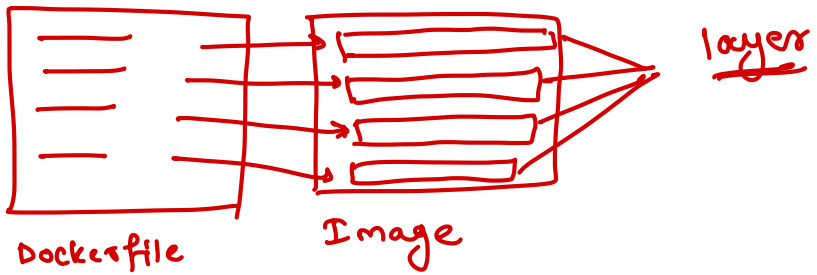

docker

Docker Images (Advanced)

Dockerfile

- The Dockerfile contains a series of instructions paired with arguments
- Each instruction should be in upper-case and be followed by an argument
- Instructions are processed from top to bottom
- Each instruction adds a new layer to the image and then commits the image
- Upon running, changes made by an instruction make it to the container



Dockerfile instructions

- FROM
- ENV
- RUN
- CMD
- EXPOSE
- WORKDIR
- ADD
- COPY
- LABEL
- MAINTAINER
- ENTRYPOINT

COPY vs ADD

COPY	ADD
Newer command	Older command
Can not copy file from urls	Can copy file from urls
Can not extract archived file to destination	Can extract contents of archived file to the destination
Can copy file from source to destination	

local machine

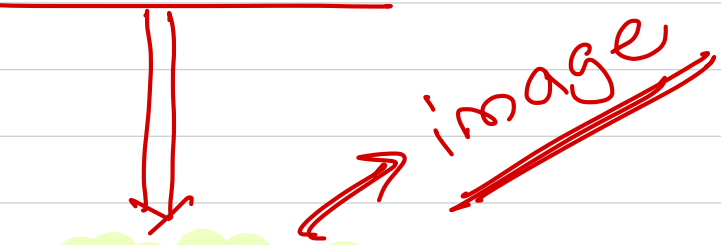
container

Multi-Stage Builds

- Used when the build system is complex
- It optimizes Dockerile and keeps them easy to read and maintain
- Helps keep size of images low
- Helps avoid having to maintain multiple Dockerfiles
- No intermediate images

