Software Architecture

Architecture

The manner in which the various components of the building are integrated to form a cohesive whole.

In the context of architectural design, a software component can be something as simple as a program module, but it can also be extended to include databases and "middleware" that enable the configuration of a network of clients and servers.

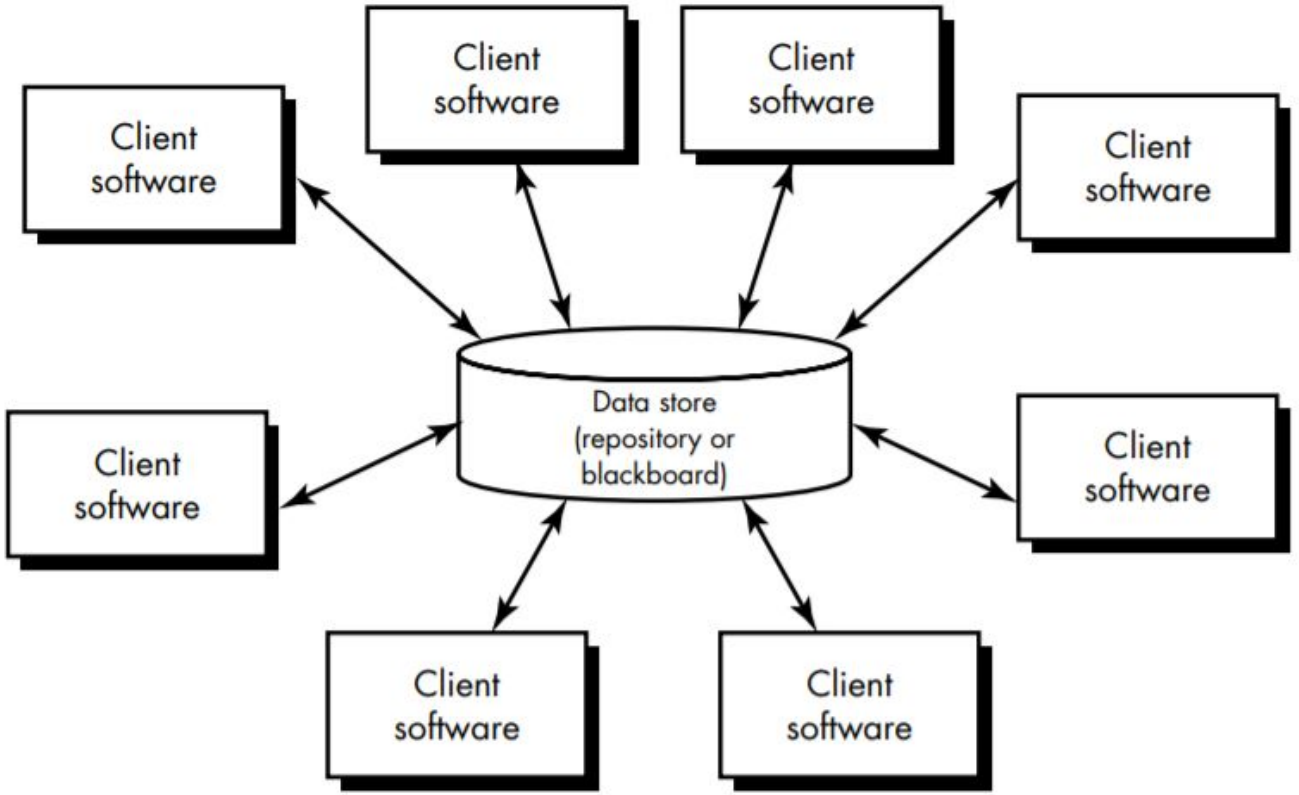
Why architecture is important?

i) Enable communication between all parties.

ii) The architecture highlights early design decisions that will have a profound impact on all software engineering work that follows.

