- k8s之原理
  - 资源调度策略解析
    - 1、Predicates
      - GeneralPredicates
        - 负责最基础的调度策略
          - PodFitsResources
          - PodFitsHost
          - PodFitsHostPorts
          - PodMatchNodeSelector
        - 负责跟容器持久化Volume相关的调度策略
          - NoDiskConflict
          - MaxPDVolumeCountPredicate
          - VolumeZonePredicate
          - VolumeBindingPredicate
        - 主要考察待调度Pod是否满足Node本身的某些条件
          - PodToreratesNodeTaints
          - NodeMemoryPressurePredicates
      - 与Pod相关的过滤规则
        - PodAffinityPredicate
    - 2、Priorities
      - 为节点打分：0-10分
      - 打分规则
        - LeastRequestedPriority
        - BalancedResourceAllocation
        - NodeAffinityPriority
        - TaintTolerationPriority
        - InterPodAffinityPriority
  - 监控体系
  - k8s本质
    - 容器组成：实际上是一个由 Linux Namespace、Linux Cgroups 和 rootfs 三种技术构建出来的进程的隔离环境
      - 一组联合挂载在 /var/lib/docker/aufs/mnt 上的 rootfs，这一部分我们称为“容器镜像”（Container Image），是容器的静态视图；
      - 一个由 Namespace+Cgroups 构成的隔离环境，这一部分我们称为“容器运行时”（Container Runtime），是容器的动态视图。
    - 其中，控制节点，即 Master 节点，由三个紧密协作的独立组件组合而成，它们分别是负责 API 服务的 kube-apiserver、负责调度的 kube-scheduler，以及负责容器编排的 kube-controller-manager。整个集群的持久化数据，则由 kube-apiserver 处理后保存在 Etcd 中。
    - 在 Kubernetes 项目中，kubelet 主要负责同容器运行时（比如 Docker 项目）打交道。而这个交互所依赖的，是一个称作 CRI（Container Runtime Interface）的远程调用接口，这个接口定义了容器运行时的各项核心操作，比如：启动一个容器需要的所有参数。
    - 此外，kubelet 还通过 gRPC 协议同一个叫作 Device Plugin 的插件进行交互。这个插件，是 Kubernetes 项目用来管理 GPU 等宿主机物理设备的主要组件，也是基于 Kubernetes 项目进行机器学习训练、高性能作业支持等工作必须关注的功能。
    - kubelet 的另一个重要功能，则是调用网络插件和存储插件为容器配置网络和持久化存储
    - 可以说，kubelet 完全就是为了实现 Kubernetes 项目对容器的管理能力而重新实现的一个组件，与 Borg 之间并没有直接的传承关系。
    - 与其他编排工具(swarm,compose)的优势
      - Kubernetes 项目最主要的设计思想是，从更宏观的角度，以统一的方式来定义任务之间的各种关系，并且为将来支持更多种类的关系留有余地。
      - 实际上，过去很多的集群管理项目（比如 Yarn、Mesos，以及 Swarm）所擅长的，都是把一个容器，按照某种规则，放置在某个最佳节点上运行起来。这种功能，我们称为“调度”。
      - 而 Kubernetes 项目所擅长的，是按照用户的意愿和整个系统的规则，完全自动化地处理好容器之间的各种关系。这种功能，就是我们经常听到的一个概念：编排。
      - 所以说，Kubernetes 项目的本质，是为用户提供一个具有普遍意义的容器编排工具。
        不过，更重要的是，Kubernetes 项目为用户提供的不仅限于一个工具。它真正的价值，乃在于提供了一套基于容器构建分布式系统的基础依赖。关于这一点，相信你会在今后的学习中，体会的越来越深。
    - 思考题
      - k8s站在上帝视角看待问题，swarm虽然也可以，目前不成熟
      - 感觉不能迁移“有状态”的容器，是因为迁移的是容器的 rootfs，但是一些动态视图是没有办法伴随迁移一同进行迁移的。
      - 1、你是否了解了 Docker Swarm（SwarmKit 项目）跟 Kubernetes 在架构上和使用方法上的异同呢？
      - 2、在 Kubernetes 之前，很多项目都没办法管理“有状态”的容器，即，不能从一台宿主机“迁移”到另一台宿主机上的容器。你是否能列举出，阻止这种“迁移”的原因都有哪些呢？
  - Service工作原理
    - 从外部访问Service的三种方式
      - 当然，如果你不显式地声明 nodePort 字段，Kubernetes 就会为你分配随机的可用端口来设置代理。这个端口的范围默认是 30000-32767，你可以通过 kube-apiserver 的--service-node-port-range 参数来修改它。
      - NodePort
      - LoadBalancer
      - External Name
      - 访问被代理的Pod
      - externalIPs
    - 其实 Service 是由 kube-proxy 组件，加上 iptables 来共同实现的。
    - 总结
      - 在理解了 Kubernetes Service 机制的工作原理之后，很多与 Service 相关的问题，其实都可以通过分析 Service 在宿主机上对应的 iptables 规则（或者 IPVS 配置）得到解决。
    - 问题
      - 1、一种典型问题，就是 Pod 没办法通过 Service 访问到自己。
        这往往就是因为 kubelet 的 hairpin-mode 没有被正确设置。关于 Hairpin 的原理我在前面已经介绍过，这里就不再赘述了。你只需要确保将 kubelet 的 hairpin-mode 设置为 hairpin-veth 或者 promiscuous-bridge 即可。
