kubectl

1 - Overview of kubectl

The kubectl command line tool lets you control Kubernetes clusters. 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.

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.

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.
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.

Operation	Syntax	Description
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.
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.
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.
exec	kubectl exec POD [-c CONTAINER] [-i] [-t] [flags] [-- COMMAND [args...]]	Execute a command against a container in a pod.
explain	kubectl explain [--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.

Operation	Syntax	Description
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.
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.

Operation	Syntax	Description
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 [flags] [options]	Display Resource (CPU/Memory/Storage) usage.
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 1.19.1.)

NAME	SHORTNAMES	APIGROUP	NAMESPACED	KIND
bindings			true	Binding
componentstatuses	cs		false	ComponentStatus

NAME	SHORTNAMES	APIGROUP	NAMESPACED	KIND
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

NAME	SHORTNAMES	APIGROUP	NAMESPACED	KIND
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
ingresses	ing	extensions	true	Ingress
flowschemas		flowcontrol.apiserver.k8s.io	false	FlowSchema

NAME	SHORTNAMES	APIGROUP	NAMESPACED	KIND
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

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.
`-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.
```

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 th
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 (s
kubectl get pods -o wide

# List the replication controller with the specified name in plain-text output fo
kubectl get replicationcontroller <rc-name>

# List all replication controllers and services together in plain-text output for
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 control
# Remember: Any pods that are created by the replication controller get prefixed
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
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, outpu
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'
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 reg
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 exec
/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 c
```

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

- Start using the kubectl commands.
- To find out more about plugins, take a look at the example cli plugin.

2 - JSONPath Support

Kubectl supports JSONPath template.

JSONPath template is composed of JSONPath expressions enclosed by curly braces {}. Kubectl uses JSONPath expressions to filter on specific fields in the JSON object and format the output. In addition to the original JSONPath template syntax, the following functions and syntax are valid:

1. Use double quotes to quote text inside JSONPath expressions.
2. Use the `range`, `end` operators to iterate lists.
3. Use negative slice indices to step backwards through a list. Negative indices do not "wrap around" a list and are valid as long as `-index + listLength >= 0`.

> **Note:**
> - The `$` operator is optional since the expression always starts from the root object by default.
> - The result object is printed as its String() function.

Given the JSON input:

```
{
  "kind": "List",
  "items":[
    {
      "kind":"None",
      "metadata":{"name":"127.0.0.1"},
      "status":{
        "capacity":{"cpu":"4"},
        "addresses":[{"type": "LegacyHostIP", "address":"127.0.0.1"}]
      }
    },
    {
      "kind":"None",
      "metadata":{"name":"127.0.0.2"},
      "status":{
        "capacity":{"cpu":"8"},
        "addresses":[
          {"type": "LegacyHostIP", "address":"127.0.0.2"},
          {"type": "another", "address":"127.0.0.3"}
        ]
      }
    }
  ],
  "users":[
    {
      "name": "myself",
      "user": {}
    },
    {
      "name": "e2e",
      "user": {"username": "admin", "password": "secret"}
    }
  ]
}
```

Function	Description	Example	Result
`text`	the plain text	`kind is {.kind}`	`kind is List`
`@`	the current object	`{@}`	the same as input

Function	Description	Example	Result
`. or []`	child operator	`{.kind}, {['kind']} or {['name\.type']}`	`List`
`..`	recursive descent	`{..name}`	`127.0.0.1 127.0.0.2 myself e2e`
`*`	wildcard. Get all objects	`{.items[*].metadata.name}`	`[127.0.0.1 127.0.0.2]`
`[start:end:step]`	subscript operator	`{.users[0].name}`	`myself`
`[,]`	union operator	`{.items[*] ['metadata.name', 'status.capacity']}`	`127.0.0.1 127.0.0.2 map[cpu:4] map[cpu:8]`
`?()`	filter	`{.users[?(@.name=="e2e")].user.password}`	`secret`
`range, end`	iterate list	`{range .items[*]} [{.metadata.name}, {.status.capacity}] {end}`	`[127.0.0.1, map[cpu:4]] [127.0.0.2, map[cpu:8]]`
`''`	quote interpreted string	`{range .items[*]} {.metadata.name}{'\t'} {end}`	`127.0.0.1 127.0.0.2`

Examples using `kubectl` and JSONPath expressions:

```
kubectl get pods -o json
kubectl get pods -o=jsonpath='{@}'
kubectl get pods -o=jsonpath='{.items[0]}'
kubectl get pods -o=jsonpath='{.items[0].metadata.name}'
kubectl get pods -o=jsonpath="{.items[*]['metadata.name', 'status.capacity']}"
kubectl get pods -o=jsonpath='{range .items[*]}{.metadata.name}{"\t"}{.status.sta
```

