



SÃO  
PAULO  
TECH  
SCHOOL

# Engenharia de Software

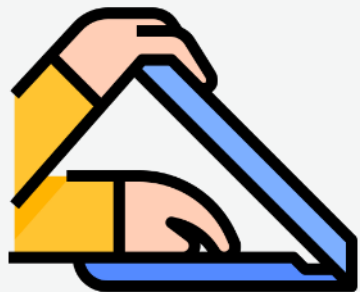
## Arquitetura de Software

Aula 9

**Fábio Figueredo**

[fabio.figueredo@sptech.school](mailto:fabio.figueredo@sptech.school)

# Regras básicas da sala de aula



1. **Notebooks Fechados:** Aguarde a liberação do professor;
2. Celulares em modo **silencioso e guardado**, para não tirar sua atenção
  - Se, caso haja uma situação urgente e você precisar **atender ao celular**, peça licença para sair da sala e atenda fora da aula.



3. **Proibido usar Fones de ouvido:** São liberados apenas com autorização do professor.

4. **Foco total no aprendizado**, pois nosso tempo em sala de aula é precioso.

- Venham sempre com o **conteúdo da aula passada em mente** e as atividades realizadas.
- **Evitem faltas** e **procure ir além** daquilo que lhe foi proposto.
- **Capricho, apresentação e profundidade** no assunto serão observados.
- **“frequentar as aulas e demais atividades curriculares aplicando a máxima diligência no seu aproveitamento”** (Direitos e deveres dos membros do corpo discente - Manual do aluno, p. 31)



# Regras básicas da sala de aula



As aulas podem e devem ser divertidas! Mas:

- **Devemos respeitar uns aos outros** – cuidado com as brincadeiras.
  - “observar e cumprir o regime escolar e disciplinar e comportar-se, dentro e fora da Faculdade, **de acordo com princípios éticos condizentes**” (Direitos e deveres dos membros do corpo discente – Manual do aluno, p. 31)

# Boas práticas no Projeto

## COMPROMISSO



COM VOCÊ:  
**ARRISQUE**, NÃO  
TENHA MEDO DE  
ERRAR



COM OS  
PROFESSORES:  
ORGANIZE A **ROTINA**  
PARA OS ESTUDOS

COM OS COLEGAS:  
**PARTICIPAÇÃO**  
**ATIVA** E PRESENTE



COM O PROJETO:  
**RESPEITO** E  
FLEXIBILIDADE

  
*Respeito*

# Boas práticas no Projeto

Reações **defensivas** não levam  
ao envolvimento verdadeiro!

Transforme cada problema e  
cada dificuldade em uma  
**OPORTUNIDADE** de aprendizado  
e crescimento.

## EVITE:

- Justificativas e Desculpas
- Transferir a culpa
- Se conformar com o que sabe
- Se comparar com o outro

# Dica: **Como ter sucesso** (Maiores índices de aprovações)

## Comprometimento

- Não ter faltas e atrasos. Estar presente (*Não fazer 2 coisas ao mesmo tempo*)
- Fazer o combinado cumprindo os prazos

## Atitudes Esperadas:

- **Profissionalismo**: Entender que não é mais ensino médio (*Atitude, comportamento, etc.*)
- **Não estar aqui só pelo** estágio ou pelo diploma
- Não ficar escondido: precisa **experimentar**
- **Trabalhar** em grupo e **participar** na aula
- **Não ser superficial** ou “achar que sabe”
- **Não se enganar** utilizando de “cola”
- Assumir a responsabilidade: Não colocar a culpa em outra coisa. **Não se vitimizar.**

# Avaliações

**Socioemocional: Binária** (reprova ou aprova). *Feedbacks durante o semestre.*

**Pesquisa e Inovação: Binária** (reprova ou aprova). *Feedback no final de cada Sprint.*

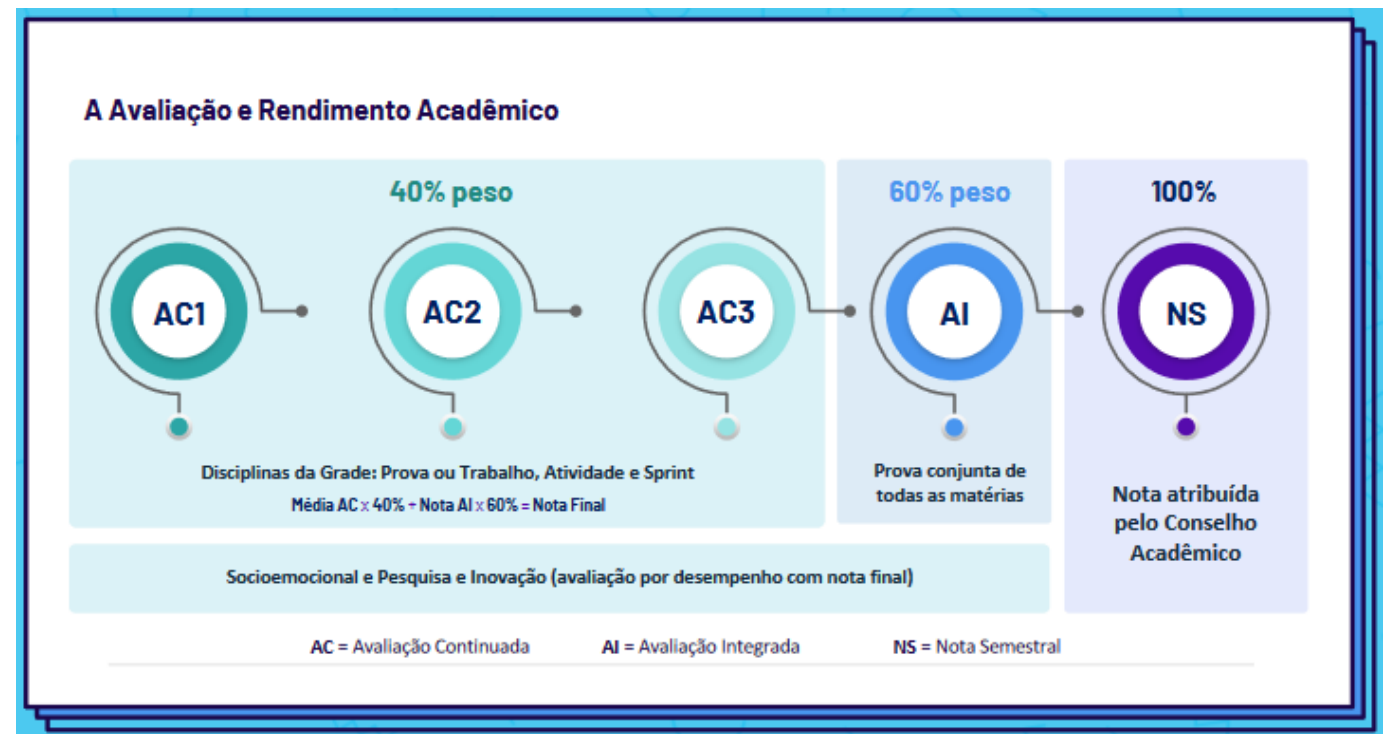
**Média = 6**

## 40% da Nota – Continuadas

*1 Continuada por Sprint com possibilidade de inspeção Individual (São 3 continuadas)*

## 60% da Nota

*1º Semestre = Projeto Individual ;  
Demais semestres: Avaliação Integrada*



Manual do Aluno



# Nosso Caminho

Legenda: Conteúdo / **Entregável PI** / Onde Estamos



S3

## Final do Semestre

- Qualidade e Testes
- Processos de Software
- Apresentação PI
- Avaliação Integrada

- **Entregável Sprint 3**

Entrega: 19/05/2022

S2

- Design de Interfaces Web
- Projeto de Software
- Arquitetura de Software

- **Desenho de Arquitetura**
- **Protótipo em Alta Resolução**
- **Diagrama de Solução de Software - Container**
- **Planilha de Arquitetura**

Entrega: 25/04/2022

S1

- ~~Apresentação~~
- ~~UI/UX~~
- ~~Fatores Humanos~~
- ~~Design de Interação~~
- ~~Design de Interfaces + Bootcamp~~

- ~~**Jornada do Usuário**~~
- ~~**Prototipação das Telas**~~

Entrega: 03/03/2022



## Break

> 10 minutos, definidos pelo professor.

**Obs:** Permanecer no andar, casos específicos me procurar.

**Atenção:** Atrasados deverão aguardar autorização para entrar na sala.

# Tópicos da Aula

- Modelos de Arquitetura de Software
- Desenho de Arquitetura de Software
- Atividade



Nosso objetivo:

**Aprender/Ensinar processos,  
métodos e ferramentas para  
construção e manutenção de  
softwares profissionais.**



Palavra-chave dessa Sprint:

# PRAGMATISMO

## **prag·má·ti·co**

adjetivo

1. Relativo à pragmática ou ao pragmatismo.
2. Que tem motivações relacionadas com a .ação ou com a eficiência. = PRÁTICO

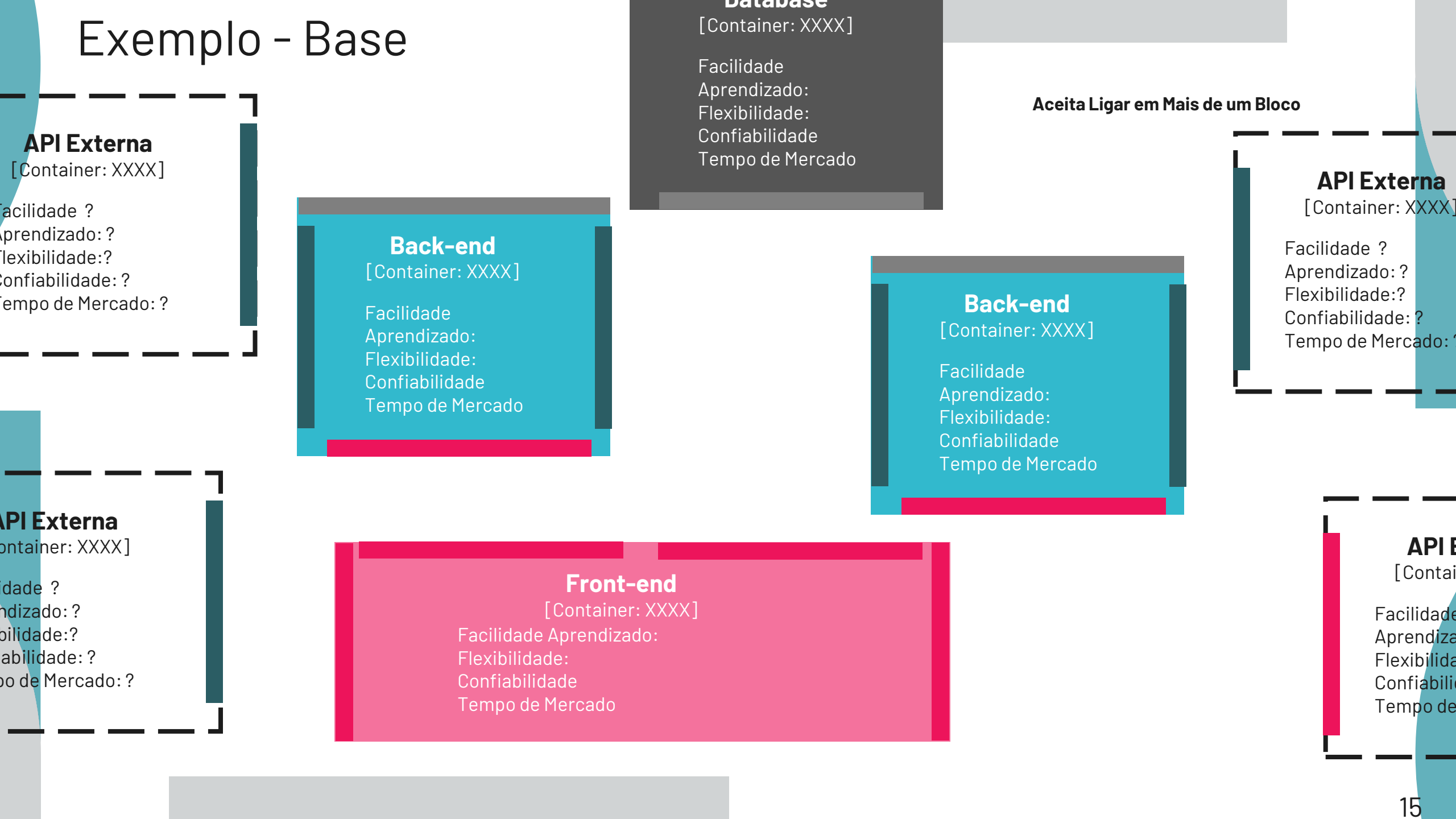
adjetivo e substantivo masculino

3. Que ou quem revela um sentido prático e sabe ou quer agir com eficácia.

The background is a dark, textured surface with a pattern of fine, glowing white lines that swirl and curve across the frame. Interspersed among these lines are numerous small, bright white dots, some of which appear to have a slight starburst or lens flare effect, giving the overall impression of a digital or cosmic network.

**ANTERIORMENTE DE ENGENHARIA  
DE SOFTWARE...**

# Exemplo - Base



The background is a dark, almost black, space filled with intricate, glowing patterns. These patterns consist of numerous thin, white, curved lines that flow and swirl across the frame, creating a sense of movement and depth. Interspersed among these lines are many small, bright white dots and star-like shapes, some of which have a soft, out-of-focus glow. The overall effect is reminiscent of a cosmic or digital landscape, with the lines and stars suggesting a complex, interconnected system.

**Vamos aprofundar...**



# **SWEBOK** Software Engineering Body of Knowledge

## **SWEBOK**

Requisitos de software

Projeto de software

Construção de software

Teste de software

Manutenção de software

Gerência de configuração de software

Gerência de engenharia de software

Processos de Engenharia de Software

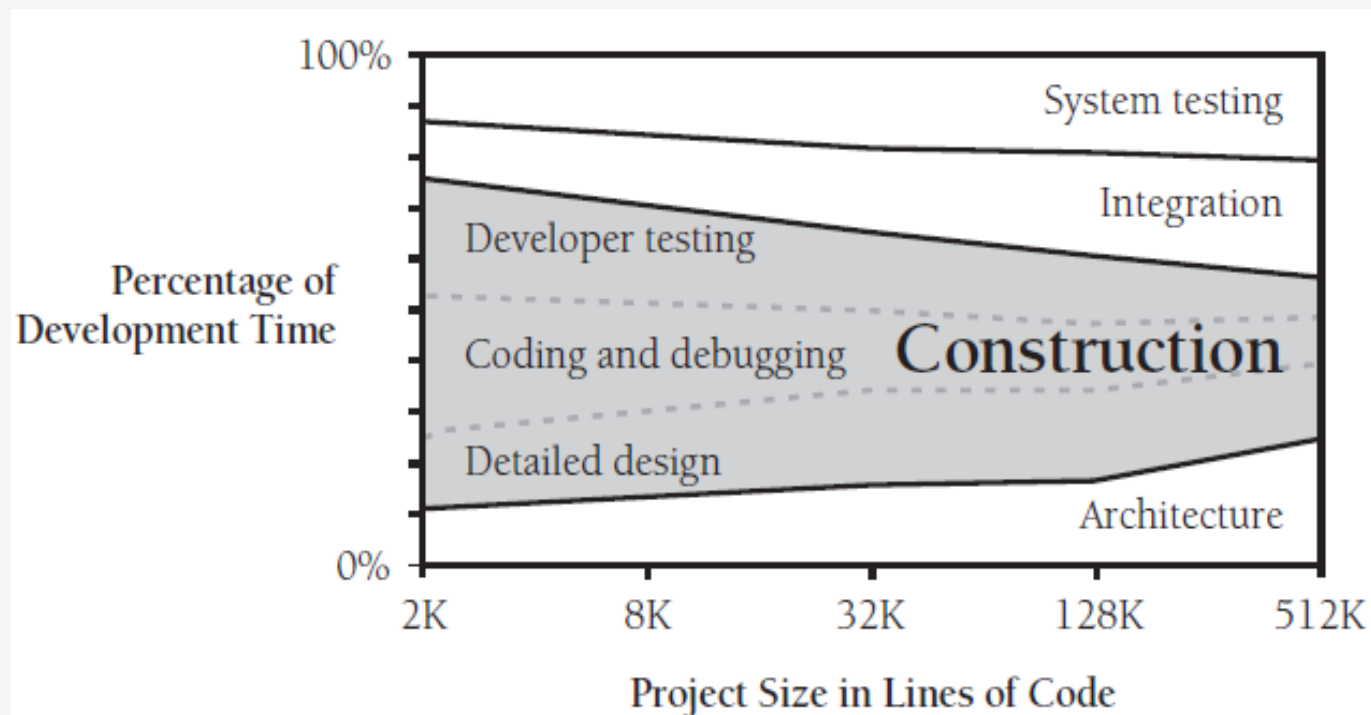
Ferramentas e Métodos de Engenharia de Software

