

Docker containers

Dott. Costantino Zazza, costantino.zazza@unitus.it

IR0000032 – ITINERIS, Italian Integrated Environmental Research Infrastructures System
(D.D. n. 130/2022 - CUP B53C22002150006) Funded by EU - Next Generation EU PNRR-
Mission 4 “Education and Research” - Component 2: “From research to business” - Investment
3.1: “Fund for the realisation of an integrated system of research and innovation infrastructures”



Docker

2

Università degli Studi di Roma "Tor Vergata"
Dipartimento di Ingegneria Civile e Ingegneria Informatica

Container-based virtualization: Docker

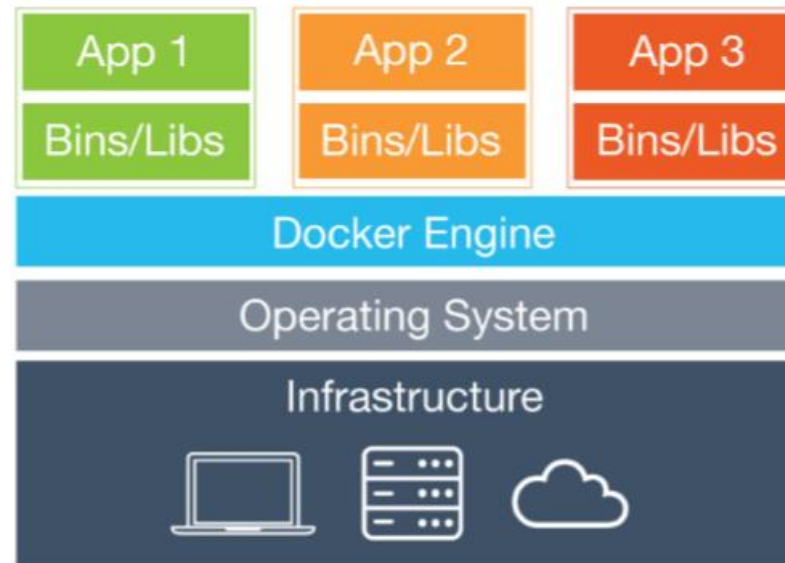
Corso di Sistemi Distribuiti e Cloud Computing
A.A. 2017/18

Valeria Cardellini
Matteo Nardelli

Case study: Docker



- Lightweight, open and secure container-based virtualization
 - Containers include the application and all of its dependencies, but share the kernel with other containers
 - Containers run as an isolated process in userspace on the host operating system
 - Containers are also not tied to any specific infrastructure



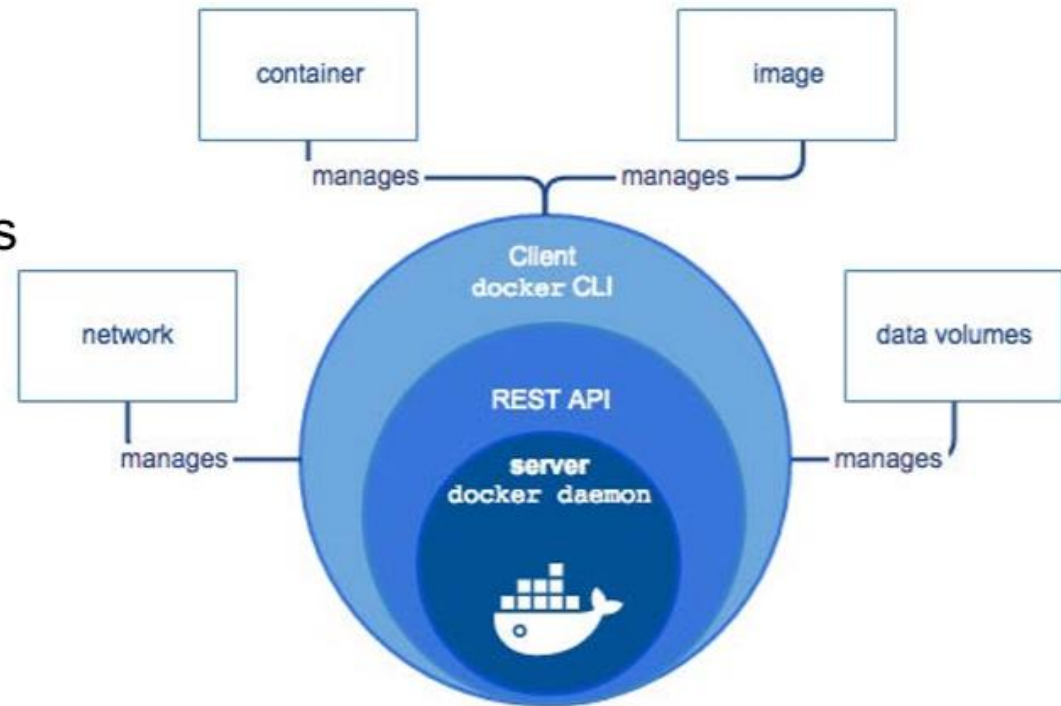
Docker internals

- Docker is written in Go language
- With respect to other OS-level virtualization solutions, Docker is a higher-level platform that exploits Linux kernel mechanisms such as **cgroups** and **namespaces**
 - First versions based on LXC
 - Now based on its own *libcontainer* runtime that uses Linux kernel namespaces and cgroups directly
 - *libcontainer* (now included in *runc*): cross-system abstraction layer aimed to support a wide range of isolation technologies
- Docker adds to LXC
 - Portable deployment across machines
 - Versioning, i.e., git-like capabilities
 - Component reuse
 - Shared libraries, see Docker Hub hub.docker.com

Docker engine

Docker Engine: client-server application composed by:

