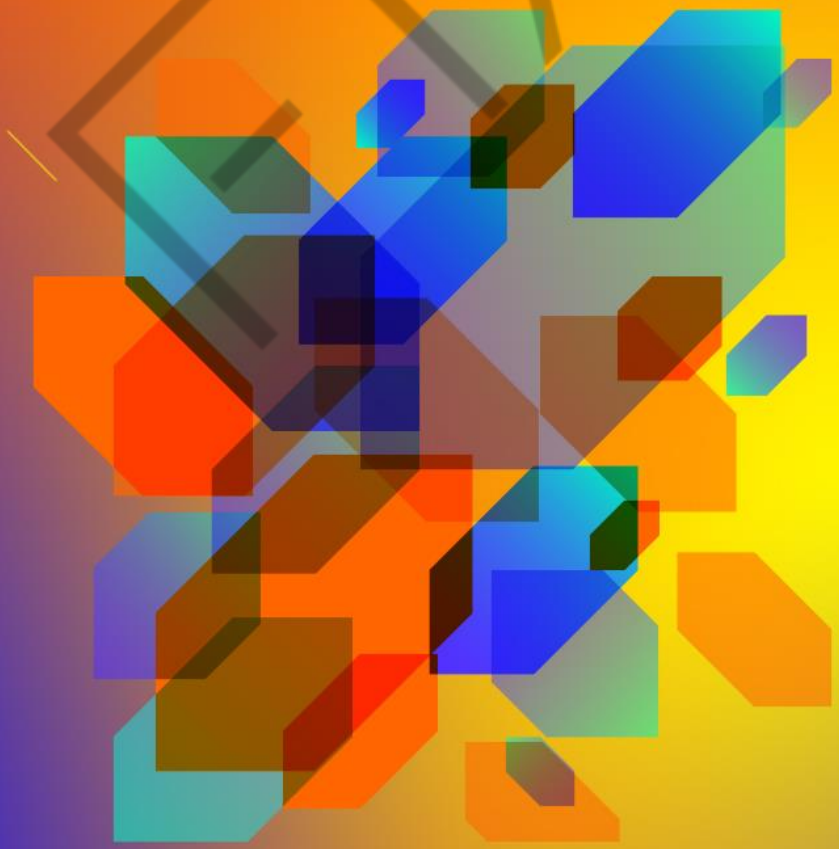
FIAP ON

SERVICES ARCHITECTURE,
API E MOBILE ARCHITECTURE

KAFKA E ESB

CHRISTIANE DE PAULA REIS



3

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE CÓDIGOS-FONTE

SUMÁRIO

3 KAFKA E ESB

Neste capítulo, você entenderá o que é e como funciona o ESB e o Kafka na Arquitetura de Microsserviços (*MSA - Microservice Architecture*). Além disso, incrementará com as novas tecnologias o desenvolvimento do projeto de microsserviços que você preparou para criar no capítulo anterior.

3.1 Qual a melhor forma de integrar Microsserviços?

A comunicação entre processos ou serviços e microsserviços é um aspecto muito importante da MSA.

Um microsserviço é desenvolvido com o objetivo de ser autônomo e de estar disponível para o consumidor cliente, mesmo que haja falha de integridade entre outros serviços que fazem parte do todo ou que haja serviços inativos. Sendo assim, quanto menos comunicações houver entre os microsserviços, melhor será para o funcionamento geral da aplicação. Porém, existem situações que demandam a necessidade de integrar os microsserviços de alguma forma. Para esses casos, evite depender de comunicação síncrona (solicitação/resposta), pois você deseja ter uma arquitetura resiliente quando alguns microsserviços falharem. Então, a dica de ouro aqui é trabalhar com a propagação de dados de forma assíncrona.

Vamos ver, em seguida, quais são as principais opções para conectar a gama de microsserviços que compõem a MSA e para trafegar dados/informações entre sistemas.

