

# API Rest e SOAP

#ninhodaaguia2022

## REST - Transferência de Estado Representacional

- Modelo de arquitetura.
- Possibilita que sistemas distribuídos se comuniquem usando protocolos existentes da Web.
- Em uma aplicação REST, os métodos mais utilizados são: GET, POST, PUT e DELETE.

## SOAP - Protocolo Simples de Acesso a Objetos

- É um formato de mensagem XML usado nas interações de serviços da Web.
- É um protocolo para troca de informações estruturadas.
- Os dados são transportados envelopados.

## As principais diferenças:

- A forma em que as informações são transportadas.
- O tempo em que estas informações levam para serem lidas.

# JSON e XML

#ninhodaaguia2022

# JSON – JavaScript Object Notation

- O formato JSON é, como o nome sugere, uma forma de notação de objetos JavaScript de modo que eles possam ser representados de uma forma comum a diversas linguagens.
- O objetivo é entregar uma solução simples de ser lida, leve e rápida, para que seja facilmente trafegado entre aplicações em quaisquer protocolos, inclusive o HTTP.

```
{
  "cliente": {
    "id": 2020,
    "nome": "Maria Aparecida",
  },
  "pagamentos": [
    {
      "id": 123,
      "descricao": "Compra do livro Cangaceiro JavaScript",
      "valor": 50.5
    },
    {
      "id": 124,
      "descricao": "Mensalidade escolar",
      "valor": 1500
    }
  ]
}
```

# JSON – JavaScript Object Notation

➤ Um JSON deve conter apenas informações que possam ser representadas em formato de texto. Por exemplo:

- Não pode ter funções;
- Não pode ter comentários;
- Todo texto sempre tem aspas duplas;
- As propriedades sempre tem aspas duplas.

```
[
  {
    "titulo": "JSON x XML",
    "resumo": "o duelo de dois modelos de representação de informações",
    "ano": 2012,
    "genero": ["aventura", "ação", "ficção"]
  },
  {
    "titulo": "JSON James",
    "resumo": "a história de uma lenda do velho oeste",
    "ano": 2012,
    "genero": ["western"]
  }
]
```

# XML – eXtensible Markup Language

- É uma linguagem de marcação para a criação de documentos com dados organizados hierarquicamente, tais como textos, banco de dados ou desenhos vetoriais.
- É classificada como extensível porque permite definir os elementos de marcação.
- As linguagens padrão para consulta XML são XPath e XQuery.

```
<?xml version="1.0">
<filmes>
  <filme id="1">
    <titulo>O XML veste prada</titulo>
    <resumo>O filme mostra a elegância da XML na representação de dados estruturados e semi
estruturados.</resumo>
    <genero>Aventura</genero>
    <genero>Documentário</genero>
    <elenco>
      <ator>Mark UPlanguage</ator>
      <ator>Mary well-Formed</ator>
      <ator>Sedna D. Atabase</ator>
    </elenco>
  </filme>
</filmes>
```

# JSON vs XML

- No caso de um número baixo de dados, o JSON e seu pareamento chave-valor torna claro o objeto a ser tratado.
- Porém, ao se tratar de um grande volume de valores complexos, a estrutura em árvore do XML torna as coisas menos difíceis de entender.

```
{  
  Servers: [  
    {  
      name: Server1,  
      owner: John,  
      created: 123456,  
      status: active  
    }  
  ]  
}
```

JSON

```
<Servers>  
  <Server>  
    <name>Server1</name>  
    <owner>John</owner>  
    <created>123456</created>  
    <status>active</status>  
  </Server>  
</Servers>
```

XML

# Semelhanças

- Os dois modelos representam informações no formato texto.
- Ambos possuem natureza auto-descritiva.
- Ambos são capazes de representar informação complexa, difícil de representar no formato tabular. Alguns exemplos: objetos compostos (objetos dentro de objetos), relações de hierarquia, atributos multivalorados, arrays, dados ausentes, etc.
- Ambos podem ser utilizados para transportar informações em aplicações AJAX.
- Ambos podem ser considerados padrões para representação de dados. XML é um padrão W3C, enquanto JSON foi formalizado na RFC 4627.
- Ambos são independentes de linguagem. Dados representados em XML e JSON podem ser acessados por qualquer linguagem de programação, através de API's específicas.



