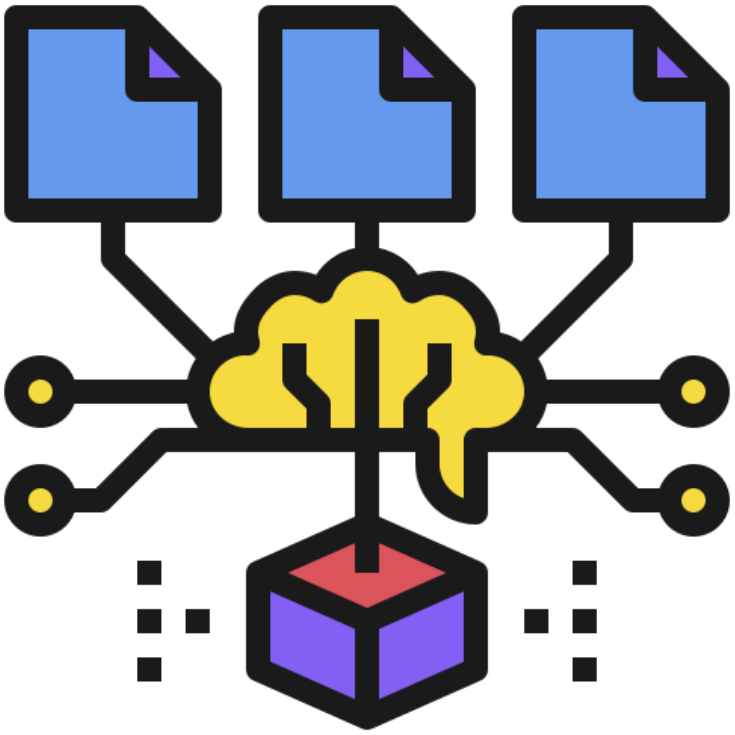
Arquitetura de software



Diogo Antonio Sperandio Xavier

SUMÁRIO

Introdução

O objetivo do material em questão é apresentar um overview sobre **Arquitetura de software. Não são necessários conhecimentos em programação** para compreender os conceitos e a ideia, mas caso possua algum, fica mais fácil o aprofundamento no tema.

Espero que esse livro possa colaborar com seu desenvolvimento de alguma forma, seja com conhecimento ou com um passo-a-passo que melhore sua venda e seus negócios. Caso isso aconteça, já estou de antemão muito feliz por isso.

Conceitos

Primeiramente, vamos expor alguns termos relevantes para o tema como **XML** (Extensible Markup Language), **URI** (Uniform Resource Identifier), **API** (Application Programming interface), **SOAP** (Simple Object Access Protocol), **WSDL** (Web Services Description Language), **XSD** (XML Schema Definition), e **REST** (Representational State Transfer).

XML

É uma recomendação da W3C para gerar linguagens de marcação para necessidades especiais. Muito comum para integrar aplicações e um grande benefício é facilitar a separação do conteúdo.

URI

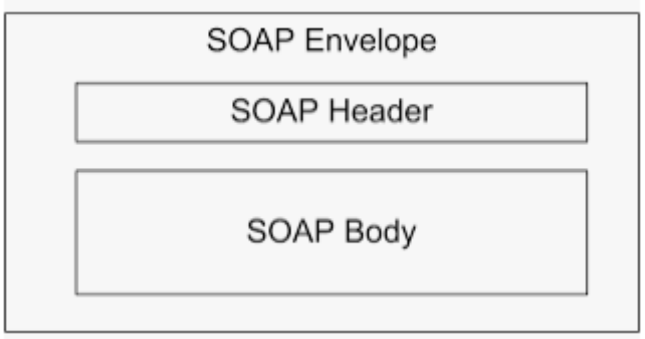
É o termo técnico que representa uma cadeia de caracteres compactada para identificação ou denominação de um recurso da internet. Seria o **URL + HTTP**.

API

É o conjunto de rotinas e padrões estabelecidos por um software para utilização das suas funcionalidades utilizando apenas seus serviços sem se envolver em suas funcionalidades.

SOAP

É um protocolo para troca de informações estruturadas em uma plataforma descentralizada e distribuída e é baseado em XML para seu formato de mensagem e normalmente baseia-se em outros protocolos da **camada de aplicação**, mais notavelmente em **chamada de procedimento remoto** (RPC) e **Protocolo de transferência de hipertexto** (HTTP), para negociação e transmissão de mensagens.



Estrutura SOAP

Ferramenta para utilização: SoapUI.

SOAP Envelope: Primeiro elemento do documento e encapsula toda a mensagem SOAP;

SOAP Header: Elemento onde possui informações de atributos e metadados da requisição. Ex: IP origem, DNS, Token, credenciais de inicialização, entre outros.

SOAP Body: Contém os detalhes da mensagem.

Ex:

```xml
<soap:Envelope xmlns:soap="https://www.w3.org/2003/05/soap-envelope/">
    <soap:Header>

    </soap:Header>

    <soap:Body>
        <m:MetodoEndereco xmlns:m="https://www.exemplo.org/endereco">
            <m:Cidade>Vila Velha</m:Cidade>
            <m:CEP>12123-123</m:CEP>
            <m:Logradouro>Rua exemplo</m:Logradouro>
            <m:Numero>20</m:Numero>
        </m:MetodoEndereco>
    </soap:Body>

</soap:Envelope>
```

WSDL

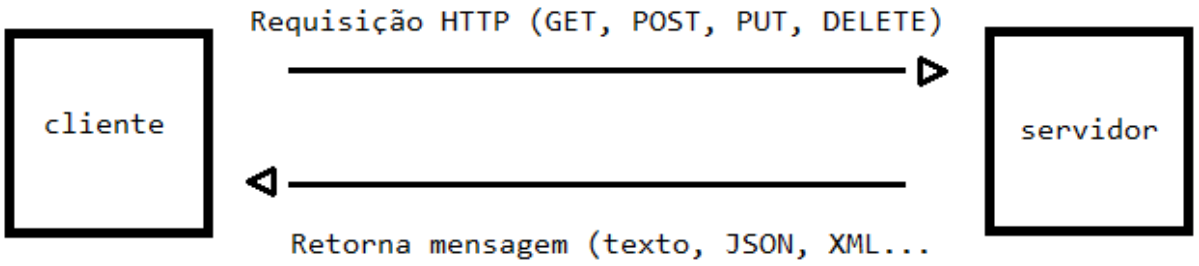
É um contrato de serviço feito em um documento XML onde é descrito o serviço, especificações de acesso, operações e métodos.

XSD

É um schema no formato XML usado para definir a estrutura de dados que será validada no XML. O XSD funciona como uma documentação de como deve ser montado o SOAP Message (XML) que será enviado através do Web Service.

REST

É um tipo de arquitetura aplicado para fornecer padrões entre sistemas de computação na web para facilitar a comunicação entre eles. Nesse estilo de arquitetura, a comunicação entre cliente e servidor pode ser feita de forma independente, sem que cada um conheça o outro, permitindo assim que o código do lado do cliente possa ser alterado a qualquer momento, sem afetar a operação do servidor (o contrário também é válido). Pode trabalhar com XML, JSON, entre outros formatos de arquivo.



