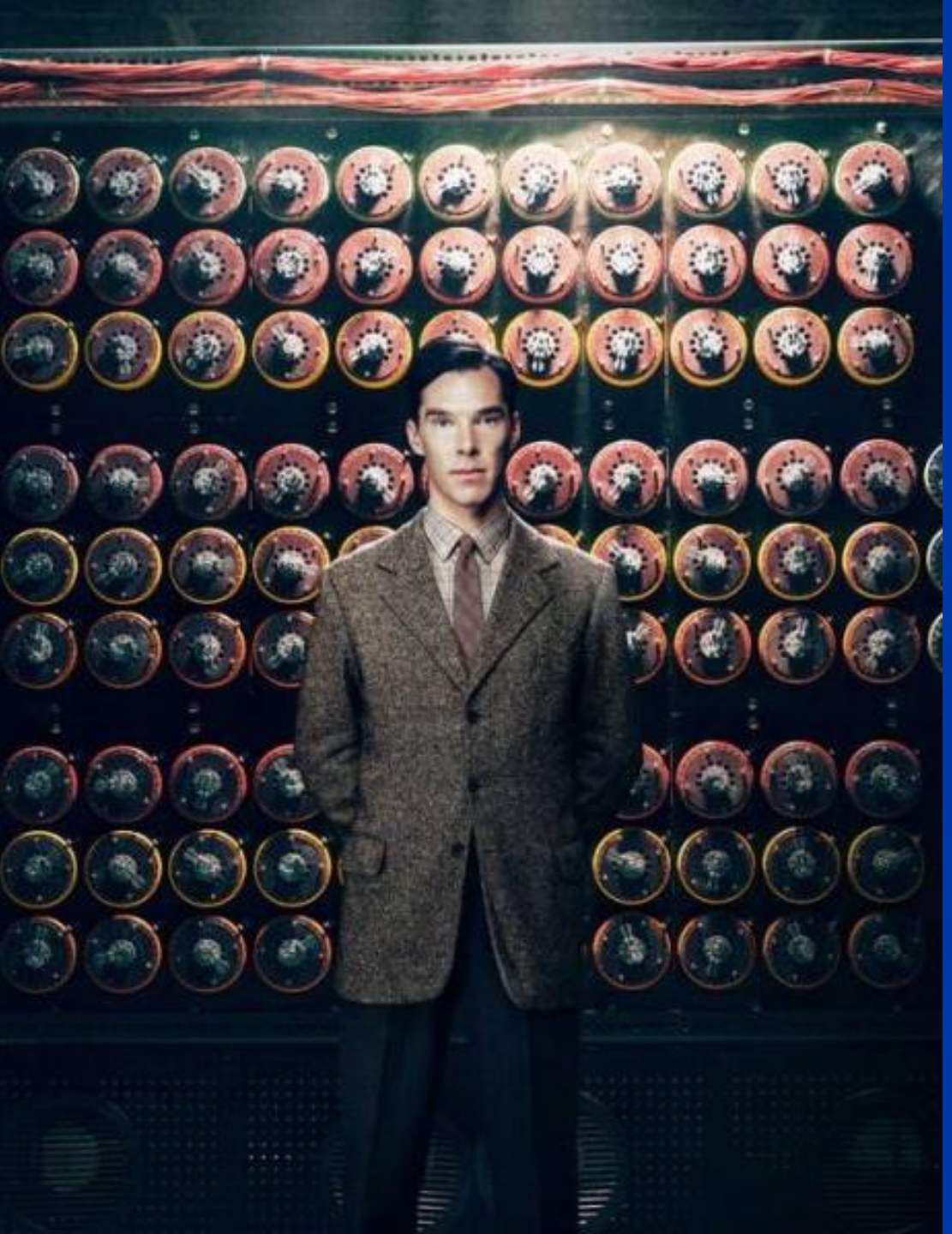
BOOTCAMP 2024
Backend Java

Syslap SOLUTIONS

Fundamentos de Engenharia de Software

SysMap SOLUTIONS



Alan Turing - Imperative

O pai da computação foi **Alan Turing**, foi ele quem desenvolveu a **maquina de Turing**, capaz de ler, interpretar e escrever dados em um dispositivo de armazenamento.

Posteriormente ele implementou essa ideia para construir uma maquina real na qual foi possível programar um *brute-force* para hackear o **enigma**.

Porém outro personagem tão importante como Alan foi seu professor **Alonzo Church**, matemático que fundamentou a teoria da ciência da computação.

SysMap SOLUTIONS

Estrutura de dados

A **Estrutura de Dados** estuda os mais variados **tipos de dados**, quando iniciamos o estudo de uma nova linguagem de programação, entre os primeiros capítulos sempre encontraremos um reservado ao assunto, normalmente apresentando os tipos primitivos de dados.

Ex: *int, boolean, string, etc...*

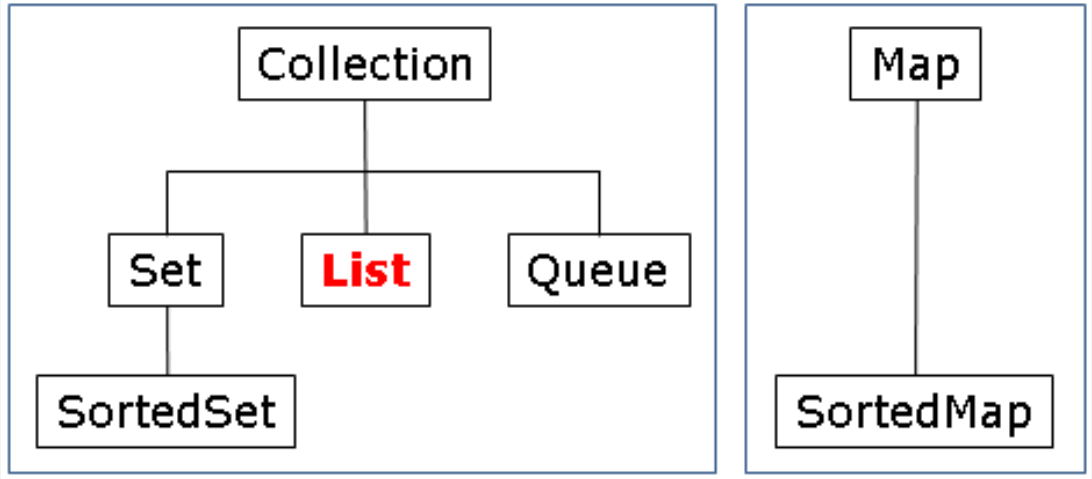
Algumas linguagens possuem tipos customizados, onde o programador consegue agrupar alguns tipos primitivos, assim organizando os dados de tal forma que reflitam as regras de negócio.

Uma outra opção é utilizar o pattern de **Value-Objects**.

Estrutura de dados

As **Arrays, Collections, Listas, Maps...** são outros tipos de estrutura de dados, porém com algumas propriedades especiais, como tamanho e uma ordem de itens indexados, usualmente são listas ligadas.

Por causa destas propriedades podemos iterar essas listas de formas diferentes e transforma-las outras estruturas como pilhas ou grafos e ainda com a capacidade de mapear para outros valores ou reduzir a último valor.



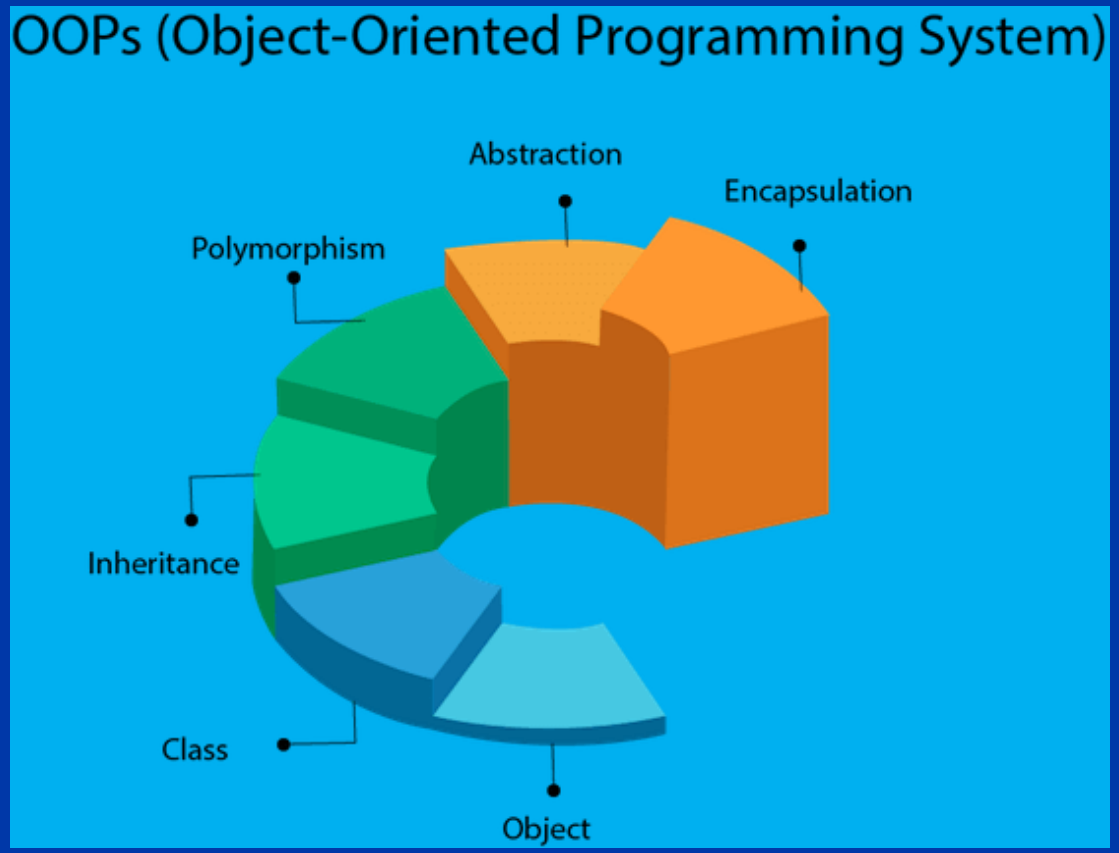
Programação Orientada a Objetos

“

Tem como objetivo modelar os dados em atributos de classes e instanciar seus respectivos objetos, alocando um estado em memoria para cada um e com a capacidade de alterar este estado quando necessário.