3.2 ESB (Enterprise Service Bus)

O **ESB** é um **barramento** para a execução de serviços através de *driver* de evento e padrões baseados em mensagens, chamadas de BUS, que, além de funcionalidades básicas, como autenticação, autorização e *logging*, dispõe de diversas funcionalidades importantes, como:

Função	Recursos
Invocação de Serviços	Comunicação através de padrões de troca de mensagens (ou MEP – *Message Exchange Pattern*). Suporta chamadas síncronas e assíncronas de serviços e, algumas vezes, *callback*. Um serviço pode ser mapeado em outro serviço.
Conversão de Protocolos	Intermedeia a comunicação entre as aplicações consumidoras e o provedor de serviços em barramentos que possuem diferentes protocolos de comunicação.
Localização de Serviços	Através do UDDI (*Universal Description, Discovery and Integration*) é possível verificar a localização e o registro dos *Web Services.*
Gerenciamento dos Serviços	Monitora, audita, mantém e reconfigura os serviços em execução no ESB.
Integração de Aplicações	Troca de informações entre diferentes aplicações, desenvolvidas em linguagem ou plataforma diferentes, podendo ser locais ou remotas.
Monitoramento de Atividades de Negócio	Comunicação com ferramentas de monitoramento de negócios (como, por exemplo, BAM – *Business Activity Monitoring*).
Segurança	Garantir o uso de políticas de segurança em conjunto com pontos de garantia de políticas, SSL (*Secure Sockets Layer*) e SAML8 (*Security Assertion Markup Language*).
Roteamento e Mediação de Mensagens	Com a criação de *Virtual Services* (serviços virtuais), que têm a finalidade de prover *endpoint* (porta de entrada) para o recebimento e o endereçamento de mensagens, é possível encaminhar requisições para serviços e enviar respostas de volta para clientes.
Tradução de Mensagens	Prepara o dado para ser trafegado entre sistemas/serviços. Traduz ou transforma mensagens enviadas por consumidores de serviços para um padrão específico esperado pelos provedores de serviço.
Garantia de Entrega	Em casos de indisponibilidade, o ESB pode persistir a mensagem em um repositório interno e tentar enviá-la novamente através do *Idempotent Retry Pattern.*

Quadro 3.1 – Funcionalidades do ESB.
Fonte: Elaborado pela autora (2019)

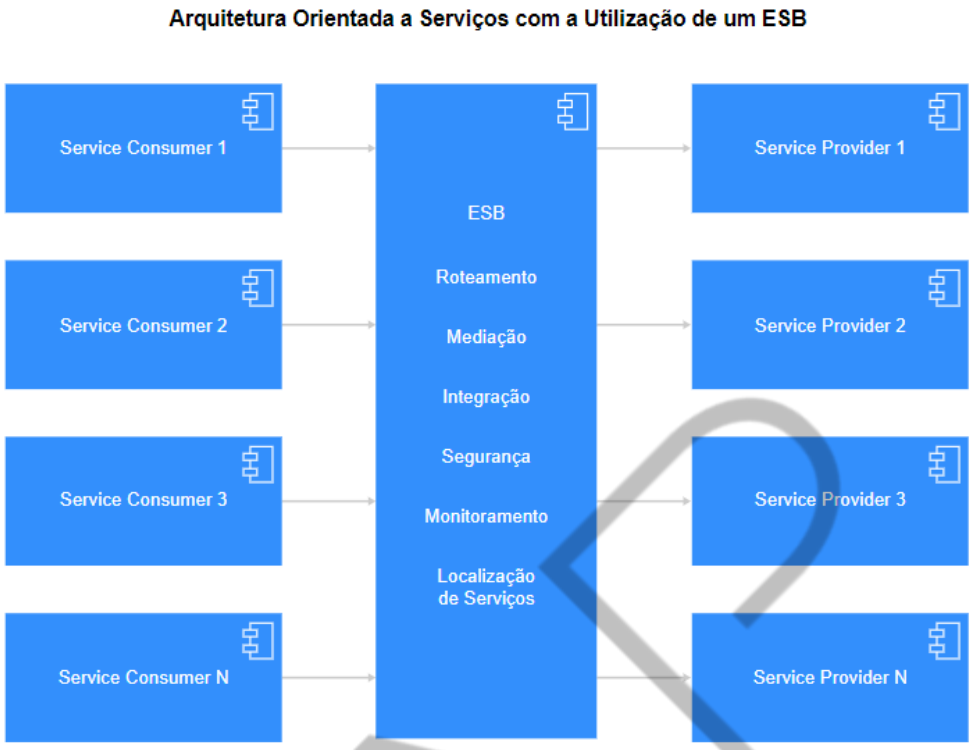


Figura 3.1 – Responsabilidades do ESB em SOA.
Fonte: Google Imagens (2019)

O barramento ESB traz um conjunto de *adapters* que possibilitam conexão a diversos protocolos e tecnologias, podendo ser inseridos também novos *adapters* de acordo com a necessidade.

