Globant

Pisa 21-02-2024

Containerization

Globant

Daniele Trezza

Senior java developer

Engineering studio

Research and development

Tutor

Nicholas Ferrari

Senior java developer

Cloud studio

Research and development

Menthor

- **33** Countries
- **5** Continents
- **+29K** Globers

We are behind the **digital transformations of brands like**

Google Santander EA Disney

Be kind culture

Career opportunities in **+3500 projects** worldwide

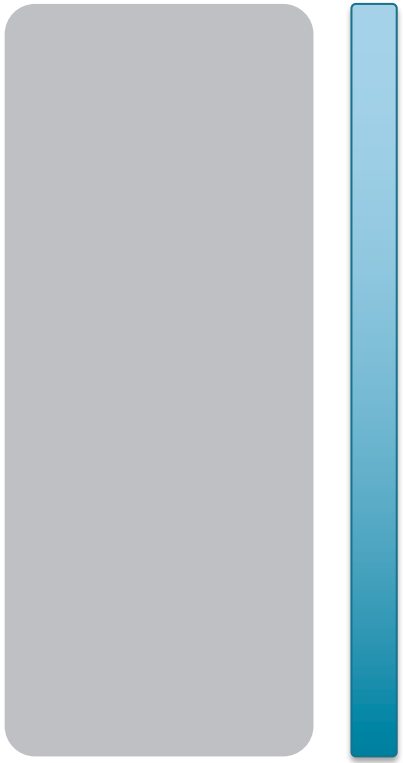
Globant

MA I CONTAINER... SERVONO?

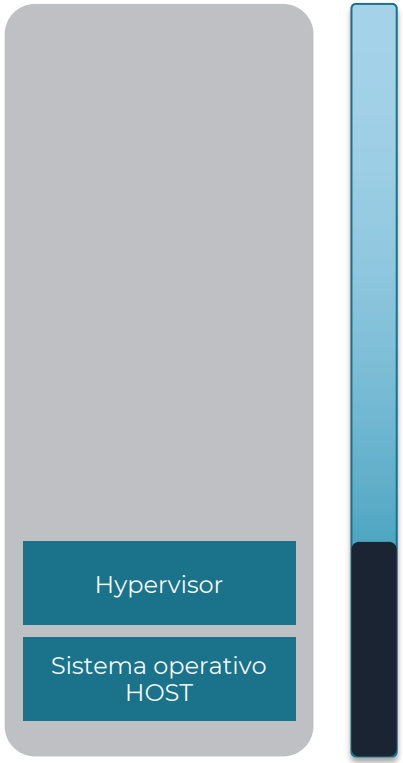
Era meglio prima (?)