GET: Solicita a representação de um recurso;

POST: Solicita a criação de um recurso;

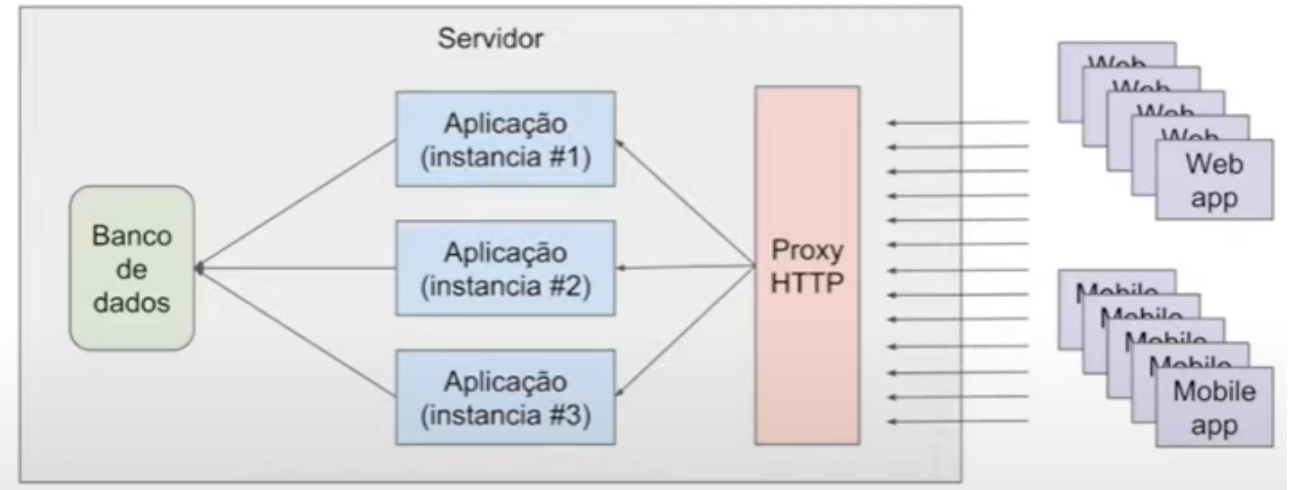
PUT: Solicita a atualização de um recurso;

DELETE: Solicita a exclusão de um recurso.

Tipos de arquitetura

Monólito

A arquitetura monolítica é um sistema único, não dividido, que roda em um único processo, uma aplicação de software em que diferentes componentes estão ligados a um único programa dentro de uma única plataforma. Se uma alteração é feita em um dos serviços, todas as outras partes do projeto terão que subir para o servidor com essa alteração.



Arquitetura monolítica

Vantagens:

- **Mais simples de desenvolver**: a organização fica concentrada em um único sistema;
- **Simples de testar**: é possível testar a aplicação de ponta a ponta em um único lugar;
- **Simples de fazer o deploy para o servidor**: a alteração é simplesmente feita e pronto;
- **Simples de escalar**: como é só uma aplicação, se for preciso adicionar mais itens, é simplesmente ir adicionando conforme a necessidade.

Desvantagens:

- **Manutenção**: a aplicação se torna cada vez maior e o código cada vez mais difícil de entender, com isso, é cada vez mais desafiador fazer alterações rápidas e subir o servidor;
- **Alterações**: para cada alteração feita é necessário realizar um novo deploy de toda a aplicação;
- **Linha de código**: uma linha de código errada pode deixar todo o sistema inoperante;
- **Linguagens de programação**: não há flexibilidade em linguagens de programação. A linguagem do começo deverá ser seguida sempre. Para

utilizar outra linguagem de programação, deve-se mudar todo o código ou mudar a arquitetura.

Microsserviços

É o nome dado a uma arquitetura que estrutura a aplicação criando uma coleção de serviços. É basicamente um monólito dividido em vários serviços separados e independentes de forma que cada um desses serviços acesse uma camada do banco de dados ou somente algum serviço externo. Para cada serviço é realizado um deploy e em caso de necessidade de acesso ao banco, por exemplo, referente a outro serviço, ocorre uma comunicação entre eles e mantém tudo independente.

Vantagens gerais:

- **Teste e manutenção fáceis:** tudo é feito de forma separada e mais rápida;
- **Independência e agilidade**: os deploys de cada microsserviços são totalmente independentes e mais rápidos;
- **Objetividade:** a organização é feita de acordo com as regras do produto e do negócio;
- **Flexibilidade:** é possível dividir em equipes para trabalhar de forma separada e totalmente independente em cada serviço. É possível criar cada serviço com uma linguagem de programação.

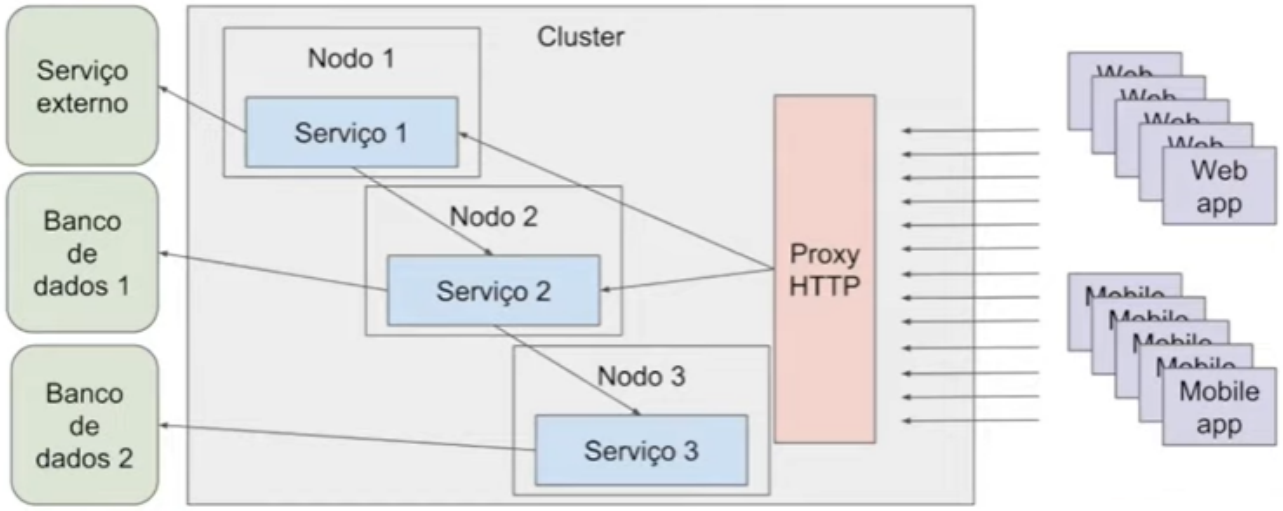
Desvantagens gerais:

- **Exige muita atenção e cuidado na divisão dos serviços:** é gasto mais tempo para dividir adequadamente o sistema de forma que no futuro a aplicação não sejam vários sistemas monolíticos que separados e com funções que se repitam;
- **Há replicação de código de resposta ou de infraestrutura:** o que existe de padrão em um serviço provavelmente existirá nos outros serviços também;
- **Complexidade no gerenciamento da aplicação:** é um ponto a se tomar cuidado para que a organização sempre exista mesmo que novas features sejam implementadas no futuro.

Gerenciamento de erros

Em processos assíncronos (exemplo abaixo microsserviços #2) e em pipeline (microsserviços #3), a complexidade é maior. A maneira de solucionar esses problemas mais comum é com o uso de **DLQ**(dead letter queue) e **filtros de re-tentativas**.

Microsserviços #1



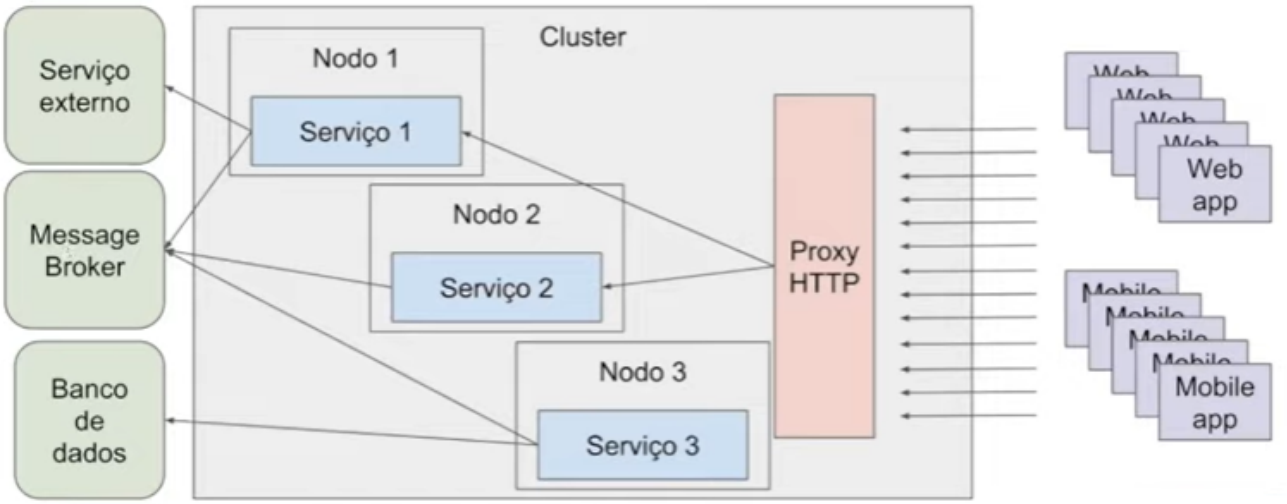
Vantagens:

- **Stack dinâmica;**
- **Simples escalabilidade.**

Desvantagens:

- **Acoplamento;**
- **Monitoramento complexo;**
- **Provisionamento complexo.**

Microsserviços #2



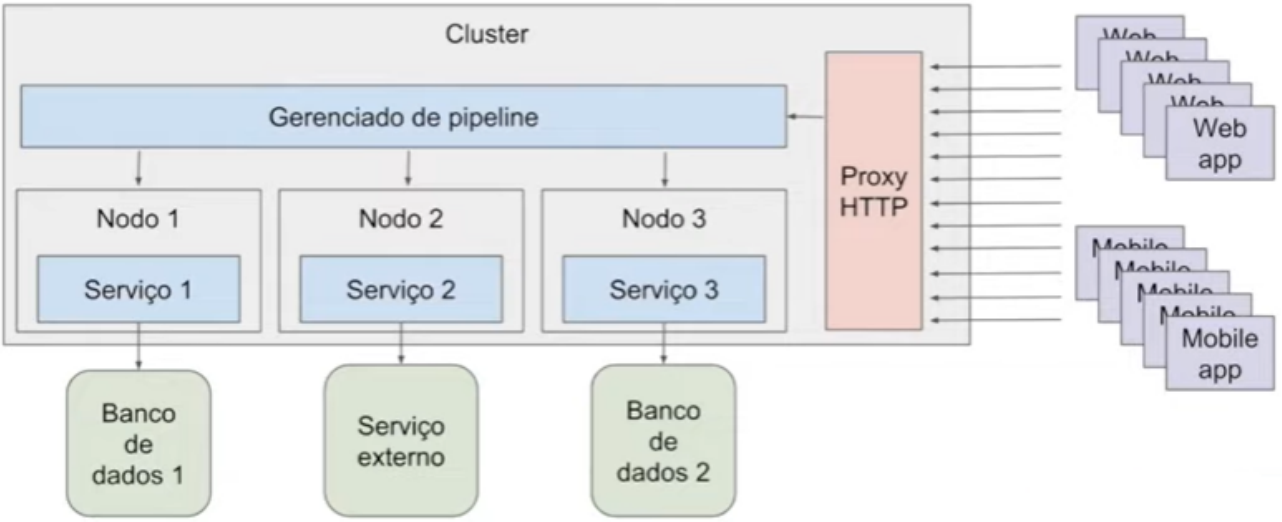
Vantagens:

- **Stack dinâmica;**
- **Simples escalabilidade;**
- **Desacoplamento.**

Desvantagens:

- **Monitoramento complexo;**
- **Provisionamento complexo.**

Microsserviços #3



Vantagens:

- **Stack dinâmica;**
- **Simples escalabilidade;**
- **Desacoplamento;**
- **Menor complexidade.**

Desvantagens:

- **Provisionamento complexo;**
- **Plataforma inteira depende do gerenciador pipeline.**

Introdução a IoT (internet das coisas)

O fato das “coisas” estarem interconectadas tem um objetivo principal por trás: Coletar dados.

Desafios

- Privacidade e segurança;
- Quantidade de dispositivos conectados;
- Capacidade de armazenar uma enorme quantidade de informações;
- Gerar valor a partir dos dados coletados.

Arquitetura

Conexão das coisas: Realizada desde protótipos até soluções industriais (hardware), como por exemplo a Alexa (home assistant) ou smartwatches. Principais características a se levar em consideração na escolha dos equipamentos: **Baixo consumo de energia, limite da rede de dados, resiliência** (projetado para falta de recursos essenciais, como por exemplo fazer buffer para compensar a falta de rede), **segurança** (principalmente para uso industrial), **customização, baixo custo.**

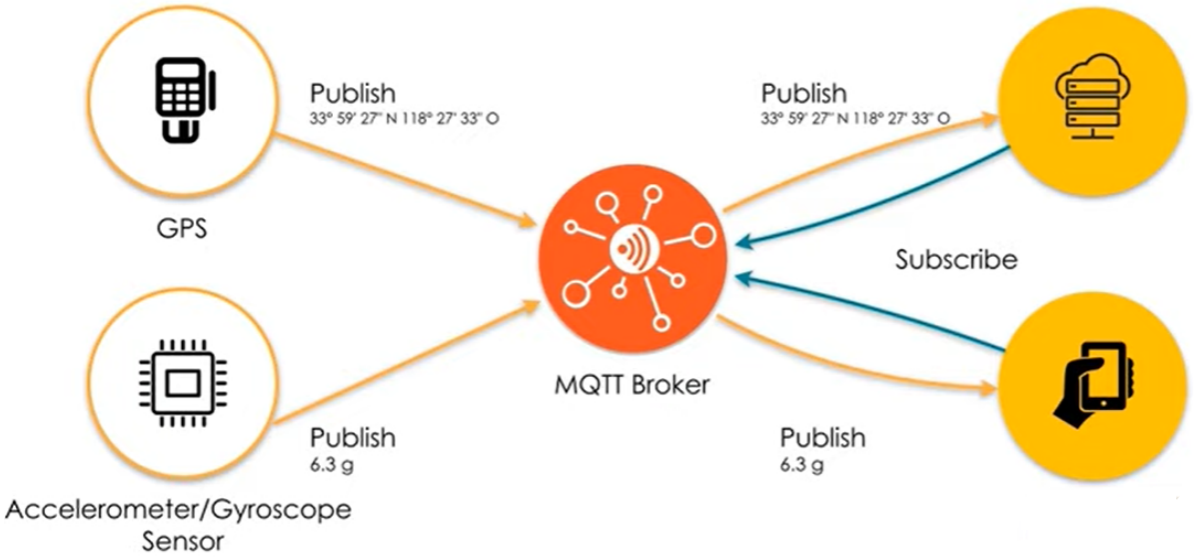
Protocolo de comunicação: MQTT. Esse é o protocolo mais utilizado para IoT com o intuito de padronizar a comunicação entre esses dispositivos. Ele foi criado pela IBM para conectar sensores de pipeline de petróleo a satélites. É um protocolo de mensagens assíncronas (não aguarda resposta) M2M (machine to machine). Seu padrão é o OASIS, suportado pelas linguagens mais populares de programação.

Abaixo alguns exemplos de comunicação.

Cliente x Servidor convencional

Cliente x Servidor convencional

Modelo Publish/Subscribe



Modelo Publish/Subscribe

Modelo de comunicação do MQTT. **Ele separa o Client entre fornecedor e consumidor**. Ou seja, no caso do GPS, ele captura os pontos geográficos e publica para um sistema (software) especializado em receber a mensagem, no caso do exemplo acima o **MQTT Broker**. Esse sistema (**MQTT Broker**) entrega as mensagens para o Clients estão inscritos para receber aquele tipo de mensagem. Um servidor pode se inscrever no Broker e receber as informações que estão sendo geradas e entregar para o Client. **A principal vantagem desse modelo é que um sistema pode alimentar vários clients que tenham interesse em receber a mensagem.**

Publish

Para publicar a mensagem no Broker, é necessário ter o endereço dele (ex: pub mqtt://my-tracker.com/d2s1m4/gps/position{‘lat:-20,3333,’lon’:-40,2833’}). A publicação nesse caso está sendo feita em JSON.

Cloud

Os números de dispositivos conectados estão cada vez maiores e com a nuvem o potencial de escalabilidade é global. A respeito de **Cloud,** recomendo a leitura do material de **AWS Cloud.**

No contexto de IoT, é importante saber que o dado deve ser armazenado e consultado com segurança e os ambientes de Cloud são uma ótima opção. A arquitetura é uma escolha tomada após compreender o problema e os componentes.

