

> Ícaro Agostino [Posts](#) [Sobre](#) [Curriculum](#) [Archive](#)

Rodando modelos [R] na nuvem - parte 2: Containers com Docker!

2020-04-20 — Written by Ícaro — 13 min read



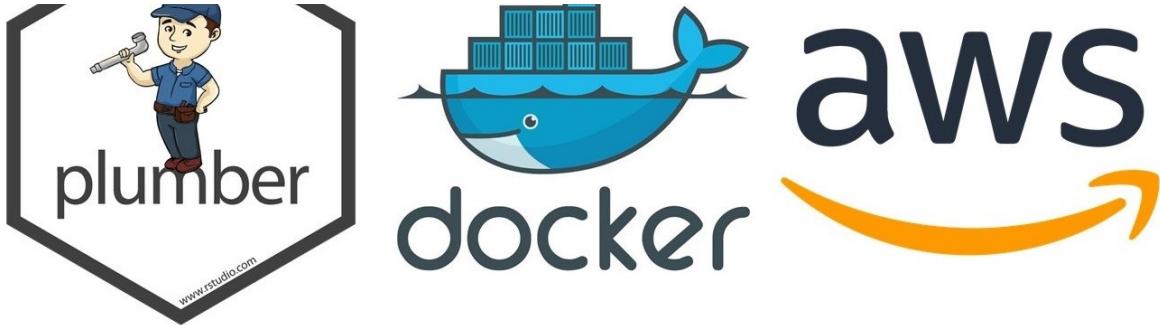
```
R %>% { docker }
```

Parte 2: Dockerizando {r}

Parte 2: Rodando aplicações em containers

Esse tutorial é a segunda parte da série **Rodando modelos [R] na nuvem**, na primeira parte aprendemos como criar APIs com o pacote `plumber` (se você ainda não leu clique [aqui](#)). Nessa parte vamos aprender a como colocar nossa aplicação em containers utilizando `Docker`. E na parte final vamos subir esse container para a cloud da Amazon ([AWS](#)).





O foco aqui será abordar o conceito de container, assim como a parte prática. Vamos entender como o Docker pode salvar nossa vida e simplificar muito o processo de subir uma aplicação para rodar em um servidor. Como falei na [parte 1](#), esse é um passo intermediário entre o desenvolvimento local de um modelo e a fase de produção, quando colocamos o modelo para rodar em um sistema real, ficando disponível para receber requisições de previsões.

Antes de tudo, um *disclaimer*, esse tutorial pode ser considerado mais avançado em relação aos anteriores, assim como mais longo com diversos detalhes e explicações. Vamos ter que operar o Docker direto de um terminal linux (**não se preocupe se você usa windows ou mac, vamos chegar lá**), não precisa ficar com receio caso você nunca tenha feito esse tipo de operação, só tenha paciência para fazer as coisas com calma e se atentar para os detalhes =)

Se você ainda é iniciante em R e tiver dificuldades para executar o tutorial, no final dessa página tem uma indicação de por onde começar com os passos básicos de R .

Todos os scripts desse tutorial estão hospedados no meu Github, clique [aqui](#) para acessar.

Vamos começar...



Aplicações em containers e o Docker

Docker é uma tecnologia muito interessante, que permite isolar uma aplicação dentro de um container, contendo o sistema operacional, os softwares instalados e claro a nossa aplicação. Essa tecnologia fornece uma camada de abstração e automação para virtualização de máquinas, na prática após desenvolver seu modelo, você pode colocá-lo em um container com tudo necessário para que o modelo funcione corretamente.

