Advance Software Engineering
Cloud Based Software

By:

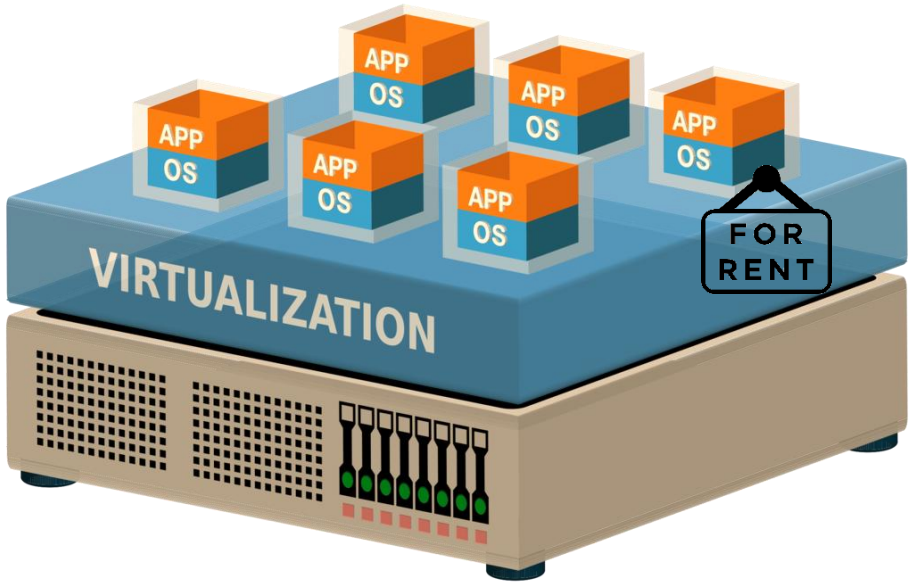
Dr. Salwa Osama

Cloud

- The cloud is made up of very large number of remote servers that are offered for rent by companies that own these servers.
  - Cloud-based servers are ‘virtual servers’, which means that they are implemented in software rather than hardware.

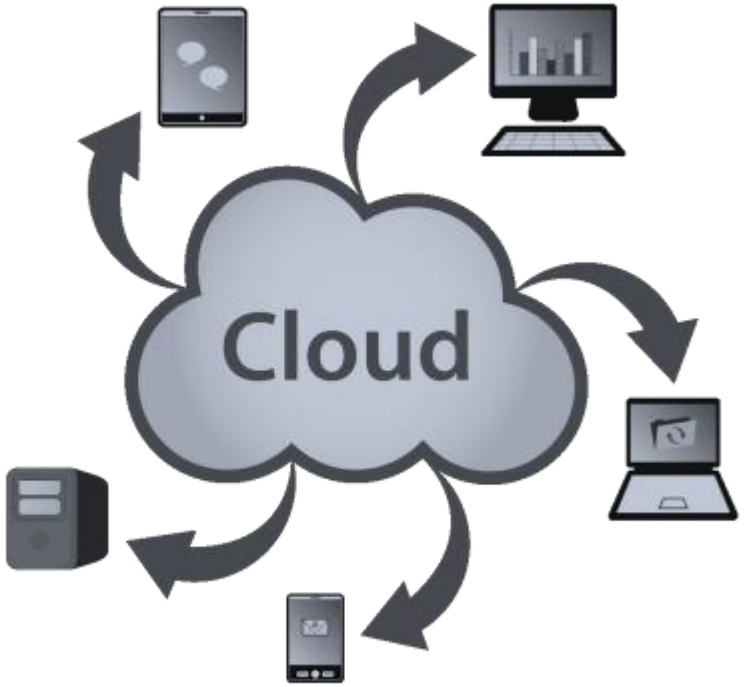
Cloud

- You can rent as many servers as you need, run your software on these servers and make them available to your customers.



Cloud

- Your customers can access these servers from their own computers or other networked devices such as a tablet or a TV.
- Cloud servers can be started up and shut down as demand changes.



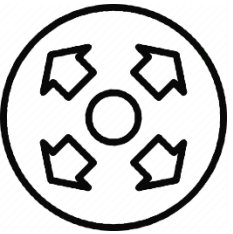
Cloud

- You may rent a server and install your own software, or you may pay for access to software products that are available on the cloud.

Benefits of using the cloud for software development rather than private server

- *Cost*
  - You avoid the initial capital costs of hardware procurement

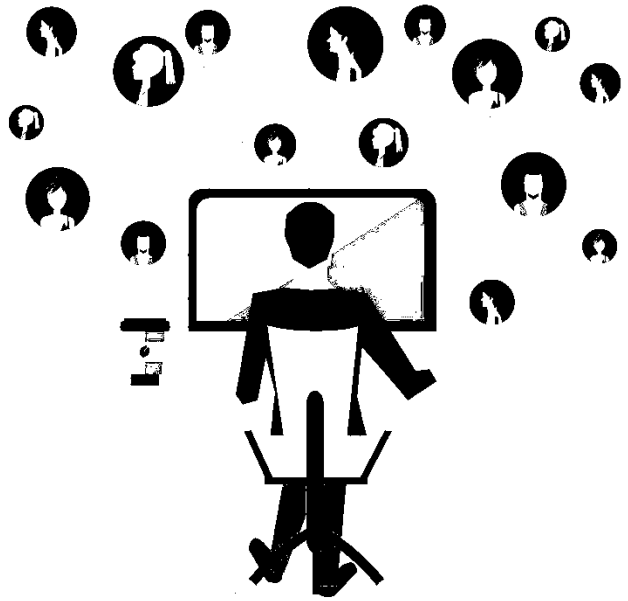
- *Startup time*
  - You don’t have to wait for hardware to be delivered before you can start work. Using the cloud, you can have servers up and running in a few minutes.



- *Server choice*
  - If you find that the servers you are renting are not powerful enough, you can upgrade to more powerful systems. You can add servers for short-term requirements, such as load testing.

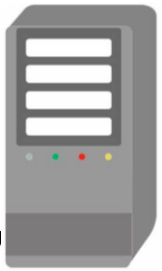
Benefits of using the cloud for software development rather than private server

- *Distributed development*
  - If you have a distributed development team, working from different locations, all team members have the same development environment and can seamlessly share all information.

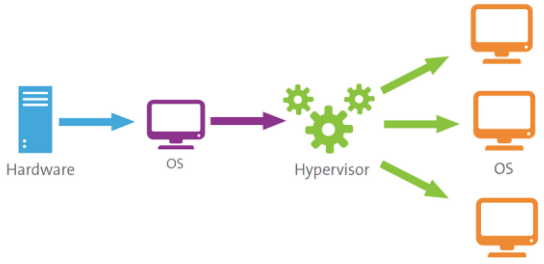


Virtual Machine &Virtual Server

- Virtual Machine:
  - It is something which acts like a real computer with an OS.
- Physical Server:
  - It is a designated or used by single user and it is not shared by multiple users.
  - Each physical server has its own OS to run programs and application, memory hard drive, processors and network connection.

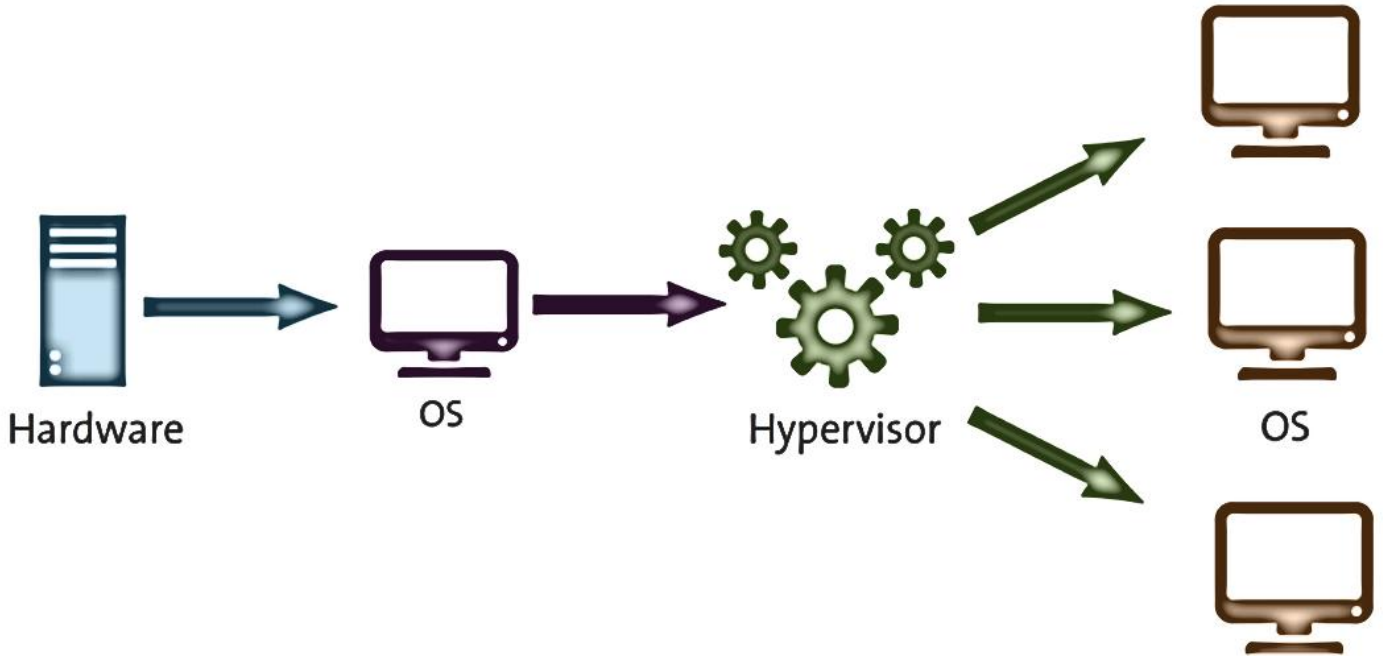


- But if we install a hypervisor on physical server we can create and manage virtual machine and then all these virtual machines will have their own resources, OS and server applications.