Source code

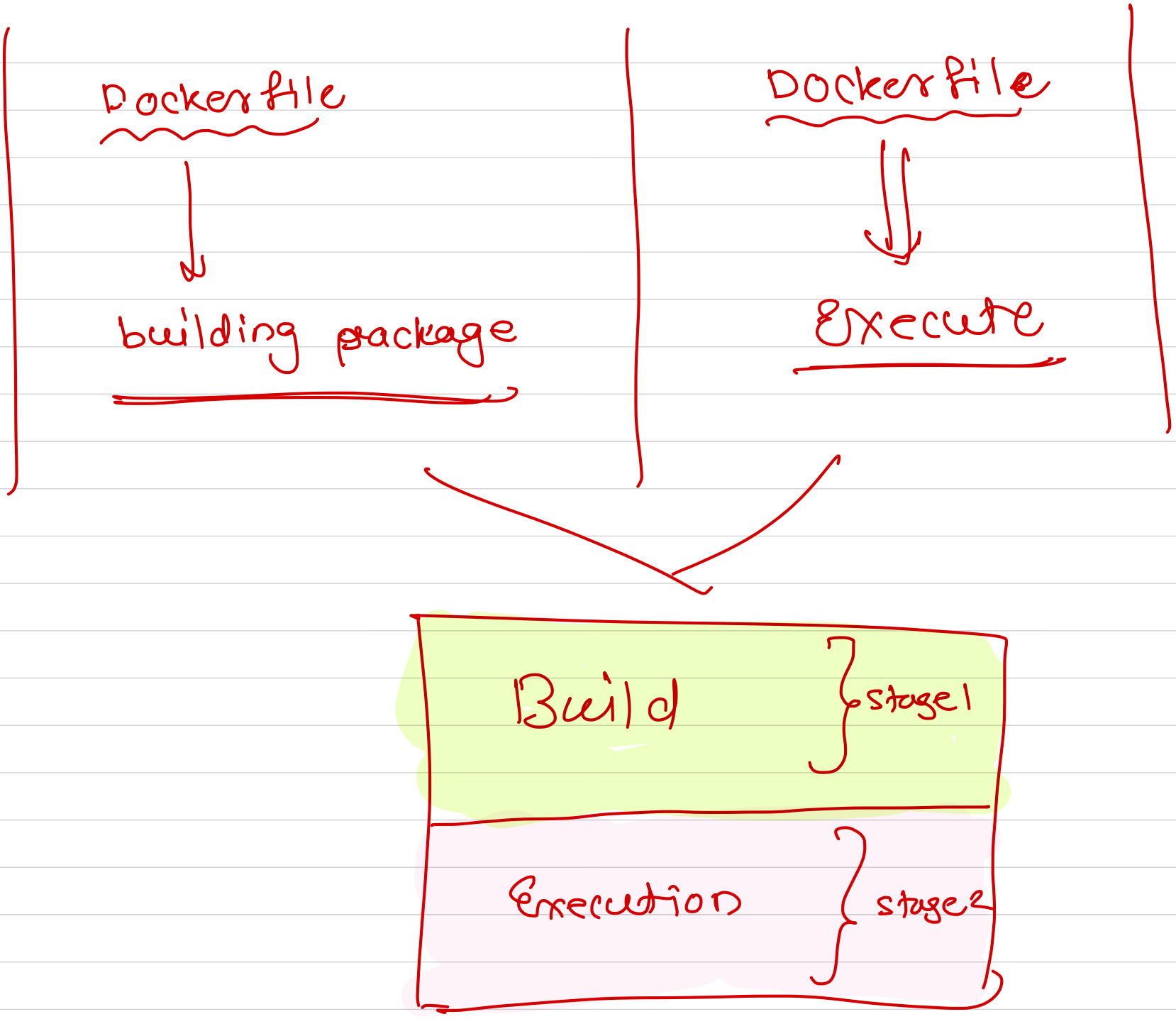
C++, Java etc



Build stage ⇒ converts source code into a package

android java app ⇒ .java + .xml + .jpg + .aar ⇒ .apk

Execution stage ⇒ container


Dockerfile
building package
Dockerfile
Execute
Build
stage1
Execution
stage2

MULTI STAGE Build

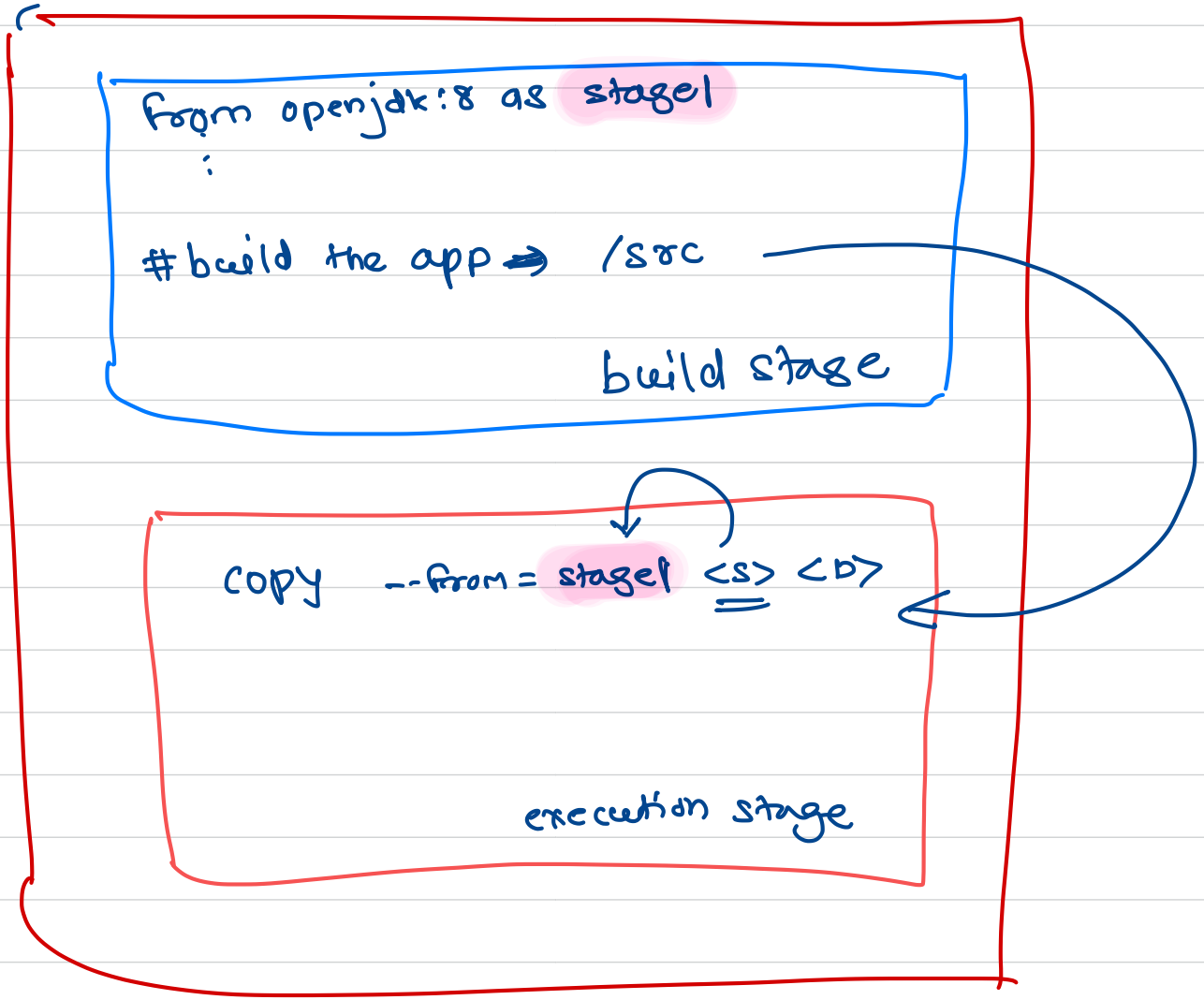


Image Save and Load

- In case of no internet environment, it becomes difficult to connect to the docker hub and download images
- Docker provides a facility to convert image to tar file and copy it to the restricted environment
- Command to create a tar file
  - **docker image save myimage -o myimage.tar**
- Command to load a tar file
  - **docker image load –i myimage.tar**

Import and Export Operations

- Docker also provides a feature to convert a container to an image by exporting all the layers
- This operation also includes writable layer
- Command to export container to an image
  - **docker export <container name> > myimage.tar**
- Command to import image
  - **docker image import myimage.tar newimage**

Sharing docker images

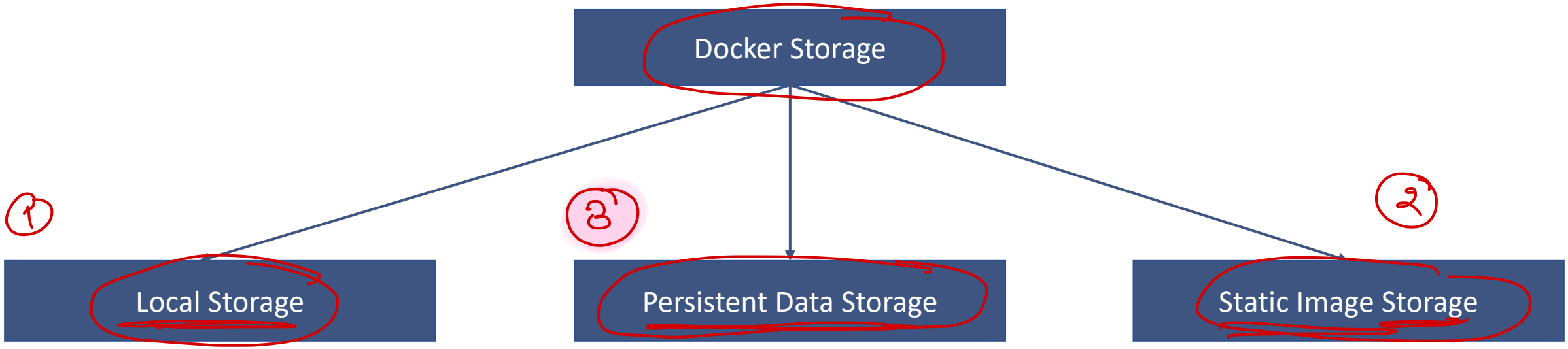
- Docker provides public docker hub to share the images
- Task
  - Create an image using Dockerfile
  - Push the image to dockerhub

Dockerfile – Best Practices

- Create slim / minimal images
- Find an official minimal image that exists → low sized image
- Only install necessary packages → JRE
- Maintain different images for different environments
  - Development – debug tools
  - Production – small and without debug tools
- Use multi-stage builds to create lean production images

Docker Volume

Overview



- Storage provided for docker image run
- Uses storage drivers to read FS layers from a container

- Saving data beyond the container lifecycle
- Data is stored outside the container

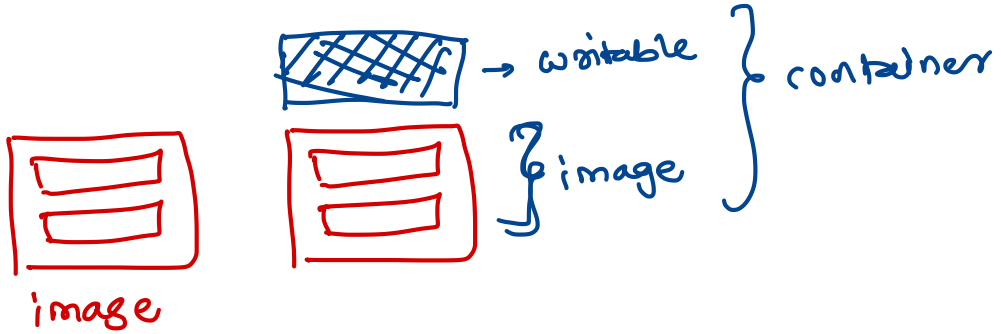
- Storing images in the docker registry
- Image stored in the registry will not run unless container is spawned

