**KUBERNETES NOTE JOEL**

1. Create an IAM AWS user with the following access

* AmazonEC2FullAccess
* AmazonRoute53FullAccess
* AmazonS3FullAccess
* AmazonVPCFullAccess

NOTE: Save the Access and Security Keys that are generated.

1. Install Pyhton 3.7 and install awscli

* $ sudo yum install python3-pip
* $ pip3 install awscli

1. Install Kops

# Linux  
curl -LO https://github.com/kubernetes/kops/releases/download/$(curl -s https://api.github.com/repos/kubernetes/kops/releases/latest | grep tag\_name | cut -d '"' -f 4)/kops-linux-amd64  
chmod +x kops-linux-amd64  
sudo mv kops-linux-amd64 /usr/local/bin/kop

1. Configure AWS by providing the access and secrete keys

$ aws configure

AWS Access Key ID [None]: AccessKeyValue  
AWS Secret Access Key [None]: SecretAccessKeyValue  
Default region name [None]: us-east-1  
Default output format [None]:

1. Create and AWS bucket

bucket\_name=imesh-kops-state-store (replace imesh with your bucket name)   
aws s3api create-bucket \ (use space in between and remove \)  
--bucket ${bucket\_name} \  
--region us-east-1

1. Save the following commands in the .bash\_profile file

$ vi .bash\_prifle

Paste the below commands under PATH  
export KOPS\_CLUSTER\_NAME=imesh.k8s.local  
export KOPS\_STATE\_STORE=s3://${bucket\_name}

1. The commands in linux engine

export KOPS\_CLUSTER\_NAME=imesh.k8s.local  
export KOPS\_STATE\_STORE=s3://${bucket\_name}

1. Create Kubernetes Cluster using Kops

kops create cluster \  
--node-count=2 \  
--node-size=t2.medium \  
--zones=us-east-1a \  
--name=${KOPS\_CLUSTER\_NAME}

1. Generate ssh key when asked

$ ssh-keygen -t -rsa

1. Run the following command

$kops create secret --name joel.k8s.local sshpublickey admin -i ~/.ssh/id\_rsa.pub

NOTE: Before we move further, we need to install “kubectl” otherwise it will give error

1. To install Kubectl use following command:

curl -o kubectl https://amazon-eks.s3-us-west-2.amazonaws.com/1.14.6/2019-08-22/*bin/linux/amd64/kubectl*

1. Apply execute permission to .kubectl

$ chmod +x ./kubectl

$ sudo mv ./kubectl /usr/local/bin/kubectl

1. Copy the binary to a folder in your PATH. If you have already installed a version of **kubectl**, then we recommend creating a $HOME/bin/kubectl and ensuring that $HOME/bin comes first in your $PATH.

$ mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$PATH:$HOME/bin

Check kubectl version

$ kubectl version

1. create the Kubernetes cluster on AWS by executing kops update command:

$ kops update cluster --name ${KOPS\_CLUSTER\_NAME} –yes

1. Run the following command to view the cluster

$ kops validate cluster

1. Getting the admin password

$ kops get secrets kube --type secret -oplaintext

MY KEY: rhTArBu0TWfCRG0xXvpbHOkVLkiF623s

1. Execute the below command to find the Kubernetes master hostname:

REFERENCES

<https://medium.com/containermind/how-to-create-a-kubernetes-cluster-on-aws-in-few-minutes-89dda10354f4>

<https://docs.aws.amazon.com/eks/latest/userguide/install-kubectl.html>

<https://kubernetes.io/docs/tasks/tools/install-kubectl/>

**KUBERNETES COMMANDS**

$ minikube version displays the minikube version installed

$ minkube start starts the cluster

$ kubectl version check if kubectl is installed

$ kubectl cluster-info check to see the details of a cluster

$ kubectl get nodes list the nodes that are created and can be used to host our applications

$ minkube dashboard open Kubernetes dashboard in a browser

$ kubectl create deployment creating a deployment to manage the pod

We need to provide the deployment name and app image location (include the full repository url for images hosted outside Docker hub).