VMs VS Containers

VMs VS Containers

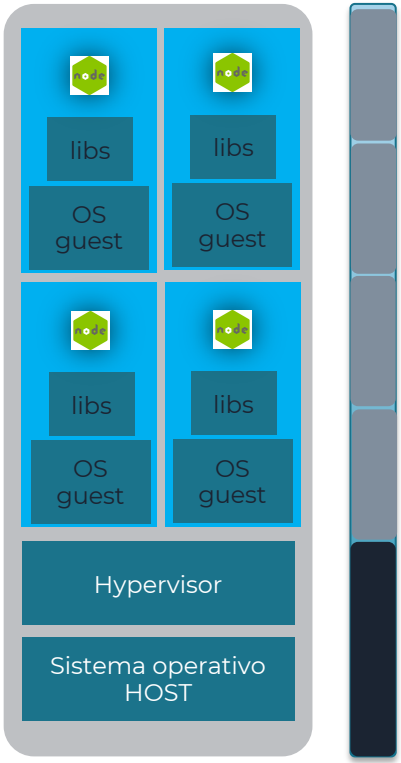


VMs VS Containers

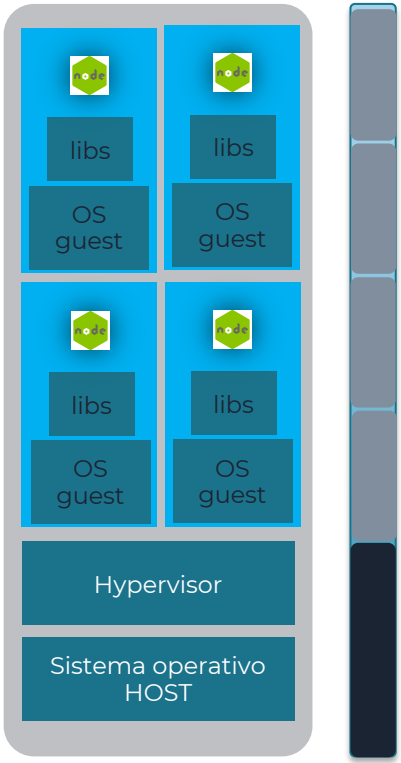


VMs VS Containers

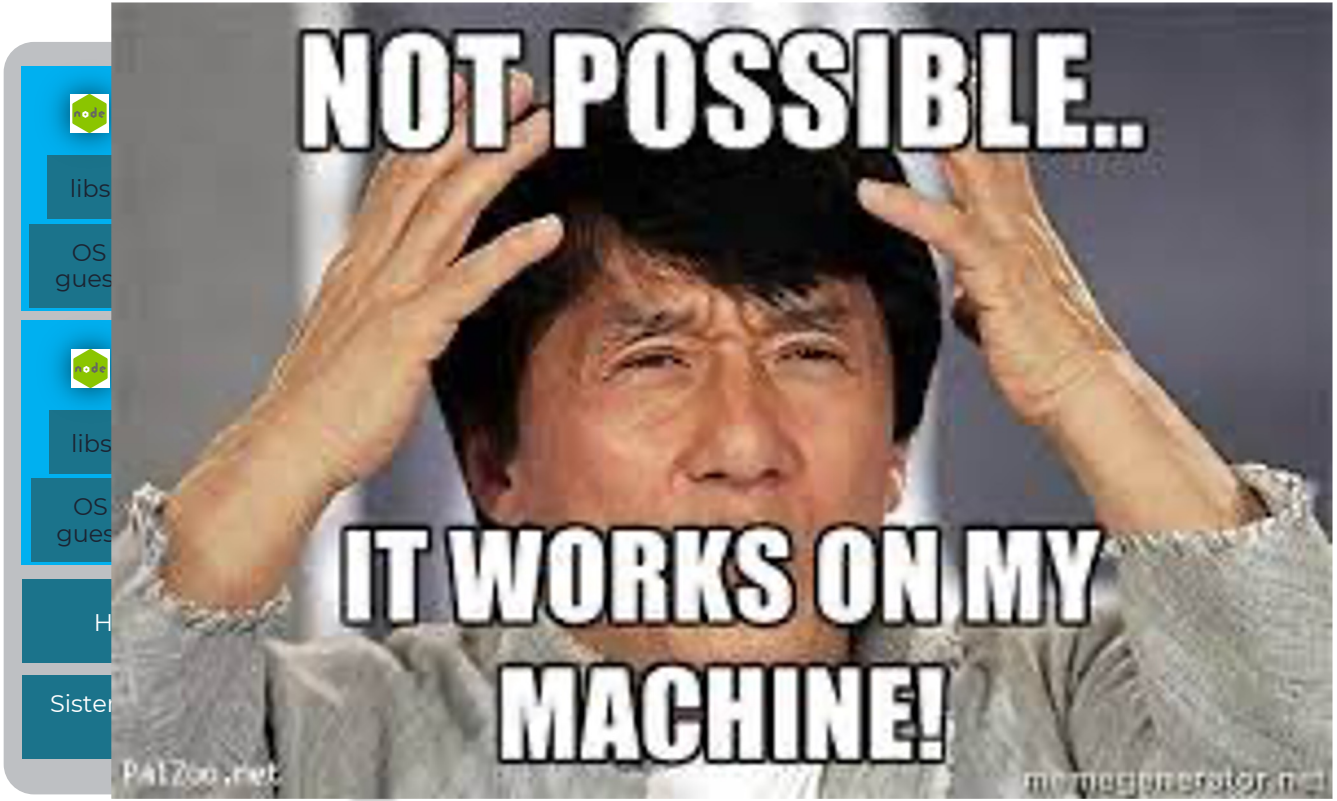
VMs VS Containers



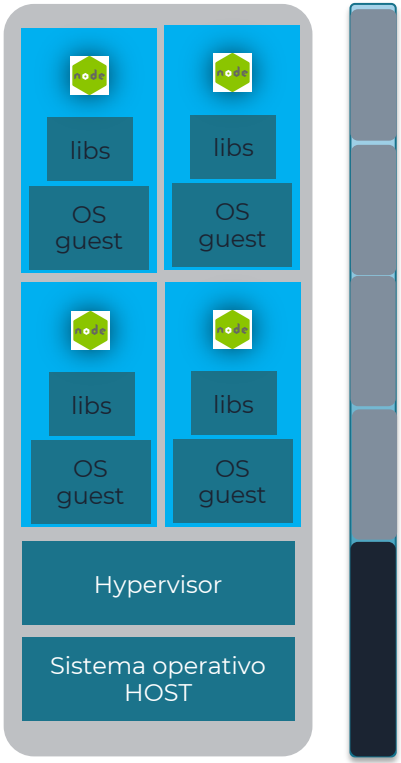
VMs VS Containers

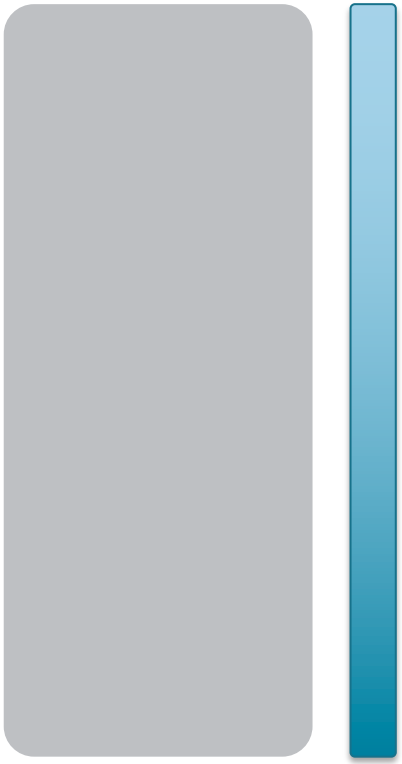


VMs VS Containers

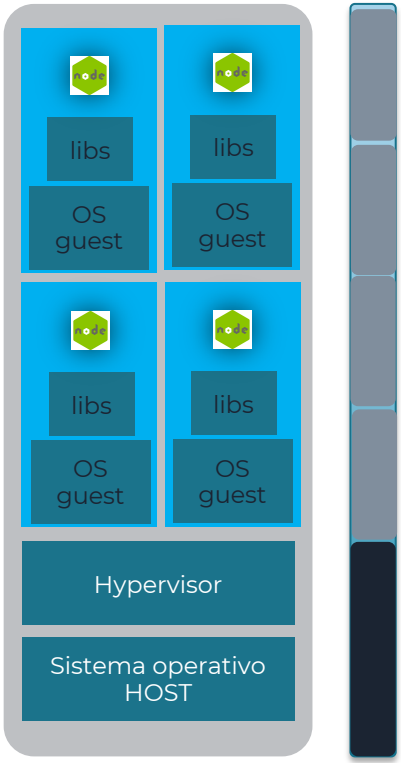


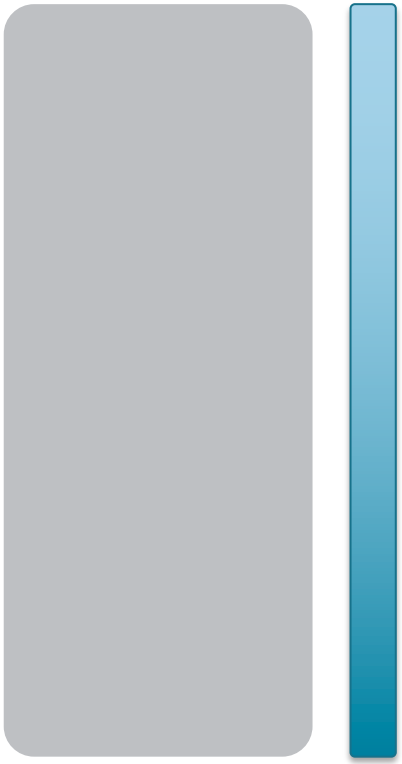
VMs VS Containers



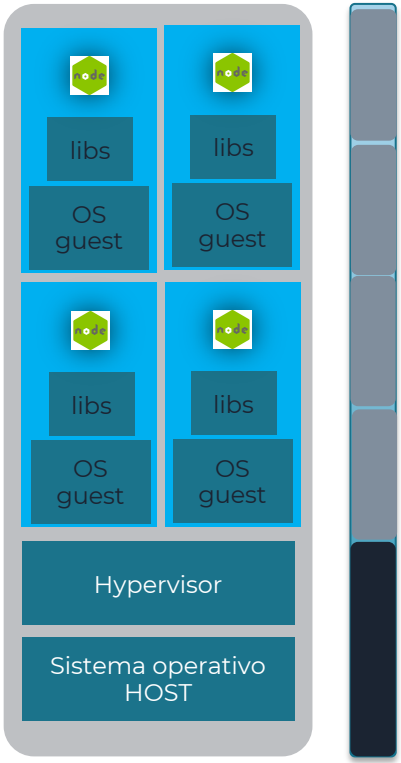


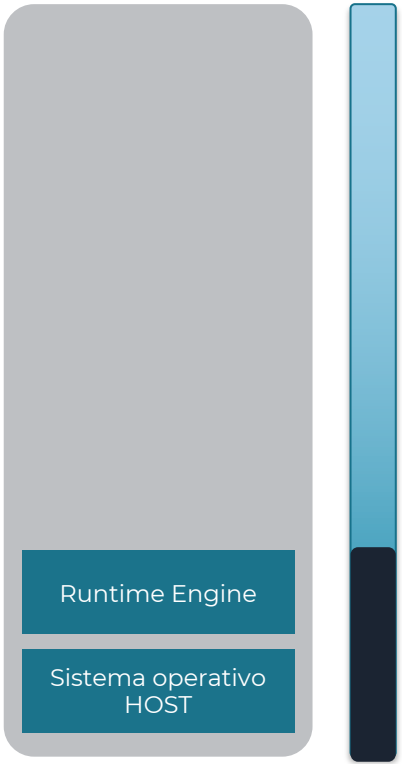
VMs VS Containers



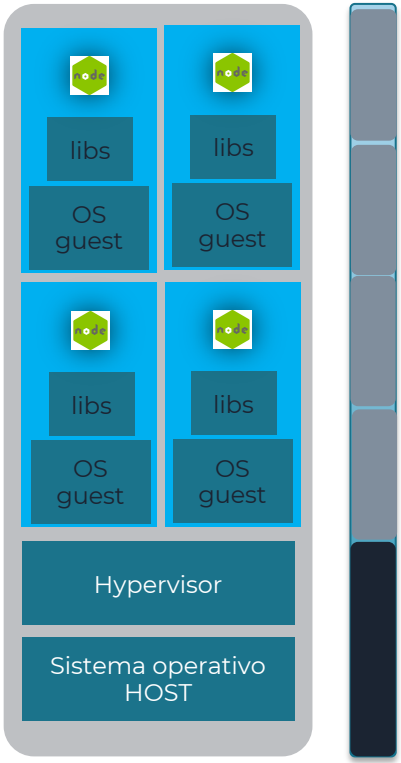


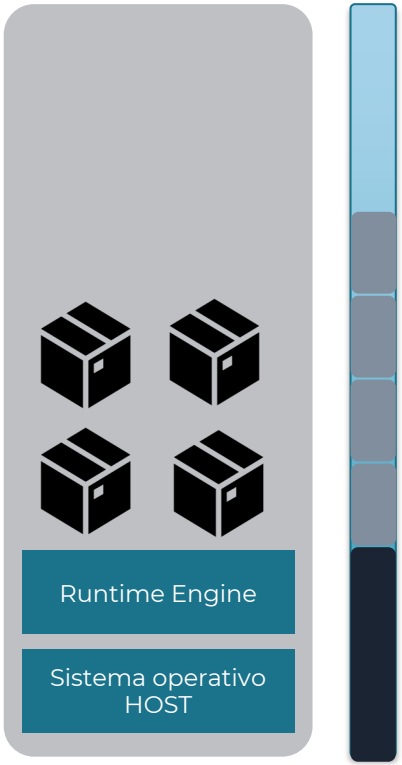
VMs VS Containers



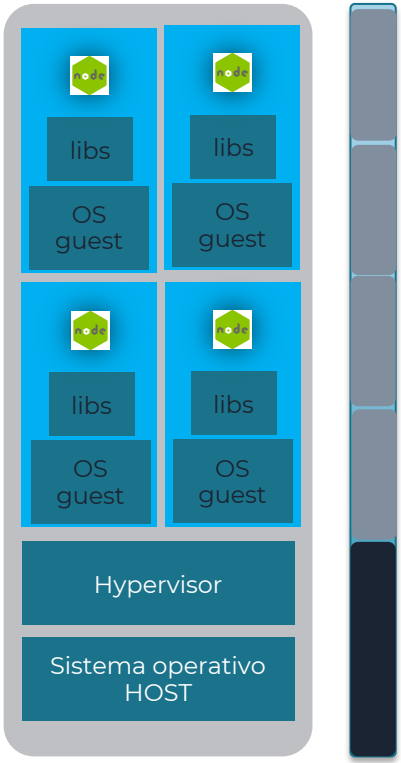


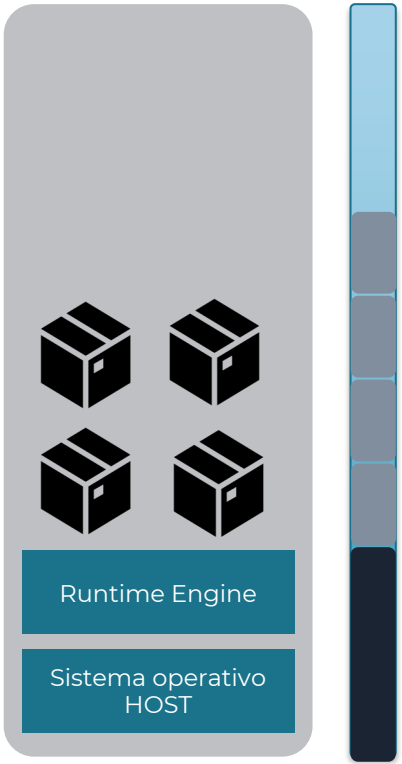
