Namespace

Namespace

- In Kubernetes, namespaces provides a mechanism for isolating groups of resources within a single cluster.
- Names of resources need to be unique within a namespace, but not across namespaces. Namespace-based scoping is applicable only for namespaced objects (e.g. Deployments, Services, etc)
- not for cluster-wide objects (e.g. StorageClass, Nodes, PersistentVolumes, etc)

Check the scope using kubectl cmd

kubectl api-resources namespaced=true

```
root@ubuntu20:/etc/kubernetes/manifests# kubectl api-resources --namespaced=true
NAME                        SHORTNAMES   APIVERSION                     NAMESPACED   KIND
bindings                                 v1                             true         Binding
configmaps                  cm           v1                             true         ConfigMap
endpoints                   ep           v1                             true         Endpoints
events                      ev           v1                             true         Event
limitranges                 limits       v1                             true         LimitRange
persistentvolumeclaims      pvc          v1                             true         PersistentVolumeClaim
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
controllerrevisions                      apps/v1                        true         ControllerRevision
daemonsets                  ds           apps/v1                        true         DaemonSet
deployments                 deploy       apps/v1                        true         Deployment
replicasets                 rs           apps/v1                        true         ReplicaSet
```

When to Use Multiple Namespaces

- Namespaces are intended for use in environments with many users spread across multiple teams, or projects.
- Namespaces provide a scope for names. Names of resources need to be unique within a namespace, but not across namespaces.
- Namespaces cannot be nested inside one another
- Avoid creating namespaces with the prefix kube-, since it is reserved for Kubernetes system namespaces.

Default Namespace

```
kubectl get namespace
```

```
NAME              STATUS   AGE
default           Active   1d
kube-node-lease   Active   1d
kube-public       Active   1d
kube-system       Active   1d
```

Kubernetes starts with four initial namespaces:

- `default` The default namespace for objects with no other namespace
- `kube-system` The namespace for objects created by the Kubernetes system

kubectl cmds for namespace

- Get the summary of namespace
  - kubectl get namespaces <name>
- Describe the specific namespace
  - kubectl describe namespaces <name>
- Create namespace
  - kubectl create namespace <insert-namespace-name-here>
- Delete namespace
  - kubectl delete namespaces <insert-some-namespace-name>

  Note: This deletes everything under the namespace!

Yaml file for name space

1. Create a new YAML file called `my-namespace.yaml` with the contents:

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: <insert-namespace-name-here>
```

Then run:

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

ResourceQuota

- ResourceQuota sets aggregate quota restrictions enforced per namespace

  apiVersion: v1

  kind: ResourceQuota

  metadata (ObjectMeta)

  spec (ResourceQuotaSpec)

Yaml file for object

```
root@ubuntu20:~# cat mem-cpu-quota.yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: mem-cpu-demo
spec:
  hard:
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
    limits.memory: 2Gi
```

Create and map object

Create the ResourceQuota:

```
kubectl apply -f https://k8s.io/examples/admin/resource/quota-mem-cpu.yaml --namespace=quota-mem-cpu-exa
```

View detailed information about the ResourceQuota:

```
kubectl get resourcequota mem-cpu-demo --namespace=quota-mem-cpu-example --output=yaml
```