kubernetes

Kubernetes provides a command line tool for communicating with a Kubernetes cluster's control plane, using the Kubernetes API.
This tool is named **kubectl.**

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

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

Context & Configuration:

Select the Kubernetes cluster that kubectl will communicate with and modify configuration data for:

(i) `$ kubectl version:`

Display the Kubernetes version running on the client and server

(ii) `$ kubectl config view:`

Show Merged kubeconfig settings and get the configuration of the cluster.

(iii) `$ kubectl config use-context <cluster-name>:`

Set the default context.

iv)
```
$ kubectl config set-cluster <cluster-name>:
```
Set a cluster entry in the kubeconfig

Create Objects:

YAML or JSON can be used to define Kubernetes manifests. You can use the `.yaml`, `.yml`, and `.json` file extensions.

i)
```
$ kubectl apply -f <file-name.yaml>:
```
Apply a configuration to an object by filename or stdin. Overrides the existing configuration.

ii)
```
$ kubectl create deployment nginx --image=nginx:
```
Start a single instance of nginx

iii)
```
$ kubectl explain <resource>:
```
Documentation of resources.

View & Find Resources:

Using the kubectl get all command, we can list every resource in a namespace, including pods, services, stateful sets, and other objects.

i)
```
$ kubectl get <resource-type>:
```
Display one or many resources

ii `$ kubectl <resource-type> <resource-name>:`

List a particular resource type

iii `$ kubectl get pods --all-namespaces:`

List all pods in all namespaces

iv `$ kubectl get pods -o wide:`

List all pods in the current namespace, with more details

v `$ kubectl describe <resource-type> <resource-name>:`

Show details of a specific resource or group of resources

vi `$ kubectl get pods --show-labels:`

Show labels for all pods (or any other Kubernetes object that supports labelling)

Update Resource:

You may manage your application deployment using a number of capabilities provided by Kubernetes, including scaling and upgrading.

i `$ kubectl rollout history deployment/frontend:`

Check the history of deployments including the revision

ii `$ kubectl rollout undo deployment/frontend:`

Rollback to the previous deployment

iii
```
$ kubectl rollout undo deployment/frontend
--to-revision=2:
```
Rollback to a specific revision

iv
```
$ kubectl rollout restart deployment/frontend:
```
Rolling restart of the "frontend" deployment

v
```
$ kubectl replace --force -f <file>:
```
Force replace, delete and then re-create the resource. Will cause a service outage.

vi
```
$ kubectl expose rc nginx --port=80 --target-port=8000:
```
Create a service for a replicated nginx, which serves on port 80 and connects to the containers on port 8000

vii
```
$ kubectl label pods my-pod new-label=awesome:
```
Add a Label

viii
```
$ kubectl label pods my-pod new-label-:
```
Remove a label

Scale Resources:

A Kubernetes cluster is updated by adding or deleting nodes, which is known as scaling.

i
```
$ kubectl scale --replicas=3 rs/foo:
```
Scale a replicaset named 'foo' to 3

ii
```
$ kubectl scale --replicas=3 -f foo.yaml:
```
Scale a resource specified in "foo.yaml" to 3

iii
```
$ kubectl scale --current-replicas=2
--replicas=3 deployment/mysql:
```
If the deployment named mysql's current size is 2, scale mysql to 3

iv
```
$ kubectl scale --replicas=5 rc/foo rc/bar rc/baz:
```
Scale multiple replication controllers

Delete Resources:

The simplest method of deleting any resource in Kubernetes is to use the specific manifest file used to create it.

i
```
$ kubectl delete -f <file>:
```
Delete a pod using the type and name specified in file

ii
```
$ kubectl delete pod unwanted --now:
```
Delete a pod with no grace period

Interacting with Resources:

When interacting with resources in Kubernetes, one may manage and orchestrate containerized applications effectively by using declarative YAML manifests to specify desired states and the Kubernetes API to add, change, or remove resources.

i.
```
$ kubectl logs [-f] [-p] <resource-type>
[-c CONTAINER]:
```
Dump pod logs

ii.
```
$ kubectl run nginx --image=nginx -n mynamespace:
```
Start a single instance of nginx pod in the namespace of mynamespace

iii.
```
$ kubectl attach <resource-type> [-c CONTAINER]:
```
Attach to Running Container

iv.
```
$ kubectl port-forward my-pod 5000:6000:
```
Listen on port 5000 on the local machine and forward to port 6000 on my-pod

v.
```
$ kubectl exec <resource-type> [-c CONTAINER]
-- COMMAND [args...]:
```
Interactive shell access to a running pod (1 container case)

vi.
```
$ kubectl top pod <POD_NAME> --containers:
```
Show metrics for a given pod and its containers

(vii) `$ kubectl cordon my-node:`

Mark my-node as unschedulable

(viii) `$ kubectl drain my-node:`

Drain my-node in preparation for maintenance

(ix) `$ kubectl uncordon my-node:`

Mark my-node as schedulable

(x) `$ kubectl top node my-node:`

Show metrics for a given node

(xi) `$ kubectl cluster-info:`

Display addresses of the master and services

Resource Type:

List of resource types that are supported, along with their short-names, API group, namespacing status, and kind:

(i) `$ kubectl api-resources`

Formatting output:

The -o (or --output) parameter should be added to a supported kubectl command in order to output details in a particular format to your terminal window.

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 a comma separated list of custom columns
`-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