SDLC

PAGE NO.
DATE:

Software development life cycle (SDLC)

① Requirements ⟶ the requirement gather from custo
② Design ⟶ architechure transfer requirment into technical
③ Implementation ⟶ Now developer devlop code
④ Verification ⟶ tester test the Application
⑤ maintaince ⟶ Now all ok on production than maintain it.

* Software development started, various Software development model have been curated.

① Waterfall Model

* it is a First Model in SDLC.
* in this method unitil the first stage is not finished the next stage will not start. (it is unidirectional
* onece move next step we can't back on priviouse step

advantage

* clear objects
* specific deadlines
* No ambiguous Requirements
* well understood milestones
* process and results are well document

Disadvantage

* Working product is not available unitil the last statage lifecycle
* poor model for large and complex project
* cannot accomodate changing requirement
* high risk and uncertianty.

② Agile Model

* To overcome the challenges faced waterfall model, we came up with Agile methodology.
* in this method work on priority request not a all request first priority and then other requirement.
* In this method creating shorter development lifecycle

Advantages	Disadvantage.
* customer statisfication is high	* not suitable for handling complex dependencies
* less planning Required	* knowledge transfer to colleagues can be difficult
* Requirement can be dynamic in nature	there is little documentation
* funtionlity can be created and tested quickly.	* success of the project depends heavily on customer interaction.

③ lean model

* lean development is a philosophy of increasing quality in software delivery by making use of agile method.
* use to overcome waste.
* in this method has its primary focus on two things Respect of front line Workers and continuous Improvement.

* Eliminate waste * Amplify learning * Decide as late as possible * Deliver as fast as possible * Empower the team * Build integrity * See the whole.	7 principle in lean methodology.

PAGE NO.
DATE:

advantages

* limiting Wastes saves the time and money
* creates positive working enviroment
* carries same advantage of Agile method.

disadvantages.

* largely dependent on the skill set of the team, therefore requires a strong team
* No room for error a missed delivery can be bad for business.

Why DevOps ?

* The developer used to run the code on his system, and then forwarded it to operation team

* but code run properly "it is not my problem"

* The operations when tried to run the code on their system it did not run!

* code faulty and send back

[This lead lots of back and forth, between the developer, and the operations, team, hence impact efficiency.]

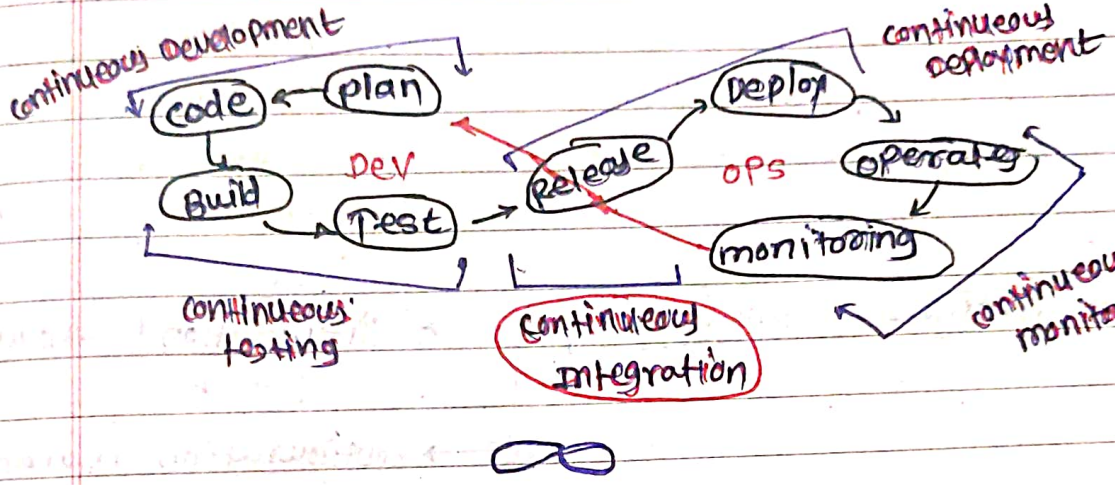
* This problem was solved using devops *

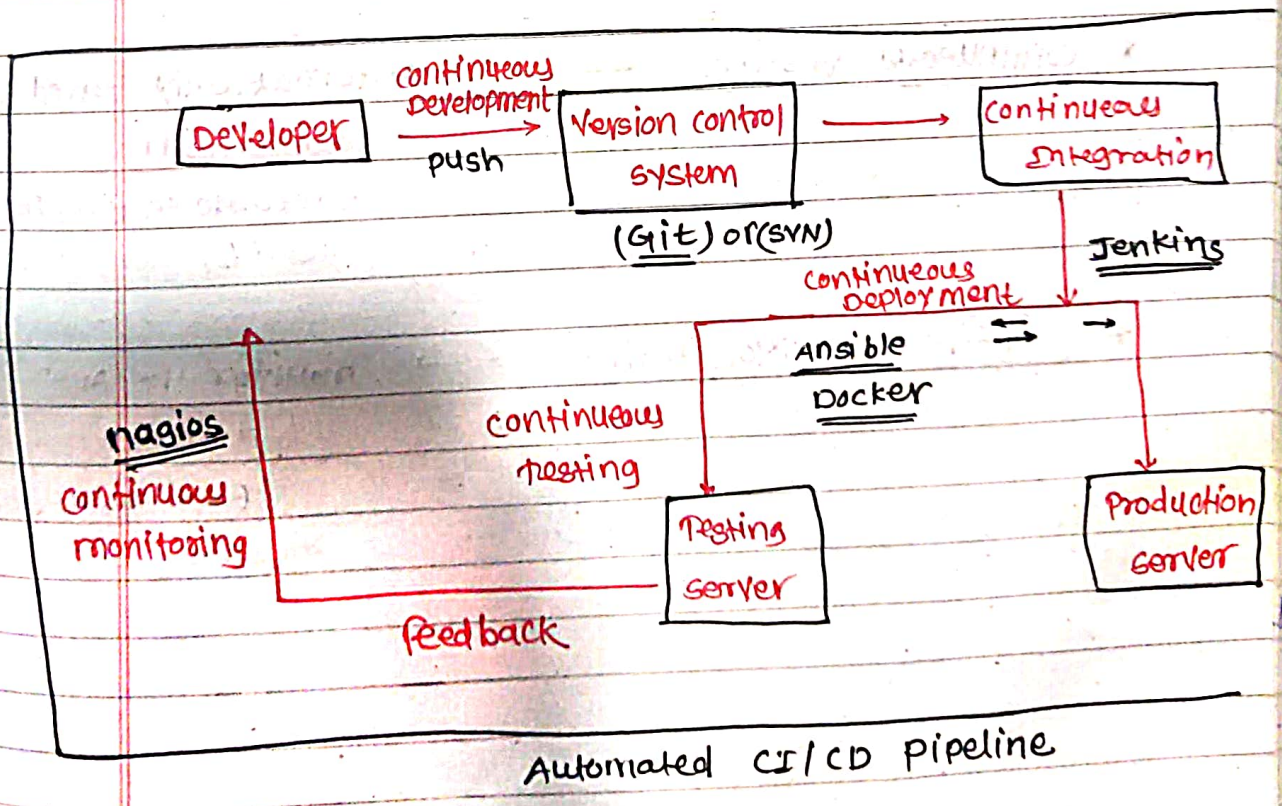
Traditional it	devops
* less productive	* more productive
* skill centric team	* Team is divided into specialized silos (tool)
* more time invested time planning	* scheduling less time in planning. (using automation)
* Difficult to achieve goal and target	* frequent release, with continuous feedback make achieving target easy.

