Cloud, Virtualization and Containers

Outline

- Cloud Infrastructure Recap
- Concept of Virtualization
- Hypervisors: Underlying Virtualization Technology
- Containerization: A New Paradigm
- Virtualization vs Containerization: Comparing Approaches
- Introduction to Docker: Containerization Made Easy

Cloud Infrastructure Recap

What are the components of cloud infrastructure?

- Hardware
- Virtualization
- Storage
- Network

Cloud Infrastructure Recap

- **Hardware Resources** –
  - Physical Hardware Distribution:
    - Positioned at Multiple Geographic Locations
  - Networking Equipment:
    - Facilitating Data Transmission
  - Storage Arrays:
    - Storing Data Securely
  - Central Element:
    - Servers Play a Vital Role

Cloud Infrastructure Recap

- Network Infrastructure:
  - Data Centers: Essential facilities hosting hardware resources, Interconnected by high-speed networks for seamless communication.
  - Varied network topologies employed within data centers, Distributes data traffic across multiple servers for optimization.
  - Robust security measures, including firewalls, ensure data protection and Interconnectivity enhances efficient data flow and component interaction.

Cloud Infrastructure Recap

- Storage Solutions:
  - Storage Types: Distinguishing Block, Object, and File Storage with Their Respective Applications
  - Data Replication: Implementing Strategies to Duplicate Data for Enhanced Availability and Disaster Recovery
  - Elastic Storage: Adapting Storage Resources to Fluctuating Demands through Scalability

Cloud Infrastructure Recap

- Virtualization –
  - Essential Role: Vital Component in Cloud Infrastructure
  - Abstraction of Resources: Separating Data Storage and Computing from Hardware
  - Enhanced Interaction: Facilitating Communication Between Users and Cloud Infrastructure
  - Common Practice: Frequently Applied to Data Storage and Computing Resources

Concept of Virtualization

- Let's explore the depths of virtualization!
  - Virtualization: An Efficient Process for Managing PCs and Hardware
  - Utilizes an Abstraction Layer Above Computer Hardware

Cont.

- What Constitutes the Hardware Components of a Single Computer?
  - Processors, memory, storage and more

- These Components Are Partitioned Across Multiple Virtual Computers, Known As:
  - Virtual machines (VMs)

Cont.

- Virtualization has become the prevailing norm in modern enterprise IT architecture.

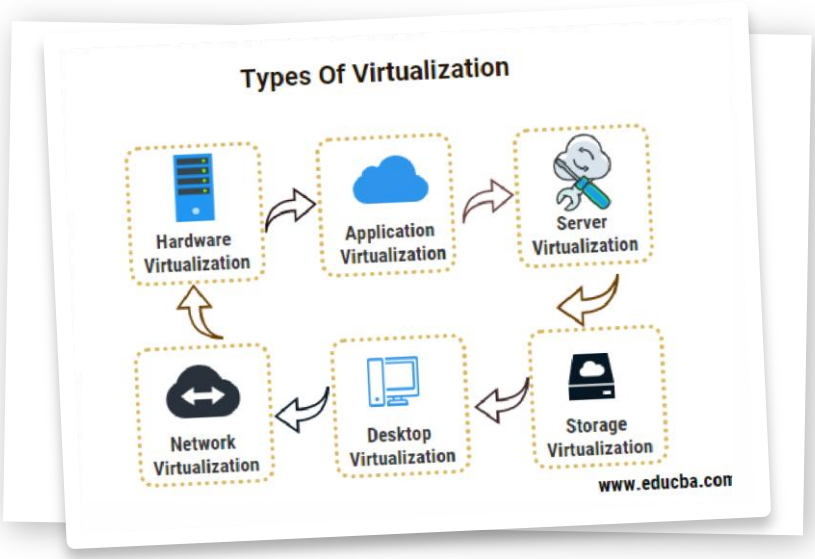
- Cloud resources exhibit cost-effectiveness as workloads expand, allowing seamless scalability.

Cont.

- Key Benefits:
  - Substantial reduction in IT expenditures.
  - Streamlined management processes.
  - Minimal downtime coupled with heightened resilience.
  - Expedited provisioning of resources.

Types of Virtualization

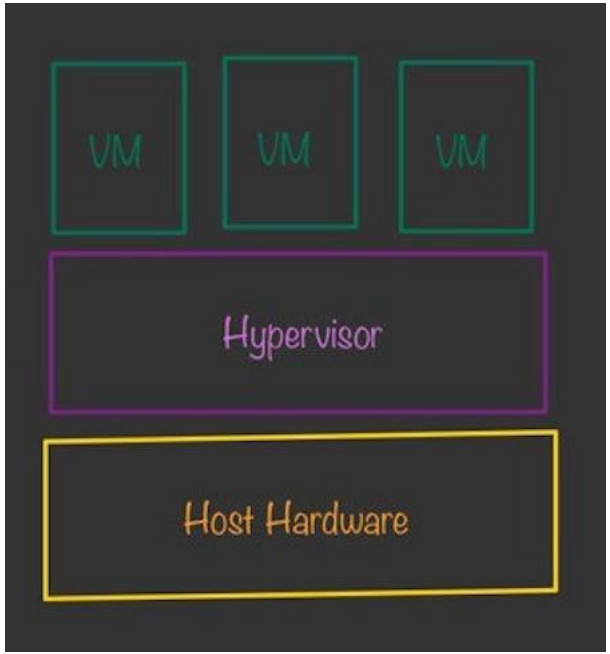
- Server Virtualization
- Desktop Virtualization
- Network Virtualization
- Storage Virtualization
- Application Virtualization



Virtual machines (VMs)

- A virtual envirnoment that simulate physical computation in software form,

- Server file of VM's configuration



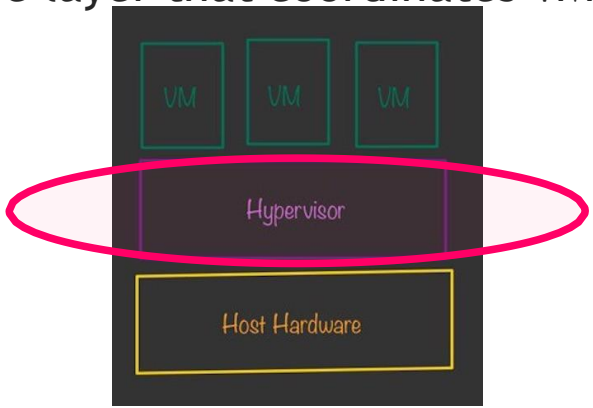
Cont.

- Types of VMs
  - MAC, Windows, Andriod, Linux and iOS
  - Java, Python
  - VMware software

- Advantadges of VM
  - Resources utilization
  - Scalibility
  - Portability
  - Flexibility &
  - Security

Cont.

- VM can’t interact with the hardware directly.
- So,how does virtualization works?
  - Hypervisior is refered as the software layer that coordinates VMs.
  - Serves as an interface.



Hypervisors

- Before hypervisors, most of the physical computers used to run on one OS at a time.
  - Downside : Wasted resources, because OS could not use all computer’s power.

Cont.

- Underlying Virtualization Technology
- Known as virtual machine monitor (VMM)
- Isolates VMs logically, assigning each VM slices of underlying computing power, memory, and storage.

Type 1 Hypervisor:

- **Type 1 Hypervisor**:

- Runs directly on top of physical hardware (usally server)
  - Bare metal
- Separate software product to create and manipulate VMs.

Types of Hypervisor

- **Type 1 Hypervisor**:

- VM resources are scheduled directly on top of hardware
- Which Type 1 hypervisor is associated with the penguin mascot?
  - Example – KVM
  - Others – ESXi and Vsphere, XEN

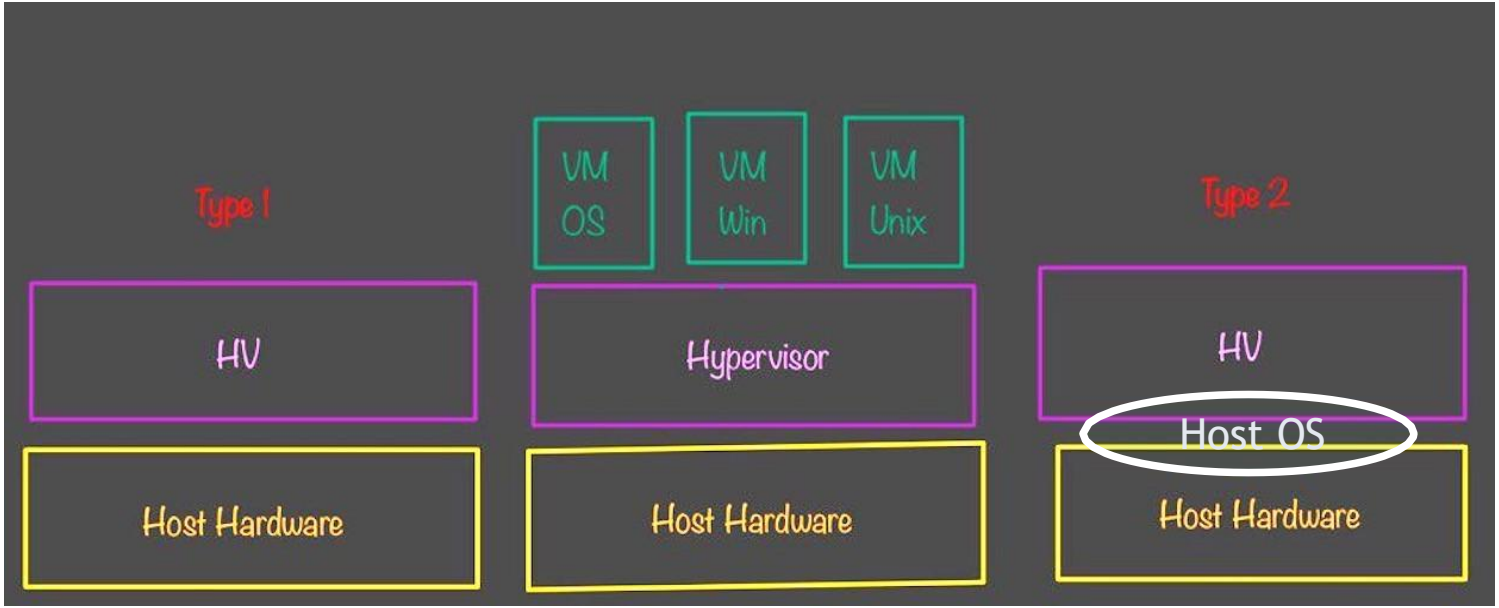
Cont.

