Corso di Digital Forensics

CdLM in Informatica

Università degli Studi di Salerno

Docente: Ugo Fiore

1bis – Virtualizzazione e Virtualbox

Lo scopo della Virtualizzazione

Lo scopo della virtualizzazione è quello di eseguire contemporaneamente più istanze di sistemi operativi “guest” in un’unica macchina fisica “host”.

I sistemi operativi “**guest**” colloquiano con le risorse messe a disposizione dalla macchina fisica “**host**” attraverso un componente software di livello intermedio generalmente denominata “**hypervisor**” o “virtual machine monitor” VMM.

Lo scopo della Virtualizzazione

Uno sviluppatore di software potrà quindi eseguire la sua applicazione in **diversi ambienti senza dover disporre di più macchine fisiche** o un amministratore di sistemi potrà testare uno scenario complesso che veda interagire più servizi su host diversi, ricreandolo su più **macchine virtuali** (VM) ospitate in una singola macchina fisica.

Ma sono gli utenti finali a trarre i maggiori benefici. Precisamente:

I benefici della Virtualizzazione

- **Aumento dell'affidabilità del sistema**

Sarà infatti possibile **dedicare** una macchina virtuale all'esecuzione di pochi servizi che notoriamente non vanno in conflitto tra di loro. Inoltre l'hypervisor è in grado di **isolare** le macchine guest in esecuzione sullo stesso host affinché eventuali problemi che compromettono il funzionamento di una singola macchina virtuale, non influenzino la stabilità delle altre.

I benefici della Virtualizzazione

- **Consolidamento dei server**

Molte aziende hanno visto crescere vistosamente il numero dei server proprio a causa dell'aumento dei servizi da fornire ai propri utenti. Attraverso la virtualizzazione si possono eseguire più macchine virtuali nella stessa macchina fisica **riducendo il numero dei server di 10 volte o più**.

Infatti è noto che la maggior parte dei server x86 ha un basso utilizzo di CPU e con le attuali tecnologie multiprocessore multicore non è raro spingersi a rapporti di consolidamento superiori.

I benefici della Virtualizzazione

- **Riduzione del Total Cost of Ownership (TCO)**

Il consolidamento ad un numero inferiore di server permette una notevole **riduzione dei costi legati all'energia** utilizzata per alimentare i server e per mantenere la temperatura ambientale adatta alle sale server. Inoltre **si riducono i costi di acquisto e** i canoni **di manutenzione** dei server fisici.

I benefici della Virtualizzazione

- **Disaster Recovery**

L'intero sistema operativo "guest" può essere facilmente salvato e ripristinato riducendo notevolmente i tempi di indisponibilità in caso di guasto.

- **Alta disponibilità**

Se è presente una infrastruttura di server fisici con delle caratteristiche hardware tra loro compatibili e questi server condividono una area dati (storage) sulla quale risiedono le macchine virtuali, sarà possibile spostare l'esecuzione di una macchina virtuale su un altro host in caso di failure. Alcuni sistemi prevedono lo spostamento automatico delle macchine virtuali tra i vari host in funzione al carico

I benefici della Virtualizzazione

- **Esecuzione di applicazioni legacy**

È frequente che alcune organizzazioni utilizzino applicazioni sviluppate per sistemi operativi che girano su **hardware ormai obsoleto**, non supportato o addirittura introvabile. Attraverso la virtualizzazione si possono continuare ad utilizzare quelle applicazioni che diversamente dovrebbero essere migrate ad una architettura più attuale affrontando i costi relativi al porting e al debug.

I benefici della Virtualizzazione

- **Sviluppo e Testing**

Possibilità di predisporre ambienti di sviluppo e di testing di varie tipologie in maniera agevole. Isolamento all'interno degli host rispetto all'ambiente di lavoro principale. Orientato sia ai server che ai client.

Virtualizzazione: che cos'è

In informatica, la virtualizzazione è un termine generico che si riferisce all'astrazione di risorse di calcolo (computing resources)

- **Platform virtualization** ovvero virtualizzazione di piattaforme hardware (concetto di VM)

- **Resource virtualization** ovvero virtualizzazione di risorse (concetto di qualità del servizio)

The basic concept of platform virtualization, was later extended to the virtualization of specific system resources, such as storage volumes, name spaces, and network resources. **Resource aggregation**, spanning, or concatenation combines individual components into larger resources or resource pools.

Platform Virtualization

- Emulation or simulation
- Native virtualization or full virtualization
- Hardware enabled virtualization
- Partial virtualization
- Paravirtualization
- Operating system-level virtualization
- Application Virtualization

Resources Virtualization

- RAID
- SAN
- Channel bondings
- VPN/NAT
- Multiprocessor and multi-core
- Cluster and Grid computing
- Partitioning

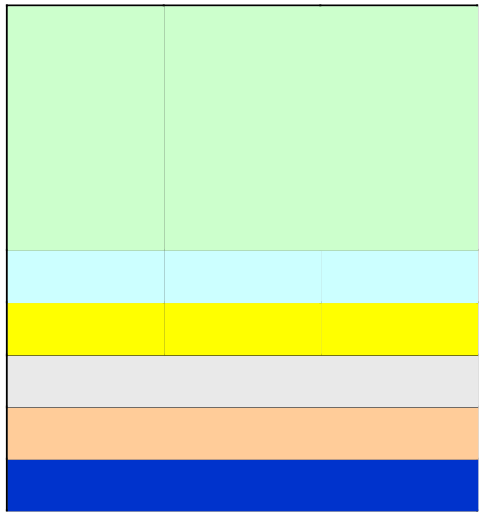
Hypervisor (VMM) e Virtual Machine (VM)

- Una **VMM** (Virtual Machine Manager o **Hypevisor**) astrae l'hardware di un singolo calcolatore.

  - **Crea e controlla molti diversi ambienti di esecuzione** (VM).

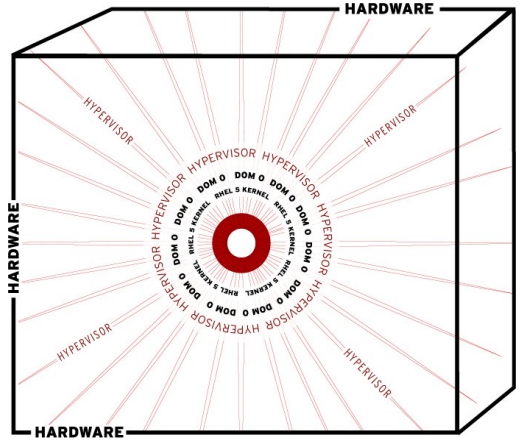
  - **Ciascuno di questi ambienti può avere un proprio sistema operativo.**

  - **Ciascuno di questi ambienti crede di controllare l'intero sistema hardware.**



Quindi una Virtual Machine (VM) è un'ambiente di esecuzione creato da un'hypervisor (VMM)

Hypervisor



- Provides **protection**, **networking**, **driver coordination**, **and resource management** so that each virtual OS sees itself as running on a bare metal server.
- **Allows you to create, control, monitor, destroy, pause, or migrate** new virtual machines.
- Virtual Machine Monitor, **Scheduling** Virtual CPU, Memory

Hypervisor

Hypervisors (VMM) are currently classified in two types:

- **Type 1** hypervisor is software that **runs** directly **on a given hardware platform** (as an operating system control program). A "guest" operating system thus runs at the second level above the hardware.
  - Recent examples are Xen, VMware's ESX Server, and Sun's Hypervisor.
- **Type 2** hypervisor is software that **runs within an operating system environment**. A "guest" operating system thus runs at the third level above the hardware.
  - Examples include VMware server and Microsoft Virtual Server.

