

# K8S 命令

---

[查询pod 及 services](#)

[重启服务](#)

[进入容器](#)

[获取 token](#)

[kubectl apply](#) - 以文件或标准输入为准应用或更新资源。

[kubectl get](#) - 列出一个或多个资源。

[kubectl describe](#) - 显示一个或多个资源的详细状态，默认情况下包括未初始化的资源。

[kubectl delete](#) - 从文件、stdin 或指定标签选择器、名称、资源选择器或资源中删除资源。

[kubectl exec](#) - 对 pod 中的容器执行命令。

[kubectl logs](#) - 打印 Pod 中容器的日志。

[查看和查找资源](#)

[更新资源](#)

[部分更新资源](#)

[对资源进行伸缩](#)

[删除资源](#)

[与运行中的 Pods 进行交互](#)

[与 Deployments 和 Services 进行交互](#)

[与节点和集群进行交互](#)

[资源类型](#)

## 查询pod 及 services

```
1 kubectl get pods,svc -n {nameSpace} -o wide
```

## 重启服务



Plain Text

复制代码

```
1  kubectl -n {namespave} rollout restart
2  deployments/{serviceName}
3
4  kubectl -n closeli rollout restart deployments/carproxy
```

## 进入容器



Plain Text

复制代码

```
1  kubectl exec -ti gamora-84779f949b-gk86m -n closeli -- /bin/bash
```

## 获取 token



Plain Text

复制代码

```
1  kubectl get secret -n kube-system | grep deploy-user
2  # deploy-user-token-vfdtf 为上一步找出来的对象名
3  kubectl describe secret deploy-user-token-vfdtf -n kube-system
```

**kubectl apply** – 以文件或标准输入为准应用或更新资源。



Plain Text

复制代码

```
1  # 使用 example-service.yaml 中的定义创建服务。
2  kubectl apply -f example-service.yaml
3
4  # 使用 example-controller.yaml 中的定义创建 replication controller。
5  kubectl apply -f example-controller.yaml
6
7  # 使用 <directory> 路径下的任意 .yaml, .yml, 或 .json 文件 创建对象。
8  kubectl apply -f <directory>
```

**kubectl get – 列出一个或多个资源。**

▼ Plain Text 复制代码

```
1 # 以纯文本输出格式列出所有 pod。
2 kubectl get pods
3
4 # 以纯文本输出格式列出所有 pod，并包含附加信息(如节点名)。
5 kubectl get pods -o wide
6
7 # 以纯文本输出格式列出具有指定名称的副本控制器。提示：你可以使用别名 'rc' 缩短和替换
  'replicationcontroller' 资源类型。
8 kubectl get replicationcontroller <rc-name>
9
10 # 以纯文本输出格式列出所有副本控制器和服务。
11 kubectl get rc,services
12
13 # 以纯文本输出格式列出所有守护程序集，包括未初始化的守护程序集。
14 kubectl get ds --include-uninitialized
15
16 # 列出在节点 server01 上运行的所有 pod
17 kubectl get pods --field-selector=spec.nodeName=server01
```

**kubectl describe – 显示一个或多个资源的详细状态，默认情况下包括未初始化的资源。**

▼ Plain Text 复制代码

```
1 # 显示名称为 <node-name> 的节点的详细信息。
2 kubectl describe nodes <node-name>
3
4 # 显示名为 <pod-name> 的 pod 的详细信息。
5 kubectl describe pods/<pod-name>
6
7 # 显示由名为 <rc-name> 的副本控制器管理的所有 pod 的详细信息。
8 # 记住：副本控制器创建的任何 pod 都以复制控制器的名称为前缀。
9 kubectl describe pods <rc-name>
10
11 # 描述所有的 pod，不包括未初始化的 pod
12 kubectl describe pods
```