Arquitetura de dados

Conceito de dados

Dados podem ser definidos como um conjunto de valores e ocorrências que descrevem algo, e com esses dados, temos informações e por consequência, conhecimento. Um modelo sustentável de dados possui as seguinte características:

- Estruturação;
- Durabilidade;
- Velocidade;
- Controle de Acesso;
- Isolamento.

Com esse modelo, é possível garantir que os dados terão valor e poderão ser bem utilizados.

O que são bancos de dados

Com a necessidade de um modelo sustentável, temos a necessidade de sistemas que gerenciam esses dados. Existem vários tipos de sistemas de bancos de dados e cada um possui uma maneira de armazenar esses dados. Esses sistemas possuem a capacidade de abstração de maneira que leve informação e conhecimento aos usuários. Esses sistemas são chamados de **SGDB's** (Sistemas Gerenciadores de Banco de Dados), e em nível de abstração, foram criadas linguagens e ferramentas que controlam o organismo. Os três principais pilares são a **Linguagem de Definição**, a **Linguagem de Manipulação** e **Dicionário de dados.**

Linguagem de Definição: linguagem que possibilita definir a estrutura dos dados;

Linguagem de Manipulação: possibilita recuperar ou alterar as informações;

Dicionário de dados: banco de dados que guarda toda essa estrutura dentro do próprio banco de dados.

Modelos de bancos

Modelo Flat: basicamente uma tabela que possibilita compreender os dados, similar a um excel;

Modelo Hierárquico: Informação dividida em forma de árvore hierárquica, como por exemplo o registro do windows, que é baseado em um modelo hierárquico. Possibilita velocidade na obtenção dos dados, porém é complicado de se manter e apresenta redundância de informação.

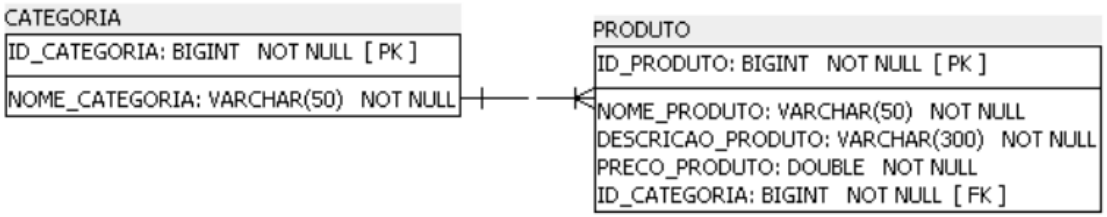
Modelo Relacional: Mais utilizado atualmente, sendo basicamente um conjunto de tabelas similar ao modelo flat que se relacionam entre elas obedecendo determinadas regras.

Existem outros modelos mais específicos, como por exemplo **Redes (Grapho)** que é similar ao modelo Hierárquico, **o modelo Orientado a Objetos, Objeto-Relacional** e **Big Data (será abordado mais adiante).**

Banco de dado relacional

É o modelo mais utilizado atualmente, de maneira direta ou indireta. Os sistemas que gerenciam esse tipo de banco são os **SGDBR** (ou RDBMS) e é composto por entidades que agrupam informações (**Tabelas**), que possuem linhas (**Tuplas**), definição das informações (**Colunas**), as chaves (**PK - chave primária e FK - chave estrangeira**).

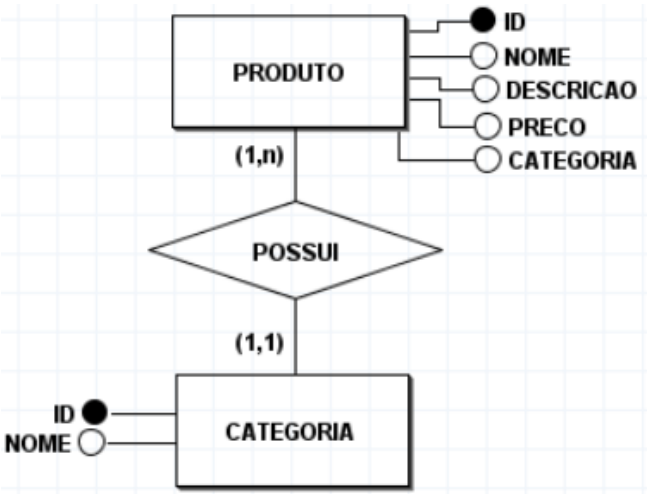
Exemplo:



Exemplo de arquitetura de um banco relacional

Modelagem

- **Modelo Conceitual** (MER - Modelo de entidade relacional). É uma abstração conceitual do modelo e que possui a relação dos dados.



Modelo Relacional (MER)

- **Modelo Lógico** (DER - Diagrama de entidade e relacionamento). Facilita a compreensão de como o modelo deverá se comportar com conceitos de normalização aplicados.

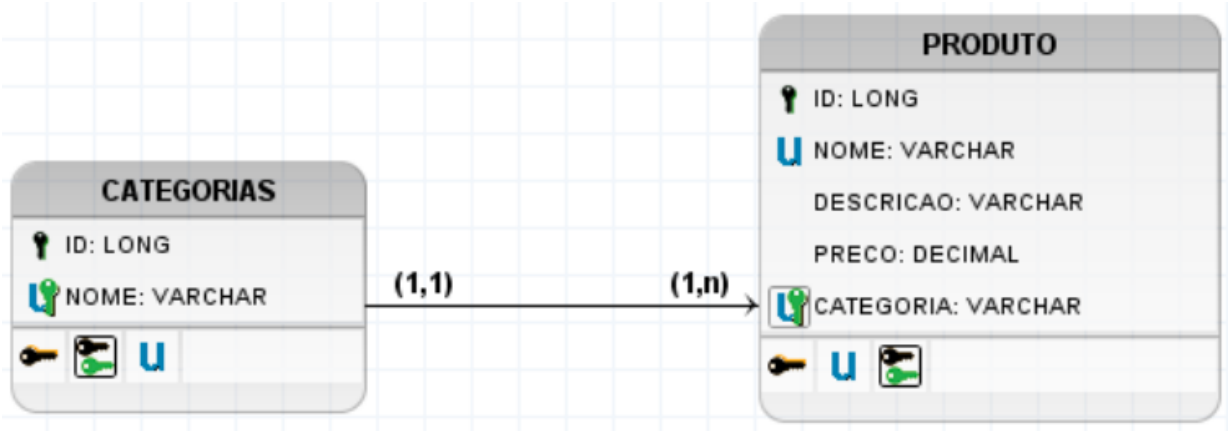


Diagrama de entidade e relacionamento (DER)

SGBDR - SQL

O significado de SGBDR é **Sistema Gerenciador de Banco de Dados Relacional.** Dentro desse sistema, é necessário ter uma linguagem que manipule-o. Essa linguagem **(SQL)** está estruturada em **DDL (Data Definition Language)** que permite definir dados**, DML(Data Manipulation Language)** podendo transformar dados e **DQL (Data Query Language)** que permite visualizar os dados.

Exemplo de **DQL:**

```
Create Table Cliente
(
Codigo number(10) Not Null Primary Key,
Nome varchar(50) Not Null,
Telefone varchar(15)
)
```

Exemplo de **DML:**

```
Insert into Cliente (Codigo,Nome,Telefone)
values (1,"Lorem ipsum","(88) 999 9999")

Delete from Cliente
Where Codigo = 1

Update Cliente
set Nome = "Lorem Ipsum"
Where Codigo = 1
```

Exemplo de **DQL:**

```
Select Codigo,
       Nome
  from Cliente
<Where> Codigo = 1
   <Group by> Profissao
   <Having> Count(1) > 0
<Order by> Nome, Codigo
```

Transactions

Ao realizar operações no banco, é necessário realizar um commit para salvar as alterações no banco ou um rollback para desfazer. Após a resolução aplicada e o commit realizado, as alterações serão efetivamente aplicadas ao banco.

ACID

As transações trabalham sobre o conceito ACID.