Principali hypervisors

- **Oracle Virtualbox**:
  - http://en.wikipedia.org/wiki/VirtualBox
- **KVM**
  - http://en.wikipedia.org/wiki/Kernel-based_Virtual_Machine
- **QEMU**
  - http://en.wikipedia.org/wiki/QEMU
- **Parallel**
  - http://en.wikipedia.org/wiki/Parallels,_Inc.

UTM [Mac]

- https://mac.getutm.app/

Principali hypervisors

- **Xen**:
  - University of Cambridge Computer Laboratory
  - Fully open sourced
  - Set of patches against the linux kernel
- **Vmware ESX** :
  - Closed source
  - Based on Linux 2.4
  - Proprietary drivers
- **Xbox 360**:
  - Closed source, used to assume full backward compatibility with the old Xbox

Tipi di Hypervisor (VMM)

- Type 0

Hardware-based solution

Commonly found in mainframe computers.

Examples: IBM LPARs, Oracle LDOMs

Tipi di Hypervisor (VMM)

- Type 1

Operating-system-like software provides virtualization

Examples: VMware ESX, Joyent SmartOS, Citrix XenServer

General-purpose OSes that provide VMM functions

Examples: Microsoft Windows Server with HyperV, **Red Hat Linux with KVM feature**

Tipi di Hypervisor (VMM)

- Type 2

Applications that run on standard OSes to provide VMM functionality to guest operating systems.

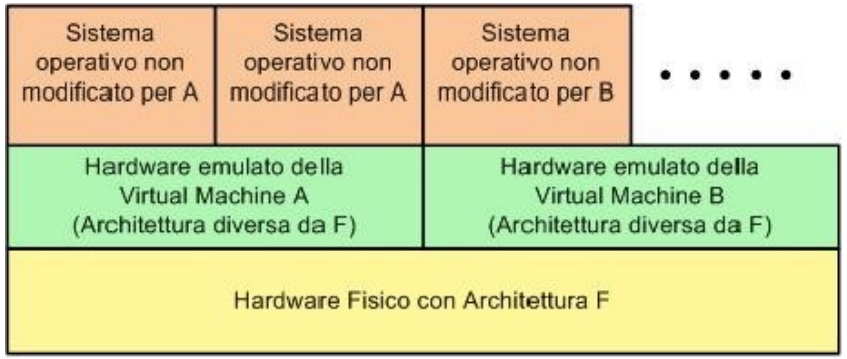
Example: VMware Workstation and Fusion, Parallels Desktop, Oracle Virtual Box.

Virtualizzazione:
3. Quali sono le principali metodologie?

- Il problema della virtualizzazione è stato affrontato in diversi modi. Tutte le **quattro metodologie di virtualizzazione** attualmente impiegate danno l'illusione di utilizzare un sistema operativo stand alone, non virtualizzato. Esse sono:
  - l'**emulation**,
  - la **full virtualization**,
  - la **paravirtualization**
  - la **operating system level virtualization**.

Emulation

Con l'emulazione l'hypervisor simula l'intero hardware set che permette al sistema operativo guest di essere eseguito senza alcuna modifica. Il software di virtualizzazione **si incarica di presentare al sistema operativo guest un'architettura hardware completa a lui nota, indipendentemente dall'architettura hardware presente sulla macchina host.**



L'emulatore simula una'architettura hardware diversa da quella fisica

I limiti della Emulation

Gli emulatori presentano al sistema operativo guest un'**architettura hardware standard** precludendo quelle che potrebbero essere le funzionalità alle quali siamo abituati, ad esempio quelle implementate in hardware.

Inoltre deve **interfacciare la CPU** (con istruzioni semanticamente equivalenti), **la memoria** (accesso esclusivo e riserva) **e l'I/O** tra sistema host e sistema guest.

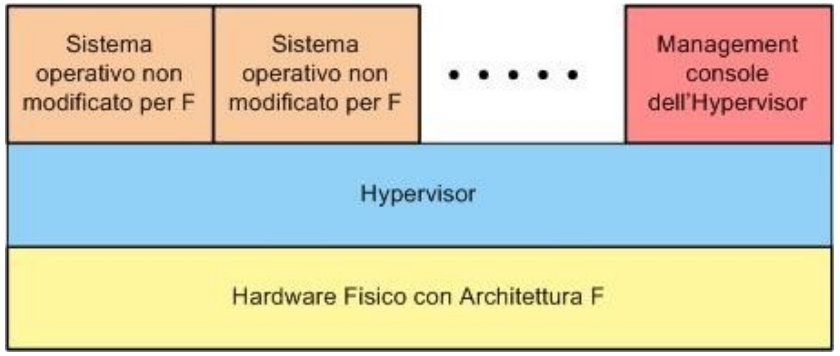
Questo carico di lavoro dell'emulatore rende **difficoltoso** l'uso di questa tecnica di virtualizzazione **quando bisogna emulare sistemi guest che richiedono processori di velocità equivalente al processore dell'host.**

Full Virtualization

La Full (o Native) Virtualization è simile alla Emulation ma **i sistemi operativi guest devono essere compatibili con l'architettura hardware della macchina fisica**. In questo modo molte **istruzioni** possono essere **eseguite direttamente sull'hardware senza bisogno di un software di traduzione garantendo prestazioni superiori** rispetto all'emulazione.

Recentemente Intel e AMD hanno introdotto **VT-x** ed **AMD-v**

Esempi di software che utilizzano la full virtualization sono VMware, Virtual Box, e Xen, limitatamente ai sistemi operativi proprietari non modificabili.

