|  |
| --- |
| Kubenetes FUNDAMENTALS - NETWORKING  September 13, 2022 |
|  |

Contents

[Common commands and general guide 1](#_Toc113826038)

[Resources 1](#_Toc113826039)

[Helpful tips 1](#_Toc113826040)

[LAB INTRODUCTION 3](#_Toc113826041)

[Kubernetes Basics 5](#_Toc113826042)

[Lab 1.1 – kubectl & kubeconfig 5](#_Toc113826043)

[LAB 1.2 – yaml files and pods 5](#_Toc113826044)

[LAB 1.3 – Deployments 7](#_Toc113826045)

[LAB 1.4 – Namespaces 10](#_Toc113826046)

[LAB 1.5 – services 13](#_Toc113826047)

[Networking in Kubernetes 18](#_Toc113826048)

[Lab 1.6 – Container Network Interface (CNI) 18](#_Toc113826049)

[Lab 1.7 – DNS 20](#_Toc113826050)

[Lab 1.8 – ClusterIP 22](#_Toc113826051)

[Lab 1.9 – NodePort 24](#_Toc113826052)

# Common commands and general guide

## Resources

|  |  |
| --- | --- |
| **Cheat sheet** | <https://kubernetes.io/docs/reference/kubectl/cheatsheet/> |
| **UDF Lab Link** | Visit link in email for UDF labs |
| **Materials** | In the SharePoint link provided |
| **Lab Env** | Access Master Node and use web shell  Use su ubuntu to access kubernetes |
| **K8s version** | 1.24.3 (use command kubectl version) |
| **Recommended Browser** | Chrome |

## Helpful tips

|  |  |
| --- | --- |
| **View next command** | Use **Tab** to view available commands after keyword |
| **Pasting of commands in web shell (chrome)** |  |
| **Dry Run** | --dry-run=client |
| **Verify after execution** | get pods / XXXX and if error, describe XXXX |
| **View various forms of output** | -o wide  -o yaml  -o json  -o template |
| **View object in all namespaces** | --all-namespaces |
| **View object in a specific namespace** | --namespace=*<namespace>*  -n *<namespace>* |
| **View help window** | kubectl help |
| **Switch User** | su *<username>* |
| **Run in privilege mode** | sudo *<command>* |
| **Unix Text Editor** | vim *<filename>* |
| **Navigating between directory** | cd .. or cd |
| **View directory** | ls |
| **Clear screen** | clear |
| **Backup file** | cp *<source file> <destination file>* |

# LAB INTRODUCTION

|  |  |
| --- | --- |
| **Starting the lab** |  |
|  |  |
| **Wait for the lab to spin up** | If color is yellow, wait. As the lab will differ in bootup time, Presenter will share the lab bootup time. |
| **Accessing Web Shell** | Once all components are green, you may start the lab. To web shell into a K8s node, click the access button and select web shell from the drop down    Access the **master node** of the K8s cluster through web shell, key in command below to switch user to ubuntu    change the directory to home directory instead of root directory, issue command below  cd ~  You will find few yaml files listed in home directory, issue command below  ls |

# Kubernetes Basics

## Lab 1.1 – kubectl & kubeconfig

|  |  |
| --- | --- |
| **Objective** | Organizing cluster access |
| **Pre-requisite** | - |
| **Steps** | On the K8 Control Plane, execute the following commands. The following commands are to retrieve the **nodes** and **pods** available within the cluster.  kubectl get nodes  kubectl get nodes -o wide  kubectl get pods  kubectl get pods -o wide |
| **References** |  |

