DEVOPS FOR AIOT
SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING, SINGAPORE POLYTECHNIC

LABORATORY 9: DOCKER & KUBERNETES IN LINUX

Objectives

By the end of the laboratory, students will be able to

- Deploying an official docker images as a pod
- Creating and configuring a deployment in kubernetes
- Declarative configuration using YAML files in Kubernetes

Activities

- Docker commands in Linux
- Kubernetes commands in Linux

Review

- Linux Virtual Machine is setup in VMware Workstation

Equipment:

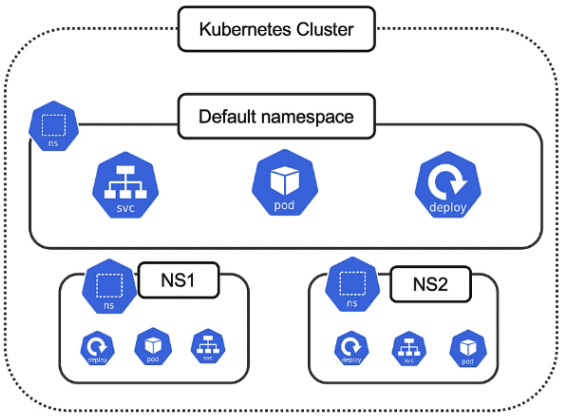
- Windows OS laptop
- VMware Workstation 16
- Linux VM
- Docker
- Kubernetes K3S in Linux

1 Kubernetes in Linux

For this lab exercise, we will run "kubectl" CLI in a Linux environment running in a Virtual Machine in VMware Workstation to communicate with Kubernetes cluster's control plane for different configurations.

1.1. Kubernetes Namespaces

Kubernetes provides Namespaces to add additional scope to the naming of the pods, deployments that must be unique within a specific namespace but can be reused with similar names across other different namespaces.



- A way to divide cluster resources via multiple users

NS = Namespace

For more information on Kubernetes Namespaces, refer to the URL below.

https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/

Exercise

- Create a new namespace called "myflask-app-nspace" and write down the command used in the box below:

```
Kubectl create namespace myflask-app-nspace
```

- Check that the Kubernetes namespace is successful created by using the appropriate "kubectl" command and write the command used in the box below:

```
Kubectl get namespace
```

1.1. Creating a Pod via Kubernetes CLI with Docker images

Docker hub provides official images for essential repositories (for example, ubuntu, nginx) that serve as a starting point for majority of users. These images are good for new users as they provide clear documentation and are designed for the most common use cases.

Exercise

Create a new pod called "**nginx-test**" on the **namespace** that was created in previous exercise with image "**nginx**". check if success:kubectl get pods --namespace=myflask-app-nspace
nginx-test 1/1 Running 0 30s

```
kubectl run nginx-test --image=nginx --namespace=myflask-app-nspace
```

1.2. Creating a Deployment via Kubernetes CLI

In the previous lab we created Kubernetes Deployments using the command below,

```
kubectl create deployment my-flask-app --image={your namespace}/flask-app
```

Exercise

- Using the Linux VM terminal run the same command above and check if that deployment is running.

- Write down the command used to check if the Kubernetes Deployment is running in the box below.

```
kubectl get deployments
```

- If the deployment is not running, create the deployment with the command given above.

- Execute the kubectl command to expose the port 5000 in the Kubernetes Deployment "my-flask-app" to the host machine and write the command used in the box below.

```
kubectl expose deployment my-flask-app --type=NodePort --port=5000
```

-delete the deployment my-flask-app. Write the 'kubectl' command used in the box below.

```
kubectl delete deployments my-flask-app
```

- Check if the Kubernetes service performing the port forwarding is still running.
- Write the "kubectl" command used in the box below.

```
kubectl get svc
```

- You will notice that the Kubernetes service is still running although the deployment has been deleted.

- Delete the Kubernetes service created to run in the port 5000 and write the "kubectl" command used in the box below.

```
kubectl delete services my-flask-app
```

1.3. Kubernetes Pod Environment Variables

Environment variables allow pods to be configured from the host operating system.

https://kubernetes.io/docs/tasks/inject-data-application/define-environment-variable-container/

Exercise

- Using kubectl, run a command that will create and run a pod named "env-pod" from image "nginx" which maps the environment variables envar1 to a value 10. Write the command used in the box below.

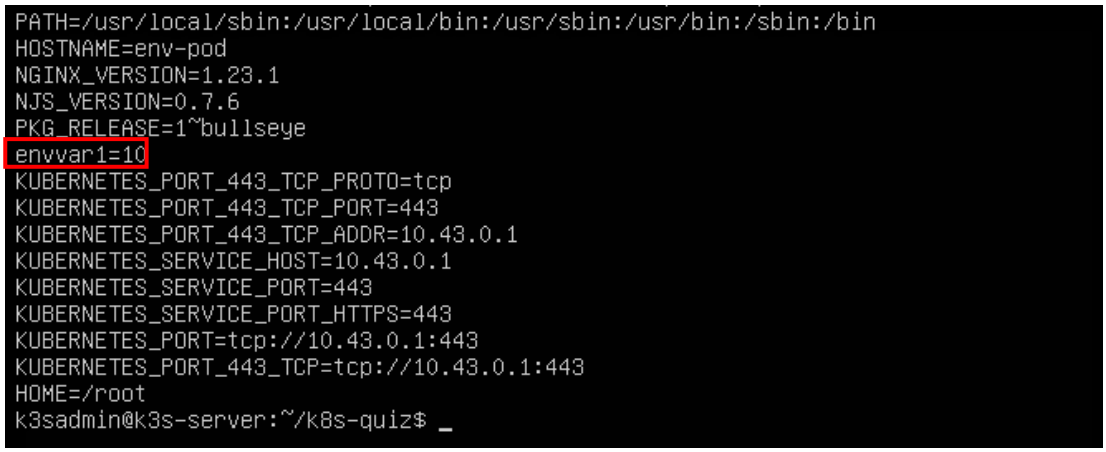
```
kubectl run env-pod --image=nginx --env="envar1=10"
```

