# **Professor: Danilo Sibov**

# **Docker | Aula 4**

## **Laboratórios**

**Este módulo aborda os seguintes tópicos:**

**● Laboratório 4 - Criando instância Venom**

**Será necessário:**

- E-mail para acesso a conta DockerHub

- Ambiente Hospedeiro para o Docker já preparado

**Referência Bibliográfica**

* Site oficial Documentação Docker -[**https://www.docker.com/**](https://www.docker.com/)
* Site oficial DockerHub -[**https://hub.docker.com/**](https://hub.docker.com/)
* Instalação Docker no Debain: <https://docs.docker.com/engine/install/ubuntu/>

**Laboratório 4 - Criando instância Venom**

Neste laboratório você aprenderá a usar o Docker, configurá-lo para utilizar a CLI, além de procurar imagens oficiais no Docker Hub, baixar imagens e aprender sobre as boas práticas.

Usaremos essa conta para a construção do nosso repositório de imagens Doc

**Etapa 1 - Baixar o Docker**

**Etapa 2 - Configurar usuário comum para acessar os comandos do Docker**

**Etapa 3 - Baixar imagem do DockerHub**

**Etapa 4 - Ensinar a mostrar imagens baixadas (docker images)**

**Etapa 5 - Boas práticas para criação de imagens Docker**

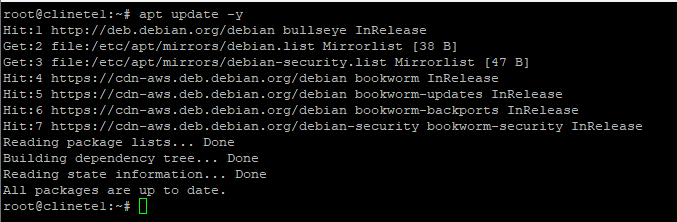
**Etapa 1 - Baixar o Docker**

1. Para baixar o Docker, acesse sua máquina virtual Debian, através do terminal e digite o seguinte **comando:** “sudo su -” para acesso como super usuário,



1. Com acesso “root”, iniciaremos atualizando o sistema

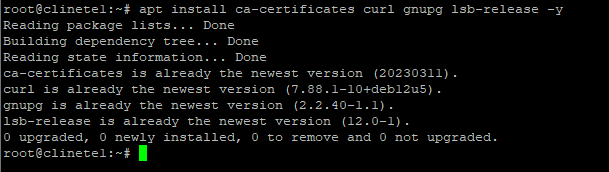




**comando:** apt-get update -y

1. Após atualizar o sistema**,** instale os seguintes pacotes:

* ***ca-certificates***
* ***curl***
* ***gnupg***
* ***lsb-release***



**comando:** apt install ca-certificates curl gnupg lsb-release -y

1. Adicione a chave **GPG oficial**

Criar o diretório “keyrings” dentro de /etc/apt





**comando:** mkdir -p /etc/apt/keyring

Após criar o diretório acima, vamos adicionar chave GPG

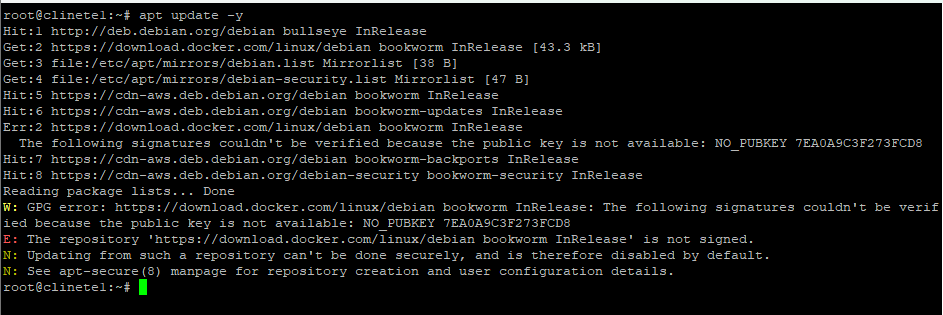
1. **adicionar a chave GPG:** curl -fsSL

