DevOps Training notes

Day 1: DevOps and GCP Basics:

CRI: Container Runtime Interface.

Basic DevOps terminologies:
Devops: methodology that brings people, process and technology together using CI/CD
CI- continuous integration of MVP with the application code.
CD- Continuous deployment/delivery-
    Non-prod(deployment)
    Prod (delivery) (deployment with a check)
IaC: **Infrastructure as Code:** creating, managing and destroying the infrastructure using code. Like it is done for GCP creating VMs by using scripts; like in terraform.

CloudFoundry is an example of **PaaS**

NIST: National Institute of Standards and Technology:
Lays/establishes the standards for cloud service providers.

Details on Cloud:
https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.500-291r2.pdf
https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf

Cloud video: https://www.youtube.com/watch?v=zDAYZU4A3w0

GCP Introduction:
Project: Billable entity in google cloud.
Resources: The services/tools that are used in a project. Multiple resources can be a part of a project.

Compute engine:
The regions in which the VM is created, will affect the latency while accessing the application. IOWA is the most cost effective region in GCP.

By default the VM created in GCP is not accessible from outside, using SSH or anything.
You need to enable a public-private key pair to access the VM.

Steps to generate Public/private key pair:

Public keys are like locks and private keys are like keys.
On Mac, run **ssh-keygen** in the terminal.
There will be 2 files created and will contain public and private keys.

Login to remote server using ssh as:
ssh -i id_rsa username@externalip
Where username is from the public key generated and external ip is VM's ip address.

Lab Work:

Lab1:

create Free tier GCP account

Lab2:

GCP Console Walkthrough
First GCP instance
Compute Engine
VM Instances
Name of a machine
Select Zone Iowa (this is most cost effective)
Select Machine type (f1 shared one in this case) (CPU series as N1)
Select Centos latest image (centos version 7)
And lets select default options here and create your machine
Once done login into the machine using GCP console (ssh option)

Lab3:

Login into your GCP account https://console.cloud.google.com
Create a project name "testproject", switch to "testproject"
Go to main navigation bar and select compute engine and click on VM Instances -> Generate your public and private key-pair (use puttygen)
Create a machine named "myserver", in Iowa region using N1 machine type and use f1-micro machine type
You have to use boot image of ubuntu:18.04
Go to security tab on the same page and put your "public" key there and click on create
Login into the machine using External IP address
Username will be the key name
Password will be your Private Key

Windows --> open a putty and go to ssh and auth and put your private key location there
MAC --> MAC Terminal --> ssh-keygen (id_rsa) and (id_rsa.pub) --> ssh -i id_rsa username@externalip

NOTE: Once done, make sure you delete the servers!

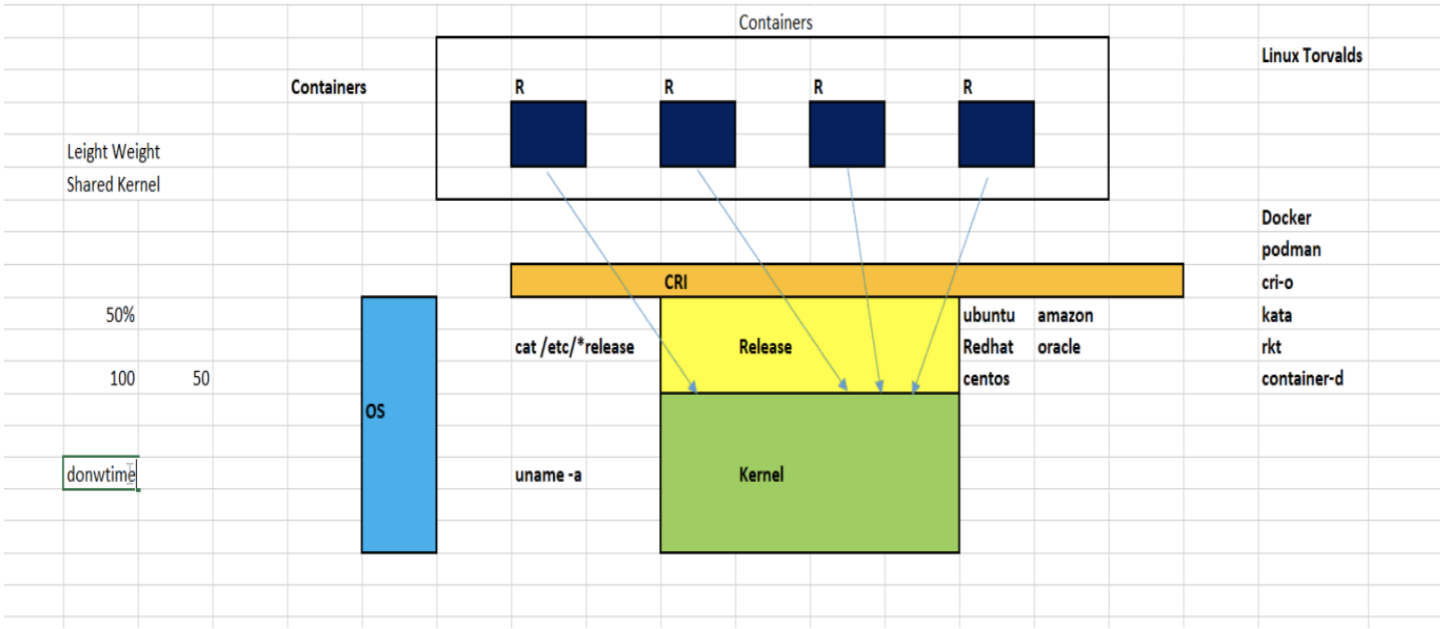
Day 2: Containers and Docker:

Journey from data centers to cloud/ virtual machines.
Initially, one server, one application was the concept.
Then came hypervisor ;

Hypervisor: Virtualises the hardware into multiple independent/ isolated hardware units; each virtual H/W unit can have its own OS
.
Limitation of Hypervisor:
Since each virtual machine on hypervisor has only hardware resources isolated and virtualized, you need to install a full fleshed OS on it that makes it bulky and difficult to manage.

