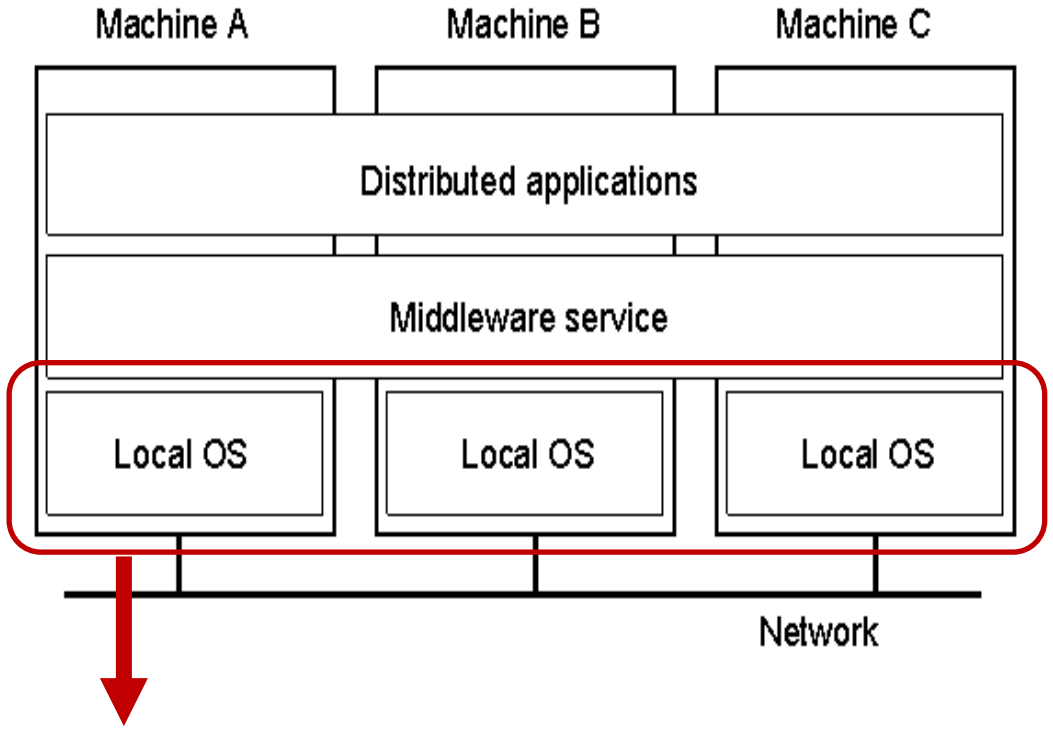
Instituto Superior de Engenharia de Lisboa

Departamento de Engenharia de Eletrónica e Telecomunicações e de Computadores

Mestrado em Engenharia Informática e de Computadores (MEIC)
Mestrado em Engenharia Informática e Multimédia (MEIM)

- **Infraestrutura de suporte à realização de laboratórios e trabalhos práticos**
  - ✓ **Virtualização e Contentores (*Containers*)**
  - ✓ **Exemplos com Docker**

Conceito de Middleware

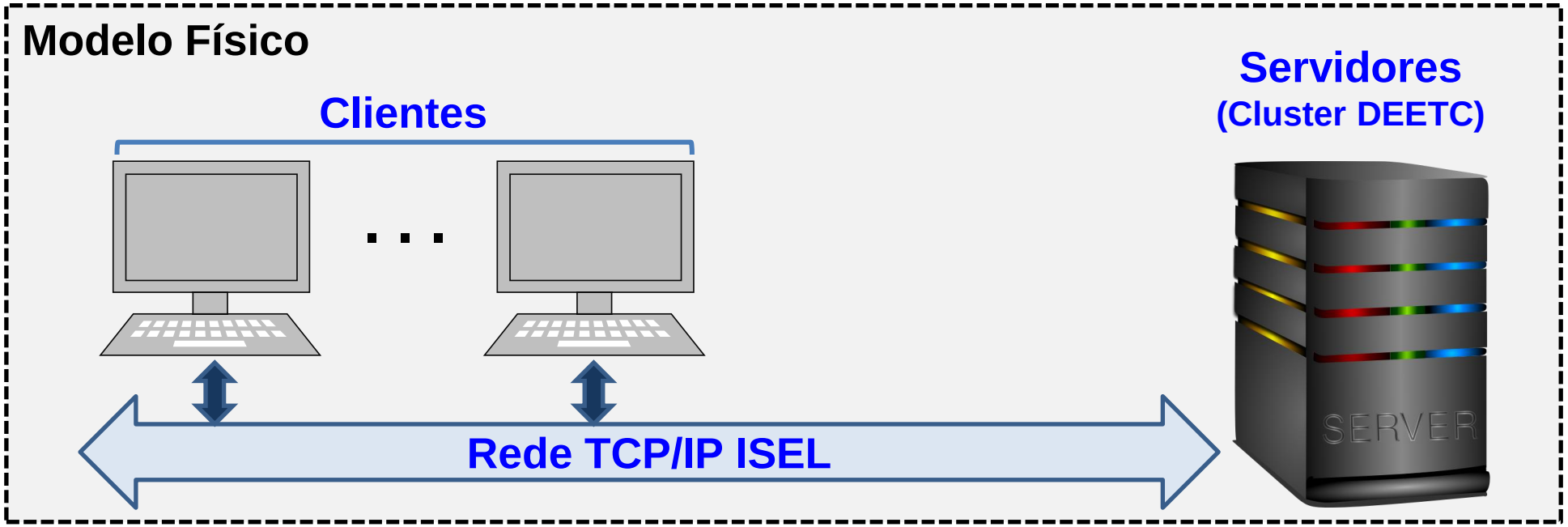


Conceito de *Middleware* numa infraestrutura local

© From: Distributed Systems, Principles and Paradigms - Andrew Tanenbaum

- **As componentes dos sistemas distribuídos, tanto ao nível de *middleware* como das aplicações podem ser alojadas e executadas:**
  - **Processos de sistema operativo numa máquina física;**
  - **Processos em máquinas virtuais (VM)**
  - ***Containers***