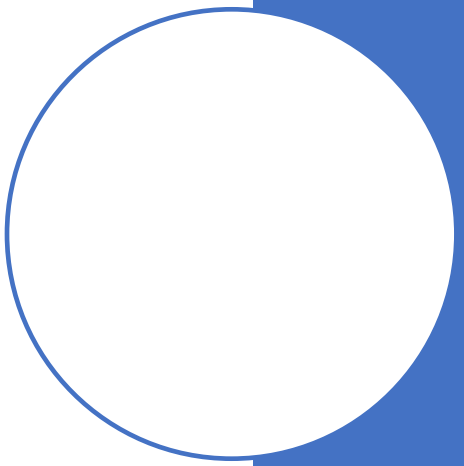
iii) It constitutes a relatively small, intellectually graspable model of how the system is structured and how its components work together.

Data Centered Architecture

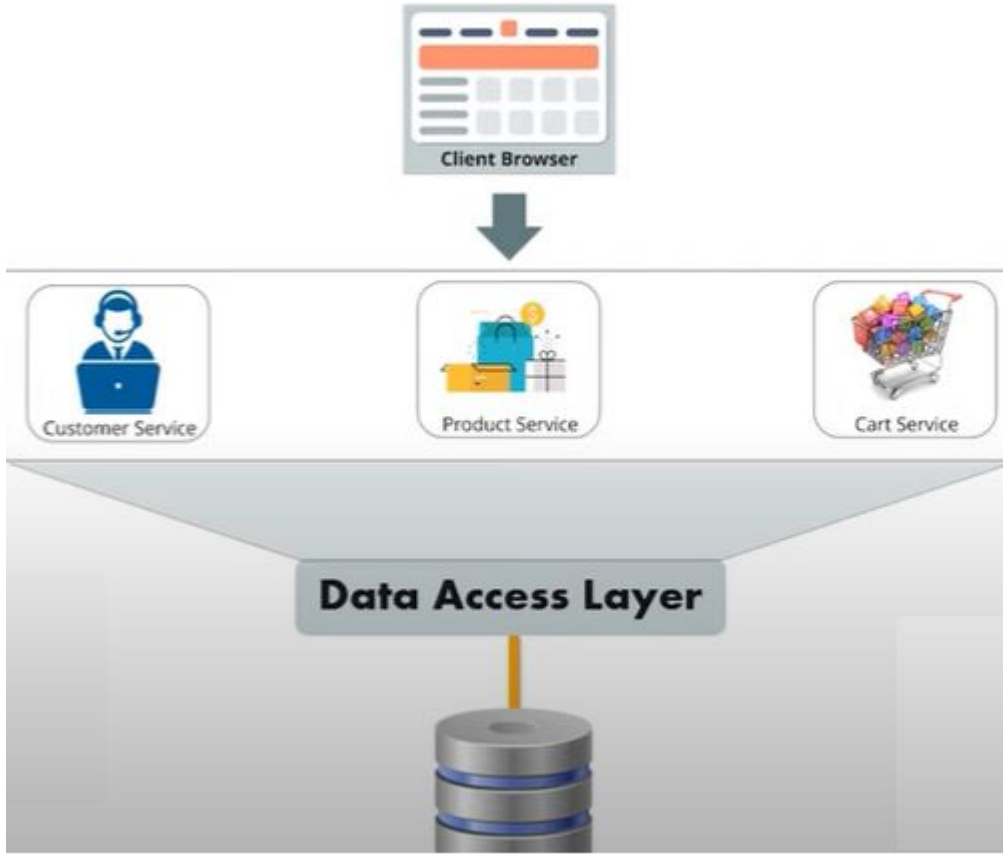


Client Server Architecture

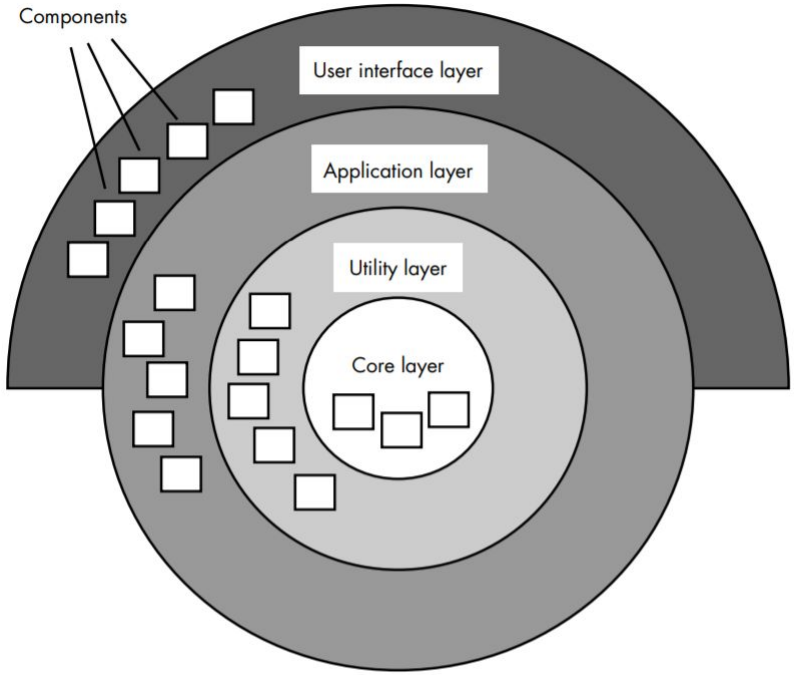
- Client – Server
- Client sends a request to server
- Server sends a response to the client
- Example World wide web (www), sending a request to printer.



Monolithic Architecture Style



Layered Architecture



