



**CENTRO UNIVERSITARIO IESB  
REDES DE COMPUTADORES**

Mateus Silva da Fonseca 16214080014

Professor: Eustáquio

Atividade  
Docker e Linux

Brasília-DF 2018

# Docker

## O que é Docker?

Docker é uma plataforma Open Source escrito em Go, que é uma linguagem de programação de alto desempenho desenvolvida dentro do Google, que facilita a criação e administração de ambientes isolados.

Como funciona?

O Docker possibilita o empacotamento de uma aplicação ou ambiente inteiro dentro de um container, e a partir desse momento o ambiente inteiro torna-se portátil para qualquer outro Host que contenha o Docker instalado. Isso reduz drasticamente o tempo de **deploy** de alguma infraestrutura ou até mesmo aplicação, pois não há necessidade de ajustes de ambiente para o correto funcionamento do serviço, o ambiente é sempre o mesmo, configure-o uma vez e replique-o quantas vezes quiser. Outra facilidade do Docker é poder criar suas imagens (containers prontos para deploy) a partir de arquivos de definição chamados Dockerfiles.

***Deploy**, podemos dizer que é a instalação da sua aplicação em um servidor de aplicações, ou seja, instalar a aplicação para disponibilizar ela para seus usuários.*

## Como criar redes no Docker? – Exemplos

O comando para gerenciar rede no Docker é “**docker network**” seguido dos parâmetros:

- **connect** – Conecta uma rede a um container
- **create** – Cria uma nova rede
- **disconnect** – Desconecta um container de uma rede
- **inspect** – Exibe informações detalhadas sobre uma ou mais redes
- **ls** – Lista as redes
- **prune** – Remove todas as redes não usadas
- **rm** – Remove uma ou mais redes

Existem vários tipos de redes:

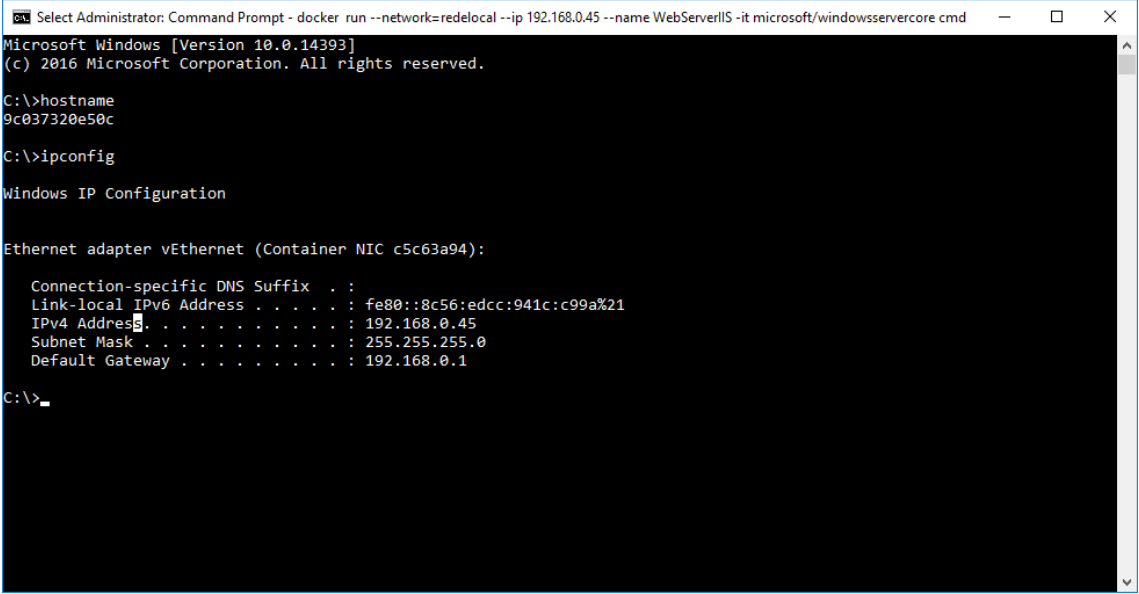
- **NAT** – Ip dinâmico alocado pelo Host Networking Service (HNS) e usa uma subnet Interna
- **Transparent** – Endereço IP estático ou dinâmico (usando um DHCP existente externo)
- **Overlay** – Endereço IP dinâmico alocado pelo engine do Swarm
- **L2 Bridge** – Endereço IP estático de um conjunto de IPs do host de container ou através do HNS Plugin)
- **L2 Tunnel** – Apenas para Azure – Endereço dinâmico

Exemplo para definir uma rede transparente chamada redelocal e conectar um container a essa rede

- `docker network create -d transparent --subnet=192.168.0.0/24 --gateway=192.168.0.1 redelocal`

Podemos então executar um container usando o endereço IP estático:

- `docker run --network=redelocal --ip 192.168.0.45 -it microsoft/windowsservercore cmd`



```

Select Administrator: Command Prompt - docker run --network=redelocal --ip 192.168.0.45 --name WebServerIIS -it microsoft/windowsservercore cmd
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\>hostname
9c037320e50c

C:\>ipconfig

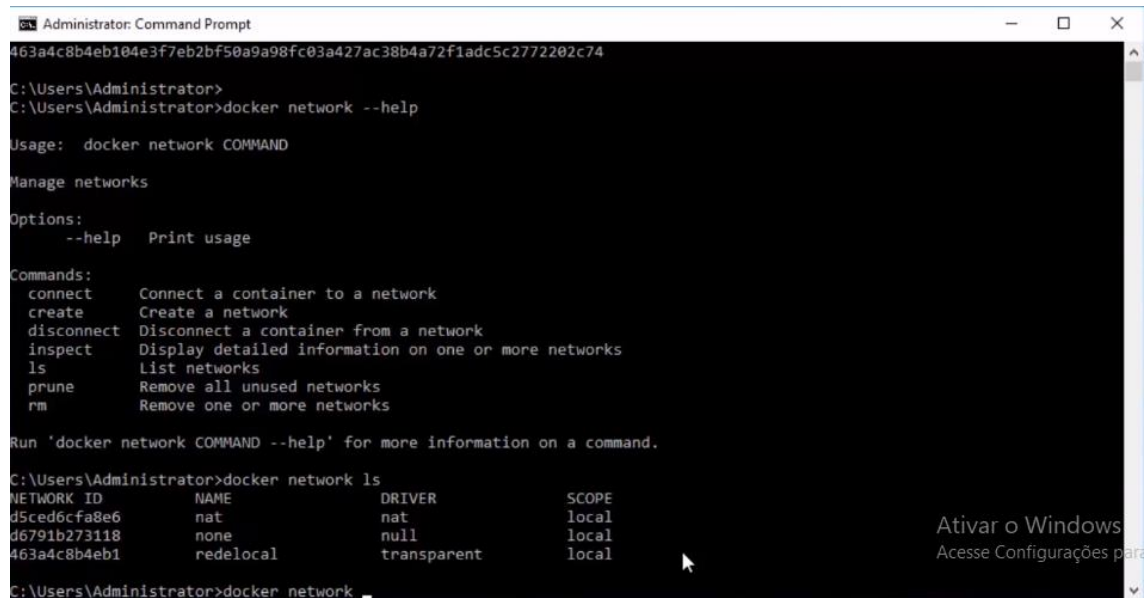
Windows IP Configuration

Ethernet adapter vEthernet (Container NIC c5c63a94):

    Connection-specific DNS Suffix  . : 
    Link-local IPv6 Address . . . . . : fe80::8c56:edcc:941c:c99a%21
    IPv4 Address. . . . . : 192.168.0.45
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.0.1

C:\>_
  
```

Após a criação, utilizarei o comando `docker network ls` para listar as redes que estão atribuídas no Container...



```
Administrator: Command Prompt
463a4c8b4eb104e3f7eb2bf50a9a98fc03a427ac38b4a72f1adc5c2772202c74
C:\Users\Administrator>
C:\Users\Administrator>docker network --help

Usage:  docker network COMMAND

Manage networks

Options:
  --help    Print usage

Commands:
  connect   Connect a container to a network
  create    Create a network
  disconnect Disconnect a container from a network
  inspect   Display detailed information on one or more networks
  ls        List networks
  prune     Remove all unused networks
  rm        Remove one or more networks

Run 'docker network COMMAND --help' for more information on a command.

C:\Users\Administrator>docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
d5ced6cfa8e6        nat                 nat                 local
d6791b273118        none               null               local
463a4c8b4eb1        redelocal          transparent         local
C:\Users\Administrator>docker network
```

Nesse Container, contam 3 networks que são: nat, none e redelocal, que foi criada com o exemplo acima.

## Como criar Volumes no Docker? - Exemplos

Um volume nada mais é do que um diretório compartilhado entre o host e o container ou entre um ou mais containers, para este volume não há as mesmas restrições ou opções de limitação do LXC, pois não há manipulação dele pela libcontainer (biblioteca responsável pelas filtragens nas chamadas de sistema).

Existem basicamente duas formas de se montar um volume no Docker:

1. Montagem host-container
2. Montagem entre containers

### 1. Montagem Host-Container

#### Prós:

- Menor overhead para acesso disco, pois não há análise pela libcontainer das chamadas para esse volume mapeado;
- É possível mapear o mesmo volume para múltiplos containers, forma primitiva de replicação de dados entre containers (lembrando que neste caso não há controle de arquivos lidos e escritos ao mesmo tempo, quem deve ser responsável por isso é o filesystem do volume, não o volume em si);
- Persistência de dados, não há risco do container cair ou ser removido e os dados desaparecerem;

### **Contra:**

- Sucessíveis possíveis brechas de segurança a nível de Filesystem, escalonamento de privilégios, pois neste ambiente não há controle pela Libcontainer;
- Não escalável, pois atrela-se o container ao host ou pelo menos ao mapeamento;
- Dificulta administração, pois há mais uma camada de gerenciamento que deve ser muito bem estruturada;

## **2. Montagem entre Containers;**

### **Prós:**

- Portável, pois o volume não está mais atrelado ao host;
- É possível mapear o mesmo volume para múltiplos containers, forma primitiva de replicação de dados entre containers (lembrando que neste caso não há controle de arquivos lidos e escritos ao mesmo tempo, quem deve ser responsável por isso é o filesystem do volume, não o volume em si);
- Persistência de dados, não há risco do container cair ou ser removido e os dados desaparecerem;

### **Contras:**

- Maior overhead para acesso disco, pois há análise pela libcontainer das chamadas para esse volume mapeado;
- Maior complexidade, pois há mais passos a serem seguidos para criação e remoção de containers;

## **1. ADICIONANDO UM VOLUME DE DADOS**

Para mapear o volume do container a uma pasta no host basta executar o comando:

```
# docker run -d --name meucontainer -v /var/app imagem
```

Será criado um novo diretório dentro do container chamado /var/app, caso o mesmo já exista, será sobrescrito por este novo.

## 2. VERIFICAR QUAIS VOLUMES UM CONTAINER POSSUI

Você pode verificar quais volumes um container possui executando o comando:

```
# docker inspect meucontainer
```

O resultado do comando será parecido com isso:

```
...
Mounts": [
  {
    "Name": "fac362...80535",
    "Source": "/var/lib/docker/volumes/fac362...80535/_data",
    "Destination": "/var/app",
    "Driver": "local",
    "Mode": "",
    "RW": true
  }
]
...
```

Por padrão todos os volumes são montados como leitura e escrita, você pode montar o volume como apenas leitura, basta executar:

```
# docker run -d --name meucontainer -v /var/app:ro imagem
```

Note o ":ro" no final do /var/app, isso quer dizer que este volume é apenas leitura.