3.3 Kafka

O **Apache Kafka** carrega as mesmas funcionalidades do ESB, mas incrementou características importantes para suportar a moderna arquitetura de nuvem, como a Arquitetura sem Servidor (*Serverless*) e a Arquitetura de Microsserviços (*MSA – Microservice Arquitecture*).

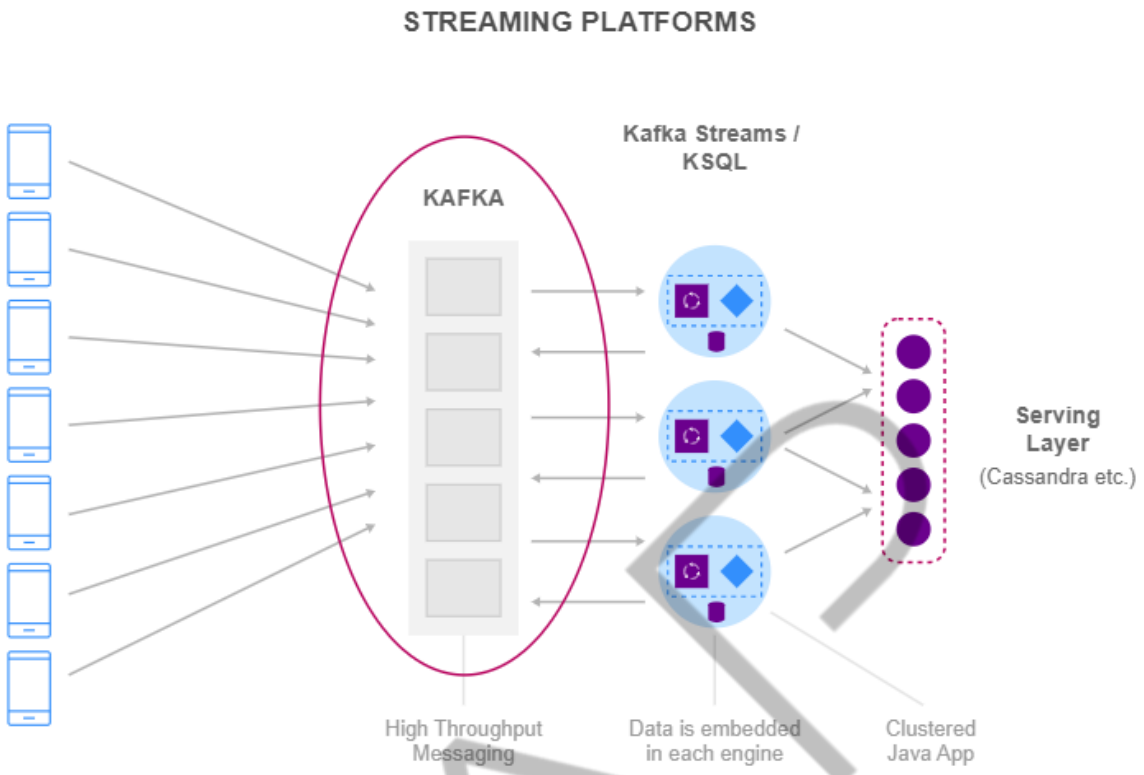


Figura 3.2 – Kafka: Streaming Plataforms.
Fonte: Stopford (2018)

O **Kafka** é uma **plataforma de mensagens** e ***streaming* distribuída** de código aberto criada pelo LinkedIn em 2011 para lidar com **alto *throughput*** (taxa de transferência), transmissão de **baixa latência** (capacidade de vazão de dados), **tolerância a falhas** e **escalabilidade**.