Qualidade de software



**IEEE – Instituto  
de  
Engenheiros,  
Eletricistas e  
Eletrônicos.**

# Por que a Eng. de Software é importante?



Livro *Code Complete* pag 654

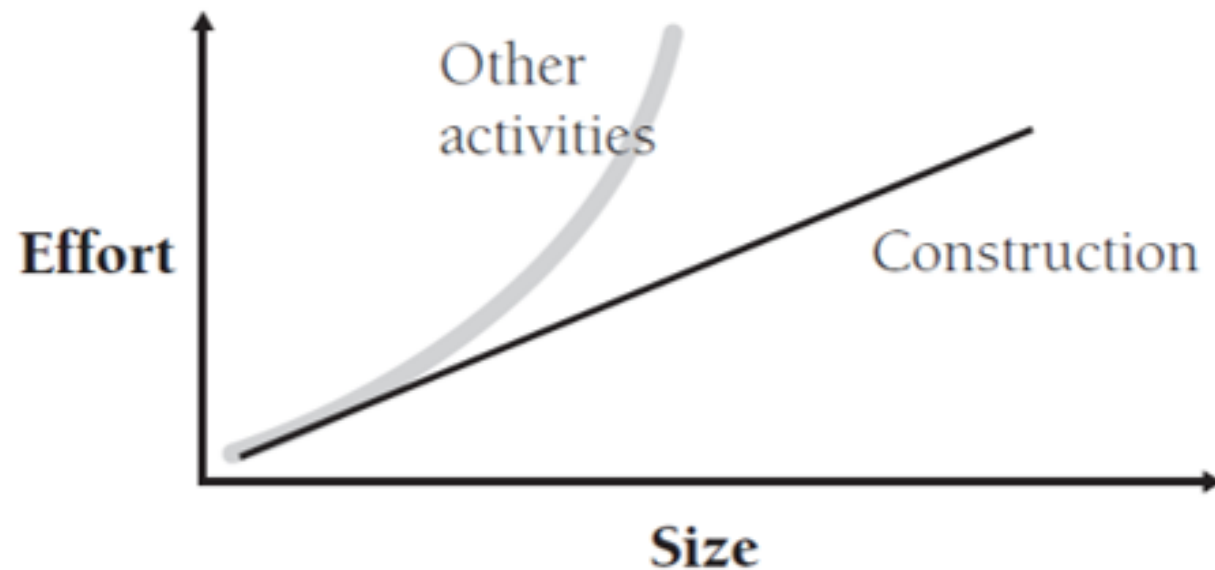
A medida que o tamanho do sistema aumenta, o tempo nas atividades de Arquitetura, Integração e Teste também aumentam.

! O tempo da construção do software reduz proporcionalmente

Atividades que aumentam:

- Comunicação
- Planejamento
- Gerenciamento
- Lev. Requisitos
- Projeto Funcional
- Arquitetura
- Integração
- Remoção de Defeitos
- Testes
- Documentação

# Por que a Eng. de Software é importante?



Quando você faz um castelo a chance de esquecer a porta aumenta muito!



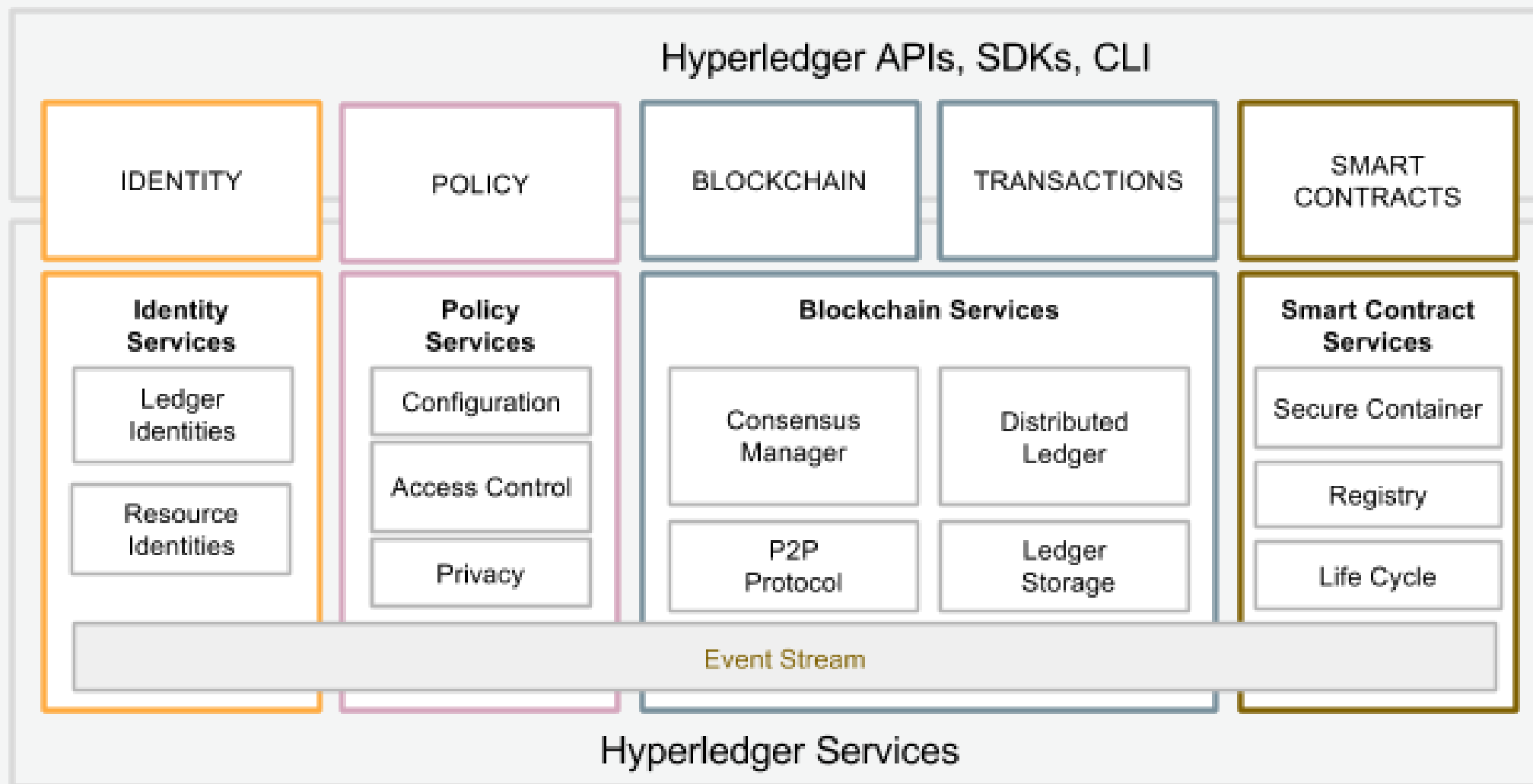
# O Desenho não precisa ser complexo...

Desenho de arquitetura de um sistema....



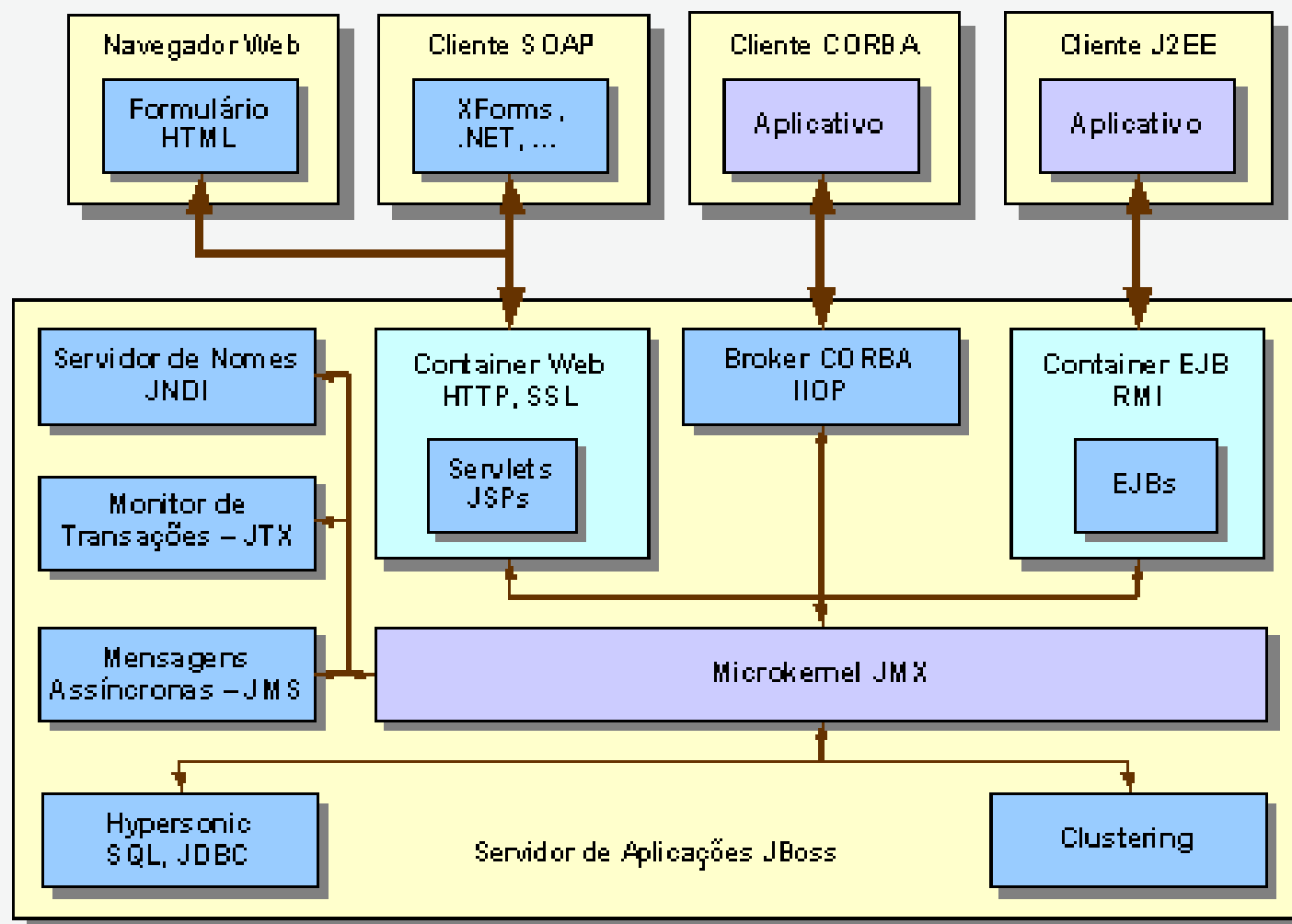
# Mas muitos podem precisar ser...

Desenho de arquitetura de um sistema de Block Chain....



# Mas muitos podem precisar ser...

Desenho de arquitetura do JBoss....



# Perguntas para decidir arquitetura [Sommerville]

1. Existe uma arquitetura conhecida que pode funcionar com o projeto?
2. Como o sistema vai utilizar as diversas capacidades de processamento?
3. Qual o estilo mais apropriado para a solução (cliente-servidor, 2 camadas, 3 camadas, etc).
4. Como será a decomposição do sistema? (módulos)
5. Como será o controle da operação do sistema? Centralizado, baseado em Eventos?
6. Como o projeto de arquitetura será avaliado?
7. Como vai ser a documentação?

...essas são questões fundamentais, outras coisas podem impactar como: mão de obra, maturidade da solução....

**Arquitetura pode tratar de ...**

**PROTOCOLOS DE  
COMUNICAÇÃO**

**EVOLUÇÃO DO SISTEMA**

**INTERAÇÃO ENTRE  
OS COMPONENTES**

***PRODUTOS***

**CONNECTIVIDADE**

**HARDWARE**

***OBJETOS***

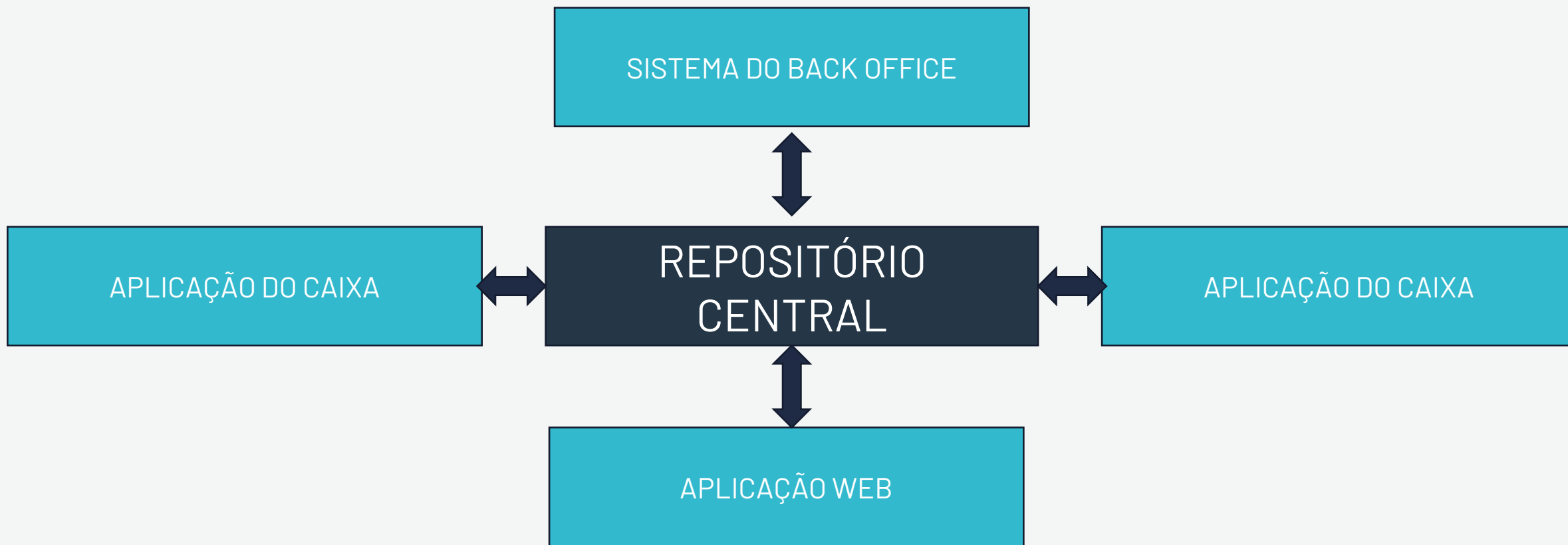
**FABRICANTES**



The background is a dark, textured surface with a complex pattern of thin, white, wavy lines that flow across the frame. Interspersed among these lines are numerous small, bright white dots and larger, out-of-focus circular bokeh lights, creating a sense of depth and movement. The overall effect is reminiscent of a starry night sky or a digital data visualization.