Note:
On Windows, you must *double* quote any JSONPath template that contains spaces (not single quote as shown above for bash). This in turn means that you must use a single quote or escaped double quote around any literals in the template. For example:

```
kubectl get pods -o=jsonpath="{range .items[*]}{.metadata.name}{'\t'}{.status.
kubectl get pods -o=jsonpath="{range .items[*]}{.metadata.name}{\"\t\"}{.statu
```

Note:
JSONPath regular expressions are not supported. If you want to match using regular expressions, you can use a tool such as `jq`.

```
# kubectl does not support regular expressions for JSONpath output
# The following command does not work
kubectl get pods -o jsonpath='{.items[?(@.metadata.name=~/^test$/)].metadata.r

# The following command achieves the desired result
kubectl get pods -o json | jq -r '.items[] | select(.metadata.name | test("tes
```

3 - kubectl

Synopsis

kubectl controls the Kubernetes cluster manager.

Find more information at: https://kubernetes.io/docs/reference/kubectl/overview/

```
kubectl [flags]
```

Options

--add-dir-header

If true, adds the file directory to the header of the log messages

--alsologtostderr

log to standard error as well as files

--as string

Username to impersonate for the operation

--as-group stringArray

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

--azure-container-registry-config string

Path to the file containing Azure container registry configuration information.

--cache-dir string Default: "$HOME/.kube/cache"

Default cache directory

--certificate-authority string

Path to a cert file for the certificate authority

--client-certificate string

Path to a client certificate file for TLS

--client-key string

Path to a client key file for TLS

--cloud-provider-gce-l7lb-src-cidrs cidrs Default: 130.211.0.0/22,35.191.0.0/16

CIDRs opened in GCE firewall for L7 LB traffic proxy & health checks

--cloud-provider-gce-lb-src-cidrs cidrs Default: 130.211.0.0/22,209.85.152.0/22,209.85.204.0/22,35.191.0.0/16

CIDRs opened in GCE firewall for L4 LB traffic proxy & health checks

--cluster string

The name of the kubeconfig cluster to use

--context string

The name of the kubeconfig context to use

--default-not-ready-toleration-seconds int Default: 300

Indicates the tolerationSeconds of the toleration for notReady:NoExecute that is added by default to every pod that does not already have such a toleration.

--default-unreachable-toleration-seconds int Default: 300

Indicates the tolerationSeconds of the toleration for unreachable:NoExecute that is added by default to every pod that does not already have such a toleration.

-h, --help

help for kubectl

--insecure-skip-tls-verify

If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

--kubeconfig string

Path to the kubeconfig file to use for CLI requests.

--log-backtrace-at traceLocation Default: :0

when logging hits line file:N, emit a stack trace

--log-dir string

If non-empty, write log files in this directory

--log-file string

If non-empty, use this log file

--log-file-max-size uint Default: 1800

Defines the maximum size a log file can grow to. Unit is megabytes. If the value is 0, the maximum file size is unlimited.

--log-flush-frequency duration Default: 5s

Maximum number of seconds between log flushes

--logtostderr Default: true

log to standard error instead of files

--match-server-version

Require server version to match client version

-n, --namespace string

If present, the namespace scope for this CLI request

--one-output

If true, only write logs to their native severity level (vs also writing to each lower severity level

--password string

Password for basic authentication to the API server

--profile string Default: "none"

Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)

--profile-output string Default: "profile.pprof"

Name of the file to write the profile to

--request-timeout string Default: "0"

The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

-s, --server string

The address and port of the Kubernetes API server

--skip-headers

If true, avoid header prefixes in the log messages

--skip-log-headers

If true, avoid headers when opening log files

--stderrthreshold severity Default: 2

logs at or above this threshold go to stderr

--tls-server-name string

Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used

--token string

Bearer token for authentication to the API server

--user string

The name of the kubeconfig user to use

--username string

Username for basic authentication to the API server

-v, --v Level

number for the log level verbosity

--version version[=true]

Print version information and quit

--vmodule moduleSpec

	comma-separated list of pattern=N settings for file-filtered logging
--warnings-as-errors	
	Treat warnings received from the server as errors and exit with a non-zero exit code

