THE DEFINITIVE CHEAT SHEET

KUBECTL

WWW.THECHIEF.IO

⭐Kubectl Sheetcheat

Note: This cheatsheet is part of **Learn Kubernetes by Building 10 projects** ebook.

Pods

List all pods in namespace `<default>`

```
kubectl get pods
```

or

```
kubectl get pod
```

or

```
kubectl get po
```

View a pod in watch mode

```
kubectl get pod <pod> --watch
```

View all pods in watch mode

```
kubectl get pods -A --watch
```

List sroted pods

```
kubectl get pods --sort-by='.status.containerStatuses[0].restartCount'
```

List pods using a different output

```
kubectl get pods -o <json|yaml|wide|custom-columns=...|custom-columns-
file=...|go-template=...|go-template-file=...|jsonpath=...|jsonpath-file=...>
```

Examples:

- JSON output

```
kubectl get pods -o json
```

or

```
kubectl get pods -ojson
```

or

```
kubectl get pods -o=json
```

- Wide output:

```
kubectl get pods -o wide
```

- Custom columns:

```
kubectl get pods -o custom-columns='DATA:spec.containers[*].image'
```

or

```
kubectl get pods -o custom-columns='DATA:spec.containers[*].volumeMounts'
```

or

```
kubectl get pods -o custom-columns='DATA:metadata.*'
```

Formatting output

To output details to your terminal window in a specific format, add the `-o` (or `--output`) flag to a supported `kubectl` command (source: K8s docs)

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

List all pods in a namespace

```
kubectl get pods -n <namespace>
```

or

```
kubectl -n <namespace> get pods
```

or

```
kubectl --namespace <namespace> get pods
```

List all pods in all namespaces

```
kubectl get pods --all-namespaces
```

or

```
kubectl get pods -A
```

Create from an image

```
kubectl run <pod> --generator=run-pod/v1 --image=<image>
```

In the following cheatsheet, we will be using images such as nginx or busybox.

Example:

```
kubectl run nginx --generator=run-pod/v1 --image=nginx
```

```
kubectl run busybox --generator=run-pod/v1 --image=busybox
```

Run pod in an interactive shell mode

```
kubectl run -i --tty nginx --image=nginx -- sh
```

Run a command after creating a pod

```
kubectl run busybox --image=busybox -- sleep 100000
```

Executing a command in a running pod

```
kubectl exec <pod> -- <command>
```

Or pass stdin to the container in TTY mode:

```
kubectl exec -it <pod> -- <command>
```

Example:

```
kubectl exec -it nginx -- ls -lrth /app/
```

Create a pod: dry run mode (without really creating it)

```
kubectl run <pod> --generator=run-pod/v1 --image=nginx --dry-run
```

Patch a pod

```
kubectl patch pod <pod> -p '<patch>'
```

Example:

```
kubectl patch pod <pod> -p  '{"spec":{"containers":[{"name":"kubernetes-serve-
hostname","image":"new image"}]}}'
```

Another example:

```
kubectl patch pod valid-pod --type='json' -p='[{"op": "replace", "path":
"/spec/containers/0/image", "value":"new image"}]'
```

Create from a YAML file

```
kubectl create -f pod.yaml
```

Export YAML from the dry run mode

```
kubectl run nginx --generator=run-pod/v1 --image=nginx --dry-run -o yaml
```

Create from STDIN

```
cat <<EOF | kubectl create -f -
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
spec:
  containers:
  - name: nginx
    image: nginx:latest
EOF
```

Create multiple resources from STDIN

```
cat <<EOF | kubectl create -f -
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
spec:
  containers:
  - name: nginx
    image: nginx:latest
---
apiVersion: v1
kind: Pod
metadata:
  name: busybox
spec:
  containers:
  - name: busybox
    image: busybox
    args:
    - sleep
    - "100"
```

Create in a namespace

```
kubectl run nginx --generator=run-pod/v1 --image=nginx -n <namespace>
```

Create in a namespace from a file

```
kubectl create -f pod.yaml -n <namespace>
```

Delete pods

```
kubectl delete pod/<pod>
```

or

```
kubectl delete pod <pod>
```

If you create the pod from a file, you can also use:

```
kubectl delete -f pod.yaml
```

To force deletion:

```
kubectl delete pod <pod> --grace-period=0 --force
```

Get pod logs

```
kubectl logs <pod>
```

or

Sometimes a pod contains more than 1 container. You need to filter the output to get logs for a specific container(s)

```
kubectl logs <pod> -c <container>
```

To follow the logs output (tail -f):

```
kubectl logs -f <pod>
```

If you need to output the logs for all pods with a label

```
kubectl logs -l <label_name>=<label_value>
```

