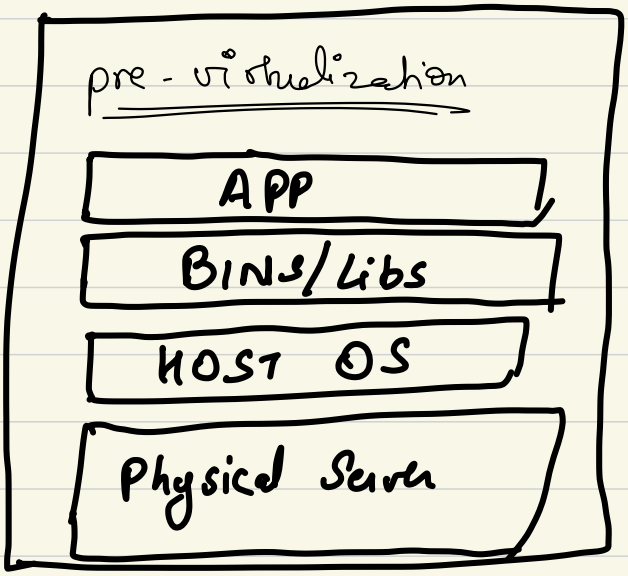
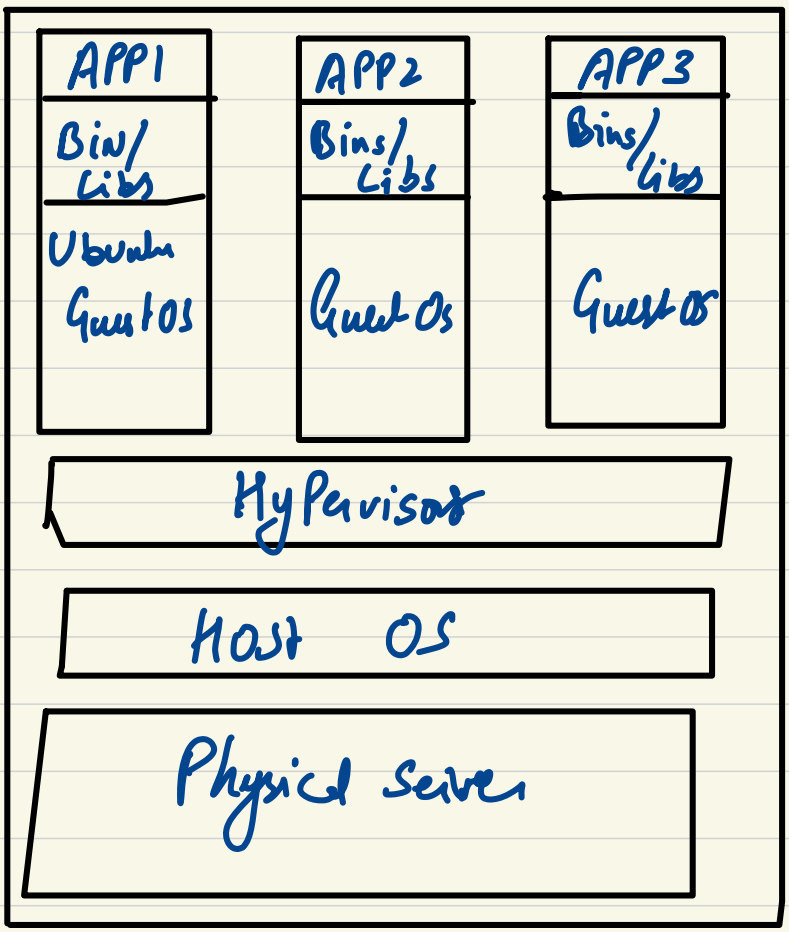
Introduction to virtualization technologies

Docker technology is one implementation of container based virtualization technologies



problem
1. purchase physical machine
2. low utilization
3. cost.
4. Deployment is slow
5. Config is very complex
6. Hard to migrate



