### **kubectl 常用命令指南**

Kubectl 命令是操作 kubernetes 集群的最直接的方式，特别是运维人员，需要对这些命令有一个详细的掌握

### **Kubectl 自动补全**

|  |
| --- |
| yum install bash-completion  source /usr/share/bash-completion/bash\_completion      #临时生效  source <(kubectl completion bash)    #永久生效  echo 'source <(kubectl completion bash)' >> ~/.bashrc  source ~/.bashrc  kubectl设置别名k自动补全：  echo 'alias k=kubectl' >>~/.bashrc  echo 'complete -F \_\_start\_kubectl k' >> ~/.bashrc  source ~/.bashrc |

### **Kubectl 上下文和配置**

设置 kubectl 命令交互的 kubernetes 集群并修改配置信息。参阅 使用 kubeconfig 文件进行跨集群验证 获取关于配置文件的详细信息。

|  |
| --- |
| # 显示合并后的 kubeconfig 配置  $ kubectl config view  # 同时使用多个 kubeconfig 文件并查看合并后的配置  $ KUBECONFIG=~/.kube/config:~/.kube/kubconfig2 kubectl config view  # 获取 e2e 用户的密码  $ kubectl config view -o jsonpath='{.users[?(@.name == "e2e")].user.password}'  # 显示当前的上下文  $ kubectl config current-context  # 设置默认上下文为 my-cluster-name  $ kubectl config use-context my-cluster-name  # 向 kubeconf 中增加支持基本认证的新集群  $ kubectl config set-credentials kubeuser/foo.kubernetes.com --username=kubeuser --password=kubepassword  # 使用指定的用户名和 namespace 设置上下文  $ kubectl config set-context gce --user=cluster-admin --namespace=foo \  && kubectl config use-context gce |

### **创建对象**

Kubernetes 的清单文件可以使用 json 或 yaml 格式定义。可以以 .yaml、.yml、或者 .json 为扩展名。

|  |
| --- |
| # 创建资源  $ kubectl create -f ./my-manifest.yaml  # 使用多个文件创建资源  $ kubectl create -f ./my1.yaml -f ./my2.yaml  # 使用目录下的所有清单文件来创建资源  $ kubectl create -f ./dir  # 使用 url 来创建资源  $ kubectl create -f https://git.io/vPieo  # 启动一个 nginx 实例  $ kubectl run nginx --image=nginx  # 获取 pod 和 svc 的文档  $ kubectl explain pods,svc  # 从 stdin 输入中创建多个 YAML 对象  $ cat <<EOF | kubectl create -f -  apiVersion: v1  kind: Pod  metadata:  name: busybox-sleep  spec:  containers:  - name: busybox  image: busybox  args:  - sleep  - "1000000"  ---  apiVersion: v1  kind: Pod  metadata:  name: busybox-sleep-less  spec:  containers:  - name: busybox  image: busybox  args:  - sleep  - "1000"  EOF  # 创建包含几个 key 的 Secret  $ cat <<EOF | kubectl create -f -  apiVersion: v1  kind: Secret  metadata:  name: mysecret  type: Opaque  data:  password: $(echo "s33msi4" | base64)  username: $(echo "jane" | base64)  EOF |

### **显示和查找资源**

|  |
| --- |
| # Get commands with basic output  # 列出所有 namespace 中的所有 service  $ kubectl get services  # 列出所有 namespace 中的所有 pod  $ kubectl get pods --all-namespaces  # 列出所有 pod 并显示详细信息  $ kubectl get pods -o wide  # 列出指定 deployment  $ kubectl get deployment my-dep  # 列出该 namespace 中的所有 pod 包括未初始化的  $ kubectl get pods --include-uninitialized  # 使用详细输出来描述命令  $ kubectl describe nodes my-node  $ kubectl describe pods my-pod  # List Services Sorted by Name  $ kubectl get services --sort-by=.metadata.name  # 根据重启次数排序列出 pod  $ kubectl get pods --sort-by='.status.containerStatuses[0].restartCount'  # 获取所有具有 app=cassandra 的 pod 中的 version 标签  $ kubectl get pods --selector=app=cassandra rc -o \  jsonpath='{.items[\*].metadata.labels.version}'  # 获取所有节点的 ExternalIP  $ kubectl get nodes -o jsonpath='{.items[\*].status.addresses[?(@.type=="ExternalIP")].address}'  # 列出属于某个 PC 的 Pod 的名字  # “jq”命令用于转换复杂的 jsonpath，参考 https://stedolan.github.io/jq/  $ sel=${$(kubectl get rc my-rc --output=json | jq -j '.spec.selector | to\_entries | .[] | "\(.key)=\(.value),"')%?}  $ echo $(kubectl get pods --selector=$sel --output=jsonpath={.items..metadata.name})  # 查看哪些节点已就绪  $ JSONPATH='{range .items[\*]}{@.metadata.name}:{range @.status.conditions[\*]}{@.type}={@.status};{end}{end}' \  && kubectl get nodes -o jsonpath="$JSONPATH" | grep "Ready=True"  # 列出当前 Pod 中使用的 Secret  $ kubectl get pods -o json | jq '.items[].spec.containers[].env[]?.valueFrom.secretKeyRef.name' | grep -v null | sort | uniq |