# Diferenças

- JSON não é uma linguagem de marcação. Não possui tag de abertura e muito menos de fechamento!
- JSON representa as informações de forma mais compacta, não permite a execução de instruções de processamento, algo possível em XML.
- JSON é tipicamente destinado para a troca de informações, enquanto XML possui mais aplicações.
  - Por exemplo: nos dias atuais existem bancos de dados inteiros armazenados em XML e estruturados em SGBD's XML nativo.

# Conceitos de Programação Orientada a Objetos

#ninhodaaguia2022

# Programação estruturada vs Programação Orientada a Objetos

## Estruturada

Um programa é composto por três tipos básicos de estruturas: **sequências**, **condições** e **repetições**.



## Orientada a objetos

Se baseia, principalmente, em dois conceitos: **classes** e **objetos**. Todos os outros, igualmente importantes, são construídos em cima desses dois.



# Classe

- É um molde utilizado na criação dos objetos.
- É um conjunto de características e métodos que definem objetos pertencentes à ela.
- Definem a quais mensagens seus atributos respondem.

```
public class Cachorro {  
    int patas = 4;  
  
    public void latir() {  
        System.out.println("AU, AU!");  
    }  
}
```

Cachorro.java

# Objeto e Método

- Objeto é uma construção que encapsula estado e comportamento.
- Ele é uma instância de uma classe.
- Métodos são os comportamentos das classes.

```
class Main {  
    public static void main(String[] args) {  
        Cachorro pitoco = new Cachorro();  
  
        System.out.println(pitoco.patas);  
        pitoco.latir();  
    }  
}
```

Cachorro.java

```
public class Cachorro {  
    int patas = 4;  
  
    public void latir() {  
        System.out.println("AU, AU!");  
    }  
  
    public void abanarORabo() {  
        System.out.println("Abanando o rabo.");  
    }  
}
```

Cachorro.java

# Encapsulamento

- O encapsulamento é um conceito que diz que deve haver uma separação entre comportamento interno de uma classe com a interface que ela disponibiliza para os seus clientes.
- Ele evita o vazamento de escopo.

```
public class Carro {  
    private int velocidade = 0;  
  
    public int getVelocidade() {  
        return this.velocidade;  
    }  
  
    public void setVelocidade(int velocidade) {  
        this.velocidade = velocidade;  
    }  
}
```

Carro.java

# Herança

- A herança é uma característica do paradigma orientado a objetos que permite que abstrações possam ser definidas em diversos níveis.
- Uma classe herda as características de uma outra classe.

```
public class Carro {  
    private String nome;  
  
    public void setNome(String nomeCarro) {  
        this.nome = nomeCarro;  
    }  
  
    public String getNome() {  
        return this.nome;  
    }  
}
```

Carro.java

# Herança

- Quando dizemos que uma classe A é um tipo de classe B, dizemos que a classe A herda as características da classe B e que a classe B é mãe da classe A, estabelecendo então uma relação de herança entre elas.

```
public class Fusca extends Carro {  
}  
  
public class Gol extends Carro {  
}  
  
public class Ferrari extends Carro {  
}
```

Fusca.java; Gol.java; Ferrari.java

```
class Main {  
    public static void main(String[] args) {  
        Fusca fusquinha = new Fusca();  
        Gol golzinho = new Gol();  
        Ferrari ferrari = new Ferrari();  
  
        fusquinha.setNome("Fusca");  
        golzinho.setNome("Gol");  
        ferrari.setNome("Ferrari");  
  
        System.out.println(fusquinha.getNome());  
        System.out.println(golzinho.getNome());  
        System.out.println(ferrari.getNome());  
    }  
}
```

Main.java



# Polimorfismo

- Polimorfismo vem do grego poli = muitas, e morphos = forma. Significa muitas formas de fazer algo.

```
public class Carro {  
    public void tocarMusica() {  
        System.out.println("Tocador de música genérico.");  
    }  
}
```

