**加我微信可进群学习交流：**

**微信号：**

luckylucky421302

也可通过扫描下面二维码添加



**课程更新的知识点会通过微信公众号免费分享给大家，可以关注我的公众号**

****

**下文需要的tomcat.tar.gz和busybox.tar.gz在百度网盘地址如下：**

链接：https://pan.baidu.com/s/1RjJiRYDjXK3UaPRIrIKV-A

提取码：387q

**什么是pod？**

**官方说明：**

Pod是Kubernetes应用程序的最基本执行单元—是你创建或部署Kubernetes对象模型中的最小和最简单的单元。 Pod表示在集群上运行的进程。Pod封装了应用程序的容器（或者在某些情况下是多个容器）、存储资源、唯一的网络标识(IP地址)以及控制容器应该如何运行的选项。 Pod表示一个部署单元：Kubernetes中的应用程序的单个实例，该实例可能由单个容器或少量紧密耦合并共享资源的容器组成。Docker是Kubernetes Pod中最常见的容器，但Pods也支持其他容器。Kubernetes集群中的Pod是如何管理容器的：

**1）pod里运行单个容器**： pod里只运行一个容器是最常见的Kubernetes使用案例。在这种情况下，你可以将Pod视为单个容器的封装，Kubernetes直接管理Pod，而不是直接管理容器。

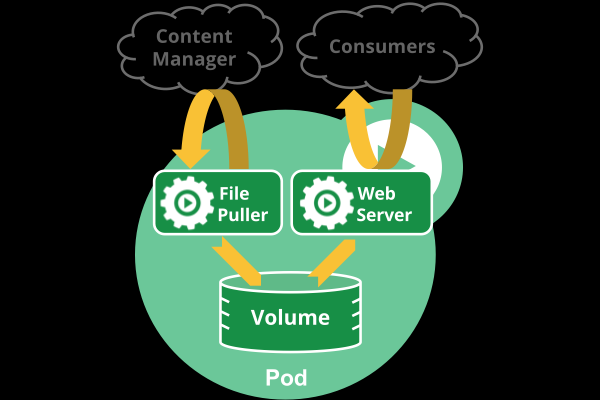
**2）pod里运行多个需要协同工作的容器**：Pod可能封装了一个应用程序，该应用程序由紧密关联并且需要共享资源的多个共同协作的容器组成。这些共同协作的容器可能形成一个统一的服务单元-一个容器将文件从共享卷提供给所有容器使用，而一个单独的“ sidecar”容器则刷新或更新这些文件。Pod将这些容器和存储资源包装在一起，成为一个可管理的实体。

**自己理解：**

pod翻译成中文是豌豆荚的意思，它是kubernetes中的最小调度单元，由一个或者多个容器组成，同一个pod中的这些容器共享存储、网络和命名空间，pod中的容器总是被同时调度，它们有共同的运行环境，运行在同一个共享上下文中，一个pod相当于一个逻辑主机--比方说我们想要部署一个tomcat应用，如果不用容器，我们可能会部署到物理机，虚拟机或者云主机上，那么出现k8s之后，我们就可以把应用部署到pod中，所以pod充当的是一个逻辑主机的角色；pod的共享上下文是一组linux命名空间，cgroup，以及其他可能隔离的方面；Pod中的容器共享IP地址和端口空间，并且可以通过localhost相互访问。他们还可以使用标准的进程间通信（如SystemV信号量或POSIX共享内存）相互通信，不同Pod中的容器具有不同的IP地址，无需[特殊配置](https://kubernetes.io/docs/concepts/policy/pod-security-policy/)即可通过IPC进行通信；在一个Pod中的应用可以访问共享的存储卷，它被认为是Pod的一部分，可以被挂接至每一个应用文件系统；与独立的应用容器一样，Pod是一个临时的实体，它有着自己的生命周期。在Pod被创建时，会被指派一个唯一的ID，并被调度到Node中，直到Pod被终止或删除。如果Pod所在的Node宕机，给定的Pod（即通过UID定义）不会被重新调度。相反，它将被完全相同的Pod所替代。这所说的具有和Pod相关生命周期的情况，例如存储卷，是说和Pod存在的时间一样长。如果Pod被删除，即使完全相同的副本被创建，则相关存储卷等也会被删除，并会为Pod创建一个新的存储卷等。Pod本身就没有打算作为持久化的实体，在调度失败、Node失败和获取其它退出（缺少资源或者Node在维护）情况下，Pod都会被删除。一般来说，用户不应该直接创建Pod，即使创建单个的Pod也应该通过控制器创建。在集群范围内，控制器为Pod提供自愈能力，以及副本和部署管理。

