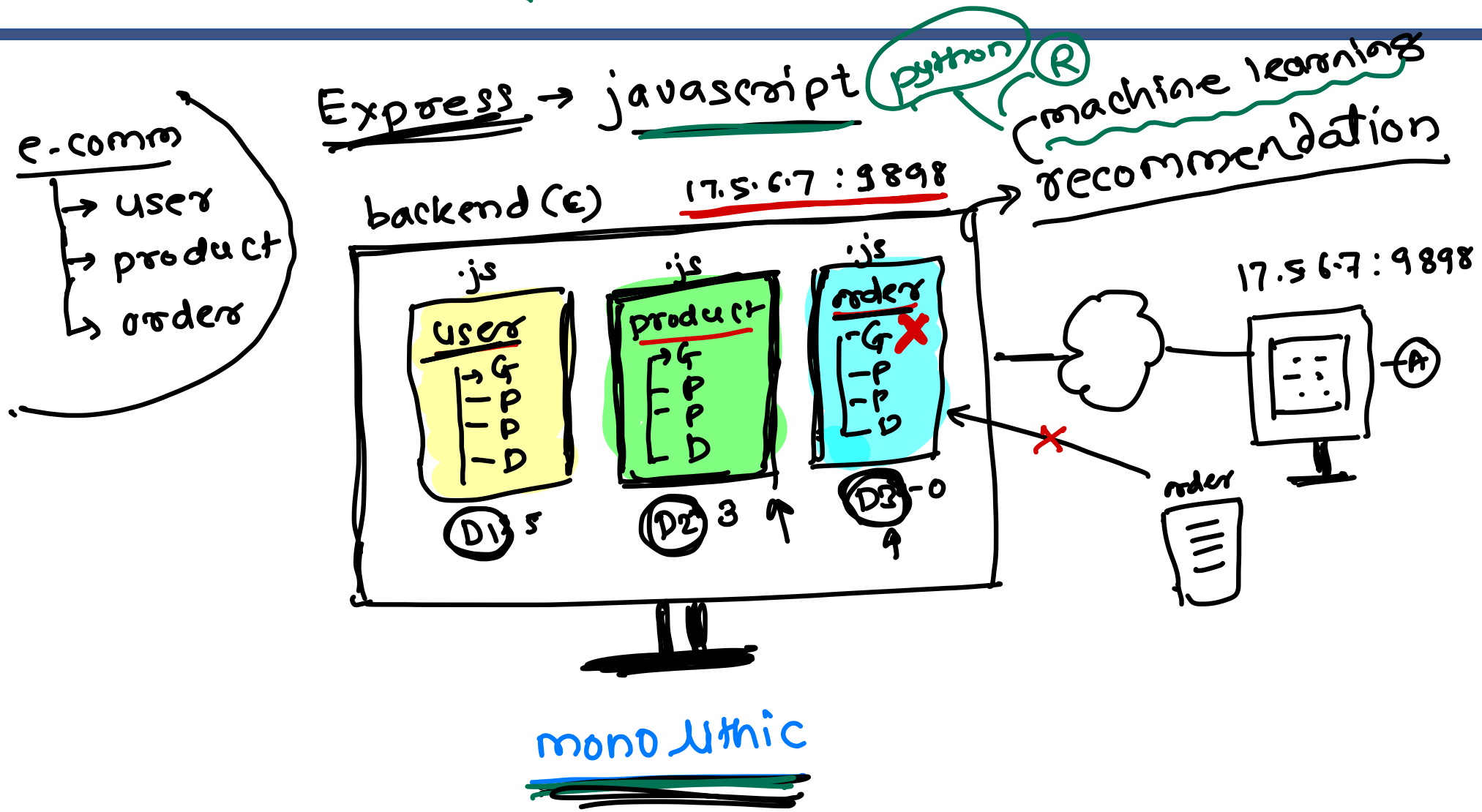