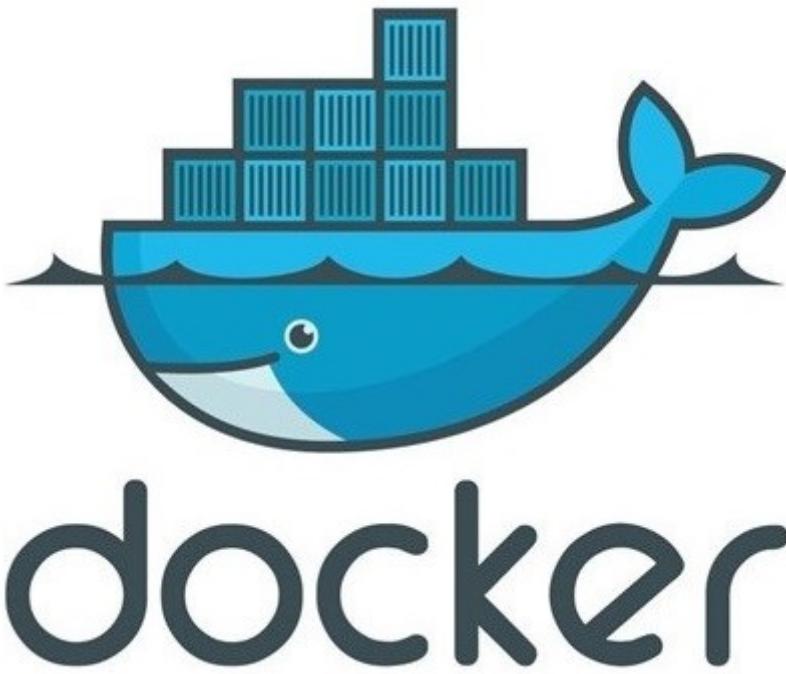


Photo by docker

Esse tipo de tecnologia evita muitas dores de cabeça, quem já precisou configurar um servidor para rodar algo sabe a complicação. Modelos desenvolvidos em R dependem da linguagem instalada, assim como todos os pacotes nas versões corretas ou facilmente podem quebrar.

Após a criação de um container é possível de forma muito simples colocar sua aplicação para rodar em um ambiente cloud, sem precisar se preocupar em instalar linguagens, pacotes ou nada disso no servidor, o container simplesmente possui tudo necessário para nossa aplicação, basta baixar o container criado e dar *play*, simples assim.

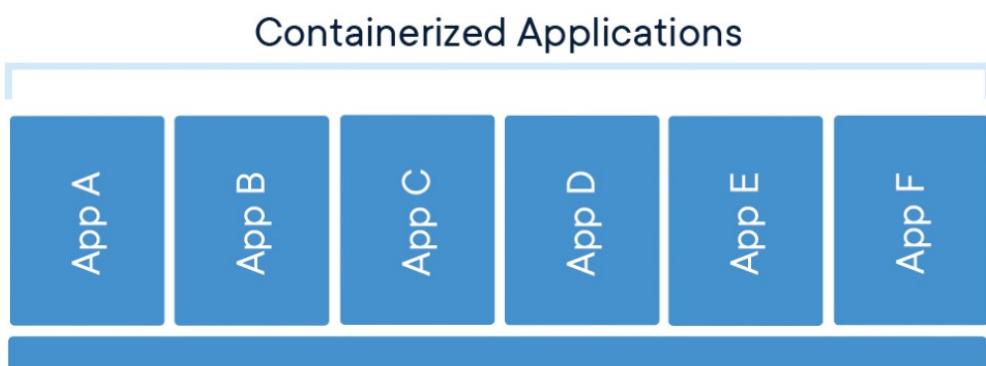




Photo by Docker

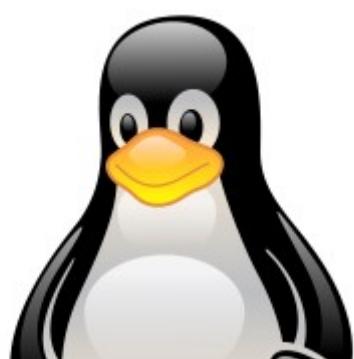
Além disso, usar `Docker` permite escalar uma aplicação, ou um conjunto de aplicações isoladas em diversos containers que podem conversar entre si e serem orquestradas utilizando tecnologias como o `docker compose` ou `kubernets`.

Linux, mas por que?

Se você é usuário de Linux ou Mac pode passar direto para o tópico [instalando docker](#).

Bem, hoje nós temos 3 grandes sistemas operacionais (OS), ou pelo menos 3 grupos: Windows, MacOS e os vários OS baseados em `Linux`. O `Docker` não roda nativamente em todas as versões do Windows, apenas na versão Pro ou Enterprise, se você utiliza windows em uma dessas versões clique [aqui](#) para baixar a versão do desktop. Porém, se você, assim como eu, usa outras versões do Windows (como a home) ainda é possível utilizar o `Docker` utilizando máquinas virtuais =)

Além disso, é importante destacar que apesar do Windows ser o OS predominante nos computadores pessoais, a esmagadora maioria dos servidores é em `Linux`, isso significa que se você está interessado em colocar seus modelos em `R` para rodar em ambientes clouds em produção, mais cedo ou mais tarde você vai esbarrar com a necessidade de aprender a mexer um pouco com `Linux`, não é nada muito complicado, também passei por essa experiência de ser um usuário de Windows tendo que aprender `Linux` para dar o próximo passo em direção ao ambiente cloud.