**kubectl delete** – 从文件、stdin 或指定标签选择器、名称、资源选择器或资源中删除资源。



Plain Text

复制代码

```
1 # 使用 pod.yaml 文件中指定的类型和名称删除 pod。
2 kubectl delete -f pod.yaml
3
4 # 删除所有带有 '<label-key>=<label-value>' 标签的 Pod 和服务。
5 kubectl delete pods,services -l <label-key>=<label-value>
6
7 # 删除所有 pod，包括未初始化的 pod。
8 kubectl delete pods --all
```

**kubectl exec** – 对 pod 中的容器执行命令。



Plain Text

复制代码

```
1 # 从 pod <pod-name> 中获取运行 'date' 的输出。默认情况下，输出来自第一个容器。
2 kubectl exec <pod-name> -- date
3
4 # 运行输出 'date' 获取在容器的 <container-name> 中 pod <pod-name> 的输出。
5 kubectl exec <pod-name> -c <container-name> -- date
6
7 # 获取一个交互 TTY 并运行 /bin/bash <pod-name>。默认情况下，输出来自第一个容器。
8 kubectl exec -ti <pod-name> -- /bin/bash
```

**kubectl logs** – 打印 Pod 中容器的日志。



Plain Text

复制代码

```
1 # 从 pod 返回日志快照。
2 kubectl logs <pod-name>
3
4 # 从 pod <pod-name> 开始流式传输日志。这类似于 'tail -f' Linux 命令。
5 kubectl logs -f <pod-name>
```

**查看和查找资源**



```
1  # get 命令的基本输出
2  kubectl get services                # 列出当前命名空间下的所有
   services
3  kubectl get pods --all-namespaces   # 列出所有命名空间下的全部的
   Pods
4  kubectl get pods -o wide            # 列出当前命名空间下的全部
   Pods, 并显示更详细的信息
5  kubectl get deployment my-dep      # 列出某个特定的 Deployment
6  kubectl get pods                   # 列出当前命名空间下的全部 Pods
7  kubectl get pod my-pod -o yaml     # 获取一个 pod 的 YAML
8
9  # describe 命令的详细输出
10 kubectl describe nodes my-node
11 kubectl describe pods my-pod
12
13 # 列出当前名字空间下所有 Services, 按名称排序
14 kubectl get services --sort-by=.metadata.name
15
16 # 列出 Pods, 按重启次数排序
17 kubectl get pods --sort-by='.status.containerStatuses[0].restartCount'
18
19 # 列举所有 PV 持久卷, 按容量排序
20 kubectl get pv --sort-by=.spec.capacity.storage
21
22 # 获取包含 app=cassandra 标签的所有 Pods 的 version 标签
23 kubectl get pods --selector=app=cassandra -o \
24     jsonpath='{.items[*].metadata.labels.version}'
25
26 # 检索带有 "." 键值, 例: 'ca.crt'
27 kubectl get configmap myconfig \
28     -o jsonpath='{.data.ca\.crt}'
29
30 # 获取所有工作节点 (使用选择器以排除标签名称为 'node-role.kubernetes.io/master'
   的结果)
31 kubectl get node --selector='!node-role.kubernetes.io/master'
32
33 # 获取当前命名空间中正在运行的 Pods
34 kubectl get pods --field-selector=status.phase=Running
35
36 # 获取全部节点的 ExternalIP 地址
37 kubectl get nodes -o jsonpath='{.items[*].status.addresses[?
   (@.type=="ExternalIP")].address}'
38
39 # 列出属于某个特定 RC 的 Pods 的名称
```

```

40 # 在转换对于 jsonpath 过于复杂的场合, "jq" 命令很有用; 可以在
    https://stedolan.github.io/jq/ 找到它。
41 sel=${$(kubectl get rc my-rc --output=json | jq -j '.spec.selector |
    to_entries | .[] | "\(.key)=\(.value),"'%?)}
42 echo $(kubectl get pods --selector=$sel --output=jsonpath=
    {.items..metadata.name})
43
44 # 显示所有 Pods 的标签 (或任何其他支持标签的 Kubernetes 对象)
45 kubectl get pods --show-labels
46
47 # 检查哪些节点处于就绪状态
48 JSONPATH='{range .items[*]}{@.metadata.name}:{range
    @.status.conditions[*]}{@.type}={@.status};{end}{end}' \
49 && kubectl get nodes -o jsonpath="$JSONPATH" | grep "Ready=True"
50
51 # 不使用外部工具来输出解码后的 Secret
52 kubectl get secret my-secret -o go-template='{range $k,$v := .data}}
    {"### "}}{{ $k }}{{ "\n" }}{{ $v|base64decode }}{{ "\n\n" }}{{ end }}'
53
54 # 列出被一个 Pod 使用的全部 Secret
55 kubectl get pods -o json | jq
    '.items[].spec.containers[].env[]?.valueFrom.secretKeyRef.name' | grep -v
    null | sort | uniq
56
57 # 列举所有 Pods 中初始化容器的容器 ID (containerID)
58 # 可用于在清理已停止的容器时避免删除初始化容器
59 kubectl get pods --all-namespaces -o jsonpath='{range
    .items[*].status.initContainerStatuses[*]}{@.containerID}{{ "\n" }}{{ end }}' |
    cut -d/ -f3
60
61 # 列出事件 (Events), 按时间戳排序
62 kubectl get events --sort-by=.metadata.creationTimestamp
63
64 # 比较当前的集群状态和假定某清单被应用之后的集群状态
65 kubectl diff -f ./my-manifest.yaml
66
67 # 生成一个句点分隔的树, 其中包含为节点返回的所有键
68 # 在复杂的嵌套JSON结构中定位键时非常有用
69 kubectl get nodes -o json | jq -c 'path(..)|[.[]|tostring]|join(".")'
70
71 # 生成一个句点分隔的树, 其中包含为pod等返回的所有键
72 kubectl get pods -o json | jq -c 'path(..)|[.[]|tostring]|join(".")'
73
74 # 假设你的 Pods 有默认的容器和默认的名字空间, 并且支持 'env' 命令, 可以使用以下脚本为
    所有 Pods 生成 ENV 变量。
75 # 该脚本也可用于在所有的 Pods 里运行任何受支持的命令, 而不仅仅是 'env'。
76 for pod in $(kubectl get po --output=jsonpath={.items..metadata.name});
    do echo $pod && kubectl exec -it $pod env; done