See Also

- kubectl annotate - Update the annotations on a resource
- kubectl api-resources - Print the supported API resources on the server
- kubectl api-versions - Print the supported API versions on the server, in the form of "group/version"
- kubectl apply - Apply a configuration to a resource by filename or stdin
- kubectl attach - Attach to a running container
- kubectl auth - Inspect authorization
- kubectl autoscale - Auto-scale a Deployment, ReplicaSet, or ReplicationController
- kubectl certificate - Modify certificate resources.
- kubectl cluster-info - Display cluster info
- kubectl completion - Output shell completion code for the specified shell (bash or zsh)
- kubectl config - Modify kubeconfig files
- kubectl cordon - Mark node as unschedulable
- kubectl cp - Copy files and directories to and from containers.
- kubectl create - Create a resource from a file or from stdin.
- kubectl debug - Create debugging sessions for troubleshooting workloads and nodes
- kubectl delete - Delete resources by filenames, stdin, resources and names, or by resources and label selector
- kubectl describe - Show details of a specific resource or group of resources
- kubectl diff - Diff live version against would-be applied version
- kubectl drain - Drain node in preparation for maintenance
- kubectl edit - Edit a resource on the server
- kubectl exec - Execute a command in a container
- kubectl explain - Documentation of resources
- kubectl expose - Take a replication controller, service, deployment or pod and expose it as a new Kubernetes Service
- kubectl get - Display one or many resources
- kubectl kustomize - Build a kustomization target from a directory or a remote url.
- kubectl label - Update the labels on a resource
- kubectl logs - Print the logs for a container in a pod
- kubectl options - Print the list of flags inherited by all commands
- kubectl patch - Update field(s) of a resource
- kubectl plugin - Provides utilities for interacting with plugins.
- kubectl port-forward - Forward one or more local ports to a pod
- kubectl proxy - Run a proxy to the Kubernetes API server
- kubectl replace - Replace a resource by filename or stdin
- kubectl rollout - Manage the rollout of a resource
- kubectl run - Run a particular image on the cluster
- kubectl scale - Set a new size for a Deployment, ReplicaSet or Replication Controller
- kubectl set - Set specific features on objects
- kubectl taint - Update the taints on one or more nodes
- kubectl top - Display Resource (CPU/Memory/Storage) usage.
- kubectl uncordon - Mark node as schedulable
- kubectl version - Print the client and server version information
- kubectl wait - Experimental: Wait for a specific condition on one or many resources.

4 - kubectl Cheat Sheet

This page contains a list of commonly used `kubectl` commands and flags.

Kubectl autocomplete

BASH

```
source <(kubectl completion bash) # setup autocomplete in bash into the current s
echo "source <(kubectl completion bash)" >> ~/.bashrc # add autocomplete permanen
```

You can also use a shorthand alias for `kubectl` that also works with completion:

```
alias k=kubectl
complete -F __start_kubectl k
```

ZSH

```
source <(kubectl completion zsh)  # setup autocomplete in zsh into the current sh
echo "[[ $commands[kubectl] ]] && source <(kubectl completion zsh)" >> ~/.zshrc #
```

Kubectl context and configuration

Set which Kubernetes cluster `kubectl` communicates with and modifies configuration information. See Authenticating Across Clusters with kubeconfig documentation for detailed config file information.

```
kubectl config view # Show Merged kubeconfig settings.

# use multiple kubeconfig files at the same time and view merged config
KUBECONFIG=~/.kube/config:~/.kube/kubconfig2

kubectl config view

# get the password for the e2e user
kubectl config view -o jsonpath='{.users[?(@.name == "e2e")].user.password}'

kubectl config view -o jsonpath='{.users[].name}'    # display the first user
kubectl config view -o jsonpath='{.users[*].name}'   # get a list of users
kubectl config get-contexts                          # display list of contexts
kubectl config current-context                       # display the current-contex
kubectl config use-context my-cluster-name           # set the default context to

# add a new user to your kubeconf that supports basic auth
kubectl config set-credentials kubeuser/foo.kubernetes.com --username=kubeuser --p

# permanently save the namespace for all subsequent kubectl commands in that cont
kubectl config set-context --current --namespace=ggckad-s2

# set a context utilizing a specific username and namespace.
kubectl config set-context gce --user=cluster-admin --namespace=foo \
  && kubectl config use-context gce

kubectl config unset users.foo                       # delete user foo
```

</>

Kubectl apply

`apply` manages applications through files defining Kubernetes resources. It creates and updates resources in a cluster through running `kubectl apply`. This is the recommended way of managing Kubernetes applications on production. See Kubectl Book.