Note: All clouds are meant for Linux and windows; not unix or any other OS.

Then came containers;
Containers are the lightweight OS that share a common kernel.



OS = kernel + Release

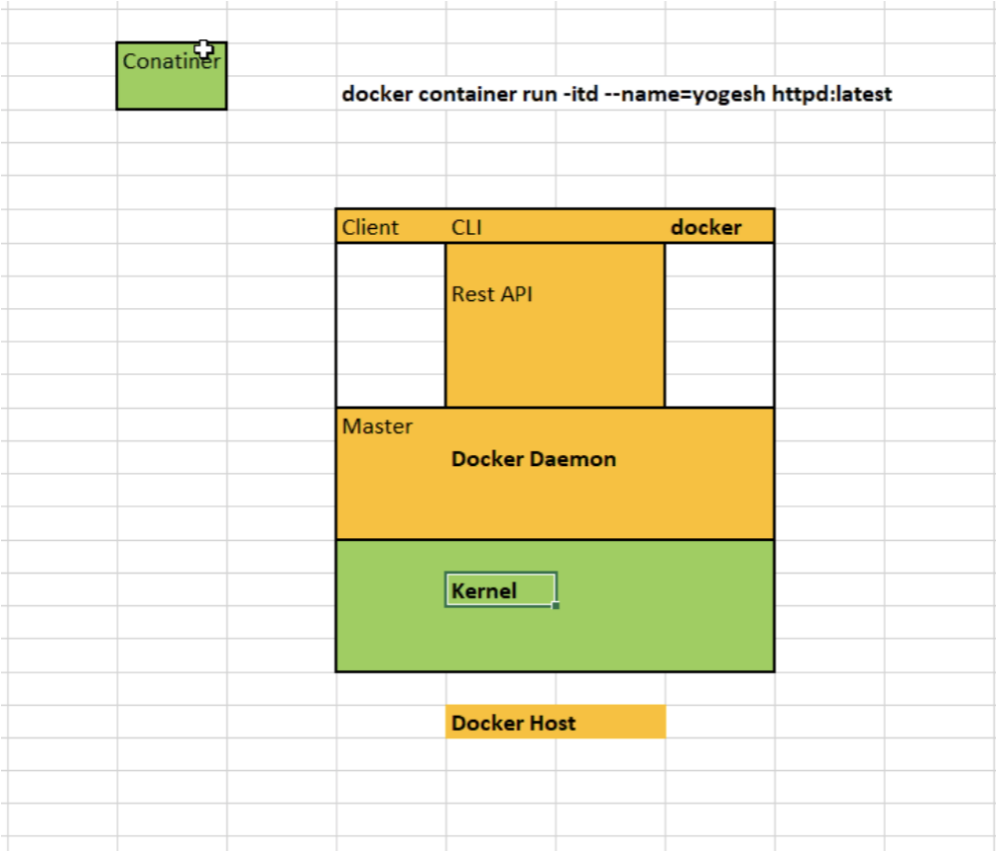
Difference between VM and containers:

VM	Containers
Full blown OS	only contains release
bulky	Lightweight
downtime and recovery time is more	less downtime and recovery time

Limitation of containers:
The container hardware resources can not exceed beyond the underlying hardware; this is where container orchestration or kubernetes comes into picture.

Docker:
Is a platform to build(image creation), ship(image deployment) and run (execute image/ manage containers) the image.
Build images, Ship/share those images to registry, and run those images into containers.

Docker architecture:



Docker and K8S videos:
https://www.youtube.com/watch?v=PMLUiPvPLEU
https://www.youtube.com/watch?v=Eic8WrGLgm4

LAB Work:

LAB 1: Docker setup

Setup docker on a VM on GCP.
Create CentOS VM with g1.small.
Login to vm using SSH.
Switch to root user:
sudo -i
Follow the steps from docs.docker.com.
(https://docs.docker.com/engine/install/centos/#install-using-the-repository)
Use **yum repolist** to see the packages enabled.

Useful commands:

systemctl status docker (to check the status of docker)
systemctl enable docker (to enable docker auto start on restart)
docker --version (to check the version)
docker system df (to check the space occupied by docker images)

Docker commands follow the following general syntax:
docker <object> <sub-commands> [Options] <arguments>
Where object may be one of: container, image, network, volume

Image lifecycle:

Pull an image from remote repository:
docker image pull <image name>
Default pull will be from docker hub (hub.docker.com) and latest image version
For pulling images from a private registry, provider full image path.
For pulling image of specific version; use:
docker image pull <image name>:<version tag>

List the images present on docker host:
docker image ls

Add new/custom tag to existing image:
docker image tag <old name>:<old tag> <new name>:<new tag>

Create image:
Image can be created from a file that holds the instructions/blueprints for the image.
Once the dockerfile is created, it can be converted into an image by executing the command:
docker image build -t <image name>:<image version> -f<docker-file-name> <path where image should be created>

Publish an image to remote registry:
To publish an image to remote registry, you must login to the registry as:
docker login

You will be prompted to enter the user name and password for the registry. Then, execute push command as:

docker push <image-repo-name>:<tag>

Remove a docker image from local:
docker image rm <image-name>:<image tag>

Lab Work:

- Create your Docker Hub account
- Search an image with a name nginx from global repo
- Download the latest nginx image to your docker host
- List the image on your docker host
- Download another available version of nginx image (any other version)
- List the images again