Example:

```
kubectl logs -l env=prod
```

You can also view logs in a multi container case with labels:

```
kubectl logs -l <label_name>=<label_value> -c <container>
```

Or view all cotainers logs with a given label:

```
kubectl logs -f -l <label_name>=<label_value> --all-containers
```

List all container id of init container of all pods

```
kubectl get pods --all-namespaces -o jsonpath='{range
.items[*].status.initContainerStatuses[*]}{.containerID}{"\n"}{end}' | cut -d/ -
f3
```

Show metrics for a given pod

```
kubectl top pod <pod>
```

Show metrics for a given pod and all its containers

```
kubectl top pod <pod> --containers
```

Deployments

Create a deployment

```
kubectl run <deployment> --image=<image>
```

or

```
kubectl create deployment <deployment> --image=<image>
```

Create a deployment with a predefined replica number

```
kubectl run <deployment> --image=<image> --replicas=<number>
```

Create a deployment with a predefined replica number and opening a port

```
kubectl run <deployment> --image=<image> --replicas=<replicas> --port=<port>
```

Example:

```
kubectl run nginx --image=nginx --replicas=2 --port=80
```

Note: The default generator for `kubectl run` is `--generator=deployment/apps.v1`.

Note: `--generator=deployment/apps.v1` is deprecated and will be removed in future versions. Use kubectl run `--generator=run-pod/v1` or `kubectl create` instead.

Create a deployment with a predefined replica number, opening a port and exposing it

```
kubectl run nginx --image=nginx --replicas=2 --port=80 --expose
```

Get a deployment

```
kubectl get deploy <deployment>
```

Watch a deployment

```
kubectl get deployment <deployment> --watch
```

or

```
kubectl get deployment <deployment> -w
```

Or using a shorter version:

```
kubectl get deploy <deployment> -w
```

Or even the longer one:

```
kubectl get deployments.apps <deployment> --watch
```

List all deployments

Same as listing pods, you have multiple options from namespace to output formatters:

```
kubectl get deploy -n <namespace>

kubectl get deploy --all-namespaces
kubectl get deploy -A

kubectl get deploy -oyaml
kubectl get deploy -owide
```

Update the image

Rolling update "nginx" containers of "nginx" deployment, updating the image:

```
kubectl set image deployment/nginx nginx=nginx:1.9.1
```

Rolling update "api" containers of "backend" deployment, updating the image:

```
kubectl set image deployment/backend api=image:v2
```

Scale a deployment

```
kubectl scale --replicas=5 deployment/<deployment>
```

Note: You can use a shorter version:

```
kubectl scale --replicas=5 deploy/<deployment>
```

Dry run and YAML output

```
kubectl run nginx --image=nginx --replicas=2 --port=80 --dry-run -o yaml
```

Create a deployment from a file

```
kubectl apply -f deployment.yaml
```

Edit a deployment

```
kubectl edit deployment/<deployment>
```

Rollback deployment

After editing your deployment, you had an error, a solution can be rolling back to the old deployment status:

```
kubectl rollout undo deployment <deployment>
```

Get rollout history

You can check the rollout history:

```
kubectl rollout history deployment <deployment>
```

```
kubectl rollout history deployment <deployment>
```

Example:

```
kubectl rollout history deployment nginx
```

gives you:

```
REVISION  CHANGE-CAUSE
2         kubectl set image deployment/nginx nginx=nginx:1.9.1 --record=true
3         <none>
```

Roll back to a previous revision

Using the information from the rollout history, we can get back our deployment to a given revision:

```
kubectl rollout undo deployment <deployment> --to-revision=<revision>
```

Example:

```
kubectl rollout undo deployment nginx --to-revision=2
```

Execute deployment rollout operations

```
kubectl rollout status deployment <deployment>
kubectl rollout pause deployment <deployment>
kubectl rollout resume deployment <deployment>
```

Port Forwarding

Choosing localhost port

```
kubectl port-forward deployment <deployment>  <locahost-port>:<deployment-port>
kubectl port-forward pod <pod>  <locahost-port>:<pod-port>
```

Example:

Forward to localhost 8090 from pod 6379:

```
kubectl port-forward redis 8090:6379
```

Listening on the same port

```
kubectl port-forward pod <pod> <port>
```

Example: Listen on ports 8000 and 9000 on localhost, forwarded from the same ports in the pod (8000 and 9000)

```
kubectl port-forward pod nginx 8000 9000
```

Listen on a random port locally

```
kubectl port-forward pod <pod> :<pod-port>
```

Example:

```
kubectl port-forward pod nginx :80
```

Listen on port on localhost + another IP

```
kubectl port-forward --address localhost,<IP.IP.IP.IP> pod <pod> <locahost-
port>:<pod-port>
```

Example:

```
kubectl port-forward --address localhost,10.10.10.1 pod redis 8090:6379
```

Listen on a forwarded port on all addresses

```
kubectl port-forward --address 0.0.0.0 pod <pod> <hosts-port>:<pod-port>
```

Services

Create a service

```
kubectl create service <clusterip|externalname|loadbalancer|nodeport> <service>
[flags] [options]>
```

Examples:

```
kubectl create service clusterip myclusterip --tcp=5678:8080
kubectl create service  loadbalancer myloadbalancer --tcp=80
```

You can use `svc` instead of `service`.