  - 存储原理
  - 资源配置与管理
    - CPU
      - 可压缩资源，典型特点：当可压缩资源不足时，pod只会饥饿，但不会退出。单位m，500m=0.5个
      - limits vs requests 区别：
        - 你也可以直接把这个配置写成 cpu=0.5。但在实际使用时，我还是推荐你使用 500m 的写法。毕竟这才是 Kubernetes 内部通用的 CPU 表示方式。
        - 这两者的区别其实非常简单：在调度的时候，kube-scheduler 只会按照 requests 的值进行计算。而在真正设置 Cgroups 限制的时候，kubelet 则会按照 limits 的值来进行设置。
    - Memory
      - 可压缩资源，典型特点：当可压缩资源不足时，pod会因为OOM被内核杀掉。单位bytes: Ei,Pi,Ti,Gi,Mi
      - limit
      - requests
    - 资源回收(Eviction)默认阈值
      - memory.available<100Mi
        nodefs.available<10%
        nodefs.inodesFree<5%
        imagefs.available<15%
      - 可配置，并分为Soft 和 Hard模式
        - 区别是Soft是优雅时间后启动Eviction
        - 当宿主机的 Eviction 阈值达到后，就会进入 MemoryPressure 或者 DiskPressure 状态，从而避免新的 Pod 被调度到这台宿主机上
        - 而当 Eviction 发生的时候，kubelet 具体会挑选哪些 Pod 进行删除操作，就需要参考这些 Pod 的 QoS 类别了。
          - 首当其冲的，自然是 BestEffort 类别的 Pod。
          - 其次，是属于 Burstable 类别、并且发生“饥饿”的资源使用量已经超出了 requests 的 Pod。
          - 最后，才是 Guaranteed 类别。并且，Kubernetes 会保证只有当 Guaranteed 类别的 Pod 的资源使用量超过了其 limits 的限制，或者宿主机本身正处于 Memory Pressure 状态时，Guaranteed 的 Pod 才可能被选中进行 Eviction 操作。
          - 当然，对于同 QoS 类别的 Pod 来说，Kubernetes 还会根据 Pod 的优先级来进行进一步地排序和选择。
      - Kubernetes 计算 Eviction 阈值的数据来源，主要依赖于从 Cgroups 读取到的值，以及使用 cAdvisor 监控到的数据。
    - Qos模型
      - Guaranteed
        - 当 Pod 里的每一个 Container 都同时设置了 requests 和 limits，并且 requests 和 limits 值相等的时候，这个 Pod 就属于 Guaranteed 类别
        - 需要注意的是，当 Pod 仅设置了 limits 没有设置 requests 的时候，Kubernetes 会自动为它设置与 limits 相同的 requests 值，所以，这也属于 Guaranteed 情况。
      - Burstable
        - 而当 Pod 不满足 Guaranteed 的条件，但至少有一个 Container 设置了 requests。那么这个 Pod 就会被划分到 Burstable 类别
      - BestEffort
        - 而如果一个 Pod 既没有设置 requests，也没有设置 limits，那么它的 QoS 类别就是 BestEffort
      - cpuset设置
        - 我们知道，在使用容器的时候，你可以通过设置 cpuset 把容器绑定到某个 CPU 的核上，而不是像 cpushare 那样共享 CPU 的计算能力。
        - 事实上，cpuset 方式，是生产环境里部署在线应用类型的 Pod 时，非常常用的一种方式。
        - 可是，这样的需求在 Kubernetes 里又该如何实现呢？
          - 首先，你的 Pod 必须是 Guaranteed 的 QoS 类型；
          - 然后，你只需要将 Pod 的 CPU 资源的 requests 和 limits 设置为同一个相等的整数值即可。
      - 三种QoS模型作用？
        - QoS 划分的主要应用场景，是当宿主机资源紧张的时候，kubelet 对 Pod 进行 Eviction（即资源回收）时需要用到的。
    - 总结
      - 正是基于上述讲述，在实际的使用中，我强烈建议你将 DaemonSet 的 Pod 都设置为 Guaranteed 的 QoS 类型。否则，一旦 DaemonSet 的 Pod 被回收，它又会立即在原宿主机上被重建出来，这就使得前面资源回收的动作，完全没有意义了。
    - 思考题
      - 为什么宿主机进入 MemoryPressure 或者 DiskPressure 状态后，新的 Pod 就不会被调度到这台宿主机上呢？
        - 被打上污点
  - Ingress工作原理
    - Ingress Controller(控制器)
      - 监控自定义Ingress的变化
        - 自定义Ingress后端是service，service后端是多个deployment类型的pod，也可以Ingress后端直接接pod
        - 匹配不到合适的service，会出现404，除非设置--default-backend-service参数
      - 如果ingress新创建，则生成nginx配置文件(/etc/nginx/nginx.conf)
      - 如果ingress或者svc更新
      - 监控自定义Ingress代理的SVC有变化，控制器是不需要重启
      - 允许通过configmap定制配置
      - 创建Ingress所需的SSL证书和秘钥
    - namespace: ingress-nginx
    - 暴露端口的SVC
      - 所有携带ingress-nginx标签的Pod的80和433端口暴露出去，type=NodePort
      - 公有云的话：type=LoadBalancer
    - 总结
      - Ingress目前只能工作在第七层，而service工作在第四层
      - 访问：curl --resolve —insecure
    - 原理图
  - Paas