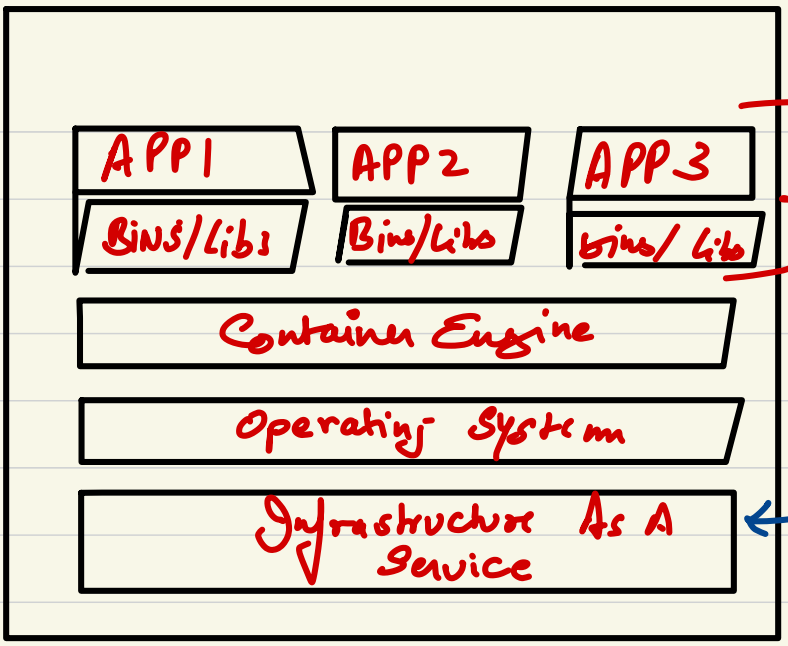
VMware or Virtual BOX

- Cost Efficient
- Easy to scale (on cloud Envt.)

Limitation

- Kernel Resource Duplication
- App portability issue.

- Virtualization happens at H/w level.



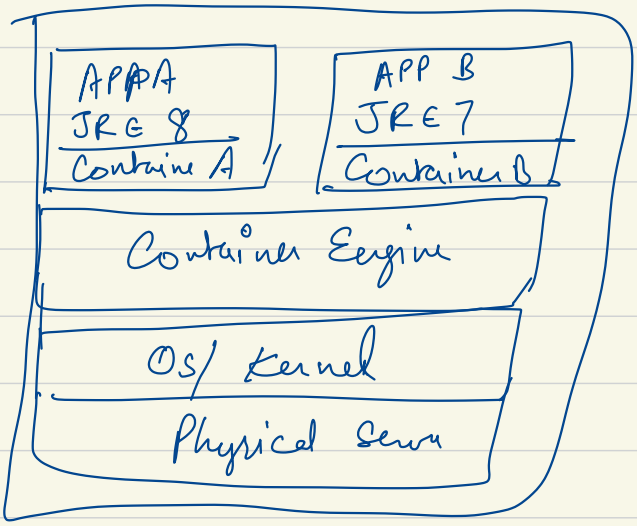
Virtualization happens at OS level

- Cost Efficient
- Fast deployment.
- Greate portability

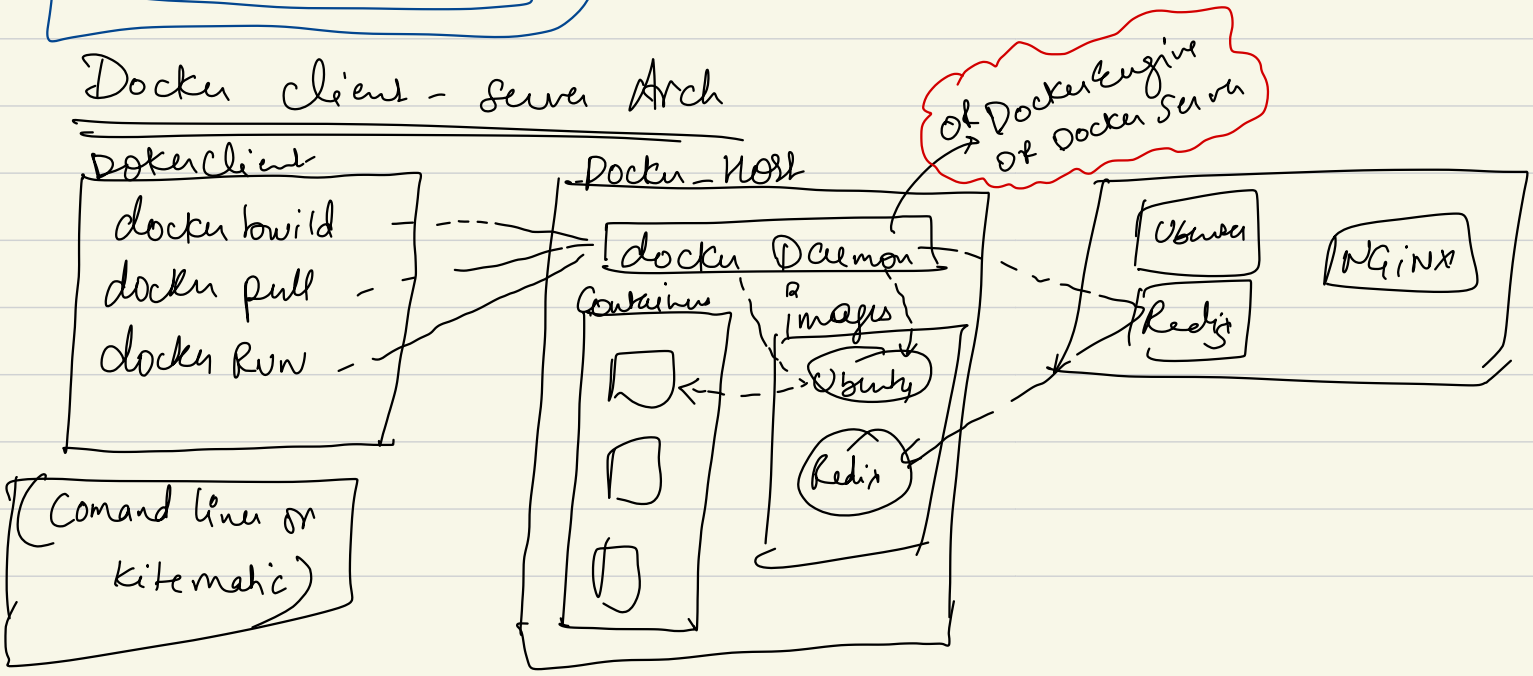
Runtime isolation.