Layers are a way to separate responsibilities and manage dependencies. Each layer has a specific responsibility. A higher layer can use services in a lower layer, but not the other way around.

N-Tier Architecture.

An N-tier application can have a **closed layer architecture** or an **open layer architecture**:

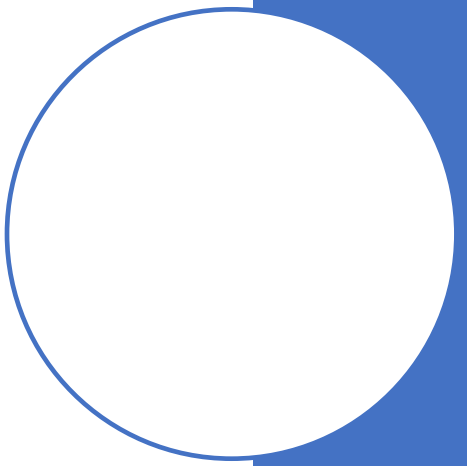
- In a closed layer architecture, a layer can only call the next layer immediately down.
- In an open layer architecture, a layer can call any of the layers below it.

Microservices Architecture

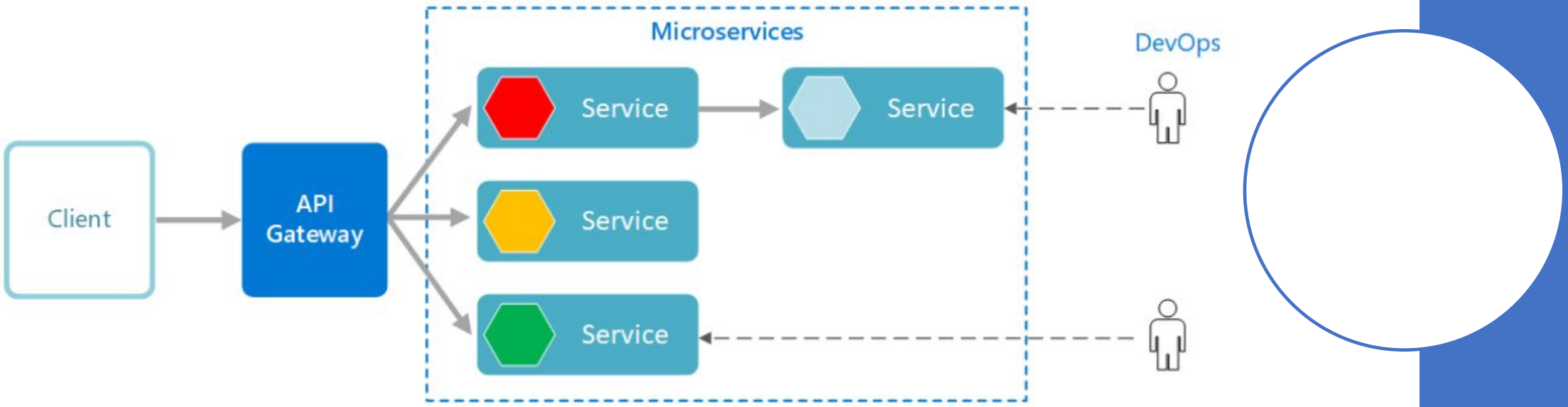
A microservices architecture consists of a collection of small, autonomous services.

