CKA-pod

讲师： 老段 RHCE/RHCA/COA/CKA

创建pod

```
apiVersion: v1
kind: Pod
metadata:
 name: static-web
 labels:
  role: myrole
spec:
 containers:
  - name: web
   image: nginx
   ports:
    - name: web
     containerPort: 80
     protocol: TCP
```

kubectl api-versions

需要的镜像
busybox
nginx:1.7.9
nginx:1.9
nginx

yaml文件的获取方法

kubectl run nginx --image=nginx --dry-run -o yaml

镜像的下载策略
Always 每次都下载最新镜像
Never 只使用本地镜像，从不下载
IfNotPresent 本地没有才下载

```
spec:
  containers:
  - image: nginx
    imagePullPolicy: Always
    name: nginx
    resources: {}
```

pod的基本操作

kubectl exec 命令

kubectl exec -it pod sh #如果pod里有多个容器，则命令是在第一个容器里执行

kubectl describe pod pod名

kubectl logs pod名

pod里运行命令及生命周期

另外一种写法

```
apiVersion: v1
kind: Pod
metadata:
 name: myapp-pod
 labels:
  app: myapp
spec:
 containers:
 - name: myapp-container
  image: busybox
  command: ['sh', '-c', 'echo OK!
&& sleep 60']
```

另外一种写法

```
apiVersion: v1
kind: Pod
metadata:
 name: myapp-pod
 labels:
  app: myapp
spec:
 containers:
 - name: myapp-container
  image: busybox
  command:
  - sh
  - -c
  - echo OK! && sleep 60
```

```
apiVersion: v1
kind: Pod
metadata:
 name: myapp-pod
 labels:
  app: myapp
spec:
 containers:
 - name: myapp-container
  image: busybox
  args:
  - sh
  - -c
  - echo OK! && sleep 60
```

在pod中使用变量

```
apiVersion: v1
kind: Pod
metadata:
 name: demo
 labels:
  purpose: demonstrate-envars
spec:
 containers:
 - name: env-demo
  image: nginx
  env:
  - name: DEMOx
   value: "Hello x sir"
  - name: DEMOy
   value: "Hello y sir"
```

```
[root@vms10 xx]# kubectl exec -it demo /bin/bash
root@demo:/# env | grep DEMO
DEMOy=Hello y sir
DEMOx=Hello x sir
root@demo:/#
```

```
apiVersion: v1
kind: Pod
metadata:
 name: demo
 labels:
  purpose: demonstrate-envars
spec:
 containers:
 - name: env-demo
  image: nginx
  env:
  - name: DEMOx
   value: "Hello x sir"
  command: ["/bin/echo"]
  args: ["$(DEMOx)"]
```

```
[root@vms10 xx]# kubectl logs demo
Hello x sir
[root@vms10 xx]#
```

```
apiVersion: v1
kind: Pod
metadata:
 name: myapp-pod
 labels:
  app: myapp
spec:
 containers:
 - name: myapp-container
  image: busybox
  command: ['sh', '-c', 'aa=1; while [ $aa -le 20 ] ; do echo $aa; let aa=$aa+1; sleep 1;  done']
```

pod的重启策略--单个容器正常退出

Always 总是重启
OnFailure 失败了才重启
Never 从不重启

容器正常运行完成之后，不再重启

```
apiVersion: v1
kind: Pod
metadata:
 name: myapp-pod
 labels:
  app: myapp
spec:
 restartPolicy: Never
 containers:
 - name: myapp-container
  image: busybox
  command: ['sh', '-c', 'echo The app is running! && sleep 60']
```

```
[root@vms10 xx]# kubectl get pods
NAME        READY     STATUS      RESTARTS   AGE
myapp-pod   0/1       Completed   0          1m
```

容器的状态

Pending pod已经建立，但是pod里还有容器没有创建完成

Running pod已经被调度到节点上，且容器工作正常

Completed pod里所有容器正常退出

Failed

pod的重启策略--单个容器异常退出

Always 总是重启
OnFailure 失败了才重启
Never 从不重启

容器需要一次completed

```
apiVersion: v1
kind: Pod
metadata:
 name: myapp-pod
 labels:
  app: myapp
spec:
 restartPolicy: OnFailure
 containers:
 - name: myapp-container
  image: busybox
  command: ['sh','-c','echoxxx The app is running! && sleep 5']
```

```
[root@vms10 xx]# kubectl get pods
NAME        READY     STATUS    RESTARTS   AGE
myapp-pod   0/1       Error     4          1m
[root@vms10 xx]#
```

初始化容器

初始化容器的概念
比如一个容器A依赖其他容器，可以为A设置多个依赖容易A1，A2，A3

A1,A2,A3要按照顺序启动，A1没有启动启动起来的话，A2,A3是不会启动的，直到所有的静态容器全部启动完毕，主容器A才会启动。

一般用于A容器运行之前，先做一些准备工作。

如果初始化容器失败，则会一直重启，pod不会创建

```
[root@vms10 xx]# kubectl get pods
NAME        READY     STATUS       RESTARTS   AGE
myapp-pod   0/1       Init:Error   4          1m
```