Show me the work -> docker image ls -->paste this out to me on chat

- Tag your latest nginx image with yourname:v1 and list the local images
- Remove yourname:v1 image from docker host
- List the images and verify the output
- At last remove all images from your Docker Host

Day 3: Container and container management:

A container is an isolated, lightweight operating system that shares a common kernel.
A container is created when an image is executed.

docker container run [options] <img name>:<tag>

Options:

- -i (interactive mode)
- -t (terminal)
- -d (detach mode/ demonised mode) (don't go inside the container)
- --name (assign name to a container)
- --mem (provide memory to container)
- --cpu (provide cpu to container)

run command under the hood performs:

- Image check; if not present, it pulls the image
- Creates container metadata
- Starts the container

Container lifecycle and imp commands:

```
docker <object> <sub-commands>    [options]    <args>
                                 -t -i -d --name

-> Create a container: docker container run -itd httpd:latest

long container id : 64 chr
short container id: 12 chr

-> List the containers: docker container ls      (it will show running containers)
                        docker container ls -a   (it will show all (running and stopped) containers)
                        docker container ls -q   (the container id's of running containers)
                        docker container ls -aq  (the container id's of all containers)

-> Stop a container: docker container stop <container-name>
-> Start a container: docker container start <container-name>

-> Login inside a container: docker container exec -it <container-name> /bin/bash
                             exit

-> Name to a container: docker container run -itd --name=yogesh httpd:latest
```

```
-> Name to a container: docker container run -itd --name=yogesh httpd:latest

-> Rename your container: docker container rename <old-container-name>  <new-container-name>

-> Metadata information about a container: docker container inspect <container-name>

-> Logs for containers: docker container logs <container-name>

-> Performance/Stats:  docker container stats

-> Removing a container (stop and remove): docker container rm <container-name>
```

Cheat sheet for docker commands:
https://thinknyx.com/docker-cheat-sheet/

Lab Work:

Lab 1: GCP automation scripts:

Automation script for installing docker:

```
#!/bin/bash
sudo yum remove docker docker-client docker-client-latest docker-common docker-latest
docker-latest-logrotate docker-logrotate docker-engine
sudo yum install -y yum-utils
sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
sudo yum install -y docker-ce docker-ce-cli http://containerd.io
sudo systemctl start docker
sudo systemctl enable docker
```

Lab 2: Container lifecycle.

Execute all the steps shown in container life cycle

Lab 3:

- Create one GCP server of centos:7
  machine type of g1.small
  use the startup script method to install Docker,
  also, don't forget to use public/private key-pair.
- Create a container named "dockercon" using nginx:latest
- Check the container status (docker container ls and docker container ls)

Show me the work done --> Paste docker container ls output to me on chat

- Stop the container
- start the container
- login inside the container
- using "docker container exec -it <container-name> /bin/bash" ---> or you can use /bin/sh if bash wont work (to come out of a container, just type exit and enter)
- Try renaming a container from "dockercon" to "yogeshcon"
- Try removing a running "yogeshcon" container
- Finally stop a container and remove a container
- Remove all other containers also
- Remove all local available images

Useful commands -> docker container run, docker container ps, docker container stop, docker container start, docker container rm

Exposing Port from docker container to docker host:
Use -p flag to provide port mapping.

docker container run -itd --name webcon -p 83:80 nginx:latest

Then on GCP create a firewall rule.

Access using ip and port on a web browser.

Port publish example:

```
Port Publish:

docker container run -itd --name=yogeshagain -p <hostport>:<containerport> httpd:latest

docker container run -itd --name=yogeshagain -p 98:80 httpd:latest

GCP level: I need to open a firewall rule for port number 98
```

Lab work:

On a GCP VM, install docker and create a container using nginx:latest image.
Create the Firewall rule on GCP
While creating firewall rule, you should allow all the ip range as the public IP of your server may keep changing as and when it is restarted/recreated. To do that, ips allowed is specified as: 0.0.0.0/0

Access the VM ip on the port exposed for receiving the traffic. The page should work and return a valid 200 response.

Creating custom images:
New images can be created using Dockerfile. Instructions in dockerfile come from docs.docker.com.

Sample docker file:

```
# This is a comment
FROM <some base image>
RUN yum -y update                # if you need to update that image
RUN yum -y install httpd         # install something atop base image
EXPOSE  80          # expose port to enable image to receive requests
CMD ["httpd", "-D", "FOREGROUND"]
```

Once the docker file is ready, we need to compile it and pack it in a distributable package called image.

Create image:
Image can be created from a file that holds the instructions/blueprints for the image. Once the dockerfile is created, it can be converted into an image by executing the command:
docker image build -t <image name>:<image version> -f<docker-file-name> <path where image should be created>

LAB Work:
Hey Yogesh, can you please create a custom web server image using centos:7 as a base image and it should show "This is my own custom Web image".
Some steps are mentioned below:

```
(After logging into vm and setting up the docker)
mkdir /var/tmp/webapp
cd /var/tmp/webapp
vi Dockerfile
# This is my first webapp Docker file
FROM centos:7
RUN yum -y update
RUN yum -y install httpd
COPY    ./index.html  /var/www/html/index.html
EXPOSE  80
CMD ["httpd", "-D", "FOREGROUND"]
(Save this docker file and exit VI edittor)
vi index.html
<h1>This is my own custom Web image</h1>
docker image build -t mywebimage:v1 .
docker image ls
```

create a container and test my image

Day 4: Container orchestration:

Limitations of containers:

No auto healing capability,
Do not ensure high availability
No auto scaling
No Load distribution

Container orchestration: It is a tool that lets us manage containerised applications effectively by addressing all the concerns of containers.