Carro.java

```
public class Fusca extends Carro {  
    public void tocarMusica() {  
        System.out.println("Toca música via fita cassete.");  
    }  
}  
  
public class Gol extends Carro {  
    public void tocarMusica() {  
        System.out.println("Toca música via Bluetooth.");  
    }  
}  
  
public class Ferrari extends Carro {  
    public void tocarMusica() {  
        System.out.println("Toca música via DVD.");  
    }  
}
```

Fusca.java; Gol.java; Ferrari.java

# Polimorfismo

- Dois objetos, de duas classes diferentes, têm um mesmo método, que é implementado de formas diferentes. Ou seja, um método possui várias formas, várias implementações diferentes em classes diferentes, mas que possuem o **mesmo efeito**.

```
class Main {  
    public static void main(String[] args) {  
        Fusca fusquinha = new Fusca();  
        Gol golzinho = new Gol();  
        Ferrari ferrari = new Ferrari();  
  
        fusquinha.tocarMusica();  
        golzinho.tocarMusica();  
        ferrari.tocarMusica();  
    }  
}
```

Main.java

```
class Main {  
    public static main(String[] args) {  
  
        Carro carro = new Fusca();  
        carro.tocarMusica();  
    }  
}
```

Main.java

# Classes abstratas

- É um tipo de classe especial que não pode ser instanciada, apenas herdada.
- Essas classes são muito importantes quando não queremos criar um objeto a partir de uma classe “geral”, apenas de suas “subclasses”.

```
public abstract class Conta {  
    private float saldo;  
  
    public void criarConta(){};  
  
    public void setSaldo(float saldo) {  
        this.saldo = saldo;  
    }  
  
    public float getSaldo() {  
        return this.saldo;  
    }  
}
```

Conta.java

```
public class ContaCorrente extends Conta {  
    @Override  
    public void criarConta() {  
        System.out.println("Criando Conta Corrente.");  
    }  
}  
  
public class ContaPoupanca extends Conta{  
    @Override  
    public void criarConta() {  
        System.out.println("Criando Conta Poupança.");  
    }  
}
```

ContaCorrente.java; ContaPoupanca.java

# Interface

- É um conjunto de métodos. Ao ser implementada(não herdada), ela força a classe a implementar os seus métodos.
- Quando uma classe implementa uma interface, é como se estivesse assinando um documento, no qual ela se compromete a implementar seus métodos.

```
public interface Automovel {  
  
    public void acelerar();  
    public void desacelerar();  
    public void acenderFarol();  
}
```

# Interface

```
public class Motocicleta implements Automovel {  
  
    @Override  
    public void acelerar() {  
        System.out.println("Acelera do jeito da Motocicleta.");  
    }  
  
    @Override  
    public void desacelerar() {  
        System.out.println("Desacelera do jeito da Motocicleta.");  
    }  
  
    @Override  
    public void acenderFarol() {  
        System.out.println("Acende o farol do jeito da Motocicleta.");  
    }  
}
```

Motocicleta.java

```
public class Carro implements Automovel {  
  
    @Override  
    public void acelerar() {  
        System.out.println("Acelera do jeito do Carro.");  
    }  
  
    @Override  
    public void desacelerar() {  
        System.out.println("Desacelera do jeito do Carro.");  
    }  
  
    @Override  
    public void acenderFarol() {  
        System.out.println("Acende o farol do jeito do Carro.");  
    }  
}
```