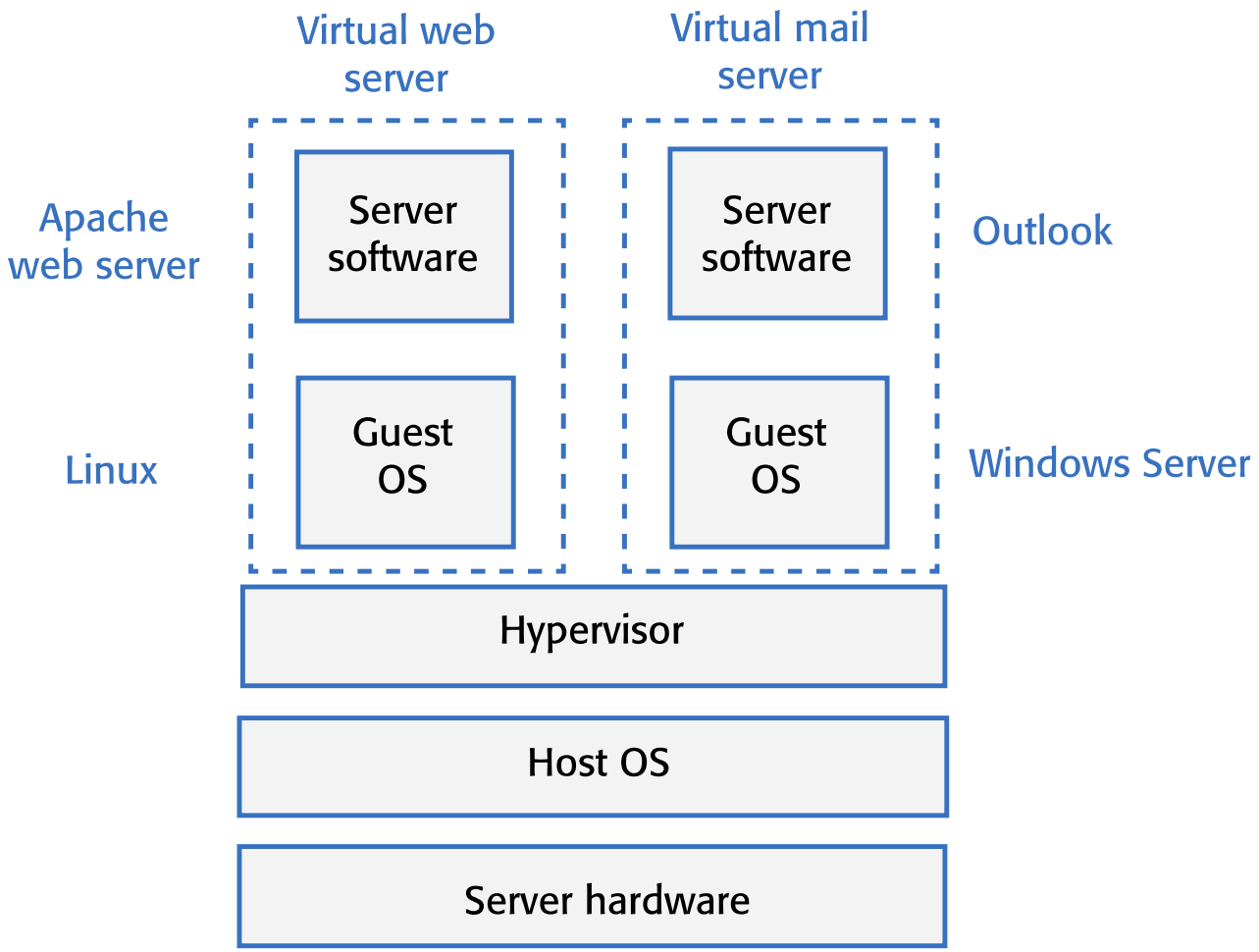
Virtual Machine & Virtual Server

- But if we install a hypervisor on physical server we can create and manage virtual machine and then all these virtual machines will have their own resources, OS and server applications.
  - Then these virtual machines only can be used for different purposes and if they are used as a server then they are called virtual server.



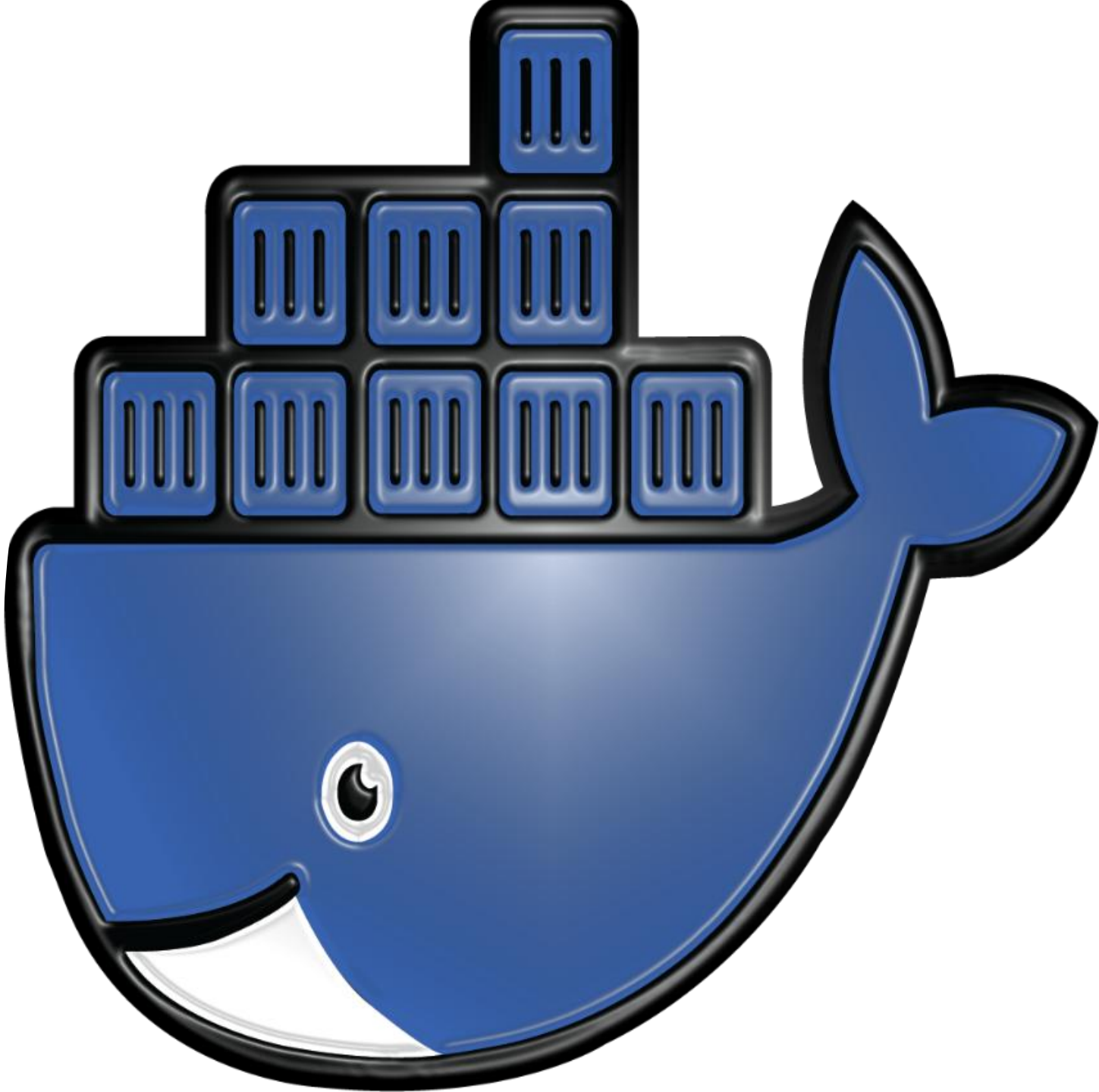
Virtual machines (VMs), running on physical server hardware, can be used to implement virtual servers.

