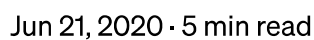




Follow



Replique facilmente as configurações de seu ambiente em qualquer máquina



Uma das vantagens da utilização do Docker em relação a ferramentas como *virtualenv* é a possibilidade de isolar também as dependências do sistema, aqueles pacotes instalados no sistema operacional. Com o Docker diferentes ambientes, previamente preparados, podem ser carregados numa mesma máquina.

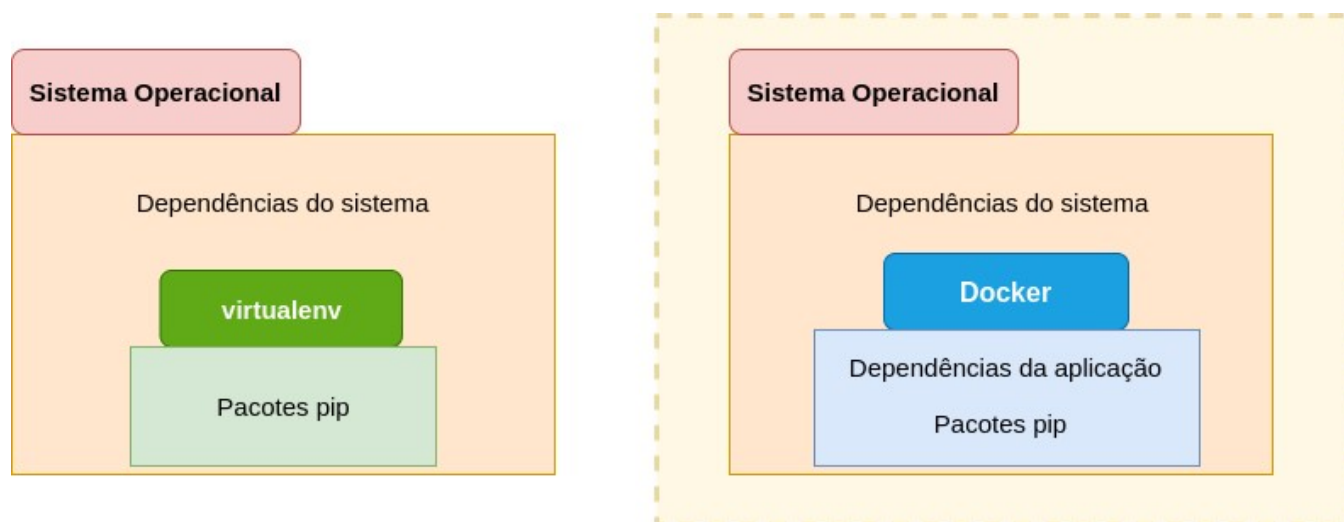


Figura 1-Ilustração do isolamento de ambiente. A esquerda temos o isolamento com virtualenv, que separa os pacotes pip de diferentes ambientes criados. A direita o ambiente docker isolando as dependências da aplicação e os pacotes pip.

Neste tutorial foi utilizado Ubuntu 18.04 como sistema operacional e Docker CE na versão 19.03.5. Para ilustrar o processo de criação de ambiente de desenvolvimento em docker, será criado um ambiente com Ubuntu versão 16.04 e Python na versão 3.5.2 com os pacotes numpy, matplotlib e scipy.

