

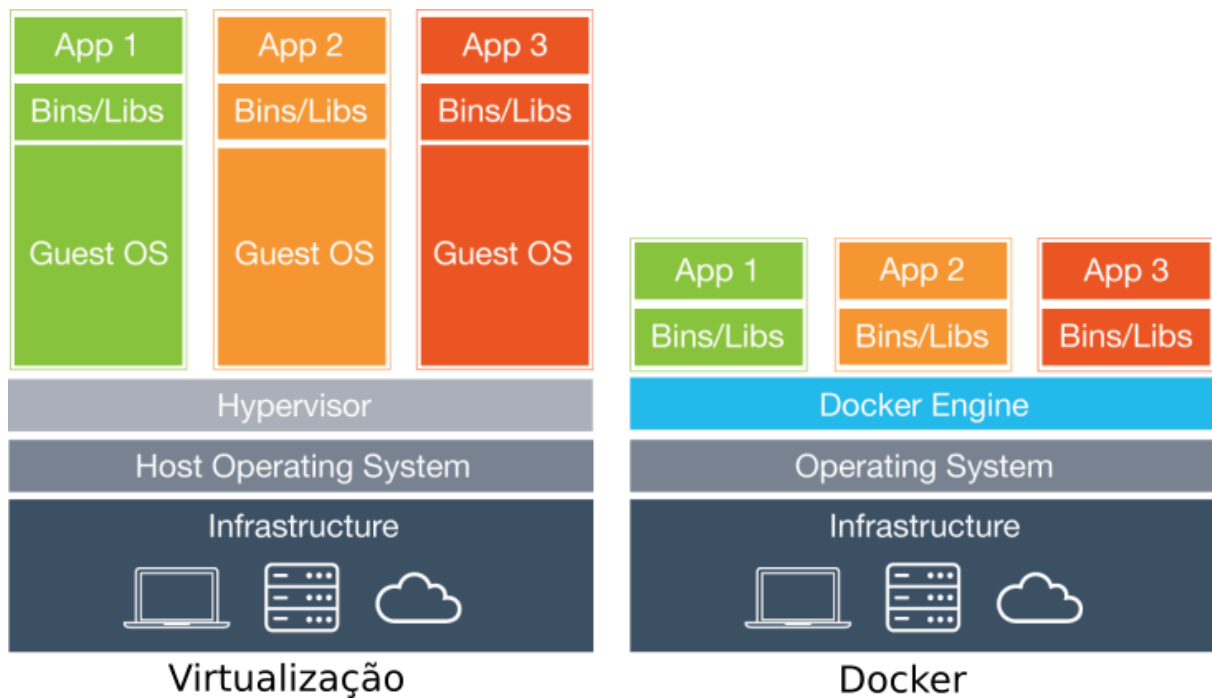


Docker

Docker é uma plataforma que facilita a criação, distribuição e execução de aplicativos em ambientes isolados chamados **containers**. Containers são leves e portáteis, permitindo que desenvolvedores empacotem um aplicativo junto com todas as suas dependências (bibliotecas, configurações, etc.) em um único pacote, que pode ser executado consistentemente em diferentes ambientes, como desenvolvimento, teste e produção.

Estrutura do Docker:

1. **Docker Engine:** O núcleo do Docker, que inclui o Docker Daemon (o serviço que gerencia os containers), a CLI (linha de comando para interagir com o daemon) e o REST API (interface de programação).
2. **Dockerfile:** Um arquivo de script que contém as instruções para criar uma imagem Docker. Ele define como o container será construído, incluindo o sistema operacional base, as dependências e os comandos a serem executados.
3. **Imagens Docker:** São modelos de read-only que contêm tudo o que o aplicativo precisa para rodar (código, dependências, variáveis de ambiente, etc.). As imagens são construídas a partir do Dockerfile.
4. **Containers:** Instâncias em execução de uma imagem Docker. Eles são isolados do sistema operacional subjacente e de outros containers, mas podem compartilhar recursos como a rede e o sistema de arquivos.
5. **Docker Hub:** Um repositório de imagens onde você pode compartilhar e baixar imagens Docker.