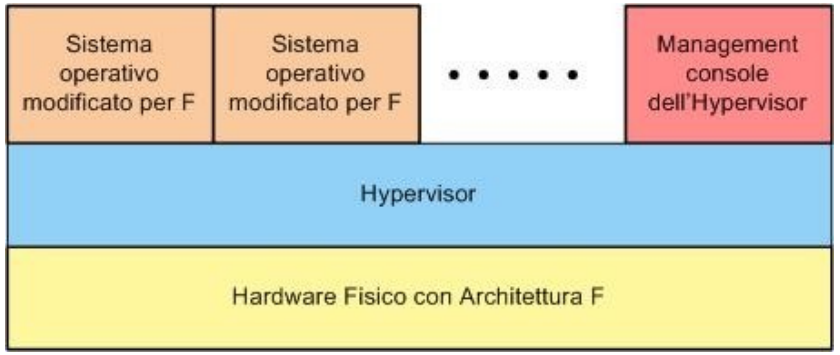


La Full Virtualization presenta al sistema operativo guest la stessa architettura hardware presente sull'host fisico

Paravirtualization

L'hypervisor presenta alle macchine virtuali un versione modificata dell'hardware sottostante, mantenendone tuttavia la medesima architettura (stessa ABI). Il sistema operativo in esecuzione sulle macchine virtuali è invece modificato per **evitare alcune particolari chiamate di sistema**. Questa tecnica **permette di ottenere un decadimento delle prestazioni minimo rispetto al sistema operativo non virtualizzato, dato che le istruzioni provenienti dalle macchine virtuali vengono eseguite quasi tutte direttamente sul processore senza che intervengano modifiche**.

Esempi di paravirtualizzazione sono Xen e User-mode Linux.

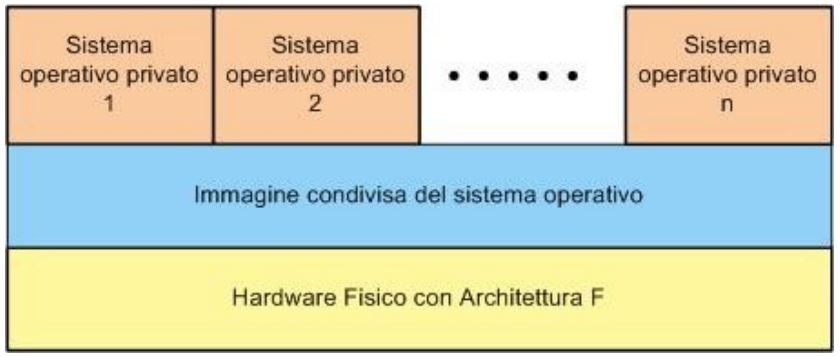


La Paravirtualization è simile alla full virtualization, ma è necessario modificare il sistema operativo guest per ottimizzare l'esecuzione sull'ambiente virtualizzato

Operating System Level Virtualization

Non si utilizza un Hypervisor, ma la virtualizzazione è creata utilizzando **copie del sistema operativo installato sull'host**. I sistemi guest creati saranno a tutti gli effetti istanze del sistema operativo host con un proprio file system, configurazione di rete e applicazioni. Il vantaggio principale di questa tecnica è il miglior utilizzo delle risorse grazie alla condivisione di spazi di memoria. Essendo i sistemi operativi delle macchine guest equivalenti a quello della macchina host, **le istanze guest non richiederanno un kernel privato**, ma utilizzeranno lo stesso con un conseguente minor utilizzo di memoria fisica. Non adatto a sistemi operativi diversi sullo stesso host. Poca stabilità, poco isolamento.

Esempi: Virtuozzo, Linux VServers, Solaris Containers, HPUX 11i Secure Resource Partitions



La Operating System level Virtualization mette a disposizione dei sistemi guest l'immagine del sistema operativo in esecuzione sull'host.

Hypervisor (VMM) di tipo 2

Example: VirtualBox

The VMM is just another process running on the host.
The host OS doesn't even know that virtualization is taking place.The VMM may run as a user application.

It might not have the administrative privileges to access the hardware assistance features of modern CPUs. Therefore, type 2 hypervisors can have poorer performance than type 0 or type 1.

No changes are required to the host operating system.

Any student can run VirtualBox to experiment and learn from different guest operating systems.

Creazione di un sistema guest

Cliccare il bottone «Nuova» direttamente sotto ai Menu.

Nome e SistemaOperativo

Inserite un nome univoco per la macchina. Ad esempio:

Mint-sistemi-operativi-2.

Scegliete la famiglia di SO: Linux.

Scegliete il tipo di SO: Linux 2.6/3.x/4.x(64-bit).

Clicate“Avanti”.

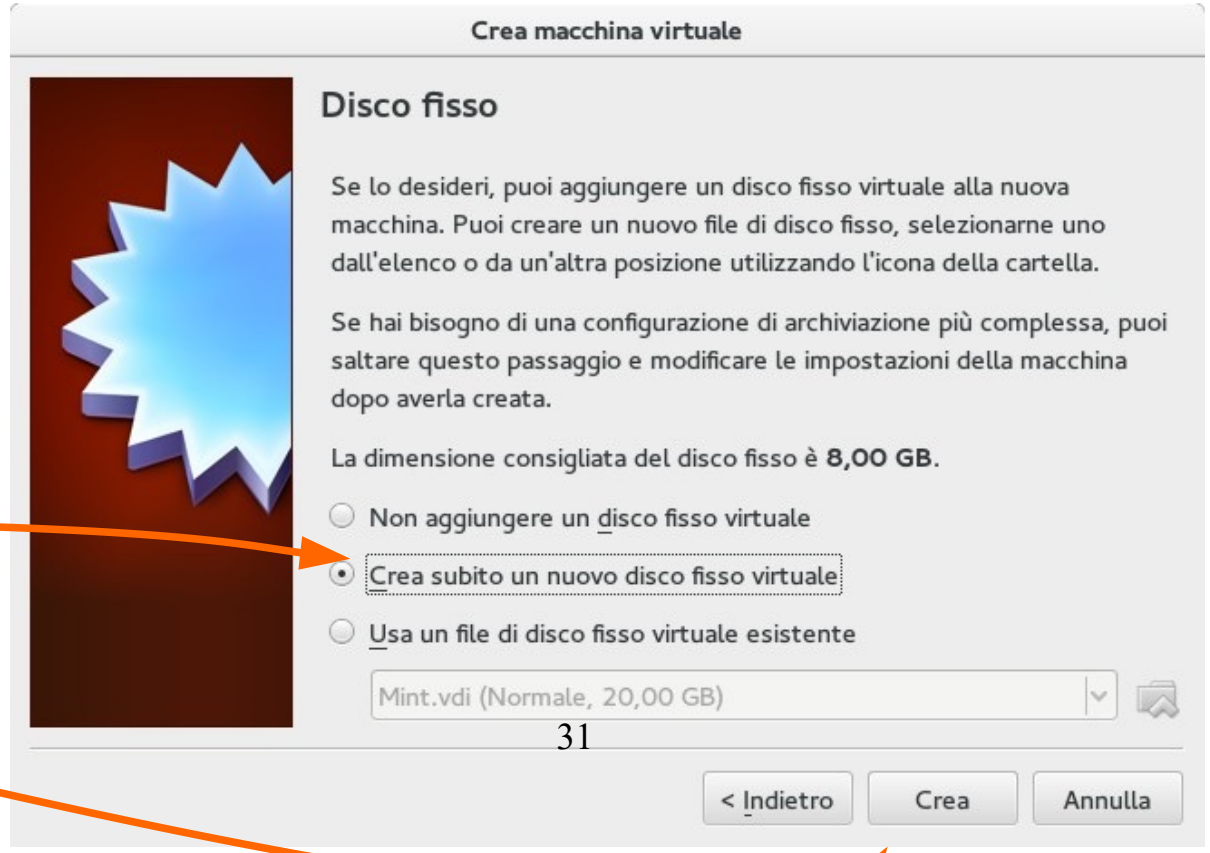
Dimensione della memoria

Impostate fino a un quarto della memoria fisica disponibile sul vostro host.