PAGE NO.
DATE:

What is devops ?

* DeVops is a software development methodology which improves the collaboration between developers and operations team using various automation tool.
* these automation tool are implemented using stages which are a part of the devops lifecycle.





* continueous Development → developers develop code.
push the code.
(Version control system) → Git

* continueous integration → automate non-human part of the software development process.
Jenkins
(it is open source automation server written in Java)

* continueous Deployment → Virtulization & containerization
Docker
→ configuration mangement
Ansible

* continueous testing → Automatically tested API
selenium
(software testing framework used for Web Appn)

* continueous Monitoring → monitor the Appn
Nagios
(open source tool)

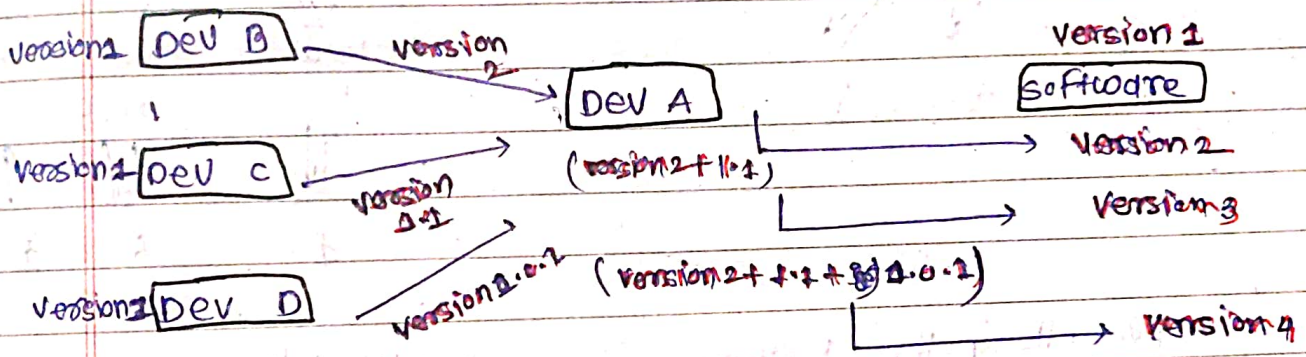
GIT

PAGE NO.
DATE:

* it is version control system tool.
* it is a system that Records / manege changes to documents, computer programs etc over time.
* it is help to tracking changes when multiple people Work on the same project.

* Before git

(manual versioning)



* versioning manual
* Team collabration was a time consuming and hetric task
* No easy access to privious version
* multiple version took a lot of spacre

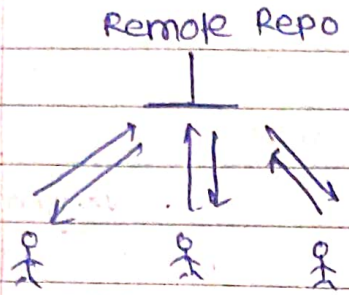
* advantages of git →

* versioning is Automatic
* Team Collaboration is simple
* Easy access to Privious versions.
* only modified code is stored across different versions, hence saves storage.

* Types of virsion control system →

binaraty files go to

* centeralized VCS

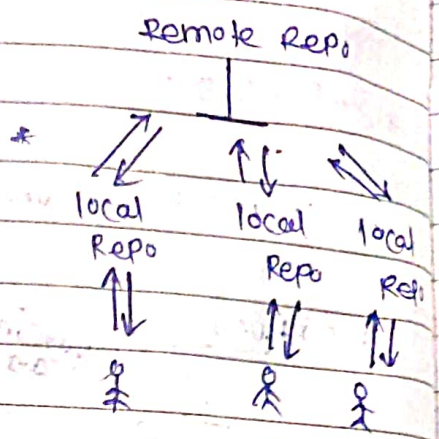


- Everything thing Required internet connection
- Developers Just have the Working copy and no version history on their local drive
- committing and retrieving action is slower since it happens on the internet
- Good For storing large files, since version history is not download
- not dependent on the number of commits

Ex. Subversion,

no binaraties files go to

* Distributed VCS



- Everything except pushing and pulling can be done without internet connection
- Every developer has full version history on local hard drive
- not Good For storing large files which are binary in nature this would increase the repo size at every commit
- if a project has a lot of commits downloading them may take a lot of time.
- committing and retrieving action is Faster since data is on local drive.

Ex - Git, perforce, mercurial

PAGE NO.
DATE:

* What is git ?

* Git is version-control system for tracking changes in computer files and coordinating work on those files among multiple people. it is primarily used for source-code management in software development, but it can be used to keep track of changes in any set of fiels.