Esse tipo de representação de dados em objetos/classes, procura aproximar o sistema que está sendo criado ao que é observado no mundo real.

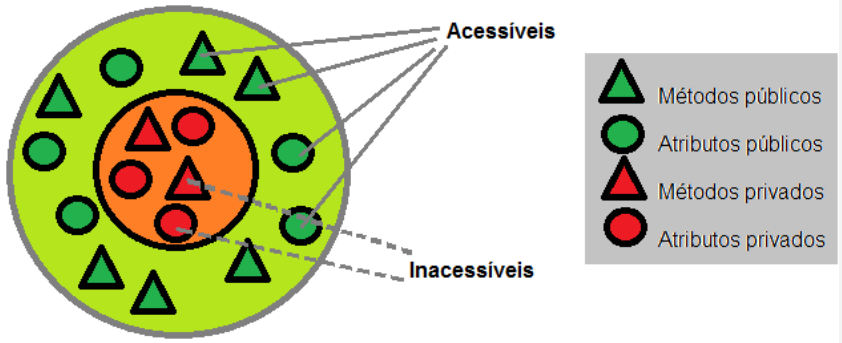
O principal objetivo da OOP é facilitar a reutilização de código e a segurança.



Progamação Orientada a Objetos

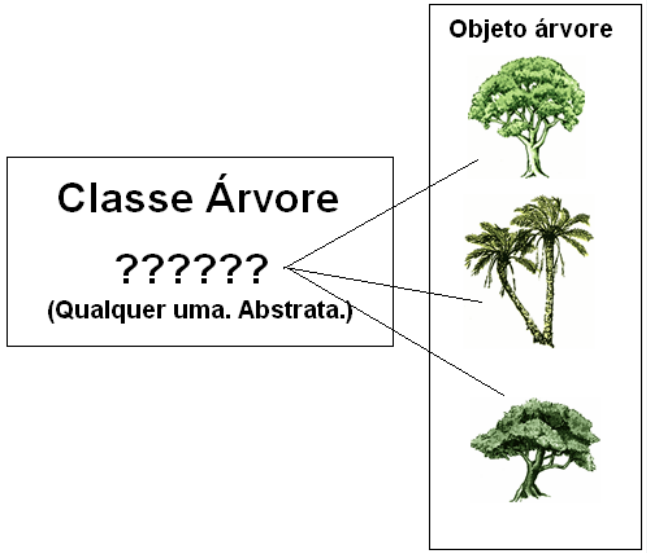
Encapsulamento

É a habilidade de um objeto em esconder dados da visão exterior, permitindo acesso somente por meio de métodos públicos.
Ex: (getters, setters).
Consiste em usar modificadores de acesso privados e fornecer métodos que possam acessá-los de forma segura. Ajuda a prevenir problemas de acesso indevido a dados.



Abstração

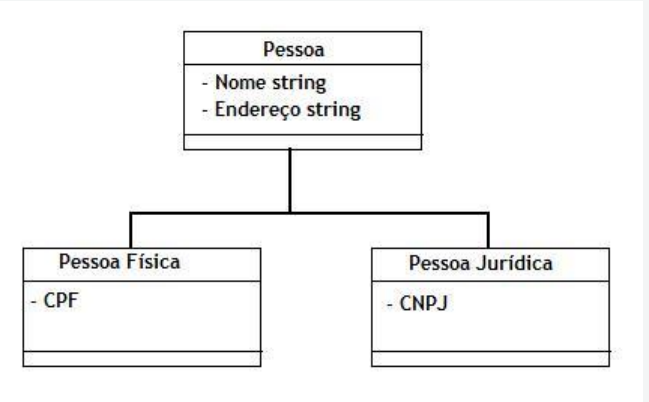
São aquelas classes que não criam instâncias e possuem métodos sem
implementação. São como um molde para serem manipulados por subclasses.
Estas subclasses sobrecarregam os métodos abstratos.



Progamação Orientada a Objetos

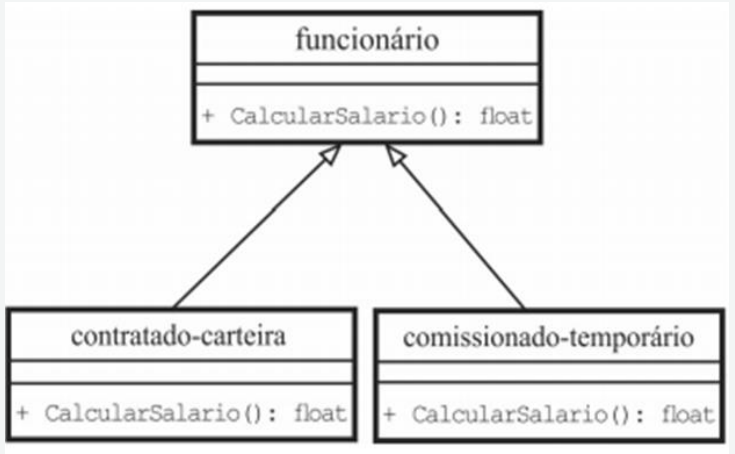
Herança

Permite criar novas classes a partir de classes já existentes, sendo estas filhas da classe base. com isso a reutilização de código e o compartilhamento de características que pode ser especificadas nas classes que as herdam.

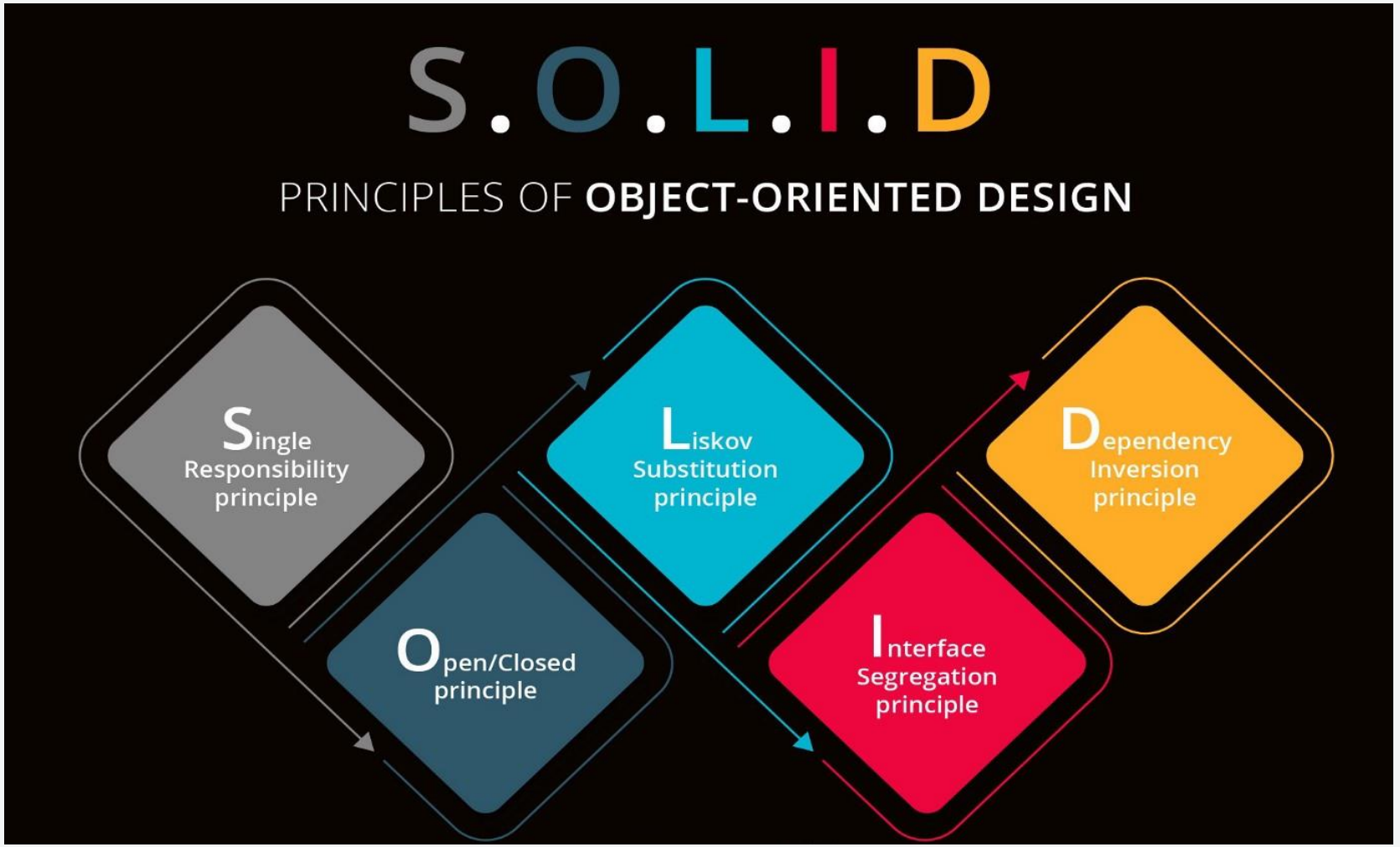


Polimorfismo

Situação na qual um objeto pode adquirir maneiras diferentes dependendo de como foi criado. As superclasses (classes base) tem seus métodos sobrecarregados por subclasses (classes filhas) podendo elas implementá-los da forma que for conveniente
Em outras palavras, o polimorfismo consiste na alteração do funcionamento interno de um método herdado de um objeto pai.



Progamação Orientada a Objetos



Domain Driven Design

O termo foi cunhado por Eric Evans em seu livro de mesmo título publicado em 2003.

"É um conjunto de princípios com foco em domínio, exploração de modelos de formas criativas e definir e falar a linguagem Ubíqua, baseado no contexto delimitado."
Evans, Eric – 2003

Tratando a questão do desenvolvimento de software sob uma ótica de semântica, apesar do core da ideia ser o **domínio**, antes de mais nada **DDD** se trata de um framework de ideias, não um framework propriamente dito como um Spring ou algo do gênero, mas sim um arcabouço que força o time a trabalhar dentro de um mesmo vocabulário, modelando o domínio de forma discreta e em conjunto com os especialistas dos mesmos além de cirandar fronteiras entre as intersecções do modelo de negócio.

Domain Driven Design

Strategic design

Ubiquitous Language

Domain

Sub-domain

Core domain

Domain model

Bounded Context

Context

Context Map

Generic Subdomain

Tactical patterns

Events

Entities

Respositories

Aggregates

Dependency Injection

Factories

Layers

Services

Modules

Domain Driven Design

Domínio

Uma esfera de conhecimento, influência ou atividade. A área temática para a qual o usuário aplica um programa é o domínio do software.