Ad esempio, nel caso della macchina del docente:4GB (un ottavo della memoria disponibile).

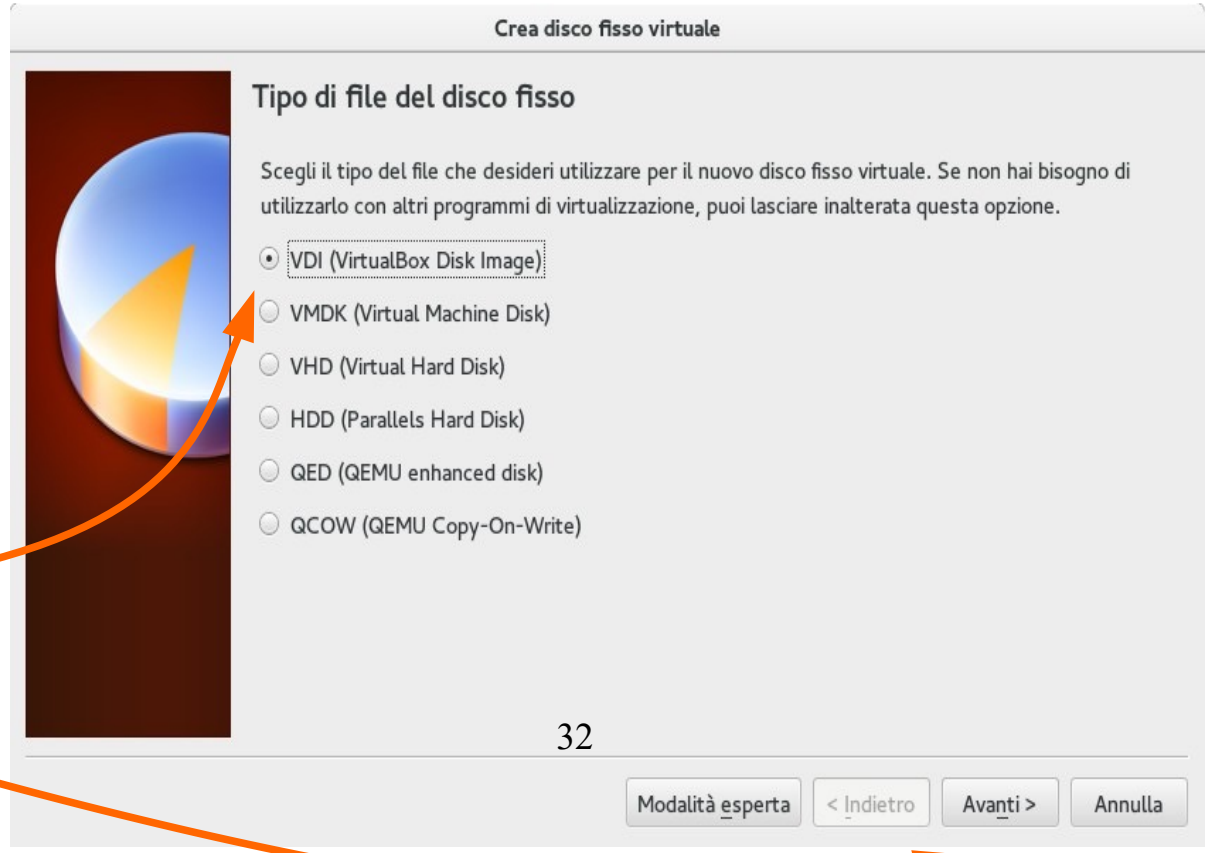
Disco fisso

Create un nuovo disco fisso virtuale.
Cliccate “Crea”.



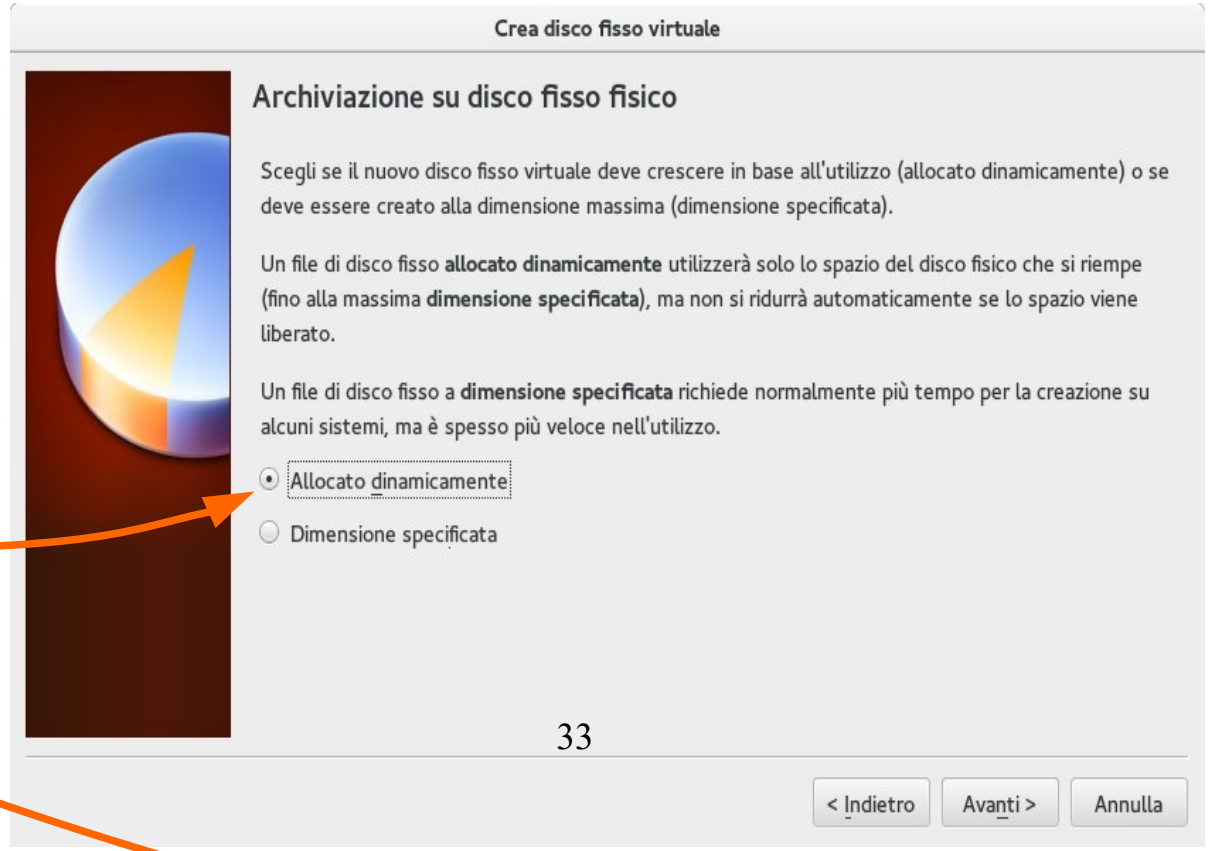
Tipo del disco fisso

Scegliete il formato del disco fisso virtuale: VDI.
Cliccate “Avanti”.



