

Projeto Proposto :

Implementação do gerenciamento dos pedidos

Objetivo

Criar um produto que vai gerenciar e calcular os produtos do pedido, sendo que os pedidos serão disponibilizados por outro produto externo A, e após nossa gestão dos pedidos, devemos disponibilizar a outro produto externo B.

Descrição

Precisamos disponibilizar uma integração com o produto externo A para o recebimento dos pedidos, para realizar em nosso produto a gestão e calculo do valor total dos produtos, somando o valor de cada produto dentro do pedido. E será necessario disponibilizar uma consulta dos pedidos e produtos, junto com seu status para o produto externo B receber os pedidos calculados.

Considerar que podemos receber 150 mil a 200 mil pedidos por dia.

Pontos extras:

- Verificação de duplicação de pedidos.
- Com a volumetria de pedidos é alta como pode garantir a disponibilidade do serviço.
- Como podemos ter maior consistencia dos dados e garantir a concorrência.
- Verificar se com essa volumetria pode engargalar o banco escolhido.

Soluções propostas

- Implementação dos requisitos mencionados.
 - Considerar que para chegar na solução pode elaborar da forma que considerar mais adequada, os pontos listados são apenas um direcionamento.
- Utilizar os protocolos de comunicação que considerar adequados.
- Considerar boas praticas.
- Considerar um novo desenho do modelo final que desenvolveu, para saber quais adequações considerou serem feitas para atingir o objetivo.

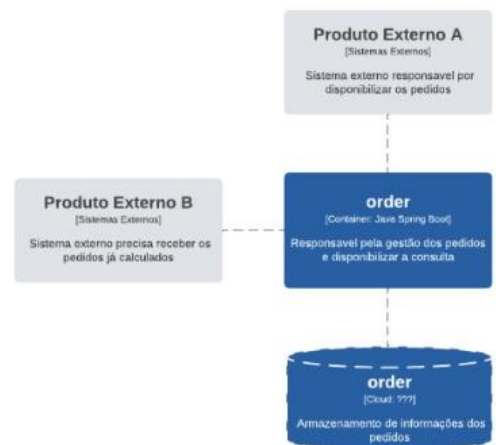
Tecnologias:

- O sistema deve ser construído usando Java com Spring Boot.
- Pode escolher qual banco de dados utilizar, o que considerar mais adequado.
- Pode utilizar as bibliotecas que forem necessario para ter a devida qualidade e atingir tornar sustentavel o produto.

Entregáveis

- Considerar uma entrega rapida, pode ser apresentado o codigo no momento da entrevista compartilhando a tela.
- Deixar rodando na maquina e exemplos das integrações externas A e B.
- Desenho final da solução que está propondo.

Desenho da Arquitetura - Exemplo
(Pode aplicar mudanças conforme considerar mais adequado)



Ao realizar o teste, tenha atenção nos seguintes pontos:

- Leia e faça com calma o teste, entenda exatamente o que ele está solicitando, a duração gira em torno de 5h a 7h.
- Aplique boas pratica
 - Organização e estrutura do código
 - Princípios de orientação objetos
 - Design patterns
 - Utilização recursos mais novos do Java
 - Perfomance e otimização

- Código limpo
- Endentação
- Aplique boas práticas de testes unitários
 - Escreva testes isolados
 - Defina bem escopo para cada teste
 - Nomenclatura e clareza
 - Boa cobertura

Execução do Projeto

- 1- Ferramentas Utilizadas
- 2- Repositório GitHub
- 3- Decisões arquiteturais
- 4- Exemplo de utilização , execução de aplicação local

Ferramentas utilizadas

Java 17 + SpringBoot

Kafka

MongoDB (Na Azure)

Repositórios GitHub:

Consumidor Kafka:

https://github.com/pvsobrinho/mouts_consumer_kafka_order

Produtor Kafka:

https://github.com/pvsobrinho/mouts_consumer_kafka_order

Decisões arquiteturais

A arquitetura utilizada para resolver o problema é baseada em uma comunicação assíncrona entre duas APIs separadas por meio do Kafka, o que se alinha bem com os requisitos de alta disponibilidade e processamento paralelo de mensagens. Eu optei por dividir as responsabilidades entre um produtor e um consumidor Kafka, cada um rodando em uma API independente, devido à **necessidade de isolamento e escalabilidade**.