VMs VS Containers



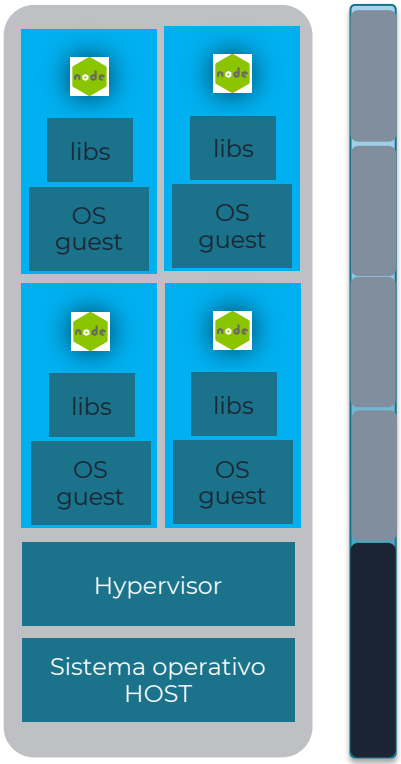


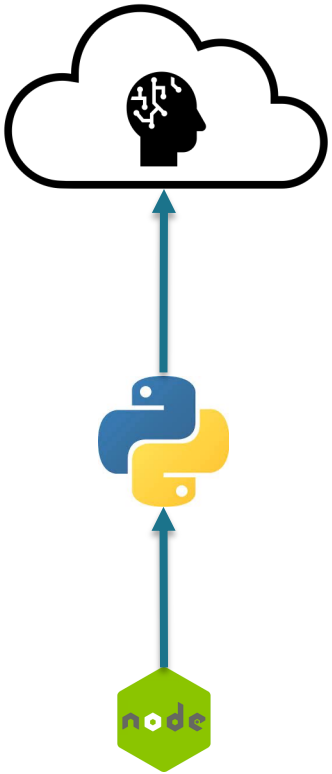
VMs VS Containers

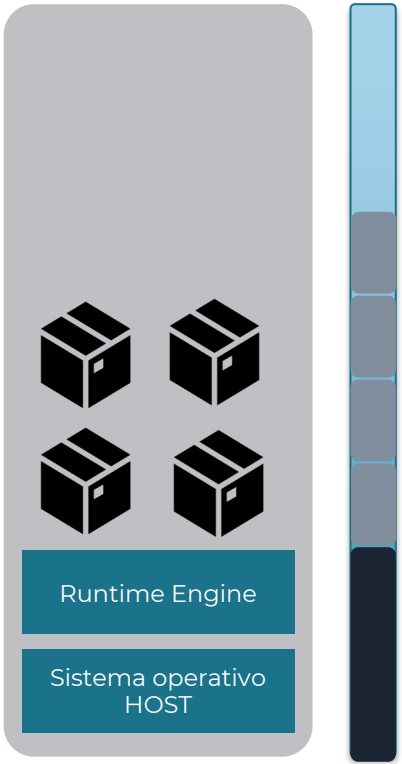




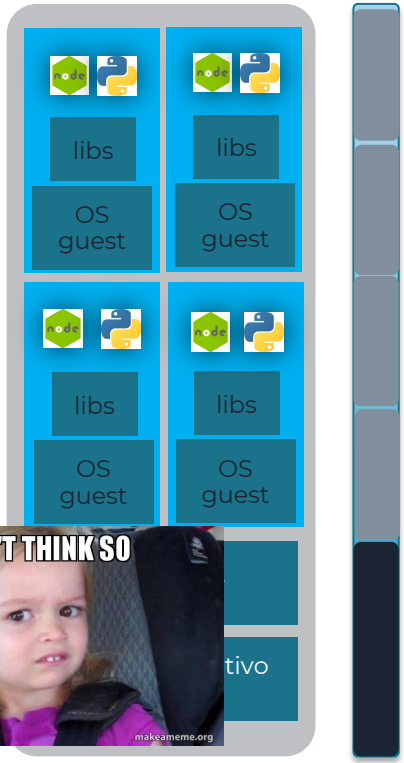
VMs VS Containers

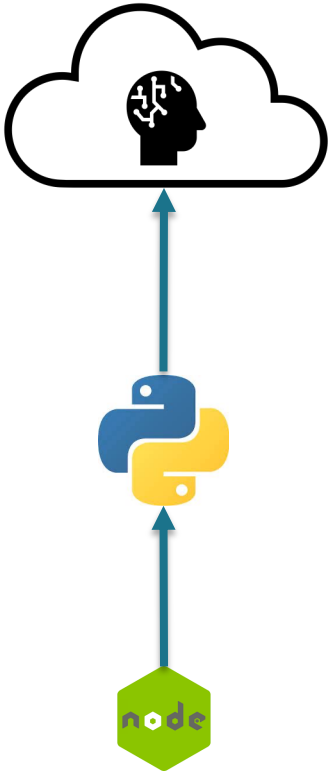


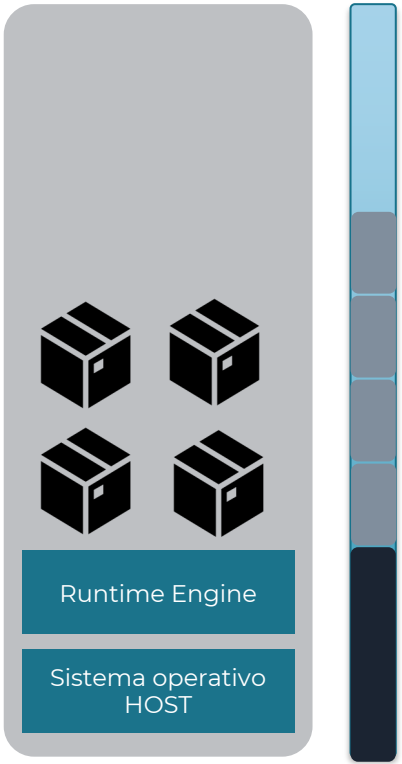




VMs VS Containers

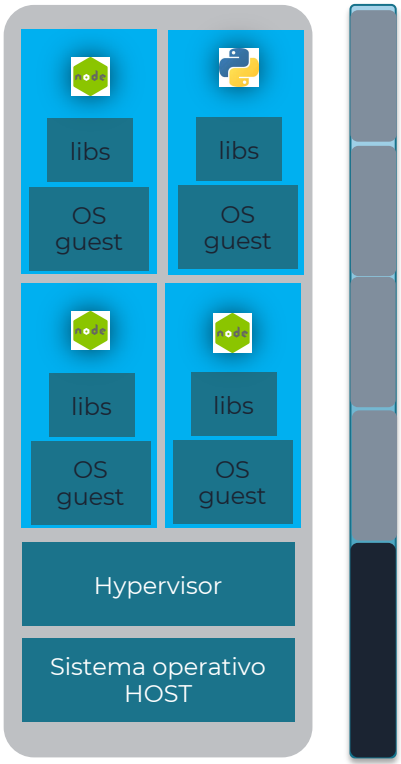


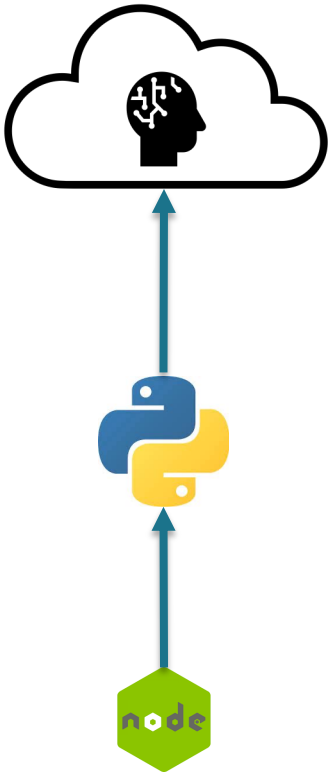


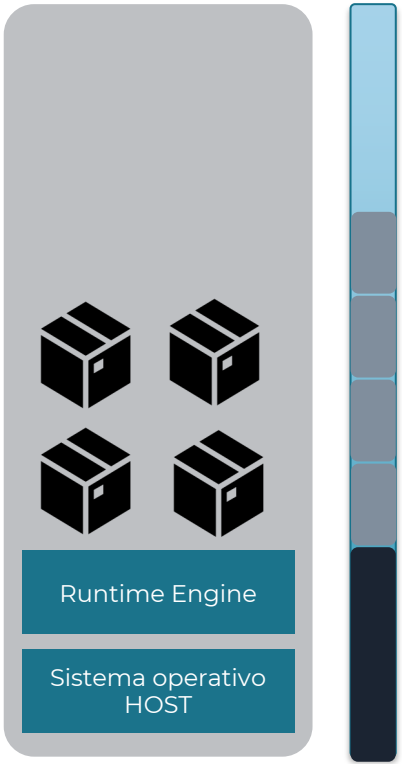


Globant

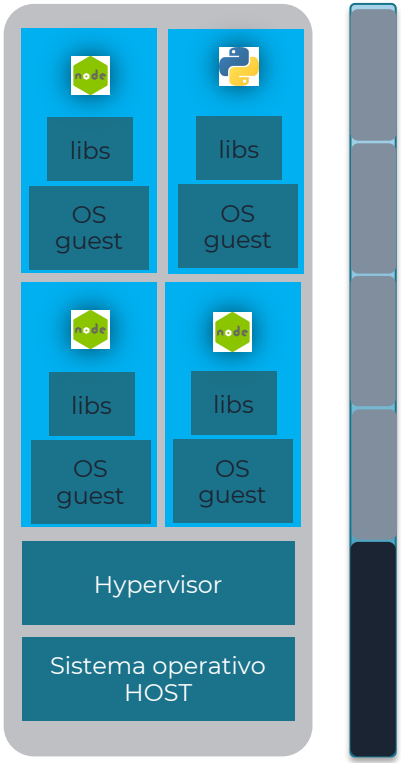
VMs VS Containers

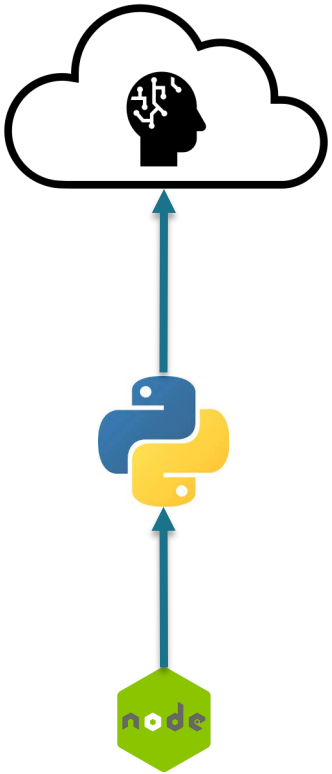


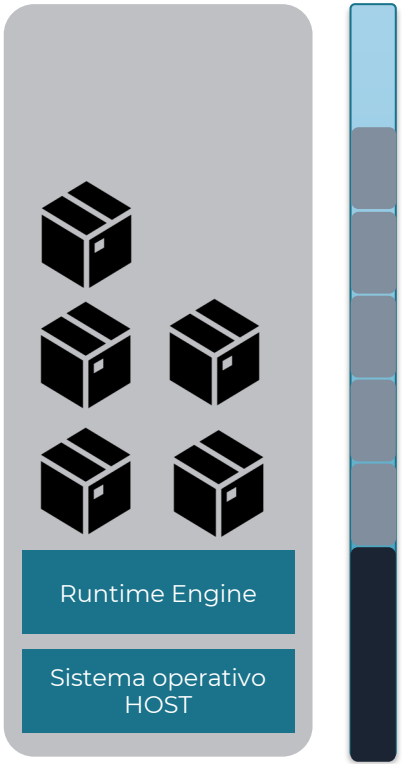




VMs VS Containers





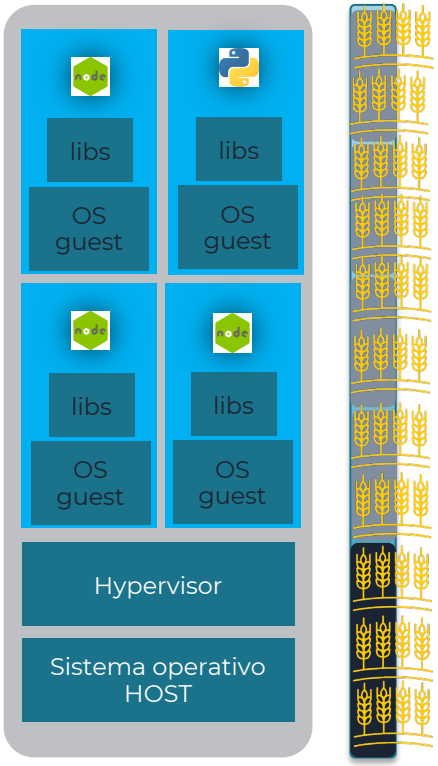


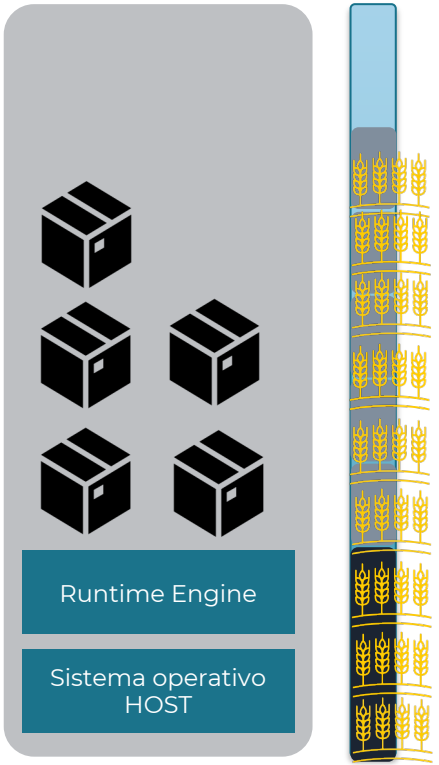
Costi

Ma io sto sul cloud...