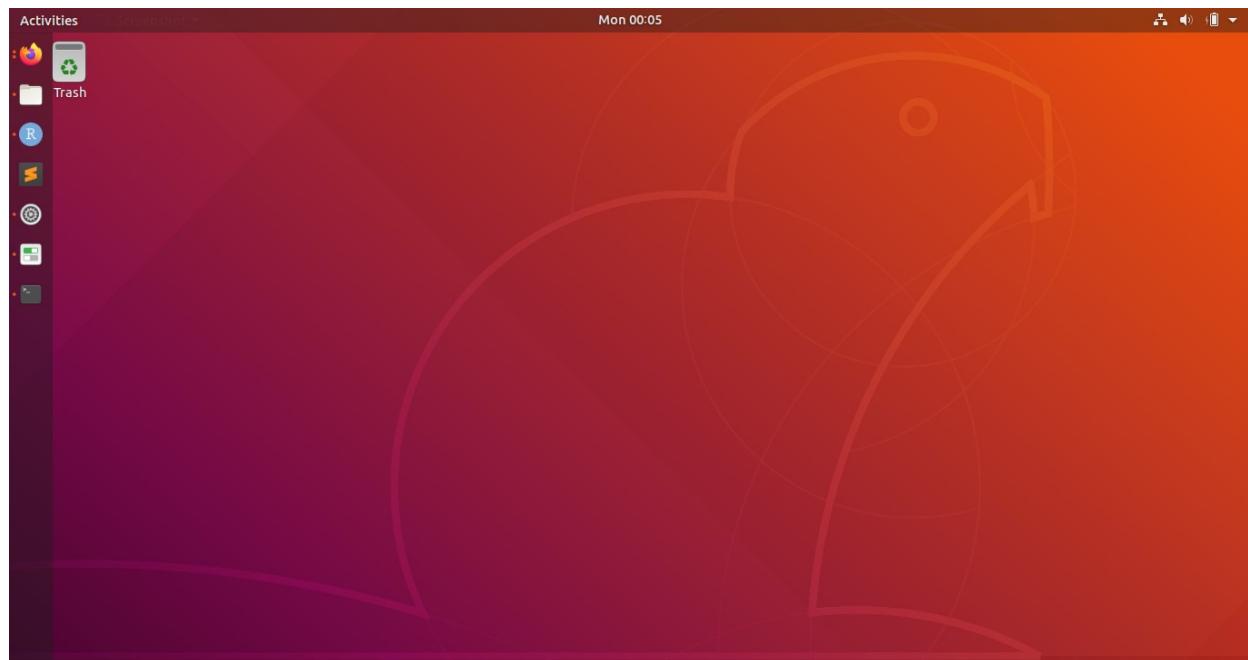
Para rodar um OS tipo `Linux` vamos utilizar **VirtualBox** que é gratuito, esse programa vai permitir rodar um sistema `Linux` virtualizado sem a necessidade de sair do windows, basicamente a máquina virtual vai ser um programa como outro qualquer em que você roda um sistema operacional inteiro dentro dela. Como distribuição, nesse tutorial vamos utilizar o OS **Ubuntu**, sendo um dos OS mais amigáveis para quem não é usuário de `Linux`. Aqui eu vou sugerir seguir o tutorial da WikiHow: [Instalando Ubuntu no Windows](#). Nesse tutorial você vai ver todos os passos de como virtualizar uma máquina Ubuntu.

Esse tutorial vai seguir a linha principal da utilização do **VirtualBox** com `Ubuntu` rodando no Windows. Essa é uma ótima forma de aprender um pouco de `Linux` sem precisar migrar de sistema =)

Instalando Docker

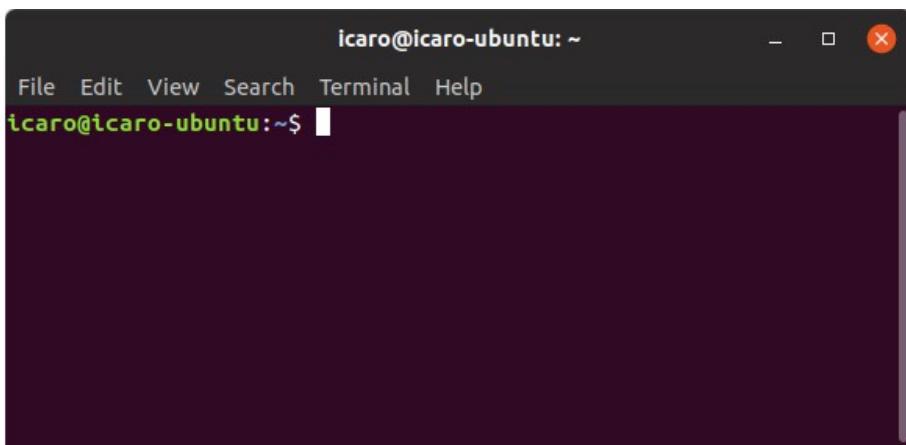
Se você está no Windows Pro ou Enterprise o docker deve ser instalado pelo link que indiquei acima ou [aqui](#). Se você usa Mac siga esse tutorial [aqui](#). Como não uso nenhum desses sistemas, não tenho como indicar o passo a passo para a instalação nessas plataformas, mas pelo que vi é como instalar um programa comum.