# **Modelos de Arquitetura**

# Modelos de Repositório



# Modelos de Repositório

---

Eficiente para compartilhar grandes quantidades de dados

---

Os sistemas precisam estar de acordo com o modelo de dados

---

Não há necessidade do sistemas saberem como os demais utilizam os dados

---

Evoluir o modelo pode ser difícil

---

Atividades de backup, controle de acesso e recuperação de erros são centralizadas

---

As políticas (ex: backup) são impostas para todos os sistemas consumidores

---

Um sistema novo ou ferramenta nova podem ser integradas diretamente (usando o modelo de dados)

---

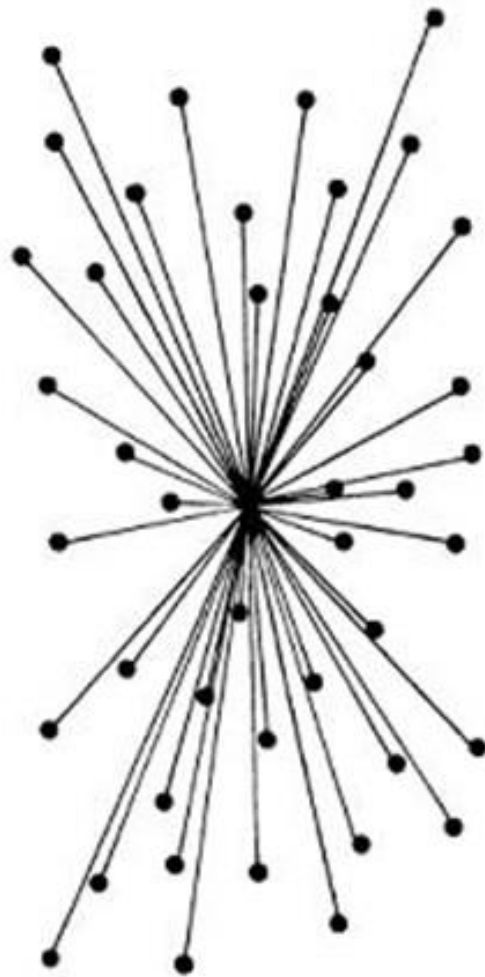
Pode ser difícil distribuir os dados para diversas máquinas (ex: geograficamente)

---

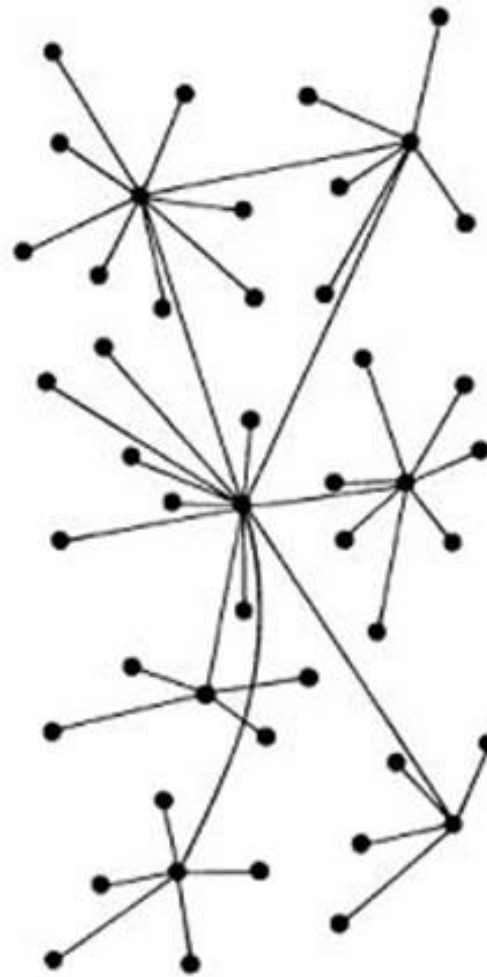


**Muitos  
sistemas  
corporativos  
usam esse  
modelo**

# Tendências...Blockchain não é assim...



CENTRALIZADO

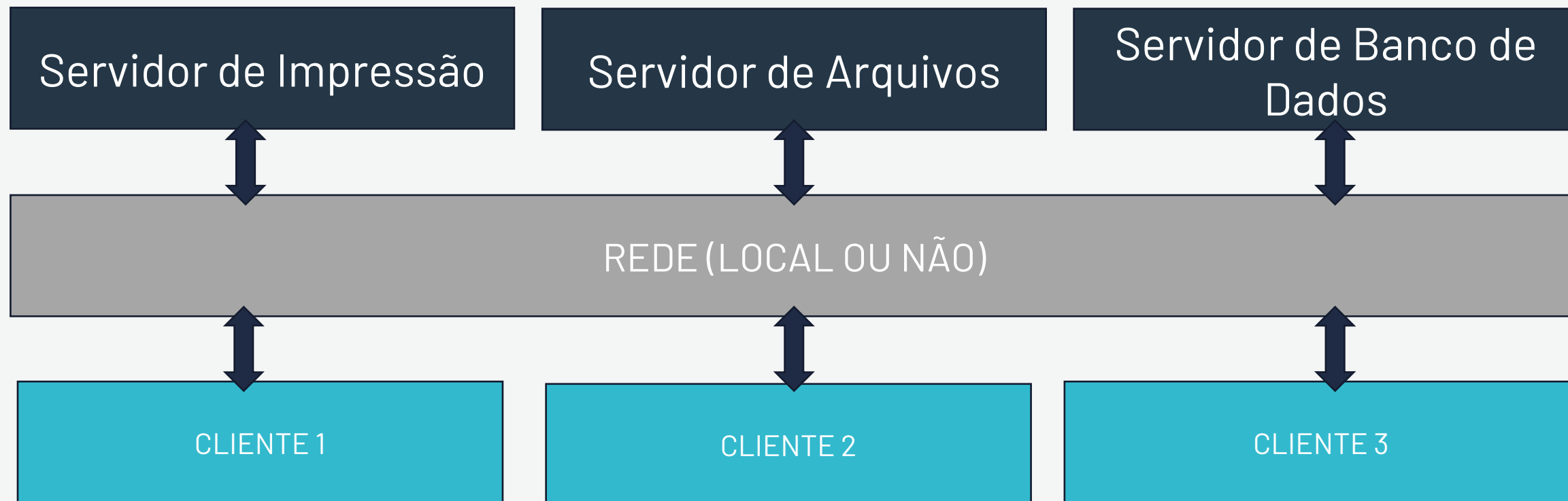


DESCENTRALIZADO



DISTRIBUÍDO

# Modelo Cliente Servidor



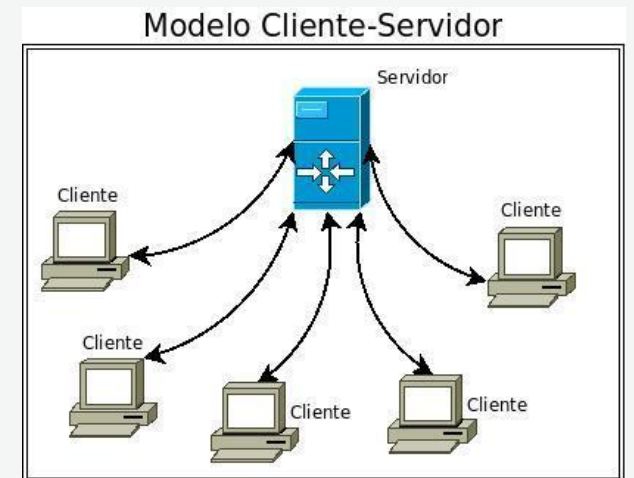
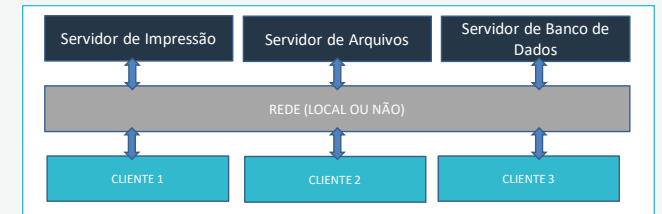
# Modelo Cliente Servidor

Um conjunto de servidores fornecem serviços para outros sistemas

Um conjunto de clientes solicitam serviços para os servidores

Existe uma rede para viabilizar o acesso

O cliente e o servidor podem estar na mesma máquina, mas isso não é prática

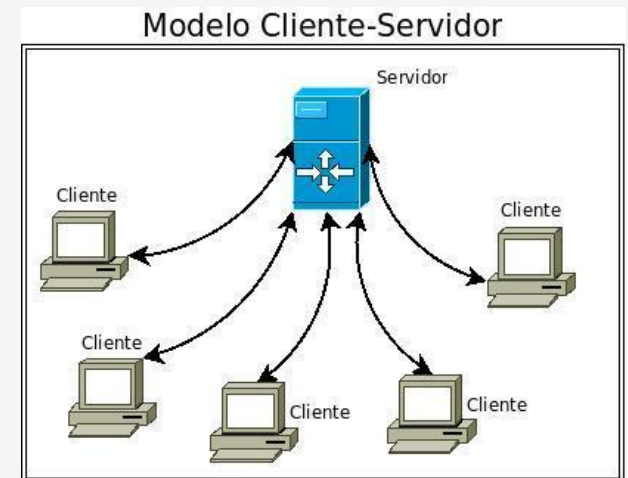
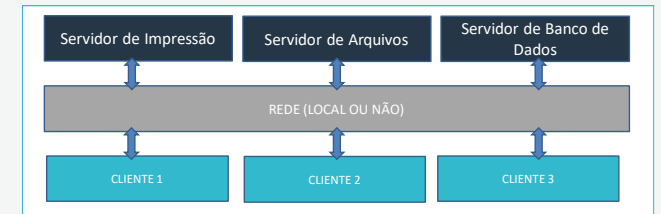


# Benefícios: Modelo Cliente Servidor

Segurança dos dados\*

Acesso centralizado aos dados

Fácil manutenção dos dados\*



# Modelo em Camadas





# Modelo em Camadas

Cada camada pode ser imaginada como um servidor/máquina abstrata

Camada OSI de redes é um exemplo de camada

O modelo de camadas apoia o modelo de componentes e desenvolvimento incremental dos sistemas

Como desvantagem, a estruturação do sistema pode ser mais difícil, assim como o gerenciamento da configuração

O desempenho também deve ser considerado quando utilizado um número grande de camadas (atenção especial para a rede).



# Benefícios: Modelo em Camadas

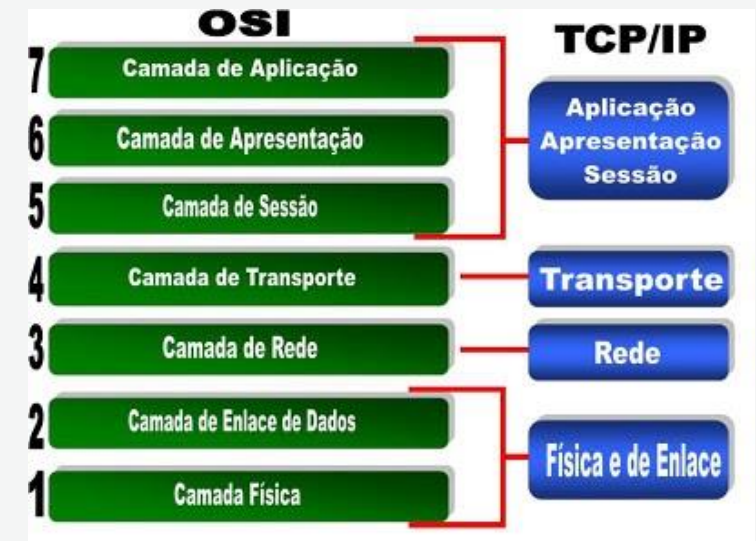
Deploy facilitado

Redução de custos

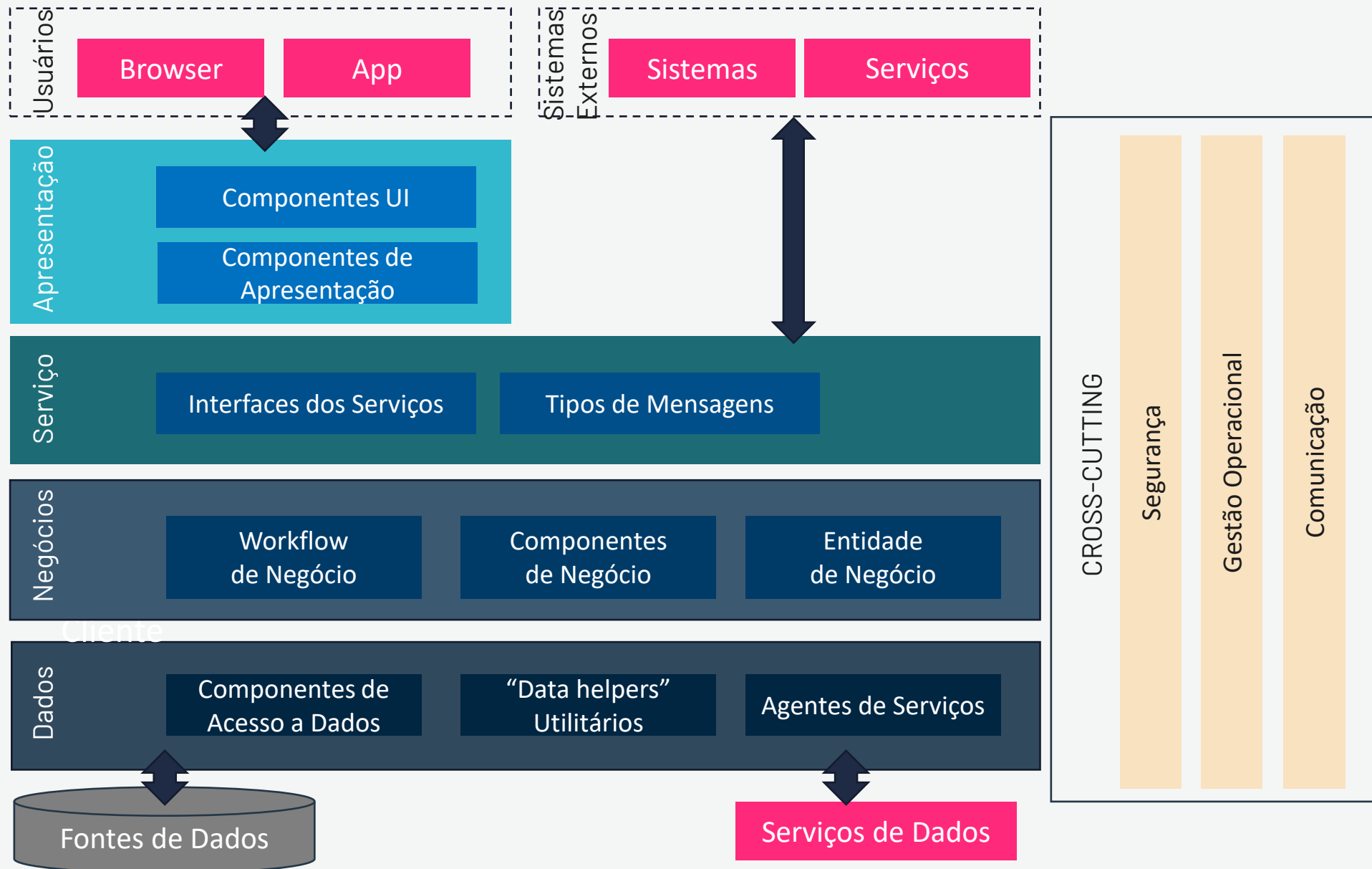
Desenvolvimento facilitado

Reutilização

Mitigação da complexidade técnica (Separação dos conceitos)



# Modelo em Camadas



Serviços são autônomos, cada serviço é mantido, desenvolvido, versionado e implantados separados.

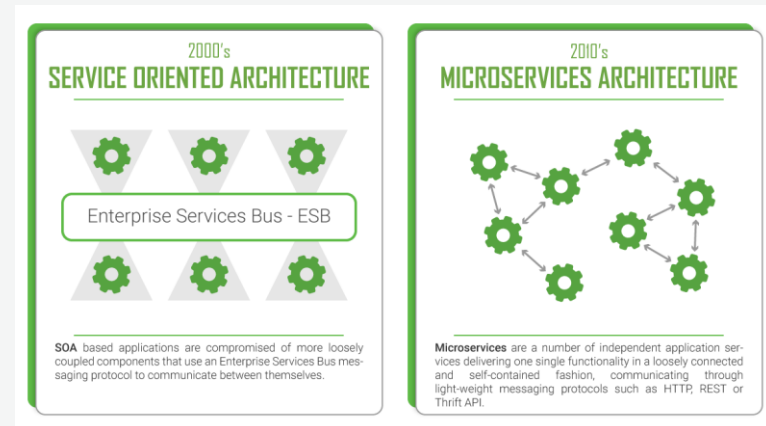
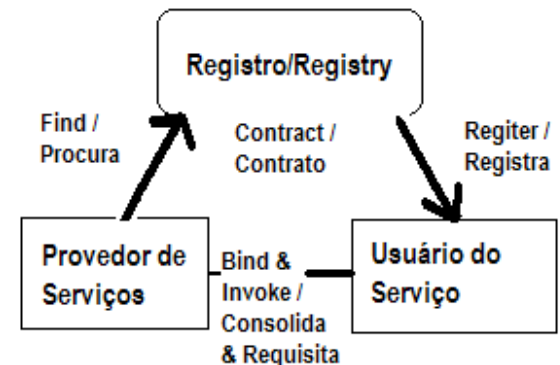
São distribuíveis (distributable), podem estar em redes e localidades remotas.