* Git lifecycle ⟶

① Working Directory ⟶

* the place where your project resides in your local disk
* The project may or may not be tracked by git
* In either case, the directory is called the working directory
* The project can be tracked by git, ~~by git~~ using the command git init
* by doing git init, it automatically creates a hidden .git folder

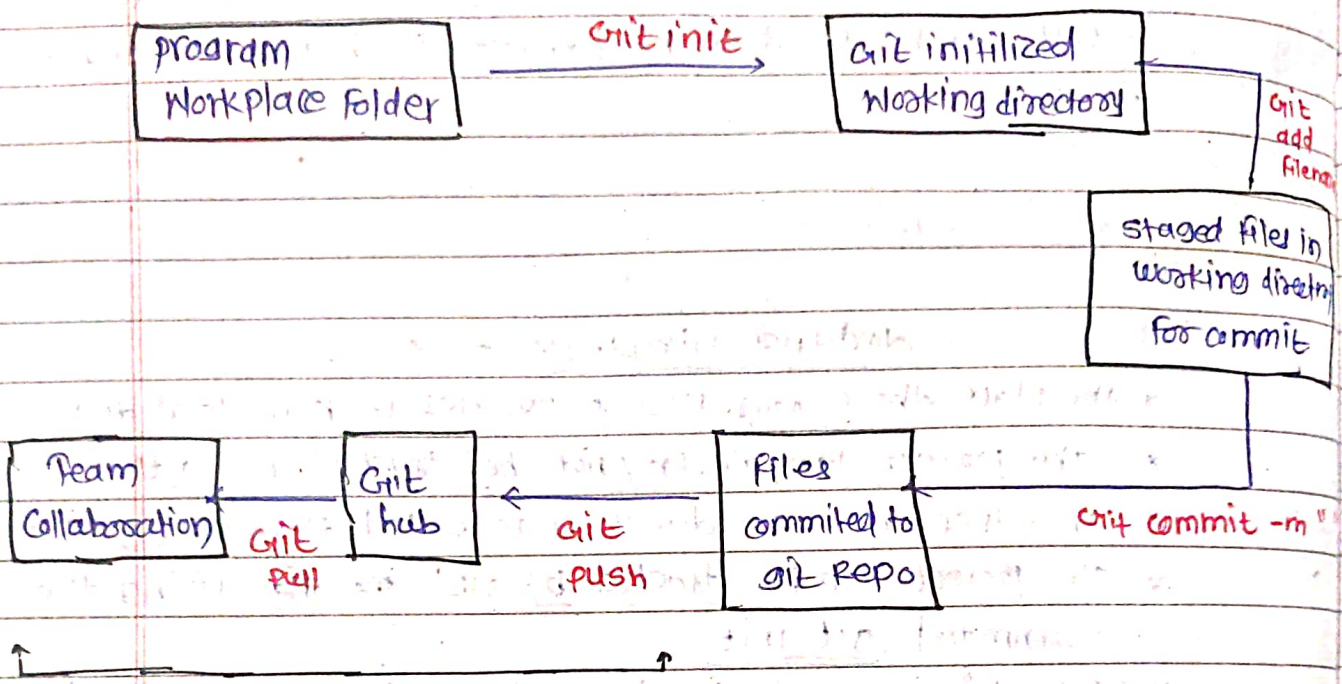
② staging area ⟶

* once we are in the Working directory, We have to specify which files are to be tracked by git
* We do not specify all files to be tracked in git, because some files could be temporary data which is being generated while excution.
* To add files in the staging area, we use command ~~git add~~ (git add .)

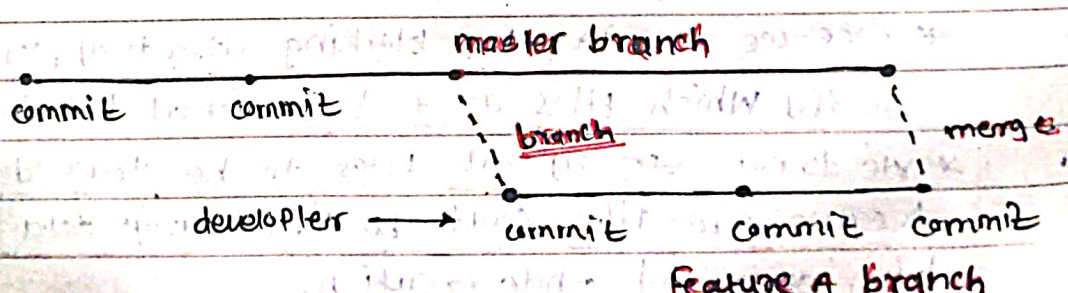
③ commit

* once the files are selected and are ready in staging area they can be saved in Repo
* saving file in Repo of git is known as as doing a commit
* when We commit a Repo in git, the commit is identified by commit id

* the command for initilizing this process is git
git commit -m "message"



* The default branch in git Repo is called the master branch.



PAGE NO.
DATE:

* common Git commands →

* ① creating Repo → * Git init (Master branch)

* ② Making changes → * Git status
* Git add . (Track files)
* Git commit -m "message"

* ③ syncing Repo with Remote →
once everything is ready on our local, we can start pushing our changes to Remote Repo
* git remote add origin "<url of Remote Repo>"
* Git push origin master
* Git clone "<URL Repo>" (copy Remote Repo to local Repo)
* git pull origin <branch> get update all files with branch (Remote)

* ④ Parallel development → multiple developers working on the same project or Repo
For that we create branch

* last commit of master which is the first commit of branch

* git branch <name of new branch> (create branch)
* git branch -D <branch name> (delete branch)
* git branch - list out all branch
* git checkout <branch name> - switch the branch.

git reset --soft HEAD^1 → Revert commit

*(git log --graph --pretty=one)

* git log → * History of Your Repo (every branch history)
* commit-id show of every commit

* git stash → want to work without commit the code.
* to switch the branch save your code without commit or staged.
* not commit the file show in your master branch.

* dangling not confused file.

* git stash pop → come back to my work which is my stash created branch

* git revert <commit-id> → * This command helps you in reverting a commit, to a priviou Version.
* commit-id get from git log
* changes in commit will be back in priviouse stage.

* after revert we want to know what changes with commit-id

* git checkout <commit-id> → * Revert all Files are committed clear and & unchang
* TO check change

* head show current commit

* git diff <commit-id1> <commit-id2> (old, new) → * This commit helps us in checking difference bet two Version in file.
git diff HEAD .

* Merging Branches →

* once the developer has finished his code/feature on his branch, the code will have to be combined with the master branch. This can be done using two ways.

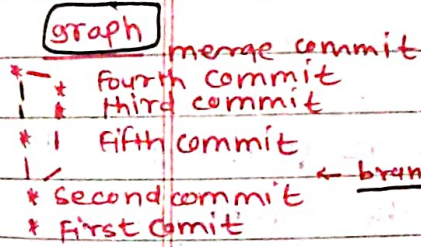
① Git Merge
② Git Rebase.

① Git merge →

* If we want to apply changes from one branch to another branch, one can use merge command.
* should be used on remote branches, since history does not change.
* create new commit, which is a merge of the two branches.
* git merge <source-branch>

(use command Where you are now)

graph



history after merge commit.