## 3. MONTANDO UM DIRETÓRIO DO HOST COMO UM VOLUME DE DADOS

Para mapear o volume do container a uma pasta no host basta executar o comando:

```
# docker run -d --name meucontainer -v
```

```
/var/app:/var/www/html imagem
```

Será criado um novo diretório dentro do container chamado /var/app, caso o mesmo já exista, será sobrescrito por este novo.

#### 4. CRIANDO E MONTANDO UM CONTAINER COMO UM VOLUME DE DADOS

É possível você criar um container com um volume que pode ser compartilhado com demais containers, isso é uma prática comum quando deseja-se ter alguma persistência de dados e que não esteja atrelado ao host onde o container está. Para isso, crie o container conforme abaixo:

```
# docker create -v /dados --name meucontainer imagem  
/bin/true
```

Agora, no próximo container basta você utilizar o parâmetro `--volumes-from` para que o volume criado no container acima seja mapeado para o novo container, algo como isso:

```
# docker run -d --volumes-from dados--name meucontainer1  
imagem
```

Você pode criar quantos containers desejar mapeando este volume, inclusive pode mapear o volume de um container, por exemplo:

```
# docker run -d --volumes-from container1 --name  
meucontainer2 imagem
```

Com isso não é necessário mapear obrigatoriamente o container de volume, e sim algum container que já utiliza ele. Outro ponto importante a se observar nesse caso é que:

- Quando removido o container de volume os demais containers não perdem os dados, pois continuará existindo neste container a pasta montada;
- Para remover o volume nesse caso é necessário remover todos os containers que foram criando mapeando este container e em seguida executar o comando: **docker em -v**

#### Como funciona o DockerFile? - Exemplos

Dockerfile é um arquivo onde colocamos tudo o que precisamos para nosso

container. De qual container ele se origina, o que você quer instalar, que serviços quer rodar.

## Preparação

Crie uma pasta chamada *docker-test*, e dentro dela crie um arquivo chamado *Dockerfile*.

## Container de origem

Nesse Exemplo, partiremos para usar o Container do Debian.

Para dizer isso para o Dockerfile, usamos **FROM**

```
FROM debian
```

Com essa diretiva, estamos partindo da última imagem do Debian presente no DockerHub.

## Assine sua imagem

Use MAINTAINER para assinar sua imagem, dizer aos outros quem é o mantenedor dela.

```
MAINTAINER Mateus
```

## Atualização e instalações

Passamos algum tempo atualizando o sistema e em seguida instalando o NGinx e o PHP-FPM. Fizemos isso de fora do container usando o comando *docker exec*.

No Dockerfile usamos RUN para criar o ambiente da nossa imagem

```
RUN apt-get update  
RUN apt-get install -y nginx  
RUN apt-get install -y php5-fpm
```



Esses 3 comandos indicam respectivamente que queremos atualizar o sistema operacional e em seguida instalar o NGinx e o php5-fpm.

## Configurações

Para configurar nosso container, entramos nele com o comando *docker attach* e criamos e alteramos o arquivo */etc/nginx/sites-available/default*. Fizemos um arquivo bem enxuto funcionando na porta *80* e apontando para */var/www/app*.

Copie o trabalho que fizemos com o arquivo default dentro do container para um arquivo local, dentro de nossa pasta *docker-test*, chame ele de *default*. Agora vamos dizer para o Dockerfile que queremos copiar esse arquivo para a configuração do Nginx.

```
COPY default /etc/nginx/sites-available/default
```

O primeiro parâmetro é o arquivo em nossa máquina, e o segundo é o lugar que ele vai parar em nossa imagem. Os passos de criar a pasta *app*, não serão necessários. Você vai ver como o container vai se virar com isso com bastante praticidade.

## Fechando sua imagem

Na parte 2 do artigo anterior criamos a imagem a partir do container que estávamos rodando. Infelizmente tínhamos de iniciar o NGinx e o PHP-FPM manualmente após fazer download da imagem para fazer as coisas acontecerem.

Aqui no Dockerfile vamos resolver esse problema.

Uma vez que nosso ambiente foi preparado com os comandos anteriores, agora vamos usar *CMD* para dizer como nosso container vai rodar.

```
CMD service nginx start && service php5-fpm start && /bin/bash
```

Repare que diferente da diretiva *RUN*, com *CMD* concatenamos os comandos usando *&&*. Precisamos disso pois só podemos ter um *CMD* no Dockerfile. Aqui respectivamente iniciamos o Nginx, em seguida o PHP-FPM e executamos */bin/bash* para que o container fique de pé quando iniciado. Caso a gente não use esse */bin/bash* aqui, teremos que passar ele quando rodarmos a imagem com *docker run*.

## Nosso Dockerfile

Já temos tudo o que precisamos no nosso *Dockerfile*. Vejam como ficou:

```
FROM debian

MAINTAINER Mateus

RUN apt-get update
RUN apt-get install -y nginx
RUN apt-get install -y php5-fpm

COPY default /etc/nginx/sites-available/default

CMD service nginx start && service php5-fpm start && /bin/bash
```

## Construindo a imagem

Para construir a imagem a partir do Dockerfile vamos executar o comando *docker build*.

```
docker build -t Mateus/php-nginx-2 .
```

Usamos o parâmetro `-t` para dar a nossa imagem um apelido, uma tag para que fique mais fácil levantar ela depois. Assim como no artigo anterior, coloque seu nome de usuário do DockerHub na frente da tag da imagem. Isso será necessário na hora do envio da imagem. No meu caso ficou *Mateus/php-nginx-2*.

O ponto no final, indica o diretório aonde está o Dockerfile que você criou. Você verá o docker executar passo a passo do seu Dockerfile em detalhes, acompanhe no seu terminal.