Tem baixo acoplamento, cada serviço é independente dos outros e podem ser atualizados independentes.

Serviços compartilham esquemas e contratos para se comunicarem (não classes).

A compatibilidade é baseada em políticas como: transporte (rede), protocolo (http) e segurança (https).

## Find-Bind-Execute



# SOA - Benefícios

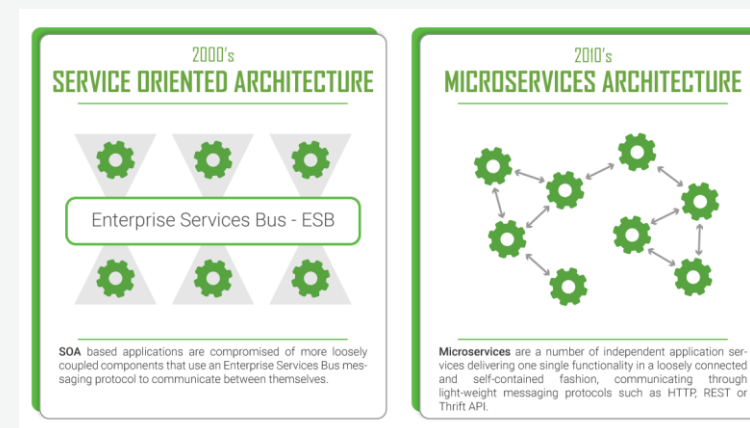
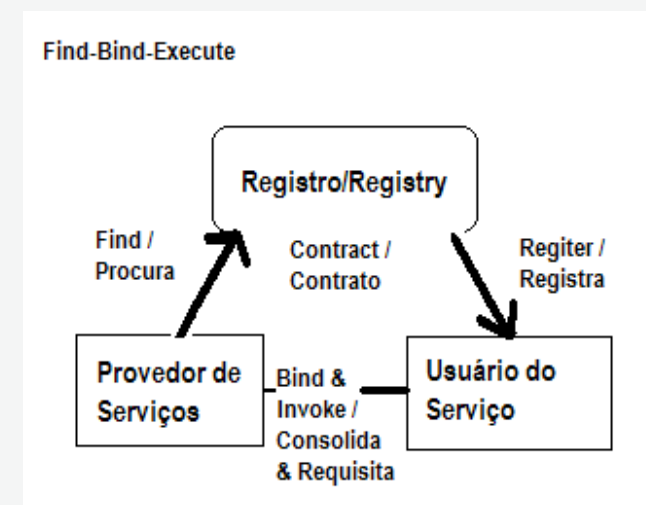
Alinhamento aos domínios de negócios

Abstração, serviços são autônomos e com baixo acoplamento. Pelo contrato é possível conhecer o serviço.

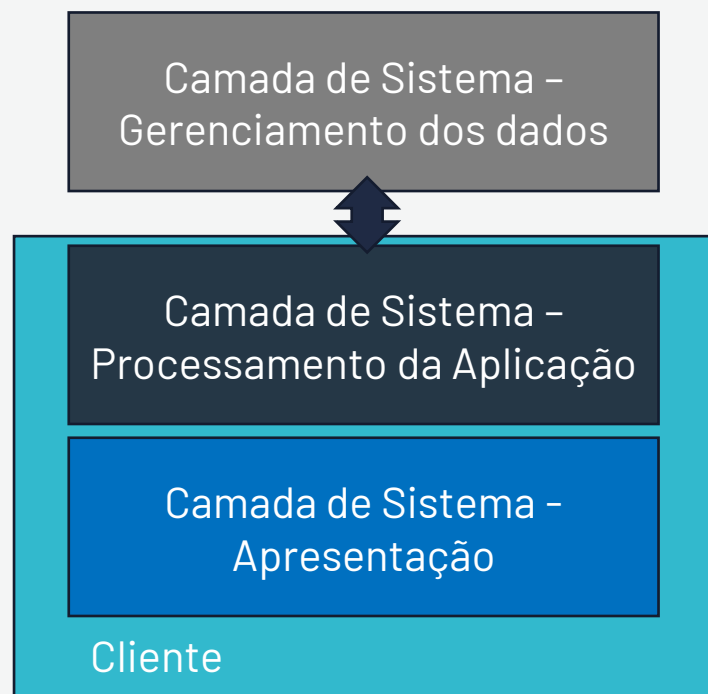
Os serviços podem expor suas descrições e permitir que as outras aplicações automaticamente possam determinar a interface.

Interoperabilidade incrementada porque os protocolos e formatos de dados são padrões de indústria.

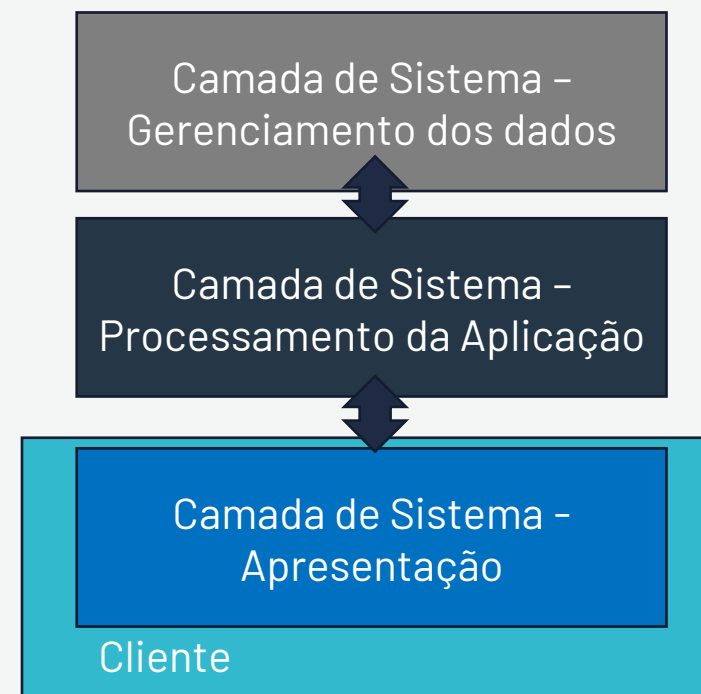
Racionalização, porque os serviços são granulares e a necessidade de duplicação é reduzida.



# Cliente Gordo vs Cliente Magro



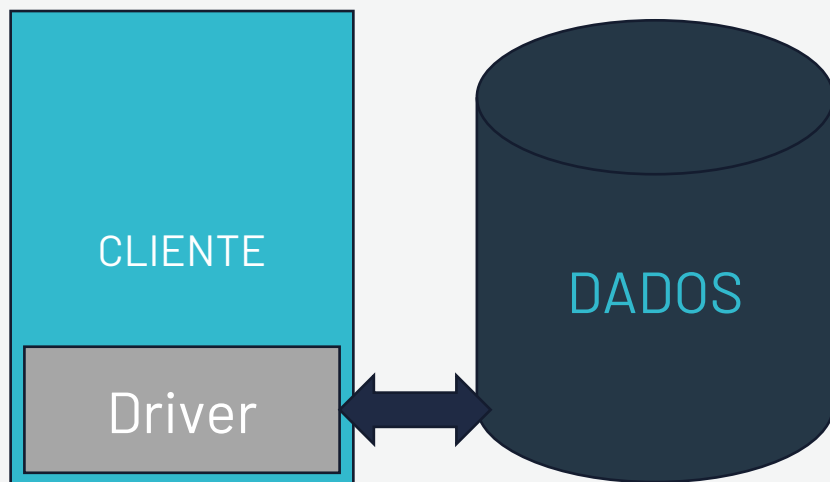
- O Hardware cliente é utilizado na sua plenitude.
- Podem ser necessárias funcionalidades apenas disponíveis em HW cliente, como acesso a periféricos, processamento diferenciado (vídeo).



- O Hardware cliente processa apenas a camada de apresentação (ex: Telas, formulários);

# Modelo de Acesso a Dados [Direto]

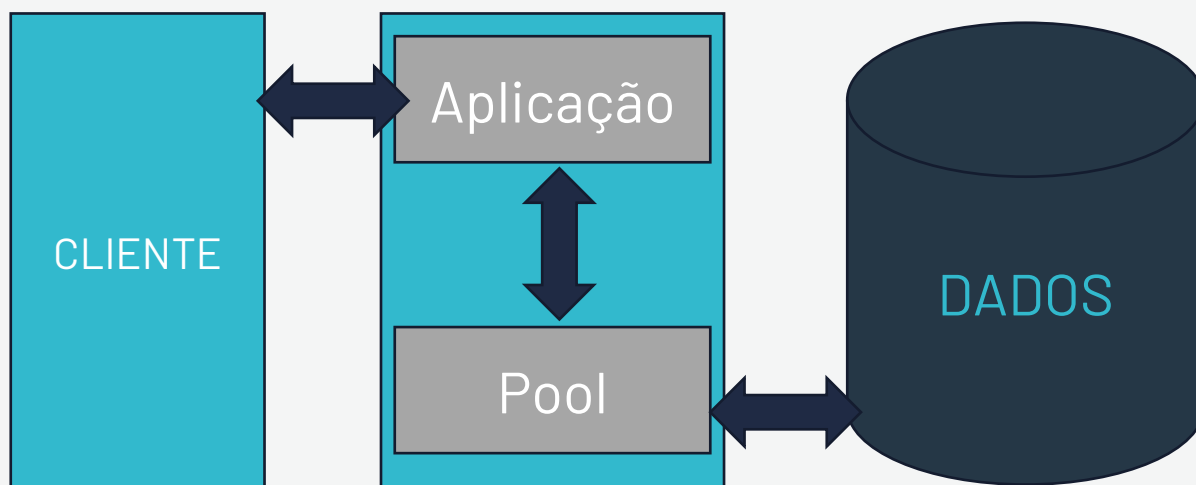
## Acesso direto aos dados



- A aplicação utiliza um driver e faz acesso direto ao Banco de Dados;
- A aplicação faz o controle da abertura e fechamento da conexão, ou seja, abre, utiliza e fecha.
- Aplicações “gordas” em modo cliente servidor normalmente utilizam deste modelo;
- Em aplicações antigas era comum a conexão ficar aberta durante todo o tempo;
- Pode haver muita regra de negócio no Database em formato de Stored Procedures;
- Exemplo: ODBC, ADO;

# Modelos de Acesso a Dados [Pooling]

## Uso de Pool de Conexões

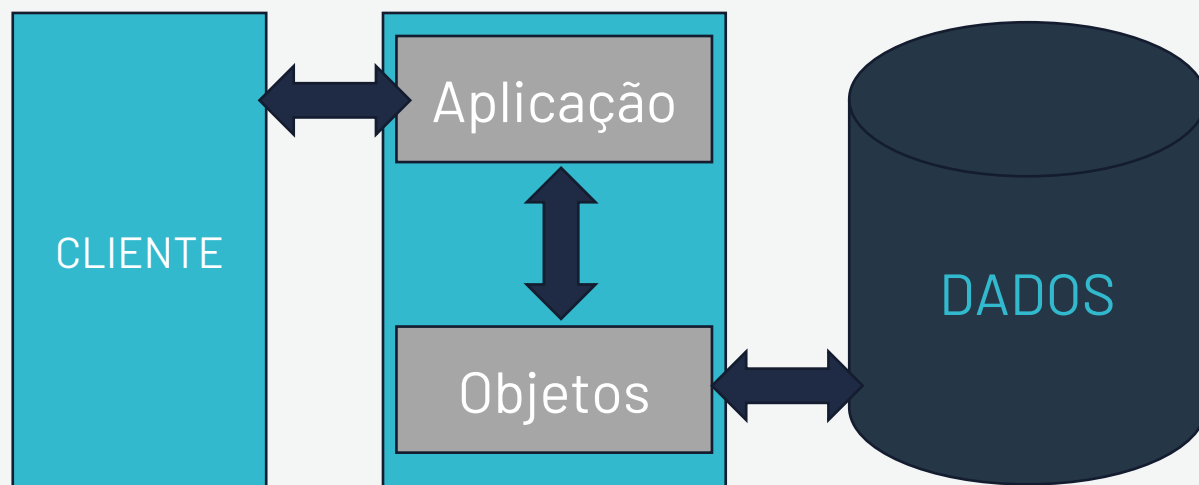


- Existe compartilhamento das conexões e há o dimensionamento do número de conexões que estarão abertas todo o tempo e o número máximo que pode ser atingido;
- A aplicação cliente chama uma aplicação que pode ou não estar na mesma camada que utiliza uma conexão que já está aberta.
- Não há a abertura e fechamento das conexões a cada transação apesar de código existir a chamada do fechamento;
- Há economia de tempo na transação e de recursos do SGDB pois ele gerenciará menos conexões;
- As regra de negócio normalmente estão na aplicação, mas podem existir em Stored Procedures;
- Normalmente o pooling é configurado no servidor de aplicações para uso comum a todas aplicações;
- Exemplos: ADO.NET, JDBC em AppServers;



# Modelos de Acesso a Dados [Objetos]

## Uso de Objetos que representam o Banco de Dados



Cliente
-codigoCliente: int
-nomeCliente: string
+getNome(codigoCliente)

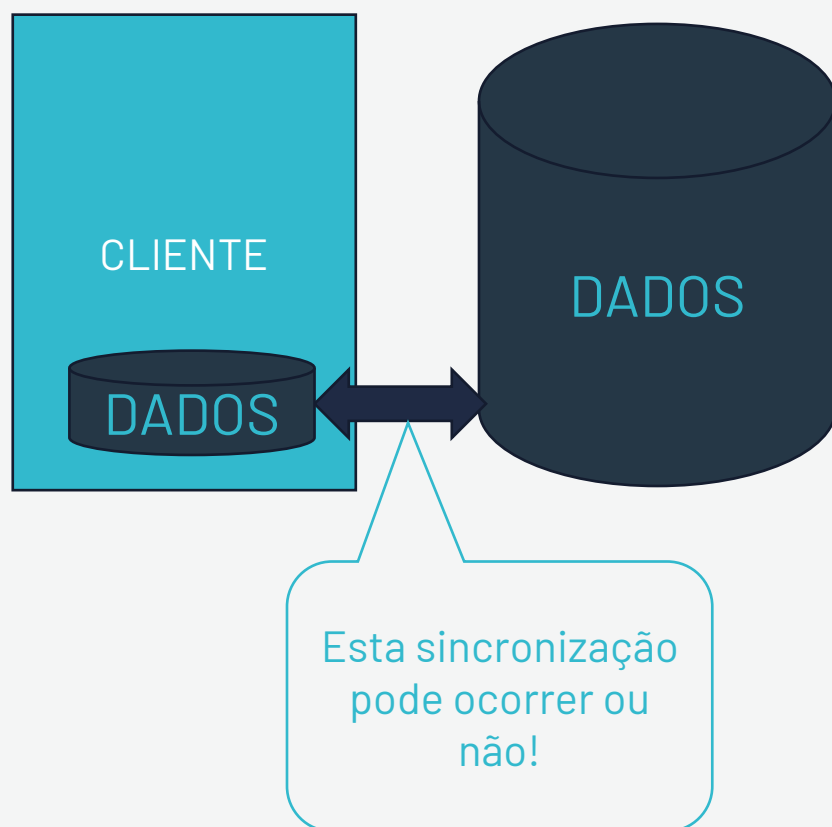


Clientes	
Id	Integer
Nome	Varchar(100)

