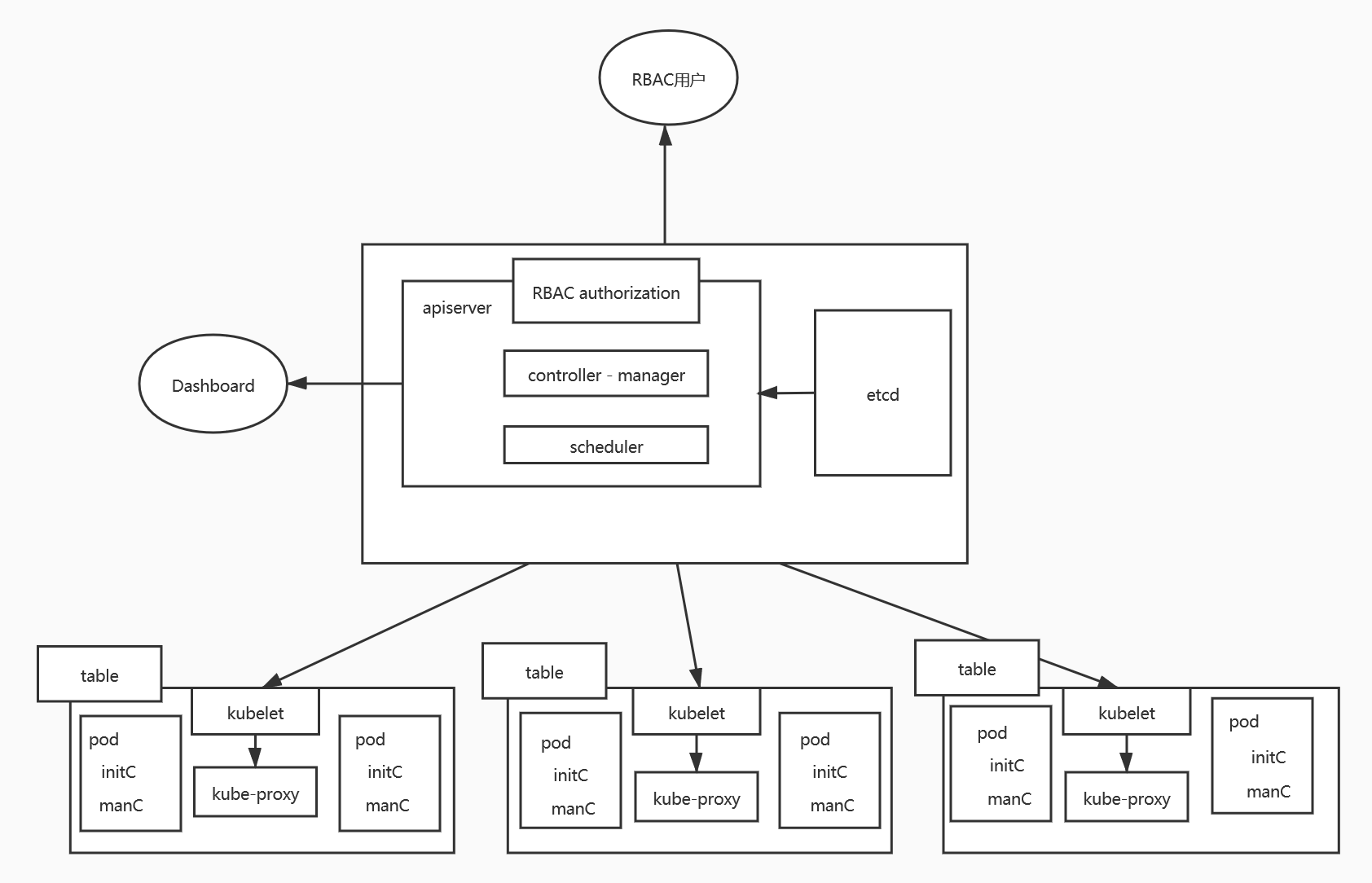
# 架构 & 基础

## Kubernetes 架构



一个完整的kubernetes 集群需要有主从两种节点。

Master 节点：

1. Kube-apiserver : 用于暴露k8s api，任何的资源请求/调用操作都是通过kube-apiserver 提供的接口进行。
2. Etcd：etcd是k8s默认的存储系统，保存所有集群数据，使用时要为etcd数据提供备份计划。
3. Kube-controller-manager：运行管理控制器，集群中处理常规任务的后台进程。

