



# docker

## Sumário

<b>O que é Docker</b> .....	2
<b>Vantagens e desvantagens</b> .....	3
Vantagens em utilizar Docker.....	3
Desvantagens em utilizar Docker.....	3
<b>Arquitetura do Docker</b> .....	4
Docker Image.....	5
Docker compose .....	5
Diferença entre máquina virtuais e Docker .....	6
Uso de Docker na sua empresa.....	6
Como instalar na sua máquina.....	7
Instalar Docker no Linux (Ubuntu) .....	7
Instalar Docker no MacOS.....	8
Instalar Docker no Windows.....	8
Docker CLI: Utilitário de linha de comando .....	8
Criar uma aplicação com Docker.....	11
<b>Develop, Ship e Run</b> .....	11
<b>Materiais para aprender Docker</b> .....	13
<b>O que é Docker</b>	



Em termos simples, o Docker é uma **plataforma de software que simplifica o processo de construção, execução, gerenciamento e distribuição de aplicativos**.

Ele faz isso **virtualizando o sistema operacional do computador** no qual está instalado e sendo executado.

Ele permite que os usuários criem ambientes independentes e isolados para iniciar e implantar seus aplicativos. Esses ambientes são chamados de **contêineres**.

Isso permitirá que o desenvolvedor execute um contêiner em qualquer máquina.

Com o Docker, **não há mais problemas de dependência ou compilação**. Tudo que você precisa fazer é iniciar seu contêiner e seu aplicativo será iniciado imediatamente.

O Docker é um programa open source desenvolvido pela Docker Inc. com a linguagem de programação GO.

Sua primeira edição foi lançada em 13 de março de 2013 e, desde seu lançamento, se tornou um software importantíssimo no mundo do desenvolvimento de tecnologia.

Podemos dizer que as palavras chaves para o **Docker** são: construir, entregar e rodar em qualquer ambiente (***build, ship and run anywhere***).

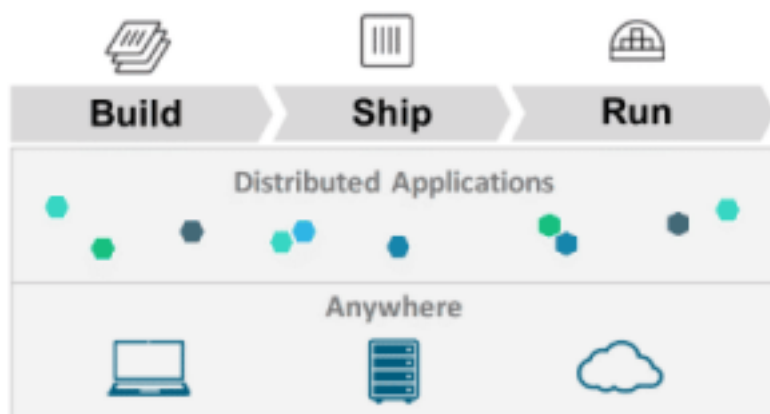


Gráfico que representa a arquitetura de Docker.

## Vantagens e desvantagens

### Vantagens em utilizar Docker

Mas o Docker não é um sistema operacional completo numa máquina virtual. O Docker só compartilhará os recursos da máquina host para executar seus ambientes. É uma ferramenta que pode realmente mudar o dia a dia de um desenvolvedor. Então escrevemos uma lista dos benefícios que você encontrará no software:

- **O Docker é rápido.** Ao contrário de uma máquina virtual, seu aplicativo inicializa em alguns segundos e para com a mesma rapidez.
- **O Docker é multiplataforma.** Você pode iniciar seu contêiner em qualquer sistema. • Os contêineres podem ser construídos e excluídos mais rápido do que em uma máquina virtual.
- **Não há mais dificuldades em configurar seu ambiente de trabalho.** Depois que seu Docker estiver configurado, você nunca mais terá que reinstalar suas dependências manualmente mesmo se mudar seu computador.
- **Você mantém seu espaço de trabalho limpo,** pois cada um de seus ambientes será isolado e você pode excluí-los a qualquer momento, sem impactar o resto.
- **É mais fácil** implantar seu projeto no servidor para colocá-lo online.