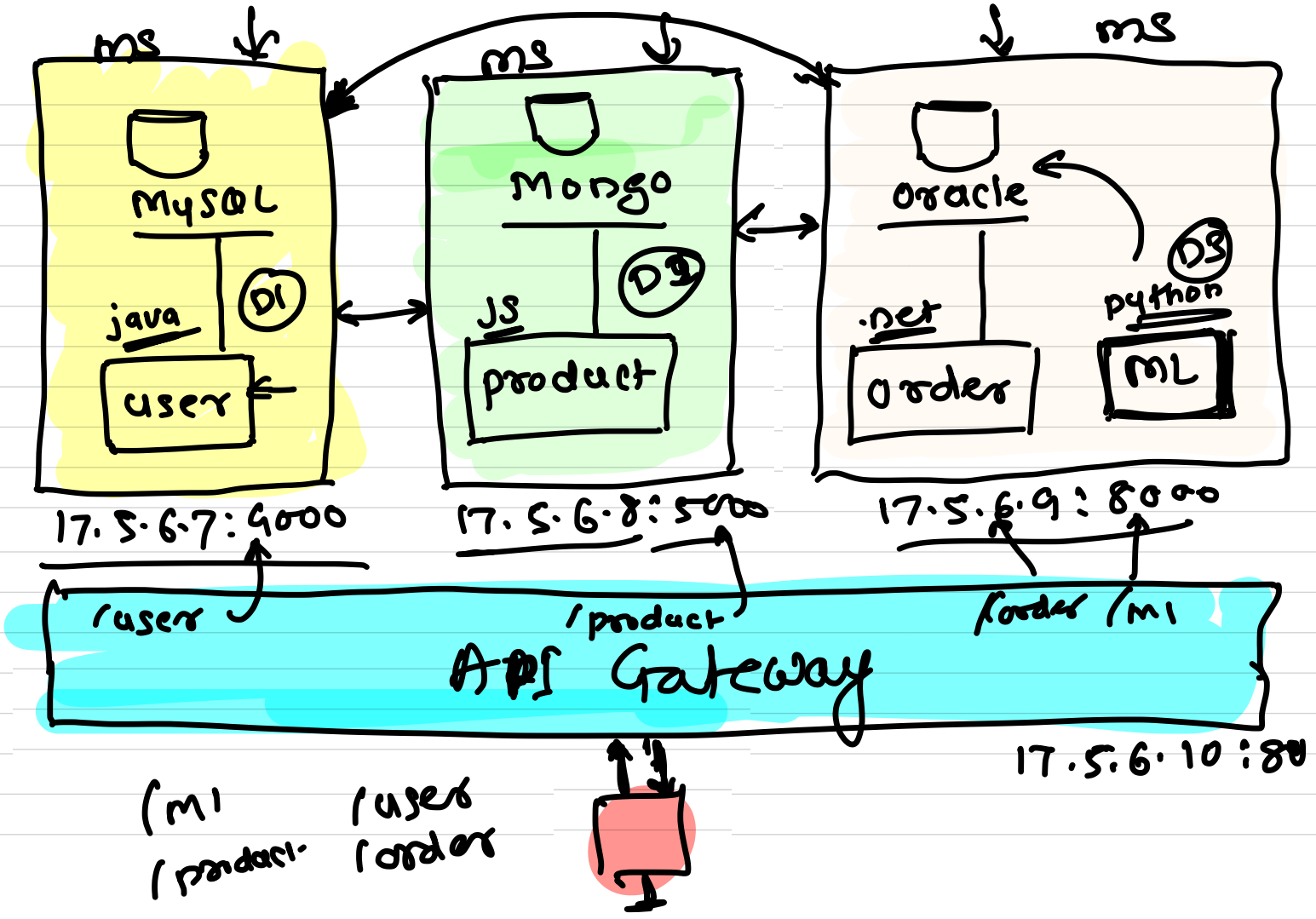
Microservices