## Checando sua imagem e dando run

Com o comando *docker images* você vê sua imagem criada localmente. Thiago

```
docker run --name usando-dockerfile -itd -p 8080:80 -v ~/dev/php:/var/www/app Mateus/php-nginx-2
```

Coloquei um nome para meu container de *usando-dockerfile*.

Passamos a referência de portas, a 8080 de nossa máquina vai bater na 80 do container.

Compartilhei uma pasta local minha *~/dev/php* para a pasta */var/www/app* do container. Tudo que for feito na pasta local, vai refletir na pasta dentro do container. */var/www/app* é o local para onde nosso config do NGinx aponta como diretório raiz.

Faça os testes com seus arquivos PHP e veja em funcionamento.

Caso queira, acesse o container e confirme que seus serviços estão rodando e que o arquivo de configuração do NGinx foi copiado.

## Enviando ao Dockerhub

Para enviar a imagem para o DockerHub, vamos fazer o mesmo processo feito anteriormente.

```
docker login
```

Caso não tenha usuário no DockerHub, não esqueça de criar antes.

## E para enviar a imagem

```
docker push Mateus/php-nginx-2
```

Use com a tag que deu para sua imagem, com seu nome de usuário.  
É isso! Acesse o DockerHub para conferir.

## Testando

Seguindo o mesmo que fizemos no final do último artigo vamos testar.  
Antes de testar, vamos eliminar tudo que temos localmente, parar o container e remover a imagem.

```
docker stop usando-dockerfile
docker rm usando-dockerfile
docker rmi Thiago/php-nginx-2
```

Agora executamos o mesmo run que fizemos acima.

```
docker run --name usando-dockerfile -itd -p 8080:80 -v ~/dev/php:/var/www/app
Mateus/php-nginx-2
```

A imagem não será encontrada localmente e será feito download dela diretamente do DockerHub.

Veja seu container rodando:

```
docker ps
```

Viram como foi muito mais fácil usar o Dockerfile para criar nosso container...

## Como funciona o Docker Compose? – Exemplos

O Docker Compose é uma ferramenta para a criação e execução de múltiplos containers de aplicação. Com o Compose, você usa um arquivo do tipo yaml para definir como será o ambiente de sua aplicação e usando um único comando você criará e iniciará todos os serviços definidos.

Compose é ótimo para desenvolvimento, testes e homologação, bem como para melhorar seu fluxo de integração contínua. Por exemplo:

- Em ambiente de desenvolvimento: Você pode utilizar ele para simular todo o ambiente de produção, ou seja, precisando de serviço redis, php, mysql? Basta definir isso em um arquivo.yml e quando você executar o docker-compose up, todo esse ambiente está disponível para você, todas as dependências que sua stack precisa estará configurada e disponível para uso. Isso sem contar que este ambiente poderá ser isolado, sem depender ou prejudicar o trabalho dos demais da equipe.
- Automação de testes: Uma das premissas básicas para desenvolvimento e integração continua é ter uma base de testes automatizada, isso para garantir qualidade e agilidade na entrega de novas releases de sua aplicação. Pensando nisso, você pode utilizar o docker compose para criar sua própria suite de testes, e precisando apenas executar um docker-compose up para testar os 50, 100, 200 requisitos que você definiu.

## **Para utilizar o docker-compose você precisa ter em mente que será necessário seguir essas três etapas:**

Definir o ambiente necessário para sua aplicação utilizando um Dockerfile (que pode ser reproduzido em qualquer lugar que tenha Docker instalado);

1. Definir no arquivo.yml quais serviços são essenciais para sua aplicação e a relação entre elas.
2. Executar o comando docker-compose up para que seu ambiente seja criado e configurado.

## **Instalação**

Bem simples, basta executar o seguinte comando (caso Linux), se você estiver utilizando em Windows via Docker ToolBox você já terá disponível o docker-compose.

```
curl -L https://github.com/docker/compose/releases/download/1.5.2/docker-  
compose-`uname -s`-`uname -m`  
  
\ > /usr/local/bin/docker-compose
```

Depois:

```
chmod +x /usr/local/bin/docker-compose
```

Compose eu escolho você

## Vamos testar com um WordPress, é fácil:

1. Criar um diretório de trabalho (mkdir /my-app);
2. Faça download do WordPress: cd /my-app; curl https://wordpress.org/latest.tar.gz | tar -xvzf –
3. Criar um arquivo Dockerfile com o seguinte código:

```
FROM orchardup/php5
```

```
ADD . /app
```

1. Criar um arquivo my-app.yml com o código:

```
web:
```

```
  build: .
```

```
  command: php -S 0.0.0.0:8000 -t /my-app
```

```
  ports:
```

```
    - "80:8000"
```

```
  links:
```

```
    - db
```

```
  volumes:
```

```
    - .:/my-app
```

```
db:
```

```
  image: orchardup/mysql
```

```
  environment:
```

```
    MYSQL_DATABASE: wordpress
```

## 2. Certifique-se de que seu wp-config esteja parecido com este:

```
<?php

define('DB_NAME', 'wordpress');

define('DB_USER', 'root');

define('DB_PASSWORD', '');

define('DB_HOST', "db:3306");

define('DB_CHARSET', 'utf8');

define('DB_COLLATE', '');


define('AUTH_KEY',          'put your unique phrase here');

define('SECURE_AUTH_KEY',   'put your unique phrase here');

define('LOGGED_IN_KEY',     'put your unique phrase here');

define('NONCE_KEY',         'put your unique phrase here');

define('AUTH_SALT',         'put your unique phrase here');

define('SECURE_AUTH_SALT',  'put your unique phrase here');

define('LOGGED_IN_SALT',    'put your unique phrase here');

define('NONCE_SALT',        'put your unique phrase here');


$table_prefix  = 'wp_';
```

```
define('WPLANG', '');

define('WP_DEBUG', false);

if ( !defined('ABSPATH') )

    define('ABSPATH', dirname(__FILE__) . '/');

require_once(ABSPATH . 'wp-settings.php');
```

