Alias

alias k=kubectl	
k api-resources	
k explain node	explain is introspection documentation
k explain node.spec	

Pods

k get po	
k get po -o wide	
k get po <pod name> -o yaml	output in yaml format
k get po <pod-name> -o yaml > pod.yaml	output in yaml format in a file
k get pods -n <pod-name>	
k get pods --all-namespaces	
kubectl get pods -A	shorter version of `--all- nam espaces`
kubectl get events -A | grep error	all events in all namespaces with errors
k get pod --show-labels	
k get pod -l <key>=<value>	
k describe po <pod name>	
k delete po <pod name>	
k delete po --all	
k apply -f <pod.yaml> -n <namespace-name>	
k edit po <pod-name>	
k run nginx --image=nginx --dry-run=client -o yaml > pod.yaml	
k exec [POD] -- [COMMAND]	
k exec nginx -- ls /	
k exec -it nginx -- /bin/sh	Get a shell to the container running in your Pod

If you are not given a pod definition file but a pod, you may extract the definition to a file.

In a multicontainer pod, containers are created and destroyed together.

initContainers is a property under spec and it has same sub properties like name, image etc.

kubectl run command creates the pod and not deployment. There is no imperative command like kubectl create po for pod creation.

Pods have a property called restartPolicy whose values can be Always, Never or OnFailure

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com

Replica set

k apply -f <replicaset-definition.yaml>	
k get rs	
k get rs -o wide	
k get rs -o yaml	
k delete rs <replicaset-name>	
k scale rs <replicaset-name> --replicas=6	
k edit rs <replicaset-name>	
k describe rs <replicaset-name>	
k delete po <rs-po-name>	Delete all pods under rs
k get rs <rs-name> -o yaml > rs.yaml	

Editing RS
- Either delete and re-create the ReplicaSet or
- Update the existing ReplicaSet and then delete all PODs, so new ones with the correct image will be created.

The value for labels in spec.selector clause and spec.template.metadata should match in replicaset.

ConfigMap

k get cm	
k describe cm <cm-name>	to check key-value pairs in config map
k create cm <cm-name> --from-file=<path to file>	colon or equals to as delimiter between keys and values
k create cm <cm-name> --from-file=<directory>	
k create cm <cm-name> --from-literal=<key1>=<value1> --from-literal=<key2>=<value2>	

Properties to remember:
configmapkeyref / env, configMapRef / envFrom, volume

Pods can consume ConfigMaps as environment variables or as configuration files in a volume mounted on one or more of its containers for the application to read.
When ConfigMap is created from a file (k create cm <cm name> --from-file=) and when that ConfigMap is mounted as volume,
then the entire file is available at mount point for the pod.

Injected into the Pod.

Secret

k get secrets	
k get secret --all-namespaces	
k describe secret <secret-name>	This shows the attributes in secret but hides the values.
k get secret <secret-name> -o yaml	To view the values (encoded).
k create secret generic <secret-name> --from-literal=<key1>=<value1> --from-literal=<key2>=<value2>	
k create secret generic <secret-name> --from-file=<path to file>	
echo -n 'string' | base64	

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com

Secret (cont)

echo -n 'encoded string' | base64 --decode

Properties to remember: secretkeyref / env, secretref / envFrom, volume

A secret can be injected into a pod as file in a volume mounted on one or more of its containers or as container environment variables.

While creating secret with the declarative approach (yaml), you must specify the secret key and value in encoded format.
When we create secret using imperative approach, secret keys and values are encoded on their own (and decoded as well).

Taints (on nodes) and Tolerations (on pods)

k taint nodes [node_name] <key>=<value>:[NoSchedule|NoExecute|Prefer-NoSchedule]	Taint effect defines what happens to pods that do not tolerate this taint.
k taint no node01 spray=mortein:NoSchedule	
k describe nodes node01 | grep -i "taint"	To check taints on a node

The property tolerations under spec has properties like key, operator, value and effect and their values come inside ""

Remove the taint on master, which currently has the taint effect of NoSchedule
$ k taint no master node-role.kubernetes.io/master:NoSchedule-

Remove from node 'foo' all the taints with key 'dedicated'
$ k taint no foo dedicated-

Horizontal Pod Auto Scaler

k get hpa

k delete hpa <hpa-name>

k autoscale deploy nginx --min=5 --max=10 --cpu-percent=80

Monitoring

k top no

k top po

k top pod --namespace=default | head -2 | tail -1 | cut -d " " -f1

k top po --sort-by cpu --no-headers

Events

k get events

k get events -n kube-system

k get events -w

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com

DaemonSets

k get ds

k get ds --all-namespaces

k describe ds <ds-name> -n <ns-name>

k describe ds <ds-name> -o yaml

Environment Variables

k run nginx --image=nginx --env=app=web # Create an nginx pod and set an environment variable

env and envFrom property is an array.
env property takes two properties - name and value. value takes only string and will always come in double quotes.

Service

k apply -f <svc.yaml>

k get svc

k get svc --show-labels

k get svc -o wide

k get svc -o yaml

k describe svc <service-name> To get to know about port, target port etc.

k delete svc <service-name>

Node

k get no

k get no -o wide

k describe no <node-name>

Network Policy

k get netpol

k describe netpol <name>

Note: While creating network policy, make sure that not only network policy is applied to the correct object but also that it allows access from (ingress) / to correct object (egress).

labels and selectors are used.

An empty podSelector selects all pods in the namespace.