- node 节点控制器

- replication 副本控制器

- endpoints SVC 和 pod 连接的 Endpoints 控制器，关联service 和 pod。当pod发生变化，会实时更新 Endpoints。

- ServiceAccount 和 token 控制器：为k8s每个命名空间下创建使用账户访问api

1. Cloud-controller-manager：云平台的管理控制器
2. Kube-scheduler：负责节点资源管理，接收来自kube-apiserver创建pods任务，并分配每个节点。
3. CoreDNS：可选的DNS服务，为每个service对象创建DNS记录，使所有的pod可以通过dns访问。

Node 节点：

1. Kubelet：是主要的节点代理工具，会监视已分配给的节点pod，管理pod容器。Kubelet会在 api-server 上注册节点信息，并定期向master汇报节点的资源使用情况。
2. Kubectl：客户端命令行工具
3. Kube-proxy：运行在每个节点上，负责pod网络代理。定时从etcd获取到svc信息来做相应的策略。
4. Container runtime：负责镜像管理以及Pod和容器的真正运行（CRI）；

## 什么是kubernetes，kubernetes可以用来做什么?

Kubernetes 是用于部署容器化应用程序的编排工具。通过编写yaml文件调用API接口实现各种工作任务。

**Kubernetes 能用来干什么?**

1. 将生产业务服务容器化，并提供部署运行、资源调度、服务发现、和动弹伸缩等一系列功能。
2. 具有完备的集群管理能力，多层次的安全防护和准入机制、多租户应用支撑能力、透明的服务注册和发现机制。
3. 內建智能负载均衡器、强大的故障发现和自我修复能力、服务滚动升级和在线扩容能力、可扩展的资源自动调度机制 。

## 3.Kubernetes 有状态、无状态服务

**无状态：deployment**  
　　- 认为所有pod都是一样的，不具备与其他实例有不同的关系。  
　　- 没有顺序的要求。  
　　- 不用考虑再哪个Node运行。  
　　- 随意扩容缩容。

**有状态：SatefulSet**  
　　- 集群节点之间的关系。  
　　- 数据不完全一致。  
　　- 实例之间不对等的关系。  
　　- 依靠外部存储的应用。  
　　- 通过dns维持身份

## Kubernetes 中的 kubelctl、kubelet、kubeadm

**Kubeadm：** 是集群二进制的部署工具，可以初始化集群，设置集群token等

**Kubelet：**是一个代理服务，它在每个节点上运行，并使从服务器与主服务器通信。

**Kubectl：**是一个命令行工具，可以使用该工具控制kubernetes集群管理器、检查集群资源、创建、删除、更新集群服务，查看程序信息等等。

## 简述Kube-proxy 作用

kube-proxy 运行在所有节点上，它监听 apiserver 中 service 和 endpoint 的变化情况，创建路由规则以提供服务 IP 和负载均衡功能。简单理解此进程是 Service 的 透明代理兼负载均衡器，其核心功能是将到某个 Service 的访问请求转发到后端的多 个 Pod 实例上。

