In Kubernetes, namespaces are an API construct to avoid naming collisions and represent a scope for object names. A good use case for namespaces is to isolate the objects by team or responsibility. In this lab, you will learn how to create, list, use, and delete namespaces.

Listing Namespaces
A Kubernetes cluster starts out with a couple of initial namespaces. You can list them with the following command:

kubectl get namespaces

The default namespace hosts object that haven't been assigned to an explicit namespace. Namespaces starting with the prefix kube- are not considered end-user namespaces. They have been created by the Kubernetes system. You will not have to interact with them as an application developer.

```
$ kubectl get namespaces
NAME              STATUS   AGE
default           Active   48s
kube-node-lease   Active   49s
kube-public       Active   49s
kube-system       Active   49s
$ A
```

Creating and Using a Namespace
You can create a namespace imperatively or declaratively. Choose just one approach.

Creating a Namespace Imperatively
To create a new namespace imperatively, use the create namespace command. The following command uses the name mystuff:

kubectl create namespace mystuff

```
$ kubectl create namespace mystuff
namespace/mystuff created
```

Creating a Namespace Declaratively
A manifest for the object in YAML form would look as follows:

```
apiVersion: v1
kind: Namespace
metadata:
  name: mystuff
```

To create the namespace from a YAML manifest file named namespace-mystuff.yaml, run the command as follows:

kubectl apply -f namespace-mystuff.yaml

Listing Namespaces
You can list all namespaces matching the name mystuff with the get namespace command:

kubectl get namespace mystuff

```
$ kubectl apply -f namespace-mystuff.yaml
Warning: resource namespaces/mystuff is missing the kubectl.kubernetes.io/last-applied-configuration annotation which is required by kubectl apply. kubectl apply should on
ly be used on resources created declaratively by either kubectl create --save-config or kubectl apply. The missing annotation will be patched automatically.
namespace/mystuff configured
$ kubectl get namespace mystuff
NAME      STATUS   AGE
mystuff   Active   75s
$
```

Managing Objects in a Namespace
Once the namespace is in place, you can create objects within it. You can do so with the command-line option --namespace or its short form, -n. The following commands create a new Pod named nginx in the namespace mystuff

kubectl run nginx --image=nginx:1.23.0 --restart=Never -n mystuff

Use the get pods command to list the available Pods in the namespace:

kubectl get pods -n mystuff

```
namespace/mystuff configured
$ kubectl get namespace mystuff
NAME       STATUS   AGE
mystuff    Active   75s
$ kubectl run nginx --image=nginx:1.23.0 --restart=Never -n mystuff
pod/nginx created
$ kubectl get pods -n mystuff
NAME    READY   STATUS              RESTARTS   AGE
nginx   0/1     ContainerCreating   0          6s
```

If you want to interact with all namespaces (for example, to list all Pods in your cluster), you can pass the --all-namespaces flag. This includes Pods in the default namespace, the kube-system namespace, and the end-user namespace mystuff.

kubectl get pods --all-namespaces

```
$ kubectl get pods --all-namespaces
NAMESPACE     NAME                                       READY   STATUS    RESTARTS   AGE
kube-system   coredns-64897985d-5zn9z                    1/1     Running   0          5m44s
kube-system   coredns-64897985d-brj8p                    1/1     Running   0          5m44s
kube-system   etcd-controlplane                          1/1     Running   0          6m
kube-system   kube-apiserver-controlplane                1/1     Running   0          5m57s
kube-system   kube-controller-manager-controlplane       1/1     Running   0          5m57s
kube-system   kube-flannel-ds-76g46                      1/1     Running   0          5m38s
kube-system   kube-flannel-ds-j4sr6                      1/1     Running   0          5m43s
kube-system   kube-proxy-6hxqz                           1/1     Running   0          5m44s
kube-system   kube-proxy-6jwzc                           1/1     Running   0          5m38s
kube-system   kube-scheduler-controlplane                1/1     Running   0          6m
mystuff       nginx                                      1/1     Running   0          57s
$
```

Deleting a Namespace
Deleting a namespace has a cascading effect on the object existing in it. Deleting a namespace will automatically delete its objects:

kubectl delete namespace mystuff

kubectl get pods -n mystuff

```
$ kubectl delete namespace mystuff
namespace "mystuff" deleted
$ kubectl get pods -n mystuff
No resources found in mystuff namespace.
$
```

Using a Namespace with a Context

If you want to change the default namespace more permanently, you can use a context. This gets recorded in a kubectl configuration file, usually located at $HOME/.kube/config. This configuration file also stores how to both find and authenticate to your cluster. For example, you can create a context with a different default namespace for your kubectl commands using:

kubectl config set-context my-context --namespace=mystuff

This creates a new context, but it doesn't actually start using it yet. To use this newly created context, you can run:

kubectl config use-context my-context

Contexts can also be used to manage different clusters or different users for authenticating to those clusters using the --users or --clusters flags with the set-context command.

```
No resources found in mystuff namespace.
$ kubectl config set-context my-context --namespace=mystuff
Context "my-context" created.
$ kubectl config use-context my-context
Switched to context "my-context".
$
```

kubectl is a powerful tool for managing your namespaces in a Kubernetes cluster. You can create, delete, list, and inspect namespaces. Once a namespace has been established, you can create objects inside of the namespace. Deleting a namespace will cascade delete all of its objects.

In Kubernetes, a context is a named cluster, user, and namespace tuple that is used to define the target Kubernetes cluster for kubectl commands. A context includes the credentials for accessing the cluster and the namespace to operate on within the cluster.

When a user runs a kubectl command, the kubectl client uses the current context to determine which cluster to target and which namespace to operate within. By changing the current context, users can easily switch between different clusters and namespaces without having to manually specify the cluster and namespace for each command.

In the context of a namespace, when a user creates or modifies Kubernetes objects within a namespace, they must have the appropriate permissions to do so. The permissions are typically defined using Kubernetes Role-Based Access Control (RBAC), which provides fine-grained control over who can perform specific actions on Kubernetes objects within a namespace.

Additionally, pod security policies, as shown in the example YAML file in a previous question, can also be used to enforce security policies on pods running within a namespace. By defining pod security policies at the namespace level, administrators can ensure that pods running in that namespace meet certain security requirements, such as restrictions on host network access or privilege escalation.