- O Banco de Dados é representado para a aplicação no formato de objetos;
- A aplicação não enxerga a estrutura de dados, ou seja, as tabelas e registros são apresentadas em formato de classes e objetos;
- O código SQL é montado automaticamente pelas bibliotecas; (é comum que problemas de performance sejam mais difíceis de serem localizados)
- Normalmente existe pooling configurado;
- A aplicação (middleware) pode fazer cache de dados em objetos o que aumenta a performance;
- Exemplos: JPA/Hibernate, Entity Framework;

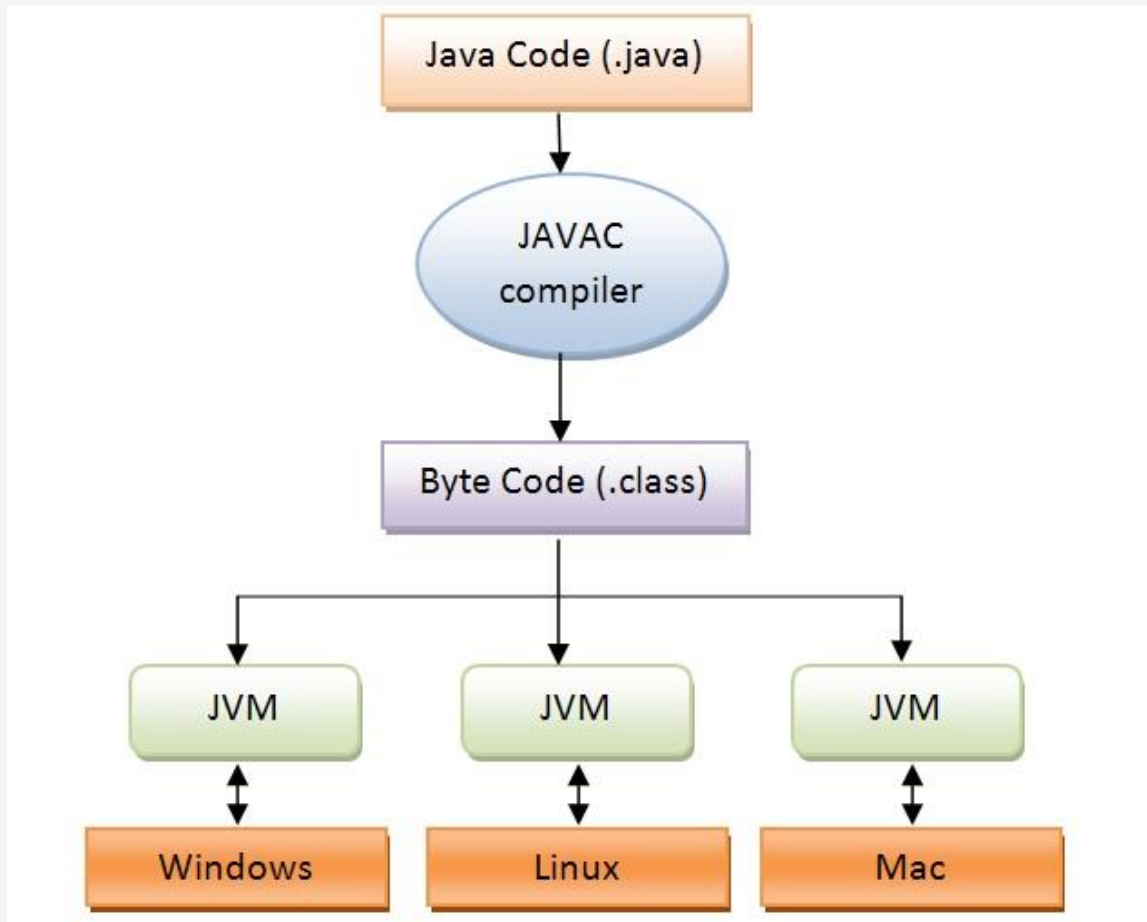
# Modelos de Acesso a Dados [Embutidos]

## Banco de dados embutidos



- A aplicação se conecta em um banco de dados local;
- Utilizado para armazenar pequenas quantidades de informações e normalmente de forma temporária;
- Muito utilizado em ambientes assíncronos ou com conexão instável;
- É comum existir posterior sincronização com outras aplicações ou bases centralizadas;
- Aplicações Mobile podem se utilizar deste recurso;
- Exemplo: SQLite; SQL Server Compact; H2;

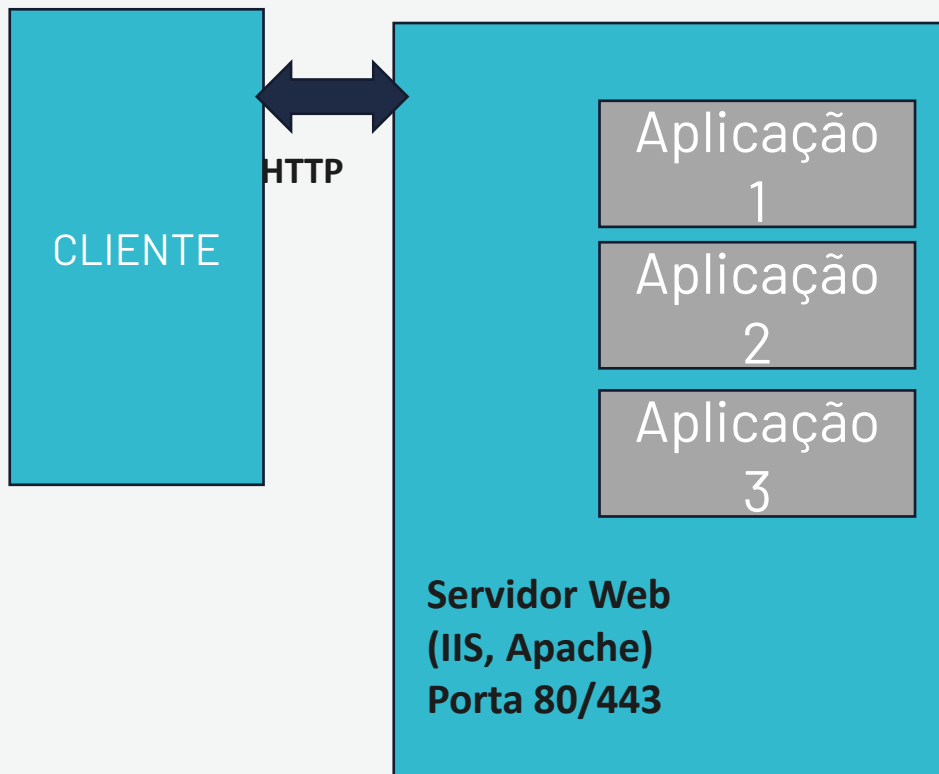
# Máquinas Virtuais



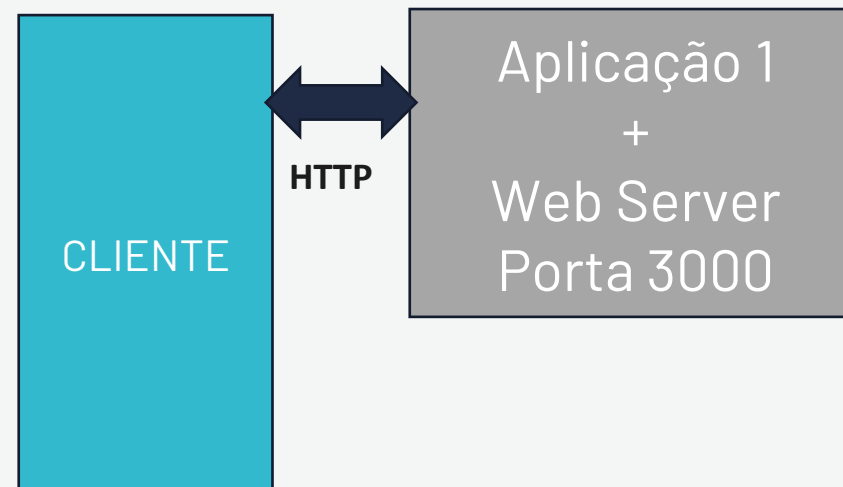
- **JVM** – Java Virtual Machine
- O código é escrito uma vez e compilado em uma linguagem intermediária;
- A Máquina virtual instalada no sistema operacional compila ou interpreta o *byte code* transformando em linguagem de máquina ;
- A aplicação escrita pode ser portada para diversas plataformas diferentes;
- O conceito de máquina virtual é amplamente utilizado. Ex: Java, Go, .NET

# Projeto com WebServer embarcado

Desenho tradicional. Exemplo

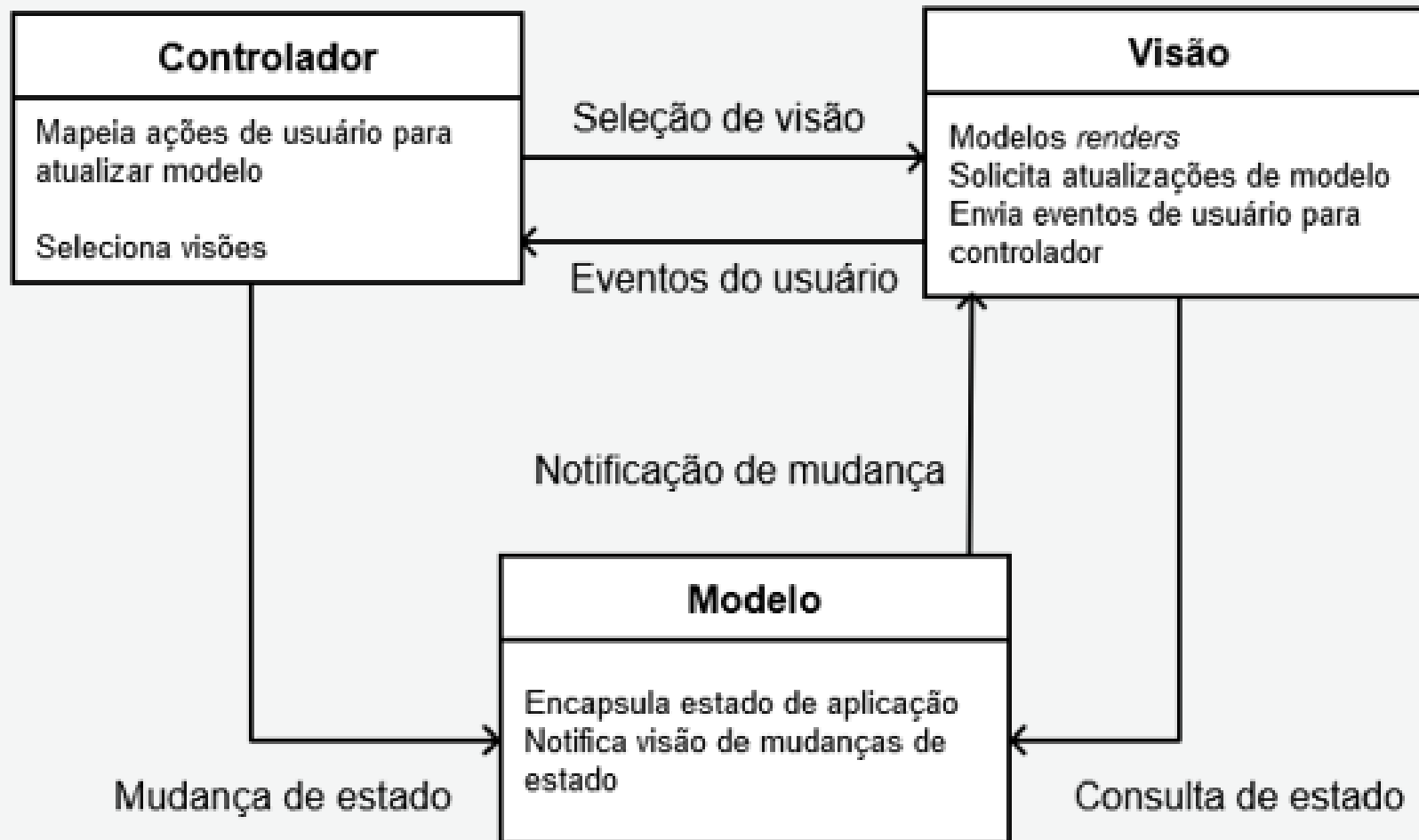


Você realiza deploy do seu projeto que é a aplicação.



Você realiza deploy do seu projeto que é a aplicação junto com um WebServer.  
Ex: Rails, Node.js, SpringBoot

# O modelo MVC [Model, View, Controller]



# MVC [Model, View, Controller]

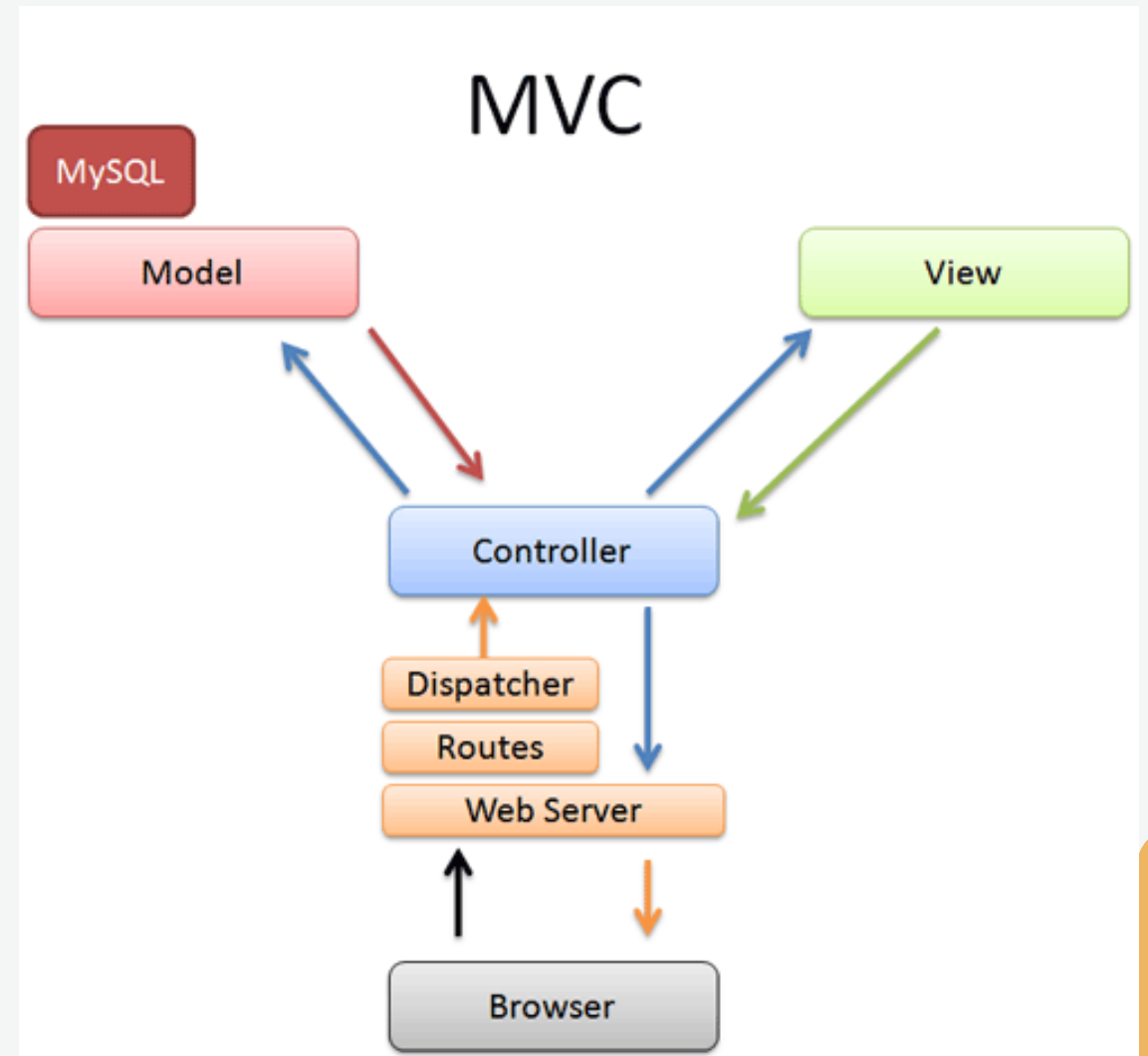
Separação da camada de apresentação da camada de controle

O controle gerencia as interações o usuário (click, tecla pressionada, ou seja, os eventos)

A camada de Modelo gerencia o acesso aos dados e como este acesso deve ser realizado

Utilizado quando há muitas formas de apresentar e interagir com os dados

Pode envolver código adicional e complexidade.



