

Professor: Danilo Sibov

Microserviços | Aula 2

Laboratórios

Este módulo aborda os seguintes tópicos:

- **Laboratório 6 - Container WEB**

Neste laboratório você aprenderá a criar um Dockerfile simples usando um site simples, vai utilizar o parâmetro **-p** e **-v** para subir um site na internet localmente (localhost)

- 1 - Criar um site http simples**
- 2 - Criar um Dockerfile simples com apache**
- 3 - Passar os arquivos do site para o container via -v**
- 4 - Enviando imagens para o Dockerhub**

Baixar APP de teste usado como exemplo para hospedagem

CÓDIGO da Aplicação: <https://github.com/cl0uD-C1SC0/DevOps-Course>

Etapas 1, 2, 3 e 4 - Criando nosso site WEB (Projeto)

1. Para começarmos a criar nosso projeto base, precisaremos de um código de uma aplicação, para isto, foi disponibilizado o código de uma aplicação simples.
2. Vamos instalar o git na nossa máquina virtual, para isto, digite: `sudo apt-get install git -y`

```
ubuntu@ip-172-31-24-3:~$ sudo apt-get install git -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Suggested packages:
  git-daemon-run | git-daemon-sysvinit git-doc git-email git-g
The following packages will be upgraded:
  git
1 upgraded, 0 newly installed, 0 to remove and 72 not upgraded
Need to get 3132 kB of archives.
After this operation, 0 B of additional disk space will be use
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-upd
Fetched 3132 kB in 0s (33.8 MB/s)
(Reading database ... 63929 files and directories currently in
Preparing to unpack .../git_1%3a2.34.1-1ubuntu1.5_amd64.deb ..
Unpacking git (1:2.34.1-1ubuntu1.5) over (1:2.34.1-1ubuntu1.4)
Setting up git (1:2.34.1-1ubuntu1.5) ...
Scanning processes...
Scanning linux images...
```

3. Depois, verifique se o git foi instalado corretamente, digite: **git --help**

```
grow, mark and tweak your common history
branch      List, create, or delete branches
commit      Record changes to the repository
merge       Join two or more development histories together
rebase      Reapply commits on top of another base tip
reset       Reset current HEAD to the specified state
switch      Switch branches
tag         Create, list, delete or verify a tag object signed with GPG

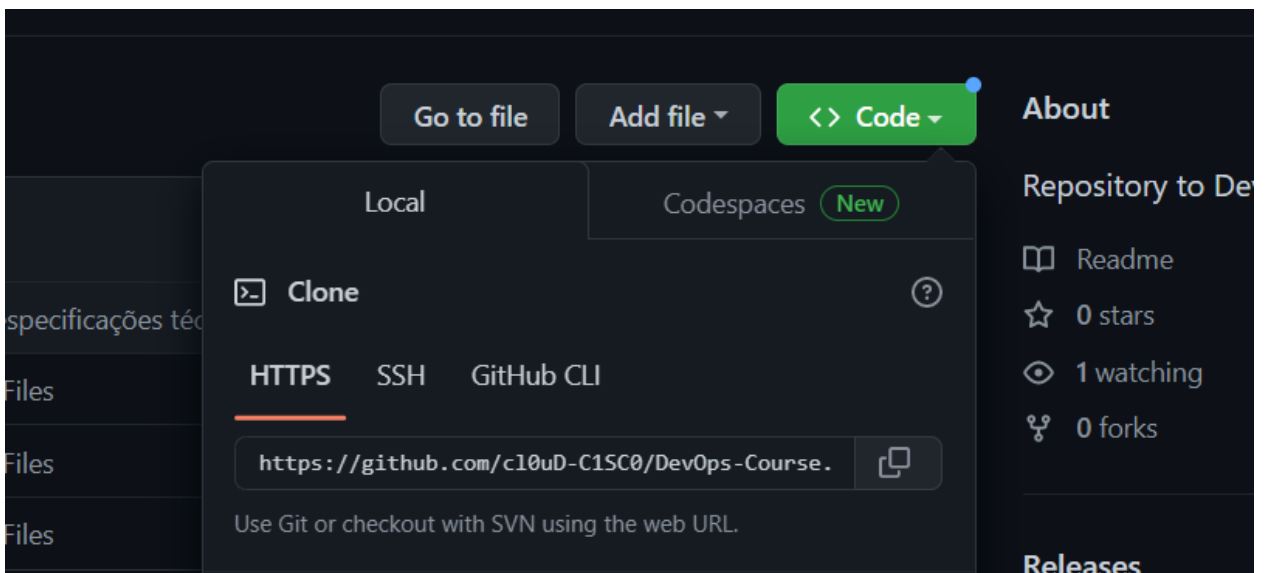
collaborate (see also: git help workflows)
fetch       Download objects and refs from another repository
pull        Fetch from and integrate with another repository or a local
push        Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
See 'git help git' for an overview of the system.
```

Se a saída do comando anterior for parecida com a foto acima, significa que o git foi instalado corretamente.