Creating objects

Kubernetes manifests can be defined in YAML or JSON. The file extension `.yaml`, `.yml`, and `.json` can be used.

```
kubectl apply -f ./my-manifest.yaml            # create resource(s)
kubectl apply -f ./my1.yaml -f ./my2.yaml      # create from multiple files
kubectl apply -f ./dir                         # create resource(s) in all manife
kubectl apply -f https://git.io/vPieo          # create resource(s) from url
kubectl create deployment nginx --image=nginx  # start a single instance of nginx

# create a Job which prints "Hello World"
kubectl create job hello --image=busybox -- echo "Hello World"

# create a CronJob that prints "Hello World" every minute
kubectl create cronjob hello --image=busybox   --schedule="*/1 * * * *" -- echo "

kubectl explain pods                           # get the documentation for pod ma

# Create multiple YAML objects from stdin
cat <<EOF | kubectl apply -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

# Create a secret with several keys
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  password: $(echo -n "s33msi4" | base64 -w0)
  username: $(echo -n "jane" | base64 -w0)
EOF
```

Viewing, finding resources

```
# Get commands with basic output
kubectl get services                          # List all services in the namespac
kubectl get pods --all-namespaces             # List all pods in all namespaces
kubectl get pods -o wide                      # List all pods in the current name
kubectl get deployment my-dep                 # List a particular deployment
kubectl get pods                              # List all pods in the namespace
kubectl get pod my-pod -o yaml                # Get a pod's YAML

# Describe commands with verbose output
kubectl describe nodes my-node
kubectl describe pods my-pod

# List Services Sorted by Name
kubectl get services --sort-by=.metadata.name

# List pods Sorted by Restart Count
kubectl get pods --sort-by='.status.containerStatuses[0].restartCount'

# List PersistentVolumes sorted by capacity
kubectl get pv --sort-by=.spec.capacity.storage

# Get the version label of all pods with label app=cassandra
kubectl get pods --selector=app=cassandra -o \
  jsonpath='{.items[*].metadata.labels.version}'

# Retrieve the value of a key with dots, e.g. 'ca.crt'
kubectl get configmap myconfig \
  -o jsonpath='{.data.ca\.crt}'

# Get all worker nodes (use a selector to exclude results that have a label
# named 'node-role.kubernetes.io/master')
kubectl get node --selector='!node-role.kubernetes.io/master'

# Get all running pods in the namespace
kubectl get pods --field-selector=status.phase=Running

# Get ExternalIPs of all nodes
kubectl get nodes -o jsonpath='{.items[*].status.addresses[?(@.type=="ExternalIP"

# List Names of Pods that belong to Particular RC
# "jq" command useful for transformations that are too complex for jsonpath, it c
sel=${$(kubectl get rc my-rc --output=json | jq -j '.spec.selector | to_entries |
echo $(kubectl get pods --selector=$sel --output=jsonpath={.items..metadata.name}

# Show labels for all pods (or any other Kubernetes object that supports labellin
kubectl get pods --show-labels

# Check which nodes are ready
JSONPATH='{range .items[*]}{@.metadata.name}:{range @.status.conditions[*]}{@.typ
 && kubectl get nodes -o jsonpath="$JSONPATH" | grep "Ready=True"

# Output decoded secrets without external tools
kubectl get secret my-secret -o go-template='{{range $k,$v := .data}}{{"### "}}{{$

# List all Secrets currently in use by a pod
kubectl get pods -o json | jq '.items[].spec.containers[].env[]?.valueFrom.secretK

# List all containerIDs of initContainer of all pods
# Helpful when cleaning up stopped containers, while avoiding removal of initCont
kubectl get pods --all-namespaces -o jsonpath='{range .items[*].status.initContai

# List Events sorted by timestamp
kubectl get events --sort-by=.metadata.creationTimestamp

# Compares the current state of the cluster against the state that the cluster wo
kubectl diff -f ./my-manifest.yaml
```

```
# Produce a period-delimited tree of all keys returned for nodes
# Helpful when locating a key within a complex nested JSON structure
kubectl get nodes -o json | jq -c 'path(..)|[.[]|tostring]|join(".")'

# Produce a period-delimited tree of all keys returned for pods, etc
kubectl get pods -o json | jq -c 'path(..)|[.[]|tostring]|join(".")'

# Produce ENV for all pods, assuming you have a default container for the pods, d
# Helpful when running any supported command across all pods, not just `env`
for pod in $(kubectl get po --output=jsonpath={.items..metadata.name}); do echo $
```