- **Type 2 Hypervisior:**
  - Doesn’t run directly on the underlaying hardware
  - Hosted as an appliction on host OS
  - Single-user desktops
  - Examples – What virtualization software is well-known?
    - Oracle Virtual Box
    - VMware Fusion, Workstation

Cont.

- Create VMs manually and install guest OS
- Must access resources via the host OS
- Latency issues, security risks and affect performance

Revisiting the Distinctions



VMs vs Bare metal servers

Virtual Machines:	Bare Metal Servers:
• Created and managed by a hypervisor. • Operate in isolated environments. • Share physical server resources. • Flexible provisioning and scaling. • Hardware-independent and portable.	• Run directly on physical hardware. • Challenges in isolation. • Dedicated resources per server. • Limited flexibility in provisioning. • Tied to specific hardware.

Security

- Security Advantages of Virtualization:
  - Malware-infected VMs can be rolled back using snapshots.
  - Isolated environments limit the impact of breaches.

- Challenges:
  - Hypervisor attacks can affect multiple VMs.

Containers represent the next stage beyond virtualization!

Containerization: A New Paradigm

- Containers hold undeniable importance in modern computing!
  - Throughout our exploration, we'll cover topics such as:
    - The reasons driving the adoption of containers.
    - How they enhance our processes.
    - The diverse domains that leverage container technology.

Cont.

- Leveraging Containerization at Google:
  - Addressing issues encountered with VM models.
- Streamlined Operating System Usage:
  - Containers avoid the necessity for a complete OS.
- Swift Start-Up and Unparalleled Portability:
  - Containers boast quick launch times and exceptional mobility.
- Effortless Migration Across Environments:
  - Moving containers between VMs, bare metal, and the cloud is a smooth process.

Containerization

- What is Containerization?
  - Packaging of a software code with just the operating system (OS), libs and dependencies
  - Creating a single lightweight executable – called **Container**
  - Resource efficient and portability

Cont.

- What is Containerization?
  - Defacto comute units of modern cloud-native applications
  - Allows developers to create and deploy application faster and securely

- Important - Allows applications to be “written once and run anywhere.”

Cont.

- What is container?
  - Often referred as “lightweight”
  - Standardized packaging for software dependencies
  - Isolate apps from each other
  - Works for all major Linux distributions, MacOS, Windows

Application Containerization

- Containers bundle an application, along with its configuration files, libraries, and dependencies, into a single executable package.
- Unlike traditional approaches, containers don't include a copy of the operating system.
- Runtime engines (e.g., Docker runtime) are used to facilitate container sharing of the host OS.

Cont.

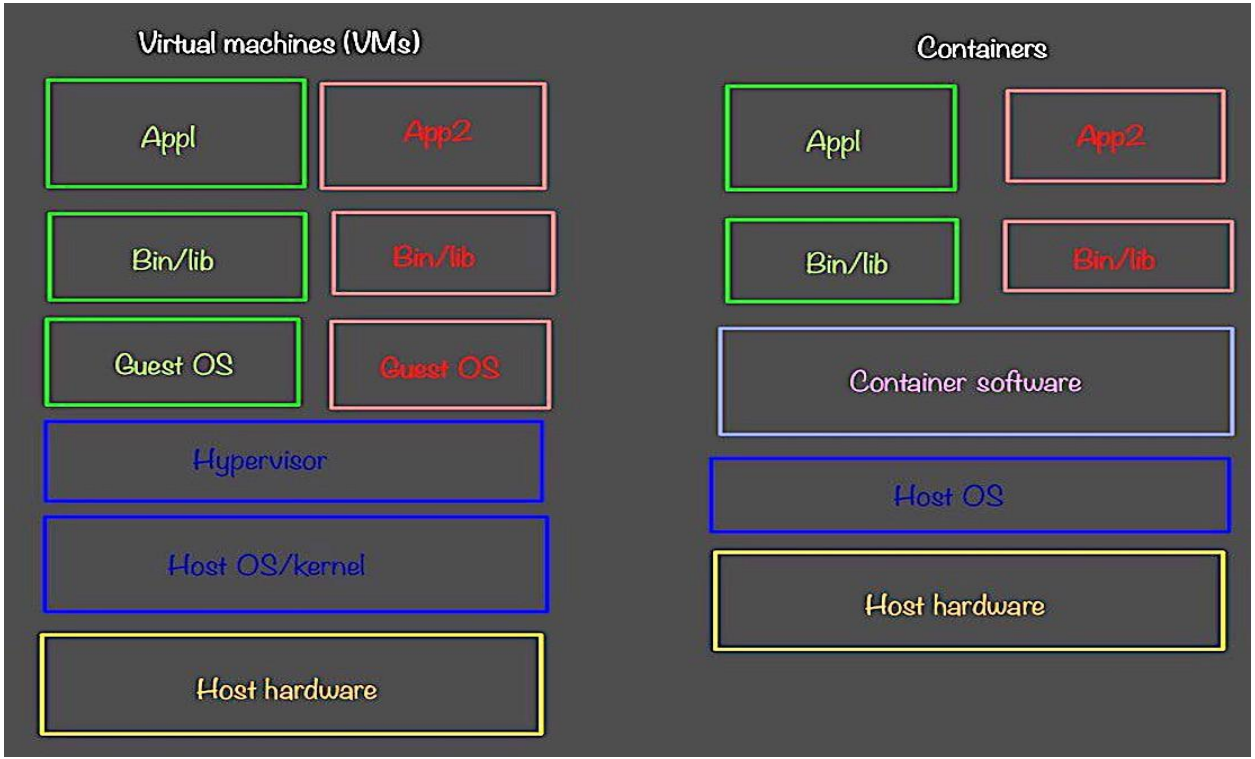
- Shared Resources and Efficiency:
  - Common bins and libraries can be shared among multiple containers.
  - Containers are more efficient in terms of capacity and start-up time compared to running a full OS with each application.
  - Smaller footprint and faster start-up lead to higher server efficiency.

Cont.

- Isolation and Security:
  - Containers isolate applications, preventing the spread of malicious code from one container to another or to the host system.

- Portability:
  - Containerized applications are portable and can run consistently across various platforms, clouds, and OS types.
  - Easily transferable between desktop, VMs, different OSes, and cloud environments.

Cont.


Virtual machines (VMs)
Appl
App2
Bin/lib
Bin/lib
Guest OS
Guest OS
Hypervisor
Host OS/kernel
Host hardware
Containers
Appl
App2
Bin/lib
Bin/lib
Container software
Host OS
Host hardware

Linux Containers (LXC)

- Linux and Container Technology:
  - Linux is an open-source operating system.
  - It features built-in container technology.
- Linux Containers:
  - Linux containers are self-contained environments.
  - They enable multiple Linux-based applications on a single host.

LXC

- Deployment for Data-Intensive Applications
  - Efficiently handle writing or reading large amounts of data.

- Linux Namespace:
  - Containers utilize Linux namespaces to allocate functionalities.
  - Namespaces isolate resources such as networking, process IDs, etc.

Virtualization vs Containerzation!

- Containers and virtual machines (VMs) are both used to run multiple software types in one environment.

- Container technology offers additional benefits beyond virtualization, becoming a preferred choice for IT professionals.

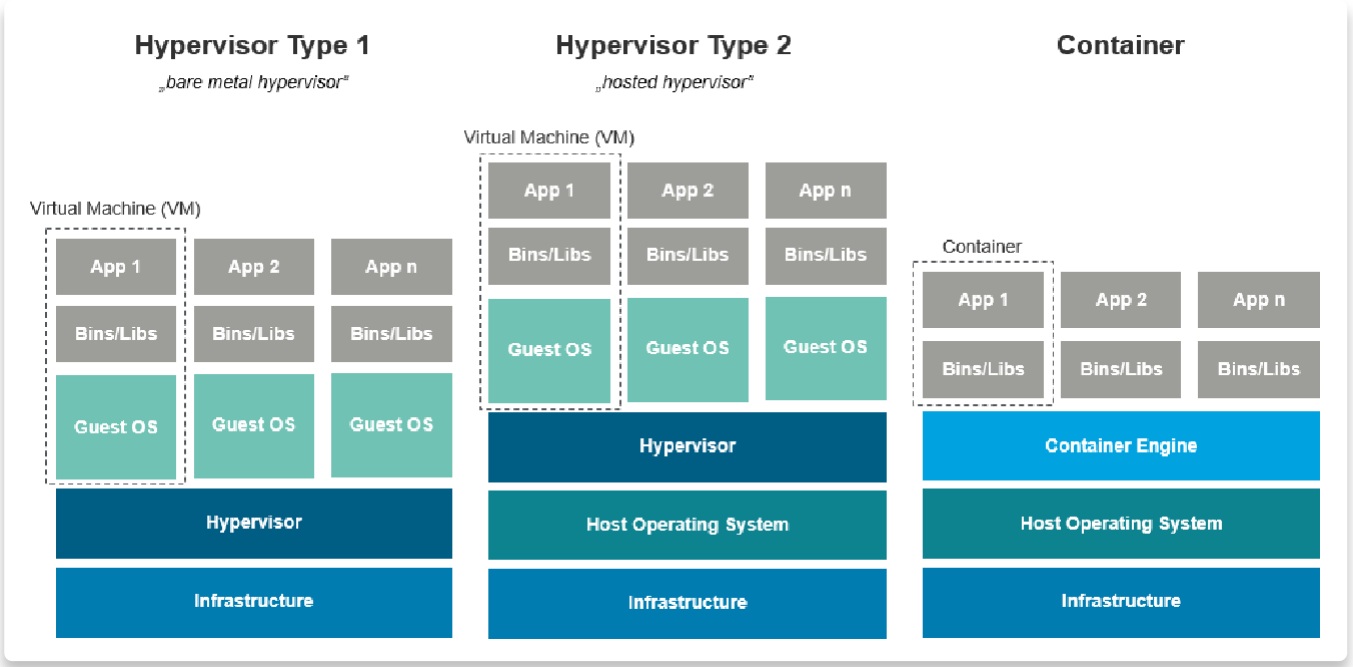
Cont.