Implementing a virtual server as a virtual machine

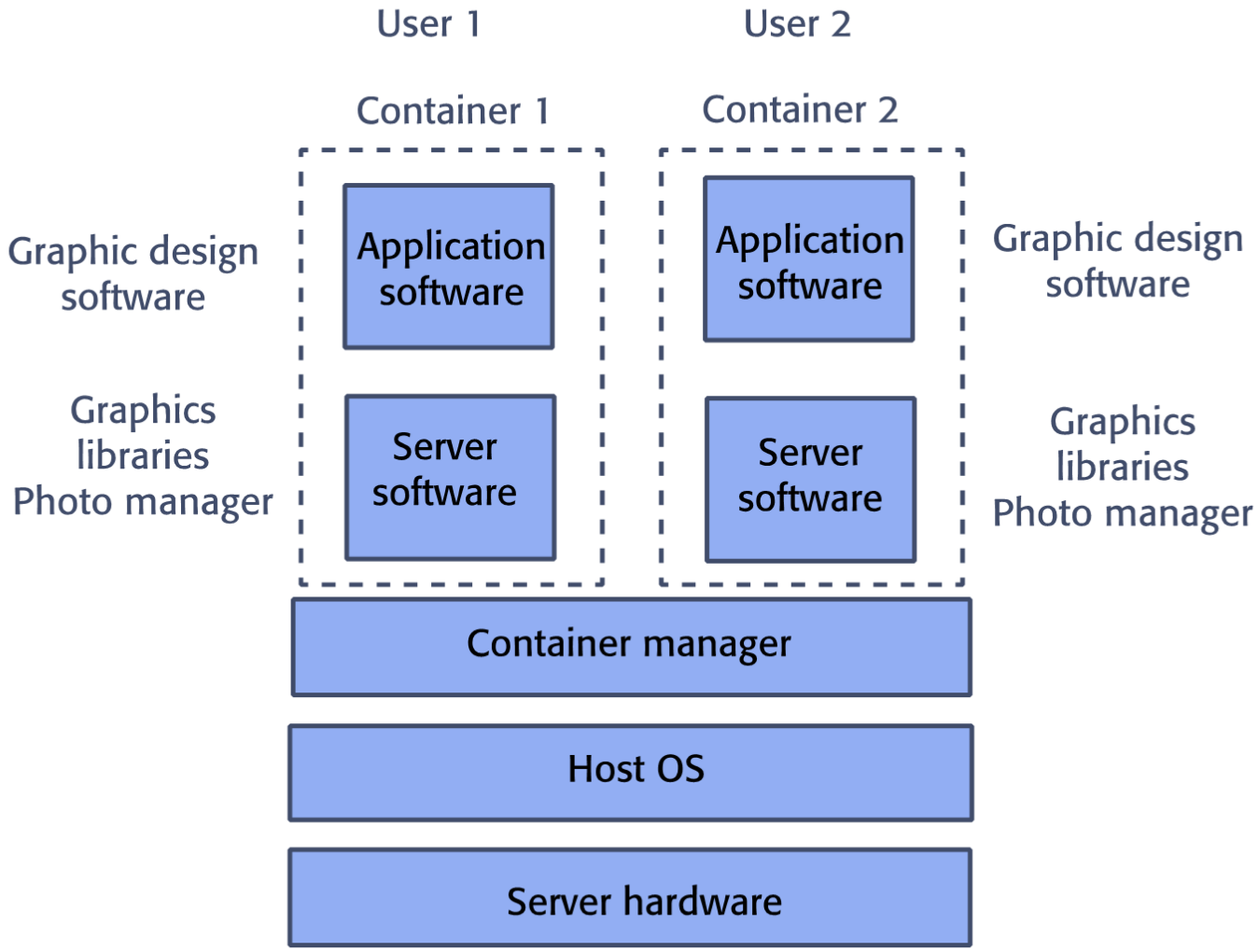


Container-based virtualization

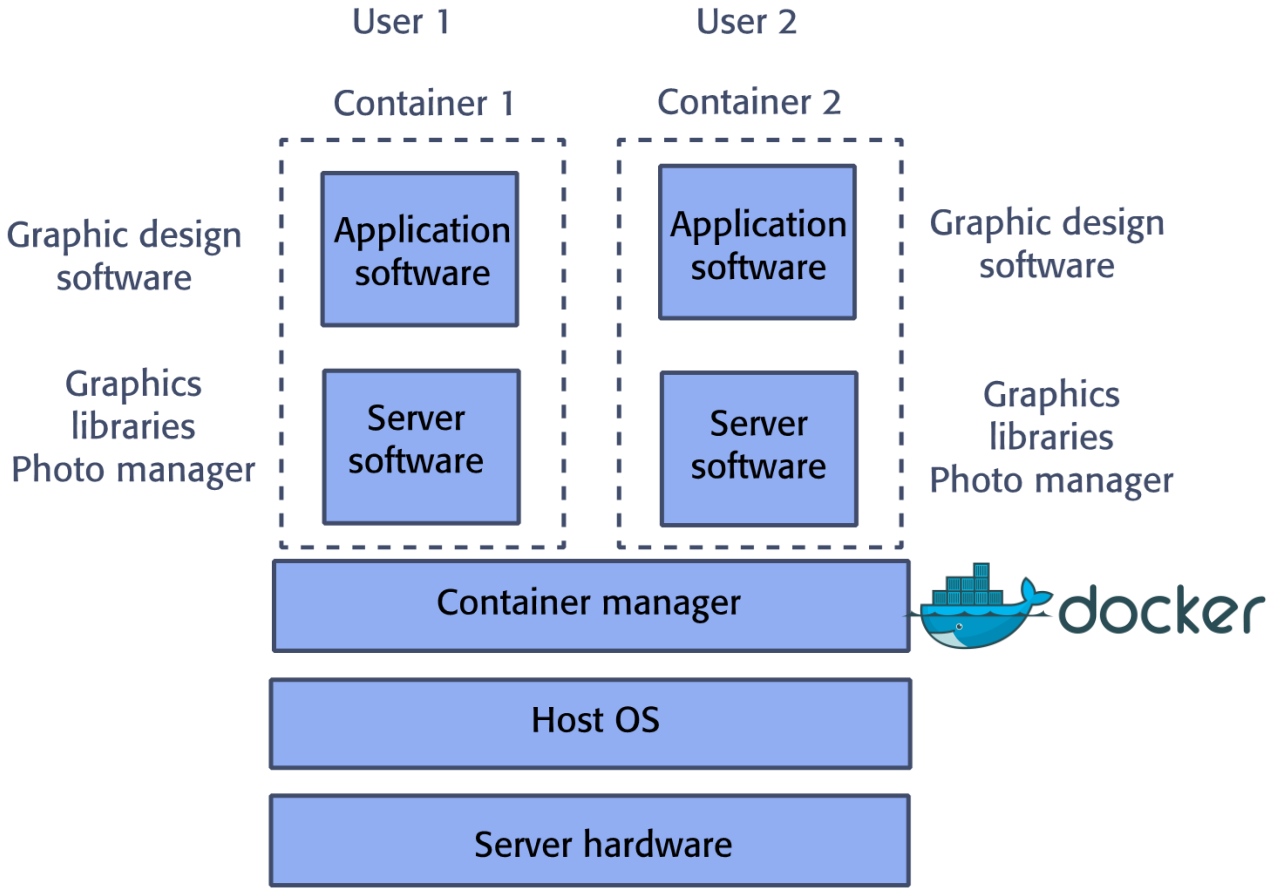
If you are running a cloud-based system with many instances of applications or services, these all use the same operating system, you can use a simpler virtualization technology called ‘containers.

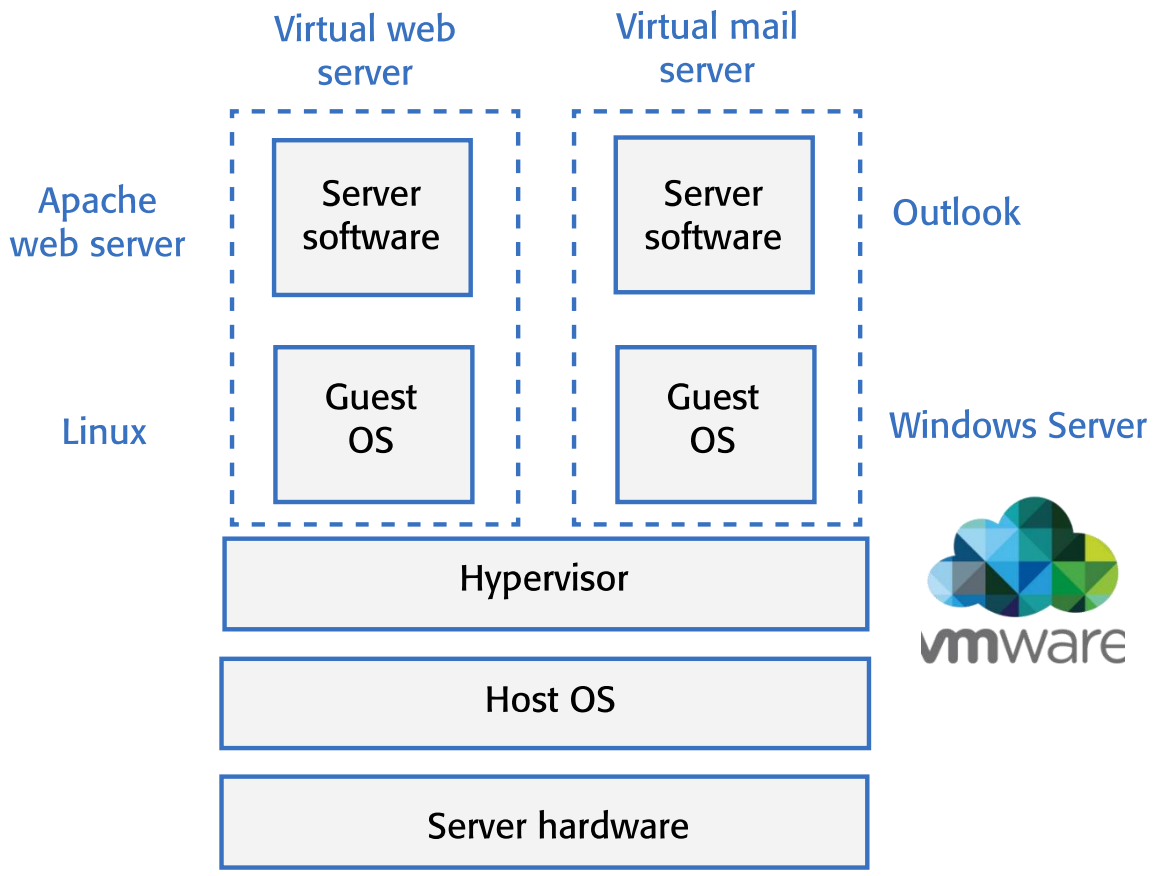


Container-based virtualization



Container-based virtualization

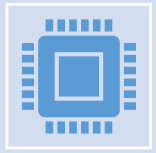




Can You Run Windows and Linux Containers on the Same Machine?

❌ **Not natively, no.**

✅ **But yes, with some conditions and help from virtualization.**



Containers rely on the **host OS kernel**.

Linux containers need a **Linux kernel**.

Windows containers need a **Windows kernel**.

