Challenges: → Env
→ shipping of code
→ Standard PKging (jar, war, dll → .net)

Docker

Java: write once run Anywhere.

PODA → Package Once, Deploy AnyWhere.

Modern App Demands:

→ ~~Zero downtime (High AVL)~~
→ Portability.
→ Scalability / elasticity
Prop. of Sy. to handle growing amount of work ~~by adding resources to the sy.~~
- scalability: Horizontal (Increasing RAM, HardDisk, processes cap.)
- Vertical S - Increasing No. of m/c (Servers)

Cloud → provides scalability easily.

O ←http→ O ←TCP→ O

→ Use right tool for right job
~~functionally~~ a tool might be able to do a job but not suitable.

TAP (Technology Agnostic Protocols)
e.g. Http, Https, TCP etc
↓
(REST)

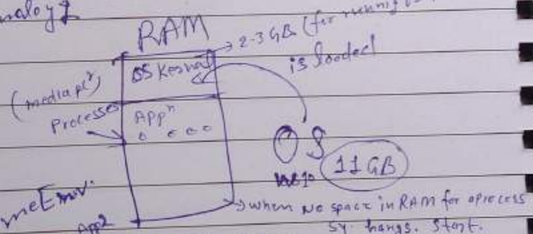
VM → H/W virtualiz^n

we enjoyed around 15 yrs.

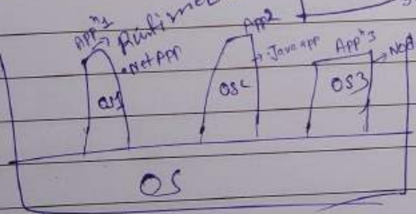
(H/W: Int^l. RAM, HardDisk, motherboard, CPU, sec storage CD/DVD; Ext^l: i/p: K.B., mouse; o/p: printer, monitor)

(H/W sharing) (companies working on this)
⇓
Hypervisors Technology → VMware, oracle VBox.

VM Backup: .vmdk file size 20GB ⇓ contain OS → Not portable easily.



→ Low spin up time.



VM → container (runtime env for app^n)

OS Sharing / OS Virtualiz^n → Containerization < Docker company / Docket company
→ lighter
→ portability.

O.S. Virt^n is supported by → Linux or OS having flavors of Linux { ubuntu, Centos, Fedora, Redhat }
not by windows

H/W virtualiz^n
OS/ V^n
AWS

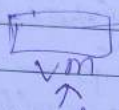
Linux OS = Kernal + supporting files (drivers for K.B Ifo etc.)

Date:

MON TUE WED THU FRI SAT SUN

Container → backed by an Image
runs as process inside ↓ set of files (app^n + dept OS)
RAM of host m/c.

app dep.



Docker file → Recipie (Ingredients)
↓ I/P
Instruction on how to build an image.
SQL S^w dep → will be downloaded & made up running.
etc.

Docker Engine

↓ o/p

Image → can be uploaded to (docker hub.com) → SAAS
hub.docker.com

docker pull <img name>

Cloud
- IAAS (Infrastructure)
- SAAS (Software As A Service) e.g. hub.docker.com (images)
- PAAS (Plateform)

We just use & don't own (gets on rent basis)

Date : ______

MON TUE WED THU FRI SAT SUN

Compute

EC2 (VM) Elastic ~~Cloud~~ Compute Cloud.

container.

→ provides secure, resizable compute capacity in the cloud.

→ we can develop & deploy app^n faster.

→ we can scale up/down any many No. of virtual servers as per need.

Storage: S3 (Simple Storage Service)

Bucket.

region: N. Virginia.