- Virtualization Efficiency:
  - Virtualization runs multiple operating systems and applications on a single physical computer.
  - Each application and its dependencies are packaged as a VM, leading to cost savings.

Cont.

- Containerization Efficiency:
  - Containers bundle application code, configuration, and dependencies into an executable package.
  - Unlike VMs, containers don't include a copy of the OS but share the host's OS kernel.
  - Container runtime engine enables sharing the OS among all containers on the system.

Hypervisor Type1 vs Type2 vs Container

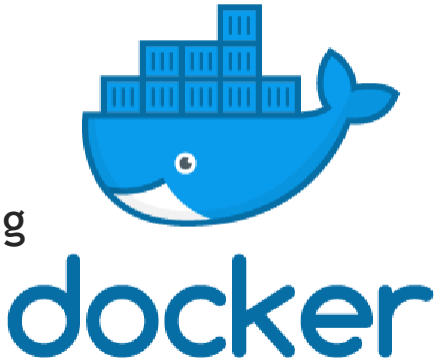


VMs vs Containers

	VIRTUAL MACHINES	CONTAINERS
Guest Operating System	Each VM runs on top of hypervisor and kernel loaded into its own memory region.	All guests share the same kernel. Kernel image is loaded in its physical memory.
Performance Efficiency	Suffers light overhead as the machine instructions get translated from guest to host OS.	Almost native performance as compared to the underlying host OS.
Security	Complete Isolation.	Isolation using namespaces.
Storage	Takes more storage.	Take less storage as the base OS is shared.
Isolation	Higher level of Isolation. Need special techniques for file sharing.	Subdirectories can be transparently mounted and can be share.
Networking	Can be linked to virtual or physical switches. Hypervisor has its own buffers to improve I/O performance.	Leverage standards like IPC mechanisms such as signals, pipes, sockets, etc. Advanced features like NIC are not available.
Bootup Time	Take a few minutes to boot.	Boot up in a few seconds.

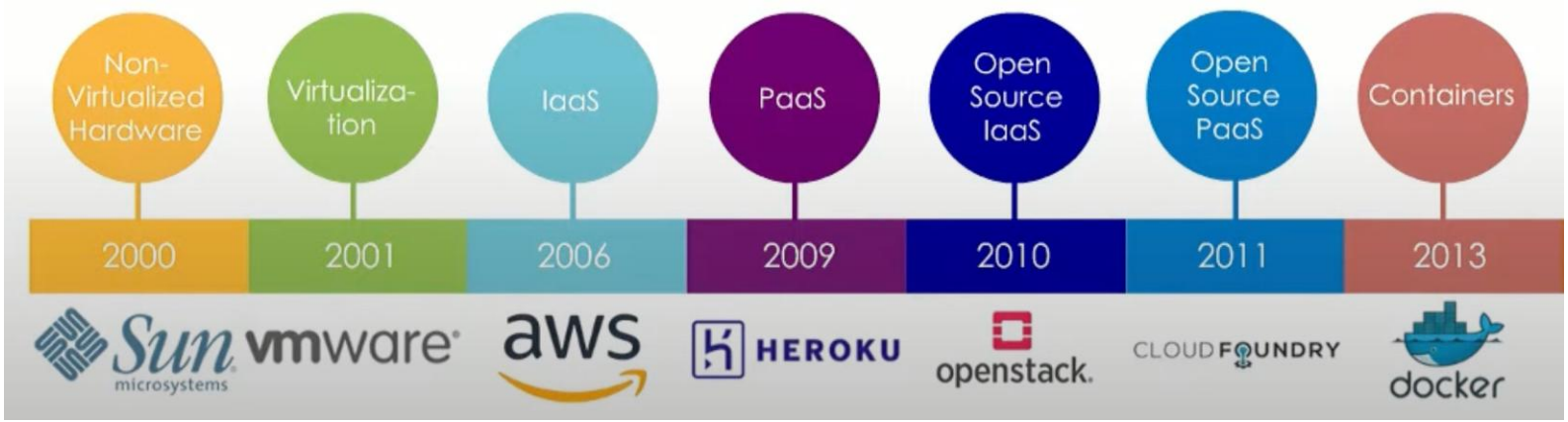
Introduction to Docker!

- Docker is a game-changing platform for modern application deployment.
- It takes containerization to the next level, making app management a breeze.



Docker!

- Docker is so popular today that “Docker” and “containers” are used interchangeably.



Cont.

- dotCloud started Docker as Platform as a service (PaaS).
- dotCloud leveraged LXC behind

(original logo) (new logo)

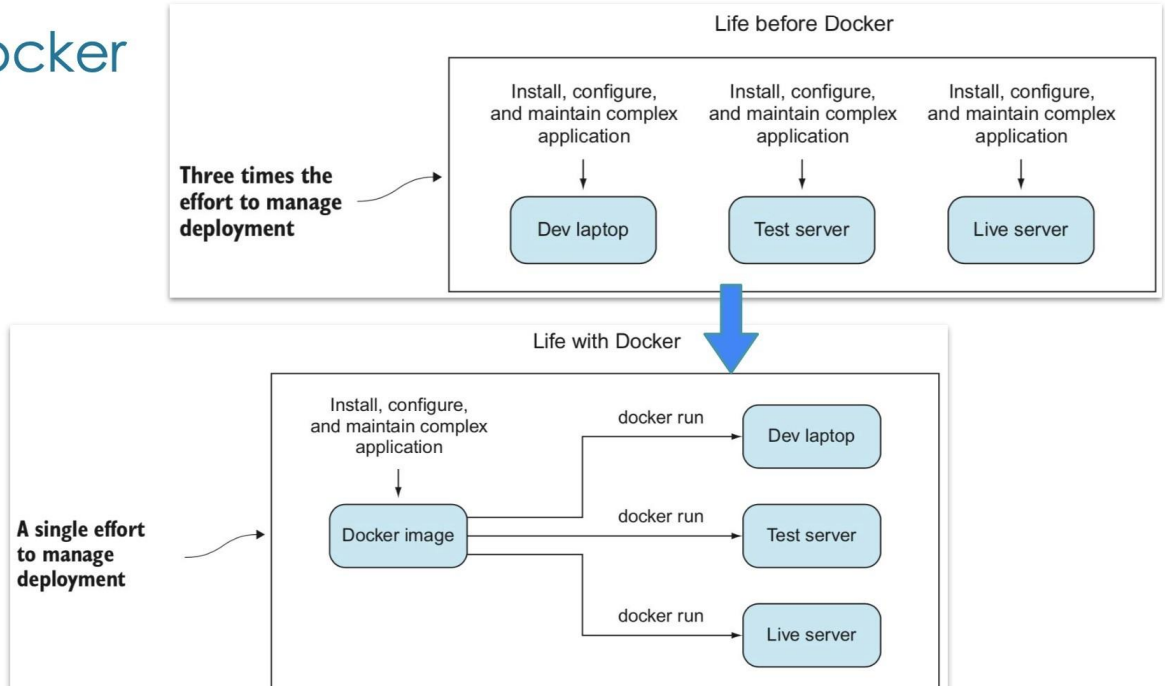
- The word “Docker” comes from a British colloquialism meaning dock worker - somebody who loads and unloads ships.

Cont.

- Why Docker is so important?
  - Docker provides what is called an *abstraction*
  - Abstraction – Allows work with complicated things in simplified terms.
  - Software community adaption of container and Docker
    - Amazon, Microsoft, and Google
  - Cross platform & open way

Cont.

Docker


Life before Docker
Install, configure, and maintain complex application
Install, configure, and maintain complex application
Install, configure, and maintain complex application
Three times the effort to manage deployment
Dev laptop
Test server
Live server
Life with Docker
Install, configure, and maintain complex application
docker run
Dev laptop
A single effort to manage deployment
Docker image
docker run
Test server
docker run
Live server

Cont.

- What is Docker?
  - Open-source platform
  - Enables developers to build, deploy, run, update and manage containers
  - Containers - Standardized, executable components that combine application source code with the operating system (OS) libraries and dependencies required to run that code in any environment.

Cont.

- Advantadges of Docker –
  - Compared to LXC, Docker offers:
    - Improved and seamless container portability
    - Even lighter weight and more granular updates

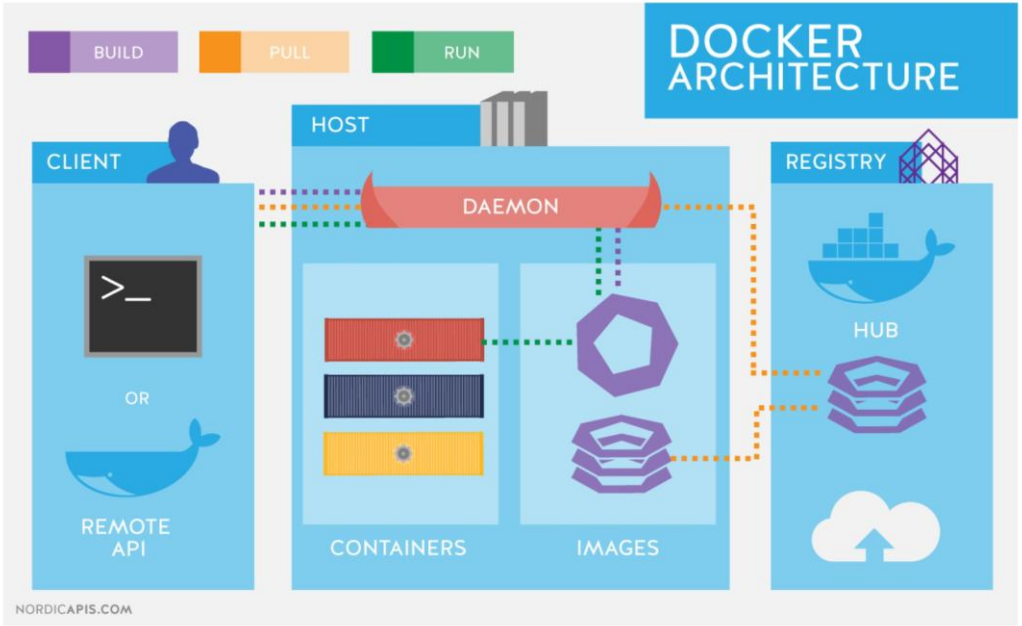
Cont.