A API do produtor é responsável por gerar e enviar mensagens, no caso, informações de produtos, para o tópico configurado no **Kafka**. Essa API encapsula a lógica de envio, utilizando um **KafkaTemplate** configurado com serialização **JSON** para garantir que as mensagens sejam enviadas no formato correto.

Já a API do consumidor se encarrega de processar essas mensagens. Ela está configurada para desserializar as mensagens recebidas em uma classe local **ProdutoRequest**, permitindo que a estrutura de dados de produtos seja própria da API consumidora, sem dependência direta da estrutura do produtor. Utilizar duas classes ProdutoRequest em pacotes diferentes permite que cada API tenha controle sobre as suas próprias implementações de dados, evitando o acoplamento e facilitando futuras mudanças.

Essa arquitetura é conhecida como Arquitetura de **Microserviços**, onde cada serviço (ou API) é independente, mas pode se comunicar de maneira eficiente através de um sistema de mensageria como o Kafka. A separação dos serviços facilita o escalonamento, pois cada API pode ser escalada conforme a demanda de forma independente, garantindo que a solução consiga lidar com um grande volume de mensagens diárias sem impactar o desempenho.

Melhorias aplicadas na arquitetura:

Decidi adicionar um **Exception Handler** centralizado para lidar com erros de forma consistente em toda a API. Dessa forma, consigo capturar e tratar exceções específicas e retorná-las ao cliente com mensagens claras e status HTTP adequados. Isso melhora a experiência do usuário, pois ele recebe feedback claro sobre o que aconteceu.

Além disso, adicionei construtores nas classes onde isso era necessário. Ao fazer isso, facilito a criação de objetos com todos os atributos obrigatórios, garantindo que a aplicação se mantenha coerente e reduzindo a possibilidade de erros de instânciação.

Também incluí **validadores** diretamente nos controladores. Com essas validações, posso garantir que os dados recebidos estejam no formato correto antes de prosseguir com a lógica de negócio. Isso ajuda a evitar problemas que poderiam surgir caso dados incorretos fossem processados, além de fornecer uma resposta rápida ao cliente caso ele envie dados inválidos.

Essas decisões têm como objetivo tornar a API mais robusta e de fácil manutenção, garantindo que erros sejam tratados de forma padronizada e que apenas dados válidos sejam processados

Escolha do MongoDB

Escolhi MongoDB porque ele atende aos requisitos de flexibilidade e escalabilidade do projeto. Como um banco de dados orientado a documentos, ele permite armazenar dados de produtos em um **formato flexível (BSON)**, ideal para estruturas que podem evoluir.

Além disso, o **MongoDB é altamente escalável**, essencial para lidar com o volume diário de **pedidos (150.000 a 200.000) sem comprometer o desempenho**. Ele também suporta operações rápidas, como buscas e verificações de duplicidade, atendendo à necessidade de **alta performance** e consistência. A integração com o Kafka para escrita assíncrona se alinha ao modelo de consistência eventual do MongoDB, mantendo a aplicação responsiva e preparada para expansão futura.

Design Pattern escolhido

A arquitetura do projeto implementa princípios sólidos de SOLID para garantir uma estrutura bem organizada e de fácil manutenção. Cada camada possui responsabilidades claramente definidas:

1. **Single Responsibility Principle:** As classes têm uma função única e específica; por exemplo, `ProdutoService` concentra a lógica de produtos, enquanto `GlobalExceptionHandler` é responsável pelo tratamento centralizado de exceções.
2. **Open/Closed Principle:** O código é projetado para ser extensível sem a necessidade de modificações diretas, permitindo a adição de funcionalidades de forma eficiente e segura.
3. **Dependency Inversion Principle:** Os serviços dependem de abstrações, tornando o sistema mais flexível e facilitando a realização de testes unitários, pois as dependências podem ser injetadas e manipuladas de forma independente.