You **can’t mix and match kernels** because containers don’t emulate operating systems — they **share the host kernel**.

So How Can You Run Both?

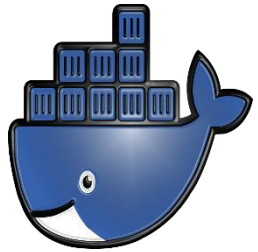
Option 1: Use Docker Desktop (on Windows)

- Docker Desktop uses **WSL 2 (Windows Subsystem for Linux v2)** to run a **Linux kernel inside Windows**.
- With Docker Desktop:
  - You can switch between **Linux containers** and **Windows containers**.
  - But you can't run both at the exact same time (natively).

Option 2: Use Virtual Machines

- Run a virtual machine:
- Linux VM on a Windows host to run Linux containers.
- Or Windows VM on Linux host for Windows containers.

Docker



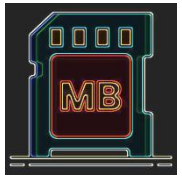
- It is an open-source standalone application which works as an engine used to run containerized applications. It is installed on your operating system, preferably on Linux, but can be also installed on window and macOS.
- Multiple Docker containers can be run on the single operating system simultaneously, you can manage those containers with Docker.
- Docker applications run in containers that can be used on any system: a laptop, on server or in the cloud.

Simply we can say Docker is a container management service.

Container-based virtualization

- Using containers accelerates the process of deploying virtual servers on the cloud.

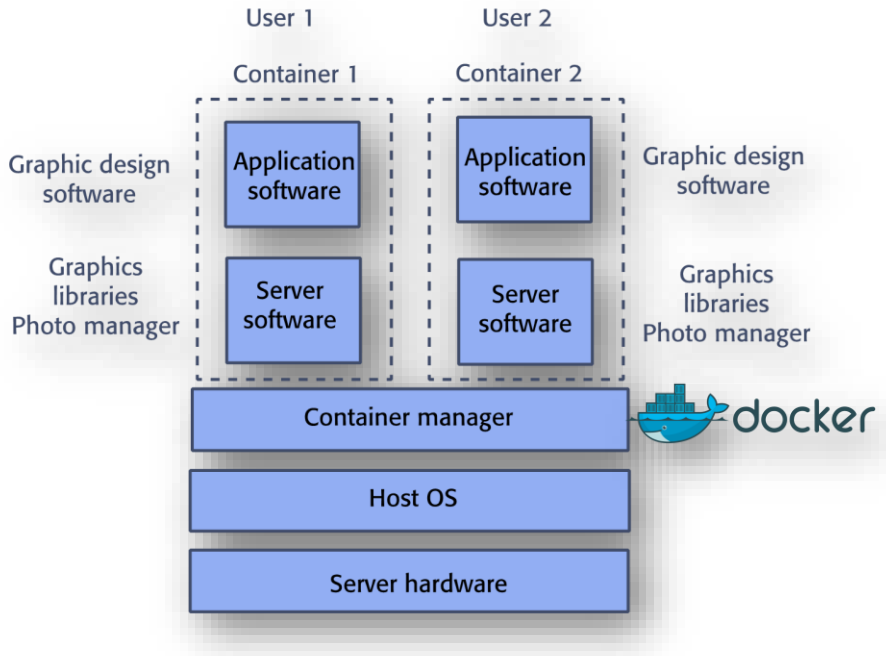
- Containers are usually megabytes in size whereas VMs are gigabytes.



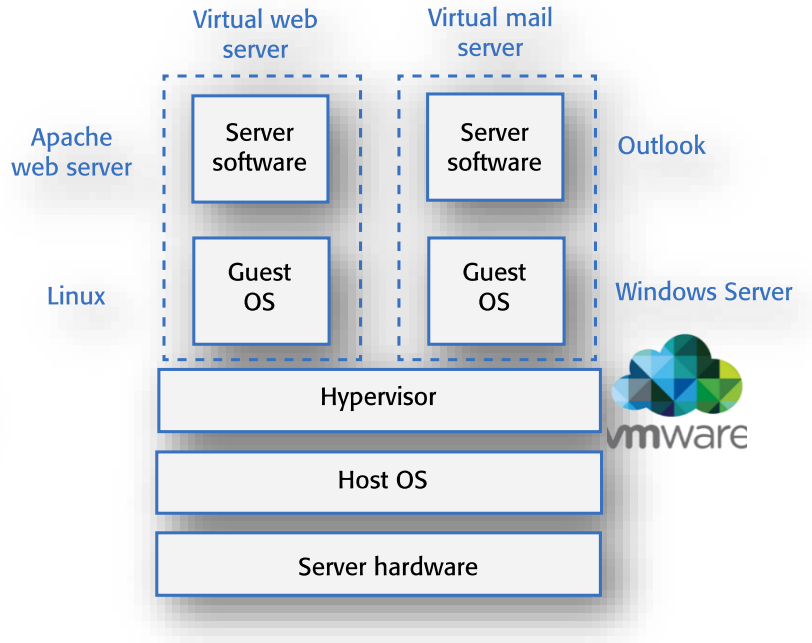
- Containers can be started and shut down in a few seconds rather than the few minutes required for a VM.

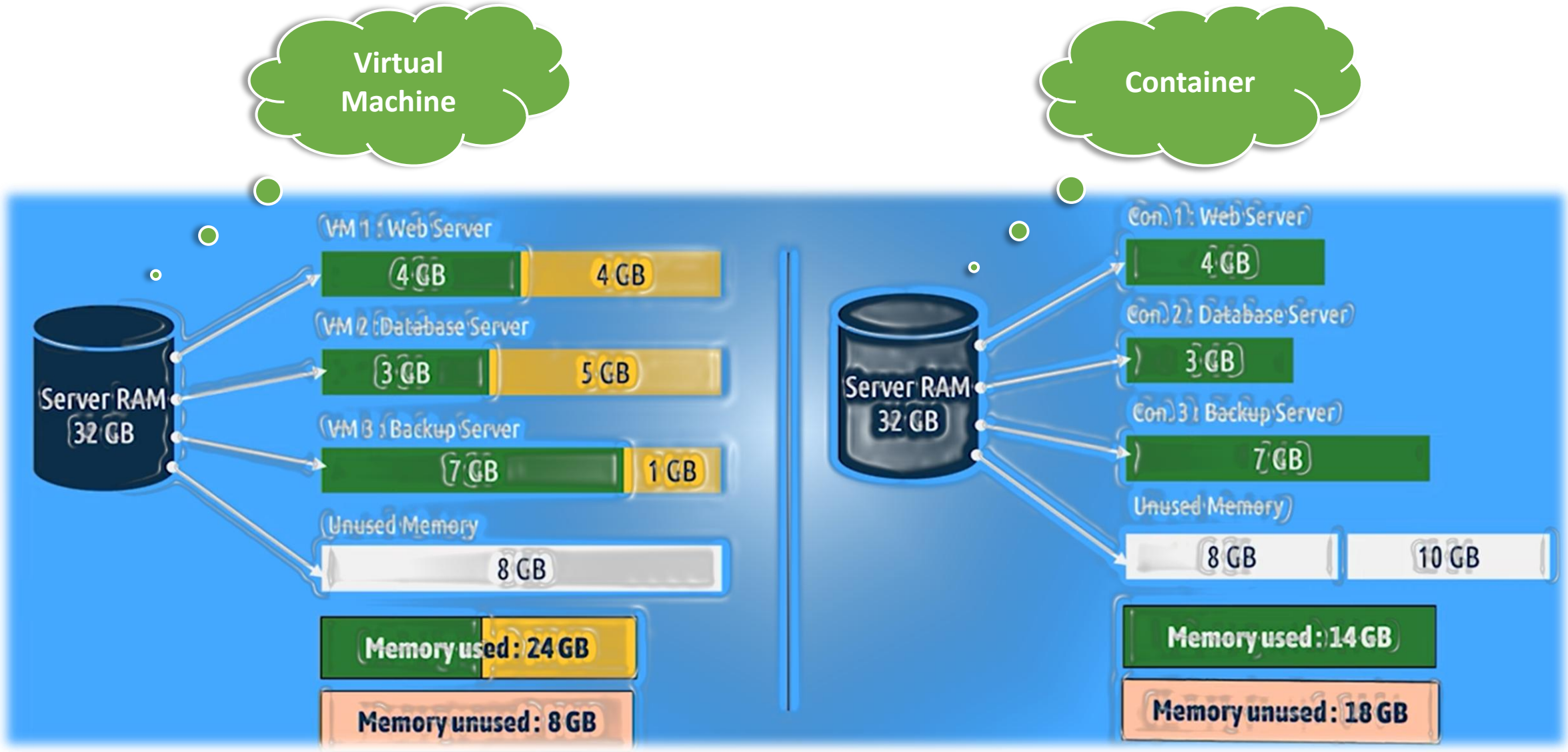
Benefits of Containerization

Portability



Lightweight




Virtual Machine
Container
Server RAM 32 GB
VM 1 : Web Server
4 GB
4 GB
VM 2 : Database Server
3 GB
5 GB
VM 3 : Backup Server
7 GB
1 GB
Unused Memory
8 GB
Memory used : 24 GB
Memory unused : 8 GB
Server RAM 32 GB
Con. 1 : Web Server
4 GB
Con. 2 : Database Server
3 GB
Con. 3 : Backup Server
7 GB
Unused Memory
8 GB
10 GB
Memory used : 14 GB
Memory unused : 18 GB