EC2 → Instances 1 → Instances → Launch Instance, VM/cont^r

- choose AMI (Amazon M/C Image) → Template (OS + app's + app)
- ubuntu [Select]
- → Add Tags — Name: mydo^r M/C
- → mydocker secgp (firewall)
  - SSH — 22 — custom
  - Http — 80 — Anywhere
- → Create Key pair.
  - Name [docker private KeyPair]
  - Download Key pair
- Launch. →

Secgroup → firewall

SSH — port 22

Cryptography

- pub key → for encryp^n
- pvt key → for decryp^n

- Sym → same key.
- ASY: Pub & Pri Key

KAAS → Kubernetis

container is a process but complete runtime env of app^n

// files^r tx — f^n → f^n → Training Content → Chand. A → MobaXTerm

hubdocker → SAAS

Session → SSH

RemoteHost [ ] PublicIP   UserName [ubuntu]

Advance SSH

Use Keypair [ ] browse key^r file

[OK]

① sudo su — ⇒ go to root

② root > apt update ⇒

③ > apt install docker.io

docker client CLI ↕ Rest ↕ Daemon S/W (Docker Engine)

select os → Actions → Instance State → stop

→ docker images

→ docker ps → To check running containers (every c^r has an ID)

→ docker ps -a → all (even exited/stopped one)

1) pull ready made Img → spawn cont^r → download the img as well if not AVL locally

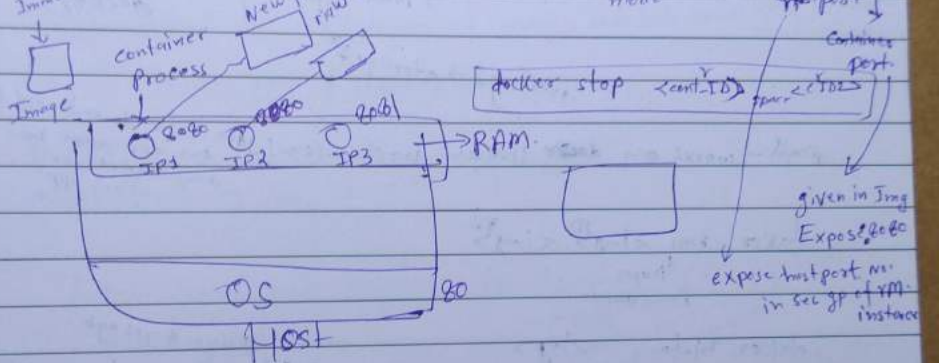
docker pull <img> → docker run <Img-Name>

docker run hello-world

- docker → client
- starts a cont^r from an img.
- name of docker img.

docker run -d --name web -p 80:8080 nginx

- -d → detached mode
- -it → interactive mode
- -p → port mapping
- nginx → Img



{ host port cannot be same for two processes

Name → maps ~~assigns~~ → Automatically to an AVL port

→ docker run -d --name -P <Img>

→ docker stop $(docker ps -aq) → only displays numerical IDs.

→ docker ps --help

{ After starting c^r, ~~image cannot be~~ deleted until cont^r is deleted.

- Knew Basic Linux commands.

To login into container (-it mode)

docker run -it --name temp ubuntu:16.04 /bin/bash
(pull img, run it & login into it.)
latest

root@hostip → -it

root@ContID pwd

ls -ltr

cat → To show content of a file
/etc/hosts → host ip details.

apt update

-ltr → show o/p as long list with various Attributes
sort by modified date
reverse sort order

@host → ls /var/lib/docker → All file sy created by docker

overlay2 → storage DR

pull → works on Union mechanism (Java file downloaded for one app won't be downloaded for other app.