② git rebase → * git rebase <sourch-b>

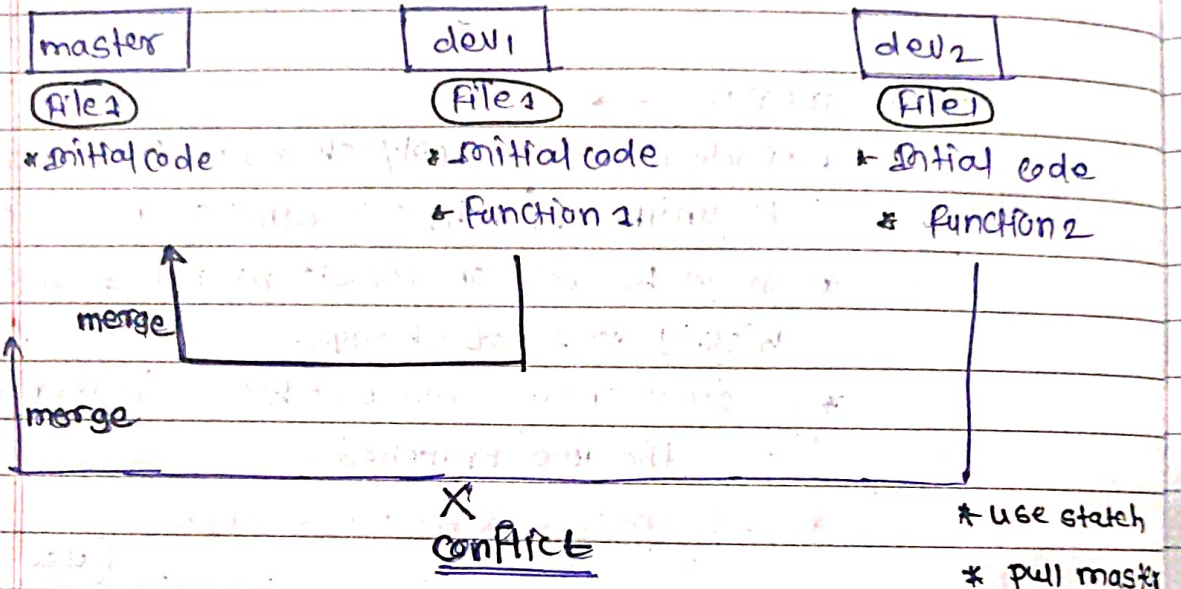
* This is an alternative to git merge command
* should be used on local branches, since history does change and will be confusing for other team members
* Does not create any new commit and results in a cleaner history
* The history is based on common commit of the two branches
* The distination's branch commit is pulled from it's "base" and "rebased" on to the latest commit on the source branch.

graph

* in local branch you use Rebase but in Remote branch use merge (ideally) (always Remember)

* Merge conflicts →

* merge conflict occur when we try to merge two branches, which have the same file updated by two different developer's.

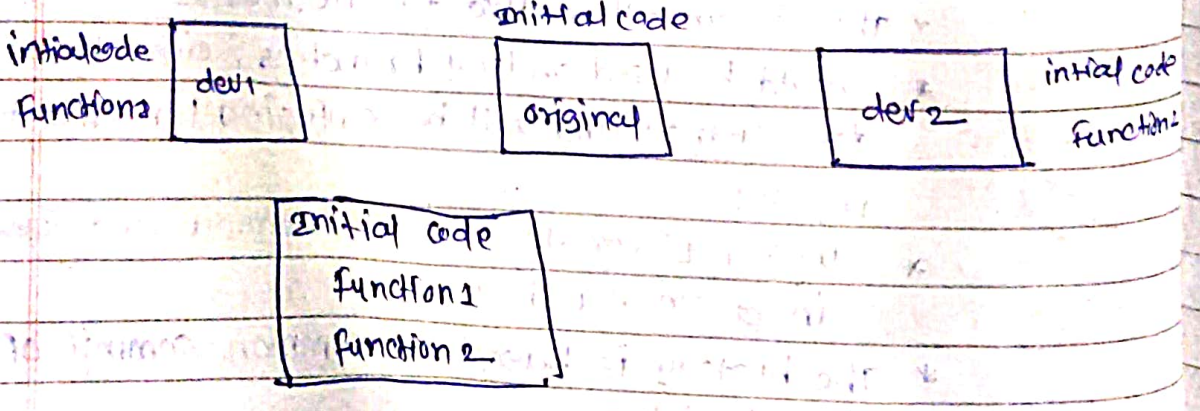


* How to resolve it?

* once we have identified, there is a merge conflict we should go ahead and use the command →

* `git mergetool`

Enter



save: wq
quit: q

PAGE NO.
DATE

* Git Workflow →

* A Git ~~workflow~~ Workflow is a recipe or recommendation for how to use it git to accomplish work in a consistant and productive manner.

* There are three popular Workflow

① Centeralized workflow

* this workflow does not required any other branch other than master
* All the changes are directly master branch and finally merged on Remote master

② feature Branching

* master only contains the production Ready code
* Any development work, is converted into a feature branch.
* once a feature is complete, the branch is merged with master.

③ Gitflow workflow

* in this workflow master, a develop, and then feature branch
* feature branch never merged directly with master, merged with develop.
* enough features in develop branch then it is merged in master branch.

PAGE NO.
DATE

* Forking in Github ⟶

Fork Repo one account to other

* Fork is a copy of Repository. Forking a repository allows you to freely experiment with changes without affecting the original project.

Docker

Before docker

* Developers used to run the code on their system it could run perfectly. But the same code did not run on the operations team system.
* dependency problem face both.
* To overcome dependancy before docker we VM on every one system. it take large space and face latency. and complex to handle.

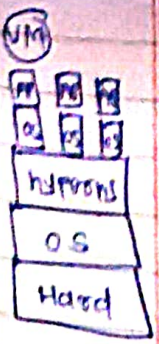
→

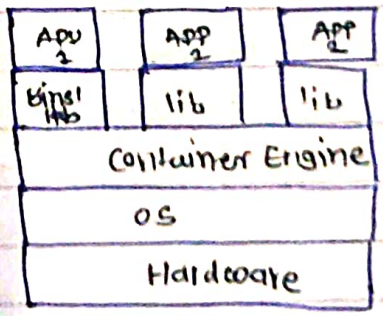
* developer environment, testing/operation environment and production environment is same. it is possible by using docker.

* What is Docker? →

docker solo release.

