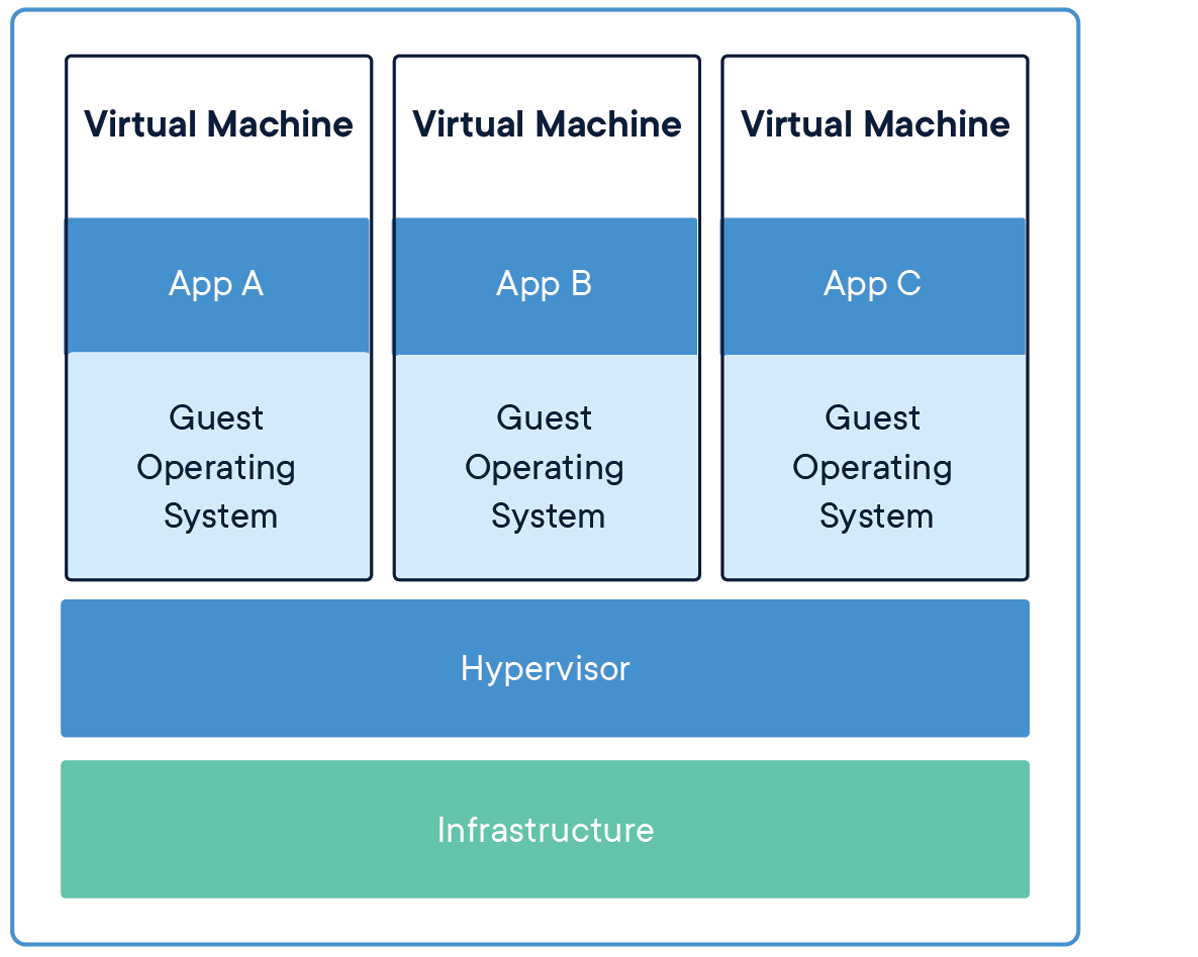
**Docker Essencial para o Desenvolvedor**