**comando:**

curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o /etc/apt/keyrings/docker.gpg

1. **Configurar repositório apt, comando:**

echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/trusted.gpg.d/docker.gpg] https://download.docker.com/linux/debian $(lsb\_release -cs) stable" | tee /etc/apt/sources.list.d/docker.list > /dev/null

1. **Atualize o sistema novamente:**

**comando:** apt update -y

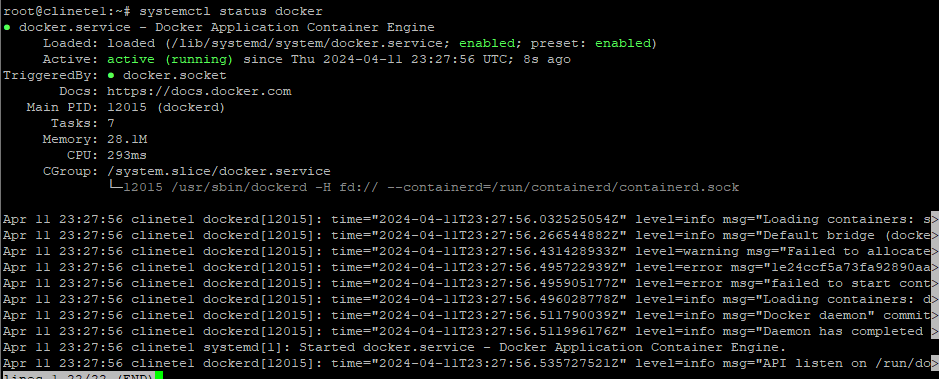
1. **Instalar todos os pacotes do Docker, comando:**

apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin -y

1. **Inicializar o Docker:**

systemctl enable docker && systemctl restart docker

1. **Verificar estado do serviço:**



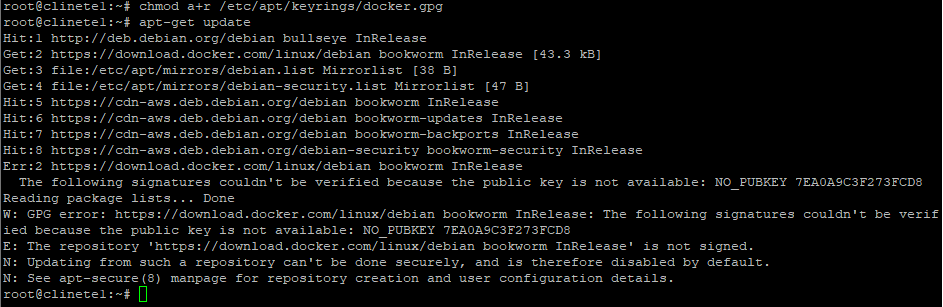
**comando:** systemctl status docker

Saída esperada: **Active (running)**, instalação foi bem sucedida.

**OBS:** Ao executar a etapa 07, caso apareça algum erro, execute os dois comandos abaixos e execute a etapa 07 novamente:

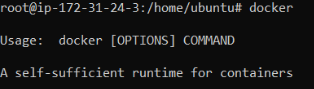
chmod a+r /etc/apt/keyrings/docker.gpg

apt-get update



**Etapa 2 - Configurar usuário comum para acessar os comandos do Docker**

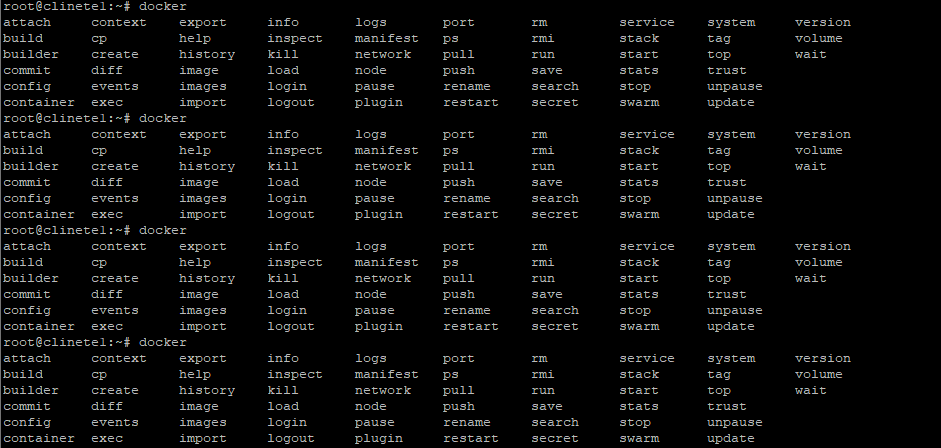
1. Como usuário **root**, verifique a instalação do Docker



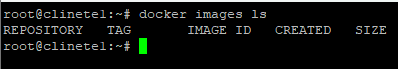
**comando:** docker

Retorno esperado na saída do comando, a instalação está correta.

**Retorno parte 1:**

****

Para verificar se o usuário comum tem acesso aos comandos do docker, digite o **comando:** docker images ls

****

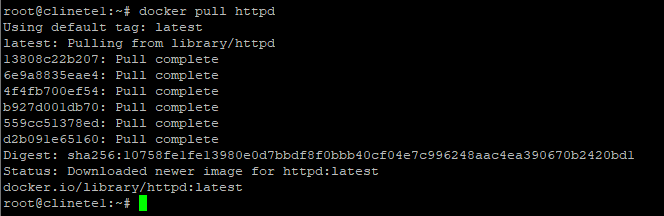
O retorno esperado do comando está na foto acima, a configuração de usuário está correta.

**Etapa 3 - Baixar imagem do DockerHub**

Neste tutorial, você vai precisar do conhecimento do LAB 3, caso não tenha feito esse laboratório, volte antes de seguir.

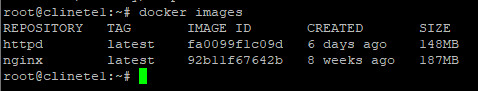
Vamos usar a imagem com Apache2: **httpd**

1. Para baixar essa imagem **httpd** em nosso servidor hospedeiro execute o **comando:** docker pull httpd



Note que o próprio docker irá começar o download da imagem.

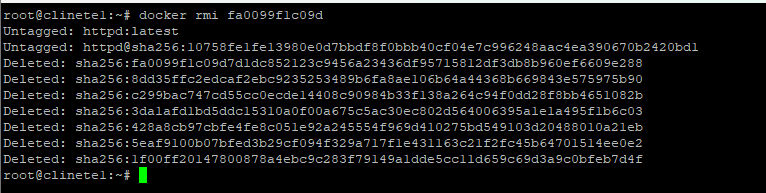
1. Para localizar e listar todas as nossas imagens baixadas no Docker, execute o **comando:** docker images

  
(não é obrigatório ter o nginx)

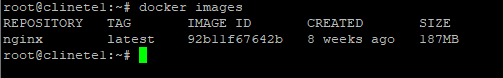
No comando acima, nós podemos ver todas as imagens baixadas, agora, entenda um pouco mais sobre os campos da saída desse comando:

**CAMPOS:**

* **REPOSITORY:** repositório da imagem
* **TAG:** tag da imagem, pode ser latest (última versão) ou v1, slim, v2.
* **IMAGE ID:** ID gerado aleatoriamente pelo Docker para identificar a imagem
* **CREATED:** Quando a imagem foi criada (muitas vezes não por você)
* **SIZE:** Um dos campos mais importantes, mostra o tamanho da imagem, pode ser de MB até GB.

