你可以使用 Kubectl 命令行工具管理 Kubernetes 集群。
`kubectl` 在 `$HOME/.kube` 目录中查找一个名为 `config` 的配置文件。
你可以通过设置 KUBECONFIG 环境变量或设置
`--kubeconfig`
参数来指定其它 kubeconfig 文件。

本文概述了 `kubectl` 语法和命令操作描述，并提供了常见的示例。
有关每个命令的详细信息，包括所有受支持的参数和子命令，
请参阅 kubectl 参考文档。
有关安装说明，请参见安装 kubectl 。

语法

使用以下语法 `kubectl` 从终端窗口运行命令：

```
kubectl [command] [TYPE] [NAME] [flags]
```

其中 `command`、`TYPE`、`NAME` 和 `flags` 分别是：

- `command`：指定要对一个或多个资源执行的操作，例如 `create`、`get`、`describe`、`delete`。
- `TYPE`：指定资源类型。资源类型不区分大小写，
  可以指定单数、复数或缩写形式。例如，以下命令输出相同的结果:

```
kubectl get pod pod1
kubectl get pods pod1
kubectl get po pod1
```

- `NAME`：指定资源的名称。名称区分大小写。
  如果省略名称，则显示所有资源的详细信息 `kubectl get pods`。

  在对多个资源执行操作时，你可以按类型和名称指定每个资源，或指定一个或多个文件：

  - 要按类型和名称指定资源：
    - 要对所有类型相同的资源进行分组，请执行以下操作：`TYPE1 name1 name2 name<#>`。

      例子：`kubectl get pod example-pod1 example-pod2`
    - 分别指定多个资源类型：`TYPE1/name1 TYPE1/name2 TYPE2/name3 TYPE<#>/name<#>`。

      例子：`kubectl get pod/example-pod1 replicationcontroller/example-rc1`
  - 用一个或多个文件指定资源：`-f file1 -f file2 -f file<#>`
    - 使用 YAML 而不是 JSON
      因为 YAML 更容易使用，特别是用于配置文件时。
      例子：`kubectl get -f ./pod.yaml`
- `flags`：指定可选的参数。例如，可以使用 `-s` 或 `-server` 参数指定 Kubernetes API 服务器的地址和端口。

> 从命令行指定的参数会覆盖默认值和任何相应的环境变量。

如果你需要帮助，从终端窗口运行 `kubectl help`。

操作

下表包含所有 kubectl 操作的简短描述和普通语法：

操作	描述
`alpha`	列出与 alpha 特性对应的可用命令，这些特性在 Kubernetes 集群中默认情况下是不启用的。
`annotate`	添加或更新一个或多个资源的注解。
`api-resources`	列出可用的 API 资源。
`api-versions`	列出可用的 API 版本。
`apply`	从文件或 stdin 对资源应用配置更改。
`attach`	附加到正在运行的容器，查看输出流或与容器（stdin）交互。
`auth`	检查授权。
`autoscale`	自动伸缩由副本控制器管理的一组 pod。
`certificate`	修改证书资源。
`cluster-info`	显示有关集群中主服务器和服务的端口信息。
`completion`	为指定的 shell （bash 或 zsh）输出 shell 补齐代码。
`config`	修改 kubeconfig 文件。有关详细信息，请参阅各个子命令。
`convert`	在不同的 API 版本之间转换配置文件。配置文件可以是 YAML 或 JSON 格式。
`cordon`	将节点标记为不可调度。
`cp`	在容器之间复制文件和目录。
`create`	从文件或 stdin 创建一个或多个资源。
`delete`	从文件、标准输入或指定标签选择器、名称、资源选择器或资源中删除资源。
`describe`	显示一个或多个资源的详细状态。
`diff`	将 live 配置和文件或标准输入做对比 (**BETA**)
`drain`	腾空节点以准备维护。
`edit`	使用默认编辑器编辑和更新服务器上一个或多个资源的定义。
`exec`	对 pod 中的容器执行命令。
`explain`	获取多种资源的文档。例如 pod, node, service 等。
`expose`	将副本控制器、服务或 pod 作为新的 Kubernetes 服务暴露。
`get`	列出一个或多个资源。