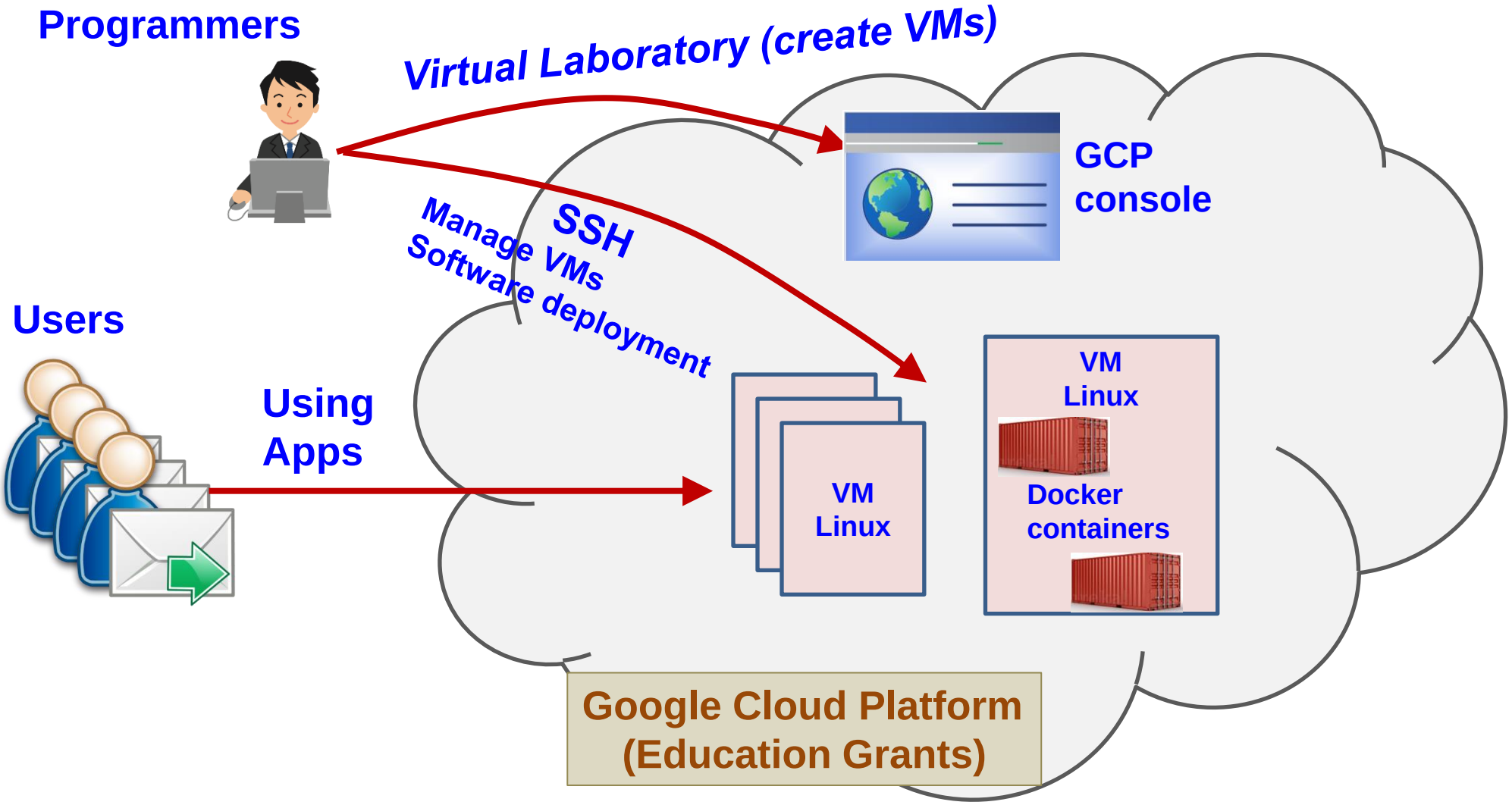
Infraestrutura de suporte à realização de laboratórios e trabalhos práticos



- ✓ **Limitação de garantir que cada grupo de alunos tivesse o número de servidores (VMs) suficiente;**
- ✓ **O endereçamento IP está limitado à rede interna do ISEL, só acessível do exterior através de VPN com bastantes limitações;**
- ✓ **Limitação da garantia da disponibilidade (*uptime*)**

Que solução ?

Usar a infraestrutura da *Google Cloud Platform* (GCP)



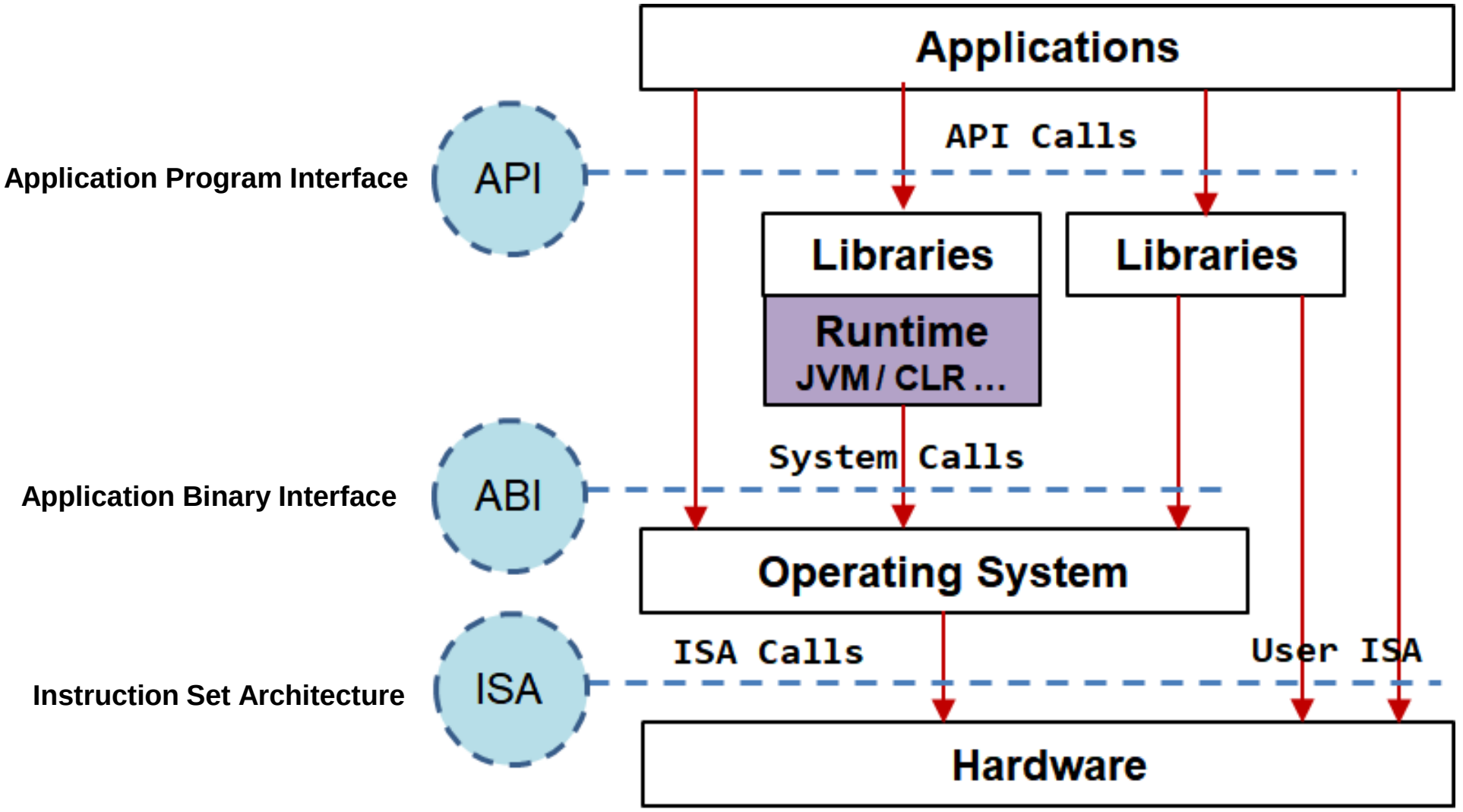
Criação de VMS na Google Cloud Platform

- **Garantir que cada grupo criou uma conta GCP:**
  - ✓ **Slides:** ***CD2324-Registo na Google Cloud Platform como Aluno.pdf***
- **Para criar VMs, seguir os passos indicados nos slides:**
  - ✓ **Slides:** ***CD2324-CriarVms-GCP.pdf***