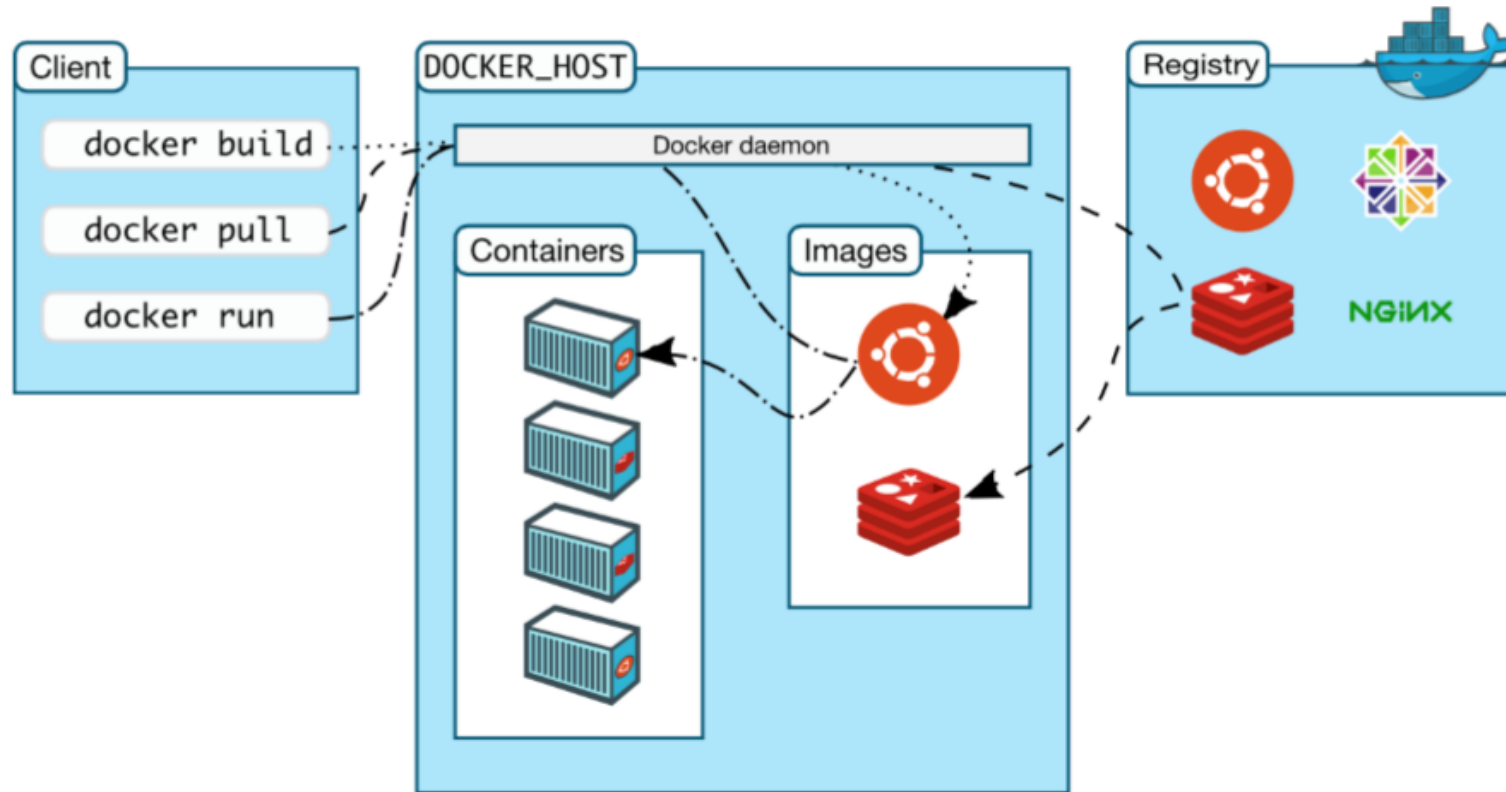
- A server, called daemon process
- A REST API which specifies interfaces that programs can use to control and interact with the daemon
- A command line interface (CLI) client



<https://docs.docker.com/engine/docker-overview/>

Docker architecture

- Docker uses a client-server architecture
 - The Docker *client* talks to the Docker *daemon*, which builds, runs, and distributes Docker containers
 - Client and daemon communicate via sockets or REST API



Docker image

- Read-only template with instructions for creating a Docker container
 - Described in text file called *Dockerfile*, with simple, well-defined syntax
 - The **Build** component of Docker
 - Enables the distribution of applications with their runtime environment: a Docker image incorporates all the dependencies and configuration necessary for it to run, eliminating the need to install packages and troubleshoot
 - Target machine must be Docker-enabled
- A Docker Image
 - A template for containers
 - Can be pulled and pushed towards a registry
 - Image names have the form `[registry/][user/]name[:tag]`
 - The default for the tag is *latest*