Após instalar o `Ubuntu` no VirtualBox e inicializar o sistema você estará num ambiente como esse:





No **Ubuntu** vamos instalar o **Docker** pelo terminal, para abrir basta apertar a tecla **Windows** ou **Super** que abrirá as aplicações, digite terminal e abra a aplicação.



Antes de instalar o **docker** vamos atualizar os pacotes do sistema, digite no terminal:

```
sudo apt update
```

Obs.: Quando usamos **sudo** no começo de um comando estamos dando permissão de administrador para o sistema, se você instalou o sistema com senha será pedido para digitar antes da execução. Esses processos de instalação podem demorar um pouco, você verá vários **códigos** subindo no terminal, é só aguardar.

Agora vamos instalar também o **git** que vai nos permitir baixar os arquivos diretos do **GitHub** na nossa máquina:

```
sudo apt install git-all
```

Com o **git** instalado e os pacotes atualizados podemos instalar o **docker**, para isso use o comando:

```
sudo apt install docker.io
```

Pronto, você já está com tudo necessário para começar a criar containers =)

Entendendo e criando containers

Agora com tudo instalado, podemos criar nossos containers. Para isso precisamos do nossos arquivos. Para evitar ter que configurar todo o R em um sistema que estamos usando apenas para criar o container, podemos baixar os arquivos do GitHub com nosso modelo desenvolvido na parte 1. Peguei o mesmo modelo que configuramos como uma API e deixei tudo pronto para a criação do container.

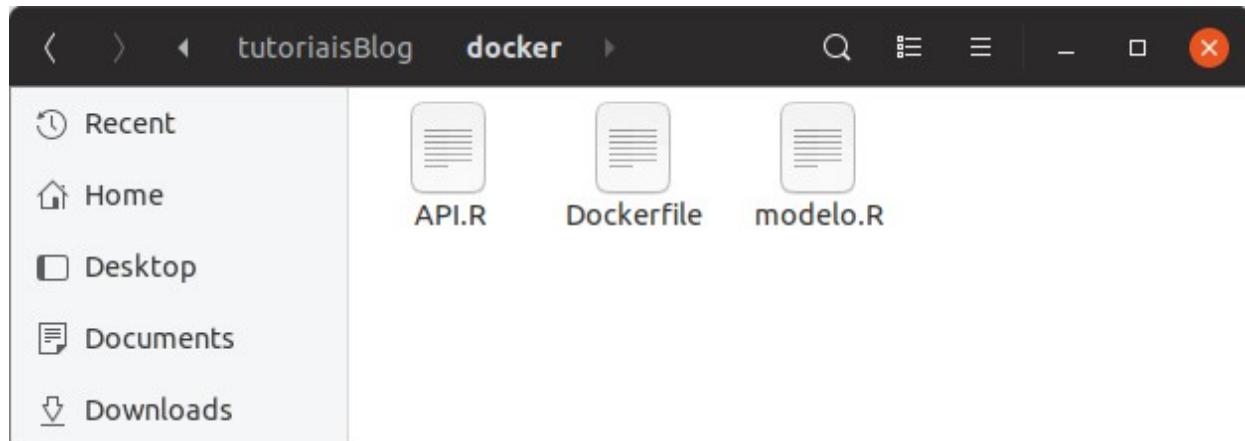
Ainda com o terminal aberto digite o comando a seguir para baixar os arquivos:

```
sudo git clone https://github.com/icaroagostino/tutoriaisBlog
```

Agora o git clonou os nossos arquivos para sua máquina pessoal, provavelmente na sua pasta /home. Agora vamos navegar para a pasta que está os arquivos necessários, digite no terminal:

```
cd tutoriaisBlog/docker
```

Agora estamos no diretório correto. Para entender quais arquivos vamos trabalhar abra o gerenciador de arquivos do Ubuntu entre na pasta tutoriaisBlog -> docker, lá teremos os seguintes arquivos:



Dois desses arquivos são da parte 1 do nosso tutorial, a novidade aqui é o Dockerfile. Se abrirmos esse arquivo, veremos o seguinte:

```
FROM rocker/r-ver:3.6.3

RUN apt-get update -qq && apt-get install -y \
    libssl-dev \
    libcurl4-gnutls-dev \
    libxml2-dev

RUN R -e "install.packages('plumber')"
```

```
COPY API.R modelo.R ./  
  
EXPOSE 8080  
  
ENTRYPOINT ["R", "-e", "source('API.R')"]
```

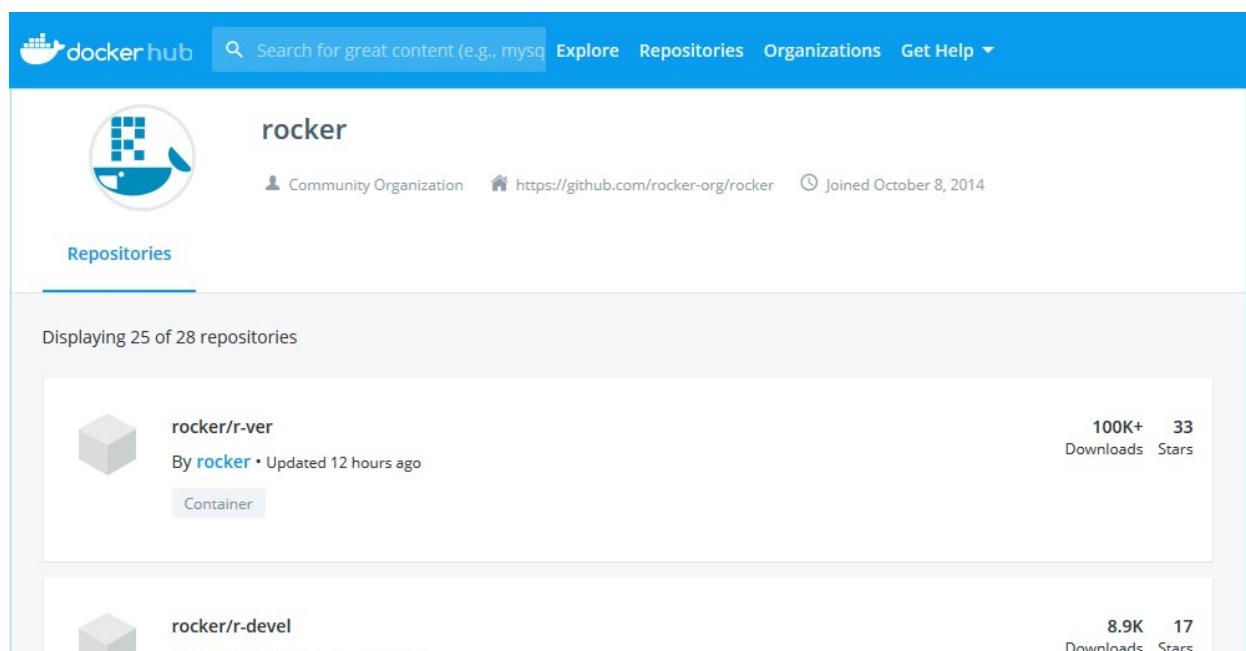
O `Dockerfile` é o arquivo que define a construção do nosso container, vamos passar por todas as linhas desse arquivo explicando para que fique claro a criação de containers utilizando `Docker`. Não precisa executar nada nos próximos parágrafos, aqui vamos apenas explicar a estrutura desse arquivo para que você possa replicar no futuro, se quiser pular a explicação do `Dockerfile`, e ir direto para a construção do container clique [aqui](#).

Entendendo a estrutura de um container

A primeira linha é a imagem base para a criação do nosso container. Aqui podemos começar com uma imagem básica de um sistema operacional e depois ir instalando o que precisamos por cima. Para nossa sorte temos diversas imagens já prontas contendo algumas ferramentas já instaladas. No nosso caso vamos instalar uma das imagens da `rocker`, que é a comunidade que mantém imagens de container configuradas com diversas versões de `R` e pacotes já instaladas.

