PÓS TECH

FULL STACK DEVELOPMENT

FASE 2 | AULA 01 -

INTRODUÇÃO AO DOCKER

SUMÁRIO

O QUE VEM POR AÍ?

Nesta aula, você aprenderá o que são containers, qual sua função e como eles ajudarão a resolver problemas com recursos do computador, tais como CPU, memória RAM, disco etc.

Você também aprenderá a instalar o subsistema Linux no Windows (o WSL), suas versões e distribuições (Ubuntu, Debian) e como instalar corretamente o Docker, além de aprender sobre o que é Docker, como ele funciona e quais são as vantagens de usá-lo. Por fim, executaremos o nosso primeiro container.

HANDS ON

Nesta primeira aula, faremos a instalação do WSL via Powershell no Windows. Depois, com o ambiente Linux rodando, faremos a instalação do Docker Desktop e a configuração para que seja feita a integração com o WSL. É no ambiente Linux que executaremos os comandos relacionados ao Docker. Por fim, executaremos alguns comandos básicos de Docker para entender um pouco sobre a sua estrutura e como funciona. Executaremos então o nosso primeiro container.

SAIBA MAIS

O Docker foi lançado em 2013 pela empresa dotCloud. É uma ferramenta Open Source que foi criada com o intuito de melhorar o isolamento e uso de recursos, além de rodar aplicações sem a necessidade de uma interface gráfica, se comparado com uma VM (ou Virtual Machine). A sua existência só foi possível graças à possibilidade de isolar processos e virtualização, funcionalidades que estão presentes no Kernel do Linux.

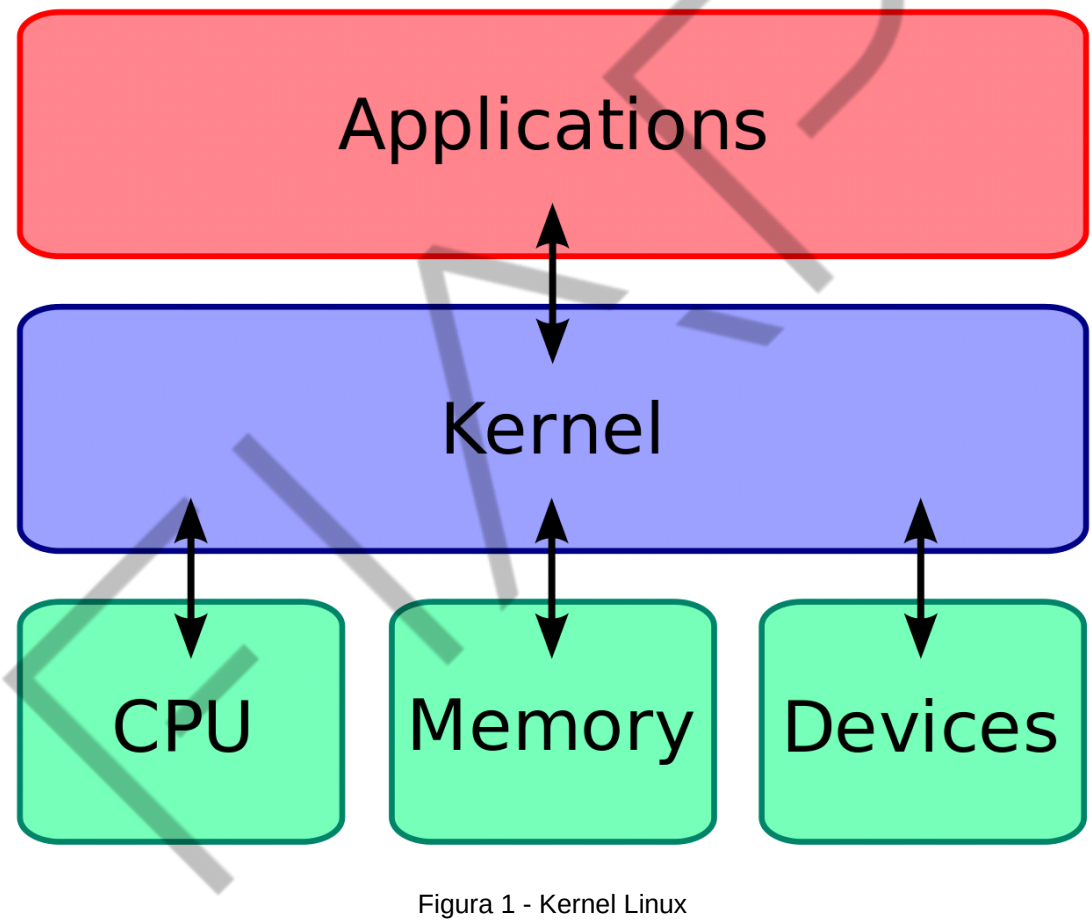

Figura 1 - Kernel Linux
Fonte: https://digilent.com/blog/demystifiying-the-linux-kernel/ (2023)

Por sua vez, o Kernel Linux é o principal componente do sistema operacional Linux. A interface central se comunica entre o hardware e seus processos, controlando todas as principais funções, e é responsável por gerenciar os recursos com eficiência. Em tradução livre, kernel significa “núcleo”.