Atomicidade: Todas as operações executadas com sucesso (commit ou rollback);

Consistência: Unicidade de chaves, restrições de integridade lógica e as outras regras que foram modeladas na criação (**DDL**);

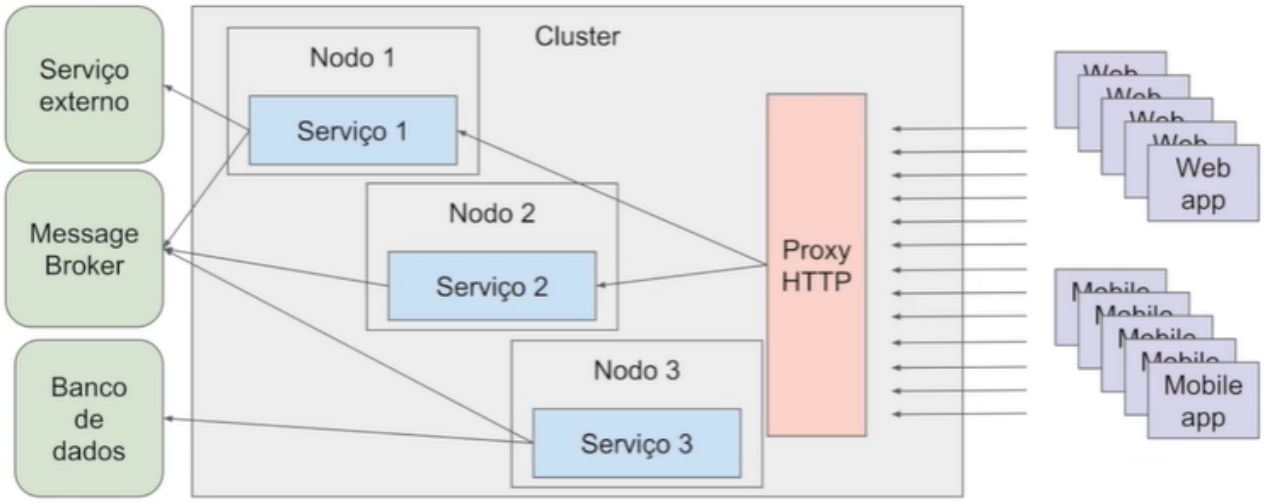
Isolamento: Várias transações podem acessar simultaneamente o mesmo registro e o banco de dados deve garantir o isolamento e garantindo que as ordem de manipulação sejam respeitadas;

Durabilidade: Depois do commit, mesmo com erros ou queda de energia, as alterações devem ser aplicadas.

Conceitos de integração de sistemas e mensageria

Arquitetura em mensageria

Esse tipo de arquitetura permite que existam serviços que se comunicam com aplicações externas e serviços que não se comunicam diretamente, mas que consomem mensagens de produtos que tiveram algum tipo de comunicação. Abaixo um exemplo:



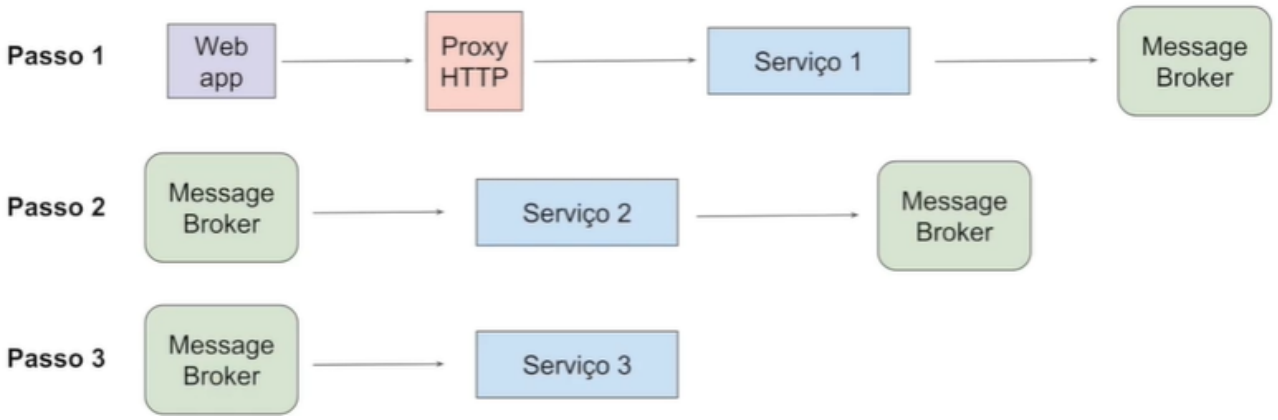
Imagine que o **Nodo 1** receba uma requisição HTTP e um usuário seja cadastrado. Ao cadastrar ele envia uma mensagem para o **Message Broker** e todos os serviços interessados em escutar a mensagem (broadcasting) o farão.

Prós: Desacoplamento, fácil plug & play, comunicação assíncrona, simples escalabilidade, broadcasting, permite event source.

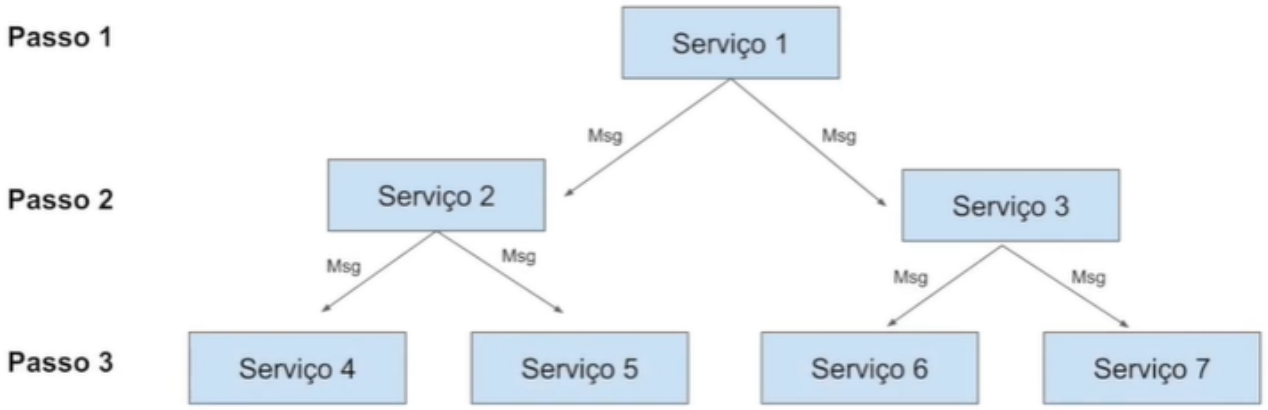
Contras: Single point of failure, difícil monitoramento.

Comunicação assíncrona entre serviços

A comunicação assíncrona pode ser definida como a transmissão de dados imediata em um fluxo estável. Abaixo temos um exemplo de comunicação assíncrona entre o serviço 1 e os serviços 2 e 3, que ocorre de maneira linear.



Outro exemplo seria uma comunicação assíncrona não linear:



A complexidade desses serviços, principalmente de monitoramento, só aumenta. Isso pode gerar uma inconsistência de dados e ocasionar na falha do serviço, já que um serviço apenas escuta, ele não sabe se outro serviço está escutando e o que será feito por ele.

Gerenciamento de erros em arquitetura assíncrona

- **Dead letter queue (filas de re-tentativas)**: Nos permite que tendo serviços de execução assíncronas, o serviço tenta interpretar a mensagem e em caso de erro, ele coloca em um fila para tentar interpretá-la novamente;

- **Monitoramento entre serviços**: É importante monitorar e rastrear os serviços e de onde vem as mensagens;

- **Rastreamento de fluxo**: Diretamente ligado ao tópico anterior, envolve entender o fluxo de mensagens.