How to run two Java apps with two different JRE on same VMs ? (conflict will happen)

Soln



Docker client - server Arch



Docker client can also connect to a remote docker daemon; typically they are on same machine (as our local)

Docker daemon can't run on non linux platforms natively because it uses linux specific kernel feature

Important Concepts.

Images → ① Images are read only templates that used to create containers
② Images are created with the docker build command, either by us or by other docker user.
③ Images are composed of layers of other images
④ Images are stored in a Docker registry

containers → ① If an image is a class, then a container is an instance of a class runtime object.
② Containers are light weight and portable encapsulations of an environment in which to run application
③ Containers are created from images. Inside that container, it has all the binaries & dependencies needed to run the application

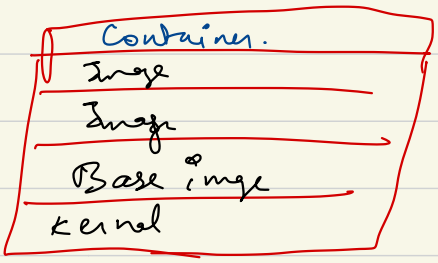
Registry & Repositories: ① A registry is where we store our images
② You can host your own registry, or you can use docker's public registry Docker HUB
③ Inside a registry, images are stored in Repositories.
④ Docker Repos are collection of different docker images with the same name, that have different tags, each tag usually represents a different version of the images

① Docker will first look for the image in the local box. If found, docker will use the local image! other wise docker will download the image from Repository.

② Docker Run: Comand will create the container using specified Image, then spin up the container and Run it

Run container in foreground: Docker run starts the process in the container and attaches the console to the process's standard input, output & standard error
Console can't be used for other commands after the container is started up.

Run Container in background: Containers started in detached mode and exits when the root process used to run the container exits.
-d is used to specify detached mode. and console is available.



Docker image layers:

A docker image is made of a list of read only layers that represent file system differences. images are stacked on top of each other to form a base of container file system.

"docker history image_name:tag" to list image layer history

1. all changes made to the running container will be written into a writable layer.
2. When a container is deleted the writable layer also deleted. but

the underlying image remains unchanged.

3. Multiple containers can share access to the same underlying image.

'Docker build' context.

① Docker build command takes the path to the build context as an argument.

② When build starts, docker client would pack all the files in the build context into a tarball then transfers the tarball file to the daemon.

③ By default docker will search for the dockerfile in the build context path.

Dockerfile in depth:

① Each Run command will execute the command on the top of writable layer of the container, then commit the container as a new image.

② The new image is then used for the next step in the dockerfile. So each RUN instruction will create a new image layer.

③ It is recommended to chain the Run instructions in the Dockerfile to reduce the number of image layers it creates.

Sort Multi-line Arguments Alphanumerically

① This will help you avoid duplication of packages and make the list much more easier to update.

