Command line tool (kubectl)

Kubernetes provides a command line tool for communicating with a Kubernetes cluster's control plane, using the Kubernetes API.

This tool is named `kubectl`.

For configuration, `kubectl` looks for a file named `config` in the `$HOME/.kube` directory. You can specify other kubeconfig files by setting the `KUBECONFIG` environment variable or by setting the `--kubeconfig` flag.

This overview covers `kubectl` syntax, describes the command operations, and provides common examples. For details about each command, including all the supported flags and subcommands, see the kubectl reference documentation.

For installation instructions, see Installing kubectl; for a quick guide, see the cheat sheet. If you're used to using the `docker` command-line tool, `kubectl` for Docker Users explains some equivalent commands for Kubernetes.

Syntax

Use the following syntax to run `kubectl` commands from your terminal window:

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

where `command`, `TYPE`, `NAME`, and `flags` are:

- `command`: Specifies the operation that you want to perform on one or more resources, for example `create`, `get`, `describe`, `delete`.
- `TYPE`: Specifies the resource type. Resource types are case-insensitive and you can specify the singular, plural, or abbreviated forms. For example, the following commands produce the same output:

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

- `NAME`: Specifies the name of the resource. Names are case-sensitive. If the name is omitted, details for all resources are displayed, for example `kubectl get pods`.

  When performing an operation on multiple resources, you can specify each resource by type and name or specify one or more files:

    - To specify resources by type and name:
        - To group resources if they are all the same type: `TYPE1 name1 name2 name<#>`.
          Example: `kubectl get pod example-pod1 example-pod2`
        - To specify multiple resource types individually: `TYPE1/name1 TYPE1/name2 TYPE2/name3 TYPE<#>/name<#>`.
          Example: `kubectl get pod/example-pod1 replicationcontroller/example-rc1`
    - To specify resources with one or more files: `-f file1 -f file2 -f file<#>`
        - Use YAML rather than JSON since YAML tends to be more user-friendly, especially for configuration files.
          Example: `kubectl get -f ./pod.yaml`
- `flags`: Specifies optional flags. For example, you can use the `-s` or `--server` flags to specify the address and port of the Kubernetes API server.

Caution: Flags that you specify from the command line override default values and any corresponding environment variables.
If you need help, run `kubectl help` from the terminal window.

In-cluster authentication and namespace overrides

By default `kubectl` will first determine if it is running within a pod, and thus in a cluster. It starts by checking for the `KUBERNETES_SERVICE_HOST` and `KUBERNETES_SERVICE_PORT` environment variables and the existence of a service account token file at `/var/run/secrets/kubernetes.io/serviceaccount/token`. If all three are found in-cluster authentication is assumed.

To maintain backwards compatibility, if the `POD_NAMESPACE` environment variable is set during in-cluster authentication it will override the default namespace from the service account token. Any manifests or tools relying on namespace defaulting will be affected by this.

`POD_NAMESPACE` environment variable

If the `POD_NAMESPACE` environment variable is set, cli operations on namespaced resources will default to the variable value. For example, if the variable is set to `seattle`, `kubectl get pods` would return pods in the `seattle` namespace. This is because pods are a namespaced resource, and no namespace was provided in the

command. Review the output of `kubectl api-resources` to determine if a resource is namespaced.

Explicit use of `--namespace <value>` overrides this behavior.

How kubectl handles ServiceAccount tokens

If:

- there is Kubernetes service account token file mounted at `/var/run/secrets/kubernetes.io/serviceaccount/token`, and
- the `KUBERNETES_SERVICE_HOST` environment variable is set, and
- the `KUBERNETES_SERVICE_PORT` environment variable is set, and
- you don't explicitly specify a namespace on the kubectl command line

then kubectl assumes it is running in your cluster. The kubectl tool looks up the namespace of that ServiceAccount (this is the same as the namespace of the Pod) and acts against that namespace. This is different from what happens outside of a cluster; when kubectl runs outside a cluster and you don't specify a namespace, the kubectl command acts against the namespace set for the current context in your client configuration. To change the default namespace for your kubectl you can use the following command:

```
kubectl config set-context --current --namespace=<namespace-name>
```

Operations

The following table includes short descriptions and the general syntax for all of the `kubectl` operations:

Operation	Syntax	Description
`alpha`	`kubectl alpha SUBCOMMAND [flags]`	List the available commands that correspond to alpha features, which are not enabled in Kubernetes clusters by default.
`annotate`	`kubectl annotate (-f FILENAME | TYPE NAME | TYPE/NAME) KEY_1=VAL_1 ... KEY_N=VAL_N [--overwrite] [--all] [--resource-version=version] [flags]`	Add or update the annotations of one or more resources.
`api-resources`	`kubectl api-resources [flags]`	List the API resources that are available.

