

Sistemas Distribuídos e Mobile

Tópicos Abordados

O que é arquitetura de software e quais são os padrões vigentes?

O que é uma arquitetura orientada a serviços, o que são microserviços, o modelo vigente e principais frameworks disponíveis?;



COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim; BLAIR, Gordon. Sistemas distribuídos. Porto Alegre: Bookman, 2013. E-book. Disponível em:
<https://integrada.minhabiblioteca.com.br/#/books/9788582600542/>. Acesso em: 17 fev. 2022

Introdução à Arquitetura de Software



A Arquitetura de Software

- A arquitetura de software é a base sobre a qual um sistema de software é construído. É um projeto de alto nível que estabelece como os componentes de software se relacionam e colaboram para atingir os objetivos do sistema.
- Uma abordagem cuidadosa ao planejamento e design arquitetônico pode impactar positivamente o sucesso a longo prazo do projeto de desenvolvimento de software, pois apoia o ciclo de vida do sistema, simplificando seu gerenciamento e otimizando a produtividade.



A Arquitetura de Software

- É comum assumir que o principal objetivo da arquitetura de software é garantir o funcionamento eficaz de um sistema. No entanto, **a sua influência direta no funcionamento do sistema é limitada.**
- Existem sistemas com arquiteturas deficientes que operam de forma surpreendentemente suave, enquanto desafios tendem a surgir em aspectos como implantação, manutenção e evolução contínua.



A Arquitetura de Software

“A arquitetura trata de coisas importantes.
Seja lá o que isso é.”

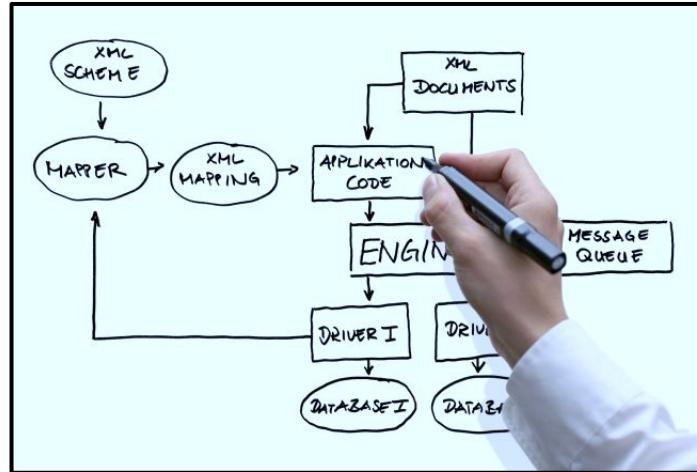
- Ralph Johnson

- A essência da arquitetura de software reside na **identificação e priorização de aspectos críticos, independentemente da sua natureza específica.** Isto envolve tomar decisões sobre quais elementos do sistema são essenciais para sua operação e desempenho.
- É importante **focar no que é essencial e garantir que estes elementos permaneçam em ótimas condições** ao longo do desenvolvimento e evolução do sistema.



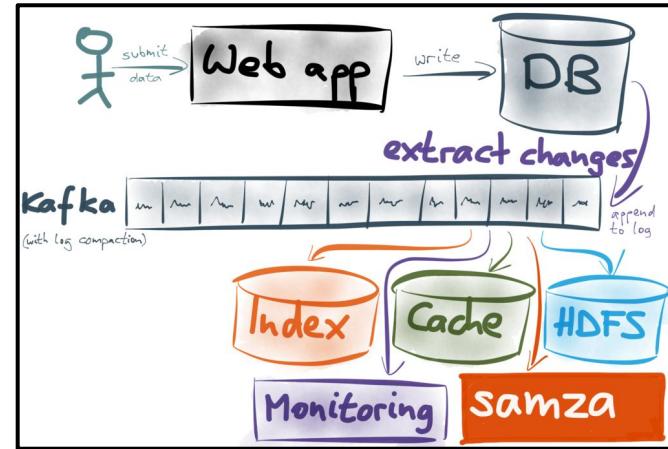
<https://martinfowler.com/architecture/>

Analogias de Arquitetura de Software



Assim como um arquiteto projeta uma casa com estrutura sólida, layout eficiente e design estético, um arquiteto de softwares planeja a estrutura e organização de um sistema de software.

Analogias de Arquitetura de Software

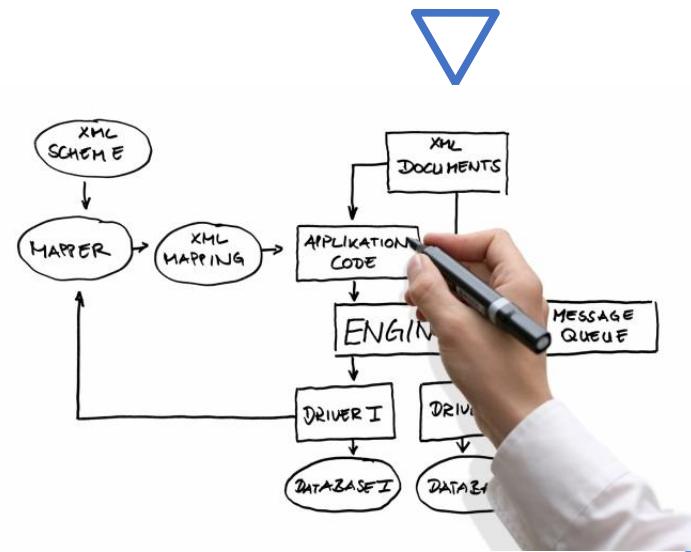


Assim como uma receita detalha os ingredientes e as etapas para preparar um prato, a arquitetura de software fornece uma estrutura e um guia para o desenvolvimento de um sistema de software, indicando como os componentes se relacionam e funcionam juntos.



O Arquiteto de Softwares

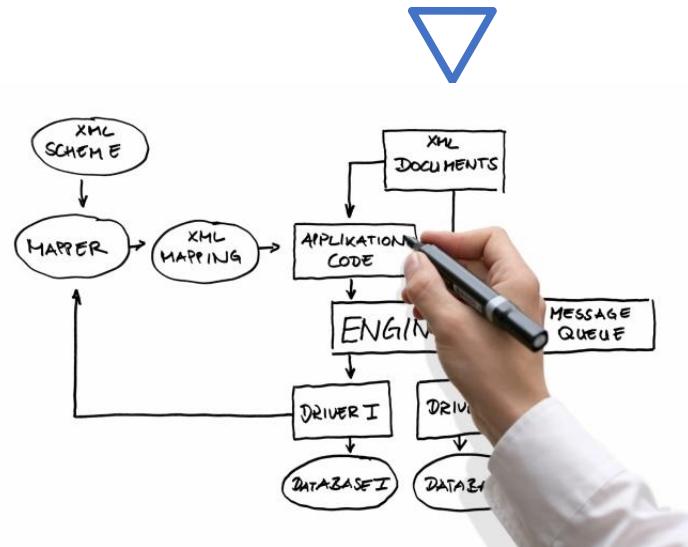
- O arquiteto de softwares é responsável pela **arquitetura de um sistema**, tomando decisões importantes sobre tecnologias e abordagens, estabelecendo padrões, resolvendo desafios técnicos complexos, supervisionando o desenvolvimento e colaborando estreitamente com a equipe.
- Sua função é garantir que o sistema seja **eficiente, escalável, de fácil manutenção e atenda aos requisitos do projeto**, orientando a equipe ao longo do ciclo de vida do software para atingir os objetivos do sistema.



<https://donmaclean.com/2012/02/18/my-role-the-software-architect/>

O Arquiteto de^oSoftwares

- Um arquiteto de softwares, em sua essência, é um programador. Embora **seu foco possa estar voltado para aspectos de alto nível**, eles não param de programar e lideram a equipe na busca por um design eficiente.
- Embora o **seu envolvimento na codificação possa não ser tão extenso como o de outros membros da equipe**, os arquitetos de software permanecem envolvidos nesta tarefa para compreender os desafios subjacentes e garantir a sua resolução adequada.



<https://donmaclennan.com/2012/02/18/my-role-the-software-architect/>

O Arquiteto de^oSoftwares

"Para que um desenvolvedor se torne um arquiteto, ele deve ser capaz de reconhecer quais elementos são importantes, identificando quais elementos podem causar sérios problemas se não forem controlados."

-Martin Fowler



<https://martinfowler.com/>

Aspectos importantes de uma boa arquitetura de software

1. **Escalabilidade:** Permite que o software seja escalável, o que significa que pode ser adaptado para lidar com um volume maior de dados ou usuários sem a necessidade de reescrever todo o sistema.
2. **Manutenibilidade:** Facilita a manutenção do software ao longo do tempo. Um design modular e bem organizado permite que alterações, correções de bugs e atualizações sejam feitas de forma eficiente e com menos risco de introdução de novos problemas.



Aspectos importantes de uma boa arquitetura de software

- 3. **Flexibilidade:** Permite incorporar novos recursos e se adaptar às mudanças nos requisitos do usuário.
- 4. **Reutilização:** Promove a reutilização de componentes e módulos, economizando tempo e recursos no desenvolvimento. Componentes reutilizáveis podem ser utilizados em diferentes partes do software ou até mesmo em projetos futuros.



Aspectos importantes de uma boa arquitetura de software

5. **Segurança:** A segurança é de vital importância para proteger a integridade dos dados e do sistema. A separação adequada de responsabilidades (SoC, *Separation of Concerns*) e o controle de acesso (AC, Access Control) são elementos fundamentais de uma arquitetura segura.
6. **Desempenho:** Influencia significativamente o desempenho do software. Um design eficiente pode otimizar o uso de recursos e acelerar a execução de tarefas críticas.



Aspectos importantes de uma boa arquitetura de software

7. **Tempo e custo:** Uma arquitetura sólida pode economizar tempo e custos no desenvolvimento. Ela evita reescritas frequentes de código e reduz a necessidade de solucionar problemas complexos e dispendiosos posteriormente no processo.

8. **Colaboração:** Uma arquitetura bem definida fornece uma base comum para as equipes de desenvolvimento colaborarem de forma eficiente. Cada membro da equipe entende como as partes do sistema se encaixam, facilitando a comunicação e a colaboração.



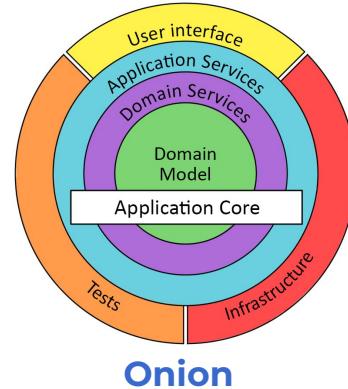
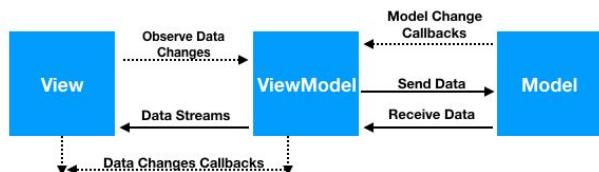
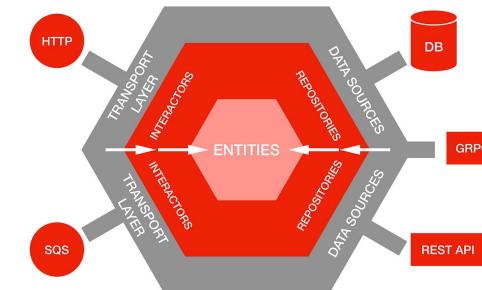
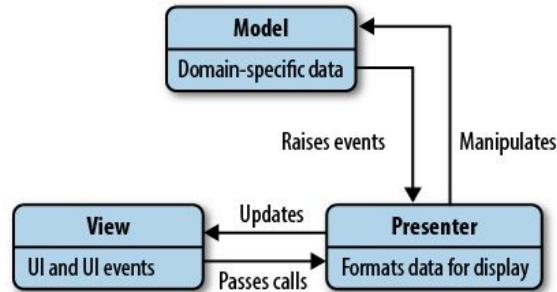
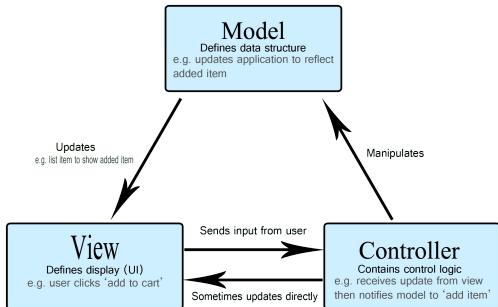
Os Padrões Arquitetônicos

- Os padrões de arquitetura são **soluções comprovadas e reutilizáveis para desafios comuns no projeto de sistemas de software**. Eles oferecem uma abordagem estruturada para organizar componentes e lógica, resultando em sistemas robustos e escaláveis.
- A adoção desses padrões é recomendada, pois **simplifica o projeto, melhora a qualidade e facilita a construção de sistemas** eficientes e de fácil manutenção. Além disso, **promovem comunicação e compreensão eficazes entre os membros da equipe** e as partes interessadas, fornecendo uma linguagem comum para descrever e discutir a arquitetura de software.



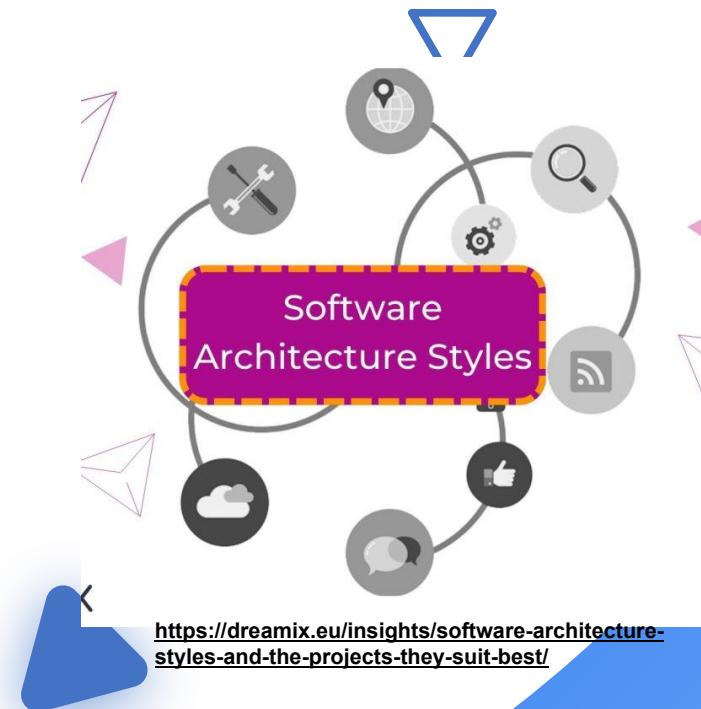
<https://www.decipherzone.com/blog-detail/software-architecture-patterns-type>

Alguns exemplos de Padrões Arquitetônicos

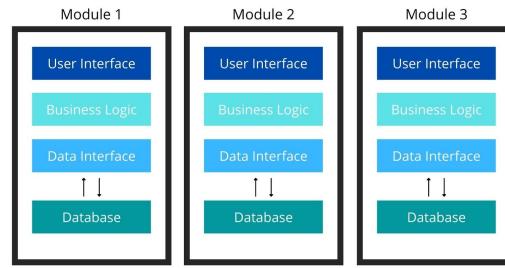
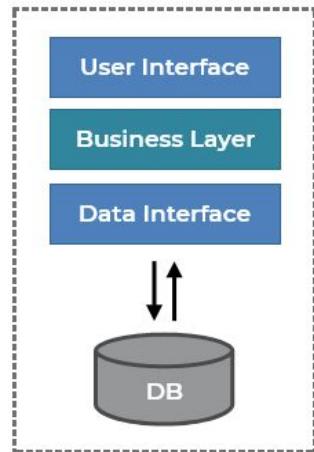


Os Estilos de Arquitetura

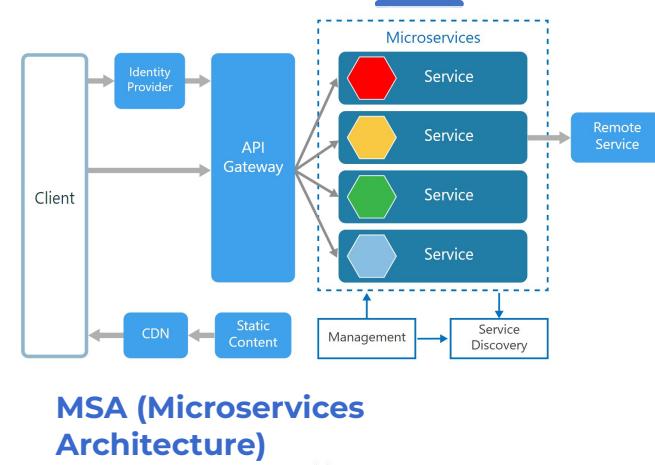
- Os estilos arquitetônicos são **diretrizes gerais e de alto nível para a organização de sistemas de software**. Eles representam abordagens arquitetônicas poderosas projetadas para resolver questões comuns de projeto.
- Em contrapartida, os padrões arquiteturais operam em um nível mais detalhado, concentrando-se na implementação e no design específico de componentes. Os estilos arquitetônicos, por outro lado, **são conceitos mais amplos que descrevem a estrutura geral de um sistema sem entrar em detalhes de implementação**.



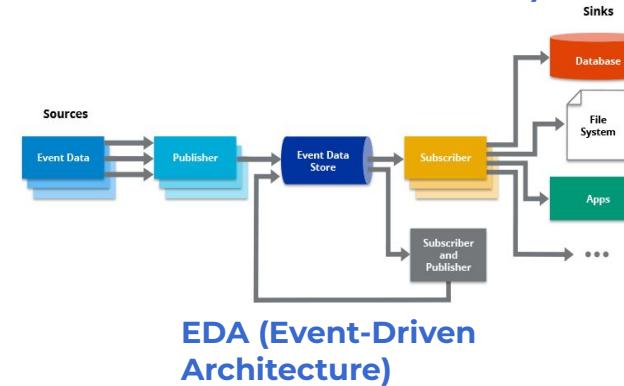
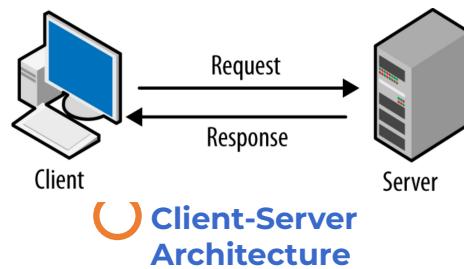
Alguns exemplos de Estilos de Arquitetura



Modular Monolith



MSA (Microservices Architecture)



EDA (Event-Driven Architecture)

Exemplo de escolha racional

- Uso carro para fins urbanos
- Sou casado e tenho 4 filhos
- Meu orçamento é limitado em R\$65.000,00
- Moro numa cidade quente



Exemplo de escolha racional



Exemplo de escolha racional



Princípios arquiteturais

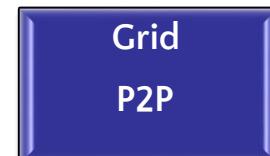
- Traduzem necessidades de negócio
- Servem como diretrizes para a arquitetura
- Riscos e restrições também podem ser condutores
- Exemplos:
 - Alta coesão e baixo acoplamento
 - Escalabilidade, Alta disponibilidade, Tolerância a falhas, Baixo tempo de resposta
 - Flexibilidade para mudanças futuras, Flexibilidade para mudanças em regras e processos
 - Integração com operadoras de cartão de crédito
 - Aplicação deve servir para várias empresas (multi empresa)
 - Segurança na Web
 - Mobilidade, Portabilidade
 - Neutralidade tecnológica, uso de tecnologias comprovadas
 - Limitar diversidade de tecnologias e fornecedores

Estilos Arquiteturais

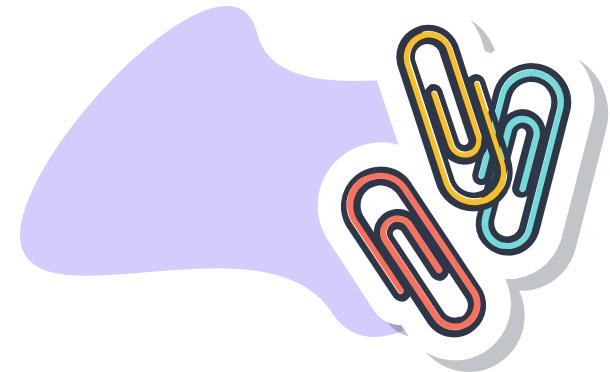
- ☐ Podem ocorrer combinações de estilos. Ou seja, uma aplicação pode ser construída sobre uma plataforma que comporta vários estilos arquiteturais.
- ☐ Consulte no apêndice os seguintes estilos clássicos:



- ☐ E em seguida os demais estilos arquiteturais:



Estilos Arquiteturais



Estilos Arquiteturais Clássicos

Cliente-
Servidor

Multicamadas
(N-layers)

Model-View-
Controller

Estilos Arquiteturais Clássicos

Cliente-
Servidor

Multicamadas
(N-layers)

Model-View-
Controller

Cliente-Servidor

□ Cliente

- Estabelece a conexão, envia mensagens para o servidor e aguarda mensagens de resposta.

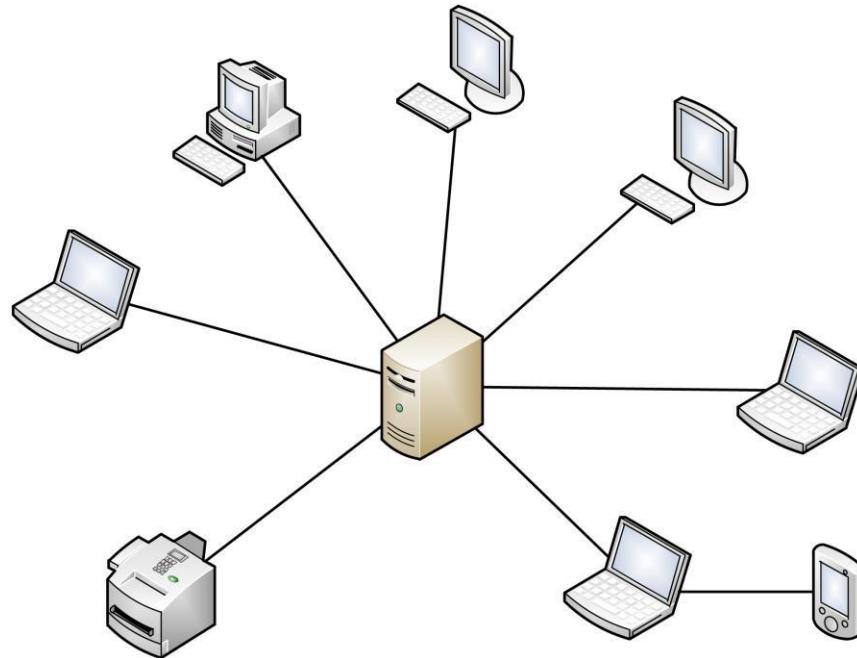
□ Servidor

- Aguarda mensagens, executa serviços e retorna resultados.



Cliente-Servidor

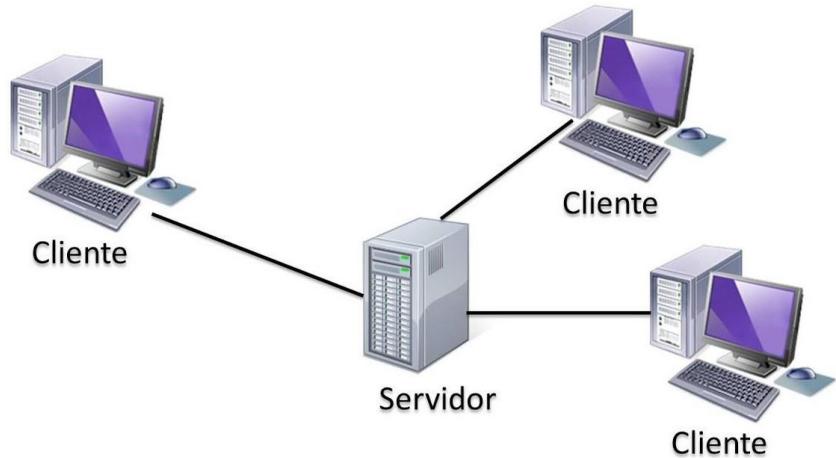
- Os primeiros sistemas cliente-servidor surgiram por causa de limitações nas arquiteturas de compartilhamento de arquivos
- Usavam um banco de dados central substituindo o servidor de arquivos
- Reduziu o tráfego de rede, uma vez que somente as consultas (queries e respostas) trafegavam, ao invés de arquivos



Cliente-Servidor

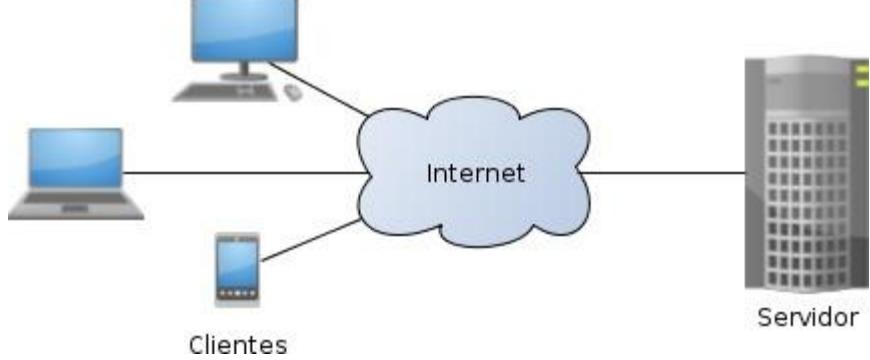
□ Arquitetura tradicional:

- Servidor de banco de dados físico local
- Clientes desktop instalados e configurados na máquina do cliente
- Reduziu o tráfego de rede, uma vez que somente as consultas (e suas respostas) trafegavam, ao invés de arquivos



□ Arquitetura atual:

- Navegador Web x Servidor Web
 - Clientes magros (browser) independente da plataforma do cliente (Mac, Win, Linux, etc.)
- Programa Java x Banco de Dados
- App Mobile x Servidor de Aplicação



Cliente-Servidor

□ Vantagens

- Utilização dos recursos do servidor sendo acessados pela rede
- Os maiores recursos estão apenas no servidor
- Escalabilidade, pois pode aumentar a capacidade computacional aumentando a quantidade de servidores
- Reduz tráfego da rede, visto que somente as consultas são feitas pela rede
- Manutenção centralizada, facilitando a evolução do software

□ Desvantagens

- Introduz complexidade
 - Do lado do servidor (alta disponibilidade, elasticidade, segurança)
 - Do lado do cliente (controle de configuração no caso de aplicações instaláveis e compatibilidade com browsers no caso de aplicações Web)
- Custos de comunicação (link adequado, segurança)

Estilos Arquiteturais Clássicos

Cliente-
Servidor

Multicamadas
(N-layers)

Model-View-
Controller

Multicamadas

□ Pontos positivos

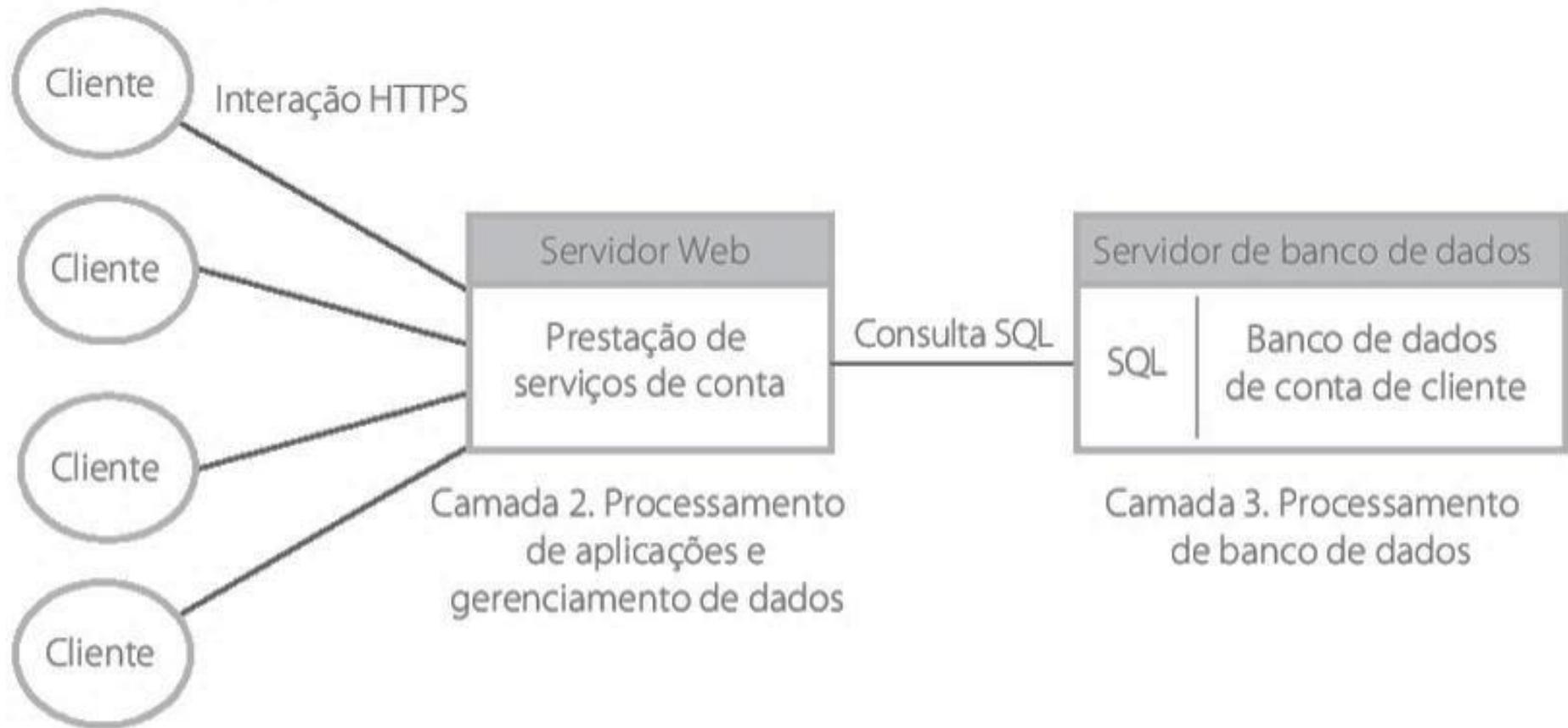
- Melhor organização do código, pois são separadas por afinidade de função
- Reuso das camadas, visto que possuem baixo acoplamento
 - Dependências tendem a permanecer “locais”
- Melhor manutenibilidade

□ Pontos negativos

- Exige código adicional para fazer a separação entre as camadas (interface bem definida), o que pode ser reduzido com uso de frameworks
- Cascateamento de alterações para as camadas superiores quando o comportamento de uma camada inferior muda, o que pode ser reduzido com uso de interfaces padronizadas

Multicamadas (cliente-servidor multicamada)