Monolithic vs Microservices



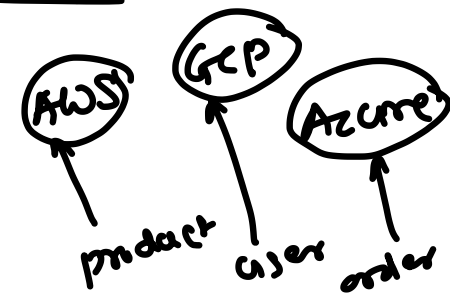

ms
MySQL
java
DI
user
17.5.6.7:9000
ms
Mongo
DB
JS
product
17.5.6.8:5000
ms
oracle
.net
order
DB
python
ML
17.5.6.9:8000
/user
/product
/order
/ML
API Gateway
17.5.6.10:80
/ML
/product
/user
/order

Overview

- 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 → small
- 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

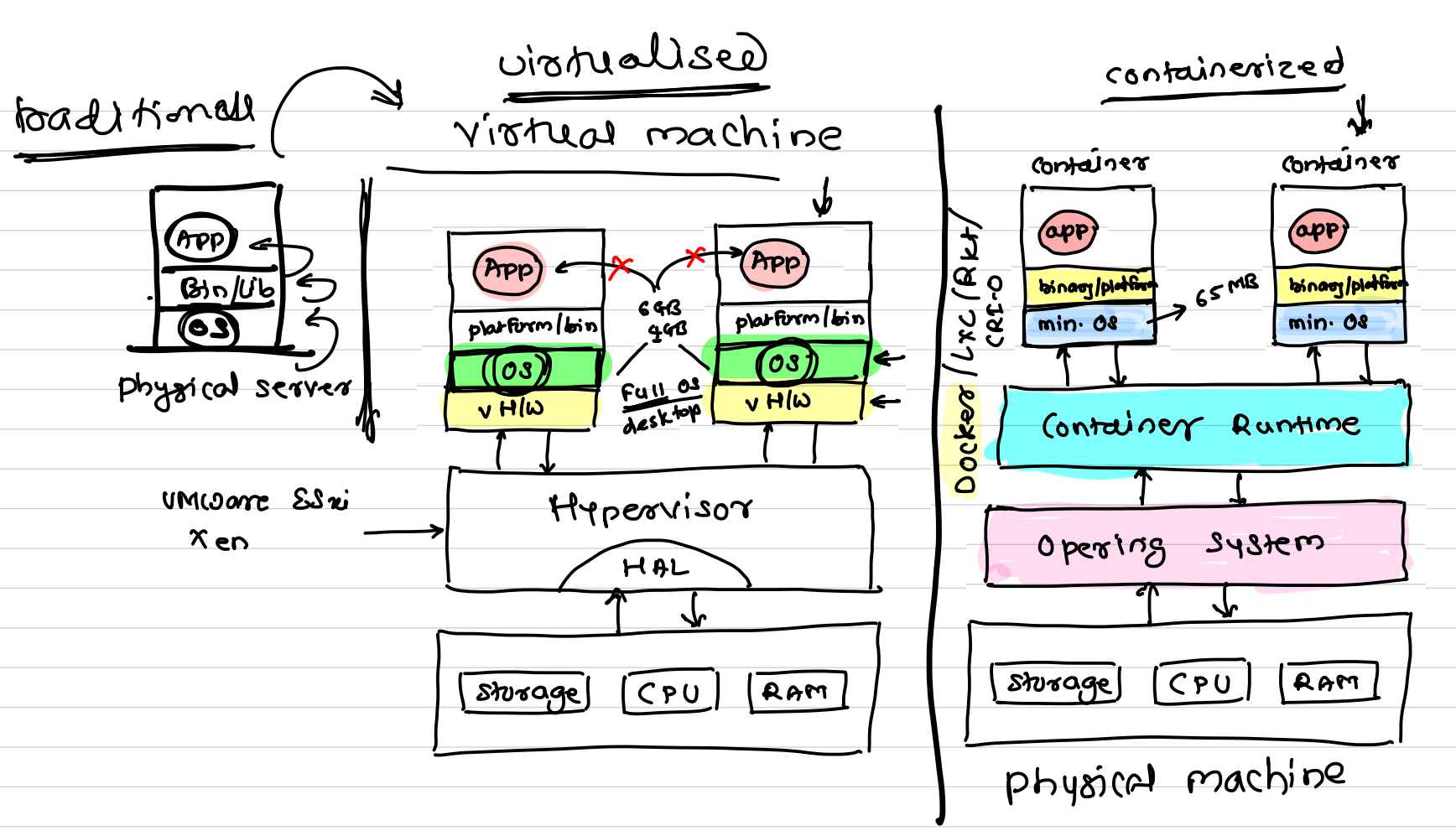


Containerization

traditional

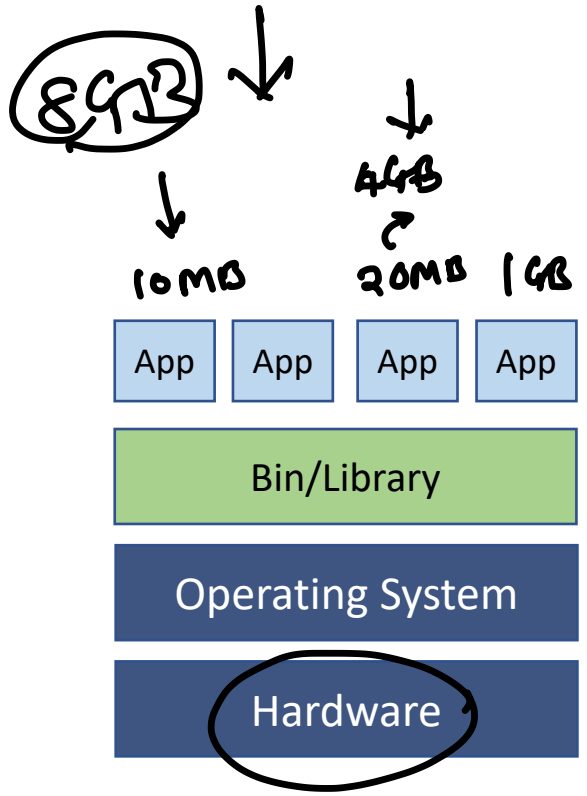
virtualised

containerized



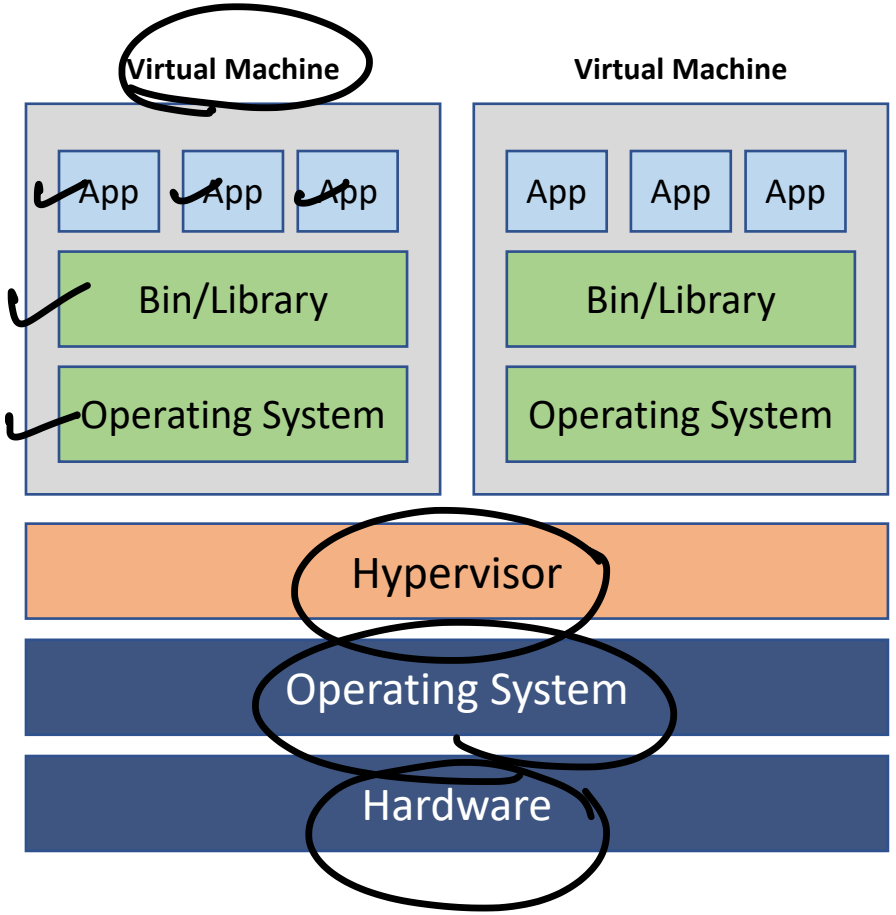
Traditional Deployment

- Early on, organizations ran applications on physical servers
- There was no way to define resource boundaries for applications in a physical server, and this caused resource allocation issues
- For example, if multiple applications run on a physical server, there can be instances where one application would take up most of the resources, and as a result, the other applications would underperform
- A solution for this would be to run each application on a different physical server
- But this did not scale as resources were underutilized, and it was expensive for organizations to maintain many physical servers