Updating resources

```
kubectl set image deployment/frontend www=image:v2               # Rolling update
kubectl rollout history deployment/frontend                      # Check the hist
kubectl rollout undo deployment/frontend                         # Rollback to th
kubectl rollout undo deployment/frontend --to-revision=2         # Rollback to a
kubectl rollout status -w deployment/frontend                    # Watch rolling
kubectl rollout restart deployment/frontend                      # Rolling restar


cat pod.json | kubectl replace -f -                              # Replace a pod

# Force replace, delete and then re-create the resource. Will cause a service out
kubectl replace --force -f ./pod.json

# Create a service for a replicated nginx, which serves on port 80 and connects t
kubectl expose rc nginx --port=80 --target-port=8000

# Update a single-container pod's image version (tag) to v4
kubectl get pod mypod -o yaml | sed 's/\(image: myimage\):.*$/\1:v4/' | kubectl r

kubectl label pods my-pod new-label=awesome                      # Add a Label
kubectl annotate pods my-pod icon-url=http://goo.gl/XXBTWq       # Add an annotat
kubectl autoscale deployment foo --min=2 --max=10                # Auto scale a d
```

Patching resources

```
# Partially update a node
kubectl patch node k8s-node-1 -p '{"spec":{"unschedulable":true}}'

# Update a container's image; spec.containers[*].name is required because it's a
kubectl patch pod valid-pod -p '{"spec":{"containers":[{"name":"kubernetes-serve-

# Update a container's image using a json patch with positional arrays
kubectl patch pod valid-pod --type='json' -p='[{"op": "replace", "path": "/spec/c

# Disable a deployment livenessProbe using a json patch with positional arrays
kubectl patch deployment valid-deployment  --type json   -p='[{"op": "remove", "p

# Add a new element to a positional array
kubectl patch sa default --type='json' -p='[{"op": "add", "path": "/secrets/1", "
```

Editing resources

Edit any API resource in your preferred editor.

```
kubectl edit svc/docker-registry                      # Edit the service named do
KUBE_EDITOR="nano" kubectl edit svc/docker-registry   # Use an alternative editor
```

Scaling resources

```
kubectl scale --replicas=3 rs/foo                                 # Scale a repli
kubectl scale --replicas=3 -f foo.yaml                            # Scale a resou
kubectl scale --current-replicas=2 --replicas=3 deployment/mysql  # If the deploy
kubectl scale --replicas=5 rc/foo rc/bar rc/baz                   # Scale multipl
```

Deleting resources

```
kubectl delete -f ./pod.json                                      # Delet
kubectl delete pod,service baz foo                                # Delet
kubectl delete pods,services -l name=myLabel                      # Delet
kubectl -n my-ns delete pod,svc --all                              # Dele
# Delete all pods matching the awk pattern1 or pattern2
kubectl get pods  -n mynamespace --no-headers=true | awk '/pattern1|pattern2/{pri
```

Interacting with running Pods

```
kubectl logs my-pod                                 # dump pod logs (stdout)
kubectl logs -l name=myLabel                        # dump pod logs, with label n
kubectl logs my-pod --previous                      # dump pod logs (stdout) for
kubectl logs my-pod -c my-container                 # dump pod container logs (st
kubectl logs -l name=myLabel -c my-container        # dump pod logs, with label n
kubectl logs my-pod -c my-container --previous      # dump pod container logs (st
kubectl logs -f my-pod                              # stream pod logs (stdout)
kubectl logs -f my-pod -c my-container              # stream pod container logs (
kubectl logs -f -l name=myLabel --all-containers    # stream all pods logs with l
kubectl run -i --tty busybox --image=busybox -- sh  # Run pod as interactive shel
kubectl run nginx --image=nginx -n
mynamespace                                         # Run pod nginx in a specific
kubectl run nginx --image=nginx                     # Run pod nginx and write its
--dry-run=client -o yaml > pod.yaml

kubectl attach my-pod -i                            # Attach to Running Container
kubectl port-forward my-pod 5000:6000               # Listen on port 5000 on the
kubectl exec my-pod -- ls /                         # Run command in existing pod
kubectl exec --stdin --tty my-pod -- /bin/sh        # Interactive shell access to
kubectl exec my-pod -c my-container -- ls /         # Run command in existing pod
kubectl top pod POD_NAME --containers               # Show metrics for a given po
kubectl top pod POD_NAME --sort-by=cpu              # Show metrics for a given po
```

Interacting with Deployments and Services

```
kubectl logs deploy/my-deployment                          # dump Pod logs for a D
kubectl logs deploy/my-deployment -c my-container          # dump Pod logs for a D

kubectl port-forward svc/my-service 5000                   # listen on local port
kubectl port-forward svc/my-service 5000:my-service-port   # listen on local port

kubectl port-forward deploy/my-deployment 5000:6000        # listen on local port
kubectl exec deploy/my-deployment -- ls                    # run command in first
```