**pod是如何管理多个容器的?**

Pod中可以同时运行多个容器。同一个Pod中的容器会自动的分配到同一个 node 上。同一个Pod中的容器共享资源、网络环境，它们总是被同时调度，在一个Pod中同时运行多个容器是一种比较高级的用法，只有当你的容器需要紧密配合协作的时候才考虑用这种模式。例如，你有一个容器作为web服务器运行，需要用到共享的volume，有另一个“sidecar”容器来从远端获取资源更新这些文件，如下图所示：



一些Pods有init容器和应用容器。 在应用程序容器启动之前，运行初始化容器。Pods为它组成的容器提供两种共享资源：网络和存储。

**网络：**

每个pod都被分配唯一的IP地址，POD中的每个容器共享网络名称空间，包括IP地址和网络端口。 Pod内部的容器可以使用localhost相互通信。 当POD中的容器与POD之外的实体通信时，它们必须使用共享网络资源（如端口）。

**存储：**

Pod可以指定一组共享存储卷。 POD中的所有容器都可以访问共享卷，允许这些容器共享数据。 卷也允许Pod中的持久数据在需要重新启动的情况下存活。 有关Kubernetes如何在POD中实现共享存储的更多信息，可参考<https://kubernetes.io/docs/concepts/storage/volumes/>

**Pod怎么工作？**

我们很少在Kubernetes中直接创建单个Pod。这是因为Pods被设计成相对短暂的、一次性的实体。 当一个POD被创建（直接创建，或间接由控制器创建）时，它被安排在集群中的节点上运行。 在进程终止、pod对象被删除、pod由于缺乏资源而被驱逐或节点失败之前，POD仍然位于该节点上

**注意：**不要将重新启动Pod中的容器与重新启动Pod混淆。POD不是一个进程，而是一个运行容器的环境。Pod一直存在直到被删除为止。

pod本身无法自我修复。如果将Pod调度到发生故障的节点，或者调度操作本身失败，则将Pod删除；同样，由于缺乏资源或Node维护，Pod也被删除。Kubernetes使用称为控制器的更高级别的抽象来处理管理相对一次性的Pod实例的工作。因此，虽然可以直接使用Pod，但在Kubernetes中使用控制器来管理Pod更为常见。

**pod和控制器关系**

你可以使控制器创建和管理多个pod。控制器在pod失败的情况下可以处理副本、更新以及自动修复。例如，如果某个节点发生故障，则控制器会注意到该节点上的Pod已停止工作，并创建了一个替换Pod。调度程序将替换的Pod放置到健康的节点上。可以使用deployment、statefulset、daemonset等控制器管理pod。

**使用pod**

Pod 可以用于托管垂直集成的应用程序栈（例如，LAMP），但最主要的目的是支持位于同一位置的、共同管理的程序，例如：

1.内容管理系统、文件和数据加载器、本地缓存管理器等。

2.日志和检查点备份、压缩、旋转、快照等。

3.数据更改监视器、日志跟踪器、日志和监视适配器、事件发布器等。

4.代理、桥接器和适配器

5.控制器、管理器、配置器和更新器

通常，不会用单个 Pod 来运行同一应用程序的多个实例。

**pod模板\*\*\*（重点）**

控制器（如deployment、daemonset、statefulset等）是通过创建pod模板来创建和管理pod的，PodTemplate是用于创建pod的规范，并且包含在deployment、job和daemonset中。每个控制器使用自己内部的Pod模板来创建实际的Pod。PodTemplate是运行应用程序所需的任何控制器的一部分。下面的示例是一个简单的Job的清单，包含一个podtemplate，这个是用来生成pod的模板。该Pod中的容器会打印一条消息，然后暂停。

cat job-template.yaml

apiVersion: batch/v1

kind: Job

metadata:

name: hello

spec:

template:

