新手必读，16 个概念入门 Kubernetes

Kubernetes 是 Google 开源的容器集群管理系统，是 Google 多年大规模容器管理技术 Borg 的开源版本，主要功能包括：

- 基于容器的应用部署、维护和滚动升级
- 负载均衡和服务发现
- 跨机器和跨地区的集群调度
- 自动伸缩
- 无状态服务和有状态服务
- 广泛的 Volume 支持
- 插件机制保证扩展性

Kubernetes 发展非常迅速，已经成为容器编排领域的领导者，接下来我们将讲解 Kubernetes 中涉及到的一些主要概念。

1、Pod

Pod 是一组紧密关联的容器集合，支持多个容器在一个 Pod 中共享网络和文件系统，可以通过进程间通信和文件共享这种简单高效的方式完成服务，是 Kubernetes 调度的基本单位。Pod 的设计理念是每个 Pod 都有一个唯一的 IP。Pod 具有如下特征：

- 包含多个共享 IPC、Network 和 UTC namespace 的容器，可直接通过 localhost 通信
- 所有 Pod 内容器都可以访问共享的 Volume，可以访问共享数据
- 优雅终止:Pod 删除的时候先给其内的进程发送 SIGTERM，等待一段时间(grace period)后才强制停止依然还在运行的进程
- 特权容器(通过 SecurityContext 配置)具有改变系统配置的权限(在网络插件中大量应用)
- 支持三种重启策略（restartPolicy），分别是：Always、OnFailure、Never
- 支持三种镜像拉取策略（imagePullPolicy），分别是：Always、Never、IfNotPresent
- 资源限制，Kubernetes 通过 CGroup 限制容器的 CPU 以及内存等资源，可以设置 request 以及 limit 值
- 健康检查，提供两种健康检查探针，分别是 livenessProbe 和 redinessProbe，前者用于探测容器是否存活，如果探测失败，则根据重启策略进行重启操作，后者用于检查容器状态是否正常，如果检查容器状态不正常，则请求不会到达该 Pod
- Init container 在所有容器运行之前执行，常用来初始化配置
- 容器生命周期钩子函数，用于监听容器生命周期的特定事件，并在事件发生时执行已注册的回调函数，支持两种钩子函数：postStart 和 preStop，前者是在容器启动后执行，后者是在容器停止前执行

2、Namespace

Namespace（命名空间）是对一组资源和对象的抽象集合，比如可以用来将系统内部的对象划分为不同的项目组或者用户组。常见的 pod、service、replicaSet

和 deployment 等都是属于某一个 namespace 的(默认是 default)，而 node, persistentVolumes 等则不属于任何 namespace。常用 namespace 操作：

- kubectlgetnamespace, 查询所有 namespace
- kubectl createnamespacens-name，创建 namespace
- kubectldeletenamespacens-name, 删除 namespace

删除命名空间时，需注意以下几点：

1. 删除一个 namespace 会自动删除所有属于该 namespace 的资源。
2. default 和 kube-system 命名空间不可删除。
3. PersistentVolumes 是不属于任何 namespace 的，但 PersistentVolumeClaim 是属于某个特定 namespace 的。
4. Events 是否属于 namespace 取决于产生 events 的对象。

3、Node

Node 是 Pod 真正运行的主机，可以是物理机也可以是虚拟机。Node 本质上不是 Kubernetes 来创建的， Kubernetes 只是管理 Node 上的资源。为了管理 Pod，每个 Node 节点上至少需要运行 container runtime（Docker）、kubelet 和 kube-proxy 服务。常用 node 操作：

- kubectlgetnodes，查询所有 node
- kubectl cordon $nodename, 将 node 标志为不可调度
- kubectl uncordon $nodename, 将 node 标志为可调度

taint(污点)使用 kubectl taint 命令可以给某个 Node 节点设置污点，Node 被设置上污点之后就和 Pod 之间存在了一种相斥的关系，可以让 Node 拒绝 Pod 的调度执行，甚至将 Node 已经存在的 Pod 驱逐出去。每个污点的组成：key=value:effect，当前 taint effect 支持如下三个选项：

- NoSchedule：表示 k8s 将不会将 Pod 调度到具有该污点的 Node 上
- PreferNoSchedule：表示 k8s 将尽量避免将 Pod 调度到具有该污点的 Node 上
- NoExecute：表示 k8s 将不会将 Pod 调度到具有该污点的 Node 上，同时会将 Node 上已经存在的 Pod 驱逐出去