Interacting with Nodes and cluster

```
kubectl cordon my-node                                                # Mark my-n
kubectl drain my-node                                                 # Drain my-
kubectl uncordon my-node                                              # Mark my-n
kubectl top node my-node                                              # Show metr
kubectl cluster-info                                                  # Display a
kubectl cluster-info dump                                             # Dump curr
kubectl cluster-info dump --output-directory=/path/to/cluster-state   # Dump curr

# If a taint with that key and effect already exists, its value is replaced as sp
kubectl taint nodes foo dedicated=special-user:NoSchedule
```

Resource types

List all supported resource types along with their shortnames, API group, whether they are namespaced, and Kind:

```
kubectl api-resources
```

Other operations for exploring API resources:

```
kubectl api-resources --namespaced=true      # All namespaced resources
kubectl api-resources --namespaced=false     # All non-namespaced resources
kubectl api-resources -o name                # All resources with simple output (
kubectl api-resources -o wide                # All resources with expanded (aka "
kubectl api-resources --verbs=list,get       # All resources that support the "li
kubectl api-resources --api-group=extensions # All resources in the "extensions" A
```

Formatting output

To output details to your terminal window in a specific format, add the `-o` (or `--output`) flag to a supported `kubectl` command.

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, and for pods, the node name is included
`-o=yaml`	Output a YAML formatted API object

Examples using `-o=custom-columns`:

```
# All images running in a cluster
kubectl get pods -A -o=custom-columns='DATA:spec.containers[*].image'

# All images running in namespace: default, grouped by Pod
kubectl get pods --namespace default --output=custom-columns="NAME:.metadata.name

 # All images excluding "k8s.gcr.io/coredns:1.6.2"
kubectl get pods -A -o=custom-columns='DATA:spec.containers[?(@.image!="k8s.gcr.i

# All fields under metadata regardless of name
kubectl get pods -A -o=custom-columns='DATA:metadata.*'
```

More examples in the kubectl reference documentation.

Kubectl output verbosity and debugging

Kubectl verbosity is controlled with the `-v` or `--v` flags followed by an integer representing the log level. General Kubernetes logging conventions and the associated log levels are described here.

Verbosity	Description
`--v=0`	Generally useful for this to *always* be visible to a cluster operator.
`--v=1`	A reasonable default log level if you don't want verbosity.
`--v=2`	Useful steady state information about the service and important log messages that may correlate to significant changes in the system. This is the recommended default log level for most systems.
`--v=3`	Extended information about changes.
`--v=4`	Debug level verbosity.
`--v=5`	Trace level verbosity.
`--v=6`	Display requested resources.
`--v=7`	Display HTTP request headers.
`--v=8`	Display HTTP request contents.
`--v=9`	Display HTTP request contents without truncation of contents.

What's next

- Read the kubectl overview and learn about JsonPath.
- See kubectl options.
- Also read kubectl Usage Conventions to understand how to use kubectl in reusable scripts.
- See more community kubectl cheatsheets.

5 - kubectl Commands

kubectl Command Reference

6 - kubectl for Docker Users

You can use the Kubernetes command line tool `kubectl` to interact with the API Server. Using kubectl is straightforward if you are familiar with the Docker command line tool. However, there are a few differences between the Docker commands and the kubectl commands. The following sections show a Docker sub-command and describe the equivalent `kubectl` command.

docker run

To run an nginx Deployment and expose the Deployment, see kubectl create deployment.
docker:

```
docker run -d --restart=always -e DOMAIN=cluster --name nginx-app -p 80:80 nginx
```

```
55c103fa129692154a7652490236fee9be47d70a8dd562281ae7d2f9a339a6db
```

```
docker ps
```

```
CONTAINER ID        IMAGE               COMMAND                  CREATED
55c103fa1296        nginx               "nginx -g 'daemon of…"   9 seconds ago
```

kubectl:

```
# start the pod running nginx
kubectl create deployment --image=nginx nginx-app
```

```
deployment.apps/nginx-app created
```

```
# add env to nginx-app
kubectl set env deployment/nginx-app  DOMAIN=cluster
```

```
deployment.apps/nginx-app env updated
```

Note: `kubectl` commands print the type and name of the resource created or mutated, which can then be used in subsequent commands. You can expose a new Service after a Deployment is created.

```
# expose a port through with a service
kubectl expose deployment nginx-app --port=80 --name=nginx-http
```

```
service "nginx-http" exposed
```

By using kubectl, you can create a Deployment to ensure that N pods are running nginx, where N is the number of replicas stated in the spec and defaults to 1. You can also create a service with a selector that matches the pod labels. For more information, see Use a Service to Access an

Application in a Cluster.