Storage drivers

- Docker supports several different storage drivers
- E.g.
  - Overlay2
    - preferred storage driver, for all currently supported Linux distributions,
    - Requires no extra configuration
  - Aufs
    - Preferred storage driver for Docker 18.06 and older, when running on Ubuntu 14.04 on kernel 3.13 which has no support for overlay2
  - Devicemapper
    - is supported, but requires direct-lvm for production environments
  - Btrfs and zfs
    - Used if they are the backing filesystem (snapshots)
  - Vfs
    - Intended for testing purposes
- Tasks
  - Get the docker disk usage
  - Get the current storage driver configured

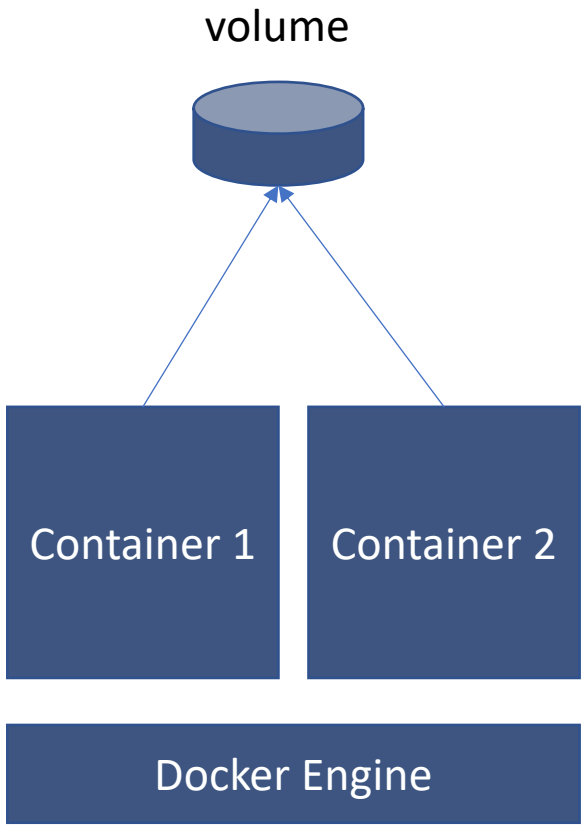
Local Storage

- Size taken by container
  - Size: data on the writable layer
  - Virtual Size: read-only image data + writable layer size
- Multiple containers share the image hence the image size will be shared
- Tasks
  - Create a container
  - Get the size information



Persistent Storage

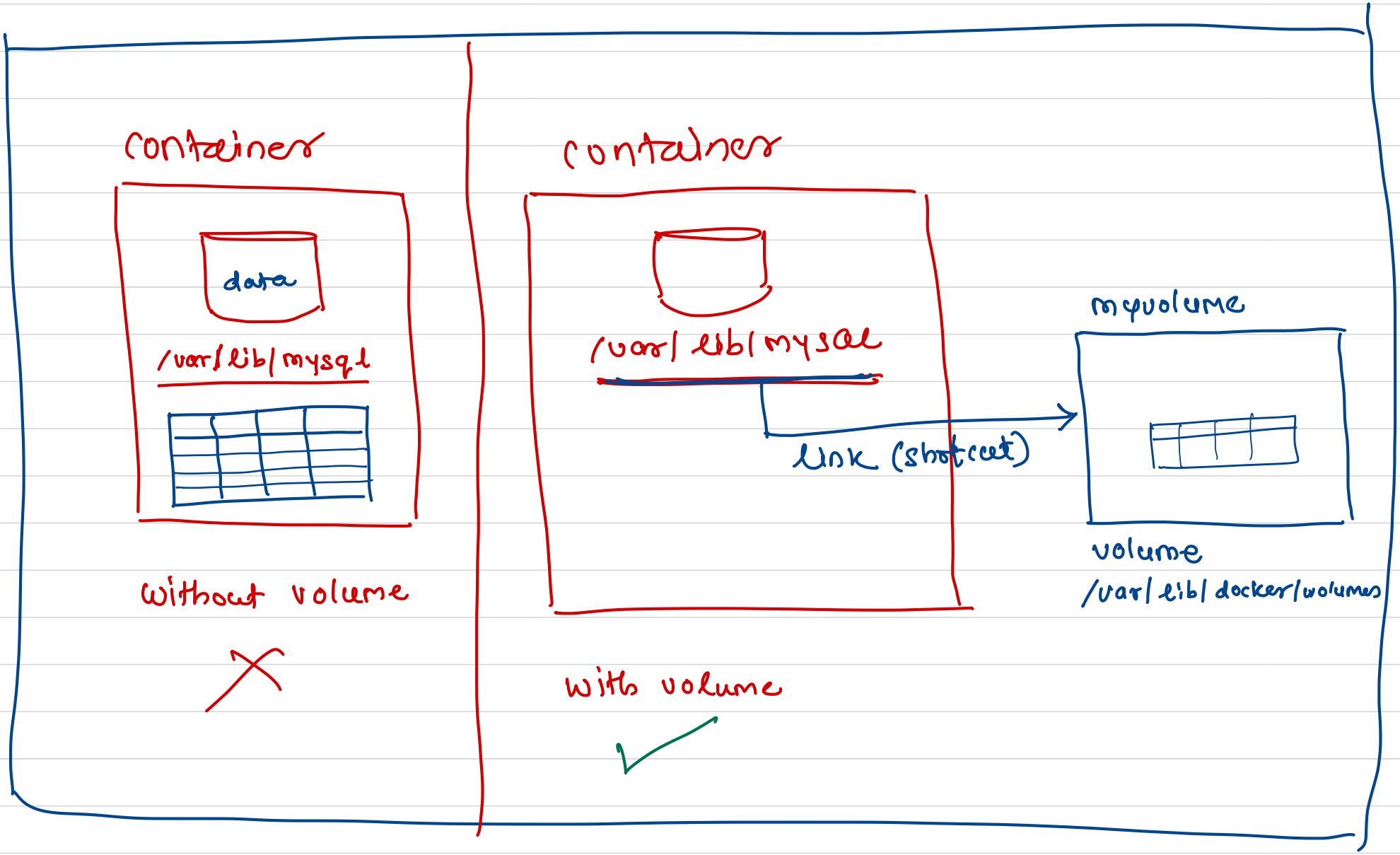
- Downsides of using local storage for containers
  - Data does not persist when container is removed
  - Writable layer is tightly coupled to the host machine
- Volume provides persistent storage
- Allows to share the data among containers
- Can be managed using the docker CLI commands
- NOTE: Volume does not increase the size of container using it



Persistent Storage

- Volumes
  - Stored in the docker managed FS of the host (/var/lib/docker/volumes)
  - Supports the use of Volume Drivers
- BindMounts
  - Stored anywhere in the host
  - E.g. you can mount a local directory with the container to share the contents
- Tmpfs Mounts
  - Temporary and stored in the host's memory
  - When the container stops, the tmpfs mount is removed
  - If the container is committed then tempfs is not saved
  - Available only with docker on linux