Carro.java

# Prática de Bancos de Dados

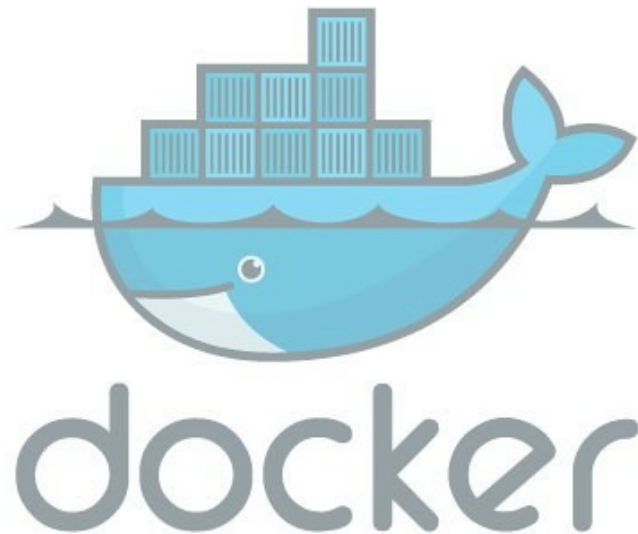
#ninhodaaguia2022

# Docker

#ninhodaaguia2022

# Explicação e conceito

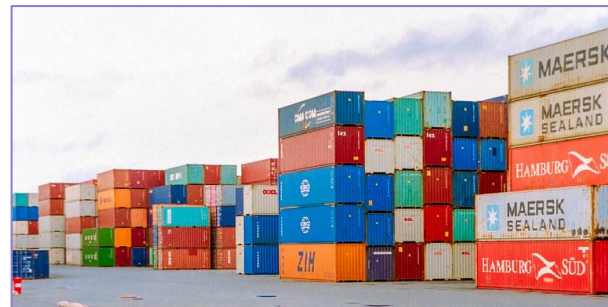
- O que é o Docker?
  - Definição
  - Docker vs Virtual Machines
- Conceitos e comandos
- Aplicação prática



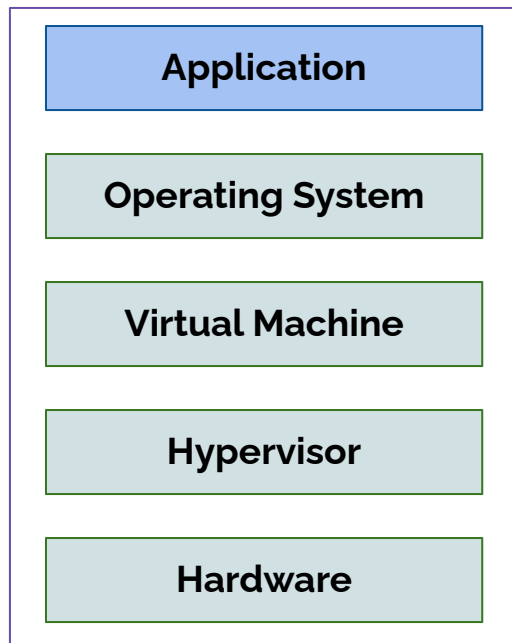


# O que é Docker?

- O Docker é uma plataforma aberta para desenvolvimento, que empacota um aplicativo com todas as suas dependências em uma unidade padronizada para desenvolvimento de software;
- Roda a aplicação sob diferentes circunstâncias;
- Separa seus aplicativos de sua infraestrutura para que você possa entregar software rapidamente;
- **Containers:** padronizar, isolar e transportar.