docker rmi <img ID> <img ID>
↑ images
rm → docker

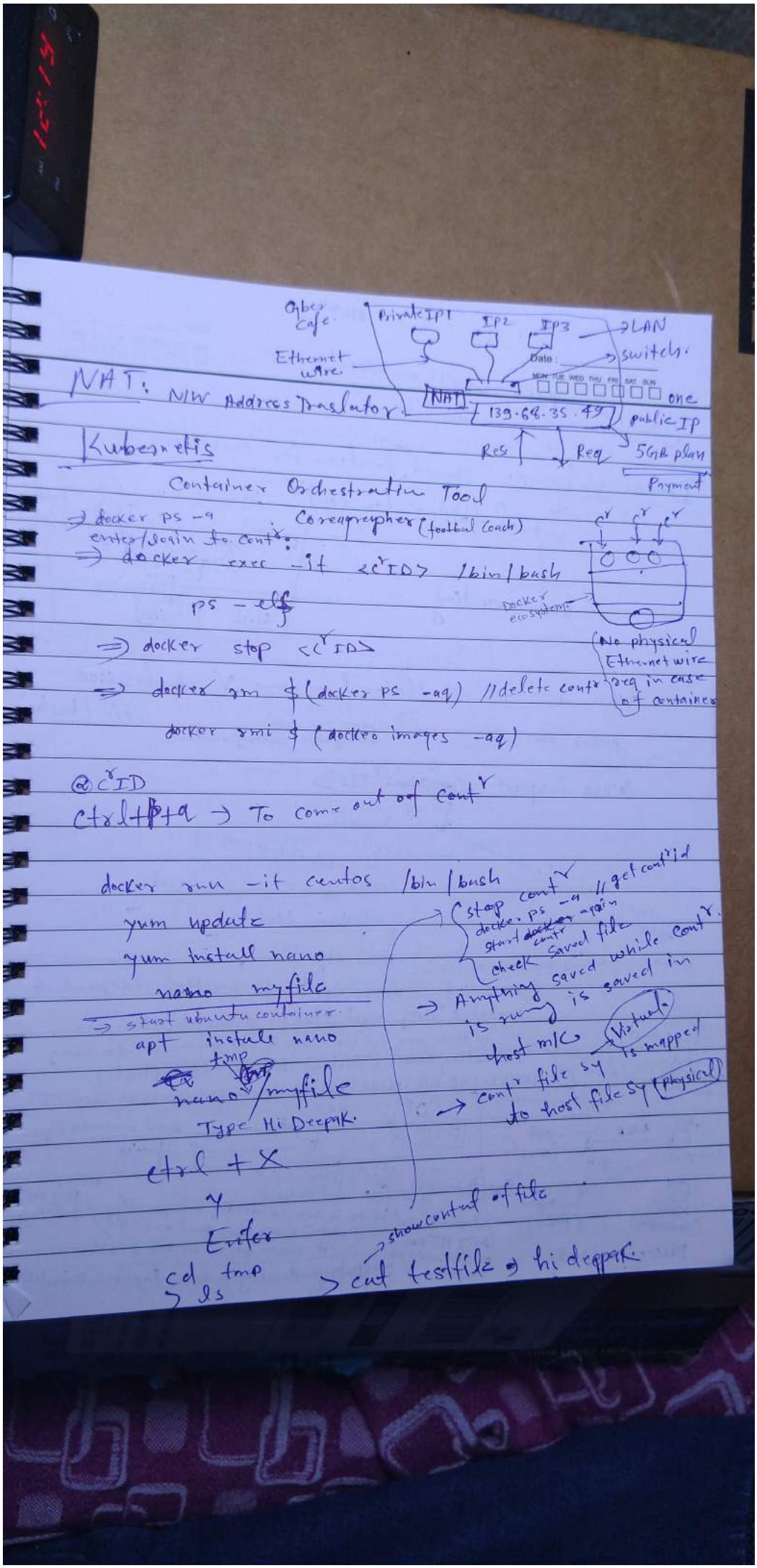
docker history <img>
Time 4 yrs ago
shows what all files & size image have.

