

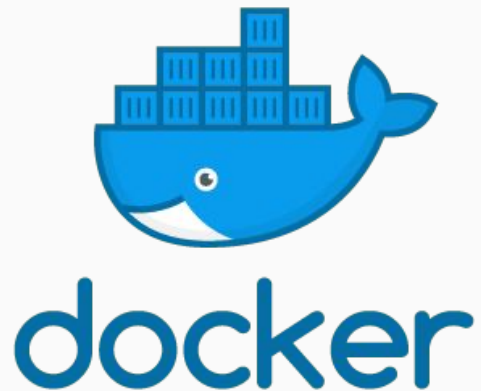
Introdução ao Docker

Aprenda Docker do zero a utilização do dia a dia



O que é Docker?

Docker é uma plataforma para desenvolvedores e administradores de sistemas criarem aplicações a serem executadas em forma de containers.



Quais as suas vantagens?

O deploy em forma de containers tem se tornado popular devido a ser:

- **Flexível:** mesmo aplicações complexas podem ser executadas em containers.
- **Leve:** containers são leves por compartilharem o kernel do servidor.
- **Intercambiável:** atualizações podem ser realizadas sem afetar a aplicação corrente (on-the-fly).



Quais as suas vantagens?

O deploy em forma de containers tem se tornado popular devido a ser:

- **Portável:** a mesma aplicação pode ser executada no desktop, nuvem, em qualquer servidor, sem alterações no código.
- **Escalável:** é possível com muita facilidade criar e incrementar o número de réplicas de um container.



Containers e máquinas virtuais

Um container é executado nativamente e compartilha o kernel do sistema operacional instalado no servidor.

Ele é executado como um processo, e não consome mais memória do que qualquer outra aplicação, fazendo com que ele seja leve e não consuma muitos recursos do servidor.



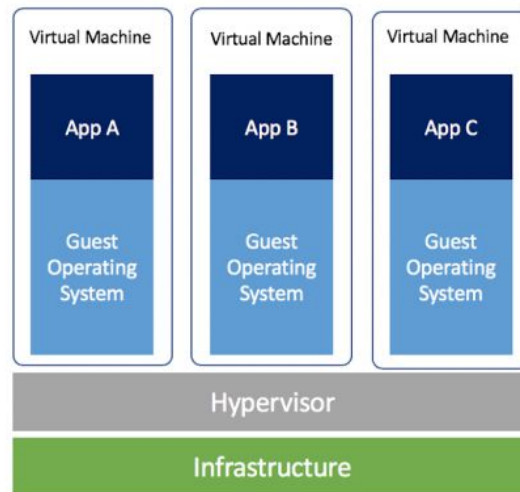
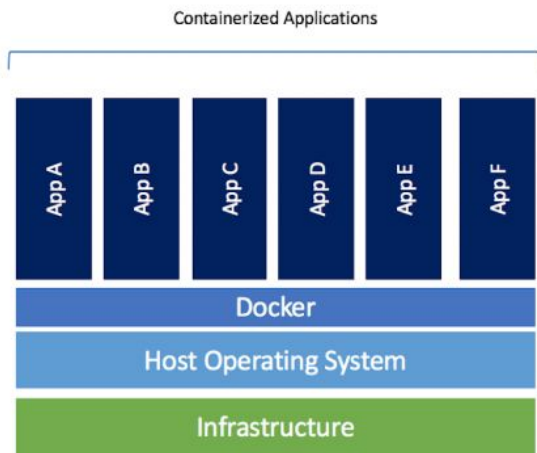
Containers e máquinas virtuais

Por outro lado, as máquinas virtuais (VM) são executadas como sistemas operacionais completos que serão executados no servidor, ao topo do sistema operacional previamente instalado, necessitando muito mais recursos do servidor para serem executados.

Em termos geral, uma aplicação executada em uma VM necessitará muito mais recursos para execução do que o necessário.



Containers e máquinas virtuais



Instalação

Instalar o Docker no Mac e Windows é bastante simples. Sua instalação consiste basicamente em fazer o download do aplicativo de instalação, executá-lo, e seguir os passos do assistente.



Instalação

Para instalar o Docker no Mac faça o download do instalador na seguinte url:
<https://docs.docker.com/docker-for-mac/install/>



Instalação

Para instalar o Docker no Windows faça o download do instalador na seguinte url:

<https://docs.docker.com/docker-for-windows/install/>



Instalação

No Linux o processo é um pouco diferente uma vez que a instalação deverá ser realizada via terminal. Para maiores informações sobre sua instalação, acesse a url correspondente a distribuição que você possui:

CentOS

<https://docs.docker.com/install/linux/docker-ce/centos/>

Debian

<https://docs.docker.com/install/linux/docker-ce/debian/>



Instalação

Fedora

<https://docs.docker.com/install/linux/docker-ce/fedora/>

Ubuntu

<https://docs.docker.com/install/linux/docker-ce/ubuntu/>



Aplicação Spring Boot de exemplo

Para seguirmos o curso e testarmos os comandos um a um, vamos criar uma aplicação bem simples usando Java 8 e Spring Boot.

Como pré-requisito, devemos ter Docker e o Java 8 instalados em nosso computador. O Java pode ser baixado em:

<https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>



Gerando a aplicação Java com Spring Boot


Para gerar a aplicação Java 8 com Spring Boot acesse a url abaixo, preencha os dados conforme a imagem exibida no próximo slide, e clique em “Generate Project” para realizar o download do projeto.

