Create single t2.medium machine in aws

sudo su

**########Now install docker###############**

sudo apt update && apt -y install docker.io

**###install Kubectl###**

curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl && chmod +x ./kubectl && sudo mv ./kubectl /usr/local/bin/kubectl

**###install Minikube####**

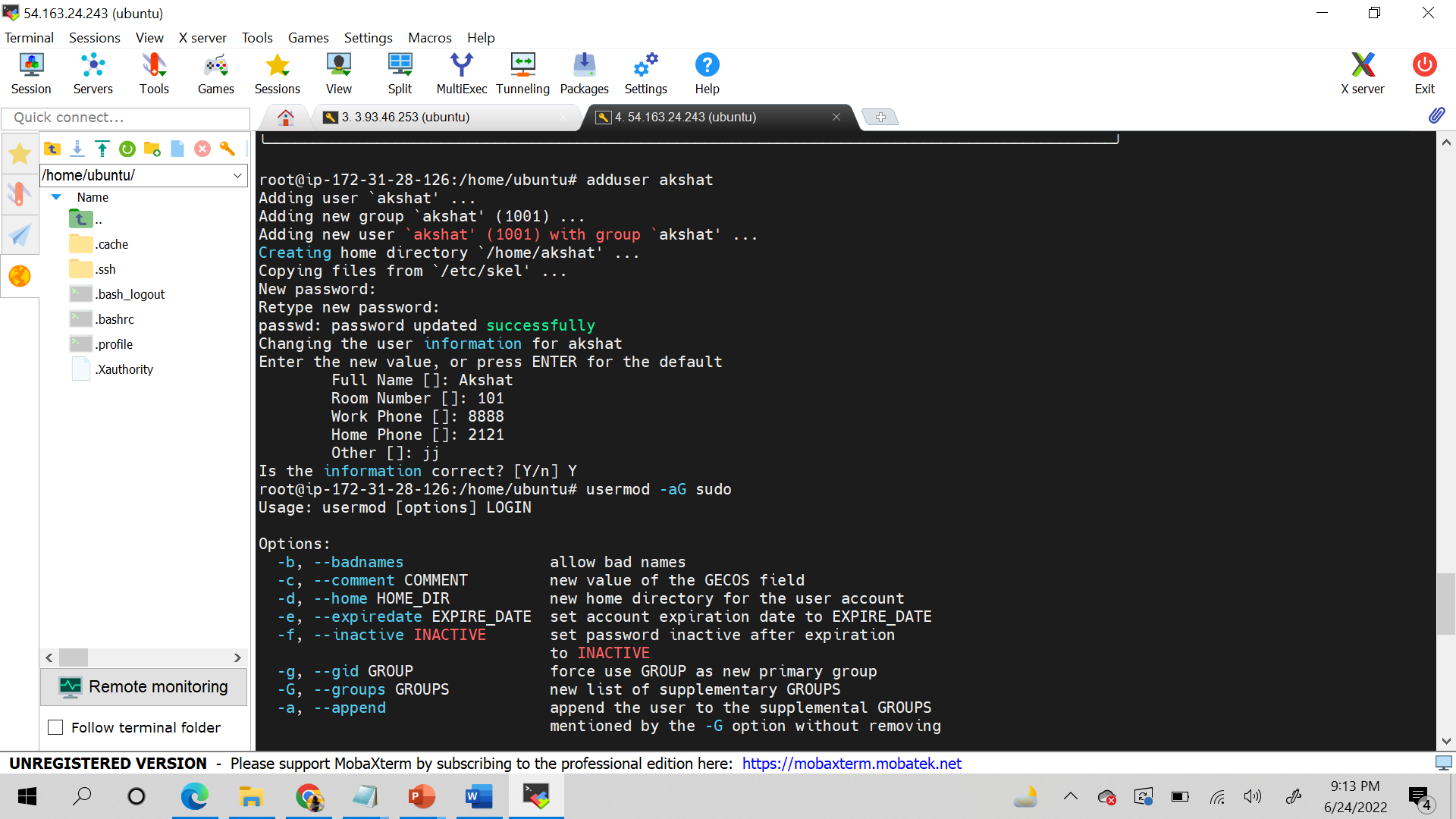
curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64 && chmod +x minikube && sudo mv minikube /usr/local/bin/