Docker image

- Images can be created from a Dockerfile and a context:
 - Dockerfile: instructions to assemble the image
 - Context: set of files (e.g., application, libraries)
 - Often, an image is based on another image (e.g., ubuntu)
- Example of a Dockerfile

```
# Use an official Python runtime as a parent image
FROM python:2.7-slim

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
ADD . /app

# Install any needed packages specified in requirements.txt
RUN pip install -r requirements.txt

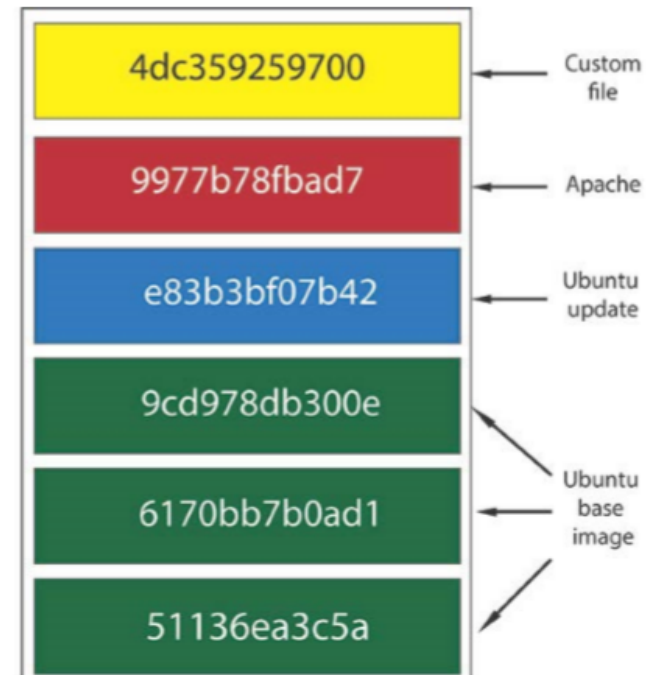
# Make port 80 available to the world outside this container
EXPOSE 80

# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python", "app.py"]
```

Docker image (2)

- Layered image
 - Each image consists of a *series of layers*
 - Docker uses *union file systems* to combine these layers into a single unified view
 - Layers are stacked on top of each other to form a base for a container's root file system
 - Based on the *copy-on-write* (COW) principle
- Layering pros
 - Enable layer reuse, installing common layers only once and saving bandwidth and storage space
 - Manage dependencies and separate concerns
 - Facilitate software specializations



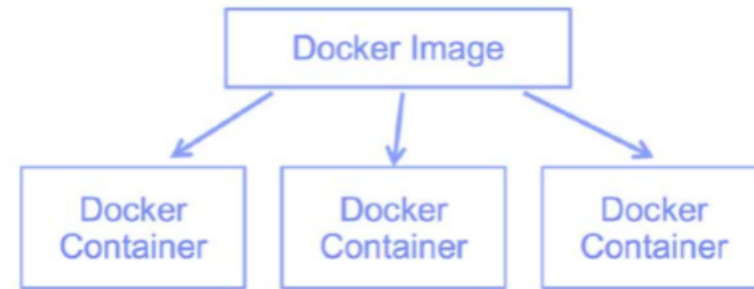
Docker image (3)

- Containers should be stateless. Ideally:
 - Very little data is written to a container's writable layer
 - Data should be written on Docker volumes
 - Nevertheless: some workloads require to write data to the container's writable layer
- The **storage driver** controls how *images* and *containers* are stored and managed on the Docker host.
- Docker supports multiple choices for the storage driver
 - Including AuFS, Device Mapper, Btrfs and OverlayFS
 - Storage driver's choice can affect the performance of the containerized applications

<https://docs.docker.com/storage/storagedriver/select-storage-driver/>

Docker container and registry

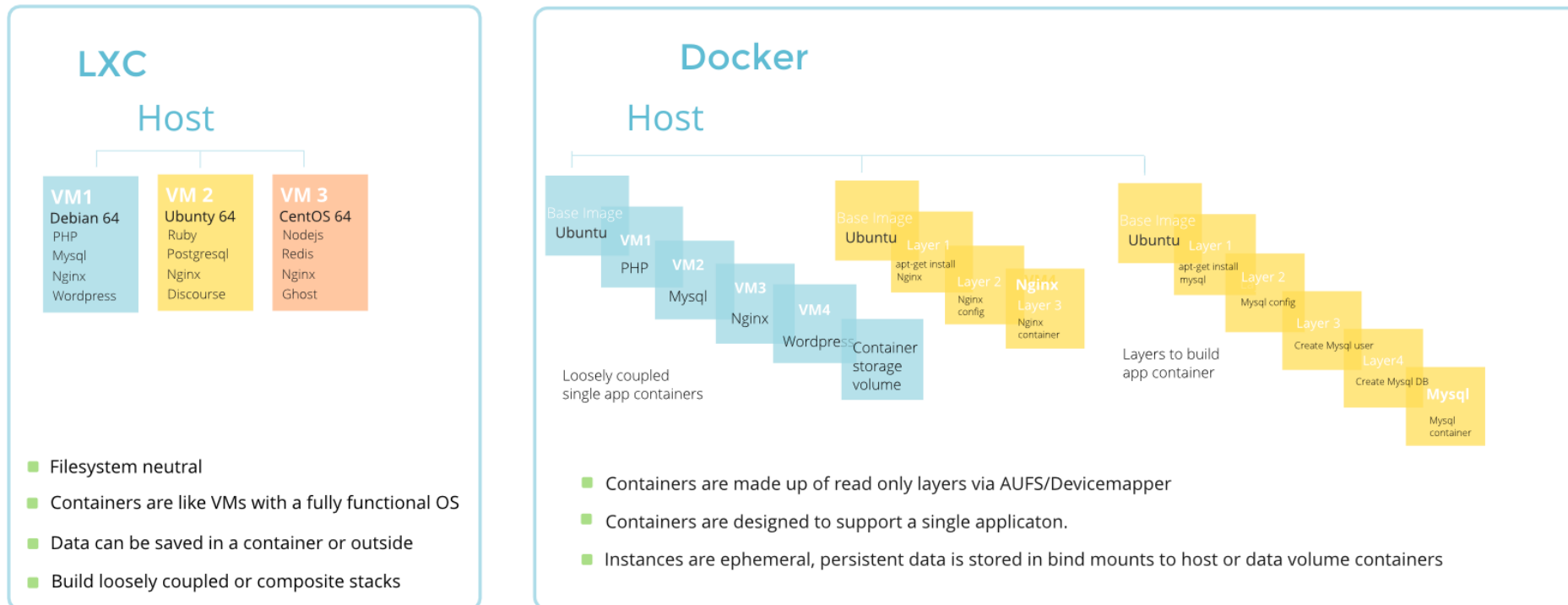
- **Docker container**: runnable instance of a Docker image
 - Run, start, stop, move, or delete a container using Docker API or CLI commands
 - The **Run** component of Docker
 - Docker containers are **stateless**: when a container is deleted, any data written that is not stored in a *data volume* is deleted along with the container
- **Docker registry**: stateless server side application that stores and lets you distribute Docker images
 - Provides an open library of images
 - The **Distribute** component of Docker
 - Docker-hosted registries: Docker Hub, Docker Store (open source and enterprise verified images)