Letter

Description automatically generated with low confidence

## LAB 1.2 – yaml files and pods

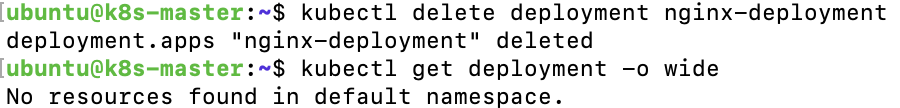
|  |  |
| --- | --- |
| **Objective** | Creating POD using YAML file |
| **Estimated Completion Time** | 5 mins |
| **Pre-requisite** |  |
| **Steps** | Yaml files have been uploaded and change path to /home/ubuntu and you will find yaml files as below.    Follow step 1 – 3 to define a yaml file and subsequently deploy it. Optionally, you may skip the step 1 and 2 and proceed directly to step 3 you can choose to deploy using the yaml file that has been uploaded on to /home/ubuntu directory.  **Step 1**: On the K8s Control Plane, execute the following command: *[Note: do not use the same name for the yaml file that has already been created]*  vi pod2.yaml  **Step 2**: Copy and paste content below. We are specifying a pod YAML configuration file with parameter of   1. image: nginx 2. name: nginx   for pod creation.  apiVersion: v1  kind: Pod  metadata:  name: nginx  labels:  name: nginx  spec:  containers:  - name: nginx  image: nginx  Enter command below to save and exit VI  :wq!  **Step 3**: Issue command below to verify pods created and then delete the pod.  cat pod.yaml  kubectl apply -f pod.yaml  kubectl get pods  kubectl get pods -o wide  kubectl delete pods nginx |
| **References** |  |
|  |  |

## LAB 1.3 – Deployments

|  |  |
| --- | --- |
| **Objective** | Creating deployment |
| **Pre-requisite** | - |
| **Steps** | Follow step 1 – 3 to define a yaml file and subsequently deploy it. Optionally, you may skip the step 1 and 2 and proceed directly to step 3 you can choose to deploy using the yaml file that has been uploaded on to /home/ubuntu directory.  **Step 1**: On the K8 Control Plane, execute the following commands. *[Note: do not use the same name for the yaml file that has already been created]*  vi deployment2.yaml  **Step 2**: Copy and paste content below. We are specifying a deployment YAML configuration file with parameter of:   1. Replica of 3 2. image: nginx:1.14.2 3. name: nginx 4. ports: 80   for pod creation.  apiVersion: apps/v1  kind: Deployment  metadata:  name: nginx-deployment  labels:  app: nginx  spec:  replicas: 3  selector:  matchLabels:  app: nginx  template:  metadata:  labels:  app: nginx  spec:  containers:  - name: nginx  image: nginx:1.14.2  ports:  - containerPort: 80  Enter command below to save and exit VI  :wq!  Step 3: Issue command below to verify pods created and then delete the deployment.  cat deployment.yaml  kubectl apply -f deployment.yaml  kubectl get deployment -o wide  kubectl describe deployment nginx-deployment  kubectl delete deployment nginx-deployment |
| **References** |  |
|  |  |
|  |  |

Text, letter

Description automatically generated



## LAB 1.4 – Namespaces

|  |  |
| --- | --- |
| **Objective** | Creating namespace |
| **Pre-requisite** | - |
| **Steps** | Follow step 1 – 3 to define a yaml file and subsequently deploy it. Optionally, you may skip the step 1 and 2 and proceed directly to step 3 you can choose to deploy using the yaml file that has been uploaded on to /home/ubuntu directory.  **Step 1:** On the K8 Control Plane, execute the following commands. *[Note: do not use the same name for the yaml file that has already been created]*  vi namespace2.yaml  **Step 2:** Copy and paste content below. We are specifying a namespace and pod YAML configuration file with parameter of:   1. namespace: production 2. image: nginx 3. name: mypod   for pod creation.  apiVersion: v1  kind: Namespace  metadata:  name: production  labels:  name: production  ---  apiVersion: v1  kind: Pod  metadata:  name: mypod  namespace: production  labels:  name: mypod  spec:  containers:  - name: mypod  image: nginx  Enter command below to save and exit VI  :wq!  **Step 3:** Issue command below to verify pods created  cat namespace.yaml  kubectl apply -f namespace.yaml  kubectl get pods  kubectl get pods -n production  kubectl describe pods mypod -n production  kubectl delete pods mypod -n production  You may need to change from the uploaded yaml file name to the yaml file name you have created if you have gone through Step 1 to 3 during pod creation. |
| **References** | Table  Description automatically generated |
|  |  |