Delete service(s)

```
kubectl delete service myclusterip
kubectl delete service myloadbalancer

kubectl delete svc myclusterip
kubectl delete svc myloadbalancer
```

or

```
kubectl delete service myclusterip myloadbalancer
```

Describe a service

```
kubectl describe service <service>
```

Nodes

Get node

```
kubectl get nodes
```

Get a specific node

```
kubectl get nodes <node>
```

Show node metrics

```
kubectl top node <node>
```

Get external IPs of cluster nodes

```
kubectl get nodes -o jsonpath='{.items[*].status.addresses[?
(@.type=="ExternalIP")].address}'
```

Describe commands with verbose output

```
kubectl describe nodes <node>
```

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"
```

Mark a node as unschedulable

```
kubectl cordon <node>
```

Drain a node for maintenance

```
kubectl drain <node>
```

Mark a node as schedulable

```
kubectl uncordon <node>
```

Namespaces

List namespaces

```
kubectl get namespaces
```

or

```
kubectl get ns
```

List or describe a namespace

```
kubectl get namespace <namespace>
kubectl describe namespace <namespace>
```

Create namespace

```
kubectl create namespace <namespace>
```

or

```
kubectl create -f namespace.yaml
```

or

```
cat <<EOF | kubectl create -f -
apiVersion: v1
kind: Namespace
metadata:
  name: mynamespace
EOF
```

Delete namespace

```
kubectl delete namespace <namespace>
```

or

```
kubectl delete -f namespace.yaml
```

Service accounts

List service accounts

```
kubectl get serviceaccounts
```

or

```
kubectl get sa
```

Get a service account

```
kubectl get serviceaccount <serviceaccount>
```

or

```
kubectl get serviceaccounts <serviceaccount>
```

or

```
kubectl get sa <serviceaccount>
```

or

```
kubectl get sa/<serviceaccount>
```

Create a service account

```
kubectl create serviceaccount <serviceaccount>
```

Delete a service account

```
kubectl delete serviceaccount <serviceaccount>
```

or

```
kubectl delete -f myserviceaccount.yaml
```

Describe a service account

```
kubectl describe serviceaccount <serviceaccount>
```

Events

List events

```
kubectl get events -A
```

List sorted events

```
kubectl get events --sort-by=<JSONPath>
```

Example: Sorted by timestamp

```
kubectl get events --sort-by=.metadata.creationTimestamp
```

List formatted events

```
kubectl get events -o <json|yaml|wide|custom-columns=...|custom-columns-
file=...|go-template=...|go-template-file=...|jsonpath=...|jsonpath-file=...>
```

Example:

```
kubectl get events -owide
```

Documentation

Get the documentation for pod manifests

```
kubectl explain pod
```

Get the documentation for service manifests

```
kubectl explain service
```

Describing resources

```
kubectl describe <resource> <reosurce_name>
```

Example:

```
kubectl describe pod busybox
```

or

```
kubectl describe nodes minikube
```

Other possible resources you can use with `describe`:

```
apiservices.apiregistration.k8s.io
certificatesigningrequests.certificates.k8s.io
clusterrolebindings.rbac.authorization.k8s.io
clusterroles.rbac.authorization.k8s.io
componentstatuses
configmaps
controllerrevisions.apps
cronjobs.batch
csidrivers.storage.k8s.io
csinodes.storage.k8s.io
customresourcedefinitions.apiextensions.k8s.io
daemonsets.apps
daemonsets.extensions
deployments.apps
deployments.extensions
endpoints
events
events.events.k8s.io
horizontalpodautoscalers.autoscaling
ingresses.extensions
ingresses.networking.k8s.io
jobs.batch
leases.coordination.k8s.io
limitranges
mutatingwebhookconfigurations.admissionregistration.k8s.io
namespaces
networkpolicies.extensions
networkpolicies.networking.k8s.io
nodes
persistentvolumeclaims
persistentvolumes
poddisruptionbudgets.policy
pods
podsecuritypolicies.extensions
podsecuritypolicies.policy
podtemplates
priorityclasses.scheduling.k8s.io
replicasets.apps
replicasets.extensions
replicationcontrollers
resourcequotas
rolebindings.rbac.authorization.k8s.io
roles.rbac.authorization.k8s.io
runtimeclasses.node.k8s.io
secrets
serviceaccounts
```