操作	描述
`kustomize`	列出从 kustomization.yaml 文件中的指令生成的一组 API 资源。参数必须是包含文件的目录的路径，或者是 git 存储库 URL，其路径后缀相对于存储库根目录指定了相同的路径。
`label`	添加或更新一个或多个资源的标签。
`logs`	在 pod 中打印容器的日志。
`options`	全局命令行选项列表，适用于所有命令。
`patch`	使用策略合并 patch 程序更新资源的一个或多个字段。
`plugin`	提供用于与插件交互的实用程序。
`port-forward`	将一个或多个本地端口转发到一个 pod。
`proxy`	运行 Kubernetes API 服务器的代理。
`replace`	从文件或标准输入中替换资源。
`rollout`	管理资源的部署。有效的资源类型包括：Deployments, DaemonSets 和 StatefulSets。
`run`	在集群上运行指定的镜像。
`scale`	更新指定副本控制器的大小。
`set`	配置应用程序资源。
`taint`	更新一个或多个节点上的污点。
`top`	显示资源（CPU/内存/存储）的使用情况。
`uncordon`	将节点标记为可调度。
`version`	显示运行在客户端和服务器上的 Kubernetes 版本。
`wait`	实验性：等待一种或多种资源的特定条件。

了解更多有关命令操作的信息，请参阅 kubectl 参考文档。

资源类型

下表列出所有受支持的资源类型及其缩写别名:

(以下输出可以通过 `kubectl api-resources` 获取，内容以 Kubernetes 1.19.1 版本为准。)

资源名	缩写名	API 分组	按命名空间	资源类型
bindings			true	Binding
componentstatuses	cs		false	ComponentStatus
configmaps	cm		true	ConfigMap
endpoints	ep		true	Endpoints
events	ev		true	Event
limitranges	limits		true	LimitRange
namespaces	ns		false	Namespace
nodes	no		false	Node
persistentvolumeclaims	pvc		true	PersistentVolumeClaim
persistentvolumes	pv		false	PersistentVolume
pods	po		true	Pod
podtemplates			true	PodTemplate
replicationcontrollers	rc		true	ReplicationController
resourcequotas	quota		true	ResourceQuota
secrets			true	Secret
serviceaccounts	sa		true	ServiceAccount
services	svc		true	Service
mutatingwebhookconfigurations		admissionregistration.k8s.io	false	MutatingWebhookConfiguration
validatingwebhookconfigurations		admissionregistration.k8s.io	false	ValidatingWebhookConfiguration
customresourcedefinitions	crd,crds	apiextensions.k8s.io	false	CustomResourceDefinition
apiservices		apiregistration.k8s.io	false	APIService
controllerrevisions		apps	true	ControllerRevision
daemonsets	ds	apps	true	DaemonSet
deployments	deploy	apps	true	Deployment
replicasets	rs	apps	true	ReplicaSet
statefulsets	sts	apps	true	StatefulSet
tokenreviews		authentication.k8s.io	false	TokenReview
localsubjectaccessreviews		authorization.k8s.io	true	LocalSubjectAccessReview
selfsubjectaccessreviews		authorization.k8s.io	false	SelfSubjectAccessReview
selfsubjectrulesreviews		authorization.k8s.io	false	SelfSubjectRulesReview
subjectaccessreviews		authorization.k8s.io	false	SubjectAccessReview
horizontalpodautoscalers	hpa	autoscaling	true	HorizontalPodAutoscaler
cronjobs	cj	batch	true	CronJob
jobs		batch	true	Job
certificatesigningrequests	csr	certificates.k8s.io	false	CertificateSigningRequest
leases		coordination.k8s.io	true	Lease
endpointslices		discovery.k8s.io	true	EndpointSlice
events	ev	events.k8s.io	true	Event

资源名	缩写名	API 分组	按命名空间	资源类型
ingresses	ing	extensions	true	Ingress
flowschemas		flowcontrol.apiserver.k8s.io	false	FlowSchema
prioritylevelconfigurations		flowcontrol.apiserver.k8s.io	false	PriorityLevelConfiguration
ingressclasses		networking.k8s.io	false	IngressClass
ingresses	ing	networking.k8s.io	true	Ingress
networkpolicies	netpol	networking.k8s.io	true	NetworkPolicy
runtimeclasses		node.k8s.io	false	RuntimeClass
poddisruptionbudgets	pdb	policy	true	PodDisruptionBudget
podsecuritypolicies	psp	policy	false	PodSecurityPolicy
clusterrolebindings		rbac.authorization.k8s.io	false	ClusterRoleBinding
clusterroles		rbac.authorization.k8s.io	false	ClusterRole
rolebindings		rbac.authorization.k8s.io	true	RoleBinding
roles		rbac.authorization.k8s.io	true	Role
priorityclasses	pc	scheduling.k8s.io	false	PriorityClass
csidrivers		storage.k8s.io	false	CSIDriver
csinodes		storage.k8s.io	false	CSINode
storageclasses	sc	storage.k8s.io	false	StorageClass
volumeattachments		storage.k8s.io	false	VolumeAttachment