- Verify that the Pod and the associated Environment Variables are created and mapped. Write the "kubectl" command used in the box below.

```
kubectl exec env-pod -- env
```

Note: There is a space in between -- and env in the command above.

In the terminal output below, you should see the list of environment variables inside the Pod “env-pod”.



1.4. Creating Secrets in Kubernetes

Secrets are used in Kubernetes to create key/value pair data based on the base64 encoding.

https://kubernetes.io/docs/concepts/configuration/secret/

Using “kubectl”, create a Kubernetes secret “test-secret” based on the following key/value pair “password=ilovedevops”

```
kubectl create secret generic test-secret --from-literal=password=ilovedevops
```

After creating the secret, run a “kubectl” command to check if the secret has been successfully created by displaying the following console output.

Write the command used to check if the secret “test-secret” was correctly created in the box below.

```
kubectl get secret test-secret -o yaml
```

Notice that the key “password” is displayed as an encoded value based on base64.

Do take note:

"aWxvdmVkZXZvcHM=" is the encoded password.

To decode the secret value to plaintext, you can use the Linux command below:

```
echo {encoded password} | base64 -d
```

For the above example the command would be:

```
echo aWxvdmVkZXZvcHM= | base64 -d
```

1.5. Creating a Pod using YAML file

In the previous lab, we created Kubernetes Pods directly using the “kubectl” Command Line Interface (CLI).

Kubernetes also provides YAML files which allow Declarative Object Configuration to define and configure Kubernetes resources such as Pods, Deployments, etc.

https://kubernetes.io/docs/tasks/run-application/run-stateless-application-deployment/

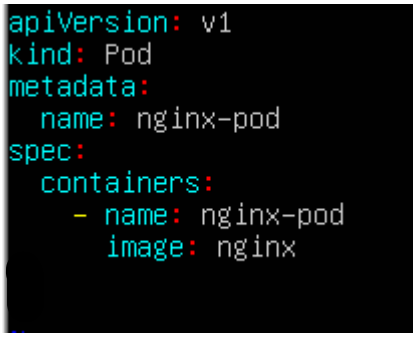
Run the example “Creating and exploring an nginx deployment” in the link above.

Exercise

- Create a new directory “yml’ in the directory path “/home/devops-admin/ET0735/lab9”

- Using the text editor, create a new YAML file “**pod.yml**”. Enter the YAML code below. Save the file in the newly created “yml” directory.

  pod.yml



```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
spec:
  containers:
    - name: nginx-pod
      image: nginx
```

don't use tab-> if here i put two spacing, so next time when i want tab, i use two spacing

- Run pod.yml using the command below

```
kubectl apply -f pod.yml
```

Note: This command runs the pod.yml file.

- Check if pod is running and write down the command used in the box below.

```
kubectl get pods
```

You should see that a pod with name "nginx-pod" has been created.

- Delete the pod "**nginx-pod**" and write down the command used in the box below.

  kubectl delete pod nginx-pod

1.6. Creating Configmap using YAML file

Write the command to create a configmap named "**my-configmap**" with the following key/value pairs:

1. var2=val2

```
kubectl create secret generic my-configmap --from-literal=var2=val2
```

Create new nginx pod named "**nginx-configmap**" using the image "**nginx**" and inject the value from variable "**var2**" as an environment variable within the container. The key of the environment variable within the container is named "**my_var**". Refer to the template below and fill up the blanks accordingly. Create a file named **lab9-configmap.yml** with the text below:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-configmap
spec:
  containers:
    - name: nginx-configmap
      image: nginx
      env:
        # Define the environment variable
        - name: ___
          valueFrom:
            configMapKeyRef:
              name: my-configmap
              # Specify the key associated with the value
              key: ___
```

Run the YML file "lab9-configmap.yml" using the command below:

```
kubectl apply -f lab9-configmap.yml
```

You should see the system return the message:

pod/nginx-configmap created.

After that, verify that a new pod "nginx-configmap" has been created using the command :

```
kubectl get pod
```

1.7. Creating persistentVolume using YAML file

Create a PersistentVolume of ***2Gi***, named "**lab9-pv**". Here are the properties of lab9-pv:

1. Access Mode of *"**ReadWriteOnce**"*
2. Mounted on hostPath **"*/tmp/lab9*"**

Refer to the template below and fill up accordingly. Create a file named lab9-pv.yml with the text below:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: lab9-pv
spec:
  capacity:
    storage: __
  accessModes:
  - ___
  hostPath:
    path: "___"
```

Run the YML file "lab9-pv.yml" using the command below:

```
kubectl apply -f lab9-pv.yml
```

You should see the system return the message:

PersistentVolume/lab9-pv is created

After that, verify that a new PersistentVolume "lab9-pv" has been created using the command:

```
kubectl get pv
```