Camada 1. Apresentação



Divisão clássica

Camada de apresentação

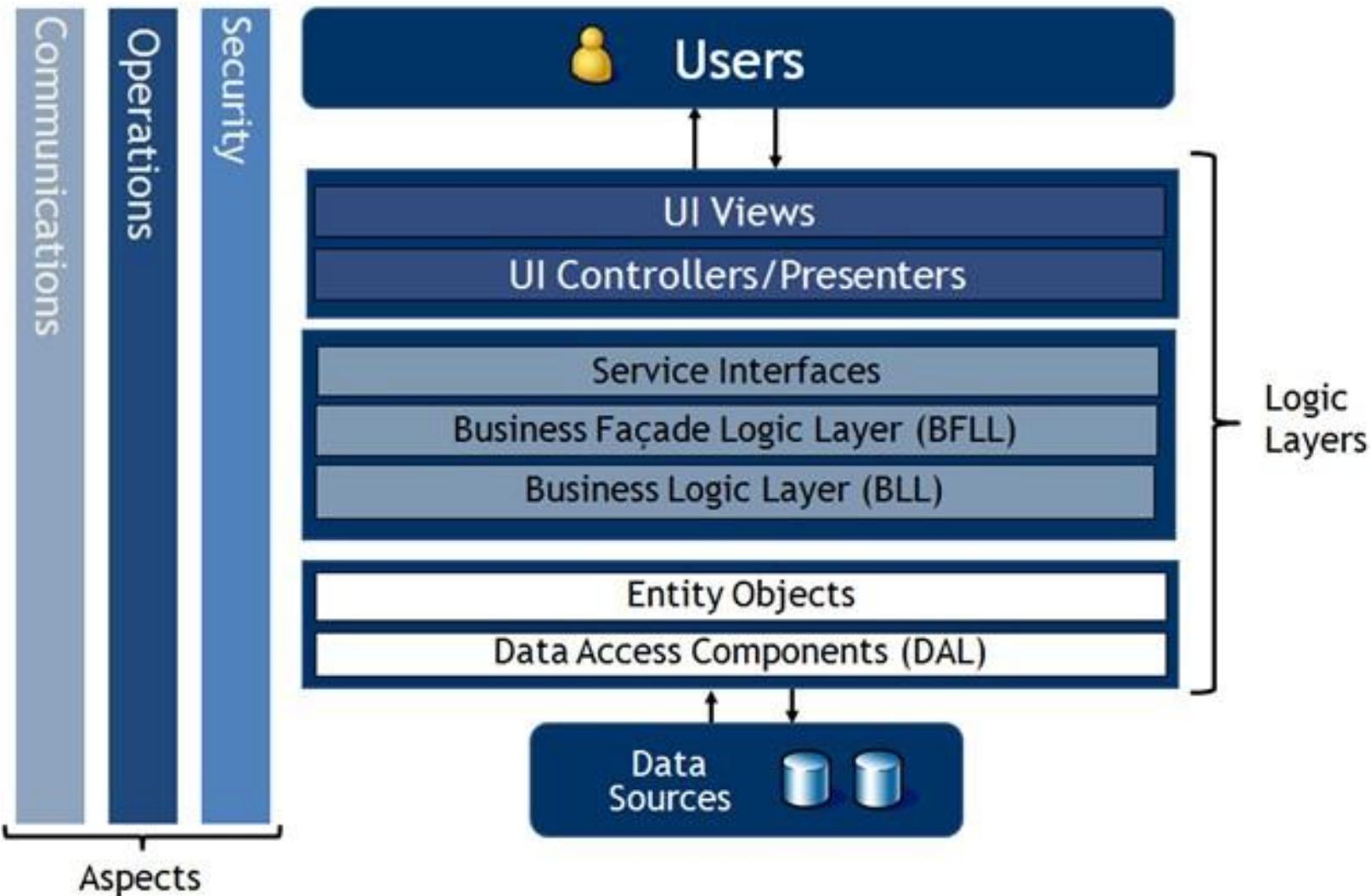
Camada de gerenciamento de dados

Camada de processamento de aplicação

Camada de banco de dados

Multi-Camadas (N-Camadas)

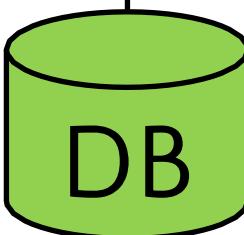
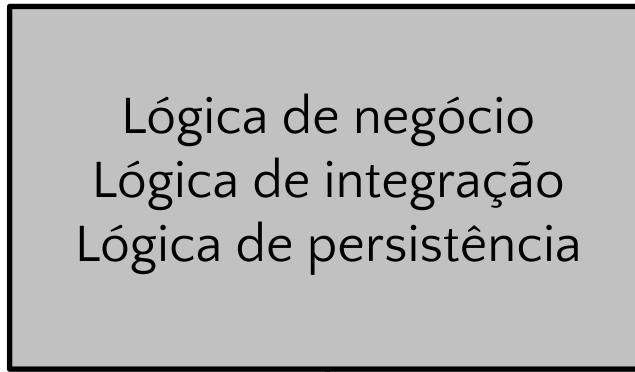
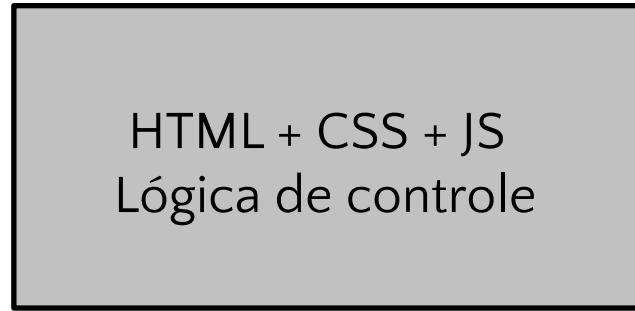
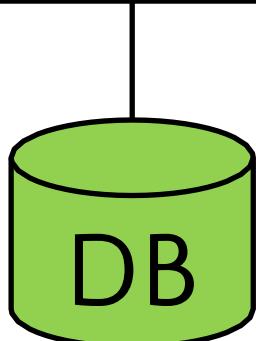
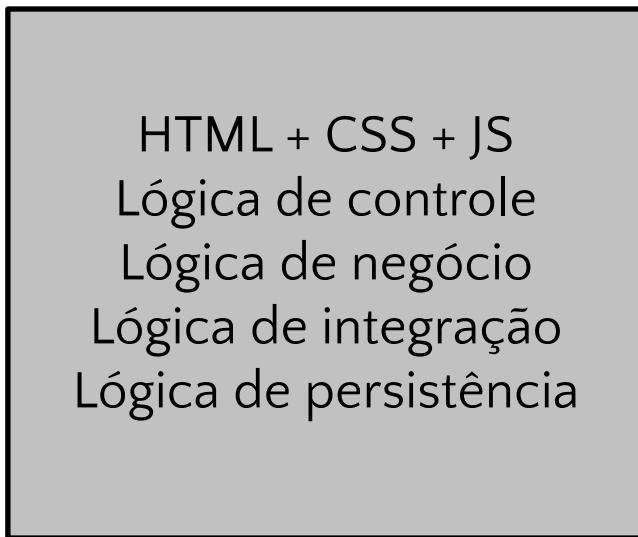
N-Layer Architecture



Multi-Camadas (N-Camadas)

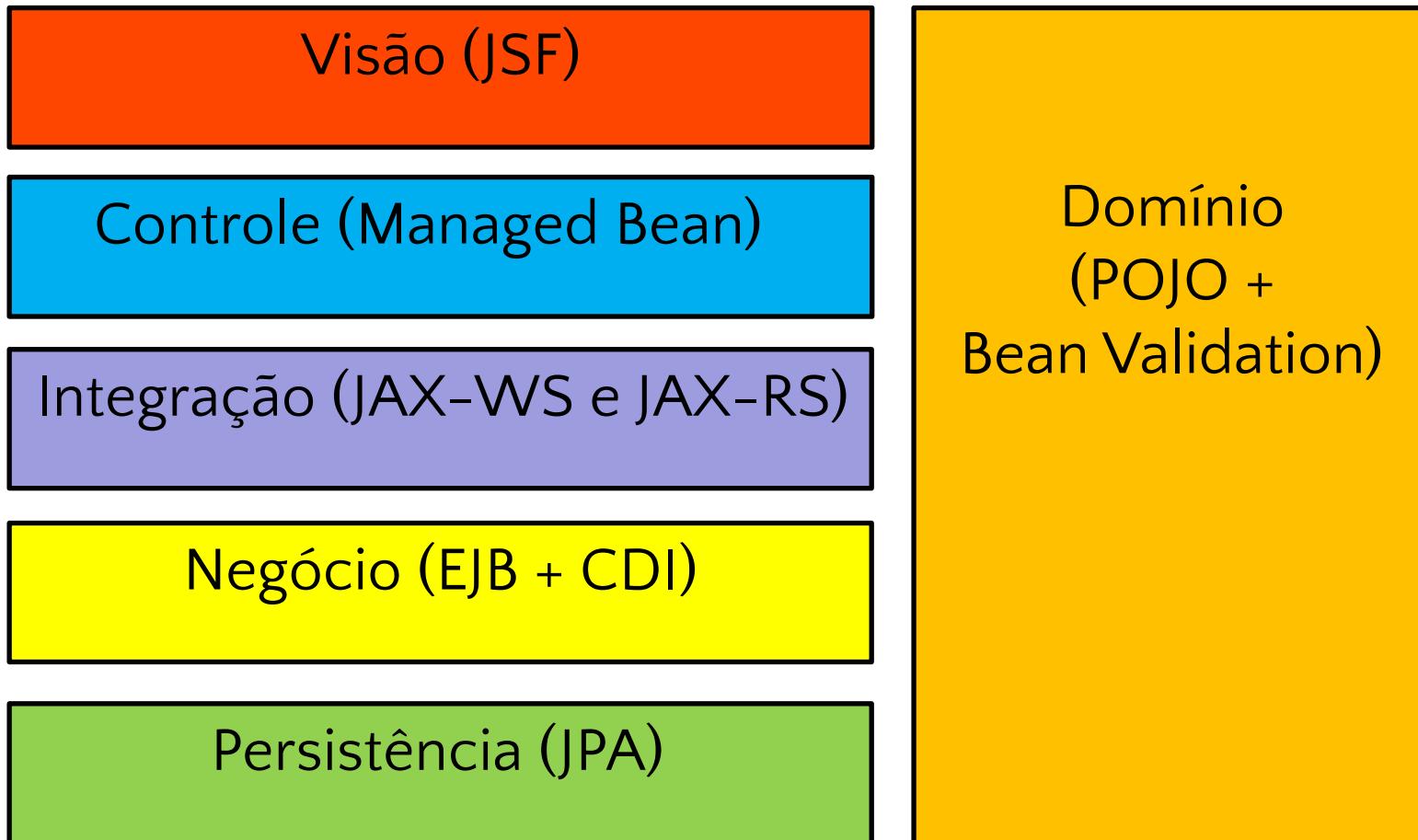
- Aplicações antigas sem divisão de camadas:

1 tier 2 tier



Multi-Camadas em Java EE

- O desenvolvimento de aplicações modernas utiliza o conceito de N-Camadas. Uma aplicação Java EE típica:



Estilos Arquiteturais Clássicos

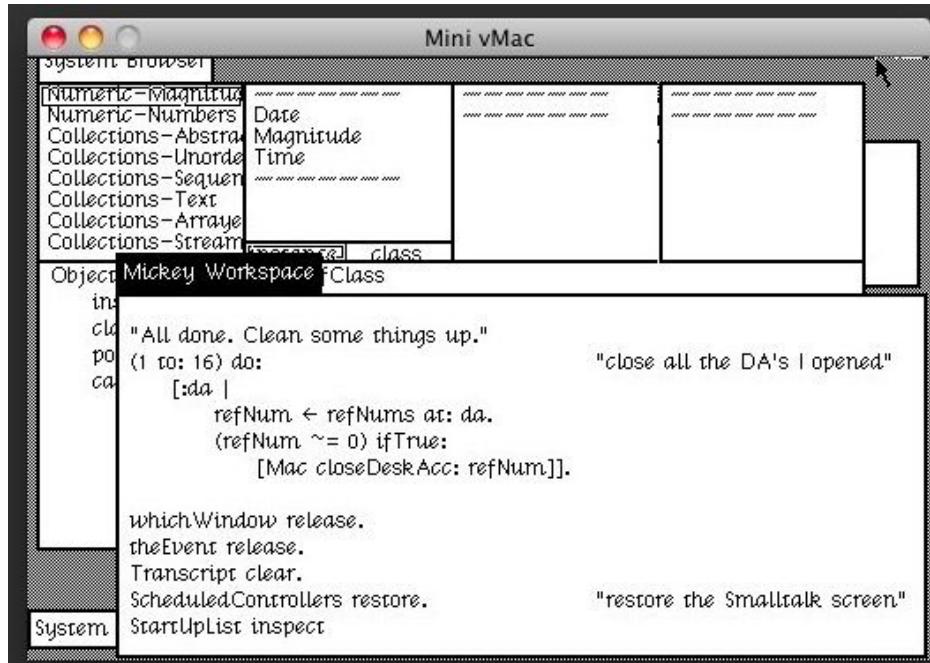
Cliente-
Servidor

Multicamadas
(N-layers)

Model-View-
Controller

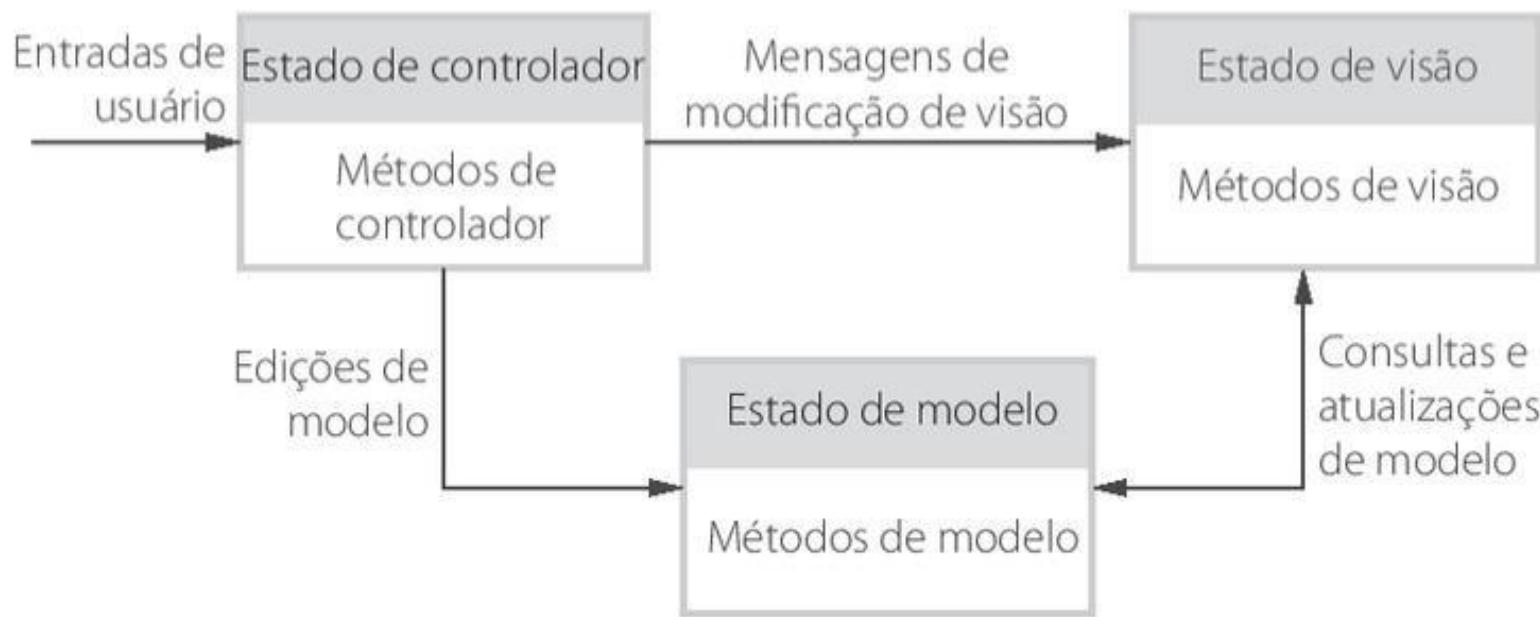
Arquitetura MVC

- Surgiu na década de 80, com a linguagem Smalltalk
- Proposta para implementar interfaces gráficas (GUIs)



Model-View-Controller (MVC)

- A aplicação é dividida em 3 componentes
 - **Model** – Contém uma representação do domínio, além da funcionalidade principal
 - **View** – Responsável pela visão, ou seja, exibe a informação aos usuários
 - **Controller** – Faz a gestão bidirecional entre entrada e saída do usuário, deixando o modelo transparente
 - **Interface do usuário** = View + Controller

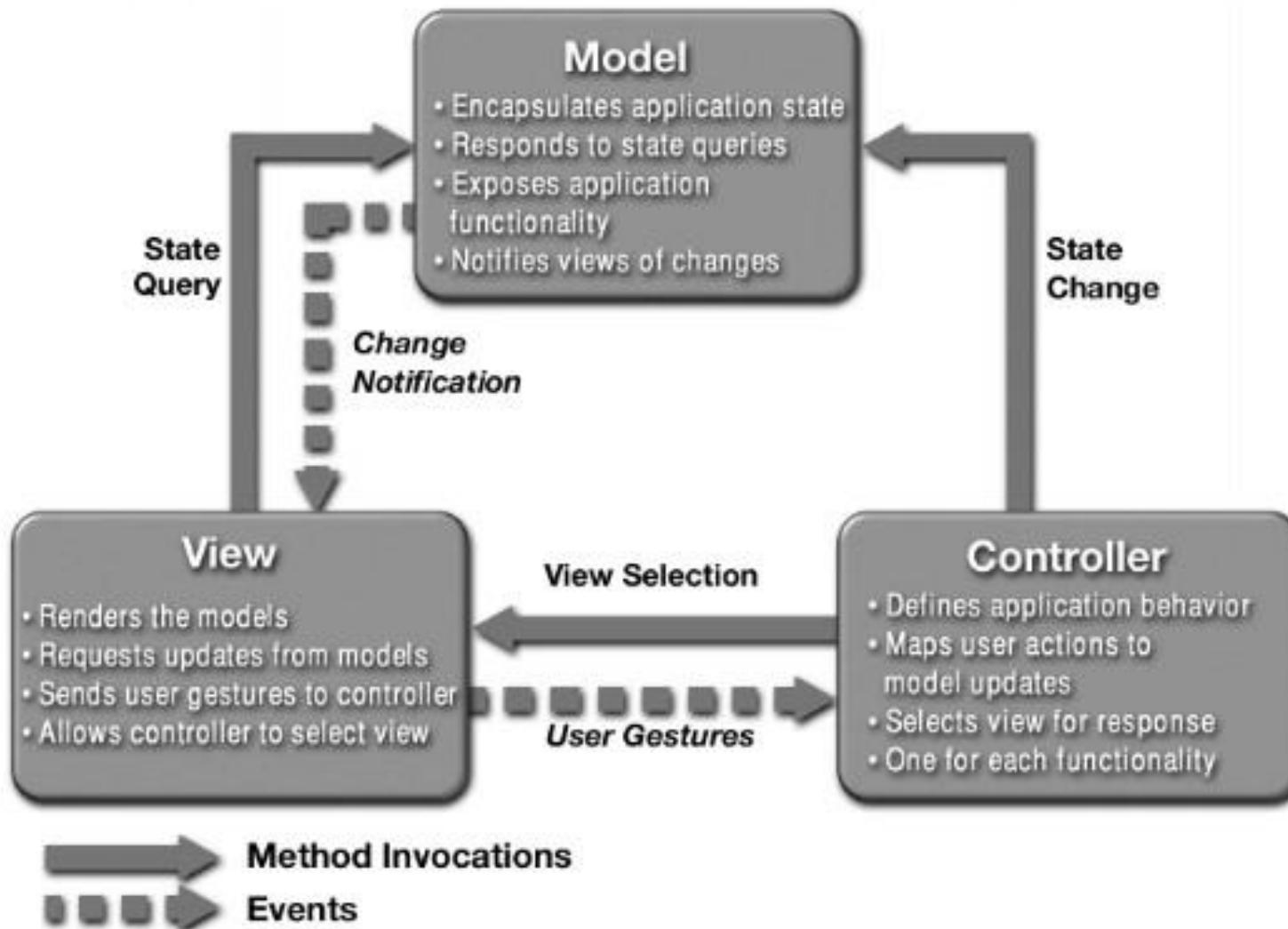


Explicando melhor...

- O **Modelo** representa o estado, estrutura e comportamento dos dados que estão sendo vistos e manipulados pelo usuário. É normalmente representado por entidades com atributos e comportamento (Domain Model). O modelo não possui link direto com a Visão, mas um link indireto através do padrão Observer. É dessa forma que o Modelo notifica a visão que os dados foram modificados.
- A **Visão** é um componente que lida com a saída de dados, é a representação visual do Modelo, sendo realizado através de telas e componentes de tela. Possui um link direto com o Modelo.
- O **Controle** é um componente que responde às entradas de dados, tais como eventos e formulários. Sua responsabilidade é formar uma ponte entre o usuário e a aplicação. Possui um link direto com o Modelo.

Model-View-Controller (MVC)

□ Funcionamento



Model-View-Controller (MVC)

□ Pontos positivos

- Melhor organização do código, facilitando a gestão e manutenção
- Aproveitamento de camadas, especialmente a de modelo
- Múltiplas “views” de um mesmo modelo “views” sincronizadas

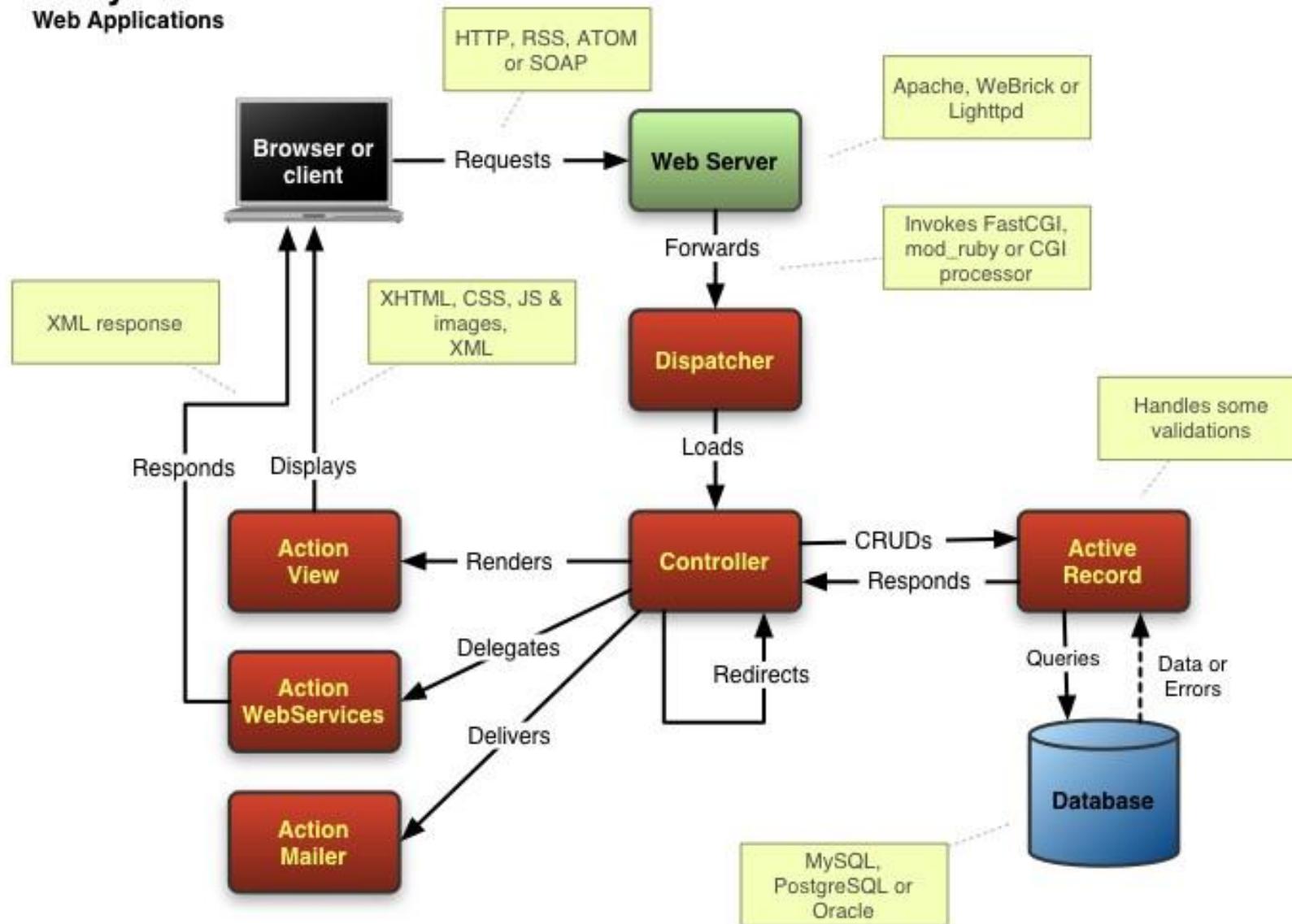
□ Pontos negativos

- Aumento da complexidade no desenvolvimento, o que pode ser reduzido com uso de um framework
- “*Controllers*” e “*Views*” tendem a ser bastante acoplados

MVC in RoR

Ruby on Rails

Web Applications



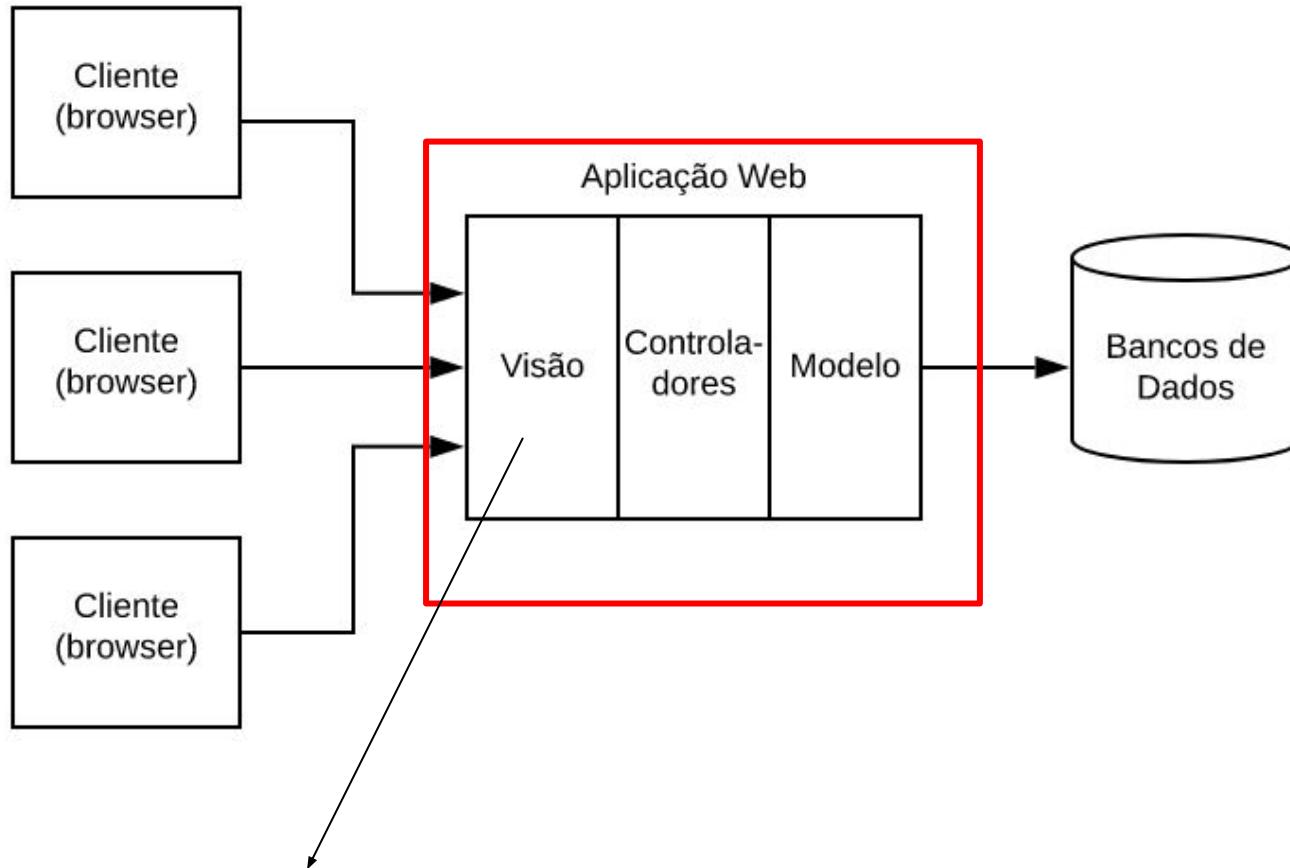
MVC nos dias de hoje

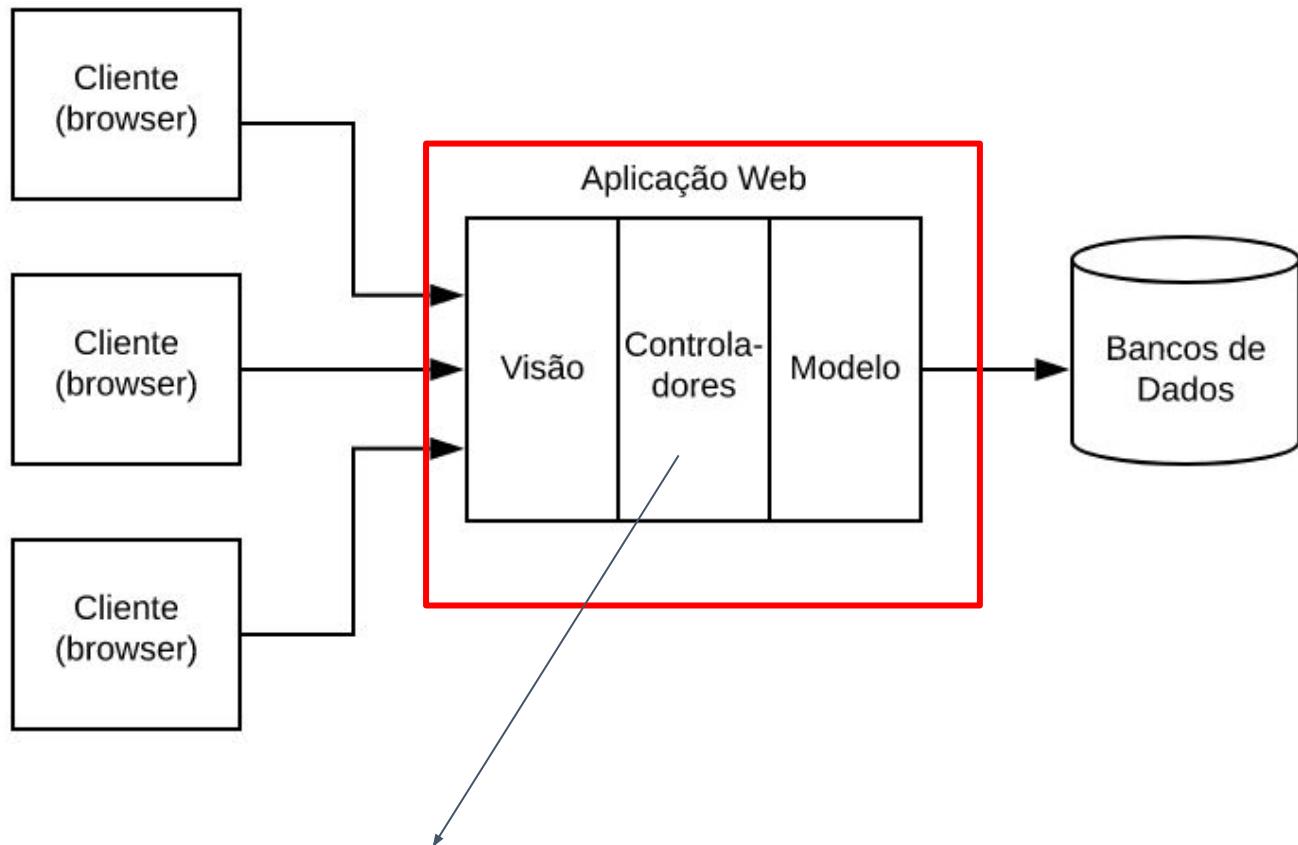
- MVC Web
- Single Page Applications

MVC Web

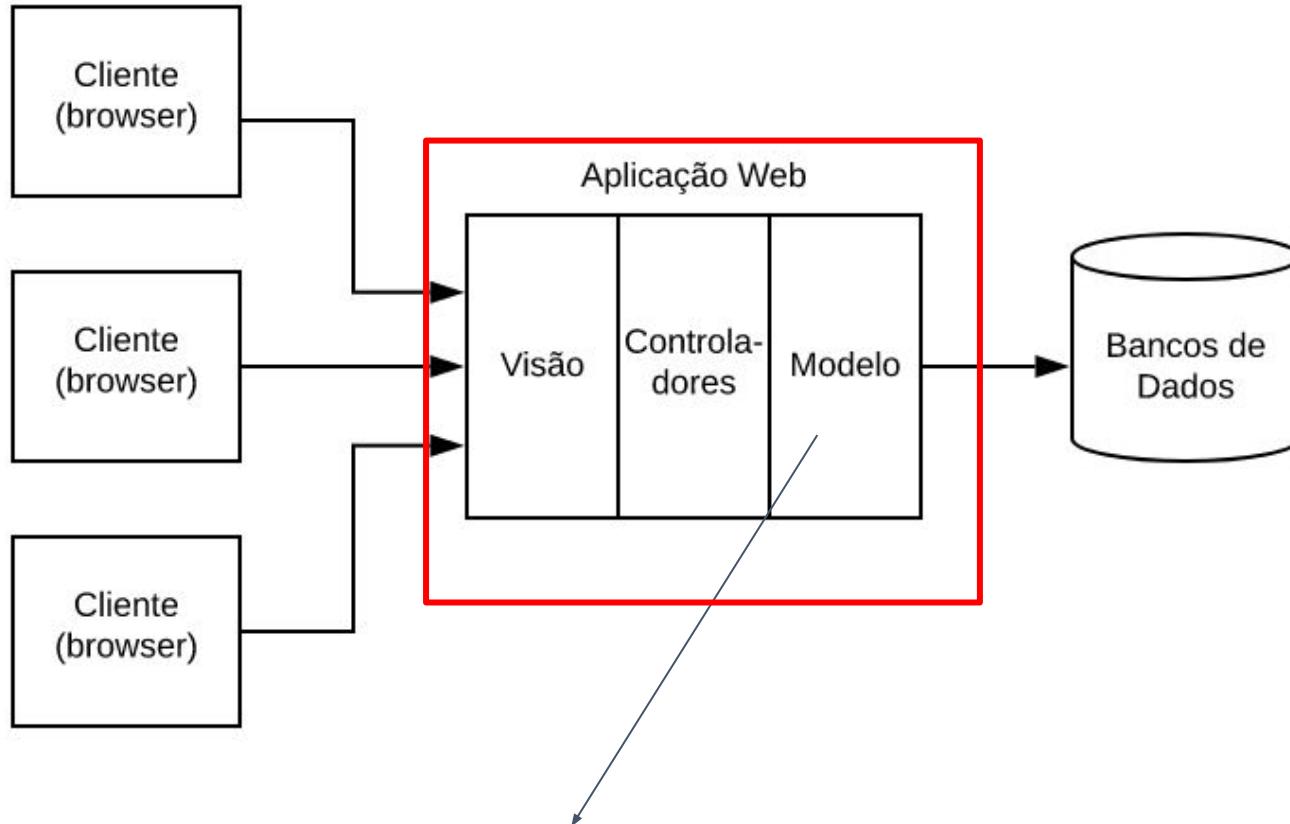
MVC Web

- Adaptação de MVC para Web, ou seja, sistemas distribuídos
- Usando Ruby on Rails, Django, Spring, PHP Laravel, etc