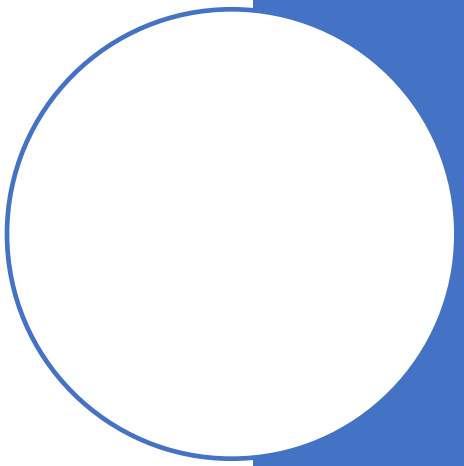
Each service is self-contained and should implement a single business capability within a bounded context.

Microservices are small, independent, and loosely coupled. Services communicate with each other by using well-defined APIs. Internal implementation details of each service are hidden from other services.



Microservices Architecture



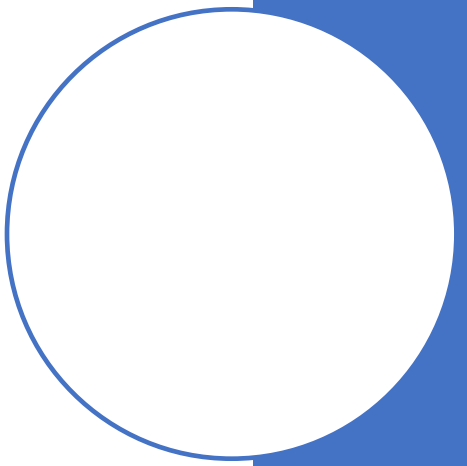


DevOps

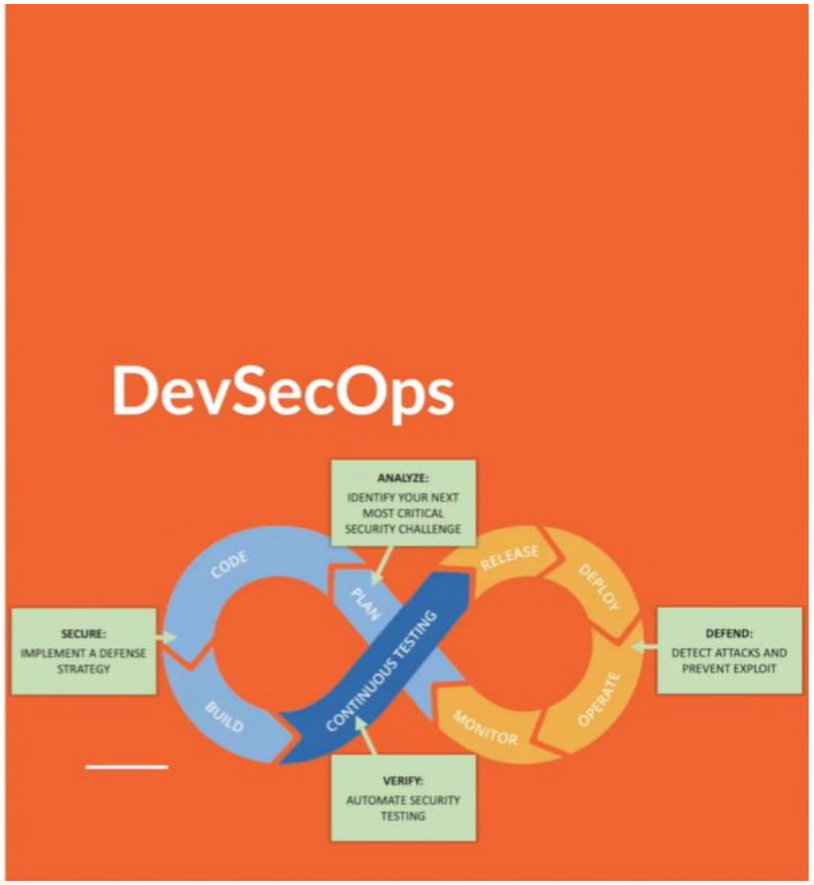
We have the

(i) Development team, and (ii) Operational team.