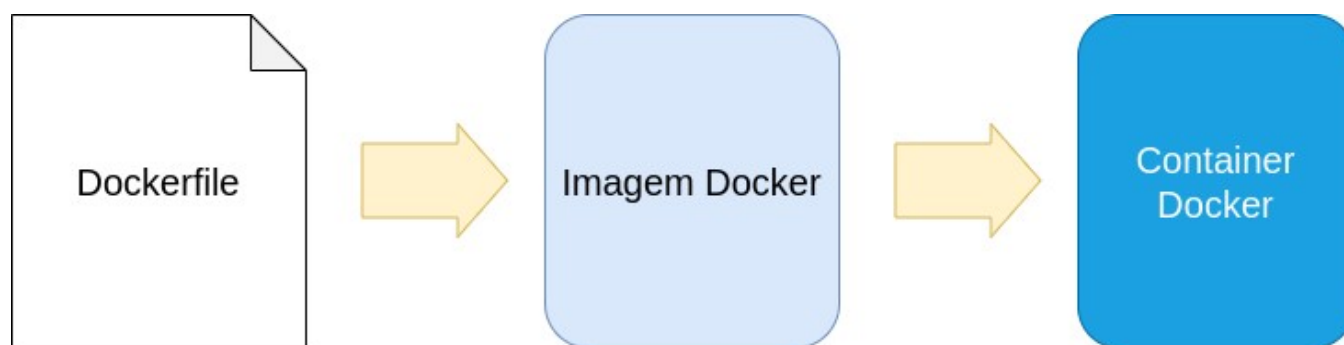


Figura 2-Processo de criação do ambiente de desenvolvimento

O processo de criação de ambiente consiste na criação de uma **imagem docker**(Item 1), a partir dela cria-se um **container**(Item 2) e é neste container que os comandos são

executados(Item 3).

1-Criação da Imagem Docker

A Imagem Docker, ou *Docker Image*, é criada a partir de um arquivo texto contendo essencialmente os comandos que devem ser executados para a criação do ambiente. Para criar uma imagem é necessário que se saiba configurar o ambiente em uma máquina qualquer utilizando o terminal de comando.

Esta é a única etapa que exige conhecimento da construção do ambiente nos mínimos detalhes. Uma vez que a imagem é escrita, toda a equipe ou comunidade de desenvolvedores podem se beneficiar desta “receita de bolo”.

Dockerfile é o nome padrão do arquivo texto com as instruções de criação da Imagem Docker. Segue abaixo o exemplo que iremos construir neste tutorial, copie e cole o conteúdo num arquivo sem extensão chamado *Dockerfile*.

#Sistema Operacional

```
FROM ubuntu:16.04
```

#Dependências do sistema

```
RUN apt-get update -y
```

```
RUN apt-get install -y python3-dev python3-pip build-essential
```

#Pacotes do Python

```
COPY requirements.txt /requirements.txt
```

```
RUN pip3 install --upgrade pip #Atualização do gerenciador de pacotes
```

```
RUN pip3 install -r requirements.txt #Instalação dos pacotes listados
```

```
RUN pip3 install jupyter #Instalação do pacote jupyter notebook
```

#Diretório do usuário docker

```
RUN mkdir -p /home/user
```

```
WORKDIR /home/user
```

O comando FROM define o Ubuntu 16.04 como sistema operacional do ambiente. Todos os comandos subsequentes devem obedecer a sintaxe do sistema escolhido. Por exemplo, se ao invés de Ubuntu fosse escolhido o CentOS, os pacotes seriam instalados

através do comando ***yum install*** e não ***apt-get install***.

Após esta primeira definição, o sistema é atualizado e as dependências são instaladas. Qualquer outra dependência poderá ser adicionada nesta linha.

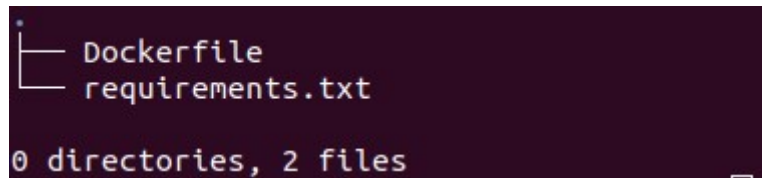


Figura 3-Arquivos do diretório trabalhado neste tutorial

O arquivo *requirements.txt* é copiado para o contexto da Imagem Docker e logo em seguida os pacotes são instalados através do comando ***pip3 install -r requirements.txt***. Esta é uma forma de instalar N pacotes utilizando apenas 1 linha do *Dockerfile*. Uma alternativa menos escalável seria colocar todos os pacotes como argumento do comando ***pip3 install***. Segue abaixo o conteúdo do arquivo texto *requirements*.

```
numpy==1.18.5
matplotlib==3.0.3
scipy==1.4.1
```

A última parte do *Dockerfile* define qual diretório o usuário será direcionado ao iniciar o ambiente.

```
$ sudo docker build --tag=desenvolvimento:python .
```

No diretório que contém os dois arquivos exibidos na Figura 3, execute o comando acima no terminal. Em seguida você poderá acompanhar no terminal a execução de cada comando listado na *Dockerfile*.