Operation	Syntax	Description
api-versions	kubectl api-versions [flags]	List the API versions that are available.
apply	kubectl apply -f FILENAME [flags]	Apply a configuration change to a resource from a file or stdin.
attach	kubectl attach POD -c CONTAINER [-i] [-t] [flags]	Attach to a running container either to view the output stream or interact with the container (stdin).
auth	kubectl auth [flags] [options]	Inspect authorization.
autoscale	kubectl autoscale (-f FILENAME | TYPE NAME | TYPE/NAME) [--min=MINPODS] --max=MAXPODS [--cpu-percent=CPU] [flags]	Automatically scale the set of pods that are managed by a replication controller.
certificate	kubectl certificate SUBCOMMAND [options]	Modify certificate resources.
cluster-info	kubectl cluster-info [flags]	Display endpoint information about the master and services in the cluster.
completion	kubectl completion SHELL [options]	Output shell completion code for the specified shell (bash or zsh).
config	kubectl config SUBCOMMAND [flags]	Modifies kubeconfig files. See the individual subcommands for details.
convert	kubectl convert -f FILENAME [options]	Convert config files between different API versions. Both YAML and JSON formats are accepted. Note - requires `kubectl-convert` plugin to be installed.
cordon	kubectl cordon NODE [options]	Mark node as unschedulable.
cp	kubectl cp <file-spec-src> <file-spec-dest> [options]	Copy files and directories to and from containers.
create	kubectl create -f FILENAME [flags]	Create one or more resources from a file or stdin.
delete	kubectl delete (-f FILENAME | TYPE [NAME | /NAME | -l label | --all]) [flags]	Delete resources either from a file, stdin, or specifying label selectors, names, resource selectors, or resources.
describe	kubectl describe (-f FILENAME | TYPE [NAME_PREFIX | /NAME | -l label]) [flags]	Display the detailed state of one or more resources.
diff	kubectl diff -f FILENAME [flags]	Diff file or stdin against live configuration.

Operation	Syntax	Description
drain	kubectl drain NODE [options]	Drain node in preparation for maintenance.
edit	kubectl edit (-f FILENAME | TYPE NAME | TYPE/NAME) [flags]	Edit and update the definition of one or more resources on the server by using the default editor.
events	kubectl events	List events
exec	kubectl exec POD [-c CONTAINER] [-i] [-t] [flags] [-- COMMAND [args...]]	Execute a command against a container in a pod.
explain	kubectl explain TYPE [--recursive=false] [flags]	Get documentation of various resources. For instance pods, nodes, services, etc.
expose	kubectl expose (-f FILENAME | TYPE NAME | TYPE/NAME) [--port=port] [--protocol=TCP|UDP] [--target-port=number-or-name] [--name=name] [--external-ip=external-ip-of-service] [--type=type] [flags]	Expose a replication controller, service, or pod as a new Kubernetes service.
get	kubectl get (-f FILENAME | TYPE [NAME | /NAME | -l label]) [--watch] [--sort-by=FIELD] [[-o | --output]=OUTPUT_FORMAT] [flags]	List one or more resources.
kustomize	kubectl kustomize <dir> [flags] [options]	List a set of API resources generated from instructions in a kustomization.yaml file. The argument must be the path to the directory containing the file, or a git repository URL with a path suffix specifying same with respect to the repository root.
label	kubectl label (-f FILENAME | TYPE NAME | TYPE/NAME) KEY_1=VAL_1 ... KEY_N=VAL_N [--overwrite] [--all] [--resource-version=version] [flags]	Add or update the labels of one or more resources.
logs	kubectl logs POD [-c CONTAINER] [--follow] [flags]	Print the logs for a container in a pod.
options	kubectl options	List of global command-line options, which apply to all commands.
patch	kubectl patch (-f FILENAME | TYPE NAME | TYPE/NAME) --patch PATCH [flags]	Update one or more fields of a resource by using the strategic merge patch process.
plugin	kubectl plugin [flags] [options]	Provides utilities for interacting with plugins.