Características	Recursos
Alto *throughput*	Publicar e assinar fluxos de registros (semelhante a uma fila de mensagens ou sistema corporativo de mensagens).
Baixa latência	Processar fluxos de registros em tempo real.
Tolerância a falhas	Tolerância a falhas ao armazenar os fluxos de registros para garantir o funcionamento do sistema.
Escalabilidade	Clusterização e escalabilidade horizontal.

Quadro 3.2 – Características e recursos do Kafka
Fonte: Elaborado pela autora (2019)

O **Kafka** nasceu com a proposta de ser um ***middleware*** robusto que permite a criação de uma **arquitetura baseada em eventos**, na qual cada nova funcionalidade pode ser plugada, desplugada e atualizada sem a necessidade de *deploy* em todos os serviços. Além disso, conecta microsserviços entre si ou com sistemas externos de forma assíncrona a fim de aumentar o **desempenho**, a **confiabilidade** e a **escalabilidade**.

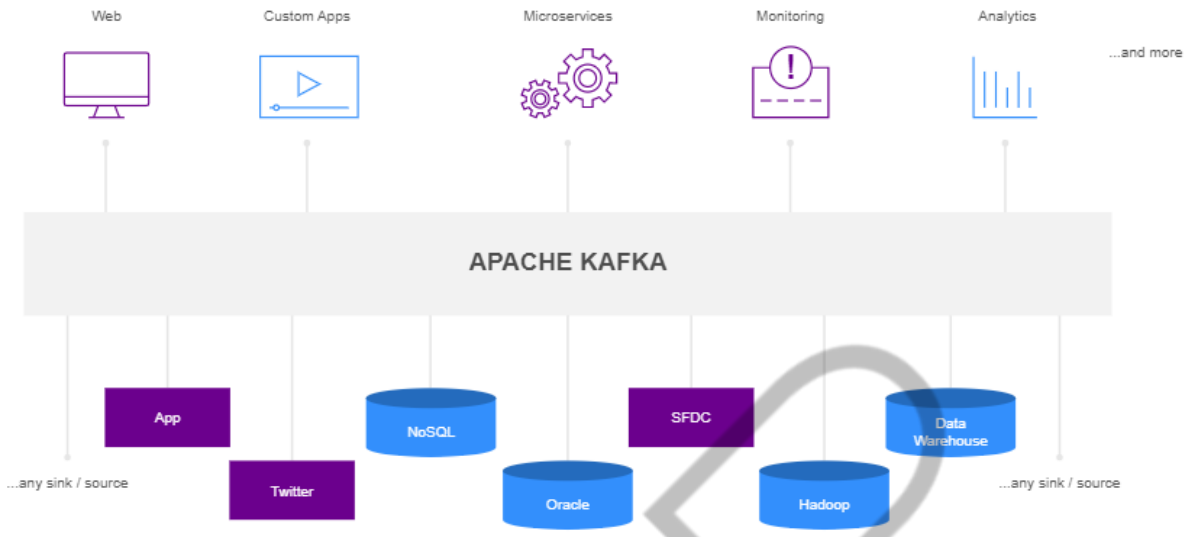


Figura 3.3 – *Apache Kafka architecture*
Fonte: Google Imagens (2019)

Fornece um rico conjunto de APIs e *clients* para diversas linguagens:

- **API *Consumer*, *Producer* e *Admin***: API principal do Kafka, usada para troca (enviar / receber) de mensagens.
- **API de fluxos**: API de processamento de fluxo de alto nível para consumir, transformar e produzir eventos entre tópicos facilmente.
- **API de conexão:** estrutura que permite integrações reutilizáveis ou padrão para eventos de fluxo dentro e fora de sistemas externos, como os bancos de dados.
- **KSQL**: interface para processar e associar eventos por meio de tópicos usando uma sintaxe semelhante a SQL (*Structured Query Language* ou Linguagem de Consulta Estruturada).

Como exemplo do mundo real, podemos pensar na situação de que novas equipes e departamentos são introduzidos com frequência na sua empresa e precisam operar de forma independente, livre das ligações síncronas que ligam o *frontend*. Como é possível conectar novos sistemas ao fluxo de eventos em tempo real?