# Dicas

1. Mantenha os padrões de projeto consistentes em cada camada. Dentro de uma camada lógica, quando possível, o design dos componentes deve ser **consistente para uma operação específica**.
2. **Não duplique a funcionalidade em um aplicativo.** Deve haver apenas um componente fornecendo uma funcionalidade específica - essa funcionalidade não deve ser duplicada em nenhum outro componente.
3. **Prefira composição à herança.** Sempre que possível, use composição sobre herança ao reutilizar a funcionalidade, pois a herança aumenta a dependência entre as classes pai e filho, limitando, assim, a reutilização de classes filhas.
4. **Estabelecer um estilo de codificação e convenção de nomenclatura para o desenvolvimento.** Verifique se a organização estabeleceu padrões de estilo de codificação e nomenclatura. Se não, você deve estabelecer padrões comuns. Isso fornece um modelo consistente que torna mais fácil para os membros da equipe revisarem o código que não escreveram, o que leva a uma melhor manutenção.

## Mais Dicas

5. Mantenha a qualidade do sistema usando técnicas automatizadas de controle de qualidade durante o desenvolvimento. **Use testes unitários** e outras técnicas automatizadas de Análise de Qualidade, como análise de dependência e análise de código estático, durante o desenvolvimento. Defina métricas comportamentais e de desempenho claras para componentes e subsistemas e use ferramentas de controle de qualidade automatizadas durante o processo de criação para garantir que as decisões locais de projeto ou implementação não afetem negativamente a qualidade geral do sistema.
6. **Considere o funcionamento de sua aplicação.** Determine quais métricas e dados operacionais são exigidos pela infraestrutura de TI para garantir a implantação e operação eficientes do seu aplicativo. Projetar os componentes e subsistemas do seu aplicativo com um entendimento claro de seus requisitos operacionais individuais facilitará significativamente a implantação e a operação geral. Use ferramentas de controle de qualidade automatizadas durante o desenvolvimento para garantir que os dados operacionais corretos sejam fornecidos pelos componentes e subsistemas do seu aplicativo.



# Diretrizes de Qualidade [Sommerville]

1. O projeto deve ser exibido em um desenho de arquitetura (compreensível e que possa ser evoluído);
2. O projeto deve ser modular
3. Deve ter representações de: dados, arquitetura, interfaces e componentes
4. Estrutura de classes adequadas e baseadas em padrões reconhecíveis
5. Componentes que tenham características funcionais independentes
6. Deve possuir interfaces que simplifiquem a conexão entre os componentes e o ambiente externo
7. O projeto deve ser obtido usando um método repetível, dirigido por informações adquiridas durante a análise de requisitos de software (Aula do Prof. Alex)
8. Um projeto deve ser representado usando uma notação que efetivamente comunique seu significado.

# Reflexão: Pêndulo Infinito

**BROWSER**

**Terminal de  
Mainframe**

**Tudo no  
DataCenter**



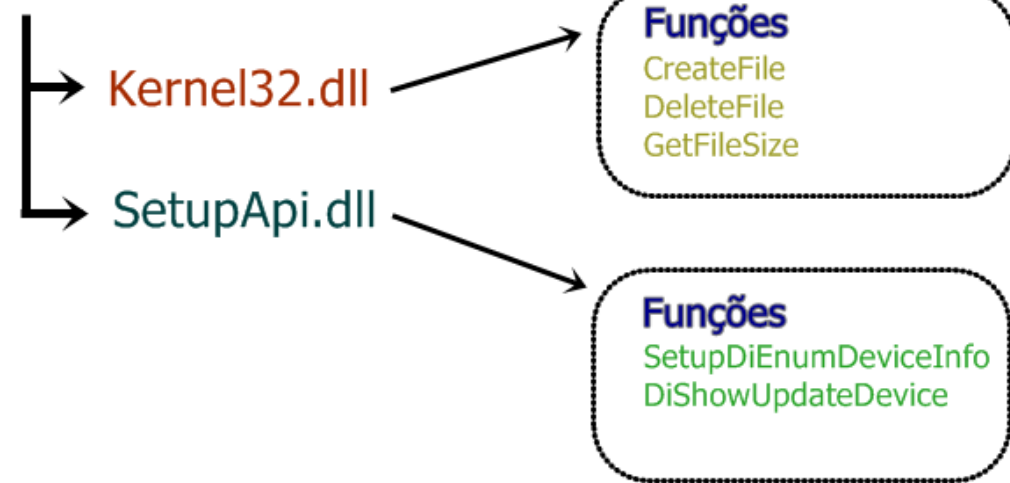
**APP  
SMARTPHONE**

***Tudo na  
Nuvem***

As **ARQUITETURAS** estão **SEMPRE INDO E VOLTANDO...**  
**AS DECISÕES RARAMENTE SÃO DO TIME DE TI...**

# API, Biblioteca e Framework

Win32 API



**API**, da sigla **Application Programming Interface** ou Interface de Programação de Aplicações, é um produto de software criado para oferecer uma interface (caminho) com regras bem definidas para integração entre sistemas, a fim de obter informações e, assim, trabalhar com elas. **É uma coleção de métodos disponibilizados para interagir com um serviço, mas sem acesso direto ao software.**

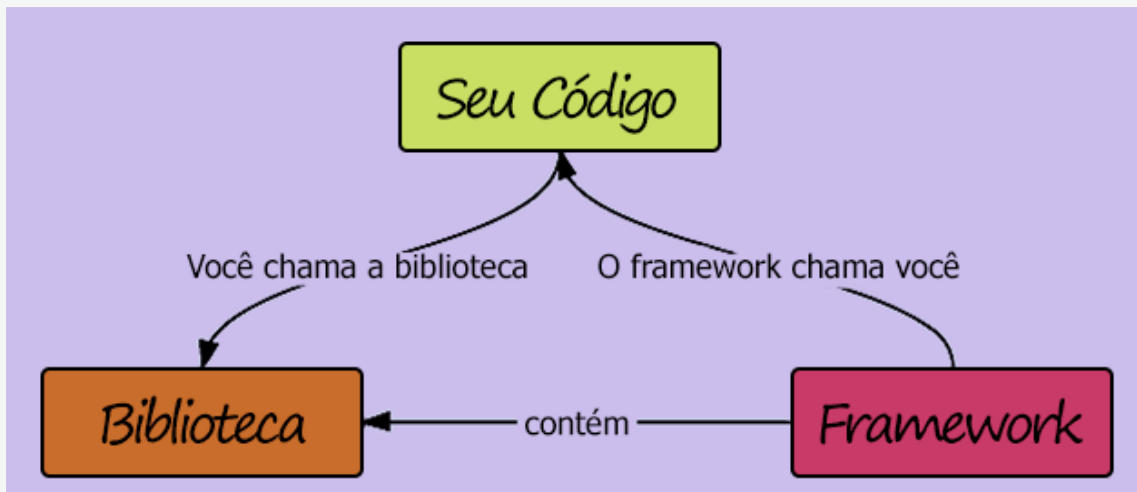
- Exemplo: Facebook, Windows

**Biblioteca** é um conjunto de classes (subprogramas ou funções), que podem ser usadas para a construção de um software. Ex: **ValidaCPF**, **ConsultaBancoOracle**.

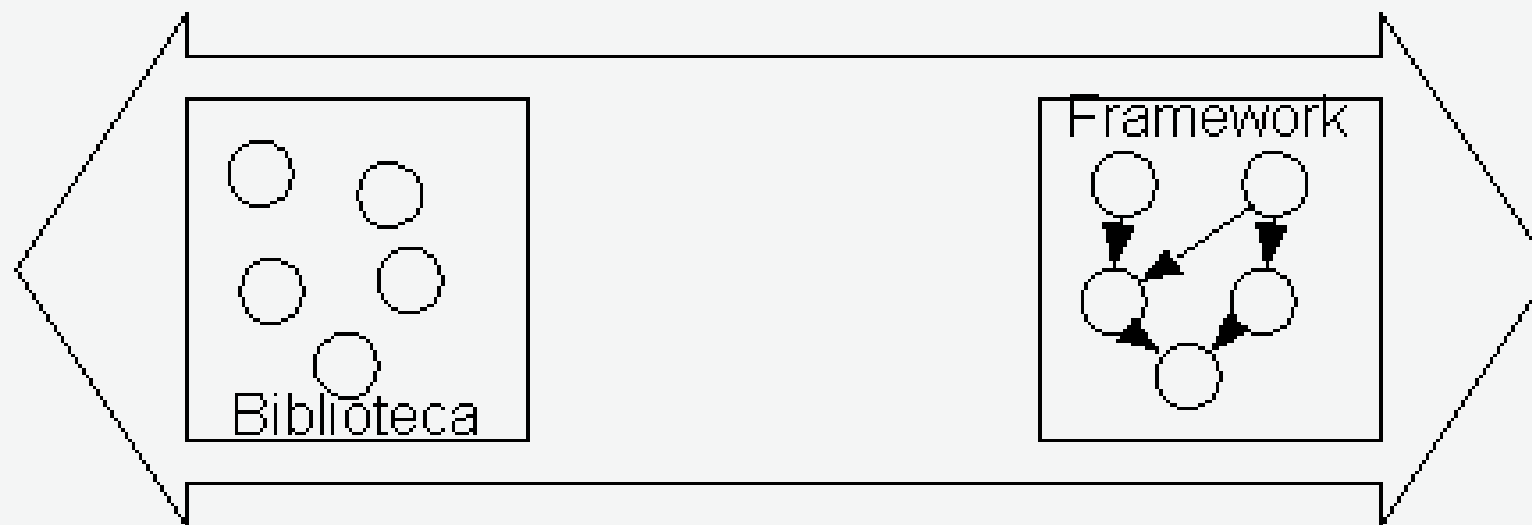
A biblioteca normalmente pode ser uma implementação real das regras de uma API

## Framework

Um framework captura a funcionalidade comum a várias aplicações. Um framework pode ser construído utilizando-se diversas bibliotecas e integrado a APIs. O framework é um conjunto de classes, bibliotecas e códigos que colaboram entre si.



# API, Biblioteca e Framework



- Classes instanciadas pelo cliente
- Cliente chama funções
- Não tem fluxo de controle predefinido
- Não tem interação predefinida
- Não tem comportamento default

- Customização com subclasse ou composição
- Chama funções da "aplicação"
- Controla o fluxo de execução
- Define interação entre objetos
- Provê comportamento default

<http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/frame/oque.htm>

Framework é uma estrutura genérica que pode ser ampliada para criar um subsistema ou aplicação mais específica. (Sommerville). O framework pode ser ampliado e pode ser necessária a adição de novas classes que vão herdar funcionalidades das classes abstratas.

# Frameworks

**Frameworks são ferramentas, não modos de vida. “Não case com o Framework!”**

**A arquitetura deve falar sobre o sistema e não sobre os frameworks que você utilizou no sistema.**

**Evite que os frameworks entrem no seu código central do sistema.**  
*(Polêmico, mas muito seguido por Empresas feitas para durar)*

# RESUMÃO

## BIBLIOTECA

Uma biblioteca se refere a uma coleção de pacotes que fornecem funções. Seu objetivo é oferecer um conjunto de funcionalidades prontas para uso sem se preocupar com outros pacotes.

### Exemplos:

- **Moment.js**: Biblioteca para converter, validar, manipular e exibir datas e horários.
- **Chart.js**: Biblioteca para criação de gráficos.
- **mo.js**: Biblioteca para criar animações com SVG.
- **React**: Biblioteca para criar interfaces de usuário.

## FRAMEWORK

Conjunto de bibliotecas, classes e códigos. Um framework não oferece apenas funcionalidades, mas também uma arquitetura para o trabalho de desenvolvimento. Você **não** cria ou inclui uma estrutura, em um framework, você integra seu código a ele.

### Exemplos:

- **Angular**: Framework para criação de aplicações web
- **Vue.js**: Framework para criação de aplicações web
- **ionic**: Framework para criar arquivos mobile
- **Express**: Framework para criar aplicações com Node.js

# Toolkit e SDK

**Toolkits** funcionam de forma mais livre, não são frameworks, similares a bibliotecas funcionando em conjunto, ou seja, você usa o que precisa.

Ex: Google Web Toolkit. Conversão de classes java para javascript.

**SDKs. Software Development Kits** podem ser toolkits ou frameworks. Contém ferramentas adicionais além das bibliotecas e documentações. Podem vir com exemplos de códigos que ajudam a usar a biblioteca adequadamente.

Ex: Java SDK (compilador, bibliotecas, utilitários)

**Conteúdo necessário para o  
Entregável e para a  
atividade de hoje.**



# Conceitos que serão utilizados

Vamos pensar em containers (não é Docker), mas pensar que o container é conjunto que precisa estar funcionando ou rodando para um software funcionar.

Exemplos de Containers (Representados por grandes quadrados):

**Server-side web application:** Aplicação backend. Ex: Spring MVC, NodeJs, Asp.NET MVC, etc.

**Client-side web application:** A aplicação Javascript que roda no Web Browser. Ex: Angular, JQuery, React.

**Client-side desktop application:** A aplicação que roda local. Ex: Java JAR, .NET Windows, C++.

**Mobile app:** Ex: App IOS, App Android, App React Native.

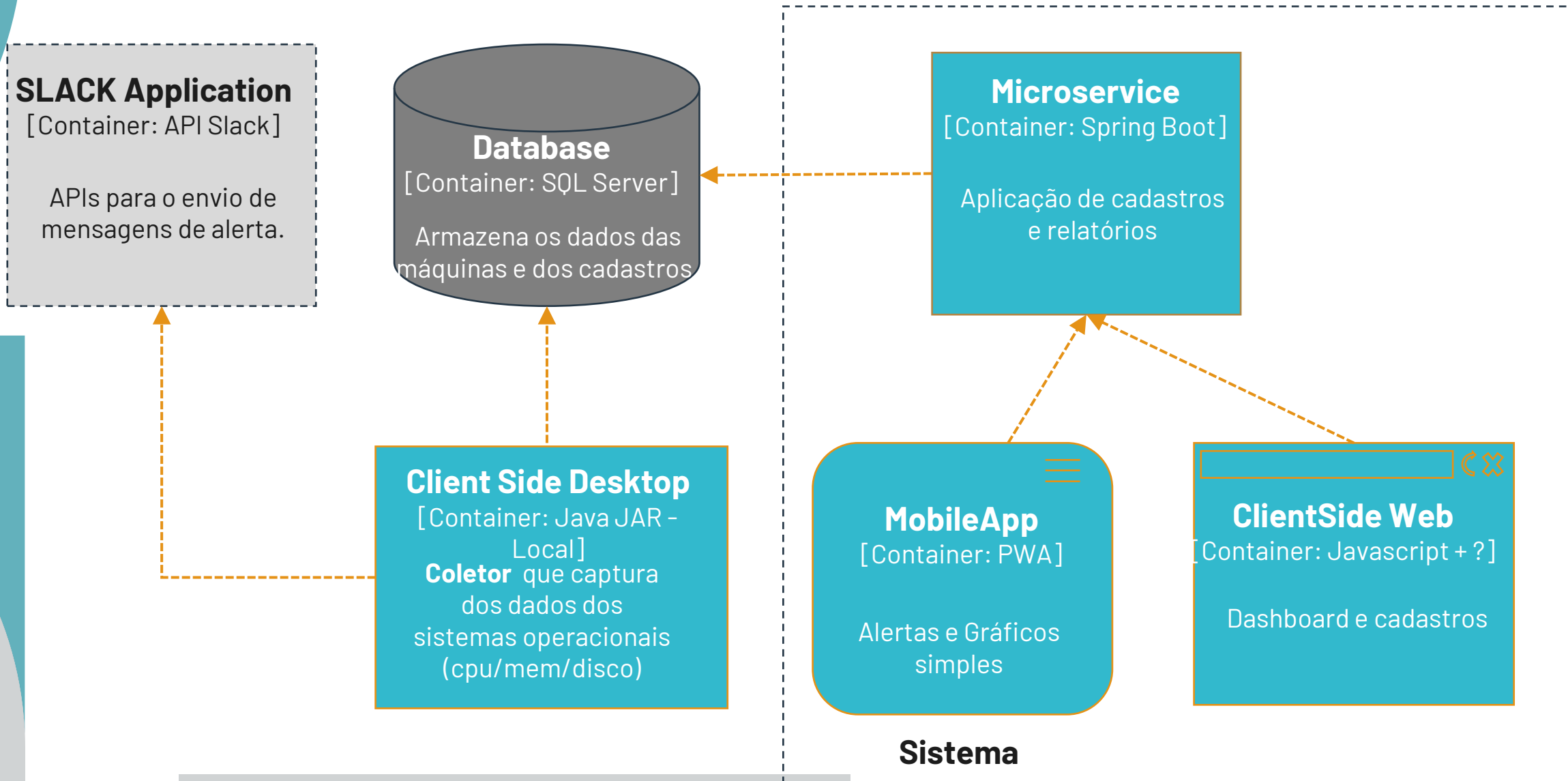
**Server-side console application:** Ex: "public static void main" application, batch, script.

