|  |
| --- |
| \*/ |

命令列表

|  |
| --- |
| kubectl –help    表格  AI 生成的内容可能不正确。 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **创建pod**    两个node导入两个nginx  创建pod的语句：   |  |  |  |  | | --- | --- | --- | --- | | **创建方式** | **创建对象** | **是否受控制器管理** | **使用场景** | | kubectl run nginx --image=nginx:latest --restart=Never / Pod YAML（kubectl apply -f pod.yaml） | 单个 Pod | ❌ 不受 Deployment 管理 | 临时测试、简单容器 | | kubectl create deployment | Deployment → Pod | ✅ Deployment 管理 | 生产环境、需要副本/自动恢复/滚动升级 |         命令创建pod少用，多是yaml创建  --port暴露的是容器内部端口；  **发布pod**    kubectl api-resources k8s支持的所有端口    通过kubectlexpose命令创建Service，设置内部通信端口和外部暴露的端口均为80、名称为nginx-service、类型是NodePort。创建Service完后，就可以通过kubectl get svc命令查看到对外暴露的端口是31825，内部通信的地址是10.107.52.98。  root@k8s-master~$>kubectl expose deployment nginx-deployment --port=80 --target-port=80 --name=nginx-service --type=NodePort **将 Deployment 暴露为一个 Service** 的命令。创建service的语句是kubectl expose  root@k8s-master~$>kubectl get pods,svc    这里是随机的  访问本机ip：30946，轮询到三个pod  外部访问流程：  “外部访问用的是该 cluster 任一 Node 的 IP，通过 kube-proxy 找到 Service（ClusterIP:ServicePort），再负载均衡到对应 Pod（TargetPort）”  **外部访问时，你访问的 Node 不一定就是运行 Pod 的 Node**。  内部访问流程：  “内部访问是 service（clusterip:service port）→ 负载均衡到 cluster 的某一 node: targetport” 也完全正确，只是**在内部访问时**不会经过 NodePort， 而且“cluster 的某一 node”只是 Pod 所在节点，**请求直接通过虚拟网络（CNI）转发过去**，并不是像外部访问那样经由物理节点端口  一个 Service 是一组 Pod 的抽象统一入口。 --port 是 Service 自己的入口端口， --target-port 是每个 Pod 的真实端口， kube-proxy 负责把请求从 Service 层负载均衡到其中一个 Pod。 如果是 NodePort 类型，还会在 Node 层额外开放一个外部访问入口   |  |  |  |  | | --- | --- | --- | --- | | **参数** | **层级** | **含义** | **示例** | | --port | Service 层 | Service 对外暴露给“集群内”的统一访问端口 | 80 | | --target-port | Pod 层 | Pod 内部容器应用实际监听的端口 | 80 | | --node-port | Node 层（仅 NodePort 类型） | 集群外部访问入口端口（系统默认 30000–32767） | 31080（例如） |   **Service 的类型（type）决定“Service 如何被访问”**   |  |  |  |  | | --- | --- | --- | --- | | **Service Type** | **访问来源** | **访问方式** | **说明** | | **ClusterIP**（默认） | 集群内 | 通过 Service 的 ClusterIP:Port | 最常用，提供集群内部访问（比如 Pod 访问 Pod） | | **NodePort** | 集群外 | 通过任意 NodeIP:NodePort | 在每个 Node 上开放端口供外部访问 | | **LoadBalancer** | 集群外（云环境） | 云平台自动分配公网负载均衡 IP | 常用于阿里云、AWS 等公有云 | | **ExternalName** | 集群外部域名 | 把 Service 映射为外部域名（DNS CNAME） | 不真正代理流量，只是 DNS 映射 |   kubectl get endpo查看 Service 背后 **真实后端 Pod IP 列表**ints #查看 Service 背后 **真实后端 Pod IP 列表**      kubectl get pod -o wide #查看pod分布node+pod ip  每个 Pod 被 Kubernetes 分配一个 **唯一的 IP**（Pod IP）  集群内 Pod 之间直接访问（比如 