Docker Terminology

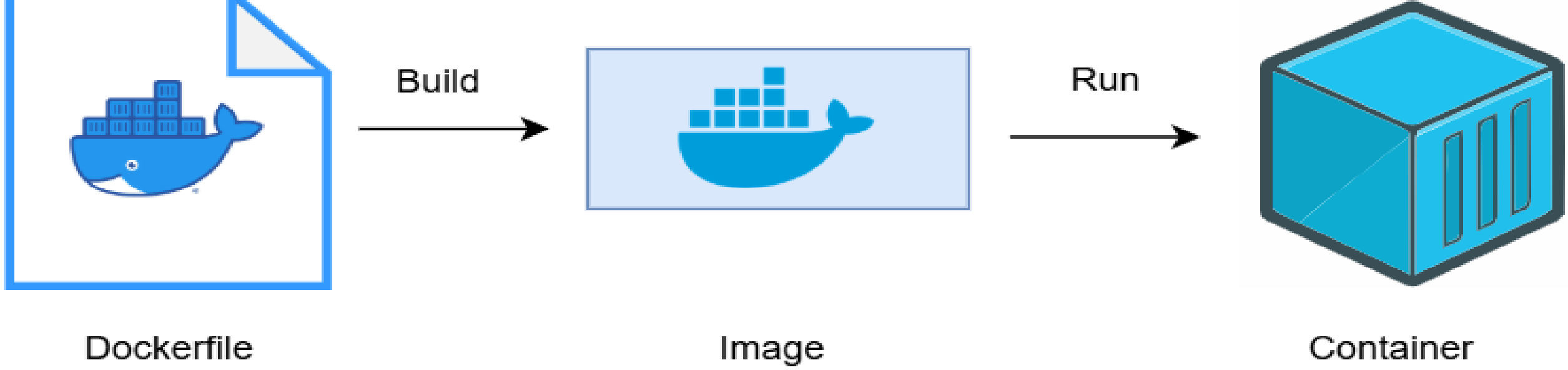
A **Docker file** is just a simple text file containing all the instructions for building an image.

It includes a series of commands and settings that specify

- which base image to use,
- what packages and dependencies to install,
- what files to include,
- and how to configure the environment.

It allows us to automate the process of creating a consistent and reproducible image for our application or service.

Once a Dockerfile is created, you can use the **docker build** command to build a Docker image based on the instructions in the file.



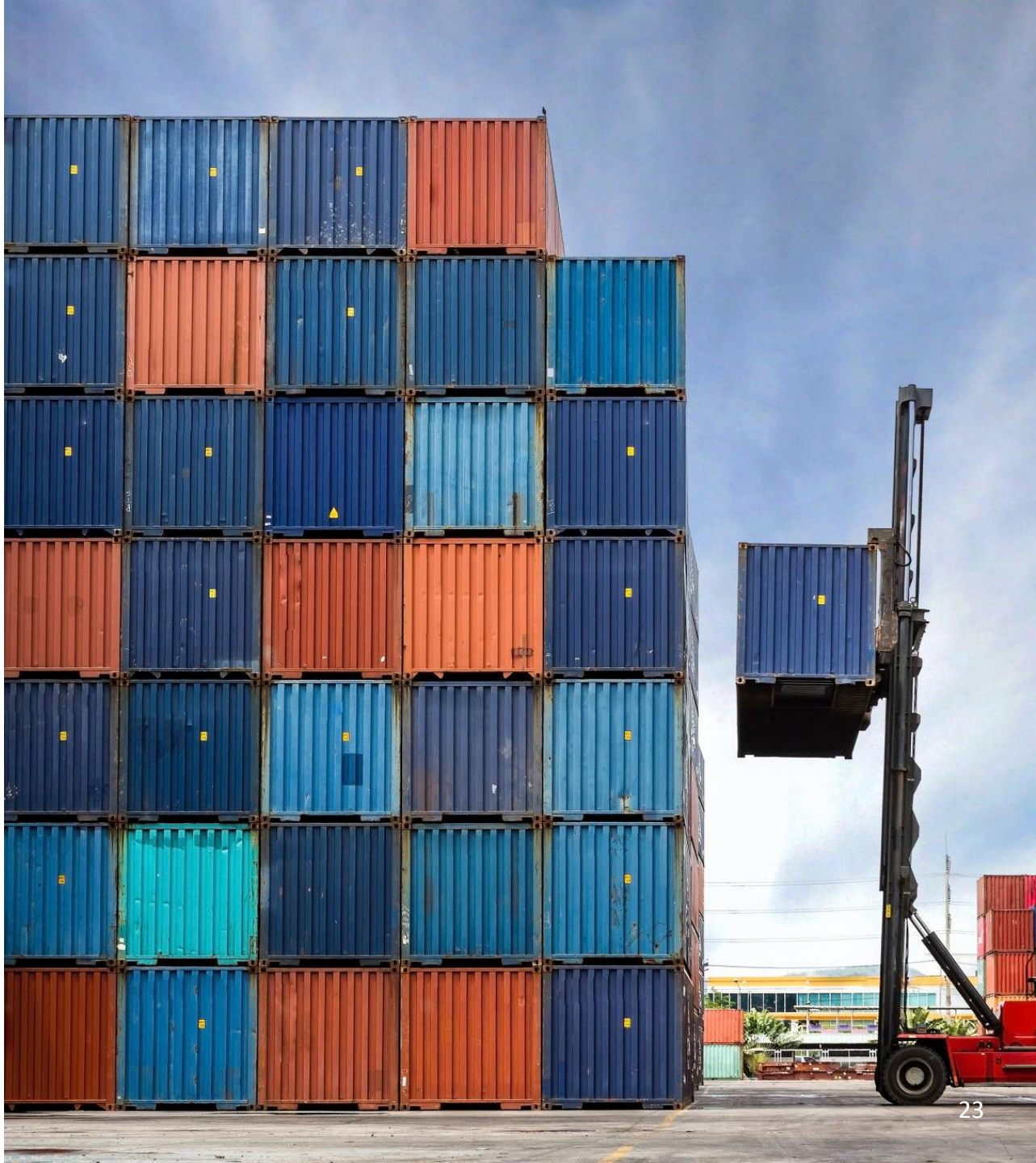
Docker Terminology

A **Docker image** is a lightweight, standalone, executable software package that contains all the necessary components to run an application, including code, runtime, system libraries, and settings.

- Once an image is built, you can share, distribute, and run it on any system that supports Docker.
- You can also store docker images in an online registry (such as Docker Hub, just one of many registries where you can store images.).
- You can also create versions of the uploaded images to track changes and update your application over time.

Docker Terminology

- A **Docker container** is like a running instance of a Docker image.
- It's your application in action, complete with all the dependencies it needs to function.
- When you run a Docker image, it becomes an active container that operates in isolation from other containers, ensuring that the application runs consistently regardless of where the container is deployed.



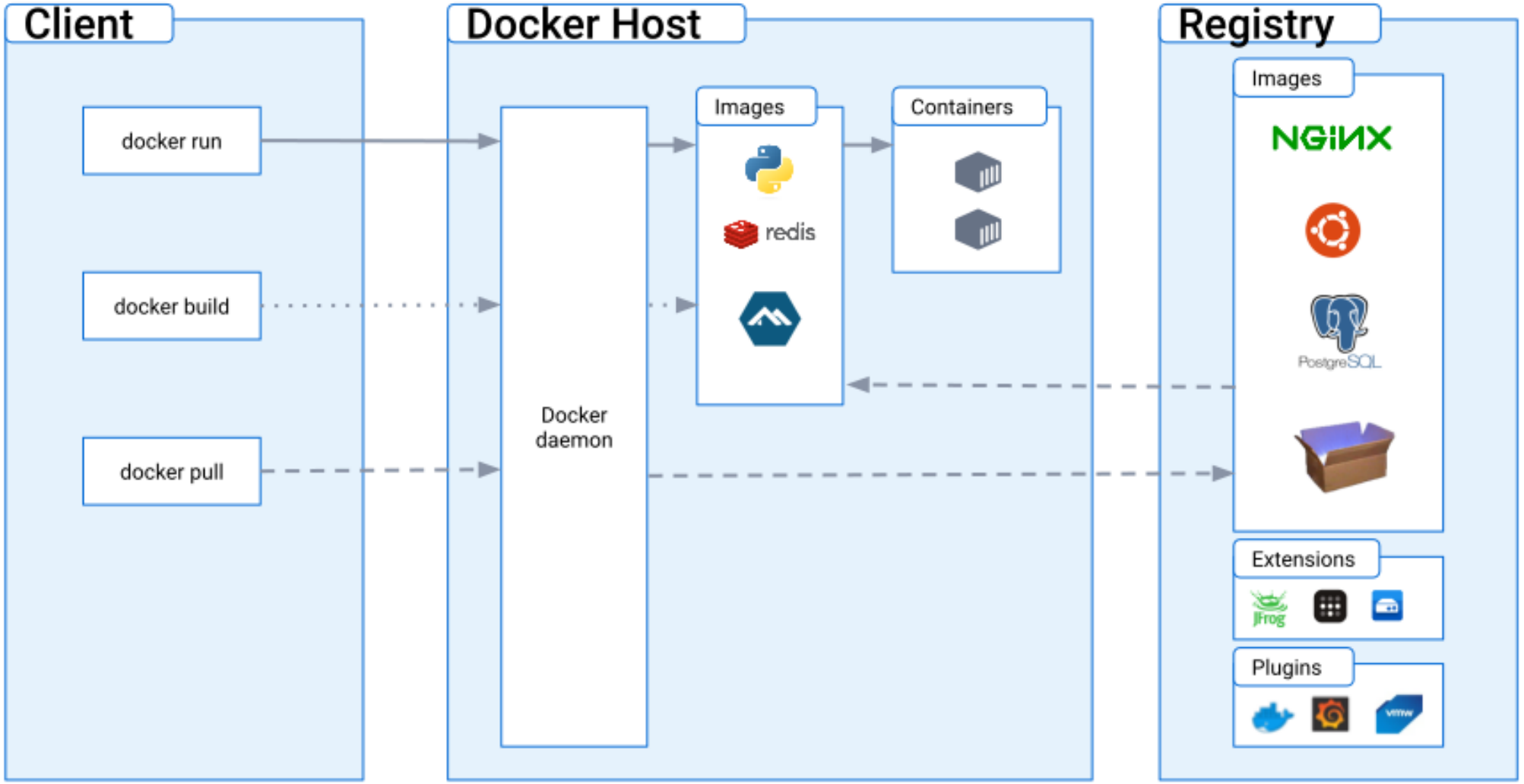
Docker Terminology

Docker daemon is the most important thing that is used for communication between these three components.