2-Construção do container

Após a criação da Imagem Docker, o Container Docker é criado ao executar o comando abaixo.

```
$ sudo docker create -t -i --name python-dev -p 4000:4000 -v  
/home/usuario/workspace:/home/user desenvolvimento:python
```

Parâmetros do comando *docker create*:

- *name* : Identificação do container criado;
- *p* : Mapeamento de portas, a primeira é do Host e a segunda é do Container;
- *v* : Mapeamento de diretórios, o primeiro *path* (“/home/usuario/workspace”) é referente ao Host e o segundo ao Container (“/home/user”). As alterações feitas em um refletem no outro.

Por último é passado como argumento o nome da imagem que o *container* irá tomar como ponto de partida, neste caso será a recém criada ***desenvolvimento:python***.

3-Navegando no terminal do container

O *container* criado necessita ser ativado. Dê um *start* no *container* através do comando abaixo:

```
$ sudo docker start python-dev
```

Observe que não é necessário que o nome do *container* seja igual a identificação da imagem. Execute o comando *exec* para entrar no terminal do *container*.

```
$ sudo docker exec -it python-dev bash
```

Note que o comando *ls* retorna todos os arquivos e diretórios do diretório mapeado na criação do *container*: “/home/usuario/workspace”. Todo arquivo criado dentro do *container* poderá ser acessado fora dele.

Para verificar a distribuição do sistema operacional do *container* basta examinar o conteúdo do arquivo “/etc/os-release”

```
cat /etc/os-release
```

3.1-Inicialização do jupyter notebook

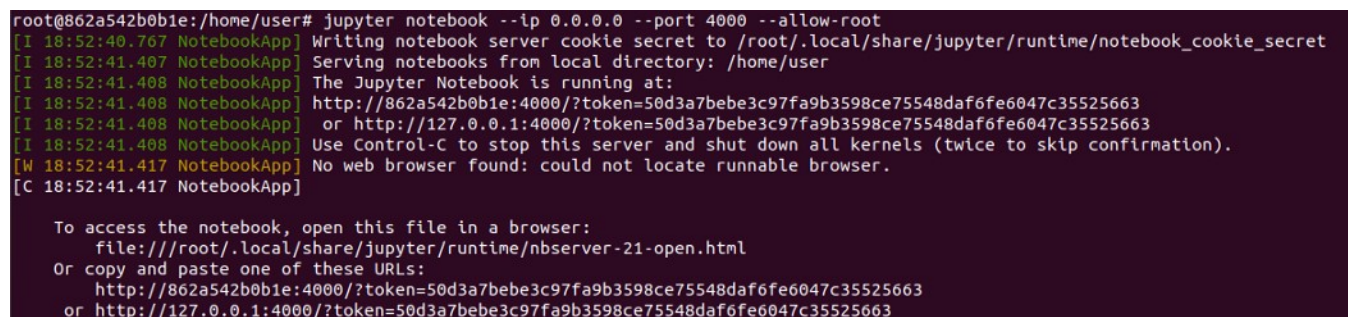
Ainda no terminal do *container python-dev*, inicie o jupyter notebook.

```
$ jupyter notebook --ip 0.0.0.0 --port 4000 --allow-root
```

Em *port* você deve indicar a mesma porta que foi mapeada na criação do *container*.

3.2-Acessando o notebook através do browser

Para acessar o notebook, insira no browser a url iniciada em “http://127.0.0.1:4000/?token”.