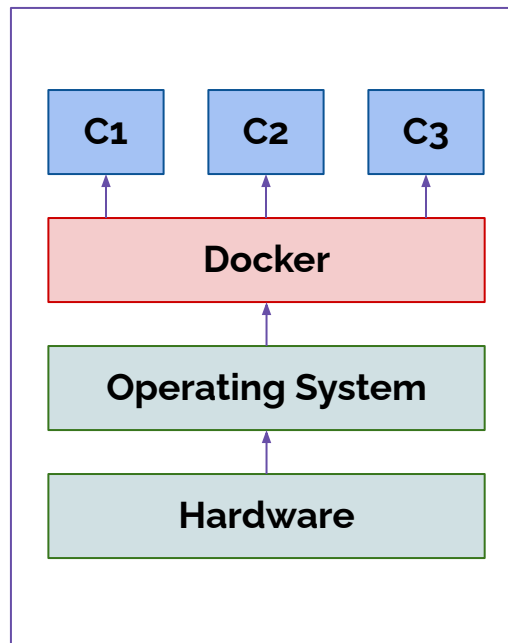


# Docker vs Virtual Machines



Virtual Machines

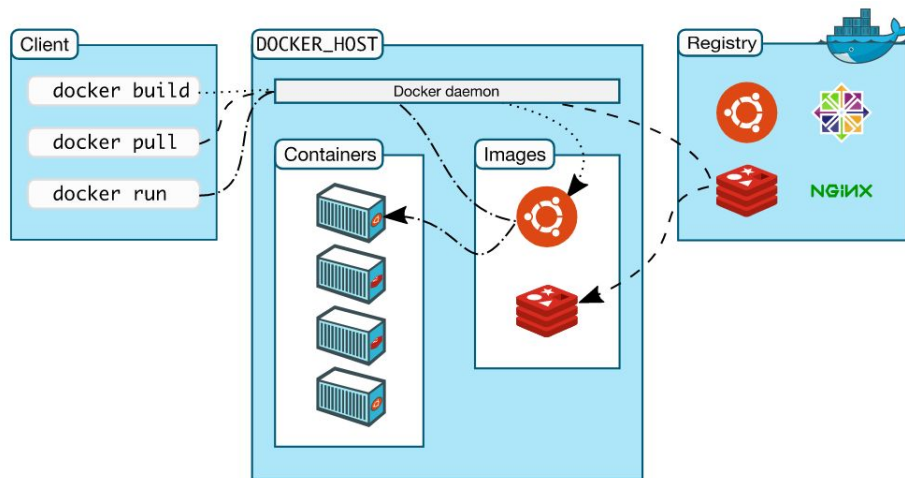
×



Docker

- **Vantagens:**
- Otimização de recursos
  - Empacotamento da aplicação
  - Facilidade no *deploy*

# Conceitos e comandos

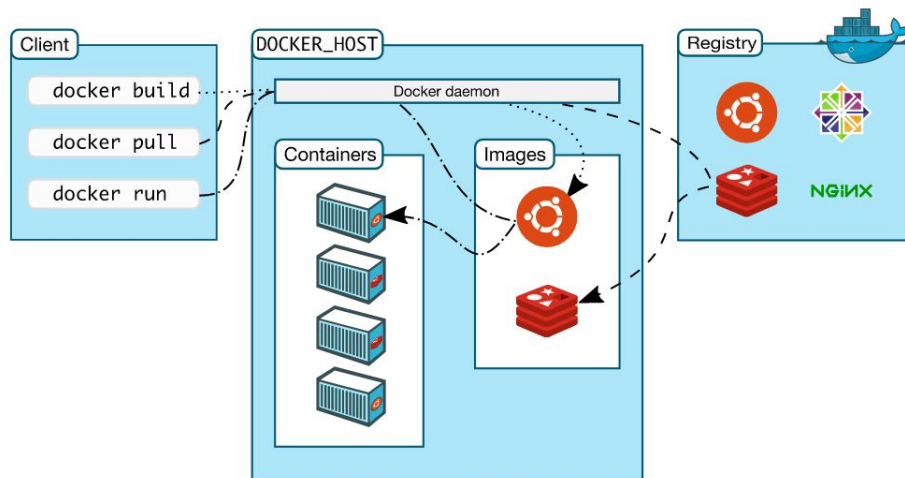


Fonte: <https://docs.docker.com/get-started/overview/>

## *Client*

- O Docker Client é a principal maneira pela qual muitos usuários do Docker interagem com o Docker.
- Pode ser uma CLI ou API Rest que aceita comandos do usuário e repassa para o Docker daemon.