The softwares used for container orchestration are:

1. Docker swarm
2. Kubernetes

Setting up K8S:

```
Setup K8S:
==========

1.) Minikube: Single Node Cluster, ISO Image (minikube), Laptop, Testing/Self-Learning/POC's. not meant for Production

-> Virtulised software (Virtual Box, VMware workstation, Hype-V, Hyper-kit, docker/podman)
-> Minikube Software
-> Kubectl Software

2.) OpenSource K8S: Min. 2 Servers (1 Master, 1 Worker), OpenSource/free, Community Support, Solid k8s, dedicated support,
learning curver is high, no inbuilt GUI, Backup and Restores. Upgrade (9 months) -> painful tasks

3.) Enterprise K8S: K8S as a software/products, Official Support, Inbuilt-GUI's, backup and restore options also, upgrade is
no more pain, Setup is easy, cost (too much expensive), learning is even higher (CRI, CO, Enterprise product as well), Huge
infra cost

OpenShift (Redhat), Rancher (SuSe), VMware Tanzu, MKE etc

4.) Managed K8S: (CaaS): I will manage k8s for you, you just come and deploy you applications, all majot flexibility, no
upfront cost (pay as you go), no dedciated k8s team is required, GUI, no be having access to the k8s master server, less
scope for cutomization

EKS (Amazon), AKS (Microsoft), GKE (Google)
```

For setting up open source K8S, refer
https://www.youtube.com/watch?v=PMLUiPvPLEU

Lab 1:

Create a test GKE cluster with 2 nodes for "Standard GKE cluster"
-> use Zonal cluster
-> use Ubuntu with Docker
-> Make sure to change the harddrive to 20GB (don't use 100 GB) under node pool section
-> You need to create 1.20.10 from static version

Once done, connect to your GKE using the CLI and execute below command:

-> kubectl get node --> this will tell you that you have 2 worker nodes in your managed k8s

Note: Paste this output on chat

Once done, select your cluster and delete it (make sure to delete it)

Lab 2: Minikube Details: Homework for your personal laptops

Windows:
========

Virtual Box: https://www.virtualbox.org/wiki/Downloads
Kubectl: https://kubernetes.io/docs/tasks/tools/install-kubectl-windows/
Minikube: https://minikube.sigs.k8s.io/docs/start/

minikube start --driver=virtualbox

MAC:
====
Virtual Box: https://www.virtualbox.org/wiki/Downloads
Kubectl: https://kubernetes.io/docs/tasks/tools/install-kubectl-macos/
Minikube: https://minikube.sigs.k8s.io/docs/start/

minikube start --driver=virtualbox

Lab 3:

Create a test GKE cluster with 3 nodes for "Standard GKE cluster"
-> use Zonal cluster
-> use Ubuntu with Docker
-> Make sure to change the harddrive to 20GB (don't use 100 GB) under node pool section
-> You need to create 1.20.10-gke-1600 from static version

Once done, connect to your GKE using the CLI and execute below command:

-> kubectl get node --> this will tell you that you have 3 worker nodes in your managed k8s
Scale out your cluster by adding another node
Scale in your cluster by removing 2 nodes

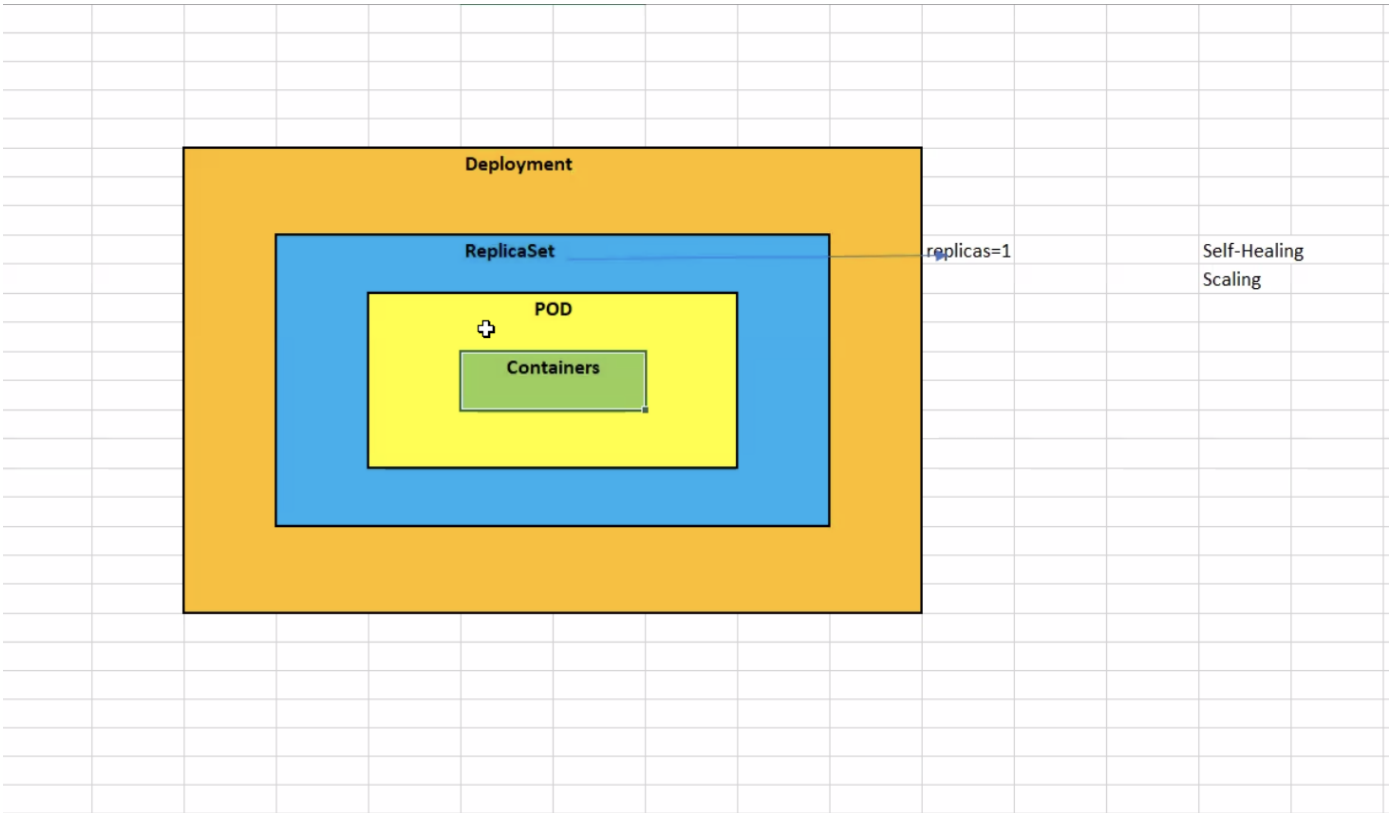
Once done, select your cluster and delete it (make sure to delete it)