<https://start.spring.io>

** Tenha certeza de ter selecionado os pacotes “WEB” e “ACTUATOR” em “Search dependencies to add”.*



Gerando a aplicação Java com Spring Boot

 **Spring Initializr**
Bootstrap your application

Project

Language

Spring Boot

Project Metadata

Dependencies
[See all](#)

Maven Project **Gradle Project**

Java **Kotlin** **Groovy**

2.2.0 M1	2.2.0 (SNAPSHOT)	2.1.4 (SNAPSHOT)	2.1.3	1.5.19
----------	------------------	------------------	--------------	--------

Group
com.demo

Artifact
docker-demo

More options

Search dependencies to add
Web, Security, JPA, Actuator, Devtools...

Selected dependencies

Web [Web]
Servlet web application with Spring MVC and Tomcat

Actuator [Ops]
Production ready features to help you monitor and manage your application

Generate Project - 🌐 + ↵

© 2013-2019 Pivotal Software
start.spring.io is powered by
[Spring Initializr](#) and [Pivotal Web Services](#)



Executando a aplicação Java com Spring Boot

Para executar a aplicação que acabamos de gerar siga os seguintes passos:

1. Acesse via terminal/console o diretório da aplicação
2. Digite no terminal, na raiz da aplicação:
`./mvnw spring-boot:run`
3. Aguarde a inicialização, e acesse a url <http://localhost:8080/actuator/health>

Essa é uma url padrão gerada pelo pacote “Actuator” que selecionamos, ao acessá-la você deverá ver no navegador a mensagem { “status”: “UP” }’.



Executando a aplicação Java com Spring Boot

Certifique-se de visualizar a mensagem conforme demonstrado no slide anterior antes de prosseguir e testar os comandos do Docker.

Para terminar a execução da aplicação, a qualquer pressiona as teclas Ctrl + C.



Dockerfile

O arquivo Dockerfile é onde adicionamos instruções sobre como o Docker construirá a imagem.

Ele consiste de um arquivo em formato texto com os comandos a serem executadas pelo Docker para construir a imagem.

A listagem de comandos possíveis é bastante extensa, e pode ser visualizada em <https://docs.docker.com/engine/reference/builder/>.



Exemplo de um arquivo Dockerfile para uma aplicação Java com Spring Boot

```
FROM openjdk:8-jre
ENTRYPOINT ["/usr/bin/java", "-jar", "docker-demo.jar"]
ARG JAR_FILE
ADD target/${JAR_FILE} docker-demo.jar
```



Exemplo de um arquivo Dockerfile para uma aplicação Java com Spring Boot

FROM	Indica qual a imagem a ser utilizada como base em nossa aplicação, que em nosso caso é uma imagem padrão contendo o Java 8.
ENTRYPOINT	Comando a ser executado sempre que o container for inicializado. No caso do Spring Boot que possui um arquivo jar executável, ficaria assim: <code>java -jar docker-demo.jar</code>
ARG	Utilizado para passar argumentos externos para o Docker. Adiante veremos como passar esses argumentos com o comando “docker build”.
ADD	Comando de cópia do jar da nossa aplicação para a imagem Docker a ser gerada.



Criando o arquivo Dockerfile em nossa aplicação Java com Spring Boot

Para criar o arquivo de configuração do Docker em nossa aplicação, crie um arquivo chamado “Dockerfile” na raiz da aplicação, com o seguinte conteúdo:

```
FROM openjdk:8-jre
ENTRYPOINT ["/usr/bin/java", "-jar", "docker-demo.jar"]
ARG JAR_FILE
ADD target/${JAR_FILE} docker-demo.jar
```



Comandos Docker para linha de comando

Vamos agora estudar os principais comandos utilizados via linha de comando para gerenciar containers com o Docker.

** Para cada comando estudado teremos um exercício prático, onde você executará os comandos na prática, utilizando como exemplo a aplicação Java com Spring Boot anteriormente.*



docker build

Comando responsável por criar uma imagem a partir de um Dockerfile.

Exemplo:

```
docker build -f Dockerfile -t demo/docker-demo .
```



docker build

-f	Especifica o caminho do arquivo Dockerfile
----	--

docker build

`--tag , -t`

Nomeia e adiciona uma tag a imagem. Pode ser chamado múltiplas vezes para adicionar múltiplas tags, como por exemplo `demo/docker-demo:latest`, `demo/docker-demo:0.0.1-SNAPSHOT`

docker build

`--build-arg`

Criam variáveis que serão passadas para o Docker na construção da imagem, como por exemplo
`JAR_FILE=docker-demo-0.0.1-SNAPSHOT.jar`

docker build

Para maiores informações, acesse a url a seguir:

<https://docs.docker.com/engine/reference/commandline/build/>



docker build - exercício

Para criar a imagem Docker precisamos primeiramente compilar nossa aplicação, gerando o arquivo jar executável.

Para isso execute o seguinte comando na raiz da aplicação:

```
./mvnw clean install
```

Após a execução do comando acima, certifique-se de que o arquivo “docker-demo-0.0.1-SNAPSHOT.jar” tenha sido gerado no diretório “target”, encontrado na raiz da aplicação.



docker build - exercício

Com o arquivo jar gerado, tenha certeza de que o Docker esteja em execução, e digite no terminal o seguinte comando:

```
docker build -f Dockerfile --build-arg JAR_FILE=docker-demo-0.0.1-SNAPSHOT.jar -t demo/docker-demo .
```

Aguarde a execução do comando, que poderá alguns minutos, e então você terá a sua primeira imagem gerada no Docker!



docker images

Comando responsável por listar todas as imagens existentes em sua instalação do Docker.