Archiviazione su disco fisso fisico

Allocate dinamicamente lo spazio sul disco fisso fisico.
Cliccate“Avanti”.



Posizione file e dimensione

Impostate 20GB di spazio su disco. Cliccate“Crea”.

Docker

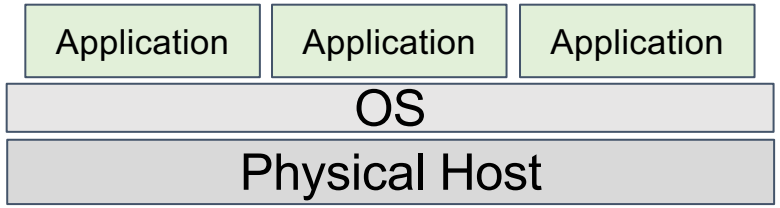
Outline

- Docker images and containers
- Docker networking
- Docker compose

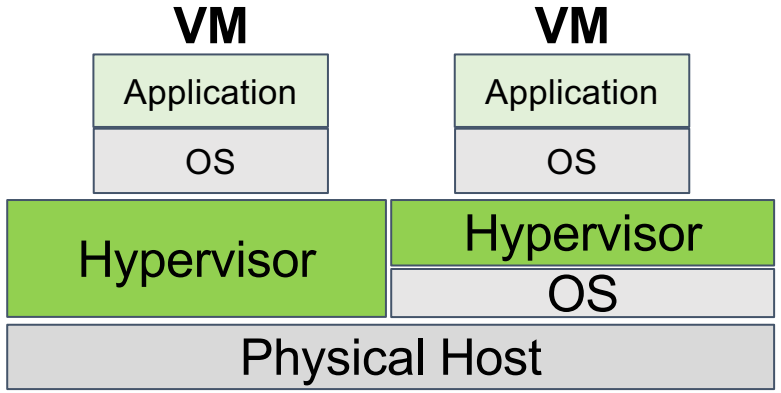
Outline

- Docker images and containers
- Docker networking
- Docker compose

Traditional vs Virtualized Deployment



- Physical Hosts run an Operating System (e.g., Windows or Linux).
- Multiple applications run on the shared OS.



- A special software, i.e., the **Hypervisor**, provides Virtual Machines.
- Examples of such technologies are *Virtualbox*[1] or *Linux KVM*[2].
- VM is a full machine running all the components, including its own Operating System, on top of the virtualized hardware

[1]https://www.virtualbox.org/

[2]https://www.linux-kvm.org

Traditional vs Virtualized Deployment

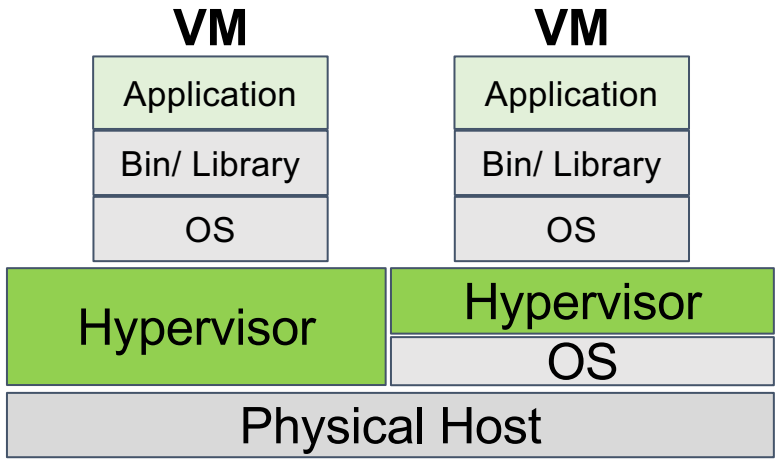
Traditional Deployment

- No way to define resource boundaries for applications.
- Isolating applications requires running them on different physical servers (expensive and resources could be underutilized).

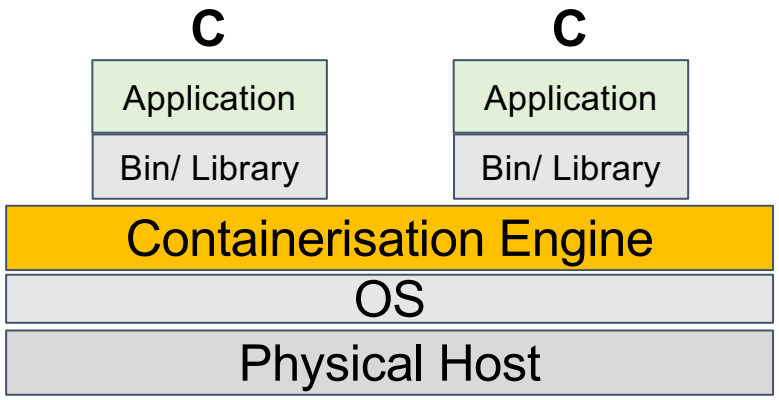
Virtualized Deployment

- Virtualization allows:
  - applications to be isolated between VMs
  - better utilization of resources in a physical server
  - better scalability.

Virtualized vs Container Deployment



- Virtual hardware
  - Each VM has an OS and Application
  - Share hardware resource from the Physical Host



- Virtual Operating Systems
  - Isolated environments, namely **containers**, sharing the same *real* operating system
  - Containers run from a distinct image that provides all files (Bin/, Library) necessary to support them
  - Examples of such technologies is *Docker*[1].

[1]https://www.docker.com/

Virtualized vs Container Deployment

Virtualized Deployment

- Heavyweight: each VM relies on a full copy of an Operating System.
- Provides full isolation.
- Best suited for when you have applications that need to run on *different* Operating System flavors.

Container Deployment

- Lightweight: sharing OS resources significantly reduces the overhead required for running containers.
- Provides a (relaxed) process-level isolation.
- Best suited for when you have applications that need to run over a *single* Operating System kernel.

