**Lab 02 A - Kubernetes Fundamentals**

**Viewing namespaces**

Let's take a look at the Namespaces before we create our own.

kubectl get ns

We'll see the namespaces that make up the **minikube** Kubernetes cluster, as follows:

NAME STATUS AGE

default Active 92m

ingress-nginx Active 4m53s

kube-node-lease Active 92m

kube-public Active 92m

kube-system Active 92m

You'll notice the **ingress-nginx** namespace which is the **add-on** we added in the previous lab.

**Creating namespaces**

Namespaces are self contained environments to separate your applications within a cluster.

**Namespaces using manifests**

Kubernetes provides 2 ways to create namespaces. Most are created using **manifests** which are YAML files containing Kubernetes API instructions, and allow you to version control your Kubernetes projects, and a starting point for being able to understand a topic you may come across later called **Helm**.

Let's create a namespace called **myfirstapp**. Note the namespace must be all lowercase to conform to the DNS naming convention. If you use any non-valid DNS characters then your configuration will fail.

1. First we will need to create the following file called **myfirstapp.yml**:

apiVersion: v1

kind: Namespace

metadata:

name: myfirstapp

labels:

name: myfirstapp

The key lines are:

* apiVersion: v1
  + This defines what version of the API we are working to in our file syntax
* kind: Namespace
  + Tells Kubernetes that we are working with a **Namespace** resource.
* name: myfirstapp
  + Underneath the **metadata** defines what the namespace will be called when this configuration is applied.

To create the namespace we will use a text editor called **nano** to add the text to the file.

1. Run the following command to enter the text editor:

nano myfirstapp.yml

You will now be in **nano**, where the bottom of the screen shows actions which can be accessed by pressing the **CTRL** key and the letter or character following the **^** symbol. For example **CTRL G** will get help on nano, and the menu will change. **CTRL X** will quit the help and put you back to the text editor.

1. Type in the code:

apiVersion: v1

kind: Namespace

metadata:

name: myfirstapp

labels:

name: myfirstapp

NOTE: Make sure you only use spaces. The above uses 2 spaces for each indent.

This file exists in the Lab02 folder.

* Save the file by pressing **CTRL X**
* Press Y to accept the save
* Press **ENTER** to accept the file name of **myfirstapp.yml**
* Run the **ls** command and you'll see your file.

1. Now we need to apply the namespace to our cluster. Run the following command to do that:

kubectl apply -f myfirstapp.yml

This will return:

namespace/myfirstapp created

On successful creation, and you'll notice the name we requested is shown.

1. Verifying the namespace in the cluster run:

kubectl get ns

You should see **myfirstapp** in the list.

If you do not run the **apply** command again.

1. What happens if you run this command again:

kubectl apply -f myfirstapp.yml

Kubernetes is **idempotent**, and will only update the configuration if something has changed. The word **unchanged** let's us know there is no difference between our code and the Kubernetes system.

1. Let's edit the namespace file and add an extra label and apply it so open the file:

nano myfirstapp.yml

1. After the last line and at the same level of indentation for the **name** label add the following:

app: myapp

1. Save the file by pressing **CTRL X**, **Y**, **ENTER**
2. Apply the change:

kubectl apply -f myfirstapp.yml

You will see the following to let you know it updated:

namespace/myfirstapp configured

The **configured** word let's us know it has been applied and changed.

**Namespaces using the command**

You can also run a simple command to create a namespace, but this leads to issues in maintaining a stable cluster. Running ad-hoc commands means we don't know what's been applied to our systems, and Kubernetes is normally controlled using automation applied using DevOps methodologies.

However, we will briefly explore the commands below just so you can see the flexibility of the kubectl CLI.

1. Create a new namespace using the following command:

kubectl create namespace anotherapp

In the above case you created a namespace called **anotherapp**, but note we cannot label it in this command.

Since it is not being created by a config file, we have to run other commands to configure it, so for example, to label it, you would need to run a command.

1. Run the following command to label your namespace.

kubectl label namespaces anotherapp name=anotherapp app=anotherapp

1. Finally, to verify that your namespace is set up, run the following command:

kubectl get ns

This lists the namespaces and you can see your new **anotherapp** namespace now added to the list.