**7. Agora que baixamos nossa imagem, vamos aprender como exclui-la, pois imagens armazenadas localmente utilizam espaço em Disco e embreve, nós aprenderemos como armazenar essas imagens na nuvem. Para deletar as imagens, execute:** docker rmi <nome-do-repository ou ID-da-imagem>

**8. Verifique se existe mais alguma imagem baixada na sua VM, digite:** docker images



Apagou a imagem do httpd

**6 - Boas práticas para criação de imagens Docker**

**1 - Utilizar imagens oficiais e recursos específicos**

Ao construir um arquivo de Dockerfile, sempre utilize imagens oficiais, ou seja, se sua aplicação é feita em NodeJS, utilize uma imagem oficial do NodeJS e assim por diante.

Isso reduz o nível de vulnerabilidade em imagens de terceiros, aumentando a segurança.

**2 - Sempre otimize sua imagem ou utilize imagens bases otimizadas**

Um conceito muito importante quando falamos de container é o seu “tamanho/peso”, se uma imagem Docker for muito pesada, fará com que o container demore para ficar **healthy** (**saúdavel**) e/ou, se estiver utilizando Kubernetes, fará com que demore para baixar a imagem.

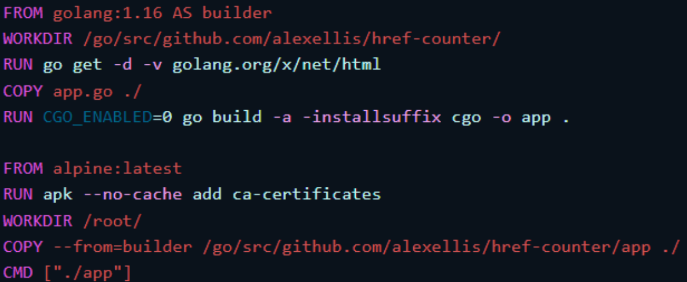
Sempre construa suas imagens Docker o mais otimizadas possível. Você pode fazer isso de diversas maneiras, um exemplo é utilizar imagens oficiais com a tag **slim**, são imagens mais leves e com menos recursos como por exemplo **curl**, além disso, você pode reduzir o número de instruções e complexidade na hora de criar seu Dockerfile.

**3 - Usufrua do multi-stage building**

Quando você quer otimizar ainda mais sua imagem, você pode utilizar um conceito em Docker chamado **multi-stage building**, que consiste em:

* Na primeira série de instruções da sua imagem, ele executará os comandos necessários para gerar um binário executável da sua aplicação, esses comandos podem ser instalar pacotes, requisitos mínimos ou comandos obrigatórios antes da aplicação ficar pronta.
* Na segunda série de instruções, basicamente, sua aplicação será apenas executada.

Isso reduz o nível de complexidade da aplicação, e também é a melhor maneira de otimizar sua aplicação, pois basicamente, na primeira etapa, baixará centenas de arquivos, por exemplo, executar diversos comandos que irão gerar **cache** e isso fará com que a imagem fique maior do que o necessário. Na segunda etapa, você elimina esse cache e pega somente a aplicação que foi construída, um exemplo, o binário executável.

Veja abaixo um exemplo prático desse conceito sendo aplicado:

**4 - Menos é mais**

Quando falamos de segurança, esse conceito que eu chamo de **menos é mais** tem que ser levado a sério, mas o que seria esse conceito?

Simplesmente menos é mais, ou seja, quando formos construir uma imagem Docker para nosso APP que precisa somente de um recurso para funcionar, por exemplo o **curl**, nós podemos procurar imagens oficiais e utilizá-las certo? Sim, porém, se utilizarmos imagens oficiais dessa forma, além do **curl** poderá vir diversos outros recursos e pacotes desnecessários o que pode comprometer a segurança do container.

Uma forma de resolver esse tipo de problema é criar por si só imagens que contenham os pacotes e recursos necessários, ou, procurar imagens específicas oficiais que contenha o mínimo de recursos possíveis.