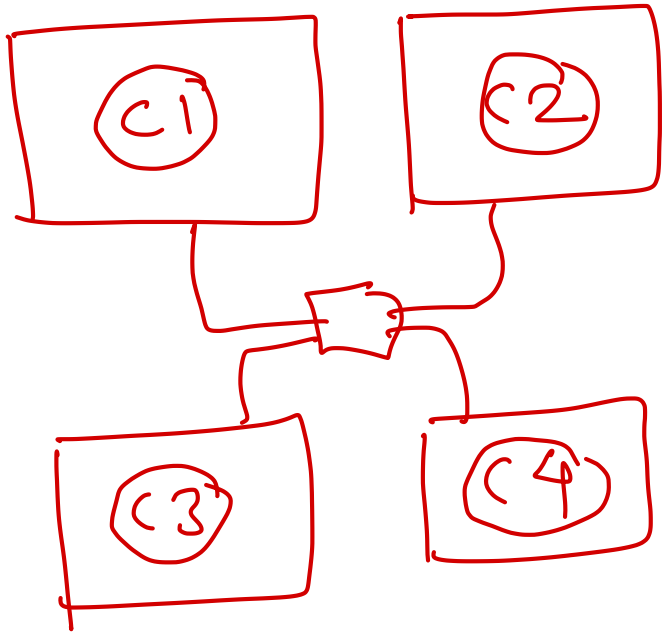
mysql

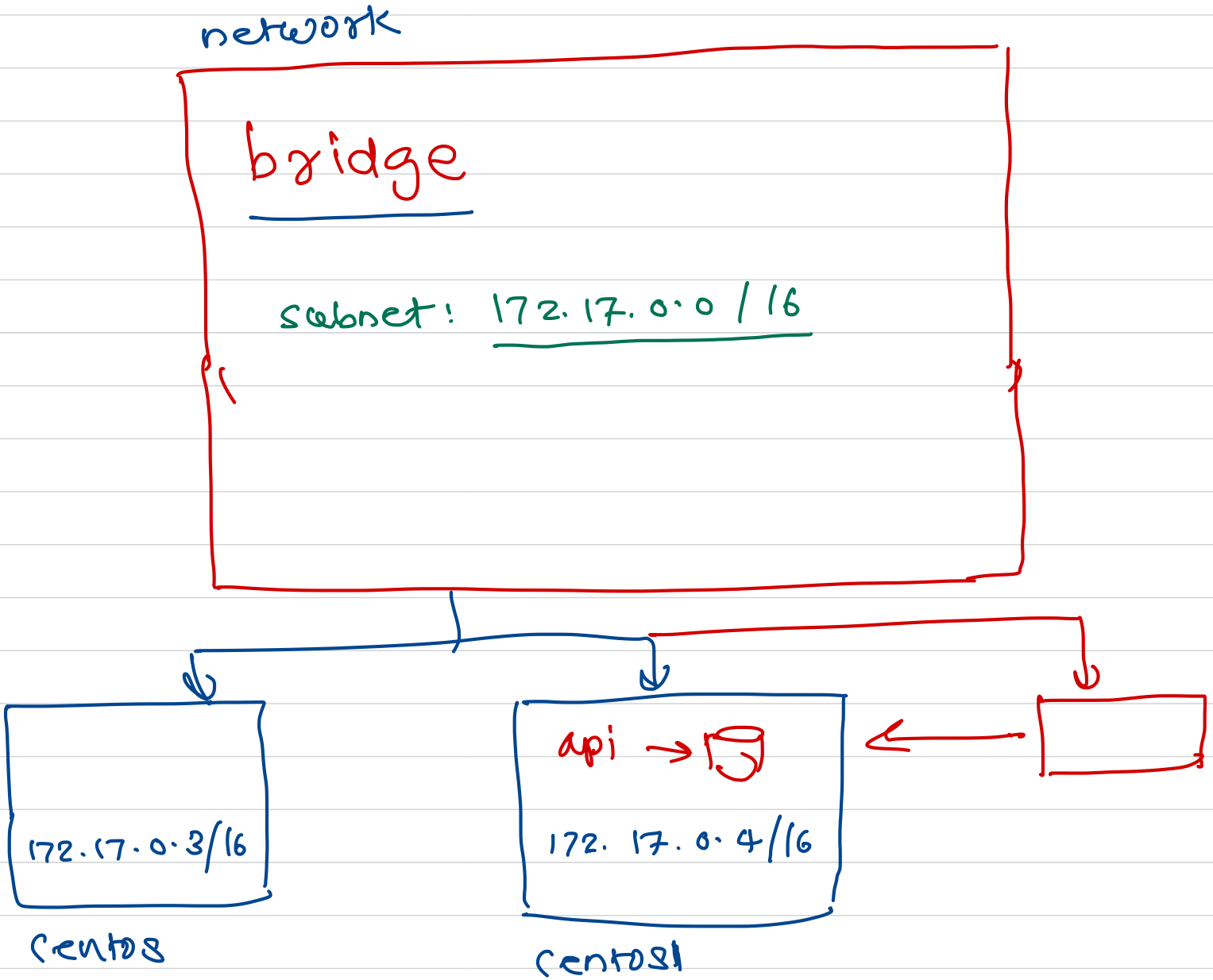

container
data
/var/lib/mysql
without volume
container
/var/lib/mysql
link (shortcut)
myvolume
volume
/var/lib/docker/volumes
with volume

Docker Network

Overview

- By default docker creates following networks on the host
  - Bridge
  - Host
  - None
- Task
  - Check the networks on the host machine
  - Get more information of any network




network
bridge
subnet: 172.17.0.0/16
172.17.0.3/16
centos
api
172.17.0.4/16
centos1

172.17.0.2/16 | 172.17.0.3/16 | 172.17.0.4/16

IP: 172.17.0.2 and 172.17.0.3 and 172.17.0.4

and

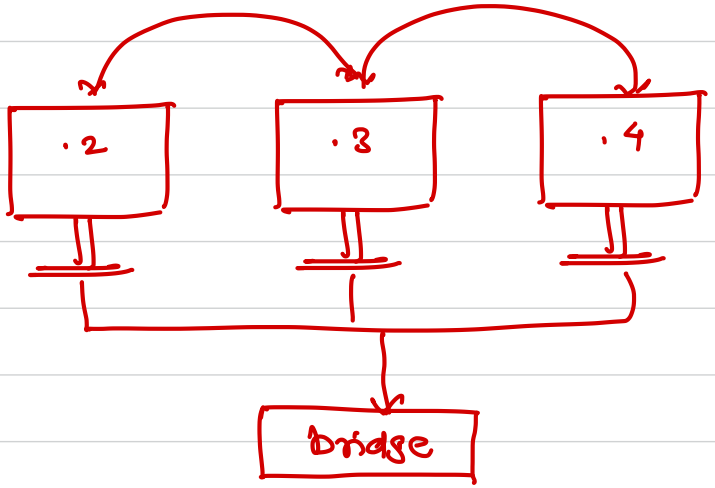
SM: 255.255.0.0 | 255.255.0.0 | 255.255.0.0

