Pod 个数达到 .spec.completions 时，Job 被视为完成。
   - 当使用 .spec.completionMode="Indexed" 时，每个 Pod 都会获得一个不同的 索引值，介于 0 和 .spec.completions-1 之间。
3. 带 工作队列 的并行 Job：
   - 不设置 spec.completions，默认值为 .spec.parallelism。
   - 多个 Pod 之间必须相互协调，或者借助外部服务确定每个 Pod 要处理哪个工作条目。 例如，任一 Pod 都可以从工作队列中取走最多 N 个工作条目。
   - 每个 Pod 都可以独立确定是否其它 Pod 都已完成，进而确定 Job 是否完成。
   - 当 Job 中 任何 Pod 成功终止，不再创建新 Pod。
   - 一旦至少 1 个 Pod 成功完成，并且所有 Pod 都已终止，即可宣告 Job 成功完成。
   - 一旦任何 Pod 成功退出，任何其它 Pod 都不应再对此任务执行任何操作或生成任何输出。所有 Pod 都应启动退出过程。



每处理完一个Pod，直接退出
直至completions达到预期值

Job completions=None and parallelism=2



执行完成的pod不退出，直至Queue清空，
有Pod成功退出，代表Job完成

CronJobSpec

```
jobTemplate: JobTemplateSpec
schedule: int32
```

```
# ┌──────────────────── minute (0 - 59)
# │ ┌────────────────── hour (0 - 23)
# │ │ ┌──────────────── day of the month (1 - 31)
# │ │ │ ┌────────────── month (1 - 12)
# │ │ │ │ ┌──────────── day of the week (0 - 6) (Sunday to Saturday;
# │ │ │ │ │                                   7 is also Sunday on some systems)
# │ │ │ │ │
# │ │ │ │ │
# * * * * * <command to execute>
```

HorizontalPodAutoscaler

需要部署集群插件，修改apiserver启动项`- --enable-aggregator-routing=true`：

```
[root@controller01 ~]# kubectl autoscale deployment --cpu-percent=50 --min=1 --max=10 -n
sep se-frontend-k8
[root@controller01 ~]# kubectl get hpa --all-namespaces
NAMESPACE   NAME             REFERENCE                   TARGETS   MINPODS   MAXPODS
REPLICAS   AGE
sep         se-frontend-k8   Deployment/se-frontend-k8   0%/60%    2         10        2
23h
```

HorizontalPodAutoscalerSpec

```
maxReplicas: int32
scaleTargetRef:
    kind: string
    name: string
    apiVersion: string
minReplicas: int32
targetCPUUtilizationPercentage: int32
```

Service

```
apiVersion: v1
kind: Service
metadata: ObjectMeta
spec:
  selector: {}
  ports:
  - name: string
    port: int32
    targetPort: IntOrString
    protocol: string
    nodePort: int32
  type: string      # ClusterIP, NodePort, LoadBalancer, ExternalName
```

当不定义selector属性，不会自动创建endpoints资源，这时可以通过手动配置endpoints引入集群外部服务。

```
apiVersion: v1
kind: Endpoints
metadata:
  name: my-service
subsets:
  - addresses:
      - ip: 192.0.2.42
    ports:
      - port: 9376
```

endpoints

```
apiVersion: v1
kind: Endpoints
metadata: ObjectMeta
subsets:
  - addresses:
    - ip: string
      hostname : string
      nodeName: string
      targetRef: ObjectReference
    ports:
      - port: int32
        protocal: string
        name: string
```

Ingress && IngressClass

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata: ObjectMeta
spec:
  defaultBackend:
    service:
      name: string
      port:
        name: string
        number: int32
  ingressClassName: string
  rules:
  - host: string
    http:
      paths:
      - backend: IngressBackend
        path: string
        pathType: string    # Exact, 精确匹配; Prefix, 基于'/'分割的URL前缀;ImplementationSpecific
  tls:
  - hosts
    secretName
```

Kind	Path(s)	Request path(s)	Matches?
Prefix	/	(all paths)	Yes
Exact	/foo	/foo	Yes

Kind	Path(s)	Request path(s)	Matches?
Exact	/foo	/bar	No
Exact	/foo	/foo/	No
Exact	/foo/	/foo	No
Prefix	/foo	/foo, /foo/	Yes
Prefix	/foo/	/foo, /foo/	Yes
Prefix	/aaa/bb	/aaa/bbb	No
Prefix	/aaa/bbb	/aaa/bbb	Yes
Prefix	/aaa/bbb/	/aaa/bbb	Yes, ignores trailing slash
Prefix	/aaa/bbb	/aaa/bbb/	Yes, matches trailing slash
Prefix	/aaa/bbb	/aaa/bbb/ccc	Yes, matches subpath
Prefix	/aaa/bbb	/aaa/bbbxyz	No, does not match string prefix
Prefix	/, /aaa	/aaa/ccc	Yes, matches /aaa prefix
Prefix	/, /aaa, /aaa/bbb	/aaa/bbb	Yes, matches /aaa/bbb prefix
Prefix	/, /aaa, /aaa/bbb	/ccc	Yes, matches / prefix
Prefix	/aaa	/ccc	No, uses default backend

ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata: ObjectMeta
binaryData: map[string][]byte
data: map[string]string
immutable: boolean
```

```
kubectl create configmap NAME [--from-file=[key=]source] [--from-literal=key1=value1] [--
dry-run=server|client|none]
```

你可以使用四种方式来使用 ConfigMap 配置 Pod 中的容器：

1. 在容器命令和参数内
2. 容器的环境变量
3. 在只读卷里面添加一个文件，让应用来读取
4. 编写代码在 Pod 中运行，使用 Kubernetes API 来读取 ConfigMap

configmap实例

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: game-demo
data:
  # 类属性键；每一个键都映射到一个简单的值
  player_initial_lives: "3"
  ui_properties_file_name: "user-interface.properties"

  # 类文件键
  game.properties: |
    enemy.types=aliens,monsters
    player.maximum-lives=5
  user-interface.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
```

configmap的使用

```
apiVersion: v1
kind: Pod
metadata:
  name: configmap-demo-pod
spec:
  containers:
    - name: demo
      image: busybox
      command: ["sleep", "3600"]
      env:
```

```yaml
        # 定义环境变量
        - name: PLAYER_INITIAL_LIVES # 请注意这里和 ConfigMap 中的键名是不一样的
          valueFrom:
            configMapKeyRef:
              name: game-demo           # 这个值来自 ConfigMap
              key: player_initial_lives # 需要取值的键
        - name: UI_PROPERTIES_FILE_NAME
          valueFrom:
            configMapKeyRef:
              name: game-demo
              key: ui_properties_file_name
      volumeMounts:
      - name: config
        mountPath: "/config"
        readOnly: true
  volumes:
    # 你可以在 Pod 级别设置卷，然后将其挂载到 Pod 内的容器中
    - name: config
      configMap:
        # 提供你想要挂载的 ConfigMap 的名字
        name: game-demo
        # 来自 ConfigMap 的一组键，将被创建为文件
        items:
        - key: "game.properties"
          path: "game.properties"
        - key: "user-interface.properties"
          path: "user-interface.properties"
```