Operation	Syntax	Description
port-forward	kubectl port-forward POD [LOCAL_PORT:]REMOTE_PORT [...[LOCAL_PORT_N:]REMOTE_PORT_N] [flags]	Forward one or more local ports to a pod.
proxy	kubectl proxy [--port=PORT] [--www=static-dir] [--www-prefix=prefix] [--api-prefix=prefix] [flags]	Run a proxy to the Kubernetes API server.
replace	kubectl replace -f FILENAME	Replace a resource from a file or stdin.
rollout	kubectl rollout SUBCOMMAND [options]	Manage the rollout of a resource. Valid resource types include: deployments, daemonsets and statefulsets.
run	kubectl run NAME --image=image [--env="key=value"] [--port=port] [--dry-run=server|client|none] [--overrides=inline-json] [flags]	Run a specified image on the cluster.
scale	kubectl scale (-f FILENAME | TYPE NAME | TYPE/NAME) --replicas=COUNT [--resource-version=version] [--current-replicas=count] [flags]	Update the size of the specified replication controller.
set	kubectl set SUBCOMMAND [options]	Configure application resources.
taint	kubectl taint NODE NAME KEY_1=VAL_1:TAINT_EFFECT_1 ... KEY_N=VAL_N:TAINT_EFFECT_N [options]	Update the taints on one or more nodes.
top	`kubectl top (POD	NODE) [flags] [options]`
uncordon	kubectl uncordon NODE [options]	Mark node as schedulable.
version	kubectl version [--client] [flags]	Display the Kubernetes version running on the client and server.
wait	kubectl wait ([-f FILENAME] | resource.group/resource.name | resource.group [(-l label | --all)]) [--for=delete|--for condition=available] [options]	Experimental: Wait for a specific condition on one or many resources.

To learn more about command operations, see the kubectl reference documentation.

Resource types

The following table includes a list of all the supported resource types and their abbreviated aliases.

(This output can be retrieved from `kubectl api-resources`, and was accurate as of Kubernetes 1.25.0)

NAME	SHORTNAMES	APIVERSION	NAMESPACED	KIND
bindings		v1	true	Binding
componentstatuses	cs	v1	false	ComponentStatus
configmaps	cm	v1	true	ConfigMap
endpoints	ep	v1	true	Endpoints
events	ev	v1	true	Event
limitranges	limits	v1	true	LimitRange
namespaces	ns	v1	false	Namespace
nodes	no	v1	false	Node
persistentvolumeclaims	pvc	v1	true	PersistentVolumeClaim
persistentvolumes	pv	v1	false	PersistentVolume
pods	po	v1	true	Pod
podtemplates		v1	true	PodTemplate
replicationcontrollers	rc	v1	true	ReplicationController
resourcequotas	quota	v1	true	ResourceQuota
secrets		v1	true	Secret
serviceaccounts	sa	v1	true	ServiceAccount
services	svc	v1	true	Service
mutatingwebhookconfigurations		admissionregistration.k8s.io/v1	false	MutatingWebhookConfiguration
validatingwebhookconfigurations		admissionregistration.k8s.io/v1	false	ValidatingWebhookConfiguration
customresourcedefinitions	crd,crds	apiextensions.k8s.io/v1	false	CustomResourceDefinition
apiservices		apiregistration.k8s.io/v1	false	APIService
controllerrevisions		apps/v1	true	ControllerRevision
daemonsets	ds	apps/v1	true	DaemonSet
deployments	deploy	apps/v1	true	Deployment
replicasets	rs	apps/v1	true	ReplicaSet
statefulsets	sts	apps/v1	true	StatefulSet
tokenreviews		authentication.k8s.io/v1	false	TokenReview
localsubjectaccessreviews		authorization.k8s.io/v1	true	LocalSubjectAccessReview

NAME	SHORTNAMES	APIVERSION	NAMESPACED	KIND
selfsubjectaccessreviews		authorization.k8s.io/v1	false	SelfSubjectAccessReview
selfsubjectrulesreviews		authorization.k8s.io/v1	false	SelfSubjectRulesReview
subjectaccessreviews		authorization.k8s.io/v1	false	SubjectAccessReview
horizontalpodautoscalers	hpa	autoscaling/v2	true	HorizontalPodAutoscaler
cronjobs	cj	batch/v1	true	CronJob
jobs		batch/v1	true	Job
certificatesigningrequests	csr	certificates.k8s.io/v1	false	CertificateSigningRequest
leases		coordination.k8s.io/v1	true	Lease
endpointslices		discovery.k8s.io/v1	true	EndpointSlice
events	ev	events.k8s.io/v1	true	Event
flowschemas		flowcontrol.apiserver.k8s.io/v1beta2	false	FlowSchema
prioritylevelconfigurations		flowcontrol.apiserver.k8s.io/v1beta2	false	PriorityLevelConfiguration
ingressclasses		networking.k8s.io/v1	false	IngressClass
ingresses	ing	networking.k8s.io/v1	true	Ingress
networkpolicies	netpol	networking.k8s.io/v1	true	NetworkPolicy
runtimeclasses		node.k8s.io/v1	false	RuntimeClass
poddisruptionbudgets	pdb	policy/v1	true	PodDisruptionBudget
podsecuritypolicies	psp	policy/v1beta1	false	PodSecurityPolicy
clusterrolebindings		rbac.authorization.k8s.io/v1	false	ClusterRoleBinding
clusterroles		rbac.authorization.k8s.io/v1	false	ClusterRole
rolebindings		rbac.authorization.k8s.io/v1	true	RoleBinding