Ex: Em um plano de streaming, é necessário adicionar o cartão por meio de uma requisição web;

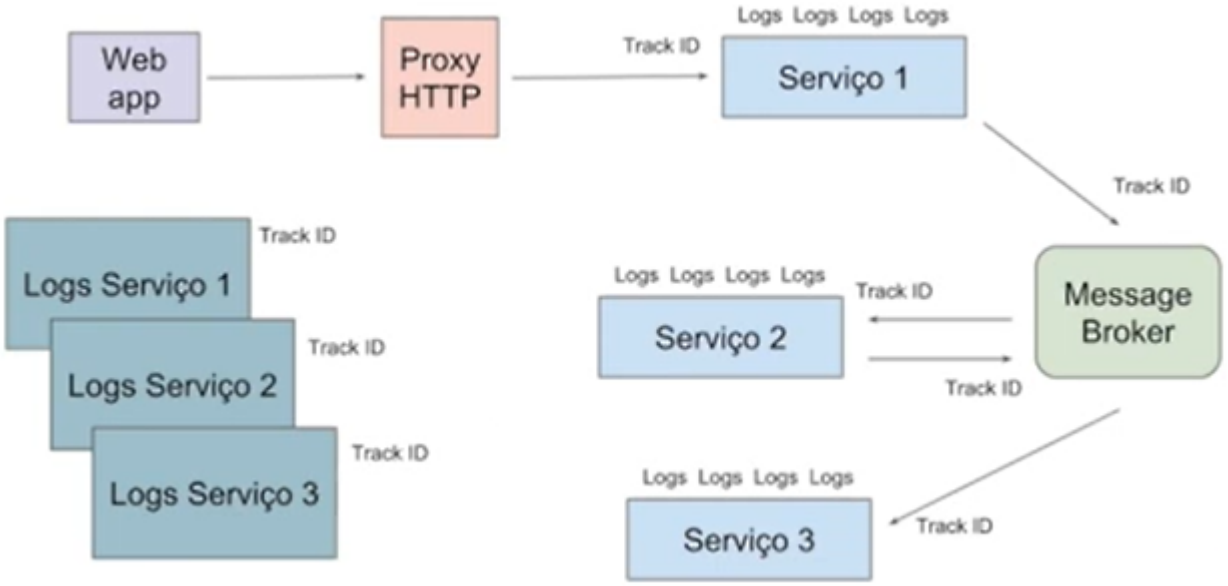
Após adicionado o cartão, o plano vai ser adquirido;

Se por algum motivo o serviço de aquisição de plano chegar antes de adicionar o cartão, o serviço irá disparar um erro. Nesse caso, a mensagem será adicionada a uma fila de re-tentativa.

Na próxima tentativa, será percebido que o cartão já foi persistido no banco e a aquisição de plano poderá ser executada.

Nesse caso, a requisição não é perdida e os eventos do passado não serão perdidos.

Ex de rastreamento de fluxo:



O Track ID é o facilitador que possibilita os serviços de se localizarem e podemos referenciar os dados por logs e esses ID’s. Dessa forma é possível compreender o caminho do sistema e direcionar o uso.

Arquitetura de dados não estruturados e business intelligence

Business intelligence em modelos de dados

Business Intelligence (BI) é um composto de ferramentas, infraestrutura, corpo técnico e dados que direcionam ações dentro de um ambiente baseado em dados de maneira mais clara. Esses dados podem vir de diversos locais, como por exemplo, operações e estudos.

Desenho de solução de BI

Data Warehouse

É uma estratégia de modelagem de dados para acomodar os dados de forma eficiente para soluções de BI. Existem dois conceitos que definem dados a nível de negócio dentro do data warehouse: **OLTP e OLAP**.

OLTP - Online Transaction Processing

É a representação das transações online, ponto a ponto, que acontecem no sistema, como por exemplo, atualizar o preço de um produto (foco operacional). Atende muitos usuários e tem muitos sistemas conectados, realizando várias operações simultaneamente.

OLAP - Online Analytical Processing

Modelo direcionado para analisar e consolidar dados colhidos no OLTP para relatórios e ferramentas estratégicas (foco estratégico). Grupo mais seleto de usuários, normalmente gestores e coordenadores.

Abaixo um comparativo em forma de tabela:

	OLAP	OLTP
Foco	Foco no nível estratégico da organização. Visa a análise empresarial e tomada de decisão.	Foco no nível operacional da organização. Visa a execução operacional do negócio.
Performance	Otimização para a leitura e geração de análises e relatórios gerenciais.	Alta velocidade na manipulação de dados operacionais, porém ineficiente para geração de análises gerenciais.
Estrutura dos dados	Os dados estão estruturados na modelagem dimensional. Os dados normalmente possuem alto nível de sumarização.	Os dados são normalmente estruturados em um modelo relacional normalizado, otimizado para a utilização transacional. Os dados possuem alto nível de detalhes.
Armazenamento	O armazenamento é feito em estruturas de *Data Warehouse* com otimização no desempenho em grandes volumes de dados.	O armazenamento é feito em sistemas convencionais de banco de dados através dos sistemas de informações da organização.
Abrangência	É utilizado pelos gestores e analistas para a tomada de decisão.	É utilizado por técnicos e analistas e engloba vários usuários da organização.
Frequência de atualização	A atualização das informações é feita no processo de carga dos dados. Frequência baixa, podendo ser diária, semanal, mensal ou anual (ou critério específico).	A atualização dos dados é feita no momento da transação. Frequência muito alta de atualizações.
Volatilidade	Dados históricos e não voláteis. Os dados não sofrem alterações, salvo necessidades específicas (por motivos de erros ou inconsistências de informações).	Dados voláteis, passíveis de modificação e exclusão.
Tipos de permissões nos dados	É permitido apenas a inserção e leitura. Sendo que para o usuário está apenas disponível a leitura.	Podem ser feito leitura, inserção, modificação e exclusão dos dados.

Comparativo OLTP e OLAP. Fonte:
https://canaltech.com.br/business-intelligence/o-que-significa-oltp-e-olap-na-pratica/

Big Data e dados não estruturados

Big data é um termo que descreve um grande volume de dados, podendo ser estruturados, semi-estruturados ou não estruturados.

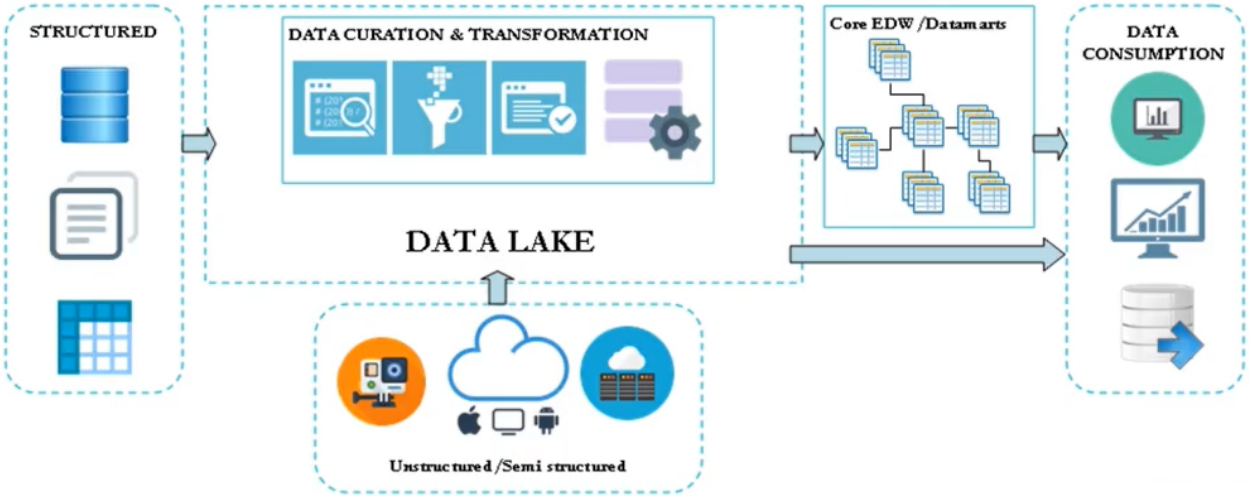
Dados estruturados: Tabela de banco de dados. Manipular os dados exige cumprimento de regras específicas;

Dados Semi-estruturados: XML, JSON, RDF, OWL. É possível alterar as informações sem ter uma regra propriamente dita;

Dados não estruturados: Dados que vem de fontes diversas sem nenhuma regra. Para isso, é necessário utilizar ferramentas que possibilitem a manipulação, como por exemplo o hadoop, apache spark e google bigquery.