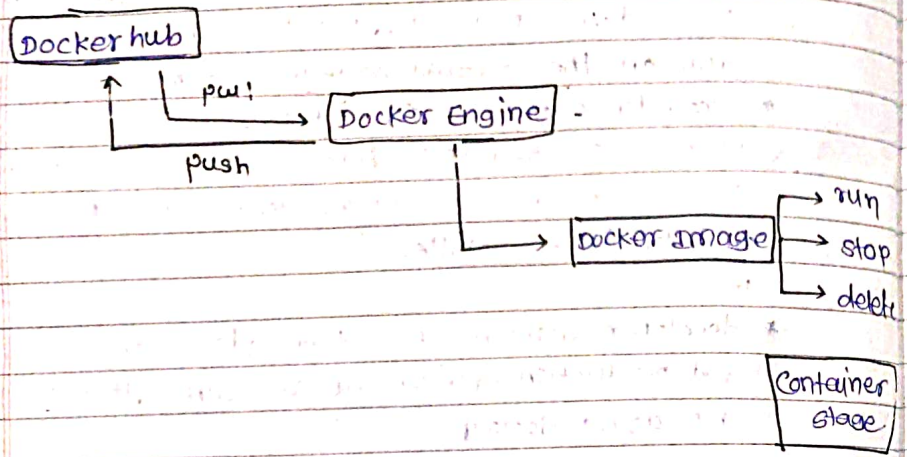
* docker is a computer program that performs operating-system-level virtulization also known as "containerization".
* Docker is used to run software pakages called 'container"
* container size is low size
* container easy to handle.



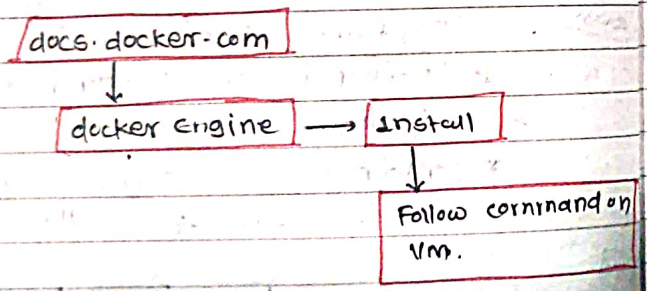


* Docker flexible, scaleble, portable, lightweight.
* Doker is client server architecture.

* Docker container lifecycle →



* Docker download →



* Docker-hub Account →

* common Docker operation →

* **docker --version** → this command help to check docker install version

* **docker pull <image-name>** → help to pull image from dockerhub on your system.

* **docker images** → To check all images you download and build.

* **docker run -d -it <image-name>** →
  * run image container in background and also open shell.
  * **-d** (backgroun run)
  * **-it** (intractive termin

* **docker ps** → * To check all running container
  * container id show

* **docker ps -a** → To check all running and exited container

* **docker exec -it <container id> /bin/bash** →
  → open interactive shell of container when container running.
  → logging | accessing container

* **docker stop <container id>** → stop the container

* **docker kill <container id>** → * Kill the containe
  * with fast
  * comparison to stop command

* rm -f <id> → running container

* docker rm <container-id> → * removing the container is exit state or stop state.

* docker rmi <image-id / Name> → * remove the image.

* docker commit --author "kshikj" -m "hi" <id> name for image

command → * make image from running container * saving changes.

* saving changes to a container →

* docker container prune → * delete all stoped container at a time

* docker image prune → * delete all image at a time

* docker inspect <container-id> → * To check info IP, Volume, network

* Mapping →

* docker run -d -it -p 80:80 --name test <nginx> → * mapping with 80 host port with 80 container port

* History →

* docker history <id/name> → * get idea about layers * optimize * Reduce instruction of dockerfile

* pushing container to docker-hub (ECR)

* after commiting the image you want to push on docker-hub.

* docker login → * login to your dockers-hub account.
username:
pass :

* not done with this command

* docker push <name:image> → Now You can push your Image

* docker image tag Imagename:version kshikjss/Repo:version → * Tag Image with local Image to Dockershub Repo

* docker image push kshikjss/Repo:version → push image on dockershub.

* rm -f <id> → running container

* docker rm <container-id> → * removing the container is exit state or stop state.

* docker rmi <image-id / name> → * remove the image.

* docker commit --author "kshikj" -m "hi" <id> name for image

command ↑

→ * make image from running container
* saving changes.

* saving changes to a container →

* docker container prune → * delete all stoped container at a time

* docker image prune → * delete all image at a time

* docker inspect <container-id> → * To check info IP, volume, network

* Mapping →

* docker run -d -it -p 80:80 --name test <nginx>

→ * mapping with 80 host port with 80 container port

* History →

* docker history <ID/name> → * get idea about layers
* optimize & Reduce instruction of dockerfile

* pushing container to docker-hub (ECR)

* after commiting the image you want to push on docker-hub.

* docker login → * login to your docker-hub account.
username:
pass :

* not done with this command

* docker push <name:image> → Now You can push your image

* docker image tag Imagename:version kshikjss/Repo:version

→ * Tag image with local image to Dockerhub Repo

* docker image push kshikjss/Repo:version → push image on dockerhub.

* Dockerfile → [File Name Dokerfile]

* A dockerfile is a text document has contains all the commands a user could call on the command line to assemble an image.
* using docker build users can create an automated build the executes several command-line instructions in succession.

* FORM → The From keyword is used to define base image, on which we building
  * FROM ubuntu

* ADD → the ADD keyword used to add files to container being built. (also link file)
  * ADD <source> <destination in container>

* RUN → * The RUN keyword is used to add layer to the base image
  * Each RUN keyword add a new layer in docker image.

* CMD → * The CMD keyword used to run commands on the start of the container
  * run only when there is no argument specified while running the container.
  * it changeable by CLI.
  * CMD ["nginx" "-D" "daemon off;"]

* ENTRYPOINT → * This keyword used strickly run run command the moment the contain
  * Difference CMD and Entrypoint is Entrypoint run irrespective of the argume or not.

initialize

* It is not changle via CLI.

* ENTRYPOINT ["nginx" "-D" "daemon-off:"]

* ENV → * The ENV keyword is used to define environment variable in container run-time.
  * ENV name DevOps
    (key) (value)
    ↳ To check
    Go into container
    * echo $name

* WORKDIR → * This keyword used to we can working inside directory of contain

* COPY → * used to copy file form local Repo to container
  * same as ADD but in copy can't transfer link file.

