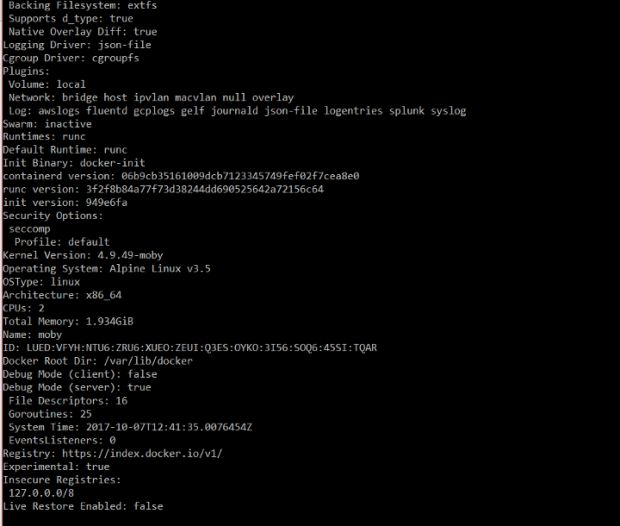
**Comandos básicos de Docker**

Existem diversos comandos e parâmetros que podemos utilizar quando nós estamos trabalhando com containers. vejamos nesse artigo alguns dos comandos considerados principais quando estamos trabalhando com containers.

**Docker INFO**

**Command: docker info**

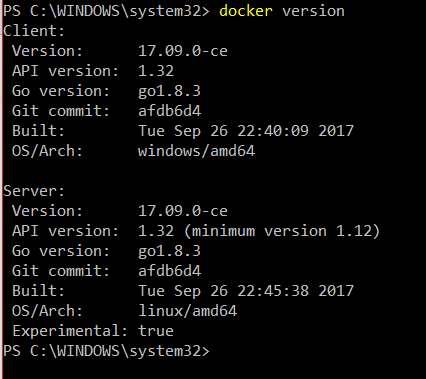
Quando acabamos de subir o Docker Engine, nós utilizamos esse comando para verificarmos as informações do nosso Docker Host.



**Docker VERSION**

**Command: docker version**

Com o version, conseguimos ver a versão do nosso Client. Para esse artigo, estou utilizando o Windows e o Server, que para esse exemplo, estamos utilizando o Linux. Quanto ao OS/Arch do server, nós podemos trabalhar com Windows ou Linux. Caso você esteja utilizando o Docker for Windows, basta clicar em **Switch to Windows Containers** que ele irá realizar essa alteração.



**Docker IMAGES**

**Command: docker images**

Utilizamos ele para listarmos as imagens que nós temos em nosso host. Como eu acabei de instalar o Docker no meu Windows, a lista está vazia, como podemos ver na imagem abaixo:

https://static.imasters.com.br/wp-content/uploads/2017/11/THIN.jpg

**Repository:** repositório;

**TAG:** tag utilizada no repositório;

**IMAGE ID:** o id na nossa imagem;

**Created:** data de quando nós criamos a nossa imagem;

**Size:** tamanho da imagem;

**Docker SEARCH**

**Command: docker search (parâmetro)**

Para procurarmos uma imagem, nós podemos utilizar o comando abaixo com o parâmetro **nome.** Ex.: Ubuntu, dotnetcore, node, etc. Assim ele irá buscar as imagens que são compatíveis com nosso server, que para este exemplo, estamos utilizando o Linux.



**Docker PULL**

Quando encontrarmos a imagem que precisamos para a nossa aplicação, nós precisamos baixar ela para nosso host. Para esse exemplo, nós iremos utilizar uma imagem do Node.js.

**command: docker pull (parametro)**

Docker pull node (baixando imagem do Node.js)

Como essa é a primeira vez que eu estou baixando o node, ele irá demorar um pouco para finalizar esse comando, mas assim que ele baixar a nossa imagem, execute o comando **docker images** para listarmos ela em nosso host.

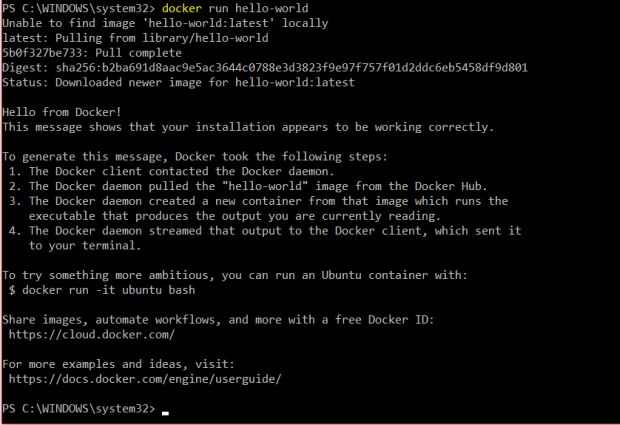
Docker images listando a imagem do nodejs