Day 5:

Two approaches are there to use K8S:

Imperative: using cli, kubectl utility, fast.
Declarative: automotive using CI/CD; using yml.

Architecture:



Objects in K8S:

Container: runs the application; do not have a network inbuilt.

POD: encapsulates the container; have a pod network inbuilt

Replica set: replica set is responsible for creating replicas of pods. self healing and scalability capabilities are added in the replica set. If the pod is deleted, rs will create new pods to ensure availability.

Deployment: creates replica set and manages the rolling of application releases automatically; without having downtime.

What happens when replica-set is deleted:

When rs is attempted to be deleted; it begins with deleting the containers it has. Once the containers are deleted, rs tends to delete itself; but it is when deployment comes into picture. Deployment instructs the replica-set to recover itself; and thereby new pods are created.

Imperative style:

Uses cli tool- kubectl.

Command structure:

kubectl <subcommand> <object> [option]

Subcommands available:

- create
- get
- describe
- delete
- scale

Object:

- Deployment, rs, pod

Life cycle:

```
-> Create a deployment:  kubectl create deployment yogeshdep --image=httpd:latest

-> List the objects: kubectl get deployment
                     kubectl get rs
                     kubectl get pods


-> Describe your objects: kubectl describe pod <pod-name>      ---> troubleshooting    (Events)
                          kubectl describe rs <rs-name>
                          kubectl describe deploy <deploy-name>

-> Test: kubectl delete pod <pod-name>    (self-healing test)

-> Decom: kubectl delete deploy <deploy-name>
```

Lab work:
GKE cluster setup.
Perform all steps in the gke lifecycle.

Scaling a K8S cluster:
There can be two kinds of scaling:

1. Manual scaling
2. Auto scaling

1. **Manual scaling:**
   Manual scaling requires human intervention. It can either be done by using Google console UI (actions-> scale) or using command line (kubectl scale).

2. **Auto scaling:**
   Auto scaling has a dependency on metric server as this kind of scaling comes to play automatically once the trigger point is reached. Whether or not the trigger point is reached is continuously evaluated from the metric server. The trigger may be xx% of CPU usage, XX% of RAM usage etc.

```
Manual Scaling
HPA (AutoScaling) --> Metric server


Services (Networking/Switch/Router/LB)
========

ClusterIP: Private Subnet , Internal Communication
NodePort:  Public Subnet , External Communication

30000-32767 (Reserved NodePort for K8S Services)
```

Services in kubernetes:
Services are required for enabling networking and load balancing in kubernetes.

Setting up network in k8s:
Containers do not have an inbuilt network. In a kubernetes ecosystem, the pod encapsulates the container. POD is the unit in k8s that has the network called pod network. Using this network, communication with apps can be established.

The communication to pods can not happen directly. It is carried out via Services.
The incoming request is received by a service, the service then communicates with the pod to resolve the request received. Also, different pods can not communicate with each other directly. The services for each pod act as an intermediate that allows the communication between pods.

Setting up communication in GKE via service:
Step 1: Create a k8s cluster with the required number of nodes.
Step 2: Create/ deploy the workload on this k8s cluster- this can either be done from UI directly or by creating a deployment in the cluster created above.
Step 3: For the created workload, you now need to expose the ports. You can expose the ports as follows:

1. From UI: in workload -> actions -> expose -> set ports -> click expose
2. From Kubectl: run command :
   kubectl expose deployment <deployment-name> --port=<service-port> --target-port=<container-port> --name=<service-name> --type=<service-type>

Step 4: the above step will map the container port with the service. This will generate one port over which the communication can be made.
Step 5: For this port number, you need to create a firewall rule.

Kubernetes cheat sheet: https://thinknyx.com/k8s-cheatsheet

Devops and K8S both depend on automation.

```
Automation:
===========


Past:
=====

Scripts: Shell/Powershell/python/perl etc     ===> Imperative in Nature

Himanshu wrote perl --> ?

2000 --> Pune --> Car --> Shimla --> How you would have been travelled

What to do and how to do --> users

2011-2012 ->


Configuration Management Tools -> CM  (Ansible/Puppet/Chef/SaltStack/Cf Engine)  ==> Declarative in nature

2021 -->Pune --> car -> Shimla --> Google Maps

what you want to do, how part will be taken the google maps
```

For example, creating a user in system:
In imperative approach:

```
#!/bin/sh
USER=$1 ; GROUP=$2 ; HOME=$3
if [ 0 -ne $(getent passwd $USER > /dev/null)$? ]
then useradd $USER -home $HOME -gid $GROUP -n ; fi
OLDGID=`getent passwd $USER | awk -F: '{print $4}'`
OLDGROUP=`getent group $OLDGID | awk -F: '{print $1}'`
OLDHOME=`getent passwd $USER | awk -F: '{print $6}'`
if [ "$GROUP" != "$OLDGID" ] && ["$GROUP" != "$OLDGROUP" ]
then usermod -gid $GROUP $USER; fi
if [ "$HOME" != "$OLDHOME" ]
then usermod -home $HOME $USER; fi
```

```
Configuration Management:

tasks:
  - name: Creating Yogesh Raheja User
    user: name=yogeshraheja state=present
```