## Kubernetes 命名空间是做什么的

命名空间是k8s中的虚拟集群，每一个命名空间都是一个隔离区。为不同的工作人员提供不同的命名空间，防止误操作。

相同命名空间内的pod默认处于同一个网段。不同的命名空间网段是不同的，如果想要跨命名空间访问需要通过 svc服务发现。

Default，kube-system，kube-public 这三个默认的命名空间不要创建任何与业务有关的pod。

## HPA 自动扩缩容机制

1. 资源使用率都有高峰和低谷，自动伸缩会提高集群整体资源利用率，让pod个数自动调整，不需要手动扩容，自动化。

2. 利用CPU的使用率来伸缩，利用内存使用大小来伸缩。

3. Metric-server 服务采集，是HPA实现伸缩的监控工具。Metric会采集节点中 pod 的使用情况（包括内存，磁盘，cpu和网络的使用率）。

**HPA的工作原理**

HPA在kubernetes中由一个controller控制，controller会间隔循环监测HPA中监控的指标是否触发伸缩条件，默认的间隔时间为15s。

一旦触发伸缩条件，controller会向kubernetes发送请求，修改伸缩对象子对象中控制pod副本数量的结构体字段。

kubernetess响应请求，修改scale结构体，然后会刷新一次伸缩对象的pod数量。伸缩对象被修改后，自然会通过list/watch机制增加或减少pod数量，达到动态伸缩的目的。

对于每个pod的资源指标(如CPU)，controller 会从 metric资源指标API中获取每一个HPA指定的pod的指标。如果设置了目标使用率，控制器会获取每个Pod中的容器资源使用情况，并计算资源使用率。如果使用原始值，将直接使用原始数据，进而计算出目标副本数。

这里注意的是，如果Pod某些容器不支持资源采集，那么该控制器将不会使用该pod的CPU使用率。

## 简述 Kubernetes 原生的服务控制器

1. [Deployment](https://kubernetes.io/zh/docs/concepts/workloads/controllers/deployment/) 和 [ReplicaSet](https://kubernetes.io/zh/docs/concepts/workloads/controllers/replicaset/) （替换原来的资源 [ReplicationController](https://kubernetes.io/zh/docs/reference/glossary/?all=true" \l "term-replication-controller" \t "http://www.bhmy.top/_blank)）。

Deployment 很适合用来管理你的集群上的无状态应用，Deployment 中的所有 Pod 都是相互等价的，并且在需要的时候被换掉。

1. StatefulSet 能够运行一个或者多个以某种方案跟踪应用状态的 pods。

例如需要数据作持久存储的服务：数据库。将每个pod 与 pv 对应起来。且可以根据所需容量动态分配到最符合需要的 pv 上进行挂载。

1. DaemonSet 定义提供节点本地支撑设施的 pods。

- Daemonset 资源会在k8s集群的每一个节点上运行，并且每个节点只能运行一个pod。

- 在每个节点做日志收集。

- 监控每个节点的运行状态。

1. Job 和 CronJob：定义一些一直运行到结束并停止的任务。

Job 是一次性任务，CronJob是规划时间并反复执行的任务。

## InitC的作用

确保MainC容器的安全性，冗余性

1. 在主容器被创建之前，需要有一些文件被创建，需要准备好一些数据，但是这些数据或文件的创建工具，加载到 Main C 之后会造成Main C容器的冗余、资源重复，随着工具越来越多会出现资源的严重浪费。
2. 所以将这些基础的配置文件，工具写入到 Init C 容器，让他在初始化的过程中帮助 Main C 将所用的文件数据提前创建出来。这样可以最大化利用 Pod 内部资源。
3. Init 容器通过 Namespace，将 Main C容器所需要的文件，通过 Init C来进行访问下载读取，直接将初始化的文件传给Main C ，保证安全性。
4. Init C 可以探测 Main C 容器是否成功启动

如果Init容器启动失败，则不会启动mainC容器。K8s会不断地重启该pod，直到Init容器启动成功为止。不过如果它的重启策略为 Never则不会重启。

## SideCar 容器

1. Sidecar 是一种容器的设计模式，在提供业务的pod基础上，定义一些专门处理其它辅助工作的容器，业务pod只处理业务响应，而其他的工作由sidcar来实现。
2. 原本在 initC容器里干的事情可以放到 sidcar中进行。举例如下：