By default images run in the background, similar to `docker run -d ...`. To run things in the foreground, use `kubectl run` to create pod:

```
kubectl run [-i] [--tty] --attach <name> --image=<image>
```

Unlike `docker run ...`, if you specify `--attach`, then you attach `stdin`, `stdout` and `stderr`. You cannot control which streams are attached (`docker -a ...`). To detach from the container, you can type the escape sequence Ctrl+P followed by Ctrl+Q.

docker ps

To list what is currently running, see kubectl get.

docker:

```
docker ps -a
```

```
CONTAINER ID        IMAGE               COMMAND                  CREATED
14636241935f        ubuntu:16.04        "echo test"              5 seconds ago
55c103fa1296        nginx               "nginx -g 'daemon of…"   About a minute ag
```

kubectl:

```
kubectl get po
```

```
NAME                         READY     STATUS      RESTARTS   AGE
nginx-app-8df569cb7-4gd89    1/1       Running     0          3m
ubuntu                       0/1       Completed   0          20s
```

docker attach

To attach a process that is already running in a container, see kubectl attach.

docker:

```
docker ps
```

```
CONTAINER ID        IMAGE               COMMAND                  CREATED
55c103fa1296        nginx               "nginx -g 'daemon of…"   5 minutes ago
```

```
docker attach 55c103fa1296
...
```

kubectl:

```
kubectl get pods
```

```
NAME              READY     STATUS    RESTARTS   AGE
nginx-app-5jyvm   1/1       Running   0          10m
```

```
kubectl attach -it nginx-app-5jyvm
...
```

To detach from the container, you can type the escape sequence Ctrl+P followed by Ctrl+Q.

docker exec

To execute a command in a container, see kubectl exec.

docker:

```
docker ps
```

```
CONTAINER ID        IMAGE               COMMAND                  CREATED
55c103fa1296        nginx               "nginx -g 'daemon of…"   6 minutes ago
```

```
docker exec 55c103fa1296 cat /etc/hostname
```

```
55c103fa1296
```

kubectl:

```
kubectl get po
```

```
NAME              READY     STATUS    RESTARTS   AGE
nginx-app-5jyvm   1/1       Running   0          10m
```

```
kubectl exec nginx-app-5jyvm -- cat /etc/hostname
```

```
nginx-app-5jyvm
```

To use interactive commands.

docker:

```
docker exec -ti 55c103fa1296 /bin/sh
# exit
```

kubectl:

```
kubectl exec -ti nginx-app-5jyvm -- /bin/sh
# exit
```

For more information, see Get a Shell to a Running Container.

docker logs

To follow stdout/stderr of a process that is running, see kubectl logs.

docker:

```
docker logs -f a9e
```

```
192.168.9.1 - - [14/Jul/2015:01:04:02 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.
192.168.9.1 - - [14/Jul/2015:01:04:03 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.
```

kubectl:

```
kubectl logs -f nginx-app-zibvs
```

```
10.240.63.110 - - [14/Jul/2015:01:09:01 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/
10.240.63.110 - - [14/Jul/2015:01:09:02 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/
```

There is a slight difference between pods and containers; by default pods do not terminate if their processes exit. Instead the pods restart the process. This is similar to the docker run option `--restart=always` with one major difference. In docker, the output for each invocation of the process is concatenated, but for Kubernetes, each invocation is separate. To see the output from a previous run in Kubernetes, do this:

```
kubectl logs --previous nginx-app-zibvs
```

```
10.240.63.110 - - [14/Jul/2015:01:09:01 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/
10.240.63.110 - - [14/Jul/2015:01:09:02 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/
```

For more information, see Logging Architecture.

docker stop and docker rm

To stop and delete a running process, see kubectl delete.

docker:

```
docker ps
```

```
CONTAINER ID        IMAGE               COMMAND                  CREATED
a9ec34d98787        nginx               "nginx -g 'daemon of"  22 hours ago
```

```
docker stop a9ec34d98787
```

```
a9ec34d98787
```

```
docker rm a9ec34d98787
```

```
a9ec34d98787
```

kubectl:

```
kubectl get deployment nginx-app
```

```
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
nginx-app    1/1     1            1           2m
```

```
kubectl get po -l app=nginx-app
```

```
NAME                         READY     STATUS    RESTARTS   AGE
nginx-app-2883164633-aklf7   1/1       Running   0          2m
```

```
kubectl delete deployment nginx-app
```

```
deployment "nginx-app" deleted
```

```
kubectl get po -l app=nginx-app
# Return nothing
```