# Conceitos e comandos

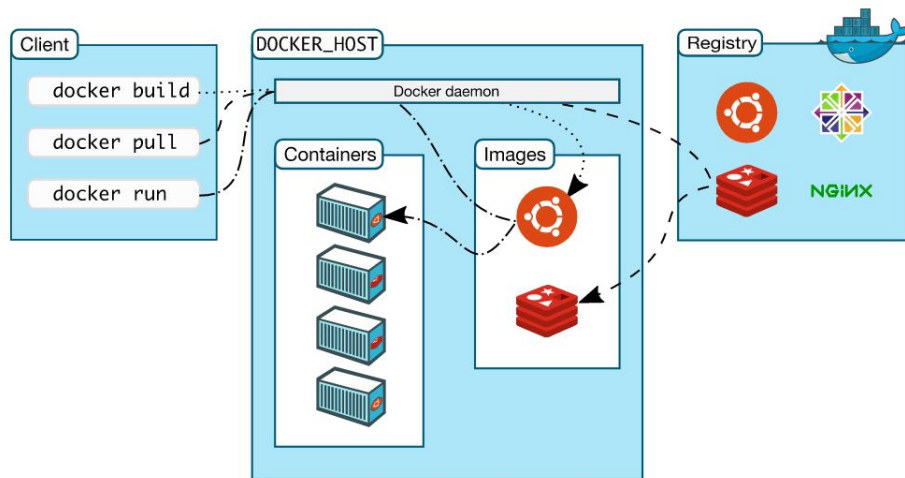


Fonte: <https://docs.docker.com/get-started/overview/>

## *Daemon*

- Software que roda na máquina em que o Docker está instalado.
- O daemon do Docker ( `dockerd` ) escuta as solicitações da API do Docker e gerencia objetos do Docker, como imagens, contêineres, redes e volumes.

# Conceitos e comandos

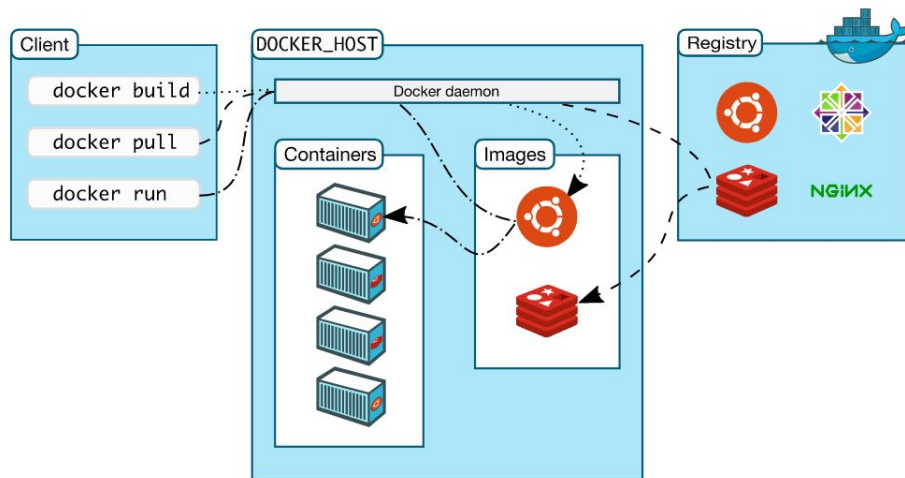


Fonte: <https://docs.docker.com/get-started/overview/>

## Registry

- São bancos que armazenam e distribuem imagens do Docker.
- Quando você usa os comandos (`docker pull`) ou (`docker run`), as imagens necessárias são extraídas do registro configurado.
- Quando você usa o comando (`docker push`), sua imagem é enviada para o registro configurado.
- Pode ser público ou privado. Exemplo de Registry: Docker Hub