Docker images and containers

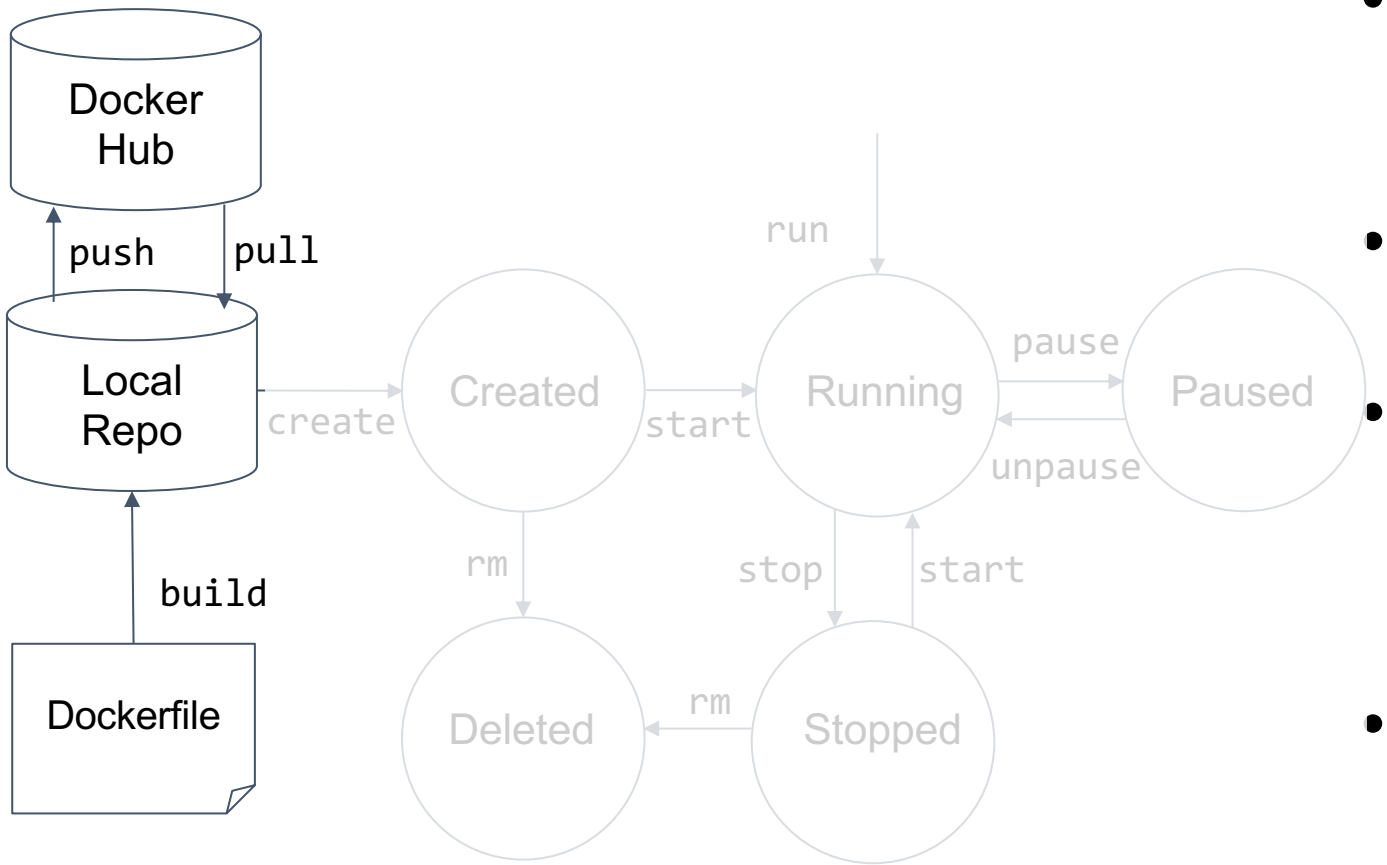
Image

- **Immutable** template for containers
- Includes everything needed to run an application
  - code, runtime, system tools, system libraries, and settings

Container

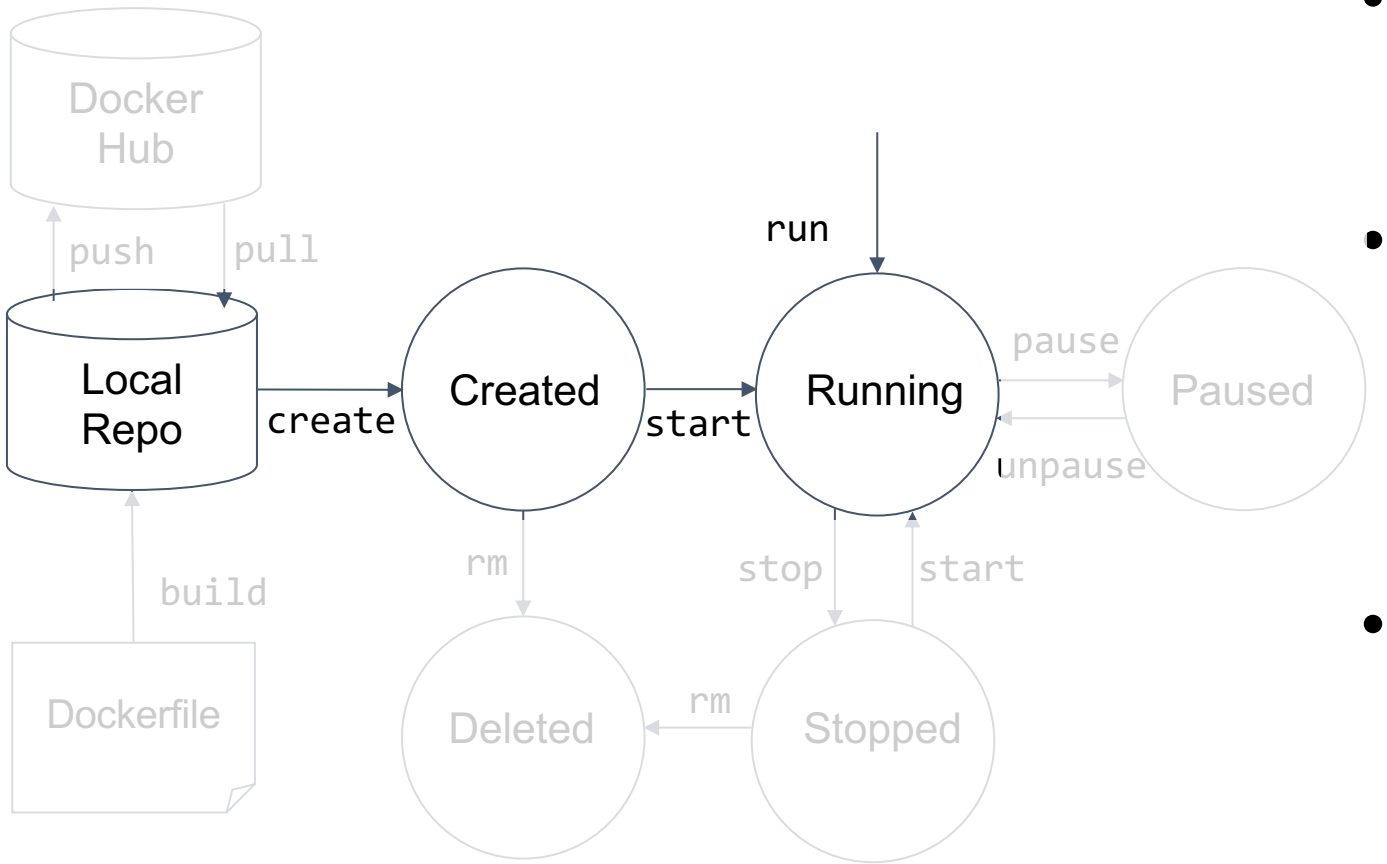
- An instance of an image
- Add a **new writable layer** on top of the underlying image
  - all changes made to the running container (e.g., writing new files or modifying existing files) are written to this writable container layer