Example

kubectl create deployment kubernetes-bootcamp --image=gcr.io/google-samples/kubernetes-bootcamp:v1

Great! You just deployed your first application by creating a deployment. This performed a few things for you:

* searched for a suitable node where an instance of the application could be run (we have only 1 available node)
* scheduled the application to run on that Node
* configured the cluster to reschedule the instance on a new Node when needed

$kubectl get deployments viewing the deployments

$ kubectl get pods view the pods

$

We now have a connection between our host (the online terminal) and the Kubernetes cluster. The proxy enables direct access to the API from these terminals.

You can see all those APIs hosted through the proxy endpoint. For example, we can query the version directly through the API using the curl command:

curl http://localhost:8001/version

**If Port 8001 is not accessible, ensure that the kubectl proxy started above is running.**

The API server will automatically create an endpoint for each pod, based on the pod name, that is also accessible through the proxy.

First we need to get the Pod name, and we'll store in the environment variable POD\_NAME:

export POD\_NAME=$(kubectl get pods -o go-template --template '{{range .items}}{{.metadata.name}}{{"\n"}}{{end}}') echo Name of the Pod: $POD\_NAME

*Note: Check the top of the terminal. The proxy was run in a new tab (Terminal 2), and the recent commands were executed the original tab (Terminal 1). The proxy still runs in the second tab, and this allowed our curl command to work using localhost:8001.*

In order for the new deployment to be accessible without using the Proxy, a Service is required which will be explained in the next modules.

When you created a Deployment in Module [2](https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-intro/), Kubernetes created a **Pod** to host your application instance. A Pod is a Kubernetes abstraction that represents a group of one or more application containers (such as Docker or rkt), and some shared resources for those containers. Those resources include:

* Shared storage, as Volumes
* Networking, as a unique cluster IP address
* Information about how to run each container, such as the container image version or specific ports to use

A Pod models an application-specific "logical host" and can contain different application containers which are relatively tightly coupled. For example, a Pod might include both the container with your Node.js app as well as a different container that feeds the data to be published by the Node.js webserver. The containers in a Pod share an IP Address and port space, are always co-located and co-scheduled, and run in a shared context on the same Node.

Pods are the atomic unit on the Kubernetes platform. When we create a Deployment on Kubernetes, that Deployment creates Pods with containers inside them (as opposed to creating containers directly). Each Pod is tied to the Node where it is scheduled, and remains there until termination (according to restart policy) or deletion. In case of a Node failure, identical Pods are scheduled on other available Nodes in the cluster.

Summary:

* Pods
* Nodes
* Kubectl main commands

*A Pod is a group of one or more application containers (such as Docker or rkt) and includes shared storage (volumes), IP address and information about how to run them.*

Pods overview

Nodes

A Pod always runs on a **Node**. A Node is a worker machine in Kubernetes and may be either a virtual or a physical machine, depending on the cluster. Each Node is managed by the Master. A Node can have multiple pods, and the Kubernetes master automatically handles scheduling the pods across the Nodes in the cluster. The Master's automatic scheduling takes into account the available resources on each Node.

Every Kubernetes Node runs at least:

* Kubelet, a process responsible for communication between the Kubernetes Master and the Node; it manages the Pods and the containers running on a machine.
* A container runtime (like Docker, rkt) responsible for pulling the container image from a registry, unpacking the container, and running the application.

*Containers should only be scheduled together in a single Pod if they are tightly coupled and need to share resources such as disk.*

Node overview

Troubleshooting with kubectl

In Module [2](https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-intro/), you used Kubectl command-line interface. You'll continue to use it in Module 3 to get information about deployed applications and their environments. The most common operations can be done with the following kubectl commands:

* **kubectl get** - list resources
* **kubectl describe** - show detailed information about a resource
* **kubectl logs** - print the logs from a container in a pod
* **kubectl exec** - execute a command on a container in a pod

You can use these commands to see when applications were deployed, what their current statuses are, where they are running and what their configurations are.

Now that we know more about our cluster components and the command line, let's explore our application.