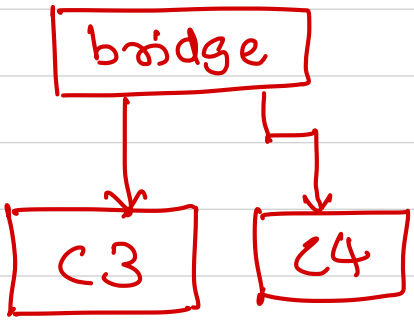
---

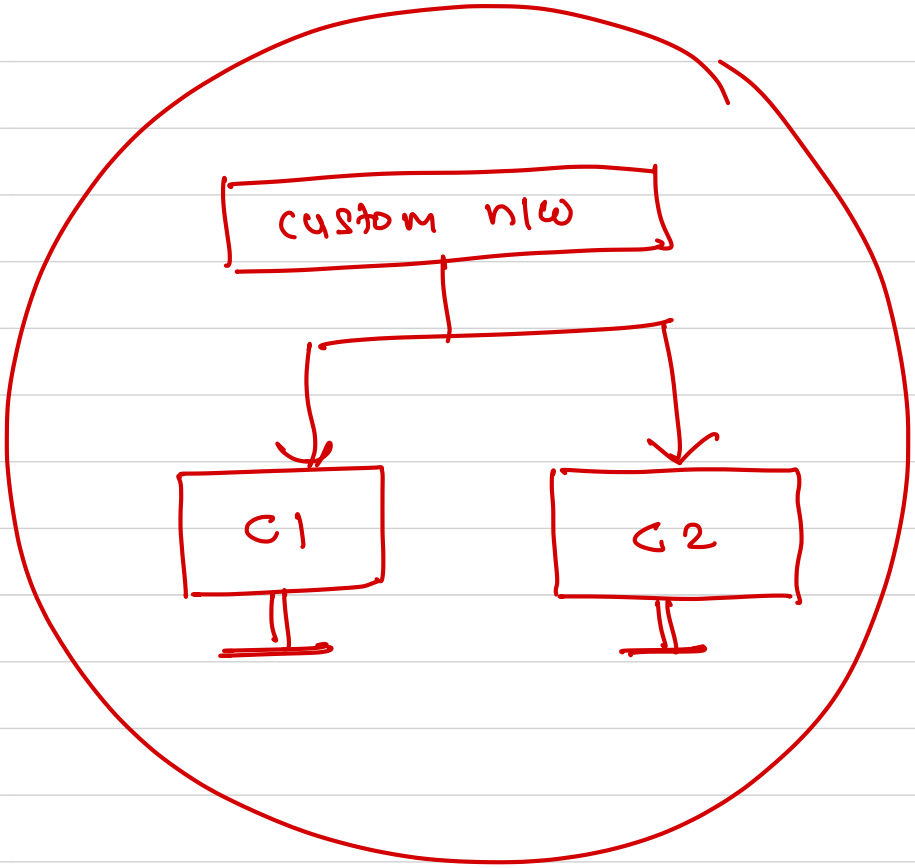
networkid: 172.17.0.0 | 172.17.0.0 | 172.17.0.0

.00000010
.00000000

.00000000
= .0

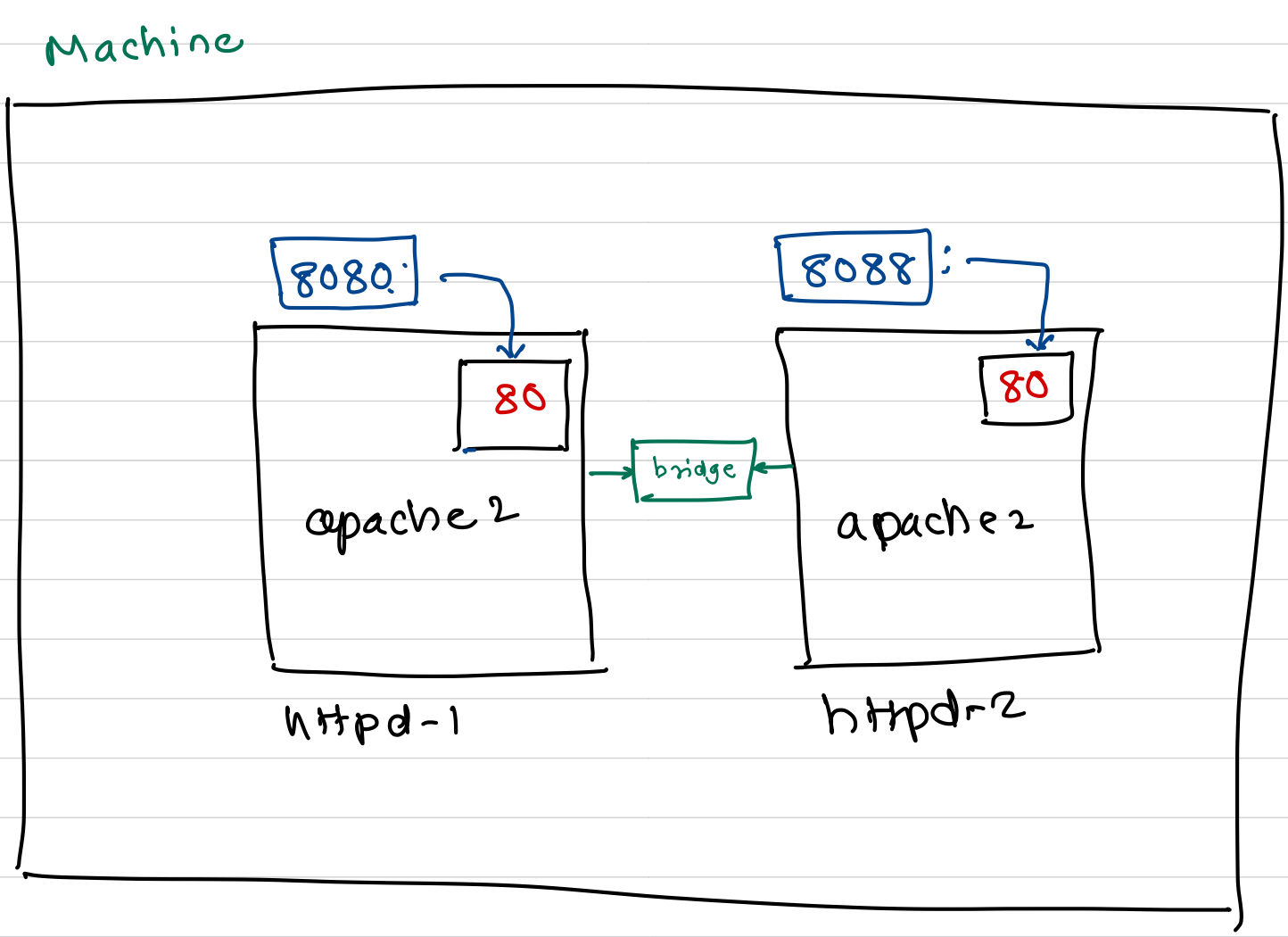



bridge
C3
C4


custom nlw
C1
C2

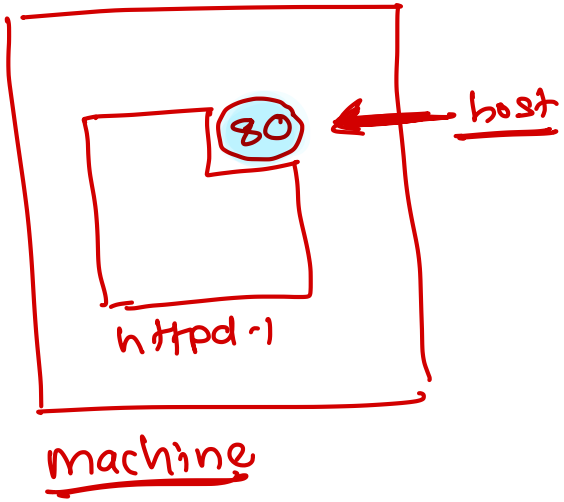
Bridge network

- Containers run on a separate network stacks, internal to docker host
- All of the containers share the external IP of the host machine using NAT
- Docker by default puts new container on bridge network
- Task
  - Get information about the bridge network
  - Run two containers on bridge network with same port published