- 日志收集处理：将sidecar的目录与主业务pod共享，由sidecar处理业务中的日志。

- 代理proxy：比如数据库分库分表，由代理proxy进行处理，写走写库，读走读库。

- 服务网格：来自单个服务的所有传入和传出网络流量都流经 Sidecar 代理。

- 打包部署，软件更新迭代后将包发送到sidecar上。

- Debug 应用

- 应用监控

1. 将辅助功能同主业务进行解耦。

## K8s 各模块如何与 Api-Server 通信

Kubernetes API Server 作为集群的核心，负责集群各功能模块之间的通信。集群内的各个功能模块通过 API Server 将信息存入 etcd，当需要获取和操作这些数据时，则通过 API Server 提供的 REST 接口（用 GET、LIST 或 WATCH 方法）来实现， 从而实现各模块之间的信息交互。

**kubelet 进程与 API Server 的交互：**

每个 Node 上的 kubelet 每隔一个时间周期， 就会调用一次 API Server 的 REST 接口报告自身状态，API Server 在接收到这些信息 后，会将节点状态信息更新到 etcd 中。

**kube-controller-manager 进程与 API Server 的交互：**

kube-controller- manager 中的 Node Controller 模块通过 API Server 提供的 Watch 接口实时监控 Node 的信息，并做相应处理。

**kube-scheduler 进程与 API Server 的交互：**

Scheduler 通过 API Server 的 Watch 接口监听到新建 Pod 副本的信息后，会检索所有符合该 Pod 要求的 Node 列 表，开始执行 Pod 调度逻辑，在调度成功后将 Pod 绑定到目标节点上。

## K8s scheduler 的作用

Scheduler 是负责Pod调度的重要功能模块，scheduler在整个系统中承担了承上启下的重要功能。“承上”是指它负责接收Controller 创建的新Pod，为其调度至目标Node；“启下”是指调度完成后，目标Node上的kubelet 服务进程接管后续工作，负责Pod接下来的生命周期。

Scheduler 的作用是将待调度的Pod按照特定的调度算法和调度策略绑定到集群中适合的Node节点上，并将绑定信息写入Etcd中。

## Kubernetes 如何保证集群安全性

Kubernetes 通过一系列机制来实现集群的安全控制，可以从三个方面来说：

1. **基础设施安全**

命名空间保证服务间的独立性、隔离性。不同命名空间下的服务网关不同，通信需要svc。

1. **权限安全**

**最小权限原则**：限制所有组件的权限，确保组件只执行它授权的行为，通过限制单个组件的能力来限制它的权限范围。

**用户权限：**划分普通用户和管理员用户，通过证书访授权 问k8s-API。

1. **集群安全**

**ApiServer 认证：**kubernetes集群中所有资源的访问和变更都是通过 API-Server 来实现的，因此需要采用更安全的 https 或 token 认证方式来验证客户端身份。然后对用户进行RBAC授权，访问集群资源。

**ApiServer 授权：**通过RBAC的方式，绑定具体的资源权限，来限制用户对资源的操作，防止误操作。

**敏感数据Secret：**对于集群敏感数据采用加密保存的方式。

## 简述Helm 及其优势

Helm 是 Kubernetes 的软件包管理工具。类似 Ubuntu 中使用的 apt、Centos 中使 用的 yum 或者 Python 中的 pip 一样。

Helm 能够将一组 K8S 资源打包统一管理, 是查找、共享和使用为 Kubernetes 构建的软件的最佳方式。Helm 中通常每个包称为一个 Chart，一个 Chart 是一个目录（一般情况下会将目录 进行打包压缩，形成 name-version.tgz 格式的单一文件，方便传输和存储）。

**Helm 优势**

在 Kubernetes 中部署一个可以使用的应用，需要涉及到很多的 Kubernetes 资源的 共同协作。使用 helm 则具有如下优势：

* 统一管理、配置和更新这些分散的 k8s 的应用资源文件；
* 分发和复用一套应用模板；
* 将应用的一系列资源当做一个软件包管理。
  + 对于应用发布者而言，可以通过 Helm 打包应用、管理应用依赖关系、管理应用版本并发布应用到软件仓库。
  + 对于使用者而言，使用 Helm 后不用需要编写复杂的应用部署文件，可以以简单的方式在 Kubernetes 上查找、安装、升级、回滚、卸载应用程序。