APIs REST HTTP apresentam problemas típicos de enfileiramento, enquanto as filas de mensagens antigas, como o RabbitMQ (implementação do AMQP – *Advanced Message Queuing Protocol*), vêm com problemas operacionais e de

dimensionamento. A resposta, então, é: você deve trabalhar com arquiteturas que fazem uso de Kafka para resolver esses problemas.

Veja algumas das grandes empresas que necessitam processar um alto volume de dados e, por isso, fazem uso da plataforma Apache Kafka. Por exemplo, a Netflix informa que processa mais de 2 trilhões de mensagens por dia usando o Kafka.

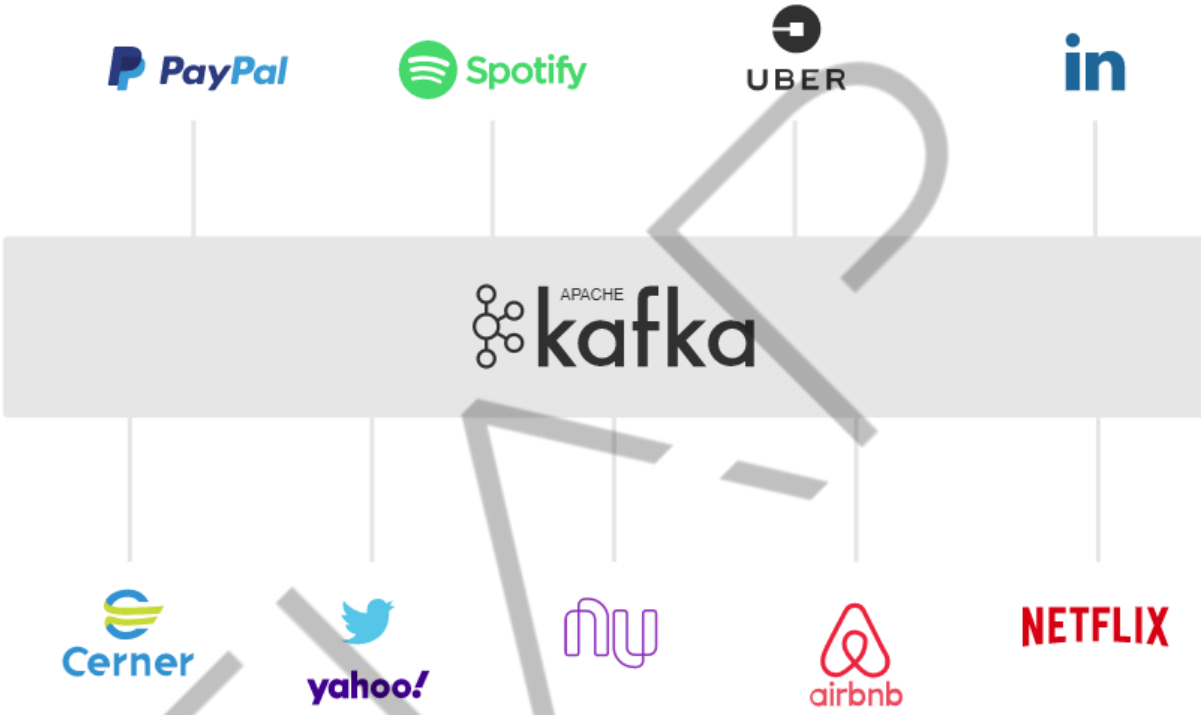


Figura 3.4 – Empresas que utilizam a plataforma Apache Kafka
Fonte: Google Imagens (2019)

Atualmente no mercado uma das principais opções para trafegar dados/informações na forma de eventos entre sistemas com escalabilidade e organização é o Apache Kafka.

3.3.1 Arquitetura do Kafka

Kafka é **agnóstica** e foi projetada no **padrão de arquitetura publicar/assinar** mensagens (*publish/subscribe* ou *pub/sub messaging*) com o objetivo de **desacoplar** partes de um sistema e permitir a troca de informações (entrega e consumo) de forma totalmente **assíncrona** e **independente**. (AMAZON, 2019).