```

## 更新资源

		Plain Text	复制代码
1	kubectl set image deployment/frontend www=image:v2	# 滚动更新 "frontend" Deployment 的 "www" 容器镜像	
2	kubectl rollout history deployment/frontend	Deployment 的历史记录, 包括版本	# 检查
3	kubectl rollout undo deployment/frontend	上次部署版本	# 回滚到
4	kubectl rollout undo deployment/frontend --to-revision=2	特定部署版本	# 回滚到
5	kubectl rollout status -w deployment/frontend	"frontend" Deployment 的滚动升级状态直到完成	# 监视
6	kubectl rollout restart deployment/frontend	启 "frontend" Deployment	# 轮替重
7			
8	cat pod.json   kubectl replace -f -	入到标准输入的 JSON 来替换 Pod	# 通过传
9			
10	# 强制替换, 删除后重建资源。会导致服务不可用。		
11	kubectl replace --force -f ./pod.json		
12			
13	# 为多副本的 nginx 创建服务, 使用 80 端口提供服务, 连接到容器的 8000 端口。		
14	kubectl expose rc nginx --port=80 --target-port=8000		
15			
16	# 将某单容器 Pod 的镜像版本 (标签) 更新到 v4		
17	kubectl get pod mypod -o yaml   sed 's/\(image: myimage\):.*\$/\1:v4/'   kubectl replace -f -		
18			
19	kubectl label pods my-pod new-label=awesome	签	# 添加标
20	kubectl annotate pods my-pod icon-url=http://goo.gl/XXBTWq	解	# 添加注
21	kubectl autoscale deployment foo --min=2 --max=10	"foo" Deployment 自动伸缩容	# 对