The Docker daemon acts as a middleman between these components, reads commands from the Client, and performs operations depending on the command.

Docker Architecture

The `docker pull` command is used to download Docker images from a registry. This command is essential for retrieving the necessary container images before running them as containers. Docker Hub is the default registry, but you can also specify other registries.



Docker Architecture

Image

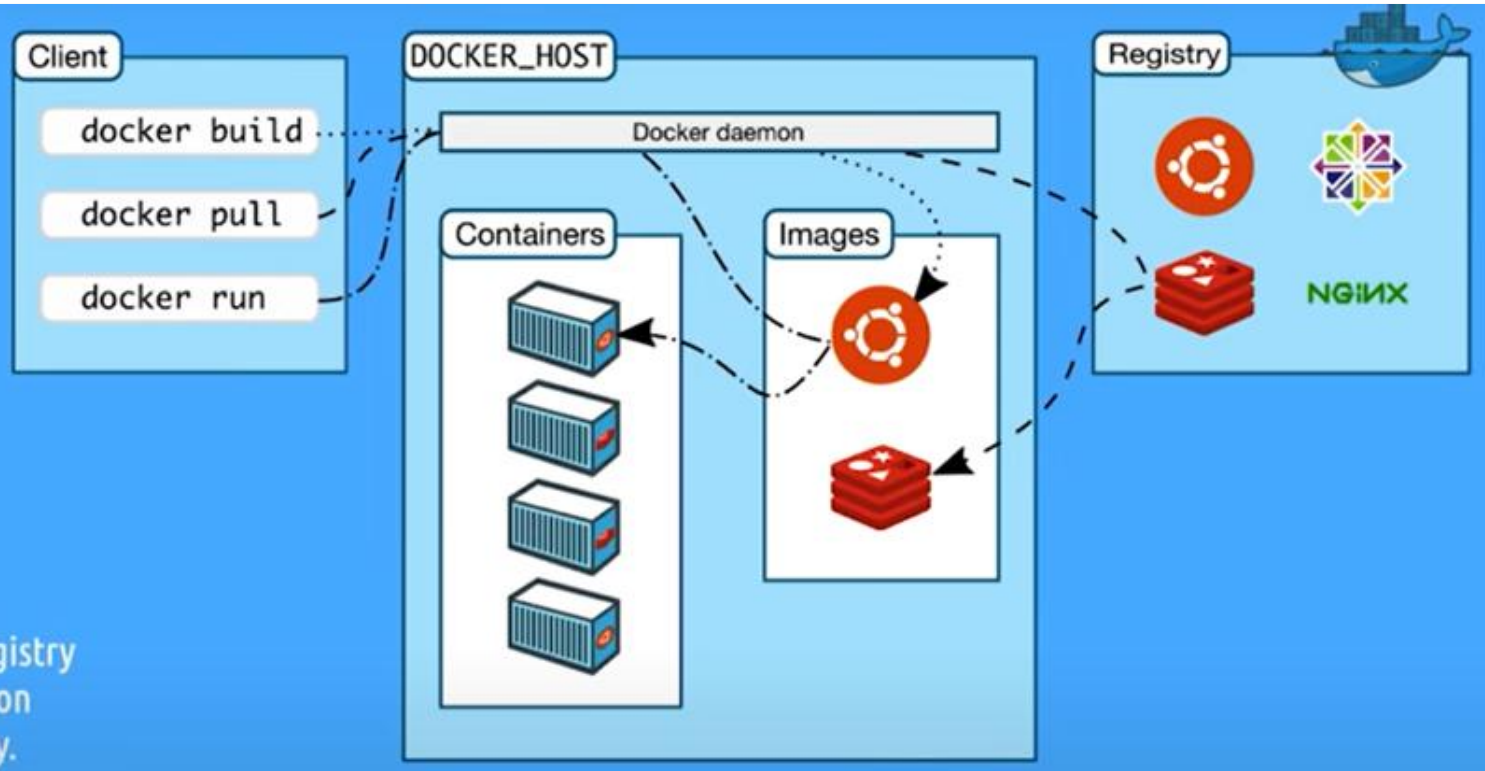
- An image is a read-only template with instructions for creating a Docker container. you may build, an image which is based on the Ubuntu image or SQL Server.

Container

- A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI.

Registry

- A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry.



Client

- The Docker client is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to dockerd, which carries them out. The docker command uses the Docker API.

Docker daemon

- The Docker daemon listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes.

Docker Architecture

Namespaces

- Docker uses a technology called `namespaces` to provide the isolated workspace called the container. When you run a container, Docker creates a set of namespaces for that container. These namespaces provide a layer of isolation. Each aspect of a container runs in a separate namespace and its access is limited to that namespace.

Docker File Example

- Dockerfile (Python HTTP Server for static website)

```
# Use an official Python image
FROM python:3.9-slim

# Set the working directory inside the container
WORKDIR /app

# Copy your website files into the container
COPY ./my-website/ .

# Expose port 8000 for the HTTP server
EXPOSE 8000

# Command to run the Python HTTP server
CMD ["python", "-m", "http.server", "8000"]
```

Example project structure

```
.
├── Dockerfile
└── my-website
    ├── index.html
    ├── style.css
    └── script.js
```

Docker File Example

1. Build the Docker image:

```bash
docker build -t simple-website .
```

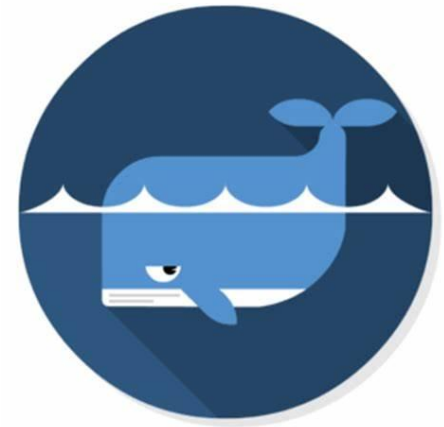
2. Run the container:

```bash
docker run -d -p 8000:8000 --name website simple-website
```

3. Open your browser and go to:

`http://localhost:8000`

Stop Docker → Cloud

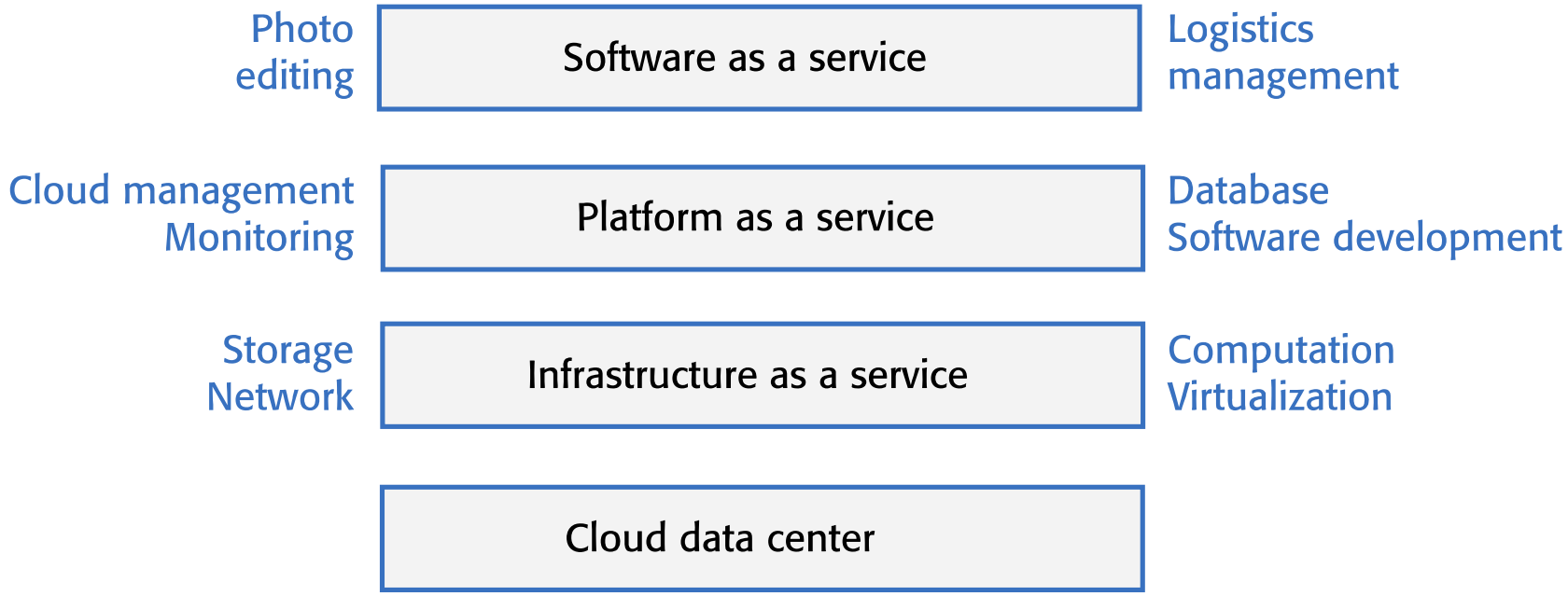


Everything as a service

The idea of a service that is rented rather than owned is fundamental to cloud computing.