- Advantadges of Docker –
  - Automated container creation
  - Container versioning
  - Container reuse
  - Shared container libraries

Cont.

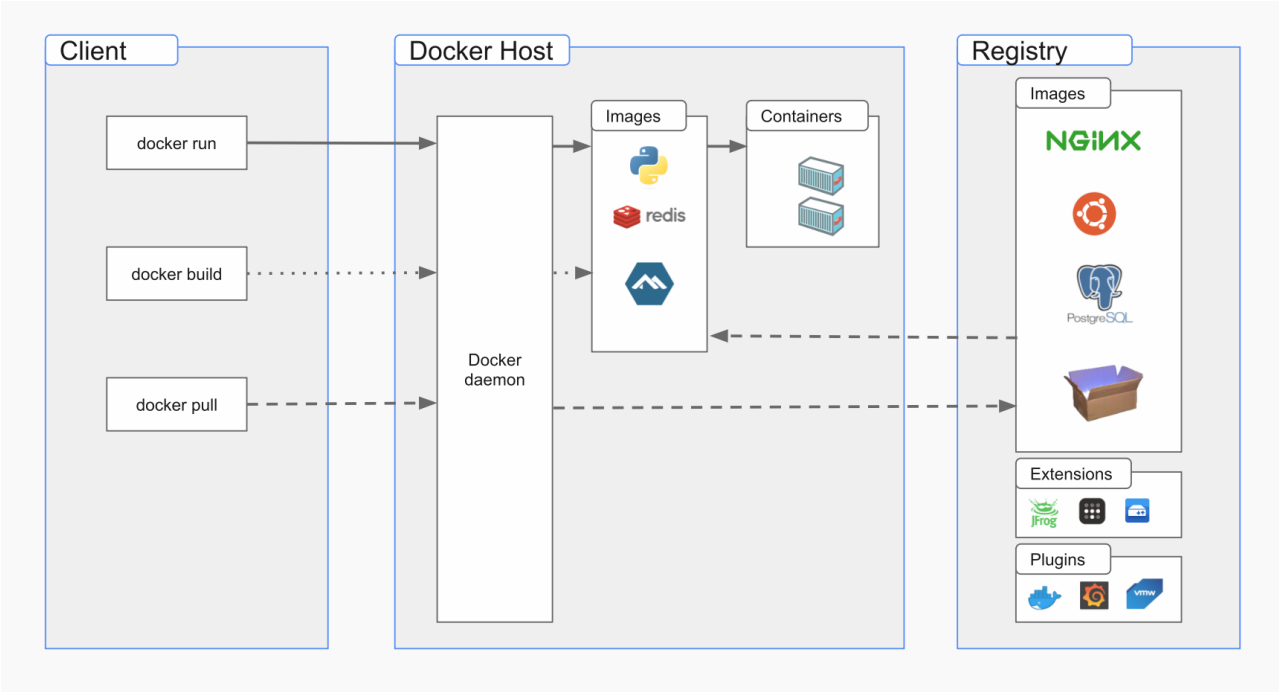
- Docker on different OS

Docker Architecture



Docker object serves a specific purpose in managing and deploying containerized applications

Docker Architecture



Docker Objects

- Image:
  - Immutable snapshot of an application and its dependencies.
  - Used to create containers.
  - Composed of layers, enhancing efficiency through caching.

Cont.

- Container:
  - Running instance of an image.
  - Isolated environment for an application.
  - Lightweight and portable.

Cont.

- Network:
  - Virtual network for container communication.
  - Enables containers to connect and communicate securely.
  - Different network drivers for various use cases.

Cont.

- Volume:
  - Persistent data storage outside containers.
  - Enables sharing data between containers and host.
  - Preserves data even when containers are removed.

Cont.

- Service:
  - Definition of how containers should behave in a specific environment.
  - Often used in orchestration tools.
  - Enables scaling, load balancing, and more.

Cont.

- Stack:
  - Collection of services that make up an application.
  - Allows managing multi-service applications together.
  - Orchestrated as a single unit.

Docker Engine.

- Docker Engine is an open-source containerization technology for building and containerizing your applications. Docker Engine acts as a client-server application with:

  - A server with a long-running daemon process dockerd.

Cont.

- APIs which specify interfaces that programs can use to talk to and instruct the Docker daemon.

- A command line interface (CLI) client docker.

- The CLI uses Docker APIs to control or interact with the Docker daemon through scripting or direct CLI commands.

Docker Images.

- Docker images encompass executable app code, tools, libraries, and dependencies.
- Running a Docker image creates one or more container instances.
- Images are built from scratch or pulled from repositories.

Cont.

- Multiple images stem from a base image, sharing stack similarities.
- Docker images consist of layers representing versions; new layers form top versions.
- Previous layers allow rollbacks or are reused in other projects.

Cont.

- Containers created from images have their own container layer.
- Container layer stores container-specific changes and exists during runtime.
- This iterative process optimizes efficiency by leveraging common stack for multiple instances.

Docker Containers.

- Docker containers are the live, running instances of Docker images.
- While Docker images are read-only files, containers are life, ephemeral, executable content.
- Users can interact with them, and administrators can adjust their settings and conditions using Docker commands.

Container vs Image

- Docker container vs docker image –
  - Docker containers serve as isolated runtime environments for application development.
  - Containers create, run, and deploy apps, separated from underlying hardware.
  - Containers utilize shared kernel, virtualize OS, and are lightweight.

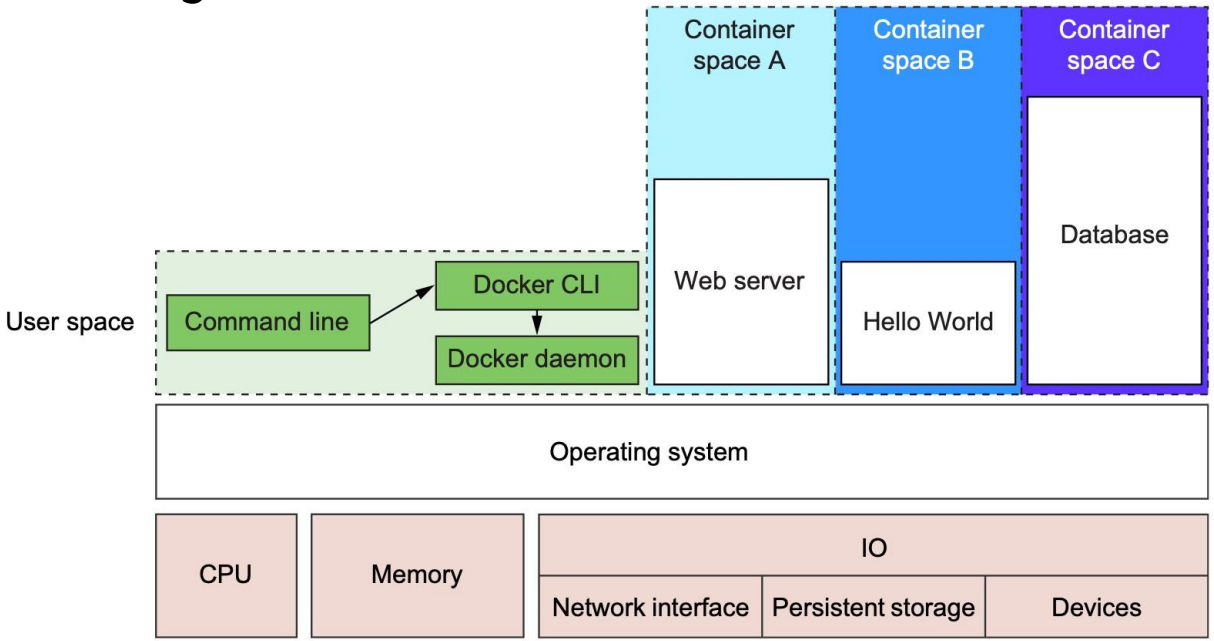
Cont.

- Docker images are snapshots of containers at specific times.
- Docker images are immutable and can be duplicated, shared, or deleted.
- Immutability benefits testing by maintaining unchanged images.
- Containers depend on images for constructing runtime environments and running apps.

Docker is running!