**##Start minikube####**

apt install conntrack

#Create user and add it to docker

adduser akshat



usermod -aG sudo akshat

su - akshat

sudo groupadd docker

sudo usermod -aG docker $USER && newgrp docker

minikube start --vm-driver=docker

minikube status

vi pod1.yml \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* press i \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

kind: Pod

apiVersion: v1

metadata:

name: testpod

annotations:

description: Our first testing pod

spec:

containers:

- name: c00

image: ubuntu

command: ["/bin/bash", "-c", "while true; do echo Test Message; sleep 5 ; done"]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Press esc and then :wq \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

###CONCEPTS OF ABOVE YAML FILE##########

RESTART POLICY:

The "restart policy" in Kubernetes refers to the behavior of the system

when a Pod (a group of one or more containers) terminates or crashes.

The possible values for the restart policy are:

>Always: The Pod will be restarted unconditionally.

>OnFailure: The Pod will be restarted only if it exits with a non-zero status.

>Never: The Pod will not be restarted.

The default restart policy is Always.

The restart policy can be specified in the Pod specification file using the "restartPolicy" field.

#######################################################

APIVERSION:

The "apiVersion" in Kubernetes refers to the version of the Kubernetes API that the resource type being used is compatible with.

For example, if you have a deployment resource, you would specify the "apiVersion" as "apps/v1" to indicate that you are using the v1 version of the apps API. The apiVersion is used by the Kubernetes API server to determine the appropriate code paths to execute when processing a request.

It is important to use the correct apiVersion for your resources, as it can affect the behavior and functionality of the resource. Newer versions of the API may add new fields or change the behavior of existing fields, so it's important to ensure that the version you are using is compatible with your intended use case.

Lets understand API Version in detail:

The "apiVersion" in Kubernetes refers to the version of the Kubernetes API that a resource type is compatible with. Each resource type in Kubernetes, such as a Deployment, a Service, or a ConfigMap, is part of a specific API group and has an associated version.

For example, the Deployment resource is part of the "apps" API group and the "v1" version of the API, so its "apiVersion" would be specified as "apps/v1".

The apiVersion field is specified in the YAML definition file for the resource, and is used by the Kubernetes API server to determine how to handle requests for the resource. The API server uses the apiVersion to select the appropriate code path to execute when processing a request for the resource.

It is important to use the correct apiVersion for your resources, as new versions of the API may add new fields or change the behavior of existing fields. For example, if you use an older version of the API, you may miss out on new features or bug fixes that have been added in a newer version.

Kubernetes provides backwards compatibility for a certain period of time to ensure that existing resources continue to work with newer versions of the API. However, it's recommended to update your resources to use the latest apiVersion to ensure that you have access to the latest features and bug fixes.

##########################################################

The command ["/bin/bash", "-c", "while true; do echo Test message; sleep 5; done"] is a shell script that runs an infinite loop in bash.

The command is executed as follows:

1. /bin/bash: This specifies the shell interpreter to use, in this case, bash.
2. -c: This option tells bash to run the command that follows as a shell script.
3. while true; do echo Test message; sleep 5; done: This is the actual shell script that is executed. The script contains a while loop that runs continuously. The loop outputs the message "Test message" to the console and then waits for 5 seconds using the "sleep" command before repeating.

This script is often used for testing purposes, for example, to check if a container is running or to continuously output log messages.