规则

1、它们总是运行到完成。
2、每个都必须在下一个启动之前成功完成。
3、如果 Pod 的 Init 容器失败，Kubernetes 会不断地重启该 Pod，直到 Init 容器成功为止。然而，如果 Pod 对应的 restartPolicy 为 Never，它不会重新启动。
4、Init 容器支持应用容器的全部字段和特性，但不支持 Readiness Probe，因为它们必须在 Pod 就绪之前运行完成。
5、如果为一个 Pod 指定了多个 Init 容器，那些容器会按顺序一次运行一个。 每个 Init 容器必须运行成功，下一个才能够运行。
6、因为 Init 容器可能会被重启、重试或者重新执行，所以 Init 容器的代码应该是幂等的。 特别地，被写到 EmptyDirs 中文件的代码，应该对输出文件可能已经存在做好准备。
7、在 Pod 上使用 activeDeadlineSeconds，在容器上使用 livenessProbe，这样能够避免 Init 容器一直失败。 这就为 Init 容器活跃设置了一个期限。
8、在 Pod 中的每个 app 和 Init 容器的名称必须唯一；与任何其它容器共享同一个名称，会在验证时抛出错误。
9、对 Init 容器 spec 的修改，被限制在容器 image 字段中。 更改 Init 容器的 image 字段，等价于重启该 Pod。

初始化容器

```
apiVersion: v1
kind: Pod
metadata:
 name: myapp-pod
 labels:
  app: myapp
spec:
 volumes:
 - name: workdir
  emptyDir: {}
 containers:
 - name: myapp-container
  image: nginx
  volumeMounts:
  - name: workdir
   mountPath: "/xx"
 initContainers:
 - name: init-poda
  image: busybox
  command: ['sh', '-c', 'touch /work-dir/aa.txt']
  volumeMounts:
  - name: workdir
   mountPath: "/work-dir"
```

```
[root@vms10 xx]# kubectl get pods
NAME        READY     STATUS     RESTARTS    AGE
myapp-pod   0/1       Init:0/1   0           2s
[root@vms10 xx]#
[root@vms10 xx]# kubectl get pods
NAME        READY     STATUS    RESTARTS    AGE
myapp-pod   1/1       Running   0           5s
[root@vms10 xx]#
```

```
[root@vms10 xx]# kubectl exec -it myapp-pod  /bin/bash
root@myapp-pod:/# ls /xx/
aa.txt
root@myapp-pod:/#
```

静态pod

所谓静态pod就是，不是master上创建的，而是需要到Node的/etc/kubelet.d/里创建一个yaml文件，然后根据这个yaml文件，创建一个pod，这样创建出来的node，是不会接受master的管理的。

当然，要是想创建静态pod的话，需要对node的kubelet配置文件进行一些设置才可以。在指定目录下面创建一个yaml文件，然后改kubelet的systemd配置，reload+重启，检查下

静态pod

在node上

systemctl status kubelet

找到
--pod-manifest-path
所对应的目录

在里面创建一个文件

```
apiVersion: v1
kind: Pod
metadata:
 name: static-web
 labels:
  role: myrole
spec:
 containers:
  - name: web
   image: nginx
   ports:
    - name: web
     containerPort: 80
     protocol: TCP
```

调度的三个对象

- 待调度Pod列表
- 可用node列表
- 调度算法

  主机过滤
  主机打分

- 调度策略

主机过滤

NoDiskConflict
PodFitsResources
PodFitsPorts
MatchNodeSelector
HostName
NoVolumeZoneConflict
PodToleratesNodeTaints
CheckNodeMemoryPressure
CheckNodeDiskPressure
MaxEBSVolumeCount
MaxGCEPDVolumeCount
MaxAzureDiskVolumeCount
MatchInterPodAffinity
GeneralPredicates
NodeVolumeNodeConflict

主机打分

LeastRequestedPriority
公式
score=cpu ((capacity - sum (requested)) * 10 / capacity) + memory
((capacity - sum (requested)) * 10 / capacity) /2

BalanceResourceAllocation
公式
score = 10 -abs (cpuFraction - memoryFraction) * 10

CalculateSpreadPriority
公式
Score = 10 * （ （maxCount -counts） / （maxCount） ）

手动指定pod运行位置

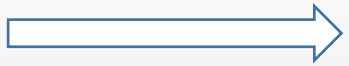
- 为节点指定标签

  kubectl get nodes --show-labels

  kubectl label node node2 disktype=ssd

  kubectl label node node2 disktype-

- 指定pod运行在指定节点



```
spec:
  containers:
  nodeSelector:
    disktype: ssd
```

```
[root@master ~]# kubectl get pods -o wide
NAME                     READY     STATUS              RESTARTS   AGE       IP           NODE
nginx-65899c769f-bcczh   1/1       Running             0          1d        10.244.2.5   node2
nginx-65899c769f-bvhtv   0/1       Terminating         0          1d        10.244.1.5   node1
nginx-6fb4c6cd69-nksr5   0/1       ContainerCreating   0          3s        <none>       node2
nginx-6fb4c6cd69-xnhqz   0/1       ContainerCreating   0          3s        <none>       node2
[root@master ~]#
```