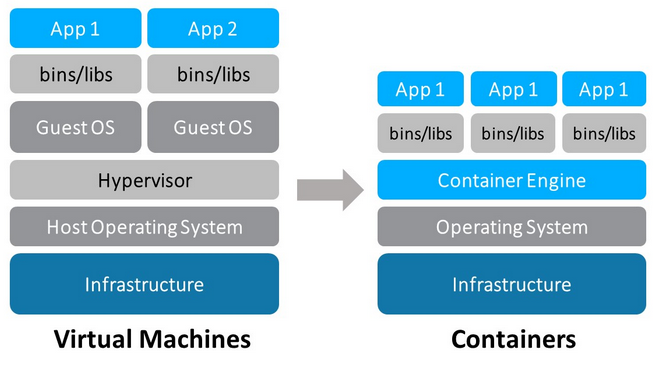
**Containers** são uma maneira de criar **ambientes isolados** que podem executar código enquanto compartilham um **único** sistema operacional.



Gerenciar os containers é difícil, pois tem que gerenciar o sistema operacional quase a nível de kernel.

O **Docker** é uma ferramenta que facilita o gerenciamento de containers. Na imagem anterior, o Docker fica na camada do *Hypervisor*, que é o que permite virtualização.

Embora seja semelhante e motivo de confusão, Containers diferem de Máquinas Virtuais. A diferença está na **arquitetura**.



Para instalar o Docker e utilizá-lo, primeiro foi instalado o Visual Studio Code e, dentro dele, as extensões YAML, Material Icon Theme e Code Runner. Após feito isso, foi instalado o próprio setup do Docker. Durante a instalação, ele pede para reiniciar após apontar um erro. Ao reiniciar, deve-se instalar o WSLI.

Após feito todos os passos de instalação, foi verificado se, de fato, o Docker estava rodando. Para isso, o PowerShell foi aberto e o seguinte comando foi digitado:

*docker run hello-world*

O *hello-world* é uma **imagem** **pra criação de contâiner**. Internamente, ele busca na nossa máquina a imagem do *hello-world*, porém não encontra. Então, ele busca na internet, encontra e instala a imagem. Veja abaixo o que foi mostrado.

Texto

Descrição gerada automaticamente

**Introdução à YAML**

**Y**et **A**nother **M**arkup **L**anguage é uma linguagem de **marcação** e **serialização de dados**.

**Serialização de dados** é a técnica que permite converter objetos em bytes, colocando-os em série, e uma vez que eles são bytes, eles podem ser **salvos em disco** ou enviados através de um **stream** (via HTTP, via Socket, entre outros).

A linguagem se integra a outras (**Python**, Java, Ruby), possui tipos de dados comuns como escalares, listas, arrays, etc, além de ser comumente utilizada como arquivo de **configuração** ou **armazenamento de dados**. Os objetivos desta linguagem são: ser lida facilmente por humanos, portátil, integrar-se facilmente com outras linguagens e facilidade de implementação e uso.

A sintaxe do YAML foi trabalhada na pasta “curso-YAML”, onde possui vários *scripts* *.yaml*, que foram rodados no VS Code. O PDF “13-recapitulando.pdf” traz, resumidamente, a sintaxe estudada.

**Seção 5: Docker e Imagens**

***O que são imagens em um container?***

**R.:** quando falamos de containers, imagens são ‘templates’ (modelos) na qual usamos para criar containers. Quando rodamos o *Docker run hello-world*, o Docker não encontrou a imagem, então ele fez o download da plataforma no Docker Hub (<https://hub.docker.com>). Nesse caso, o *Docker run hello-world* baixou da internet o *template* *hello-world* para criarmos um container a partir dela.

Existem algumas formas de utilizar uma imagem, duas delas são: *Docker run*, *Docker pull*.

*Docker pull hello-world*

O comando acima indica que você irá **baixar** a imagem hello-world e **criar** um container **a partir dela**.

Ao rodar

*Docker run hello-world*

Nós baixamos (caso não exista localmente) e já criamos o container, ele **já vem pronto**.