Como Funciona:

1. **Criação da Imagem:** O desenvolvedor escreve um Dockerfile e usa o comando `docker build` para criar uma imagem Docker.
2. **Execução do Container:** A partir da imagem, o Docker cria e inicia um container usando `docker run`, que executa o aplicativo em um ambiente isolado.
3. **Compartilhamento:** Imagens podem ser enviadas para o Docker Hub ou outros repositórios privados para serem compartilhadas com outros desenvolvedores ou para serem usadas em diferentes ambientes (desenvolvimento, teste, produção).
4. **Escalabilidade:** Docker facilita a escalabilidade, pois múltiplos containers podem ser executados em paralelo, replicando a aplicação de forma eficiente.

Clone do Projeto

Antes de darmos inicio ao docker, precisamos realizar um clone do projeto que será utilizado para realizarmos nossos estudos.

```
https://github.com/BuntyRaghani/spring-boot-hello-world
```

Comandos do Docker:

- Para ver quantos e quais containers estão rodando na nossa máquina:

```
docker ps
```

Caso de erro de permissão, é preciso adicionar seu usuário ao

- Para rodar uma imagem deve se usar o **run**, para isso vamos usar uma imagem oficial do docker de testes:

```
docker run hello-world
```

⚠ ATENÇÃO: o docker só irá ficar rodando, deve existir um processo ativo dentro do container. Caso não tenha um processo ativo, o docker vai rodar a imagem e depois irá parar.

- Para ver o histórico de execução do docker na nossa máquina, utilizamos o:

```
docker ps -a
```

- Para rodar o Ubuntu na nossa máquina, devemos utilizar o comando:

```
docker run ubuntu
```

- Para deixar um processo ativo para que o ubuntu fique executando, utilizamos o comando:
- **-it** ⇒ para conseguir interagir

- `bash` ⇒ inicia um processo no ubuntu, abre outro terminal.

```
docker -it ubuntu bash
```

- Para parar a execução de todos os processos que estão rodando dentro do container:

```
docker stop {idContainer}
```

- Para rodar iniciar de novo o mesmo container:

```
docker start {idContainer}
```

- Para voltar a interagir com o ubuntu

```
docker exec -it {idContainer} bash
```

Projeto

Crie um arquivo em algum lugar do seu projeto com o nome "Dockerfile" na raiz do projeto. Depois precisamos passar nesse arquivo qual imagem usaremos, no nosso caso será o Maven.

1 - Para isso utilizaremos o comando **FROM** para buscar de uma imagem já existente. Depois disso vamos chamar esse comando de build com o **AS**

```
FROM maven:3.8.4-jdk-8 AS build
```

2 - Devemos trazer para nosso container todos os arquivos que estão na nossa aplicação, então vamos copiá-los. Dentro do nosso container vai ter uma pasta `app` e dentro dela vai ter a pasta `src`.

Além disso, também devemos copiar o `pom.xml` para o nosso container.

```
COPY src /app/src  
COPY pom.xml /app
```

3 - Devemos mudar para o diretório /app. Depois devemos fazer as instalações das dependências com o maven e o build da nossa aplicação spring.

```
WORKDIR /app  
RUN mvn clean install
```

4 - Precisamos pegar o jdk da imagem da Amazon

```
FROM openjdk:9-jre-alpine
```

5 - Devemos agora copiar o arquivo jar gerado na nossa aplicação para dentro de /app/jar.

```
COPY --from=build /app/target/spring-boot-2-hello-world-1.0.2-SNAPSHOT.jar
```

6 - Trocamos de novo nosso WORKDIR para pois não estamos mais dentro do build sim do jdk.

```
WORKDIR /app
```

7 - Agora vamos expor a porta 8080 para esse container.

```
EXPOSE 8080
```

8 - E por fim vamos colocar o comando que vai executar/iniciar a execução da aplicação.

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

Nosso arquivo Dockerfile então deve ficar assim:

```
FROM maven:3.8.4-jdk-8 AS build
```

```
COPY src /app/src
```

```
COPY pom.xml /app
```

```
WORKDIR /app
```

```
RUN mvn clean install
```

```
FROM openjdk:9-jre-alpine
```

```
COPY --from=build /app/target/spring-boot-2-hello-world-1.0.2-SNAPSHOT.jar
```

```
WORKDIR /app
```

```
EXPOSE 8080
```

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