Porquê a virtualização

- **Consolidação de recursos**
  - **Menos espaço para mais serviços (menos energia, recursos humanos, ...)**
- **Uso de sistemas legados (*legacy applications*)**
- **Isolamento**
  - **VM e/ou *Containers* isolam falhas de segurança ou erros em componentes de software**
- **Ambientes de desenvolvimento e de investigação**
  - **Facilidade de criação de ambientes com *stack* de software bem definido**
- **Rapidez de aprovisionamento e escalabilidade**
  - **Para melhor acomodar um aumento de carga (ex: número de pedidos)**
- **Migração e balanceamento de carga**
  - **Migração de VMs para consolidar e otimizar a utilização de hardware**
- **Backups e recuperação de desastres**

Interfaces de um sistema computational

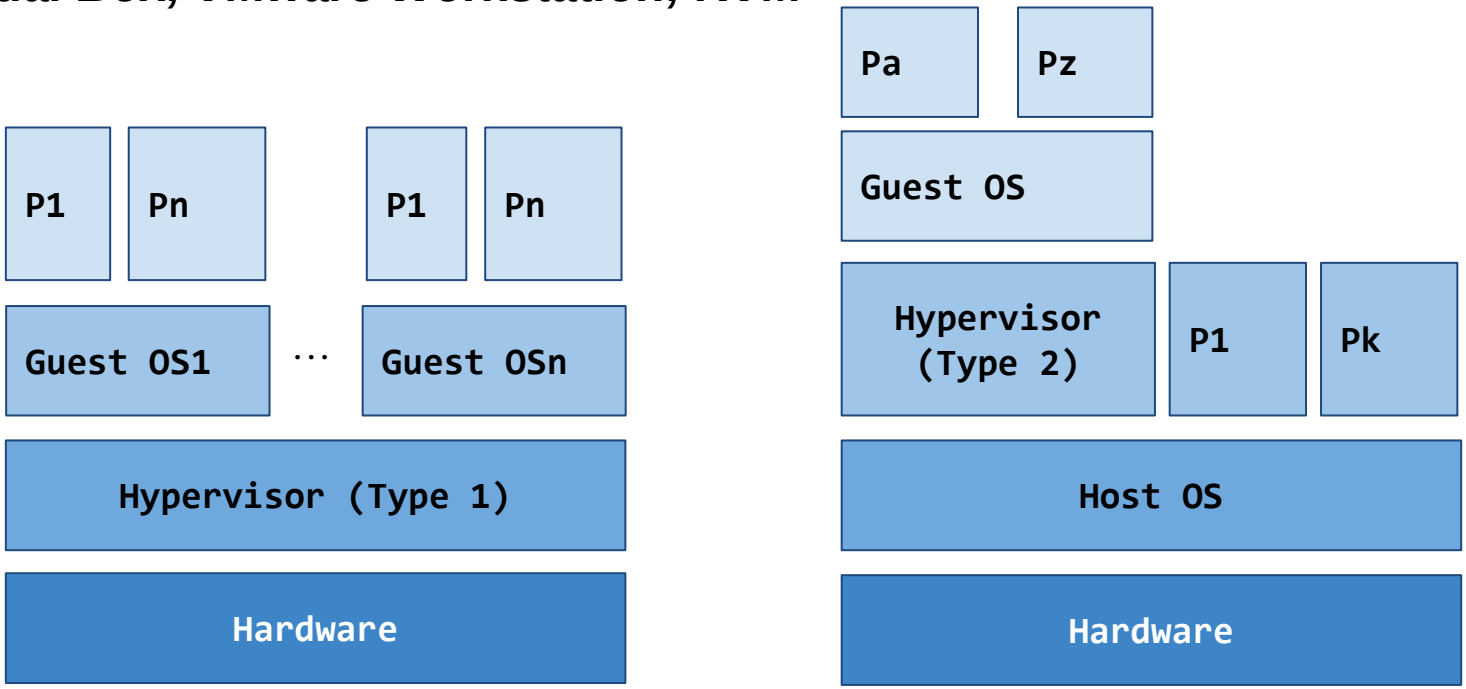


Máquinas virtuais de processo ou sistema

- O que é a "máquina virtual" depende da perspetiva do processo ou do sistema operativo
  - Na perspetiva do processo, a máquina é representada pela ABI, e na perspetiva da aplicação é representada pela API
  - Na perspetiva do SO, a máquina é representado pela ISA
- O software que suporta uma "máquina virtual" de processo é designado de *runtime (ex: JVM, CLR)*
- O software que suporta uma "máquina virtual" de sistema (VM) é referido como *virtual machine monitor* (VMM) ou *hypervisor*.
- O sistema operativo de uma VM é o *guest* (convidado)
- O software que suporta a VM é o *host* (hospedeiro)

Hypervisor e execução privilegiada

- O *hypervisor* de um ambiente virtualizado é classificado do Tipo 1 ou do Tipo 2, se respetivamente, não depende, ou depende da existência de um sistema operativo
- O Tipo 1 (ou *bare metal*) interage diretamente com o *hardware* e não necessita de um sistema operativo, introduzindo menos *overhead. Exemplos: Citrix/Xen, VMware ESXi; Microsoft Hyper-V*
- O Tipo 2 executa-se sobre um sistema operativo, tirando partido da transparência que esse sistema tem a diferentes *hardwares. Exemplos: Microsoft Virtual PC, Oracle Virtual Box, VMware Workstation; KVM*



Cloud Computing

- ***"A model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.", NIST 2011***

Note a analogia com o modelo do slide 2

Service Class	Access & Tools	Service contents
SaaS Software as a Service	Web Browser	**Cloud Applications:** *Social Networks, Email, Office suites (Google docs), ERP, CRM, IAM (Identity and Access Management), Video processing, …*
PaaS Platform as a Service	Development Environments	**Cloud Platform:** Programming languages, frameworks, *Mashups* editors, Web APIs, Data Storage models (Relational, NoSQL), *Data Analytics, …*
IaaS Infrastructure as a Service	Virtualization Manager	**Cloud Infrastructure:** Computer servers, Data Storage, Firewall, Load Balancer, IP Addressing, VPN, ….

Virtualização: Organizações, Clouds públicas, …

Centros de dados das organizações

. . .