Modelo

Um sistema de abstrações que descreve aspectos selecionados de um domínio e pode ser usado para resolver problemas relacionados a esse domínio.

Linguagem Ubíqua

Uma linguagem estruturada em torno do modelo de domínio e usada por toda a equipe membros dentro de um contexto limitado para conectar todas as atividades da equipe com o software.

Contexto

A configuração em que uma palavra ou declaração aparece que determina seu significado. As declarações sobre um modelo só podem ser entendidas em um contexto.

Contexto Limitado

Uma descrição de um limite (normalmente um subsistema ou o trabalho de um determinado equipe) dentro do qual um determinado modelo é definido e aplicável.

GoF

O Gang Of Four Design Patterns foi criado por 4 autores, sendo eles Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides. Esse padrão de design foi apresentado, pela primeira vez, no livro *"Design Patterns: Elements of Reusable Object-Oriented Software"*

O padrão de design nada mais é do que aquela solução que, em algum momento, será reutilizada para resolver um problema que é comumente encontrado em um software.

Creational

São padrões de criação de objetos.
Exemplos: Singleton, Factory, Prototype, ...

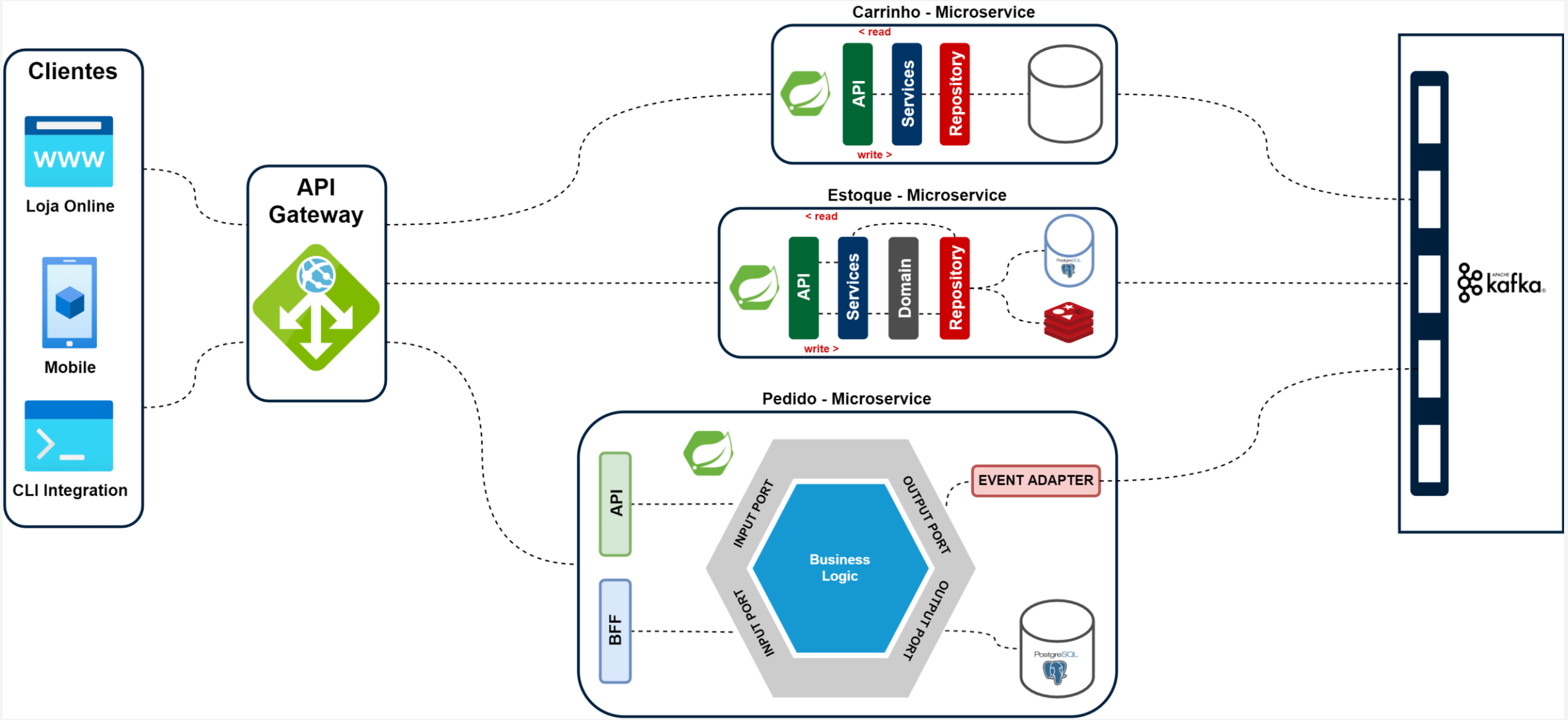
Structural

São padrões que iram facilitar a conclusão do projeto e apoiar de forma ampla no desenvolvimento.
Exemplos: Adapter, Decorator, Facade, ...

Behavioral

São padrões de comunicação entre objetos, apoiando na divisando de responsabilidades.
Exemplos: Mediator, Command, Observer, ...

Design Patterns



Material de Apoio

Microservices // Dicionário do Programador - YouTube

O que são Microsserviços? (Microservices) #HipstersPontoTube - YouTube

Microsserviços ou Monolíticos? Qual você deve escolher? | por André Baltieri #balta - YouTube

Microserviços Java na prática com DDD, CQRS e Event Sourcing - Parte 1 - YouTube

SOLID fica FÁCIL com Essas Ilustrações - YouTube

A Solid Guide to SOLID Principles | Baeldung

Clean Code // Dicionário do Programador - YouTube

Descomplicando "Event Sourcing" - YouTube

Professional Tips

SysMap SOLUTIONS

```ruby
class Repeat
  def print_message
    puts "I Will Not Repeat My Code"
    puts "I Will Not Repeat My Code"
    puts "I Will Not Repeat My Code"
    puts "I Will Not Repeat My Code"
    puts "I Will Not Repeat My Code"
    puts "I Will Not Repeat My Code"
    puts "I Will Not Repeat My Code"
  end
end
```

DRY - Don't Repeat Yourself

DRY é sobre a duplicação do conhecimento, da intenção. Trata-se de expressar a mesma ideia em dois lugares diferentes, possivelmente de duas maneiras totalmente diferentes.

O Argumento do Dry é que cada peça de software deve ter uma representação única, inequívoca e dentro de um sistema.

KISS - Keep It Simple, Stupid

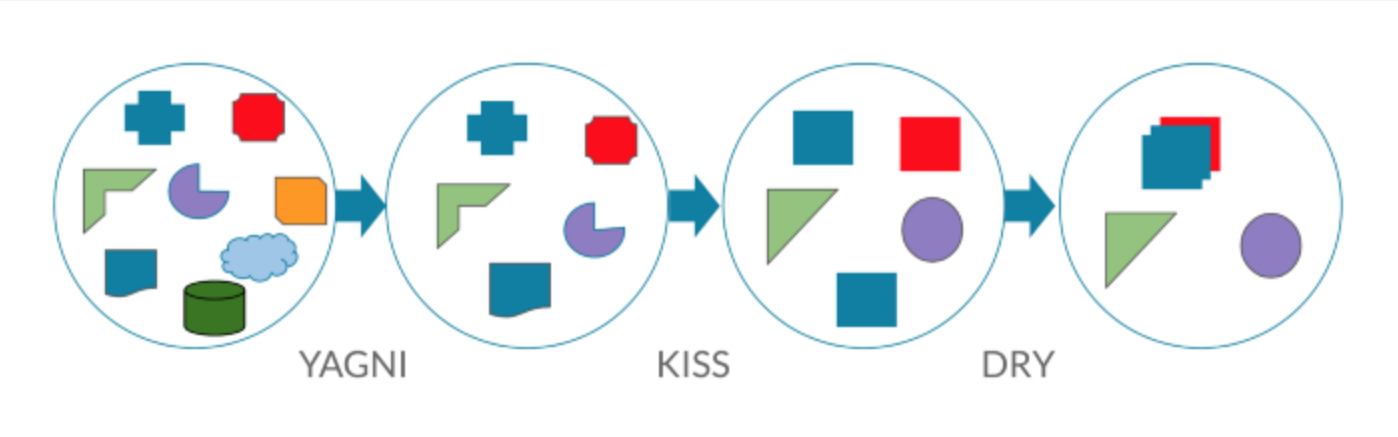
KISS é sobre manter poucas funções em cada modulo, e cada função conter o mínimo possível de linhas, este número pode variar, mas não passa de 50/60 linhas por função.

O Argumento do Kiss é que o código é escrito por devs para ser lido por devs, manter-se simples agiliza inclusive o onboard de novos membros e cria um ambiente saudável para manutenção.

YAGNI - You Ain't Gonna Need It

YAGNI é sobre se manter no core da aplicação, evitando otimizações prematuras ou ainda criação de boilerplates tentando *"facilitar tal feature no futuro"*.

O Argumento do Yagni é sobre gastar tempo e esforço de desenvolvimento com features que estão planejadas para o momento, desta forma evitando o crescimento de complexidade e focando no que realmente importa.



Vamos falar do Java

O Java é uma poderosa linguagem de programação em constante crescimento, sendo uma das mais populares entre os desenvolvedores. Com um forte desenvolvimento orientado a objetos, ela favorece o desenvolvimento de sistemas flexíveis e extensíveis.

O Java é compilado para um bytecode que é interpretado por uma máquina virtual. Sendo assim, as aplicações feitas em Java podem ser executadas em qualquer plataforma que possua a Java Virtual Machine (JVM) instalada.

Tipos de aplicações com Java:

- **Java SE –** Composto por APIs e bibliotecas básicas para possibilitar o desenvolvimento de aplicativos de linha de comando e desktop;

- **Java ME –** A Java Micro Edition é a plataforma voltada para dispositivos móveis e embarcados, com capacidade de processamento reduzida, como os utilizados na criação de produtos para a Internet das Coisas;

- **Java EE –** Voltada para o desenvolvimento de soluções web e corporativas, a Java Enterprise Edition é uma especificação que agrupa diversas outras aplicações;

JVM

Na máquina virtual Java, ou JVM, é onde a sua aplicação será executada. É ela, também, a responsável pela característica multiplataforma do Java. Um programa escrito nessa linguagem será executado em qualquer plataforma, seja ela um notebook, smartphone ou torradeira, que possua uma máquina virtual Java implementada.

