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.
`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.
`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.
`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.
`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.

Operation	Syntax	Description
`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 1.25.0)

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
`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
`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.
`-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.

1 - 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 shell, bash-completion package should be installed first.
echo "source <(kubectl completion bash)" >> ~/.bashrc # add autocomplete permanently to your bash shell.
```

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

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

ZSH

```
source <(kubectl completion zsh)  # setup autocomplete in zsh into the current shell
echo '[[ $commands[kubectl] ]] && source <(kubectl completion zsh)' >> ~/.zshrc # add autocomplete permanently to your zsh shell
```

A Note on --all-namespaces

Appending `--all-namespaces` happens frequently enough where you should be aware of the shorthand for `--all-namespaces`:

`kubectl -A`

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-context
kubectl config use-context my-cluster-name           # set the default context to my-cluster-name

kubectl config set-cluster my-cluster-name           # set a cluster entry in the kubeconfig

# configure the URL to a proxy server to use for requests made by this client in the kubeconfig
kubectl config set-cluster my-cluster-name --proxy-url=my-proxy-url

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

# permanently save the namespace for all subsequent kubectl commands in that context.
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

# short alias to set/show context/namespace (only works for bash and bash-compatible shells, current context to be set before using kn to set namespace)
alias kx='f() { [ "$1" ] && kubectl config use-context $1 || kubectl config current-context ; } ; f'
alias kn='f() { [ "$1" ] && kubectl config set-context --current --namespace $1 || kubectl config view --minify | grep namespace | cut -d" " -f6 ; } ; f'
```

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 manifest files in dir
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:1.28 -- echo "Hello World"

# create a CronJob that prints "Hello World" every minute
kubectl create cronjob hello --image=busybox:1.28   --schedule="*/1 * * * *" -- echo "Hello World"

kubectl explain pods                          # get the documentation for pod manifests

# 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:1.28
    args:
    - sleep
    - "1000000"
---
apiVersion: v1
kind: Pod
metadata:
  name: busybox-sleep-less
spec:
  containers:
  - name: busybox
    image: busybox:1.28
    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 namespace
kubectl get pods --all-namespaces            # List all pods in all namespaces
kubectl get pods -o wide                     # List all pods in the current namespace, with more details
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}'

# Retrieve a base64 encoded value with dashes instead of underscores.
kubectl get secret my-secret --template='{{index .data "key-name-with-dashes"}}'

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

# 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")].address}'

# List Names of Pods that belong to Particular RC
# "jq" command useful for transformations that are too complex for jsonpath, it can be found at https://stedolan.github.io/jq/
sel=${$(kubectl get rc my-rc --output=json | jq -j '.spec.selector | to_entries | .[] | "\(.key)=\(.value),"')%?}
echo $(kubectl get pods --selector=$sel --output=jsonpath={.items..metadata.name})

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

# Check which nodes are ready
JSONPATH='{range .items[*]}{@.metadata.name}:{range @.status.conditions[*]}{@.type}={@.status};{end}{end}' \
 && 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}}{{"### "}}{{$k}}{{"\n"}}{{$v|base64decode}}{{"\n\n"}}{{end}}'

# List all Secrets currently in use by a pod
kubectl get pods -o json | jq '.items[].spec.containers[].env[]?.valueFrom.secretKeyRef.name' | grep -v null | sort | uniq

# List all containerIDs of initContainer of all pods
# Helpful when cleaning up stopped containers, while avoiding removal of initContainers.
kubectl get pods --all-namespaces -o jsonpath='{range .items[*].status.initContainerStatuses[*]}{.containerID}{"\n"}{end}' | cut -d/ -f3

# 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 would be in if the manifest was applied.
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 'paths|join(".")'

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

# Produce ENV for all pods, assuming you have a default container for the pods, default namespace and the `env` command is supported.
# 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 $pod && kubectl exec -it $pod -- env; done

# Get a deployment's status subresource
kubectl get deployment nginx-deployment --subresource=status
```

Updating resources

```
kubectl set image deployment/frontend www=image:v2              # Rolling update "www" containers of "frontend" deployment, updating the image
kubectl rollout history deployment/frontend                     # Check the history of deployments including the revision
kubectl rollout undo deployment/frontend                        # Rollback to the previous deployment
kubectl rollout undo deployment/frontend --to-revision=2        # Rollback to a specific revision
kubectl rollout status -w deployment/frontend                   # Watch rolling update status of "frontend" deployment until completion
kubectl rollout restart deployment/frontend                     # Rolling restart of the "frontend" deployment


cat pod.json | kubectl replace -f -                             # Replace a pod based on the JSON passed into stdin

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

# Create a service for a replicated nginx, which serves on port 80 and connects to the containers on port 8000
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 replace -f -

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

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 merge key
kubectl patch pod valid-pod -p '{"spec":{"containers":[{"name":"kubernetes-serve-hostname","image":"new image"}]}}'

# Update a container's image using a json patch with positional arrays
kubectl patch pod valid-pod --type='json' -p='[{"op": "replace", "path": "/spec/containers/0/image", "value":"new image"}]'