### Desvantagens em utilizar Docker

Bom, nenhum software é perfeito e sempre apresenta alguma limitação, então vejamos a seguir algumas das principais críticas de desenvolvedores:

- Há uma tonelada de solicitações de recursos para upgrade que ainda estão em andamento (como capacidade de autorregistro e autoinspeção de contêineres, cópia de arquivos do host para o contêiner e muito mais).
- Há momentos em que um container fica inativo, então depois disso, ele precisa de uma estratégia de backup e recuperação, embora existam várias soluções, mas que não são automatizadas ou nem muito escaláveis ainda.
- Em comparação com as máquinas virtuais, os contêineres Docker oferecem menos

sobrecarga, mas não sobrecarga zero.

- O principal problema é que se um aplicativo projetado para ser executado em um contêiner do Docker no Windows, ele não pode ser executado no Linux ou vice-versa. No entanto, as máquinas virtuais não estão sujeitas a essa limitação.
- Podemos dizer que, para aplicativos que requerem interfaces ricas, o Docker não é uma boa solução.

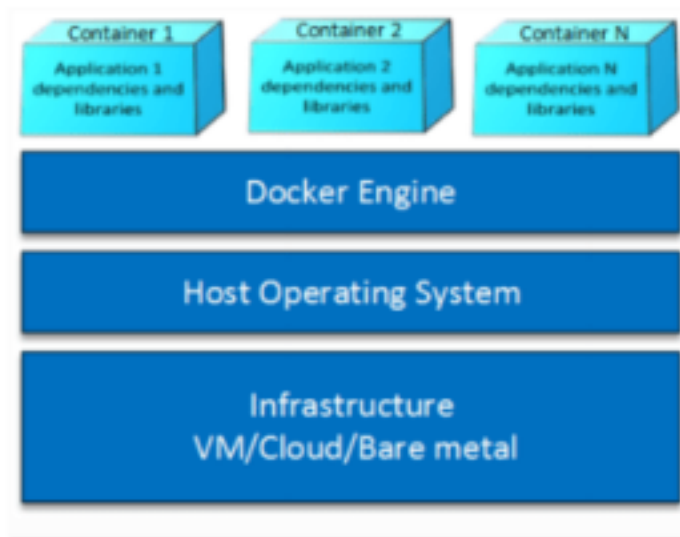
**Resumidamente**, é preciso avaliar os riscos de segurança específicos do Docker para garantir que você poderá lidar com eles antes de mover as cargas de trabalho para o Docker.

## Arquitetura do Docker



Os principais componentes da arquitetura envolvem:

- **Docker** para Mac, Linux e Windows – versões que permitem instalar e executar containers nos sistemas operacionais de forma isolada.
- **Docker Daemon** – Software que roda na máquina onde o Docker está instalado. Usuário não interage diretamente com o daemon.
- **Docker Client** – CLI ou REST API que aceita comandos do usuário e repassa estes comandos ao Docker daemon.
- **Docker Image** – É um template. Uma imagem contém todos os dados e metadados necessários para executar *containers* a partir de uma imagem.
- **Docker Container** – Detém tudo que é necessário para uma aplicação ser executada. Cada *container* é criado a partir de uma imagem. Cada container é uma aplicação isolada independente.
- **Docker Engine** – Usado para criar imagens e containers.
- **Docker Registry** – Uma coleção de imagens hospedadas e rotuladas que juntas permitem a criação do sistema de arquivos de um container. Um registro pode ser público ou privado.
- **Docker Hub** – Este é um registro usado para hospedar e baixar diversas imagens. Pode ser visto como uma plataforma SAAS de compartilhamento e gerenciamento de imagens.
- **Dockerfile** – Um arquivo texto contendo uma syntax simples para criação de novas imagens.
- **Docker Compose** – Usado para definir aplicações usando diversos containers.
- **Docker Swarm** – É uma ferramenta que permite o agrupamento (*clustering*) de Containers Docker.