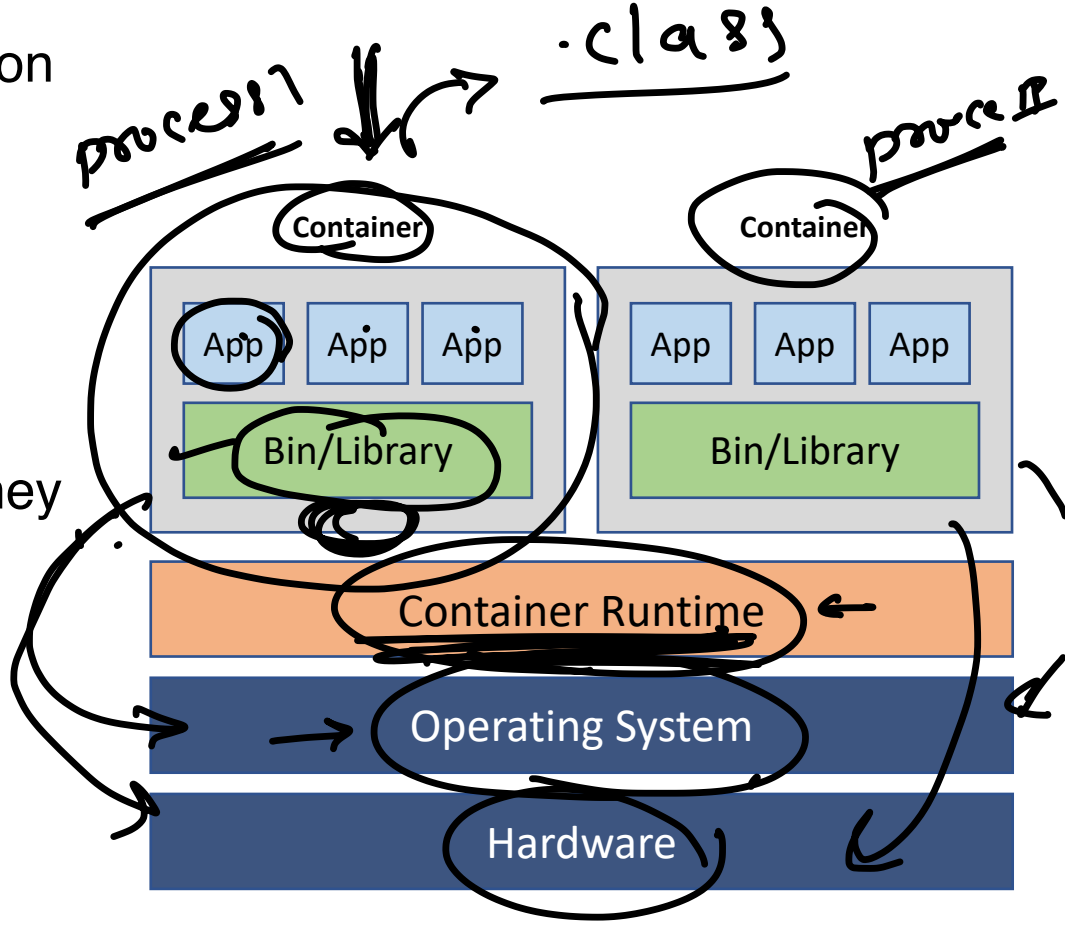
Virtualized Deployment

- It allows you to run multiple Virtual Machines (VMs) on a single physical server's CPU
- Virtualization allows applications to be isolated between VMs and provides a level of security as the information of one application cannot be freely accessed by another application
- Virtualization allows better utilization of resources in a physical server and allows better scalability because
  - an application can be added or updated easily
  - reduces hardware costs
- With virtualization you can present a set of physical resources as a cluster of disposable virtual machines
- Each VM is a full machine running all the components, including its own operating system, on top of the virtualized hardware