As principais funções do Kernel Linux são gerenciar memória, processos, drives, chamadas do sistema e segurança.

Junto do Kernel temos duas outras tecnologias que isolam o uso dos containers, chamadas de “Cgroups” e “Namespaces”.

As Cgroups são responsáveis por definir limites no uso de recursos em processos Linux. Com essa tecnologia, podemos definir o quanto de CPU um determinado processo pode usar, priorizando os que forem mais importantes. Com um único comando, também é possível colocar limites no uso de recursos como CPU e memória, por exemplo, mantendo o controle de quais processos podem ou devem ser reiniciados, interrompidos ou congelados. A tecnologia Cgroups também é usada para controlar processos em execução em um container e é essencial para o funcionamento correto e sem gargalos do processo.

Já a tecnologia “Namespaces” permite criar e lidar com diferentes cenários em um mesmo sistema. Com ela, conseguimos analisar propriedades globais diferentes e isoladas em cada contexto. O container tem um ambiente isolado, similar ao de uma VM (Virtual Machine), mas podem existir containers diferentes, com aplicações diferentes, rodando em paralelo em um mesmo servidor.

Os Namespaces são responsáveis por criar esses isolamentos de forma transparente e lógica, podendo cada container ter um conjunto de usuários, rede diferente, disco e assim por diante. Dessa forma, caso algum dos containers rodando quebre por algum motivo (como usar 100% de uma CPU, por exemplo), os demais containers rodando em paralelo não terão o mesmo problema. Isso garantirá que aplicações diferentes e que não se conversam não sejam afetadas por conta do container "ruim".

Imagine um navio cargueiro carregando vários containers. Caso algum container caia no mar, os demais não serão afetados, nem mesmo o navio cargueiro (que representa o host onde os containers estão rodando). A figura “Containers” representa essa ideia.

Figura 2 - Containers
Fonte: wtsnet.com.br (2023)

A diferença entre VMs e Containers são poucas, mas mudam completamente o modo de execução e isolamento das aplicações. As duas imagens a seguir retratam um pouco esse cenário.

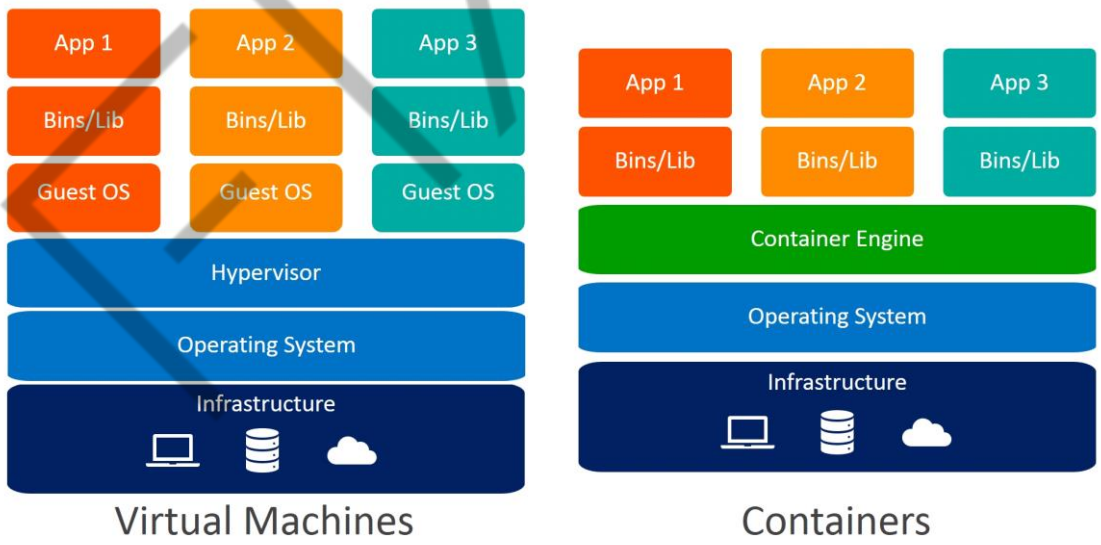

Figura 3 - Diferenças entre VMs e Containers
Fonte: Elaborado pelo autor (2023)