## Docker Image

Nesta etapa, você escreve um Dockerfile que cria uma imagem do Docker. A imagem contém todas as dependências que o aplicativo Python precisa, incluindo o próprio Python.

Então, no diretório do projeto, crie um arquivo chamado Dockerfile e cole o seguinte:

```
# syntax=docker/dockerfile:1
FROM python:3.7-alpine
WORKDIR /code
ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0
RUN apk add --no-cache gcc musl-dev linux-headers
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
EXPOSE 5000
COPY . .
CMD ["flask", "run"]
```

Isso vai dizer ao Docker para:

- Criar uma imagem começando com a imagem do Python 3.7.
- Definir o diretório de trabalho para /code.
- Definir as variáveis de ambiente usadas pelo comando flask.
- Instalar o gcc e outras dependências
- Copiar o requirements.txt e instalar as dependências do Python.
- Adicionar metadados à imagem para descrever que o contêiner está escutando na porta 5000
- Copiar o diretório atual no projeto para o workdir na imagem.
- Definir o comando padrão para o contêiner para flask run.

## Docker compose

Crie um arquivo chamado docker-compose.yml no diretório do projeto e cole o seguinte:

```
version: "3.9"
services:
  web:
```

```
build: .
ports:
- "5000:5000"
redis:
image: "redis:alpine"
```

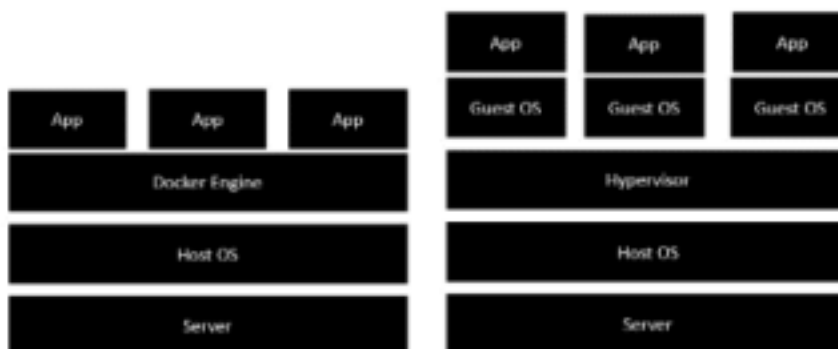
Esse arquivo Compose define dois serviços web e redis.

O serviço da Web usa uma imagem criada a partir do Dockerfile no diretório atual. Em seguida, ele vincula o contêiner e a máquina hospedeira à porta exposta, 5000. Este serviço de exemplo usa a porta padrão para o servidor web Flask, 5000. Já o serviço redis usa uma imagem pública do Redis extraída do registro do Docker Hub.

Por exemplo, se no diretório do projeto você iniciar seu aplicativo executando `docker-compose up`, o Compose extrai uma imagem do Redis, cria uma imagem para seu código e inicia os serviços que você definiu.

Nesse caso, o código é copiado estaticamente na imagem em tempo de compilação. Diferença entre máquinas virtuais e Docker

Se formos compara com a tecnologia de virtualização teremos o seguinte



cenário:

É possível perceber que esta nova arquitetura simplifica bastante o uso de recursos e a sobrecarga do sistema para podermos configurar e distribuir nossas aplicações, independentemente do provedor que estaremos hospedando.

A principal vantagem desta arquitetura é que não é necessário ter hardware extra para o sistema operacional hospedeiro. Tudo será tratado como Container.

### Uso de Docker na sua empresa

Uma regra geral é que, **se você for uma organização que usa máquinas virtuais**, provavelmente se dará melhor com a estratégia de contêineres.



Deve-se observar que seus **desenvolvedores individuais podem se beneficiar do uso do Docker**, mesmo que você decida não adotá-lo em toda a empresa para o fluxo completo de DevOps.