### **更新资源**

|  |
| --- |
| $ kubectl rolling-update frontend-v1 -f frontend-v2.json # 滚动更新 pod frontend-v1  $ kubectl rolling-update frontend-v1 frontend-v2 --image=image:v2 # 更新资源名称并更新镜像  $ kubectl rolling-update frontend --image=image:v2 # 更新 frontend pod 中的镜像  $ kubectl rolling-update frontend-v1 frontend-v2 --rollback # 退出已存在的进行中的滚动更新  $ cat pod.json | kubectl replace -f - # 基于 stdin 输入的 JSON 替换 pod  # 强制替换，删除后重新创建资源。会导致服务中断。  $ kubectl replace --force -f ./pod.json  # 为 nginx RC 创建服务，启用本地 80 端口连接到容器上的 8000 端口  $ kubectl expose rc nginx --port=80 --target-port=8000  # 更新单容器 pod 的镜像版本（tag）到 v4  $ kubectl get pod mypod -o yaml | sed 's/\(image: myimage\):.\*$/\1:v4/' | kubectl replace -f -  # 添加标签  $ kubectl label pods my-pod new-label=awesome  # 添加注解  $ kubectl annotate pods my-pod icon-url=http://goo.gl/XXBTWq  # 自动扩展 deployment “foo”  $ kubectl autoscale deployment foo --min=2 --max=10 |

### **修补资源**

使用策略合并补丁并修补资源。

|  |
| --- |
| # 部分更新节点  kubectl patch node k8s-node-1 -p '{"spec":{"unschedulable":true}}'  # 更新容器镜像；spec.containers[\*].name 是必须的，因为这是合并的关键字  $ kubectl patch pod valid-pod -p '{"spec":{"containers":[{"name":"kubernetes-serve-hostname","image":"new image"}]}}'  # 使用具有位置数组的 json 补丁更新容器镜像  $ kubectl patch pod valid-pod --type='json' -p='[{"op": "replace", "path": "/spec/containers/0/image", "value":"new image"}]'  # 使用具有位置数组的 json 补丁禁用 deployment 的 livenessProbe  $ kubectl patch deployment valid-deployment --type json -p='[{"op": "remove", "path": "/spec/template/spec/containers/0/livenessProbe"}]' |

### **编辑资源**

在编辑器中编辑任何 API 资源

|  |
| --- |
| # 编辑名为 docker-registry 的 service  $ kubectl edit svc/docker-registry  # 使用其它编辑器  $ KUBE\_EDITOR="nano" kubectl edit svc/docker-registry |

### **Scale 资源**

|  |
| --- |
| # Scale a replicaset named 'foo' to 3  $ kubectl scale --replicas=3 rs/foo  # Scale a resource specified in "foo.yaml" to 3  $ kubectl scale --replicas=3 -f foo.yaml  # If the deployment named mysql's current size is 2, scale mysql to 3  $ kubectl scale --current-replicas=2 --replicas=3 deployment/mysql  # Scale multiple replication controllers  $ kubectl scale --replicas=5 rc/foo rc/bar rc/baz |

### **删除资源**

|  |
| --- |
| # 删除 pod.json 文件中定义的类型和名称的 pod  $ kubectl delete -f ./pod.json  # 删除名为“baz”的 pod 和名为“foo”的 service  $ kubectl delete pod,service baz foo  # 删除具有 name=myLabel 标签的 pod 和 serivce  $ kubectl delete pods,services -l name=myLabel  # 删除具有 name=myLabel 标签的 pod 和 service，包括尚未初始化的  $ kubectl delete pods,services -l name=myLabel --include-uninitialized  # 删除 my-ns namespace 下的所有 pod 和 serivce包  $ kubectl -n my-ns delete po,svc --all |