Data Lake VS Big Data

Data lake é um tipo de big data, porém mais reservado e tratado(coorporativo). Precisa ter um sistema de **Data Accurate** para armazenar, pré-filtrar e exibir esses dados de maneira mais coerente.



Arquitetura de aplicação em nuvem

Escrevi um material um pouco mais detalhado a respeito de cloud, mais especificamente AWS. Acesse o **Material sobre AWS Cloud** para entender um pouco mais a respeito caso necessário.

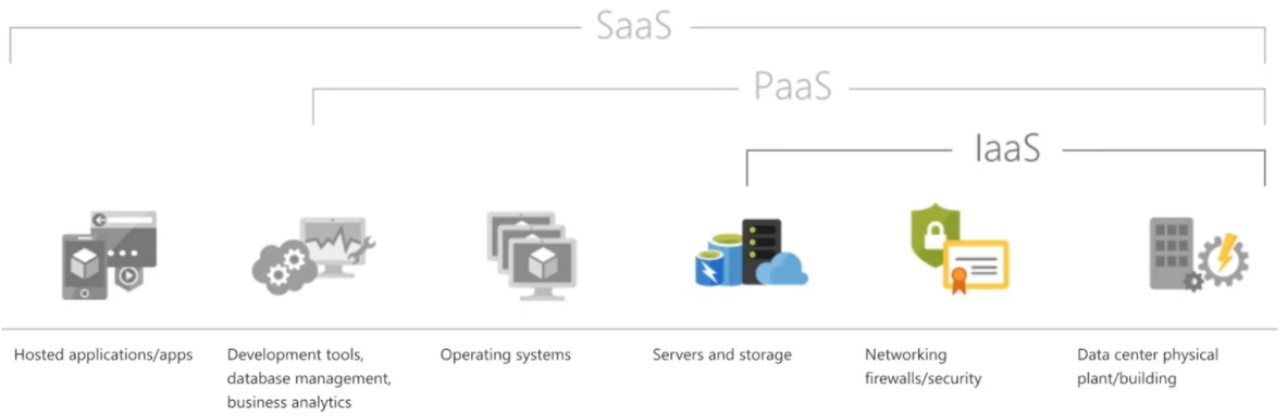
Cloud Computing

É um provedor ou serviço que faz o gerenciamento de hardware e software que pode ser pago "on demand".

IaaS - Infrastructure as a service: As máquinas permitem instalar o que for necessário para utilizá-las, pagando apenas pelo uso da infraestrutura como serviço;

PaaS - Plataform as a service: Com o avanço, agora o uso da acesso a uma plataforma, permitindo modificar a forma que as máquinas são utilizadas ou adicionar ou remover capacidade, e possibilitou a criação através de código com ferramentas como o Teraform. Tudo isso possibilita o rastreamento e customização do sistema.

BaaS - Backend as a service: Basicamente é uma ferramenta que provê um backend como serviço, como por exemplo o Firebase. Essa ferramenta lida com dados, autenticação e todo o backend.



ORganização visual SaaS, PaaS e IaaS - Fonte: https://www.microsoft.com/pt-br

Disponibilidade

IaaS - hardware e internet. A principal vantagem é não precisar se preocupar com o ambiente físico;

PaaS - auto scale on the go. Dada uma aplicação funcionando e em determinado momento ocorra um pico de uso que aumenta a demanda, o serviço de PaaS provisiona mais máquinas conforme a necessidade aumenta através do sistema automatizado pré configurado;

BaaS - não há backend service. Para desenvolvimento mobile ou frontend, é bem útil devido ao fato de não ter a necessidade de monitorar o backend da aplicação.

Atualmente (2021) para disponibilidade de serviços e orquestração de containers estão sendo mais utilizados o kubernetes (k8S). Para melhor disponibilidade, é recomendado o sistema de múltiplos nodos (3 serviços rodando kubernetes) devidamente balanceados para suprir aumento de demanda (Criando novas máquinas) e em caso de um serviço parar, o kubernetes consegue redistribuir de forma que a aplicação se mantenha disponível. Outro ponto importante é ficar atento ao load balancer para garantir o desempenho e equilíbrio das cargas e por consequência a disponibilidade do serviço.

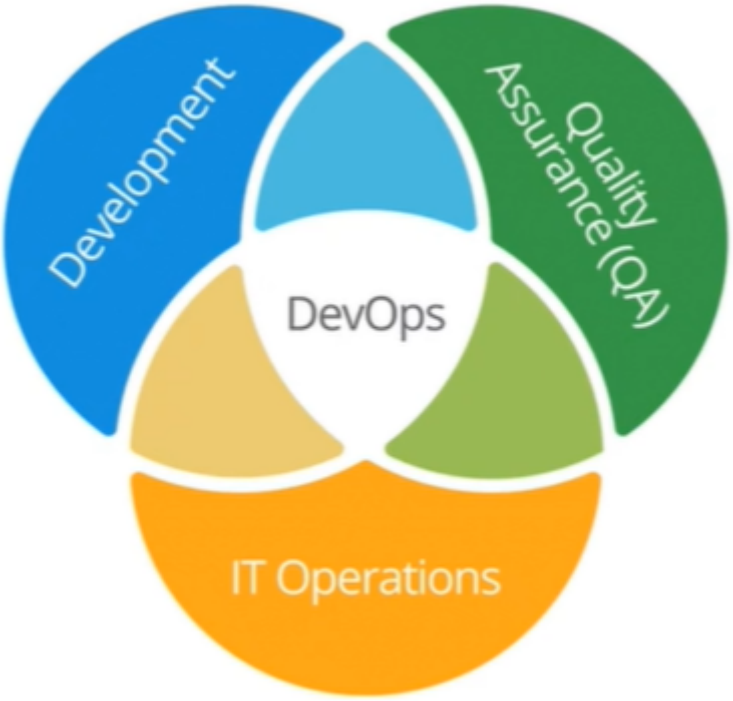
Serverless

Basicamente significa a ausência de servidor (nem na cloud computing), sem um serviço sendo executado (executado caso necessário o código) e sem down time, já que o serviço não está sendo executado. Algumas desvantagens são o custo, a quantidade de funções lambda (função executada ao utilizar o serverless) e o tamanho dessas funções, já que, no caso da AWS, por exemplo, o tempo limite de execução de uma lambda é 15 minutos.

Desenvolvimento e operação de software integrado

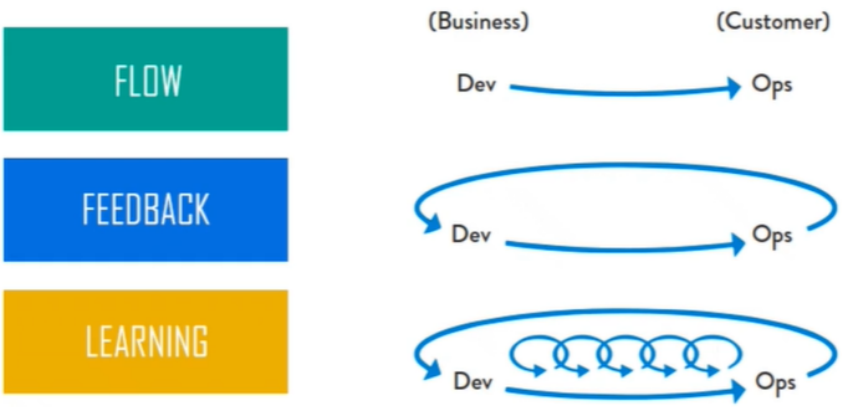
DevOps

O DevOps é um conjunto de práticas que proporciona integração e automatização de processos entre as equipes de desenvolvimento, operações e de apoio para a produção eficiente de software otimizando o tempo. DevOps se baseia em colaboração entre as equipes, mudando a mentalidade e a forma de agir do time.