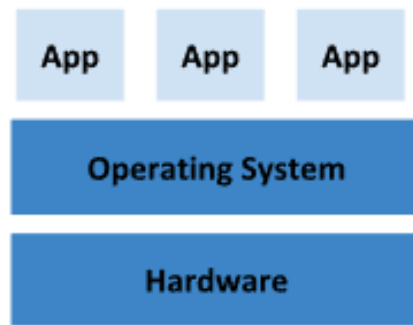


Docker vs LXC

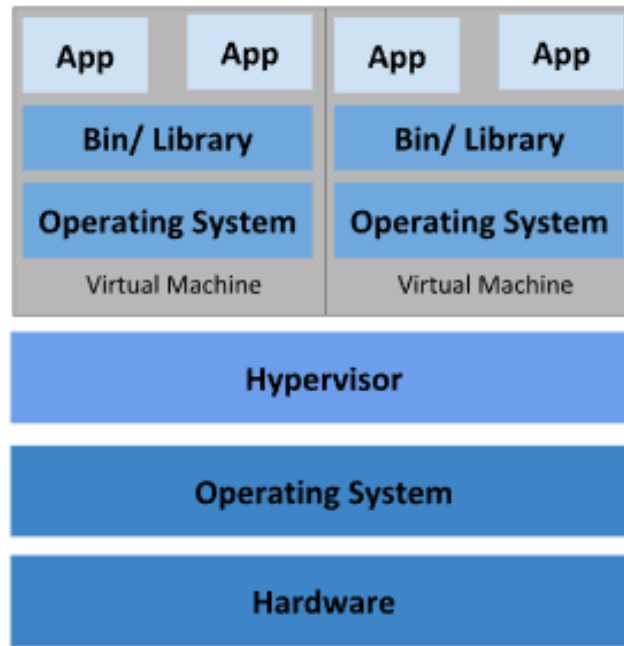
- Non overlapping solutions
- Main differences in the figure