Esses princípios promovem uma arquitetura robusta, garantindo que o sistema seja escalável, testável e de fácil manutenção, permitindo a evolução contínua do projeto

Configurando o MongoDB na nuvem Azure (Exemplo de configuração)


Microsoft Azure

Pesquisar recursos, serviços e documentação

[Página inicial](#) > [puc-mongo-graphql](#) > [Marketplace](#) >

Azure Cosmos DB for MongoDB

Microsoft



Azure Cosmos DB for MongoDB

Microsoft | Azure Service

★ 4.5 (104 classificações)

Plano

Azure Cosmos DB for MongoDB

Criar

Visão geral

Planos

Informações de Uso + Suporte

Classificações e Análises

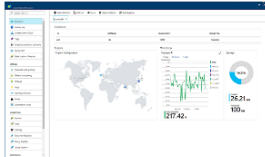
The [Azure Cosmos DB for MongoDB](#) makes it easy to use Cosmos DB as if it were a MongoDB database. You can leverage your MongoDB experience and continue using MongoDB drivers, SDKs, and tools by pointing your application to the API for MongoDB account's connection string.

The API for MongoDB has [numerous added benefits](#) of being built on Azure Cosmos DB:

- Cost efficient, granular, unlimited, instantaneous scalability
- Automatic and built-in sharding
- Five 9's of availability and turnkey geo distribution
- Affordable, flexible pricing with serverless billing and free tier
- Fully managed upgrades and maintenance
- Real time analytics (HTAP) at any scale with Azure Synapse

To learn more, check out the [Azure Cosmos DB for MongoDB Documentation](#).

Mídia



A configuração escolhida foi o Cluster Free Tier. Apenas como exemplo já que não vamos precisar de um desempenho de um sistema produtivo e não gerar custos na nuvem.

Microsoft Azure

Pesquisar recursos, serviços e documentação

[Página inicial](#) > [Azure Cosmos DB for MongoDB](#) > [Create Azure Cosmos DB Account - Choose Architecture](#) >

Create Azure Cosmos DB for MongoDB cluster

Microsoft

Global distribution is currently available only in preview. Enable access to this feature on the Basics tab by selecting 'Access to global distribution (preview)'. [Learn more](#)

Noções básicas

Global distribution (preview)

Rede

Marcas

Examinar + criar

Project details

Selecione a assinatura para gerenciar os recursos e os custos implantados. Use grupos de recursos como pastas para organizar e gerenciar todos os seus recursos.

Assinatura * ⓘ

Azure for Students

Resource group * ⓘ

puc-mongo-graphql

[Criar novo](#)

Cluster details

Cluster name * ⓘ

motus

Access to global distribution (preview) ⓘ

☐

Free tier ⓘ

☒

Localização * ⓘ

(Asia Pacific) South India

Cluster tier ⓘ

Shards: 1, no high availability (HA)

Free tier, 0.00 USD / month

MongoDB version ⓘ

7.0

Administrator account

Microsoft Azure

Página inicial > Azure Cosmos DB for MongoDB > Create Azure Cosmos DB Account - Choose Architecture >

Create Azure Cosmos DB for MongoDB cluster

Microsoft

Product details

Azure Cosmos DB for MongoDB cluster by Microsoft

Estimated cost per month0.00 USD

Terms of use

Terms

By clicking "Create", I (a) agree to the legal terms and privacy statement(s) associated with the Marketplace offering(s) list above; (b) authorize Microsoft to bill my current payment method for the fees associated with the offering(s), with the same billing frequency as my Azure subscription; and (c) agree that Microsoft may share my contact, usage and transactional information with the provider(s) of the offering(s) for support, billing and other transactional activities. Microsoft does not provide rights for third-party offerings. For additional details see [Azure Marketplace Terms](#)

Noções básicas

Assinatura	dff51c3a-2cbf-4bdb-b02a-2dd8a959e2d4
Grupo de Recursos	puc-mongo-graphql
Localização	(Asia Pacific) South India
Cluster name	motus
Server admin login name	motuser
Shards	1
Shard compute + storage	Free tier
High availability	Não
MongoDB version	7.0
Preview features	Não

Global distribution (preview)

Geo-replica	Não
-------------	-----

Rede

Método de conectividade	Acesso público (allowed IP addresses)
-------------------------	---------------------------------------

Criar

< Anteriores

Próximo >

Baixar um modelo para automação

A configuração do Azure para liberar todos os IPS para acessar este recurso. Já que é uma aplicação de teste e no FreeTier.