Um exemplo de framework é o CALMS. Ele é dividido entre **cultura, automação, lean, measurement e sharing.**

Os fluxos de entrega devem ser entregues baseados nos três caminhos:



Flow: A otimização do fluxo tem como objetivo eliminar os desperdícios, gargalos do processo, transferência de responsabilidades e tempos de espera. Esse caminho é trilhado entre a demanda e a entrega. A chave para este caminho é a aplicação de metodologias ágeis e a automatização dos processos do desenvolvimento à release, como a integração contínua ou entrega contínua;

Feedback: Ciclos rápidos de feedbacks visam resolver problemas da maneira mais rápida possível, testando tudo, alertando falhas e considerando todas as métricas coletadas no ambiente produtivo sobre o valor entregue. O monitoramento é a chave que ajuda a gerar informações relevantes constantemente. Com feedbacks rápidos, tanto a falha quanto a solução vem de maneira rápida e o sistema logo é normalizado.

Learning: O aprendizado contínuo visa gerar conhecimento durante o desenvolvimento do software. Hipóteses (brainstorming) são melhores do que uma certeza imediata. Este caminho é fruto do processo científico e produz segurança psicológica. A chave é o trabalho dinâmico, com times realizando experimentos em seu trabalho diário para gerar novas melhorias. Eliminar a cultura da culpa e aumentar a colaboração e o compartilhamento de conhecimento.

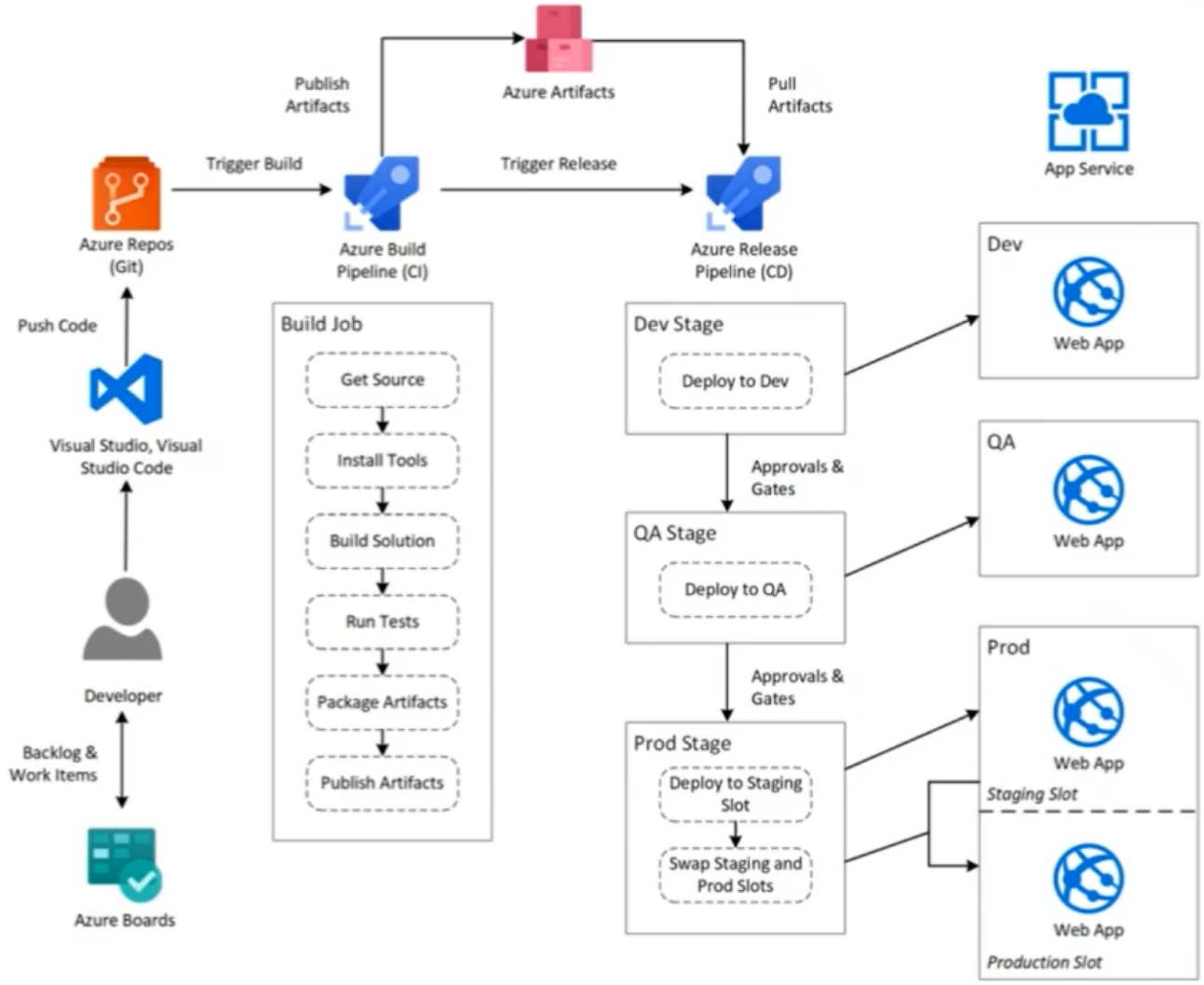
Entregando o software - Ferramentas

- **Plan:** Jira, draw.io, confluence, microsoft teams. balsamiq, roadmunk ...
- **Code:** Visual Studio, Git, Eclipse, Pycharm, vs code, notepad ...
- **Build:** Containerd, docker, nuget, npm, cri-o, .NET CLI, MSBuild ...
- **Test:** Unit.net, postman, sonarqube, codacy, loader.io, meter, selenium, code climate ...
- **Release:** Disponibiliza o software para o cliente (ferramentas atreladas ao deploy);
- **Deploy:** Jenkins, GitLab CI, Travis CI, AppVeyor, circleci, Azure Pipelines ...
- **Operate:** Kubernetes, rancher, microsoft Azure, puppet, terraform, chef, open shift, AWS ...
- **Monitor:** Datadog, prometheus, appmetrics, rollbar, zabbix, new relic, pushover, seq, monitis ...

Continuous Integration (CI/CD)

Quando falamos de integração contínua (CI), o limite dela é a geração do artefato, ela não faz implementação na produção. Após o pipeline de integração contínua, a implementação contínua é feita de maneira automática para o ambiente desejado. A etapa de entrega contínua (CD) é parecida com a de implementação contínua, porém necessita de alguém para aprová-la.

Pipelines



Continuous Inspection

Inspeção contínua da qualidade do código serve para resolver problemas de análise de qualidade como a complexidade ciclomática (quantidade de caminhos que seu código pode seguir), complexidade cognitiva, vulnerabilidade/code smell, padronização e estilo. Com isso a ferramenta (sonarqube, por exemplo) bloqueia de acordo com a configuração e o desenvolvedor precisa melhorar o código.

Outras funcionalidades das ferramentas são a relações de débito técnico, que colaboram com a gestão dos processos e indicadores de qualidade, cobertura de testes e métricas.

Após as validações, o pull-request é ou não autorizado para liberar o merge.

Referências

1. www.w3c.br
2. https://pt.wikipedia.org/wiki/Camada_de_aplica%C3%A7%C3%A3o
3. https://pt.wikipedia.org/wiki/Chamada_de_procedimento_remoto
4. https://pt.wikipedia.org/wiki/Hypertext_Transfer_Protocol
5. https://www.soapui.org/
6. https://www.ibm.com/docs/en/ibm-mq/9.1?topic=components-dead-letter-queues
7. https://mqtt.org/
8. https://developer.ibm.com/br/articles/iot-mqtt-why-good-for-iot/
9. https://github.com/dsperax/pdf-for-download/blob/main/AWS%20Cloud%20(PT-BR)/AWS%20Cloud%20(PT-BR).pdf
10. https://pt.wikipedia.org/wiki/Broadcasting_(rede_de_computadores)
11. https://canaltech.com.br/business-intelligence/o-que-significa-oltp-e-olap-na-pratica/
12. https://github.com/dsperax/pdf-for-download/blob/main/AWS%20Cloud%20(PT-BR)/AWS%20Cloud%20(PT-BR).pdf
13. https://www.microsoft.com/pt-br
14. https://www.atlassian.com/devops/frameworks/calms-framework
15. Manual De DevOps: Como obter agilidade, confiabilidade e segurança em organizações tecnológicas - Livro por Patrick Debois