DevOps is the culture of collaboration between these two teams.



DevOps and DevSecOps



Practices involved in DevOps

- **Continuous integration (CI)** – merges code changes to ensure the most recent version is available to developers. As an example, using version control software e.g., git, SVN, Microsoft teams etc.

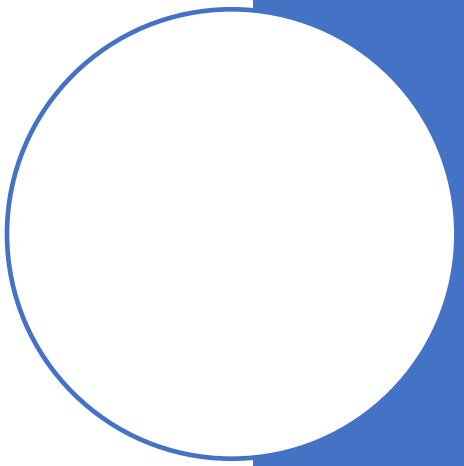
- **Continuous delivery and continuous deployment (CD)** – automates the process of releasing updates to increase efficiency e.g., processes automation using Jenkins. Containerization tool e.g., docker.

- **Microservices**

DevSecOps

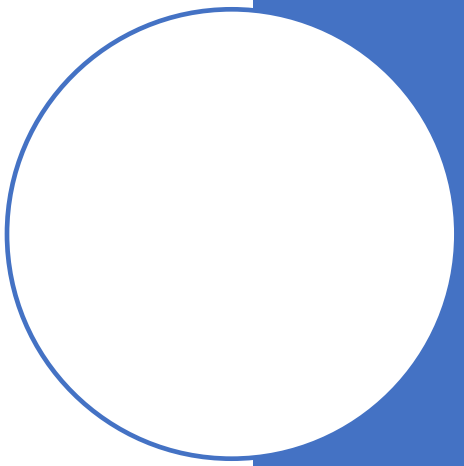
Through this method, application security begins at the outset of the build process, instead of at the end of the development pipeline.

DevSecOps emphasizes that developers should create code with security in mind and aims to solve the issues with security that DevOps doesn’t address.



Practices involved in DevSecOps Process

- **Common weaknesses enumeration (CWE)** – improves the quality of code and increases the level of security during the CI and CD phases.
- **Threat modeling** – implements security testing during the development pipeline to save time and cost in future.
- **Automated security testing** – test for vulnerabilities in new builds on regular basis.
- **Incident management** – creates a standard framework for responding to security incidents.



Virtualization

In 1960s, the problem of sharing the computer resources among multiple applications is faced. Only one application could run at a time.

Later on, several mechanisms were proposed to share the resources among applications, but a new problem was faced. What, if one application consumes all the resources?