Docker container lifecycle



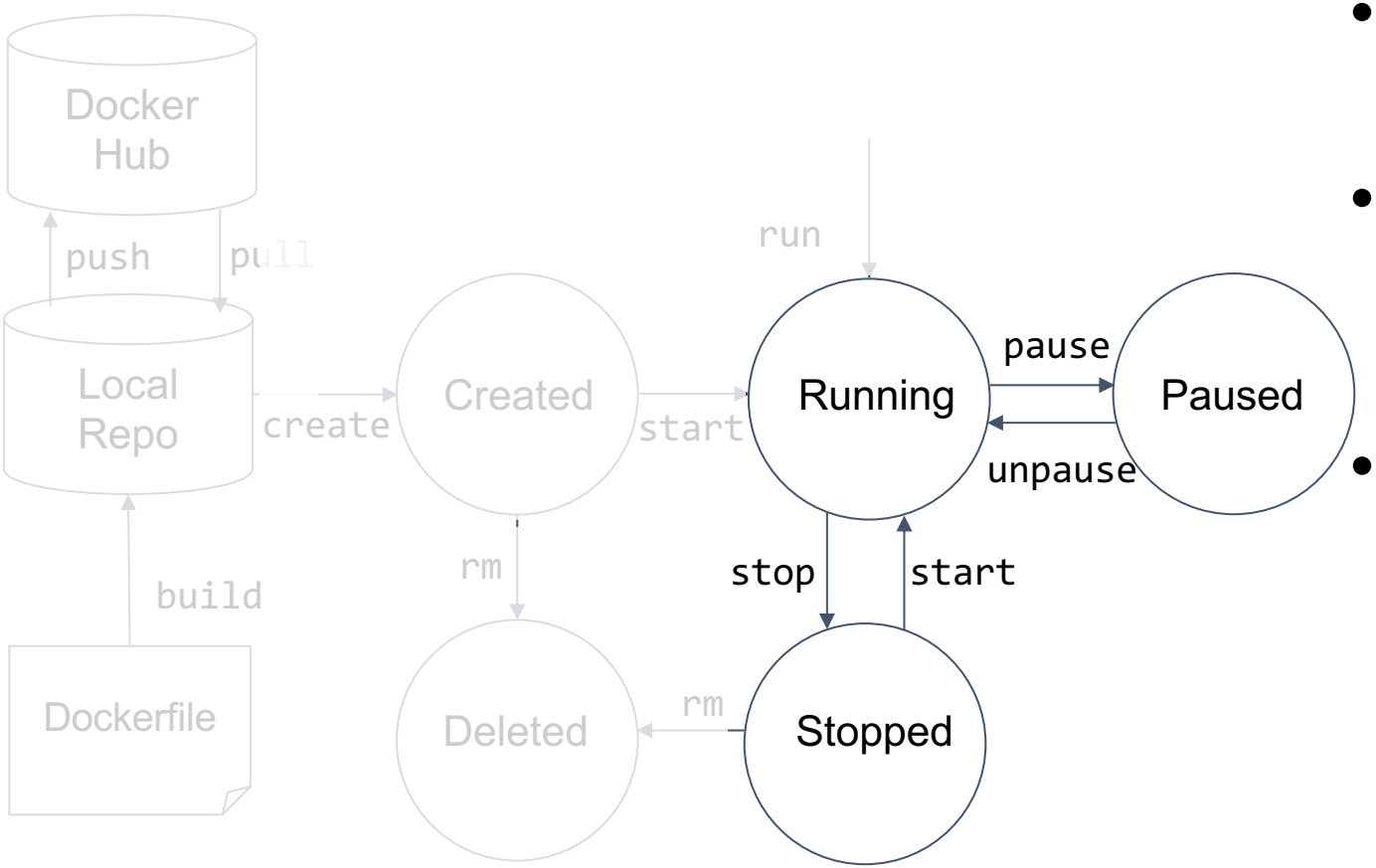
- Docker images can be pulled from a central repository (Docker Hub)
- Pulled images are saved in a local repository
- Custom images can be created and saved starting from a specific configuration file (Dockerfile)
- Custom images can be pulled to the central repository