Um dos recursos mais conhecidos da JVM é o Garbage Collection. É ele que é acionado com certa frequência para limpar da memória objetos que não estão sendo utilizados, evitando desperdício de espaço e que sua aplicação deixe de funcionar por falta dela.

Collections no Java

O Java dispõe das estruturas de **arrays** e as classes **Vector** e **Hashtable**.

Para atender estruturas utilizando arrays, a partir do Java 1.2, foi criado um conjunto de interfaces e classes denominado Collections Framework, que faz parte do pacote **java.util**.

A Collections Framework contém os seguintes elementos:

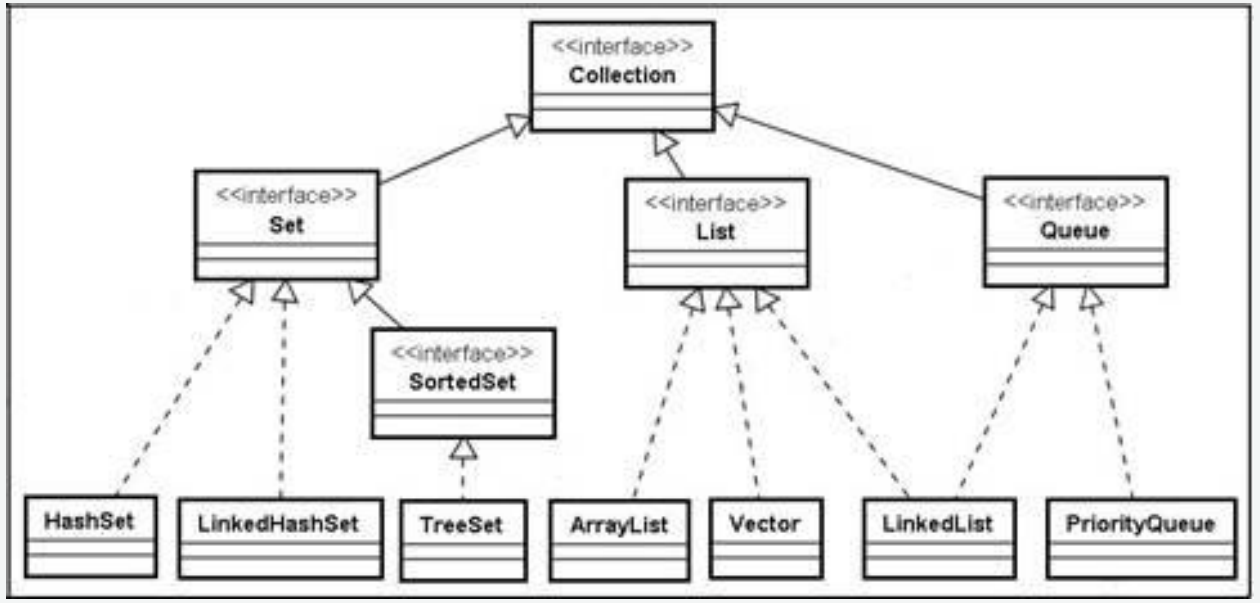
Interfaces: Tipos abstratos que representam as coleções.

Implementações: São as implementações concretas das interfaces;

Algoritmos: São os métodos que realizam as operações sobre os objetos das coleções, tais como busca e ordenação.

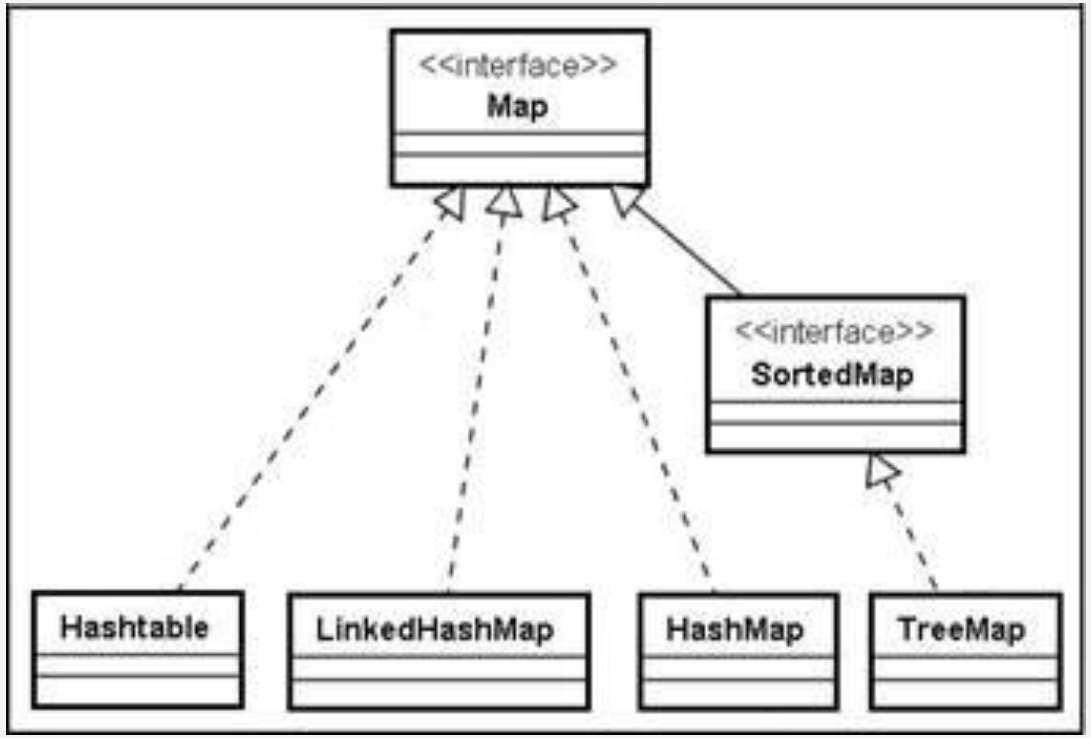
Collections no Java

Abaixo temos a árvore da hierarquia de interfaces e classes da Java Collections Framework que são derivadas da interface **Collection**.



Collections no Java

A hierarquia da Collections Framework tem uma segunda árvore. São as classes e interfaces relacionadas a mapas, que não são derivadas de **Collection**.



Packages Repository

Maven:
Trata-se da ferramenta de gerenciamento de dependências mais utilizada para aplicações Java. Uma ferramenta que te dá apoio na automatização no build e na publicação de suas aplicações.

Trata-se de uma ferramenta de gestão de dependências e um task runner. Em outras palavras, o Maven automatiza os processos de obtenção de dependências e de compilação de projetos Java.

Quando criamos um projeto Maven, este projeto fica atrelado a um arquivo principal: o **pom.xml**. Neste arquivo POM (Project Object Model), nós descrevemos as dependências de nosso projeto e a maneira como este deve ser compilado. Com o Maven, é possível, por exemplo, automatizar a execução de testes unitários durante a fase de build, entre outras automatizações.

Com o Maven, não temos mais a necessidade de baixarmos as dependências de nosso projeto e as configurar dentro do Build Path/Classpath de nossas aplicações.

Packages Repository

Gradle:

O Gradle trata-se de uma ferramenta de build open source bastante poderosa que nos permite configurar arquivos de build por meio da sua DSL (Domain Specific Language) baseada na linguagem Groovy.

Sua principal característica e vantagem é que ele une o melhor da flexibilidade do Ant com o gerenciamento de dependências e as convenções do Maven.

Como são baseados em scripts, os arquivos do Gradle permitem que você realize tarefas de programação em seu arquivo de configuração. Há, ainda, um sistema de plugins que adicionam funcionalidades extras ao core.

Além da DSL, a ideia do Gradle é permitir configurações baseando-se em tasks, ou seja, quando queremos criar algum novo comportamento durante o build, vamos criar uma task!

Entendendo do Spring

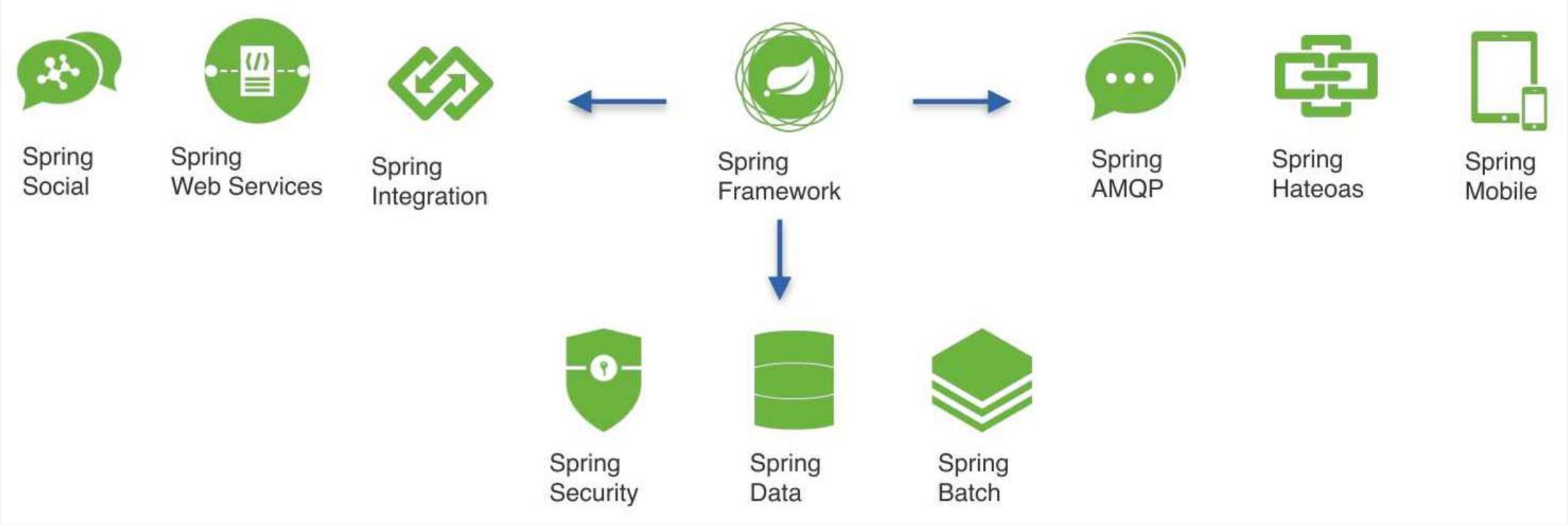
Spring Framework

Fornece um modelo abrangente de programação e configuração para aplicativos corporativos modernos baseados em Java.