10.244.1.5），即工作容器访问  kubectl get pod -o wide查看  Service 的内部 IP（ClusterIP）   当你创建一个 Service（默认 ClusterIP）时，Kubernetes 分配一个虚拟 IP   这个 IP **不是 Pod 的真实 IP**，而是 Service 的逻辑入口（集群内部访问入口）   Service IP 用于：   * 集群内部访问统一入口（curl http://nginx-service:80） * kube-proxy 通过 Endpoints 将请求转发到后端 Pod   **进入容器**  kubectl exec [options] <pod-name> [flags] -- <command>  kubectl exec -it <pod-name> -- /bin/bash  -it是进入，参数  多个container必须pod内指定container-c <container-name> 参数  --后面跟命令    **版本更新**  kubectl get pod -w #持续监控pod  kubectl set image deployment nginx-test nginx=nginx:1.21.5 –record 用于 **更新资源（Pod/Deployment/DaemonSet 等）的容器镜像,并记录这次更新历史，从而触发滚动更新保证服务不中断**  **滚动更新就是 Deployment 逐步替换旧 Pod 为新 Pod 的过程，触发滚动意味着模板变化导致 Deployment 开始按顺序更新 Pod，以保证服务不中断。**  Kubernetes 会自动触发 **滚动更新**：   1. 逐个删除旧 Pod 2. 创建新 Pod 使用新镜像 3. 保证应用可用性不下降   升级pod内containe升级：  实时监控pod状态  再开窗口：kubectl set image deployment nginx-test nginx=ngin:1.21.5 –record    **版本回滚**    kubectl rollout 是用来管理 **Deployment / DaemonSet / StatefulSet 的滚动更新（Rolling Update）和版本历史** 的命令集合   |  |  | | --- | --- | | **命令** | **作用** | | kubectl rollout status deployment <name> | 查看 Deployment 当前滚动更新的状态，判断更新是否完成 | | kubectl rollout history deployment <name> | 查看 Deployment 的更新历史（每次修改 Deployment 的模板都会记录） | | kubectl rollout undo deployment <name> | 回滚 Deployment 到上一个版本 | | kubectl rollout pause deployment <name> | 暂停 Deployment 的滚动更新 | | kubectl rollout resume deployment <name> | 恢复暂停的滚动更新 |   **滚动更新触发的方式**   * **滚动更新是由 Deployment 模板变化触发的**，比如：   + 镜像版本变化（kubectl set image deployment ...）   + 资源限制/环境变量/标签等变化 * 一旦模板变了，Deployment 会自动 **逐步替换 Pod** * kubectl rollout 只是用来 **查看/管理**这个过程   **查看资源详细信息**    **文件yaml创建资源对象**          模板：  apiVersion: apps/v1 # 顶层  kind: Deployment # 顶层  metadata: # 顶层  name: nginx-deploy # metadata 下的一级  labels: # metadata 下的一级  app: nginx # labels 下的二级  spec: # 顶层  replicas: 3 # spec 下的一级  selector: # spec 下的一级  matchLabels: # selector 下的一级  app: nginx # matchLabels 下的二级  template: # spec 下的一级  metadata: # template 下的一级  labels:  app: nginx  spec: # template 下的一级  containers: # spec 下的一级  - name: nginx # containers 是列表  image: nginx:1.21 # 列表项下的一级  ports:  - containerPort: 80    编写完yaml资源文件：    执行资源文件：  kubectl apply -f nginx-deployment.yml      资源文件创建pod资源对象：    资源文件创建deployment资源对象：    kubectl delete：删除 Kubernetes 资源  -f <文件或目录>：指定 YAML 文件或目录，里面定义了要删除的资源（Deployment、Service、ConfigMap 等）  **kubectl create**和**kubectl apply区别：**   |  |  |  |  | | --- | --- | --- | --- | | **命令** | **功能** | **特点** | **使用场景** | | kubectl create -f <file> | 根据 YAML/JSON 文件创建资源 | **只创建**，如果资源已存在会报错 | 第一次创建资源 | | kubectl apply -f <file> | 根据 YAML/JSON 文件创建或更新资源 | **创建或更新**（根据资源的 metadata.name 和 namespace 判断） | 想反复修改并应用配置时，持续管理资源 |   **kubectl create**  kubectl create -f deployment.yaml  |  +-- 检查资源是否存在  |  +-- 不存在 → 创建资源  +-- 已存在 → 报错  **kubectl apply**  kubectl apply -f deployment.yaml  |  +-- 检查资源是否存在  |  +-- 不存在 → 创建资源  +-- 已存在 → 比较当前状态与 YAML 文件  |  +-- 差异 → 更新资源（只改 YAML 定义的部分）  关键点：apply 使用 **声明式配置**（Declarative），记录上一次应用的配置（kubectl.kubernetes.io/last-applied-configuration annotation），便于增量更新；create 是 **命令式**，只会尝试创建。  所以说不用create  **Service资源文件**        **命令创建yaml模板**  **deployment**  kubectl create deployment tomcat --image=tomcat:8 -0 yaml --  dry-run=client > tomcat-deployment.yaml    -o yaml **输出格式**为 YAML（而不是直接创建资源）  --dry-run=client **模拟执行**，不会实际创建资源，只在客户端生成对象定义  > tomcat-deployment.yaml **重定向输出**到文件 tomcat-deployment.yaml   --dry-run=client：不提交到 API Server，客户端本地生成对象   -o yaml：把生成的 Deployment 对象以 YAML 格式输出   >：把输出写入 tomcat-deployment.yaml 文件  **service**  kubectl expose deployment tomcat --port=8080 --target-  port=8080 --type=NodePort -o yaml --dry-run=client >tomcat\_service.yaml    怎么看某个service有哪些pod？  **通过 label selector**  每个 Service 定义了一个 selector，Service 会把匹配这些标签的 Pod 作为后端。 假设 Service 名为 my-service：  kubectl get service my-service -o yaml #看service定义的selector部分  会看到类似：  spec:  selector:  app: myapp  然后根据 selector 查 Pod：  kubectl get pods -l app=myapp #根据laobel查询pod  这里 -l 表示按标签筛选 Pod。   Service 和 Pod 是通过 **selector（标签）** 关联的   查看方法：   1. kubectl get pods -l <selector> 2. kubectl get endpoints <service> #看的是pod的ip：端口 3. kubectl describe service <service> #找selector部分的标签，在用1找pod   怎么看某个deployment有哪些pod？  **通过 Pod 名称找 Deployment**  **方法一：根据前缀匹配**  如果你知道 Deployment 名为 nginx-deployment：  kubectl get pods | grep nginx-deployment  匹配出来的 Pod 通常就是 Deployment 创建的。  **方法二：通过 label selector**  更规范的方法是看 Deployment 的 selector，然后查 Pod：  kubectl get deployment nginx-deployment -o yaml  会看到类似：  spec:  selector:  matchLabels:  app: nginx  然后查 Pod：  kubectl get pods -l app=nginx  推荐这种方法，因为 **Pod 名称不一定完全依赖 Deployment 名**（Pod 名称可能因为 ReplicaSet 或者滚动更新而变化），标签才是官方可靠关联。  **方法三：kubectl describe**  kubectl describe deployment nginx-deployment  输出中会列出 **ReplicaSets**，通过 ReplicaSet 名称也可以找到 Pod。   **简单粗暴**：通过 Pod 名前缀匹配 Deployment 名   **官方推荐**：通过 Deployment 的 selector 标签查 Pod，更可靠 |