**Microservice:** Ex: Spring Boot.

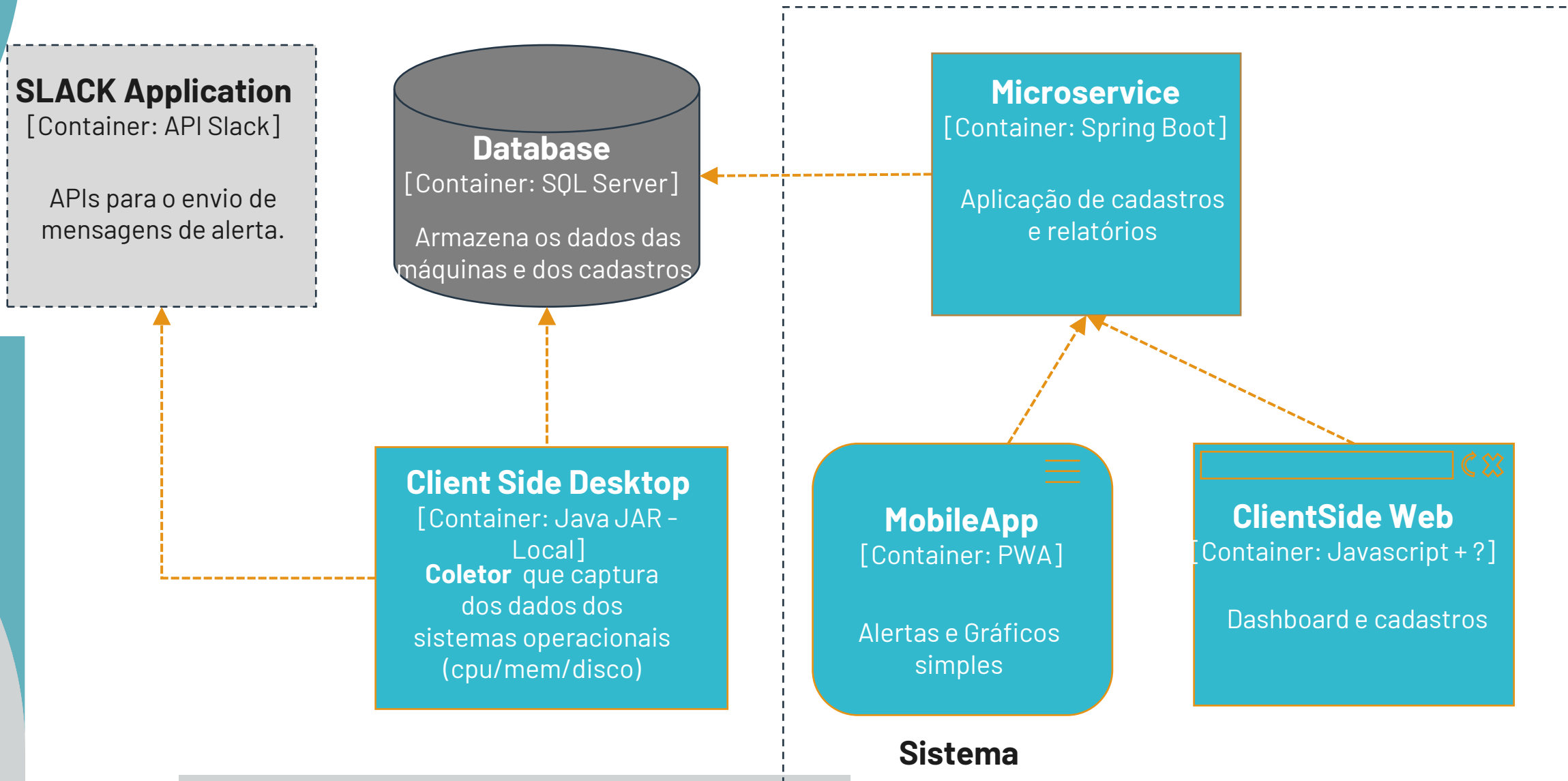
**Serverless function:** Uma função que independe se servidor. Ex: Amazon Lambda, Azure Function.