Infrastructure as a service (IaaS)	• Cloud providers offer different kinds of infrastructure service such as a compute service, a network service and a storage service that you can use to implement virtual servers.
Platform as a service (PaaS)	• This is an intermediate level where you use libraries and frameworks provided by the cloud provider to implement your software. These provide access to a range of functions, including SQL and NoSQL databases.
Software as a service (SaaS)	• Your software product runs on the cloud and is accessed by users through a web browser or mobile app.

Figure 5.5 Everything as a service



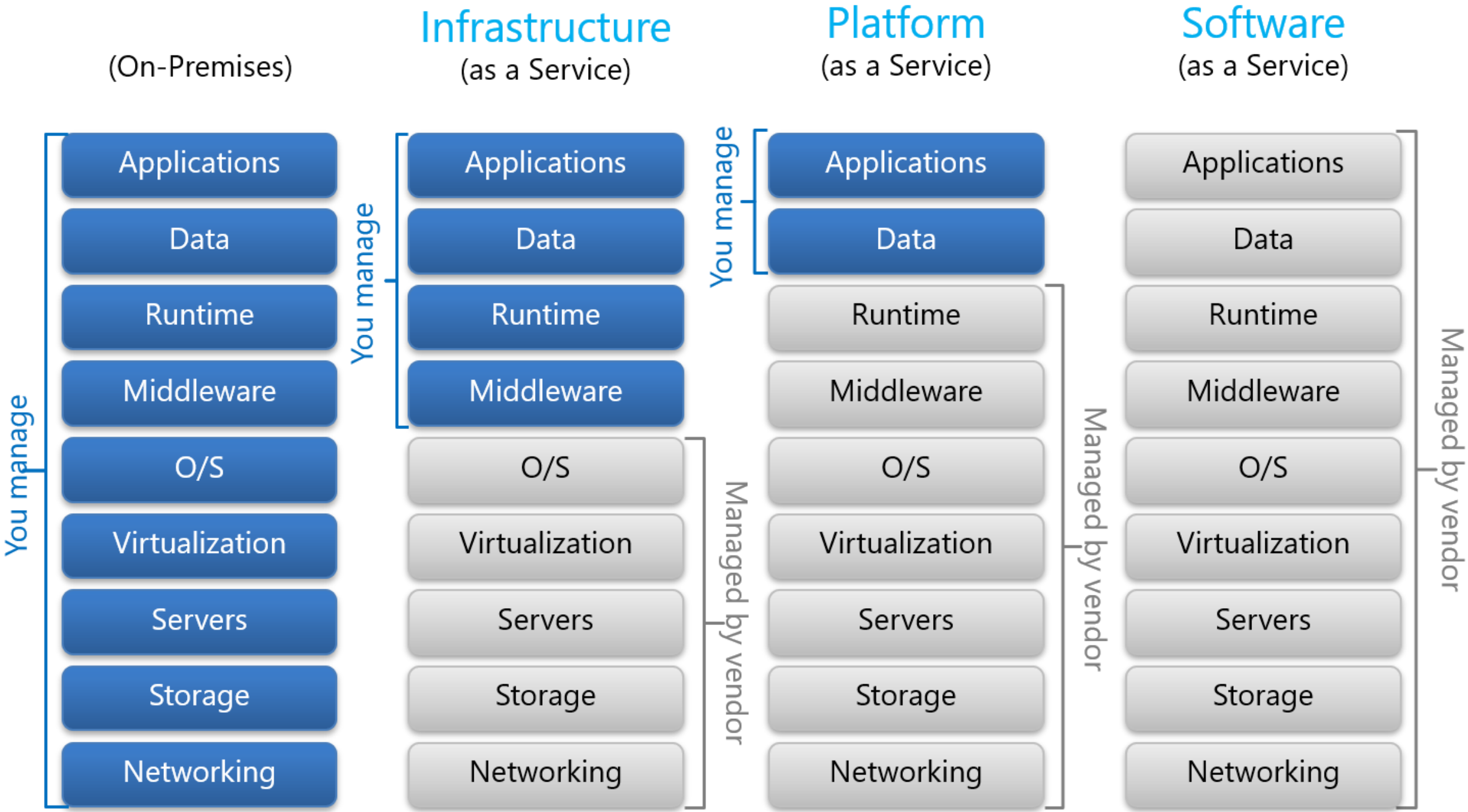
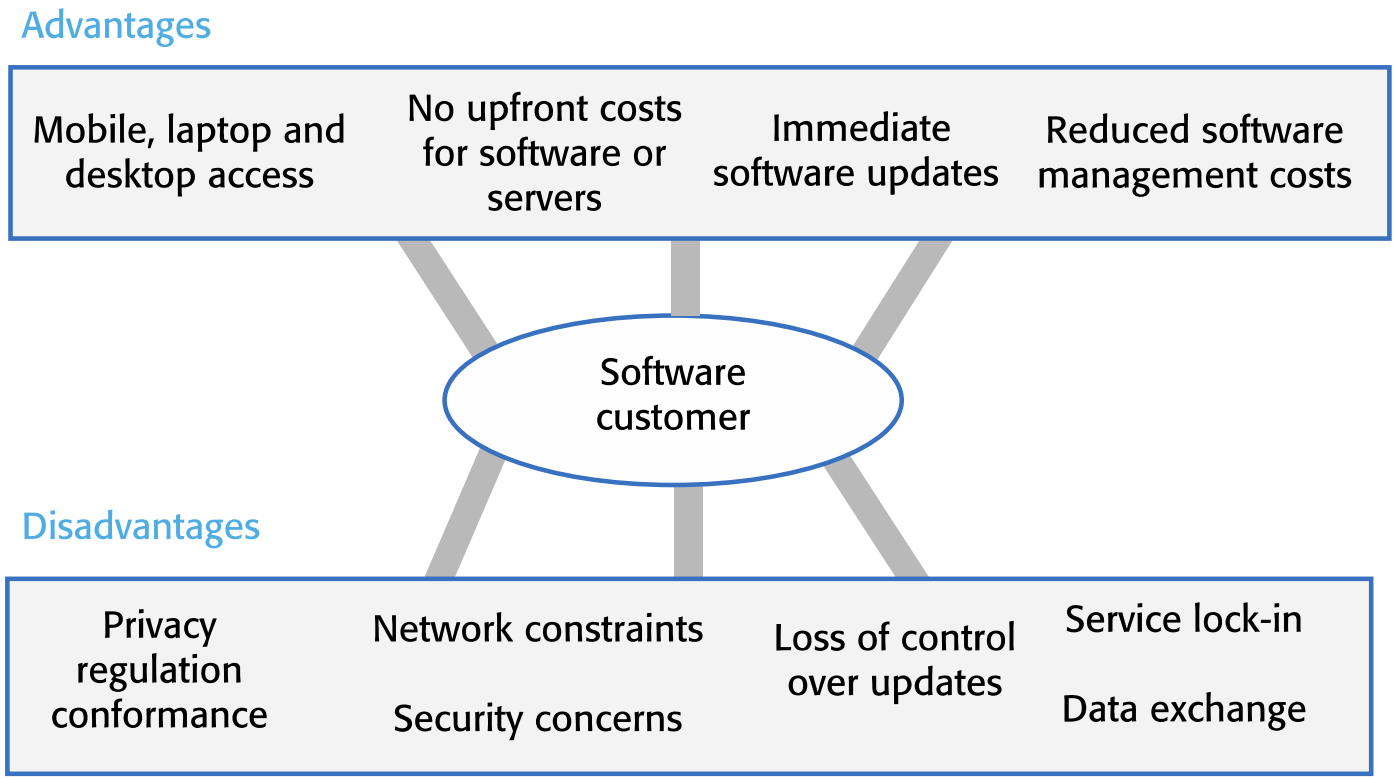

(On-Premises)
Infrastructure
(as a Service)
Platform
(as a Service)
Software
(as a Service)
You manage
Applications
Data
Runtime
Middleware
O/S
Virtualization
Servers
Storage
Networking
You manage
Applications
Data
Runtime
Middleware
O/S
Virtualization
Servers
Storage
Networking
Managed by vendor
You manage
Applications
Data
Runtime
Middleware
O/S
Virtualization
Servers
Storage
Networking
Managed by vendor
Applications
Data
Runtime
Middleware
O/S
Virtualization
Servers
Storage
Networking
Managed by vendor

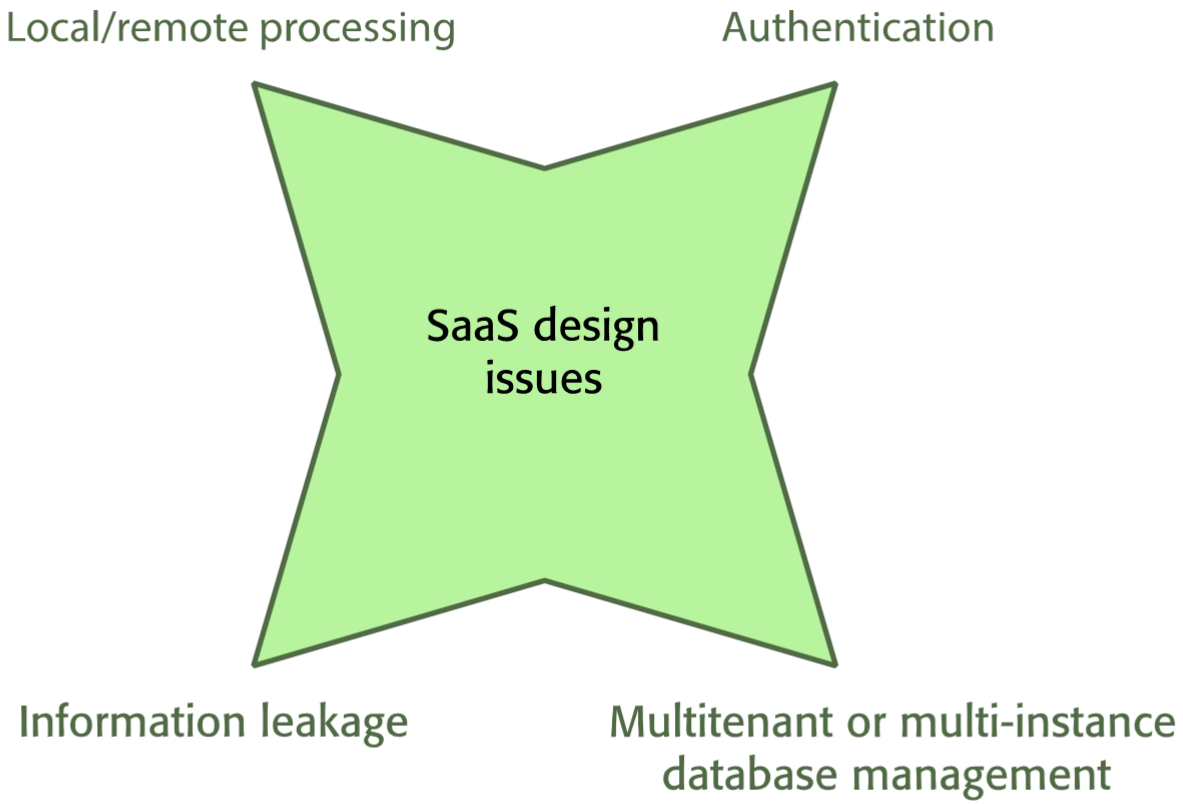
Figure 5.8 Advantages and disadvantages of SaaS for customers