Em suma, utilizar *pull* implica em **criar** um container a partir da imagem; *run*, **rodar** um container **pronto**.

**Exemplo:**

“Uma aplicação será desenvolvida com Python e banco de dados PostgreSQL e será executada no servidor web Nginx.”

Uma possível infra para esse exemplo seria utilizar **3 containers**, onde cada um recebe um “serviço”.

- Um container usará uma **imagem** com a linguagem Python contendo suporte a se comunicar com o banco de dados PostgreSQL;

- Outro container usará uma **imagem** com o servidor do banco de dados PostgreSQL;

- Outro container usará outra **imagem** com o servidor web Nginx.

***Obs.: as imagens de containers são diferentes de imagens de serviço cloud como AWS, Google Cloud Plataform e Microsoft Azure.***

***Obs. 2: um container após criado é inicializado, ou seja, não fica aguardando um comando para que fique funcional.***

***Obs. 3: uma imagem é criada a partir de um Dockerfile.***

**Cada container é criado com uma imagem?**

**R.:** é possível que mais de um container utilize a mesma imagem sem baixa-la mais de uma vez. Quando criamos um container a partir de uma imagem, esta fica armazenada no nosso computador local e sempre que quisermos utilizá-la para um novo container, não precisamos instalar novamente.

***Sistemas de arquivos em camadas***

Os containers são uma tecnologia **Linux**, portanto, segue o sistema de arquivos em camadas do Linux. Primeiro, vamos entender o ***filesystem***, que é o sistema de arquivos utilizado pela ferramenta, chamado de **Layered**. Isso significa que é um **sistema de arquivos em camadas**.

Veja na imagem a seguir o sistema de arquivos de um sistema Linux/Unix.

Diagrama

Descrição gerada automaticamente

Ele é composto por **duas camadas**.

**Bootfs (boot file system):** é onde fica o Boot do sistema operacional, assim como o **Kernel.**

**Rootfs:** inclui o sistema de arquivos do sistema operacional, incluindo a arquitetura de diretórios como /dev, /proc, /bin, /etc, /lib, /usr e /tmp, assim como os arquivos de configuração e binários do sistema operacional.

Quando o sistema operacional é iniciado, ele carrega o ***rootfs*** em modo leitura, verifica a integridade e, em seguida, remonta-o como leitura/escrita tornando-se disponível para o usuário e aplicações.

No Docker, tem essa mesma arquitetura, mas com um pequeno diferencial: a camada de escrita que o processo/aplicação visualiza não é o mesmo rootfs base do sistema, mas sim uma **camada de abstração** do rootfs.

Isso faz com que um container se torne **portável**, pois as modificações realizadas **não** são aplicadas ao sistema origem do container e sim na camada a qual o sistema visualiza.

Em termos práticos, o bootfs se torna compartilhado com o rootfs e o container via AUFS, enquanto o rootfs é isolado por camadas. O AUFS monta uma camada de leitura/escrita em cima do filesystem em **somente leitura**, pois isso que garantirá que as modificações feitas dentro do container **não afetem o sistema de arquivos** do host. Um detalhe nessa arquitetura é que a cada modificação e *commit* do container é gerado uma nova camada.

**Imagens são compartilhadas entre containers Docker,** pois este tipo de sistema de arquivos em camadas busca sempre a eficiência.

**Exercício Prático: comandos importantes**

CMD

**# Versao do Docker**

docker –version

**# Lista de todos os comandos do Docker**

docker –help

**# Listagem (ls) de todos os containers Docker**

docker container ls

# **Lista dos comandos disponíveis para aquele container**

docker container –help

# **Lista todos os containers EM EXECUCAO**

docker ps

# **Lista os containers EM EXECUÇAO** e os **PARADOS**

docker ps -a

# **Help do comando especificado**

docker container start –help

docker container [comando\_qualquer] –help

# **Roda o container hello-world**

docker run hello-world

**11:09**