Recebem dados de entrada e fornecem
informações para páginas de saída



Lógica da aplicação (regras de negócio) e
fazem a interface com o banco de dados

Exemplo muito simples de um sistema MVC Web

<https://replit.com/@engsoftmoderna/ExemploArquiteturaMVC>

Este exemplo não usa nenhum framework e possui uma interface Web muito simples, tudo por motivos didáticos

Controlador

```
public class ControladorPesquisaLivros {  
    ...  
    public void start() {  
        ...  
        get("/", (req, res) -> {  
            res.redirect("index.html");  
            return null;  
        }) ;  
        ...  
    }  
}
```

Browser (index.html)

Biblioteca MVC

Pesquisa de Livros

Informe o nome do autor

Nomes válidos: valente, fowler e gof

Controlador

```
public class ControladorPesquisaLivros {  
    ServicoPesquisaLivros pesq;  
    PaginaDadosLivro pagina;  
    ...  
    public void start() {  
        ...  
        get("/pesquisa", (req, res) -> {  
            String autor = req.queryParams("autor");  
            Livro livro = pesq.pesquisaPorAutor(autor);  
            return pagina.exibeLivro(livro.getTitulo(),  
                livro.getAutor(), livro.getISBN());  
        } );  
        ...  
    }  
}
```

Modelo

```
public class ServicoPesquisaLivros {  
  
    public Livro pesquisaPorAutor(String autor) {  
        try(Connection con = DriverManager.getConnection(...)) {  
            String query = "select * from livros where autor = ?";  
            PreparedStatement stmt = con.prepareStatement(query);  
            stmt.setString(1, autor);  
            ResultSet rs = stmt.executeQuery();  
            String isbn = rs.getString("isbn");  
            String titulo = rs.getString("titulo");  
            return new Livro(isbn, autor, titulo);  
        } catch (SQLException e) {  
            System.out.println(e.getMessage());  
            return null;  
        }  
    }  
}
```

Controlador

```
public class ControladorPesquisaLivros {  
    ServicoPesquisaLivros pesq;  
    PaginaDadosLivro pagina;  
    ...  
    public void start() {  
        ...  
        get("/pesquisa", (req, res) -> {  
            String autor = req.queryParams("autor");  
            Livro livro = pesq.pesquisaPorAutor(autor);  
            return pagina.exibeLivro(livro.getTitulo(),  
                livro.getAutor(), livro.getISBN());  
        } );  
        ...  
    }  
}
```

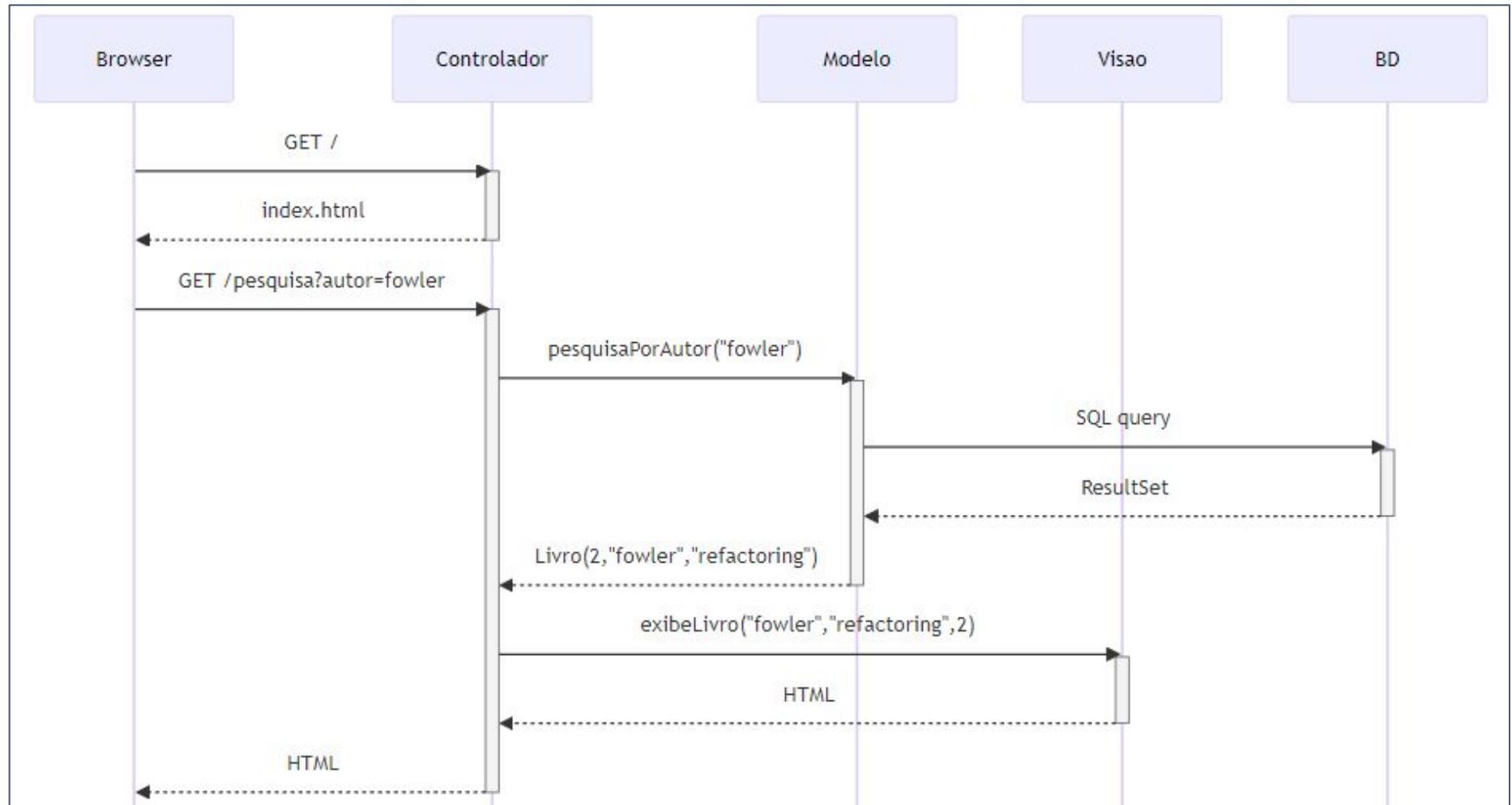
Visão

```
public class PaginaDadosLivro {  
  
    public String exibeLivro(String titulo, String autor, String isbn) {  
        String res = "<h4> Dados do Livro Pesquisado </h4>";  
        res += "<ul>";  
        res += "<li> Título: " + titulo + " </li>";  
        res += "<li> Autor: " + autor + " </li>";  
        res += "<li> ISBN: " + isbn + " </li>";  
        res += "</ul>";  
        return res;  
    }  
}
```

Browser

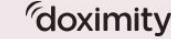
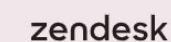
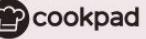
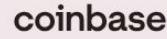
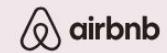
Dados do Livro Pesquisado

- Título: Refactoring
- Autor: fowler
- ISBN: 2



MVC Frameworks continuam sendo relevantes!

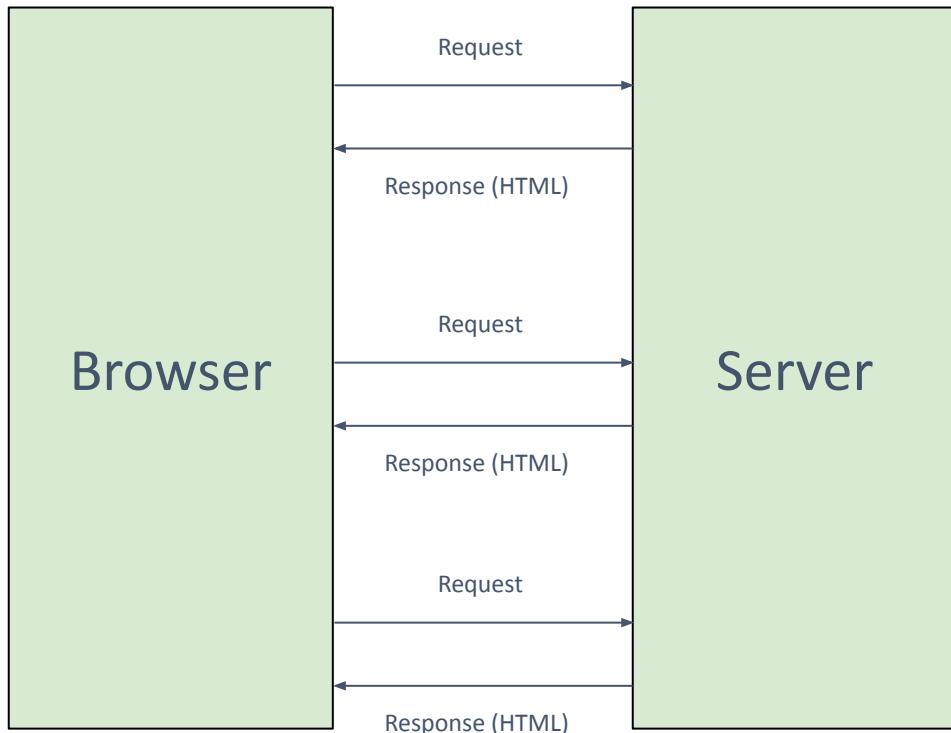
Over the past two decades, Rails has taken countless companies to millions of users and billions in market valuations.



<https://rubyonrails.org> (April 2024)

Single Page Applications (SPAs)

Aplicações Web Tradicionais



Problema: interface menos responsiva

Multiple Page Applications (MPAs)

Single Page Applications

- Roda no browser; logo, mais independente do servidor
- "Menos burra": manipula sua interface e armazena dados
- Pode acessar o servidor para buscar mais dados
- Exemplo: GMail, Google Docs, Facebook, Figma, etc
- Implementadas em JavaScript, usando React, Vue, etc

```
import { useState } from 'react';

export default function MyApp() {
  return (
    <div>
      <h1>Counters that update separately</h1>
      <MyButton />
      <MyButton />
    </div>
  );
}

function MyButton() {
  const [count, setCount] = useState(0);

  function handleClick() {
    setCount(count + 1);
  }

  return (
    <button onClick={handleClick}>
      Clicked {count} times
    </button>
  );
}
```



Counters that update separately

Clicked 6 times
Clicked 4 times

useState() retorna dois valores: (1) uma variável com o estado corrente e (2) uma função que você deve chamar toda vez que precisar alterar este estado.

Fonte: <https://react.dev/learn>



Exemplo: Aplicação Simples usando Vue.js

```
<h3>Uma Simples SPA</h3>

<div id="ui">
  Temperatura: {{ temperatura }}
  <p><button v-on:click="incTemperatura">Incrementa
  </button></p>
</div>
```

```
<script>
var model = new Vue({
  el: '#ui',
  data: {
    temperatura: 60
  },
  methods: {
    incTemperatura: function() {
      this.temperatura++;
    }
})
</script>
```

Interface (Web)
HTML

```
<h3>Uma Simples SPA</h3>

<div id="ui">
  Temperatura: {{ temperatura }}
  <p><button v-on:click="incTemperatura">Incrementa
  </button></p>
</div>

<script>
var model = new Vue({
  el: '#ui',
  data: {
    temperatura: 60
  },
  methods: {
    incTemperatura: function() {
      this.temperatura++;
    }
})
</script>
```

Uma Simples SPA

Temperatura: 60

Incrementa

A diagram illustrating a Single Page Application (SPA) structure. On the left, the code defines an HTML template with a heading and a div element containing a temperature display and an increment button. It also defines a Vue.js instance 'model' with a 'temperatura' data property set to 60 and an 'incTemperatura' method that increments the temperature. On the right, the resulting application interface is shown in a box. It displays the heading 'Uma Simples SPA'. Below it, the temperature is shown as 'Temperatura: 60' next to a text input field containing the value '60'. To the right of the input is a button labeled 'Incrementa'. Red arrows from the code point to each corresponding part of the application interface: one arrow points from the 'h3' tag to the heading, another from the 'Temperatura:' text to the input field, and a third from the 'Incrementa' button to the button itself.

```
<h3>Uma Simples SPA</h3>

<div id="ui">
  Temperatura: {{ temperatura }}
  <p><button v-on:click="incTemperatura">Incrementa
  </button></p>
</div>
```

```
<script>
var model = new Vue({
  el: '#ui',
  data: {
    temperatura: 60
  },
  methods: {
    incTemperatura: function() {
      this.temperatura++;
    }
})
</script>
```

Modelo

<h3>Uma Simples SPA</h3>

```
<div id="ui">
  Temperatura: {{ temperatura }}
  <p><button v-on:click="incTemperatura">Incrementa
  </button></p>
</div>
```

```
<script>
var model = new Vue({
  el: '#ui',
  data: {
    temperatura: 60
  },
  methods: {
    incTemperatura: function() {
      this.temperatura++;
    }
})
</script>
```

Modelo

Dados

Métodos

```
<h3>Uma Simples SPA</h3>

<div id="ui">
  Temperatura: {{ temperatura }}
  <p><button v-on:click="incTemperatura">Incrementa
  </button></p>
</div>

<script>
var model = new Vue({
  el: '#ui',
  data: {
    temperatura: 60
  },
  methods: {
    incTemperatura: function() {
      this.temperatura++;
    }
  }
)
</script>
```

```
<h3>Uma Simples SPA</h3>

<div id="ui">
  Temperatura: {{ temperatura }}
  <p><button v-on:click="incTemperatura">Incrementa
  </button></p>
</div>

<script>
var model = new Vue({
  el: '#ui',
  data: {
    temperatura: 60
  },
  methods: {
    incTemperatura: function() {
      this.temperatura++;
    }
  }
)
</script>
```

```
<h3>Uma Simples SPA</h3>

<div id="ui">
  Temperatura: {{ temperatura }}
  <p><button v-on:click="incTemperatura">Incrementa
  </button></p>
</div>

<script>
var model = new Vue({
  el: '#ui',
  data: {
    temperatura: 60
  },
  methods: {
    incTemperatura: function() {
      this.temperatura++;
    }
  }
)
</script>
```

Resumo

MVC Tradicional
(Smalltalk): aplicações
gráficas, pré-Web



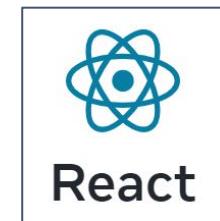
MVC Web: adaptação de
MVC para Web
(fullstack)



django



SPA: adaptação de
MVC para uso no
front-end

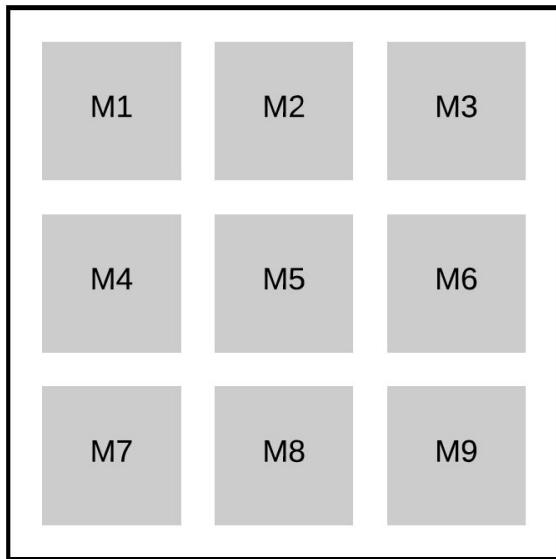


Microserviços

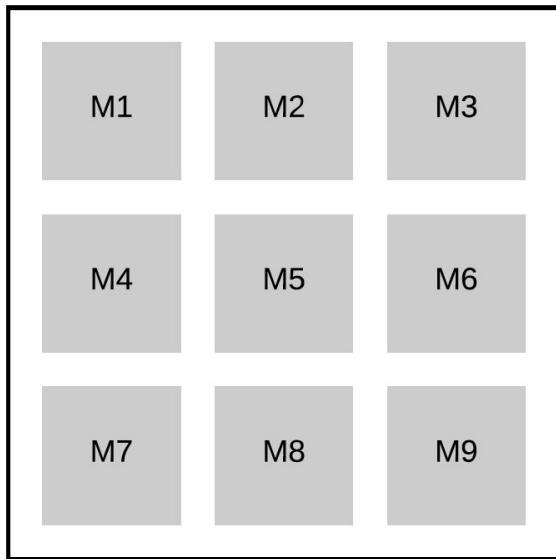
Microserviços

- Módulos (ou conjuntos de módulos) viram processos independentes em tempo de execução
- Esses módulos são menores do que de um monolito
- Daí o nome **microserviço**

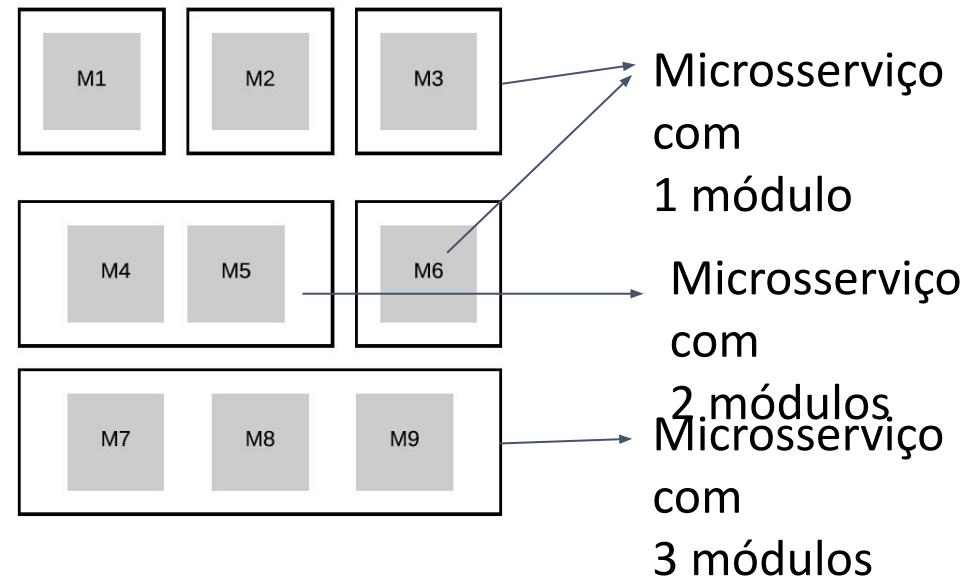
Arquitetura Monolítica



Arquitetura Monolítica



Arquitetura baseada em Microsserviços



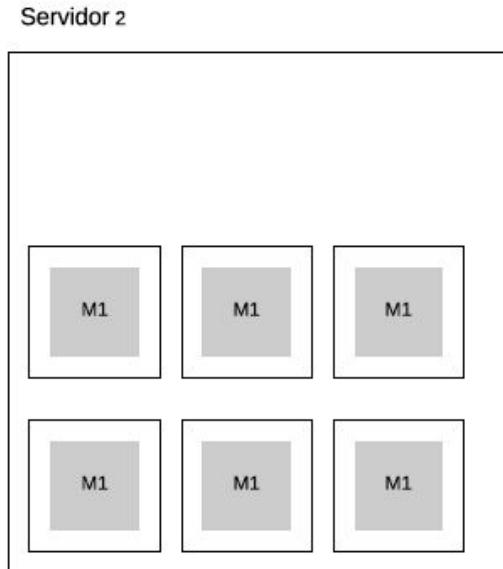
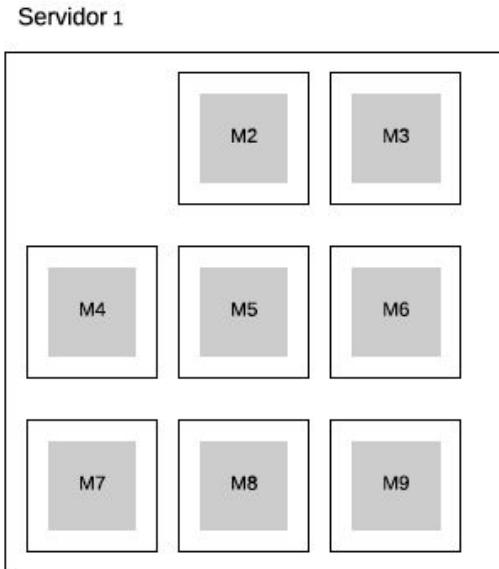
microsserviço = processo
(run-time, sistema
operacional)

Explicando de outra maneira

- Suponha um sistema com n endpoints (ou rotas)
- Monolito: todas as n rotas em um mesmo processo
- Microsserviços: rotas divididas em múltiplos processos

Vantagem #1: Escalabilidade

- Pode-se escalar apenas o módulo com problema de desempenho



Só contém instâncias de M1, pois apenas ele é o gargalo de desempenho

Vantagem #2: Flexibilidade para Releases

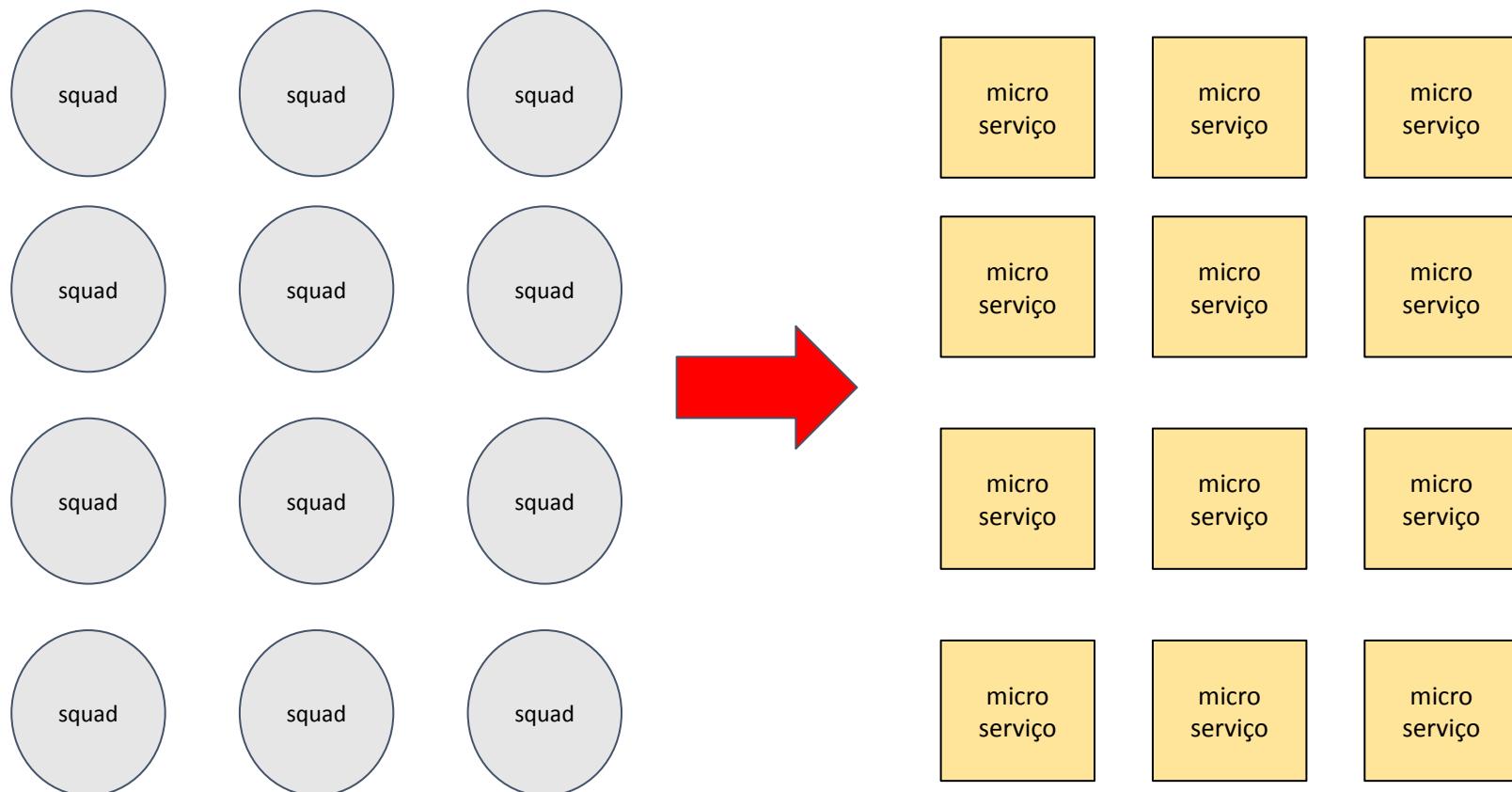
- Times ganham autonomia para colocar microsserviços em produção
- Processo = espaço de endereçamento próprio
- Chances de interferências entre processos são menores

Outras vantagens

- Tecnologias diferentes
- Falhas parciais. Exemplo: apenas o sistema de recomendação pode ficar fora do ar

Lei de Conway (1968)

- Arquitetura de software espelha a arquitetura da organização



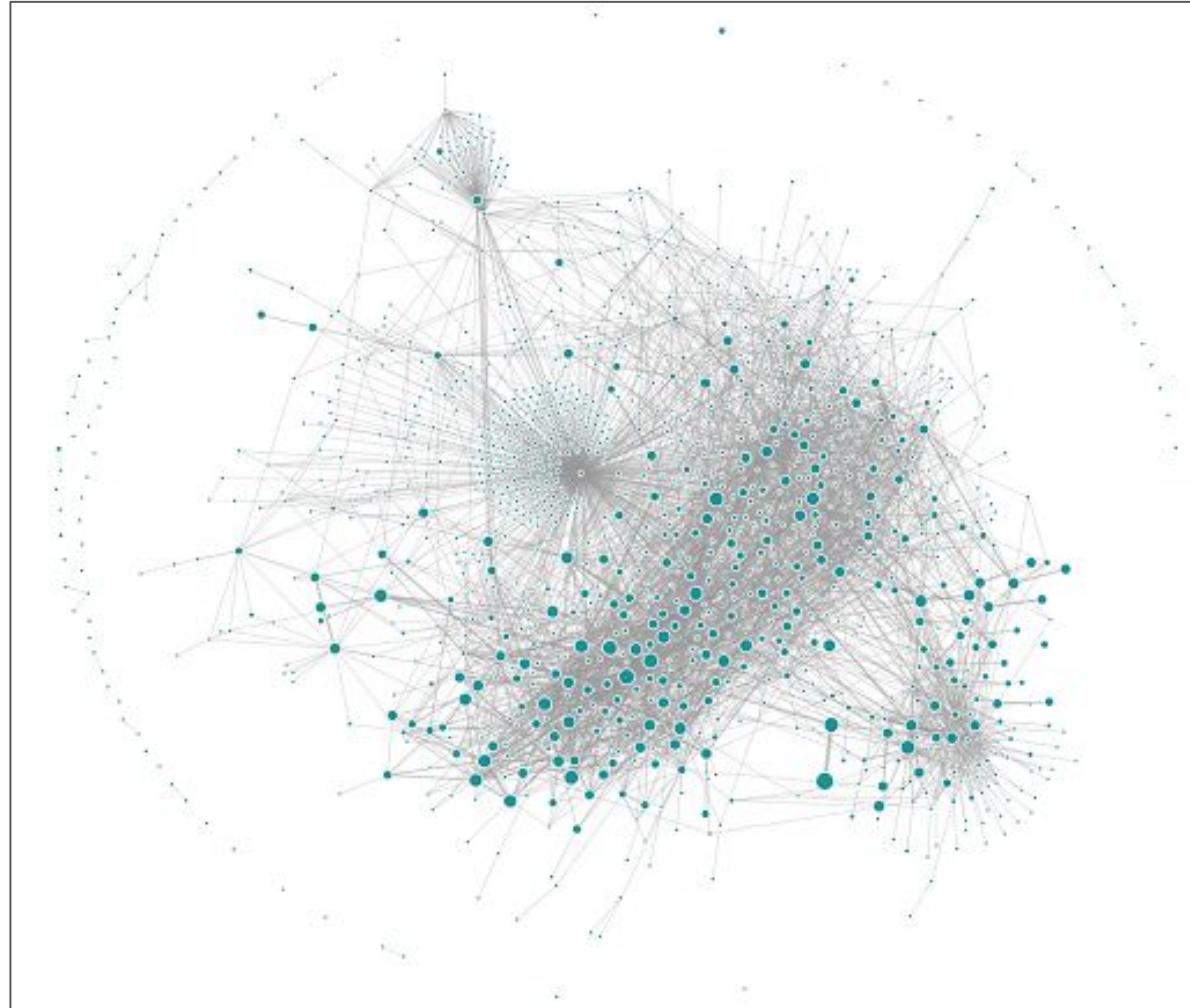
Quem usa microsserviços?

- Grandes empresas como Netflix, Amazon, Google, etc

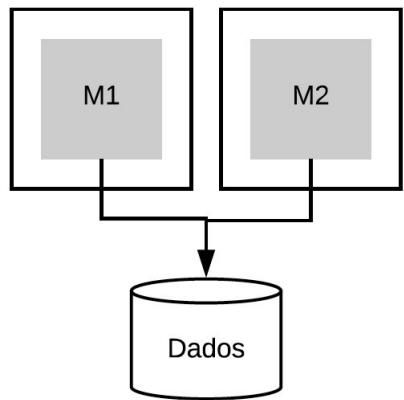


Cada nodo é um microsserviço

Exemplo: Uber (~2018)

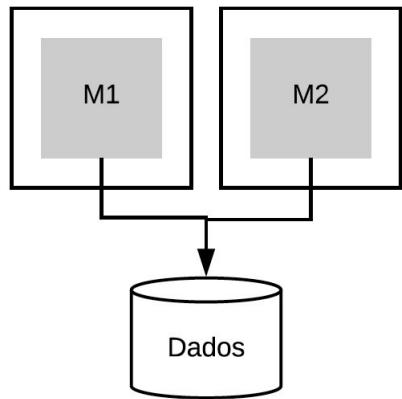


Gerenciamento de Dados com Microsserviços

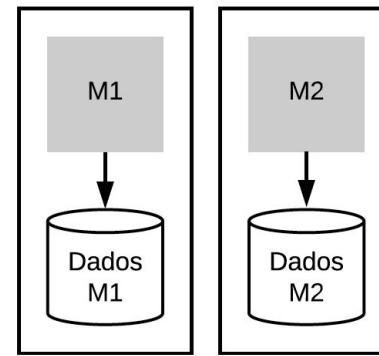


Arquitetura que **não** é
recomendada. Motivo: aumenta
acoplamento entre M1 e M2

Gerenciamento de Dados com Microsserviços



Arquitetura que não é recomendada. Motivo: aumenta acoplamento entre M1 e M2



Arquitetura recomendada.
Motivo: não existe acoplamento de dados entre M1 e M2. Logo, M1 e M2 podem evoluir de modo independente.
Se M1 precisar usar serviços de M2 (ou vice-versa), isso deve ocorrer via interfaces

Quando não usar microsserviços?