# Kubenetes 策略

## k8s-Api 镜像下载策略，image状态

**K8s的镜像下载策略有三种：Always、Never、IFNotPresent；**

1. **Always：**镜像标签为latest时，总是从指定的仓库中获取镜像；
2. **Never**：禁止从仓库中下载镜像，也就是说只能使用本地镜像；
3. **IfNotPresent**：仅当本地没有对应镜像时，才从目标仓库中下载。

默认的镜像下载策略是：当镜像标签是latest时，默认策略是Always；当镜像标签是自定义时（也就是标签不是latest），那么默认策略是IfNotPresent。

**Image 状态:**

1. **Running**：Pod所需的容器已经被成功调度到某个节点，且已经成功运行，
2. **Pending**：APIserver创建了pod资源对象，并且已经存入etcd中，但它尚未被调度完成或者仍然处于仓库中下载镜像的过程。
3. **Unknown**：APIserver无法正常获取到pod对象的状态，通常是其无法与所在工作节点的kubelet通信所致

## K8s - 节点亲和性策略

**延申问题：k8s某个集群节点需要关机维护该如何操作？**

1.cordon：标记节点不可调度，后续新的pod不会被调度到此节点，但是该节点上的pod可以正常对外服务；

2.drain：驱逐节点上的pod至其他可调度节点；

3.uncordon：恢复标记节点可调度；

## k8s - pod 对外暴露策略

HostNetwork：使用宿主机网络。直接暴露在宿主机网络环境中。

HostPort：将容器的端口映射到宿主机的端口，这样用户就可以通过宿主机的IP来访问Pod。

NodePort：集群服务暴露外网的方式，在30000~32767之间选择一个端口进行映射。

loadBalancer：是公有云提供的负载均衡器

Ingress：原生的nginx-ingress反向代理负载均衡

Istio+envoy(vs+gateway)：网关服务

## K8s - pod 重启策略

Always：但凡pod对象终止就重启，此为默认策略。

OnFailure：仅在pod对象出现错误时才重启

Never：不论容器运行状态如何，kubelet 都不会重启该容器。

## Kubernetes 主机调度策略

Spec.nodename 使用集群节点名字进行调度

Spec.nodeSelector 使用节点标签进行调度

## pv的访问模式

ReadWriteOnce：只能以读写的方式挂载到单个节点；

ReadOnlyMany：能以只读的方式挂载到多个节点；

ReadWriteMany：能以读写的方式挂载到多个节点。

## pv的回收策略

recycle：清除PV的数据，然后自动回收；

Retain：需要手动回收；

delete：删除云存储资源，云存储专用；

## storageClass 动态存储绑定模式

Immediate：一旦创建pvc就绑定。

WaitForFirstConsumer：延迟绑定，直到使用该pvc的pod被创建。

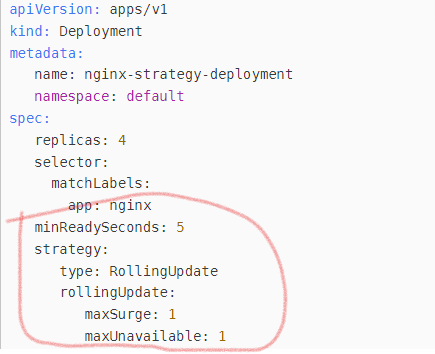
## SVC 负载分发策略有那些？

RoundRobin：轮询模式，即轮询将请求转发到后端的各个Pod上

SessionAffinity：基于客户端IP地址进行会话保持的模式，第一次客户端访问后端某个Pod，之后的请求都转发到这个Pod上

## Deployment滚动更新策略

滚动更新策略主要用来解决发布时遇到的 “太快不稳定”或者“太慢体验差”的情况。



**minReadySeconds:**

表示kubernetes在等待设置时间后才进行升级，如果没有设置该值kubernete会假设容器启动起来就提供服务了，默认值是0

**type：RollingUpdate / recreate**

RollingUpdate表示，设置更新策略为滚动更新，可以设置为recreate和RollingUpdate两个值，recreate表示全部重新创建，默认值就是RollingUpdate。