常用命令如下：

- kubectl taint node node0 key1=value1:NoShedule，为 node0 设置不可调度污点
- kubectl taint node node0 key-，将 node0 上 key 值为 key1 的污点移除
- kubectl taint node node1 node-role.kubernetes.io/master=:NoSchedule，为 kube-master 节点设置不可调度污点
- kubectl taint node node1 node-role.kubernetes.io/master=PreferNoSchedule ， 为 kube-master 节点设置尽量不可调度污点

容忍(Tolerations)设置了污点的 Node 将根据 taint 的 effect：NoSchedule、PreferNoSchedule、NoExecute 和 Pod 之间产生互斥的关系，Pod 将在一定程度上不会被调度到 Node 上。 但我们可以在 Pod 上设置容忍(Toleration)，意思是设置了容忍的 Pod 将可以容忍污点的存在，可以被调度到存在污点的 Node 上。

4、Service

Service 是对一组提供相同功能的 Pods 的抽象，并为他们提供一个统一的入口，借助 Service 应用可以方便的实现服务发现与负载均衡，并实现应用的零宕机升级。Service 通过标签(label)来选取后端 Pod，一般配合 ReplicaSet 或者 Deployment 来保证后端容器的正常运行。service 有如下四种类型，默认是 ClusterIP：

- ClusterIP: 默认类型，自动分配一个仅集群内部可以访问的虚拟 IP
- NodePort: 在 ClusterIP 基础上为 Service 在每台机器上绑定一个端口，这样就可以通过 NodeIP:NodePort 来访问该服务
- LoadBalancer: 在 NodePort 的基础上，借助 cloud provider 创建一个外部的负载均衡器，并将请求转发到 NodeIP:NodePort
- ExternalName: 将服务通过 DNS CNAME 记录方式转发到指定的域名

另外，也可以将已有的服务以 Service 的形式加入到 Kubernetes 集群中来，只需要在创建 Service 的时候不指定 Label selector，而是在 Service 创建好后手动为其添加 endpoint。

5、Volume 存储卷

默认情况下容器的数据是非持久化的，容器消亡以后数据也会跟着丢失，所以 Docker 提供了 Volume 机制以便将数据持久化存储。Kubernetes 提供了更强大的 Volume 机制和插件，解决了容器数据持久化以及容器间共享数据的问题。Kubernetes 存储卷的生命周期与 Pod 绑定

- 容器挂掉后 Kubelet 再次重启容器时，Volume 的数据依然还在
- Pod 删除时，Volume 才会清理。数据是否丢失取决于具体的 Volume 类型，比如 emptyDir 的数据会丢失，而 PV 的数据则不会丢

目前 Kubernetes 主要支持以下 Volume 类型：

- emptyDir：Pod 存在，emptyDir 就会存在，容器挂掉不会引起 emptyDir 目录下的数据丢失，但是 pod 被删除或者迁移，emptyDir 也会被删除
- hostPath：hostPath 允许挂载 Node 上的文件系统到 Pod 里面去
- NFS（Network File System）：网络文件系统，Kubernetes 中通过简单地配置就可以挂载 NFS 到 Pod 中，而 NFS 中的数据是可以永久保存的，同时 NFS 支持同时写操作。
- glusterfs：同 NFS 一样是一种网络文件系统，Kubernetes 可以将 glusterfs 挂载到 Pod 中，并进行永久保存
- cephfs：一种分布式网络文件系统，可以挂载到 Pod 中，并进行永久保存
- subpath：Pod 的多个容器使用同一个 Volume 时，会经常用到

- secret：密钥管理，可以将敏感信息进行加密之后保存并挂载到 Pod 中
- persistentVolumeClaim：用于将持久化存储（PersistentVolume）挂载到 Pod 中
- ...

6、PersistentVolume(PV) 持久化存储卷

PersistentVolume(PV)是集群之中的一块网络存储。跟 Node 一样，也是集群的资源。PersistentVolume (PV)和 PersistentVolumeClaim (PVC)提供了方便的持久化卷：PV 提供网络存储资源，而 PVC 请求存储资源并将其挂载到 Pod 中。PV 的访问模式(accessModes)有三种：

- ReadWriteOnce(RWO):是最基本的方式，可读可写，但只支持被单个 Pod 挂载。
- ReadOnlyMany(ROX):可以以只读的方式被多个 Pod 挂载。
- ReadWriteMany(RWX):这种存储可以以读写的方式被多个 Pod 共享。

不是每一种存储都支持这三种方式，像共享方式，目前支持的还比较少，比较常用的是 NFS。在 PVC 绑定 PV 时通常根据两个条件来绑定，一个是存储的大小，另一个就是 访问模式。PV 的回收策略(persistentVolumeReclaimPolicy)也有三种

- Retain，不清理保留 Volume(需要手动清理)
- Recycle，删除数据，即 rm -rf /thevolume/* (只有 NFS 和 HostPath 支持)
- Delete，删除存储资源