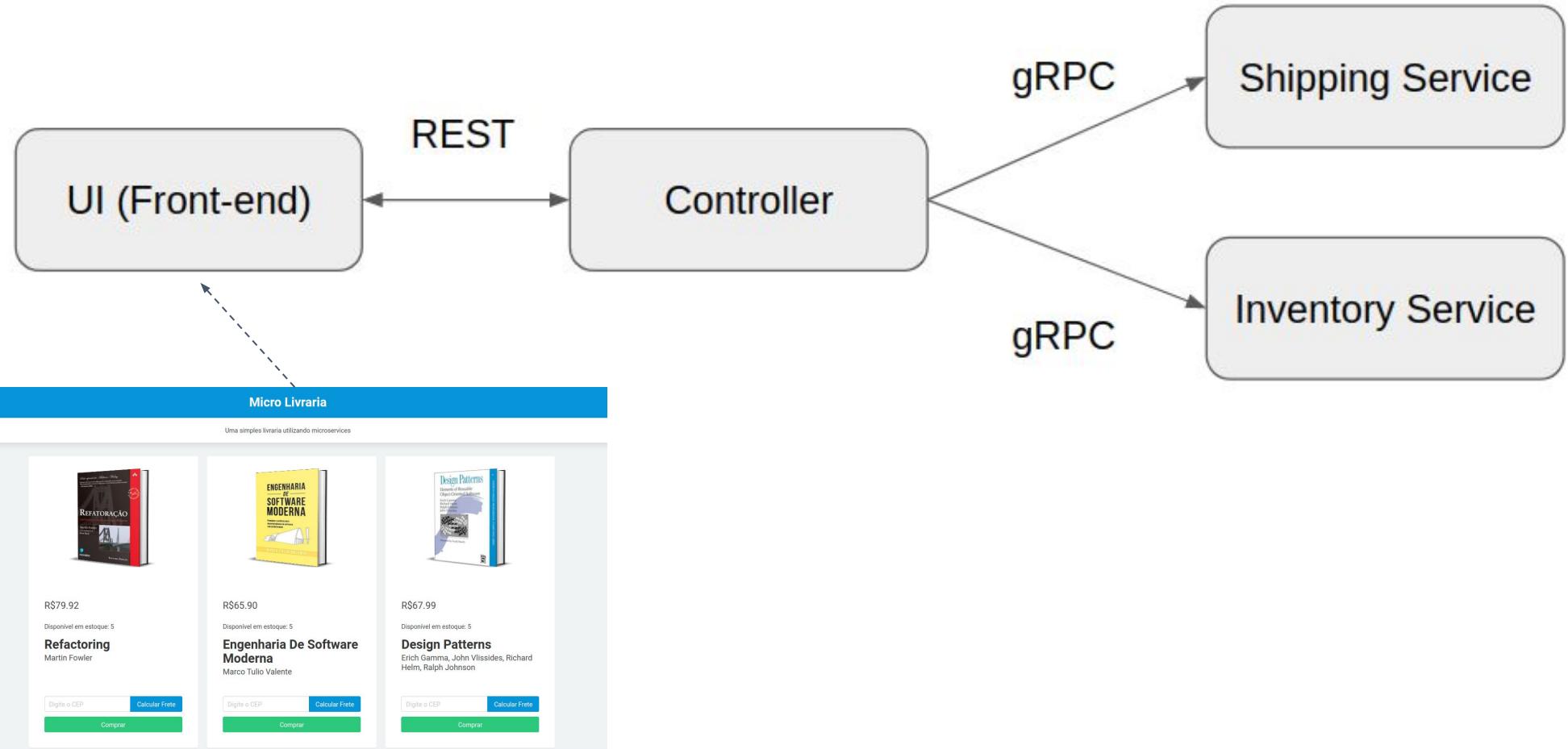
- Arquitetura com microsserviços é mais complexa
 - Sistema distribuído (gerenciar centenas de processos)
 - Latência (comunicação é via rede)
 - Transações distribuídas

Recomendação: começar com monolitos

- É migrar para microsserviços se:
 - Monolito apresentar problemas de desempenho
 - Monolito estiver atrasando as releases
- Migração pode ser gradativa...

Roteiro prático sobre microsserviços

<https://github.com/aserg-ufmg/micro-livraria>

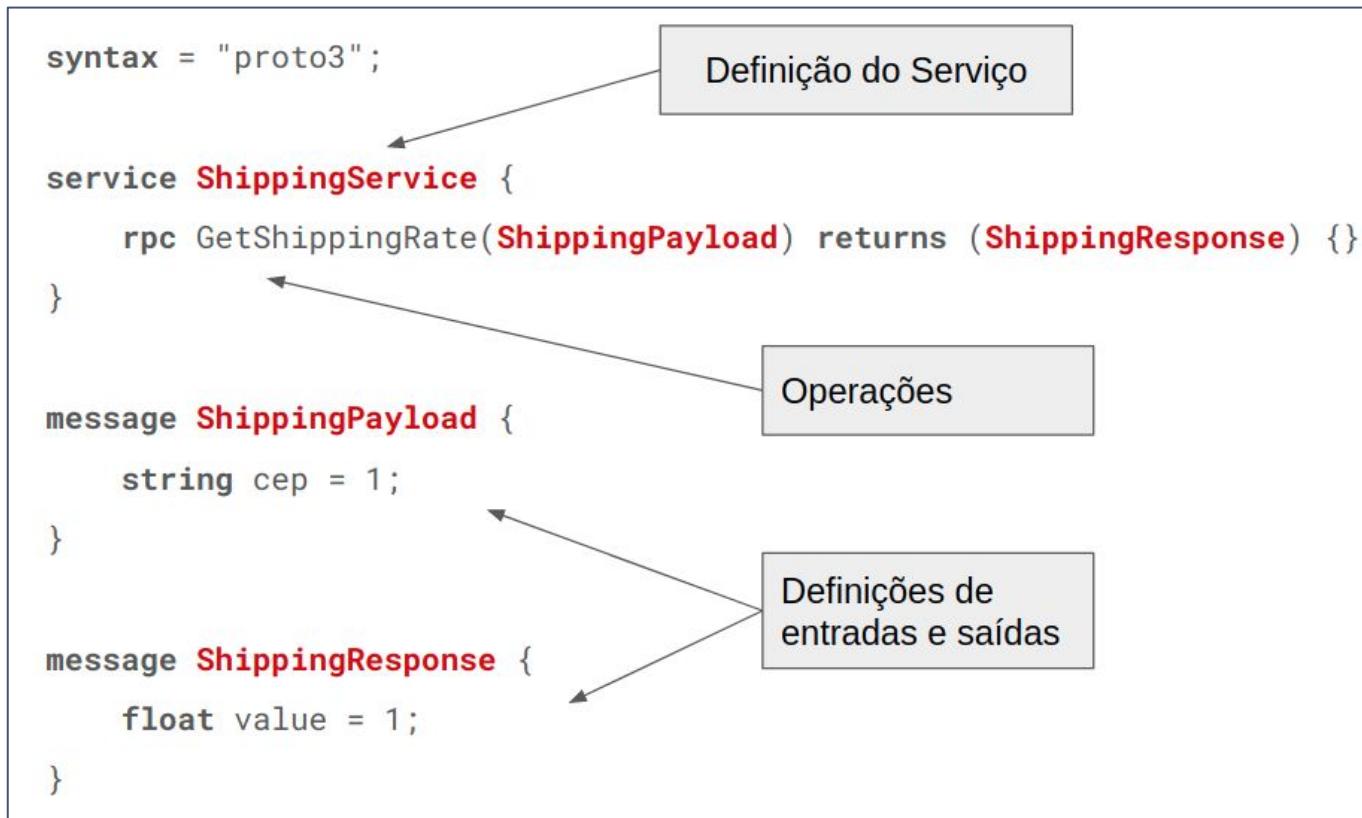


gRPC

- Sistema que viabiliza comunicação entre aplicações
- Baseado em Chamada Remota de Procedimentos
- Inclui um protocolo binário para comunicação distribuída
- E também uma linguagem para definição de interfaces
 - Clientes: chamam métodos dessa interface
 - Servidores: implementam métodos

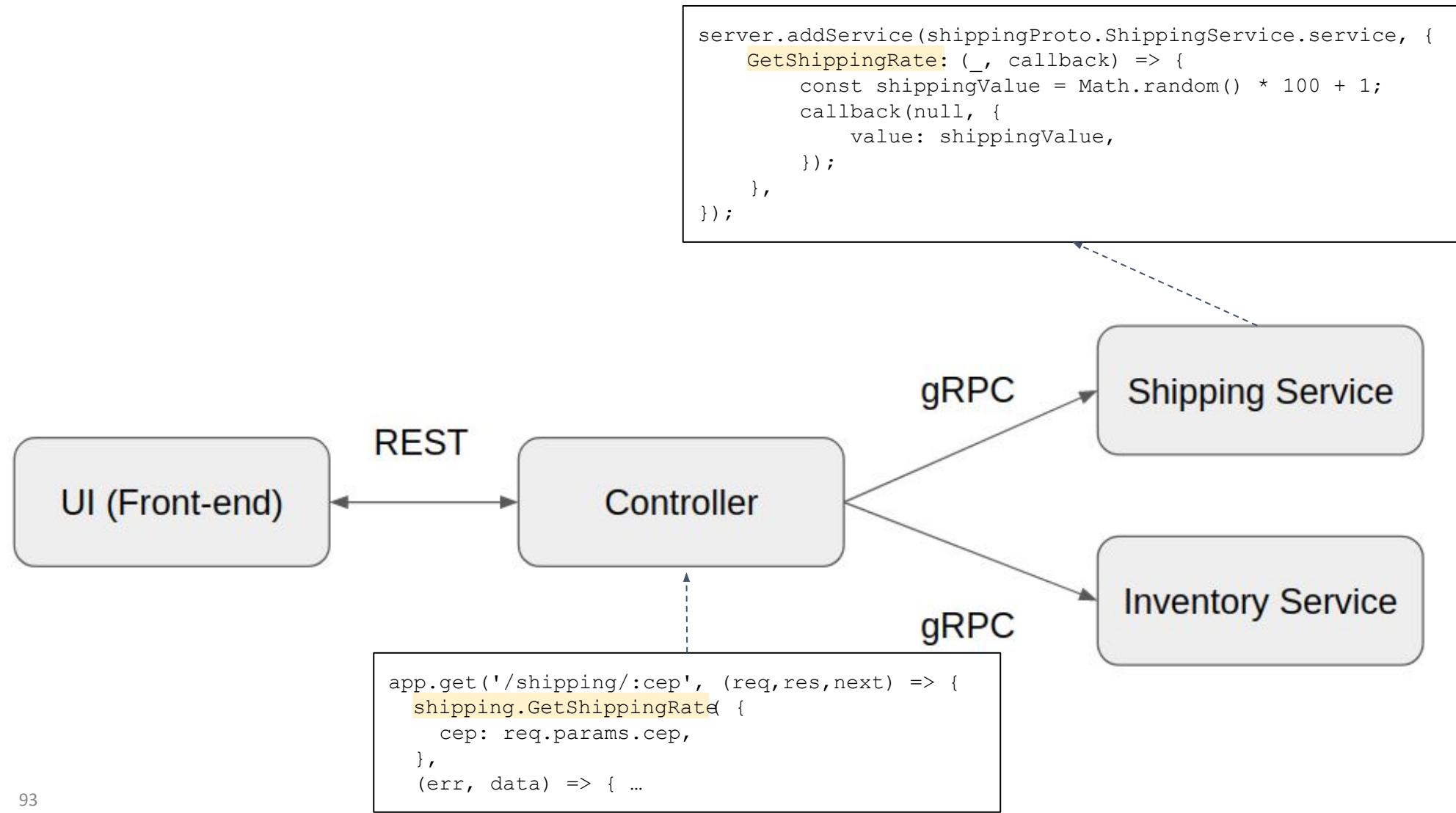
gRPC: Linguagem para definição de interfaces

```
syntax = "proto3";  
  
service ShippingService {  
    rpc GetShippingRate(ShippingPayload) returns (ShippingResponse) {}  
}  
  
message ShippingPayload {  
    string cep = 1;  
}  
  
message ShippingResponse {  
    float value = 1;  
}
```



The diagram illustrates the structure of a gRPC service definition. It shows a block of Protocol Buffer code with annotations:

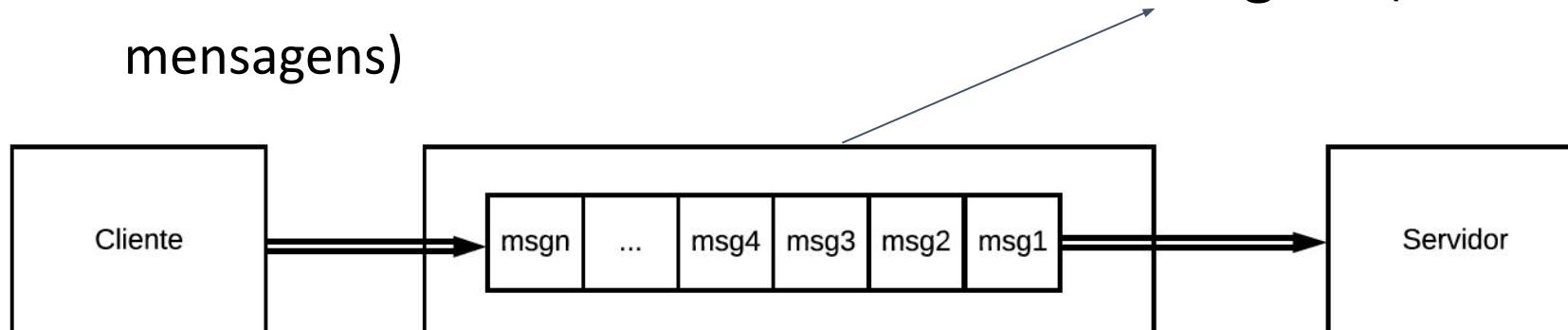
- An arrow points from the `syntax = "proto3";` line to a box labeled **Definição do Serviço**.
- An arrow points from the `service ShippingService {` line to a box labeled **Operações**.
- Two arrows point from the `message` definitions (`ShippingPayload` and `ShippingResponse`) to a box labeled **Definições de entradas e saídas**.



Arquitetura Orientada a Mensagens

Arquitetura orientada a Mensagens

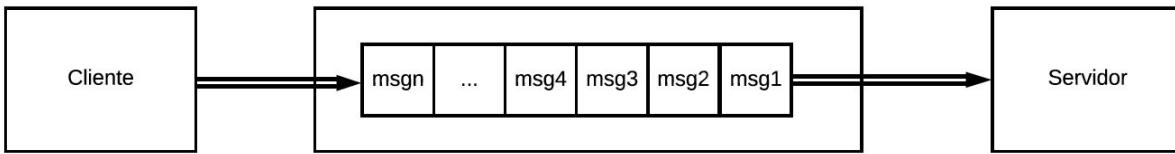
- Arquitetura para aplicações distribuídas
- Clientes não se comunicam diretamente com servidores
- Mas com um intermediário: **fila de mensagens** (ou broker de mensagens)



Exemplo: RabbitMQ

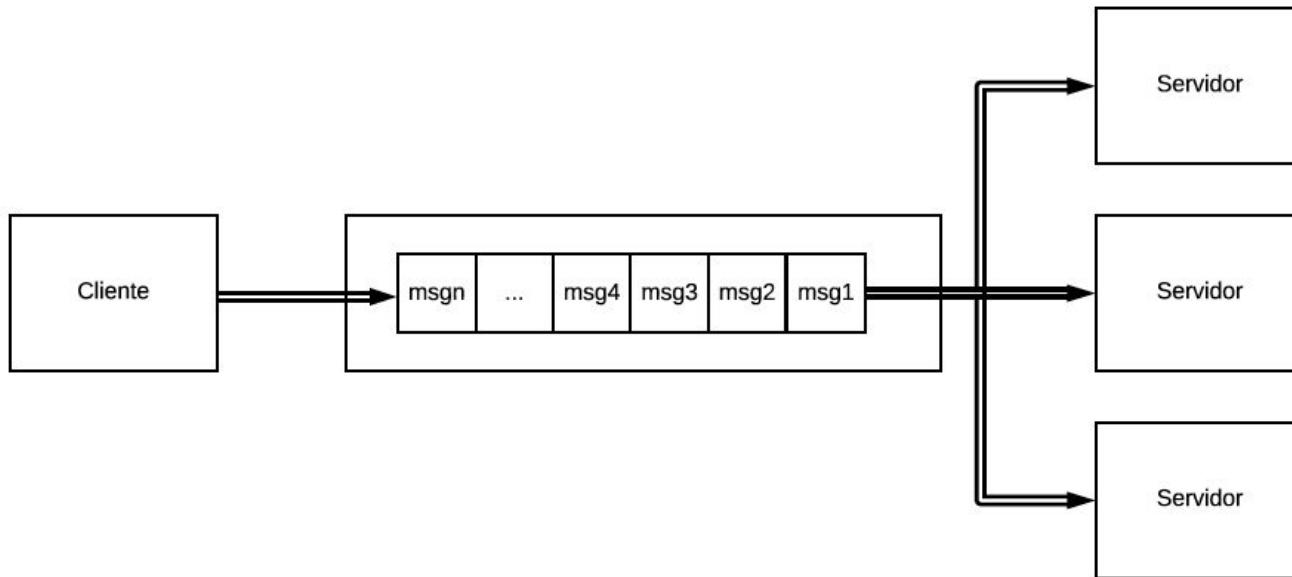
Vantagem #1: Tolerância a Falhas

- Não existe mais mensagem: "servidor fora do ar"
- Assumindo que a fila de mensagens roda em um servidor bastante robusto e confiável



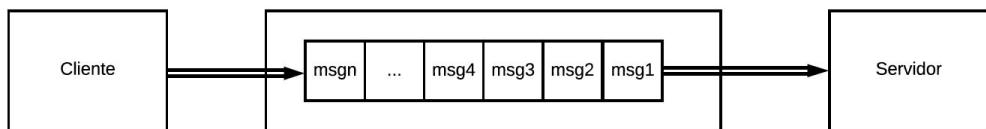
Vantagem #2: Escalabilidade

- Mais fácil acrescentar novos servidores (e mais difícil sobrecarregar um servidor com excesso de mensagens)



Comunicação Assíncrona

- Comunicação entre clientes e servidores é **assíncrona**
- Cria um acoplamento fraco entre clientes e servidores
- **Desacoplamento no espaço:** clientes não conhecem servidores e vice-versa
- **Desacoplamento no tempo:** clientes e servidores não precisam estar simultaneamente no ar



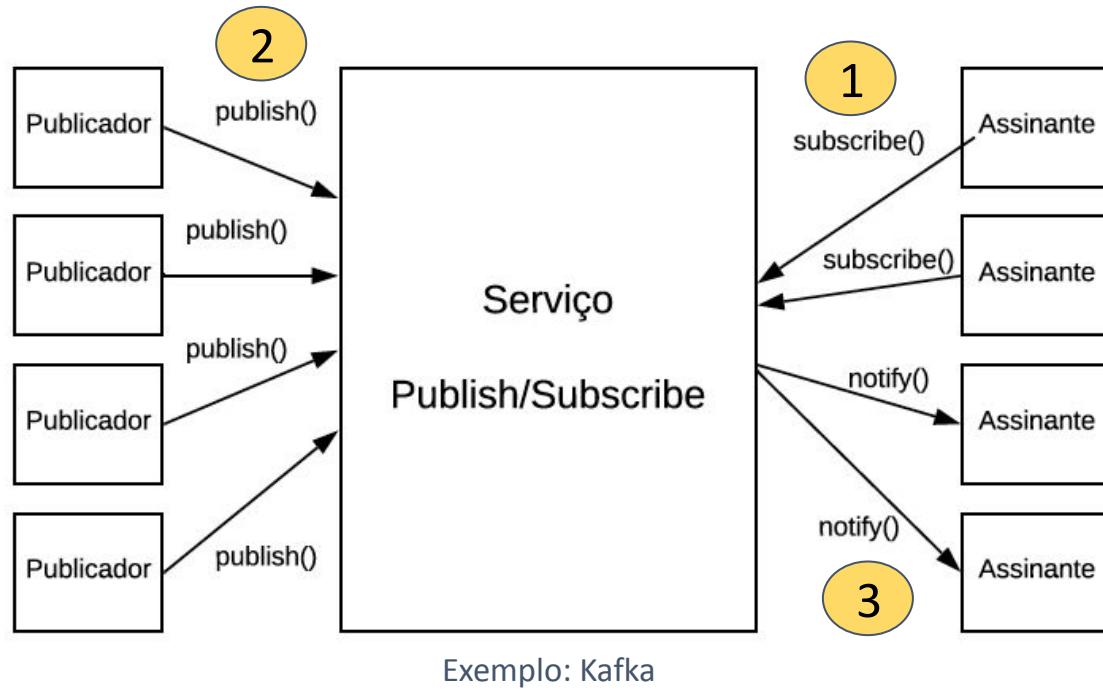
Arquitetura Publish/Subscribe

Publish/Subscribe

- "Aperfeiçoamento" de fila de mensagens
- Mensagens são chamadas de **eventos**

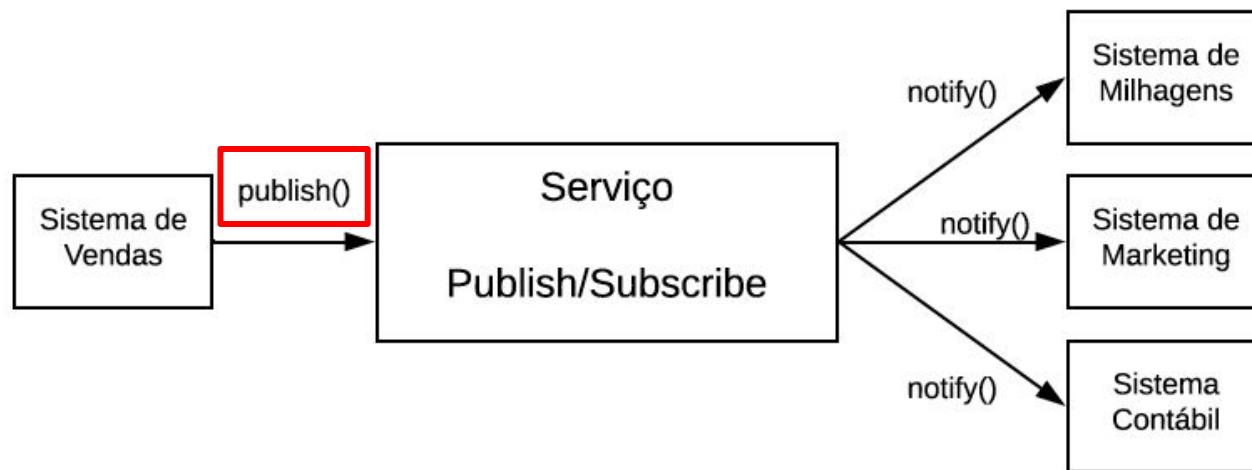
Publish/Subscribe

- Sistemas podem: (1) assinar eventos; (2) publicar eventos; (3) serem notificados sobre a ocorrência de eventos



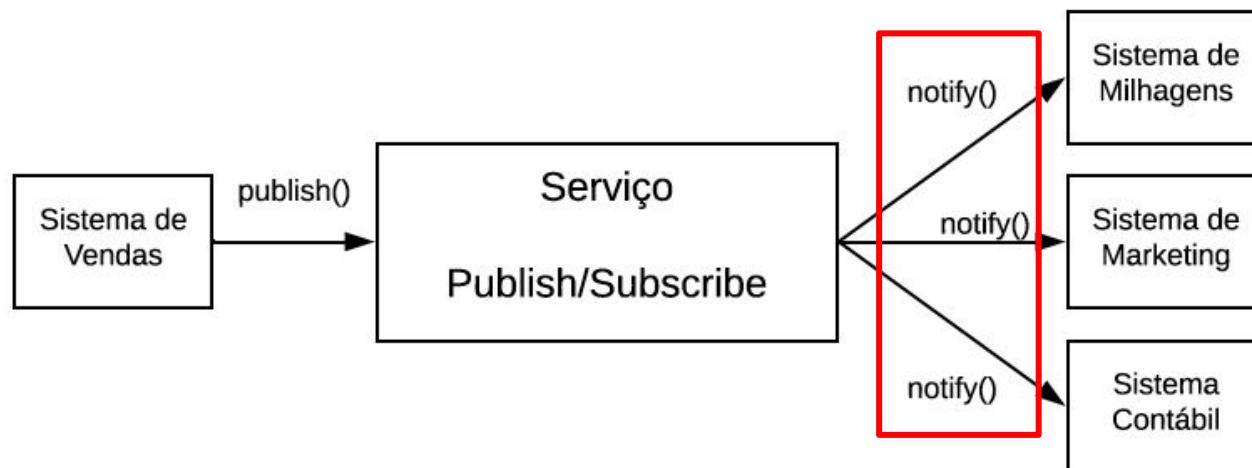
Exemplo: Sistema de Companhia Aérea

- Evento: venda de passagem (com dados da venda)



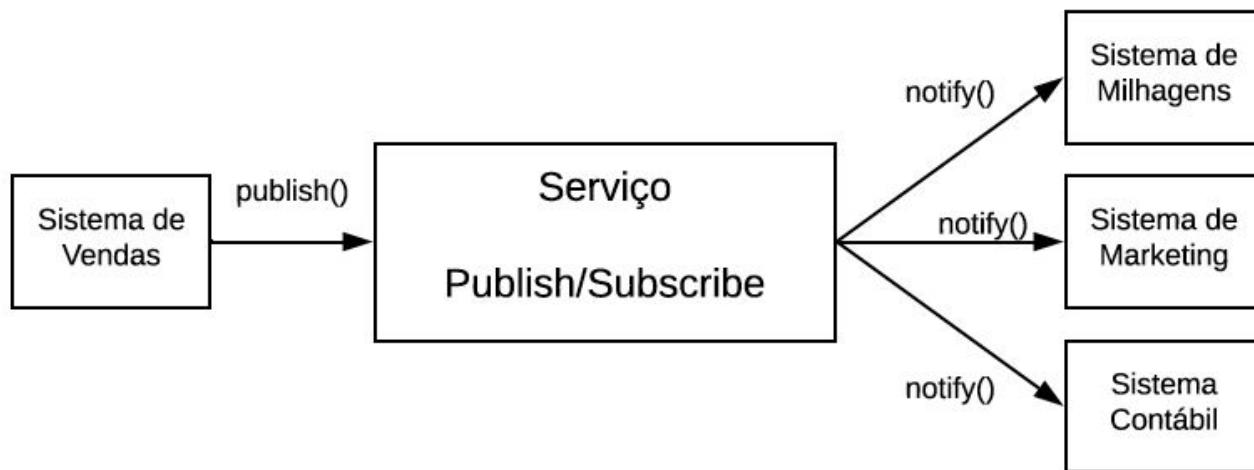
Exemplo: Sistema de Companhia Aérea

- Evento: venda de passagem (com dados da venda)



Exemplo: Sistema de Companhia Aérea

- Evento: venda de passagem (com dados da venda)



Comunicação em grupo: um sistema publica eventos, n sistemas assinam e são notificados da publicação

Roteiro prático sobre Publish/Subscribe

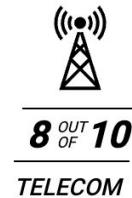
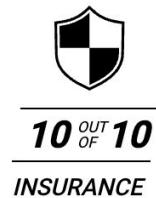
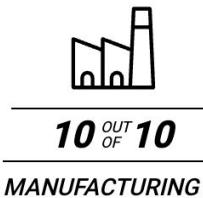
<https://github.com/aserg-ufmg/pub-sub-store>

Exemplo de Sistema Pub/Sub

APACHE KAFKA

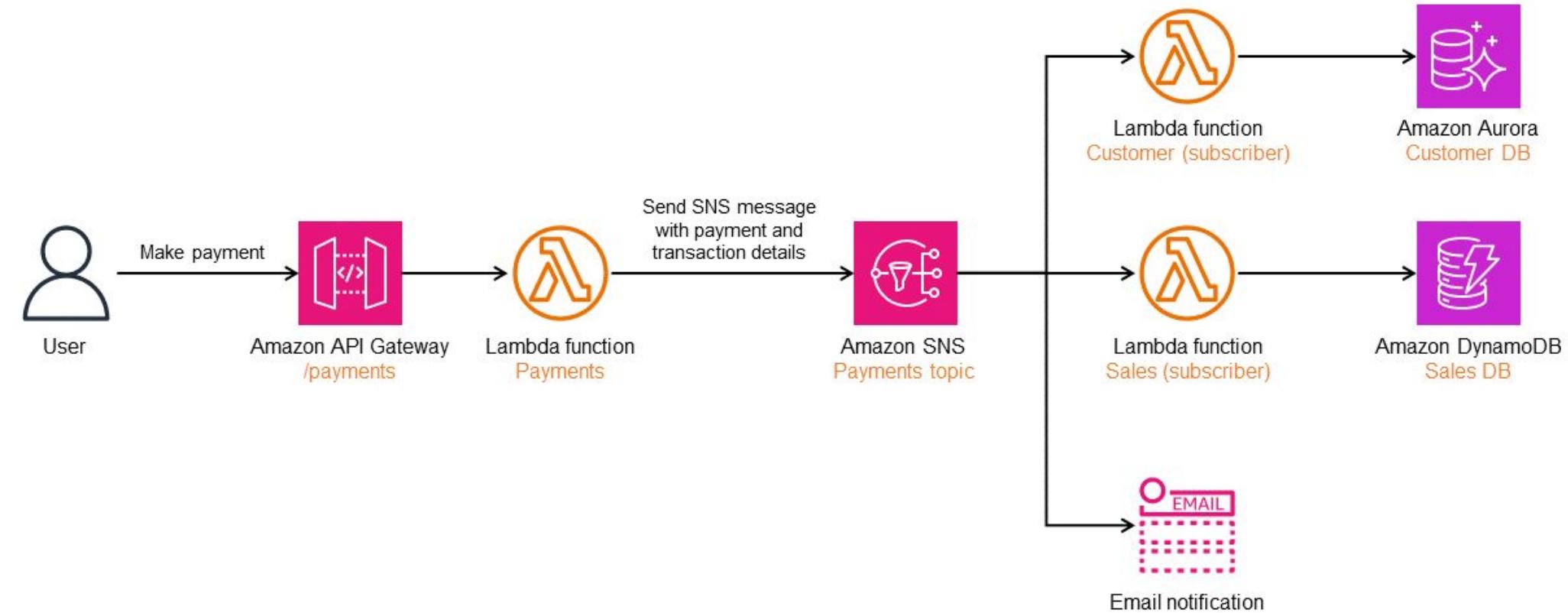
More than 80% of all Fortune 100 companies trust, and use Kafka.

Apache Kafka is an open-source distributed event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications.



<https://kafka.apache.org>

AWS



Outros Padrões Arquiteturais

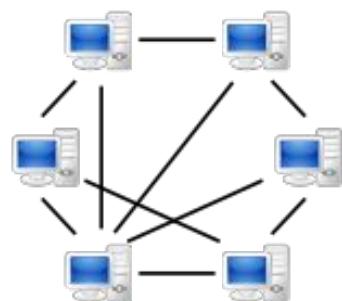
(1) Pipes e Filtros

- Programas são chamados de **filtros** e se comunicam por meio de **pipes** (que agem como buffers)
- Arquitetura bastante flexível. Usada por comandos Unix.
- Exemplo: `ls | grep csv | sort`



(3) Peer-to-Peer

- Todo nodo pode ser cliente e/ou servidor
- Isto é, pode ser consumidor e/ou provedor de recursos
- Exemplo: sistemas para compartilhamento de arquivos (usando BitTorrent); Blockchain



Exercícios

1. Suponha uma empresa de streaming que deseja implementar um sistema detector de problemas de qualidade em seus vídeos (por exemplo, problemas nas legendas, no áudio, congelamento de imagens, etc). Os vídeos estão todos armazenados em um sistema de storage, isto é, em memória secundária. A empresa está avaliando duas arquiteturas:

Arquitetura #1: cada detector é um microsserviço, que recebe o nome do vídeo como parâmetro, carrega o mesmo do storage e executa um algoritmo de detecção de problemas de qualidade nesse vídeo.

Arquitetura #2: os detectores são módulos de um monolito. Então, o vídeo é carregado uma única vez do storage para a memória principal e compartilhado por todos os detectores de qualidade.

Supondo que a empresa de streaming tem milhões de clientes (ou seja, é uma empresa do porte da Amazon Prime Video) qual arquitetura você considera mais escalável? Justifique.

Observação: este exercício é baseado em um [artigo](#) do blog da Amazon Prime Video



Outros Estilos Arquiteturais

Web Standards

RIA - Rich
Internet
Application

Web 2.0

Aplicações
Móveis

Baseada em
Serviços (SOA)

Baseado em
computação em
Grid / P2P /
Componentes
distribuídos

Microserviços

IoT

Web Standards

- Quando usar:
 - Padrões para aplicações Web
- Vantagens:
 - Compatibilidade de browsers
 - Existem vários frameworks que facilitam a vida do desenvolvedor
- Desvantagens:
 - Escolha do melhor framework
 - Estudo do framework
 - Área em constante evolução

Web Standards - <https://www.w3.org/standards/>

- *Web Standards* é um conjunto de normas, diretrizes, recomendações, notas, artigos, tutoriais e afins de caráter técnico, produzidos pela [W3C](#).
- É destinado a orientar fabricantes, desenvolvedores e projetistas para o uso de práticas que possibilitem a criação de uma [Web](#) acessível a todos.
- Definições:
 - [**Web Design e Aplicações**](#)
 - Padrões para o desenvolvimento de páginas Web, incluindo HTML & CSS, Script e Ajax, Gráficos, Áudio e Vídeo, Acessibilidade, Internacionalização, Mobile Web, Privacidade e Matemática na Web.
 - [**Arquitetura Web**](#)
 - Foco nos princípios da Arquitetura Web, Identificadores (URL, URI e IRI), Protocolos (HTTP, XML, etc.), Meta Formatos e Internacionalização.
 - [**Tecnologia XML**](#)
 - Tecnologias XML, incluindo XML, XML Namespaces, XML Schema, Efficient XML Interchange (EXI) e outros padrões relacionados.