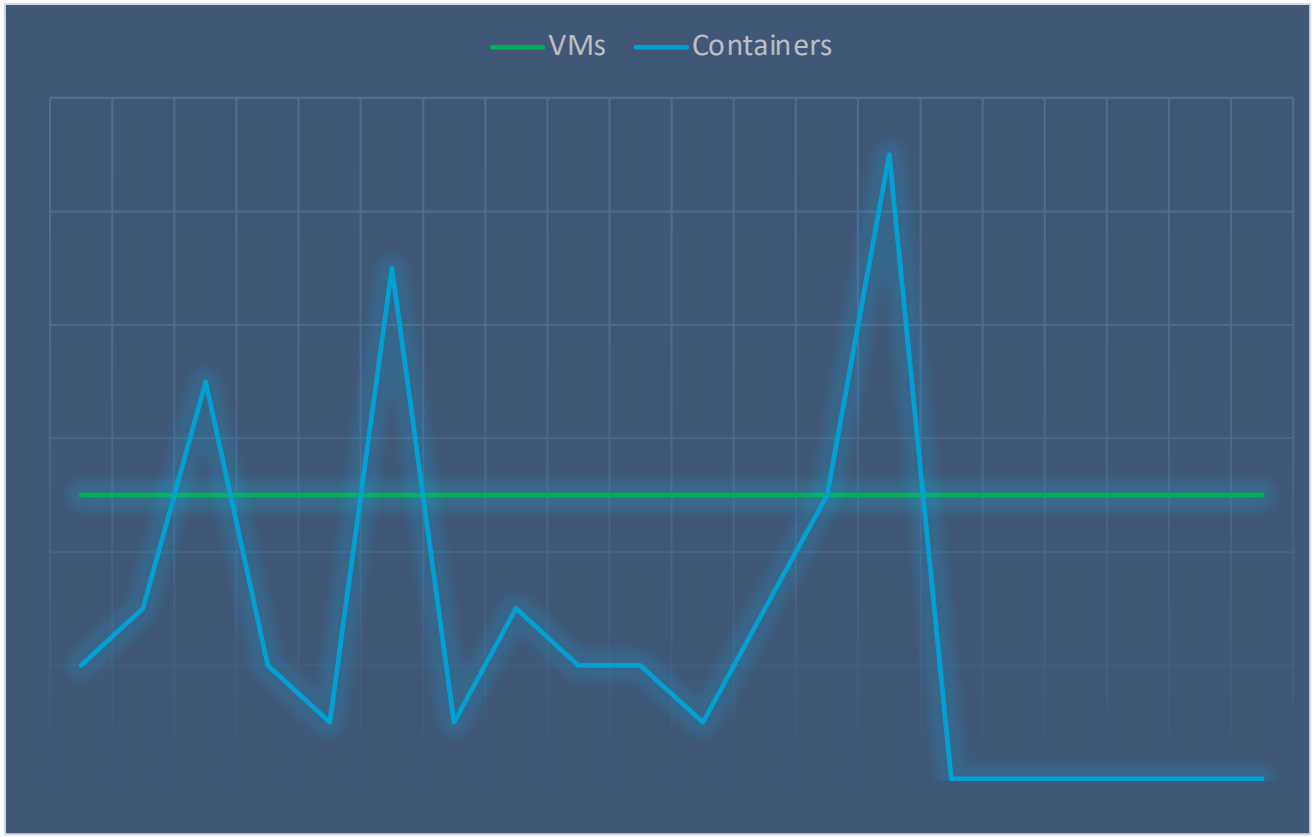
Non ho hardware...

Costi

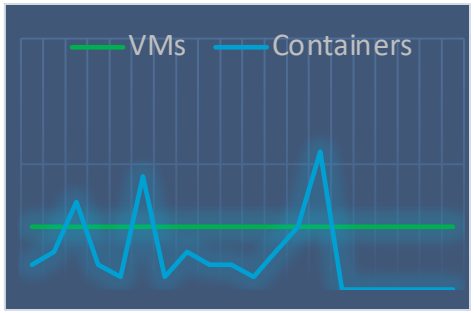

libs
OS guest
libs
OS guest
libs
OS guest
libs
OS guest
Hypervisor
Sistema operativo HOST


Runtime Engine
Sistema operativo HOST

Costi



Costi



Per il programmatore:

-Pensa modulare

-Test semplificato

-Troubleshooting rapido

-Divisione delle responsabilità

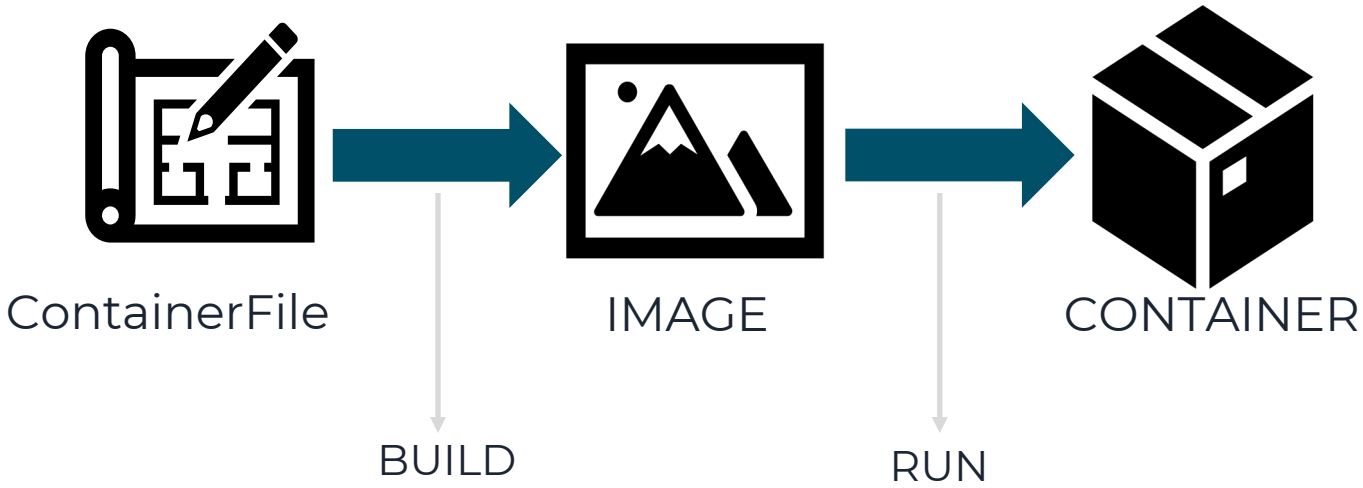
TERMINOLOGIA

TERMINOLOGIA

Il vocabolario base

Globant

From the image to the container



TERMINOLOGIA

Layering

Globant

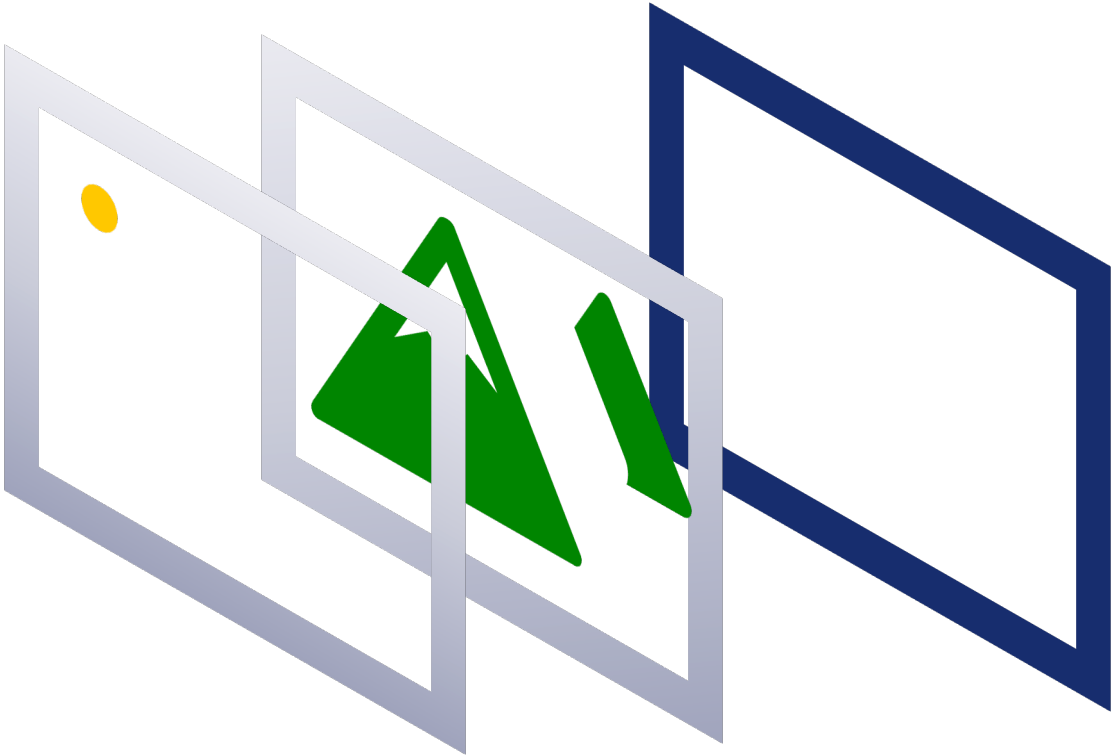
TERMINOLOGIA

Layering

Globant

TERMINOLOGIA

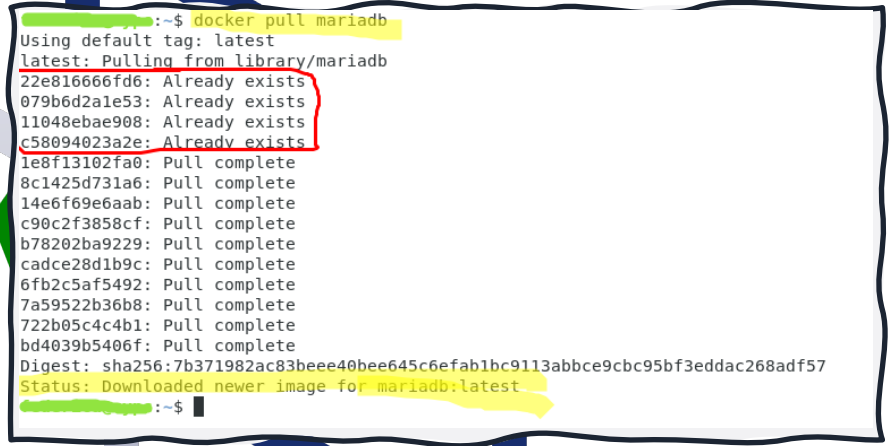
Layering



Globant

TERMINOLOGIA

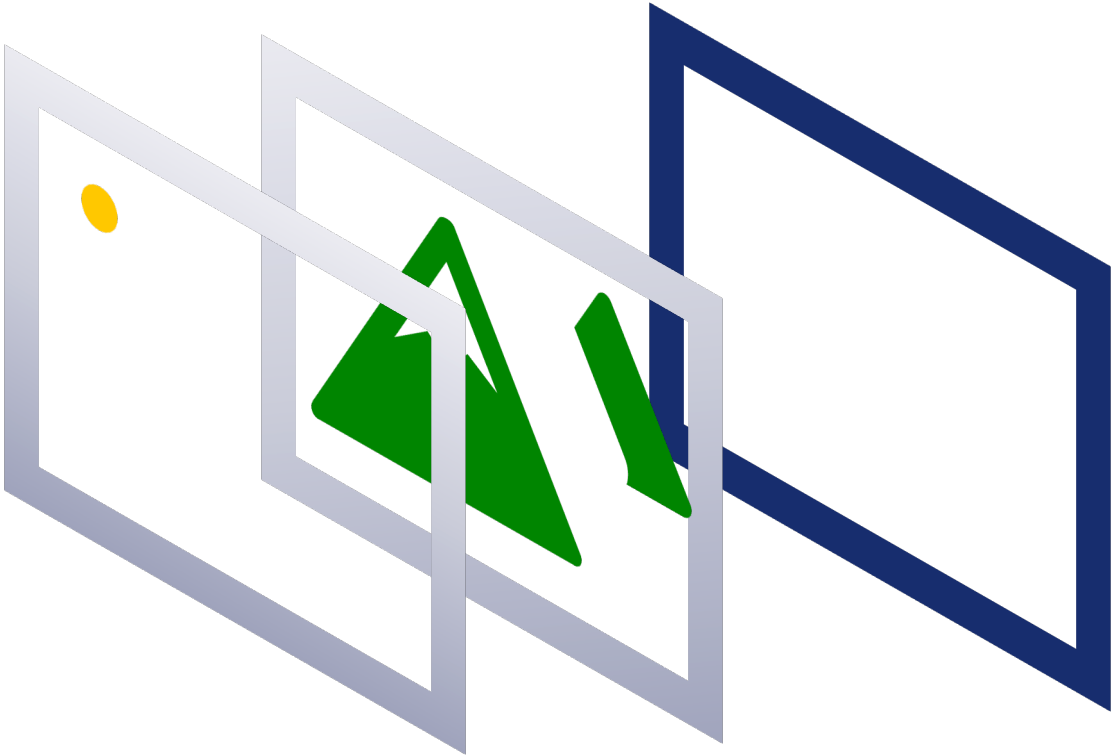
Layering



Globant

TERMINOLOGIA

Layering



Globant

TERMINOLOGIA

Layering

Globant

TERMINOLOGIA

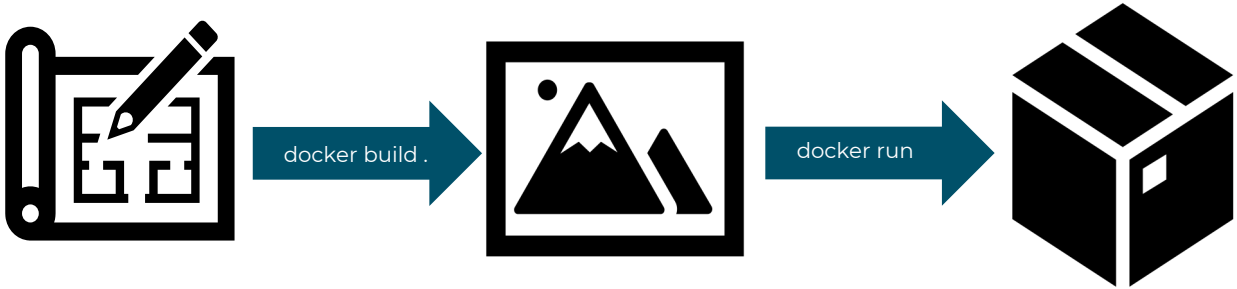
Layering

Globant

From the image to the container

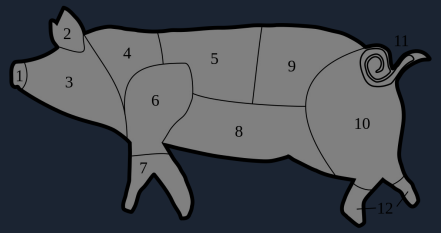
TERMINOLOGIA

From the image to the container



Globant

ANATOMY OF A CONTAINERFILE



ANATOMY OF A CONTAINERFILE

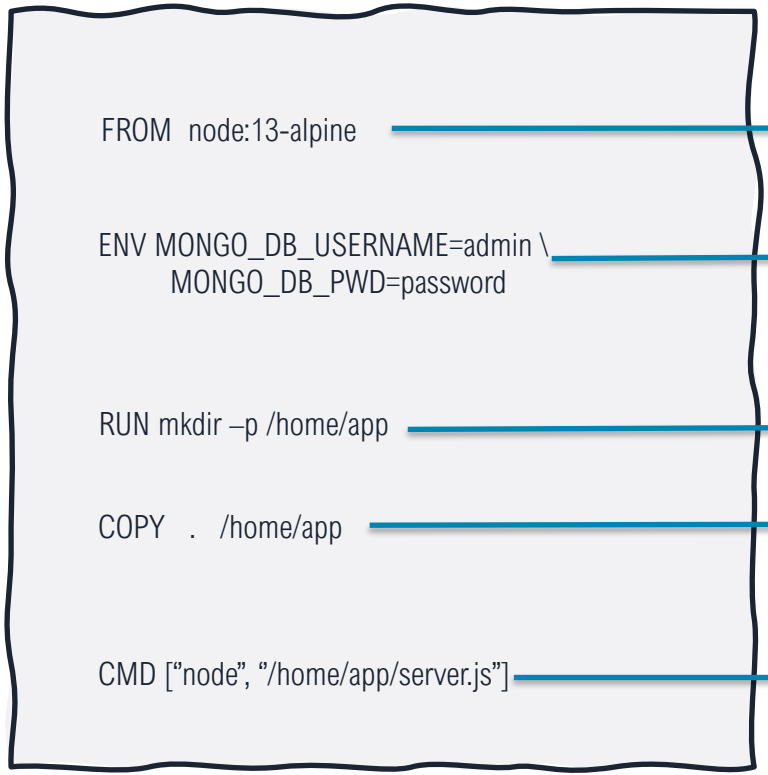
Basic commands

```
FROM  node:13-alpine

ENV MONGO_DB_USERNAME=admin \
        MONGO_DB_PWD=password

RUN mkdir –p /home/app

COPY  .  /home/app

CMD [“node”, “/home/app/server.js”]
```

Globant

ANATOMY OF A CONTAINERFILE

Basic commands



Ogni containerfile ha sempre come primo comando il FROM e il nome di una **immagine da cui partire**.

La definizione di variabili di ambiente è **opzionale**.
Queste variabili saranno visibili solo **dall’interno** del container.

Permette di eseguire **qualunque comando** linux, **all’interno** del container.

Copia i file **dall’host al container**.
Unico comando che agisce anche sull’HOST.

Il **primo comando** da eseguire quando parte il container.

Globant

ANATOMY OF A CONTAINERFILE

PRO commands

ARG : il FROM può prendere degli argomenti dichiarati dalle istruzioni ARG che lo precedono.
ARG può essere usato anche per dichiarare delle variabili interne.

LABEL : aggiunge metadati.

EXPOSE : informa che il container ascolterà su una determinata porta. NON pubblica la porta. Per pubblicare la porta si usa l'opzione –p quando si fa la run.

ADD : è un copy più potente. Si può specificare un URL da copiare nel filesystem del container. Se si passa come primo argomento un file compresso, questo verrà decompresso nel container.

ENTRYPOINT : comando da eseguire quando il container parte. Senza Entrypoint di default si userà /bin/sh -c bash. CMD rimane obbligatorio, e rappresenta il parametro di default.

VOLUME : specifica il volume, ossia dove va mantenuto lo stato del container.

WORKDIR : setta la directory di lavoro per le istruzioni che lo seguono.

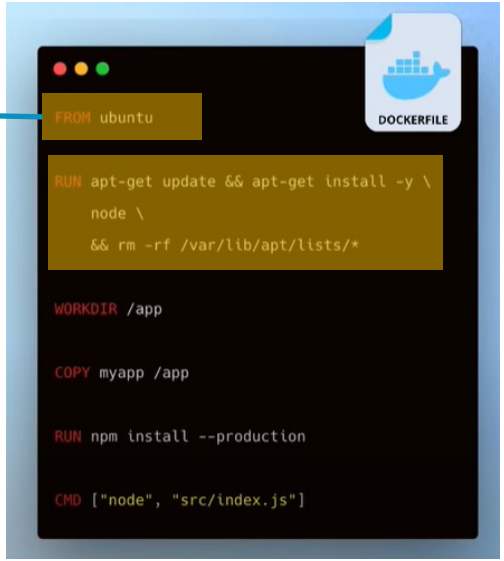
ONBUILD : Utile per immagini che so saranno usate per derivare altre immagini. Permette di impostare i primi comandi che verranno eseguiti alla creazione della prossima immagine. Vedi Maven.