NAME	SHORTNAMES	APIVERSION	NAMESPACED	KIND
roles		rbac.authorization.k8s.io/v1	true	Role
priorityclasses	pc	scheduling.k8s.io/v1	false	PriorityClass
csidrivers		storage.k8s.io/v1	false	CSIDriver
csinodes		storage.k8s.io/v1	false	CSINode
csistoragecapacities		storage.k8s.io/v1	true	CSIStorageCapacity
storageclasses	sc	storage.k8s.io/v1	false	StorageClass
volumeattachments		storage.k8s.io/v1	false	VolumeAttachment

Output options

Use the following sections for information about how you can format or sort the output of certain commands. For details about which commands support the various output options, see the kubectl reference documentation.

Formatting output

The default output format for all `kubectl` commands is the human readable plain-text format. To output details to your terminal window in a specific format, you can add either the `-o` or `--output` flags to a supported `kubectl` command.

Syntax

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

Depending on the `kubectl` operation, the following output formats are supported:

Output format	Description
`-o custom-columns=<spec>`	Print a table using a comma separated list of custom columns.
`-o custom-columns-file=<filename>`	Print a table using the custom columns template in the `<filename>` file.
`-o json`	Output a JSON formatted API object.
`-o jsonpath=<template>`	Print the fields defined in a jsonpath expression.
`-o jsonpath-file=<filename>`	Print the fields defined by the jsonpath expression in the `<filename>` file.
`-o name`	Print only the resource name and nothing else.
`-o wide`	Output in the plain-text format with any additional information. For pods, the node name is included.

Output format	Description
`-o yaml`	Output a YAML formatted API object.

Example

In this example, the following command outputs the details for a single pod as a YAML formatted object:

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

Remember: See the kubectl reference documentation for details about which output format is supported by each command.

Custom columns

To define custom columns and output only the details that you want into a table, you can use the `custom-columns` option. You can choose to define the custom columns inline or use a template file: `-o custom-columns=<spec>` or `-o custom-columns-file=<filename>`.

Examples

Inline:

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

Template file:

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

where the `template.txt` file contains:

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

The result of running either command is similar to:

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

Server-side columns

`kubectl` supports receiving specific column information from the server about objects. This means that for any given resource, the server will return columns and rows relevant to that resource, for the client to print. This allows for consistent

human-readable output across clients used against the same cluster, by having the server encapsulate the details of printing.

This feature is enabled by default. To disable it, add the `--server-print=false` flag to the `kubectl get` command.

Examples

To print information about the status of a pod, use a command like the following:

```
kubectl get pods <pod-name> --server-print=false
```
The output is similar to:

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

Sorting list objects

To output objects to a sorted list in your terminal window, you can add the `--sort-by` flag to a supported `kubectl` command. Sort your objects by specifying any numeric or string field with the `--sort-by` flag. To specify a field, use a jsonpath expression.

Syntax

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

Example

To print a list of pods sorted by name, you run:

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

Examples: Common operations

Use the following set of examples to help you familiarize yourself with running the commonly used `kubectl` operations:

`kubectl apply` - Apply or Update a resource from a file or stdin.

```
# Create a service using the definition in example-service.yaml.
kubectl apply -f example-service.yaml
```

```
# Create a replication controller using the definition in example-controller.yaml.
kubectl apply -f example-controller.yaml

# Create the objects that are defined in any .yaml, .yml, or .json file within the <directory> directory.
kubectl apply -f <directory>
```

`kubectl get` - List one or more resources.

```
# List all pods in plain-text output format.
kubectl get pods

# List all pods in plain-text output format and include additional information (such as node name).
kubectl get pods -o wide

# List the replication controller with the specified name in plain-text output format. Tip: You can shorten and replace the 'replicationcontroller' resource type with the alias 'rc'.
kubectl get replicationcontroller <rc-name>

# List all replication controllers and services together in plain-text output format.
kubectl get rc,services

# List all daemon sets in plain-text output format.
kubectl get ds

# List all pods running on node server01
kubectl get pods --field-selector=spec.nodeName=server01
```