**maxUnavailable：**

和期望ready的副本数比，不可用副本数最大值，这个值越小，越能保证服务稳定，更新越平滑；

**maxSurge：**

和期望ready的副本数比，超过期望副本数最大值，这个值调的越大，副本更新速度越快。

# Kubernetes - Pod 知识点

## kubernetes pod 的所有状态

Pending：API Server 已经创建该 Pod，且 Pod 内还有一个或多个容器的镜像没

有创建，包括正在下载镜像的过程。

Running：Pod 内所有容器均已创建，且至少有一个容器处于运行状态、正在启

动状态或正在重启状态。

Succeeded：Pod 内所有容器均成功执行退出，且不会重启。

Failed：Pod 内所有容器均已退出，但至少有一个容器退出为失败状态。

Unknown：由于某种原因无法获取该 Pod 状态，可能由于网络通信不畅导致。

## 简述kubernetes创建一个pod的主要流程

1. 客户端提交pod 的yaml文件调用 k8s-apiServer
2. ApiServer 接收到指令后，通过 controller 控制器创建资源对象。
3. Controller-manager 通过 api-server 将pod的配置信息存储到 etcd 数据中心。
4. Kube-scheduler 检测到 pod信息后会开始调度预选，过滤掉不符合pod资源配置要求的节点，然后根据调度策略开始调度，将pod的资源配置发送到node节点的kubelet上。
5. Kubelet 根据 shceduler 发来的资源配置单运行pod，运行成功后将pod的运行信息返回给 scheduler。Scheduler 将pod的运行信息存储到etcd上。

## Kubernetes Pod重启策略

Always：但凡pod对象终止就重启，此为默认策略。

OnFailure：仅在pod对象出现错误时才重启

Never：不论容器运行状态如何，kubelet 都不会重启该容器。

**不同 controller 控制器的默认重启策略：**

1. RC 和 DaemonSet：必须设置为 Always，需要保证该容器持续运行；
2. Job：OnFailure 或 Never，确保容器执行完成后不再重启；

## Kubernetes pod健康检查方式

对 Pod 的健康检查可以通过两类探针来检查：LivenessProbe 和 ReadinessProbe。

**LivenessProbe 探针：用于判断容器是否存活（running 状态）。**

如果 LivenessProbe 探针探测到容器不健康，则 kubelet 将杀掉该容器，并根据容器 的重启策略做相应处理。若一个容器不包含 LivenessProbe 探针，kubelet 认为 该容器的 LivenessProbe 探针返回值用于是“Success”。

**ReadineeProbe 探针：用于判断容器是否启动完成（ready 状态）。**

如果 ReadinessProbe 探针探测到失败，则 Pod 的状态将被修改。Endpoint Controller 将从 Service 的 Endpoint 中删除包含该容器所在 Pod 的 Eenpoint。

**startupProbe 探针：启动检查机制**

应用一些启动缓慢的业务，避免业务长时间 启动而被上面两类探针 kill 掉。

## Kubernetes pod 的常见调度方式

1. **Deployment 或 RC控制器调度：**该调度方式会自动部署一个容器应用的多个副本，可以持续监控副本数量，在集群内始终维持用户指定的副本数量。
2. **NodeSelector：定向调度**，需要手动将Pod调度到特定节点上。也可以通过Node标签进行调度。
3. **NodeAffinity 亲和性调度**：亲和性调度机制极大的扩展了 Pod 的调度能力，目前