Com **spring** você pode desenvolver para qualquer tipo de plataforma!

> **O maior benefício do Spring Boot é que ele nos deixa mais livres para pensarmos nas regras de negócio da nossa aplicação!**

Entendendo do Spring



Entendendo do Spring

Spring Boot

O Spring Boot é um módulo que torna fácil a criação de aplicações Spring. Permite ainda selecionarmos os módulos que desejamos utilizar.

Com configurações rápidas, você consegue, por exemplo, disponibilizar uma aplicação baseada no Spring MVC, utilizando o Hibernate + JPA...

Entendendo do Spring

Spring Data

Tem por objetivo facilitar nosso trabalho com persistência de dados de uma forma geral. E além do Spring Data JPA, ele possui vários outros projetos:

Spring Data Commons
Spring Data Gemfire
Spring Data KeyValue
Spring Data LDAP
Spring Data MongoDB
Spring Data REST
Spring Data Redis
Spring Data for Apache Cassandra

Entendendo do Spring

Spring Security

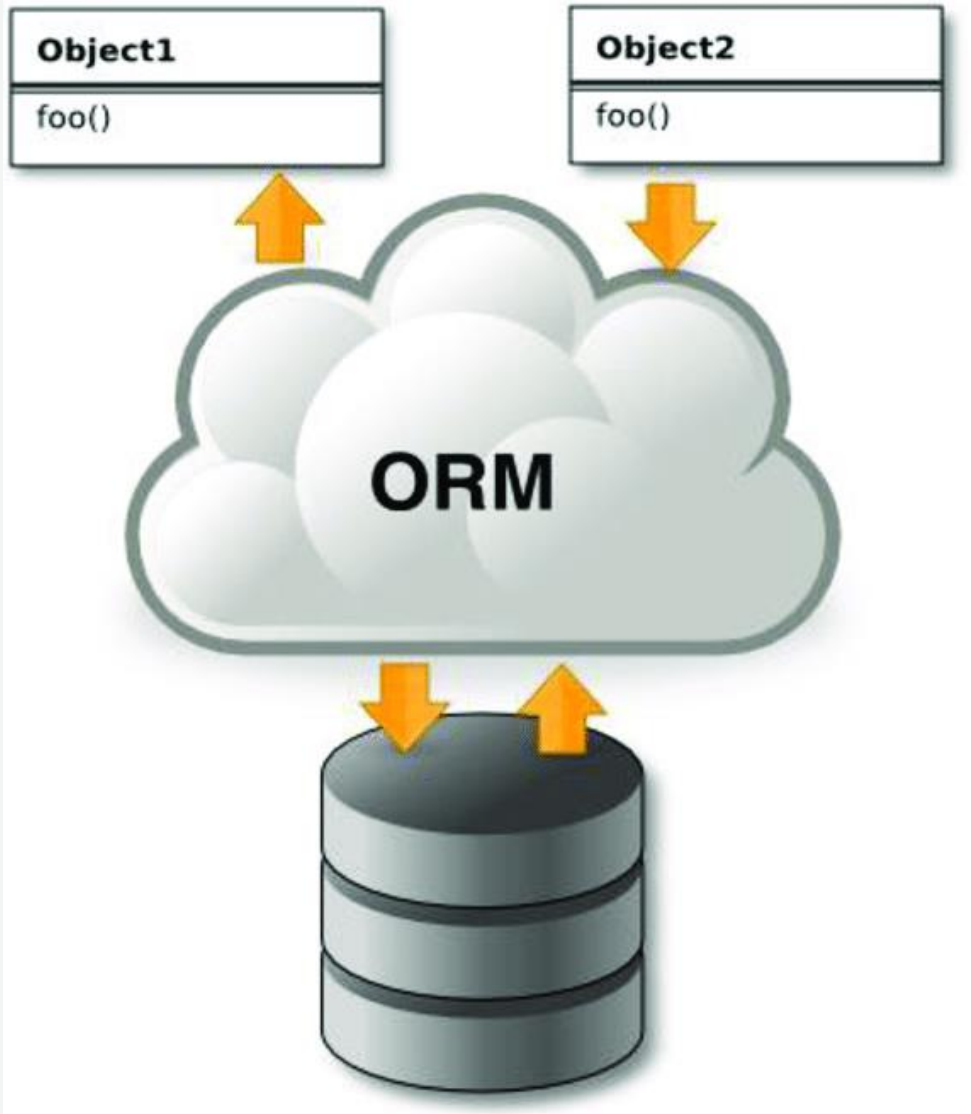
Spring Security tem o foco em tornar a parte de autenticação e autorização uma coisa simples de fazer.

Através das permissões que atribuímos aos usuários autenticados, podemos proteger as requisições web, com uso de tokens por exemplo.

Object Relational Mapper ORM

O ORM cria uma abstração do banco de dados em código, desta forma torna mais prático o entendimento e manipulação das entidades/tabelas e campos.

Alguns ORMs mais conhecidos: JPA, Spring Data, nHibernate, Entity Framework.



Entendendo do Spring Loggin

Log em sistemas é muito importante, principalmente no cenário atual, no qual existem inúmeras integrações entre sistemas. Assim, é importante saber quais informações são trafegadas de um sistema para o outro, se os dados chegaram corretamente, entre outras informações.

Para a correta inclusão de instruções de Log nas aplicações, é necessário ter bem claros os conceitos de Log.

Quais informações devem ser logadas? Qual é o melhor formato para armazenar estes dados? Estas são apenas duas das perguntas que nos auxiliam na decisão do que logar.

Como alternativa para o Logging oferecido nativamente pelo Java, surgiu o **Log4j** que realiza em suma as mesmas função do Common Logging porém algumas funcionalidades extras.

Log4J - Log4j divide os logs em 5 níveis hierárquicos, São eles: DEBUG, INFO, WARN, ERROR, FATAL. Com configurações globais e simplificadas, rapidamente você terá a coleta de logs sendo feita em sua aplicação.

Entendendo do Spring
Testes Unitários

Log em sistemas é muito importante, principalmente no cenário atual, no qual existem inúmeras integrações entre sistemas. Assim, é importante saber quais informações são trafegadas de um sistema para o outro, se os dados chegaram corretamente, entre outras informações.

Para a correta inclusão de instruções de Log nas aplicações, é necessário ter bem claros os conceitos de Log.

Quais informações devem ser logadas? Qual é o melhor formato para armazenar estes dados? Estas são apenas duas das perguntas que nos auxiliam na decisão do que logar.

Como alternativa para o Logging oferecido nativamente pelo Java, surgiu o **Log4j** que realiza em suma as mesmas função do Common Logging porém algumas funcionalidades extras.

Log4J - Log4j divide os logs em 5 níveis hierárquicos, São eles: DEBUG, INFO, WARN, ERROR, FATAL. Com configurações globais e simplificadas, rapidamente você terá a coleta de logs sendo feita em sua aplicação.

Entendendo do Spring Documentando APIs

REST: Representational State Transfer refere-se a um grupo de restrições de design dentro da arquitetura de software que geram sistemas distribuídos eficientes, confiáveis e escaláveis. Um sistema é denominado RESTful quando adere a todas essas restrições.

A ideia básica do REST é que um recurso, por exemplo um documento, seja transferido com seu estado bem definido, padronização de operações e formatos, ou serviços que se autodenominem RESTful, quando modificam diretamente o tipo de documento, ao invés de desencadear ações em algum lugar.

JSON: JSON é basicamente um formato leve de troca de informações/dados entre sistemas. Mas JSON significa **JavaScript Object Notation**.

Além de ser um formato leve para troca de dados é também muito simples de ler. Mas quando dizemos que algo é simples, é interessante compará-lo com algo mais complexo para entendermos tal simplicidade não é? Neste caso podemos comparar o JSON com o formato XML.

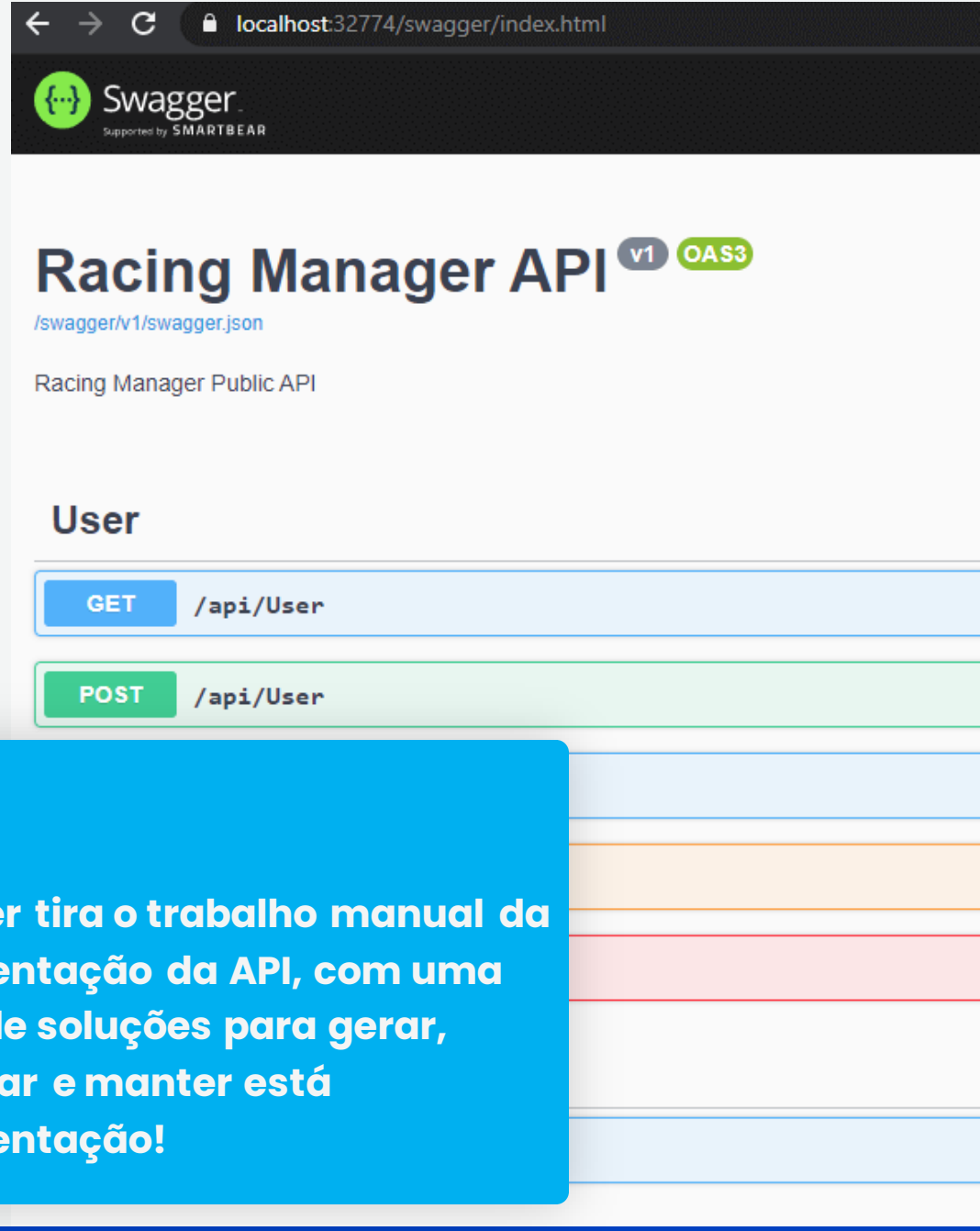
Entendendo do Spring

Documentando APIs

Swagger

Usado em um conjunto de ferramentas de software de código aberto para projetar, construir, documentar e usar serviços REST.

