Page No.
Date

Exp 1 (Collaboration)

Steps

i) Go to IAM module
ii) On Dashboard Click on Users → Add Users
iii) Enter username for 1st then add another user and enter uname
iv) Give access type → Password → Custom password
Enter the custom password
v) Don't allow them to change password
vi) Next → Add users to grp
vii) Create group → Give grp name
viii) Then give rights to access cloud 9 i.e AWS Cloud9 Users
→ Create group.
ix) Next. Tags → Next Create Click on create users
x) Url will be generated copy and paste if somewhere
xi) Click on Url

xii) Sign with 1st user → Go to Cloud9 → Create environment
→ Give the name of environment, short description
xiii) Environment settings: Environ → Keep everything default
platform → Ubuntu 30 minutes time then → Next
Create environment

- Create a tab in incognito mode (or other browser)
- Copy and paste the link → sign with 2nd user
- Cloud 9
- Go to user 1 → share option → invite member using uname
Done
- Go to user 2 → dashboard of Cloud 9 → shared with u → open IDE

Page No.
Date

CICD

Exp 2

1-) Go to amazon elastic beanstalk. Create application.

2) Give name, choose platform -> tomcat, sample application everything else default then create app

3) Go to github -> repositories -> consist of yaml file

4) Create Codepipeline

-> create pipeline -> (name) -> everything default -> next
source -> Github (version 2).

Connection -> Connect to github -> only select repo
(watch record) -> Connect
Choose repos name, branch name else default -> next

5) Build provider -> AWS codeBuild
Create project -> name, OS -> Ubuntu, runtime -> standard
image -> any
Buildspec -> Use a buildspec file
else default -> continue to codepipeline

Singe build -> next

6) Deploy stage -> elastic beanstalk -> next

7) Review create pipeline

Page No.
Date

After pipeline is created successfully

Go the Deploy → elastic beanstalk → Url

→ Edit code it github and commit

Page No.
Date

Exp 8

1) Go to EC2 → Ubuntu 20.04 → 2 instance
Storage - Default → ~~Security grp~~ along
Security grp add with default
i) HTTP → port 80 source - Anywhere
ii) HTTPS → 443 " " " "
iii) Custom TCP → ~~8~~ 6443 " " " "

⇒ Launch instances, Name them :- K8s Master K8s Worker

> Now, select master and the connect
⇒ same with worker
→ clear master and worker, sudo su

> Now change host name of master :-
• hostnamectl set-hostname master-node
• ~~exec bash~~ then exit
• sudo su

→ Same with worker node

To change color (optional)
→ master → ~~tput setaf~~ tput setaf 02
worker → tput setaf 01

Now commands

→ apt-get update (both)
→ apt-get install docker.io -y (both)
→ docker --version (both) (to check version)

→ systemctl enable docker
→ systemctl status docker } (both)
→ systemctl ~~status~~ start docker

→ Add Kubernetes signing key (both)
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add

→ Add software Repositories (both)

~~apt-get~~ .
apt-add-repository "deb http://apt.kubernetes.io/ kubernetes-xenial main"

→ apt-get install kubeadm kubelet kubectl -y (both)
→ apt-mark hold kubeadm kubelet kubectl (both)
→ kubeadm version (both)

→ swapoff -a (both)

→ kubeadm init --pod-network-cidr=10.244.0.0/16 --ignore-preflight-errors=all (Master)

→ Now copy the output & get head paste it somewhere (your kubernetes secure)

→ then enter the 3 commands (master)

→ Then Deploy Pod Network to Cluster (master)

kubectl ~~ap~~ apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

→ kubectl get pods --all-namespaces (master)

→ Now copy paste from word document ((worker)
kubeadm join ..... (command)

→ kubectl get nodes (master)

3rd ~~ex~~ exp done

Page No.
Date

Exp 4

&

1) Create deployment named nginx

- Kubectl create deployment nginx --image = nginx

→ kubectl expose deploy nginx --port 80 --target-port 80
--type NodePort

→ kubectl get services

* Here we will get port number in ports column greater than 30000

* Now go to instances running select worker node instance then open public ip address

* then in browser to put : portnumber we get (preferable) of any (master or worker)

* This will not work, go to security grp and add another grp : custom TCP with ~~source~~ port number and source - Anywhere IPv4 Save rule.

* Now again open the browser from worker public ip address and further put : portnumber. Also remove https ka s as the site is not secured.

Page No.
Date

Scale up replicas

→ kubectl scale --current-replicas=1 --replicas=2 deployment/nginx

Output : you will see 2/2

→ kubectl describe deployment/nginx (gives detail of service deployed)

→ kubectl delete service nginx

→ Now refresh that page in web-browser u won't be able to access

→ To check if it has been deleted
→ kubectl get services

Page No.
Date

Exp 5.

→ First launch instance keeping everything default
→ then connect
→ sudo su, dnf
→ wget (then link address terraform site)

→ Then we need to unzip

→ ~~apt~~ - apt install unzip
→ then unzip the zip file using "unzip & filename

→ ls → we get list of files

→ Setting path → mv terraform /usr/local/bin/

→ terraform

Exp 11-12

AWS lambda Theory:

→ Code run on AWS lambda is called lambda function.

AWS supports serverless
:- we don't need to manage servers. AWS does it.

Step 1 :- Create I am role.

→ Go to IAM → roles → Create role

type → AWS service → Lambda → next
Permissions-policies
→ lambda, full key words search → Tick on AWS lambda - full acce
→ S3. full access tick
→ cloudwatch :- full access
.
→ next,

Role details :
→ Give Role name :- AWSdemo → Create role
↑ Now role is created

IAM is over

:→ ~~Now go to lambda~~

Page No.
Date

Now get to S3

Step2 : Create a bucket in AWS S3 to upload image.
Ensure region of bucket is same as that of lambda function

→ Go to S3 → Create bucket

→ Uncheck block all public access (we don't want to block public access)

→ Tick "I acknowledge"
→ Then click on Create bucket

Step3 : Create a lambda function

→ Go to lambda → Create Function
→ Use a blueprint
→ Filter blueprints → Blueprint attribute = S3
→ Now go with S3 object (s3-get-object) → Then click on configure

Basic Info

→ function name → Use an existing role → select the role created just now
&
S3 trigger
→ Bucket select bucket created ; keep everything default, tick I acknowledge then create function

→ Now get to configuration and increase timeout to 20 sec

→ Now get to Test
. . add event name
→ Now go to S3 and upload an image

→ Now again go to lambda check monitor → view logs on cloudwatch

→ Now when we click on log we will see what all happened

~~Pass~~ ***

If want customization (upload image through code)

→ Go to the test → Create event → enter event name
→ Now we need to change arn so go to S3.
→ Select S3 bucket and click on copy ARN now paste this ARN in code
→ Now in code also replace the bucket name with yours
→ Now in code → object → key → change to image eg 2.png
→ Go to the bucket where image is uploaded and then go to Settings and enable e-Tag
→ Now copy paste the e-Tag in code
→ save and test
→ Execution result: succeeded (logs) → click on logs
→ U will be redirect to cloudwatch

→ Go to Code in lambda → index.js → In try block add console.log ('An image has been added')

→ Now deploy & test
→ Take ss of image has been added