有两种节点亲和力表达：（软硬规则）

- 硬规则：必须满足指定的规则，调度器才可以调度 Pod 至 Node 上。

- 软规则：优先调度至满 足的 Node 的节点，但不强求，多个优先级规则还可以设置权重值。

1. **Taints 和 Tolerations（污点和容忍）：**

- Taint：使 Node 拒绝特定 Pod 运行；

- Toleration：为 Pod 的属性，表示 Pod 能容忍（运行）标注了 Taint 的 Node。

## Kubernetes 对外暴露 pod 的方式有哪些

HostNetwork：使用宿主机网络。直接暴露在宿主机网络环境中。

HostPort：将容器的端口映射到宿主机的端口，这样用户就可以通过宿主机的IP来访问Pod。

NodePort：集群服务暴露外网的方式，在30000~32767之间选择一个端口进行映射。

LoadBalancer：只能在service上定义，是公有云提供的负载均衡器

Ingress：ingress controller是由K8s管理的负载均衡容器，它的镜像包含一个nginx或HAProxy负载均衡器和一个控制器守护进程。

## 一个请求到Pod接收响应，中间经过哪些过程？

1. 首先是通过网络I/O到达k8s主节点的外网网卡并建立TCP三次握手。
2. 通过网桥将网络I/O转到flanned网段。
3. 然后根据请求域名在k8s集群中找到对应7层负载的API，比如IngressNginx，treafike所设定的负载均衡域名。
4. 由负载均衡域名找到对应的服务发现域名和映射端口号，每一个服务发现的名字都会对照一个pod容器。
5. 到达pod容器后根据请求路径反馈数据。

## Pod被调度到一个节点上的过程

1. 客户端提交Pod的配置信息（可以是yaml文件定义好的信息）到kube-apiserver；

2） Apiserver收到指令后，通知给controller-manager创建一个资源对象；

3） Controller-manager通过api-server将pod的配置信息存储到ETCD数据中心中；

4） Kube-scheduler检测到pod信息会开始调度预选，会先过滤掉不符合Pod资源配置要求的节点，然后开始调度调优，主要是挑选出更适合运行pod的节点，然后将pod的资源配置单发送到node节点上的kubelet组件上。

5） Kubelet根据scheduler发来的资源配置单运行pod，运行成功后，将pod的运行信息返回给scheduler，scheduler将返回的pod运行状况的信息存储到etcd数据中心。

## Pod如何平滑升级

Deployment滚动更新的功能，需要准备一个新的镜像。

1. Kubectl patch 更新容器的镜像版本，同时可以控制是否要增加几个pod（命令比较复杂）
2. Kubectl set 命令直接修改镜像版本（命令比较简单）
3. Kubectl edit 直接修改镜像版本

# Kubernetes controller

## 简述 Kubernetes deployment 升级过程？

## kubernetes deployment 升级策略？

## 不同控制器的重启策略

1. RC 和 DaemonSet：必须设置为 Always，需要保证该容器持续运行；
2. Job：OnFailure 或 Never，确保容器执行完成后不再重启；

# Kubernetes 持久化

## Kubernetes 怎么做持久化

**1.持久化存储类型分为两种：volume 和 storageClass**

**Volume 静态数据卷有4种：**

1. emptryDir 临时卷，当容器被删除后，该卷也删除
2. HostPath 本地卷，将节点的目录挂载到容器中，也可以通过存储服务器将块设备共享到节点上，在通过hostpath挂载共享块设备。
3. Local 本地卷，需要配合节点亲和性指定标签来指定节点。
4. NFS 共享服务器，共享目录

**StorageClass 动态类存储**

1. StorageClass 是管理员才能创建的 类存储。
2. 类存储 通过：服务质量等级 、 策略 、自定义任意策略，来实现映射。
3. 为持久卷(pv)，动态分配 类存储实现持久化。当有pod要使用持久化时，通过包含的三个字段，自动分配到存储空间。