Machine
8080
80
apache 2
httpd-1
bridge
8088
80
apache 2
httpd-2

Host network

- Containers behave just as any other process running in the docker host
- Host network adds the containers on the host's network stack
- There will be no isolation between the host machine and the container
- Does not perform any operation on incoming traffic (NAT)
- Task
  - Run a container on host network and verify the IP address
  - Run two containers on host network with same port published

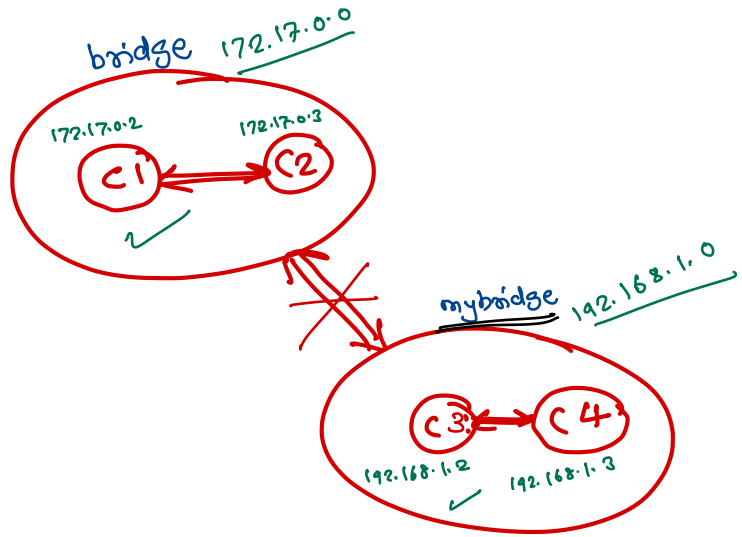


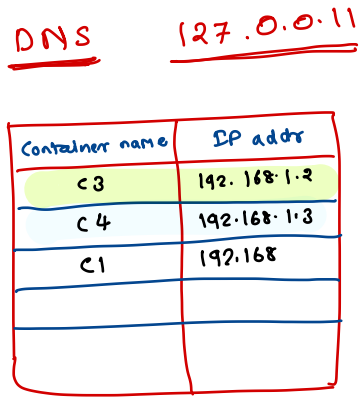
Modify network settings on container

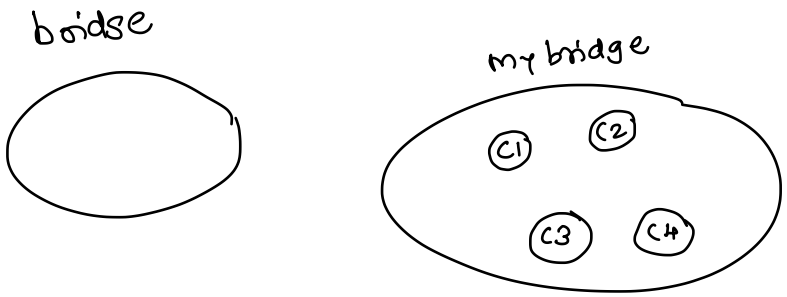
- Docker allows to modify the network settings without the need to restart the container
- Tasks
  - Start a container on none network
  - Disconnect the none network
  - Connect to bridge network

Custom network

- Docker network command can be used to create custom networks
- To create a custom network we have to use a driver
- If driver is not mentioned then docker uses bridge by default
- We can create as many networks as we need
- Tasks
  - Create a custom bridge network
  - Check the network interfaces on the docker host
  - Run a container using the newly created network







Remove the network

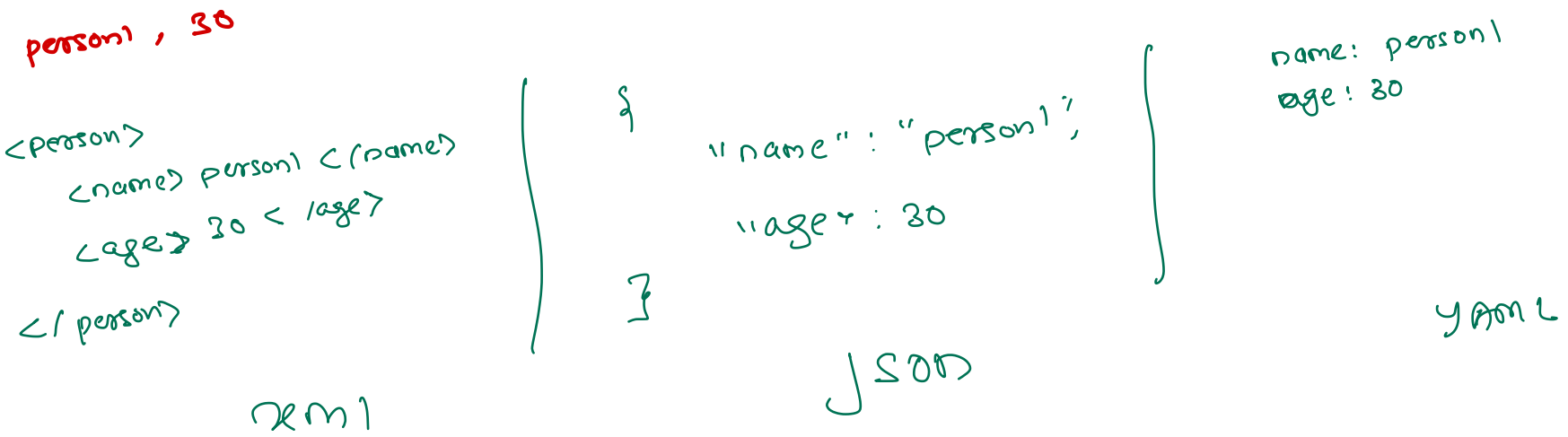
- Default networks can not be removed
- Active networks can not be removed
- Tasks
  - Create a custom network
  - Remove that custom network

- Prune command can be used to remove all unused networks

YAML

Overview

- YAML is the abbreviated form of “YAML Ain’t markup language”
- It is a data serialization language which is designed to be human -friendly and works well with other programming languages for everyday tasks
- It is useful to manage data and includes Unicode printable characters
- Easily readable by humans



Basics

- YAML is case sensitive
- The files should have **.yaml** or **.yml** as the extension
- YAML does not allow the use of tabs while creating YAML files; spaces are allowed instead
- Comment starts with #
- Comments must be separated from other tokens by whitespaces.