@CId → ubuntu specific
apt update
apt install docker.io
Need to start daemons inside cont^r
ping 8.8.8.8 ⇒ cont^r is lightest form of OS
apt install iputils-ping (ping utility needs to installed in cont^r)

→ docker exec -it <c^r Id> /bin/bash ⇒ run a new command in a running container

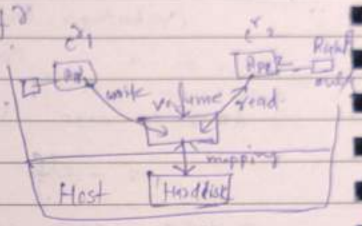
→ docker start -a -i <c^r Id> : To start a stopped container.

↪ docker restart -t 10 <c^r Id> To restart a running c^r. -t: time to wait for stop (sec) before killing c^r


Cyber Cafe
Private IP1
IP2
IP3
LAN
Ethernet wire
Date :
switch
MON TUE WED THU FRI SAT SUN
one
NAT
139.68.35.47
public IP
Req
Res
5GB plan
Payment
Container Orchestration Tool
Coreographer (football coach)
Docker ecosystem
No physical Ethernet wire req in case of containers
NAT: NW Address Translator
Kubernetis
docker ps -a
enter/login to cont^r:
docker exec -it <c^rID> /bin/bash
ps -elf
docker stop <c^rID>
docker rm $(docker ps -aq) //delete cont^r
docker rmi $(docker images -aq)
@c^rID
ctrl+p+q → To come out of cont^r
docker run -it centos /bin/bash
yum update
yum install nano
nano myfile
→ start ubuntu container
apt install nano
tmp
nano myfile
Type Hi Deepak.
ctrl + X
Y
Enter
cd tmp
ls
cat testfile → hi deepak
show content of file
stop cont^r
docker ps -a // get cont^r id
start docker cont^r -ai
check saved file
→ Anything saved while cont^r is running is saved in host m/c (Virtual)
→ cont^r file sy is mapped to host file sy (physical)

Cloud Native App → works on cloud:
Kubernetis

Docker volume: shared storage location for containers.
→ virtual folder for cont^r
↕
host mounting




```
docker run -it -v /test-val --name V ubuntu:16:00 /bin/bash
```
→ spawn the cont^r along with creating a volume.

```
docker ps
```

```
docker inspect <~~docker~~ c-ID>
```
→ c^r Kundali

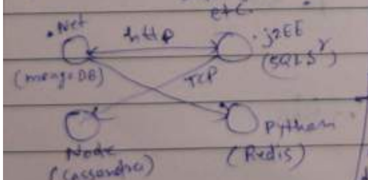
mounts{
source — – – – /test-val → Actual host mapping
volume — /test-val
}

more Downtime (whole s^r needs to be stopped)
Not scalable

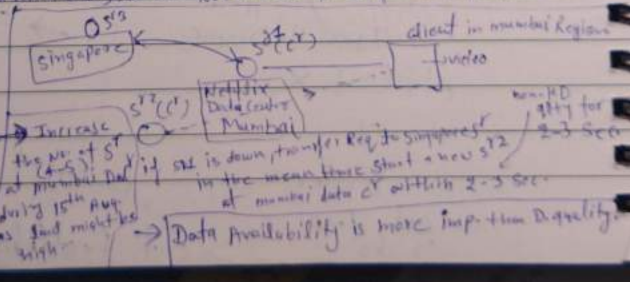
monolithic → microservices (→ ~~Kubernetes~~)
→ Break the large App^n into smaller independent modules "using right tool for right job". They can run in their own run time env & can communicate using Technology Agnostic Protocol (HTTP, TCP, rest etc)

- .Net Tech stack
- J2EE stack
- Node Tech stack
- Python

Similarly one D.B cannot be suitable for all types job.
cart → Redis (In RAM D.B.) → RDBMS not suitable from performance pt of view
shipping → mongo D.B.
etc.



Microservices →



Creating a Docker Image:

Date: MON TUE WED THU FRI SAT SUN

Dockerfile: → A text document that contains all the commands a user could call on the command line to assemble an image.