3. O que falta? Docker-compose up no diretório onde você criou o Dockerfile e o my-app.yml.

Agora é só acessar: <http://ipdamaquina> e prosseguir com a instalação do Wordpress normalmente, o que o Compose fez então? No passo 3 nós criamos um arquivo Dockerfile contendo a descrição da imagem que queremos criar e que será utilizada como base para o nosso ambiente, no passo 4 nós definimos qual era esse ambiente, veja que definimos 1 container para atender requisições web e 1 banco de dados, quando executamos o docker-compose up ele criará a imagem baseado no Dockerfile e criar os containers de serviços que definimos no my-app.yml. O mais legal, você pode quer escalar seu ambiente, ter mais containers web? Docker-compose scale web=5 e o Compose criará e iniciará 5 containers do serviço web que definimos no my-app.yml.

## **Alguns dados importantes sobre o Docker Compose:**



- O Compose preserva todos os volumes atribuídos aos seus serviços, por exemplo, quando você executa o `docker-compose up`, se você já tiver algum outro container utilizando o volume informado, o Compose copiará o volume do container antigo para o novo container, sem perder nenhuma informação.
- O Compose recriará apenas containers cujas configurações foram modificadas, caso contrário ele se baseará nas configurações que ele já tem armazenada em cache, isso é muito útil principalmente para provisionar e escalar seu ambiente de forma muito mais rápida.

Você ainda pode utilizar o Docker Compose juntamente com o Docker Machine e provisionar seu ambiente em qualquer lugar que precisar, isso é ainda mais útil para quem tem seu serviço de infraestrutura terceirizado com outros provedores (AWS, Digital Ocean, etc).

## **LINUX**

### **Sistema de arquivos**

Então um sistema de arquivos é uma espécie de gerenciador e organizador que irá permitir o sistema operacional ler os arquivos que estão no HD. Esta é a finalidade básica de um sistema de arquivos.

#### **EXT2**

Um dos primeiros sistemas de arquivos utilizando nas primeiras versões das distros Linux foi o Second Extended FileSystem, ou simplesmente EXT2. Embora ele tenha sido uma espécie de padrão não era muito eficiente.

#### **EXT3**

O EXT3 acabou se popularizando nas distros por ter um suporte mais eficiente do que seu antecessor. Este suporte recebe o nome de journaling. Logo abaixo será explicado um pouco melhor sua função.

#### **EXT4**

Este é uma espécie de versão do EXT3. Este sistema de arquivos surgiu com a prerrogativa de melhorar o desempenho de compatibilidade, formatos e limites de armazenamentos.

## **REISERFS**

O sistema de arquivos ReiserFS foi criado recentemente. Mas atualmente quase todas as distribuições Linux o suportam. Sua performance é muito boa, principalmente para um número muito grande de arquivos pequenos. ReiserFS também possui suporte a journaling.

## **XFS**

O sistema de arquivos XFS também possui suporte a journaling. Foi desenvolvido originalmente pela Silicon Graphics e posteriormente disponibilizado o código fonte. O XFS é considerado um dos melhores sistemas de arquivos para banco de dados, pois é muito rápido na gravação. XFS utiliza muitos recursos de cache com memória RAM, e para utilizar XFS é recomendado utilizar sistemas que possuem redundância de energia.

## **SWAP**

SWAP é um espaço reservado para troca de dados com a memória RAM.

Em alguns lugares ele não é mencionado como um Sistema de Arquivos, mas resolvi descrever aqui pois faz parte deste artigo.

## **VFAT**

O sistema de arquivos VFAT não possui suporte a journaling. É utilizado normalmente para transferir dados entre sistemas M\$ Windows e o Linux instalados no mesmo disco, pois pode ser lido e escrito por ambos os sistemas operacionais. O sistema de arquivos VFAT está longe de ser um sistema de arquivos utilizado para Sistemas Linux, exceto para compartilhamento/compatibilidade entre o M\$ Windows e Linux. Se você utilizar VFAT no Linux, esteja certo de perder alguns atributos, tais como: permissão de execução, links simbólicos, entre outras coisas.

## **Estrutura de diretórios**

Para manter a organização, desenvolvedores de distribuições Linux e softwares diversos seguem o Filesystem Hierarchy Standard (padrão para sistema de arquivos hierárquico), ou FHS, uma espécie de referência que padroniza quais pastas do sistema recebem determinados tipos de arquivo.

### **O diretório raiz (/)**

Todos os arquivos e diretórios do sistema Linux instalado no computador partem de uma única origem: o diretório raiz. Mesmo que estejam armazenados em outros dispositivos físicos, é a partir do diretório raiz – representado pela barra (/) – que você poderá acessá-los. O único usuário do sistema capaz de criar ou mover arquivos do diretório raiz é o root, ou seja, o usuário-administrador.

### **Binários executáveis: /bin**

No diretório /bin estão localizados os binários executáveis que podem ser utilizados por qualquer usuário do sistema. São comandos essenciais, usados para trabalhar com arquivos, textos e alguns recursos básicos de rede, como o cp, mv, ping e grep.

### **Binários do sistema: /sbin**

Assim como o /bin, este diretório armazena executáveis, mas com um diferencial: são aplicativos utilizados por administradores de sistema com o propósito de realizar funções de manutenção e outras tarefas semelhantes.

### **Programas diversos: /usr**

O /usr reúne executáveis, bibliotecas e até documentação de softwares usados pelos usuários ou administradores do sistema.

### **Configurações do sistema: /etc**

No diretório /etc ficam arquivos de configuração que podem ser usados por todos os softwares, além de scripts especiais para iniciar ou interromper módulos e programas diversos.

### **Bibliotecas: /lib**

Neste ponto do sistema de arquivos ficam localizadas as bibliotecas usadas pelos comandos presentes em /bin e /sbin.