Nas várias plataformas Cloud

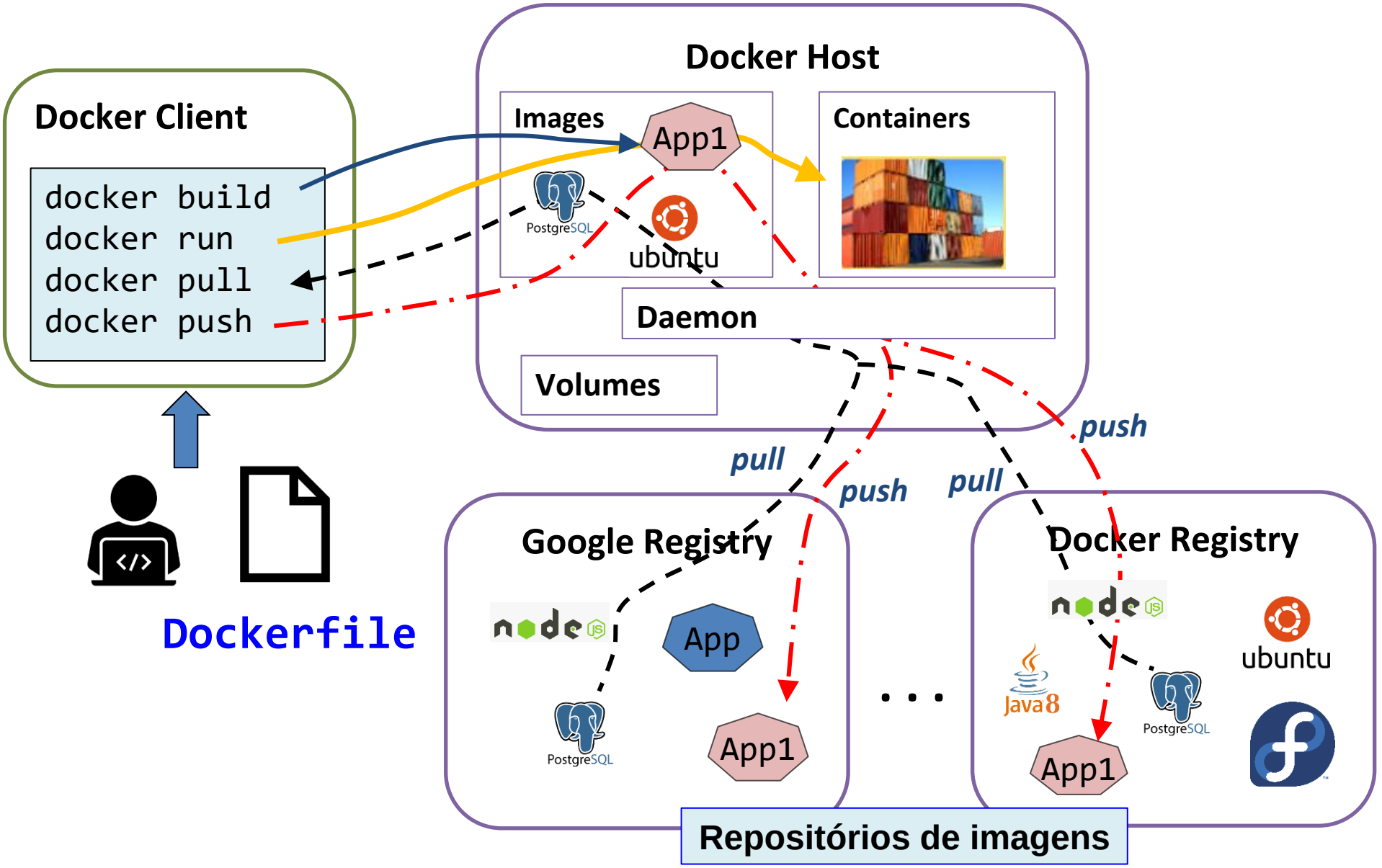
Deployment genérico de software

- Uma aplicação pode ter várias dependências que precisam de estar presentes no sistema alvo
  - *runtime*; bibliotecas; outras aplicações
- O *deployment* pode ser feito com base em imagens de VMs
  - No entanto, as imagens podem não ser portáveis entre diferentes *infraestruturas* com suporte de virtualização, nomeadamente na *Cloud*
- Uma abordagem diferente tem sido usada nos últimos anos com *containers* para executar imagens binárias, partilhadas de forma independente entre infraestruturas heterogéneas
  - Um *container* facilita o processo de desenvolvimento, teste, *deployment* e operação de sistemas:
    - Separação de responsabilidades entre componentes do sistema
    - Divisão de tarefas entre equipas
    - Isolamento e transparência face aos recursos computacionais
    - Automatização das operações de *deployment* e monitorização em produção
  - Mas! continuam a existir os desafios de segurança, coordenação e interação entre as partes bem como o tratamento das falhas parciais

Empacotamento em imagens de *containers*

- Os *containers* são, no essencial, processos que correm no contexto do sistema operativo, geridos por um *runtime*
- Fornecem um isolamento inferior ao das VMs mas superior ao de processos regulares do sistema operativo, virtualizando o acesso ao sistema de ficheiros e utilizando os recursos (CPU, Mem, I/O) com algumas restrições
- Os *containers* executam imagens binárias, previamente construídas, e comprometidas com uma *Application Binary Interface* (ABI) (ex: linux, *windows*)
  - Para determinados ambientes de execução, incluindo *runtimes*, *middlewares* e aplicações, existem normalmente imagens para diferentes plataformas de *hardware*: arm64, amd64, …
- Existem várias concretizações de *runtimes* de *containers*:
  - Linux *Containers*: LXC, LXD, RKT, CRI-O, . . .
  - Docker (que iremos usar como base para desenvolver aplicações distribuídas)

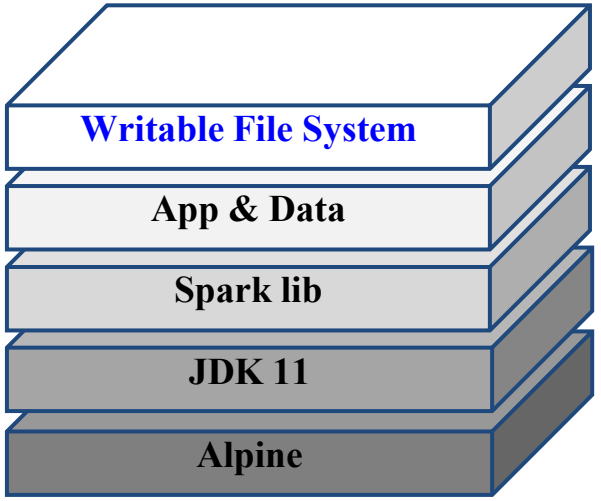
Sistema Docker



Imagens e *Containers*

- Uma imagem de *container* representa um ou mais ficheiros (obtidos de um Docker *Registry*, ex: `hub.docker.com`), e usados localmente para iniciar um *container*
- Existem formatos diferentes de imagens mas a generalidade das ferramentas suporta o formato aberto OCI
  (*Open Container Image*: https://www.opencontainers.org/)
  - Cada imagem depende de outra e adiciona algum *middleware* e/ou aplicação
  - Existem imagens base Linux e Windows e já com vários *middleware*
- Quando o *container* se inicia tem disponível um *file system* isolado do do sistema operativo *host*
  - As alterações no *file system* não são persistidas