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com

Annotations

k annotate po nginx desc="Hello World"	
k annotate po nginx author=Avnish	
k annotate po nginx desc-	Remove this annotation from the pod
k annotate no <node-name>	

Labels & Selectors

k get [pod|deploy|all] --show-labels	Show labels
k get label [node|pod|deploy|etc] <key>=<value>	Syntax
k label pod nginx env=lab	To label pod
k label deploy my-webapp tier=frontend	To label deployment
k label node node01 size=large	To label nodes
k label po nginx <key>-	Remove the label
k label po nginx env=lab1 --overwrite	To overwrite a label
k get po --selector=app=App1	
k get po --selector=app!=App1	
k get all --selector=env=prod	
k get po --selector=env=prod,bu=finance,tier=frontend	Equivalent of && in programming languages

Volumes

k get pv
k describe pv
k delete pv <pv-name>
k get pvc
k describe pvc
k delete pvc <pvc-name>

Ingress

k get ingress
k describe ingress <ingress name>
k edit ingress <ingress name>
k apply -f <ingress.yaml>

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com

Logging

k logs -f <pod-name> <container-name>	Follow the logs
k logs <pod-name> --previous	Dump pod logs for a previous instantiation of a container
k logs --tail=20 <pod-name>	

Deployments

k create deploy nginx --image=nginx --replicas=2	
k get deploy	
k get deploy -n <namespace-name>	
k get deploy <deployment-name>	
k get deploy <deployment-name> -o yaml | more	
k get deploy -o wide	
k describe deploy <deployment-name>	
k apply -f deploy.yaml	
k apply -f deploy.yaml --record	Record the change-cause in revision history
k rollout status deploy <deployment-name>	
k rollout history deploy <deployment-name>	
k rollout history deploy nginx --revision=2 # to get detailed history for a specific revision.	
k rollout undo deploy <deployment-name>	
k rollout undo deploy <deployment-name> --to-revision=1	
k rollout pause deploy <deployment-name>	
k rollout resume deploy <deployment-name>	
k delete deploy <deployment-name>	
k create deploy nginx --image=nginx --dry-run=client -o yaml > deploy.yaml	
k scale deploy <deployment-name> --replicas=3 # To scale up / down a deployment. Not recorded in revision history.	
k create deploy <deployment-name> --image=redis -n <namespace-name>	
k edit deploy <deployment-name> -n <namespace-name>	
k expose deploy <deployment-name> --port=80 --type=NodePort|ClusterIp	
k autoscale deployment <deployment-name> --max 6 --min 3 --cpu-percent 50	Autoscale deployment

Deployments can be paused and resumed. When paused, no changes are recorded to revision history.

RollingUpdateStrategy in define upto how many pods can be down/up during the update at a time.
spec.strategy.type==RollingUpdate|Recreate
spec.strategy.rollingUpdate.maxUnavailable
spec.strategy.rollingUpdate.maxSurge

updating images or new deployements, triggers rollout. A new rollout creates a new deployment revision.

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com

Service Account

k create sa <sa-name>	
k get sa	
k get sa <sa-name> -o yaml	
k describe sa <sa-name>	Fetch token: gives secret name
k describe secret <secret-name>	Fetch token: gives token stored in secret
k run nginx --image=nginx --serviceaccount=myuser --dry-run=client -o yaml > pod.yaml	nginx pod that uses 'myuser' as a service account

When we use service account inside the pod, the secret for that service account is mounted as volume inside the pod.

Property to remember: spec -> serviceAccountName # set at pod level
Injected into the Pod.
For a deployment, can set service account in pod template.

A user makes a request to API server through k using user account.
A process running inside a container makes a request to API server using service account.
A service account just like user account has certain permissions.

CronJobs

k get cj	
k create cj busybox --image=busybox --schedule="*/1 *" -- /bin/sh -c "date; echo Hello from Kubernetes cluster"	cron job with image busybox that runs on a schedule and writes to standard output

In a cronjob, there are 2 templates - one for job and another for pod.

In a cronjob, there are 3 spec sections - one for cronjob, one for job and one for pod (in order).

Properties to remember: spec -> successfulJobHistoryLimit, spec -> failedJobHistoryLimit

Jobs

k create job busybox --image=busybox -- /bin/sh -c "echo hello;sleep 30;echo world"	
k get jobs	
k logs busybox-qhcnx	pod under job
k delete job <job-name>	

restartPolicy defaults to Never.

Job has 2 spec sections - one for job and one for pod (in order).

restartPolicy has to be OnFailure or Never. If the restartPolicy is OnFailure, a failed container will be re-run on the same pod. If the restartPolicy is Never, a failed container will be re-run on a new pod.

Job properties to remember: completions, backoffLimit, parallelism, activeDeadlineSeconds, restartPolicy.

By default Pods in a job are created in sequence

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com

Namespace	
k get ns	
k get ns -o yaml	
k create ns <namespace-name>	
k apply -f <namespace.yaml>	
k get po -n kube-system	
k describe ns <namespace-name>	
k delete ns <namespace-name>	
k config set-context $(k config current-context) --namespace=dev	To switch to a namespace permanently
k run redis --image=redis -n <namespace-name>	Create/run in a specific ns
k exec -it <pod-name> -n <namespace-name> -- sh	
k create ns <namespace-name> --dry-run=client -o yaml	

By **mansourj**
cheatography.com/mansourj/

Not published yet.
Last updated 1st November, 2023.

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com