Advantages:

Time saving
Complexity reduced

```
2007: Puppet/Chef --> Ruby --> complex --> performance (agents) --> Pull based tools

2012: Ansible/SaltStack --> Python --> Simple --> Agentless -> Push based tool --> 2015 Oct (RedHat) --> 2017 --> Number 1
```

```
Server Automation
Network Automation
```

Architecture of Ansible:

Master - Worker topology.
Master ----------> Worker
Master is where Ansible is installed and is also called controller
Master has an inventory file that manages the worker list.
Worker is the one that is managed by the master. All the operations are executed on the worker through ansible installed on master.

Terminology:

```
Terminology:
============

Ansible Core/Ansible Engine  ==> Open Source Ansible


Controller Node: Automation Master server

Managed Node: Server managed by Ansible controller node

Inventory: is a list of Ansible Managed Servers

Modules: Pre-built python scripts by Ansible

Adhoc-requests: One time execution to perform automation (temporary automation)

YAML: Yet another markup language  (Homework)

Playbooks: Automation files in Ansible written YAML language

Roles: Directory structure for writting playbooks

Collections: Combination of Roles

Ansible Galaxy: A supermarket for roles and collections

Ansible Tower: GUI for Ansible Core/Ansible Engine ==> Enterprise offering by RedHat

SSH/WINRM
```

Ansible documentation: docs.asible.com

Lab 1: Ansible setup:

```
Ansible Setup:
==============
Lab1: Pre-req:
==============

-> Create two servers using GCP
   -> centos7, g1-small

-> Networking (servers should talk to each other -> ping test)
   -> ip addr
   -> vi /etc/hosts and put the entries of master in worker and worker in master

exmaple: like this    10.128.0.30     ansibleworker

-> Password less authentication  (keys)
   -> create a key pair on ansiblemaster ==> ssh-keygen ( Private key = /root/.ssh/id_rsa and pub key =
/root/.ssh/id_rsa.pub)
   -> cat /root/.ssh/id_rsa.pub
   -> Go to your ansibleworker and create one file "mkdir /root/.ssh" and then "touch /root/.ssh/authorized_keys"
   -> copy your public key in this file /root/.ssh/authorized_keys
   -> Change the permission of your authorized file "chmod 600 /root/.ssh/authorized_keys"
   -> On your ansibleworker open a file "/etc/ssh/sshd_config" and check a parameter named "PermitRootLogin" if this is no
then chnage/set it to yes
   -> then restart sshd service on your ansibleworker
       -> systemctl restart sshd
       -> systemctl status sshd
   -> Once done go back to your master and do ssh "ansibleworker" and see if you are able to login into the worker without
pasword, if not troubleshoot it!
```

Lab 2: installing ansible:

```
Lab2: Ansible Installation

Installation:
==============

-> docs.ansible.com
   -> community documentation --> installation -> installing ansible for specific os

-> install ansible
   -> Only on your ansiblemaster run below steps:
   -> yum install epel-release
   -> yum install ansible
   -> Check the installed ansible version by running "ansible --version"

-> Inventory Setup
   -> Default inventory path "/etc/ansible/hosts" for ansible
   -> vi /etc/ansible/hosts
      just add your managed server name here


-> Ansible test (ansible ping test)
   -> ansible "all" -m ping    -------> Adhoc request

Paste me the output of above command.
```

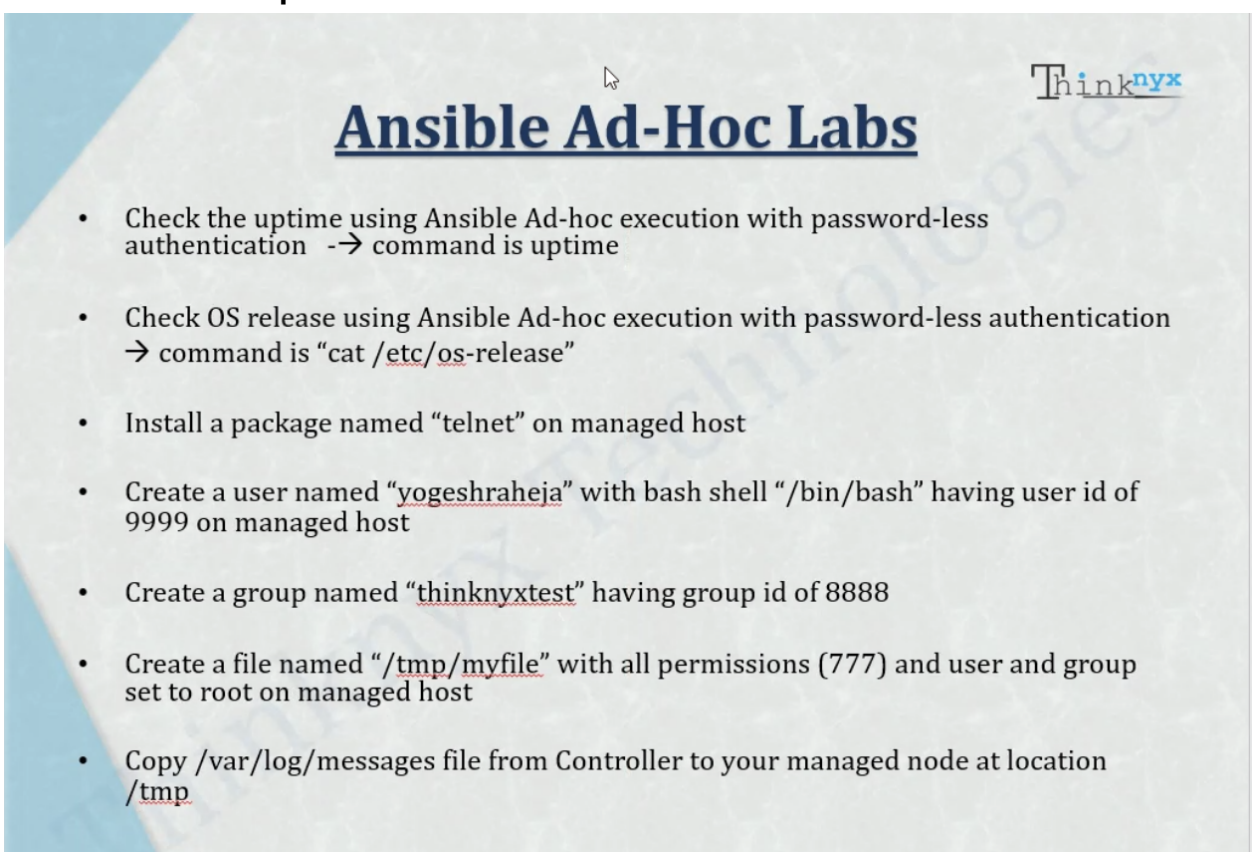
Kinds of request in Ansible:

There can be two kinds of requests that are executed in ansible master:

1. Ad-hoc requests:
   These are one-time requests and are not needed to be executed repeatedly. These can be executed directly from ansible cli available in ansible master. Through these requests, you can invoke the predefined modules that does the job for you.

```
Adhoc Requests:
===============

ansible "list of servers/inventory" -m <Module> -a "Attributes/parameters"

ansible "all" -m ping
```

Ansible ad-hoc requests lab:



Answers:

```
--> ansible "all" -m command -a "cat /etc/os-release"

Install a package named “telnet” on managed host

--. ansible "all" -m yum -a "name=telnet state=present"

-> rpm -qa| grep -i telnet   on worker run this to ensure the pkg is installed

Create a user named “yogeshraheja” with bash shell “/bin/bash” having user id of 9999 on managed host

--> ansible "all" -m user -a "name=yogeshraheja uid=9999 shell=/bin/bash state=present"

Create a group named “thinknyxtest” having group id of 8888

--> ansible "all" -m group -a "name=thinknyxtest gid=8888 state=present"

Create a file named “/tmp/myfile” with all permissions (777) and user and group set to root on managed host

--> ansible "all" -m file -a "path=/tmp/myfile mode=777 owner=root group=root state=touch"

Copy /var/log/messages file from Controller to your managed node at location /tmp

--> ansible "all" -m copy -a "src=/var/log/messages dest=/tmp/messages"
```

2. Repeated requests:
   These requests are not limited to cli. These are created and managed in the form of yaml. These files are called playbooks.

   YAML: yet another markup language:
   https://www.redhat.com/en/topics/automation/what-is-yaml#:~:text=YAML%20is%20a%20data%20serialization,is%20for%20data%2C%20not%20documents

   Steps of playbook creation:
   1. Create
   2. Syntax check
   3. Dry run
   4. Execute (Run)

   Commands:

```
-> Heu yogesh, can you develope an ansible playbook for webserver setup (application installation, customization - custom web
page, start the application and want us to ensure that the application should automatically come up after server reboot also
```

```
-> Create a playbook
-> Syntax-checking:          ansible-playbook websetup.yml  --syntax-check
-> Dry-run (Check mode):     ansible-playbook websetup.yml  --check
-> Real-run (Run mode):      ansible-playbook websetup.yml
```