# Disable a deployment livenessProbe using a json patch with positional arrays
kubectl patch deployment valid-deployment  --type json   -p='[{"op": "remove", "path": "/spec/template/spec/containers/0/livenessProbe"}]'

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

# Update a deployment's replica count by patching its scale subresource
kubectl patch deployment nginx-deployment --subresource='scale' --type='merge' -p '{"spec":{"replicas":2}}'
```

Editing resources

Edit any API resource in your preferred editor.

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

Scaling resources

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

Deleting resources

```
kubectl delete -f ./pod.json                                      # Delete a pod using the type and name specified in pod.json
kubectl delete pod unwanted --now                                 # Delete a pod with no grace period
kubectl delete pod,service baz foo                                # Delete pods and services with same names "baz" and "foo"
kubectl delete pods,services -l name=myLabel                      # Delete pods and services with label name=myLabel
kubectl -n my-ns delete pod,svc --all                             # Delete all pods and services in namespace my-ns,
# Delete all pods matching the awk pattern1 or pattern2
kubectl get pods  -n mynamespace --no-headers=true | awk '/pattern1|pattern2/{print $1}' | xargs  kubectl delete -n mynamespace pod
```

Interacting with running Pods

```
kubectl logs my-pod                                 # dump pod logs (stdout)
kubectl logs -l name=myLabel                        # dump pod logs, with label name=myLabel (stdout)
kubectl logs my-pod --previous                      # dump pod logs (stdout) for a previous instantiation of a container
kubectl logs my-pod -c my-container                 # dump pod container logs (stdout, multi-container case)
kubectl logs -l name=myLabel -c my-container        # dump pod logs, with label name=myLabel (stdout)
kubectl logs my-pod -c my-container --previous      # dump pod container logs (stdout, multi-container case) for a previous instantiation of a container
kubectl logs -f my-pod                              # stream pod logs (stdout)
kubectl logs -f my-pod -c my-container              # stream pod container logs (stdout, multi-container case)
kubectl logs -f -l name=myLabel --all-containers    # stream all pods logs with label name=myLabel (stdout)
kubectl run -i --tty busybox --image=busybox:1.28 -- sh  # Run pod as interactive shell
kubectl run nginx --image=nginx -n mynamespace      # Start a single instance of nginx pod in the namespace of mynamespace
kubectl run nginx --image=nginx --dry-run=client -o yaml > pod.yaml
                                                    # Generate spec for running pod nginx and write it into a file called pod.yaml
kubectl attach my-pod -i                            # Attach to Running Container
kubectl port-forward my-pod 5000:6000               # Listen on port 5000 on the local machine and forward to port 6000 on my-pod
kubectl exec my-pod -- ls /                         # Run command in existing pod (1 container case)
kubectl exec --stdin --tty my-pod -- /bin/sh        # Interactive shell access to a running pod (1 container case)
kubectl exec my-pod -c my-container -- ls /         # Run command in existing pod (multi-container case)
kubectl top pod POD_NAME --containers               # Show metrics for a given pod and its containers
kubectl top pod POD_NAME --sort-by=cpu              # Show metrics for a given pod and sort it by 'cpu' or 'memory'
```

Copy files and directories to and from containers

```
kubectl cp /tmp/foo_dir my-pod:/tmp/bar_dir            # Copy /tmp/foo_dir local directory to /tmp/bar_dir in a remote pod in the current namespace
kubectl cp /tmp/foo my-pod:/tmp/bar -c my-container    # Copy /tmp/foo local file to /tmp/bar in a remote pod in a specific container
kubectl cp /tmp/foo my-namespace/my-pod:/tmp/bar       # Copy /tmp/foo local file to /tmp/bar in a remote pod in namespace my-namespace
kubectl cp my-namespace/my-pod:/tmp/foo /tmp/bar       # Copy /tmp/foo from a remote pod to /tmp/bar locally
```

Note: `kubectl cp` requires that the 'tar' binary is present in your container image. If 'tar' is not present,`kubectl cp` will fail. For advanced use cases, such as symlinks, wildcard expansion or file mode preservation consider using `kubectl exec`.

```
tar cf - /tmp/foo | kubectl exec -i -n my-namespace my-pod -- tar xf - -C /tmp/bar           # Copy /tmp/foo local file to /tmp/bar in a remote pod in namespace my-namespace
kubectl exec -n my-namespace my-pod -- tar cf - /tmp/foo | tar xf - -C /tmp/bar    # Copy /tmp/foo from a remote pod to /tmp/bar locally
```

Interacting with Deployments and Services

```
kubectl logs deploy/my-deployment                         # dump Pod logs for a Deployment (single-container case)
kubectl logs deploy/my-deployment -c my-container         # dump Pod logs for a Deployment (multi-container case)

kubectl port-forward svc/my-service 5000                  # listen on local port 5000 and forward to port 5000 on Service backend
kubectl port-forward svc/my-service 5000:my-service-port  # listen on local port 5000 and forward to Service target port with name <my-service-port>

kubectl port-forward deploy/my-deployment 5000:6000       # listen on local port 5000 and forward to port 6000 on a Pod created by <my-deployment>
kubectl exec deploy/my-deployment -- ls                   # run command in first Pod and first container in Deployment (single- or multi-container cases)
```

Interacting with Nodes and cluster