Key differences between LXC and Docker

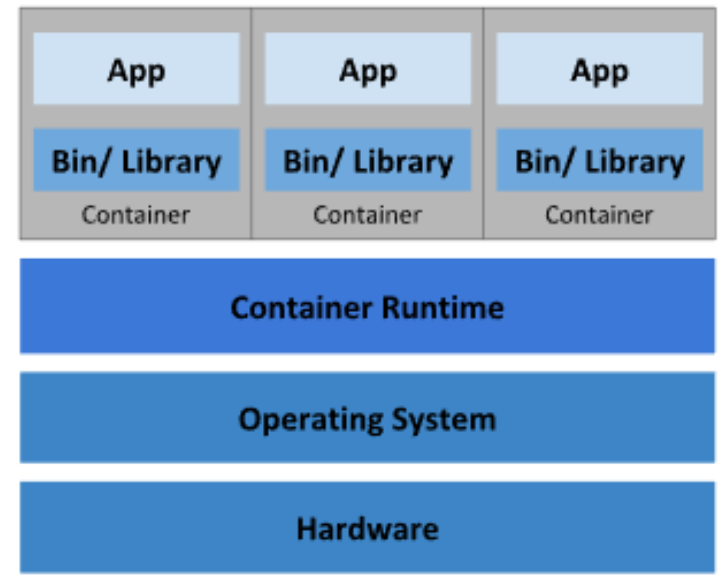




Traditional Deployment



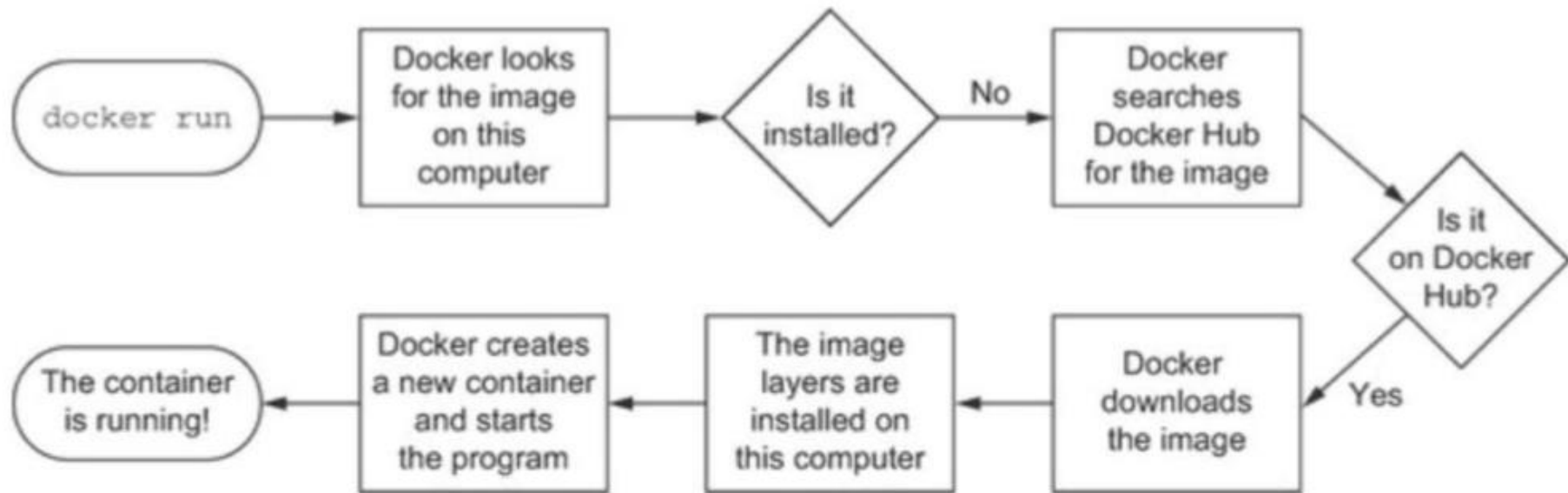
Virtualized Deployment



Container Deployment

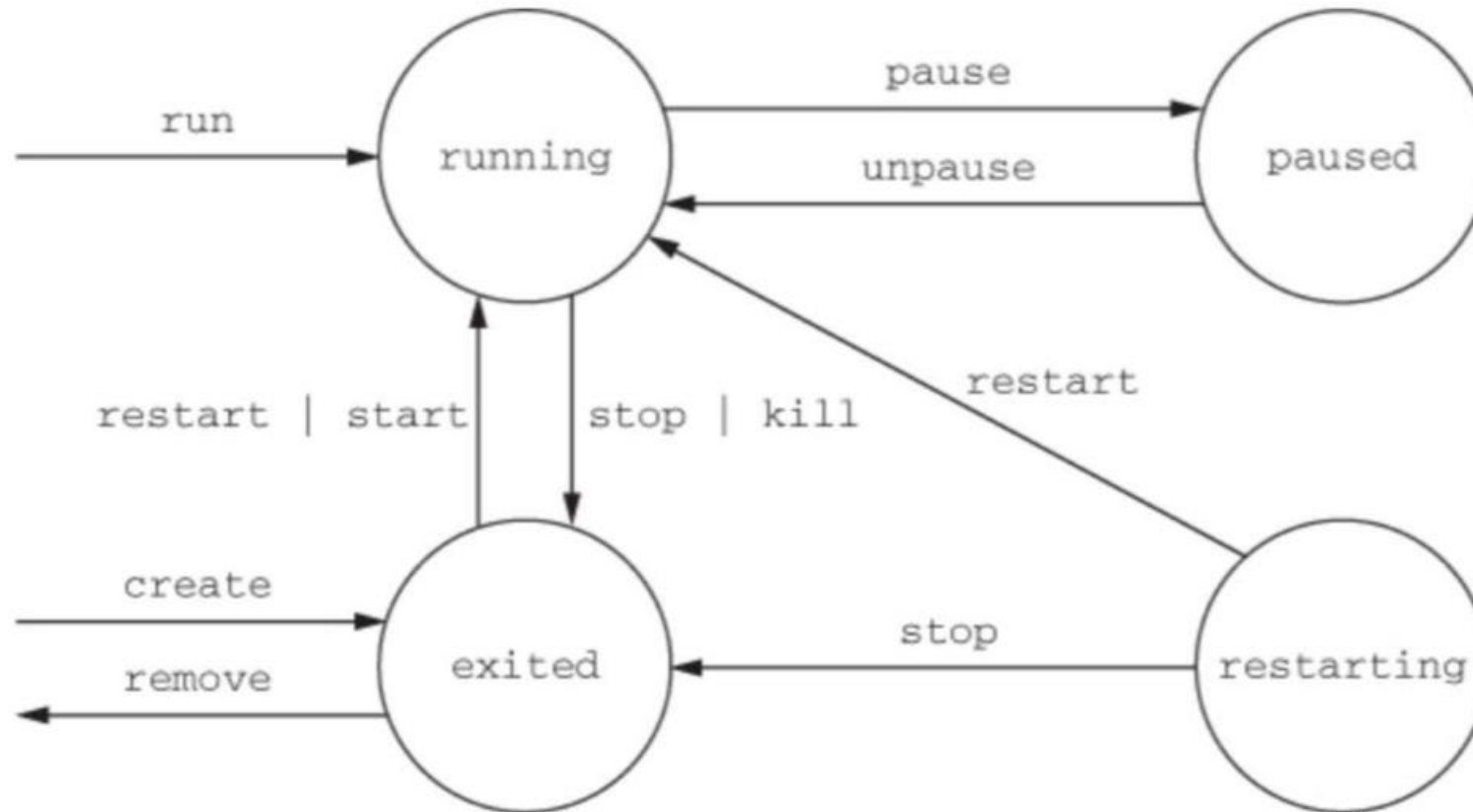
Docker: run command

- When you run a container whose image is not yet installed but is available on Docker Hub



Courtesy of "Docker in Action" by J. Nickoloff

State transitions of Docker containers



Courtesy of "Docker in Action" by J. Nickoloff

Commands: Info and image handling

- Obtain detailed info on your Docker installation
`$ docker info`
E.g., to know the used storage driver (e.g., AuFS)
- Download/update an image from the registry
`$ docker pull image-id`
- Upload an image to the registry
`$ docker push image-id`
- List images
`$ docker images`
- Inspect an image
`$ docker inspect image-id`
- Remove an image
`$ docker rmi image-id`