Web Standards

▪ Web Semântica

- Conheça as tecnologias e ferramentas para dar suporte à "Web dos dados", viabilizando pesquisas como num banco de dados. Aqui você irá aprender sobre Dados Linkados, Vocabulários e Ontologias, Consultas, Inferência e Aplicações verticais.

▪ Web Services

- Refere-se às tecnologias como HTTP, XML, SOAP, WSDL, SPARQL, entre outras, que possibilitam a integração e comunicação entre diferentes aplicações.

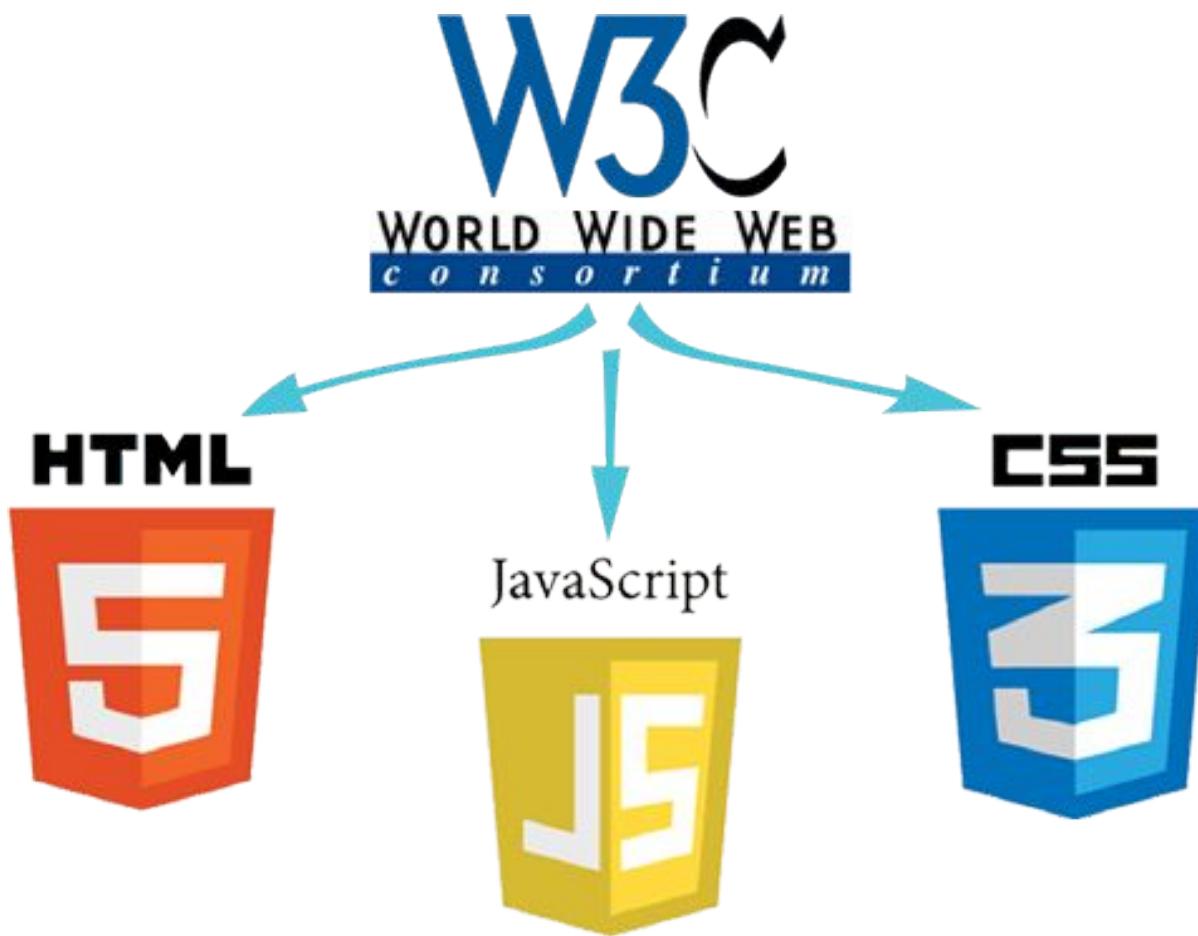
▪ Web de Dispositivos

- Tecnologias que permitem o acesso à Web por todos, de qualquer lugar, a

▪ Navegadores e Ferramentas de Autoria

- Nós devemos ser capazes de publicar e acessar conteúdos independentemente do software que utilizamos, do computador que temos, idioma que falamos, se estamos conectados sem fio ou não, se a tela é grande ou pequena, etc. Estas normas tem o objetivo de aprimorar constantemente a web que é aberta para todos nós.

Web Standards



Web Standards - HTML 5

What Does HTML5 Do?

Key features of the next Web programming standard.



<STORAGE>

Data can be stored on a user's computer or mobile device, so Web apps work without an Internet connection.



<TYPE>

Web pages can have flashier type with more fonts, shadows, colors and other effects.



< MOTION >

Objects move on Web pages and react to the movements of a cursor.



<GAMES>

Interactive games can run with just a Web browser without installing other software or plug-ins.



<VIDEO>

Video can be embedded in a Web page without a plug-in. Browser makers have not agreed on formats.



<3D>

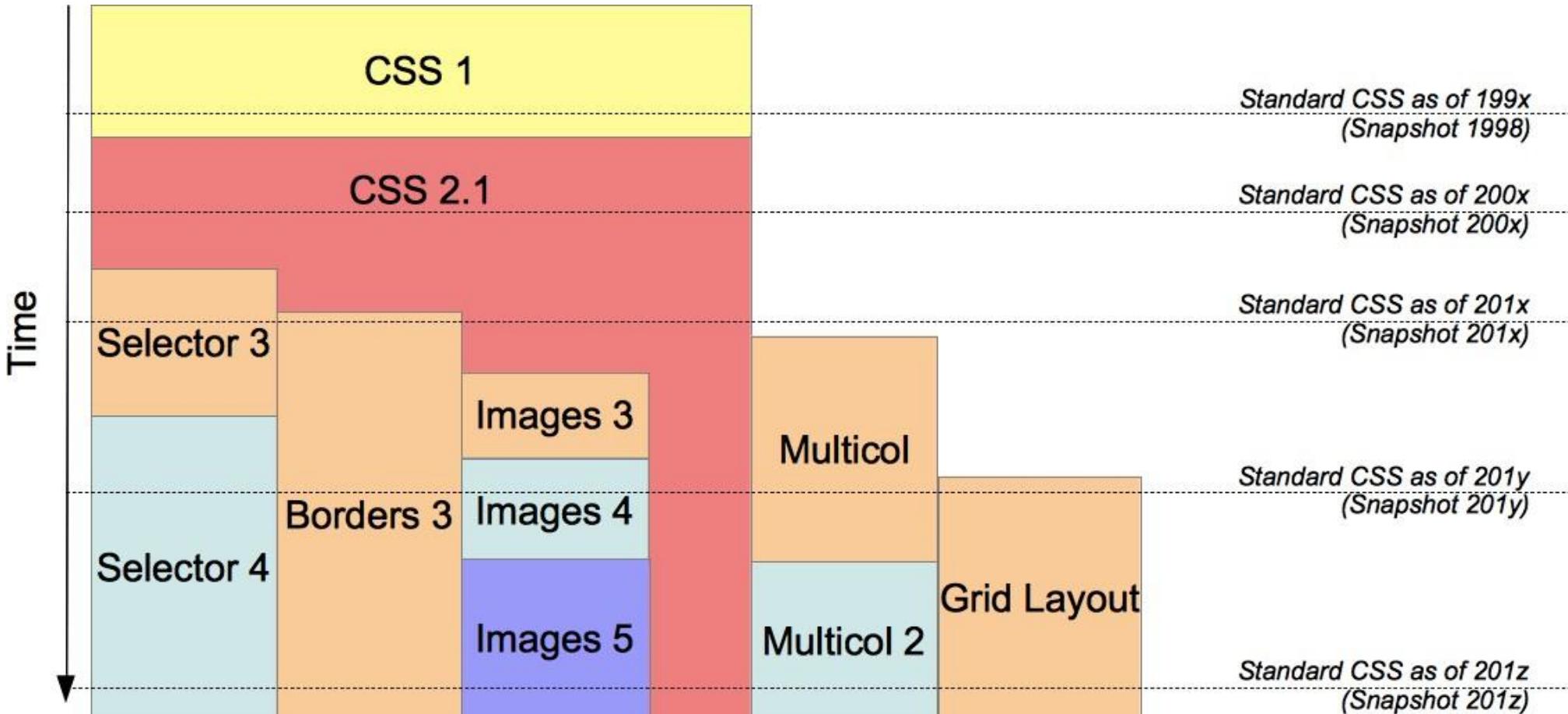
A technology called WebGL can create interactive 3-D effects using a computer's graphics processor.



<AUDIO>

Audio is played without a plug-in. Browser makers have not agreed on formats.

Web Standards – CSS 3



Web Standards – Javascript

MEAN STACK



Mongo DB
(database system)



Express
(back-end web
framework)



Angular.js
(front-end
framework)



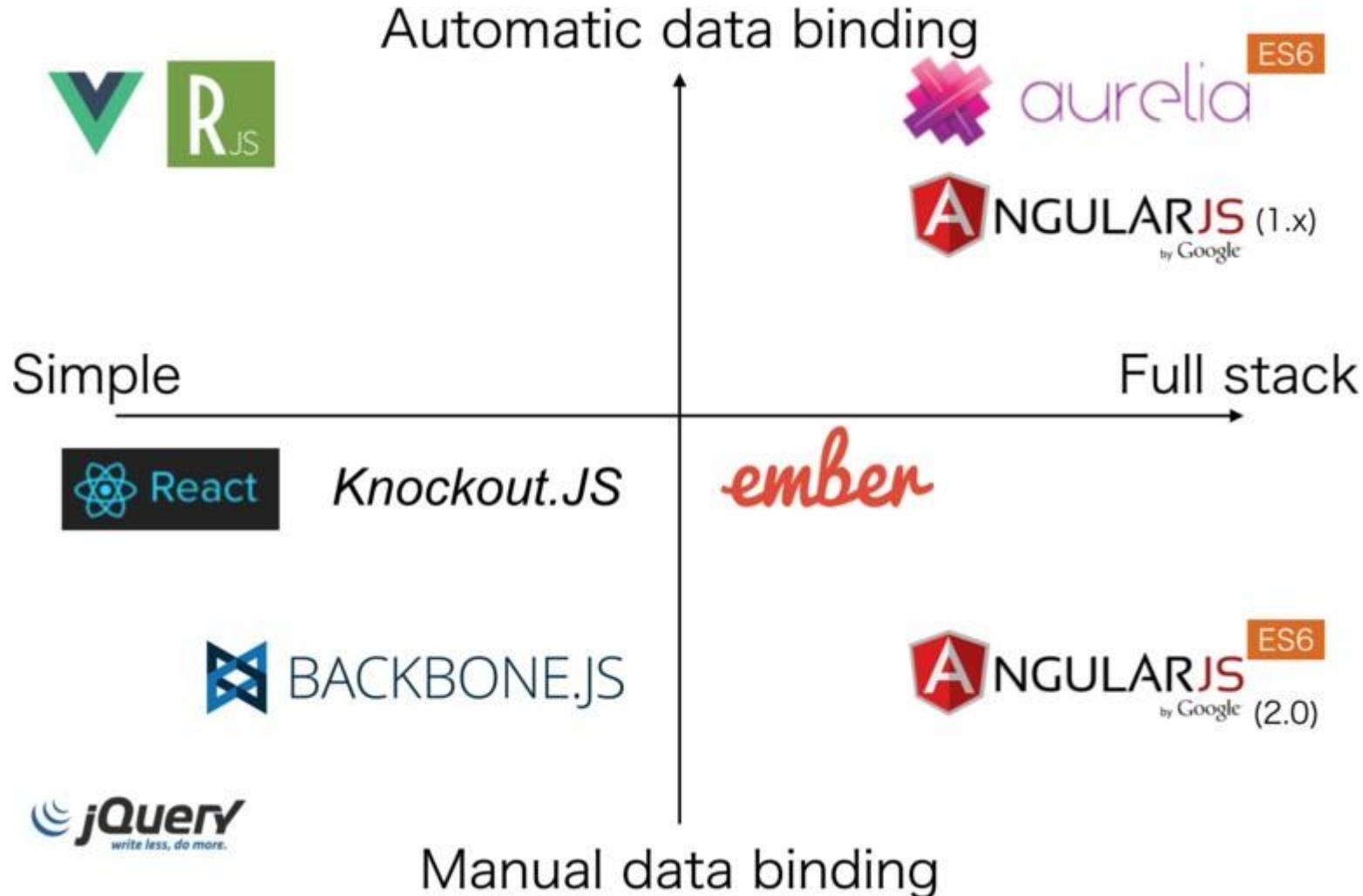
Node.js
(back-end runtime
environment)

Web Standards – Javascript



<http://noeticforce.com/best-javascript-frameworks-for-single-page-modern-web-applications>

Web Standards – Javascript



Outros Estilos Arquiteturais

Web Standards

RIA - Rich
Internet
Application
Web 2.0

Aplicações
Móveis

Baseada em
Serviços (SOA)

Baseado em
computação em
Grid / P2P /
Componentes
distribuídos

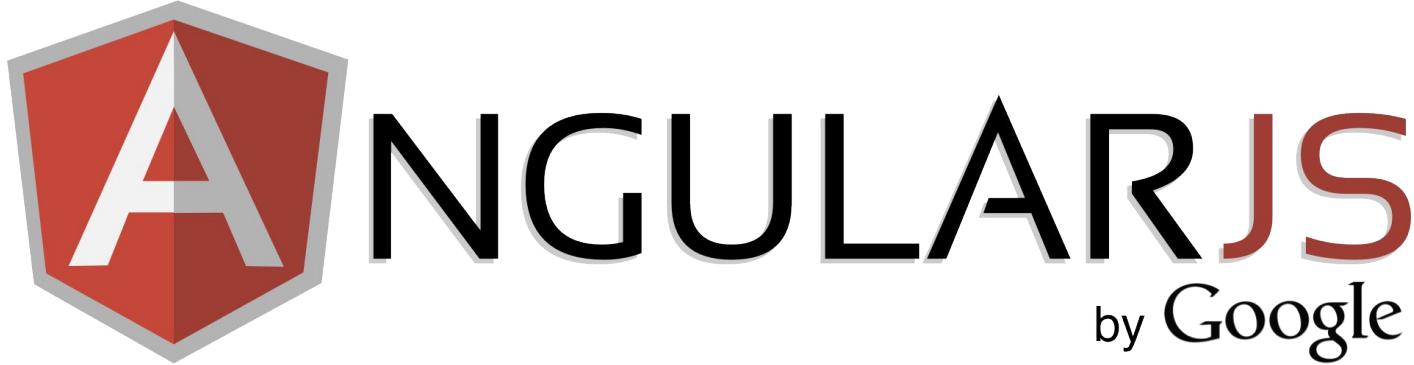
Microserviços

IoT

RIA – Rich Internet Application / Web 2.0

- Quando usar:
 - Criação de interfaces profissionais na Web
- Vantagens:
 - Economia no desenvolvimento de interfaces com usuário
 - Existem diversos frameworks
- Desvantagens:
 - Escolha do melhor framework
 - Estudo do framework
 - Área em constante evolução

RIA best frameworks



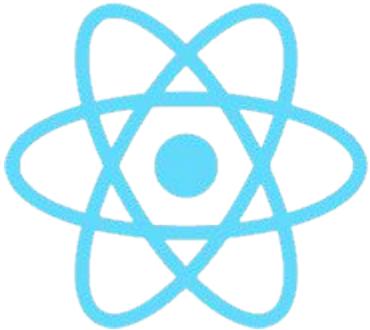
- Usa o padrão MVC
- Criado e mantido pela Google
- <https://angularjs.org/>

RIA best frameworks



- Framework PHP lançado em 2011
- Usa o padrão MVC
- <https://laravel.com/>

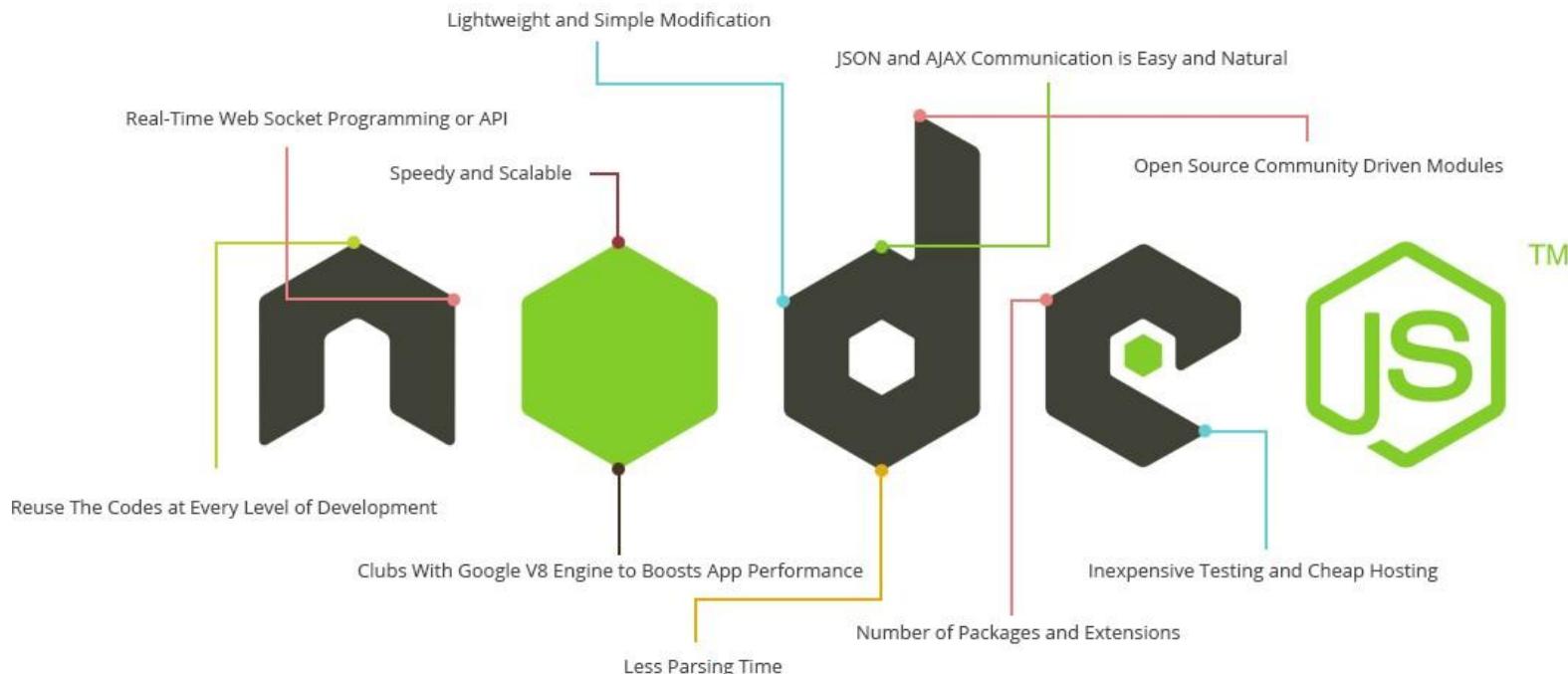
RIA best frameworks



React

- Mantido pelo Facebook
- Ideal para aplicações Web cujos dados mudam constantemente
- Sendo reescrito para o React Fiber
- <https://facebook.github.io/react/>

RIA best frameworks



- Não é apenas um framework, mas um ambiente completo
- Ambiente de execução JavaScript baseado no engine JS do Chrome V8
- É um servidor JS escalável que resolve o problema C10K (non-blocking I/O model & event-driven)
- <https://nodejs.org/en/>

RIA best frameworks



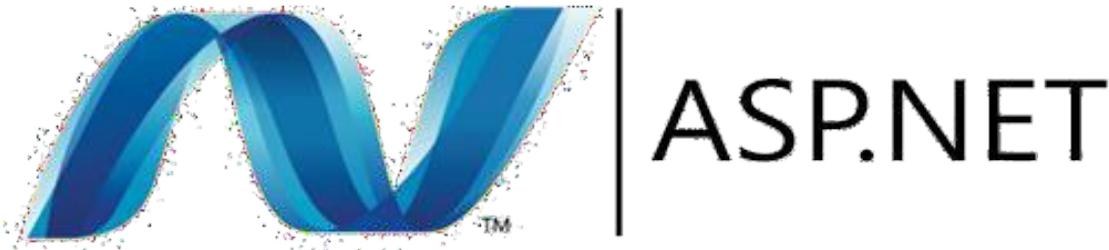
- Promete aumentar velocidade e facilidade no desenvolvimento de sites orientados a banco de dados (*database-driven web sites*)
- Cria aplicações web com base em estruturas pré-definidas
- Usa o padrão MVC
- <http://rubyonrails.org/>

RIA best frameworks



- Framework PHP para aplicações empresariais grandes e complexas
- Segue o padrão MVC
- Projetado para permitir que os desenvolvedores apliquem princípios ágeis do desenvolvimento e foquem na regras de negócio
- <https://symfony.com/>

RIA best frameworks



- Framework Web da Microsoft desde 2002, mas é open-source
- Facilidade na criação de aplicações Web
- Permite a escrita usando várias linguagens
- Permite o uso do Visual Studio 2017 gratuitamente
- <https://www.asp.net/>

RIA best frameworks



- Framework Web para PHP
- Utiliza a metodologia Rapid Application Development (RAD)
- O nome Yii (pronunciado I ou Di) representa as palavras Yes It Is! (Sim isto é!), em resposta às mais comuns perguntas ao que escolhem a plataforma Yii: Isto é seguro? ... Isto é rápido? ... Isto é profissional? ... Isto é certo para o meu próximo projeto?
- <http://www.yiiframework.com/>

RIA best frameworks



- É um framework JS escrito usando o Node.JS e integrado ao MongoDB
- Ideal para prototipação rápida com código cross-plataform (Android, IOS, Web)
- É uma plataforma completa com Javascript no frontend e backend
- <https://www.meteor.com/>

RIA best frameworks



- Framework Web para PHP
- Utiliza o modelo MVC, com diversos extensores
- Possui muito material de treinamento disponível
- Possui um gerador de código Bake para *database-driven web sites*
- <https://cakephp.org/>

RIA best frameworks



- Framework progressivo JS para interfaces com usuário
- Ideal para a construção de aplicações Single-Page
- É atualmente um dos mais procurados e ativos
- <https://vuejs.org/>

RIA best frameworks



- Um dos melhores framework Web para Java
- Baseado em eventos Ajax e componentes ricos
- Utiliza o padrão MVVM
- <https://www.zkoss.org/>

RIA best frameworks



- Framework para aplicações RIA em Java
- Utiliza o GWT – Google Web Toolkit
- <https://vaadin.com/>

RIA best frameworks



- Framework padronizado para interfaces Java EE
- Utiliza o padrão MVP
- Consiste em um padrão definido pelo JCP (**Java** Community Process)
- Possui diversas implementações, tais como: [PrimeFaces](#),
[MyFaces](#), [IceFaces](#), etc.

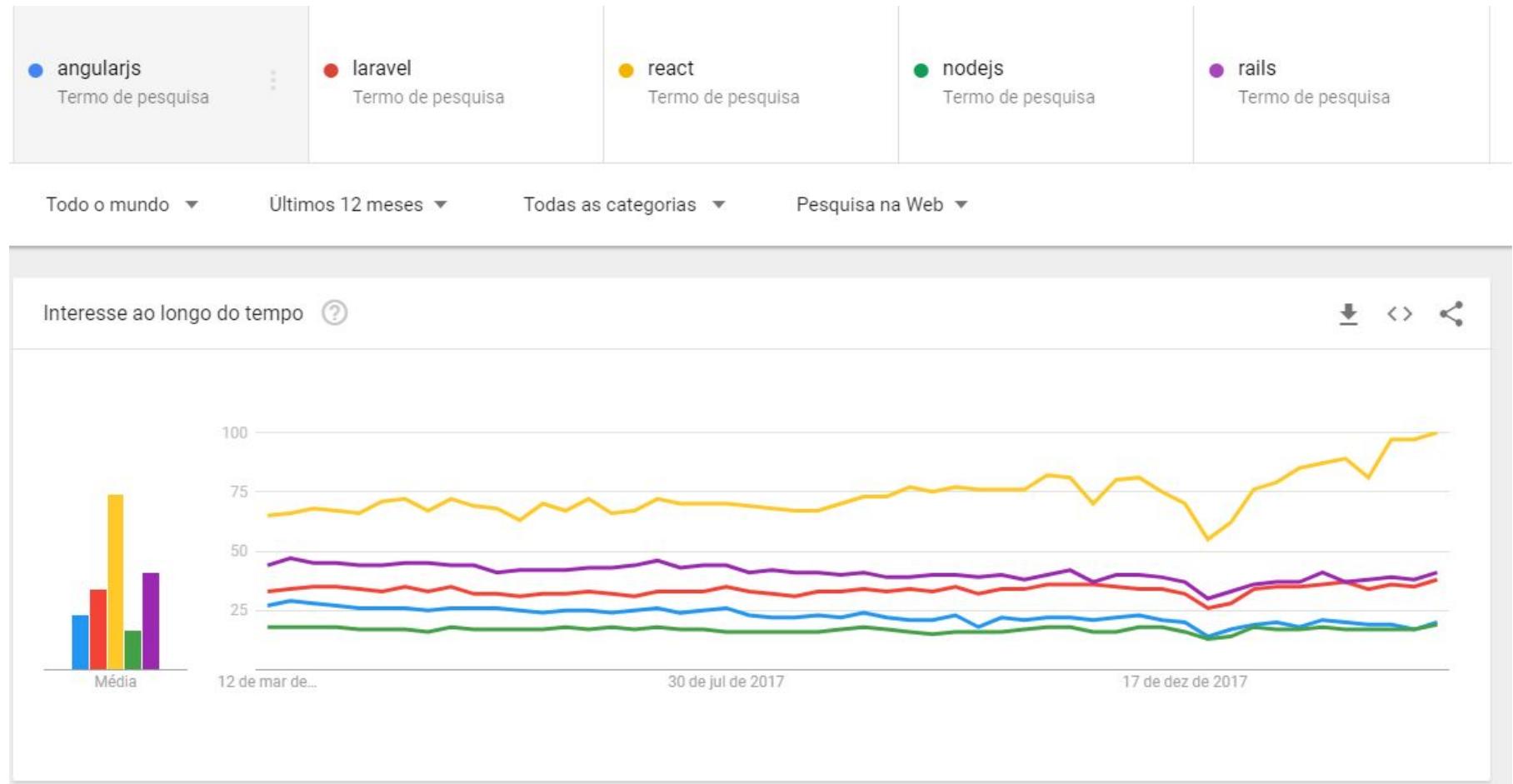
RIA frameworks

□ Dentre outros...



Como escolher...

- Uma boa dica é ver o Google Trends <https://trends.google.com.br/trends/>



Outros Estilos Arquiteturais

Web Standards

RIA - Rich
Internet
Application

Web 2.0

Baseada em
Serviços (SOA)

Aplicações
Móveis

Baseado em
computação em
Grid / P2P /
Componentes
distribuídos

Microserviços

IoT

Aplicações Móveis

- Quando usar:
 - Criação de Apps para smartphones
- Vantagens:
 - Existem diversos frameworks
- Desvantagens:
 - Evolução constante de dispositivos e versões de Sistemas Operacionais



Aplicações Móveis - Opções

▪ Hybrid Mobile App Choices (Cross Platform)

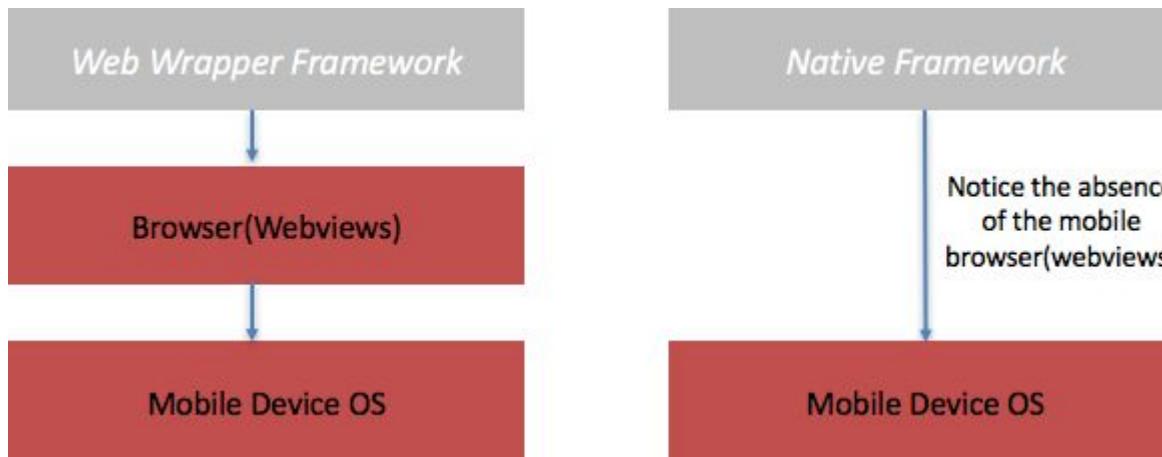
- Permite utilizar Web Standards (HTML 5, CSS 3 e JS) para ser empacotada e distribuída em um dispositivo (Ex: [Apache Cordova](#), [Phonegap](#), [Ionic](#))

▪ Native Mobile App Choices (Cross Platform)

- Permite utilizar Web Standards (HTML 5, CSS 3 e JS) mas depois gera um código nativo para a plataforma (Ex: [React Native](#), [Angular 2 Nativescript](#), [Xamarin](#))

▪ Fully Native Choices (Platform Specific)

- Utiliza a linguagem própria do SDK do Sistema Operacional do dispositivo (Ex: [Swift/Objective C \(IOS\)](#), [Java Android Studio](#), [Windows Phone*](#)(fim dez/2019))



Aplicações Móveis



APACHE
CORDOVA™

Mobile apps with **HTML, CSS & JS**

Target multiple platforms with **one code base**

Free and **open source**

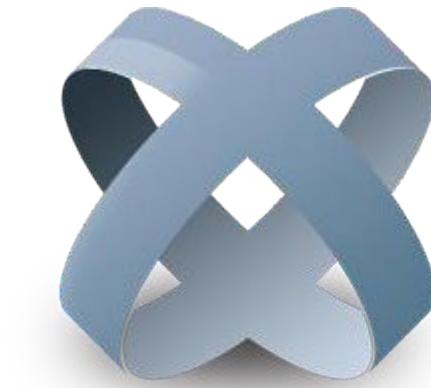
•4 more...

[GET STARTED](#) [DOCUMENTATION](#)

The image shows the Apache Cordova landing page. It features a large, stylized logo on the left consisting of a white hexagon containing a dark hexagon with two glowing blue vertical bars. To the right of the logo, the word "APACHE" is in small capital letters, followed by "CORDOVA" in large, bold, white capital letters with a trademark symbol. Below the title, there are three main bullet points: "Mobile apps with HTML, CSS & JS", "Target multiple platforms with one code base", and "Free and open source". Under the last point, there is a link labeled "•4 more...". At the bottom, there are two blue rectangular buttons with white text: "GET STARTED" and "DOCUMENTATION".