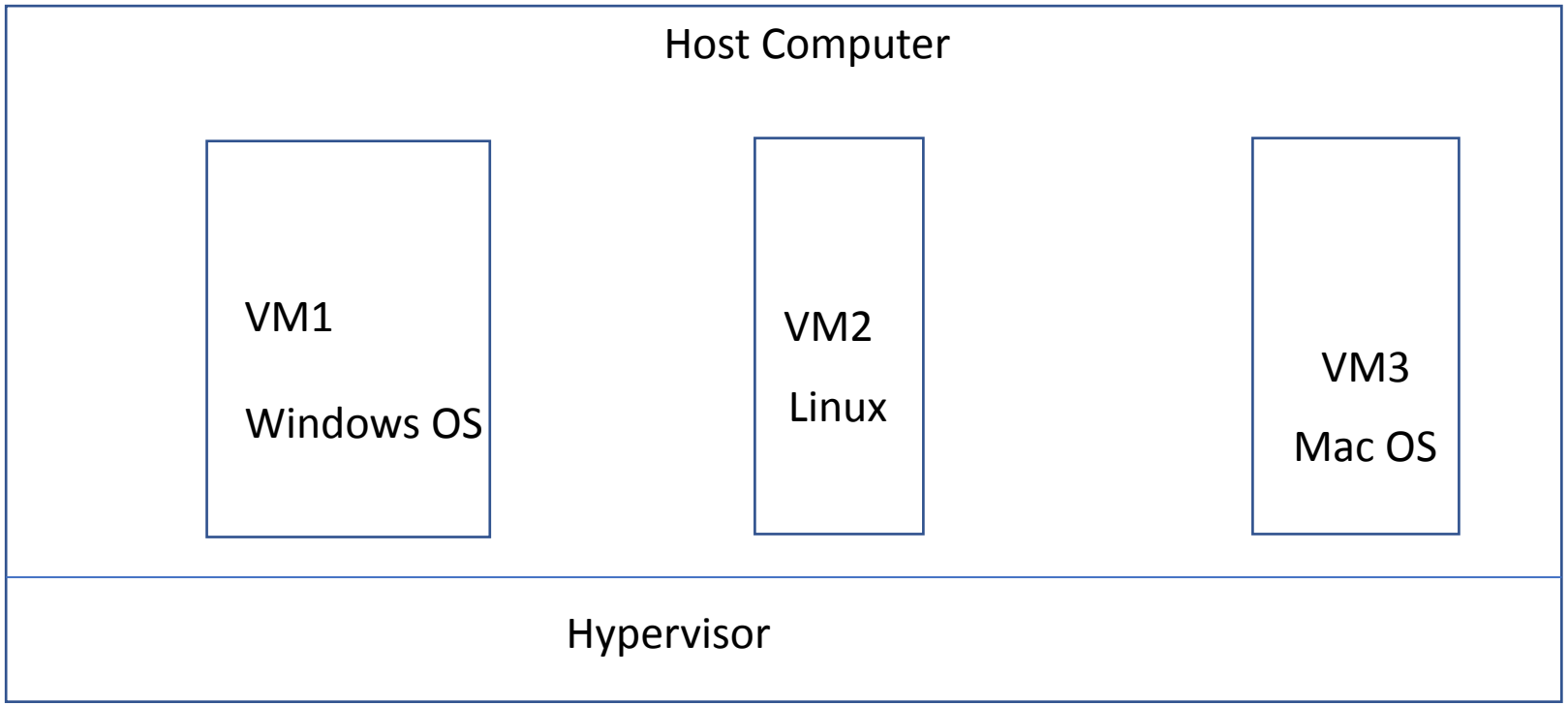
Virtual Machines

Virtualization (using VMs) solve the problem using isolation and sharing mechanisms among resources, i.e., process scheduling (schedular assign threads to CPUs), using virtual memory to complement physical memory, disk controller to access the disk and allowing only the valid threads to access the disk content, and a network isolation is achieved through the identification of messages.

Every VM has an address used to identify messages to or from that VM. The hypervisor implements network infrastructure within the physical machine that allows VMs to share and isolate use of physical network interface using the same approaches and protocols used for networking between physical machines.

VMs allow the execution of multiple simulated, or virtual, computers in a single physical computer.

Virtualization



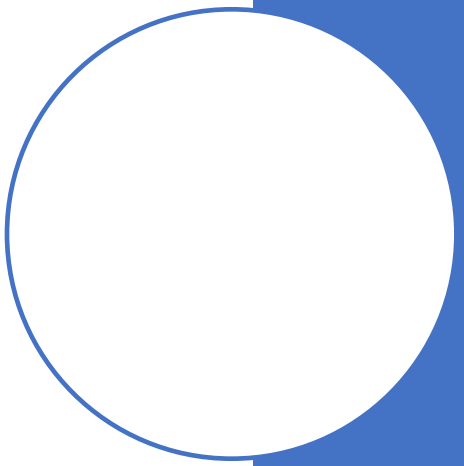
Containers

VMs solve the problem of sharing resources and maintaining isolation but VM images can be large and is time consuming to transfer VM images around the network i.e., transferring 8GB VM image to 2000 machines.