`kubectl describe` - Display detailed state of one or more resources, including the uninitialized ones by default.

```
# Display the details of the node with name <node-name>.
kubectl describe nodes <node-name>

# Display the details of the pod with name <pod-name>.
kubectl describe pods/<pod-name>

# Display the details of all the pods that are managed by the replication controller named <rc-name>.
# Remember: Any pods that are created by the replication controller get prefixed with the name of the replication controller.
kubectl describe pods <rc-name>

# Describe all pods
kubectl describe pods
```

Note: The `kubectl get` command is usually used for retrieving one or more resources of the same resource type. It features a rich set of flags that allows you to customize the output format using the `-o` or `--output` flag, for example. You can

specify the `-w` or `--watch` flag to start watching updates to a particular object. The `kubectl describe` command is more focused on describing the many related aspects of a specified resource. It may invoke several API calls to the API server to build a view for the user. For example, the `kubectl describe node` command retrieves not only the information about the node, but also a summary of the pods running on it, the events generated for the node etc.

`kubectl delete` - Delete resources either from a file, stdin, or specifying label selectors, names, resource selectors, or resources.

```
# Delete a pod using the type and name specified in the pod.yaml file.
kubectl delete -f pod.yaml

# Delete all the pods and services that have the label '<label-key>=<label-value>'.
kubectl delete pods,services -l <label-key>=<label-value>

# Delete all pods, including uninitialized ones.
kubectl delete pods --all
```

`kubectl exec` - Execute a command against a container in a pod.

```
# Get output from running 'date' from pod <pod-name>. By default, output is from the first container.
kubectl exec <pod-name> -- date

# Get output from running 'date' in container <container-name> of pod <pod-name>.
kubectl exec <pod-name> -c <container-name> -- date

# Get an interactive TTY and run /bin/bash from pod <pod-name>. By default, output is from the first container.
kubectl exec -ti <pod-name> -- /bin/bash
```

`kubectl logs` - Print the logs for a container in a pod.

```
# Return a snapshot of the logs from pod <pod-name>.
kubectl logs <pod-name>

# Start streaming the logs from pod <pod-name>. This is similar to the 'tail -f' Linux command.
kubectl logs -f <pod-name>
```

`kubectl diff` - View a diff of the proposed updates to a cluster.

```
# Diff resources included in "pod.json".
kubectl diff -f pod.json

# Diff file read from stdin.
cat service.yaml | kubectl diff -f -
```

Examples: Creating and using plugins

Use the following set of examples to help you familiarize yourself with writing and using `kubectl` plugins:

```
# create a simple plugin in any language and name the resulting executable
file
# so that it begins with the prefix "kubectl-"
cat ./kubectl-hello
#!/bin/sh

# this plugin prints the words "hello world"
echo "hello world"
```

With a plugin written, let's make it executable:

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

# and move it to a location in our PATH
sudo mv ./kubectl-hello /usr/local/bin
sudo chown root:root /usr/local/bin

# You have now created and "installed" a kubectl plugin.
# You can begin using this plugin by invoking it from kubectl as if it
were a regular command
kubectl hello
hello world
# You can "uninstall" a plugin, by removing it from the folder in your
# $PATH where you placed it
sudo rm /usr/local/bin/kubectl-hello
```

In order to view all of the plugins that are available to `kubectl`, use the `kubectl plugin list` subcommand:

```
kubectl plugin list
```

The output is similar to:

```
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` also warns you about plugins that are not executable, or that are shadowed by other plugins; for example:

```
sudo chmod -x /usr/local/bin/kubectl-foo # remove execute permission
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
```

You can think of plugins as a means to build more complex functionality on top of the existing kubectl commands:

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

The next few examples assume that you already made `kubectl-whoami` have the following contents:

```
#!/bin/bash

# this plugin makes use of the `kubectl config` command in order to output
# information about the current user, based on the currently selected
context
kubectl config view --template='{{ range .contexts }}{{ if eq .name
"'$(kubectl config current-context)'" }}Current user: {{ printf "%s\n"
.context.user }}{{ end }}{{ end }}'
```

Running the above command gives you an output containing the user for the current context in your KUBECONFIG file:

```
# make the file executable
sudo chmod +x ./kubectl-whoami

# and move it into your PATH
sudo mv ./kubectl-whoami /usr/local/bin

kubectl whoami
Current user: plugins-user
```

What's next

- Read the `kubectl` reference documentation:
  - the kubectl command reference
  - the command line arguments reference
- Learn about `kubectl` usage conventions
- Read about JSONPath support in kubectl
- Read about how to extend kubectl with plugins
  - To find out more about plugins, take a look at the example CLI plugin.