**四层负载均衡**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 集群内部cluster ip示例：  创建deployment    Pod端口80，标签 app=nginx，副本3个  定义service    80-80 443-443    可以看见：    可以看到svc ip为10.106.251.239  可以看到三个pod ip全部加入到同名endpoint里    可以访问到    查看每个pod的日志，可以看到是轮询访问：    访问pod应用的方法：  Pod ip：内端口  Service ip ：port  外部访问nodeport示例：  externalTrafficPolicy  为 Local或cluster  意思是k8s集群的每个node都有所有service信息，通过任意node所在主机的ip：nodeport，找到node主机（这里是你说的本节点），然后通过nodeport查找service信息，找到service，service根据externalTrafficPolicy使用负载策略，如果是local在本node，对照endpoint找pod是否存在，存在则调度到本节点的该pod，不存在则丢弃请求；如果是Cluster，则根据负载均衡到endpoint的随机节点，更加均匀的分布请求到后端  **外部请求流向 NodePort / LoadBalancer（带 externalTrafficPolicy）**  1️⃣ **请求到达任意节点**（客户端访问 NodePort 或云 LB IP） 2️⃣ **节点上的 kube-proxy 查找 Service 信息**   * Kubernetes 集群的每个 Node 上都维护了一份 Service 对应的规则和 Endpoint 列表（通过 iptables 或 IPVS） 3️⃣ **Service 根据 externalTrafficPolicy 决定调度策略** * **Local**   + 节点只查本节点上有没有该 Service 的 Pod（即 Endpoint 指向本节点的 Pod）   + 有 Pod → 转发流量到 Pod   + 没有 Pod → 请求丢弃 * **Cluster**（默认）   + 节点可以把流量转发到集群中任意节点上的 Pod   + 流量分布均匀，Pod 数量越多，负载越均衡   **🔑 核心理解**   * **每个 Node 都“知道” Service 和 Endpoint 信息**，所以请求可以在任意节点入站 * **Local 模式 = 节点局部调度 + 保留客户端 IP** * **Cluster 模式 = 全集群调度 + 均匀负载，但客户端 IP 会丢失**   ✅ **一句话总结**  externalTrafficPolicy 控制的是 **节点如何选择后端 Pod**，Local 限定本节点 Pod，Cluster 跨节点调度，Service 仍然是流量的抽象层。  创建service      Type为nodeport  内部访问不用加端口：    外部访问加端口：（得注释掉externalTrafficPolicy）      看流量调度策略：    Externelname类型service：访问集群外服务  ExternalName Service **不映射任何 Pod**，它只是做 DNS 别名，把集群内部访问的 Service 名解析到外部域名。  **什么是 ExternalName 的 DNS 别名解析**   * ExternalName Service **本质上就是一个 DNS CNAME** * 功能：把 **Service 名称** 映射到一个 **外部域名**    Pod 发起请求：curl http://external-api.default.svc.cluster.local   集群 DNS（CoreDNS）解析：   * 查到 Service 类型是 ExternalName → 返回 api.example.com 作为 CNAME    Pod 继续 DNS 解析 api.example.com → 得到实际 IP   请求发往外部 IP  Pod解析流程：  Pod 访问 Service 名 → CoreDNS 返回 externalName → Pod 再解析到最终 IP   **同一 Namespace 内**的 Pod 可以通过 <service-name>.<namespace>.svc.cluster.local 访问 Service。   **跨 Namespace**访问需要写全 DNS 名或通过 Ingress/外部访问。  总结就是externalname相当于一个namespace范围所有pod内container的cname记录，<service-name>.<namespace>.svc.cluster.local->externalName  定义service对象：    定义pod对象：      Nodebanlancer类型service    应用后：    Externel-ip部分实际是：是SLB的分配的，这里实验没继续  综合示例：  安装mariadb：  定义service：    在 Service（type: ClusterIP 等）中：  ports:  - port: 3306  如果你**没有显式写 targetPort**，Kubernetes 会自动让  targetPort = port  也就是默认目标端口和暴露端口一致。  应用：    定义endpoint对象：    这里定义的endpoint的作用是定义外部服务作为后端负载均衡集群的一员  明确告诉 Kubernetes：Service 名为 mysql 的服务后端地址，不是集群内的 Pod，而是外部的某个 IP（比如物理机、虚拟机或外部数据库服务器）。   * metadata.name: mysql 👉 必须和对应的 Service（同命名空间下的）名称 **一致**。 Service 会根据名字自动“绑定”这个 Endpoints。 * subsets.addresses.ip: 192.168.80.46 👉 指定外部服务所在主机的 **真实 IP**。 * subsets.ports.port: 3306 👉 对应服务端口（如外部 MySQL）。   应用endpoint：    访问：    Endpoint的两种使用情况：  **Endpoints 的两种使用场景**  **① 内部 Pod 场景（默认自动生成）**   * 当你创建一个普通的 Service（例如 ClusterIP 或 NodePort），并设置了 selector 时， Kubernetes 控制器会**自动生成一个同名的 Endpoints 对象**。 * 这个 Endpoints 里记录的是所有匹配该 selector 的 Pod 的 IP + Port。 * Service 根据这些 Endpoints 实现内部负载均衡。   📘 举例：  apiVersion: v1  kind: Service  metadata:  name: nginx-svc  spec:  selector:  app: nginx  ports:  - port: 80  自动生成的 Endpoints（示例）：  apiVersion: v1  kind: Endpoints  metadata:  name: nginx-svc  subsets:  - addresses:  - ip: 10.244.1.5  - ip: 10.244.2.7  ports:  - port: 80  **② 外部服务场景（手动创建）**   * 当 **Service 没有 selector** 时，Kubernetes 不会自动生成 Endpoints。 * 你可以**手动创建一个同名的 Endpoints**，把外部服务 IP 写进去。 * Service 就会把请求转发到这些外部地址，实现“外部服务的负载均衡或代理”。   Nodeport和cluster ip类型的service区别：   |  |  |  | | --- | --- | --- | | **关键点** | **ClusterIP** | **NodePort** | | **type** | ClusterIP（默认） | NodePort | | **是否有 ports.nodePort 字段** | ❌ 无 | ✅ 有（可指定或自动分配 30000–32767） | |