### **Opcionais: /opt**

Aplicativos adicionais, que não são essenciais para o sistema, terminam neste diretório.

### **Arquivos pessoais: /home**

No diretório /home ficam os arquivos pessoais, como documentos e fotografias, sempre dentro de pastas que levam o nome de cada usuário.

### **Inicialização: /boot**

Arquivos relacionados à inicialização do sistema, ou seja, o processo de boot do Linux, quando o computador é ligado, ficam em /boot.

### **Volumes e mídias: /mnt e /media**

Para acessar os arquivos de um CD, pendrive ou disco rígido presente em outra máquina da rede, é necessário "montar" esse conteúdo no sistema de arquivos local, isso é, torná-lo acessível como se fosse apenas mais um diretório no sistema.

### **Serviços: /srv**

Dados de servidores e serviços em execução no computador ficam armazenados dentro desse diretório.

### **Arquivos de dispositivos: /dev**

No Linux, tudo é apresentado na forma de arquivos. Ao plugar um pendrive no computador, por exemplo, um arquivo será criado dentro do diretório /dev e ele servirá como interface para acessar ou gerenciar o drive USB. Nesse diretório, você encontra caminhos semelhantes para acessar terminais e qualquer dispositivo conectado ao computador, como o mouse e até modems.

### **Arquivos variáveis: /var**

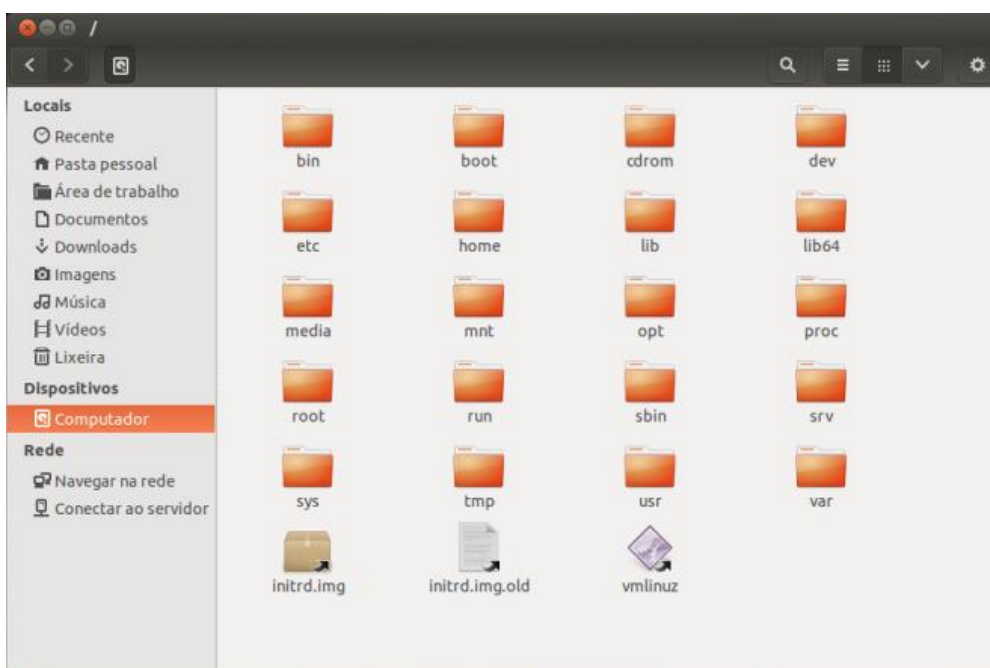
Todo arquivo que aumenta de tamanho ao longo do tempo está no diretório de arquivos variáveis.

### **Processos do sistema: /proc**

Nesse diretório são encontrados arquivos que revelam informações sobre os recursos e processos em execução no sistema.

### **Arquivos temporários: /tmp**

Arquivos e diretórios criados temporariamente tanto pelo sistema quanto pelos usuários devem ficar nesse diretório. Boa parte deles é apagada sempre que o computador é reiniciado.



As permissões de acesso protegem o sistema de arquivos Linux do acesso indevido de pessoas ou programas não autorizados. A permissão de acesso do GNU/Linux também impede que um programa mal intencionado, por exemplo, apague um arquivo que não deve, envie arquivos especiais para outra pessoa ou forneça acesso da rede para que outros usuários invadam o sistema.

## **Donos, Grupos e outros usuários**

A idéia básica da segurança no sistema GNU/Linux é definir o acesso aos arquivos por donos, grupos e outros usuários:

### **dono**

É a pessoa que criou o arquivo ou o diretório. O nome do dono do arquivo/diretório é o mesmo do usuário usado para entrar no sistema GNU/Linux. Somente o dono pode modificar as permissões de acesso do arquivo.

As permissões de acesso do dono de um arquivo somente se aplicam ao dono do arquivo/diretório. A identificação do dono também é chamada de user id (UID).

A identificação de usuário ao qual o arquivo pertence é armazenada no arquivo `/etc/passwd` e do grupo no arquivo `/etc/group`. Estes são arquivos textos comuns e podem ser editados em qualquer editor de texto, mas utilize preferencialmente os comandos `vipw` e `vigr` que executa procedimentos adicionais de checagem de uids e grupos após a alteração. Tenha cuidado para não modificar o campo que contém a senha do usuário encriptada (que pode estar armazenada no arquivo `/etc/passwd` caso não estiver usando senhas ocultas).

### **grupo**

Permite que vários usuários diferentes tenham acesso a um mesmo arquivo (já que somente o dono poderia ter acesso ao arquivo). Cada usuário pode fazer parte de um ou mais grupos e então acessar arquivos que pertençam ao mesmo grupo que o seu (mesmo que estes arquivos tenham outro dono).