* EXPOSE → * using this keyword expose the port of container.

* Build Dockerfile ⟶

* docker build -t Image name:version -F DockerFile.Txt .

→ docker file build with image name with tag.

ECR

* push Image on ECR (Elastic container Registry)

Transfer Image form local to AWS ECR.

* ECR ⟶
* it is a artifact management
* next Version code savers

go to ECR
↓
Make private Repo
↓
mutability
↓
Encription
↓
Image scanning setting off
↓
create.

Now

Click on Repo
↓
View on push command
[login, tag, push]
↓
copy command one by one
paste in local machine cmd
↓
push Image

* Docker Volume ⟶ * persisting data.
* we can attach volum to our container
* When container delete but volume will save it can not delete.

* docker volume create <name> → * create a Volume

* docker volume ls ⟶ * To check volume list

* docker volume inspect <name> →
* get all info volume
* also get path of Volume (distination

* docker run -d --name < > -V <name>:path (source) (dist)
nginx
<image>

→ * map the Volume with your Image container run.
* give them path.

* docker run -d bindmount <name> nginx
<image>

→ otherway to mount Volume.

Three types Volume ⟶
① bind mount
② Volume
③ tmpfs mount.

* Docker Network →

* docker network create -d bridge my-net →
  <Name>
  creat a network

* docker network ls → list down a network.

* docker run --network=<Name> -it -d nginx. →
  → attach network to container.

Types of network
1. driver
2. none
3. bridge.

AWS

ECS

Elastic container service

* it is a orchestration tool of AWS
* How to your code deploy in production environment for this we use ECS
* efficient deploy, manege and scale containerised Application
* it is a cloudnative.

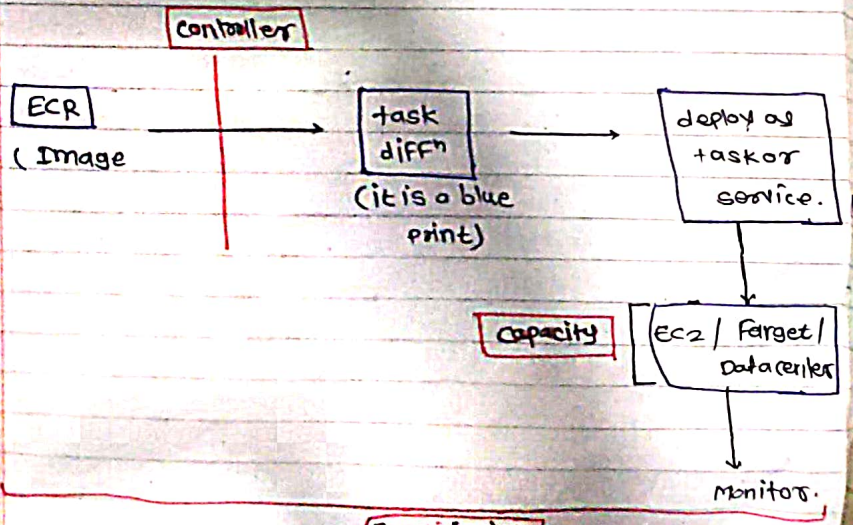
* Types of ECS →

1. capacity → Infrastrure to container run
2. controller → help to deploy and manege
3. provisioning → it is help to How to deploy and manege our App^n & container

SDK (software development kit)

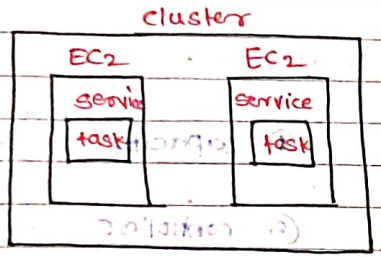
CDK (cloud development kit)

* How it Work? →



* cluster → 1.
  * task / service run on cluster
  * it is logical grouping

Image → Task diffination → cluster → service / task.



* Steps For creating cluster & deploy the Appn →

DockerFile
↓
build Image
↓
Push Image on ECR
↓
Copy URI of Image

* Task diffination create

Now
↓
Go to Ecs
↓
task diff. create New
↓
diffination family name
↓
Infra Required — EC2 * / Fargate *
↓
OS
↓
Network Mode
↓
task Role
↓
task-excution Role (Create)
↓
Name — container-1
URI
Essential Container-Yes
↓
port mapping
↓
log collection off
↓
create

* create Role and add policy ←
  * ecsexicutiontaskRole

* Now cluster creation →

Create cluster (Go to cluster)

↓ Name

↓ Default Namespace optional

↓ Infra

EC2 *

Fargate. * (Serverless)

* task create there use Infra is same in cluster

↓ create

* Now created a service in cluster →

* In service we add task
* and service inside the cluster.

Go to cluster

↓ Services

↓ Create

↓ launch type

EC2 / Fargate
(Infra selected in task diff⊕ cluster)

↓ Plate form version

↓ Application type

↓ service

↓

↓ Select family name

↓ Revision

↓ service name

↓ service type (Replica)

↓ Desired task

↓ Networking

YPC , subnet (3)

* create YPC
* create 5 subnet
  3 → private
  2 → public
* create NAT in pub sub
* Entry of NAT in private Route t
* private subnet give to service network
* using public subnet create load balancer using ip target group.

↓ Security group

↓ load balancer

* type
* Create or exiting
* listner Rule
* target group

↓ Create.