Se a App escrever num ficheiro e o *container* terminar os dados são perdidos



Exemplo: Instalar Docker *engine* em CentOS

***Host* com sistema Linux (CentOS 8), numa VM GCP**

https://docs.docker.com/engine/install/centos/

- `sudo yum install -y yum-utils`
- `sudo yum-config-manager --add-repo \`
  `https://download.docker.com/linux/centos/docker-ce.repo`
- `sudo yum install docker-ce docker-ce-cli containerd.io`
- `sudo systemctl start docker`
- `sudo docker run hello-world`

Comando descrito em múltiplas linhas

Repetir sempre que se faz *restart* à VM

Para evitar usar *sudo* ao interagir com o *deamon* docker, é possível adicionar o utilizador Linux da VM GCP ao grupo privilegiado `docker` e `ativar` as mudanças no grupo

- `sudo usermod -aG docker $USER`

```
// newgrp is used to change the current GID (group ID) during a login session
```

- `sudo newgrp - docker`

Exemplo: *release* do SO dentro do *container*

- **Execução de *shell* "`/bin/bash`" em *container* com imagem de sistema operativo Fedora**

```
docker run -i -t fedora
```

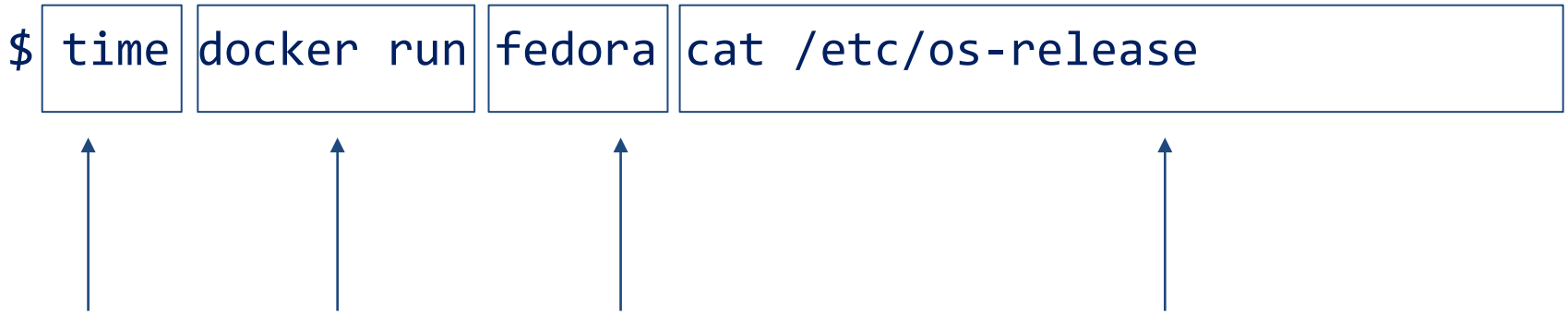
```
-i iterativo
-t pseudo-terminal (/bin/bash)
```

Primeira vez que a imagem é usada localmente é feito download do repositório de imagens

```
Unable to find image 'fedora:latest' locally
latest: Pulling from library/fedora
4c69497db035: Already exists
Digest: sha256:ee55117b3058f2f12961184fae4b9c392586e400487626c6bd0d15b4eae94ecc
Status: Downloaded newer image for fedora:latest
[root@57cad972cfe4 /]# cat /etc/os-release
NAME=Fedora
VERSION="31 (Container Image)"
ID=fedora
VERSION_ID=31
VERSION_CODENAME=""
PLATFORM_ID="platform:f31"
PRETTY_NAME="Fedora 31 (Container Image)"
```

Shell dentro do container

Iniciar *container* e execução de comando




```
real    0m0.599s
user    0m0.029s
sys     0m0.022s
```

- **A execução de um *container* `fedora` e, dentro deste, o comando `cat /etc/os-release`, demorou no total ~0.6 seg.**
- **As VMs podem demorar algumas dezenas de segundos a iniciar**

Os 4 passos do ciclo de desenvolvimento e produção

1. **Desenvolver a aplicação usando a linguagem de programação, bibliotecas e ambiente de execução que sejam apropriados**

2. **Criar a imagem binária com a aplicação desenvolvida, referindo uma imagem base que tenha apenas o sistema operativo ou já com alguns dos componentes (bibliotecas, *middleware*, etc.)**

3. **Publicar (*push*) a imagem num repositório (*Docker Registry*) de imagens**

4. **Em qualquer sistema, com suporte para *containers*, é possível lançar em execução um *container* a partir da imagem binária publicada no ponto 3**

Exemplo de construção de imagem a partir de JAR

Artefacto:

ServiceREST.jar

```
Serviço http REST no porto 7500 com as rotas:
   http://<host ip>:7500/ping
   http://<host ip>:7500/hello/<some name>
   http://<host ip>:7500/calc/number{+,-,*}number
   Ex: http://<host ip>:7500/calc/5*3 retorna 15
```

Dockerfile

```
FROM openjdk:11

RUN mkdir /usr/servicerest

WORKDIR /usr/servicerest

COPY  ServiceREST.jar .

CMD ["java", "-jar", "ServiceREST.jar"]
```