① cd

mkdir /test

cd /test

Create file → touch /test/Dockerfile (uppercase, lowercase)

→ nano /test/index.html → paste some html content; ctrl+X → Y → Enter

<h1> Hi Deepak </h1>

nano Dockerfile — ctrl+X → Y → ENTER

FROM centos:latest → Take centos OS

MAINTAINER CITUSTEC

RUN yum -y install httpd (server)

COPY index.html /var/www/html/ (copy the index.html file on server given path)

To run the app^n packaged in Img → CMD ["/usr/sbin/httpd", "-D", "FOREGROUND"] → start the server

EXPOSE 80

Build Img:

To give a tag name to image.

```
docker build /test/ -t webserver:v1
                       tagName
```

↓ Dockerfile & indexfile both are at same location

↓ spawn cont^r

```
docker run -d -p 1234:80 webserver:v1
```

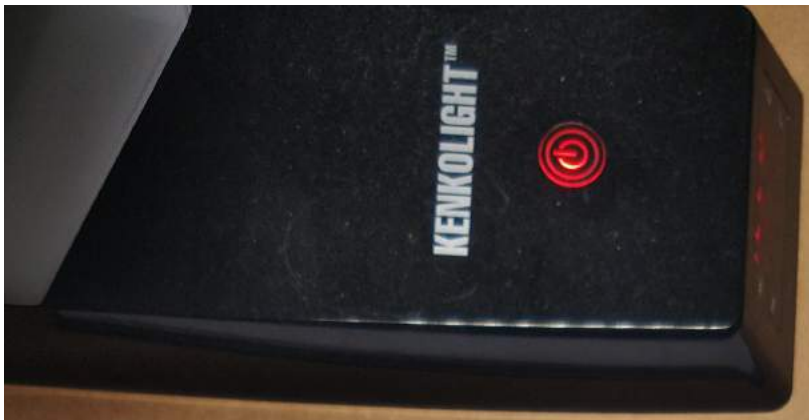
To see result on web browser: <I.P. Address>:1234

→ specific module S^r can be set up at loc^n where load is more monolithic cannot do it

→ As per load on a part^r modules, we can increase the No of S^r for that specific module

→ but in case of monolithic we have to replicate S^r for all modules.

→ All these headache of creating, running, monitoring of S^r (containers) can be taken care of by Kubernetes.



Date :

MON TUE WED THU FRI SAT SUN

Container-linking : So that Two apps running in two containers can communicate each other
→ It is better option than exposing ports.

docker run -idt --name vote --link redis:alias-src -P schoolof devops/vote

↓ already running container Name

↓ Alias for running container Name.

while spawning a new container just link the already running container.

> env ⇒ will show the new env variables for linking with the already container.

→ We can create a docker image from a running container
We spawn the container
We installed some new S/W
& now we would all the changes done in new img

Docker commit : To create new img from container's changes

docker commit <C. Name> <New img-Name>

Dockerfile is input to docker Engine
docker-compose.yml is i/p to { docker-compose, docker swarm, or Kubernetis

Docker-compose: ⇒ A tool for defining & running multi-container docker apps
→ With compose, we use a YAML file to configure your app's services
→ Then with a single command, we create & start all the services from our config

docker-compose.yml

↓

utility to run yml file.

paste content
ctr+X, Y → Enter

[root]/home/ubuntu#> nano docker-compose.yml

→ apt install docker-compose

docker-compose up -d

Using Compose is a 3 step process:

1. Define your app's env with a Dockerfile so it can be reproduced anywhere.
2. Define the services (VI, Services, DB etc) that make up your app in docker-compose.yml so they can be run together in an isolated env.
3. Run 'docker-compose up' & compose starts & runs your entire app

Sample

```
1. version: '3'
2. services:
     web:
       build: .          → path where Dockerfile is kept
                         → root of proj
       ports:
         - "5000:80"     → C port (app running inside the C is exposed at port 80)
           ↓
         host port

       volumes:
         - .:/code
         - logvolume01:/var/log

       links:
         - redis

     redis:
       image: redis:alpine
3. volumes:
     logvolume01: {}
```

To access it http://{HostIP}:5000

Layers :→ UI → controller → Service → DAO → D.B.
(DATA Access)
object

In CI/CD → an isolated test env is created & e2e tests are run & env is destroyed.