Command: Run

```
$ docker run [OPTIONS] IMAGE [COMMAND] [ARGS]
```

- Most common options
 - name assign a name to the container
 - d detached mode (in background)
 - it foreground with attached pseudo-tty and STDIN (interactive)
 - expose=[] expose a range of ports inside the container
 - p=[] publish a container's port or a range of ports to the host
 - v bind and mount a volume
 - e set environment variables
 - link=[] link to other containers
- The “Hello World” container

```
$ docker run ubuntu /bin/echo 'Hello world'
```

Command: Management

- List containers
 - Only running containers: `$ docker ps`
 - All containers (including exited ones): `$ docker ps -a`
- Container lifecycle
 - Stop container
 - `$ docker stop containerid`
 - Start stopped container
 - `$ docker start containerid`
 - Kill running container
 - `$ docker kill containerid`
 - Remove container
 - `$ docker rm containerid`
- Copy files from and to docker container
 - `$ docker cp containerid:path localpath`
 - `$ docker cp localpath containerid:path`

Some examples of using Docker (2)

- Running a web application in Docker
 - Also bind the container to a specific port

```
$ docker run -d -p 80:5000 training/webapp python app.py
```

- Stopping and removing a container

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	...
26ea7a6908bd	training/webapp	"python app.py"	4 seconds ago	...

...	STATUS	PORTS	NAMES	
...	Up 3 seconds	0.0.0.0:32768->5000/tcp	angry_chandrasekhar	

```
$ docker stop 26ea7a6908bd
```

```
$ docker rm 26ea7a6908bd
```

Some examples of using Docker (3)

- Running a Web server inside a container and sending an HTTP request through an interactive container

```
$ docker run -d --name web nginx:latest
```

```
$ docker run -i -t --link web:web  
    --name web_test busybox:latest  
    /bin/sh wget -O - http://web:80/ exit
```

- Checking the logs of a container

```
$ docker logs web
```

Docker Compose

To easily coordinate the execution of multiple services, we can use **Docker Compose**

- Read more at <https://docs.docker.com/compose/>



Docker Compose:

- is not bundled within the installation of Docker (on Linux)
- it can be installed following the official Docker documentation
 - <https://docs.docker.com/compose/install/>
- Allows to easily express the container to be instantiated at once, and the relations among them
- By itself, Docker compose runs the composition on a single machine; however, in combination with **Docker Swarm**, containers can be deployed on multiple nodes

Docker Compose

- We specify how to compose containers in a easy-to-read file, by default named `docker-compose.yml`
- To start the Docker composition (in background with -d):

```
$ docker-compose up -d
```

- To stop the Docker composition:

```
$ docker-compose down
```

- By default, Docker-compose looks for the `docker-compose.yml` file in the current working directory; we can change the file with the configuration using the `-f` flag

Docker Compose

- There are different versions of the Docker compose file format
- Latest: version 3 is supported from Docker Compose 1.13

```
version: '3'

services:
  storm-nimbus:
    image: storm
    container_name: nimbus
    command: storm nimbus
    depends_on:
      - zookeeper
    links:
      - zookeeper
    ports:
      - "6627:6627"
```

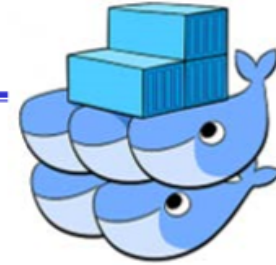


```
zookeeper:
  image: zookeeper
  container_name: zookeeper
  ports:
    - "2181:2181"

worker1:
  image: storm
  command: storm supervisor
  depends_on:
    - storm-nimbus
    - zookeeper
  links:
    - storm-nimbus
    - zookeeper
```

On the Docker compose file format: <https://docs.docker.com/compose/compose-file/>

Docker: Swarm mode



Docker includes the **swarm mode** for natively managing a cluster of Docker Engines, which is called **swarm**

- Read more at <https://docs.docker.com/engine/swarm/>

A *task* is a running container which is part of a swarm service

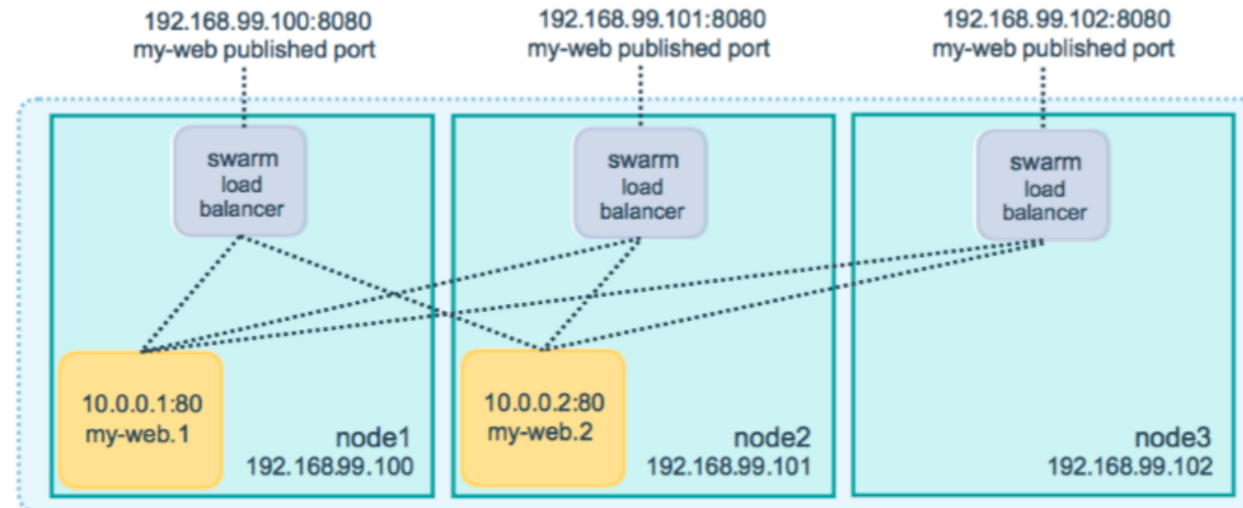
Basic features of the swarm mode:

- **Scaling:** it allows to declare the number of tasks for each service
- **State reconciliation:** swarm monitors the cluster state and reconciles any differences w.r.t. the expressed desired state
- **Multi-host networking:** it allows to specify an overlay network among services
- **Load balancing:** it allows to expose the ports for services to an external load balancer. Internally, the swarm lets you specify how to distribute service containers between nodes

Docker: Swarm mode

A swarm consists of multiple Docker hosts which run in swarm mode

- Node: an instance of the Docker engine
 - **Manager node** dispatches tasks to worker nodes
 - **Worker nodes** receive and execute tasks
- Load balancing
 - The swarm manager can automatically assign the service a (configurable) PublishedPort
 - External components can access the service on the PublishedPort. All nodes in the swarm route ingress connections to a running task



Commands: Swarm cluster

- Create a swarm: Manager node

```
$ docker swarm init --advertise-addr <MANAGER-IP>  
Swarm initialized: current node (<nodeid>) is now a manager.  
To add a worker to this swarm, run the following command:
```

```
docker swarm join --token <token> <manager-ip>:port
```

Create a swarm: Worker node

```
$ docker swarm join --token <token> <manager-ip>:port
```

- Inspect status

```
$ docker info
```

```
$ docker node ls
```

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS
<nodeid> *	controller	Ready	Active	Leader
<nodeid>	storage	Ready	Active	

Commands: Swarm cluster

- Leave the swarm

```
$ docker swarm leave
```

If the node is a manager node, you will receive a warning about maintaining the quorum. To override the warning, pass the `--force` flag

- After a node leaves the swarm, you can run the `docker node rm` command on a manager node to remove the node from the node list

```
$ docker node rm node-id
```

Commands: Manage Services

- Deploy a service to the swarm (from the manager node)

```
$ docker service create -d --replicas 1 \  
  --name helloworld alpine ping docker.com
```

- List running services

```
$ docker service ls
```

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
<serviceid>	helloworld	replicated	1/1	alpine:latest	

Commands: Manage Services

- Inspect the service

```
$ docker service inspect --pretty <SERVICE-ID>
$ docker service ps <SERVICE-ID>
```

ID	NAME	IMAGE	NODE	DESIRED ST	CURRENT ST	ERROR	PORTS
<cont.id1>	helloworld.1	alpine:latest	controller	Running	Running ...		
<cont.id2>	helloworld.2	alpine:latest	storage	Running	Running ...		

- Inspect the container

```
$ docker ps <cont.id1>
```

Manager node

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	... NAMES
<cont.id1>	alpine:latest	"ping docker.com"	2 min ago	Up 2 min	helloworld.1.iuk1sj...

Worker node

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	... NAMES
<cont.id2>	alpine:latest	"ping docker.com"	2 min ago	Up 2 min	helloworld.2.skfos4...

Commands: Manage Services

- Scale the service

```
$ docker service scale <SERVICE-ID>=<NUMBER-OF-TASKS>
```

The swarm manager will automatically enact the updates

- Apply rolling updates to a service

```
$ docker service update --image redis:3.0.7 redis
```

```
$ docker service update --replicas 2 helloworld
```