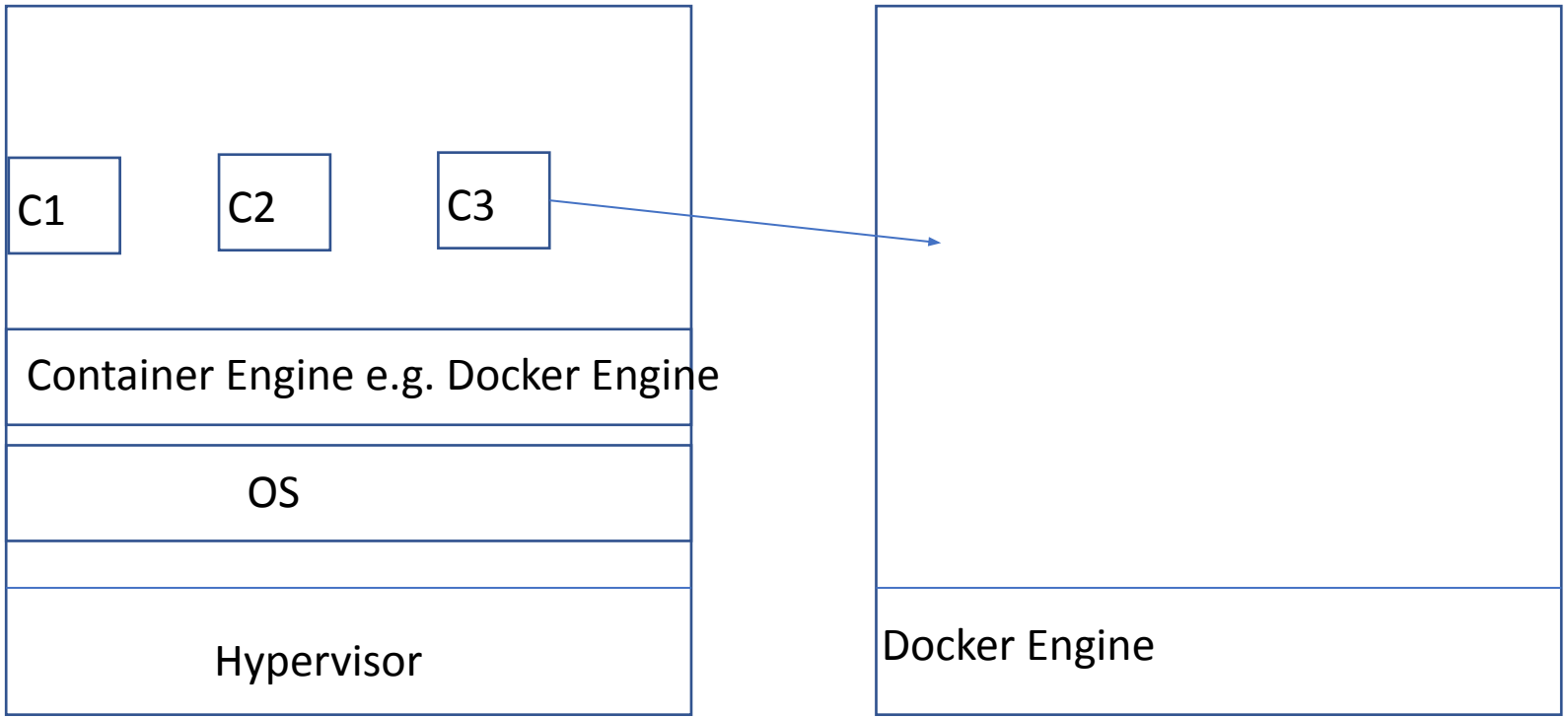
Once the image is transferred, it needs to boot the OS and start your services, which still takes more time.

Containers are a mechanism to maintain the advantages of virtualization while reducing the image transfer time and start-up time.

Example: a single container can be considering a package containing a “software, its dependencies, configuration etc.”.



Container


C1
C2
C3
Container Engine e.g. Docker Engine
OS
Hypervisor
Docker Engine