Scalars

- Scalars in YAML are written in block format using a literal type
- E.g.
  - Integer
    - 20
    - 40
  - String
    - Steve
    - “Jobs”
    - ‘USA’
  - Float
    - 4.5
    - 1.23015e+3

Mapping

- Represents key-value pair
- The value can be identified by using unique key
- Key and value are separated by using colon (:)
- E.g.
  - name: person1
  - address: “India”
  - phone: +9145434345
  - age: 40
  - hobbies:
  - - reading
  - - playing

Sequence

- Represents list of values
- Must be written on separate lines using dash and space
- Please note that space after dash is mandatory
- E.g.
  - # pet animals
  - - cat
  - - dog

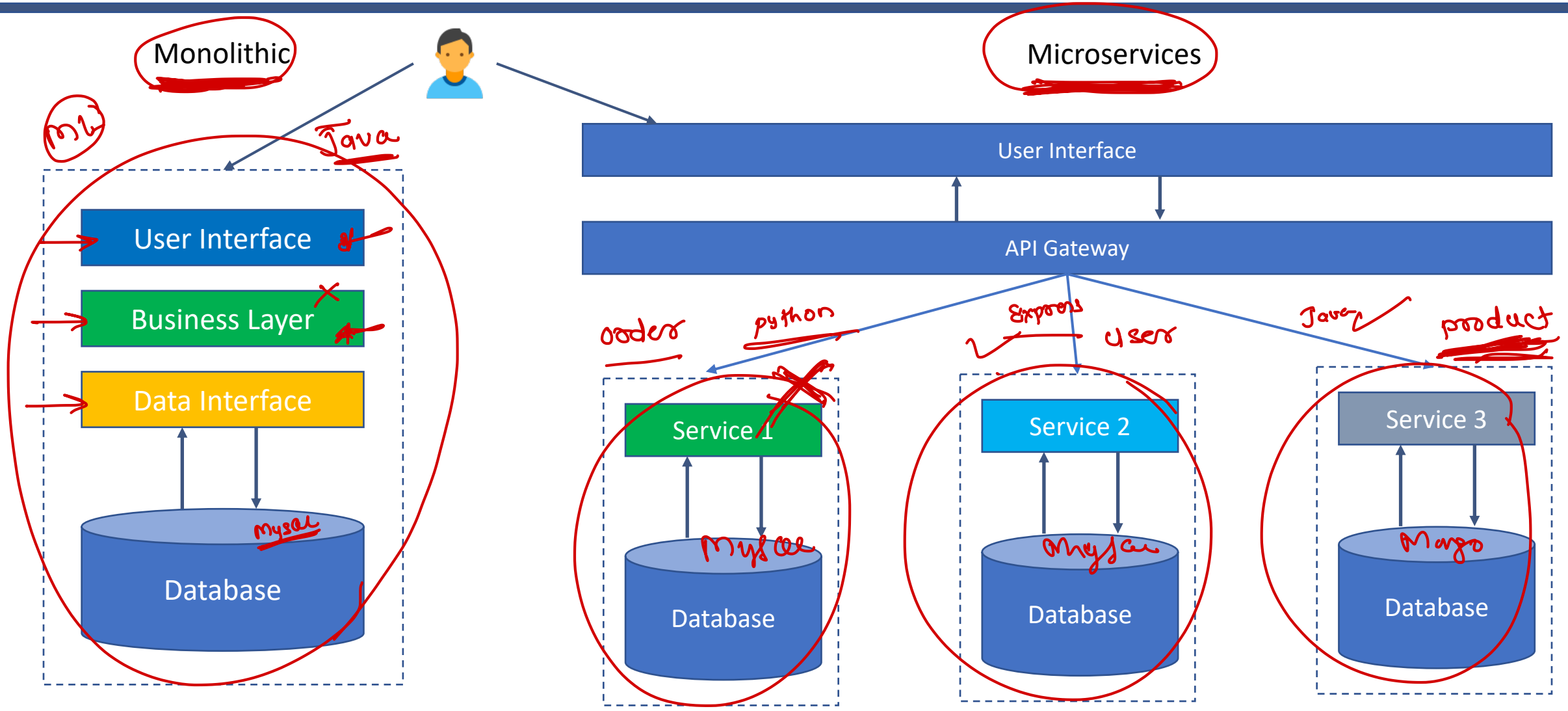
  - # programming languages
  - - C
  - - C++
  - - Java

Sequence

- Sequence may contains complex objects
- E.g.
  - products:
    - - title: product 1
    - price: 100
    - description: good product
    - - title: product 2
    - price: 300
    - description: useful product

Docker Compose

Monolithic vs Microservice



Microservice

- Distinctive method of developing software systems that tries to focus on building single-function modules with well-defined interfaces and operations
- Is an architectural style that structures an application as a collection of services that are
  - Highly maintainable and testable
  - Loosely coupled
  - Independently deployable
  - Organized around business capabilities

Benefits

- Easier to Build and Maintain Apps
- Improved Productivity and Speed
- Code for different services can be written in different languages
- Services can be deployed and then redeployed independently without compromising the integrity of an application
- Better fault isolation; if one microservice fails, the others will continue to work
- Easy integration and automatic deployment; using tools like Jenkins
- The microservice architecture enables continuous delivery.
- Easy to understand since they represent a small piece of functionality, and easy to modify for developers thus they can help a new team member become productive quickly
- Scalability and reusability, as well as efficiency
- Components can be spread across multiple servers or even multiple data centers
- Work very well with containers, such as Docker

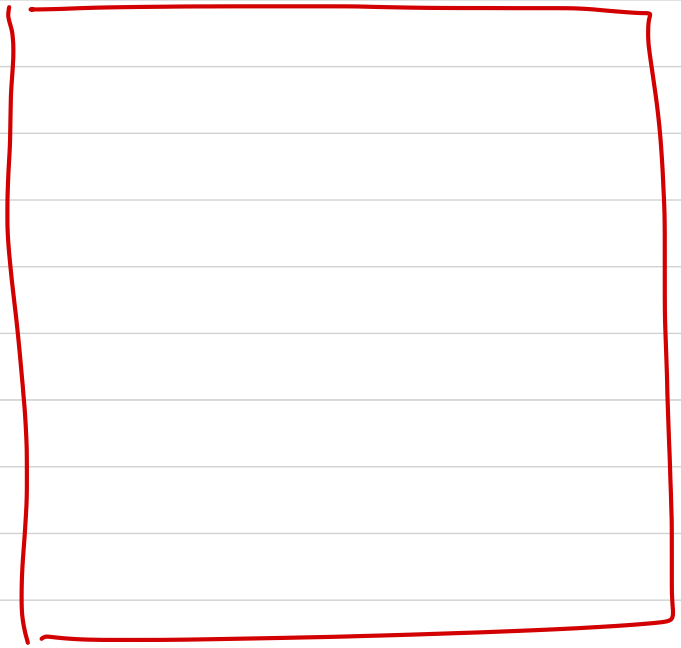
What is Docker compose ?