Geralmente, os desafios a seguir indicam que sua organização pode se beneficiar de uma estratégia de contêiner:

- O fluxo atual de DevOps está gerando um alto custo de infraestrutura por conta do uso excessivo de recursos necessários às máquinas virtuais;
- É difícil para a empresa gerenciar a infraestrutura do Sitecore devido às dependências variadas entre as diferentes versões;
- A empresa está perdendo muito tempo configurando ambientes Sitecore;
- Os vários ambientes são implantados em ambientes de hospedagem em nuvem diversos;
- A equipe precisa ser capaz de replicar ambientes específicos rapidamente, como dar suporte para problemas de produção relatados;
- A equipe precisa trabalhar em diferentes sites/soluções/projetos, que também podem usar diferentes versões do Sitecore e exigir diferentes dependências (versões do SQL Server, sistemas operacionais, versões do Solr, etc.);
- Os desenvolvedores mudam de um projeto Sitecore para outro, precisando começar rapidamente e depois desmontá-lo rapidamente;
- A equipe está lutando para encontrar uma maneira de testar isoladamente antes da implantação.

## **Como instalar na sua máquina**

### **Instalar Docker no Linux (Ubuntu)**

Se o seu sistema operacional for linux, Ubuntu, por exemplo, você deve seguir os seguintes passos:

#### **1. Para instalar o Docker Engine, você precisa da versão de 64 bits de uma destas versões do Ubuntu:**

- Ubuntu Impish 21.10
- Ubuntu Hirsute 21.04
- Ubuntu Focal 20.04 (LTS)
- Ubuntu Bionic 18.04 (LTS)

#### **2. Abra o terminal com o atalho Ctrl + Alt + T. Baixe as últimas atualizações do sistema.**

```
$ sudo apt update && sudo apt upgrade
```

#### **3. Instale utilizando o repositório do Ubuntu 18.04**

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

#### **4. Inicie o Docker**

```
$ sudo systemctl start docker
```

#### **5. Entretanto, garanta que ele seja iniciado após a reinicialização**

```
$ sudo systemctl enable docker
```

## 6. Caso queira verificar a versão instalada

**\$ docker -v**

### Instalar Docker no MacOS

Se você estiver utilizando MacOS, certifique-se de estar utilizando MAC OS X Sierra 10.12 ou superior.

1. **Baixe o Docker para Mac;**
2. Dê duplo clique no arquivo DMG, autorize a instalação e informe sua senha de administrador, depois dê um duplo clique em Docker.app para iniciar.

Após baixar, rode em uma janela do terminal os seguintes comandos para verificar se está tudo ok:

**\$ docker --version**

**\$ docker-compose --version**

**\$ docker-machine --version**

### Instalar Docker no Windows

Se você estiver utilizando Windows, siga os seguintes passos:

1. Verifique se o suporte a virtualização está habilitado em seu computador;
2. Seu Windows deve ser 64bits versão 1607 e build: 14393.0;
3. Você deve habilitar o recurso do hyper-v;

### Baixe o Docker para Windows.

Docker CLI: Utilitário de linha de comando

Depois de entender a arquitetura e realizar a instalação, é chegada a hora de conhecer o utilitário de linha de comando. Utilizar o docker consiste em informar a ele opções e argumentos da seguinte forma:

**\$ docker [option] [command] [arguments]**

Para exibir as informações globais de sistema sobre o Docker, digite:

**\$ docker info**

Caso deseje ver todos os subcomandos do utilitário digite no terminal:

**\$ docker**

**attach** Attach to a running container

**build** Build an image from a Dockerfile

**commit** Create a new image from a container's changes

**cp** Copy files/folders between a container and the local filesystem

**create** Create a new container

**diff** Inspect changes on a container's filesystem

**events** Get real time events from the server

**exec** Run a command in a running container



**export** Export a container's filesystem as a tar archive  
**history** Show the history of an image  
**images** List images  
**import** Import the contents from a tarball to create a filesystem image  
**info** Display system-wide information  
**inspect** Return low-level information on a container or image  
**kill** Kill a running container  
**load** Load an image from a tar archive or STDIN  
**login** Log in to a Docker registry  
**logout** Log out from a Docker registry  
**logs** Fetch the logs of a container  
**network** Manage Docker networks  
**pause** Pause all processes within a container  
**port** List port mappings or a specific mapping for the CONTAINER  
**ps** List containers  
**pull** Pull an image or a repository from a registry  
**push** Push an image or a repository to a registry  
**rename** Rename a container  
**restart** Restart a container  
**rm** Remove one or more containers  
**rmi** Remove one or more images  
**run** Run a command in a new container  
**save** Save one or more images to a tar archive  
**search** Search the Docker Hub for images  
**start** Start one or more stopped containers  
**stats** Display a live stream of container(s) resource usage statistics  
**stop** Stop a running container  
**tag** Tag an image into a repository  
**top** Display the running processes of a container  
**unpause** Unpause all processes within a container  
**update** Update configuration of one or more containers  
**version** Show the Docker version information  
**volume** Manage Docker volumes  
**wait** Block until a container stops, then print its exit code

Para conhecer as opções adicionais de um comando específico de seu interesse,

digite: **\$ docker subcomando --help**

Agora vamos rodar um container da imagem do Ubuntu, para aquecer os motores e iniciar nossa jornada nesse novo mundo. Em uma janela do terminal ou *prompt* de comando, digite:

**\$ docker run -t -i ubuntu /bin/bash**

A saída resultante do comando acima, deve indicar que o Docker está executando corretamente a imagem do Ubuntu, em modo interativo, note que executei um comando `top` para listar os processos que estão executando internamente ao container:

```
Selecionar root@56cb0400dea7: /
Microsoft Windows [versão 10.0.17134.765]
(c) 2018 Microsoft Corporation. Todos os direitos reservados.

C:\Users\...>docker run -t -i ubuntu /bin/bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
5abc03819f3e: Pull complete
05731e63f211: Pull complete
0bd67c50d6be: Pull complete
Digest: sha256:f08638ec7ddc90065187e7eabdfac3c96e5ff0f6b2f1762cf31a4f49b53000a5
Status: Downloaded newer image for ubuntu:latest
root@56cb0400dea7:/# ls
bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys usr var
root@56cb0400dea7:/#
```

Neste ponto você já pode executar qualquer comando dentro do container, não é preciso utilizar usuário sudo, porque você já está operando com privilégio de super usuário (root).

Teste, executando os comandos:

**\$ apt-get update**

...e depois instale o **node.js** para fazer um breve teste:

**\$ apt-get install -y nodejs**

Repare que utilizamos o comando run do utilitário e as opções -i -t:

```
Selecionar root@56cb0400dea7: /
top - 13:30:07 up 19:39, 0 users, load average: 0.20, 0.22, 0.24
Tasks: 2 total, 1 running, 1 sleeping, 0 stopped, 0 zombie
%Cpu(s): 2.3 us, 1.0 sy, 0.0 ni, 91.3 id, 4.7 wa, 0.0 hi, 0.7 si, 0.0 st
KiB Mem : 2027844 total, 120704 free, 1072404 used, 834736 buff/cache
KiB Swap: 1048572 total, 1043668 free, 4904 used, 797840 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM    TIME+  COMMAND
    1 root        20   0   18504   3448   3040  S   0.0   0.2   0:00.06  bash
   11 root        20   0   36620   3104   2620  R   0.0   0.2   0:00.00  top
```

Em outro terminal rode o comando abaixo para listar os Containers:

**\$ docker ps**

A saída resultante será:

```
Selecionar Prompt de Comando
Microsoft Windows [versão 10.0.17134.765]
(c) 2018 Microsoft Corporation. Todos os direitos reservados.

C:\Users\...>docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
56cb0400dea7       ubuntu             "/bin/bash"        12 minutes ago     Up 12 minutes      0.0.0.0:22->22      objective_hamilton
```

Para exibir o último containers criado, informe ao comando a chave -l:

**\$ docker ps -l**

Se quiser exibir todos os containers – ativos e inativos, passe ao comando a chave

**-a: \$ docker ps -a**

Para exibir um container executado ou ativo é só digitar:

**\$ docker stop id-do-container**

O container-id pode ser encontrado na saída do comando docker ps.