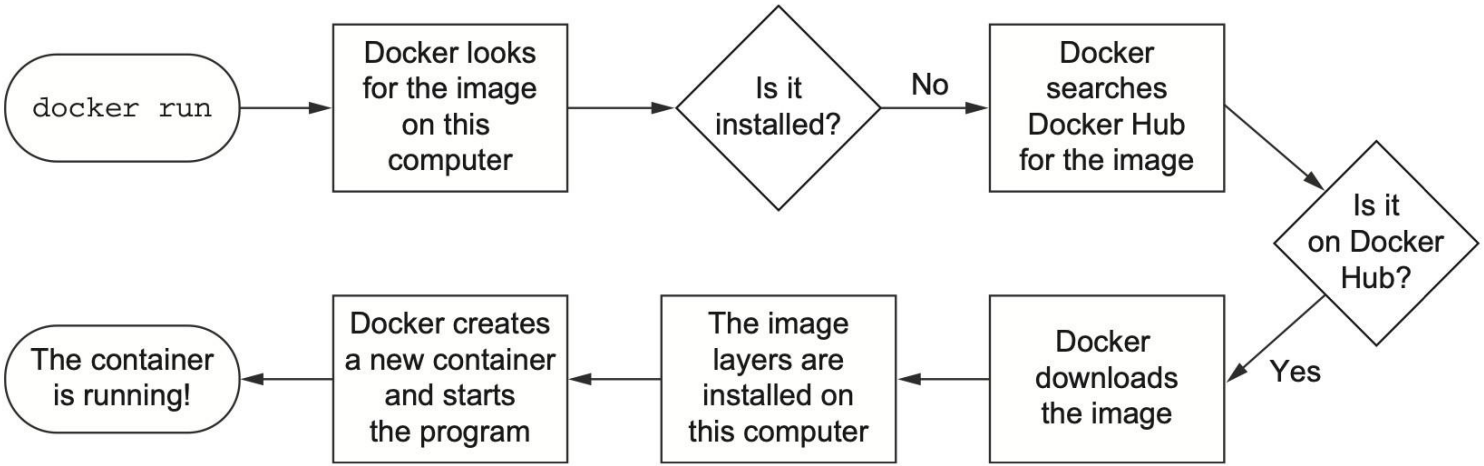
Cont.

- Docker running!



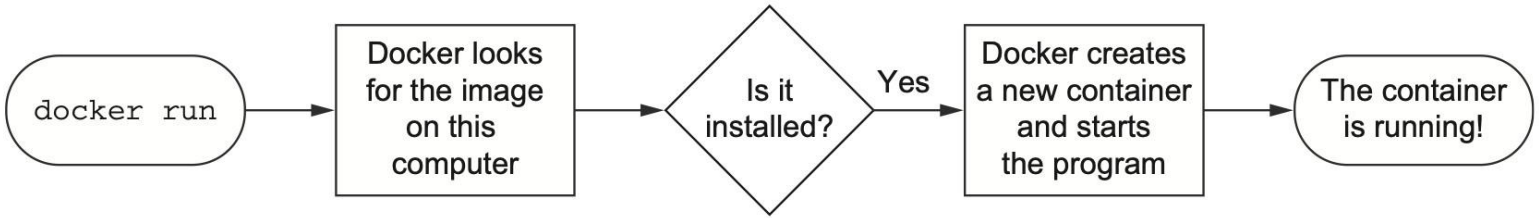
Cont.

- Docker run command

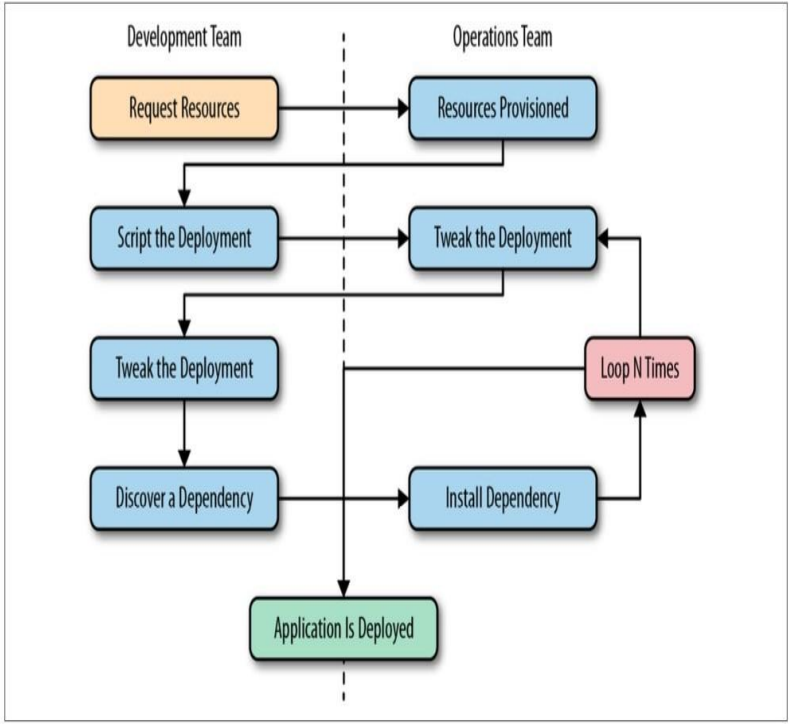


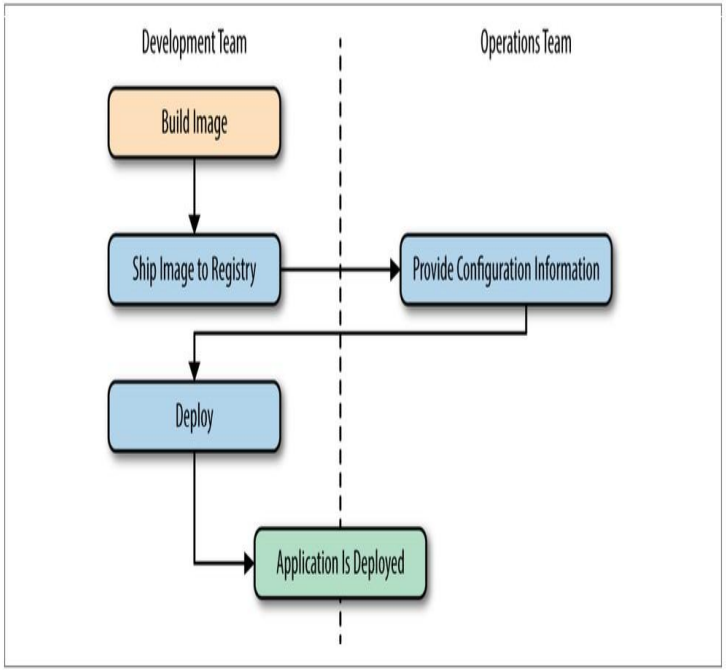
Cont.

- Docker run second time!



Docker Deployment





Docker Tools

- Docker Desktop:
  - Application for Mac/Windows.
  - Includes Docker Engine, CLI, Compose, Kubernetes, and more.
  - Provides access to Docker Hub.

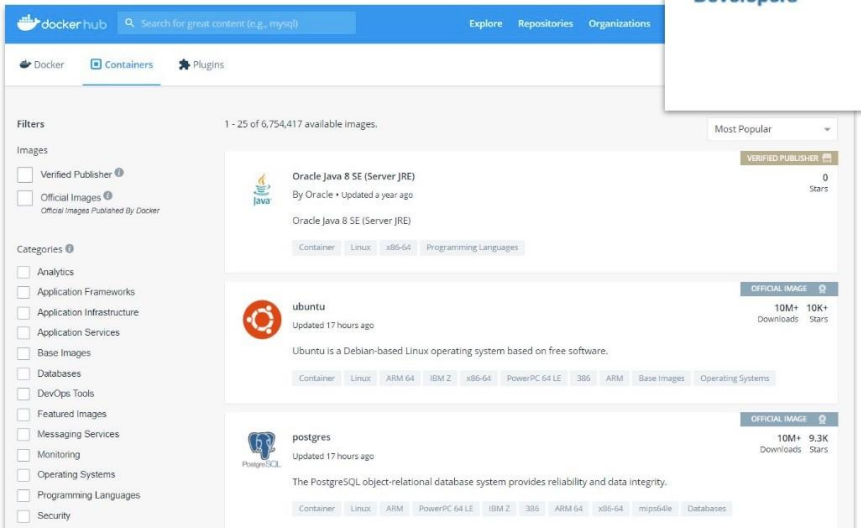
Cont.

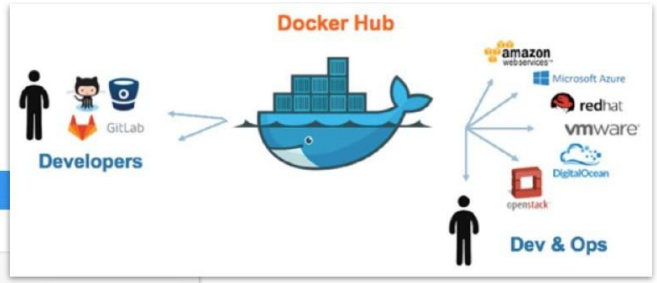
- Docker Daemon:
  - Manages Docker images.
  - Responds to client commands.
  - Control center of Docker implementation.
  - Runs on Docker host.

Cont.

- Docker Registry:
  - Open-source storage and distribution system.
  - Tracks image versions in repositories.
  - Uses tagging for identification.
  - Utilizes git, a version control tool.

Docker Hub.

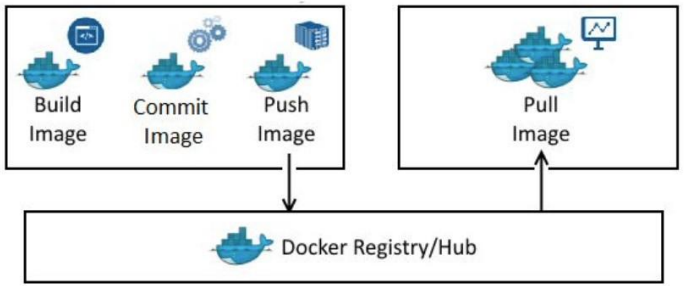

Docker Hub
docker hub
Search for great content (e.g., mysql)
Explore
Repositories
Organizations
Docker
Containers
Plugins
Filters
1 - 25 of 6,754,417 available images.
Most Popular
Images
Verified Publisher
Official Images
Official Images Published By Docker
Categories
Analytics
Application Frameworks
Application Infrastructure
Application Services
Base Images
Databases
DevOps Tools
Featured Images
Messaging Services
Monitoring
Operating Systems
Programming Languages
Security
VERIFIED PUBLISHER
Oracle Java 8 SE (Server JRE)
By Oracle • Updated a year ago
Oracle Java 8 SE (Server JRE)
0
Stars
Container
Linux
x86-64
Programming Languages
OFFICIAL IMAGE
ubuntu
Updated 17 hours ago
10M+
Downloads
10K+
Stars
Ubuntu is a Debian-based Linux operating system based on free software.
Container
Linux
ARM 64
IBM Z
x86-64
PowerPC 64 LE
386
ARM
Base Images
Operating Systems
OFFICIAL IMAGE
postgres
Updated 17 hours ago
10M+
Downloads
9.3K
Stars
The PostgreSQL object-relational database system provides reliability and data integrity.
Container
Linux
ARM
PowerPC 64 LE
IBM Z
386
ARM 64
x86-64
mips64le
Databases


Docker Hub
GitLab
Developers
amazon web services
Microsoft Azure
redhat
vmware
DigitalOcean
openstack
Dev & Ops

Cont.