# This is the pod template

spec:

containers:

- name: hello

image: busybox

command: ['sh', '-c', 'echo "Hello, Kubernetes!" && sleep 3600']

restartPolicy: OnFailure

# The pod template ends here

修改pod template或切换到新的pod tmplate对已经存在的pod没有影响。 POD不直接接收模板更新,而是创建一个新的POD来匹配修改后的POD模板。例如，控制器可确保正在运行的Pod与当前Pod模板匹配。如果模板已更新，则控制器必须删除现有的Pod并根据更新的模板创建新的Pod。每个控制器都实现自己的规则来处理Pod模板的更改。在节点上，[kubelet](https://kubernetes.io/docs/reference/generated/kubelet)不直接观察或管理有关Pod模板和更新的任何详细信息。

**和pod相关的api对象**

kubectl explain pods

上面命令可以看到和pod相关的api对象有哪些，也就是通过资源清单yaml部署一个pod时需要哪些字段

**apiVersion**

apiVersion定义了此对象表示的版本化模式。服务器应将已识别的模式转换为最新的内部值，并可能拒绝无法识别的值。更多信息参考：

<https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources>

查看k8s集群支持的apiVersion有哪些，可以使用下面的命令：

**kubectl api-versions**

显示如下支持apiVersion信息：

admissionregistration.k8s.io/v1

admissionregistration.k8s.io/v1beta1

apiextensions.k8s.io/v1

apiextensions.k8s.io/v1beta1

apiregistration.k8s.io/v1

apiregistration.k8s.io/v1beta1

apps/v1

authentication.k8s.io/v1

authentication.k8s.io/v1beta1

authorization.k8s.io/v1

authorization.k8s.io/v1beta1

autoscaling/v1

autoscaling/v2beta1

autoscaling/v2beta2

batch/v1

batch/v1beta1

certificates.k8s.io/v1beta1

coordination.k8s.io/v1

coordination.k8s.io/v1beta1

crd.projectcalico.org/v1

discovery.k8s.io/v1beta1

events.k8s.io/v1beta1

extensions/v1beta1

metrics.k8s.io/v1beta1

networking.k8s.io/v1

networking.k8s.io/v1beta1

node.k8s.io/v1beta1

policy/v1beta1

rbac.authorization.k8s.io/v1

rbac.authorization.k8s.io/v1beta1

scheduling.k8s.io/v1

scheduling.k8s.io/v1beta1

storage.k8s.io/v1

storage.k8s.io/v1beta1

v1

**kind**

Kind是表示此对象表示的REST资源的字符串值。服务器可以从客户端提交请求的端点推断出这一点，说白了就是表示我们要创建什么资源，如deployment、statefulset、pod、service、ingress

查看更详细信息可参考：

https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

**metadata**

标准对象的元数据。更多信息：

<https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata>

**spec**

指定容器的所需行为。更多信息：

https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status

**status**

最近观察到的pod的状态。此数据可能不是最新的。Status不需要在pod或者其他资源中定义，这个默认是存在的，更多信息：https：//git.k8s.io/community/contributors/devel/api-conventions.md#spec-and-status

**怎么创建pod？**

**1.通过定义资源清单yaml文件（就是以yaml结尾的文件）创建pod，在k8s的master节点操作**

查看定义资源清单需要哪些字段

kubectl explain pods

kubectl explain pods.apiVersion

kubectl explain pods.kind

kubectl explain pods.metadata

kubectl explain pods.spec

**cat pod.yaml**

apiVersion: v1

kind: Pod

metadata:

name: web

namespace: default

labels:

web1: tomcat

spec:

containers:

- name: tomcat1

image: tomcat:8.5-jre8-alpine

imagePullPolicy: IfNotPresent

#通过kubectl apply创建一个pod

kubectl apply -f pod.yaml

#查看pod创建的情况

kubectl get pods

显示如下：

NAME READY STATUS RESTARTS AGE

web 0/1 ContainerCreating 0 37s

#查看pod的详细信息

kubectl describe pods web

# pod.yaml定义的所有资源都删除掉

kubectl delete -f pod.yaml yaml

#查看pod调度到哪个节点

kubectl get pods -o wide

显示如下：

NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

web 1/1 Running 0 5m30s 10.244.1.21 node1 <none> <none>

#查看pod日志

kubectl logs web

#查看pod里指定容器的日志

kubectl logs -c tomcat1 web

#进入到刚才创建的pod，刚才创建的pod名字是web

kubectl exec -it web -- /bin/bash

#假如pod里有多个容器，进入到pod里的指定容器，按如下命令：

kubectl exec -it web -c tomcat1 -- /bin/bash

我们上面创建的pod是一个自主式pod，也就是通过pod创建一个应用程序，如果pod出现故障停掉，那么我们通过pod部署的应用也就会停掉，不安全，所以还有一种pod，

还有一种控制器管理的pod，通过控制器创建pod，可以对pod的生命周期做管理，如定义pod的副本数，如果有一个pod意外停掉，那么会自动起来一个pod替代之前的pod，之后会讲解pod的控制器

kubectl get pods

可查看到刚才创建的pod

**pod的持久性**

一般来说，用户不需要直接创建 Pod。他们几乎都是使用控制器进行创建，即使对于单例的 Pod 创建也一样使用控制器，例如[Deployments](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/)控制器提供集群范围的自修复以及副本数和滚动管理。 像[StatefulSet](https://kubernetes.io/docs/concepts/workloads/controllers/statefulset.md)这样的控制器还可以提供支持有状态的Pod。

**Pod生命周期**

同一个pod中可以运行多个容器，我们在创建一个pod时可以通过创建多个容器来实现pod的整个生命周期，一个pod的创建包含如下过程：

**Init容器（也叫做初始化容器）：**

Init容器就是做初始化工作的容器。可以有一个或多个，如果多个按照定义的顺序依次执行，只有所有的初始化容器执行完后，主容器才启动。由于一个Pod里的存储卷是共享的，所以Init Container里产生的数据可以被主容器使用到，Init Container可以在多种K8S资源里被使用到，如Deployment、DaemonSet, StatefulSet、Job等，但都是在Pod启动时，在主容器启动前执行，做初始化工作。

**主容器：**

**容器钩子：**

**对于pod资源来说，容器钩子是在pods.spec.containers.lifecycle下定义的**

初始化容器启动之后，开始启动主容器，在主容器启动之前有一个post start hook（容器启动后钩子）和pre stop hook（容器结束前钩子）

**postStart**：该钩子在容器被创建后立刻触发，通知容器它已经被创建。如果该钩子对应的hook handler执行失败，则该容器会被杀死，并根据该容器的重启策略决定是否要重启该容器，这个钩子不需要传递任何参数

**preStop**：该钩子在容器被删除前触发，其所对应的hook handler必须在删除该容器的请求发送给Docker daemon之前完成。在该钩子对应的hook handler完成后不论执行的结果如何，Docker daemon会发送一个SGTERN信号量给Docker daemon来删除该容器，这个钩子不需要传递任何参数

#查看postStart怎么定义的，可以用如下命令：

kubectl explain pods.spec.containers.lifecycle.postStart

#查看preStop怎么定义的，可以用如下命令：

kubectl explain pods.spec.containers.lifecycle.preStop

**容器探针：**

livenessProbe：

指示容器是否正在运行。如果存活探测失败，则 kubelet 会杀死容器，并且容器将受到其[重启策略](https://k8smeetup.github.io/docs/concepts/workloads/pods/pod-lifecycle/" \l "restart-policy)的影响。如果容器不提供存活探针，则默认状态为Success。

readinessProbe：指示容器是否准备好服务请求。如果就绪探测失败，端点控制器将从与 Pod 匹配的所有Service 的端点中删除该 Pod 的 IP 地址。初始延迟之前的就绪状态默认为Failure。如果容器不提供就绪探针，则默认状态为Success