Exemplo:

`docker images`



docker images

Para maiores informações, acesse a url a seguir:

<https://docs.docker.com/engine/reference/commandline/images/>



docker images - exercício

Com a imagem gerada anteriormente no Docker, digite no terminal o seguinte comando:

`docker images`

Você deverá ver algo similar a:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
demo/docker-demo	latest	eb60a6956674	8 seconds ago	461MB



docker run

Comando responsável por executar uma imagem em um container.

Exemplo:

```
docker run --name docker-demo -d -p 8085:8080 demo/docker-demo:latest
```



docker run

-d	Executa o container em background (não bloqueia o terminal) e exibe no mesmo o seu ID
-p, --publish	Publica o container na porta especificada, onde o primeiro valor é a porta externa, e a segunda (após os dois pontos) é a porta onde a aplicação será executada internamente no container.

docker run

Para maiores informações, acesse a url a seguir:

<https://docs.docker.com/engine/reference/commandline/run/>



docker run - exercício

Vamos agora inicializar a imagem criada anteriormente, para isso digite no terminal:

```
docker run --name docker-demo -d -p 8085:8080 demo/docker-demo
```

Após a inicialização acesse <http://localhost:8085/actuator/health>

Você deverá ver a mensagem { "status": "UP" }.



docker ps

Comando responsável por exibir no terminal os containers que estão em execução.

Exemplo:

`docker ps`



docker ps

Para maiores informações, acesse a url a seguir:

<https://docs.docker.com/engine/reference/commandline/ps/>



docker ps - exercício

Com o container do passo anterior ainda em execução, execute o seguinte comando no terminal:

`docker ps`

Você verá algo similar a:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
b1b9ccc5f37d	demo/docker-demo	"/usr/bin/java -jar ..."	4 minutes ago	Up 3 minutes	0.0.0.0:8085->8080/tcp	docker-demo



docker port

Comando responsável por listar as portas/mapeamento de um container.

Exemplo:

`docker container port docker-demo`

`docker port docker-demo`



docker port

Para maiores informações, acesse a url a seguir:

<https://docs.docker.com/engine/reference/commandline/port/>



docker port - exercício

Com o nosso container docker-demo em execução, execute um dos seguintes comandos (ou os dois em sequência).

```
docker container port docker-demo  
docker port docker-demo
```

Você verá a seguinte mensagem:

```
8080/tcp -> 0.0.0.0:8085
```



docker top

Comando responsável por exibir os processos em execução de um container.

Exemplo:

`docker top docker-demo`



docker top

Para maiores informações, acesse a url a seguir:

<https://docs.docker.com/engine/reference/commandline/top/>



docker top - exercício

Com o nosso container docker-demo em execução, execute o seguinte comando.

```
docker top docker-demo
```

Você verá algo parecido com:

PID	USER	TIME	COMMAND
2351	root	0:16	/usr/bin/java -jar docker-demo.jar



docker stats

Comando responsável por exibir dados em tempo real sobre os containers, como estatísticas de uso de cpu e memória.

Exemplo:

`docker stats`

docker stats

Para maiores informações, acesse a url a seguir:

<https://docs.docker.com/engine/reference/commandline/stats/>



docker stats - exercício

Com o nosso container docker-demo em execução, execute o seguinte comando.

`docker stats`

Você verá algo parecido com:

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
b1b9ccc5f37d	docker-demo	0.37%	237.5MiB / 1.952GiB	11.88%	3.69kB / 2.03kB	1.87MB / 0B	31



docker logs

Comando responsável por exibir os logs de um container.

Exemplo:

```
docker logs -f --tail 10 docker-demo
```



docker logs

<code>-f, --follow</code>	Exibe os logs em modo contínuo no terminal
<code>--tail n</code>	Limita a quantidade de linhas (n) a serem exibidas a partir do fim do arquivo

docker logs

Para maiores informações, acesse a url a seguir:

<https://docs.docker.com/engine/reference/commandline/logs/>



docker logs - exercício

Com o nosso container docker-demo em execução, execute o seguinte comando.

```
docker logs -f --tail 5 docker-demo
```



docker logs - exercício

Você verá algo parecido com:

```
2019-03-26 21:33:52.317 INFO 1 --- [      main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with
context path "/"
2019-03-26 21:33:52.322 INFO 1 --- [      main] c.demo.dockerdemo.DockerDemoApplication : Started DockerDemoApplication in 5.164
seconds (JVM running for 5.829)
2019-03-26 21:34:46.539 INFO 1 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet
'dispatcherServlet'
2019-03-26 21:34:46.539 INFO 1 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet    : Initializing Servlet 'dispatcherServlet'
2019-03-26 21:34:46.552 INFO 1 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet    : Completed initialization in 13 ms
```



docker container ls

Comando responsável por listar os containers existentes.

Exemplo:

docker container ls



docker container ls

Para maiores informações, acesse a url a seguir:

https://docs.docker.com/engine/reference/commandline/container_ls/



docker container ls - exercício

Com o nosso container docker-demo em execução, execute o seguinte comando.

`docker container ls`

Você verá algo parecido com:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
b1b9ccc5f37d	demo/docker-demo	"/usr/bin/java -jar ..."	23 minutes ago	Up 23 minutes	0.0.0.0:8085->8080/tcp	docker-demo



docker stop

Comando responsável por terminar a execução de um container.