Imagem base

Diretoria de trabalho dentro do *container*

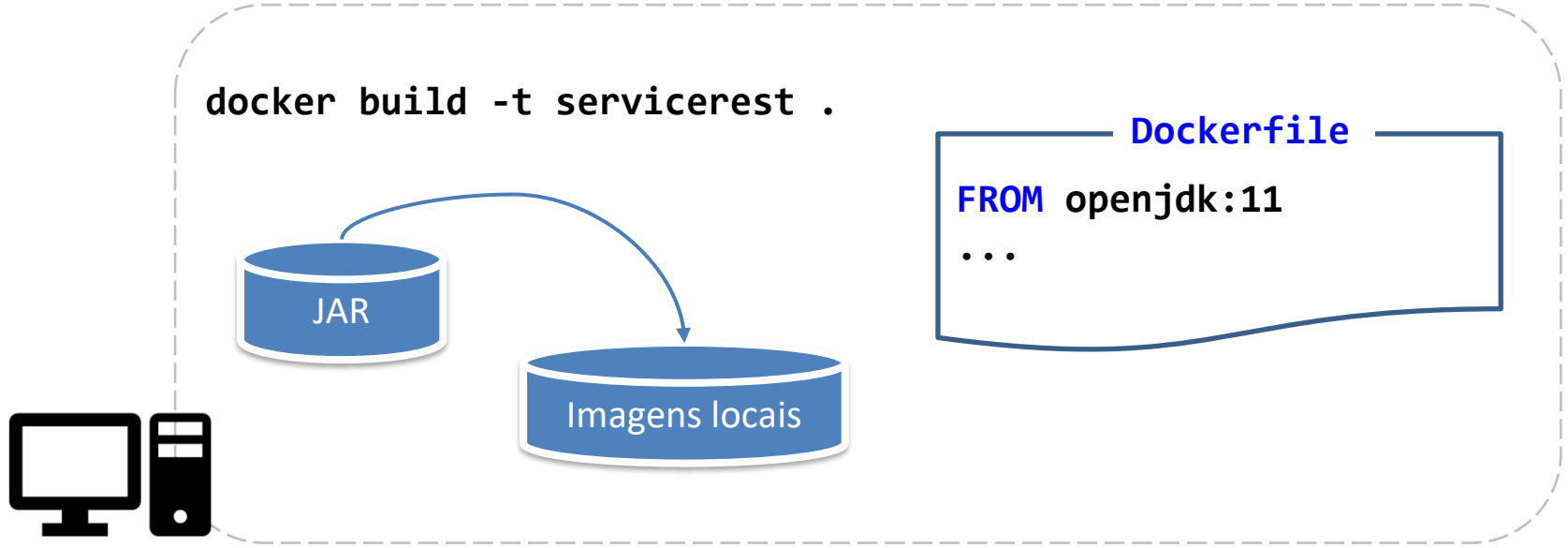
Copia JAR do *host* para a WORKDIR da imagem

Comando a executar quando se iniciar o *container*

https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

“Dockerizar” uma aplicação

- O ficheiro *`Dockerfile`* refere as imagens base e as aplicações que têm de ser copiadas do host para a imagem
- O comando *`docker build`* constrói novas imagens a partir de um ficheiro *`Dockerfile`*

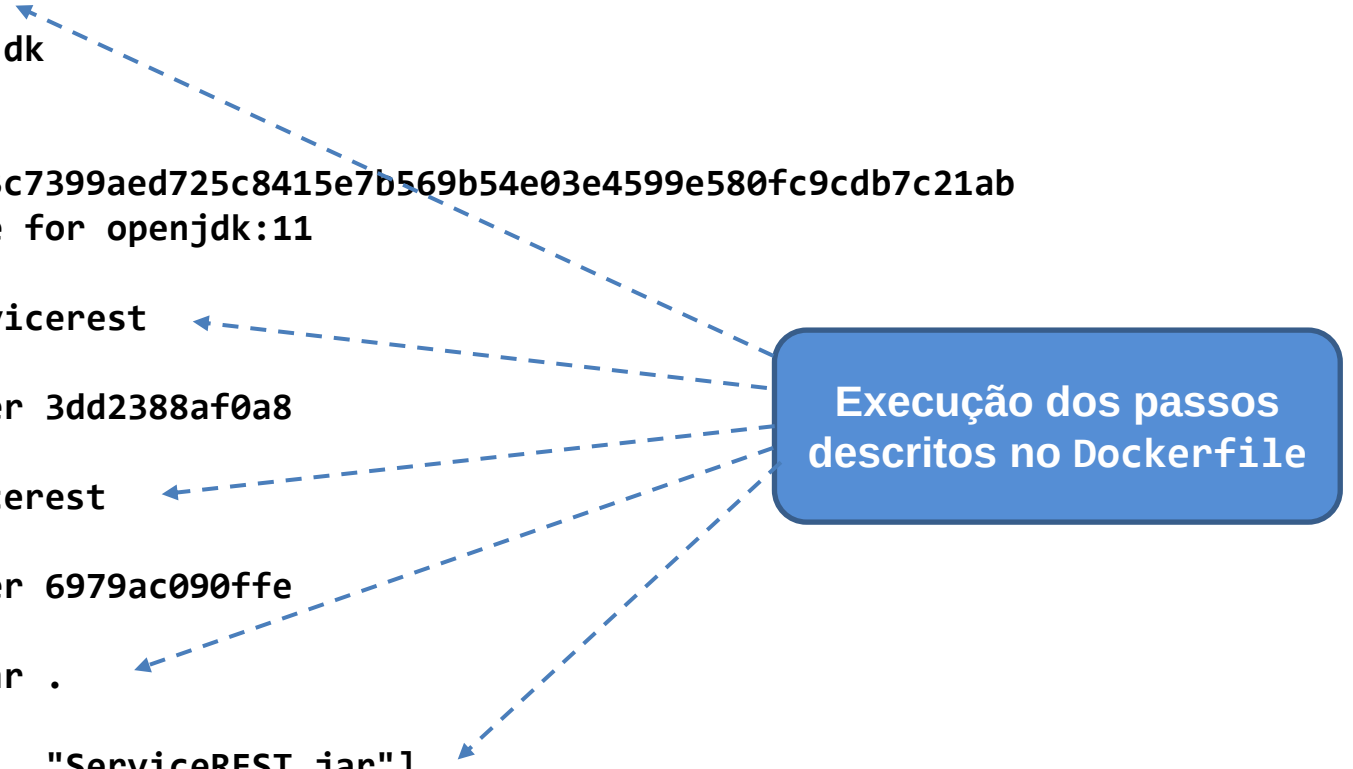


- Imagem fica guardada localmente com um nome e uma *tag* (no exemplo,
- `servicerest:latest`)

Exemplo de execução do comando build

```
$ docker build -t servicerest  .
```

```
Sending build context to Docker daemon  3.146MB
Step 1/5 : FROM openjdk:11
11: Pulling from library/openjdk
001c52e26ad5: Pull complete
d9d4b9b6e964: Pull complete
Digest: sha256:99bac5bf83633e3c7399aed725c8415e7b569b54e03e4599e580fc9cdb7c21ab
Status: Downloaded newer image for openjdk:11
 ---> 47a932d998b7
Step 2/5 : RUN mkdir /usr/servicerest
 ---> Running in 3dd2388af0a8
Removing intermediate container 3dd2388af0a8
 ---> f3211de0336b
Step 3/5 : WORKDIR /usr/servicerest
 ---> Running in 6979ac090ffe
Removing intermediate container 6979ac090ffe
 ---> 6ab6c1689100
Step 4/5 : COPY ServiceREST.jar .
 ---> 45d1b55bf27f
Step 5/5 : CMD ["java", "-jar", "ServiceREST.jar"]
 ---> Running in 35ddb2da17c7
Removing intermediate container 35ddb2da17c7
 ---> 6ecdb6caca3b
Successfully built 6ecdb6caca3b
Successfully tagged servicerest:latest
```



Ciclo de vida de imagem num *container*

- **Imagens**

```
$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
servicerest   latest    6ecdb6caca3b   29 minutes ago   657MB
openjdk       11        47a932d998b7   6 weeks ago      654MB
fedora        latest    98ffdbffd207   4 months ago     163MB
hello-world   latest    feb5d9fea6a5   11 months ago    13.3kB
```