输出选项

有关如何格式化或排序某些命令的输出的信息，请使用以下部分。有关哪些命令支持各种输出选项的详细信息，请参阅kubectl 参考文档。

格式化输出

所有 `kubectl` 命令的默认输出格式都是人类可读的纯文本格式。要以特定格式向终端窗口输出详细信息，可以将 `-o` 或 `--output` 参数添加到受支持的 `kubectl` 命令中。

语法

```
kubectl [command] [TYPE] [NAME] -o=<output_format>
```

根据 `kubectl` 操作，支持以下输出格式：

Output format	Description
`-o custom-columns=<spec>`	使用逗号分隔的自定义列列表打印表。
`-o custom-columns-file=<filename>`	使用 `<filename>` 文件中的自定义列模板打印表。
`-o json`	输出 JSON 格式的 API 对象
`-o jsonpath=<template>`	打印 jsonpath 表达式定义的字段
`-o jsonpath-file=<filename>`	打印 `<filename>` 文件中 jsonpath 表达式定义的字段。
`-o name`	仅打印资源名称而不打印任何其他内容。
`-o wide`	以纯文本格式输出，包含任何附加信息。对于 pod 包含节点名。
`-o yaml`	输出 YAML 格式的 API 对象。

示例

在此示例中，以下命令将单个 pod 的详细信息输出为 YAML 格式的对象：

```
kubectl get pod web-pod-13je7 -o yaml
```

请记住：有关每个命令支持哪种输出格式的详细信息，请参阅 kubectl 参考文档。

自定义列

要定义自定义列并仅将所需的详细信息输出到表中，可以使用该 custom-columns 选项。你可以选择内联定义自定义列或使用模板文件：`-o=custom-columns=<spec>` 或 `-o=custom-columns-file=<filename>`。

示例

内联：

```
kubectl get pods <pod-name> -o custom-columns=NAME:.metadata.name,RSRC:.metadata.resourceVersion
```

模板文件：

```
kubectl get pods <pod-name> -o custom-columns-file=template.txt
```

其中，`template.txt` 文件包含：

```
NAME          RSRC
metadata.name metadata.resourceVersion
```

运行任何一个命令的结果类似于:

```
NAME            RSRC
submit-queue    610995
```

Server-side 列

`kubectl` 支持从服务器接收关于对象的特定列信息。
这意味着对于任何给定的资源，服务器将返回与该资源相关的列和行，以便客户端打印。
通过让服务器封装打印的细节，这允许在针对同一集群使用的客户端之间提供一致的人类可读输出。

此功能默认启用。要禁用它，请将该 `--server-print=false` 参数添加到 `kubectl get` 命令中。

例子:

要打印有关 pod 状态的信息，请使用如下命令:

```
kubectl get pods <pod-name> --server-print=false
```

输出类似于:

```
NAME        AGE
pod-name    1m
```

排序列表对象

要将对象排序后输出到终端窗口，可以将 `--sort-by` 参数添加到支持的 `kubectl` 命令。通过使用 `--sort-by` 参数指定任何数字或字符串字段来对对象进行排序。要指定字段，请使用 jsonpath 表达式。

语法

```
kubectl [command] [TYPE] [NAME] --sort-by=<jsonpath_exp>
```

示例

要打印按名称排序的 pod 列表，请运行:

```
kubectl get pods --sort-by=.metadata.name
```

示例：常用操作

使用以下示例集来帮助你熟悉运行常用 kubectl 操作:

`kubectl apply` - 以文件或标准输入为准应用或更新资源。

```
# 使用 example-service.yaml 中的定义创建服务。
kubectl apply -f example-service.yaml

# 使用 example-controller.yaml 中的定义创建 replication controller。
kubectl apply -f example-controller.yaml

# 使用 <directory> 路径下的任意 .yaml, .yml, 或 .json 文件 创建对象。
kubectl apply -f <directory>
```

`kubectl get` - 列出一个或多个资源。

```
# 以纯文本输出格式列出所有 pod。
kubectl get pods

# 以纯文本输出格式列出所有 pod，并包含附加信息(如节点名)。
kubectl get pods -o wide

# 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。
kubectl get replicationcontroller <rc-name>

# 以纯文本输出格式列出所有副本控制器和服务。
kubectl get rc,services

# 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。
kubectl get ds --include-uninitialized

# 列出在节点 server01 上运行的所有 pod
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。