Configure IP address in firewall rules

You need to configure at least one IP address in Public access (allowed IPs) to enable access to this cluster. If you continue without configuring IP address then you must configure the IP address later to allow access to this cluster. [Learn more](#)

Create cluster without firewall rules

Return to add firewall rules

Microsoft Azure

Pesquisar recursos, serviços e documentos (G+J)

Copilot


Página inicial

>

Grupos de recursos

>

puc-mongo-graphql



puc-mongo

Azure Cosmos DB for MongoDB (vCore)

☆

...

Pesquisar

Atualizar

Reset password

Excluir

Visão geral

Log de atividade

IAM (Controle de acesso)

Marcações

Diagnosticar e resolver problemas

Início rápido (preview)

Configurações

Monitoramento

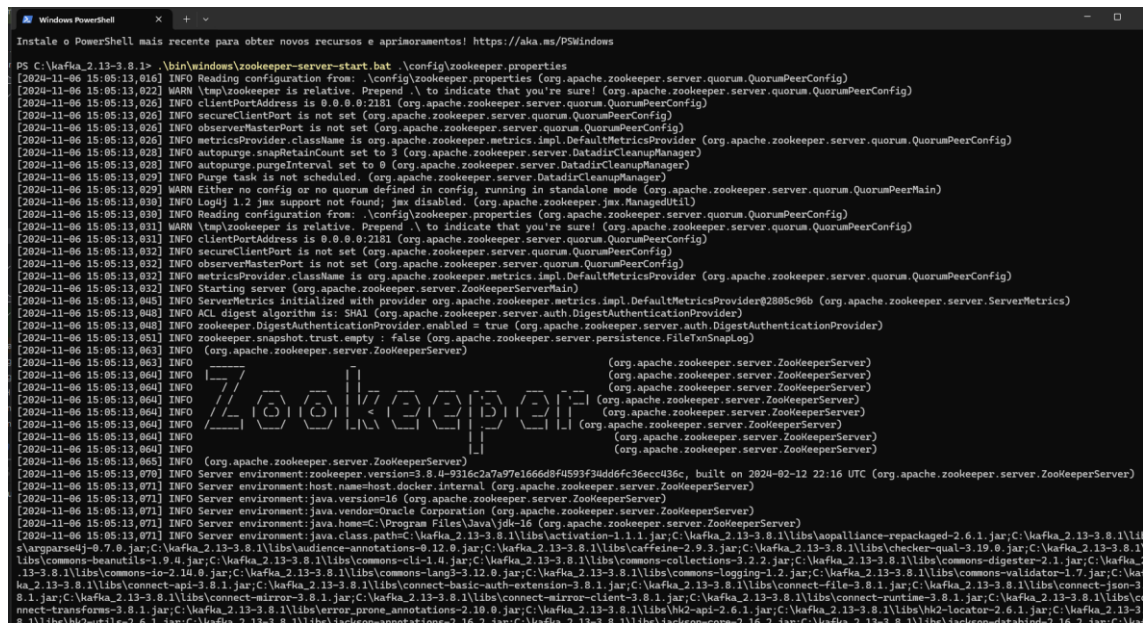
Automação

Ajuda

Fundamentos

Status	: Ready	Cluster tier	: Free tier
Grupo de recursos	: puc-mongo-graphql	Shard count	: 1
Assinatura	: Azure for Students	Disk size	: 32
ID da Assinatura	: dff51c3a-2cbf-4bdb-b02a-b2d8a959e2d4	Método de conectividade	: Public access
Localização	: South India	High availability	: Não
MongoDB version	: 7.0		
Admin username	: pucmongo1		
Rótulos	: Adicionar marcas		

<https://kafka.apache.org/downloads>



```
.\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties
```

```
.\bin\windows\kafka-server-start.bat .\config\server.properties
```

Enviando um produto da API produto para API ordem via mensagem assíncrona com o Kafka

localhost:8080/swagger-ui/index.html#/produto-a-controller/enviarProduto

Swagger
Supports Swagger 2.0

/v3/api-docs

API de Produtos - Produtora 1.0 OAS3

[/v3/api-docs](#)