### **与运行中的 Pod 交互**

|  |
| --- |
| # dump 输出 pod 的日志（stdout）  $ kubectl logs my-pod  # dump 输出 pod 中容器的日志（stdout，pod 中有多个容器的情况下使用）  $ kubectl logs my-pod -c my-container  # 流式输出 pod 的日志（stdout）  $ kubectl logs -f my-pod  # 流式输出 pod 中容器的日志（stdout，pod 中有多个容器的情况下使用）  $ kubectl logs -f my-pod -c my-container  # 交互式 shell 的方式运行 pod  $ kubectl run -i --tty busybox --image=busybox -- sh  # 连接到运行中的容器  $ kubectl attach my-pod -i  # 转发 pod 中的 6000 端口到本地的 5000 端口  $ kubectl port-forward my-pod 5000:6000  # 在已存在的容器中执行命令（只有一个容器的情况下）  $ kubectl exec my-pod -- ls /  # 在已存在的容器中执行命令（pod 中有多个容器的情况下）  $ kubectl exec my-pod -c my-container -- ls /  # 显示指定 pod 和容器的指标度量  $ kubectl top pod POD\_NAME --containers |

### **与节点和集群交互**

|  |
| --- |
| # 标记 my-node 不可调度  $ kubectl cordon my-node  # 清空 my-node 以待维护  $ kubectl drain my-node  # 标记 my-node 可调度  $ kubectl uncordon my-node  # 显示 my-node 的指标度量  $ kubectl top node my-node  $ kubectl cluster-info  # 将当前集群状态输出到 stdout  $ kubectl cluster-info dump  # 将当前集群状态输出到 /path/to/cluster-state  $ kubectl cluster-info dump --output-directory=/path/to/cluster-state  # 如果该键和影响的污点（taint）已存在，则使用指定的值替换  $ kubectl taint nodes foo dedicated=special-user:NoSchedule |

#### **set 命令**

配置应用的一些特定资源，也可以修改应用已有的资源

使用 kubectl set --help查看，它的子命令，env，image，resources，selector，serviceaccount，subject。

语法：resources (-f FILENAME | TYPE NAME) ([—limits=LIMITS & —requests=REQUESTS]

#### **kubectl set resources 命令**

这个命令用于设置资源的一些范围限制。资源对象中的Pod可以指定计算资源需求（CPU-单位m、内存-单位Mi），即使用的最小资源请求（Requests），限制（Limits）的最大资源需求，Pod将保证使用在设置的资源数量范围。对于每个Pod资源，如果指定了Limits（限制）值，并省略了Requests（请求），则Requests默认为Limits的值。

|  |
| --- |
| 可用资源对象包括(支持大小写)：replicationcontroller、deployment、daemonset、job、replicaset。 |

例如：

|  |
| --- |
| # 将deployment的nginx容器cpu限制为“200m”，将内存设置为“512Mi”  $ kubectl set resources deployment nginx -c=nginx --limits=cpu=200m,memory=512Mi  # 设置所有nginx容器中 Requests和Limits  $ kubectl set resources deployment nginx --limits=cpu=200m,memory=512Mi --requests=cpu=100m,memory=256Mi  # 删除nginx中容器的计算资源值  $ kubectl set resources deployment nginx --limits=cpu=0,memory=0 --requests=cpu=0,memory=0 |

#### **kubectl set selector 命令**

设置资源的 selector（选择器）。如果在调用”set selector”命令之前已经存在选择器，则新创建的选择器将覆盖原来的选择器。

selector必须以字母或数字开头，最多包含63个字符，可使用：字母、数字、连字符” - “ 、点”.”和下划线” \_ “。如果指定了—resource-version，则更新将使用此资源版本，否则将使用现有的资源版本。

注意：目前selector命令只能用于Service对象。

|  |
| --- |
| 语法：selector (-f FILENAME | TYPE NAME) EXPRESSIONS [—resource-version=version] |