7、Deployment 无状态应用

一般情况下我们不需要手动创建 Pod 实例，而是采用更高一层的抽象或定义来管理 Pod，针对无状态类型的应用，Kubernetes 使用 Deloyment 的 Controller 对象与之对应。其典型的应用场景包括：

- 定义 Deployment 来创建 Pod 和 ReplicaSet
- 滚动升级和回滚应用
- 扩容和缩容
- 暂停和继续 Deployment

常用的操作命令如下：

- kubectl run www – image=10.0.0.183:5000/hanker/www:0.0.1 – port=8080 生成一个 Deployment 对象
- kubectlgetdeployment–all-namespaces 查找 Deployment
- kubectl describe deployment www 查看某个 Deployment
- kubectl edit deployment www 编辑 Deployment 定义
- kubectldeletedeployment www 删除某 Deployment
- kubectl scale deployment/www–replicas=2 扩缩容操作，即修改 Deployment 下的 Pod 实例个数

- kubectlsetimage deployment/nginx-deployment nginx=nginx:1.9.1 更新镜像
- kubectl rollout undo deployment/nginx-deployment 回滚操作
- kubectl rollout status deployment/nginx-deployment 查看回滚进度
- kubectl autoscale deployment nginx-deployment–min=10–max=15–cpu-percent=80 启用水平伸缩（HPA – horizontal pod autoscaling），设置最小、最大实例数量以及目标 cpu 使用率
- kubectl rollout pause deployment/nginx-deployment 暂停更新 Deployment
- kubectl rollout resume deploy nginx 恢复更新 Deployment

更新策略.spec.strategy 指新的 Pod 替换旧的 Pod 的策略，有以下两种类型

- RollingUpdate 滚动升级，可以保证应用在升级期间，对外正常提供服务。
- Recreate 重建策略，在创建出新的 Pod 之前会先杀掉所有已存在的 Pod。

Deployment 和 ReplicaSet 两者之间的关系

- 使用 Deployment 来创建 ReplicaSet。ReplicaSet 在后台创建 pod，检查启动状态，看它是成功还是失败。
- 当执行更新操作时，会创建一个新的 ReplicaSet，Deployment 会按照控制的速率将 pod 从旧的 ReplicaSet 移 动到新的 ReplicaSet 中

8、StatefulSet 有状态应用

Deployments 和 ReplicaSets 是为无状态服务设计的，那么 StatefulSet 则是为了有状态服务而设计，其应用场景包括：

- 稳定的持久化存储，即 Pod 重新调度后还是能访问到相同的持久化数据，基于 PVC 来实现
- 稳定的网络标志,即 Pod 重新调度后其 PodName 和 HostName 不变,基于 Headless Service(即没有 Cluster IP 的 Service)来实现
- 有序部署，有序扩展，即 Pod 是有顺序的，在部署或者扩展的时候要依据定义的顺序依次进行操作(即从 0 到 N-1，在下一个 Pod 运行之前所有之前的 Pod 必须都是 Running 和 Ready 状态)，基于 init containers 来实现
- 有序收缩，有序删除(即从 N-1 到 0)

支持两种更新策略：

- OnDelete:当 .spec.template 更新时，并不立即删除旧的 Pod，而是等待用户手动删除这些旧 Pod 后自动创建新 Pod。这是默认的更新策略，兼容 v1.6 版本的行为
- RollingUpdate:当 .spec.template 更新时，自动删除旧的 Pod 并创建新 Pod 替换。在更新时这些 Pod 是按逆序的方式进行，依次删除、创建并等待 Pod 变成 Ready 状态才进行下一个 Pod 的更新。

9、DaemonSet 守护进程集

DaemonSet 保证在特定或所有 Node 节点上都运行一个 Pod 实例，常用来部署一些集群的日志采集、监控或者其他系统管理应用。典型的应用包括:

- 日志收集，比如 fluentd，logstash 等
- 系统监控，比如 Prometheus Node Exporter，collectd 等
- 系统程序，比如 kube-proxy, kube-dns, glusterd, ceph，ingress-controller 等

指定 Node 节点 DaemonSet 会忽略 Node 的 unschedulable 状态，有两种方式来指定 Pod 只运行在指定的 Node 节点上：

- nodeSelector:只调度到匹配指定 label 的 Node 上
- nodeAffinity:功能更丰富的 Node 选择器，比如支持集合操作
- podAffinity:调度到满足条件的 Pod 所在的 Node 上

目前支持两种策略

- OnDelete: 默认策略，更新模板后，只有手动删除了旧的 Pod 后才会创建新的 Pod
- RollingUpdate: 更新 DaemonSet 模版后，自动删除旧的 Pod 并创建新的 Pod