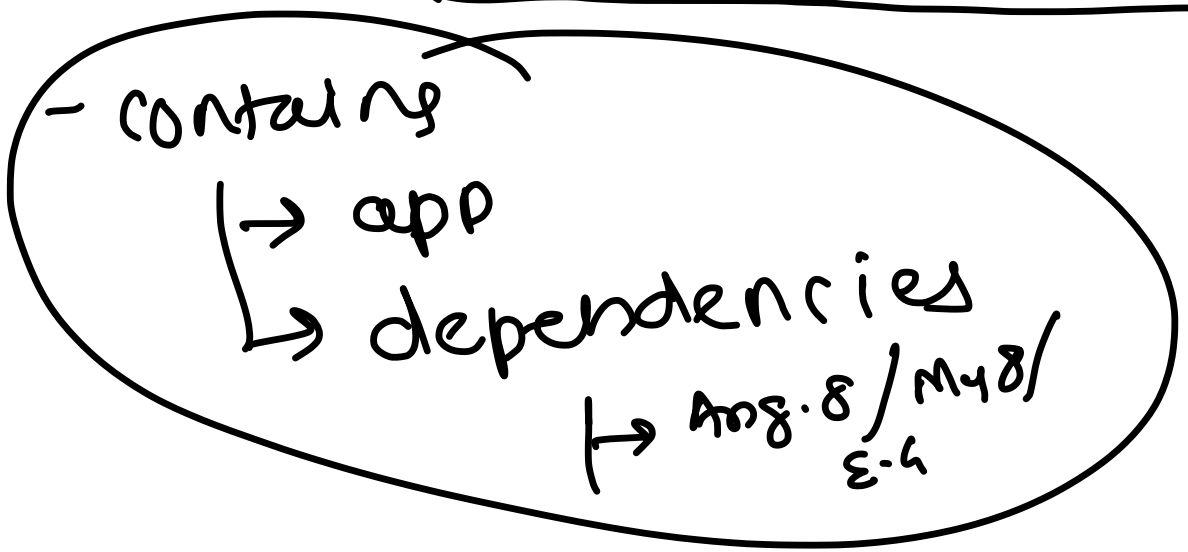
Container deployment

- Containers are similar to VMs, but they have relaxed isolation properties to share the Operating System (OS) among the applications
- Therefore, containers are considered lightweight
- Similar to a VM, a container has its own filesystem, CPU, memory, process space, and more
- As they are decoupled from the underlying infrastructure, they are portable across clouds and OS distributions



Containerization

- Lightweight alternative or companion to a virtual machine
- Involves encapsulating or packaging up software code and all its dependencies so that it can run uniformly and consistently on any infrastructure
- Allows developers to create and deploy applications faster and more securely