Todas essas imagens de containers ficam disponíveis gratuitamente para baixar no `docker hub`, que funciona como um `GitHub` só que para containers. Acesse a página do `rocker` no `docker hub` para ver as possibilidades:

Link: <https://hub.docker.com/u/rocker>



The screenshot shows the Docker Hub interface. At the top, there's a search bar with placeholder text 'Search for great content (e.g., mysql)', navigation links for 'Explore', 'Repositories', 'Organizations', and 'Get Help', and the Docker Hub logo. Below the header, the 'rocker' organization page is displayed. It features a circular profile picture with a blue and white geometric pattern. To the right of the picture, the organization name 'rocker' is shown, along with its status as a 'Community Organization' and a link to its GitHub repository (<https://github.com/rocker-org/rocker>). It also shows the date it was joined: October 8, 2014. A 'Repositories' tab is selected, indicated by a blue underline. Below this, a message says 'Displaying 25 of 28 repositories'. Two repository cards are visible: 'rocker/r-ver' and 'rocker/r-devel'. Each card includes a small icon, the repository name, the maintainer ('By rocker'), the last update time ('Updated 12 hours ago'), a 'Container' tag, and download statistics ('100K+ Downloads', '33 Stars' for r-ver, and '8.9K Downloads', '17 Stars' for r-devel).



Temos diversas imagens disponíveis, como por exemplo `R` base com apenas a linguagem, imagens com `tidyverse`, `shiny` e outras versões. Aproveite que você está na página do `docker hub` e faça um cadastro, vamos precisar no final para subir nosso container para o `docker hub`.

Aqui vamos utilizar a imagem `r-ver:3.6.3` que vai nos permitir acessar a imagem que sabemos que será compatível com o modelo que desenvolvemos na parte 1, por isso a primeira linha do nosso `Dockerfile` é:

```
FROM rocker/r-ver:3.6.3
```

Logo estamos falando pro `docker` criar nossa imagem com base na imagem já pronta da `rocker`, isso é um grande atalho, permitindo construir imagens com aplicações utilizando imagens já configuradas. E claro, temos containers com outras linguagem como `Python`, ou com aplicações já prontas como banco de dados.

O comando seguinte do nosso `Dockerfile` é para instalar algumas dependências no `linux` que roda dentro do nosso container. Na sequencia vamos rodar um comando para instalar o pacote `plumber`:

```
RUN R -e "install.packages('plumber')"
```

O próximo passo é copiar os nossos arquivos para dentro do container, em uma aplicação futura você provavelmente terá outros arquivos, com outras estruturas, basta colocar os nomes dos arquivos assim como fiz a seguir, não esquecendo de deixar o `./` no final:

```
COPY API.R modelo.R ./
```

Agora temos nossos arquivos dentro do container, vamos então expor uma porta para que possamos acessar nossa aplicação dentro do container (lembra que na parte 1 desse tutorial escolhemos a porta 8080 para acessar nossa aplicação):

```
EXPOSE 8080
```

E por fim vamos definir o que o container vai fazer quando ele for iniciado:

```
ENTRYPOINT [ "R", "-e", "source('API.R')"]
```

Ufa! terminamos de entender o básico da estrutura de criação de containers, essa parte do tutorial poderia ser pulada, já que eu configurei esse arquivo previamente e poderia só ensinar a execução, mas acredito que essa breve explicação pode auxiliar você no futuro quando for criar seu próprio container com outras necessidades, baseadas em outras imagens, com outras portas e etc.

Criando o container

Finalmente vamos criar nosso container, e esse passo é realmente muito simples, de volta ao terminal (certifique-se que ele está rodando na pasta certa, se você fechou e reabriu deve usar o comando `cd tutoriaisBlog/docker`).

Agora basta digitar o próximo comando (se atente para o ponto no final da linha):

```
sudo docker build -t modelo-linear .
```

Aqui o processo é simples de explicar, o `docker` vai utilizar o arquivo `Dockerfile` que está na pasta que acabamos de explicar. Esse arquivo passa uma sequência de instruções para criação da imagem, esse processo pode demorar alguns minutos. Seja paciente, pois o tempo que gastamos aqui criando o container vamos poupar todas as vezes que precisarmos iniciar nossa aplicação que já vai estar pronta e configurada.

O `docker` traz essa vantagem, pois aqui estamos isolando nossa aplicação em um container com todas as dependências corretamente, sem sequer precisar instalar o `R` na máquina virtual, da mesma forma que não precisaremos instalar a linguagem e os pacotes na cloud na parte final.

Após passar o processo de criação podemos ver nossa imagem e colocar ela para rodar. Para ver a lista de imagens use o comando:

```
sudo docker images
```

Aqui podemos ver duas imagens, a nossa e a da `rocker` que foi baixada:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
modelo-linear	latest	ce4703adbb16	2 hours ago	901MB
rocker/r-ver	3.6.3	b1dcbafd194d	11 hours ago	656MB

Agora vamos colocar nosso container para rodar:

```
sudo docker run -d -p 8080:8080 modelo-linear
```

O comando `run` inicia o container, o comando `-d` serve para rodar o container em segundo plano, se tirarmos essa parte a aplicação vai inicializar e abrir no terminal, não é o que queremos. O comando `-p` serve para dizer como o container se comunica, conectando a porta interna do container com a externa da sua máquina, aqui usamos `8080:8080`. E por fim o nome do container `modelo-linear`.