```
# 显示名称为 <node-name> 的节点的详细信息。
kubectl describe nodes <node-name>

# 显示名为 <pod-name> 的 pod 的详细信息。
kubectl describe pods/<pod-name>

# 显示由名为 <rc-name> 的副本控制器管理的所有 pod 的详细信息。
# 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。
kubectl describe pods <rc-name>

# 描述所有的 pod，不包括未初始化的 pod
kubectl describe pods
```

> `kubectl get` 命令通常用于检索同一资源类型的一个或多个资源。
> 它具有丰富的参数，允许你使用 `-o` 或 `--output` 参数自定义输出格式。你可以指定 `-w` 或 `--watch` 参数以开始观察特定对象的更新。
> `kubectl describe` 命令更侧重于描述指定资源的许多相关方面。它可以调用对 `API 服务器` 的多个 API 调用来为用户构建视图。
> 例如，该 `kubectl describe node` 命令不仅检索有关节点的信息，还检索在其上运行的 pod 的摘要，为节点生成的事件等。

`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。

```
# 使用 pod.yaml 文件中指定的类型和名称删除 pod。
kubectl delete -f pod.yaml

# 删除所有带有 '<label-key>=<label-value>' 标签的 Pod 和服务。
kubectl delete pods,services -l <label-key>=<label-value>

# 删除所有 pod，包括未初始化的 pod。
kubectl delete pods --all
```

`kubectl exec` - 对 pod 中的容器执行命令。

```
# 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。
kubectl exec <pod-name> -- date

# 运行输出 'date' 获取在容器的 <container-name> 中 pod <pod-name> 的输出。
kubectl exec <pod-name> -c <container-name> -- date

# 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。
kubectl exec -ti <pod-name> -- /bin/bash
```

`kubectl logs` - 打印 Pod 中容器的日志。

```
# 从 pod 返回日志快照。
kubectl logs <pod-name>

# 从 pod <pod-name> 开始流式传输日志。这类似于 'tail -f' Linux 命令。
kubectl logs -f <pod-name>
```

示例：创建和使用插件

使用以下示例来帮助你熟悉编写和使用 `kubectl` 插件：

```
# 用任何语言创建一个简单的插件，并为生成的可执行文件命名
# 以前缀 "kubectl-" 开始
cat ./kubectl-hello
```

```
#!/bin/sh

# 这个插件打印单词 "hello world"
echo "hello world"
```

这个插件写好了，把它变成可执行的：

```
sudo chmod a+x ./kubectl-hello

# 并将其移动到路径中的某个位置
sudo mv ./kubectl-hello /usr/local/bin
sudo chown root:root /usr/local/bin

# 你现在已经创建并"安装了"一个 kubectl 插件。
# 你可以开始使用这个插件，从 kubectl 调用它，就像它是一个常规命令一样
kubectl hello
```

```
hello world
```

```
# 你可以"卸载"一个插件，只需从你的路径中删除它
sudo rm /usr/local/bin/kubectl-hello
```

为了查看可用的所有 `kubectl` 插件，你可以使用 `kubectl plugin list` 子命令：

```
kubectl plugin list
```

输出类似于：

```
The following kubectl-compatible plugins are available:

/usr/local/bin/kubectl-hello
/usr/local/bin/kubectl-foo
/usr/local/bin/kubectl-bar

```

`kubectl plugin list` 指令也可以向你告警哪些插件被运行，或是被其它插件覆盖了,例如:

```
sudo chmod -x /usr/local/bin/kubectl-foo # 删除执行权限
kubectl plugin list
```

```
The following kubectl-compatible plugins are available:

/usr/local/bin/kubectl-hello
/usr/local/bin/kubectl-foo
  - warning: /usr/local/bin/kubectl-foo identified as a plugin, but it is not executable
/usr/local/bin/kubectl-bar

error: one plugin warning was found
```

你可以将插件视为在现有 kubectl 命令之上构建更复杂功能的一种方法：

```
cat ./kubectl-whoami
```

接下来的几个示例假设你已经将 `kubectl-whoami` 设置为以下内容：

```
#!/bin/bash

#这个插件利用 `kubectl config` 命令基于当前所选上下文输出当前用户的信息
kubectl config view --template='{{ range .contexts }}{{ if eq .name "'$(kubectl config current-context)'" }}Current user: {{ printf "%s\n" .context.user }}{{ end }}{{ end }}'
```

运行以上命令将为你提供一个输出，其中包含 KUBECONFIG 文件中当前上下文的用户：

```
#!/bin/bash
# 使文件成为可执行的
sudo chmod +x ./kubectl-whoami

# 然后移动到你的路径中
sudo mv ./kubectl-whoami /usr/local/bin

kubectl whoami
Current user: plugins-user
```