```
kubectl cordon my-node                                                # Mark my-node as unschedulable
kubectl drain my-node                                                 # Drain my-node in preparation for maintenance
kubectl uncordon my-node                                              # Mark my-node as schedulable
kubectl top node my-node                                              # Show metrics for a given node
kubectl cluster-info                                                  # Display addresses of the master and services
kubectl cluster-info dump                                             # Dump current cluster state to stdout
kubectl cluster-info dump --output-directory=/path/to/cluster-state   # Dump current cluster state to /path/to/cluster-state

# View existing taints on which exist on current nodes.
kubectl get nodes -o='custom-columns=NodeName:.metadata.name,TaintKey:.spec.taints[*].key,TaintValue:.spec.taints[*].value,TaintEffect:.spec.taints[*].effect'

# If a taint with that key and effect already exists, its value is replaced as specified.
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 (only the resource name)
kubectl api-resources -o wide                # All resources with expanded (aka "wide") output
kubectl api-resources --verbs=list,get       # All resources that support the "list" and "get" request verbs
kubectl api-resources --api-group=extensions # All resources in the "extensions" API group
```

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,IMAGE:.spec.containers[*].image"

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

# 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

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.

2 - kubectl Commands

kubectl Command Reference

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

Environment variables

KUBECONFIG

Path to the kubectl configuration ("kubeconfig") file. Default: "$HOME/.kube/config"

KUBECTL_COMMAND_HEADERS

When set to false, turns off extra HTTP headers detailing invoked kubectl command (Kubernetes version v1.22 or later)

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 - 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
`.` 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.startTime}{"\n"}{end}'
```

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

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.name}'

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

5 - 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             STATUS              PORTS                NAMES
55c103fa1296        nginx               "nginx -g 'daemon of…"   9 seconds ago       Up 9 seconds        0.0.0.0:80->80/tcp   nginx-app
```

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              STATUS                     PORTS                NAMES
14636241935f        ubuntu:16.04        "echo test"              5 seconds ago        Exited (0) 5 seconds ago                        cocky_fermi
55c103fa1296        nginx               "nginx -g 'daemon of…"   About a minute ago   Up About a minute          0.0.0.0:80->80/tcp   nginx-app
```

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             STATUS              PORTS                NAMES
55c103fa1296        nginx               "nginx -g 'daemon of…"   5 minutes ago       Up 5 minutes        0.0.0.0:80->80/tcp   nginx-app
```

```
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             STATUS              PORTS                NAMES
55c103fa1296        nginx               "nginx -g 'daemon of…"   6 minutes ago       Up 6 minutes        0.0.0.0:80->80/tcp   nginx-app
```

```
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/7.35.0" "-"
192.168.9.1 - - [14/Jul/2015:01:04:03 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.35.0" "-"
```

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/7.26.0" "-"
10.240.63.110 - - [14/Jul/2015:01:09:02 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.26.0" "-"
```

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/7.26.0" "-"
10.240.63.110 - - [14/Jul/2015:01:09:02 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.26.0" "-"
```

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            STATUS              PORTS                         NAMES
a9ec34d98787        nginx               "nginx -g 'daemon of"  22 hours ago       Up 22 hours         0.0.0.0:80->80/tcp, 443/tcp   nginx-app
```

```
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+a3d1dfa6f4335", GitCommit:"9b77fed11a9843ce3780f70dd251e92901c43072", GitTreeState:"dirty", BuildDate:"2017-08-29T20:32:58Z", OpenPaasKubernetesVersion:"v1.03.02", GoVersion:"go1.7.5", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"6", GitVersion:"v1.6.9+a3d1dfa6f4335", GitCommit:"9b77fed11a9843ce3780f70dd251e92901c43072", GitTreeState:"dirty", BuildDate:"2017-08-29T20:32:58Z", OpenPaasKubernetesVersion:"v1.03.02", GoVersion:"go1.7.5", Compiler:"gc", Platform:"linux/amd64"}
```

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/kube-dns/proxy
kubernetes-dashboard is running at https://203.0.113.141/api/v1/namespaces/kube-system/services/kubernetes-dashboard/proxy
Grafana is running at https://203.0.113.141/api/v1/namespaces/kube-system/services/monitoring-grafana/proxy
Heapster is running at https://203.0.113.141/api/v1/namespaces/kube-system/services/monitoring-heapster/proxy
InfluxDB is running at https://203.0.113.141/api/v1/namespaces/kube-system/services/monitoring-influxdb/proxy
```

6 - 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.

Subresources

- You can use the `--subresource` alpha flag for kubectl commands like `get`, `patch`, `edit` and `replace` to fetch and update subresources for all resources that support them. Currently, only the `status` and `scale` subresources are supported.
- The API contract against a subresource is identical to a full resource. While updating the `status` subresource to a new value, keep in mind that the subresource could be potentially reconciled by a controller to a different value.

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.

`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.