CMD instructions:

- CMD instruction specifies what command you want to Run when the container starts up.
- If we don't specify CMD instructions in the docker file, Docker

will use the default command defined in the base image.

- The CMD instructions doesn't RUN when building the image, it only Runs when container starts up.
- You can specify the command in either exec form which is preferred or in shell form.

Docker Cache

- Each time docker executes an instruction it builds a new image layer.
- The next time, if the instruction doesn't change, docker will simply reuse the existing layer.

Aggressive caching

Advantage is fast build

Disadvantage is in this case apt-get update will never RUN will get outdated git & curl

```
From ubuntu:14.04
RUN apt-get update
RUN apt-get install -y git
```

Since each line executes on new Writtable image Docker caches the old image and reuse it when necessary.

```
FROM ubuntu:14.04          ← Reuse cache
RUN apt-get update         ← Reuse cache
RUN apt-get install -y git curl.   ← (New Image)
```

Solution to this is chain apt-get update

Solution 2: Tell Docker to invalidate cache.

```
RUN apt-get update && apt-get install -y \
  git \
  curl
```

```
docker build -t ak/debian . --no-cache =true.
```

Copy instruction :- The copy instruction copies new files or directories from build context & adds them to the file system of the container

Add instruction :- ① Add instruction can not only copy files but also allows you to download a file from internet and copy to container.

② Add Instructions also has the ability to automatically unpack compressed files.

③ The rule is : use COPY for the sake of transparency, Unless you are absolutely sure you need ADD.

Push Images to docker hub

① list images using Docker Images. Choose image you want to push

② RUN : Docker tag {Your_Image_Id} dockerhubAccName/RepoName:{tag}

- If tag is not provided by default 'latest' tag is used.
- A lot of repositories use it to tag the most up-to-date stable image, However, this is still only a convention and is entirely not being enforced.
- Images which are tagged latest will not be updated automatically when a newer version of the image is pushed to the repository.
- Avoid using latest.

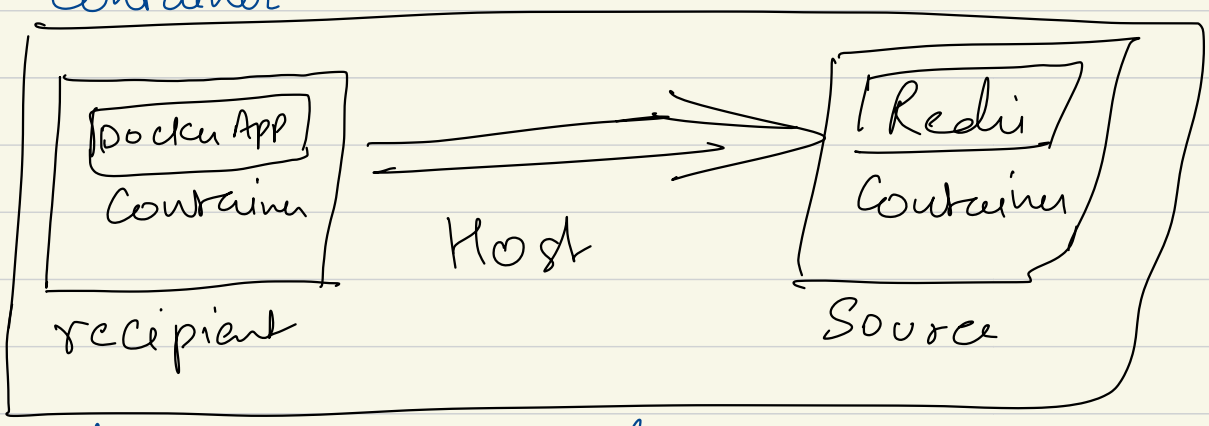
③ check your new image using Docker images

④ Docker login -- userName : yourUserName.

⑤ Docker push RepoName : tag.

Docker Container links:

Container links allow containers to discover each other and securely transfer info about one container to another container



When you setup a link, you create a contract/conduit b/w a source container and recipient container. The recipient container then can access select data about the source container.

The links are established by using container names.

How links work internally

① Go to pyApp Container
docker exec -it {container id} bash

② Open /etc/hosts file.
When a linux machine gets started it would need to know mapping of some hosts names of the IP addresses before DNS can be referenced. this mapping is kept in this file.

③ You'll find entry for redis container, with IP, Container name & container id you can verify it using docker inspect for Redis.

with docker ps (lists)

Benifits of container linking

1. The main use for docker container links is whenwe build an application with a microservice architecture we are able to run many independent components in different containers.