调度：警戒线cordon

kubectl run nginx --image=nginx --replicas=10

kubectl scale --replicas=20 deployment/nginx

kubectl cordon vms11.rhce.cc

kubectl uncordon vms11.rhce.cc

```
[root@vms10 ~]# kubectl get nodes
NAME            STATUS                     ROLES     AGE       VERSION
vms10.rhce.cc   Ready,SchedulingDisabled   master    34d       v1.11.1
vms11.rhce.cc   Ready,SchedulingDisabled   <none>    34d       v1.11.1
vms12.rhce.cc   Ready                      <none>    34d       v1.11.1
[root@vms10 ~]#
```

如果一个node被标记为cordon，新创建的pod不会被调度到此node上

已经调度上去的还在运行，需要被删除让其重新生成

用于节点的维护

调度:节点的drain

kubectl drain vms11.rhce.cc --ignore-daemonsets

kubectl uncordon vms11.rhce.cc

用于节点的维护

如果一个节点被设置为drain，则此节点不再被调度pod，
且此节点上已经运行的pod会被驱逐(evicted)到其他节点

drain包含：

- cordon
- evicted

调度:节点taint及pod的tolerations

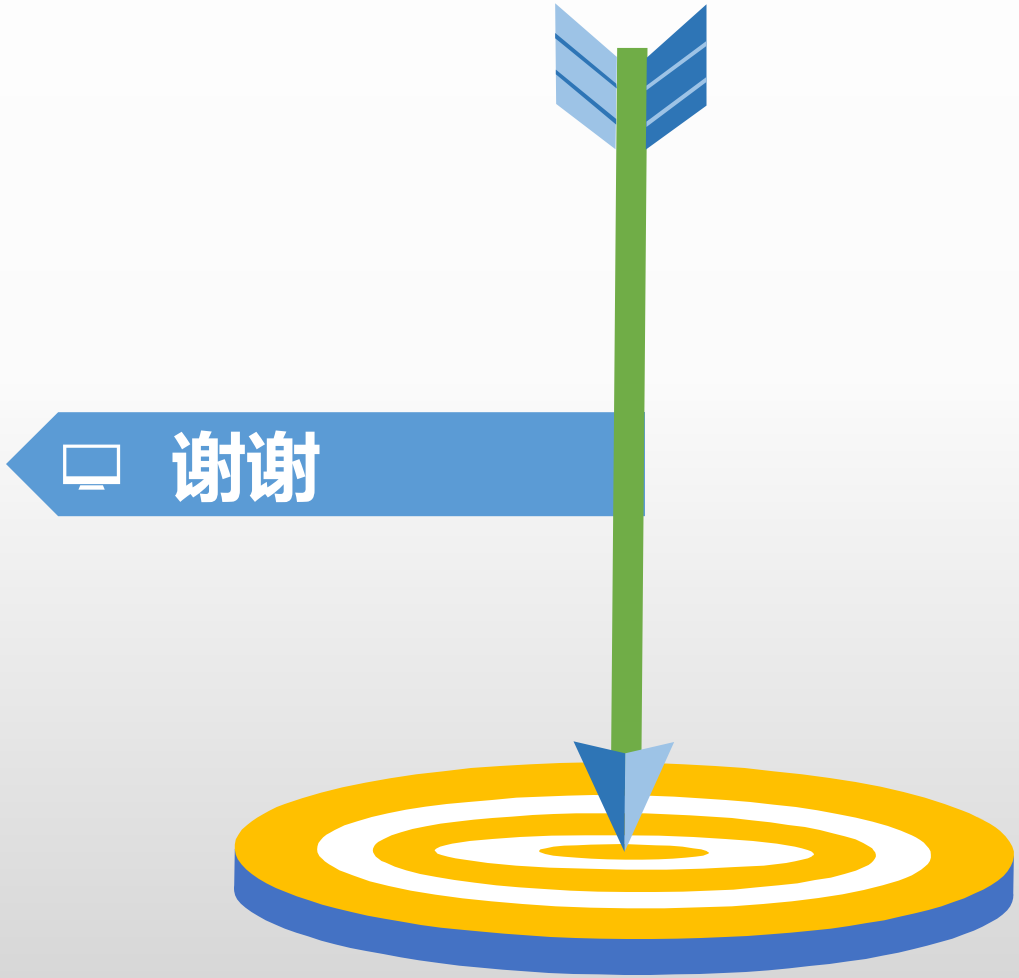
kubectl describe nodes vms11.rhce.cc | grep -A1 Tain

kubectl taint nodes vms11.rhce.cc keyxx=valuexx:NoSchedule

kubectl taint nodes vms11.rhce.cc keyxx:NoSchedule-

只有标签为keyxx=valuexx的pod才能调度到此node，否则不能

```
apiVersion: v1
kind: Pod
metadata:
 name: myapp-pod
 labels:
  app: myapp
spec:
 restartPolicy: OnFailure
 tolerations:
 - key: "keyxx"
  operator: "Equal"
  value: "valuexx"
  effect: "NoSchedule"
 containers:
 - name: myapp-container
  image: nginx
```


谢谢