Swagger inclui documentação automatizada, geração de código e geração de casos de teste.



> **Swagger tira o trabalho manual da documentação da API, com uma gama de soluções para gerar, visualizar e manter está documentação!**

Entendendo do Spring Brokers de Mensageria

Brokers, são tipicamente servidores de fila que implementam as técnicas de pub/sub e ou RPC entre outras que tem como proposito fazer a comunicação entre os serviços ou microsserviços.

Antes de definirmos a utilização de um broker devemos nos atentar para algumas características do mesmo, por exemplo temos

Protocolo: AMQP, MQTT, XMPP, (G)RPC...

Tipo da Mensagem: Protobuf, Json, Texto, Binário, Avro...

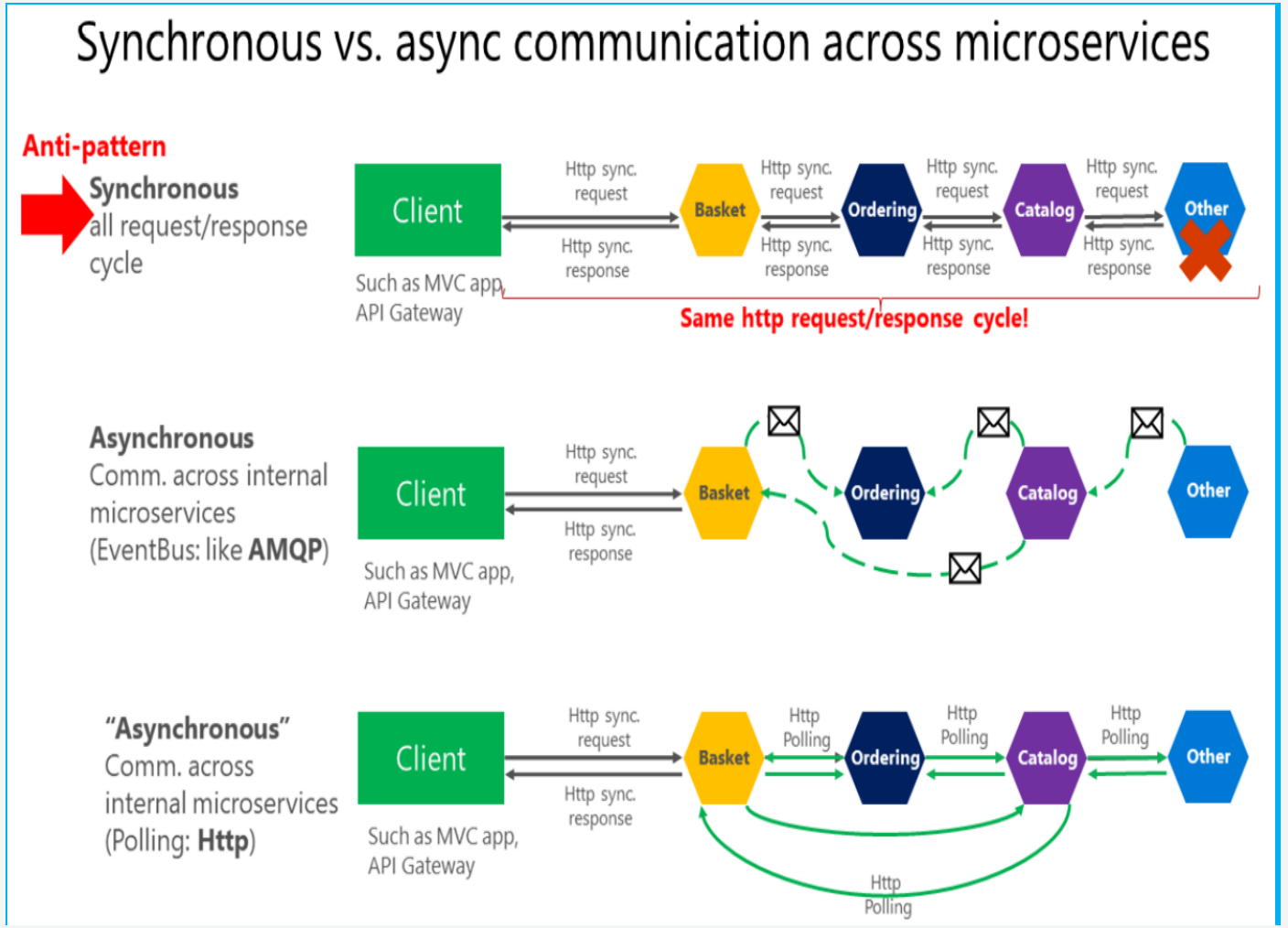
Arquitetura do broker:

- **RabbitMQ (AMQP):** Como um servidor de filas, a regra fica armazenada no próprio servidor baseada em rotas e dispatchers, onde a tendencia do servidor eh a fila zerar;
- **Kafka (RPC):** Como um log storage, a regra fica a cargo de quem produz ou consome as filas, onde a tendencia do servidor eh a fila crescer eternamente;
- **Mosquito (MQTT):** Como um servidor de filas, a regra fica a cargo de quem produz ou consome as filas, a tendencia do servidor é manter um buffer de mensagens fazendo um reload a fim de não estourar o espaço;
- **EJabberD (XMPP):** Como um servidor de filas, a regra fica armazenada no próprio servidor baseada em rotas e dispatchers, onde a tendencia do servidor é a fila zerar;

Comunicação **Síncrona** vs **Assíncrona**

“

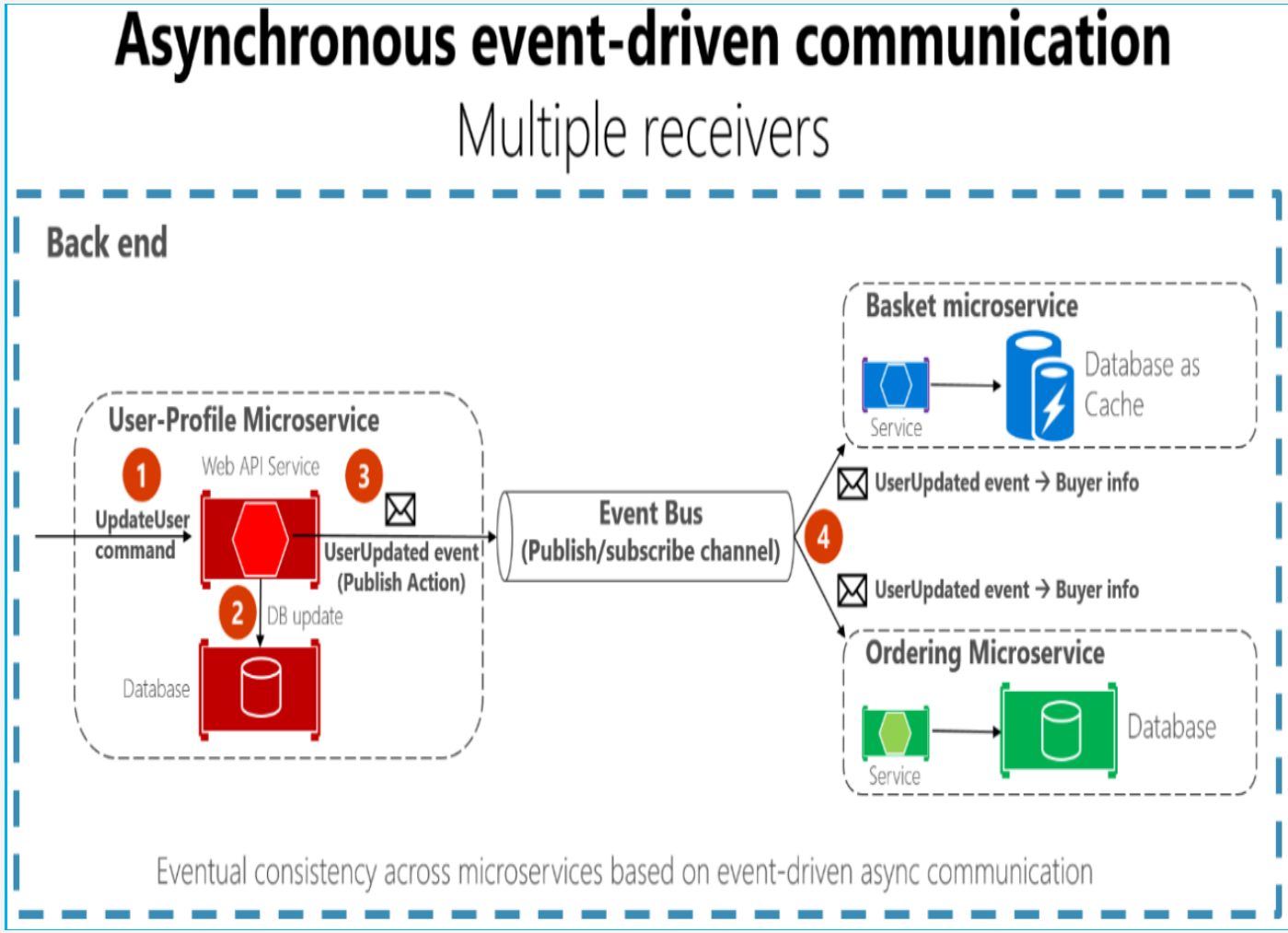
Em um ecossistema de software distribuído é recomendado o uso de comunicação assíncrona entre os microsserviços!