2. Docker creates a secure tunnel b/w containers that doesn't need to expose any ports externally on the container.

Docker Compose

Why docker-compose?
Manual linking containers and configuring services become impractical when the number of containers grows.

Docker compose is a tool for defining and running multi-container Docker applications. With compose, you use a compose file to configure your application's services. (docker-compose.YML)
Then using single command, you create and start all the services from your config.

Docker-compose is a very handy tool to quickly get docker environment up & running.

Docker-compose uses Yaml files to store the configuration of all containers, which removes burden to maintain our scripts for docker or Orchestration.

Useful Commands (Run from compose file directory only)

docker-compose ps → list containers run by Compose
docker-compose logs → show logs
docker-compose logs -f → show logs and follow logs
docker-compose logs {container_name} → specific logs

docker-compose stop
docker-compose rm
docker-compose down

A typical docker-compose workflow usually invokes making changes to the source code or images and rebooting the container afterwards.

* if we make changes to our code and run docker-compose up, changes will not reflect in container, because docker-compose up never rebuilds image if it already exists

* use docker-compose build to rebuild image. then run docker-compose up.

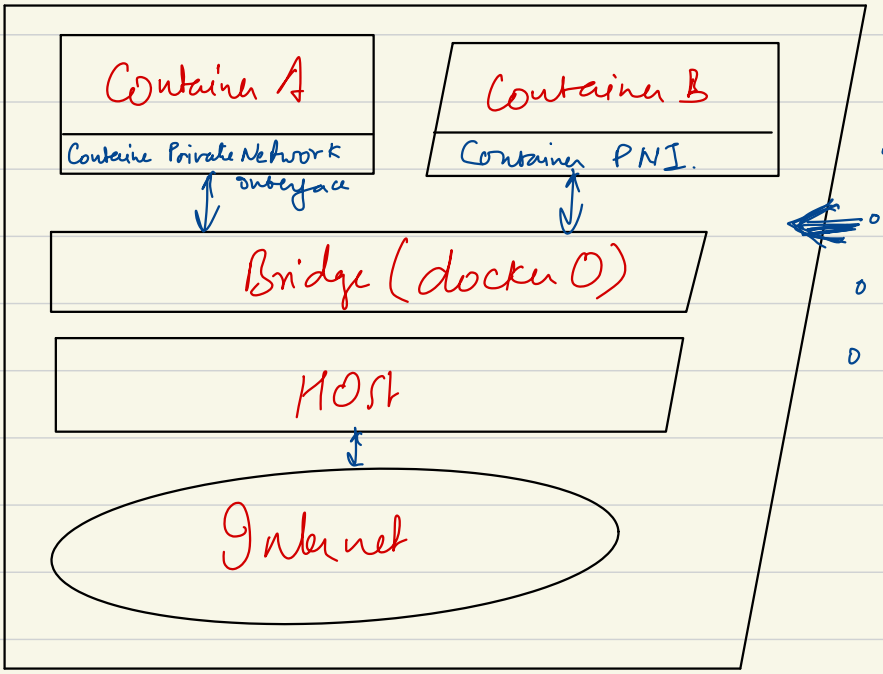
Docker-Container Networking

Docker uses the networking capabilities of the host machine OS to provide networking support for the container running on the host machine.
Once the docker daemon is installed on the host machine, A bridge network interface "Docker 0" is provisioned on the host

which will be used to bridge the traffic from the outside network to the internal containers hosted on the host machine.

- Each container connects to the bridge network through its container network interface.
- Containers can connect to each other and connect to the outside world through this bridge network interface.

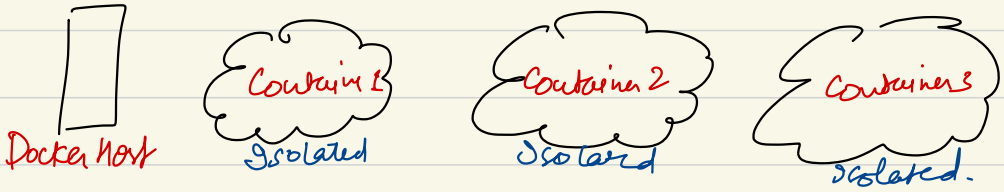
This is how default docker network model called bridge network looks like.



Docker Network types

- Closed / None network
- Bridge Network
- Host Network
- Overlay network.

1) **None Network**. This network doesn't have any access to outside world. the none network adds a container to the container-specific network stack. That container lacks a network interface.