As mensagens são organizadas em tópicos (***topics***), os tópicos são divididos em partições (***partition***), e as partições são replicadas (***replication***) entre os nós

denominados ***brokers***. ***Streams*** permitem que o envio de dados entre fontes distintas seja realizado sem dependências entre os produtores (***producer***) e os consumidores (***consumer***).

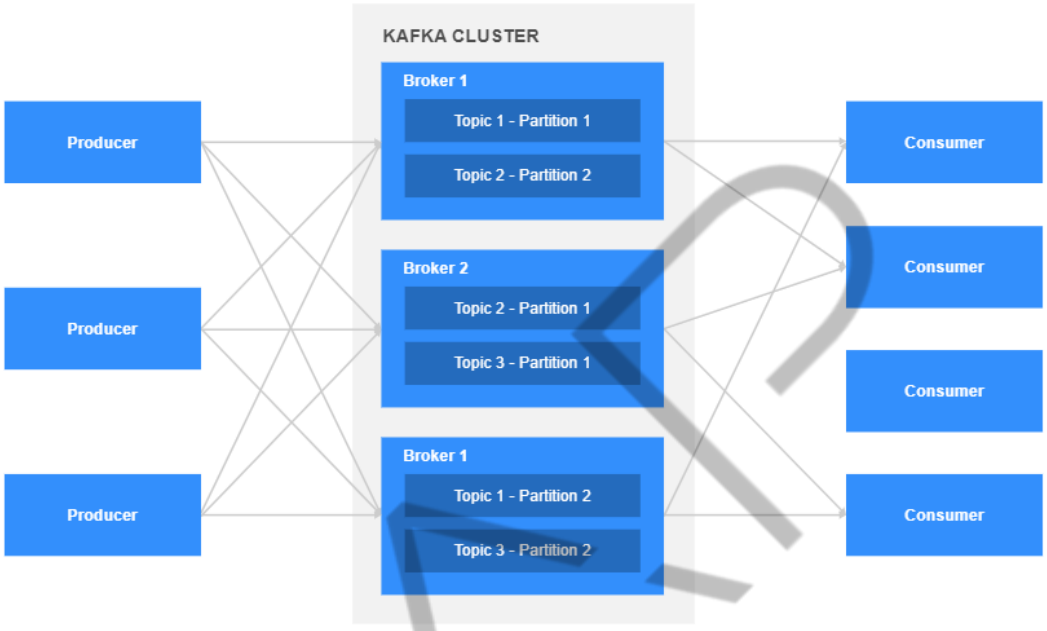


Figura 3.5 – Kafka Cluster
Fonte: Google Imagens (2019)

3.3.2 Instalando e configurando o Kafka

1. Instalar Kafka como serviço na sua máquina local usando o Quickstart.

Última Versão Quickstart: 2.3.0 (já inclui o ZooKeeper)

Os próximos comandos devem ser executados no terminal / *prompt* de comando.

2. Extraia o Kafka com o comando:

```
tar -xzf kafka_2.12-2.3.0.tgz
```

Código-fonte 3.1 – Comando para extrair o Kafka.
Fonte: Elaborado pela autora (2019)

3. Acesse a pasta *bin* do Kafka com o comando:

```
cd kafka_2.12-2.3.0/bin
```

Código-fonte 3.2 – Comando para acessar a pasta *bin* do Kafka.
Fonte: Elaborado pela autora (2019)

4. Inicie o servidor Zookeeper com o comando:

```
zookeeper-server-start.sh config/zookeeper.properties
```

Código-fonte 3.3 – Comando para iniciar o servidor Zookeeper.
Fonte: Elaborado pela autora (2019)

5. Inicie o servidor Kafka com o comando:

```
kafka-server-start.sh config/server.properties
```

Código-fonte 3.4 – Comando para iniciar o servidor Kafka.
Fonte: Elaborado pela autora (2019)

As propriedades contidas nos arquivos *zookeeper.properties* e *server.properties,* que ficam dentro da pasta *config,* são algumas configurações-padrão disponibilizadas ao baixar o Kafka, mas podem ser alteradas de acordo com as necessidades do projeto.