```
services
statefulsets.apps
storageclasses.storage.k8s.io
validatingwebhookconfigurations.admissionregistration.k8s.io
volumeattachments.storage.k8s.io
```

Editing resources

Edit a service

```
kubectl edit service <service>
```

Edit a service with your favorite text editor

```
KUBE_EDITOR="vim" edit service <service>
```

Note: Change `service` by any editable resource type like pods.

Deleting Resources

Delete a resource using the type and name specified in `<file>`

```
kubectl delete -f <file>
```

Delete pods and services with same names

```
kubectl delete pod,service <name1> <name2>
```

Delete pods and services with a custom label

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

Delete all pods and services in a namespace

```
kubectl -n <namespace> delete pods,services --all
```

Delete all resources in a namespace

```
kubectl delte <namespace>
```

All get commands

```
kubectl get all
kubectl get pods
kubectl get replicasets
kubectl get services
kubectl get nodes
kubectl get namespaces
kubectl get configmaps
kubectl get endpoints
```

Abbreviations / Short forms of resource types

Resource type	Abbreviations
componentstatuses	cs
configmaps	cm
daemonsets	ds
deployments	deploy
endpoints	ep
event	ev
horizontalpodautoscalers	hpa
ingresses	ing
limitranges	limits
namespaces	ns
nodes	no
persistentvolumeclaims	pvc
persistentvolumes	pv
pods	po
podsecuritypolicies	psp
replicasets	rs
replicationcontrollers	rc
resourcequotas	quota
serviceaccount	sa
services	svc

Verbose Kubectl

```
kubectl run nginx  --image=nginx --v=5
```

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=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.

(Table source: K8s docs)

Cluster

Display addresses of the master and services

```
kubectl cluster-info
```

Dump cluster state to STDOUT

```
kubectl cluster-info dump
```

Dump cluster state to a file

```
kubectl cluster-info dump --output-directory=</file/path>
```

Compares the current cluster state against the state that the cluster would be in if the manifest was applied

```
kubectl diff -f ./my-manifest.yaml
```

List all images running in a cluster

```
kubectl get pods -A -o=custom-columns='DATA:spec.containers[*].image'
```

Kubectl context

Show merged kubeconfig settings

```
kubectl config view
```

Use multiple kubeconfig

```
KUBECONFIG=~/.kube/config1:~/.kube/config2:~/.kube/config3
```

Get a list of users

```
kubectl config view -o jsonpath='{.users[*].name}'
```

Display the first user

```
kubectl config view -o jsonpath='{.users[].name}'
```

Get the password for the "admin" user

```
kubectl config view -o jsonpath='{.users[?(@.name == "admin")].user.password}'
```

Display the current context

```
kubectl config current-context
```

Display list of contexts

```
kubectl config get-contexts
```

Set the default context to `<cluster>`

```
kubectl config use-context <cluster>
```

Sets a user entry in kubeconfig

```
kubectl config set-credentials <username> [options]
```

Sets a user with a client key

```
kubectl config set-credentials <user> --client-key=~/.kube/admin.key
```

Sets a user with basic auth

```
kubectl config set-credentials --username=<username> --password=<password>
```

Sets a user with client certificate

```
kubectl config set-credentials <user> --client-certificate=<path/to/cert> --
embed-certs=true
```

Set a context utilizing a specific config file

```
kubectl config --kubeconfig=<config/path> use-context <cluster>
```

Set a context utilizing a specific username and namespace.

```
kubectl config set-context gce --user=cluster-admin --namespace=foo \
  && kubectl config use-context gce
```

Alias

Create an alias on *nix

```
alias k=kubectl
```

Create an alias on Windows

```
Set-Alias -Name k -Value kubectl
```

Kubectl imperative (create) vs declarative (apply)

Create

You tell your cluster what you want to create, replace or delete, not how you want you it to look like.

```
kubectl create -f <filename|url>

kubectl delete deployment <deployment-name>
kubectl delete deployment <deployment-filename>
kubectl delete deployment <deployment-url>
```

Apply

You tell your cluster how you want it to look like.

The creation, deletion and modification of objects is done via a single command. The declarative approach is a statement of the desired end result.

```
kubectl apply -f <filename|url>

kubectl delete -f <deployment-filename>
kubectl apply -f <deployment-filename>
```

If the deployment is deleted in `<deployment-filename>`, it will also be deleted from the cluster.