**Docker RUN**

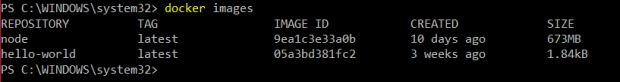
**Command: docker pull (nome da imagem)**

Para criarmos um container, nós precisamos de uma imagem. Caso você não tenha essa imagem em seu host ainda, ele irá até o repositório central e irá baixar ela para seu host, e em seguida, ele criará o container. Vamos criar um exemplo com o famoso Hello World. Para isso, execute o comando abaixo:

**command: docker run hello-world**

Hello World Docker

Notem na imagem acima que o Docker procurou a imagem em nosso host, e como ele não encontrou, ele baixou ela do docker hub. Agora execute o comando docker images novamente no seu terminal e note que temos uma nova imagem.

docker images hello world

**Docker PS**

**Command: docker ps**

Agora vamos verificar o status do nosso container. Execute o comando acima no seu terminal e ele retornará a imagem abaixo:

https://static.imasters.com.br/wp-content/uploads/2017/11/RIG.pngdocker os

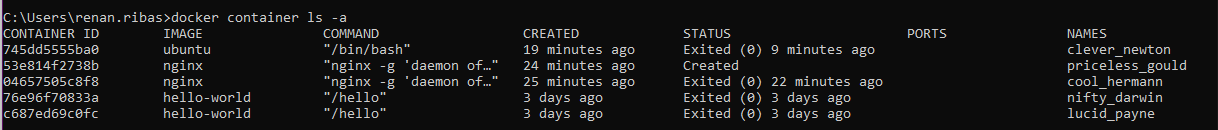
Notem que ele está retornando uma lista vazia. Isso acontece porque os containers nascem e morrem quando executamos o comando **docker run hello-world,** ele realizou todos os processos e morreu. Para vermos ele, podemos executar o comando **docker ps** com o parâmetro**-a.** Dessa forma, teremos uma lista de todos os containers do nosso host.

**Command: docker ps -a**

https://static.imasters.com.br/wp-content/uploads/2017/11/W.jpgdocker ps -a (listando todos os nosso containers)

**Comando alternativo do PS**

**command: docker container ls -a**



**Docker STATUS**

**Command: docker stats (id ou apelido do container)**

https://static.imasters.com.br/wp-content/uploads/2017/11/smalal.jpg

Para termos informações sobre um container, nós executamos o comando acima. Ele nos retorna dados como:

**CONTAINER — ID** do Container

**CPU** **%**— uso de CPU em porcentagem

**MEM USAGE** / LIMIT — Memória usada/Limite que você pode ter setado

**MEM — uso** de memória em porcentagem

**NET** **I/O** — I/O de Internet

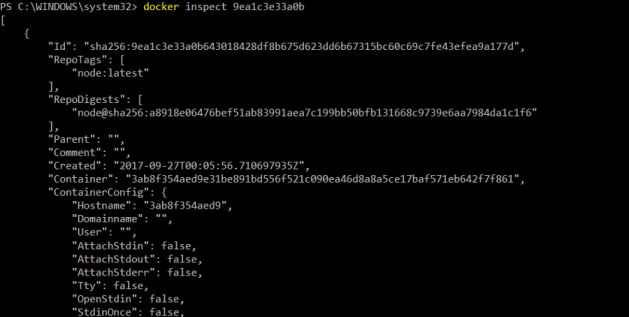
**BLOCK** **IO** — Outros processos de I/O

docker stats (buscando dados sobre o nosso contêiner)

**Docker INSPECT**

Caso precise de mais detalhes sobre a sua imagem ou o seu container, podemos utilizar o comando **inspect.** Ele retornará um json com todas as informações relacionadas a nossa busca. No exemplo abaixo, nós estamos executando ele em nossa imagem node.

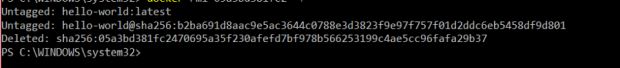
**Command: docker inspect (id da imagem ou container)**

docker inspect (imagem)

**Docker RMI**

**Command: docker rmi (nome da imagem)**

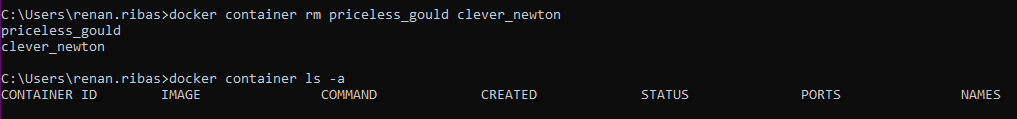
Caso você tenha baixado uma imagem errada ou queira deletar alguma por um outro motivo, basta executar o comando acima que ele deleta ela do seu host. Vamos deletar a nossa imagem que baixamos no exemplo anterior.

docker rmi (deletando uma imagem)

**Docker Container RM**

Caso você tenha criado um container errada ou queira deletar algum por um outro motivo, basta executar o comando abaixo que ele deleta ele do seu host. Vamos deletar dois containers no exemplo anterior.