## 部分更新资源



```

1  # 部分更新某节点
2  kubectl patch node k8s-node-1 -p '{"spec":{"unschedulable":true}}'
3
4  # 更新容器的镜像; spec.containers[*].name 是必须的。因为它是一个合并性质的主键。
5  kubectl patch pod valid-pod -p '{"spec":{"containers":
6    [{"name":"kubernetes-serve-hostname","image":"new image"}]}}'
7
8  # 使用带位置数组的 JSON patch 更新容器的镜像
9  kubectl patch pod valid-pod --type='json' -p='[{"op": "replace", "path":
10    "/spec/containers/0/image", "value":"new image"}]'
11
12 # 使用带位置数组的 JSON patch 禁用某 Deployment 的 livenessProbe
13 kubectl patch deployment valid-deployment --type json -p='[{"op":
14   "remove", "path": "/spec/template/spec/containers/0/livenessProbe"}]'
15
16 # 在带位置数组中添加元素
17 kubectl patch sa default --type='json' -p='[{"op": "add", "path":
18   "/secrets/1", "value": {"name": "whatever" } }]'
19
20 编辑资源

```

## 对资源进行伸缩

```

1  kubectl scale --replicas=3 rs/foo # 将名为
   'foo' 的副本集伸缩到 3 副本
2  kubectl scale --replicas=3 -f foo.yaml # 将在
   "foo.yaml" 中的特定资源伸缩到 3 个副本
3  kubectl scale --current-replicas=2 --replicas=3 deployment/mysql # 如果名
   为 mysql 的 Deployment 的副本当前是 2, 那么将它伸缩到 3
4  kubectl scale --replicas=5 rc/foo rc/bar rc/baz # 伸缩多
   个副本控制器

```

## 删除资源

```
1  kubectl delete -f ./pod.json
   # 删除在 pod.json 中指定的类型和名称的 Pod
2  kubectl delete pod,service baz foo
   # 删除名称为 "baz" 和 "foo" 的 Pod 和服务
3  kubectl delete pods,services -l name=myLabel
   # 删除包含 name=myLabel 标签的 pods 和服务
4  kubectl -n my-ns delete pod,svc --all
   # 删除在 my-ns 名字空间中全部的 Pods 和服务
5  # 删除所有与 pattern1 或 pattern2 awk 模式匹配的 Pods
6  kubectl get pods -n mynamespace --no-headers=true | awk
   '/pattern1|pattern2/{print $1}' | xargs kubectl delete -n mynamespace
   pod
```