- Sevice provided by Docker for finding and sharing container images

- **Repositories** allow sharing container images with the Docker community



- Container images can be pushed to a repository or pulled from it
  - **Official images** (provided by Docker)
    - Clear documentation, best practices, design for most common use cases, scanned for security vulnerabilities
  - **Publisher images** (provided by external vendors)

Dockerfile.

- Docker containers begin with a text file guiding image construction.
- Docker File automates image creation using CLI instructions.
- Docker commands are extensive yet standardized, ensuring consistent operations.
- Docker builds images automatically by reading the instructions from a Dockerfile.

Cont.

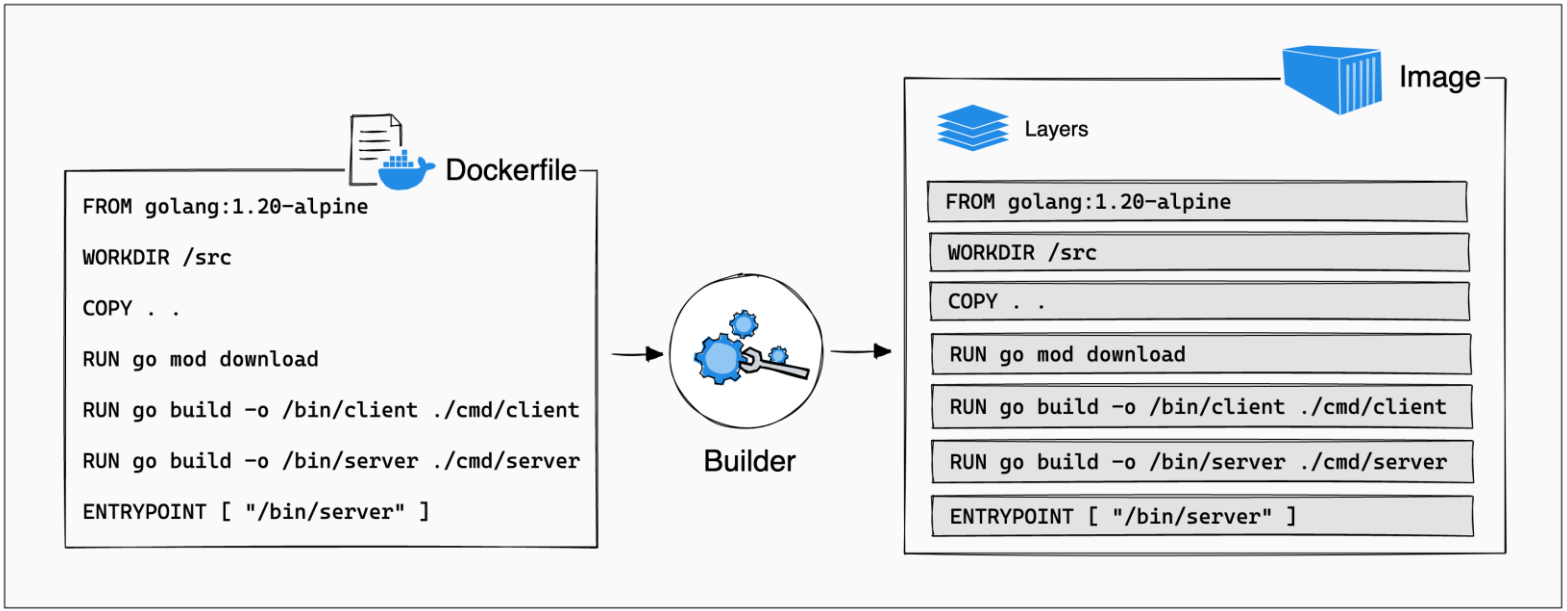
- # syntax=docker/dockerfile:1

- FROM ubuntu:22.04
- COPY . /app
- RUN make /app
- CMD python /app/app.py

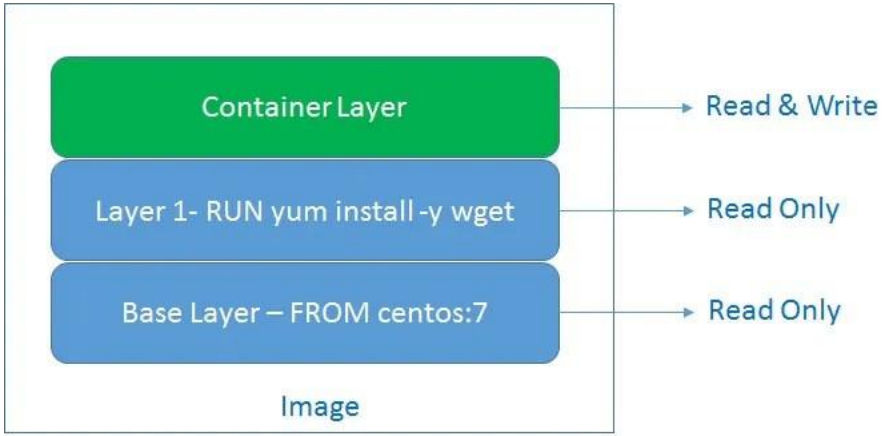
Cont.

- Each instruction creates one layer:
  - FROM creates a layer from the ubuntu:22.04 Docker image.
  - COPY adds files from your Docker client's current directory.
  - RUN builds your application with make.
  - CMD specifies what command to run within the container.
- Adding new writable layer is called as the container layer

Docker Layers



Cont.

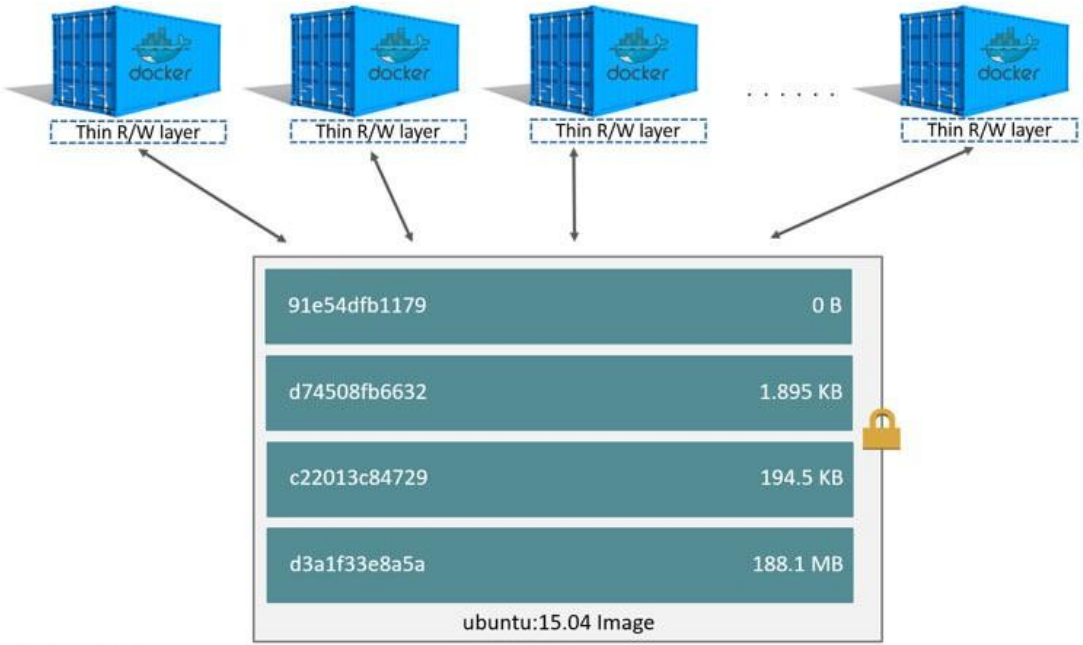


- Image has many layers,
- Only one read-write layers,
- Changes are made to editable to R/W layers, not to underlaying layers,
- CoW (Copy on Write) in storage drivers.

Cont.

- Advantages of using Docker layers –
  - Good storage management
  - Faster builds
  - Faster deployments
  - Sharing across multiple containers
  - Enhanced scalability

Cont.


Thin R/W layer
Thin R/W layer
Thin R/W layer
Thin R/W layer
91e54dfb1179 0 B
d74508fb6632 1.895 KB
c22013c84729 194.5 KB
d3a1f33e8a5a 188.1 MB
ubuntu:15.04 Image

Building and Managing Images

- **Anatomy of Docker Images:**
  - Layered Architecture: Images composed of stackable layers.
  - Image Construction: Steps to create an image defined in Dockerfile.
  - Effective Docker files: Writing clear and concise instructions.

Cont.

- **Best Practices for Optimizing Images:**
  - Layer Caching: Reuse unchanged layers for faster builds.
  - Efficiency: Reduce redundancy in layers for streamlined images.
  - Minimizing Image Size: Eliminate unnecessary files and dependencies.
  - Image Security: Ensuring images are free from vulnerabilities.

Working with Containers

- Containerization Essentials:
  - Running Containers: Initiate containers with docker run command.
  - Container Lifecycle: Manage container states – start, pause, stop.

Cont.

- Advanced Container Configuration:
  - Port Mapping: Associate container ports with host ports.
  - Volume Mounting: Connect containers to persistent storage.
  - Environment Variables: Set runtime configuration inside containers.

Cont.