BEST PRACTICES

BEST PRACTICES

1 - USA IMMAGINI DI PARTENZA UFFICIALI E VICINE ALLA TUA NECESSITA’

FROM node



Così il containerFile viene più pulito, e in più l’immagine di partenza sarà già costruita seguendo le best practices

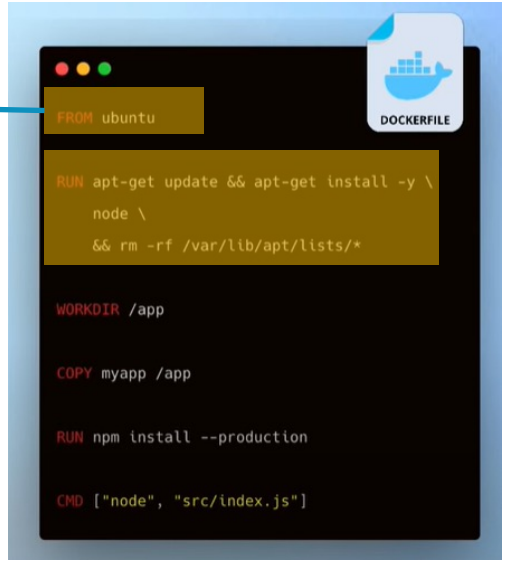
Globant

BEST PRACTICES

2 – EVITA IL LATEST

Non puoi prevedere che la versione latest supporterà sempre le tue funzionalità
Due build diverse fatte in momenti diversi potrebbero generare risultati diversi

FROM node17.0.1



Globant

BEST PRACTICES

3 – OTTIMIZZA LA TUA CACHE

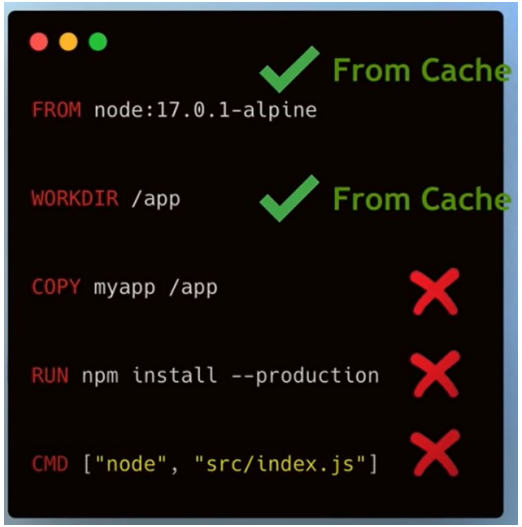
Ho un containerFile in cui prima faccio la copia della mia app node e poi installo le dipendenze.

Globant

BEST PRACTICES

3 – OTTIMIZZA LA TUA CACHE

Se faccio una modifica al codice, MA non alle dipendenze, il comando npm install non verrà preso da cache!



I layer che stanno sopra al layer modificato vengono tutti cancellati dalla cache

Globant

BEST PRACTICES

3 – OTTIMIZZA LA TUA CACHE

Vorrei che npm install venga eseguito solo quando il file package.json viene modificato

Prima copio il file package, faccio npm install e alla fine copio dentro l'applicazione.

```
FROM node:17.0.1-alpine

WORKDIR /app

COPY package.json package-lock.json .

RUN npm install --production

COPY myapp /app

CMD ["node", "src/index.js"]
```

IN GENERALE: Ordinare i comandi dal meno soggetto a cambiamenti al più soggetto a cambiamenti

BEST PRACTICES

4 – IL FILE .containerignore è tuo amico

.dockerignore

Una applicazione in run non ha bisogno di tutti i file che abbiamo sul repository....

```
# ignore .git and .cache folders
.git
.cache

# ignore all markdown files (md)
*.md

# ignore sensitive files
private.key
settings.json
```

CARTELLE AUTO-GENERATE... README... CARTELLA DI GIT...

Globant

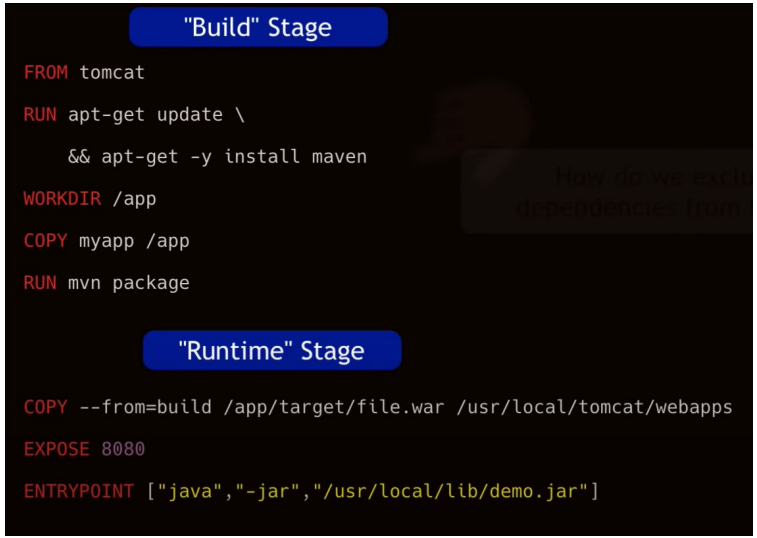
BEST PRACTICES

5 – MULTI-STAGE BUILDS

Ci sono contenuti di cui hai bisogno in fase di build, ma non quando l'applicazione gira
DEVELOPMENT TOOLS...BUILD TOOLS...TEST DEPENDENCIES... TEMPORARY FILES

Esempi:

In una Java Application, hai bisogno della **JDK** solo per compilarla, ma non per farla girare.
Maven o **Gradle** non sono necessari quando in run.



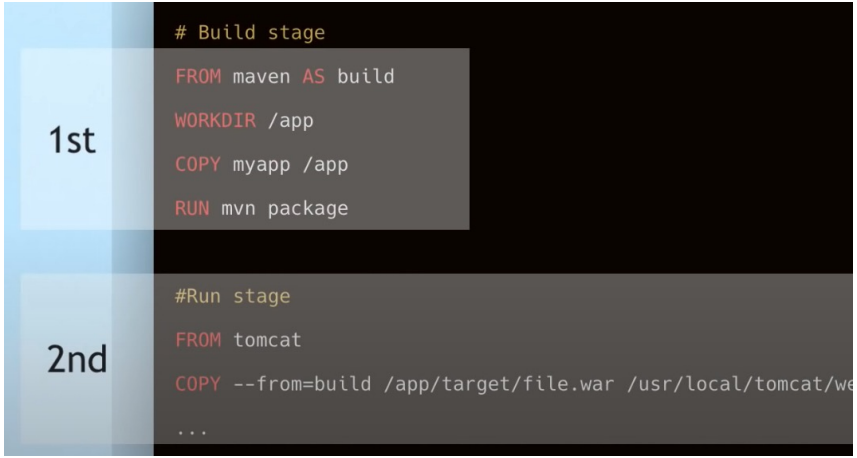
COME SI FA A SEPARARE LA FASE DI BUILD DALLA FASE DI RUNTIME?

Globant

BEST PRACTICES

5 – MULTI-STAGE BUILDS

Ci sono contenuti di cui hai bisogno in fase di build, ma non quando l'applicazione gira
DEVELOPMENT TOOLS…BUILD TOOLS…TEST DEPENDENCIES… TEMPORARY FILES



Ogni From inizia una immagine nuova, con –from= build richiamo l'artifact di una immagine precedente.

Solo l'ultima sarà il risultato della build

Globant

BEST PRACTICES

6 – USA UTENTI CON POCHI PRIVILEGI

Soprattutto se stai usando *docker*...

CON UN UTENTE CREATO AD-HOC

```
...

# create group and user
RUN groupadd -r tom && useradd -g tom tom

# set ownership and permissions
RUN chown -R tom:tom /app

# switch to user
USER tom

CMD node index.js
```

ALCUNE IMMAGINI INCLUDONO Già UN UTENTE

```
FROM node:10-alpine

...

# set ownership and permissions
RUN chown -R node:node /app

# switch to node user
USER node

CMD [“node”, “index.js”]
```

Globant

BEST PRACTICES

7 – SCAN con SNYC


```
✗ Low severity vulnerability found in freetype/freetype
  Description: CVE-2020-15999
  Info: https://snyk.io/vuln/SNYK-ALPINE310-FREETYPE-1019641
  Introduced through: freetype/freetype@2.10.0-r0, gd/libgd@2.2.5-r2
  From: freetype/freetype@2.10.0-r0
  From: gd/libgd@2.2.5-r2 > freetype/freetype@2.10.0-r0
  Fixed in: 2.10.0-r1

✗ Medium severity vulnerability found in libxml2/libxml2
  Description: Out-of-bounds Read
  Info: https://snyk.io/vuln/SNYK-ALPINE310-LIBXML2-674791
  Introduced through: libxml2/libxml2@2.9.9-r3, libxslt/libxslt@1.1.33-r3, nginx-module-
xslt/nginx-module-xslt@1.17.9-r1
  From: libxml2/libxml2@2.9.9-r3
  From: libxslt/libxslt@1.1.33-r3 > libxml2/libxml2@2.9.9-r3
  From: nginx-module-xslt/nginx-module-xslt@1.17.9-r1 > libxml2/libxml2@2.9.9-r3
  Fixed in: 2.9.9-r4
```

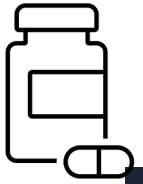
Si può configurare in molti repository, o integrarlo nella pipeline ci CI/CD