4. Agora, acesse o link que se encontra no início deste PDF e faça login em sua conta do Github.

5. Depois disso, acesse o link novamente, e procure pelo botão verde: **< > Code**



6. Feito isto, selecione a opção HTTPS e clique nos dois quadrados para copiar o link.

7. Depois, em seu terminal, digite: `git clone <link-copiado>`

```
ubuntu@ip-172-31-24-3:~$ git clone https://github.com/cl0uD-C1SC0/DevOps-Course.git
Cloning into 'DevOps-Course'...
remote: Enumerating objects: 11, done.
remote: Counting objects: 100% (11/11), done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 11 (delta 0), reused 8 (delta 0), pack-reused 0
Receiving objects: 100% (11/11), done.
ubuntu@ip-172-31-24-3:~$
```

8. Feito isso, para esse APP funcionar, nós vamos precisar do NodeJS e o NPM instalados em nossa máquina, siga o roteiro abaixo para instalar ambos:

```
curl -fsSL https://deb.nodesource.com/setup\_16.x | sudo -e bash -
```

```
sudo apt-get install nodejs -y
```

Verifique a versão do NodeJS: `node --version`

Verifique a versão do NPM: `npm --version`

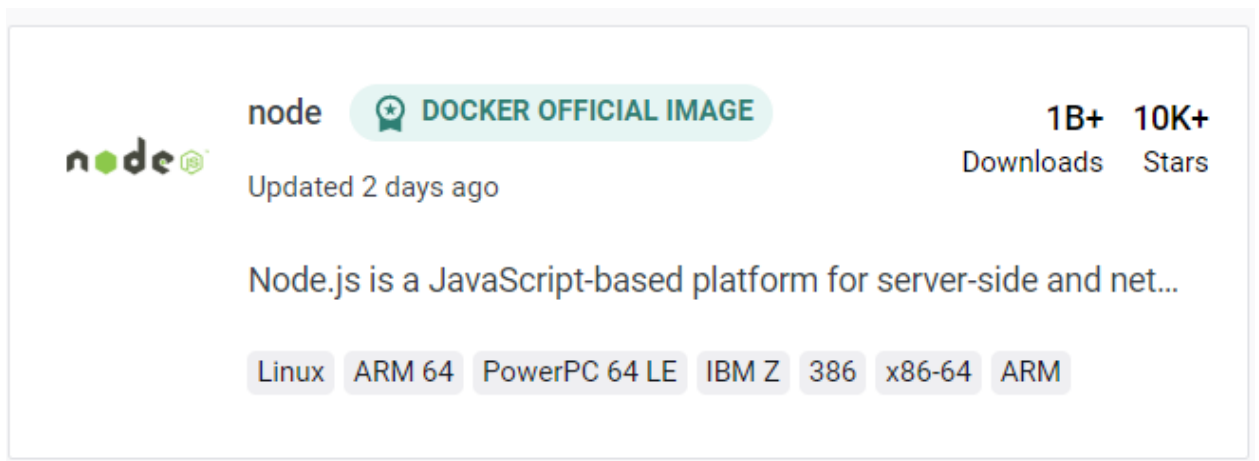
9. Após executar os passos anteriores, vamos agora criar o nosso Dockerfile.

10. Digite no seu terminal: `vim Dockerfile`

11. Aperte a tecla “i” para inserir caracteres no Dockerfile

12. Para evitar problemas de instalação do Node e NPM, vamos utilizar uma imagem original do NodeJS.

13. Acesse o Dockerhub e procure pela imagem **Node**



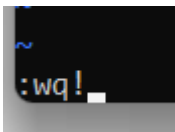
14. Para fins de tutorial, vamos estar utilizando a imagem **19-bullseye-slim**, portanto, construa o seu Dockerfile da seguinte forma:

```
FROM node:19-bullseye-slim
WORKDIR /app/
COPY DevOps-Course/ .
RUN npm install -s express \
    npm install -s mongoose \
    npm install -s body-parser \
    npm install -s socket.io \
    npm install -s http \
    npm install -s dotenv
EXPOSE 3000
CMD ["node", "server.js"]
```

Colocamos o comando **RUN** para ele instalar todas as dependências (bibliotecas) do nosso APP

O comando **node server.js**, ele é necessário para inicializar o nosso backend juntamente com o frontend.