* Go to Route 53 → Register DNS to Domain

*

Route 53

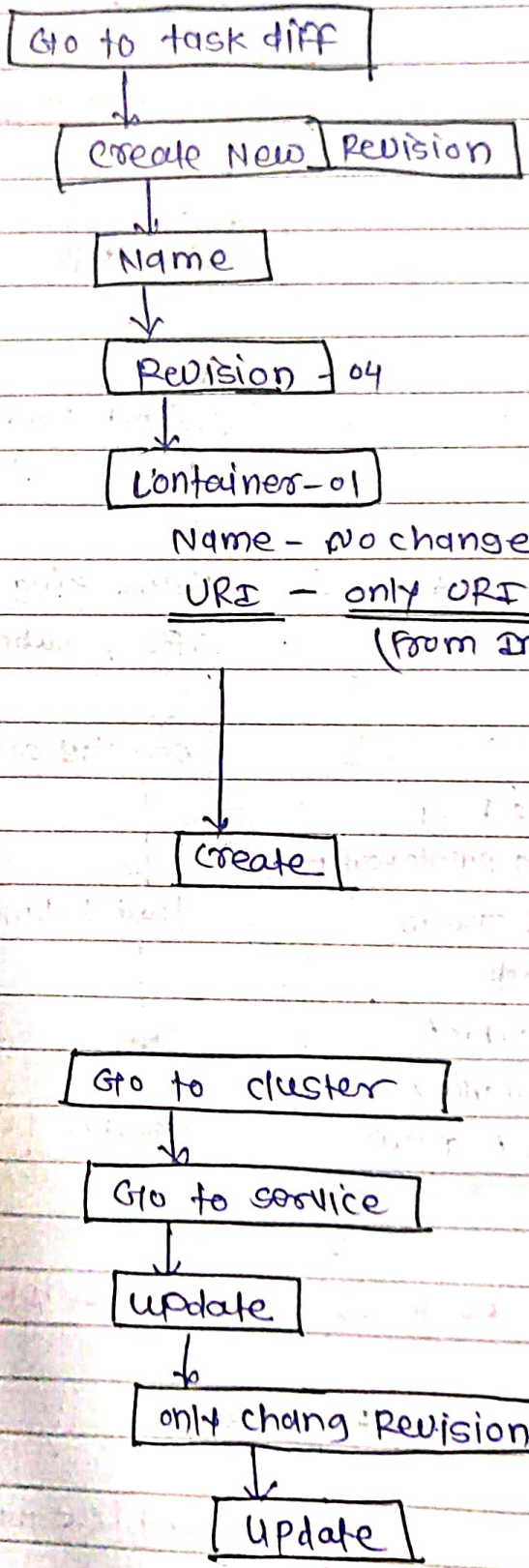
↓ Create Record (C-Name)

↓ Paste DNS

↓ Create

PAGE NO.
DATE

* Reversioning of your Appn code. ⟶



* All is same in task diffenation

Now

Kubernetes [K8's]

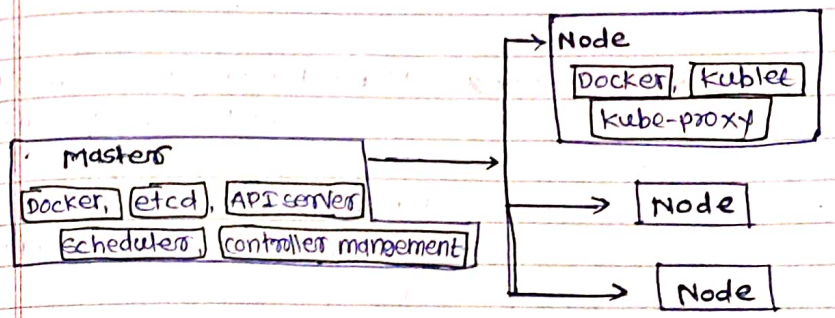
* Container orchestration using Kubernetes.
* it is developed by google date to publicly 21st 2015
* it is open-source tool.
* K8's based on bule-green deployment method.
* Features of K8's →

* pods
* Replication controller
* Storage management
* Resource Monitoring
* Health checks
* service Discovery
* Networking
* secreat Management
* Rolling Updates

* Difference between Docker swarm Vs K8's

Docker swarm	K8's
* based on popularity is less	* popularity is more and widely serches.
* Easy to install and intilize	* complex procedure to install K8's
* faster When compared to K8's	* slower when compared with Docker swarm
* Not Reliable and has less features	* more reliable and comparatively has more features

* K8's Architecture →



K8's

* Master compoent →

① etcD → * it is a highly available distributed key Value store. Which is used to store Cluster wide secrets.
* it is only accessible by K8's API server as it has sensitive info
* etcD is storage of K8's

② API-server → * it expose K8's API.
* The K8's API is the front-end For K8's controle plane, and is used to deploy and excute all operations in K8's

③ schedular → * The schedular takes care of scheduling of all the processes, dynamic Resources management and manages present and future events on the cluster.

④ Controller-manager → * the controller maneger run all the controllers on the K8's cluster
* manage running pod
* Killpod, cpu utilization, Network command.
* Endpoint controller
* manage Replication controller
* Node controller.
* token Controller

K8's

* Slave Node compoents →

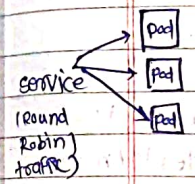
① Kubelet → * kublet takes the specification form API server and ensure the Application is running according to the specificat which were Mentioned.
* Each node has it's kublets service

② kube-proxy → * this proxy service runs on each node and helps in making service available to the external host.
* it helps to connection forwarding to the correct Resources
* it is also capable of doing primitive load blancing.

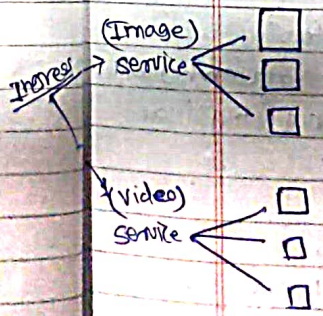
* Working of K8's →

* pods → * pods can have one or more containers coupled together.
* They are the basic unit of K8's
* To Increase high availibity, we always Prefer pods to be in replicas.
* Replication happen in pod level not in container level

* service → * service are used to load balance the traffic among the pods.
* it follows round robin distribution among the healthy pods.



* Ingress → * An Ingress is an object that allow access to your K8's services from outside the K8's cluster.
* Your configure access by creating a collection of rules that define which inbound connections reach with services.



* Make cluster First → by using Note1

* Deployment in K8's

  * Deployment in k8's is a controller which hepls your Applications reach the desired state, the desired state is defined inside the development file

* create yaml.File →
  * Yaml syntax for Deployments.
  * make a yaml syntax for a pod of Nginx server.
  * it is deployment File.

* kubectl create -F nginx.yaml <~~name~~>
  <Name>
  → once File create, to
  * deploy the deployment use this syntax
  * using this create deployment

* kubectl get pods →
  * To view the pod which set deployment File.
  * you can see matching with the number of Replicas set in file.

* kubectl get pods --all-namespaces →
  * To show all App'n pod in k8s cluster

* Kubectl APPly -F nginx.yaml → * this is command for you changes in Yaml file and Apply

* kubectl get pods -o wide →
  * show pods where exit (which node)
  * shows all pods IP this IP which is k8's cluster Network
  * it is not outside Accessible.
  * this IP accessible into a cluster.

* curl <IP> → * To check the what is inside the pod.

* creating a service →

* A service is basically a round-robin load balancer for all the pods
* it is constantly monitors the pods.
* in case pod get unhealthy, the service will start deploying the traffic to the other healthy pod.