Aplicações Móveis

□ Dentre outras...



titanium™

Aplicações Móveis

□ Lojas



Outros Estilos Arquiteturais

Web Standards

RIA - Rich
Internet
Application

Web 2.0

Baseada em
Serviços (SOA)

Aplicações
Móveis

Baseado em
computação em
Grid / P2P /
Componentes
distribuídos

Microserviços

IoT

Baseada em Serviços (SOA)

□ Quando usar:

- Quando você quer potencializar o legado
- Necessidade de integração entre diversos sistemas a nível empresarial

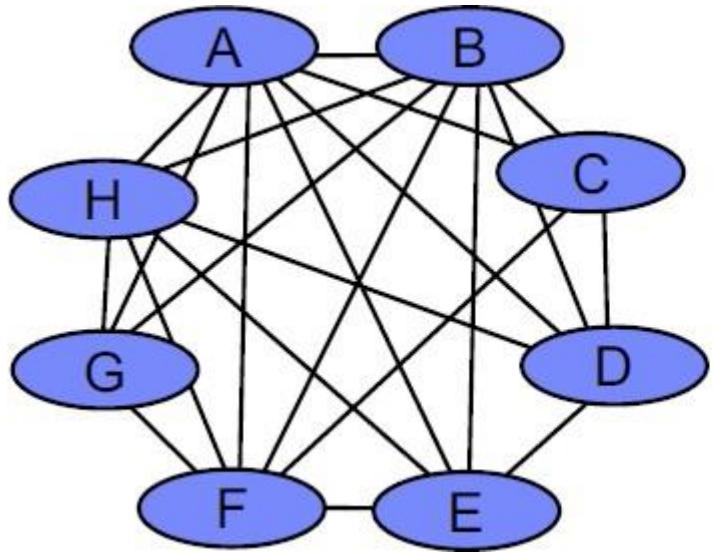
□ Vantagens:

- Maior governança, gestão, controle dos serviços
- Maior disponibilidade e qualidade dos serviços

□ Desvantagens:

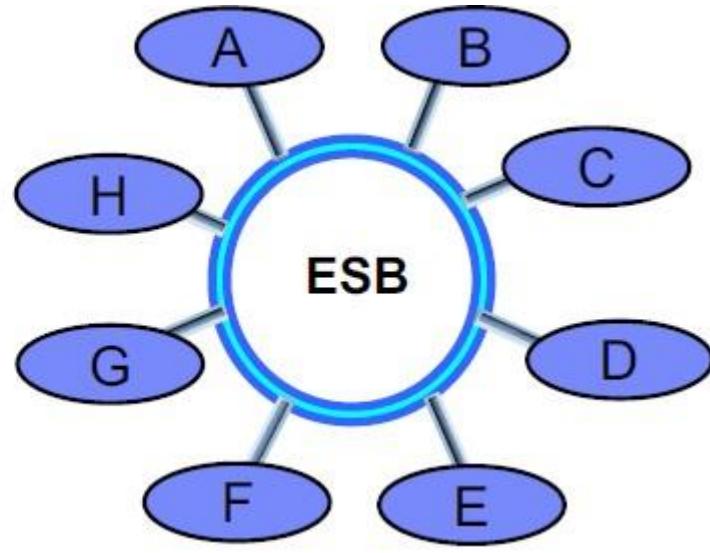
- Mudança cultural, treinamento
- Necessário uma equipe responsável por gerir os serviços
- Maior complexidade no desenvolvimento e manutenção
- Adiciona novos componentes na infraestrutura
- Soluções robustas de ESB são caras

ESB – Enterprise Service Bus



□ P2P – Peer to Peer

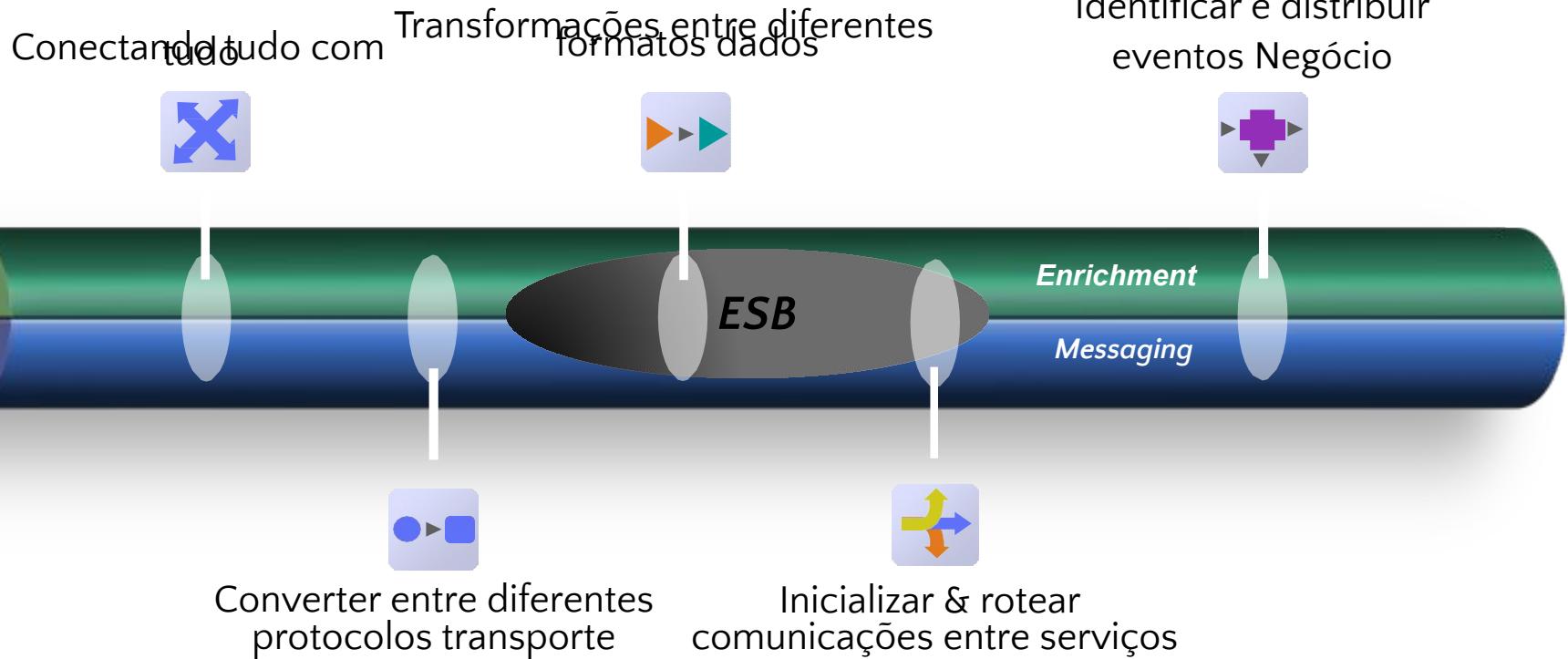
- Cada componente tem uma API privada para outro componente
- A complexidade e o número de transformações de dados cresce exponencialmente
- Modificações físicas nos componentes requerem alterações em todos os outros componentes



□ ESB

- Cada componente se conecta a um barramento que padroniza conexão e transformação
- A complexidade e número de transformações de dados cresce linearmente
- Componentes podem sofrer alterações físicas sem impacto na aplicação

ESB – Enterprise Service Bus



Lógica de Conectividade, e não Lógica de Negócio!

ESB – Enterprise Service Bus

- Soluções ESB disponíveis no mercado:
- Comerciais:
 - (TIBCO) ActiveMatrix™ BusinessWorks
 - IBM WebSphere ESB
 - IBM WebSphere Message Broker
 - Microsoft BizTalk Server
 - Windows Azure Service Bus
 - Oracle Enterprise Service Bus (BEA Logic)
 - Progress Sonic ESB
 - Red Hat JBoss Fuse

ESB – Enterprise Service Bus

- Soluções ESB disponíveis no mercado:
- Open Source:
 - [Apache ServiceMix](#)
 - [Apache Synapse](#)
 - [JBoss ESB](#)
 - [NetKernel](#)
 - [Petals ESB](#)
 - [Spring Integration](#)
 - [Open ESB](#)
 - [WSO2 ESB](#)
 - [Mule](#)
 - [UltraESB](#)
 - [Red Hat Fuse ESB](#) (based on Apache ServiceMix)

ESB – Enterprise Service Bus

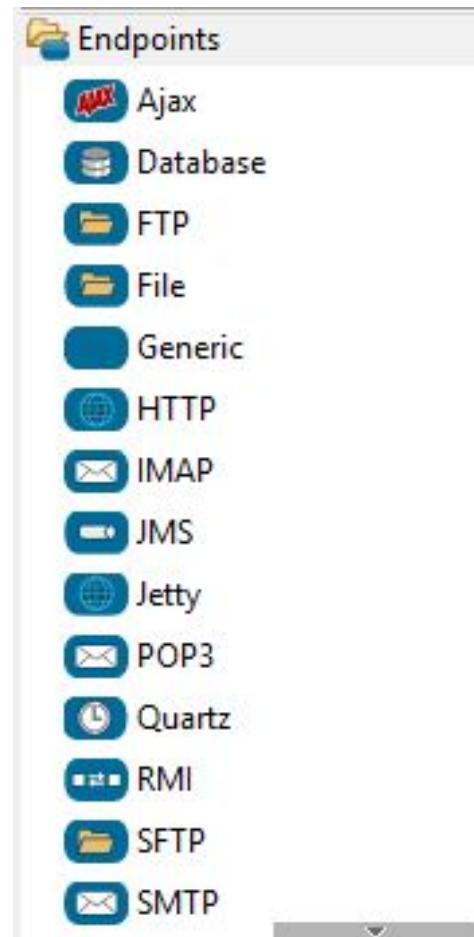
- Soluções ESB disponíveis no mercado:
- Pagas:
 - IBM Websphere Enterprise Service Bus
 - IBM Message Broker
 - IBM DataPower
 - IBM WSRR Websphere Service Registry and Repository
 - TIBCO
 - Progress Aurea Sonic ESB
 - Fuse ESB
 - Pegasystems

Exemplo MuleSoft - www.mulesoft.com

- Estamos utilizando a versão:
 - MuleStudio-for-win-32bit-3.5.0
- Para utilizar, rode o MuleStudio:

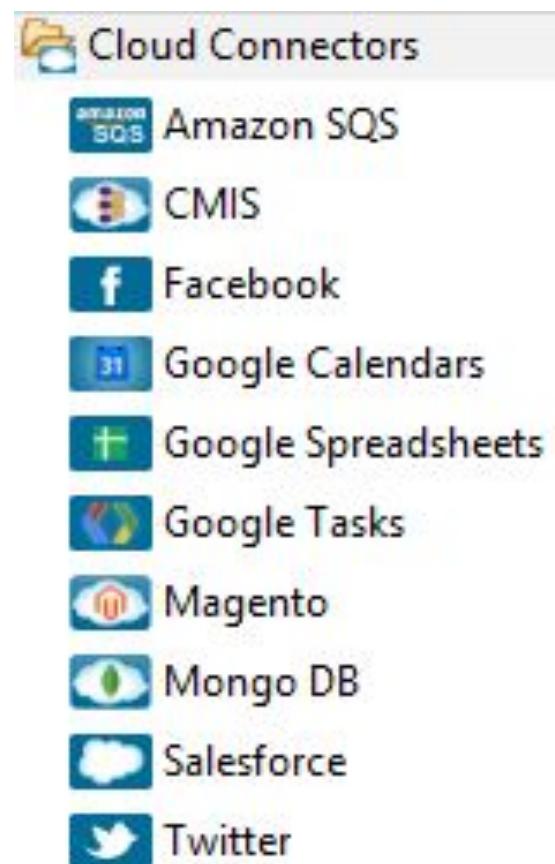
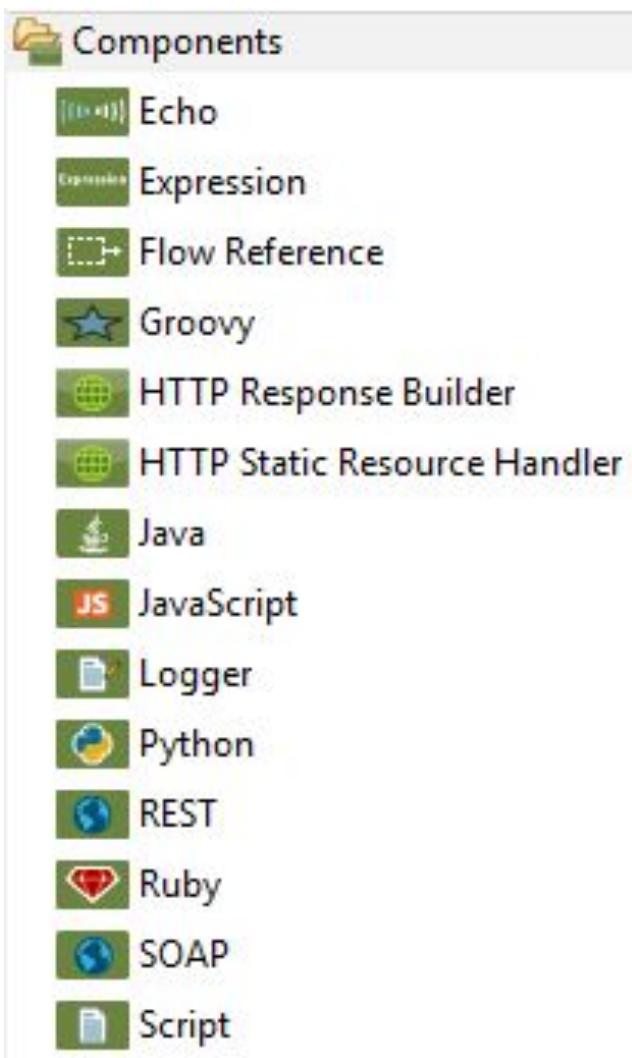


- Ele possui vários endpoints:



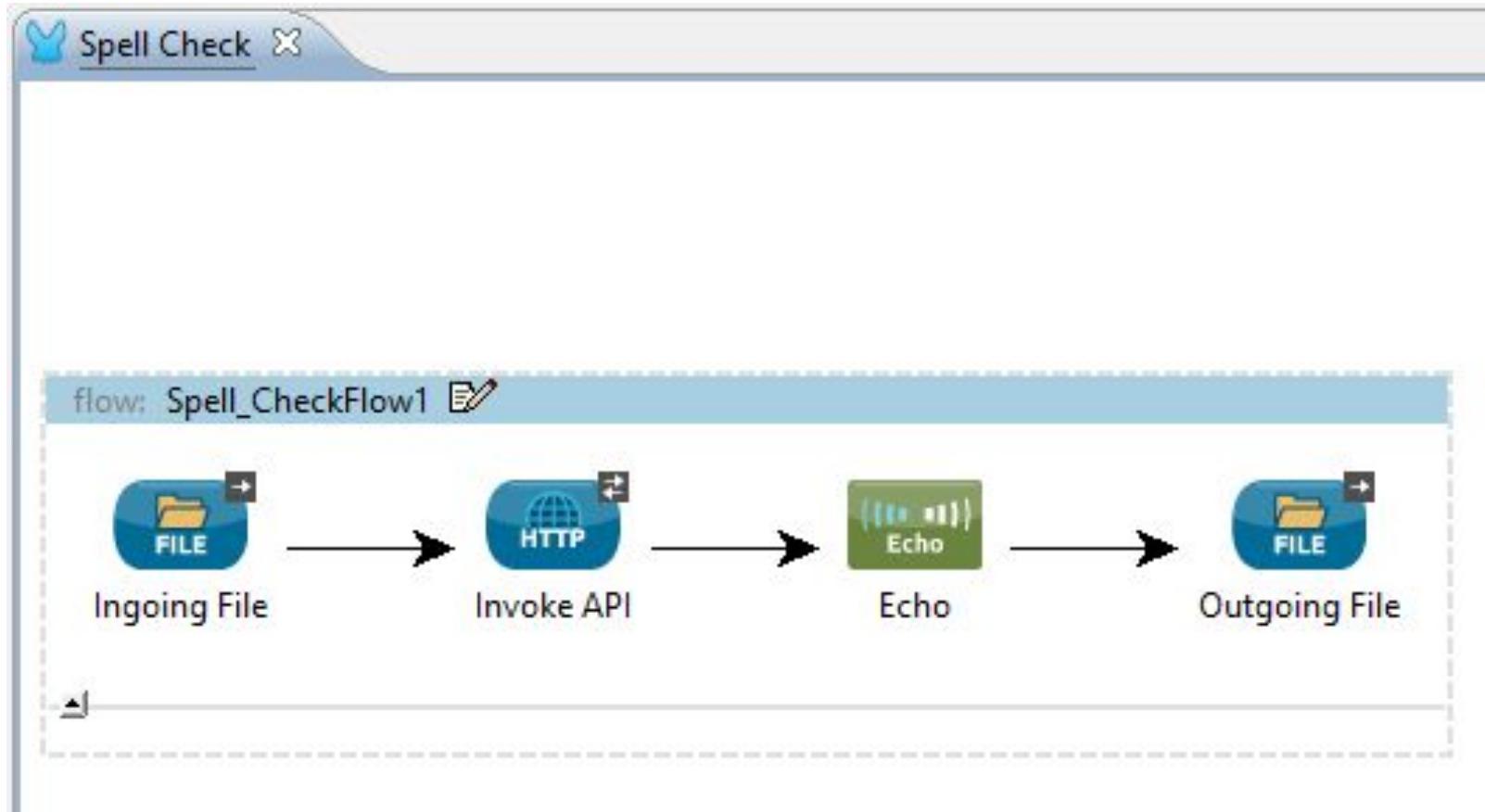
Exemplo MuleSoft - www.mulesoft.com

- Vários componentes e conectores Cloud:



Exemplo MuleSoft - www.mulesoft.com

- Esses endpoints, componentes e serviços permitem criar orquestração de serviços:



Outros Estilos Arquiteturais

Web Standards

RIA – Rich
Internet
Application
Web 2.0

Aplicações
Móveis

Baseada em
Serviços (SOA)

Baseado em
computação em
Grid / P2P /
Componentes
distribuídos

Microsserviços

IoT

Computação em Grid / Cluster

- Computação em **GRID** é um modelo computacional capaz de alcançar uma **alta taxa de processamento** dividindo as tarefas entre diversas máquinas, podendo ser em rede local ou rede de longa distância, que formam uma máquina virtual.
- Eses processos podem ser executados no momento em que as máquinas não estão sendo utilizadas pelo usuário, assim evitando o desperdício de processamento da máquina utilizada.
- Um **cluster** consiste normalmente em computadores fortemente acoplados em uma mesma rede que trabalham em conjunto, de modo que, em muitos aspectos, podem ser considerados como um único sistema. Diferentemente **grid**, computadores em cluster têm cada conjunto de nós para executar a mesma tarefa controlado e programado por software.

Aplicações

- Protocolos (NFS)
- Peer-to-Peer (P2P)
 - Gnutella e BitTorrent – troca de arquivos
 - ICQ e Jabber – comunicação direta
 - SETI@Home – processamento doméstico de radiotelescópios
 - VOIP – Voz sobre IP
- SaaS – Software as a Service
- Mineração de dados
- Mineração de criptomoedas
- Virtualização (XenServer, VMWare)
- Cloud Computing (AWS, Bluemix, Azure, Google Cloud)
- Big Data (Hadoop, ELK)
- Microsserviços (Docker)
- Inteligência Artificial / Deep learning

Exemplo: SETI@home



Exemplo: Peer-to-Peer (P2P)

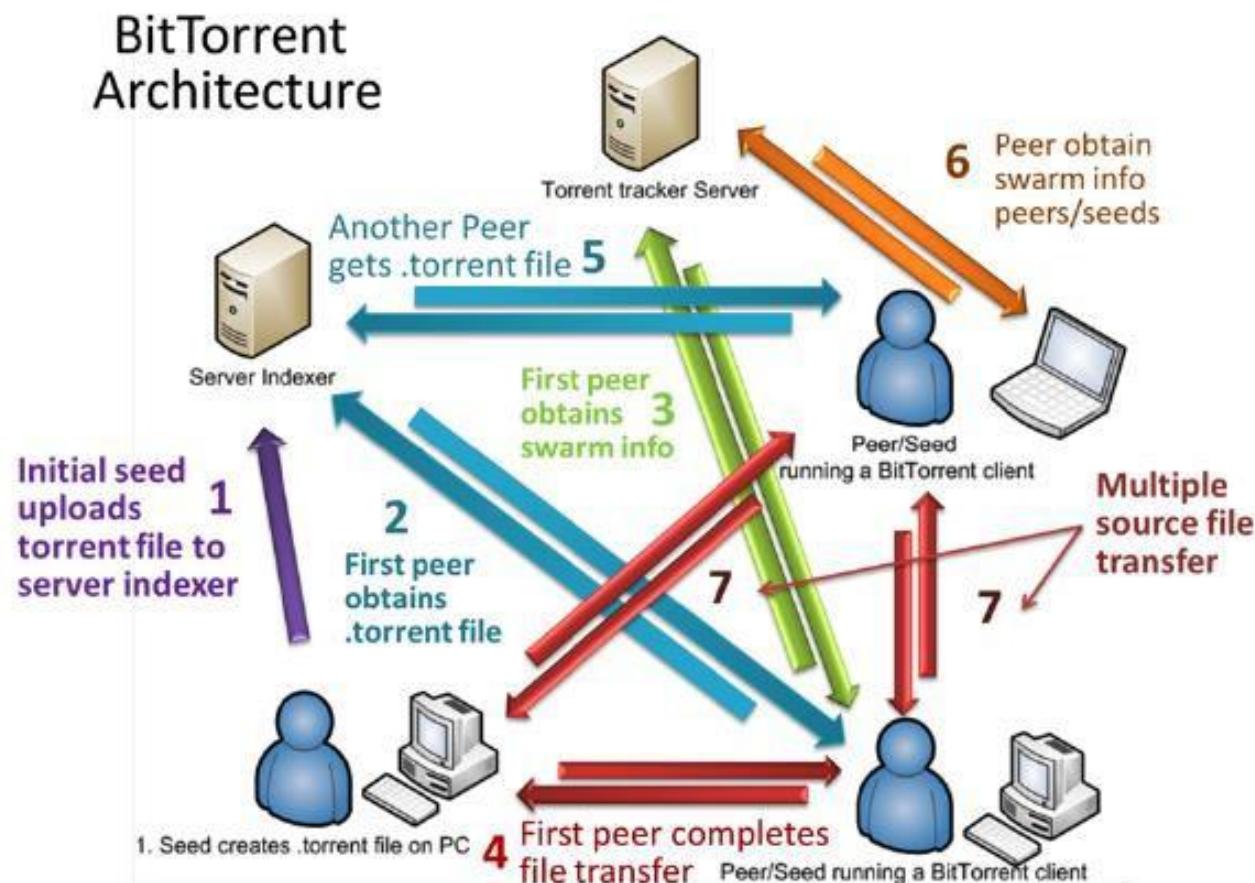
□ Características:

- Descentralizados
- Processamento realizado por qualquer nó da rede
- Sem distinções entre clientes e servidores
- Utiliza todo o poder computacional e armazenamento de uma rede grande
- Cada nó executa uma cópia da aplicação



Exemplo: BitTorrent

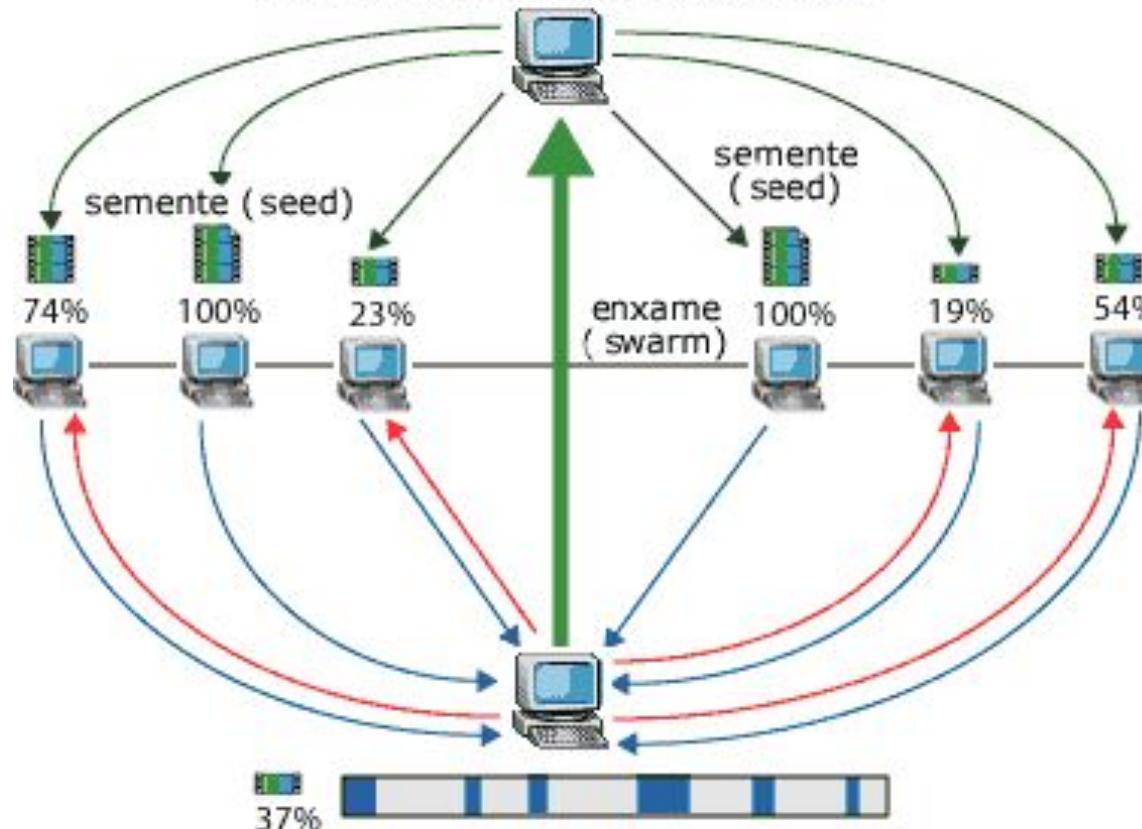
- BitTorrent é um protocolo de rede que permite ao utilizador realizar downloads (descarga) de arquivos indexados. Esse protocolo introduziu o conceito de partilhar o que já foi descarregado, maximizando o desempenho e possibilitando altas taxas de transferência



Exemplo: **BitTorrent**



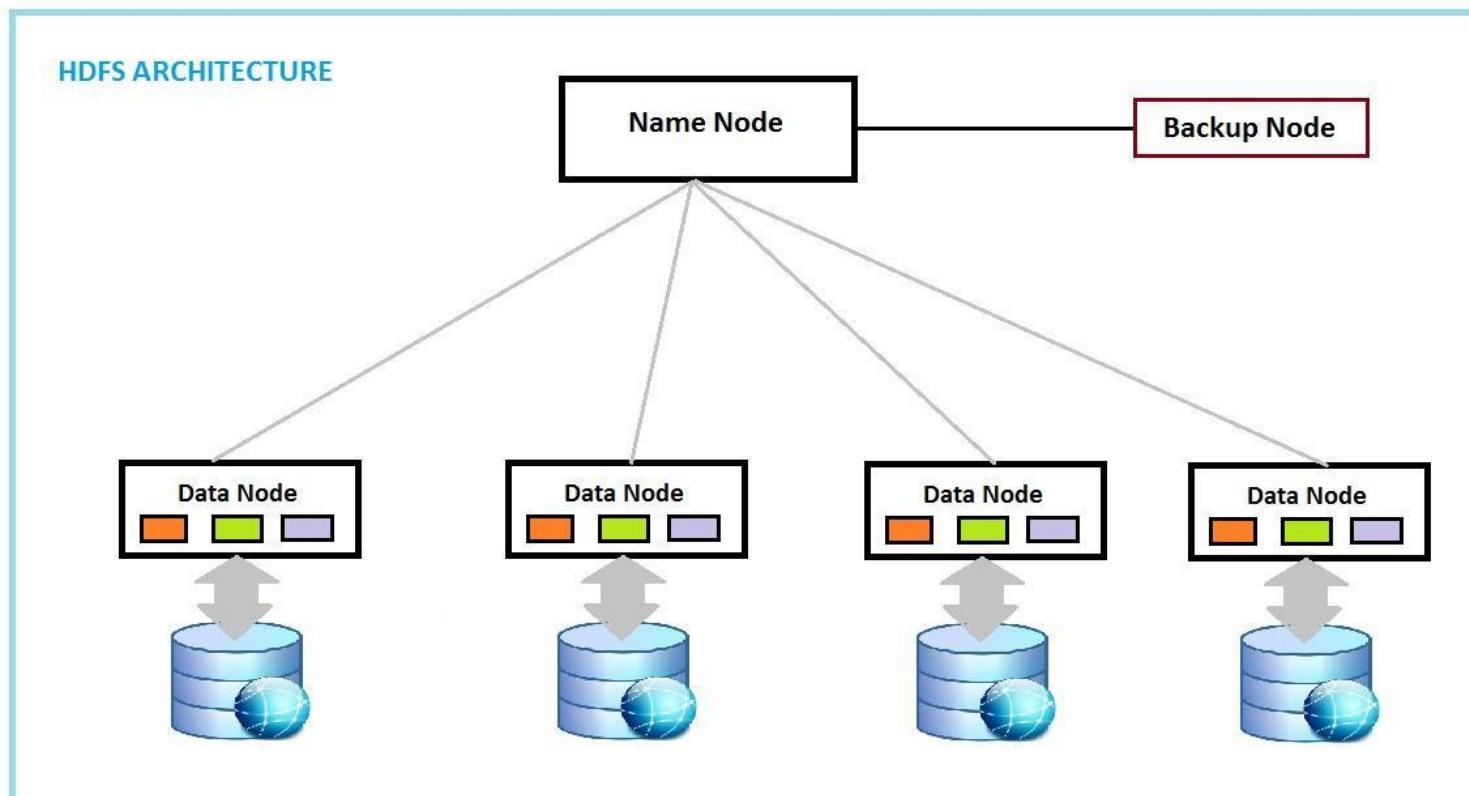
O rastreador (tracker) do BitTorrent identifica o enxame (swarm) e ajuda o programa cliente a trocar fragmentos do arquivo que você quer com outros computadores.



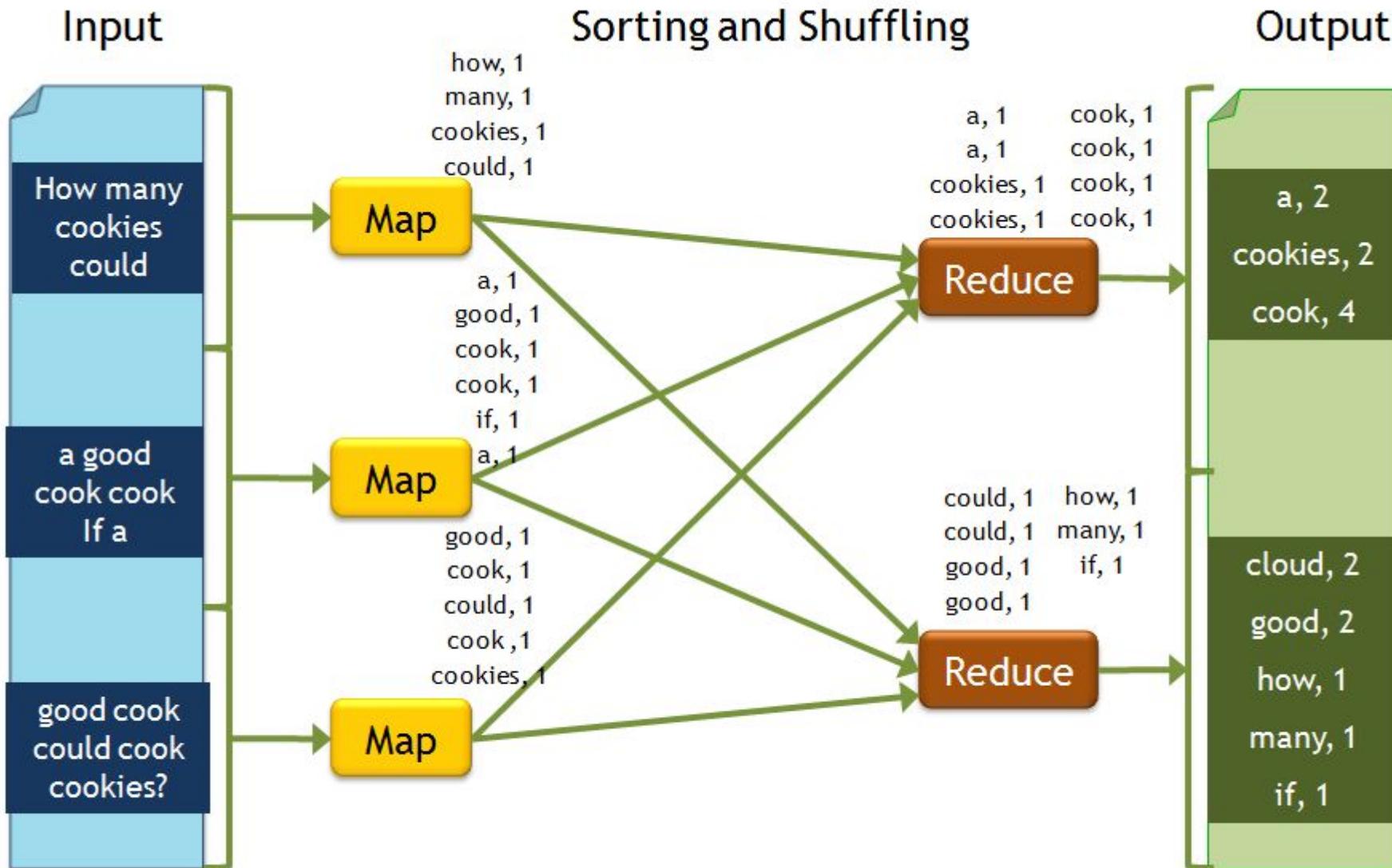
Computador com programa cliente
BitTorrent recebe e envia múltiplos
fragmentos do arquivo simultaneamente.