Por padrão, a porta usada para o Kafka é 9092 e para o Zookeeper é 2183.

3.4 ESB vs. Kafka

Ambas as tecnologias, ESB e Kafka, surgiram com o objetivo de integrar aplicações para viabilizar o processo de negócio da empresa. Diversos fatores são considerados para que haja integração:

- ✓ **Plataformas e linguagens de programação**: como Cobol, Java, .NET, Go ou Python.
- ✓ **Tecnologias**: i) formatos de dados, como JSON, XML, Apache Avro ou Protocol Buffers; ii) padrões, como SOAP, REST, JMS ou MQTT; iii) estruturas abertas, como Nginx ou Kubernetes; iv) interfaces proprietárias, como EDIFACT ou SAP BAPI.
- ✓ **Paradigmas de comunicação**: como solicitação-resposta, processamento em lote, processamento em tempo real, ativação e desativação, consultas contínuas e retrocesso, publicação de assinatura.
- ✓ **Arquiteturas de aplicativos**: como monolítica, *Client Server*, Orientada a Serviços (*SOA – Service-Oriented Architecture*), Orientada a Microsserviços (*MSA – Microservice Arquitecture*) ou sem Servidor (*Serverless*).

O uso de barramento **ESB**, apesar de disponibilizar uma variedade de conectores e realizar um ótimo processamento de transações complexas, envolve alguns pontos de atenção, como:

I. escalabilidade complexa da infraestrutura;

II. ii) poucos recursos próprios para Governança, Monitoramento e Monetização;

III. iii) suporte limitado para Segurança, exemplo, Oauth 2.0, OpenID, JWT etc.;

IV. iv) demanda a aquisição de muitos produtos em conjunto (como BAM, BPEL, Service registry etc.) para alcançar um bom estado de funcionamento. Além disso, o ESB é tido como pesado e sem agilidade para a realidade do desenvolvimento em cloud, mas pode ser altamente recomendado como padrão de integração corporativa.

Por sua vez, o grande diferencial da plataforma Apache **Kafka** é que ela combina mensagens, armazenamento e processamento de eventos em uma única plataforma. Além disso, permite que diferentes consumidores leiam dados independentemente uns dos outros e em diferentes velocidades. É indicada também para uso em demandas de escabilidade extremamente alta. Todo o conjunto de características da plataforma Kafka é o que a destaca das soluções existentes de integração e mensagens e promove seu sucesso em empresas de grande porte.

Desta forma, ambas as tecnologias podem ser usadas em âmbitos e funcionalidades diferentes, até de forma complementar, mas sempre considerando a tríade propósito x escopo x orçamento do seu projeto.

3.5 Exercício

Implementar dois microsserviços com node.js através do framework Express, usando o Springboot e Maven.

1. Utilizar Kafka para conversação entre os microsserviços.
2. Implementar *Service Discovery* e *Loadbalance.*
3. Armazenar em container Docker.

4. Fazer *deploy* na AWS ou IBM Cloud.

Para este exercício, além das ferramentas que já instalamos e configuramos anteriormente, é necessário ter instalado em seu computador:

- ✓ Docker CE (Community Edition) – Ubuntu.
- ✓ Docker Compose para Windows, Mac OS X ou Linux/Unix.
- ✓ Git para Windows, Mac OS X ou Linux/Unix.

DICA: Não instalar Docker e Docker Compose através de instaladores de pacotes, pois podem conter versões desatualizadas.

CONCLUSÃO

O que você viu neste capítulo foram diversas tecnologias que são essenciais para o funcionamento geral da Arquitetura de Microsserviços. Siga em frente, pois vamos ampliar ainda mais seu potencial de desenvolvedor web.

REFERÊNCIAS

AMAZON. **What is Pub/Sub Messaging?** Disponível em: <https://aws.amazon.com/pt/pubsub-messaging/. Acesso em: 10 out.. 2019.

STOPFORD, Ben. **Designing event driven systems. Concepts and patterns for streaming services with Apache Kafka.** O'Reilly Media, 2018.

GLOSSÁRIO

API – Application Programming Interface	API é uma estrutura de códigos de programação padronizados que permitem a conexão entre sistemas.