Globant

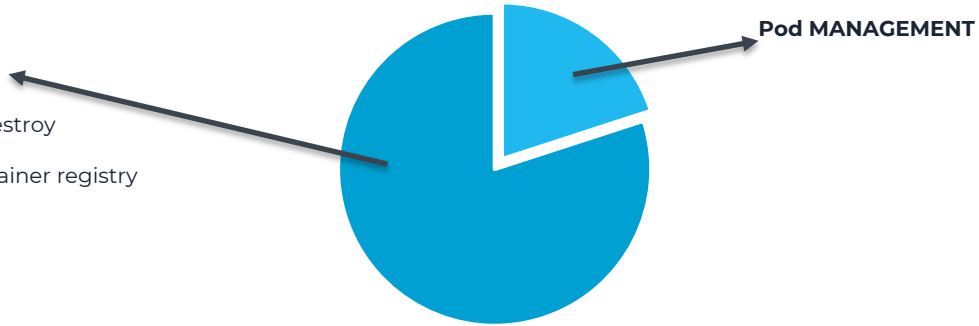
PODMAN

PODMAN

BASIC COMMANDS



```
alias docker=podman
```



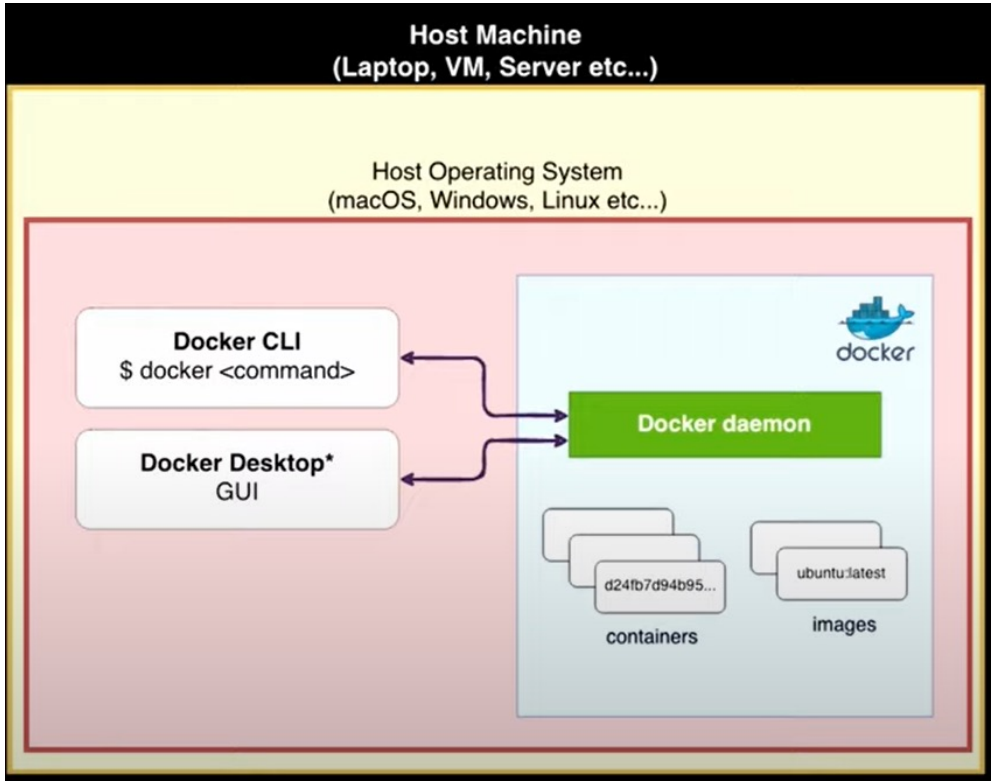
Docker-inspired
- Image build, update destroy
- Container run, stop
- Upload images to container registry

Globant

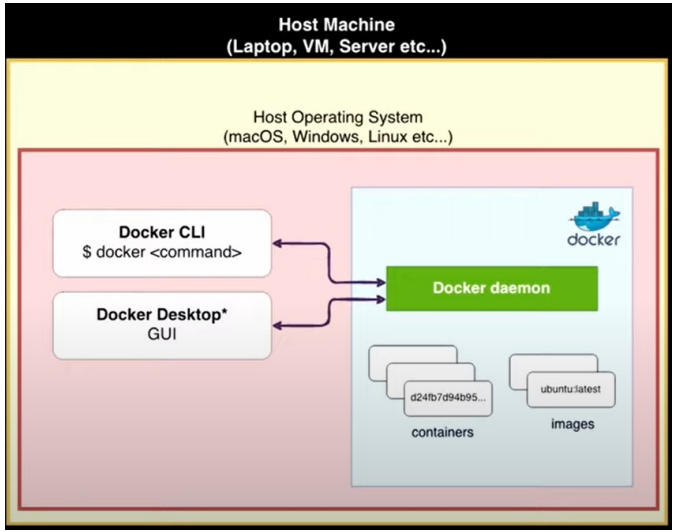
Ma allora... cosa c'è di diverso?

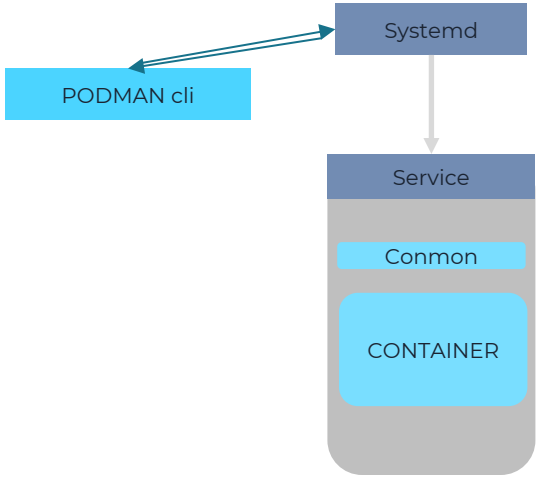
DAEMON
ROOT
LESS

Ma allora... cosa c'è di diverso?

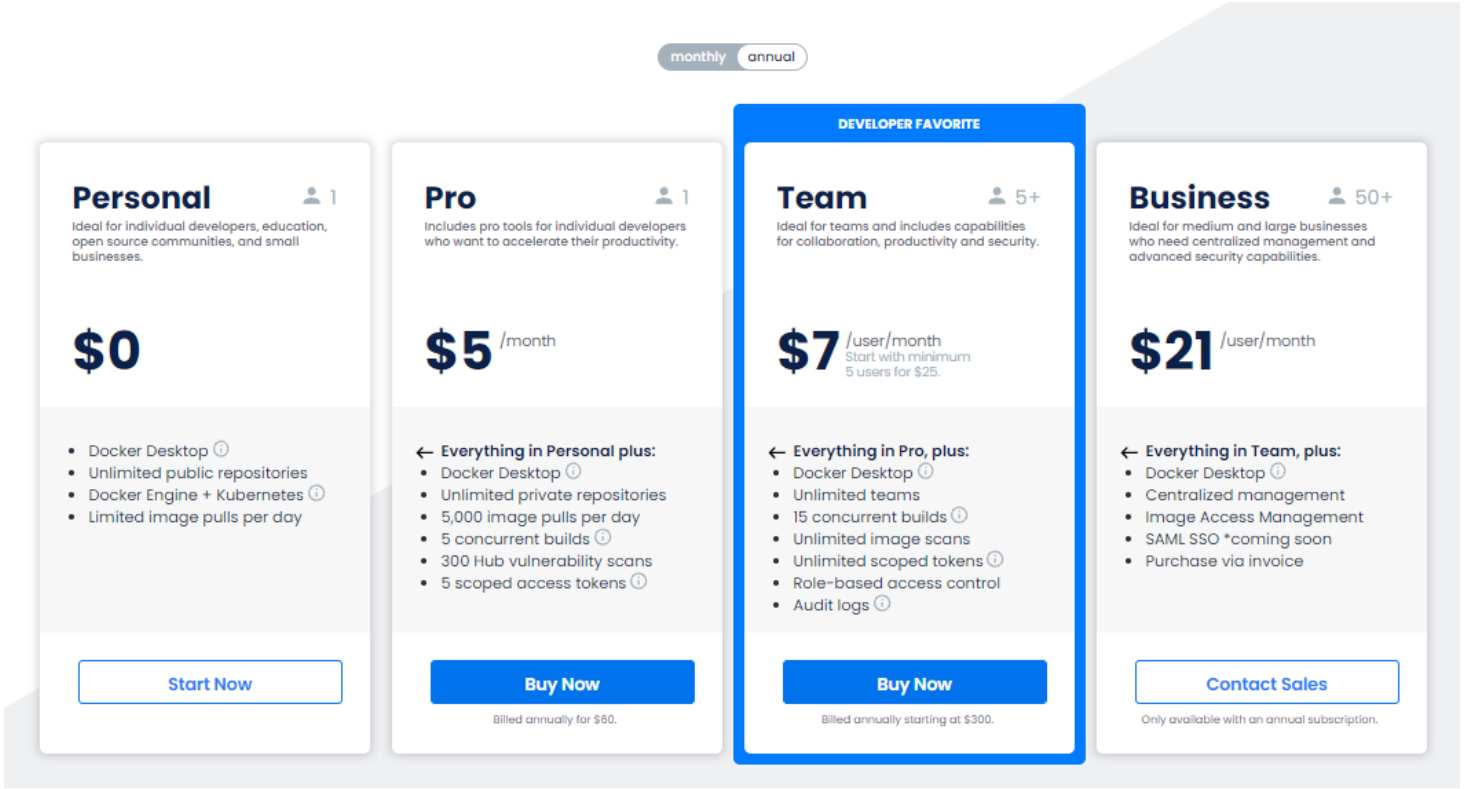


Ma allora... cosa c'è di diverso?



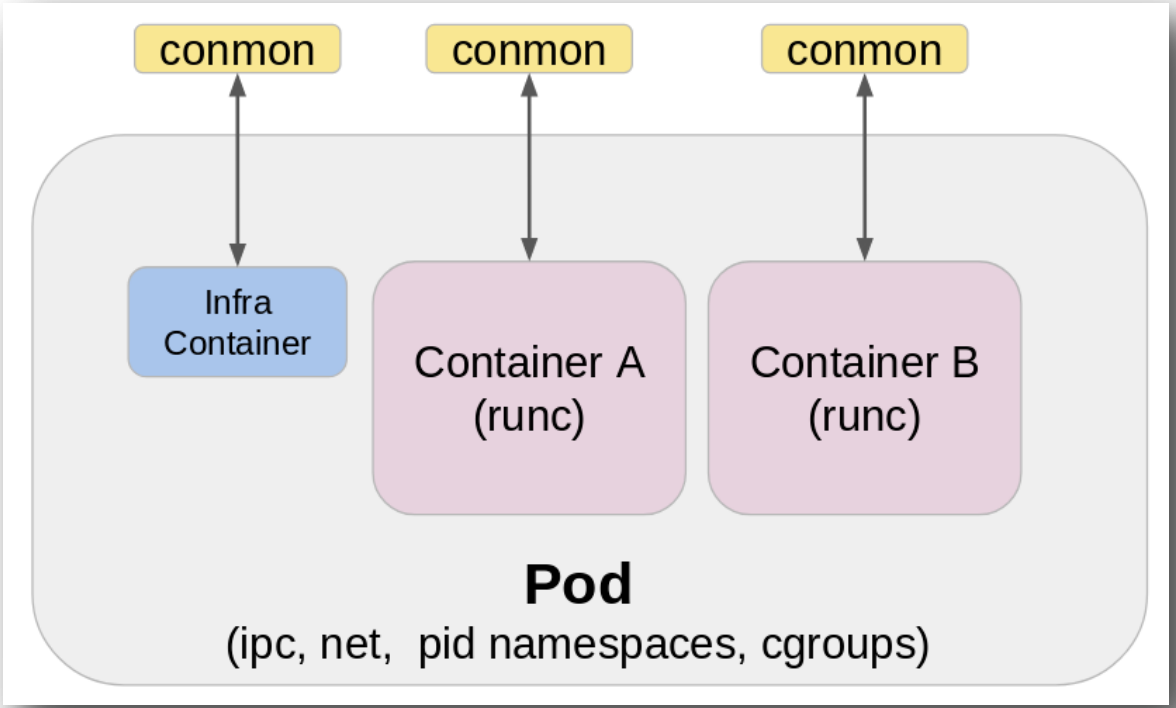


Docker si paga!!!