Exemplo: Hadoop HDFS

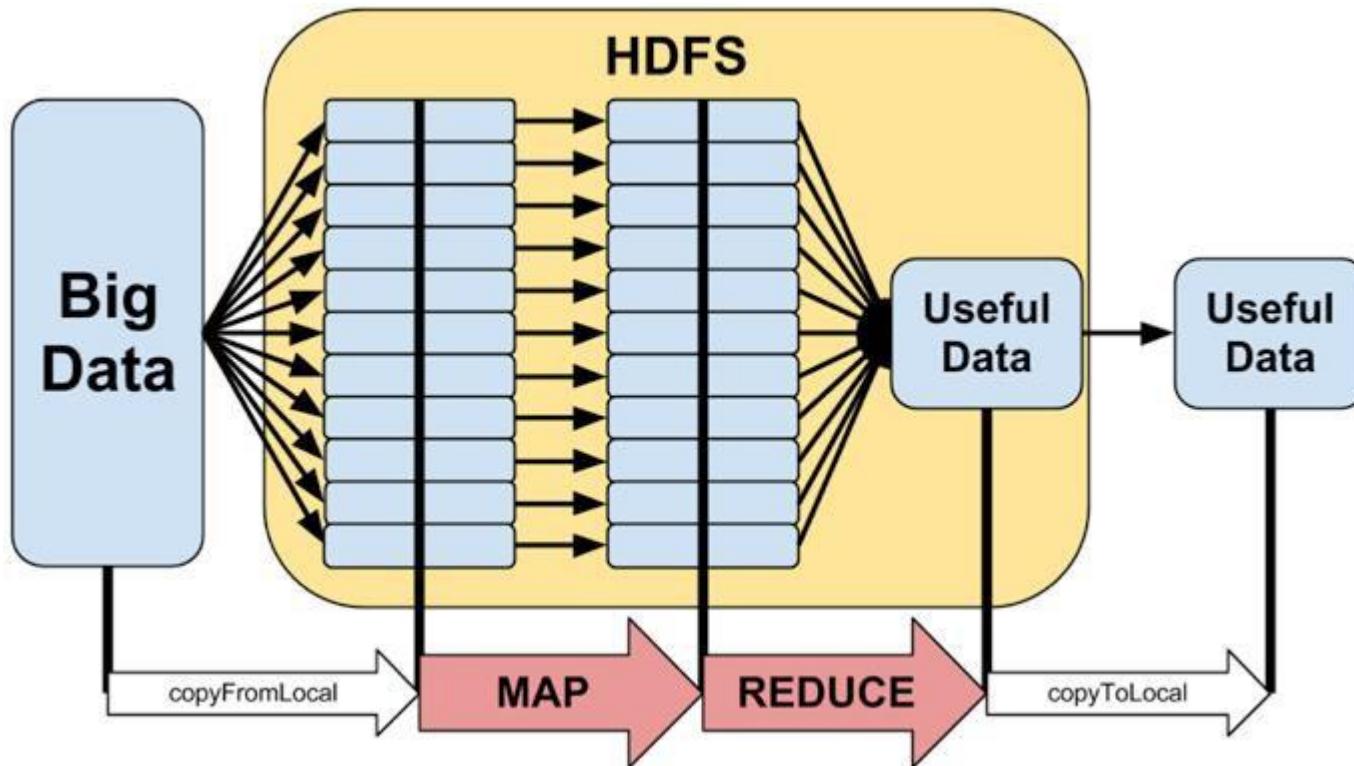
- Hadoop é uma plataforma de software em Java de computação distribuída voltada para clusters e processamento de grandes volumes de dados, com atenção a tolerância a falhas.
- Hadoop Distributed File System (HDFS) - Sistema de arquivos distribuído que armazena dados em máquinas dentro do cluster, sob demanda, permitindo uma largura de banda muito grande em todo o cluster.



Exemplo: Hadoop HDFS



Exemplo: Hadoop HDFS



Exemplo: Elasticsearch



- Elasticsearch é um servidor de buscas distribuído baseado no Apache Lucene. Sua arquitetura possui os seguintes componentes:



Kibana gives shape to your data and is the extensible user interface for configuring and managing all aspects of the Elastic Stack.



Elasticsearch is a distributed, JSON-based search and analytics engine designed for horizontal scalability, maximum reliability, and easy management.

Ingest any data, from any source, in any format.



Beats

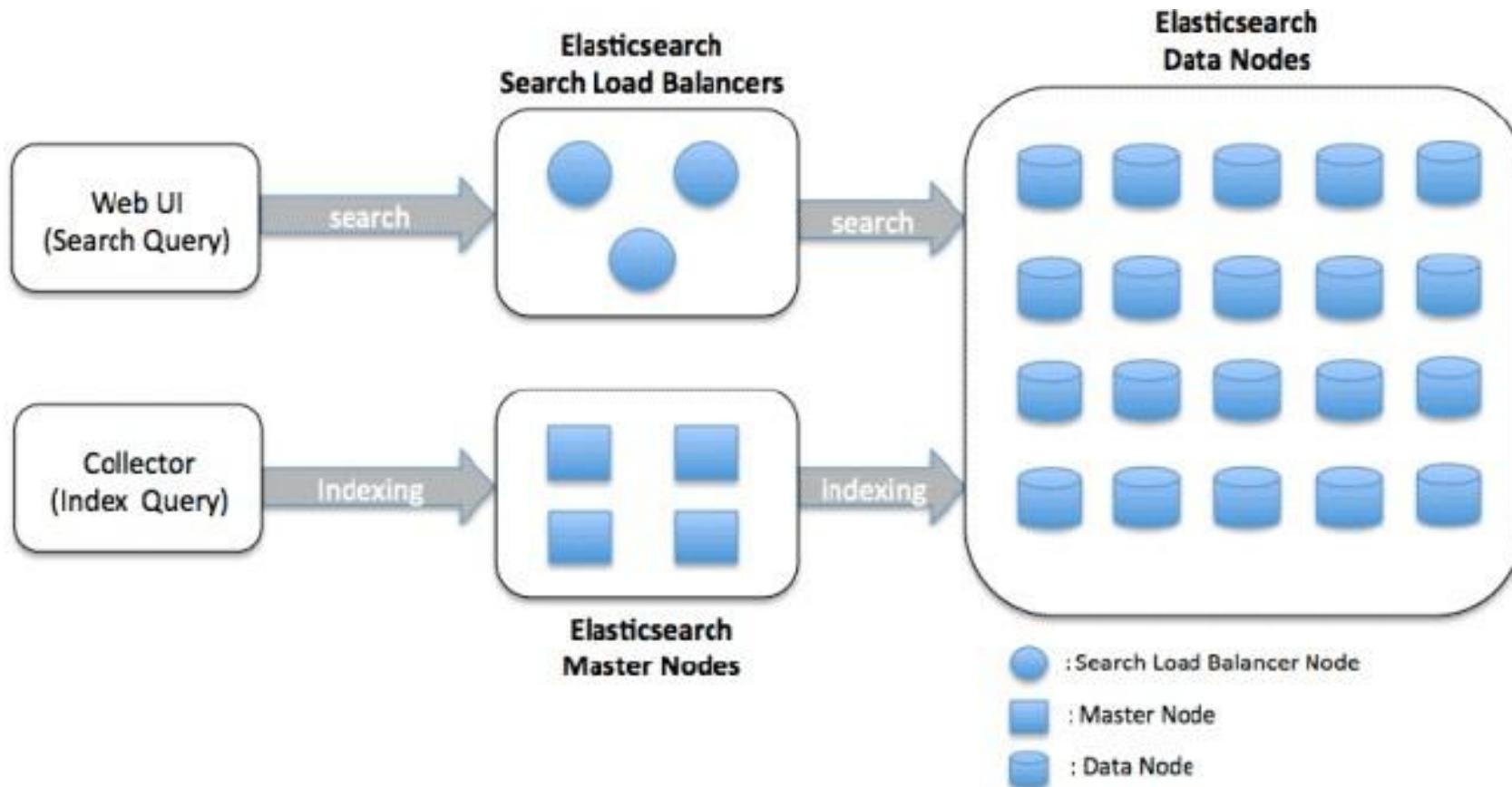
Beats is a platform for lightweight shippers that send data from edge machines to Logstash and Elasticsearch.

Logstash

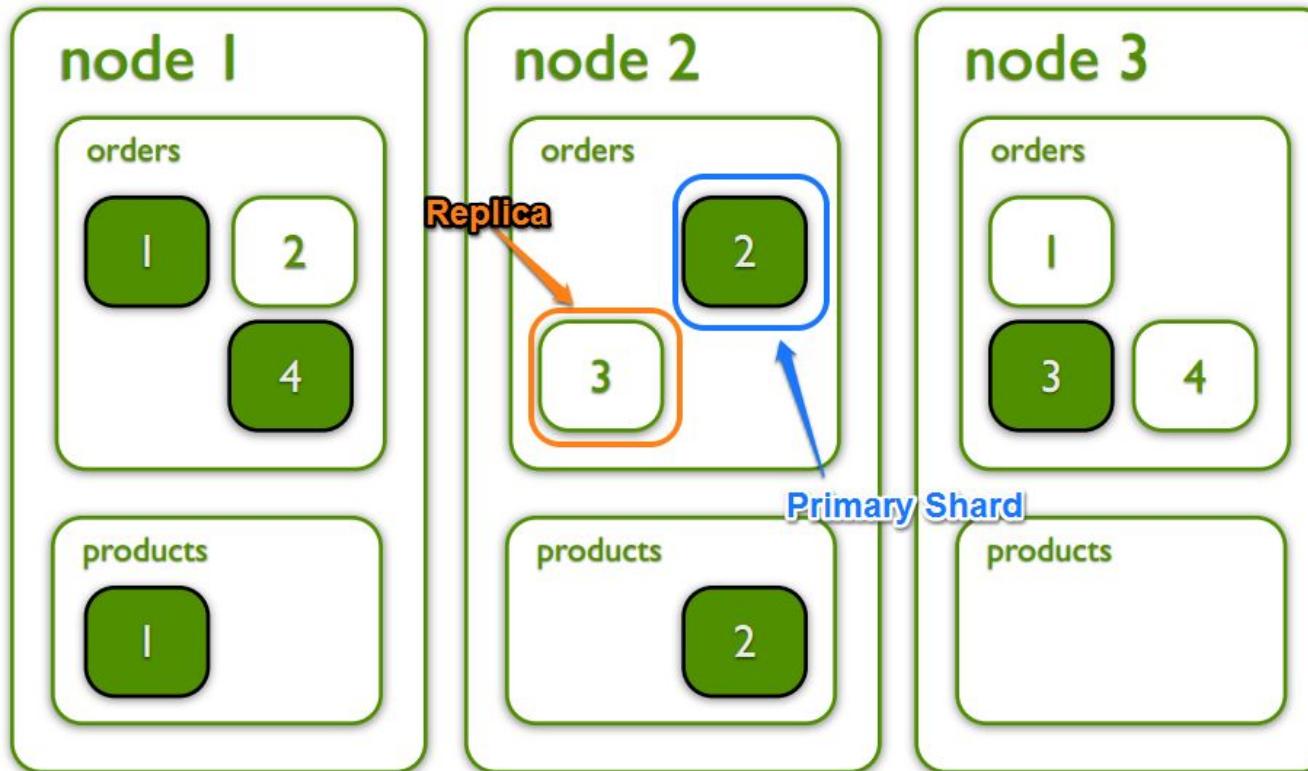
Logstash is a dynamic data collection pipeline with an extensible plugin ecosystem and strong Elasticsearch synergy.



Exemplo: Elasticsearch



Exemplo: Elasticsearch



Exemplo: Elasticsearch



	shop_de_v1 shards: 5 * 3 docs: 169119 size: 54.59MB de	shop_nl_v1 shards: 5 * 3 docs: 131858 size: 33.57MB nl	shop_pl_v1 shards: 5 * 3 docs: 130069 size: 37.16MB pl
★ spelwiese inet [REDACTED] XlzfWekStqP9JISuCJnIQ load: 0.00 heap: 222.97MB/332.25MB	<div>0</div> <div>1</div> <div>2</div> <div>3</div> <div>4</div>	<div>0</div> <div>1</div> <div>2</div> <div>3</div> <div>4</div>	<div>0</div> <div>1</div> <div>2</div> <div>3</div> <div>4</div>
★ nl-1 inet [REDACTED] x05f9CNkSBCWmXL8YI2uBA load: 0.00 heap: 1.52GB/7.97GB	<div>0</div> <div>1</div> <div>2</div> <div>3</div> <div>4</div>	<div>0</div> <div>1</div> <div>2</div> <div>3</div> <div>4</div>	<div>0</div> <div>1</div> <div>2</div> <div>3</div> <div>4</div>
★ PL-1 inet [REDACTED] mfB03OhRw2ntD5QwVu0nQ load: 0.00 heap: 95.27MB/3.98GB	<div>0</div> <div>1</div> <div>2</div> <div>3</div> <div>4</div>	<div>0</div> <div>1</div> <div>2</div> <div>3</div> <div>4</div>	<div>0</div> <div>1</div> <div>2</div> <div>3</div> <div>4</div>
❗ unassigned shards			

Baseado em computação em Grid / P2P

□ Outros exemplos:

- Content Distribution Networks (CDNs)
- DNS
- Torrent
- Jogos
- Netflix
- Spotify
- Hadoop HDFS
- Elasticsearch
- Blockchain

Outros Estilos Arquiteturais

Web Standards

RIA – Rich
Internet
Application
Web 2.0

Portais / CMS

GED / ECM

Aplicações
Móveis

Baseado em
Processos de
Negócio (BPMS)

Baseada em
Serviços (SOA)

Baseado em
Regras de
Negócio (BRMS)

Aplicações de
Processamento
BATCH

Aplicações
Georreferenciadas
(GIS)

Tomada de
Decisão e BI
(DW/DM)

MDM / Data Hub
/ Data Lake

BigData /
Analytics

Baseado em
computação em
Grid / P2P /
Componentes
distribuídos

Microsserviços

IoT

Microsserviços

□ Quando usar:

- Quando você pretende criar aplicações cujas partes possam ser reutilizadas, desenvolvidas separadamente e/ou escaladas individualmente

□ Vantagens:

- Reutilização, escalabilidade em grão fino, desenvolvimento isolado de cada microsserviço
- Existem plataformas para automatizar e prover o ambiente de execução

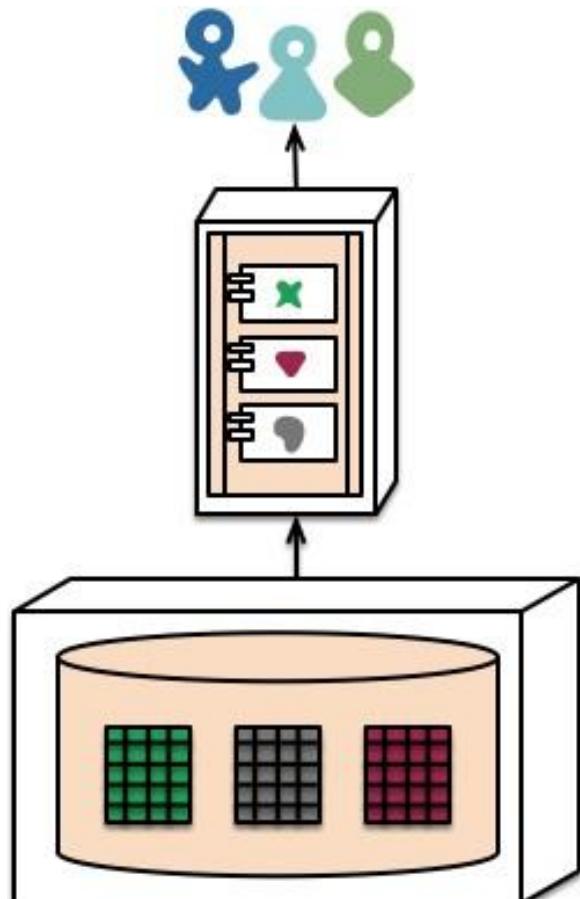
□ Desvantagens:

- Adiciona maior complexidade ao desenvolvimento
- Mudança de cultura do desenvolvimento de aplicações

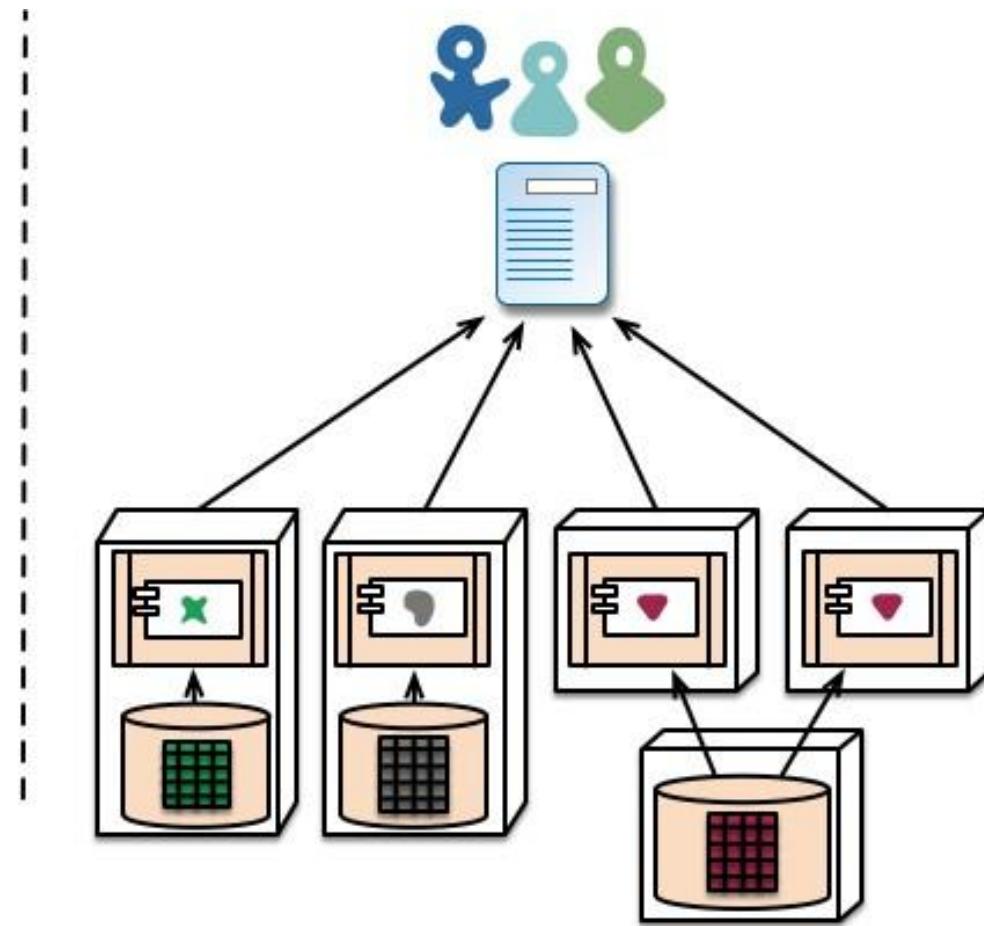
Microsserviços

- O estilo de arquitetura de microsserviços é uma abordagem que desenvolve um aplicativo único como uma **suite de pequenos serviços**, cada um executando seu próprio processo e se comunicando através de mecanismos leves, muitas vezes em uma API com recursos HTTP.
- Esses serviços são construídos em torno de capacidades de negócios e funcionam através de mecanismos de deploy independentes totalmente automatizados.
- Há o mínimo possível de gerenciamento centralizado desses serviços, que podem ser escritos em diferentes linguagens de programação e utilizam diferentes tecnologias de armazenamento de dados.

Microservices



monolith - single database



microservices - application databases

Microsserviços

□ Plataformas:

- Conheça: <https://github.com/mfornos/awesome-microservices>
- Docker
- Cisco Microservices
- Cocaine
- Deis
- Fabric8
- Hook.io
- Lattice
- Lightbend
- Netflix OSS
- Spring Cloud Netflix
- VAMP

Docker

- O Docker é um projeto de código aberto que automatiza a implantação de aplicativos dentro de contêineres de software, fornecendo uma camada adicional de abstração e automação de virtualização de nível de sistema operacional no Linux.



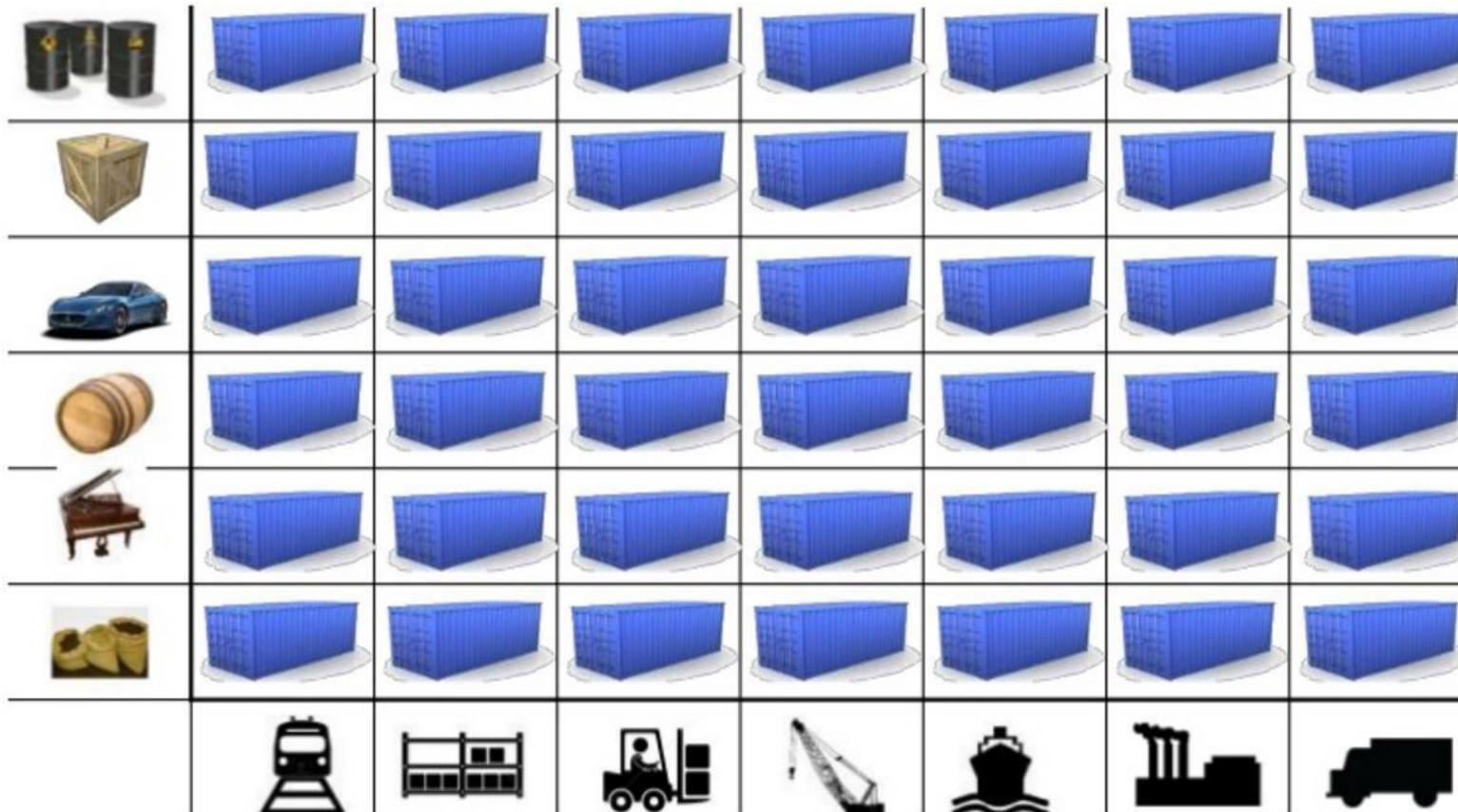
Docker: motivação

- Complexidade e diversidade das configurações de software atualmente

Static website	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Background workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?
	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers
							

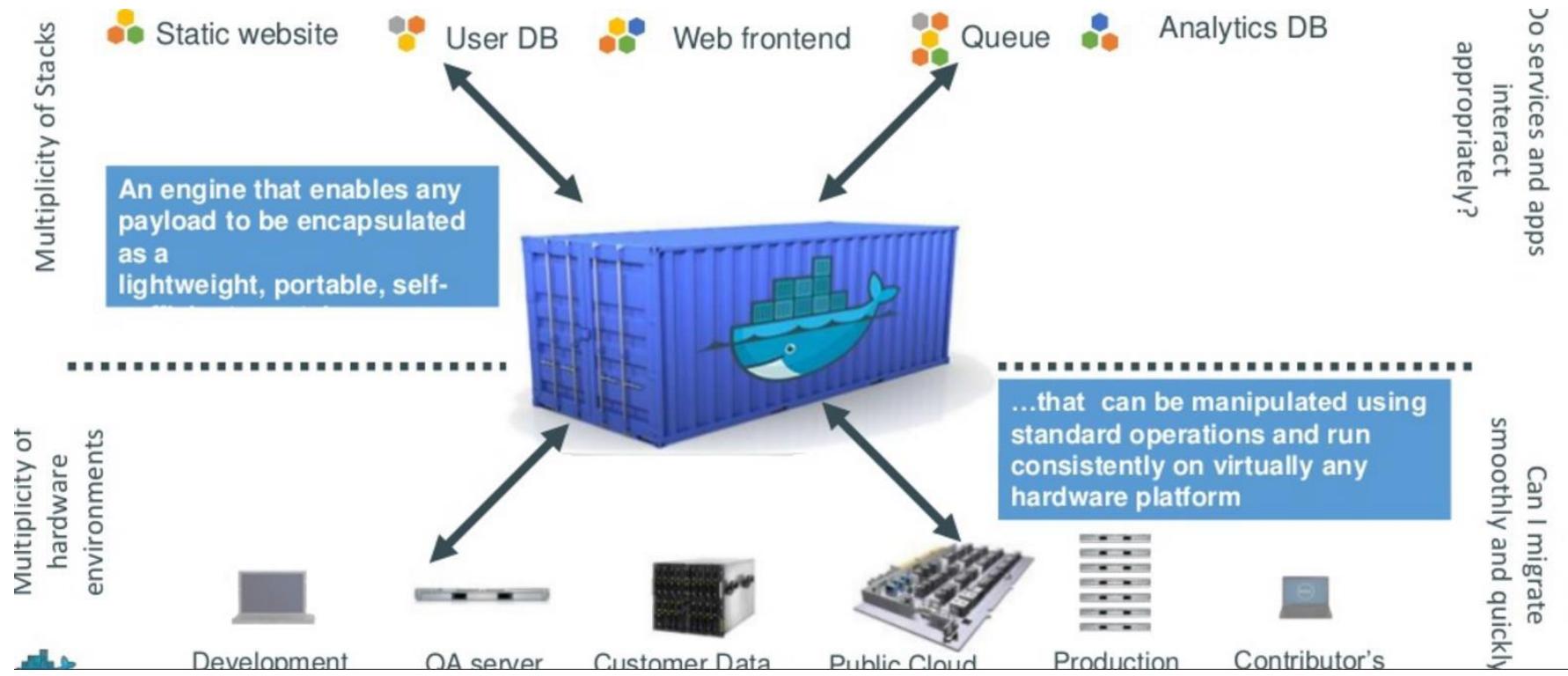
Docker: motivação

- No caso de produtos físicos isso foi resolvido com containers



Docker: motivação

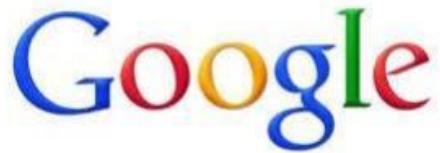
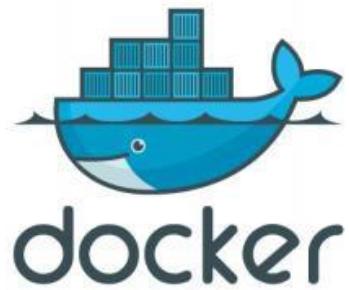
- Por quê não usar containers em software?





Docker: motivação

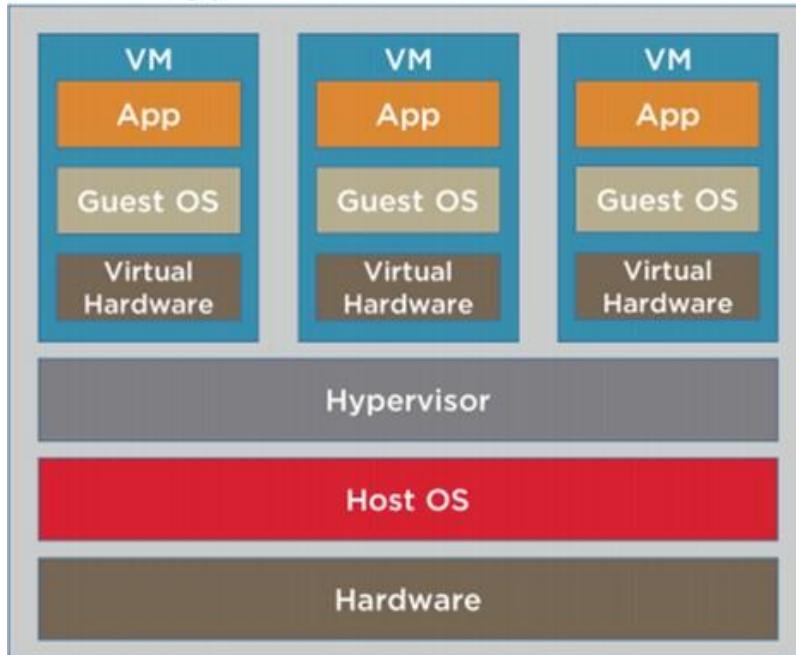
- E usufruir a computação distribuída através das nuvens existentes?