## 与运行中的 Pods 进行交互

```

1  kubectl logs my-pod # 获取 pod 日志（标准输出）
2  kubectl logs -l name=myLabel # 获取含 name=myLabel
   标签的 Pods 的日志（标准输出）
3  kubectl logs my-pod --previous # 获取上个容器实例的 pod
   日志（标准输出）
4  kubectl logs my-pod -c my-container # 获取 Pod 容器的日志
   （标准输出，多容器场景）
5  kubectl logs -l name=myLabel -c my-container # 获取含 name=myLabel
   标签的 Pod 容器日志（标准输出，多容器场景）
6  kubectl logs my-pod -c my-container --previous # 获取 Pod 中某容器的上
   个实例的日志（标准输出，多容器场景）
7  kubectl logs -f my-pod # 流式输出 Pod 的日志
   （标准输出）
8  kubectl logs -f my-pod -c my-container # 流式输出 Pod 容器的日
   志（标准输出，多容器场景）
9  kubectl logs -f -l name=myLabel --all-containers # 流式输出含
   name=myLabel 标签的 Pod 的所有日志（标准输出）
10 kubectl run -i --tty busybox --image=busybox -- sh # 以交互式 Shell 运行
    Pod
11 kubectl run nginx --image=nginx -n mynamespace # 在指定名字空间中运行
    nginx Pod
12 kubectl run nginx --image=nginx # 运行 ngins Pod 并将其
    规约写入到名为 pod.yaml 的文件
13   --dry-run=client -o yaml > pod.yaml
14
15 kubectl attach my-pod -i # 挂接到一个运行的容器中
16 kubectl port-forward my-pod 5000:6000 # 在本地计算机上侦听端口
    5000 并转发到 my-pod 上的端口 6000
17 kubectl exec my-pod -- ls / # 在已有的 Pod 中运行命
    令（单容器场景）
18 kubectl exec --stdin --tty my-pod -- /bin/sh # 使用交互 shell 访问正
    在运行的 Pod（一个容器场景）
19 kubectl exec my-pod -c my-container -- ls / # 在已有的 Pod 中运行命
    令（多容器场景）
20 kubectl top pod POD_NAME --containers # 显示给定 Pod 和其中容
    器的监控数据
21 kubectl top pod POD_NAME --sort-by=cpu # 显示给定 Pod 的指标并
    且按照 'cpu' 或者 'memory' 排序

```

## 与 Deployments 和 Services 进行交互

▼ Plain Text 复制代码

```
1  kubectl logs deploy/my-deployment          # 获取一个
   Deployment 的 Pod 的日志（单容器例子）
2  kubectl logs deploy/my-deployment -c my-container # 获取一个
   Deployment 的 Pod 的日志（多容器例子）
3
4  kubectl port-forward svc/my-service 5000      # 侦听本地端口
   5000 并转发到 Service 后端端口 5000
5  kubectl port-forward svc/my-service 5000:my-service-port # 侦听本地端口
   5000 并转发到名字为 <my-service-port> 的 Service 目标端口
6
7  kubectl port-forward deploy/my-deployment 5000:6000 # 侦听本地端口
   5000 并转发到 <my-deployment> 创建的 Pod 里的端口 6000
8  kubectl exec deploy/my-deployment -- ls        # 在 Deployment
   里的第一个 Pod 的第一个容器里运行命令（单容器和多容器例子）
```

与节点和集群进行交互

▼ Plain Text 复制代码

```
1  kubectl cordon my-node                      #
   标记 my-node 节点为不可调度
2  kubectl drain my-node                      #
   对 my-node 节点进行清空操作，为节点维护做准备
3  kubectl uncordon my-node                   #
   标记 my-node 节点为可以调度
4  kubectl top node my-node                   #
   显示给定节点的度量值
5  kubectl cluster-info                      #
   显示主控节点和服务的地址
6  kubectl cluster-info dump                 #
   将当前集群状态转储到标准输出
7  kubectl cluster-info dump --output-directory=/path/to/cluster-state #
   将当前集群状态输出到 /path/to/cluster-state
8
9  # 如果已存在具有指定键和效果的污点，则替换其值为指定值。
10 kubectl taint nodes foo dedicated=special-user:NoSchedule
```

资源类型

```
1  # 列出所支持的全部资源类型和它们的简称、API 组，是否是名字空间作用域 和 Kind。
2  kubectl api-resources
3
4  # 用于探索 API 资源的其他操作
5  kubectl api-resources --namespaced=true      # 所有命名空间作用域的资源
6  kubectl api-resources --namespaced=false     # 所有非命名空间作用域的资源
7  kubectl api-resources -o name                # 用简单格式列举所有资源（仅显示
   资源名称）
8  kubectl api-resources -o wide                # 用扩展格式列举所有资源（又称
   "wide" 格式）
9  kubectl api-resources --verbs=list,get       # 支持 "list" 和 "get" 请求动
   词的所有资源
10 kubectl api-resources --api-group=extensions # "extensions" API 组中的所有
   资源
```