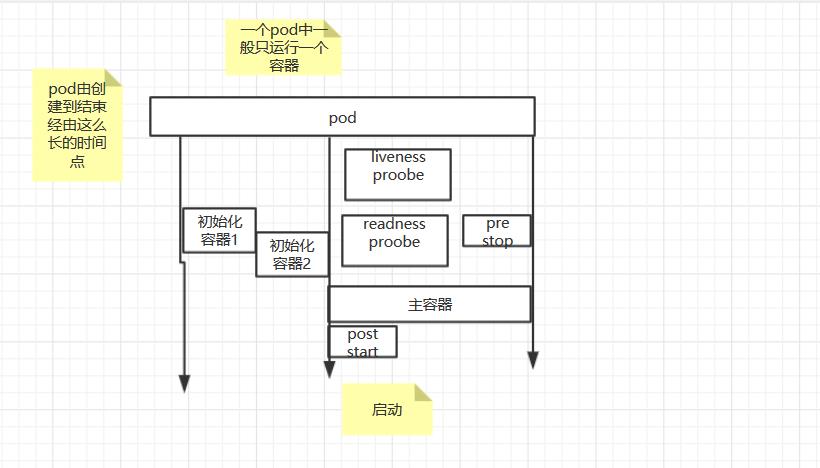
#查看livenessProbe帮助命令：

kubectl explain pods.spec.containers.livenessProbe

#查看readinessProbe帮助命令：

kubectl explain pods.spec.containers.readinessProbe

整个图如下：



从上面可以看出，我们pod在从创建到结束之前，会一直处于某种状态之中，有一些状态：

**常见的pod状态**

（1）Pending：挂起，我们在请求创建pod时，条件不满足，调度没有完成，没有任何一个节点能满足调度条件。已经创建了但是没有适合它运行的节点叫做挂起，调度没有完成。

（2）Running：运行状态

（3）Failed：表示失败

（4）Succeeded：表示成功状态

（5）Unknown：未知状态，所谓pod是什么状态是apiserver和运行在pod节点的kubelet进行通信获取状态信息的，如果节点之上的kubelet本身出故障，那么apiserver就连不上kubelet，得不到信息了，就会看Unknown

**创建pod时大概经历哪些阶段：**

初始化容器-->主容器

**pod重启策略**

Always：只要容器挂了就重启

OnFailure：只有容器状态为错误的时候才重启

Never：从不重启容器

默认重启策略就是Always

#查看重启策略restartPolicy怎么定义的：

kubectl explain pods.spec.restartPolicy

**名称空间-namespace**

namespace叫做命名空间，可以把k8s集群划分成多个名称空间，然后对不同的名称空间的资源做隔离，可以控制各个名称空间的入栈，出栈策略，是一种在多个用户之间划分群集资源的方法

#查看k8s集群当前有哪些名称空间：

kubectl get namespace

**pod label**

（1）什么是标签？

标签其实就一对 key/value ，被关联到对象上，比如Pod,标签的使用我们倾向于能够标示对象的特殊特点，并且对用户而言是有意义的（就是一眼就看出了这个Pod是干什么的），标签可以用来划分特定组的对象（比如版本，服务类型等），标签可以在创建一个对象的时候直接给与，也可以在后期随时修改，每一个对象可以拥有多个标签，但是，key值必须是唯一的

（2）查看所有pod资源对象的标签

kubectl get pods --show-labels

（3）查看带有指定标签的pod

kubectl get pods -L web1

显示所有资源对象下web1这个标签的标签值

（4）kubectl get pods -l web1 --show-labels

查看拥有web1这个标签的资源对象，并且把标签显示出来

（5）想修改资源的标签，比方说想给web加上个release标签

给资源对象打标签要使用label命令，指定给某个类型下的某个资源打标签，资源中的key/value可以是多个，因此在web（pod名字）这个资源下再打个标签release，用如下命令

kubectl label pods web release=new

查看标签是否打成功：

kubectl get pods --show-labels

显示如下,显示如下，说明标签达成功了；

NAME READY STATUS RESTARTS AGE LABELS

web 1/1 Running 1 21h release=new,web1=tomcat

（6）k8s的标签选择器：

与name和UID不同，label不提供唯一性。通常，我们会看到很多对象有着一样的label。通过标签选择器，客户端/用户能方便辨识出一组对象。

API目前支持两种标签选择器：基于等值的和基于集合的标签选择器。一个label选择器可以由多个必须条件组成，由逗号分隔。在多个必须条件指定的情况下，所有的条件都必须满足，因而逗号起着AND逻辑运算符的作用。