Se após realizar uma série de configurações de uma imagem docker você quiser disponibilizá-la ou até mesmo hospedá-la no **Docker Hub** para uso futuro, você pode.

Basta criar uma conta no Docker Hub e, depois de efetuar login, faça o carregamento da imagem para o repositório.

**\$ docker login -u username-docker**

Após efetuar o login no Docker Hub você poderá então enviar a imagem que desejar, utilizando:

**S docker push username-docker/nome-da-imagem**

Após finalizar o upload da imagem, você poderá acessar no endereço do Docker Hub.

Criar uma aplicação com Docker

Suponha que você queira criar uma aplicação que possua todas as dependências do Node.js já instalados, sem instalar as dependências manualmente ou diretamente em sua máquina.

Usando Docker você pode fazer isso de maneira isolada com alguns comandos declarados no arquivo de configuração. Isso se chama “Dockerfile”.

## **Develop, Ship e Run**

Veja um exemplo de conteúdo de um Dockerfile:

```
FROM node:10-alpine
RUN mkdir -p /home/node/app/node_modules && chown -R node:node
/home/node/app WORKDIR /home/node/app
# Install app dependencies
COPY package*.json ./
USER node
RUN npm install
COPY --chown=node:node . .
EXPOSE 8080
CMD [ "node", "app.js" ]
```

Se olharmos para o arquivo app.js que define uma aplicação básica em Node.js e express, nós teremos:

```
const express = require('express');
const app = express();
app.get('/', function (req, res) {
res.send('Geek Hunter!');
```

```
});  
app.listen(3000, function () {  
  console.log('Servidor Geek Hunter rodando na porta 3000!');  
});
```

Agora que já definimos nossa aplicação em app.js e já temos nosso Dockerfile, vamos criar uma imagem para poder rodar nossa aplicação e realizar o *deploy* na imagem que será carregada no *container*.

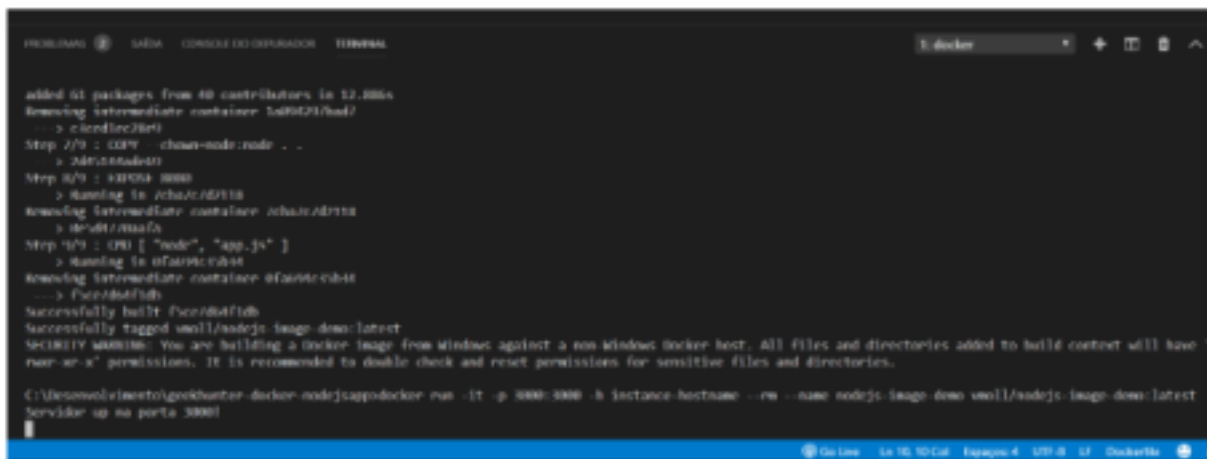
Vamos criar uma imagem executando o comando:

```
$ docker build -t vmoll/nodejs-image-demo .
```

A opção -t serve para informarmos uma tag para a imagem que estamos criando.