#### **kubectl set image 命令**

用于更新现有资源的容器镜像。

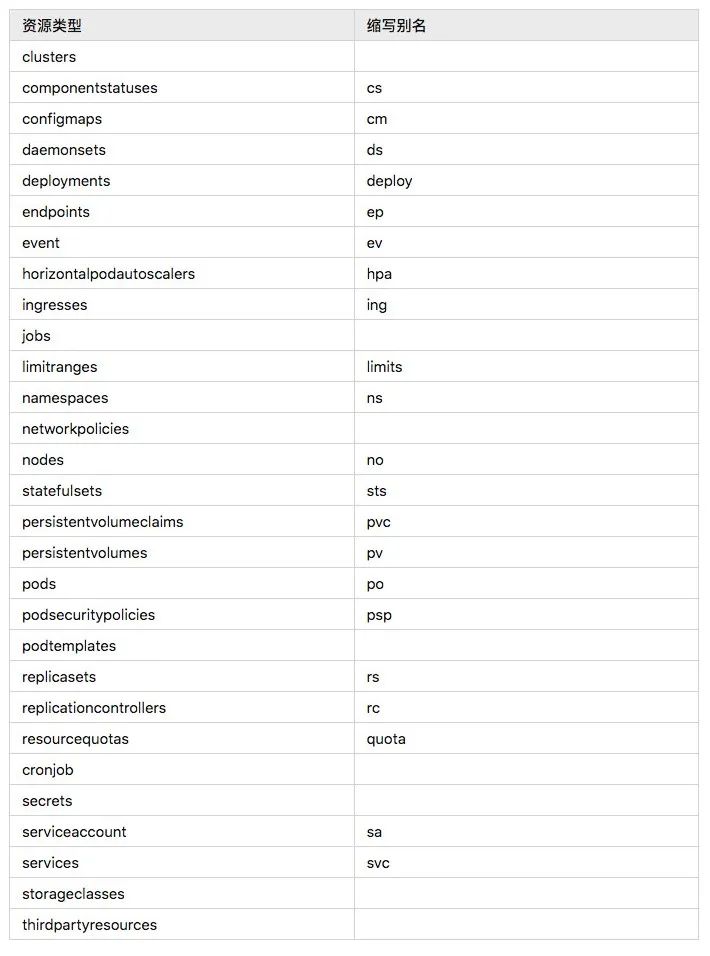
可用资源对象包括：pod (po)、replicationcontroller (rc)、deployment (deploy)、daemonset (ds)、job、replicaset (rs)。

|  |
| --- |
| 语法：image (-f FILENAME | TYPE NAME) CONTAINER\_NAME\_1=CONTAINER\_IMAGE\_1 … CONTAINER\_NAME\_N=CONTAINER\_IMAGE\_N |

|  |
| --- |
| # 将deployment中的nginx容器镜像设置为“nginx：1.9.1”  $ kubectl set image deployment/nginx busybox=busybox nginx=nginx:1.9.1  # 所有deployment和rc的nginx容器镜像更新为“nginx：1.9.1”  $ kubectl set image deployments,rc nginx=nginx:1.9.1 --all  # 将daemonset abc的所有容器镜像更新为“nginx：1.9.1”  $ kubectl set image daemonset abc \*=nginx:1.9.1  # 从本地文件中更新nginx容器镜像  $ kubectl set image -f path/to/file.yaml nginx=nginx:1.9.1 --local -o yaml |

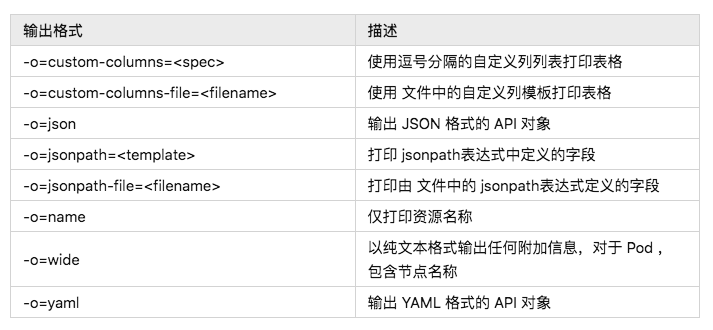
### **资源类型**

下表列出的是 kubernetes 中所有支持的类型和缩写的别名



### **格式化输出**

要以特定的格式向终端窗口输出详细信息，可以在 kubectl 命令中添加 -o 或者 -output 标志。



**Kubectl 详细输出和调试**



使用 -v 或 --v 标志跟着一个整数来指定日志级别

来源：https://zhuanlan.zhihu.com/p/394967586