Containers vs Virtual machines

Virtual Machine	Container
Hardware level virtualization	OS virtualization
Heavyweight (bigger in size) — 6GB	Lightweight (smaller in size) - 65 MB
Slow provisioning	Real-time and fast provisioning
Limited Performance	Native performance
Fully isolated	Process-level isolation
More secure	Less secure
Each VM has separate OS	Each container can share OS resources
Boots in minutes	Boots in seconds
Pre-configured VMs are difficult to find and manage	Pre-built containers are readily available
Can be easily moved to new OS	Containers are destroyed and recreated → immutable
Creating VM takes longer time	Containers can be created in seconds

Advantages

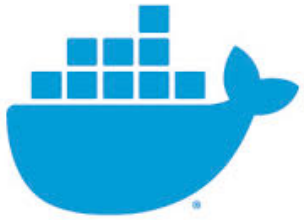
- **Portability**
  - A container creates an executable package of software that is abstracted away from (not tied to or dependent upon) the host operating system, and hence, is portable and able to run uniformly and consistently across any platform or cloud
- **Agility**
  - The open source Docker Engine for running containers started the industry standard for containers with simple developer tools and a universal packaging approach that works on all operating systems
- **Speed**
  - Containers are often referred to as “lightweight,”
  - Meaning they share the machine’s operating system (OS) kernel and are not bogged down with this extra overhead
- **Fault isolation**
  - Each containerized application is isolated and operates independently of others
  - The failure of one container does not affect the continued operation of any other containers

Advantages

- **Efficiency**
  - Software running in containerized environments shares the machine's OS kernel, and application layers within a container can be shared across containers
  - Thus, containers are inherently smaller in capacity than a VM and require less start-up time
- **Ease of management**
  - A container orchestration platform automates the installation, scaling, and management of containerized workloads and services
  - Container orchestration platforms can ease management tasks such as scaling containerized apps, rolling out new versions of apps, and providing monitoring, logging and debugging, among other functions
- **Security**
  - The isolation of applications as containers inherently prevents the invasion of malicious code from affecting other containers or the host system
  - Additionally, security permissions can be defined to automatically block unwanted components from entering containers or limit communications with unnecessary resources