# Conceitos e comandos

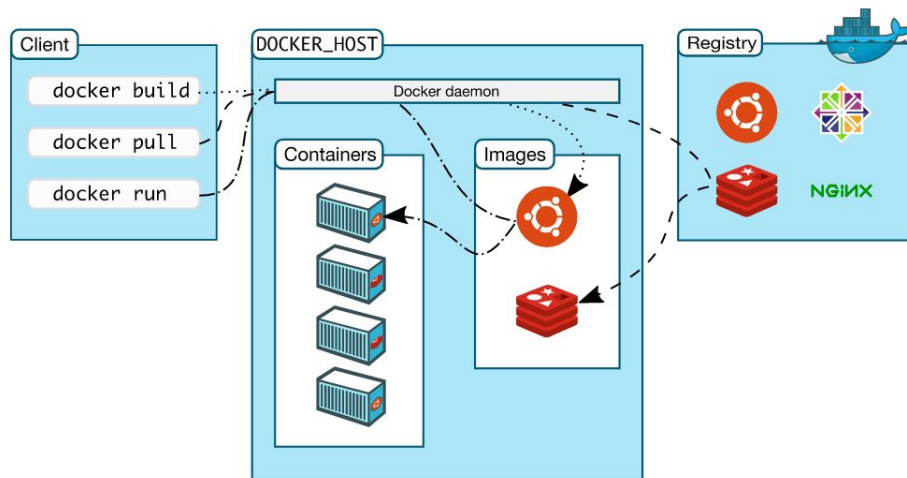


Fonte: <https://docs.docker.com/get-started/overview/>

## *Image:*

- Todos os dados e metadados necessários para executar containers.
- Muitas vezes, uma imagem é baseada em outra imagem, com alguma personalização adicional
- Você pode criar suas próprias imagens ou usar apenas aquelas criadas por outras pessoas e publicadas em um registro.

# Conceitos e comandos



Fonte: <https://docs.docker.com/get-started/overview/>

## Containers:

- Um contêiner é uma aplicação executável de uma imagem. Você pode criar, iniciar, parar, mover ou excluir um contêiner usando a API ou CLI do Docker.
- Você pode conectar um contêiner a uma ou mais redes, anexar armazenamento a ele ou até mesmo criar uma nova imagem com base em seu estado atual.
- Por padrão, um contêiner é relativamente bem isolado de outros contêineres e de sua máquina host.

# Conceitos

## Docker Hub

- O Docker Hub é um registry público que qualquer pessoa pode usar
- o Docker está configurado para procurar imagens nele por padrão. Você pode até mesmo executar seu próprio registro privado.

The screenshot shows the 'Create Repository' page on Docker Hub. The header includes the Docker Hub logo, a search bar, and navigation links for Explore, Repositories, Organizations, and Get Help. The main form has a 'Name' field with a dropdown set to 'docker' and a 'Description' field. The 'Visibility' section shows 'Public' as the selected option, with a note that public repositories appear in search results. The 'Private' option is also visible. Below this, the 'Build Settings' section is optional, showing 'Autobuild' as a feature that triggers a new build with every git push. At the bottom, there are three buttons: 'Cancel', 'Create', and 'Create & Build'. On the right side, there is a 'Pro tip' section with a code block showing the CLI commands for tagging and pushing a new image to a repository.

dockerhub Search for great content (e.g., mysql) Explore Repositories Organizations Get Help

Using 7 of 51 private repositories. [Get more](#)

Create Repository

docker Name

Description

Visibility

Using 7 of 51 private repositories. [Get more](#)

☐ Public Public repositories appear in Docker Hub search results

☒ Private Only you can view private repositories

Build Settings (optional)

Autobuild triggers a new build with every git push to your source code repository. [Learn More](#)

Disconnected Disconnected

Cancel Create Create & Build

Pro tip

You can push a new image to this repository using the CLI

```
docker tag local-image:tagname new-repo:tagname
docker push new-repo:tagname
```

Make sure to change tagname with your desired image repository tag.



# Conceitos

## *Dockerfile*

É um arquivo texto responsável pela criação de novas imagens.

## *Docker Compose*

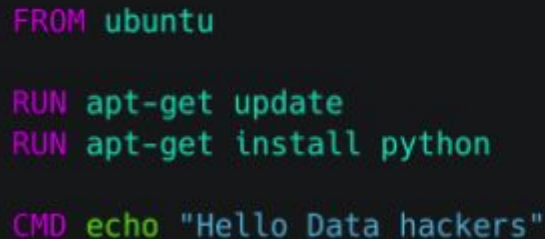
É usado para definir aplicações usando diversos containers.

## *Docker Swarm*

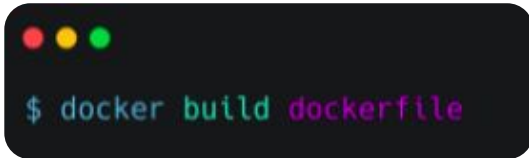
Ferramenta que permite o agrupamento e gerenciamento de Containers nativo do Docker.

## *Docker Engine*

Responsável por criar imagens e containers.



```
FROM ubuntu  
  
RUN apt-get update  
RUN apt-get install python  
  
CMD echo "Hello Data hackers"
```



```
$ docker build dockerfile
```

# Prática de Docker

#ninhodaaguia2022

**Obrigado!**