Exemplo:

```
docker container stop demo-docker
```

```
docker stop demo-docker
```



docker stop

Para maiores informações, acesse a url a seguir:

<https://docs.docker.com/engine/reference/commandline/stop/>



docker stop - exercício

Com o nosso container docker-demo em execução, execute o seguinte comando.

```
docker stop docker-demo
```

Após a execução do comando anterior, execute o comando a seguir para certificar-se que nenhum container está em execução.

```
docker container ls
```



docker start

Comando responsável por inicializar um container.

Exemplo:

`docker container start docker-demo`

`docker start docker-demo`



docker start

Para maiores informações, acesse a url a seguir:

<https://docs.docker.com/engine/reference/commandline/start/>



docker start - exercício

Execute o comando a seguir para certificar-se de que nenhum container está em execução.

`docker container ls`

Com o nosso container docker-demo em execução, execute o seguinte comando.

`docker start docker-demo`



docker start - exercício

Após a execução do comando anterior, execute o comando a seguir para certificar-se que nenhum container está em execução.

docker container ls

Você verá o seguinte.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
b1b9ccc5f37d	demo/docker-demo	"/usr/bin/java -jar ..."	42 minutes ago	Up 8 seconds	0.0.0.0:8085->8080/tcp	docker-demo



docker rm

Comando responsável por remover definitivamente um container do Docker.

Exemplo:

```
docker container rm -f docker-demo
```

```
docker rm -f docker-demo
```



docker rm

-f, --force	Força a remoção de um container, removendo também suas dependências
-------------	---

docker rm

Para maiores informações, acesse a url a seguir:

<https://docs.docker.com/engine/reference/commandline/rm/>



docker rm - exercício

Execute o comando a seguir para remover o nosso container docker-demo.

```
docker rm -f docker-demo
```

Execute o comando a seguir para certificar-se de que o container foi removido com sucesso.

```
docker container ls
```



docker rmi

Comando responsável por remover uma imagem do Docker.

Exemplo:

```
docker rmi -f ac08d794e567
```



docker rmi

-f, --force	Força a remoção de uma imagem, removendo também suas dependências
-------------	---

docker rmi

Para maiores informações, acesse a url a seguir:

<https://docs.docker.com/engine/reference/commandline/rmi/>



docker rmi - exercício

Execute o seguinte comando para listar as imagens no Docker e obter o ID do container demo/docker-demo.

`docker images`

Digite o comando a seguir substituindo o IMAGE_ID abaixo pelo ID do container listado com o comando acima.

`docker rmi -f IMAGE_ID`



docker rmi - exercício

Execute novamente o comando a seguir e veja que a imagem não é mais listada.

docker images



Deploy no Heroku

Agora que você já aprendeu a criar e gerenciar as imagens com o Docker localmente, chegou a hora de aprender como realizar o deploy de uma imagem em um servidor na nuvem.

Para isso usaremos o [Heroku](#), que utiliza o Amazon AWS como base, mas nos provê uma interface mais amigável e simples de utilizar.



Deploy no Heroku

Vamos utilizar a nossa mesma aplicação Spring Boot criada aqui no curso para isso.

Portanto siga os próximos passos para realizar o deploy, lembrando que o Heroku é gratuito (para os passos que executaremos) e não é necessário nenhum tipo de cartão para realizar o cadastro no mesmo.



Deploy no Heroku

O primeiro passo é se cadastrar no Heroku, portanto acesse a url a seguir, clique em “Sign up” e realize o cadastro.

<http://heroku.com>



Deploy no Heroku

Após a criação da conta devemos instalar o utilitário do Heroku em nosso computador, para isso acesse a url a seguir e faça o download e instalação do aplicativo referente ao seu sistema operacional.

<https://devcenter.heroku.com/articles/heroku-cli#download-and-install>

O processo é bastante simples, basta realizar o download do instalador e o executá-lo localmente.



Deploy no Heroku

Para certificar de que a instalação foi realizada com sucesso, digite no terminal:

```
heroku --version
```

Você deverá ver algo parecido com:

```
heroku/7.22.7 darwin-x64 node-v11.10.1
```



Deploy no Heroku

Chegou a hora de realizar o login em seu computador, para que assim o mesmo reconheça a sua conta criada anteriormente. Para isso digite no terminal:

`heroku login`

Faça o login com suas credenciais para finalizar.



Deploy no Heroku

Como estamos fazendo o deploy de uma imagem Docker, também se faz necessária a execução de outro comando de login, mas agora para o container.

Portanto, execute no terminal:

```
heroku container:login
```



Deploy no Heroku

Agora vamos fazer duas pequenas alterações na configuração do nosso projeto Spring Boot para facilitar o deploy no Heroku.

A primeira delas é adicionar no arquivo `application.properties`, encontrado em `/src/main/resources`, uma configuração para a porta no servidor ser injetada de modo dinâmico.

Isso se faz necessário porque o Heroku alocará dinamicamente uma porta ao inicializar a aplicação, sendo necessário torná-la dinâmica.



Deploy no Heroku

Para configurar a porta de modo dinâmico, adicione no arquivo `application.properties`:

```
server.port=${PORT}
```

O Spring Boot automaticamente obterá o valor de `PORT`, que é uma variável de ambiente criada pelo Heroku no servidor, e inicializará o servidor com o seu valor.