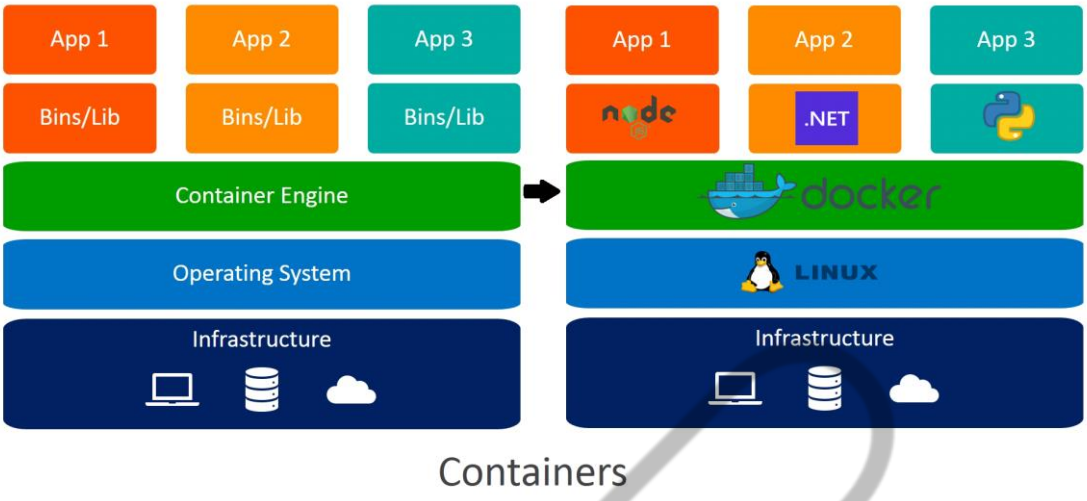

Figura 4 – Ilustração do conceito de containers
Fonte: Elaborado pelo autor (2023)

O Docker tem algumas funcionalidades vitais para criar e rodar um container. As primeiras são as imagens. Elas são responsáveis por empacotar o sistema operacional, dependências, portas expostas e variáveis de ambiente. Existem milhares de imagens personalizadas já criadas, mas também é possível criar a sua própria imagem e compartilhar com a comunidade.

Imagine que para rodar uma aplicação NodeJS seja necessário instalar a versão do Node compatível com o seu código, instalar as dependências pelo NPM/YARN e realizar a configuração padrão (que toda aplicação node precisa ter pré-instalada) para que o código criado possa funcionar. As imagens personalizadas têm esse papel na comunidade.

O Docker Hub é basicamente um Github voltado para imagens Docker. Assim como o Github tem repositórios públicos que qualquer pessoa consegue clonar, fazer o fork e usar o código criado por outra pessoa, o Docker Hub funciona de maneira muito similar. Nele, encontraremos imagens oficiais e imagens não oficiais (criadas pela comunidade), seja para ajudar outras pessoas ou para benefício próprio, como meio de salvar as imagens em um local seguro (privado). Podemos usar essas imagens fazendo o "pull" por meio da URL e tag da imagem.

O que são as Tags?

As tags são utilizadas para definir uma versão da imagem criada, como forma de controle, atualização e organização. As tags ajudam a aplicar melhorias constantes nas imagens, seja para corrigir bugs, melhorar o processo ou até mesmo para realizar correções de segurança. Também é bem parecido com o Github.

Para que essas imagens sejam criadas de forma personalizada, é preciso declarar os comandos, instalações, variáveis, portas etc.

Isso é feito por meio de um arquivo chamado “Dockerfile” (o nome do arquivo já é a sua extensão). É nele que as instruções devem ser feitas. As figuras “Dockerfile simples” e “Dockerfile multistage” ilustram um Dockerfile básico e outro avançado:

Dockerfile simples:

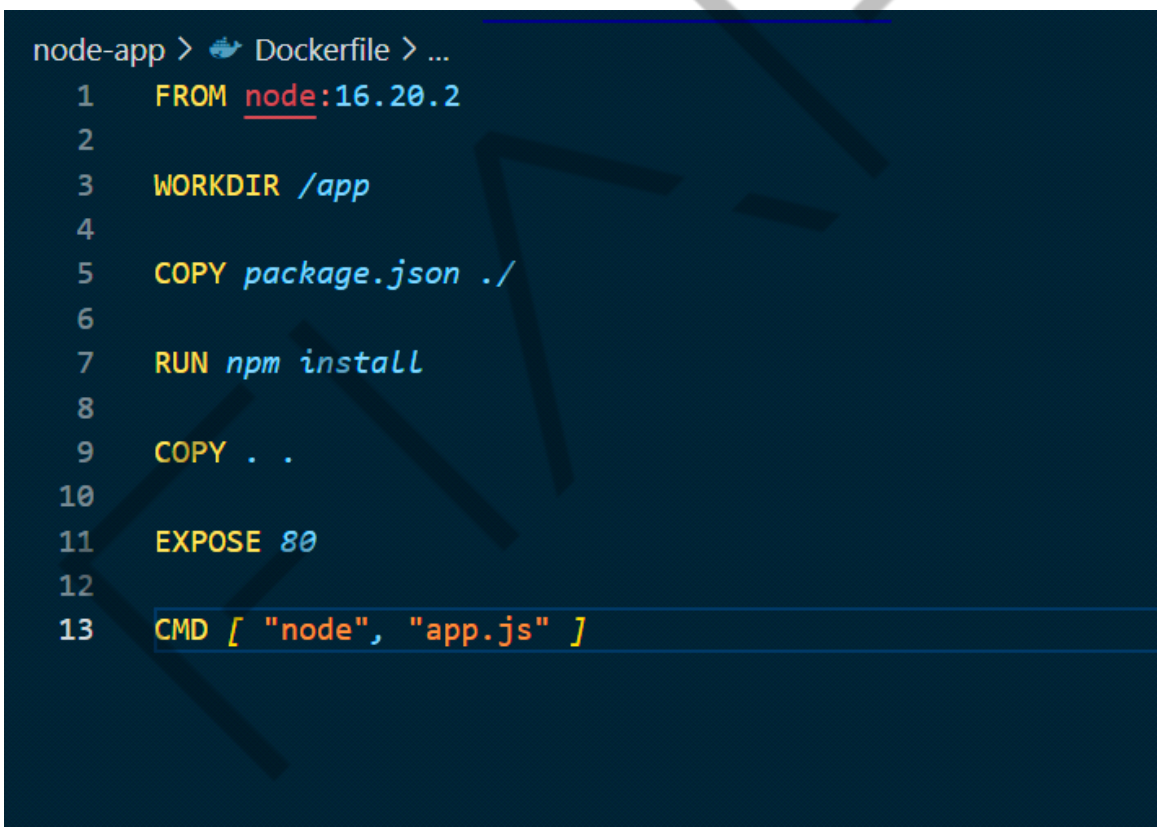

Figura 5 - Dockerfile simples
Fonte: Elaborado pelo autor (2023)

Dockerfile multistage:

Dockerfile-multistage > ...

```
1   FROM node:18 as base
2
3   RUN apt-get update && \
4       apt-get upgrade -y && \
5       npm install -g newrelic && \
6       newrelic save
7
8   COPY newrelic.js /app/newrelic
9   ENV NEWRELIC_LICENSEKEY \
10      NEWRELIC_APPNAME
11
12  COPY . /app
13
14  FROM node:18 as builder
15
16  RUN npm build stage
17  COPY --from=base /app/newrelic /app
18
19  CMD newrelic start
20
21  FROM node:18-buster-slim as deploy
22
23  RUN
24  COPY --from=builder /app /app
25
26  CMD npm start
```

Figura 6 - Dockerfile multistage
Fonte: Elaborado pelo autor (2023)

O QUE VOCÊ VIU NESTA AULA?

Nesta aula, você viu como os containers funcionam. Vimos também a distribuição Linux e a possibilidade de usá-los no Windows por meio do WSL.

Você também aprendeu sobre Docker e seus principais comandos, criou o seu primeiro container e entendeu como ele funciona e qual o seu papel. O conhecimento básico sobre esses temas está disponível nos formatos de texto e vídeo, além de atividades complementares. Aproveitem todo o conteúdo disponível na plataforma. Até as próximas aulas!

REFERÊNCIAS

Como instalar o Linux no Windows com o WSL. Learn Microsoft. 2023. Disponível em: <https://learn.microsoft.com/pt-br/windows/wsl/install>. Acesso em: 08 jan. 2024.

Containers: o que são namespaces e cgroups. ETCD.DEV. 2021. Disponível em: <https://etcd.dev/2021/09/20/containers-o-que-sao-namespaces-e-cgroups/>. Acesso em: 08 jan. 2024.

O que é o Kernel Linux. Red Hat. 2019. Disponível em: <https://www.redhat.com/pt-br/topics/linux/what-is-the-linux-kernel>. Acesso em: 08 jan. 2024.

PALAVRAS-CHAVE

Container. Docker. Linux. WSL.