**Database:** Um banco de dados relacional ou de objetos. Ex: MySQL, SQL Server, Oracle Database, MongoDB.C

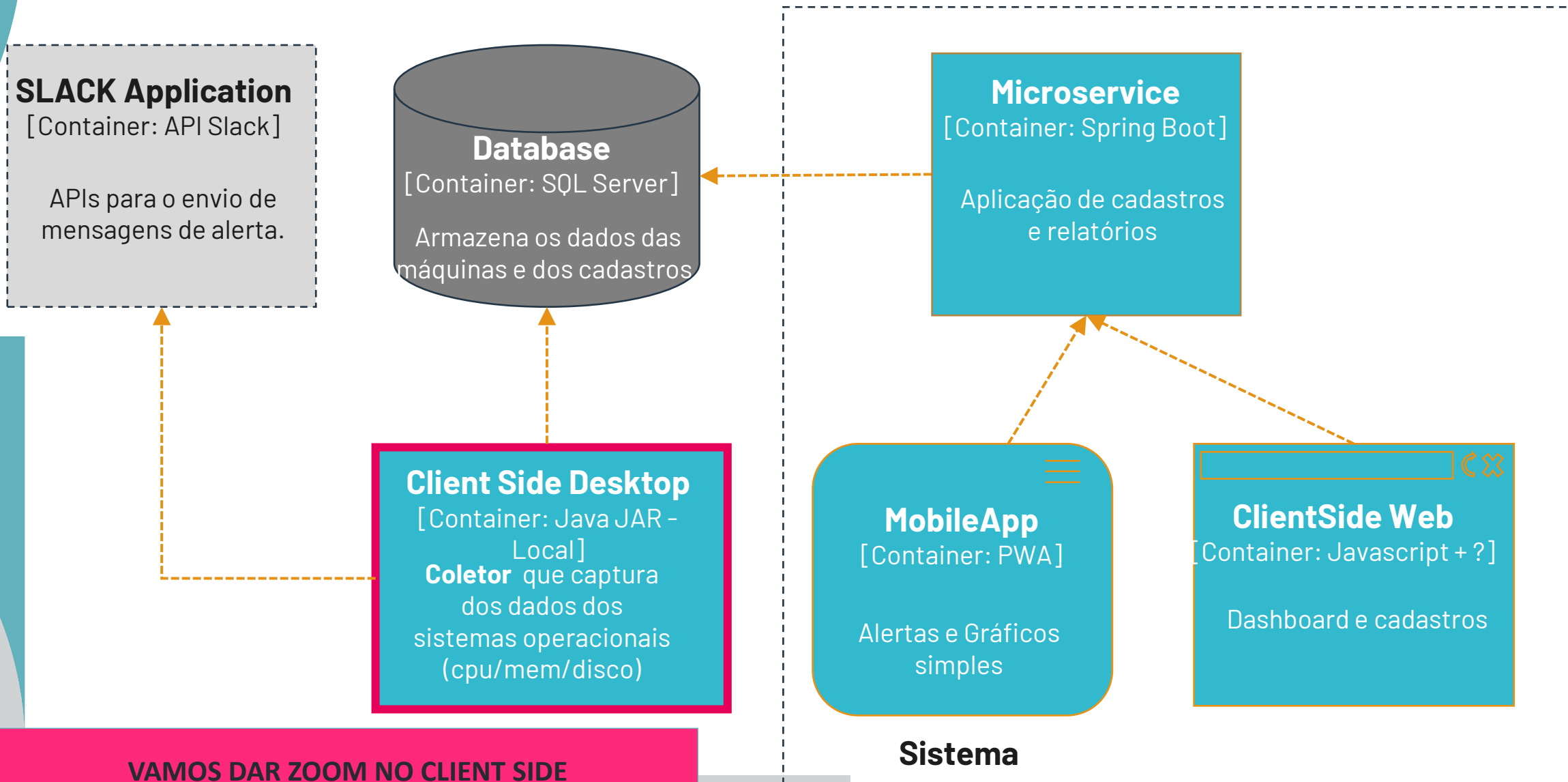
# Diagrama – Visão – Containers



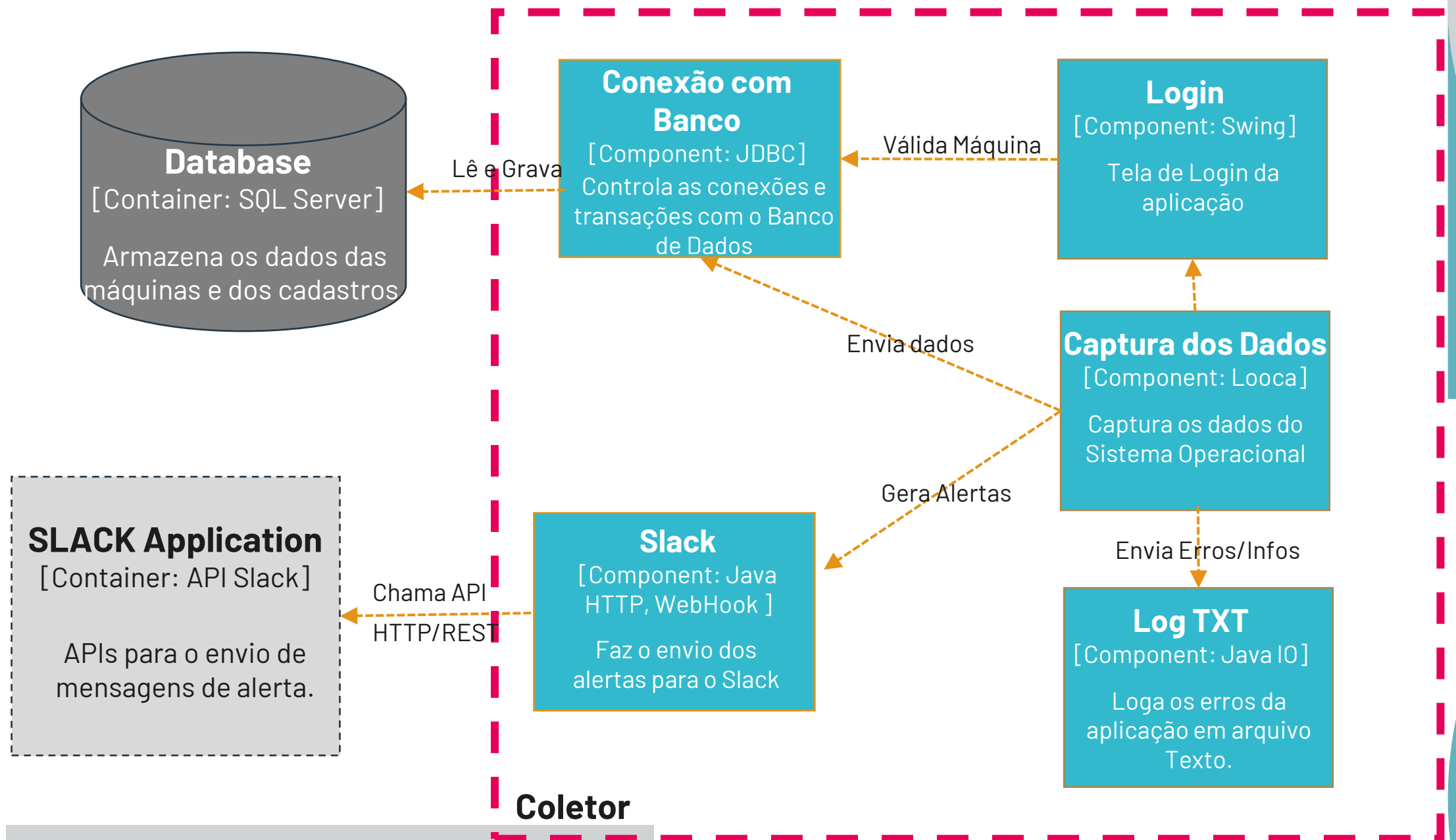
# Diagrama – Visão – Containers



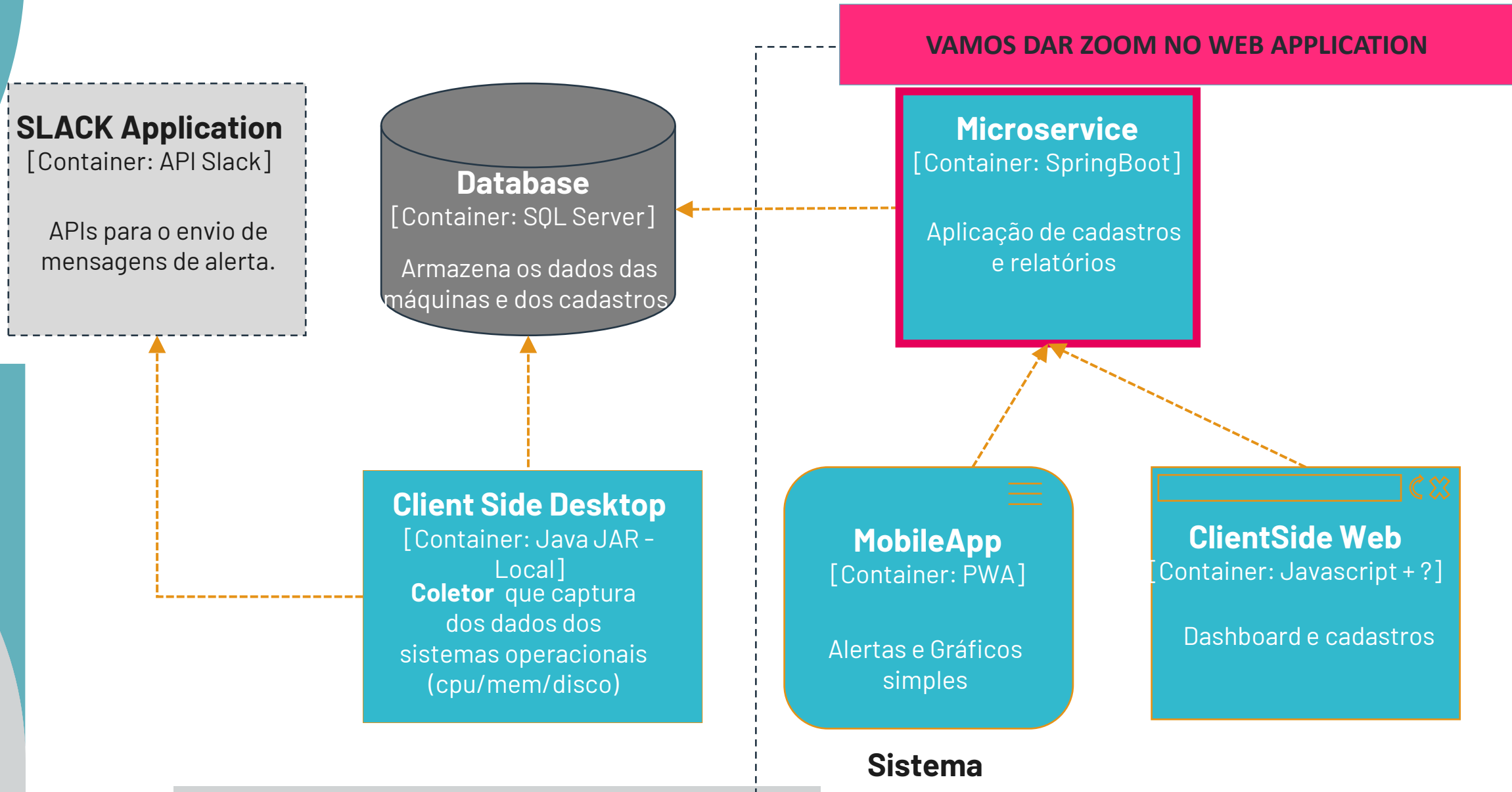
# Diagrama – Visão – Containers



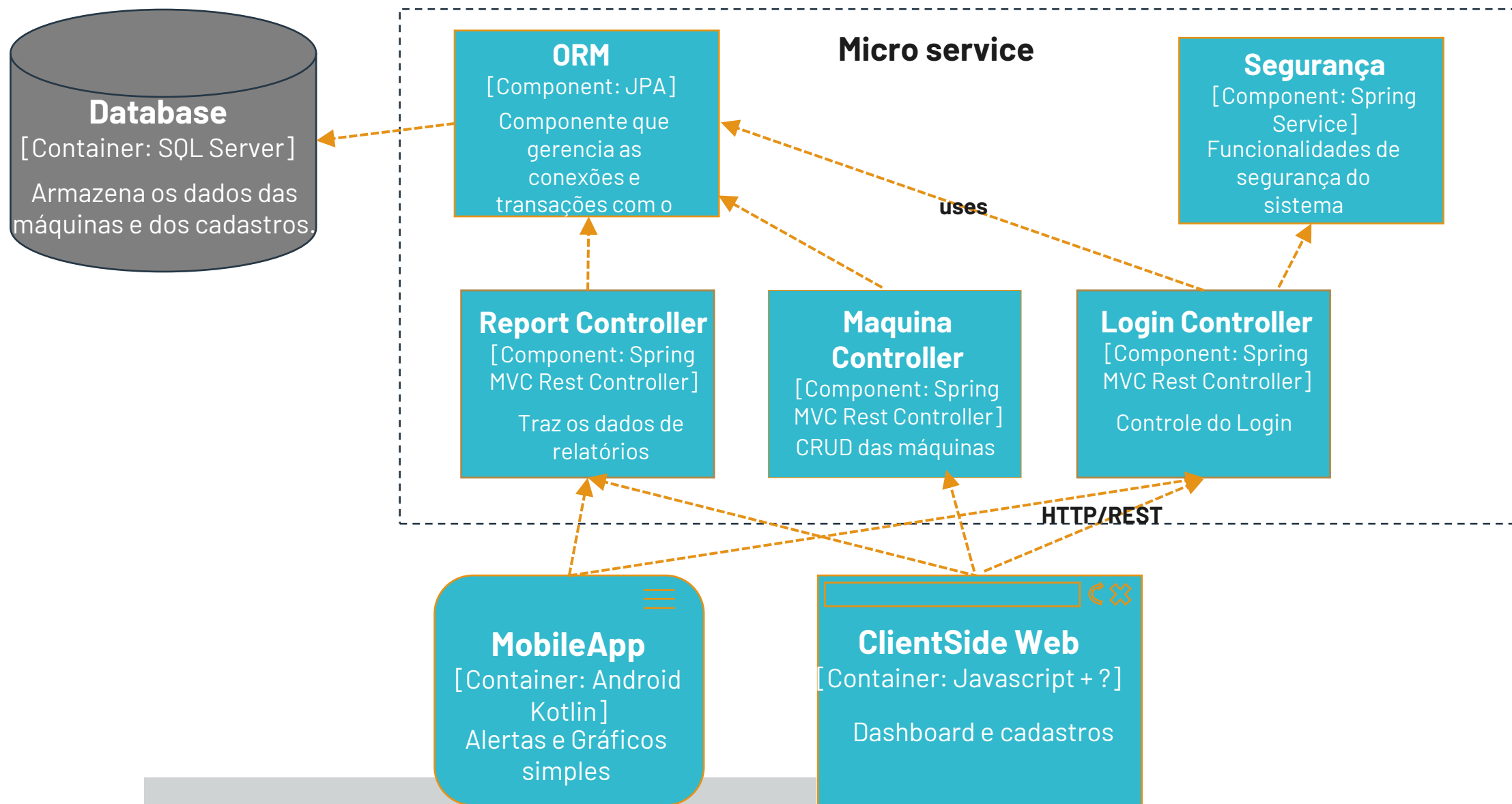
# Diagrama – Visão – Componentes – Coletor



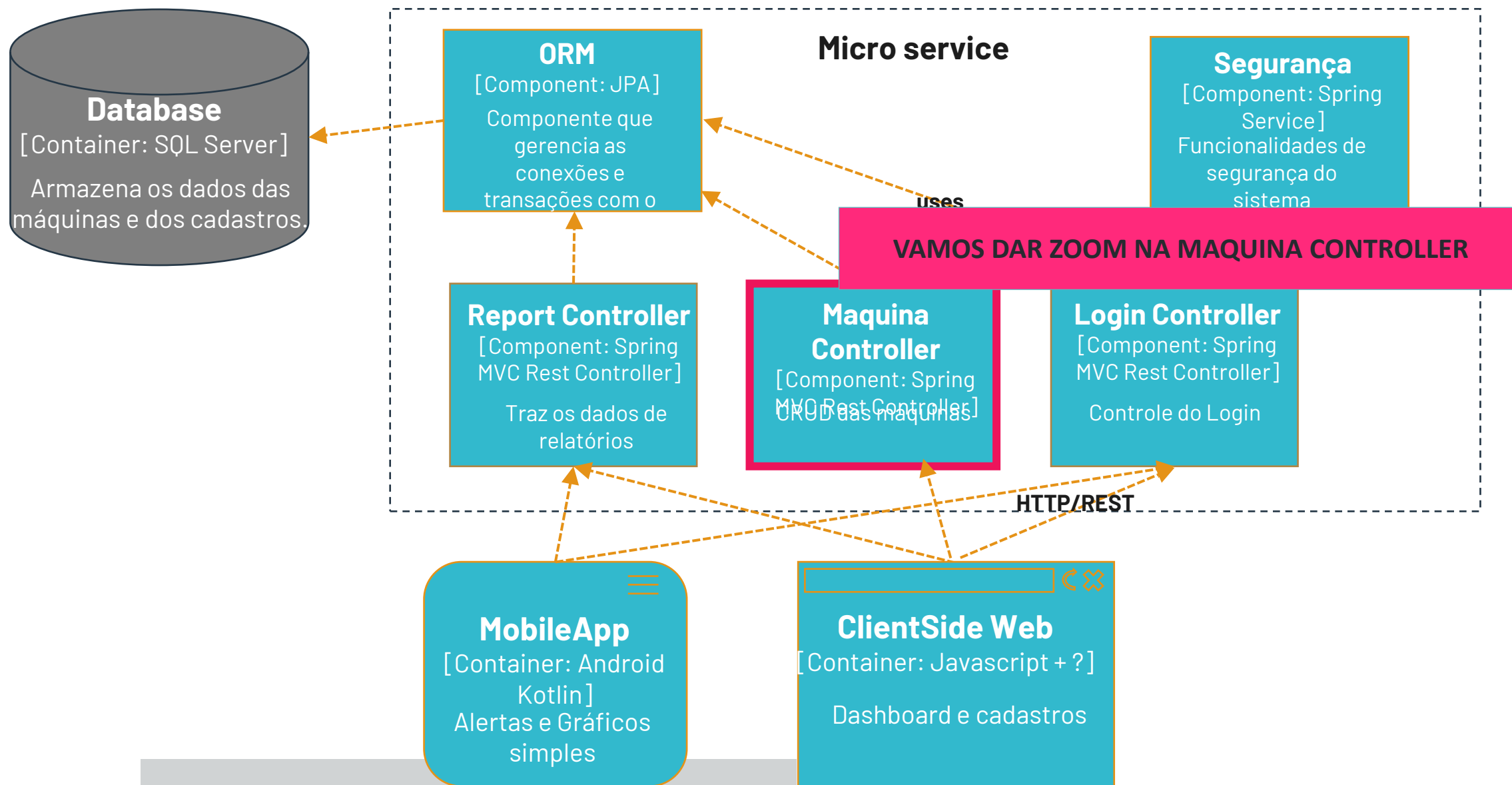
# Desenho da Solução – Containers



# Diagrama – Visão – Componentes – Web Application

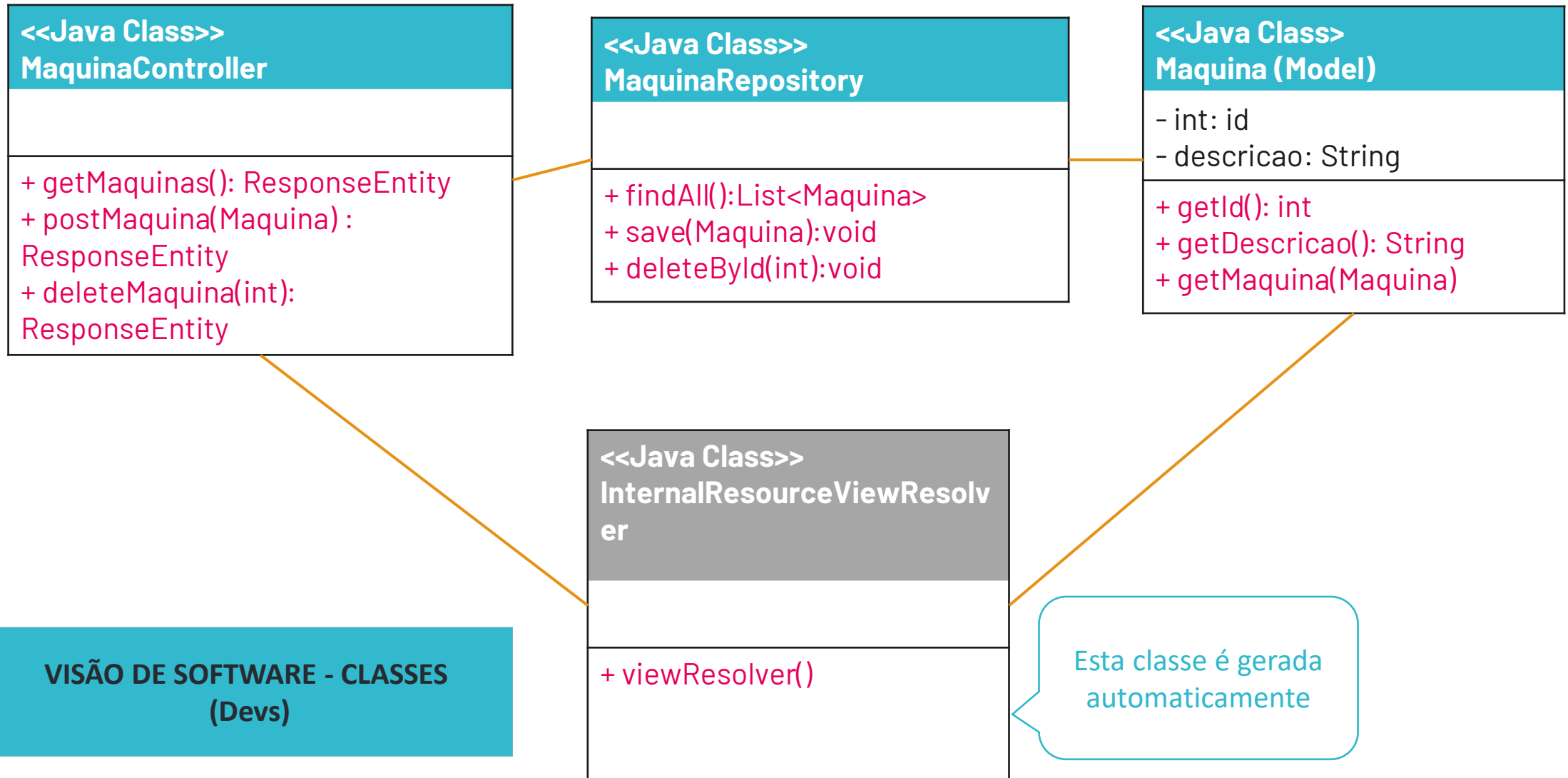


# Diagrama – Visão – Componentes – Web Application





# Diagrama de Classes – Maquina Controller



# Passo a passo – Desenho de Arquitetura (Entregável de PI)

1. Identificar os Objetivos da Arquitetura
2. Cenários Chave
  1. O que é crítico para o negócio?
  2. O que gera alto impacto?
3. Fazer a visão global (overview) da Aplicação
  1. Determinar o tipo da sua aplicação (WEB, Mobile, etc)
  2. Identificar as restrições no desenvolvimento (Rede, Segurança, Sistema Operacional)
  3. Identificar estilos importantes de arquitetura (Camadas, SOA) – Vamos ver mais a frente.
  4. Determinar as tecnologias relevantes (Spring, Node.JS)
4. Desenhar no quadro ou folha de papel
5. Identificar os assuntos chaves (Key Issues: Qualidade, Deploy, Execução, Usabilidade)
6. Cuidar dos itens Transversais (Caching, Comunicação, Autenticação, etc).

# Atividade em Sala

The background is a dark, monochromatic abstract composition. It features a series of thin, white, wavy lines that flow across the frame, creating a sense of movement and depth. Interspersed among these lines are numerous small, bright white dots and larger, fainter circular bokeh-like shapes, which contribute to a cosmic or digital aesthetic. The overall effect is one of dynamic energy and modern design.

# Exercício em Classe

## Migração de Sistema escrito em Visual Basic 6.0

Um Supermercado cresceu muito, saiu de 2 lojas para 30 lojas e contratou sua empresa para um projeto de reescrever o aplicativo existente em uma nova linguagem, utilizando linguagens e framework modernos.

A atual aplicação utiliza a arquitetura de Cliente -> Servidor, sendo que a aplicação são executáveis que rodam no Windows XP e a base de dados é em SQL Server 6.5. São 3 caixas por loja e mais 2 máquinas administrativas, o cliente tem como foco utilizar soluções nas estações que não tenham custo de licenciamento. Existem 4 executáveis na aplicação:

- **Frente de Caixa:** Faz interface com vários dispositivos como por exemplo: impressora fiscal, leitor de código de barras.
- **Módulo de Suprimentos:** Tem o cadastro dos produtos, das compras, dos recebimentos e tem um leitor de código de barra.
- **Módulo de Gerenciamento:** Contém os cadastros. módulos de controle de estoque e de exportação de dados para o sistema Financeiro.
- **Módulo de Relatórios:** Relatórios operacionais que podem ser exportados ou impressos.

O cliente já comprou os computadores novos que vão rodar nos caixas, todos vieram com Linux, ele também já comprou as impressoras fiscais que se comunicarão com os computadores via USB, assim como os leitores de códigos de barra. A empresa já tem um grupo de desenvolvedores e você precisará repassar conhecimento para o time, assim como tudo que será desenvolvido.

O cliente sonha em poder acessar o módulo de gerenciamento da casa dele e em fazer análises em Dashboards e também gostaria que o Site Institucional recebesse uma atenção!

# Exercício em Classe

**Grupo 1:** Front-end Frente de Caixa – Desempenho, Segurança, Usabilidade e Escalabilidade;

**Grupo 3:** Módulo de Suprimentos– Desempenho, Usabilidade e Segurança;

**Grupo 4:** Módulo de Relatórios –Integridade das informações

**Grupo 5:** Back-end – Compatibilidade, Segurança, Disponibilidade e Escalabilidade;

**Grupo 6:** Back-end – APIs de integração com Sistemas Legados – Segurança, Reusabilidade e Rastreabilidade;

**Grupo 7:** Banco de Dados – Compatibilidade, Portabilidade, Segurança e Disponibilidade;

**Grupo 8:** Dashboard – Usabilidade, Segurança;

**Grupo 2:** Aplicativo das aplicações Desktops– Usabilidade, Confiabilidade e Escalabilidade.

**Grupo 9:** Site Institucional – Usabilidade e Manutenibilidade.

**Agradeço**  
a sua atenção!

**Fábio Figueredo**

fabio.figueredo@sptech.school

SÃO  
PAULO  
TECH  
SCHOOL