Deploy no Heroku

A segunda modificação é simplificar no arquivo **Dockerfile**, encontrado na raiz da aplicação, ficando assim:

```
FROM openjdk:8-jre
```

```
CMD ["/usr/bin/java", "-jar", "docker-demo.jar"]
```

```
ADD target/docker-demo-0.0.1-SNAPSHOT.jar docker-demo.jar
```

Removemos a definição de **JAR_FILE** para simplificar o processo de build, e substituímos o **ENTRYPOINT** por **CMD**, que embora ambos sejam parecidos, o Heroku depende do **CMD** para funcionar.



Deploy no Heroku

Por ter alterado nosso projeto, precisamos compilar novamente o projeto, para isso digite no terminal, na raiz do projeto:

```
./mvnw clean install
```



Deploy no Heroku

Vamos agora executar os passos para o deploy no Heroku.

O Heroku baseia seu processo de envio de arquivos com base no **Git**, portanto na raiz da aplicação digite o comando a seguir para inicializar o **Git** no projeto:

git init

**Caso você não possua o Git instalado no seu computador, acesse <https://git-scm.com/downloads> para realizar o download do mesmo.*



Deploy no Heroku

Agora devemos criar nosso **Dyno**, que é o termo usado pelo Heroku para representar nosso servidor remoto, e ao mesmo tempo fazer o link dele com nosso projeto.

Na raiz do projeto execute:

```
heroku create
```

Na sequência digite **git remote** para ver o link foi criado, você deverá ver **heroku** no terminal após a execução do comando.



Deploy no Heroku

Agora chegou a hora de gerar a imagem Docker e enviar para o Heroku. Tenha certeza de ter inicializado o seu Docker localmente, a imagem será adicionada a ele.

Para isso digite no terminal, na raiz do projeto:

heroku container:push web

**Esse comando pode demorar vários minutos, portanto aguarde todo o processo ser finalizado.*



Deploy no Heroku

Digite **docker images** no terminal e verifique se uma nova imagem foi adicionada a ela, agora semelhante a:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
registry.heroku.com/serene-forest-91545/web	latest	b60881749f4c	6 minutes ago	461MB



Deploy no Heroku

Para inicializar o container no Heroku, digite o seguinte comando no terminal, na raiz do projeto:

```
heroku container:release web
```

Digite o seguinte para verificar os logs da aplicação e ter certeza de que ela foi inicializada sem problemas:

```
heroku logs -t
```



Deploy no Heroku

Para visualizar a aplicação digite:

`heroku open`

O navegador será aberto com a url da nossa aplicação. Repare que aparecerá uma tela de erro, pois nossa aplicação apenas possui a url `/actuator/health` configurada, portanto adicione esse sub-path na url para visualizar o status do projeto, que deverá ser:

`{"status":"UP"}`



Deploy no Heroku

Pronto, o deploy foi realizado com sucesso!!!

Acesse no navegador o painel de administração do Heroku (login) para visualizar dados da sua conta, Dynos, entre outros.



Usando o Docker para executar uma imagem existente no repositório Docker Hub

Muitas vezes queremos utilizar o Docker somente para executar uma aplicação em específica, como por exemplo o MySQL, Redis, Nginx, MongoDB..., nos poupando assim de instalar essas aplicações em nosso computador.

Para isso existe o Docker Hub (<https://hub.docker.com>), que é o repositório público do Docker, que possui as mais diversas imagens disponíveis e prontas para uso.



Usando o Docker para executar uma imagem existente no repositório Docker Hub

Ao acessar o website do Docker Hub, você pode fazer uma busca pela aplicação que deseja executar no Docker.

Por exemplo, se quisermos instalar o MySQL como container no Docker, basta fazer uma busca no Docker Hub por Mysql, e selecionar uma das opções listadas.



Usando o Docker para executar uma imagem existente no repositório Docker Hub

No caso do MySQL, basta acessar diretamente a url https://hub.docker.com/_/mysql, verificando o comando para realizar o “pull”, ou seja, baixar a imagem para o seu Docker.

No caso do MySQL, o comando a ser executado é o **docker pull mysql**, portanto execute esse comando no terminal e aguarde o download da imagem.



Usando o Docker para executar uma imagem existente no repositório Docker Hub

Repare que na mesma página encontramos toda a documentação necessária para executar e gerenciar a aplicação, que no nosso caso é o MySQL.

Portanto, para executar a imagem, digite a seguinte instrução (conforme descrito na documentação).

```
docker run --name mysql -e MYSQL_ROOT_PASSWORD=root -d -p 3306:3306  
mysql
```



Usando o Docker para executar uma imagem existente no repositório Docker Hub

Repare que definimos a senha para o usuário **root** também como **root**, mantendo todos os outros valores como padrão, como por exemplo a porta 3306, que é a padrão do MySQL.

Seguindo a documentação, se quisermos acessar via terminal o container do MySQL, basta executar o seguinte comando.

```
docker exec -it mysql bash
```



Usando o Docker para executar uma imagem existente no repositório Docker Hub

Então podemos acessar o MySQL com o seguinte comando.

```
mysql -p
```

Digitando a senha **root** no prompt.



Usando o Docker para executar uma imagem existente no repositório Docker Hub

Somente para certificar de que tudo esteja funcionando, liste as tabelas existentes digitando.

`show databases;`

A listagem de tabelas será exibida.

Digite `quit;` para sair do MySQL, e `exit` para finalizar a execução do terminal no container.



Docker compose

O Docker compose é uma ferramenta que permite configurar e executar múltiplos containers em uma mesma aplicação.