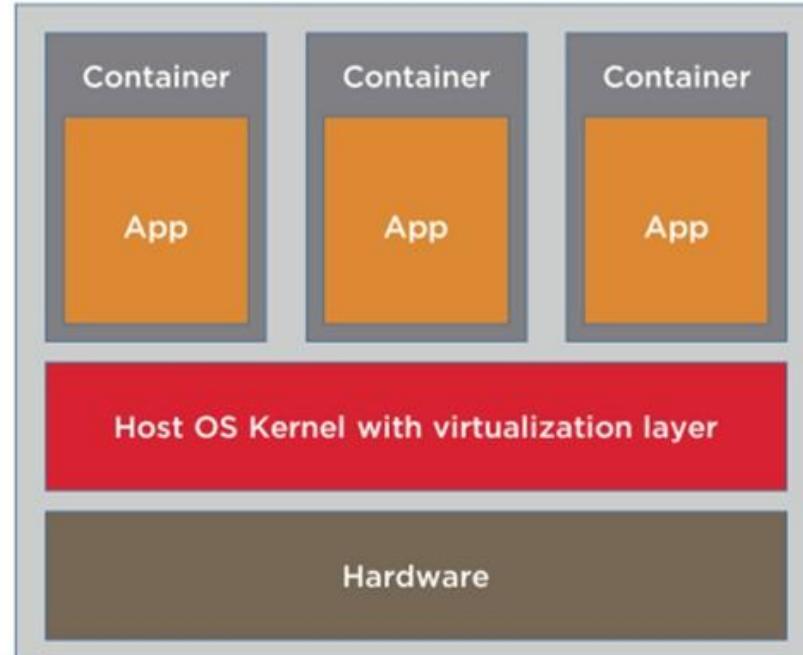


Docker vs. Virtual Machine

Hypervisor virtualization

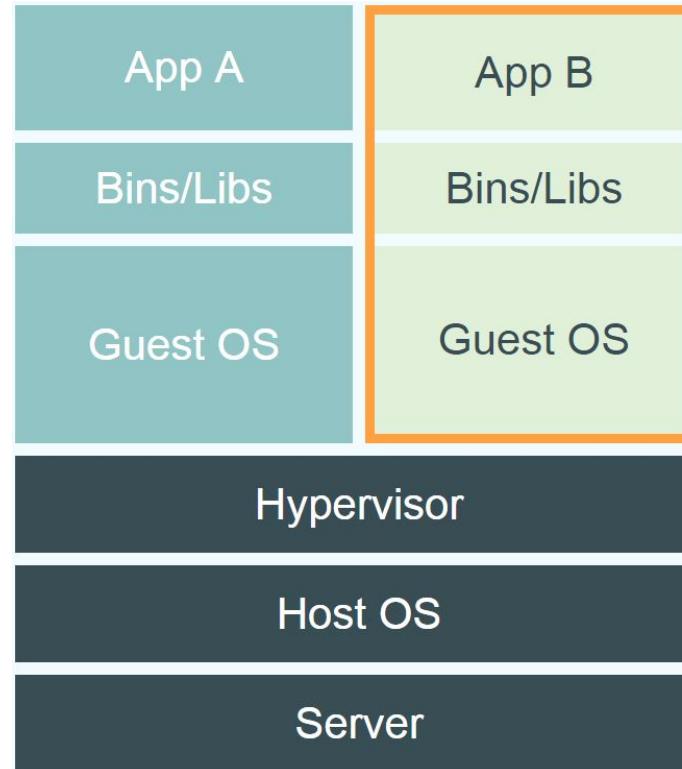
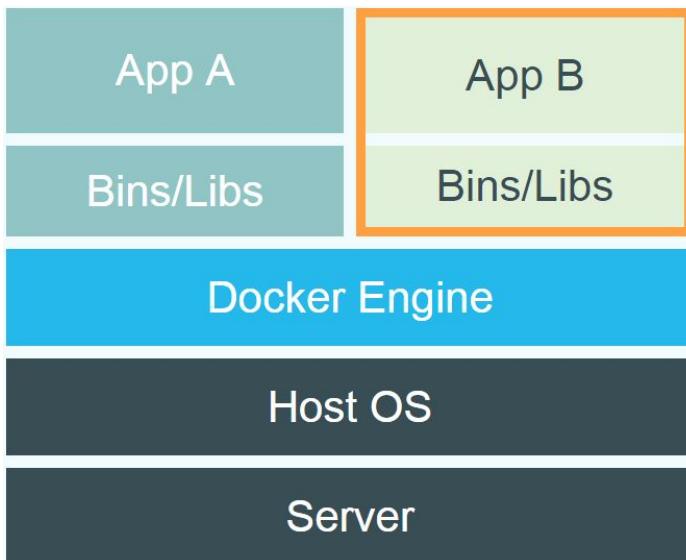


Container virtualization

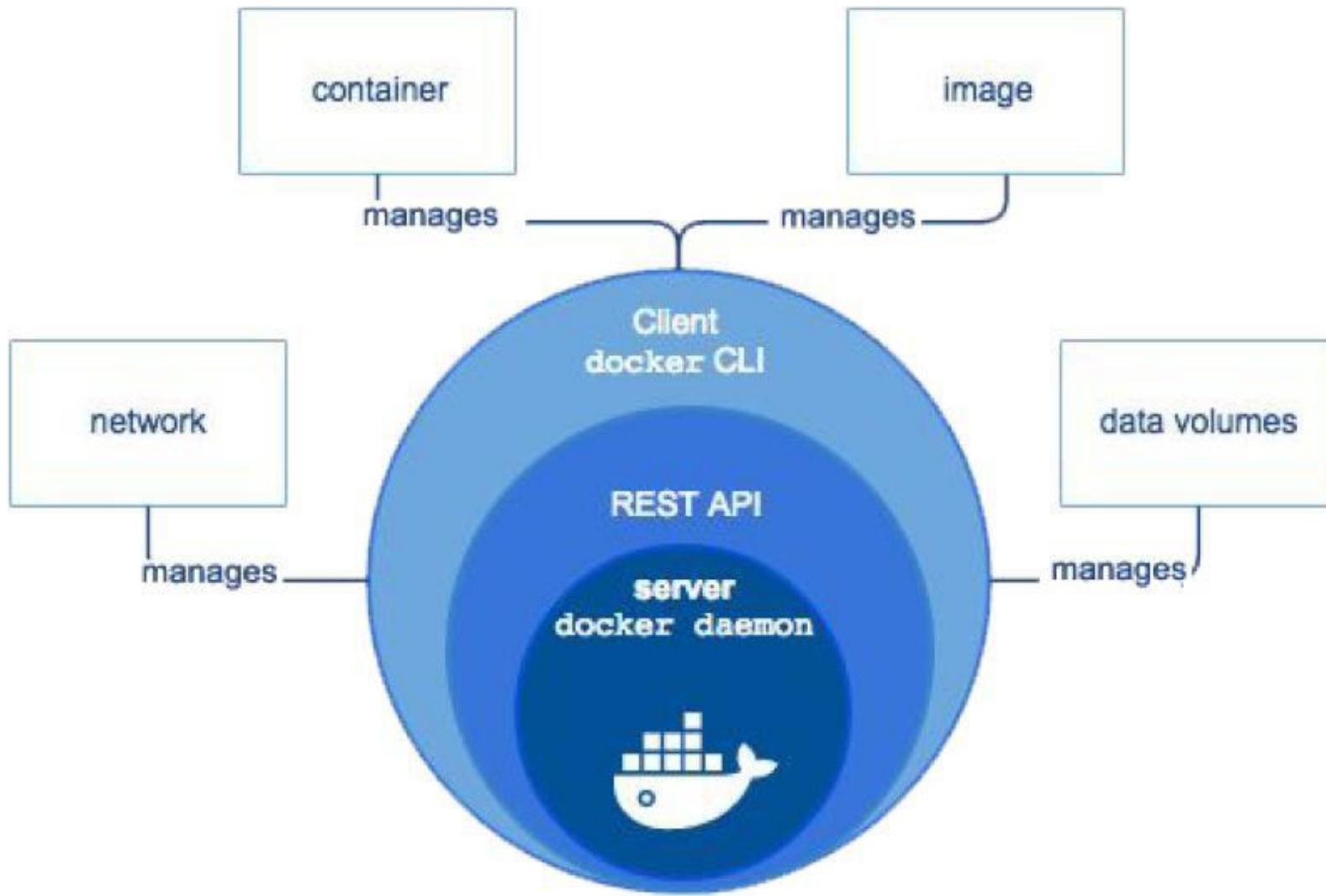




Docker vs. Virtual Machine

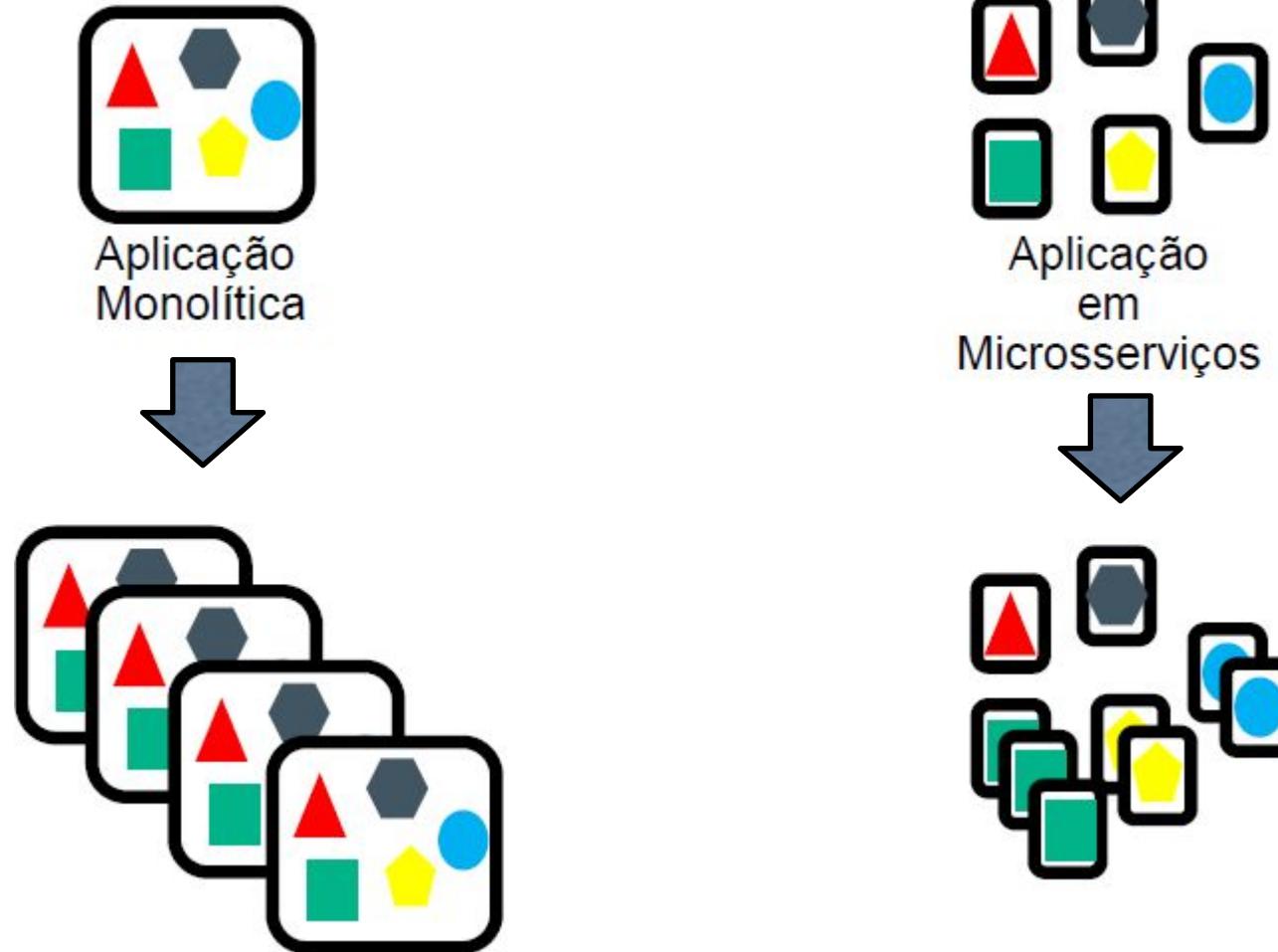


Arquitetura Docker



Microsserviços

- “Abordagem de desenvolvimento de uma única aplicação como sendo uma unidade da ‘diminuta’ das suas componentes”





Microcontainers

- Microcontainers = Microsserviços + Tecnologia de Container
- Que usa microsserviços?





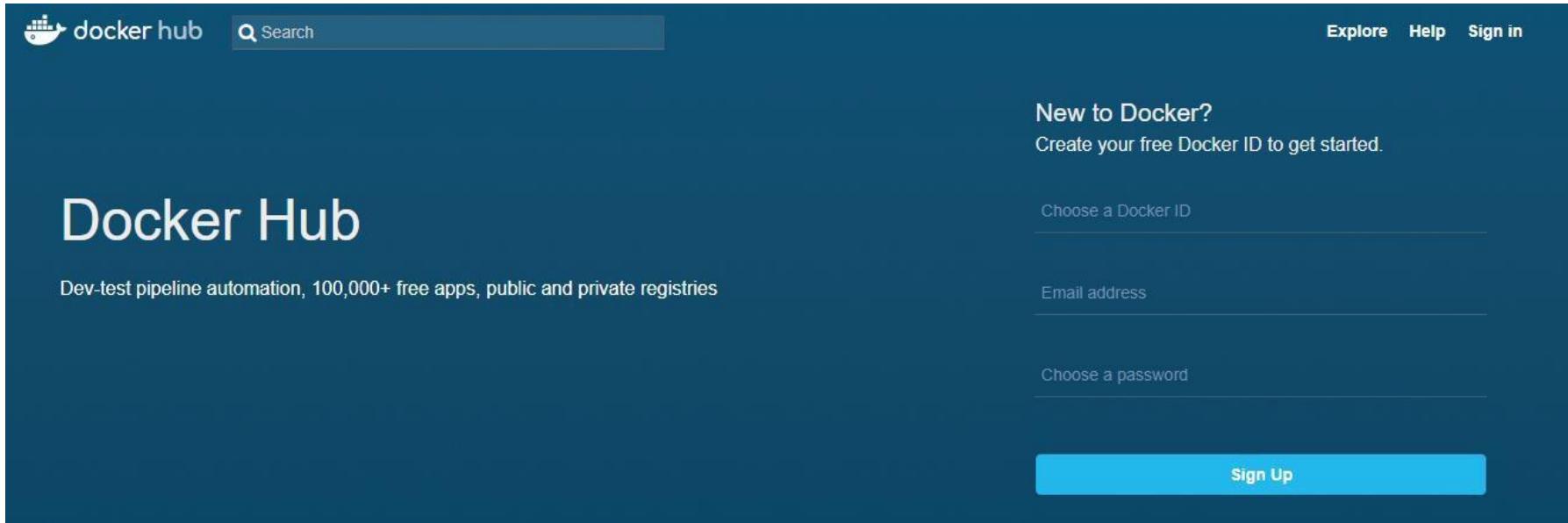
Docker: How To

- Documentação oficial:
 - <https://docs.docker.com/>
- Comandos básicos:
 - <https://imasters.com.br/desenvolvimento/comandos-basicos-docker/?trace=1519021197&source=single>
 - <https://woliveiras.com.br/posts/comandos-mais-utilizados-no-docker/>



<https://hub.docker.com>

- Não comece do zero, utilize um container do Hub



The screenshot shows the Docker Hub sign-up page. At the top left is the Docker Hub logo and a search bar. At the top right are links for "Explore", "Help", and "Sign in". The main heading "Docker Hub" is in large white letters. Below it, a sub-headline reads "Dev-test pipeline automation, 100,000+ free apps, public and private registries". To the right, there's a "New to Docker?" section with a "Create your free Docker ID to get started." button. This section includes fields for "Choose a Docker ID", "Email address", and "Choose a password". A prominent blue "Sign Up" button is at the bottom of the form.



Ecossistema

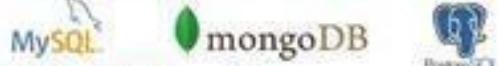
Service Providers



Dev Tools



Official Repositories



Operating Systems



Configuration Management



Big Data



Service Discovery



Orchestration



System Integrators



Instalando do Docker

- Para sistemas com suporte nativo a Hyper-V (exemplo: Windows Pro / Enterprise)
 - <https://www.docker.com/>
- Para sistemas sem suporte a Hyper-V (exemplo: Windows Home / Single Language)
 - <https://www.docker.com/products/docker-toolbox>





kubernetes

Kubernetes <https://kubernetes.io>

- ☐ Kubernetes é um sistema free para automatizar implantação, dimensionamento e gerenciamento de aplicativos em

Automatic binpacking

Automatically places containers based on their resource requirements and other constraints, while not sacrificing availability. Mix critical and best-effort workloads in order to drive up utilization and save even more resources.

Horizontal scaling

Scale your application up and down with a simple command, with a UI, or automatically based on CPU usage.

Automated rollouts and rollbacks

Kubernetes progressively rolls out changes to your application or its configuration, while monitoring application health to ensure it doesn't kill all your instances at the same time. If something goes wrong, Kubernetes will rollback the change for you. Take advantage of a growing ecosystem of deployment solutions.

Storage orchestration

Automatically mount the storage system of your choice, whether from local storage, a public cloud provider such as GCP or AWS, or a network storage system such as NFS, iSCSI, Gluster, Ceph, Cinder, or Flocker.

Self-healing

Restarts containers that fail, replaces and reschedules containers when nodes die, kills containers that don't respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve.

Service discovery and load balancing

No need to modify your application to use an unfamiliar service discovery mechanism. Kubernetes gives containers their own IP addresses and a single DNS name for a set of containers, and can load-balance across them.

Secret and configuration management

Deploy and update secrets and application configuration without rebuilding your image and without exposing secrets in your stack configuration.

Batch execution

In addition to services, Kubernetes can manage your batch and CI workloads, replacing containers that fail, if desired.

Outros Estilos Arquiteturais

Web Standards

RIA – Rich
Internet
Application
Web 2.0

Portais / CMS

GED / ECM

Aplicações
Móveis

Baseado em
Processos de
Negócio (BPMS)

Baseada em
Serviços (SOA)

Baseado em
Regras de
Negócio (BRMS)

Aplicações de
Processamento
BATCH

Aplicações
Georreferenciadas
(GIS)

Tomada de
Decisão e BI
(DW/DM)

MDM / Data Hub
/ Data Lake

BigData /
Analytics

Baseado em
computação em
Grid / P2P /
Componentes
distribuídos

Cloud Computing

Microsserviços

IoT

IoT – Internet of Things

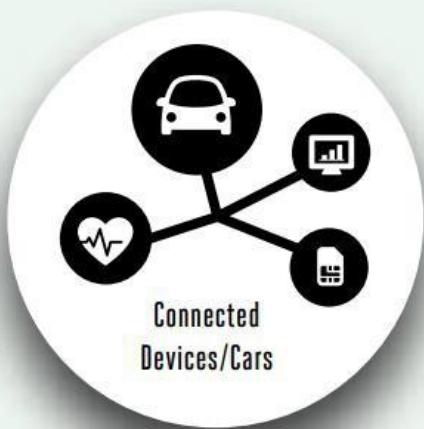
- Quando usar:
 - Quando você pretende adicionar sensores, microcomputadores, atuadores dentre outros dispositivos para compor seu processo de negócio
- Vantagens:
 - Existem várias plataformas, sensores e atuadores de baixo custo
- Desvantagens:
 - Exige conhecimento específico
 - Mudança de cultura do desenvolvimento (necessidade de código otimizado, metodologia de testes, etc.)

IoT – Internet of Things



Evolução constante

On the go...



Connected
Devices/Cars

in the home...



Connected Home

in the city...



Smart Cities

and beyond



Agriculture

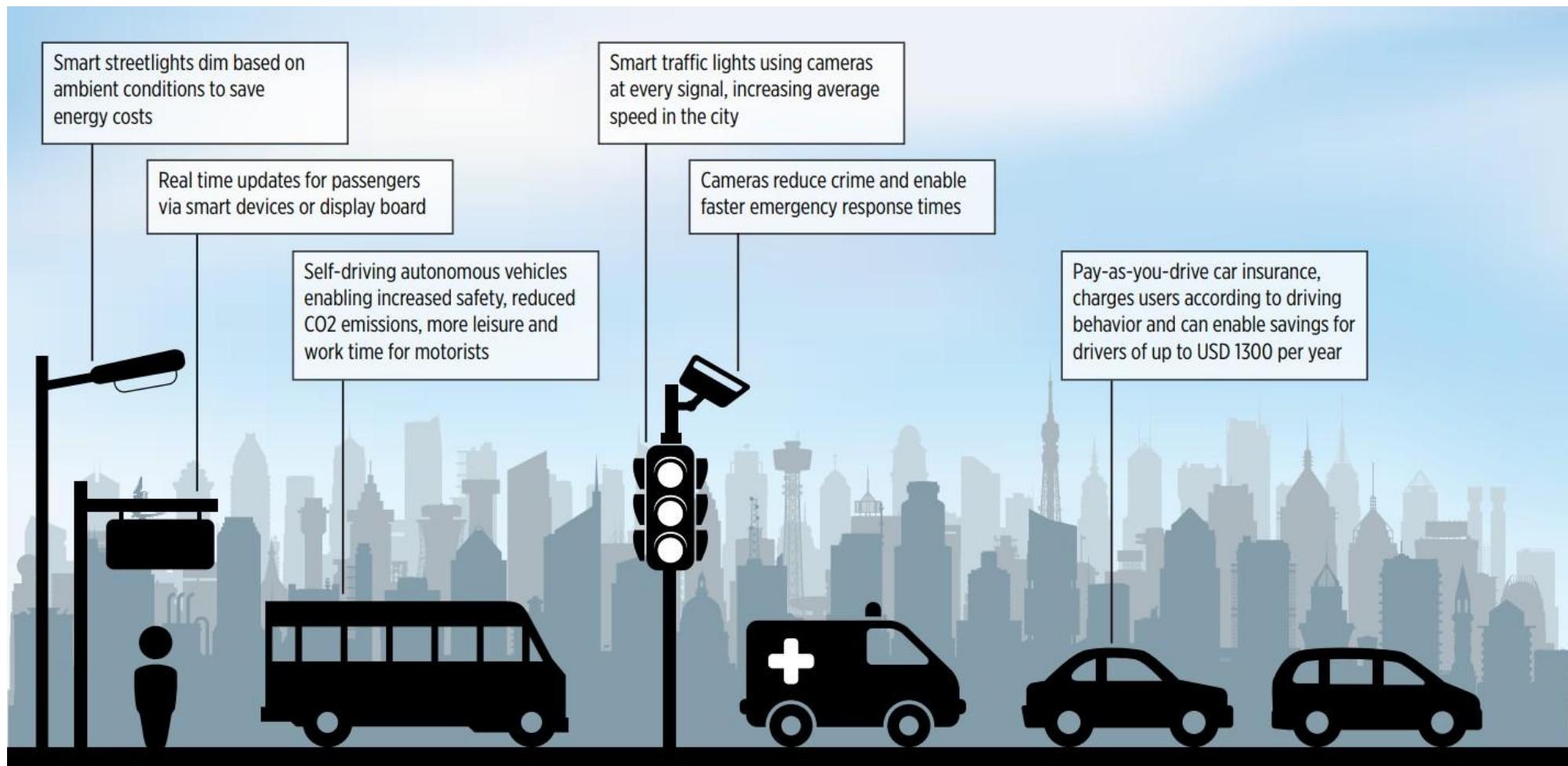
Pervasive **smart connectivity** brings consumers' physical and digital life closer together

Connected **intelligent buildings** bring the benefits home by driving dramatic improvements in (energy) **efficiency and security** and extending benefits of **health** and **education** to the home

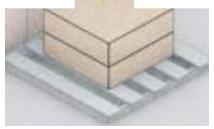
Smart cities ensures a networked urban society shares in the benefits of **intelligent traffic management, smart energy grids and security**

While spreading the benefits to rural areas by enabling innovation in **agriculture** and improving **access to key services** such as **education** and **health**

Smart Cities (Cidades Inteligentes)



Smart Ventilation



bound Delivery IO:
145300415
Product Code:
23704B Location;
T023134 Volume
Count: 2/20 Cond
ition: OK

Warning: forklift ahead



Belt ID: 003B4

Status: gearbox requires immediate maintenance
Action: appointment scheduled



t
•

Outbound Delivery Order ID:
4n9S8B
Order Status:
complete Condition: OK



Smart
Lighting



Smart living



Shipment ID: H5800243

Location: HHDE_7F240

Temperature: 15°C

Humidity: 53%

Light: container opened for customs

Shock: no shock events detected



! Engine sensor: 9R003

Status: serious malfunction detected

Action: maintenance team alerted



Shipment ID: CR55024

Location: HHDE 50331

Speed: 70km/H

ETA: 20:30 04/05/15

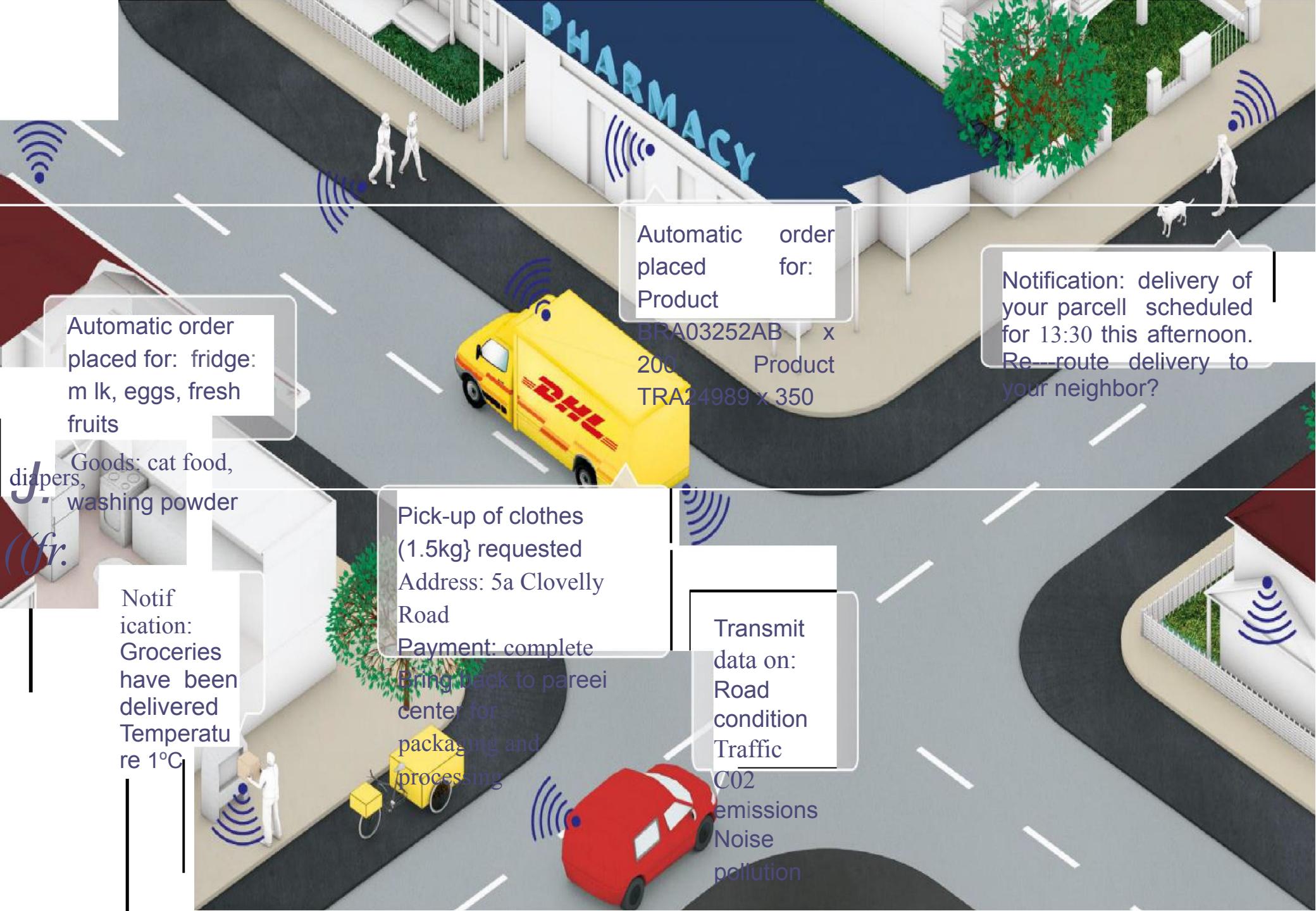
! Material damage likely
Maintenance check scheduled

! Driver fatigue detected
Pull over at next stop

Shipment ID: DE235104

Location: HHDE002-45-8

Temperature: OK



FUTURE FARMS

small and smart



FARMING DATA

The farm generates vast quantities of rich and varied data. This is stored in the cloud. Data can be used as digital evidence reducing time spent completing grant applications or carrying out farm inspections saving on average £5,500 per farm per year.

TEXTING COWS

Sensors attached to livestock allowing monitoring of animal health and wellbeing. They can send texts to alert farmers when a cow goes into labour or develops infection increasing herd survival and increasing milk yields by 10%.

SURVEY DRONES

Aerial drones survey the fields, mapping weeds, yield and soil variation. This enables precise application of inputs, mapping spread of pernicious weed blackgrass could increase wheat yields by 2-5%.

FLEET OF AGRIBOTS

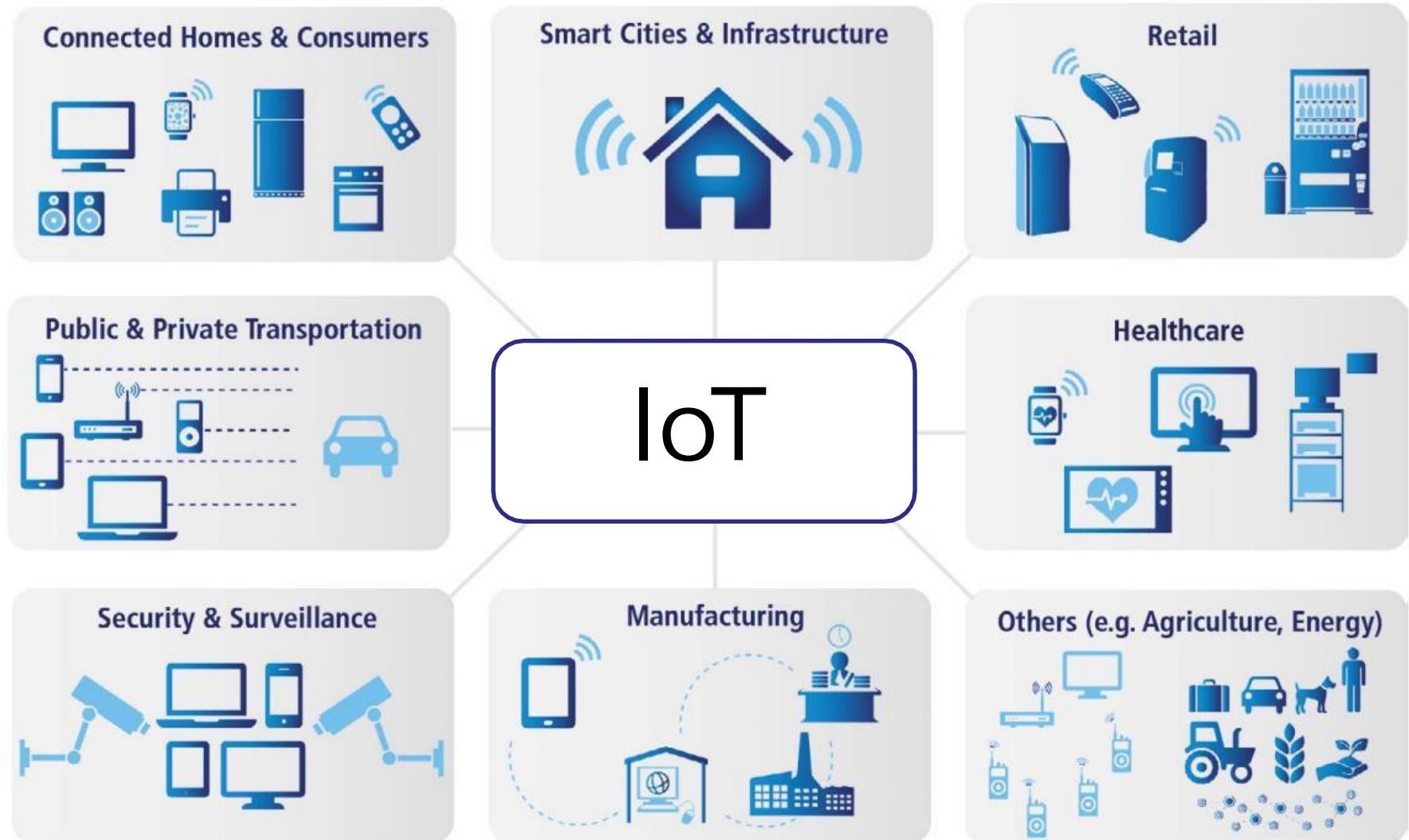
A herd of specialised agribots tend to crops, weeding, fertilising and harvesting. Robots capable of microdot application of fertiliser reduce fertiliser cost by 99.9%.



SMART TRACTORS

GPS controlled steering and optimised route planning reduces soil erosion, saving fuel costs by 10%.

Resumindo...



IoT □ IoE

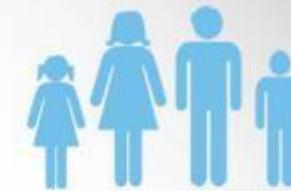
Data

Leveraging data into more useful information for decision making



People

Connecting people in more relevant, valuable ways



Things

Physical devices and objects connected to the Internet and each other for intelligent decision making, often called **Internet of Things (IoT)**



Process

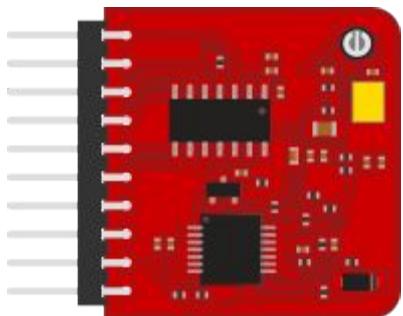
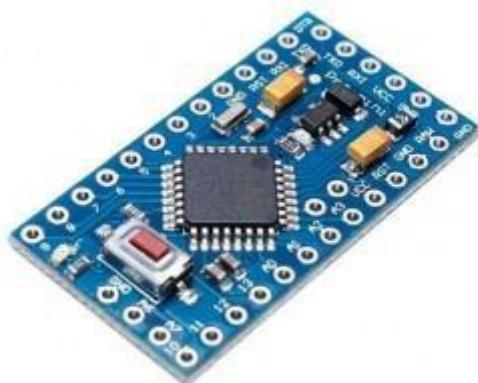
Delivering the right information to the right person (or machine) at the right time



IoT – Internet of Things

□ Plataformas:

- Conheça: <https://github.com/HQarroum/awesome-iot>
- Arduino
- BeagleBoard
- HummingBoard
- Intel Galileo
- Microduino
- Node MCU
- OLinuXino
- Odroid
- Particle
- Pinoccio
- Raspberry Pi
- Tessel
- UDOO



23