- **Execução com exposição do porto 7500 do *container* como porto 8000 do *host***

```
$ docker run -d –p 8000:7500 servicerest
```

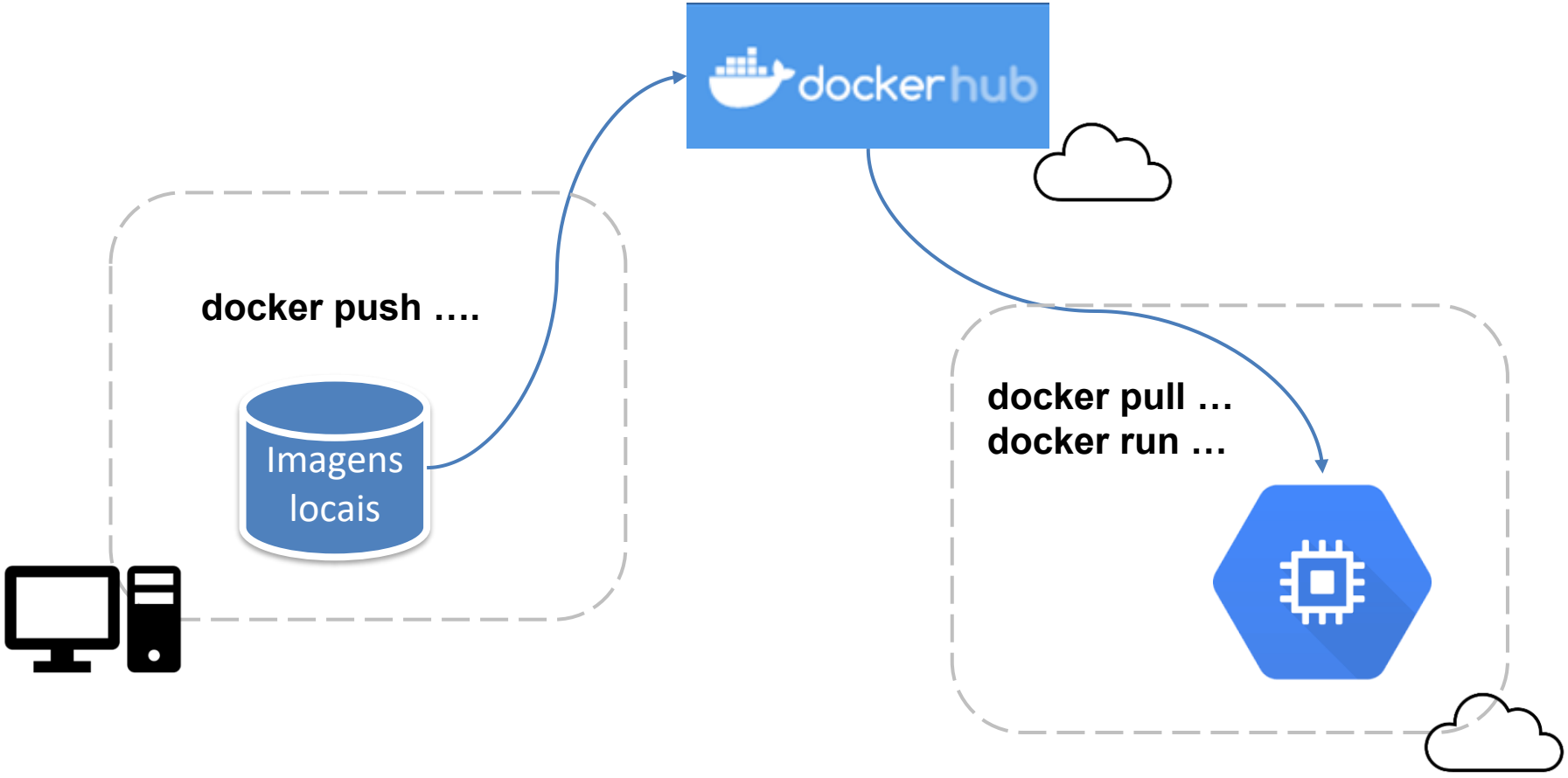
- **Observação de execução**

```
$ docker ps -all
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS
PORTS                                          NAMES
494ad3d036d0   servicerest    "java -jar ServiceRE…"   14 minutes ago   Up 14
minutes   0.0.0.0:8000->7500/tcp, :::8000->7500/tcp    elegant_tharp
```

- **Destruição do container em execução**

```
$ docker kill 494
494
```

Push, Pull e execução *anywhere*



Se ainda não tem, deve criar uma conta no *dockerhub*

Publicar imagem em repositório Dockerhub (1)

- **Para publicar no Dockerhub é preciso estar autenticado**

```
$ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have
a Docker ID, head over to https://hub.docker.com to create one.
Username: user
Password: password
Login Succeeded
```

- **A imagem local tem de ser marcada (*tag*) com o formato adequado ao repositório remoto (<user>/<image name>[:<tag>])**

```
$ docker tag servicerest jslaisel/servicerest
```

alternativa

```
$ docker tag servicerest jslaisel/servicerest:v1
```

```
REPOSITORY              TAG       IMAGE ID       CREATED          SIZE
jslaisel/servicerest    latest    6ecdb6caca3b   41 minutes ago   657MB
servicerest             latest    6ecdb6caca3b   41 minutes ago   657MB
openjdk                 11        47a932d998b7   6 weeks ago      654MB
fedora                  latest    98ffdbffd207   4 months ago     163MB
hello-world             latest    feb5d9fea6a5   11 months ago    13.3kB
```

- **O comando *`build`* pode usar este formato para publicar em repositório externo**

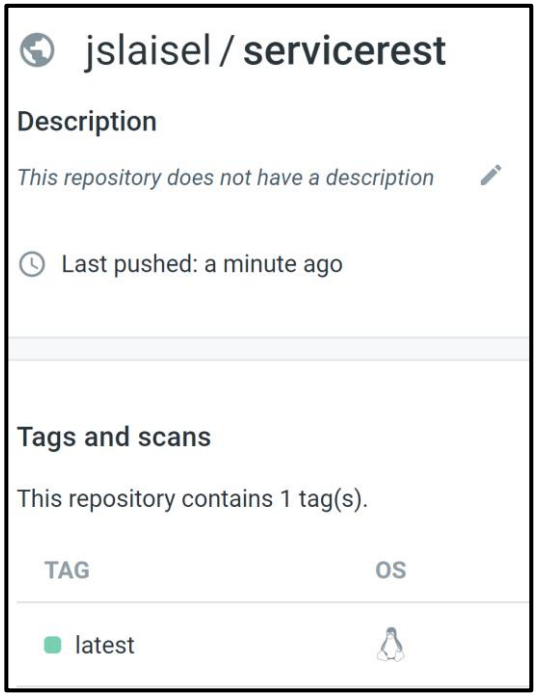
```
$ docker build -t jslaisel/servicerest:v1 .
```