Um exemplo muito comum é quando precisamos adicionar um banco de dados em nossa aplicação, não seria mais fácil executar apenas um único comando e ter todo o ambiente configurado e pronto para uso?

É exatamente isso o que o Docker compose nos permite fazer!



Docker compose - instalação

O Docker Compose já vem instalado por padrão quando instalamos o Docker no Mac e Windows, porém se você utiliza Linux terá de fazer a instalação manualmente.

Para maiores detalhes de como proceder com sua instalação no Linux, acesse:

<https://docs.docker.com/compose/install/>



Docker compose - docker-compose.yml

Para configurar o Docker Compose em nossa aplicação, precisaremos criar na raiz do projeto o arquivo **docker-compose.yml**.

Este arquivo conterá toda a configuração dos containers utilizados pela aplicação.

Vamos imaginar uma aplicação em Java e Spring Boot que tenha como dependência o banco de dados MySQL.

O arquivo **docker-compose.yml** ficaria conforme o código a seguir:



Docker compose - docker-compose.yml

version: '3'

services:

mysql:

image: mysql:latest

environment:

- MYSQL_ROOT_PASSWORD=root
- MYSQL_DATABASE=tarefas
- MYSQL_USER=usuario
- MYSQL_PASSWORD=senha

volumes:

- /data/mysql

ports:

- 3306:3306

web:

image: tarefas

build:

context: ./

dockerfile: Dockerfile

depends_on:

- mysql

ports:

- 8080:8080

**Para melhor visualização o arquivo foi dividido em duas colunas, mas todo o código é parte do mesmo arquivo docker-compose.yml*



Docker compose - docker-compose.yml

version	Formato do arquivo docker-compose.yml, no caso versão 3.
services	Define os serviços que fazem parte da aplicação.
"mysql", "web"	Nome dos serviços, utilize nomes que sejam fácil de identificar o serviço.
image	Nome da imagem local/remota que será utilizada no serviço.
environment	Permite definir e passar para a imagem variáveis de ambiente.
volumes	Monta um diretório para ser utilizado para armazenar os dados.
depends_on	Cria uma hierarquia de execução, dizendo qual serviço depende de qual.
ports	Porta onde a aplicação será executada, em nosso caso 8080.
build	Utilizado para informar como a imagem deverá ser criada.



Docker compose - docker compose up

Uma vez configurado o arquivo `docker-compose.yml`, utilizamos o seguinte comando para inicializar todas as dependências:

`docker-compose up -d`

O `-d` é opcional, e tem como função executar os comandos em segundo plano, não bloqueando o terminal.



Docker compose - docker compose down

Para parar a execução dos serviços/containers da aplicação, basta executar o seguinte comando na raiz da aplicação:

`docker-compose down`



Docker compose - exercício

Para ver todos esses conceitos na prática, vamos criar uma aplicação bastante simples utilizando o Spring Boot e MySQL.

Essa aplicação consistirá em uma API RESTful para a criação e gerenciamento de tarefas utilizando o Spring Data REST.



Docker compose - exercício


Para criar a aplicação Spring Boot acesse:

<http://start.spring.io>

Adicione as dependências **JPA**, **MySQL** e **Rest Repositories**, conforme a imagem a seguir:



Docker compose - exercício

 **Spring Initializr**
Bootstrap your application

Project

Language

Spring Boot

Project Metadata

Dependencies
[See all](#)

Maven Project **Gradle Project**

Java **Kotlin** **Groovy**

2.2.0 M1

2.2.0 (SNAPSHOT)

2.1.5 (SNAPSHOT)

2.1.4

1.5.20

Group
com.example

Artifact
tarefas

More options

Search dependencies to add
Web, Security, JPA, Actuator, Devtools...

JPA [SQL]
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate

MySQL [SQL]
MySQL JDBC driver

Rest Repositories [Web]
Exposing Spring Data repositories over REST via Spring Data REST

Generate Project - 📄 + ↵

© 2013-2019 Pivotal Software
starts.spring.io is powered by
[Spring Initializr](#) and [Pivotal Web Services](#)



Docker compose - exercício

Abra o projeto em sua IDE de preferência, ou apenas edite manualmente os arquivos.

Crie um diretório chamado **entidades** dentro de **com/exemplo/tarefas**, e adicione o código ao lado nele.

O código ao lado cria a entidade que representará a tabela no banco de dados do MySQL.

```
package com.exemplo.tarefas.entidades;
```

```
import javax.persistence.Entity;
```

```
import javax.persistence.GeneratedValue;
```

```
import javax.persistence.GenerationType;
```

```
import javax.persistence.Id;
```

```
@Entity
```

```
public class Tarefa {
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.AUTO)
```

```
    private long id;
```

```
    private String nome;
```

```
    //getters e setters aqui
```

```
}
```

Docker compose - exercício

Crie um diretório chamado **repositorios** dentro de **com/exemplo/tarefas**, e adicione o código ao lado nele.

O código ao lado cria o repositório que criará todas as ações de gerenciamento de tarefas de modo automático.

Com o código ao lado já teremos todos os endpoints implementados para uso.

```
package com.exemplo.tarefas.repositorios;
```

```
import com.exemplo.tarefas.entidades.Tarefa;
```

```
import org.springframework.data.repository.PagingAndSortingRepository;
```

```
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
```

```
@RepositoryRestResource(collectionResourceRel = "tarefas", path = "tarefas")
```

```
public interface TarefaRepository extends PagingAndSortingRepository<Tarefa, Long> {  
}
```



Docker compose - exercício

Por fim edite o arquivo `application.properties`, que se encontra em `src/main/resources`, e adicione o código ao lado nele.