Data storage and management issues for SaaS

Issue	Explanation
Regulation	Some countries, such as EU countries, have strict laws on the storage of personal information. These may be incompatible with the laws and regulations of the country where the SaaS provider is based. If an SaaS provider cannot guarantee that their storage locations conform to the laws of the customer's country, businesses may be reluctant to use their product.
Data transfer	If software use involves a lot of data transfer, the software response time may be limited by the network speed. This is a problem for individuals and smaller companies who can't afford to pay for very high speed network connections.
Data security	Companies dealing with sensitive information may be unwilling to hand over the control of their data to an external software provider. As we have seen from a number of high-profile cases, even large cloud providers have had security breaches. You can't assume that they always provide better security than the customer's own servers.
Data exchange	If you need to exchange data between a cloud service and other services or local software applications, this can be difficult unless the cloud service provides an API that is accessible for external use.

Figure 5.9 Design issues for software delivered as a service



SaaS design issues (1)

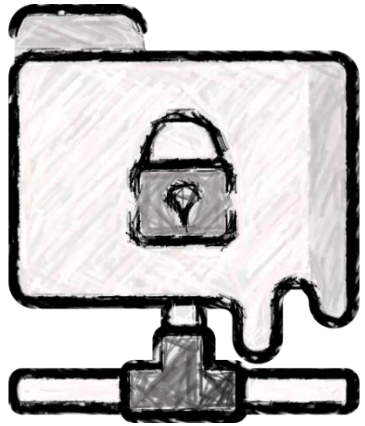
- Local/remote processing
  - A software product may be designed so that some features are executed locally in the user's browser or mobile app and some on a remote server.
  - Local execution reduces network traffic and so increases user response speed. This is useful when users have a slow network connection.
  - Local processing increases the electrical power needed to run the system.

- Authentication

  - If you set up your own authentication system, users have to remember another set of authentication credentials.
  - Many systems allow authentication using the user's Google, Facebook or LinkedIn credentials.
  - For business products, you may need to set up a federated authentication system, which delegates authentication to the business where the user works.

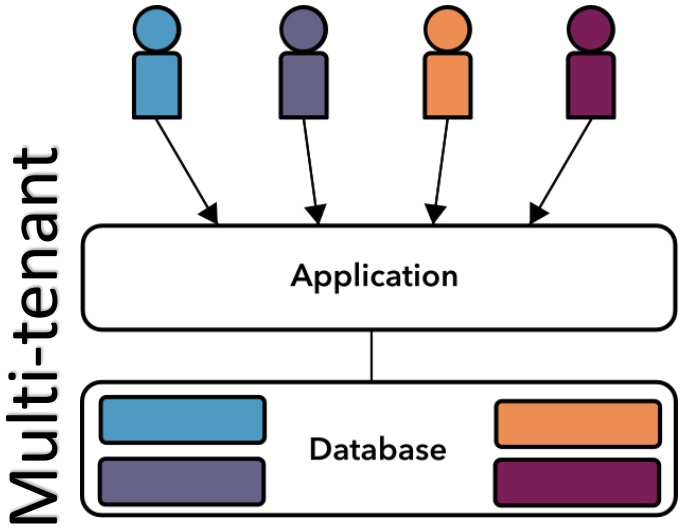
SaaS design issues (2)

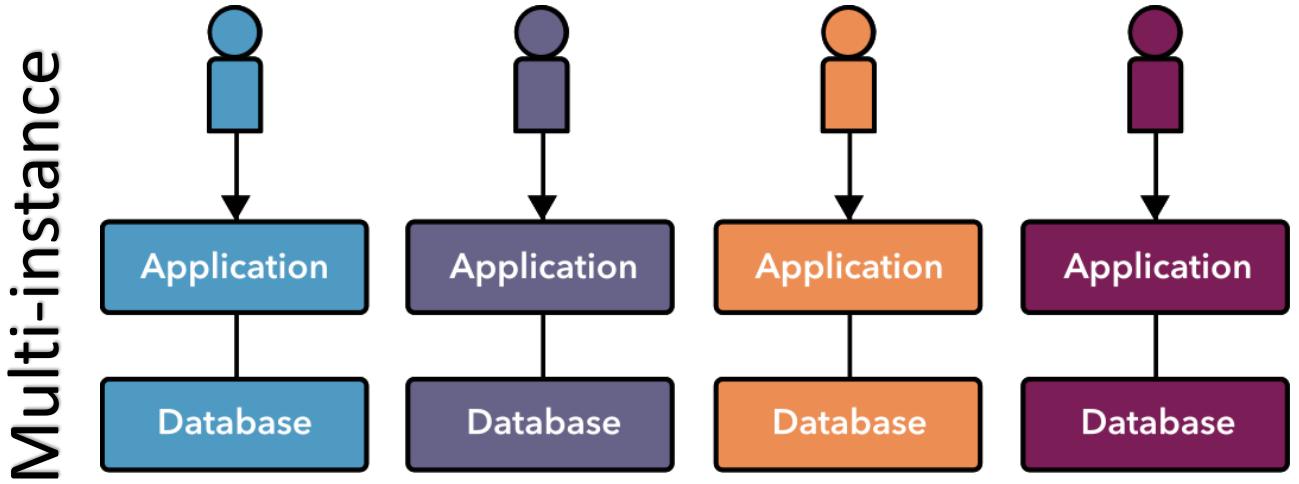
- Information leakage
  - If you have multiple users from multiple organizations, a security risk is that information leaks from one organization to another.
  - There are a number of different ways that this can happen, so you need to be very careful in designing your security system to avoid this.



SaaS design issues (3)

- Multi-tenant and multi-instance systems
  - In a multi-tenant system, all customers are served by a single instance of the system and a multitenant database.
  - In a multi-instance system, a separate copy of the system and database is made available for each user.





Multi-tenant systems

- A multi-tenant database is partitioned so that customer companies have their own space and can store and access their own data.
  - There is a single database schema, defined by the SaaS provider, that is shared by all of the system's users.
  - Items in the database are tagged with a tenant identifier, representing a company that has stored data in the system. The database access software uses this tenant identifier to provide 'logical isolation', which means that users seem to be working with their own database.

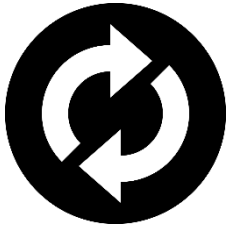
Figure 5.10 An example of a multi-tenant database

Stock management					
Tenant	Key	Item	Stock	Supplier	Ordered
T516	100	Widg 1	27	S13	2017/2/12
T632	100	Obj 1	5	S13	2017/1/11
T973	100	Thing 1	241	S13	2017/2/7
T516	110	Widg 2	14	S13	2017/2/2
T516	120	Widg 3	17	S13	2017/1/24
T973	100	Thing 2	132	S26	2017/2/12

Table 5.5 Advantages of multi-tenant databases

- *Resource utilization*
  The SaaS provider has control of all the resources used by the software and can optimize the software to make effective use of these resources.

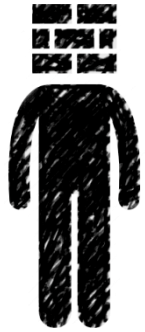
- *Security*
  Multitenant databases have to be designed for security because the data for all customers is held in the same database. They are, therefore, likely to have fewer security vulnerabilities than standard database products. Security management is simplified as there is only a single copy of the database software to be patched if a security vulnerability is discovered.

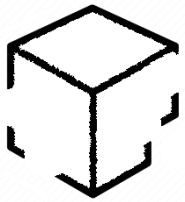


- *Update management*
  It is easier to update a single instance of software rather than multiple instances. Updates are delivered to all customers at the same time so all use the latest version of the software.

Table 5.5 Disadvantages of multi-tenant databases

- *Inflexibility*
  Customers must all use the same database schema with limited scope for adapting this schema to individual needs. I explain possible database adaptations later in this section.
- *Security*
  As data for all customers is maintained in the same database, then there is a theoretical possibility that data will leak from one customer to another. In fact, there are very few instances of this happening. More seriously, perhaps, if there is a database security breach then it affects all customers.
- *Complexity*
  Multitenant systems are usually more complex than multi-instance systems because of the need to manage many users. There is, therefore, an increased likelihood of bugs in the database software.





THANK YOU!