> **Note:** When you use kubectl, you don't delete the pod directly. You have to first delete the Deployment that owns the pod. If you delete the pod directly, the Deployment recreates the pod.

docker login

There is no direct analog of `docker login` in kubectl. If you are interested in using Kubernetes with a private registry, see Using a Private Registry.

docker version

To get the version of client and server, see kubectl version.

docker:

```
docker version
```

```
Client version: 1.7.0
Client API version: 1.19
Go version (client): go1.4.2
Git commit (client): 0baf609
OS/Arch (client): linux/amd64
Server version: 1.7.0
Server API version: 1.19
Go version (server): go1.4.2
Git commit (server): 0baf609
OS/Arch (server): linux/amd64
```

kubectl:

```
kubectl version
```

```
Client Version: version.Info{Major:"1", Minor:"6", GitVersion:"v1.6.9+a3d1dfa6f433
Server Version: version.Info{Major:"1", Minor:"6", GitVersion:"v1.6.9+a3d1dfa6f433
```

docker info

To get miscellaneous information about the environment and configuration, see kubectl cluster-info.

docker:

```
docker info
```

```
Containers: 40
Images: 168
Storage Driver: aufs
 Root Dir: /usr/local/google/docker/aufs
 Backing Filesystem: extfs
 Dirs: 248
 Dirperm1 Supported: false
Execution Driver: native-0.2
Logging Driver: json-file
Kernel Version: 3.13.0-53-generic
Operating System: Ubuntu 14.04.2 LTS
CPUs: 12
Total Memory: 31.32 GiB
Name: k8s-is-fun.mtv.corp.google.com
ID: ADUV:GCYR:B3VJ:HMPO:LNPQ:KD5S:YKFQ:76VN:IANZ:7TFV:ZBF4:BYJO
WARNING: No swap limit support
```

kubectl:

```
kubectl cluster-info
```

```
Kubernetes master is running at https://203.0.113.141
KubeDNS is running at https://203.0.113.141/api/v1/namespaces/kube-system/services
kubernetes-dashboard is running at https://203.0.113.141/api/v1/namespaces/kube-sy
Grafana is running at https://203.0.113.141/api/v1/namespaces/kube-system/services
Heapster is running at https://203.0.113.141/api/v1/namespaces/kube-system/service
InfluxDB is running at https://203.0.113.141/api/v1/namespaces/kube-system/service
```

7 - kubectl Usage Conventions

Recommended usage conventions for `kubectl`.

Using `kubectl` in Reusable Scripts

For a stable output in a script:

- Request one of the machine-oriented output forms, such as `-o name`, `-o json`, `-o yaml`, `-o go-template`, or `-o jsonpath`.
- Fully-qualify the version. For example, `jobs.v1.batch/myjob`. This will ensure that kubectl does not use its default version that can change over time.
- Don't rely on context, preferences, or other implicit states.

Best Practices

`kubectl run`

For `kubectl run` to satisfy infrastructure as code:

- Tag the image with a version-specific tag and don't move that tag to a new version. For example, use `:v1234`, `v1.2.3`, `r03062016-1-4`, rather than `:latest` (For more information, see Best Practices for Configuration).
- Check in the script for an image that is heavily parameterized.
- Switch to configuration files checked into source control for features that are needed, but not expressible via `kubectl run` flags.

You can use the `--dry-run=client` flag to preview the object that would be sent to your cluster, without really submitting it.

> **Note:** All `kubectl run` generators are deprecated. See the Kubernetes v1.17 documentation for a list of generators and how they were used.

Generators

You can generate the following resources with a kubectl command, `kubectl create --dry-run=client -o yaml`:

- `clusterrole`: Create a ClusterRole.
- `clusterrolebinding`: Create a ClusterRoleBinding for a particular ClusterRole.
- `configmap`: Create a ConfigMap from a local file, directory or literal value.
- `cronjob`: Create a CronJob with the specified name.
- `deployment`: Create a Deployment with the specified name.
- `job`: Create a Job with the specified name.
- `namespace`: Create a Namespace with the specified name.
- `poddisruptionbudget`: Create a PodDisruptionBudget with the specified name.
- `priorityclass`: Create a PriorityClass with the specified name.
- `quota`: Create a Quota with the specified name.
- `role`: Create a Role with single rule.
- `rolebinding`: Create a RoleBinding for a particular Role or ClusterRole.
- `secret`: Create a Secret using specified subcommand.
- `service`: Create a Service using specified subcommand.
- `serviceaccount`: Create a ServiceAccount with the specified name.

`kubectl apply`

- You can use `kubectl apply` to create or update resources. For more information about using kubectl apply to update resources, see Kubectl Book.