Comunicação **Baseada em Eventos**

“

Na comunicação assíncrona com eventos, um microsserviços publica eventos em uma fila e muitos microsserviços podem se inscrever nesta fila, para serem notificados e processarem os eventos recebidos!



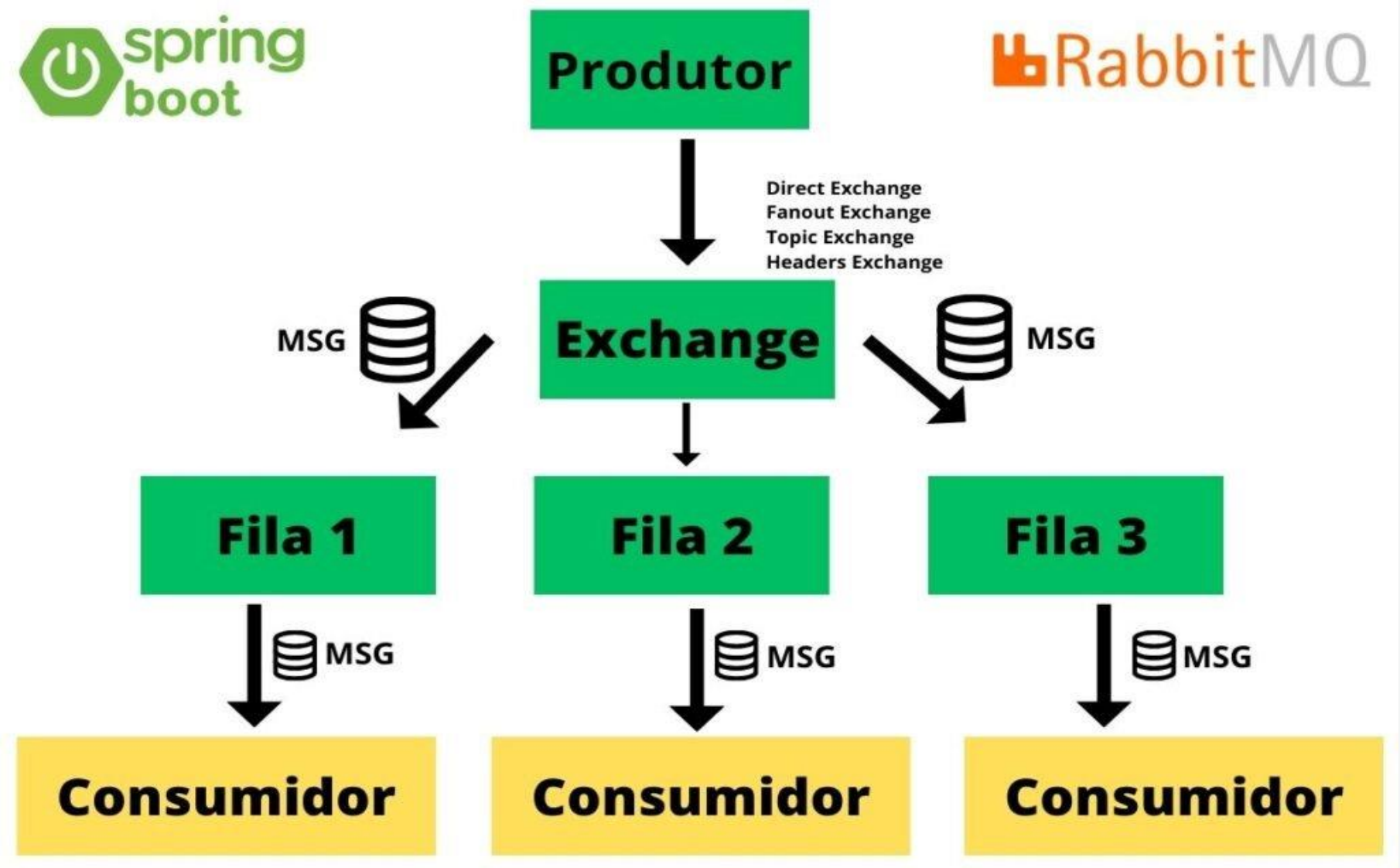
Entendendo o Rabbit MQ

Plataforma open-source message-broker, altamente consolidado e utilizado por quem trabalha com comunicação entre sistemas

Operando de forma assíncrona, ele age como um intermediário que processa as nossas mensagens entre produtores e consumidores, além de contar com filas que possuem diversas opções de encaminhamento.

> **O projeto tem objetivo de orquestrar com performance e baixa latência um grande volume de mensagens!**

Entendendo o Rabbit MQ



Entendendo o Rabbit MQ

O fluxo básico de mensagens no RabbitMQ

1.O produtor publica uma mensagem em uma Exchange específica, especificando a chave de roteamento (routing key).

2.A Exchange, baseada na chave de roteamento e nas regras definidas, encaminha a mensagem para a(s) fila(s) correta(s).

3.Os consumidores estão continuamente verificando as filas em busca de novas mensagens. Quando uma mensagem chega, o RabbitMQ a entrega a um consumidor disponível.

4.O consumidor processa a mensagem e confirma ao RabbitMQ que a mensagem foi recebida e processada com sucesso (acknowledgment). Nesse momento, o RabbitMQ remove a mensagem da fila.

Entendendo o Rabbit MQ

O RabbitMQ é altamente escalável e pode lidar com grandes volumes de mensagens. Ele suporta vários protocolos de mensagens, incluindo:

AMQP (Advanced Message Queuing Protocol)
MQTT (Message Queuing Telemetry Transport)
STOMP (Simple Text Oriented Messaging Protocol)

Entendendo o Rabbit MQ

Producer

Responsável por enviar uma mensagem para um tópico específico.

O próprio Rabbit organiza as mensagens em uma partição, garantindo sempre a ordem das mensagens produzidas.

Entendendo o Rabbit MQ

Consumer

Consumer é possível ler essas mensagens nas fila de forma ordenada.

Importante saber que, ao consumir uma mensagem, a mesma não é processada como read.

Material de Apoio

https://virandoprogramador.com.br/o-que-e-rabbitmq/

https://fullcycle.com.br/como-funciona-o-rabbitmq/

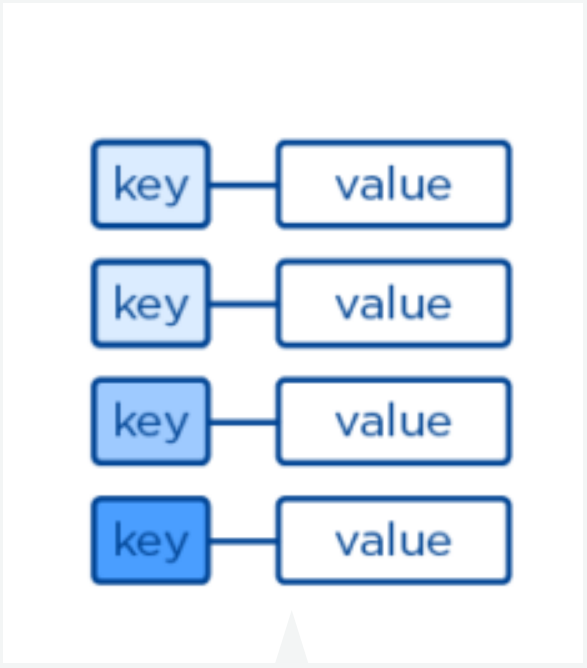
https://dev.to/gabrielgcj/o-que-e-rabbitmq-e-qual-sua-funcao-na-programacao-468j

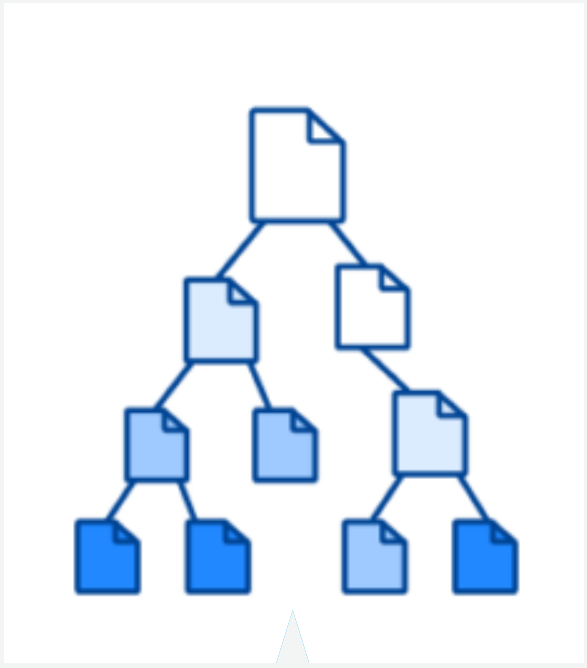
https://dev.to/efoscarini/primeiros-passos-com-rabbitmq-3p9

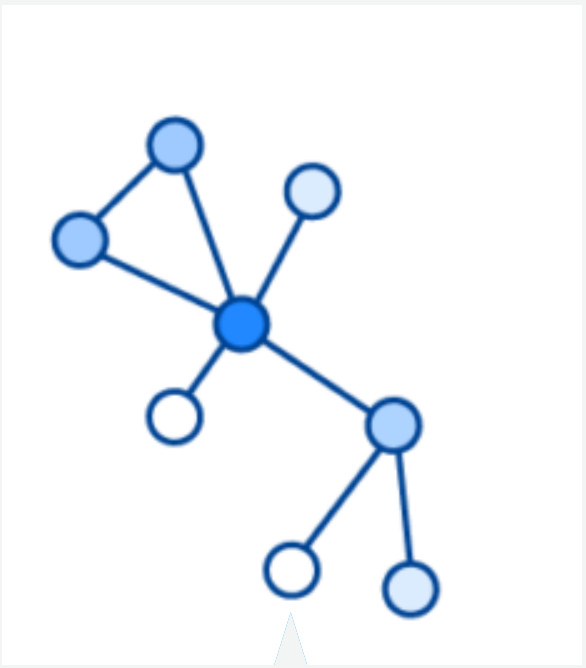
NoSQL

São considerados Banco de Dados porém não obedecem aos requisitos de Bancos de Dados Relacionais, possuem um sistema de persistência, formas de acesso aos dados, com a capacidade de realizar escrita e leitura porém com as habilidades de clusterização e Tolerância a Partições.

