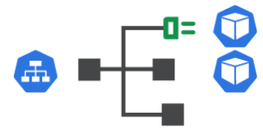
Service

基本概念

- Pod的局限
  - Pod生命周期不确定 — Pod的生命周期是短暂的，可以随时被启动或者是删除，新创建的Pod的IP地址不能固定
  - IP地址不可预知 — k8s只有在Pod在调度到了某一个确定的节点上之后才会为其分配IP地址，在此之前，客户端并不能知道Pod确切的IP地址
- 正因为Pod的多变性，所以k8s提供了一个单一不变的接入点资源，在该资源的生命周期内，其IP地址和端口均不会发生改变，该资源就是Service
- Endpoint
  - Service和Pod之间并不是之间连接的，而是通过Endpoint资源作为资源媒介。当某一个Pod不可用时，该Pod将会从Endpoint列表中摘除。而当有新的Pod被创建时，则会在Endpoint列表中添加



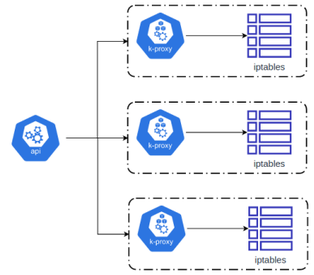
- Service的访问模式
  - VIP (Virtual IP) — 直接使用k8s为Service提供的虚拟集群IP来进行访问，例如 10.0.17.85。在访问到Service之后，根据Endpoint列表来负载均衡的访问服务中的Pods
  - DNS访问
    - 每个Service都会从内部的DNS服务器获取到一个DNS条目，而后客户端就可以通过Service-Name来进行访问了 — <svc-name>.<namespace>.svc.cluster.local
      - srv-name — Service名称
      - namespace — Service所在的namespace

      这种方式又称之为全限定域名(FQDN)访问
    - Normal Service — 在这种处理方式下，访问<svc-name>.<namespace>.svc.cluster.local该域名将会解析成Service的VIP，通过VIP进行访问
    - Headless Service
      - **将Service.spec中的clusterIP字段设置为None会使该Service成为Headless Service，k8s将不会为该服务分配Service VIP**
      - 在这种处理方式下，访问<svc-name>.<namespace>.svc.cluster.local该域名将会直接解析成某一个Pod的IP地址，直接访问Pod，负载均衡功能通过DNS轮询机制实现
      - Headless Service提供了一种通过Pod的域名来直接访问Pod的能力，这将是实现StatefulSet的一个非常重要的前提: 保证Pod的网络标志 — <pod-name>.<svc-name>.<namespace>.svc.cluster.local
- 需要特别注意的是，Service的IP地址为虚拟地址，并没有分配给任何节点的任何网络接口，也就是说，Service IP是无法ping通的

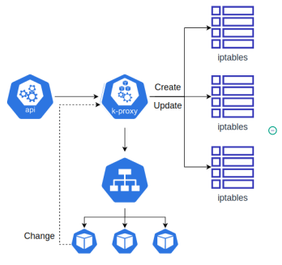
基于iptables实现的Service

Service是由运行在每个节点上的kube-proxy进程，以及节点上的iptables共同实现的

- API-Server与kube-proxy
  - 当用户通过API-Server创建一个Service时，虚拟IP地址将会被立即分配，随后API-Server会通知所有节点的kube-proxy进程，有一个新的Service被创建了
  - 然后，每个kube-proxy都会让该Service在自己的工作节点上可寻址，即建立一系列的iptables规则



- 实例
  - 上下文 — Service上挂3个Pod，Pod IP地址分别为: 10.32.0.4，10.32.0.5，10.32.0.6，Service VIP地址为: 10.108.131.2
  - 查看iptables内容
    - 首先查看kube-proxy对iptables所做的修改
      - command — sudo iptables-save
      - results
        - -A KUBE-SERVICES -d 10.108.131.2/32 -p tcp -m comment --comment "default/hugo_service:default cluster IP" -m tcp --dport 80 -j KUBE-SVC-ODX2UBAZM7RQWOIU
        - 规则含义: 目标地址为10.108.131.2、目标端口为80的IP包，全部转到KUBE-SVC-ODX2UBAZM7RQWOIU这条iptables链上进行处理
    - 查看 KUBE-SVC-ODX2UBAZM7RQWOIU iptables链行为
      - command — sudo iptables-save | grep KUBE-SVC-ODX2UBAZM7RQWOIU
      - results
        - -A KUBE-SVC-ODX2UBAZM7RQWOIU -m comment --comment "default/hugo_service:default" -m statistic --mode random --probability 0.33333333349 -j KUBE-SEP-XKSQDM4YDD4KTZGD
        - -A KUBE-SVC-ODX2UBAZM7RQWOIU -m comment --comment "default/hugo_service:default" -m statistic --mode random --probability 0.50000000000 -j KUBE-SEP-7SURG5ZVQA5YHJ2T
        - -A KUBE-SVC-ODX2UBAZM7RQWOIU -m comment --comment "default/hugo_service:default" -j KUBE-SEP-WEDMYEKG7DO6QEO5
        - 从 --mode random 命令可以看出，这是一个组合随机链，随机链的目的地分别是KUBE-SEP-XKSQDM4YDD4KTZGD、KUBE-SEP-7SURG5ZVQA5YHJ2T、KUBE-SEP-WEDMYEKG7DO6QEO5这三条链又是干嘛的?
    - 查看随机链的行为
      - command — sudo iptables-save | grep -E "KUBE-SEP-XKSQDM4YDD4KTZGD|KUBE-SEP-7SURG5ZVQA5YHJ2T|KUBE-SEP-WEDMYEKG7DO6QEO5"
      - results
        - -A KUBE-SEP-XKSQDM4YDD4KTZGD -p tcp -m comment --comment "default/hugo_service:default" -m tcp -j DNAT --to-destination 10.32.0.4:9090
        - -A KUBE-SEP-7SURG5ZVQA5YHJ2T -p tcp -m comment --comment "default/hugo_service:default" -m tcp -j DNAT --to-destination 10.32.0.5:9090
        - -A KUBE-SEP-WEDMYEKG7DO6QEO5 -p tcp -m comment --comment "default/hugo_service:default" -m tcp -j DNAT --to-destination 10.32.0.6:9090
        - 10.32.0.4，10.32.0.5，10.32.0.6正是Pod的地址
        - 可以看到，这是一个网络地址转换的iptables规则，并且是将目的IP地址进行NAT的规则。而这条DNAT的规则通过查询得知属于PREROUTING链
        - 也就是说，目的地址为10.108.131.2:80的IP包将会随机的分发至三个随机链，而这三个随机链将会在路由前将IP包的目标地址DNAT到Pod的地址上



- 同时，Pod状态的变更也需要发送至kube-proxy进程，kube-proxy需要对iptables进行相应的修改
- 基于iptables实现的Service所能承载的Pod数量有限，想象一下，当集群中有成千上万的Pod时，节点需要花费更多的资源在创建/更新iptables规则上，甚至可能吃掉工作节点所有的CPU资源

基于IPVS实现的Service

- 前面我们提到了基于iptables实现的Service在大规模集群中会出现频繁地更新巨大iptables规则，导致节点资源紧张的问题。并且，一旦节点上承载了成百上千的iptables规则时，运维以及问题排查将会是一个非常痛苦且低效的过程
- 为了解决这个问题，k8s提供了基于IPVS的解决方案，虽然IPVS在内核中仍然是基于netfilter的NAT实现的，但是IPVS不需要为每个Pod设置iptables规则，并且规则的处理也是在内核中进行，极大降低了维护规则的代价