A terminal window showing the output of the command 'jupyter notebook --ip 0.0.0.0 --port 4000 --allow-root'. The output includes several status messages from the NotebookApp, such as 'Writing notebook server cookie secret', 'Serving notebooks from local directory: /home/user', and 'The Jupyter Notebook is running at: http://862a542b0b1e:4000/?token=50d3a7bebe3c97fa9b3598ce75548daf6fe6047c35525663'. It also provides instructions on how to access the notebook via a browser, either by opening a file or by copying and pasting the URL. The terminal text is as follows:

```
root@862a542b0b1e:/home/user# jupyter notebook --ip 0.0.0.0 --port 4000 --allow-root
[I 18:52:40.767 NotebookApp] Writing notebook server cookie secret to /root/.local/share/jupyter/runtime/notebook_cookie_secret
[I 18:52:41.407 NotebookApp] Serving notebooks from local directory: /home/user
[I 18:52:41.408 NotebookApp] The Jupyter Notebook is running at:
[I 18:52:41.408 NotebookApp] http://862a542b0b1e:4000/?token=50d3a7bebe3c97fa9b3598ce75548daf6fe6047c35525663
[I 18:52:41.408 NotebookApp] or http://127.0.0.1:4000/?token=50d3a7bebe3c97fa9b3598ce75548daf6fe6047c35525663
[I 18:52:41.408 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[W 18:52:41.417 NotebookApp] No web browser found: could not locate runnable browser.
[C 18:52:41.417 NotebookApp]

To access the notebook, open this file in a browser:
file:///root/.local/share/jupyter/runtime/nbserver-21-open.html
Or copy and paste one of these URLs:
http://862a542b0b1e:4000/?token=50d3a7bebe3c97fa9b3598ce75548daf6fe6047c35525663
or http://127.0.0.1:4000/?token=50d3a7bebe3c97fa9b3598ce75548daf6fe6047c35525663
```

Figura 4-Output do comando ‘jupyter notebook’ digitado no terminal

No meu caso a url é : **http://127.0.0.1:4000/?token=50d3a7bebe3c97fa9b3598ce75548daf6fe6047c35525663**

A partir do acesso via browser, o notebook pode ser utilizado normalmente. A diferença é que os pacotes que serão importados nos notebooks são todos aqueles listados no

arquivo *requirements.txt* e não necessitam estar instalados fora do *container*.

Caso os arquivos criados estejam sem alguma permissão, altere-a através do comando *chmod*.

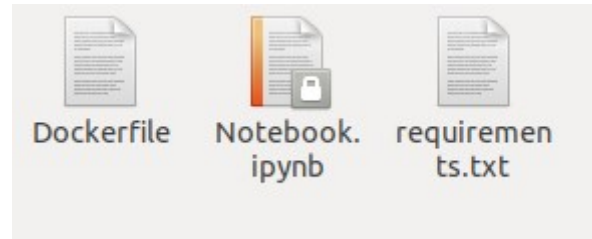


Figura 5-Arquivo criado pelo jupyter notebook iniciado de dentro do container.

Se estiver num ambiente onde não haja nenhuma restrição quanto a visualização, execução e escrita, dê todas as permissões aos arquivos:

```
$ sudo chmod 777 <arquivo>
```

Seguem alguns comandos docker muito úteis. Mais comandos na [documentação](#).

#Criação de imagem

```
sudo docker build -t=<nome:tag> .
```

#Criação container

Estes são os passos necessários para a criação de um ambiente totalmente isolado para o jupyter notebook. Uma vez criado, pode-se compartilhar e usar o ambiente em larga

#Iniciar um container

```
sudo docker start <container>
```

Neste tutorial me concentrei especificamente na criação de um ambiente que utiliza

jupyter notebook. Mas este é apenas um caso, a utilização do Docker como ambiente de

desenvolvimento é muito ampla. Não conheço os limites para a utilização do Docker

como ferramenta de desenvolvimento.

#Remoção imagem

Apesar de ser muito mais trabalhoso, é possível incluir a IDE dentro do Docker e, desta forma, isolar completamente o ambiente de desenvolvimento.

Gostou do artigo? Deixe sua opinião e dúvidas nos comentários.

[DevOps](#)[Python](#)[Jupyter Notebook](#)[Data Visualization](#)[Data Science](#)[About](#) [Write](#) [Help](#) [Legal](#)

Get the Medium app