- Docker compose is a tool for defining and running multi-container applications
- One microservice represents on containerized application
- It is difficult to manage every container individually
- With compose, you use a YAML file to configure your application services
- YAML contains the configuration of all the services
- With a single command, you create and start all the services from your configuration
- Docker compose relies on Docker Engine

Docker compose workflow

- Build the container image as part of compose or use a pre-created image
- Define the services that make up the application
- Start the entire application with a single command
- Tasks
  - Containerize the application
  - Create docker-compose file and deploy the application

/order



Node - Express

/ml

python - flask

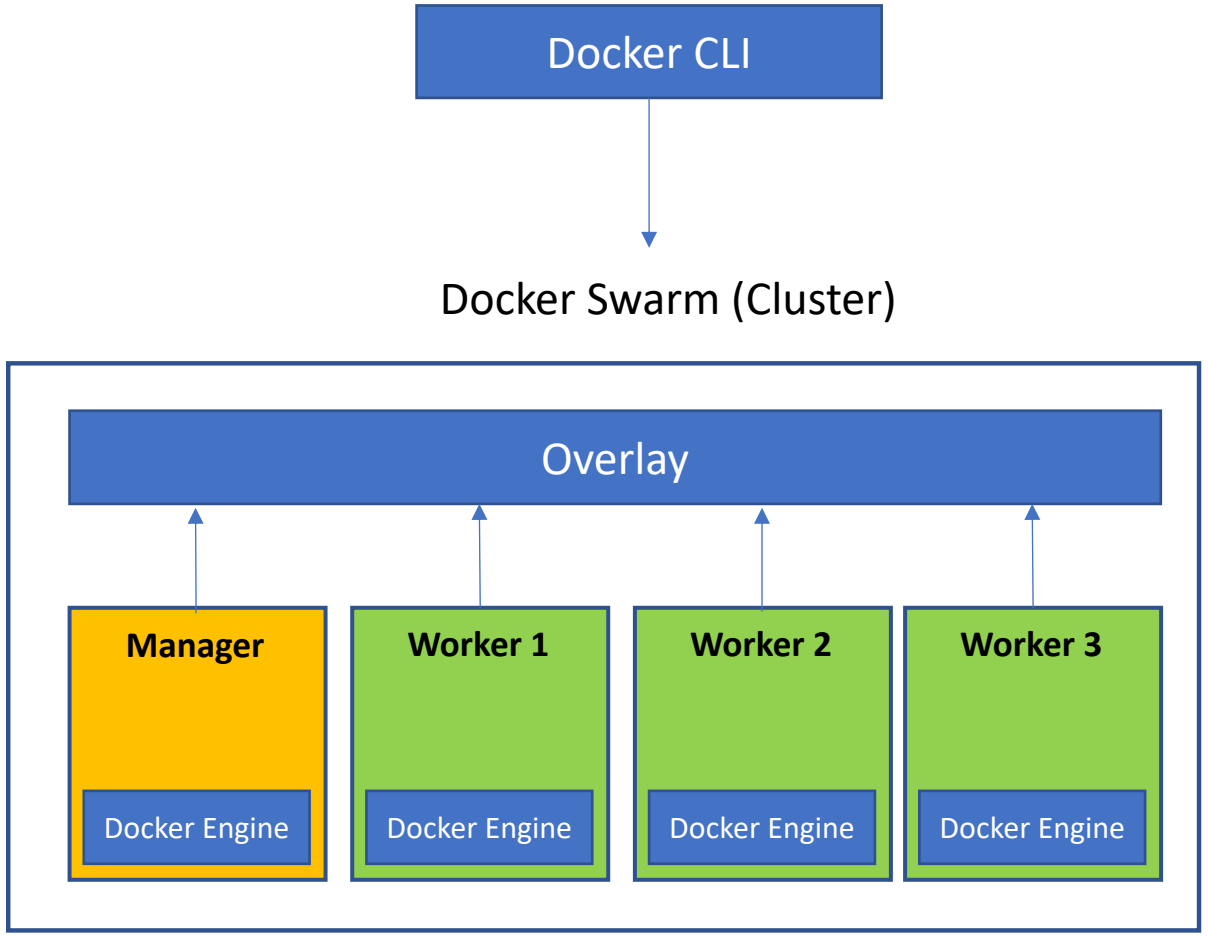
Docker Swarm

Overview

- Container orchestration solution provided by Docker
- It is multi-container and machine machine (node) setup
- A swarm is a group of machines (known as nodes) that are running Docker engine and joined together into a cluster
- A node can be physical or virual
- Docker CLI to create and manage a swarm
- Cluster management is integrated in the Docker engine
- It is secure by default
- It is built using Swarmkit

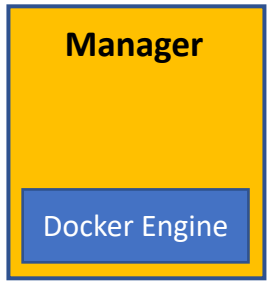
Create a Swarm

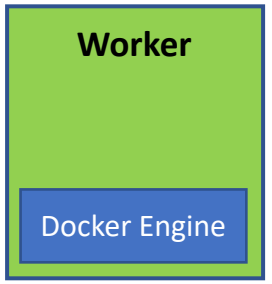
- A swarm consists of multiple docker hosts (nodes)
- There are two types of nodes in swarm
  - Manager node
    - The controller
    - Handles the cluster
    - Management and orchestration functions
    - By default manager acts as a worker
  - Worker node
    - The nodes where the containers run
    - These nodes run swarm services



Create a Swarm and add workers to swarm

- In our swarm we are going to use 3 nodes (one manager and two workers)
- Every node must have Docker Engine 1.12 or newer installed
- Following ports are used in node communication
  - TCP port 2377 is used for cluster management communication
  - TCP and UDP port 7946 is used for node communication
  - UDP port 4789 is used for overlay network traffic





docker swarm init --advertise-addr <ip>

docker swarm join --token <token>

Overlay Network

- It is a computer network built on top of another network
- Sits on top of the host-specific networks and allows container, connected to it, to communicate securely
- When you initialize a swarm or join a host to swarm, two networks are created
  - An overlay network called as ingress network
  - A bridge network called as docker_gwbridge
- Ingress network facilitates load balancing among services nodes
- Docker_gwbridge is a bridge network that connect overlay networks to individual docker daemon's physical network

Swarm nodes

- A node is a machine which has docker engine running
- A node can be physical or virtual
- Tasks
  - List the nodes in the swarm
  - Get information about a node

Swarm nodes

- Promote a node
  - A worker node can be promoted to a manager node
- Demote a node
  - A manager node can be demoted to a worker node
- Tasks
  - Get the list of nodes
  - Promote a worker to manger
  - Demote a manager to worker

Service

- Definition of tasks to execute on Manager or Worker nodes
- Declarative Model for Services
- Scaling
- Desired state reconciliation
- Service discovery
- Rolling updates
- Load balancing
- Internal DNS component

Demo

- Tasks
  - Create a new service
  - List the services
  - Inspect a service
  - List all the tasks started by service
  - Scale a service
  - Update a service
  - Remove a service

Service mode

- Replicated
  - We specify the number of required identical tasks
  - Swarm decides on which node the task can run
  - By default service starts in replicated mode
- Global
  - Guarantees to run the one task on each node
  - Can be used for antirust scanners, logging or monitoring agents
- Tasks
  - Create a global service

Publishing port

- Port can be published in order to make the application accessible outside
- There are two ways you can publish the port
  - Swarm mode (routing mesh)
    - Application can be accessed using any of the nodes
  - Host mode
    - Application can be accessed only on those nodes where the service is running
- Tasks
  - Create a service in routing mesh mode
  - Create a service in host mode

Control service placement

- The service will go in pending state until a node is available with the required resources
- Constraint flag can be used to restrict based on placement requirements
- Tasks
  - Add label to a node
  - Start service only on the respective node

Stack

- Multiple services can be started and orchestrated together with a stack
- Stack uses a YAML file configuration to group the services
- Task
  - Create a configuration YAML file
  - Create a stack
  - List the services created using stack
  - Remove stack