**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](https://kubernetes.io/docs/user-guide/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](https://kubernetes.io/docs/reference/kubectl/overview/#custom-columns). |
| -o custom-columns-file=<filename> | Print a table using the [custom columns](https://kubernetes.io/docs/reference/kubectl/overview/#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](https://kubernetes.io/docs/reference/kubectl/jsonpath/) expression. |
| -o jsonpath-file=<filename> | Print the fields defined by the [jsonpath](https://kubernetes.io/docs/reference/kubectl/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. |
| -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](https://kubernetes.io/docs/user-guide/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:

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 in kubectl 1.11 and higher. 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

Output looks like this:

NAME READY STATUS RESTARTS AGE

pod-name 1/1 Running 0 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](https://kubernetes.io/docs/reference/kubectl/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, including uninitialized ones, in plain-text output format.*

kubectl get ds --include-uninitialized

*# 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, not including uninitialized ones*

kubectl describe pods --include-uninitialized=false

**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 name=<label-name>.*

kubectl delete pods,services -l name=<label-name>

*# Delete all the pods and services that have the label name=<label-name>, including uninitialized ones.*

kubectl delete pods,services -l name=<label-name> --include-uninitialized

*# 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>

**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/bash

*# this plugin prints the words "hello world"*

echo "hello world"

*# with our plugin written, let's make it executable*

sudo chmod +x ./kubectl-hello

*# and move it to a location in our PATH*

sudo mv ./kubectl-hello /usr/local/bin

*# we have now created and "installed" a kubectl plugin.*

*# we can begin using our plugin by invoking it from kubectl as if it were a regular command*

kubectl hello

hello world

# we can "uninstall" a plugin, by simply removing it from our PATH

sudo rm /usr/local/bin/kubectl-hello

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

kubectl plugin list

The following kubectl-compatible plugins are available:

/usr/local/bin/kubectl-hello

/usr/local/bin/kubectl-foo

/usr/local/bin/kubectl-bar

# this command can also warn us about plugins that are

# not executable, or that are overshadowed by other

# plugins, for example

sudo chmod -x /usr/local/bin/kubectl-foo

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

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

cat ./kubectl-whoami

#!/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: {{ .context.user }}{{ end }}{{ end }}'

Running the above plugin gives us an output containing the user for the currently selected context in our KUBECONFIG file:

*# make the file executable*

sudo chmod +x ./kubectl-whoami

*# and move it into our PATH*

sudo mv ./kubectl-whoami /usr/local/bin

kubectl whoami

Current user: plugins-user

To find out more about plugins, take a look at the [example cli plugin](https://github.com/kubernetes/sample-cli-plugin)