一个空的label选择器（即有0个必须条件的选择器）会选择集合中的每一个对象。

一个null型label选择器（仅对于可选的选择器字段才可能）不会返回任何对象。

**基于等值关系的标签选择器：**

=，==，!=

基于相等性或者不相等性的条件允许用label的键或者值进行过滤。匹配的对象必须满足所有指定的label约束，尽管他们可能也有额外的label。有三种运算符是允许的，“=”，“==”和“!=”。前两种代表相等性（他们是同义运算符），后一种代表非相等性。例如：

environment = production tier != frontend

第一个选择所有键等于 environment 值为 production 的资源。后一种选择所有键为 tier 值不等于 frontend 的资源，和那些没有键为 tier 的label的资源。

要过滤所有处于 production 但不是 frontend 的资源，可以使用逗号操作符， environment=production,tier!=frontend 。

**第二类叫做基于集合的标签选择器：**

基于集合的label条件允许用一组值来过滤键。支持三种操作符: in ， notin ,和 exists(仅针对于key符号) 。例如：

environment in (production, qa)

tier notin (frontend, backend)

第一个例子，选择所有键等于 environment ，且value等于 production 或者 qa 的资源。 第二个例子，选择所有键等于tier且值是除了frontend 和 backend 之外的资源，和那些没有label的键是 tier 的资源。 类似的，逗号操作符相当于一个AND操作符。因而要使用一个 partition 键（不管值是什么），并且 environment 不是 qa 过滤资源可以用 partition,environment notin (qa) 。

基于集合的选择器是一个相等性的宽泛的形式，因为 environment=production 相当于environment in (production) ，与 != and notin 类似。

基于集合的条件可以与基于相等性 的条件混合。例如， partition in (customerA,customerB),environment != qa 。

标签选择器根据定义的标签可以选择匹配到的资源对象

**更多详细信息可参考：**

<https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/>

**node label**

**（1）查看nodes节点的标签**

kubectl get nodes --show-labels

**（2）给node节点打标签：**

kubectl label nodes node1 node011=haha

kubectl get nodes --show-labels

可以看到node01上有node011这个标签了

**（3）节点选择器nodeSelector**

**#查看nodeSelector帮助命令**

kubectl explain pods.spec.nodeSelector

nodeSelector <map[string]string>

NodeSelector is a selector which must be true for the pod to fit on a node.

Selector which must match a node's labels for the pod to be scheduled on

that node. More info:

https://kubernetes.io/docs/concepts/configuration/assign-pod-node/

上面有一个nodeSelector，这个是节点标签选择器，可以限制pod运行在哪个节点上

kubectl get pods -o wide

从上面可以看到web运行在node1上，如果我们想要让它运行在master1上，就需要用到节点选择器

nodeSelector:

node011: haha

#这个node011是我们给node1节点打的标签，在上面已经操作过

cat pod.yaml 看到完整的文件如下：

apiVersion: v1

kind: Pod

metadata:

name: web

namespace: default

labels:

web1: tomcat

spec:

containers:

- name: tomcat1

image: tomcat:8.5-jre8-alpine

nodeSelector：

node011：haha

kubectl delete -f pod.yaml

kubectl apply -f pod.yaml

kubectl get pods -o wide

显示pod运行在node1上

如果node1和node2都有node011这个标签，那么nodeSelector则根据调度策略调度pod到相应的node节点上。

**更多详细信息可参考：**

<https://kubernetes.io/docs/concepts/configuration/assign-pod-node/>

**节点名称**

nodeName：指定pod节点运行在哪个具体node上，不存在调度说法

查看 nodeName帮助命令：

kubectl explain pods.spec.nodeName