O código ao lado configura os dados de conexão com o banco de dados MySQL, que será criado com o Docker Compose a seguir.

```
spring.jpa.hibernate.ddl-auto=create
spring.datasource.url=jdbc:mysql://localhost:3306/tarefas
spring.datasource.username=usuario
spring.datasource.password=senha
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
```

*Se você estiver utilizando Mac, altere a linha da url para:

```
spring.datasource.url=jdbc:mysql://host.docker.internal:3306/tarefas
```

Isso se faz necessário pois é assim que o Docker reconhece o "localhost" no Mac.



Docker compose - exercício

Crie o arquivo Dockerfile na raiz da aplicação, com o seguinte conteúdo:

```
FROM openjdk:8-jre  
ENTRYPOINT ["/usr/bin/java", "-jar", "tarefas.jar"]  
ADD target/tarefas-0.0.1-SNAPSHOT.jar tarefas.jar
```



Docker compose - exercício

Por fim crie o arquivo `docker-compose.yml` na raiz da aplicação, com o seguinte conteúdo:

version: '3'

services:

mysql:

image: mysql:latest

environment:

- MYSQL_ROOT_PASSWORD=root
- MYSQL_DATABASE=tarefas
- MYSQL_USER=usuario
- MYSQL_PASSWORD=senha

volumes:

- /data/mysql

ports:

- 3306:3306

web:

image: tarefas

build:

context: ./

dockerfile: Dockerfile

depends_on:

- mysql

ports:

- 8080:8080

**Para melhor visualização o arquivo foi dividido em duas colunas, mas todo o código é parte do mesmo arquivo `docker-compose.yml`*



Docker compose - exercício

Agora compile a aplicação para gerar o arquivo “.jar” que será utilizado para a criação da imagem Docker.

Para isso execute o seguinte comando no terminal, na raiz da aplicação:

```
./mvnw clean install -DskipTests
```

**Usamos o “skipTests” porque não queremos executar os testes neste momento*



Docker compose - exercício

Agora vamos criar as imagens e executar ambos containers.

Para isso digite no terminal, na raiz do projeto:

```
docker-compose up -d
```

Este processo pode demorar vários minutos, portanto aguarde todo o processo terminar.



Docker compose - exercício

Digite o seguinte comando para visualizar as imagens geradas:

```
docker images
```

Digite o seguinte comando para verificar os containers em execução:

```
docker ps
```

Os nomes poderão variar, mas deverão ser exibidos os containers com os nomes `tarefas_mysql_1` e `tarefas_web_1`.



Docker compose - exercício

Como ambos containers são inicializados ao mesmo tempo, é bastante provável que a aplicação Sprint Boot falhe na inicialização devido ao MySQL não estar pronto.

Caso isso ocorra, execute `docker start tarefas_web_1` para a iniciar novamente.



Docker compose - exercício

Para testar a aplicação, acesse a seguinte URL no navegador:

<http://localhost:8080/tarefas>

Você deverá ver algo parecido com o objeto JSON ao lado:

```
{
  "_embedded": {
    "tarefas": []
  },
  "_links": {
    "self": {
      "href": "http://localhost:8080/tarefas?page,size,sort",
      "templated": true
    },
    "profile": {
      "href": "http://localhost:8080/profile/tarefas"
    }
  },
  "page": {
    "size": 20,
    "totalElements": 0,
    "totalPages": 0,
    "number": 0
  }
}
```



Docker compose - exercício

Para adicionar uma nova tarefa, vamos executar um POST com CURL via terminal (você pode utilizar o POSTMAN se quiser fazer o mesmo com uma interface gráfica):

```
curl -H "Content-type: application/json" -X POST -d '{"nome":"Estudar Docker"}' http://localhost:8080/tarefas
```

Acesse novamente <http://localhost:8080/tarefas> para visualizar a nova tarefa na listagem recém incluída.



Docker compose - exercício

Ao término dos testes você pode parar a execução de ambos containers executando no terminal o seguinte comando:

`docker-compose down`



Docker e Angular

Agora que temos uma aplicação Java com Spring Boot no Docker, porque não aprender também como fazer o mesmo para uma aplicação Angular?

Vamos então aprender a criar uma imagem Docker com o Angular e Nginx (servidor web), já no formato final para ir para produção!



Docker e Angular

O primeiro passo então será criar a nossa aplicação Angular, então tenha certeza de ter instalado em seu computador a uma versão recente do NodeJS (≥ 8), e também o Angular CLI.

O NodeJS pode ser baixado em <https://nodejs.org>, e o Angular CLI pode ser instalado via terminal com o seguinte comando (que pode requerer ser executado como administrador do sistema):

```
npm install -g @angular/cli
```



Docker e Angular

Com o NodeJS e o Angular CLI instalados, crie um projeto Angular com o seguinte comando:

```
ng new ng-docker-demo
```

Ao término da execução do comando acima (que pode demorar alguns minutos), acesse o diretório da aplicação recém criada com:

```
cd ng-docker-demo
```



Docker e Angular

Antes de configurarmos nosso Dockerfile, vamos compilar a aplicação para produção, e garantir que temos o caminho correto dos arquivos compilados para o deploy. Para isso execute na raiz da aplicação:

```
ng build --prod
```

Você deverá ter agora em seu projeto o diretório **dist/ng-docker-demo**, e dentro dele teremos o código fonte compilado.