- Roll back an update

```
$ docker service rollback [OPTIONS] <SERVICE-ID>
```

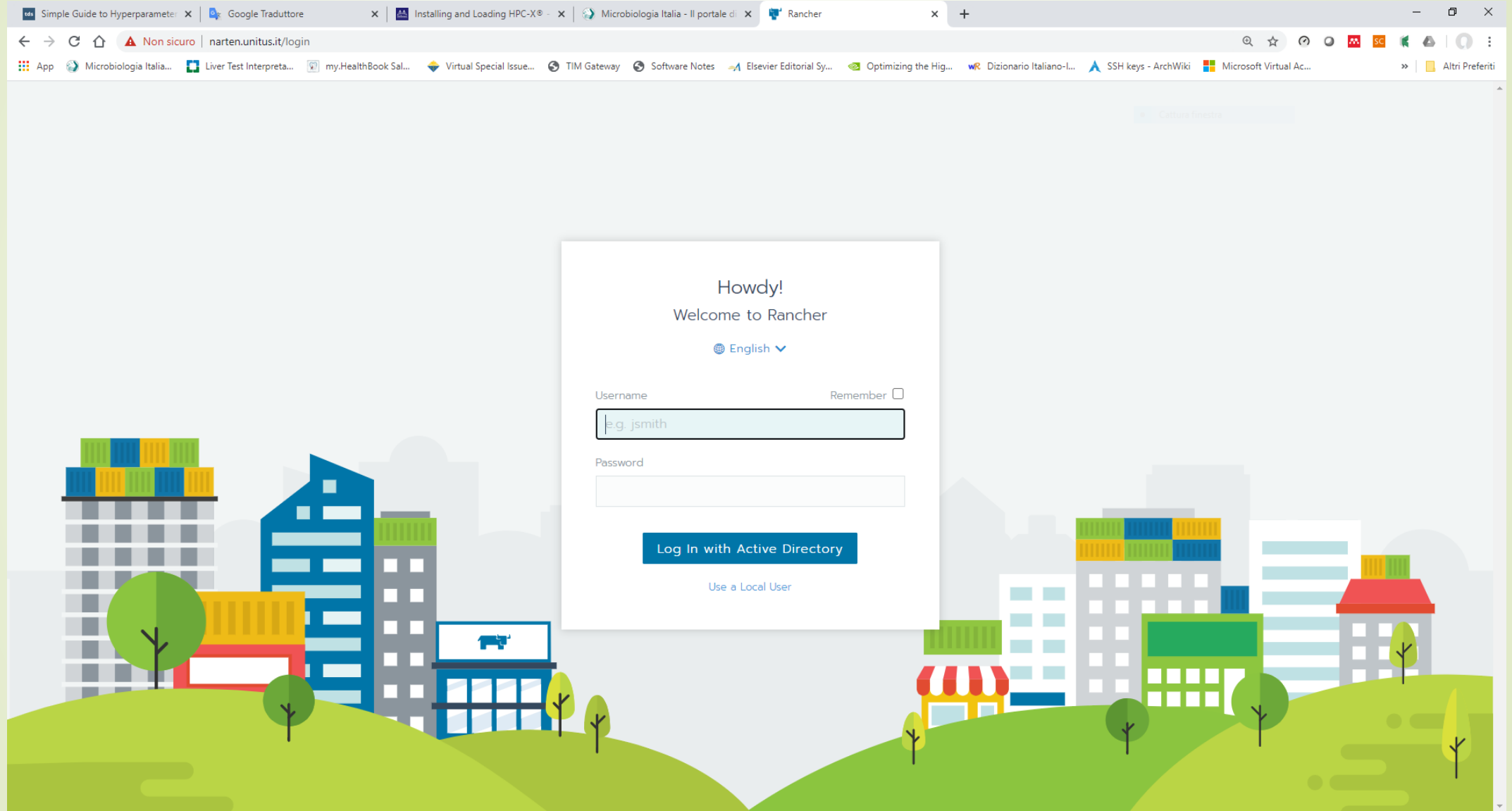
- Remove a service

```
$ docker service rm <SERVICE-ID>
```

All in all, for the impatient 😊

<https://docs.docker.com/get-started/>

Containers' orchestrator... Rancher



VM & Containers' orchestrator... Google Cloud

The screenshot shows the Google Cloud Platform (GCP) dashboard for a project named 'My First Project'. The interface is in Italian and displays various monitoring and management widgets.

- Informazioni sul progetto:** Shows project details such as 'Nome progetto: My First Project', 'ID progetto: gothic-space-276417', and 'Numero progetto: 940221018682'. It includes a link to 'AGGIUNGI PERSONE A QUESTO PROGETTO' and 'Vai alle impostazioni del progetto'.
- Risorse:** Lists 'Compute Engine' with '1 istanza'.
- Traccia:** States 'Non esistono dati traccia negli ultimi 7 giorni' and provides a link to 'Guida introduttiva di Stackdriver Trace'.
- Introduzione:** Offers links to 'API Esplora e abilita le API' and 'Implementa una soluzione preimpostata'.
- Compute Engine:** Features a line graph for 'CPU (%)' utilization. The y-axis ranges from 0.02 to 0.14. A legend indicates 'instance/cpu/utilization: 0,04'. A link 'Vai a Compute Engine' is provided.
- API API:** Features a line graph for 'Richieste (richieste/sec)'. The y-axis ranges from 0 to 0.8. A legend indicates 'Richieste: 0,20'.
- Stato di Google Cloud Platform:** Displays 'Tutti i servizi sono normali' and a link 'Vai alla dashboard dello stato di Cloud'.
- Fatturazione:** Shows 'Addebiti stimati: 0,00 EUR €' for the period '01-14 giu 2020' and a link 'Visualizza le spese dettagliate'.
- Monitoring:** Includes options to 'Configura criteri di avviso', 'Crea controlli di uptime', and 'Visualizza tutte le dashboard'. A link 'Vai a Monitoring' is also present.
- Error Reporting:** States 'Non ci sono errori. Hai configurato Error Reporting?' and a link 'Scopri come impostare Error Reporting'.


VM & Containers' orchestrator... Azure Cloud

Microsoft Azure

Cerca risorse, servizi e documentazione (G+)

Benvenuti in Azure.


Non si ha una sottoscrizione? Vedere le opzioni seguenti.



Inizia con una versione di valutazione gratuita

Offri un credito gratuito di \$ 200 per l'acquisto di prodotti e servizi di Azure, oltre a 12 mesi di [servizi gratuiti](#) tra quelli più diffusi.


[Avvia](#) [Altre informazioni](#)



Gestisci Azure Active Directory

Gestire gli accessi, impostare criteri intelligenti e ottimizzare la sicurezza con Azure Active Directory.

[Vista](#) [Altre informazioni](#)












Accedi ai vantaggi per gli studenti





Dopo aver verificato il livello accademico, sarà possibile ottenere software gratuito, credito Azure o l'accesso ad Azure Dev Tools for Teaching.

[Esplora](#) [Altre informazioni](#)





Servizi di Azure

[Crea una risorsa](#)  Macchine virtuali  Servizi app  Account di archiviazione  Database SQL  Database di Azure per i...  Azure Cosmos DB  Servizi Kubernetes  App per le funzioni  Altri servizi

Esplora

 Sottoscrizioni  Gruppi di risorse  Tutte le risorse  Dashboard

Strumenti

 Microsoft Learn [Approfondire le conoscenze](#)  Monitoraggio di Azure [Monitorare le app e](#)  Centro sicurezza [Proteggere le app e](#)  Gestione dei costi [Analizzare e ottimizzare](#)

<https://portal.azure.com/#create/hub>