Após a criação será possível rodar a aplicação utilizando o seguinte comando:

```
$ docker run -it -p 3000:3000 -h instance-hostname --rm --name nodejs-image-demo  
vmoll/nodejs-image-demo:latest
```

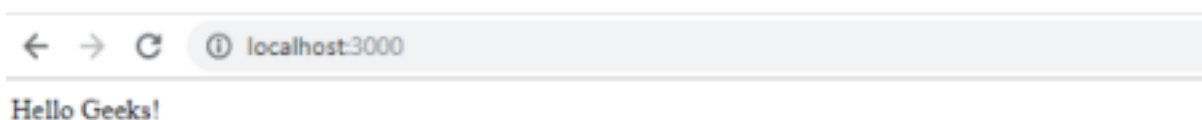


```
added 64 packages from 48 contributors in 12.881s  
building intermediate container 1a09291bad7  
-> e7e0d1ec28e9  
Step 2/9 : COPY --chown=node:node . .  
-> 3d61a1a4e011  
Step 3/9 : CMD ["node", "app.js"]  
-> 1a09291bad7  
building intermediate container 1a09291bad7  
-> e7e0d1ec28e9  
Step 4/9 : CMD ["node", "app.js"]  
-> 1a09291bad7  
building intermediate container 1a09291bad7  
-> e7e0d1ec28e9  
Successfully built 1a09291bad7  
Successfully tagged vmoll/nodejs-image-demo:latest  
SECURITY WARNING: You are building a Docker image from Windows against a non-Windows Docker host. All files and directories added to build context will have 'root' permissions. It is recommended to double check and reset permissions for sensitive files and directories.  
C:\Users\leandro\geekhunter> docker run -it -p 3000:3000 -h instance-hostname --rm --name nodejs-image-demo vmoll/nodejs-image-demo:latest  
Servidor up na porta 3000!
```

No comando que acabamos de executar informamos -p . 3000:3000.

Esse comando é responsável por realizar o bind ou vinculação da porta local para uma porta externa do serviço, que será disponibilizado na máquina que está hospedando a aplicação.

Veja como a aplicação está funcional e operante, acessando <http://localhost:3000>, e receberá a mensagem “Hello Geeks!”:



Todo o **código da aplicação de exemplo**, juntamente com seu Dockerfile, estão disponíveis na minha página do Github, linkada acima.

**Observação importante:** criei aqui uma aplicação bem simples, apenas para demonstrar que é possível realizar o deploy na imagem que criamos. Nosso foco principal é a criação e execução de containers usando Docker.

Caso queira criar uma aplicação completa, você pode utilizar algum gerador como o express generator. Para isso execute os seguintes comandos:

```
$ npm install express-generator -g
$ express minhaAplicacao
$ cd minhaAplicacao
$ npm install
$ npm start
```

Após isso abra uma janela do browser em: <http://localhost:3000>

Como última dica, gostaria de deixar aqui um link onde existem diversos exemplos de aplicações e recomendações que podem ser estudados e seguidos para uma melhor utilização. Acesse aqui:

## >> **Projetos Docker**

Galera, espero ter ajudado vocês a compreenderem de maneira simples os benefícios da utilização do Docker em nossas aplicações.

Creio que, se você ainda não está fazendo uso dessa tecnologia, logo passará a utilizá-la em seus novos projetos.

## **Materiais para aprender Docker**

Com o ecossistema Docker evoluindo e mudando a uma velocidade enorme, muitos tutoriais, livros e outros materiais instrucionais tendem a ficar desatualizados rapidamente.

Mas saiba que aprender Docker é rápido, em geral leva algumas horas apenas e se manter atualizado(a), como sabemos, é um trabalho do dia a dia para todo dev.

Então vamos dar uma olhada em materiais atuais e alguns que continuam sendo atualizados constantemente:

## **Links úteis e referências:**

1. **Como instalar**
2. **Como instalar o Docker Compose**
3. **Docker Hub**
4. **Documentação**
5. **Exemplos de aplicações**