Por padrão, quando um novo usuário é criado e não especificar nenhum grupo, ele pertencerá ao grupo de mesmo nome do seu grupo primário (este

comportamento é controlado pelo parâmetro USERGROUPS=yes do arquivo /etc/adduser.conf. A identificação do grupo é chamada de GID (group id). Um usuário pode pertencer a um ou mais grupos.

## **outros**

É a categoria de usuários que não são donos ou não pertencem ao grupo do arquivo. Cada um dos tipos acima possuem três tipos básicos de permissões de acesso que serão vistas na próxima seção.

## **Tipos de Permissões de Acesso**

Quanto aos tipos de permissões que se aplicam ao dono, grupo e outros usuários, temos 3 permissões básicas:

r - Permissão de leitura para arquivos. Caso for um diretório, permite listar seu conteúdo (através do comando ls, por exemplo).

w - Permissão de gravação para arquivos. Caso for um diretório, permite a gravação de arquivos ou outros diretórios dentro dele.

Para que um arquivo/diretório possa ser apagado, é necessário o acesso a gravação.

x - Permite executar um arquivo (caso seja um programa executável). Caso seja um diretório, permite que seja acessado através do comando cd.

As permissões de acesso a um arquivo/diretório podem ser visualizadas com o uso do comando ls -la. As 3 letras (rwx) são agrupadas da seguinte forma:

```
-rwxr-xr-- gleydson users teste
```

A primeira letra diz qual é o tipo do arquivo. Caso tiver um "d" é um diretório, um "l" um link a um arquivo no sistema, um "-" quer dizer que é um arquivo comum, etc.

Da segunda a quarta letra (rwx) dizem qual é a permissão de acesso ao dono do arquivo. Neste caso gleydson ele tem a permissão de ler (r - read), gravar (w - write) e executar (x - execute) o arquivo teste.

Da quinta a sétima letra (r-x) diz qual é a permissão de acesso ao grupo do arquivo. Neste caso todos os usuários que pertencem ao grupo users tem a permissão de ler (r), e também executar (x) o arquivo teste.

Da oitava a décima letra (r--) diz qual é a permissão de acesso para os outros usuários. Neste caso todos os usuários que não são donos do arquivo teste tem a permissão somente para ler o programa.

### **Etapas para acesso a um arquivo/diretório**

O acesso a um arquivo/diretório é feito verificando primeiro se o usuário que acessará o arquivo é o seu dono, caso seja, as permissões de dono do arquivo são aplicadas. Caso não seja o dono do arquivo/diretório, é verificado se ele pertence ao grupo correspondente, caso pertença, as permissões do grupo são aplicadas. Caso não pertença ao grupo, são verificadas as permissões de acesso para os outros usuários que não são donos e não pertencem ao grupo correspondente ao arquivo/diretório.

Após verificar aonde o usuário se encaixa nas permissões de acesso do arquivo (se ele é o dono, pertence ao grupo, ou outros usuários), é verificado se ele terá permissão de acesso para o que deseja fazer (ler, gravar ou executar o arquivo), caso não tenha, o acesso é negado, mostrando uma mensagem do tipo: "Permission denied" (permissão negada).

O que isto quer dizer é que mesmo que você seja o dono do arquivo e definir o acesso do dono (através do comando `chmod`) como somente leitura (r) mas o acesso dos outros usuários como leitura e gravação, você somente poderá ler este arquivo mas os outros usuários poderão ler/gravá-lo.

As permissões de acesso (leitura, gravação, execução) para donos, grupos e outros usuários são independentes, permitindo assim um nível de acesso diferenciado.

### **Exemplo de acesso a um arquivo**

Abaixo um exemplo e explicação das permissões de acesso a um arquivo no GNU/Linux (obtido com o comando `ls -la`, explicarei passo a passo cada parte:

```
-rwxr-xr-- 1 gleydson user 8192 nov 4 16:00 teste
```



-rwxr-xr--

Estas são as permissões de acesso ao arquivo teste. Um conjunto de 10 letras que especificam o tipo do arquivo, permissão do dono do arquivo, grupo do arquivo e outros usuários. Veja a explicação detalhada sobre cada uma abaixo:

-rwxr-xr--

A primeira letra (do conjunto das 10 letras) determina o tipo do arquivos. Se a letra for um d é um diretório, e você poderá acessá-lo usando o comando cd. Caso for um l é um link simbólico para algum arquivo ou diretório no sistema. Um - significa que é um arquivo normal.

-rwxr-xr--

Estas 3 letras (da segunda a quarta do conjunto das 10 letras) são as permissões de acesso do dono do arquivo teste. O dono (neste caso gleydson) tem a permissão para ler (r), gravar (w) e executar (x) o arquivo teste.

-rwxr-xr--

Estes 3 símbolos (do quinto ao sétimo do conjunto de 10) são as permissões de acesso dos usuários que pertencem ao grupo user do arquivo teste. Os usuários que pertencem ao grupo user tem a permissão somente para ler (r) e executar (x) o arquivo teste não podendo modificá-lo ou apagá-lo.

-rwxr-xr--

Estes 3 símbolos (do oitavo ao décimo) são as permissões de acesso para usuários que não são donos do arquivo teste e que não pertencem ao grupo user. Neste caso, estas pessoas somente terão a permissão para ver o conteúdo do arquivo teste.

gleydson

Nome do dono do arquivo teste.

user

Nome do grupo que o arquivo teste pertence.

teste

Nome do arquivo.