Publicar imagem em repositório Dockerhub (2)

- **A imagem pode agora ser publicada no repositório *Dockerhub***

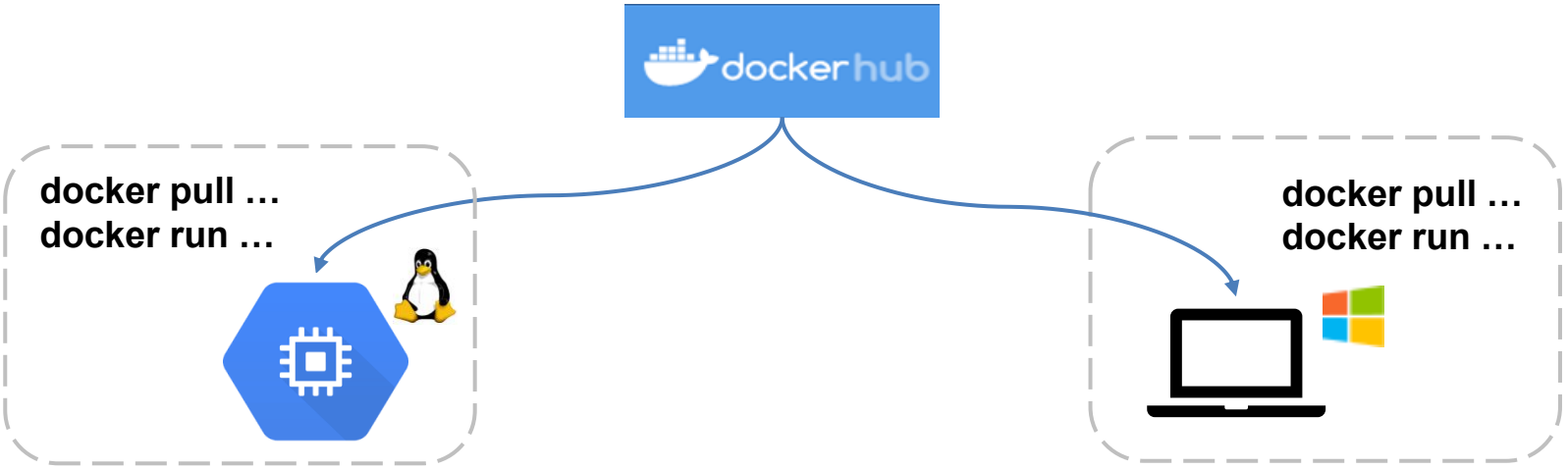
```
$ docker push jslaisel/servicerest
Using default tag: latest
The push refers to repository [docker.io/jslaisel/servicerest]
...
```

https://hub.docker.com/repository/docker/jslaisel/servicerest



- **Mais comandos**
  - https://www.docker.com/sites/default/files/d8/2019-09/docker-cheat-sheet.pdf

Exemplo de *pull* e *run*

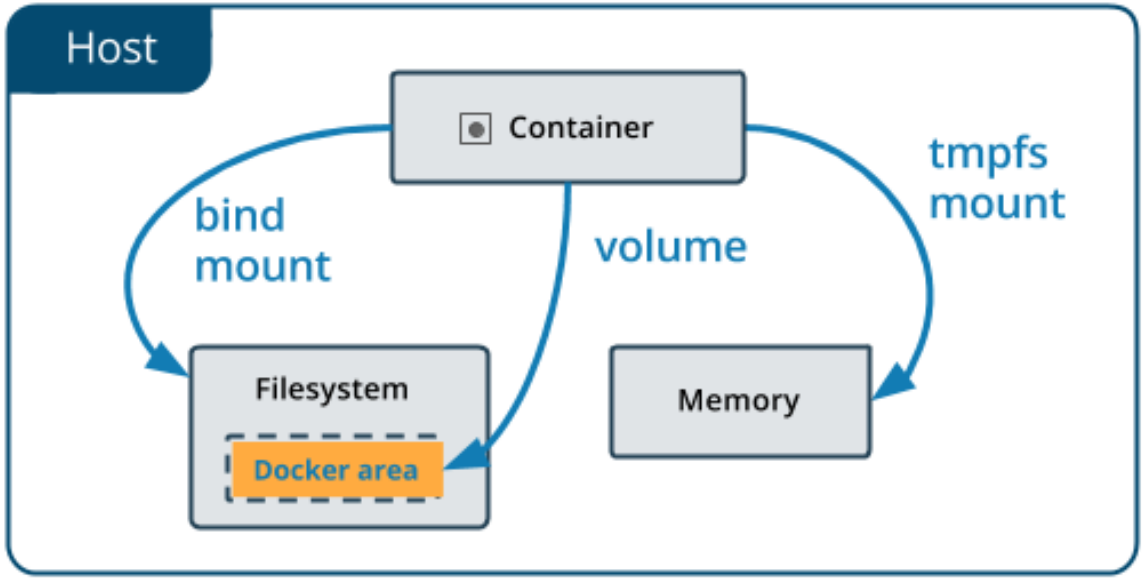



```
$ docker pull jslaisel/servicerest
Using default tag: latest
latest: Pulling from jslaisel/servicerest
Digest: sha256:e0776de7dd50ad1f68b5c5b60bb6a0999c6c0a85667bb8c4d424f1b677fab892
Status: Image is up to date for jslaisel/servicerest:latest
docker.io/jslaisel/servicerest:latest

$ docker run jslaisel/servicerest
```

Persistência de dados

- **A camada de escrita do *container* não persiste as alterações feitas no sistema de ficheiros após o *container* ser destruído**
- **Para persistir dados ao nível do sistema de ficheiros do *host*, o *container* tem de partilhar uma pasta com o sistema de ficheiros do *host***



https://docs.docker.com/storage/volumes/

Exemplo com *volumes*

containers a partilhar um volume de nome *volshare*

```
$ docker volume create volshare
volshare
$ docker run --name fedora-1 -it -v volshare:/myshare fedora
[root@80c38b8c7d31 /]# ls /myshare
[root@80c38b8c7d31 /]# date > /myshare/fedora-1.txt
[root@80c38b8c7d31 /]# ls /myshare
fedora-1.txt  fedora-2.txt
[root@80c38b8c7d31 /]# cat /myshare/fedora-2.txt
Fri May 29 08:29:22 UTC 2020
[root@80c38b8c7d31 /]#
```

```
$ docker run --name fedora-2 -it -v volshare:/extshare fedora
[root@fafa04f367f1 /]# ls /extshare
fedora-1.txt
[root@fafa04f367f1 /]# date > /extshare/fedora-2.txt
[root@fafa04f367f1 /]# ls /extshare
fedora-1.txt  fedora-2.txt
[root@fafa04f367f1 /]# cat /extshare/fedora-1.txt
Fri May 29 08:26:51 UTC 2020
[root@fafa04f367f1 /]#
```

```
$ docker volume ls
DRIVER      VOLUME NAME
local       volshare
```