Popular container platforms

- Linux Containers (LXC)
- Docker
- Windows Server
- CoreOS rkt



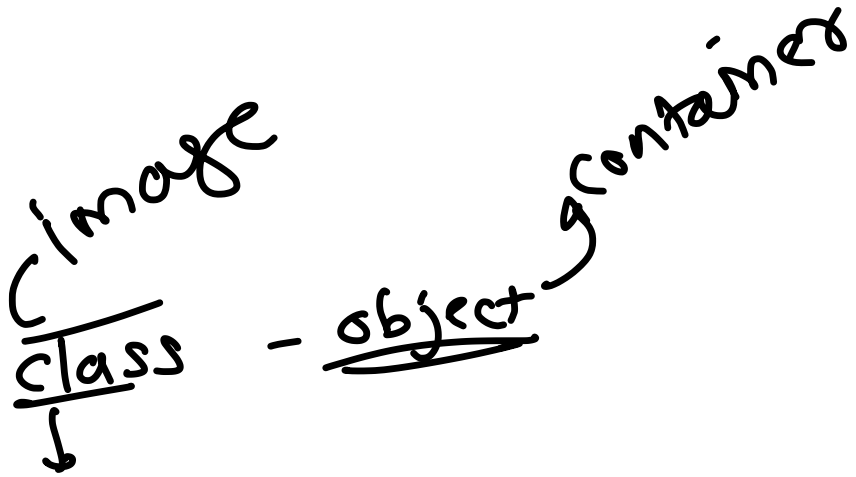
Docker

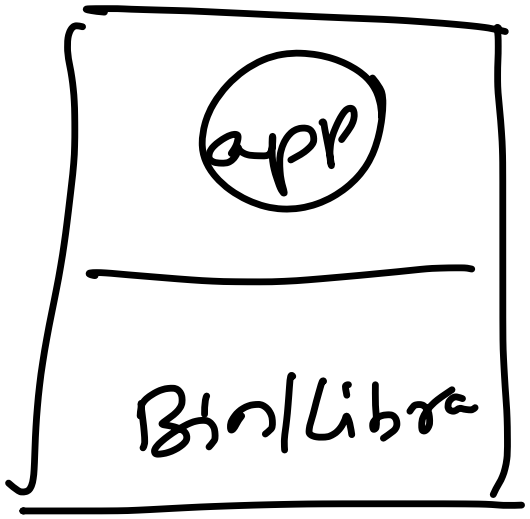
Overview

- Docker is software to create, manage and orchestrate containers
- It supports all major Operating Systems
- Docker Inc, started by Solomon Hykes, is behind the docker tool
- Docker Inc started as PasS provider called as dotCloud which was using the Linux Containers behind the screen to run contianers
- In 2013, the dotCloud became Docker Inc
- It comes in two editions
  - Enterprise Edition (EE) ⇒ $$
  - Community Edition (CE) ⇒ free

Images

- Object that contains an OS filesystem and an application
- You can think of it as a class in Object Oriented Programming language
- Docker provides various pre-built images on Docker Hub





Commands related to images

- **List all the images**

  > docker image ls

- **Download an image from docker hub**

  > docker image pull <image name>

- **Get the details of selected image**

  > docker image inspect <image name>

- **Delete an image**

  > docker image rm <image name>

Commands related to images

- **Push the image to docker hub**

  > docker image push <image name>

- **Tag an image**

  > docker image tag <image name> <tag>

- **Build an image with Dockerfile**

  > docker image build <Dockerfile>

Containers

- It is created using docker image
- You can think of container as an object created by using class
- It consists of
  - Your application code
  - Dependencies
  - Networking
  - Volumes

Commands related to containers

- **List the running containers**

  > docker container ls

- **List all the containers (including stopped)**

  > docker container ls –a

- **Create and start container**

  > docker container run <image>

- **Start a stopped/created container**

  > docker container start <container>

Commands related to containers

- **Stop a container**

  > docker container stop <container>

- **Remove a container**

  > docker container rm <container>

- **Get the stats of selected container**

  > docker container stats <container>

- **Execute a command in a container**

  > docker container exec <container>

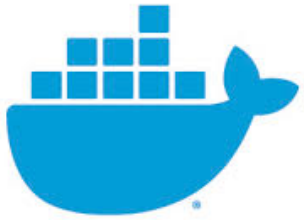
Commands related to containers

- **Create an image from current state of a container**

  > docker container commit <container>

- **Get the processes running in the container**

  > docker container top <container>



Docker Compose

Microservices

- 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

Docker Compose

- Compose is a tool for defining and running multi-container Docker applications
- With Compose, you use a YAML file to configure your application's services
- Then, with a single command, you create and start all the services from your configuration

Features

- Manages multiple services easily
- Multiple isolated environments on a single host
- Only recreate containers that have changed
- Variables and moving a composition between environments

Installation

- Run this command to download the current stable release of Docker Compose

  > sudo curl -L "https://github.com/docker/compose/releases/download/1.24.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

- Apply executable permissions to the binary:

  > sudo chmod +x /usr/local/bin/docker-compose

Start using docker compose

- Docker compose uses docker-compose.yml
- Following is sample docker-compose file

```
version: ‘3’
services:
  web:
    build: .
    ports:
      - “9090: 80”
```

Build and run the application

- To run the application use

  > docker-compose up

- To stop the containers

  > docker-compose stop

- To remove the containers

  > docker-compose down

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

Features

- Matches native data structures of agile methodology and its languages such as Perl, Python, PHP, Ruby and JavaScript
- YAML data is portable between programming languages
- Includes data consistent data model
- Easily readable by humans
- Supports one-direction processing
- Ease of implementation and usage

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