$ docker-compose up -d
$ ./run-tests
$ docker-compose down.

⇒ Create 3 VMs (EC2) on AWS :

VM1 — Mr — 30GB RAM
VM1 (ubuntu 16.04)
↳ Linux m/c

VM2 — 30GB — Node1

VM3 — 30GB — Node2

⇒ To use Kubernetes/docker we need VMs
⇓
Linux based
⇓
As it support OS virtualiz'n

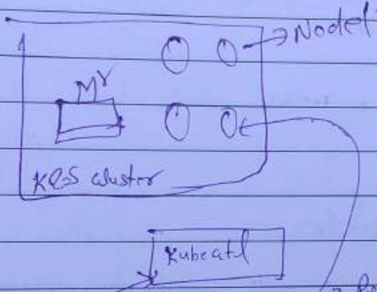
Kubernetes → for Automated app^n deployment, scaling & mgmt
→ open source
contr^r orchestr^n sy.

Date: ______
MON TUE WED THU FRI SAT SUN

docker swarm → alternative of Kubernetes but less powerful than K

→ orchestrator for microservice apps.

K8S cluster → Kubernetes cl.



Master Compt:

1 kube - api server. gives instr^n to below 3, Restful

kubectl → client tool to comm with master

2. Kube - Controller-Manager (Node Control^r) → watchman (Desired state, Actual state) the ReplicaSet → To scale out the pod if some node is experiencing heavy traffic.

3. Kube - Scheduler → Assign works to node.

4 Cluster store (Source of truth) Backup plan

⇓

All 4 runs as pods

→ containers are wrapped in pods in K8S. In docker c^r used to be naked.

GCS: Google cloud Service
ACS: Azure cloud S "
ECS: Elastic " "

Nodes are minions

1) Kubelet (main K8s agent) → listens to API server & takes care of spawning / stopping containers
2) Container engine (docker)
3) Kube proxy : Network of pods → to put containers (running inside RAM of nodes)
   NW where processes can communicate.

Reg' Node with cluster
Instantiate pods
~~Exposes endpoints on 10255~~

Kubectl get nodes

Kubectl ~~config~~ get pods --all-namespaces
⇓
To see running pods of Master

→ Pod → env (c^r)
→ one pod can have multiple cont^rs.
(docker engine takes care of OS virtualizⁿ)
→ if c^rs are tightly coupled we can put them in one pod
→ But if c^rs are independent → use 1 c^r per pod.

Pods are mortal in Nature

So Accessed by services

Air object → Services → has fixed ip & port, DNS (myservice)
Abstⁿ layer

Kubectl api-resources

{ --namespace
-n

Date:

Kubectl create -f pod.yml → Replica controller
rc.yml → has selector
→ ensures scalability

Kubectl get pods (--all-namespaces)
-n kube-system

- namespaces: all pods which are part of same app given same namespace.

Kubectl delete pod hello-pod

Kubectl create -f rc.yml
apply svc.yml.

Kubectl get rc

Service Type: (svc)
- clusterIP (Int'l Access)
- NodePort (Ext'l Access)
- Load Balancer

Based on from where its being accessed

Kubectl get svc
Kubectl delete svc hello-svc

How to update running container with Zero Downtime
(Update Deployment)

Date:
MON TUE WED THU FRI SAT SUN

Kubectl ~~create~~ apply -f deploy.yml

Kubectl get rs (replica set)

Kubectl describe deployment hello-deploy

Activities

→ ~~Creating~~ own Img & pushing it to docker hub

→ How to create pvt repo in docker hub

→ How ~~configure~~ create custom registry (maven Repo of hub)