10、Ingress

Kubernetes 中的负载均衡我们主要用到了以下两种机制：

- Service：使用 Service 提供集群内部的负载均衡，Kube-proxy 负责将 service 请求负载均衡到后端的 Pod 中
- Ingress Controller：使用 Ingress 提供集群外部的负载均衡

Service 和 Pod 的 IP 仅可在集群内部访问。集群外部的请求需要通过负载均衡转发到 service 所在节点暴露的端口上，然后再由 kube-proxy 通过边缘路由器将其转发到相关的 Pod，Ingress 可以给 service 提供集群外部访问的 URL、负载均衡、HTTP 路由等，为了配置这些 Ingress 规则，集群管理员需要部署一个 Ingress Controller，它监听 Ingress 和 service 的变化，并根据规则配置负载均衡并提供访问入口。常用的 ingress controller：

- nginx
- traefik
- Kong
- Openresty

11、Job & CronJob 任务和定时任务

Job 负责批量处理短暂的一次性任务 (short lived>CronJob 即定时任务，就类似于 Linux 系统的 crontab，在指定的时间周期运行指定的任务。

12、HPA（Horizontal Pod Autoscaling） 水平伸缩

Horizontal Pod Autoscaling 可以根据 CPU、内存使用率或应用自定义 metrics 自动扩展 Pod 数量（支持 replication controller、deployment 和 replica set)。

- 控制管理器默认每隔 30s 查询 metrics 的资源使用情况(可以通过 –horizontal-pod-autoscaler-sync-period 修改)

- 支持三种 metrics 类型
    - 预定义 metrics(比如 Pod 的 CPU)以利用率的方式计算
    - 自定义的 Pod metrics，以原始值(raw value)的方式计算
    - 自定义的 object metrics
- 支持两种 metrics 查询方式:Heapster 和自定义的 REST API
- 支持多 metrics

可以通过如下命令创建 HPA：kubectl autoscale deployment php-apache –cpu-percent=50 –min=1 –max=10

13、Service Account

Service account 是为了方便 Pod 里面的进程调用 Kubernetes API 或其他外部服务而设计的**授权** Service Account 为服务提供了一种方便的认证机制，但它不关心授权的问题。可以配合 RBAC(Role Based Access Control)来为 Service Account 鉴权，通过定义 Role、RoleBinding、ClusterRole、ClusterRoleBinding 来对 sa 进行授权。

14、Secret 密钥

Sercert-密钥解决了密码、token、密钥等敏感数据的配置问题，而不需要把这些敏感数据暴露到镜像或者 Pod Spec 中。Secret 可以以 Volume 或者环境变量的方式使用。有如下三种类型：

- Service Account:用来访问 Kubernetes API，由 Kubernetes 自动创建，并且会自动挂载到 Pod 的 /run/secrets/kubernetes.io/serviceaccount 目录中;
- Opaque:base64 编码格式的 Secret，用来存储密码、密钥等;
- kubernetes.io/dockerconfigjson: 用来存储私有 docker registry 的认证信息。

15、ConfigMap 配置中心

ConfigMap 用于保存配置数据的键值对，可以用来保存单个属性，也可以用来保存配置文件。ConfigMap 跟 secret 很类似，但它可以更方便地处理不包含敏感信息的字符串。ConfigMap 可以通过三种方式在 Pod 中使用，三种分别方式为:设置环境变量、设置容器命令行参数以及在 Volume 中直接挂载文件或目录。可以使用 kubectl create configmap 从文件、目录或者 key-value 字符串创建等创建 ConfigMap。也可以通过 kubectl create-f value.yaml 创建。

16、Resource Quotas 资源配额

资源配额(Resource Quotas)是用来限制用户资源用量的一种机制。资源配额有如下类型：

- 计算资源，包括 cpu 和 memory
    - cpu, limits.cpu, requests.cpu
    - memory, limits.memory, requests.memory

- 存储资源，包括存储资源的总量以及指定 storage class 的总量
    - requests.storage:存储资源总量，如 500Gi
    - persistentvolumeclaims:pvc 的个数
    - storageclass.storage.k8s.io/requests.storage
    - storageclass.storage.k8s.io/persistentvolumeclaims
- 对象数，即可创建的对象的个数
    - pods, replicationcontrollers, configmaps, secrets
    - resourcequotas, persistentvolumeclaims
    - services, services.loadbalancers, services.nodeports

它的工作原理为：

- 资源配额应用在 Namespace 上，并且每个 Namespace 最多只能有一个 ResourceQuota 对象
- 开启计算资源配额后，创建容器时必须配置计算资源请求或限制(也可以 用 LimitRange 设置默认值)
- 用户超额后禁止创建新的资源