15. Feito isto, salve o arquivo apertando a tecla **ESC** e digitando **:wq!**



16. Depois da construção do nosso Dockerfile, está na hora de “buildar”, digite: `docker build -t <nome-da-sua-imagem> .`

```
ubuntu@ip-172-31-24-3:~$ docker build -t node-app .
Sending build context to Docker daemon 88.58kB
Step 1/4 : FROM node:19-bullseye-slim
19-bullseye-slim: Pulling from library/node
e9995326b091: Already exists
c723fd0ba54b: Pull complete
783ffe98099c: Pull complete
c381bba3b17b: Pull complete
fc456a1679e4: Pull complete
Digest: sha256:c50fa1ec635d2e2523490b968f15422982c50
Status: Downloaded newer image for node:19-bullseye-
---> 2e250cf46aeb
Step 2/4 : WORKDIR /app
---> Running in 54218b314b98
Removing intermediate container 54218b314b98
---> 4e6b4e3a41a4
Step 3/4 : EXPOSE 3000
---> Running in 927b42c7741a
Removing intermediate container 927b42c7741a
---> 66e933a14fee
Step 4/4 : CMD ["node", "server.js"]
---> Running in b6e592013f17
Removing intermediate container b6e592013f17
---> 43c4a06d007a
Successfully built 43c4a06d007a
Successfully tagged node-app:latest
ubuntu@ip-172-31-24-3:~$
```

17. Feito isto, vamos iniciar o nosso container, passando como parametro o `-v` que será o nossos arquivos (`index.html`, `server.js`, etc...) para dentro do container, execute: `docker run -d -p 3000:3000 -v /home/<nome-do-seu-usuario>/DevOps-Course/./etc --name web-app <nome-da-sua-imagem>`

OBS: Substitua o `<nome-da-sua-imagem>` para o nome da sua imagem Docker construída previamente. Substitua também o `<nome-do-seu-usuario>` para o nome do seu usuário.

```
ubuntu@ip-172-31-24-3:~$ docker run -d -p 3000:3000 -v /home/ubuntu/DevOps-Course/./etc --name web-app node-app
526fc890447e8d484caded8c2147bc9f2800c4c5b313d6a066a11b8e9233b185
```

18. Agora, verifique se o container está UP/Healthy, execute: `docker ps`

```
ubuntu@ip-172-31-24-3:~$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED
6ff4103def55   node-app   "docker-entrypoint.s..." About a minute ago
```

19. Vamos verificar se os nossos arquivos estão sendo do `/etc` no container, entre no container, execute: `docker exec -it <nome/ID do container> bash`

20. Se você estiver utilizando a mesma imagem que utilizamos, ao entrar no container, ficará assim:

```
bash-5.1#
```

21. Digite `pwd`

```
bash-5.1# pwd
/app
bash-5.1#
```

Note que estamos no diretório `/app`, o motivo é por conta que no Dockerfile definimos a `Workdir` do nosso container, ou seja, sua área de trabalho por assim dizer.

22. Digite `cd /etc/`

```
bash-5.1# cd /etc
bash-5.1#
```

23. Depois, dentro do diretório `/etc/`, digite `ls`

```
bash-5.1# ls
README.md  hostname  hosts  index.html  resolv.conf  script.js  server.js
bash-5.1#
```

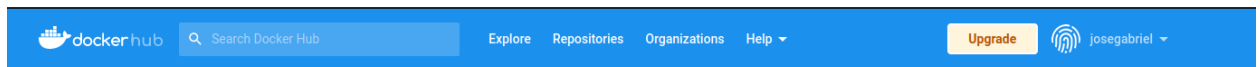
Note que todos os arquivos que nós temos no nosso repositório, também tem dentro do Container.

Essa é uma das principais funções da flag **-v** onde basicamente monta um “ volume “ dentro do container. É bastante comum utilizar isso quando temos um banco de dados em container (o que não é o correto), mas utilizamos o **-v** para salvarmos os arquivos do container localmente ou enviar arquivos para o container.