Ansible cheat sheet: https://thinknyx.com/ansible-cheatsheet/

Terraform cheatsheet: https://thinknyx.com/terraform-cheat-sheet/

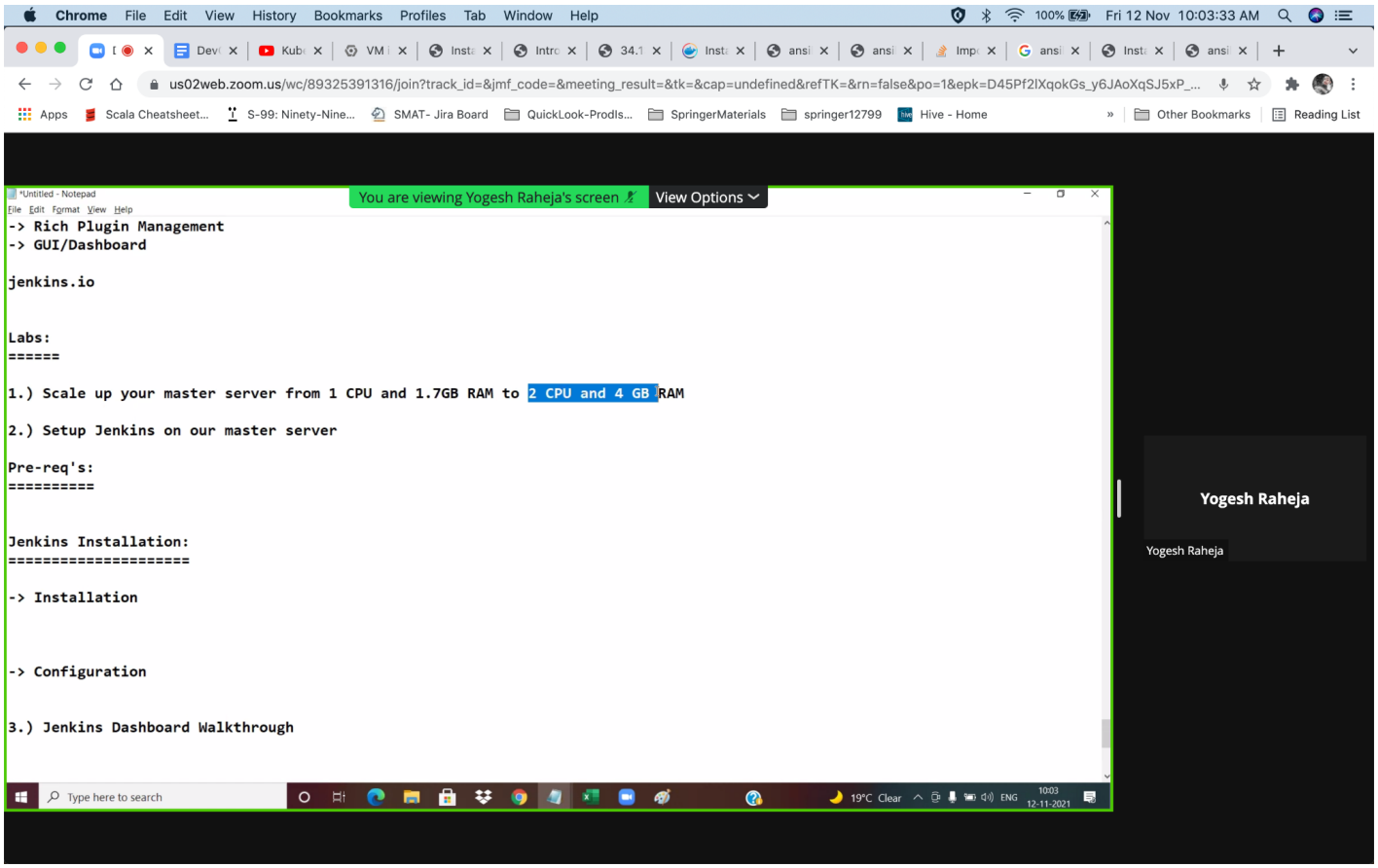
Lab Work:
Create an Ansible playbook for installing the docker in a linux machine.

Answer: Playbook like this:

```
---
-
  name: Docker Setup Playbook
  hosts: all

  tasks:
    -
      name: Removing old docker pkgs
      yum:
        name:
          - docker
          - docker-client
          - docker-client-latest
          - docker-common
          - docker-latest
          - docker-latest-logrotate
          - docker-logrotate
          - docker-engine
        state: absent

    -
      name: Installing Pre-reqs
      yum:
        name: yum-utils
        state: present

    -
      name: Seeting up the repo's for Docker
      get_url:
        url: https://download.docker.com/linux/centos/docker-ce.repo
        dest: /etc/yum.repos.d/docker-ce.repo
-- INSERT --
```

```
      - docker-latest-logrotate
      - docker-logrotate
      - docker-engine
    state: absent

-
  name: Installing Pre-reqs
  yum:
    name: yum-utils
    state: present

-
  name: Seeting up the repo's for Docker
  get_url:
    url: https://download.docker.com/linux/centos/docker-ce.repo
    dest: /etc/yum.repos.d/docker-ce.repo

-
  name: Installing Docker Software
  yum:
    name:
      - docker-ce
      - docker-ce-cli
      - containerd.io

-
  name: Starting and enabling Docker Services
  service:
    name: docker
    state: started
    enabled: yes
```

Day 8: CI/CD using Jenkins:



Installing Java:

```
Labs:
=====

1.) Scale up lab for upgrading your master server from (1 CPU and 1.7 GB RAM ---> 2 CPU and 4 GB RAM)

2.) Setup the Jenkins (jenkins.io --> Click on download)

Pre-req:
========
-> Java is a pre-req for Jenkins, install java on your master server

yum list | grep -i openjdk
yum install java-1.8.0-openjdk.x86_64
java -version
```

Installing Jenkins:

```
Jenkins Installation Steps:
===========================

Repo setup:
===========
yum repolist
yum install wget
wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo
yum repolist

Key Import:
============
rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io.key

Install Jenkins:
================
yum install jenkins
rpm -qa | grep -i jenkins

edit /etc/sysconfig/jenkins file and change JENKINS_USER to root  (it will show user as Jenkins, change that to root)

JENKINS_USER = "root"

systemctl start jenkins
systemctl status jenkins
```

Configure Jenkins GUI:

```
Jenkins Configuration:
======================

-> We need create firewall rule to open port 8080 in GCP
-> Go to browser put your masterip:8080
-> On your master "cat /var/lib/jenkins/secrets/initialAdminPassword" and paste the returned string in the UI
-> Click on select plugins to install
-> Select NONE  (at the top of your screen) so that no plugin should get installed
-> Put the requested details
-> Press start using jenkins
-> Logout and Login back with your provided credentails
```

Plugin Management:

```
Lab1.6: Plugin Management
=======

-> Go to your Jenkins console, dashboard, manage jenkins and then manage plugins
-> Search and install GIT plugin

Note: When you select Git, select install without restart
      Once plugins gets install, go to the bottom of the same screen and tick restart jenkins
```

NOTE:

var/lib/jenkins ----> backup and restore this directory and all the data will be back.

Lab work:
Mini CI/CD project --> GCP, Docker, Ansible, Github, Jenkins

Github and create one repo --> sndemo (Remote repo)
Install Git on my master
Clone the remote repo
-- Pre-ansible setup (Inventory)
-- Install Docker on my master (ansible playbook) ---> ansible to use locally
-- Dockerfile (with relevant code like html/python/java etc etc)
-- Build the image and push the image to the registry --->(sndemo:v1) --> image name
-- Ansible playbook to deploy containers for us on managed node
-- Check our application