Use o comando seguinte para ver os containers que estão rodando:

```
sudo docker ps
```

Se tudo deu certo até aqui você verá algo assim:

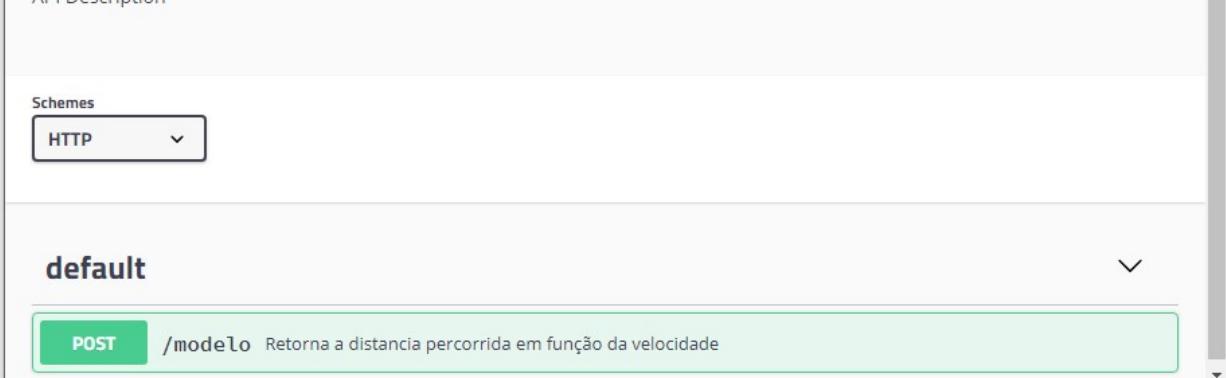
CONTAINER ID	IMAGE	COMMAND	CREATED
6d2ce98af45a	modelo-linear	"R -e source('API.R')"	11 seconds ago
STATUS	PORTS	NAMES	
Up 7 seconds	0.0.0.0:8080->8080/tcp	peaceful_driscoll	

Agora você pode abrir o firefox que já vem instalado no Ubuntu e digitar na pesquisa:

- `http://localhost:8080/_swagger_/_`

E novamente veremos nossa aplicação rodando com `swagger`:





A screenshot of a Docker API interface. At the top, there's a dropdown menu labeled "Schemes" with "HTTP" selected. Below it, a section titled "default" contains a green button labeled "POST" next to the URL "/modelo". A tooltip for this URL says "Retorna a distancia percorrida em função da velocidade".

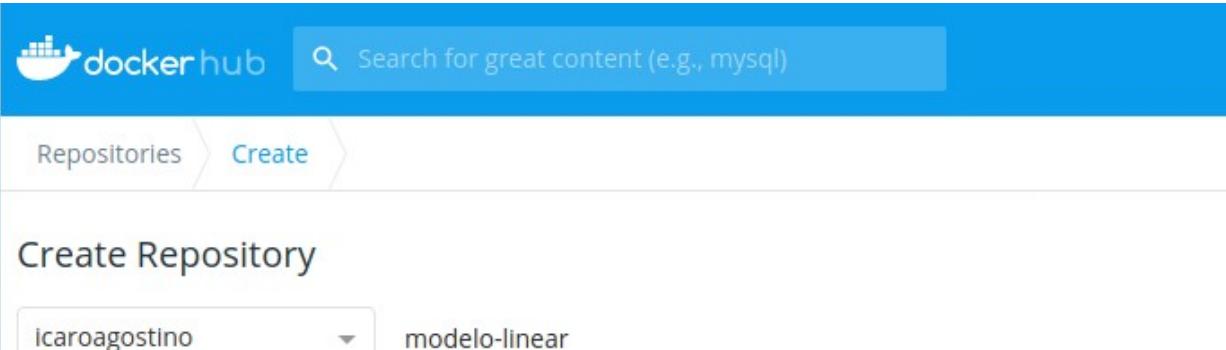
Aqui você pode testar novamente sua aplicação, colocar alguns valores e usar o `post`. Lembre-se que estamos rodando o modelo criado na parte 1 sem instalar `R` ou pacotes na nossa máquina virtual, está tudo rodando isoladamente dentro de um container.