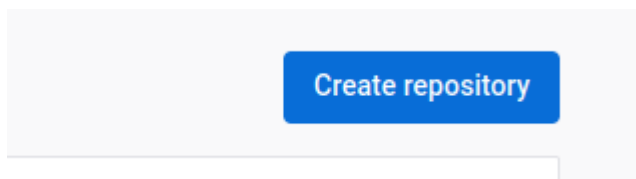
24. Para enviar sua imagem ao Dockerhub, primeiro, acesse:

<https://hub.docker.com/search?q=>

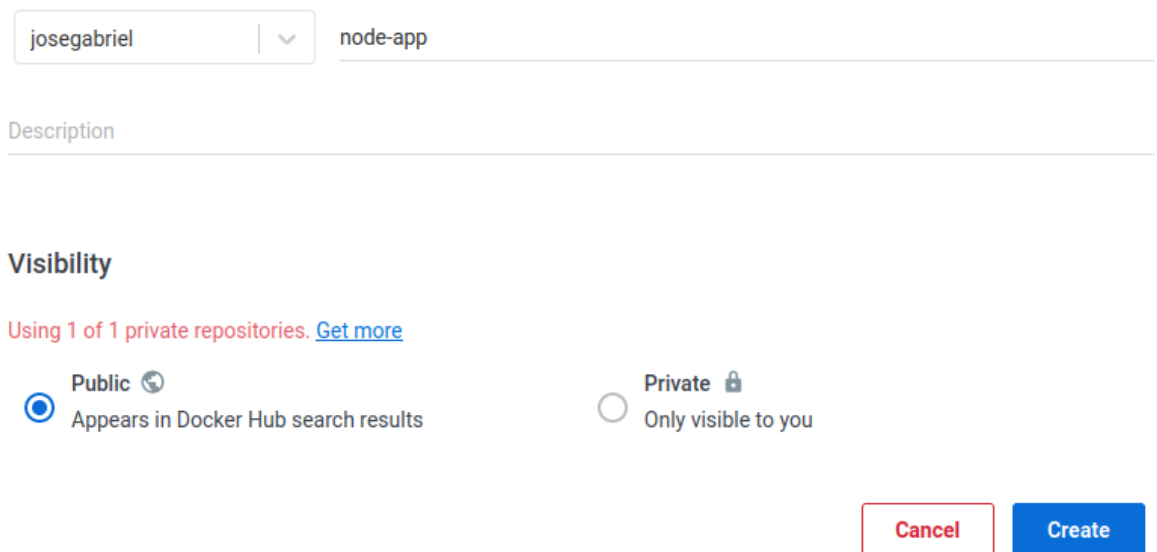
25. No menu, localizado na parte superior do site, selecione Repositories



26. Ao clicar no repositores, clique no botão em azul mais escuro **Create repository**



27. O Nome do repositório, deverá ser o nome da imagem, o nome da imagem, neste caso, é **node-app**, por tanto o nome do repositório será o mesmo



josegabriel | node-app

Description

Visibility

Using 1 of 1 private repositories. [Get more](#)

☒ Public Appears in Docker Hub search results

☐ Private Only visible to you

Cancel Create

Clique em [Create](#), para criar o repositório.

28. Após isso, execute o comando: `docker tag node-app:latest <nome-do-seu-user-no-dockerhub>/node-app:v1`

29. Depois disso, execute: `docker push <nome-do-seu-user-no-dockerhub>/node-app:v1`



```
$ docker push josegabriel/node-app:v1
The push refers to repository [docker.io/josegabriel/node-app]
5c1f75acb53c: Preparing
ae79363dd741: Pushing [=====>] 8.704kB
96abd31ec240: Pushing 1.536kB
aae1fab7b8e1: Preparing
25478c00b6e8: Preparing
61cf42256970: Waiting
f1479df99400: Waiting
882fd36bfd35: Waiting
d1dec9917839: Waiting
d38adf39e1dd: Waiting
4ed121b04368: Waiting
d9d07d703dd5: Waiting
```

30. Depois de ter enviado a imagem para o seu repositório de imagens no DockerHub, recarregue a página e verifique se a imagem foi criada:

Tags and scans

VULNERABILITY SCANNING - DISABLED
[Enable](#)

This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
 v1		Image	---	2 minutes ago

[See all](#)[Go to Advanced Image Management](#)