

Microservices



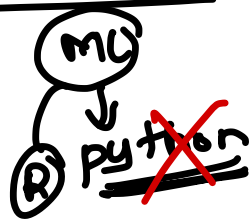
Monolithic vs Microservices

E-commerce app - backend

- product -
- order -
- user -
- category -
- reporting -

X

- ① language
- ② resilient X
- ③ scalable X



> mkdir server

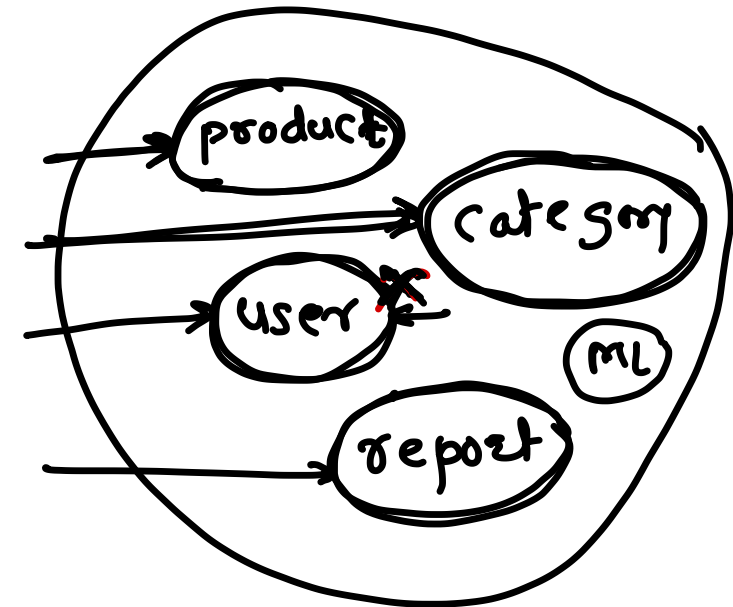
> server.js ←

- product.js
- category.js
- user.js

> node server.js



- user -
- product -
- category -

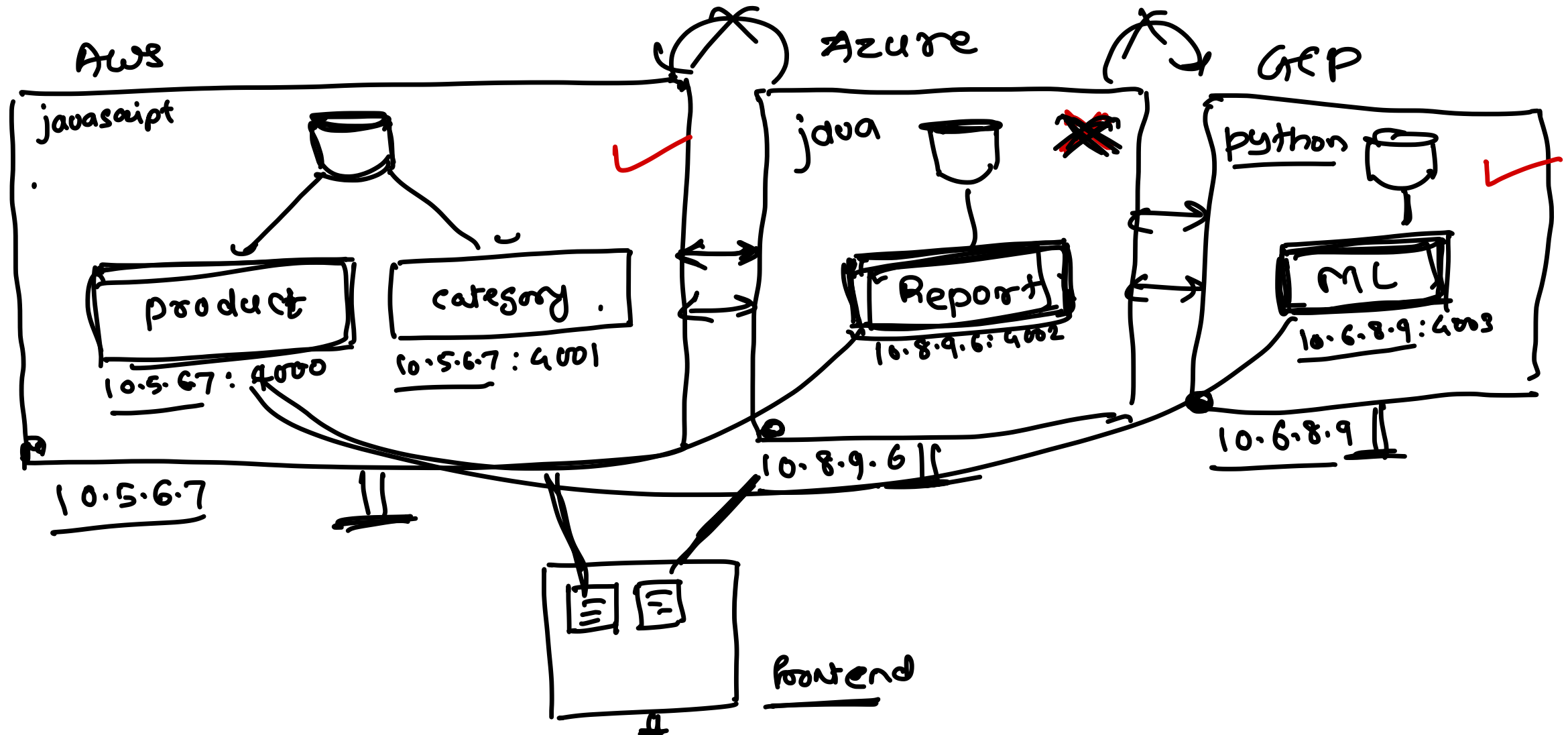


Server

monolithic →

http://localhost:4000

Monolithic vs Microservices



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
- 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 — *messaging*
- Components can be spread across multiple servers or even multiple data centers
- Work very well with containers, such as Docker



