ABOUT KUBERNETES:

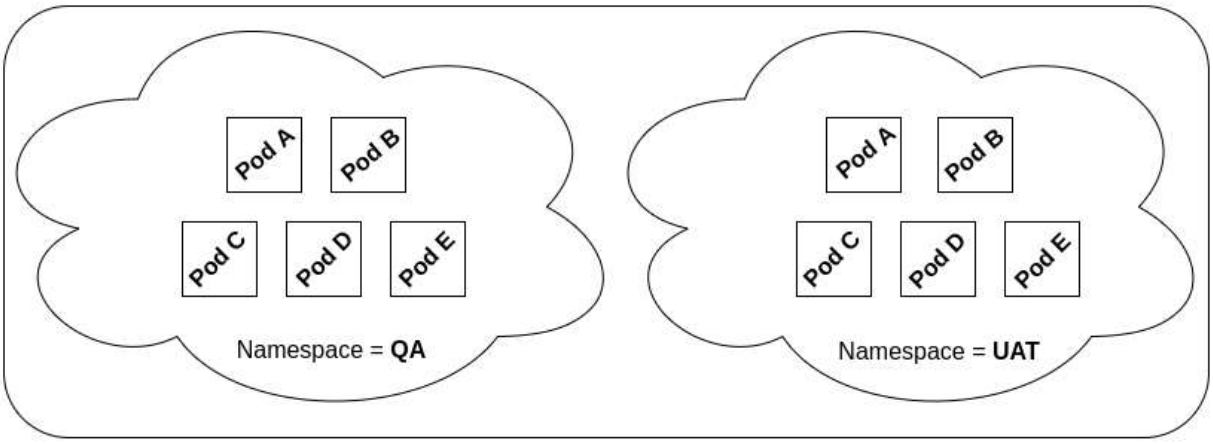
Kubernetes is an open-source tool that manages deployment, scaling, and orchestration of containerized applications. It groups containers that make up an application into logical units for easy management and discovery.

WHAT ARE NAMESPACES?

Kubernetes namespaces can be seen as a logical entity used to represent cluster resources for usage of a particular set of users. This logical entity can also be termed as a virtual cluster. One physical cluster can be represented as a set of multiple such virtual clusters (namespaces). The namespace provides the scope for names. Names of resources within one namespace need to be unique.

WHAT A KUBERNETES NAMESPACE DOES?

Namespace sets up virtual clusters over the same physical cluster. Rather than the need to set up an entire physical cluster for a new environment. We can set up several different virtual cluster over the same physical cluster using namespaces.



Kubernetes Cluster

By default, Kubernetes starts with the following three namespaces:

- **Default**: Catch-all namespace for all objects not belonging to either of the kube-public or kube-system namespaces. The default namespace is used to hold the default set of pods, services, and deployments used by the cluster.
- **Kube-public**: Namespace for resources that are publicly available/readable by all
- **Kube-system**: Namespace for objects/resources created by Kubernetes systems

WORKING WITH KUBERNETES NAMESPACES:

The following command can be used to get a list of all namespaces:

```
kubectl get namespaces
```

The following command displays namespaces with labels:

```
kubectl get namespaces --show-labels
```

In real-world scenarios, you can create a namespace for your development (dev), testing (QA), and production (prod) environments. The objects in the dev/QA namespaces such as pods, services, and deployments will be available for developers/testers respectively to build and run applications.

There will be lesser restrictions on modifying the resources in the dev/QA namespaces.

In the production (prod) namespace, there will be greater control on who can manage the resources.

In this article, we will look into the creation/deletion of namespaces for dev/QA/prod.

HOW TO CREATE NAMESPACES:

Let's create namespaces for 0ur development/QA/prod environments. The following would be required to be done:

- Create JSON files representing the namespaces
- Execute the kubectl command for creating namespaces

Create JSON Files Representing Namespaces:

Development namespace: Save the filename as namespace-dev.json:

```
{
  "kind": "Namespace",
  "apiVersion": "v1",
  "metadata": {
    "name": "dev",
    "labels": {
      "name": "dev"
    }
  }
}
```

QA Namespace: Save the filename as namespace-qa.json:

```
{
  "kind": "Namespace",
  "apiVersion": "v1",
  "metadata": {
    "name": "qa",
    "labels": {
      "name": "qa"
    }
  }
}
```

Production Namespace: Save the filename as namespace-prod.json:

```
{
  "kind": "Namespace",
  "apiVersion": "v1",
  "metadata": {
    "name": "prod",
    "labels": {
      "name": "prod"
    }
  }
}
```

EXECUTE THE KUBECTL COMMAND FOR CREATING NAMESPACES:

The following commands will create the namespaces for Dev/QA/Prod environments:

```
# Namespace for Developers
kubectl create -f namespace-dev.json
# Namespace for Testers
kubectl create -f namespace-qa.json
# Namespace for Production
kubectl create -f namespace-prod.json
```

Check whether the namespaces got created by executing the following command:

```
kubectl get namespaces --show-labels | grep name=
```

HOW TO WORK WITH NAMESPACES:

Once the namespaces have been created, in order to have kubectl commands work with a specific namespace, the following needs to be done:

- Assign a context to each namespace
- Switch to the appropriate context for working with that namespace

ASSIGN A CONTEXT TO EACH NAMESPACE:

```
# Assign dev context to development namespace
kubectl config set-context dev --namespace=dev --cluster=minikube --user=minikube
# Assign qa context to QA namespace
kubectl config set-context qa --namespace=qa --cluster=minikube --user=minikube
# Assign prod context to production namespace
kubectl config set-context prod --namespace=prod --cluster=minikube --user=minikube
```

SWITCH TO THE APPROPRIATE CONTEXT:

Execute the following commands to switch to the specific context. Once switched to a context, any execution of kubectl command would create/update/delete objects in that namespace.

```
# Switch to Dev context
kubectl config use-context dev
# Switch to QA context
kubectl config use-context qa
# Switch to Prod context
kubectl config use-context prod
```

Just to check what the current context is, execute the following command:

```
kubectl config current-context
```

Once switched to a context, commands such as the following would give object detail for that namespace:

```
kubectl get pods
kubectl get deployments
```

HOW TO DELETE NAMESPACES:

The commands below would delete the namespaces:

```
# Delete dev namespace
kubectl delete namespaces dev
# Delete qa namespace
kubectl delete namespaces qa
# Delete prod namespace
kubectl delete namespaces prod
```

LISTING PODS OF EACH NAMESPACE WITHOUT SWITCHING CONTEXT:

The pods of each namespace can be listed by following commands

kubectl get pods **–namespace=qa** or
kubectl get pods **–namespace=uat**

All resources created in a particular namespace can be listed using the parameter **"–namespace="** after any Kubernetes command.

The usage and implementation of a namespace is very simple but it solves an essential use case of segregating multiple environments without the need to launch multiple clusters.

This helps in reducing the **infrastructure cost and efforts to manage applications** on different environments.

CAVEATS:

While you can run staging and production environments in the same cluster and save resources and money by doing so, you will need to be careful to set up resource limits so that your staging environment doesn't starve production for CPU, memory, or disk resources. Setting resource limits properly, and testing that they are working takes a lot of time and effort so unless you can measurably save money by running production in the same cluster as staging or test, you may not really want to do that.

Whether or not you run staging and production in the same cluster, namespaces are a great way to isolate different apps within the same cluster. Namespaces will also serve as a level where you can apply resource limits so look for more resource management features at the namespace level in the future.

CONCLUSION:

The best example for **namespaces** lies in the Kubernetes base setup itself ☺