NoSQL







Chave-Valor

Colunar

Documental

Grafo

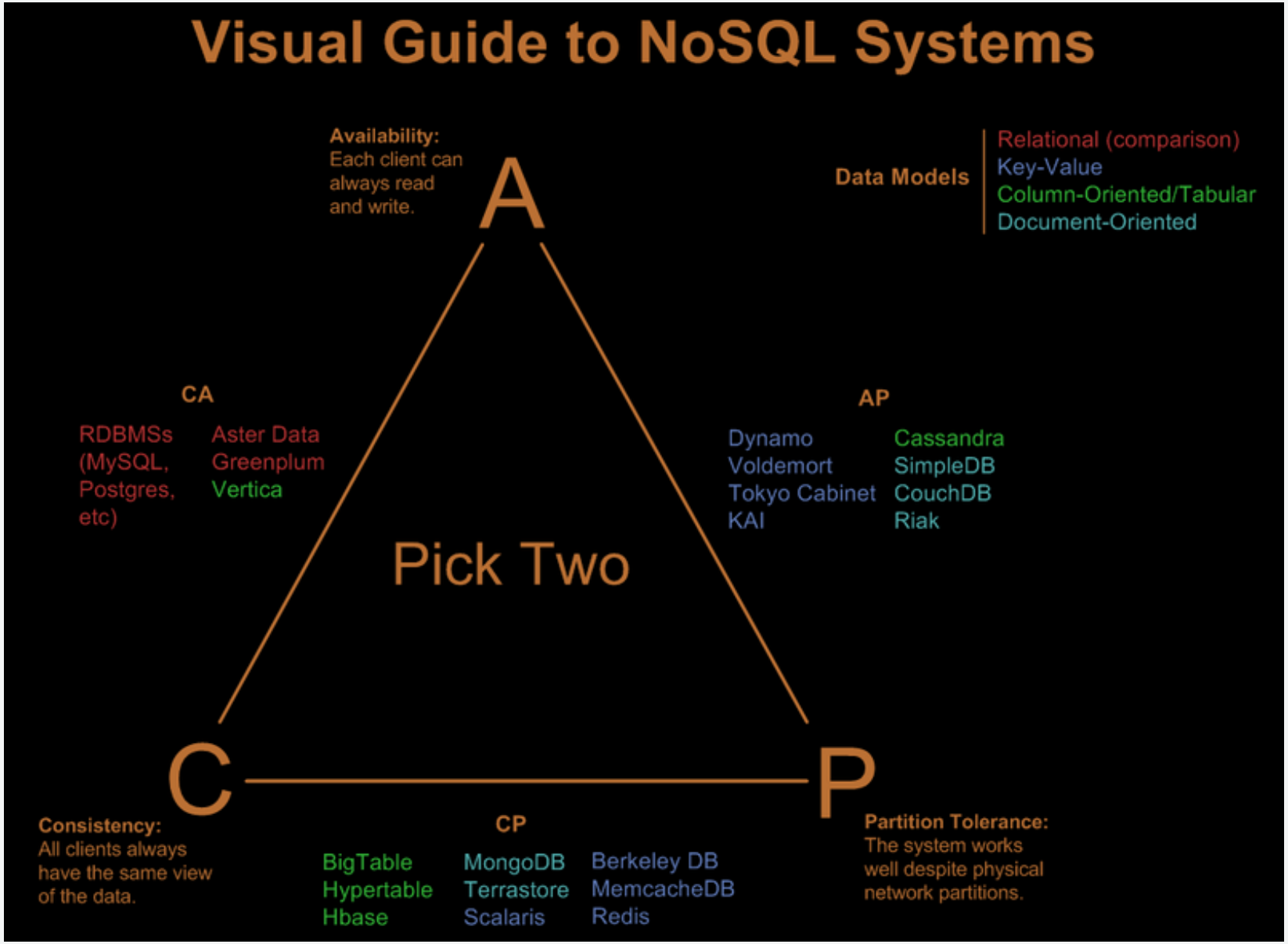
Casualmente distribuídos em 4 tipos

CAP Theorem

A ideia por traz do teorema CAP é escolher a sua base de dados baseado em princípios de tolerância a partição. Bancos de dados relacionais (CA) conseguem apenas escalar verticalmente, ou seja, só é possível aumentar a capacidade de resposta da base aprimorando o hardware, enquanto que, as bases NoSQL (AP/CP) possuem a habilidade de escalar horizontalmente, isto é, adicionar novos nós no cluster e por consequência aumentar a capacidade de resposta das bases de maneira virtualmente ilimitada, porém para isso precisamos escolher entre disponibilidade ou consistência.

AP: Nos Bancos NoSQL considerados eventualmente consistente, a base passa por momentos de inconsistência e isso significa que as versões dos dados podem esta desatualizadas, porem sempre esta disponível e isso significa que mesmo com uma partição de rede ainda consegue receber leituras e escritas.

CP: Bancos NoSQL considerados consistente, a base sempre esta consistente e isso significa que todos os clientes possuem as mesmas versões dos dados e mesmo com uma partição a base consegue receber as requisições de leitura e escrita, porem em caso de uma condição de corrida em escritas algumas requisições podem retornar erros de escrita.



[C] Consistency
[A] Availability
[P] PartitionTolerance

PostgreSQL

O PostgreSQL ou somente “postgres” é um sistema de gerenciamento de banco de dados objeto relacional (**SGBDOR**), desenvolvido como projeto de código aberto com mais de 30 anos de desenvolvimento ativo sob licença do TLP (The PostgreSQL License). .

PostgreSQL

PostgreSQL

Algumas características desse SGBDOR são:

- Consultas complexas.
- Software robusto e de alta qualidade com código de manutenção e bem comentado.
- Chaves estrangeiras.
- Integridade transacional.
- Controle de concorrência multi-versão.
- Suporte ao modelo híbrido objeto relacional.
- Facilidade de acesso.
- Linguagem procedural em várias linguagens para procedimentos armazenados.
- Indexação por texto.

PostgreSQL

PostgreSQL

Tipo de Dados

- **Primitivos:** Integer, Numeric, String, Boolean
- **Estruturada:** Date/Time, Array, Range, UUID
- **Documento:** JSON/JSONB, XML, Key-value (Hstore)
- **Geometria:** Ponto, Linha, Círculo, Polígono
- **Customizações:** Tipos compostos e personalizados

PostgreSQL

Material de Apoio

https://blog.geekhunter.com.br/tutorial-postgresql-introducao-pratica-ao-servico/

https://www.devmedia.com.br/postgresql-tutorial/33025

https://medium.com/@habbema/postgresql-badfbba3e533

Versionamento – Git/GitHub

O git é um sistema de controle de versão distribuído gratuito e de código aberto projetado para lidar com tudo, desde projetos pequenos a muito grandes com velocidade e eficiência.

O **Github** é um "repositório" de **Gits** online dividido por usuários.
Onde o mundo constrói software, milhões de desenvolvedores e empresas criam, enviam e mantêm seus softwares.

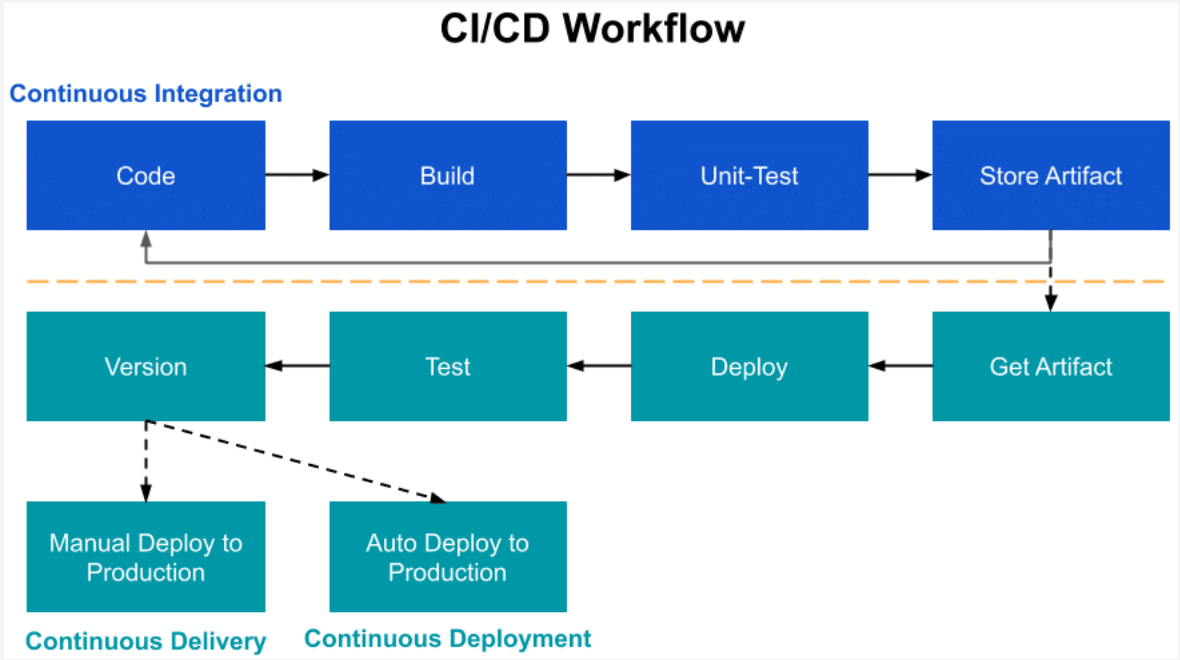
Alguns repositórios **git** são criados no modelo **Monorepo** e precisam de ferramentas especiais para manipula-los

CI/CD – Botando em Produção

CI/CD preenche as lacunas entre as atividades e equipes de desenvolvimento e operação, reforçando a automação na compilação, teste e implantação de aplicativos.

O processo contrasta com os métodos tradicionais, onde todas as atualizações eram integradas em um grande lote antes de lançar a versão mais recente.

As práticas modernas de DevOps envolvem desenvolvimento contínuo, teste contínuo, integração contínua, implantação contínua e monitoramento contínuo de aplicativos de software ao longo de seu ciclo de vida de desenvolvimento.



Obrigado!

Andryev Lemes