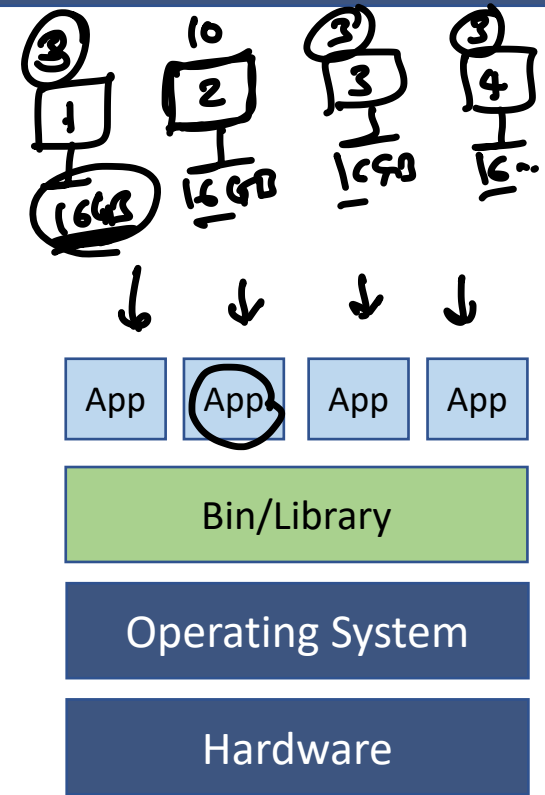


Containerization



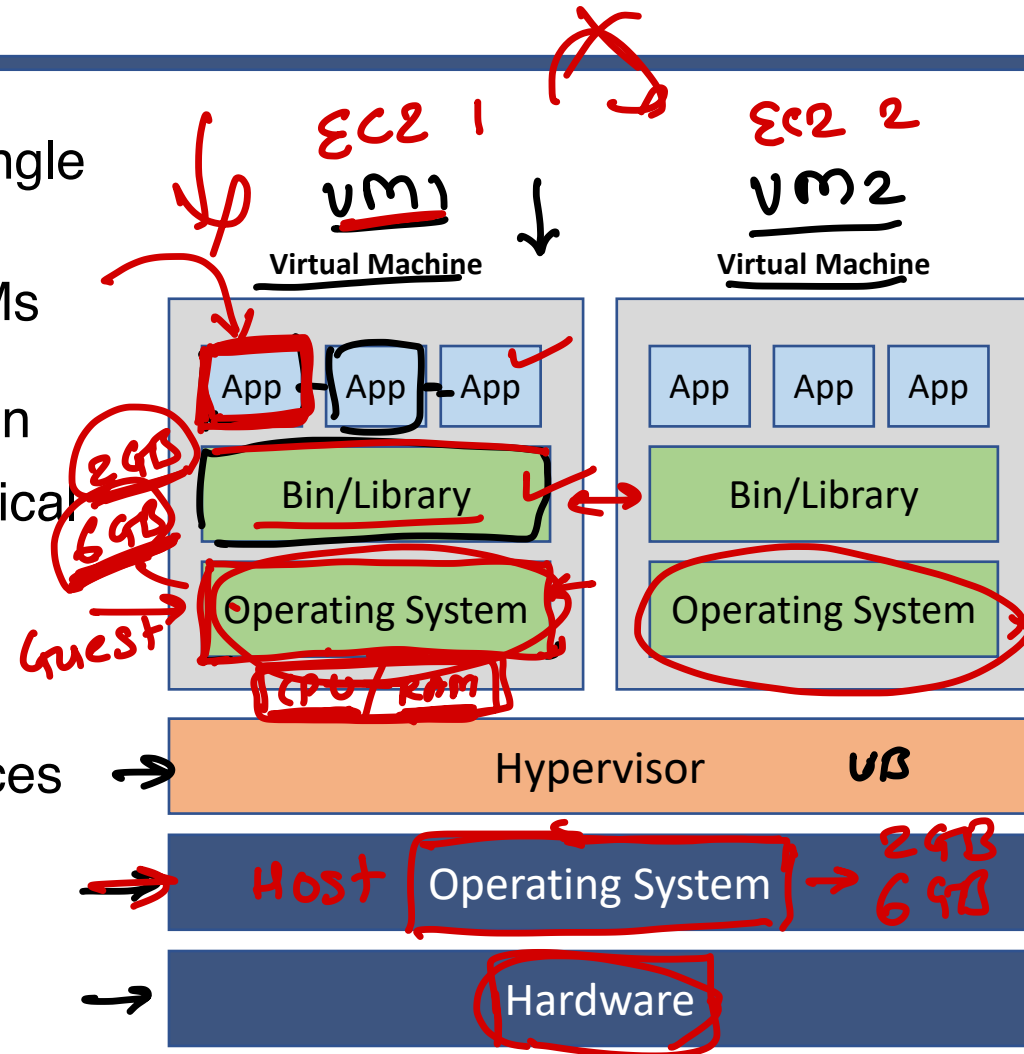
Traditional Deployment

- 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



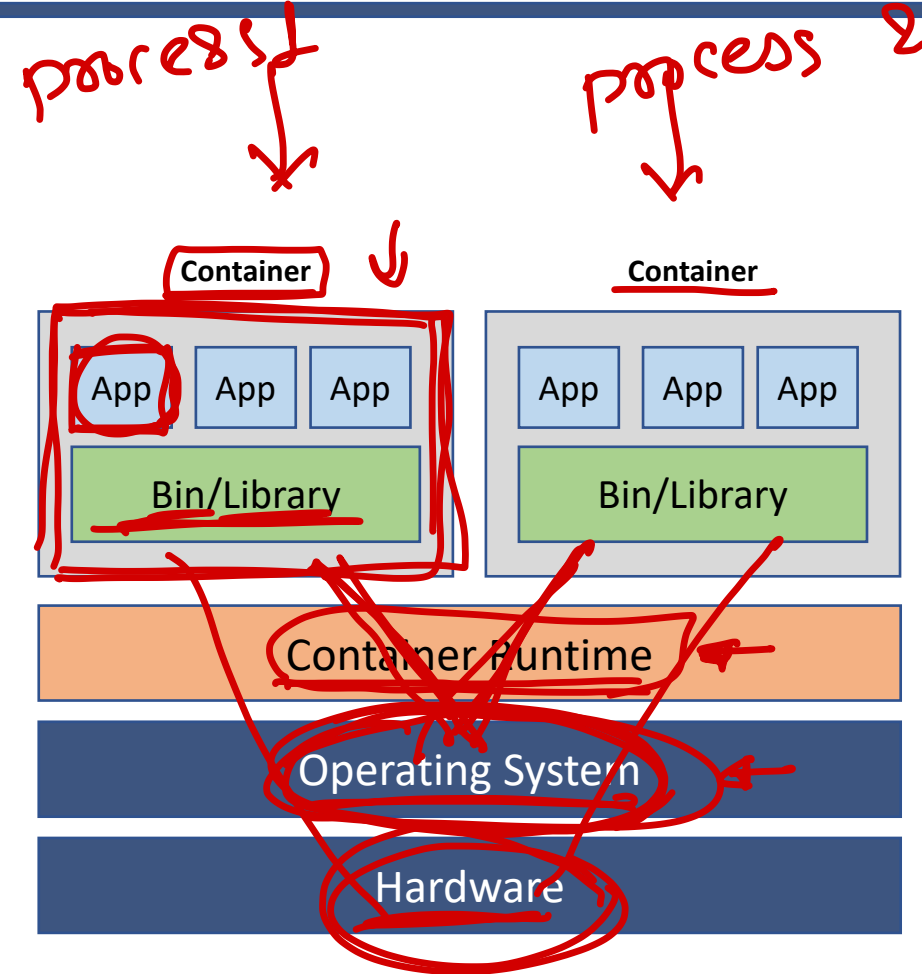
Virtualized Deployment

- 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 are immutable



Containers vs Virtual machines

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



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 PaaS provider called as dotCloud which was using the Linux Containers behind the screen to run containers
 - 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>