- Inspecting and Managing Containers:
  - **docker ps**: List active containers along with key details.
  - **docker exec**: Execute commands inside a running container.
  - Additional Commands: Pause, resume, remove containers, etc.

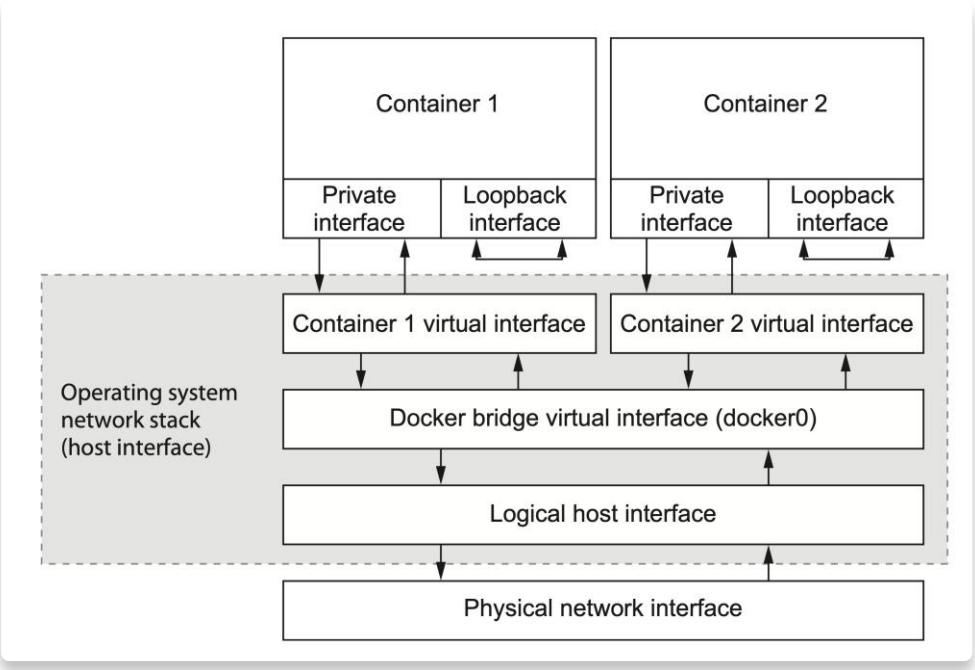
Docker Network

- Exploring Docker’s Network Functionality:
  - Docker’s Network Interaction
  - Construction of Network Containers
  - Tailoring Container Networks
  - Enabling Network Accessibility
  - Container Discovery

Cont.

- Docker handles two main networking types: single-host virtual networks and evolving multi-host networks.
- Single-host networks ensure container isolation within a host, vital for security-conscious setups.
- Networked app developers must grasp containerization's impact on deployment strategies.

Cont.



The default local Docker network topology

Cont.

- Why Docker networking is different from VM or physical machine networking?
  - VMs vs Docker:
  - VMs: Flexible with NAT and host networking.
  - Docker: Primarily uses bridge network, Linux for host networking.
  - Network Isolation: Docker uses namespace, not separate stack.
  - Container Scaling: Docker handles many containers per host; VMs run fewer processes.

Cont.

- Docker Network Drivers –
  - Allows to create three different types of network drivers:
    - The Bridge Driver
    - The Host Driver
    - The None Driver
  - & User-specific:
    - The Overlay Driver
    - The Macvlan Driver

Cont.

- Bridge -

```
$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
5077a7b25ae6   bridge    bridge    local
7e25f334b07f   host      host      local
475e50be0fe0   none      null      local
```

- Use Docker inspect for more details on the connectivity

Cont.

- Bridge Driver Drawbacks:
  - Production Caution: Not advised for production due to IP-based communication instead of service discovery.
  - Dynamic IPs: Containers get different IP addresses with each run, not suitable for production stability.
  - Security Concern: Allows unrelated containers to communicate, posing a potential security threat.
- Custom Bridge Networks: Later, learn about creating secure custom bridge networks.

Cont.

- Host –

- Shares host's networking stack, directly uses host's network.
- Available on Linux hosts only.
- Highest performance due to minimal network abstraction.
- Limited isolation: Containers share host's network namespace, potentially less secure.

Cont.

- None –

- No networking for the container.
- Useful for cases where a container doesn't require network connectivity.
- Isolated container environments.

Cont.

- Overlay -

- Designed for multi-host networking, spanning multiple Docker hosts.
- Utilizes the VXLAN protocol for encapsulation.
- Enables communication between containers on different hosts.
- Supports automatic service discovery, beneficial for distributed applications.
- Suitable for complex, distributed architectures.

Cont.

- Macvlan –

- Assigns a MAC and IP address to containers from the physical network.
- Containers appear as separate devices on the network.
- Useful when containers need direct network access with unique IPs.
- Can be complex to set up and manage.

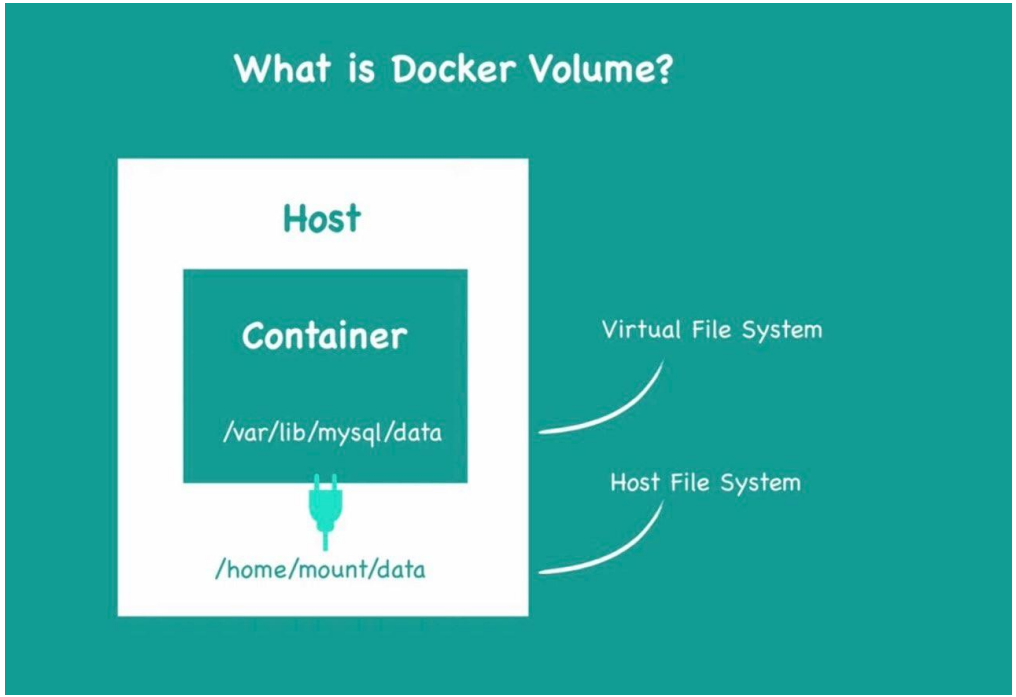
Storag
e

- Managing data in Docker –

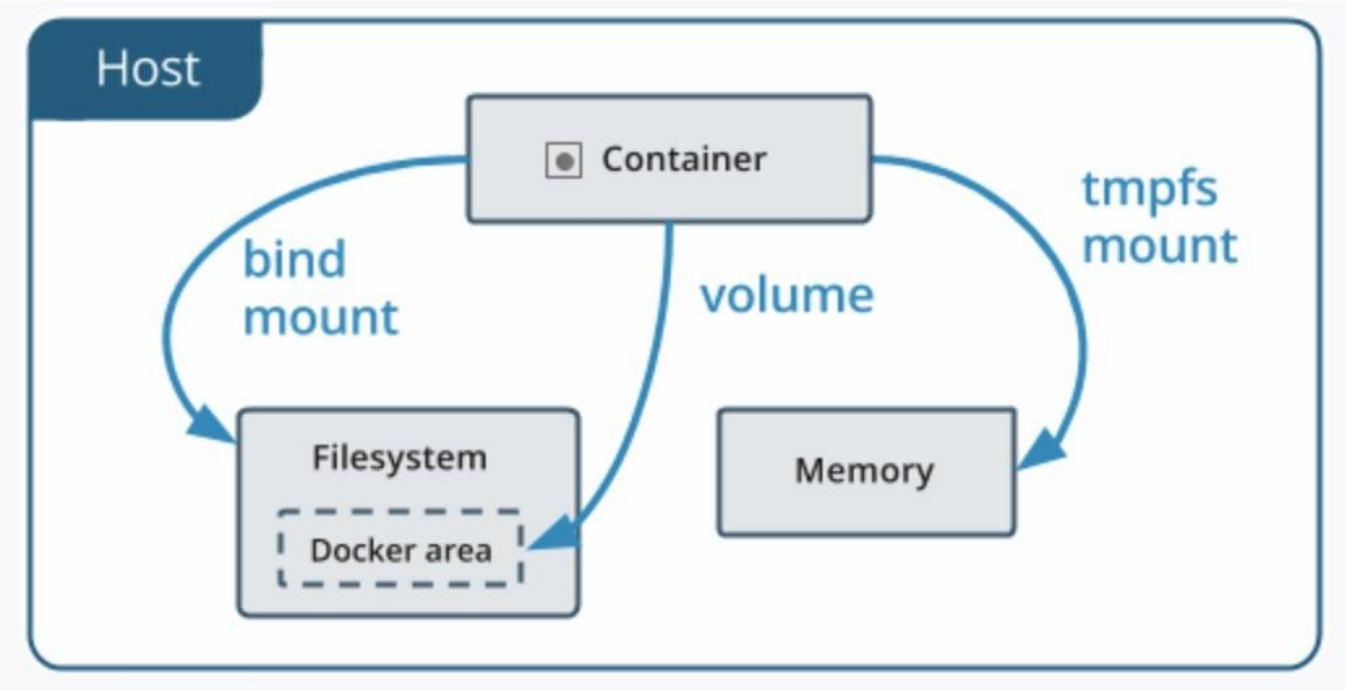
- Container Data Storage:
  - Default: Data in container's writable layer isn't persistent.
  - Challenge: Moving data from a container is complex.