Exemplo de acesso a um diretório

Abaixo um exemplo com explicações das permissões de acesso a um diretório no GNU/Linux:

drwxr-x--- 2 gleydson user 1024 nov 4 17:55 exemplo

drwxr-x---

Permissões de acesso ao diretório exemplo. É um conjunto de 10 letras que especificam o tipo de arquivo, permissão do dono do diretório, grupo que o diretório pertence e permissão de acesso a outros usuários.

drwxr-x---

A primeira letra (do conjunto das 10) determina o tipo do arquivo. Neste caso é um diretório porque tem a letra d.

drwxr-x---

Estas 3 letras (da segunda a quarta) são as permissões de acesso do dono do diretório exemplo. O dono do diretório (neste caso gleydson) tem a permissão para listar arquivos do diretório (r), gravar arquivos no diretório (w) e entrar no diretório (x).

drwxr-x---

Estas 3 letras (da quinta a sétima) são as permissões de acesso dos usuários que pertencem ao grupo user. Os usuários que pertencem ao grupo user tem a permissão somente para listar arquivos do diretório (r) e entrar no diretório (x) exemplo.

drwxr-x---

Estes 3 símbolos (do oitavo ao décimo) são as permissões de acesso para usuários que não são donos do diretório exemplo e que não pertencem ao grupo user. Com as permissões acima, nenhum usuário que se encaixe nas condições de dono e grupo do diretório tem a permissão de acessá-lo.

gleydson

Nome do dono do diretório exemplo.

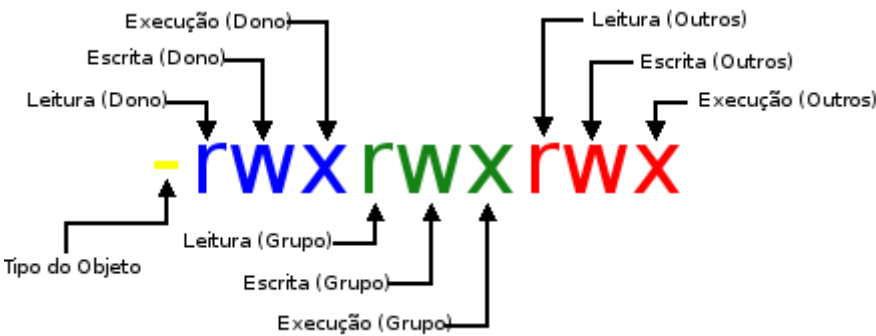
user

Nome do grupo que diretório exemplo pertence.

exemplo

Nome do diretório.

Permissão	Binário	Decimal
---	000	0
--X	001	1
-W-	010	2
-WX	011	3
r--	100	4
r-X	101	5
rw-	110	6
rwX	111	7



## **Permissão de acesso stick bit**

### **STICKY (STICKY BIT)**

Em arquivos executáveis, a propriedade Sticky faz com que o sistema mantenha uma imagem do programa em memória depois que o programa finalizar. Esta capacidade aumenta o desempenho, pois será feito um cache do programa para a memória e da próxima vez que ele for executado, será carregado mais rápido.

Em diretórios, a propriedade Sticky impede que outros usuários deletem ou renomeam arquivos dos quais não são donos. Isso normalmente é utilizado para aumentar a segurança, pois o diretório estará em modo append-only (somente incremento). Sendo assim, somente o usuário que é Dono do arquivo, poderá deletar ou renomear os arquivos dentro de um diretório com a propriedade Sticky aplicada.

A permissão especial Sticky pode ser especificada somente no campo outros usuários das permissões de acesso.

Exemplo:

No diretório /tmp, todos os usuários devem ter acesso para que seus programas possam criar os arquivos temporários, mas nenhum pode apagar arquivos dos outros. Então é interessante aplicar a propriedade Sticky no diretório /tmp.

Comandos:

#### **Aplicando Sticky:**

Aplicando a propriedade Sticky em um arquivo executável utilizando formato octal (1):

```
# chmod 1750 programa_pesado.sh
```

```
# ls -lah programa_pesado.sh
```

```
-rwxr-x--T 1 root root 2,9M 2006-09-26 23:51 programa_pesado.sh
```

Aplicando a propriedade Sticky em um arquivo executável utilizando formato simbólico (t):

```
# chmod o+t programa_pesado.sh
```

```
# ls -lah programa_pesado.sh
```

```
-rwxr-x--T 1 root root 2,9M 2006-09-26 23:51 programa_pesado.sh
```

Aplicando a propriedade Sticky em um diretório utilizando formato simbólico (t):

```
# chmod o+t /tmp
```

```
# ls -lah /tmp
```

```
drwxrwxrwt 8 root root 264 2006-09-26 23:22 .
```

Aplicando a propriedade Sticky em um diretório utilizando formato octal (1):

```
# chmod 1777 /tmp
```

```
# ls -lah /tmp
```

```
drwxrwxrwt 8 root root 264 2006-09-26 23:22 .
```

Retirando Sticky:

```
# chmod o-t /tmp
```

```
# ls -lah /tmp
```

```
drwxrwxrwx 8 root root 264 2006-09-26 23:22 .
```

Procurando Sticky:

Aplicando a propriedade Sticky em um diretório utilizando simbólico (t):

```
# find /home/roberto/ -perm /o=t
```

/home/roberto/programa\_pesado.sh

Aplicando a propriedade Sticky em um diretório utilizando formato octal (1):

```
# find /home/roberto/ -perm -1000
```

/home/roberto/programa\_pesado.sh

## **Funcionamento do Git**

Git é um sistema de controle de versão de arquivos. Através deles podemos desenvolver projetos na qual diversas pessoas podem contribuir

simultaneamente no mesmo, editando e criando novos arquivos e permitindo que os mesmos possam existir sem o risco de suas alterações serem sobrescritas.

Uma das aplicações do git é justamente essa, permitir que um arquivo possa ser editado ao mesmo tempo por pessoas diferentes. Por mais complexo que isso seja, ele tenta manter tudo em ordem para evitar problemas para nós desenvolvedores.

Outro fator importante do git (e essa é um dos seus diferenciais em relação ao svn – caso vc o conheça) é a possibilidade de criar, a qualquer momento, vários snapshots do seu projeto

## **GitHub**

O Github é um serviço web que oferece diversas funcionalidades extras aplicadas ao git.