**2.持久化的方式通过 pv 和pvc**

****PV**** 是由管理员设置的存储，PV也是集群中的持久化资源，每一个pv都会对应一个不同的存储服务。不受pod 或用户管理，独立于 PV 的 Pod 之外的生命周期。

**PVC** 是pv的请求方案，通过pvc来寻找合适的pv进行匹配。如果有两个pv同时满足需求，会选择一个最贴近需求的存储卷来进行匹配。Pod 消耗节点资源，PVC 消耗 PV 资源。

## PV 生命周期内的阶段？

答：某个 PV 在生命周期中可能处于以下 4 个阶段（Phaes）之一。

l Available：可用状态，还未与某个 PVC 绑定。

l Bound：已与某个 PVC 绑定。

l Released：绑定的 PVC 已经删除，资源已释放，但没有被集群回收。

l Failed：自动资源回收失败。

## pv的访问模式

ReadWriteOnce：只能以读写的方式挂载到单个节点；

ReadOnlyMany：能以只读的方式挂载到多个节点；

ReadWriteMany：能以读写的方式挂载到多个节点。

## Pv的回收策略

recycle：清除PV的数据，然后自动回收；

Retain：需要手动回收；

delete：删除云存储资源，云存储专用；

## 4.storageClass 动态存储绑定模式

Immediate：一旦创建pvc就绑定。

WaitForFirstConsumer：延迟绑定，直到使用该pvc的pod被创建。

# Kubernetes SVC

## 1.简述 Kubernetes Service 类型？

答：通过创建 Service，可以为一组具有相同功能的容器应用提供一个统一的入口地址，

并且将请求负载分发到后端的各个容器应用上。其主要类型有：

* ClusterIP：虚拟的服务 IP 地址，该地址用于 Kubernetes 集群内部的 Pod 访问， 在 Node 上 kube-proxy 通过设置的 iptables 规则进行转发；
* NodePort：使用宿主机的端口，使能够访问各 Node 的外部客户端通过 Node

的 IP 地址和端口号就能访问服务；

* LoadBalancer：使用外接负载均衡器完成到服务的负载分发，需要在

spec.status.loadBalancer 字段指定外部负载均衡器的 IP 地址，通常用于公有云

## 简述 Kubernetes 外部如何访问集群内的服务？

Kubernetes 集群内的服务：

1. **映射 Pod 到物理机：**将 Pod 端口号映射到宿主机，即在 Pod 中采用 hostPort 方

式，以使客户端应用能够通过物理机访问容器应用。

1. **映射 Service 到物理机：**将 Service 端口号映射到宿主机，即在 Service 中采用

nodePort 方式，以使客户端应用能够通过物理机访问容器应用。

1. **映射 Sercie 到 LoadBalancer：**通过设置 LoadBalancer 映射到云服务商提供的

LoadBalancer 地址。这种用法仅用于在公有云服务提供商的云平台上设置 Service 的场景。

1. **Istio gateway网关映射SVC**
2. **Nginx-ingress 网关映射SVC**

# Kubernetes 集群网络

## 1.Flannel网络通信

Flannel：使用vxlan技术为各节点创建一个可以互通的Pod网络，使用的端口为UDP 8472（需要开放该端口，如公有云AWS等）。

flanneld第一次启动时，从etcd获取配置的Pod网段信息，为本节点分配一个未使用的地址段，然后创建flannedl.1网络接口（也可能是其它名称，如flannel1等）。

flannel将分配给自己的Pod网段信息写入 /run/flannel/subnet.env 文件，docker后续使用这个文件中的环境变量设置docker0网桥，从而从这个地址段为本节点的所有Pod容器分配IP。

## 2 Calico 网络组件通信

Calico 是一个基于 BGP 的纯三层的网络方案，与 OpenStack、Kubernetes、AWS、GCE 等云平台都能够良好地集成。Calico 在每个计算节点都利用 Linux Kernel 实现了一个高效的 vRouter 来负责数据 转发。每个 vRouter 都通过 BGP 协议把在本节点上运行的容器的路由信息向整个 Calico 网络广播，并自动设置到达其他节点的路由转发规则。

Calico 保证所有容器之间的数据流量都是通过 IP 路由的方式完成互联互通的。Calico 节点组网时可以直接利用数据中心的网络结构（L2 或者 L3），不需要额外的 NAT、 隧道或者 Overlay Network，没有额外的封包解包，能够节约 CPU 运算，提高网络 效率。