## LAB 1.5 – services

|  |  |
| --- | --- |
| **Objective** | Creating service |
| **Pre-requisite** | - |
| **Steps** | Follow step 1 – 3 to define a yaml file and subsequently deploy it. Optionally, you may skip the step 1 and 2 and proceed directly to step 3 you can choose to deploy using the yaml file that has been uploaded on to /home/ubuntu directory.  **Step 1:** On the K8 Control Plane, execute the following commands. *[Note: do not use the same name for the yaml file that has already been created]*  vi service2.yaml  **Step 2:** Copy and paste content below. We are specifying a service and pod YAML configuration file with parameter of:   1. image: nginx 2. name: nginx 3. service type: NodePort 4. nodePort: 30080   for pod creation.  apiVersion: v1  kind: Pod  metadata:  name: nginx  labels:  name: nginx  spec:  containers:  - name: nginx  image: nginx  ---  apiVersion: v1  kind: Service  metadata:  name: nginx  labels:  name: nginx  spec:  type: NodePort  ports:  - port: 80  nodePort: 30080  name: http  - port: 443  nodePort: 30443  name: https  selector:  name: nginx  Enter command below to save and exit VI  :wq!  **Step 3:** Issue command below to verify pods created  kubectl apply -f service.yaml  kubectl get pods  kubectl get service  kubectl describe pods  kubectl describe service  Curl from jumphost  Curl http://[worker node IP]:30080 |
| **References** | Text, letter  Description automatically generated with medium confidence  Table  Description automatically generated  Table  Description automatically generated  Table  Description automatically generated  Text, letter  Description automatically generated |

# Networking in Kubernetes

## Lab 1.6 – Container Network Interface (CNI)

|  |  |
| --- | --- |
| **Objective** | Within a node, CNI will automatically provision IP Addresses for the Pods |
| **Pre-requisite** | - |
| **Steps** | On the K8 Control Plane, execute the following commands  Create a pod. The control plane should create a pod on the available worker node.  kubectl run alpine --image=busybox42/alpine-pod --restart=Never  Check if the pod is created successfully (Status: Running) and note the worker node as well as the IP Address. The parameter -o wide reveals the extended details of the pods which includes the Pod IP Address as well as Node  kubectl get pods alpine -o wide  Web shell to the work node and do a ping on the alpine’s Pod IP Address.  ping -c4 <your pod IP address>  The ping should succeed. You have successfully communicated with a pod with its auto assigned IP Address.  **Bonus**: You can choose to simulate a production pod life cycle and delete the pods  kubectl delete pods alpine  And recreating them. You will find that IP Addresses changes each time. Therefore, there is a need to have a service that acts as a proxy for these ephemeral pods. |
| **References** |  |
|  |  |

## Lab 1.7 – DNS

|  |  |
| --- | --- |
| **Objective** | K8 supports DNS to translate Name to IP Address |
| **Pre-requisite** | Lab 1.5 - Service, the service named nginx is created |
| **Steps** | Retrieve pod details of nginx  kubectl get pods nginx  Remember, a pod is like a mini–Virtual Machine. You can shell into the pod  kubectl exec -ti nginx -- /bin/bash  The pod name is not the fully qualified name. We can identify the fully qualified name by locating DNS resolution file  cat /etc/resolv.conf  Let’s test it out. Install and use nslookup on the fully qualified service namespace (service name + namespace). Notice the name server is as declared in the DNS resolution file, take note of the translated IP Address which we will verify against the service details later.  apt-get update  apt-get install busybox  busybox nslookup nginx.default.svc.cluster.local  Let us verify the results with the service details. Exit back to control plane and inspect the service. You will notice that the Service details matches the results from the nslookup.  exit  kubectl describe svc nginx  **Bonus**: What about the name server, who is exposing the translating service? (Hint: it is an out of the box service) |
| **References** |  |
|  |  |
|  |  |
|  |  |
|  |  |