#####################################################################

kubectl apply -f pod1.yml

Now ,  
kubectl get pods

(you will find a pod with the name testpod)

kubectl get pods -o wide

kubectl describe pod testpod

kubectl logs -f testpod (to see what is running in container)

kubectl logs -f testpod -c c00 (c00 is the container name as mentioned in the pod1.yml file )

kubectl get nodes

kubectl exec testpod -it -c c00 -- /bin/bash (to enter inside the container)

exit

\*\*\*\*Delete a pod\*\*\*\*\*\*\*\*\*\*\*\*\*

kubectl delete pod testpod

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**\*\*\*\*\*\*\*\*MULTI CONTAINER POD ENVIRONMENT (If you want to create multiple container in a single pod)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*vi testpod3.yml \*\*\***

kind: Pod

apiVersion: v1

metadata:

name: testpod3

spec:

containers:

- name: c00

image: ubuntu

command: ["/bin/bash", "-c", "while true; do echo My first message; sleep 5 ; done"]

- name: c01

image: ubuntu

command: ["/bin/bash", "-c", "while true; do echo Hello-Devops; sleep 5 ; done"]

\*\*then escape and :wq\*\*

kubectl apply -f testpod3.yml

Now ,  
kubectl get pods

(you will find a pod with the name testpod)

kubectl get pods -o wide

kubectl describe pod testpod3

kubectl get pods testpod3 -o jsonpath='{.spec.containers[\*].name}' (list containers in pod)

kubectl logs -f testpod3 c00 (to see what is running in container)

kubectl logs -f testpod3 c01

kubectl get nodes

kubectl exec testpod3 -it -c c00 -- /bin/bash

(to enter inside the container)

exit

\*\*\*\*Delete a pod\*\*\*\*\*\*\*\*\*\*\*\*\*

kubectl delete pod testpod3

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**LAB \*\*\*\*\*\*\*EXAMPLE OF LABELS\*\*\*\*\*\*\*\*\*\***

vi pod7.yml

kind: Pod

apiVersion: v1

metadata:

name: testlabel

labels:

env: development

class: pods

spec:

containers:

- name: c00

image: ubuntu

command: ["/bin/bash", "-c", "while true; do echo Welcome to the training lets learn labels in pod; sleep 5 ; done"]

**\*\*Press escape and :wq \*\***

kubectl apply -f pod7.yml

kubectl get pods

kubectl get pods -o wide

kubectl get pods --show-labels

kubectl label pods testlabel env= (manual command to add a label to existing pod)

kubectl get pods –show -labels

kubectl get pods -l env=development

**\*\*\*\*\*\*\*LAB: EXAMPLE OF REPLICATION CONTROLLER\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**vi myrc.yml**

kind: ReplicationController

apiVersion: v1

metadata:

name: myreplica

spec:

replicas: 2

selector:

myname: akshat

template:

metadata:

name: testpod6

labels:

myname: akshat

spec:

containers:

- name: c00

image: ubuntu

command: ["/bin/bash", "-c", "while true; do echo Hello-Akshat; sleep 5 ; done"]

kubectl scale --replicas=10 rc -l myname=akshat

kubectl apply -f myrc.yml

kubectl get rc

kubectl describe rc myreplica

kubectl get pods

kubectl delete pod <<pod name>>

kubectl get pods

<<again you will see 2 pods as replication is enabled>>

Kubectl get rc

Kubectl describe rc myreplica

Kubectl get pods —-show-labels

kubectl scale --replicas=8 rc -l myname=akshat

(scale up)

Kubectl get rc

Kubectl get pods

kubectl scale --replicas=1 rc -l myname=akshat (scale down)

Kubectl get pods

Kubectl get rc

Kubectl get pods –show-labels

Kubectl delete -f myrc.yml

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**LAB 8: EXAMPLE OF REPLICA SET\*\*\*\*\*\*\*\*\*\*\*\***

kind: ReplicaSet

apiVersion: apps/v1

metadata:

name: myrs

spec:

replicas: 2

selector:

matchExpressions: # these must match the labels

- {key: myname, operator: In, values: [akshat, Akshat, Akshu]}

- {key: env, operator: NotIn, values: [production]}

template:

metadata:

name: testpod7

labels:

myname: Akshat

spec:

containers:

- name: c00

image: ubuntu

command: ["/bin/bash", "-c", "while true; do echo Happy new year; sleep 5 ; done"]