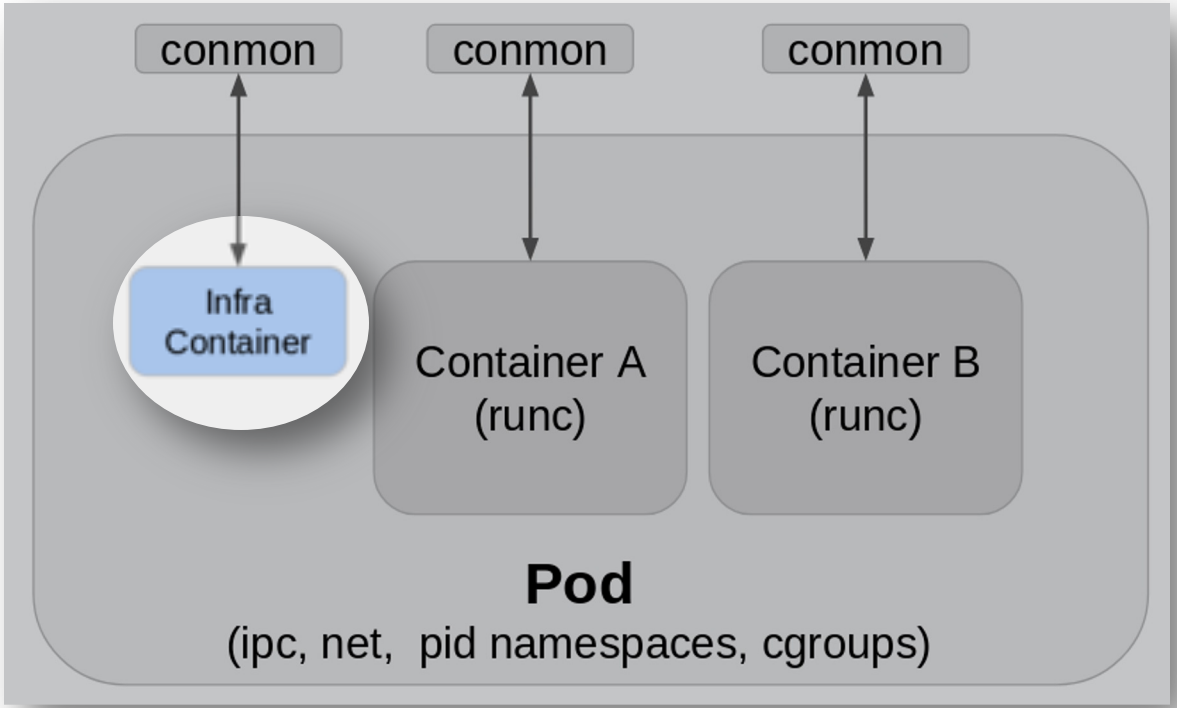


Globant

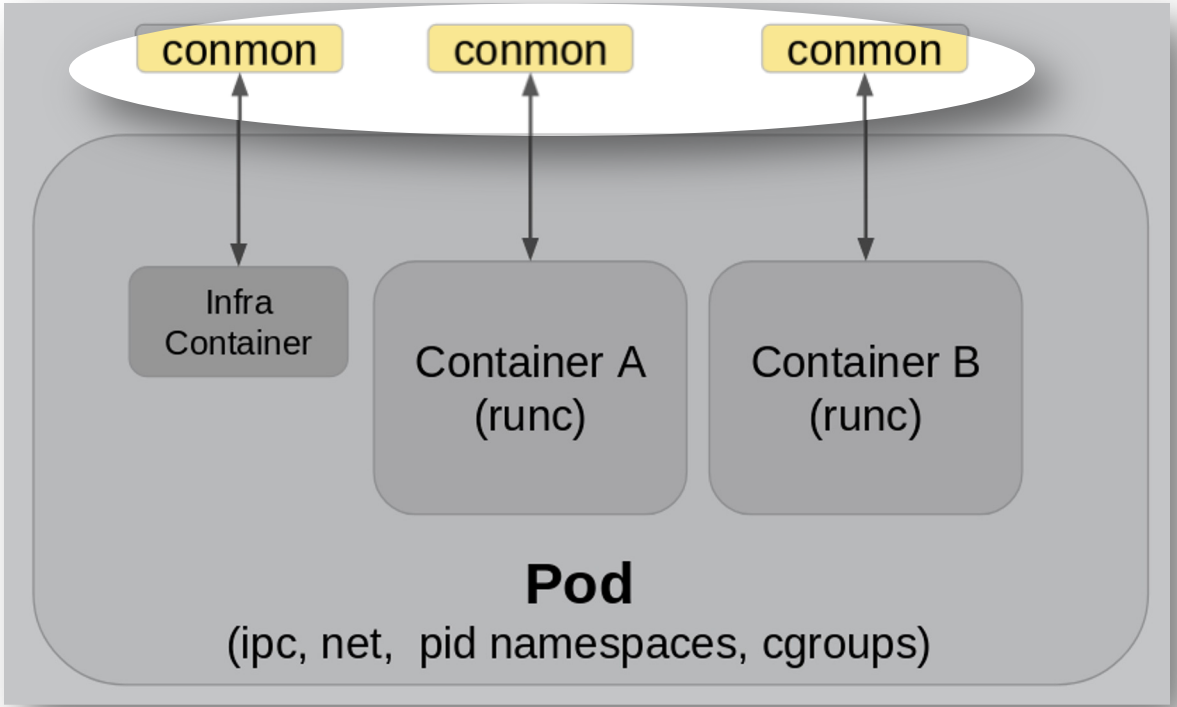
I POD


conmon
conmon
conmon
Infra Container
Container A (runc)
Container B (runc)
Pod
(ipc, net, pid namespaces, cgroups)

I POD


conmon
conmon
conmon
Infra Container
Container A (runc)
Container B (runc)
Pod
(ipc, net, pid namespaces, cgroups)

I POD


conmon
conmon
conmon
Infra Container
Container A (runc)
Container B (runc)
Pod
(ipc, net, pid namespaces, cgroups)

I POD

```
dtrezza@SYS-DTREZZA:/mnt/c/Users/Daniele$ podman pod create --name pod_di_daniele
d3b4d7e6272dd720ee511d1efecc0a78b2203fbed06203579ab5d5069dfc4f5b
```

```
dtrezza@SYS-DTREZZA:/mnt/c/Users/Daniele$ podman pod ps
POD ID        NAME            STATUS      CREATED         INFRA ID      # OF CONTAINERS
d3b4d7e6272d  pod_di_daniele  Created     26 seconds ago  1b9a8d346182  1
```

```
podman run --pod pod_di_daniele --name primo_container hello-world
```

```
podman pod restart pod_di_daniele
```

```
podman pod stop pod_di_daniele
```

```
podman pod kill pod_di_daniele
```

I POD

```
podman generate kube
```

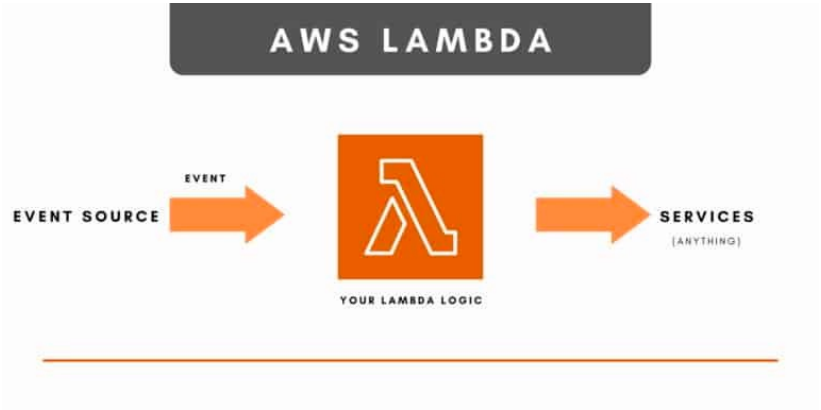
```
dtrezza@SYS-DTREZZA:/mnt/c/Users/Daniele$ podman generate kube pod_di_daniele
# Generation of Kubernetes YAML is still under development!
#
# Save the output of this file and use kubectl create -f to import
# it into Kubernetes.
#
# Created with podman-3.3.1
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2021-11-09T15:25:43Z"
  labels:
    app: poddidaniele
  name: pod_di_daniele
spec:
  containers:
  - command:
    - /hello
    env:
    - name: PATH
      value: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
    - name: TERM
      value: xterm
    - name: container
      value: podman
    image: docker.io/library/hello-world:latest
    name: primocontainer
    resources: {}
    securityContext:
      allowPrivilegeEscalation: true
      capabilities:
        drop:
        - CAP_MKNOD
        - CAP_NET_RAW
        - CAP_AUDIT_WRITE
      privileged: false
      readOnlyRootFilesystem: false
      seLinuxOptions: {}
    workingDir: /
  dnsConfig: {}
  restartPolicy: Never
status: {}
```

Globant

Un progetto vero

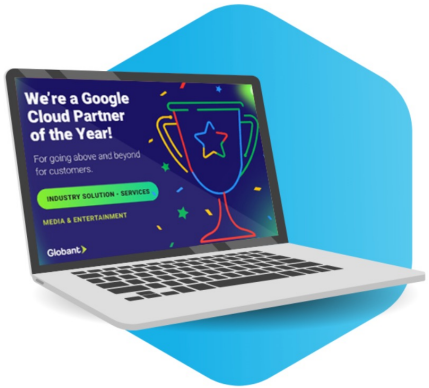
Docker images and AWS lambda

```dockerfile
FROM public.ecr.aws/lambda/python:3.9 as stage

RUN yum install -y -q sudo unzip

# Current stable version of Chromium
ENV CHROMIUM_VERSION=1002910

# Install Chromium
COPY install-browser.sh /tmp/
RUN /usr/bin/bash /tmp/install-browser.sh

COPY chrome-deps.txt /tmp/
RUN yum install -y $(cat /tmp/chrome-deps.txt)

COPY requirements.txt /tmp/

RUN python3 -m pip install --upgrade pip -q
RUN python3 -m pip install -r /tmp/requirements.txt -q

COPY --from=stage /opt/chrome /opt/chrome
COPY --from=stage /opt/chromedriver /opt/chromedriver
COPY . ${LAMBDA_TASK_ROOT}

CMD [ "run.handler" ]
```



Globant

Cloud Providers partnerships



aws partner network

Globant

Grazie per l’attenzione,

A presto!

Daniele Trezza daniele.trezza@globant.com

Nicholas Ferrari nicholas.ferrari@globant.com