Sudo docker run -d --net none {Image}

- provides that maximum level of network protection
- Not a good choice if network/Internet is required.
- Suits well where the container requires the max level of network security & network access is not necessary.

2) Bridge Network : This is the default type of networks in docker containers. All the containers in the same bridge network are connected with each other and they can connect to the outside world via the bridge network interface.

- In bridge network containers have access to two network interfaces.
  - A loopback interface (same as None)
  - A private interface (to connect bridge)
- All containers in the same bridge network can communicate with each other.
- Containers from different bridge network can't communicate with each other by-default (we can manually connect them)
- Reduces the level of Isolation in favour of better outside connectivity.
- Most suitable where you want to setup a relatively small network on a single host.

3) Host Network : - The least protected network model, it adds a container on the host network stack.

- Containers deployed on the host stack have full access to the host's interface.
- This kind of containers are usually called open containers.
- When you see if config of host network it will show all the network interfaces that are available from host. of the container

- Minimum network security level.
- No isolation on this type of open containers, thus leave the container widely unprotected.
- Containers running in the host network stack should see a higher level of performance than those traversing the docker0 bridge and iptable port mappings.

4) **Overlay Network:** All above network models have one limitation which is they can only be deployed on single host.
If you want to create a network accross multiple host machines, you need overlay network.
- Docker-engine support multi-host networking out-of-the box. through overlay network driver.

- Overlay Network requires some pre-existing conditions before it can be created.
    1) Running Docker engine in swarm mode. you can create overlay network on manager node.
    2) Else A key-value store is require such as Consul.

- Widely used in production.
- Latest Docker swarm mode creates overlay network automatically.

Network in Compose: By default docker compose sets up a single network for services each container will join the default network and is reachable by other containers on that network. We can also provide network isolation using custom networks

Eg: We create 2 Networks CDN_1 & CDN2,
frontend connects to CDN1, Backend connects to both.
DB connects to CDN2

UNIT Test In Containers

- Unit tests should test some basic functionality of our docker app code, with no reliance on external services.
- Unit tests should run as quickly as possible so that developers can iterate much faster without being blocked by waiting for test results.
- Docker containers can spin up in seconds & can create a clean and isolated envt. which is great to run unit tests with.

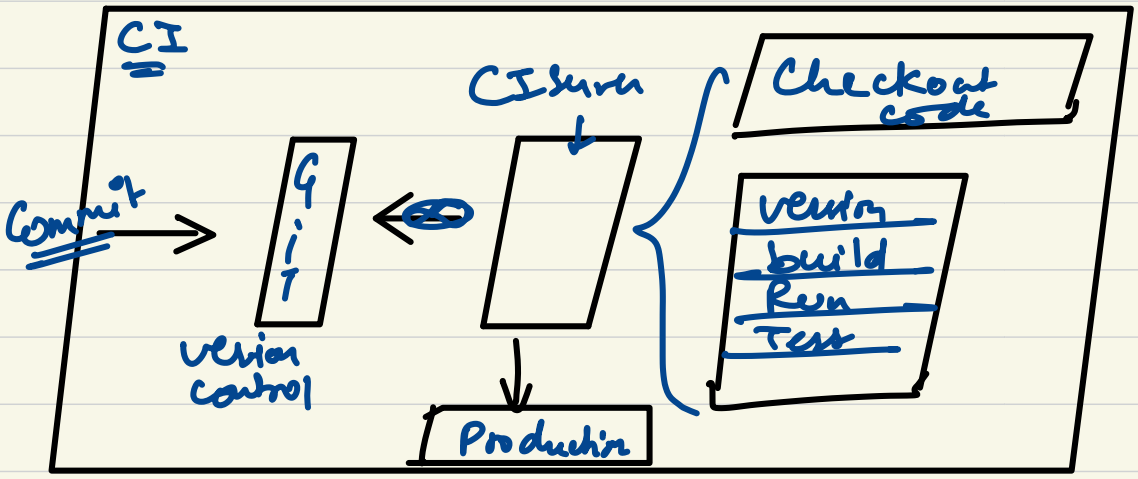
Pros: A single image is used through development, testing & prod. which greatly ensures the reliability of our tests.

Cons: It increases the size of image.

Fit docker into Continuous Integration (CI)

- CI is a S/w engineering practice in which isolated changes are immidiately tested and reported when they are added to large code base
- The goal of CI is to provide rapid feedback so that if a defect is introduced into the code base, it can be identified and corrected ASAP.

CI w/o docker



CI with docker