## Lab 1.8 – ClusterIP

|  |  |
| --- | --- |
| **Objective** | K8 Networking. K8 supports ClusterIP which provides communication between pods within the same cluster |
| **Pre-requisite** | Unix Jump Host with k8 Controlplane  ClusterIP service setup  Worker 1:  busybox42(a)  busybox42(b) |
| **Steps** | On the K8 Control Plane, let us create a file  sudo vim cluster-ip.yaml  Let us create Pod and Service using the following yaml configuration  Notice that up to now, our files are not step by step instructions, we configure with the end state in mind. That is because Kubernetes configurations are **Declarative** in nature. The usual practice is to deploy via files as it is repeatable as well as we can version control. Also note that we can deploy several K8s object in a single file *(demarcated by the three dashes---)*  Copy the details into the file and run the deployment file using the command  kubectl apply -f cluster-ip.yaml  **Yaml File details**  kind: Deployment  metadata:  apiVersion: apps/v1  kind: Deployment  metadata:  name: cluster-ip-deployment  labels:  app: cluster-ip-deployment  spec:  replicas: 2  selector:  matchLabels:  app: nginx-app  template:  metadata:  labels:  app: nginx-app  spec:  containers:  - name: nginx-hello  image: nginxdemos/hello:plain-text  ports:  - containerPort: 80  ---  apiVersion: v1  kind: Service  metadata:  name: cluster-ip-service  spec:  type: ClusterIP  selector:  app: nginx-app  ports:  - name: http  port: 80 # ClusterIp Service Port  Let’s test the ClusterIP Service. We can consume the service with a curl command by doing the following (1) Find a pod in the same node. [*If there are no pods in the same node, feel free to create more pods via commands, we have learnt this in Lab 1.6 step 1*] (2) we will shell into the chosen pod and do a curl.  Retrieve node information  kubectl get pods -o wide  Shell into a different pod of the same worker node.  kubectl exec -ti nginx -- /bin/bash  Ping / curl the ClusterIP service to simulate internal cluster service consumption  curl cluster-ip-service  You can also curl the service’s IP address directly. Inspect the service and note the IP Address and Port. Do note that the target port is used to select the containerPort of the Pod, which is declared in the deployment  exit  kubectl describe service  kubectl exec -ti nginx -- /bin/bash  curl <Service IP Address:Port> (e.g. 10.244.126.28:80)  **Bonus**: try curling the other pod in the different worker node. Does it work for you? This shows that direct pod-to-pod communication will only work within the same node.  **Tip**: you can run the following command to reverse the yaml file execution. **Don’t** do it unless you intend to reverse to troubleshoot failure, we still require ClusterIP Service to compare differences with NodePort.  kubectl delete -f cluster-ip.yaml |
| **References** | |  |
|  | |  |
|  | |  |

## Lab 1.9 – NodePort

|  |  |
| --- | --- |
| **Objective** | K8 supports NodePort which provides communication between pods of different clusters |
| **Pre-requisite** | Unix Jump Host with k8 Controlplane  Service without NodePort – Lab 1.9 – service: cluster-ip-deployment  Service with NodePort – use Lab 1.5 – service nginx |
| **Steps** | **Let’s attempt to access a service type ClusterIP as a consumer outside the cluster**. We have a virtual machine that is outside the cluster, let’s use that.  Previously in Lab 1.9, a service has already been created with ClusterIP. On the K8 controlplane, execute the following commands to retrieve the IP Address and port number. Copy it  kubectl describe service cluster-ip-service  Open new tab and web shell into the lab’s Jumphost / Client, try consuming the service with that IP Address and port number  curl <Cluster-IP Service IP Address:Port> (e.g. 10.244.126.28:80)  you should expect to receive error: *Failed to connect XXX. Connection timeout*  **Now, let’s attempt to access a service type NodePort as a consumer outside the cluster**  Previously in Lab 1.5, a service has already been created with NodePort.  Back in the master node web shell, Inspect the service, retrieve the communication details for us to consume.  Retrieve the port number from the **NodePort** from the service **and IP Address from the nodes’s internal ip address.** The pod should be created in both worker nodes so either node will work fine.  kubectl get nodes -o wide  kubectl describe service nginx  if service is not created, please use the following command and inspect again  kubectl expose pod nginx --type=NodePort --name=nginx-service --port=80  On the Jumphost / Client / controlplane, execute the following commands:  curl <Node IP Address:NodePort> (e.g. 10.1.1.7:30080) |
| **References** | |  |
|  | |  |
|  | |  |
|  | |  |
|  | |  |