**command: Docker container rm (nome ou id do container)**



**Docker EXEC**

**Command: docker exec (id\_container ou nome\_container)**

Com o exec, nós podemos executar qualquer comando em nossos containers sem precisarmos estar na console deles.

Vejamos abaixo alguns dos parâmetros que podemos utilizar com ele.

-i permite interagir com o container

-t associa o seu terminal ao terminal do container

-it é apenas uma forma reduzida de escrever -i -t

–name algum-nome permite atribuir um nome ao container em execução

-p 8080:80 mapeia a porta 80 do container para a porta 8080 do host

-d executa o container em background

-v /pasta/host:/pasta/container cria um volume ‘/pasta/container’ dentro do container com o conteúdo da pasta ‘/pasta/host’ do host

Para testarmos ele, iremos baixar uma imagem do Ubuntu. Para isso, execute o comando abaixo no seu terminal:

**Command: docker run -it -d ubuntu /bin/bash**

Execute o comando **docker ps** agora e note que o nosso contêiner está em up.

**Command: docker ps**

https://static.imasters.com.br/wp-content/uploads/2017/11/SUEPR.jpg

Até aqui não temos nenhuma novidade, vamos agora criar um novo diretório dentro do nosso container sem precisarmos entrar nele.

**Command: docker exec 8b54c76e81b7 mkdir /temp/**

Agora vamos criar um arquivo dentro do nosso diretório temp que acabamos de criar dentro do nosso container:

**Command: docker exec 8b54c76e81b7 touch /temp/dotnetsp.txt**

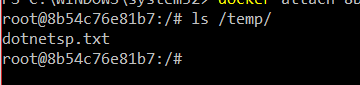
**Docker ATTACH**

**sommand: docker attach (id\_container ou nome\_container)**

Vamos agora entrar dentro de nosso container e verificar o novo arquivo criado. Para isso, execute o comando acima + o nome ou id do seu container. Podemos observar que a execução dele irá nos dar acesso de root em nosso container.

https://static.imasters.com.br/wp-content/uploads/2017/11/1-r4Lv79FXZeIiy0MfDEAOiQ.png

Agora execute o comando ls /temp para verificar nosso arquivo dentro diretório que criamos, chamado temp:



Para sair do container, execute o comando exit.

**command: exit**

**Docker START**

**Command: docker start (id do container)**

Quando saímos do nosso container com o comando exit, ele mata o nosso processo. Notem na imagem a baixo que ele não está mais sendo listado no nosso comando docker ps:

https://static.imasters.com.br/wp-content/uploads/2017/11/UPPP.jpgDocker ps (depois que saímos do nosso contêiner com o comando exit)

**Docker STOP**

Para pararmos um contêiner que esteja sendo executado, utilizamos o comando **stop.** Sua execução é bem simples, passamos os mesmos parâmetros que utilizamos no **start.**

**Command: docker stop (id ou nome container)**

https://static.imasters.com.br/wp-content/uploads/2017/11/looop.jpg

**Mapeamento de Portas**

**-p host:container**

Para testarmos o mapeamento de portas, iremos baixar a imagem do ngnix e passarmos para ele a porta 80. Para isso, execute o comando abaixo em seu terminal:

**Command: docker run -it -p 8080:80 nginx**

Ele irá baixar a imagem do nginx para seu host e abrir ele na porta 8080. Para testar assim que ele finalizar a execução do comando acima, abra no seu navegador o endereço: http://localhost:8080/. Podemos ver seu retorno na imagem abaixo.

**container nginx**

Mas executando ele dessa forma, ficamos com nosso console travado, conforme na imagem abaixo:



Para termos um container sendo executado em background, podemos utilizar eles em segundo plano. Para isso, só precisamos passar o comando -d. Caso seu container ainda esteja sendo executado, execute o comando **control c** para sair dele, e em seguida, execute o mesmo comando anterior, só que dessa vez com o parâmetro -d:

**Command: docker run -it -d -p 8080:80 nginx**

Por fim, execute o comando docker ps e veja ele na lista:

https://static.imasters.com.br/wp-content/uploads/2017/11/RAI.jpg