Docker container lifecycle



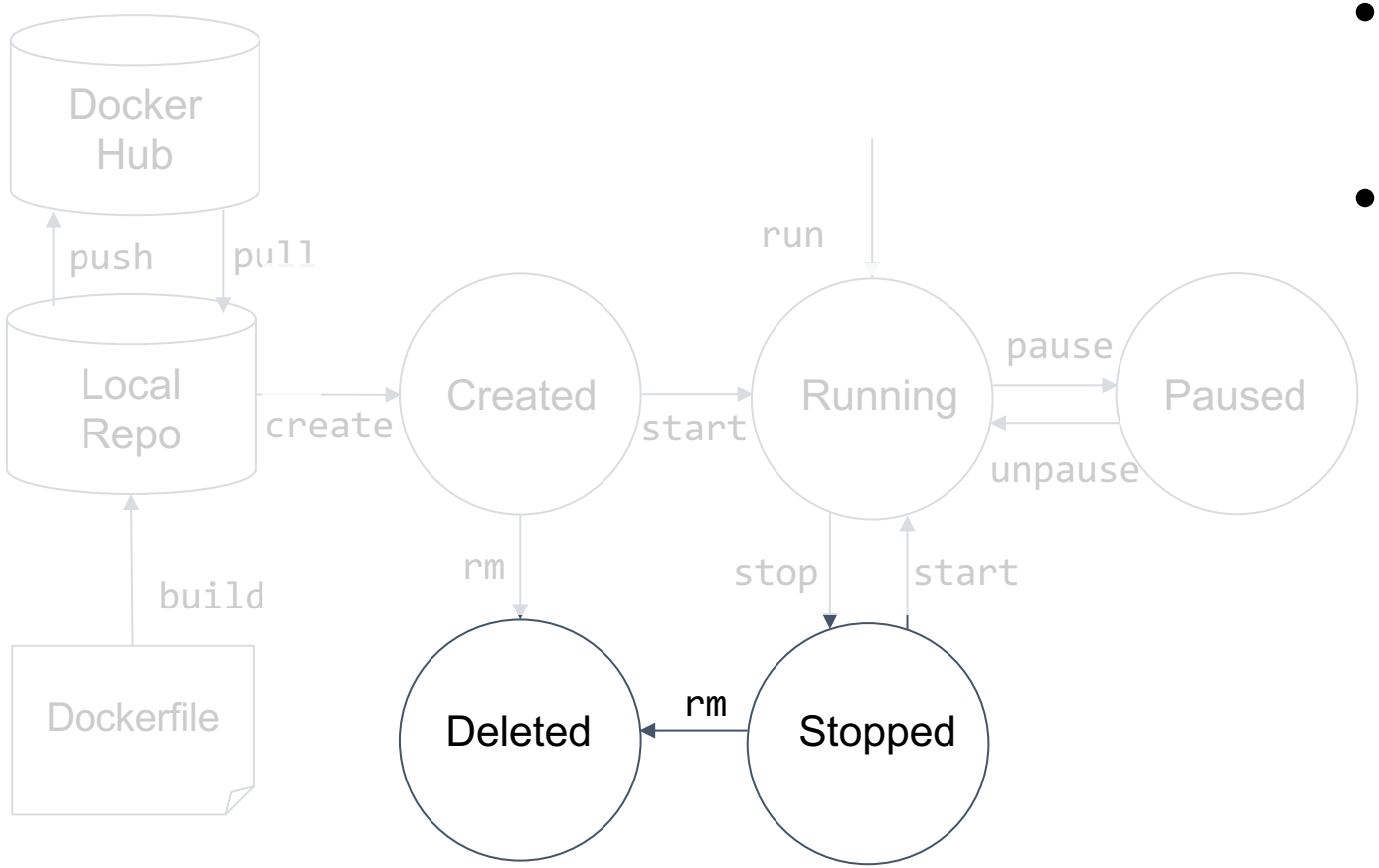
- A saved image can be used for creating a container (a writeable layer is added)
- Starting a container means running a default command contained in the image, namely the entrypoint (can be overridden)
- A container can be created and started using a single run command

Docker container lifecycle



- After executing the entrypoint, the container stops
- A **foreground** process specified as the entrypoint can keep running the container
- A running container can be paused or stopped

Docker container lifecycle



- After a container is stopped, the writeable layer still exists
- Deleting a container permanently removes the associated writable layer

Docker lifecycle example: images

- Pull an image from the central hub
  - docker pull <image>
- Build an image from a Dockerfile
  - docker build -t <image>
- List images saved in the local repository
  - docker images
- Delete an image
  - docker rmi <image>

Docker lifecycle example: containers

- Start a container
  - docker run <image>
- Start a container overriding default entrypoint with <newcmd>
  - docker run --entrypoint <newcmd> <image>
- Execute a command <cmd> (e.g., /bin/bash) in a running container
  - docker exec -it <containerID> <cmd>
- Stop a container
  - docker stop <containerID>
- Remove a container
  - docker rm <containerID>
- List running and stopped containers
  - docker ps –a

Docker volumes

- Volumes can be used to save (persist) data and to share data between containers
- Volume is unrelated to the container layers: deleting a container does not involve deleting an associated volume
- A volume can be:
  - (anonymous/)named: managed internally by Docker itself
  - host: refers to a filesystem location of the host running Docker

Docker volumes: example

- Create a named volume
  - docker volume create volumename
- Run a container using the named volume
  - docker run –v volumename:/path/in/container_filesystem
- Running a container using a host volume
  - docker run –v /path/on/host_filesystem:/path/in/container_filesystem

Dockerfile

- Docker can build custom images automatically by reading the instructions from a Dockerfile
- Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image
- The *docker build* allows the execution of an automated build of an image starting from a Dockerfile

(Dockerfile reference: https://docs.docker.com/engine/reference/builder/)

Dockerfile example

Command	Description	Example
FROM <image>	Start building from this (base) image	FROM ubuntu
RUN <cmd>	Run the specified command	RUN apt install apache2
COPY <src> <dest>	Copy a file to the image fs	COPY vh.conf /etc/apache2/conf/
CMD ["exec", "param1“, …]	Configure the default command when container starts	CMD ["apache2", "-D", " FOREGROUND "]

Outline

- Docker images and containers
- Docker networking
- Docker compose

Docker networking

Docker supports different configurations, the two main ones being

- *Bridge* (default)
  - isolated layer 3 networks enabling connected containers to communicate
  - (can) allow the access to external networks masquerading connections with the host network configuration
- *Host*
  - containers use the host network
  - listening ports are exposed to the outside world

Docker networking: published ports

- A container connected to bridges is isolated and does not expose any of its ports to the outside world
- A published port can be made available to services running outside the container
- A published port is mapped to a port on the Docker host

Docker networking: examples

- Create network with a configured subnet and gateway address
  - docker network create --driver bridge <networkname> --subnet=<ip/mask> --gateway=<ip/mask>
- Connect a container to a network
  - docker network connect <networkname> <containerid>
- Run a container and expose a port
  - docker run –h <host-name> -p <internal-port>:<exposed-port> --name <container-name> <image-name>

Outline

- Docker images and containers
- Docker networking
- Docker compose

Docker compose

Compose is a tool for defining and running multi-container Docker applications

- A single file for providing configurations
- A single set of commands for configuring, building, and running all the containers

Docker compose configuration

- The Compose file (docker-compose.yaml) uses a standard, human-readable syntax, namely YAML* syntax. It defines
  - Services: configuration that is applied to each container (much like passing command-line parameters to *docker run*)
  - Networks (optional): define configuration of networks to be created
  - Volumes (optional): define configuration of volumes to be created

* https://yaml.org/

Docker compose configuration: example

```
services:
 # frontend container
 app:
  # use a custom image
  build:
   # directory containing Dockerfile
   context: ./app
  # image name
  image: custom_image
  # exposed ports (host:container)
  ports:
    - 8080:80
  # a mapped host volume
  volumes:
    -  ./config/config.json:/etc/config.json
  # connected networks (defined in networks..)
  networks:
    ext:
      ipv4_address: 192.168.100.100
    int:

 # backend container
 db:
  # pull an existing image
  image: mariadb
  # a mapped named volume
  volumes:
    - db-content:/var/lib/mysql
  networks:
    int:

volumes:
 db-content:

networks:
 ext:
  driver: bridge
  ipam:
   driver: default
   config:
     - subnet: 192.168.100/24
 int:
  driver: bridge
```

Docker compose: commands example

- (build images and) run services
  - docker-compose up –d
- stop services
  - docker-compose stop
- start services
  - docker-compose start -d
- stop and remove containers and networks
  - docker-compose down
- show logs (entrypoint output)
  - docker-compose logs -f [service_name]