* service types →

① **cluster IP** → Exposes the service on cluster-intern. IP

② **Nodeport** → Exposes the service on each Node's IP at a static port (outside the world)

③ **load balancer** → Exposes the service externally using cloud provider's load balancer. (always in pending state)

④ **External Name** → Maps the service to the contents of the External Name

* creating service syntax. (on Master)

* create service →
  * **kubectl create service nodeport** (or clusterIP or load balancer) **<name>** (of service) **--tcp=** <port-of service> : <port of. cont>

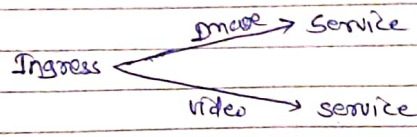
(clusterip) only work in cluster get page not in browser

* check all service →
  * **kubectl get svc**

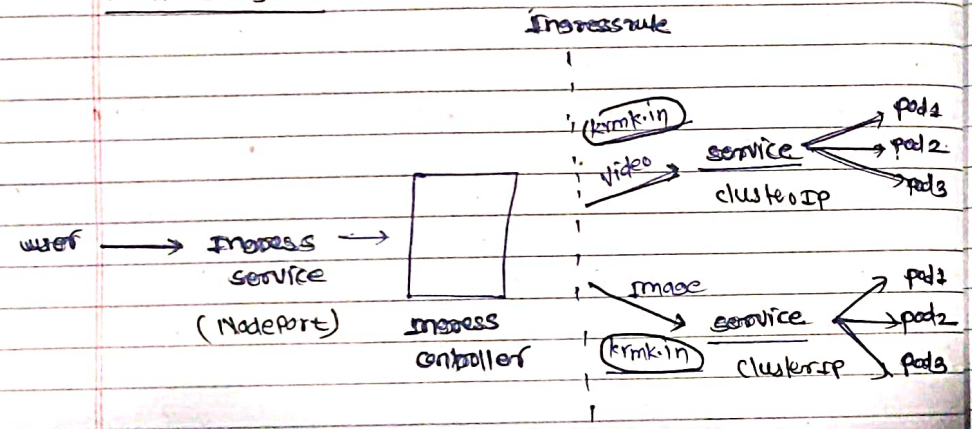
* delete of service →
  * **kubectl delete service <name>**

* Creating of Ingress

- Kubernetes ingress is a collection of routing rules that govern how external users access services running in a kubernetes cluster.



* What is ingress →



* Installing ingress controller →

Nodeport

nginx ingress controller (search)
↓
get github page
↓
deploy
↓
Readme
↓
docs/deploy.

* mandatory command
* bare-metal service

pod created
service created

* check nodeport http with IP

* kubectl get pod --all-namespaces
* kubectl get svc

* define Ingress rule →

* The following rule (ramal) will redirect traffic which ask for /Foo to nginx service.
* All the other requests will be redirect to ingress controller's default page.

```
apiVersion : extension/v1beta1
kind : Ingress
metadata :
  name : nginx
  annotations :
    nginx.ingress.kubernetes.io/rewrite-target:/

spec:
  rules :
    - http:
        Paths :
          - path: /Foo
            backend :
              Servicename : nginx
              Serviceport : 80
```

* kubectl create service clusterip nginx --tcp=80:80
* nano demo.yaml
* Kubectl create -f demo.yaml
* Kubectl get ing

* kubectl delete ingress name → PAGE NO. ip DATE 194 master

* k8's DashBoard →

* Kubernetes object →

* 11 types of object in k8's
* object are the blueprints
* scale, manage, deploy (without zero down time)

① pod → * scale, manage, containerization App
* all container always run on a single worker node.
* pod run one or more container

② Depolyment → * used to manage the lifecycle of one or more identical pod.
* using deployment versioning done.

③ Replica set → * set the desired, min, max value of pod
* deployment don't manage directly.
* act like as autoscal

④ Stateful set → * use to manage stateful application
* it ensure that each pod uniquely identified by a number starting with zero
* same pod is replace with privious identity.

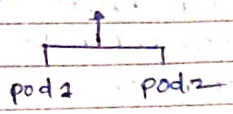
⑤ Daemon sets → * a daemonset ensures that a copy of pod
* monitoring agents running accross all

it is task diff
✓ depolyment
service
ingress

⑥ persistant Volume → * it is represent a piece of storage that can attach your pod
* it is a volume a store the data.

pod ✗ Volume ✓

⑦ service → * it is a way access a group of pods that provide same functionality.



⑧ namespaces → * it is a way of divide single k8's cluster into multiple virtual clusters.

⑨ configmaps → * it is very imp
* store the non-sensitive config value

⑩ secrets → * hold sensitive value (pass, token)

⑪ Job → * run specific task

Terraform

* What is the infrastructure as code (Iac) →

(not using management concept)

* Infrastructure as a code is the management of infrastructure (network, VM's, load balancers, and connection topology) in a descriptive model, using a same versioning as Devops team uses for source code.
* iac envolved to solve the problem of enviroment drift in the release pipeline.

iac	configuration (cm) management.
① can create/destroy hardware architectures	① Work only with softwares
② can install software while bootstrapping servers	② can' not coook on hardware level, but can install any software
③ should not be used as a replacement to cm tools	⑤ cannot be used as a replacement to iac

* introduction of terraform →

* Terraform is an open-source infrastructure as code software tool created by Hashicorp.
* it enables users to define and provision a datacenter infrastructure using a high-level configuration language known as Hashicorp configuration language, or optionally Json.

* Basic Terraform operations →

* Make Folder
* use command * terraform init
  - → make a empty folder
  - → also you have code download all plugin for provider.
* provider.tf →

```
provider "aws" {
    region = "ap-south-1"
    access_key = "  "
    secret_key = "  "
}

resource "aws_instance" "name" {
    ami = "  id  "
    instancetype = "  t2.micro  "
}
```

* terraform init → intilize and download all plugin
* terraform plan → show all the value which is deployed, give info you check all things.
* (~ change, + add, - delete)

1+ add 0 change 0 destroy

* terraform apply → create your deployment and make you want all things.

* terraform destroy → your need is end you can delete your Infra using this command. all Resource are delete.

* What happen when use this command: →

* terraform init → make . terraform folder
. terraform.lock.hcl

* terraform plan → you get {} terraform.TFstate
(not make just give idea)

* terraform apply → create {} terraform.tfstate which all info your code running.

* terraform destroy → create {} terraform.backup.tfstate