Docker e Angular

Agora podemos configurar o Docker na aplicação, portanto vamos criar o arquivo **Dockerfile** na raiz da aplicação, com o seguinte conteúdo (o dividirei em duas páginas por ele ser um pouco longo):



Docker e Angular

```
### Primeira parte, build do projeto ###  
# Imagem base com NodeJS utilizada para o build da aplicação  
FROM node:9.6.1 as builder  
# Define o diretório base na imagem  
RUN mkdir /usr/src/app  
WORKDIR /usr/src/app  
# Adiciona `/usr/src/app/node_modules/.bin` ao $PATH para acesso aos executáveis  
ENV PATH /usr/src/app/node_modules/.bin:$PATH  
# Instala as dependências  
COPY package.json /usr/src/app/package.json  
RUN npm install  
RUN npm install -g @angular/cli@1.7.1 --unsafe  
# Adiciona o código fonte ao diretório da aplicação  
COPY . /usr/src/app  
# Executa o build para compilar e gerar os arquivos finais do projeto  
RUN ng build --prod
```



Docker e Angular

Parte 2 - criação da imagem de produção

Imagem base com Nginx

FROM nginx:1.13.9-alpine

Copia os arquivos do projeto gerados no passo anterior para a nova imagem

COPY --from=builder /usr/src/app/dist/ng-docker-demo /usr/share/nginx/html

Expõe a porta 80 na imagem

EXPOSE 80

Inicializa o Nginx

CMD ["nginx", "-g", "daemon off;"]



Docker e Angular

Certifique-se de ter o conteúdo dos dois últimos códigos exibidos juntos no arquivo Dockerfile, localizado na raiz da aplicação.

Agora finalmente poderemos gerar a imagem docker, para isso digite na raiz da aplicação:

```
docker build -f Dockerfile -t ng-docker-demo .
```

Aguarde todo o processo de criação, que poderá demorar alguns minutos.



Docker e Angular

Após a criação da imagem, digite o comando abaixo para verificar se a imagem foi criada corretamente:

`docker images`

Você deverá ver algo similar a:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ng-docker-demo	latest	8776b69740c8	7 minutes ago	18.2MB



Docker e Angular

Agora chegou a hora de criar o container e verificar nossa aplicação, para isso digite no terminal, na raiz da aplicação:

```
docker run --name ng-docker-demo -d -p 4200:80 ng-docker-demo
```

Acesse <http://localhost:4200> para verificar se está tudo funcionando corretamente.

Você deverá ver a tela a seguir:



Docker e Angular

Welcome to ng-docker-demo!



Here are some links to help you start:

- [Tour of Heroes](#)
- [CLI Documentation](#)
- [Angular blog](#)



Docker e Angular

Parabéns, a sua aplicação Angular agora está sendo executada como um container no Docker!

Utilize agora os comandos estudados previamente para verificar logs, verificar o status do container, terminar sua execução, excluí-la.



Comandos úteis para o uso do Docker

docker build

Comando responsável por criar uma imagem a partir de um Dockerfile.

- f** Especifica o caminho do arquivo Dockerfile
- tag, -t** Nomeia e adiciona uma tag a imagem. Pode ser chamado múltiplas vezes para adicionar múltiplas tags, como por exemplo demo/docker-demo:latest, demo/docker-demo:0.0.1-SNAPSHOT

Ex.: `docker build -f Dockerfile -t demo/docker-demo .`

docker images

Comando responsável por listar todas as imagens existentes em sua instalação do Docker.

Ex.: `docker images`

docker port

Comando responsável por listar as portas/mapeamento de um container.

Ex.: `docker container port docker-demo`
`docker port docker-demo`

docker stats

Comando responsável por exibir dados em tempo real sobre os containers, como estatísticas de uso de cpu e memória.

Ex.: `docker stats`

docker start

Comando responsável por inicializar um container.

Ex.: `docker container start docker-demo`
`docker start docker-demo`

docker rm

Comando responsável por remover definitivamente um container do Docker.

- f, --force** Força a remoção de um container, removendo também suas dependências

Ex.: `docker container rm -f docker-demo`
`docker rm -f docker-demo`

docker logs

Comando responsável por exibir os logs de um container.

- f, --follow** Exibe os logs em modo contínuo no terminal
- tail n** Limita a quantidade de linhas (n) a serem exibidas a partir do fim do arquivo

Ex.: `docker logs -f --tail 10 docker-demo`

docker run

Comando responsável por executar uma imagem em um container

- d** Executa o container em background e exibe no mesmo o seu ID
- p, --publish** Publica o container na porta especificada, onde o primeiro valor é a porta externa, e a segunda (após os dois pontos) é a porta onde a aplicação será executada internamente no container.

Ex.: `docker run --name docker-demo -d -p 8085:8080 demo/docker-demo:latest`

docker ps

Comando responsável por exibir no terminal os containers que estão em execução.

Ex.: `docker ps`

docker top

Comando responsável por exibir os processos em execução de um container.

Ex.: `docker top docker-demo`

docker container ls

Comando responsável por listar os containers existentes.

Ex.: `docker container ls`

docker stop

Comando responsável por terminar a execução de um container.

Ex.: `docker container stop demo-docker`
`docker stop demo-docker`

docker rmi

Comando responsável por remover uma imagem do Docker.

- f, --force** Força a remoção de uma imagem, removendo também suas dependências

Ex.: `docker rmi -f ac08d794e567`



Fim