Mas e agora, como salvar e guardar esse container para no futuro subir para uma cloud?

Utilizando o Docker Hub e subindo nossa imagem

Lembra que visitamos a página do `docker hub` para ver as imagens disponíveis, você vai precisar criar uma conta como já falei anteriormente. Após criar sua conta, clique `Create`, você vai agora colocar um nome no seu repositório (sugiro utilizar `modelo-linear` como fizemos antes), a descrição é opcional e podemos escolher deixar público ou privado. Por fim, clique em `Create`.



A screenshot of the Docker Hub website. The header has the "docker hub" logo and a search bar with the placeholder "Search for great content (e.g., mysql)". Below the header, there are navigation links for "Repositories" and "Create". The main area is titled "Create Repository". It shows a dropdown menu with "icaroagostino" selected and a text input field containing "modelo-linear".

Exemplo para o tutorial do Blog

Public 
Public repositories appear in Docker Hub search results

Private 
Only you can view private repositories

Create

Agora de volta ao terminal precisamos renomear nossa imagem com o nosso ID do docker hub , a seguir vou fazer o exemplo com a minha conta e você pode replicar para a sua trocando por seu ID, para renomear usamos o seguinte:

```
sudo docker tag modelo-linear icaroagostino/modelo-linear
```

Utilizando sudo docker images podemos ver que temos duas imagens, uma com seu ID no começo e outra não, elas tem o mesmo IMAGE ID e não ocupam duas vezes o mesmo tamanho, na verdade o docker faz referência de uma imagem para a outra.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
icaroagostino/modelo-linear	latest	ce4703adbb16	2 hours ago	901MB
modelo-linear	latest	ce4703adbb16	2 hours ago	901MB
rocker/r-ver	3.6.3	b1dcbafd194d	11 hours ago	656MB

Agora por fim, vamos subir nosso container para o docker hub , utilizando o comando push :

Não esqueça de trocar o meu nome de usuário pelo seu.

```
sudo docker push icaroagostino/modelo-linear
```

Após o processo terminar podemos voltar a página do docker hub , na aba tags podemos ver que nosso container já está hospedado:

General **Tags** Builds Timeline Collaborators Webhooks Settings

Action Sort by Latest

IMAGE	Actions
latest	docker pull icaroagostino/modelo-linear: latest 
Last updated a few seconds ago by icaroagostino	

DIGEST 1ca4b1a679af	OS/ARCH linux/amd64	COMPRESSED SIZE ⓘ 322.29 MB
------------------------	------------------------	--------------------------------

Podemos baixar nossa imagem de container em qualquer computador, servidor ou cloud com `docker` utilizando o comando `pull`, o próprio `docker hub` já te diz o comando para fazer isso utilizando:

```
sudo docker pull icaroagostino/modelo-linear
```

Pronto, agora temos nosso modelo dentro de um container e hospedado na nuvem pronto para usar em outros locais. Na próxima parte desse tutorial vamos colocar esse container para rodar na cloud da Amazon (AWS) utilizando uma máquina gratuita, que será acessível de qualquer local da internet.

Para aprender a mexer mais no `docker` sugiro sempre ir na documentação oficial do projeto, que é bastante clara e objetiva: <https://docs.docker.com/>

Chegamos ao fim

O papo hoje foi bem mais denso e longo, a continuação para a parte 3 onde abordaremos a AWS está disponível clicando [aqui](#). Se esse tutorial está sendo útil para você e foi possível executar tudo, ou mesmo se teve algum problema me marque em um tweet, meu user é [@icaroagostino](#). Você pode me seguir nas minhas redes (nos botões a seguir), se quiser interagir comigo o twitter é a plataforma que mais uso para se comunicar com a comunidade de [R](#).



Os dados utilizados nesse exemplo são públicos, para mais detalhes baixem os scripts [neste repositório](#).

`R` é uma linguagem de programação open source. Todos os recursos, bibliotecas, dados e implementações são gratuitos e desenvolvidos pela comunidade.

Se você tem interesse de aprender a instalar o R e os passos básicos para iniciar nesse mundo sugiro o tutorial básico desenvolvido pelo pessoal do curso-r: <http://material.curso-r.com/installacao/>

Até o próximo post, bons estudos em  :D

Você pode compartilhar esse post nas redes sociais utilizando os botões no fim dessa página.



READ OTHER POSTS

[← Testando o pacote Cor...](#)

[Rodando modelos \[R\] ... →](#)

Compartilhe em:



› [Ícaro Agostino](#) 

© 2020 Powered by [Hugo](#)

Theme: [panr](#)