- Storage Drivers:
  - Storage driver manages filesystem for container data.
  - Affects performance due to union filesystem.

Cont.


What is Docker Volume?
Host
Container
/var/lib/mysql/data
/home/mount/data
Virtual File System
Host File System

Cont.


Host
Container
bind
mount
volume
tmpfs
mount
Filesystem
Docker area
Memory

Cont.

- Persistent Options:
  - Volumes: Managed by Docker, best for persistence.
  - Bind Mounts: Host files in containers, good for sharing.
  - tmpfs Mounts: Memory-only, non-persistent data.

Cont.

- Volume Benefits:
  - Safest for data persistence.
  - Shared among containers, supports backup.
  - Volume drivers for remote storage.
  - Managed by Docker in `/var/lib/docker/volumes/` on Linux.
  - Can be mounted into multiple containers simultaneously.

Cont.

- Bind Mounts:
  - Share host files, high performance.
  - Be cautious with system files.
  - Can be located anywhere on the host.
  - Allows modification by non-Docker processes.
  - Offers high performance but relies on host filesystem structure.
  - Ideal for sharing configuration files and artifacts.

Cont.

- tmpfs Mounts:
  - Resides in host system's memory, not persisted on disk.
  - Used for non-persistent data or sensitive information.
  - Utilized by Docker's internal processes like swarm services.

Cont.

- Usage Tips:
- Volumes: Preferred for persistence.
- Bind Mounts: For sharing and artifacts.
- tmpfs Mounts: Short-lived memory data.

Docker Compose

- Docker Compose Basics:
  - Definition: Tool for defining and running multi-container Docker applications with a single configuration file.
  - Purpose: Simplifies managing and orchestrating multi-container apps.

Cont.

- Services and Containers:
  - Service: Represents each container in the application.
  - Configuration: Define images, ports, environment variables, volumes, dependencies, etc.

Cont.

- Defining Services:
  - Image: Specify the Docker image.
  - Build: Use a Dockerfile to build the image.
  - Ports: Map container ports to host ports.
  - Environment Variables: Set variables for containers.
  - Volumes: Define data volumes.
  - Depends On: Establish service dependencies.

Cont.

- Use Cases:
  - Local Development: Simplify complex setups.
  - Testing: Define test environments.
  - Staging and Production: Consistency between environments.
- Customization and Extensibility:
  - Modify default behaviour.
  - Integration with external tools like CI/CD.

Cont.

- Multi-Container Applications:
  - Scenario: Compose is designed for applications with multiple containers that need to work together.
  - Example: Web server + database for a web application.

Docker in CI/CD

- Streaming development and deployment –
  - Building a website is great but hosting it can lead to problems like glitches and downtime.
  - Traditional approaches involve deploying after writing the entire code, causing long delays and inconveniences.
  - Enter CI/CD Docker: A continuous approach to software development that combines Docker containers for smoother workflows.

Cont.

- What is CI/CD Docker?
  - CI/CD: Continuous Integration/Continuous Delivery, a methodology for SDLC (Software Development Life Cycle).
  - CI/CD Docker: Implementing CI/CD using Docker containers for seamless and efficient development.

Cont.

- Role of Docker in CI/CD:
  - Docker simplifies container creation and deployment.
  - Detects coding errors during development with container technology.

- Docker's role:
  - Streamlines processes.
  - Arranges pipeline steps logically.
  - Facilitates concurrent building and testing.
  - Integrates with tools like GitHub and Jenkins.

Cont.

- Benefits of CI/CD Docker:
  - Rapid error identification and resolution.
  - Time and cost savings.
  - Effective real-time server performance assessment.

Docker Orchestration

- Automation and Scaling:
  - As apps scale, automation is crucial.
  - Tools needed for maintenance, failure recovery, updates, and reconfigurations.
- Orchestrators Defined:
  - Orchestrators manage, scale, and maintain containerized apps.
  - Kubernetes and Docker Swarm are popular options.

Cont.

- Benefits of Container Orchestration:
  - Automation reduces complexity and effort.
  - Supports agile and DevOps approaches.
  - Facilitates rapid, iterative development and deployment.

Cont.

- Benefits of Container Orchestration:
  - Enhances containerization benefits:
  - Efficient resource utilization.
  - Automated health monitoring and availability.

Cont.

- Kubernetes and Docker Swarm:

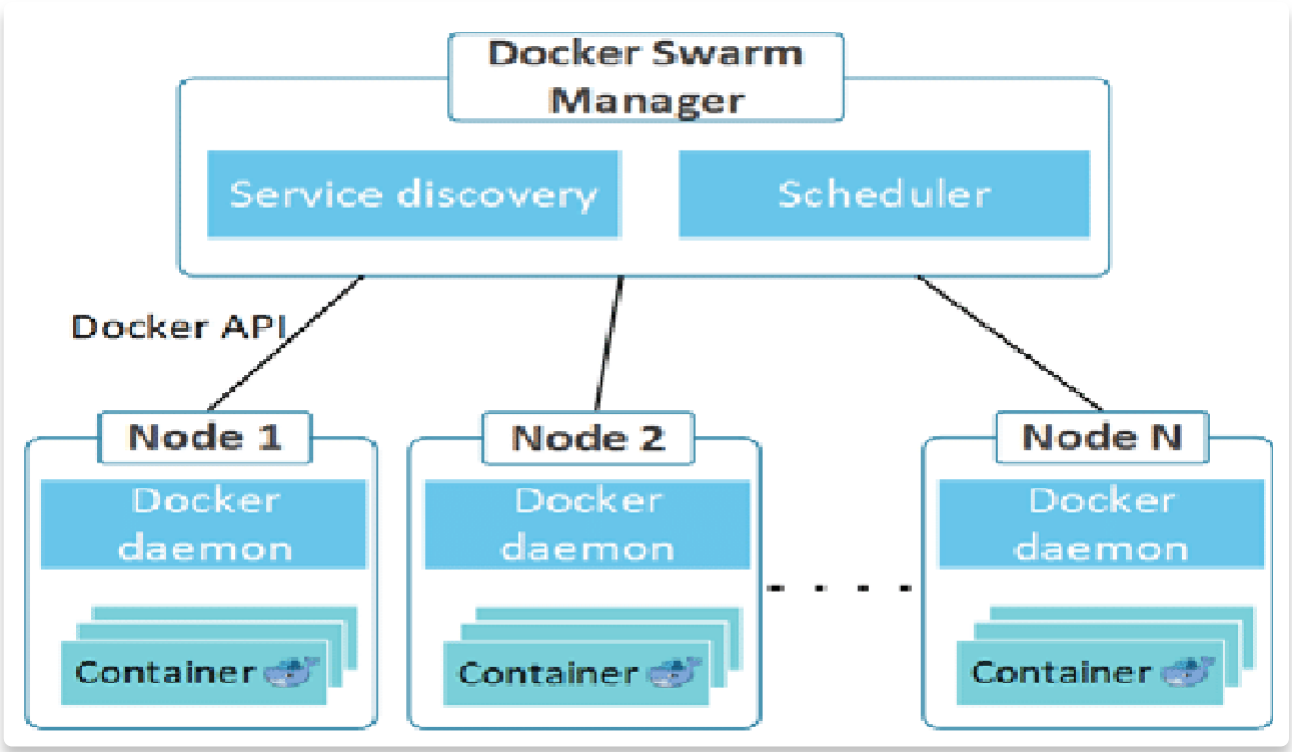
  - Kubernetes: Powerful orchestration tool.
  - Docker Swarm: Another orchestration choice.
  - Both help in scaling, maintenance, and updates.

Cont.

- Docker Swarm –

- Docker's native container orchestration tool.
- Manages containerized applications across a cluster of machines.

Cont.


Docker Swarm Manager
Service discovery
Scheduler
Docker API
Node 1
Docker daemon
Container
Node 2
Docker daemon
Container
Node N
Docker daemon
Container

Cont.

- Activation and Nodes:

- Activated using docker swarm init.
- Nodes: Machines in the Swarm.
- Manager Nodes: Control service orchestration.
- Worker Nodes: Execute tasks.

Cont.

- Services: Define container behaviour.
- Replicas: Desired number of container instances.
- Load balancing among replicas.
- Rolling updates with minimal downtime.
- Stacks: Groups of services with shared dependencies.
- Each service gets a DNS name.
- Communication between containers using service name.

Cont.

- Automated TLS certificates for security.
- Easy scalability with load distribution.
- Fault tolerance with container recovery.
- Docker CLI extended for Swarm management.

- Limitations:
- Simplified compared to Kubernetes.
- Suitable for straightforward use cases.

Troubleshooting

- Container Logs: Check logs with docker logs <container_id> for errors.
- Image Check: Verify image tags, pull with docker pull if needed.
- Networking: Confirm ports are exposed and mapped correctly.
- Resources: Ensure sufficient CPU, memory, disk space; monitor with docker stats.
- Docker Daemon: Restart if needed, check logs at /var/log/docker.log.

Cont.

- Compose Issues: Validate docker-compose.yml, debug services step by step.
- Volumes: Verify paths, permissions; use docker volume ls to inspect.
- Permissions: Ensure user is in docker group, manage container user permissions.
- Network/DNS: Check connectivity, resolve domain names, inspect proxy settings.
- Disk Space: Clear unused items with docker system prune, manage disk usage.

Security & Best practices

- Official Images: Prefer trusted Docker Hub images.
- Updates: Keep Docker, images, and containers up to date.
- Image Scanning: Use tools to spot vulnerabilities in images.
- Minimal Images: Build lightweight images to reduce risk.
- Container Isolation: Use Docker's isolation features.

Cont.

- Least Privilege: Restrict container capabilities.
- Secrets Management: Handle sensitive data securely.
- Network Segmentation: Isolate containers with networks.
- Monitoring: Implement logging and runtime security.
- Host Security: Secure hosts, control API access, and backup data.