Documentação da API para envio de produtos

Servers

produto-a-controller

POST	/produto-a/enviar
Parameters	
No parameters	
Request body <small>required</small>	
Example Value Schema	
<pre>{ "id": "string", "nome": "string", "preco": 0, "quantidade": 0}</pre>	

Teste enviando um produto :

POST

/produto-a/enviar

Parameters

No parameters

Request body required

```
{
  "id": "1",
  "nome": "TESTE",
  "preco": 110,
  "quantidade": 10
}
```

Execute

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8080/produto-a/enviar' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "id": "1",
    "nome": "TESTE",
    "preco": 110,
    "quantidade": 10
  }'
```

Request URL

http://localhost:8080/produto-a/enviar

Produto foi recebido no consumidor:

```
public class KafkaConsumerConfig {  
    public ConcurrentKafkaListenerContainerFactory<String, ProdutoRequest> kafkaListenerContainerFactory() {  
        factory.setConsumerFactory(consumerFactory());  
        return factory;  
    }  
  
    @Bean  
    public DefaultKafkaConsumerFactory<String, ProdutoRequest> consumerFactory() {  
        Map<String, Object> props = new HashMap<>();  
        props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");  
        props.put(ConsumerConfig.GROUP_ID_CONFIG, "order-group");  
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);  
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, JsonDeserializer.class);  
        props.put(JsonDeserializer.TRUSTED_PACKAGES, "com.example.motus_order.dto");  
        props.put(JsonDeserializer.VALUE_DEFAULT_TYPE, "com.example.motus_order.dto.ProdutoRequest");  
        props.put(JsonDeserializer.USE_TYPE_INFO_HEADERS, false);  
  
        return new DefaultKafkaConsumerFactory<>(props);  
    }  
}
```

Console Output:

```
2024-11-06T16:51:54.845-03:00 INFO 15084 --- [main] o.e.k.c.c.internals.LegacyKafkaConsumer : (Consumer clientId=consumer-order-group-1, groupId=order-group) Subscribed to topic  
2024-11-06T16:51:54.856-03:00 INFO 15084 --- [main] c.e.motus_order.MotusOrderApplication : Started MotusOrderApplication in 1.171 seconds (process running for 1.58)  
2024-11-06T16:51:55.091-03:00 INFO 15084 --- [ntainer#0-0-C-1] org.apache.kafka.clients.Metadata : [Consumer clientId=consumer-order-group-1, groupId=order-group] Cluster ID: E5vjQ5f  
2024-11-06T16:51:55.094-03:00 INFO 15084 --- [ntainer#0-0-C-1] o.e.k.c.c.internals.ConsumerCoordinator : [Consumer clientId=consumer-order-group-1, groupId=order-group] Discovered coord  
2024-11-06T16:51:55.106-03:00 INFO 15084 --- [ntainer#0-0-C-1] o.e.k.c.c.internals.ConsumerCoordinator : [Consumer clientId=consumer-order-group-1, groupId=order-group] (Re-)joining group  
2024-11-06T16:51:55.108-03:00 INFO 15084 --- [ntainer#0-0-C-1] o.e.k.c.c.internals.ConsumerCoordinator : [Consumer clientId=consumer-order-group-1, groupId=order-group] Request joining group  
2024-11-06T16:51:55.113-03:00 INFO 15084 --- [ntainer#0-0-C-1] o.e.k.c.c.internals.ConsumerCoordinator : [Consumer clientId=consumer-order-group-1, groupId=order-group] (Re-)joining group  
2024-11-06T16:51:55.116-03:00 INFO 15084 --- [ntainer#0-0-C-1] o.e.k.c.c.internals.ConsumerCoordinator : [Consumer clientId=consumer-order-group-1, groupId=order-group] Successfully joined  
2024-11-06T16:51:55.117-03:00 INFO 15084 --- [ntainer#0-0-C-1] o.e.k.c.c.internals.ConsumerCoordinator : [Consumer clientId=consumer-order-group-1, groupId=order-group] Finished assignment  
2024-11-06T16:51:55.119-03:00 INFO 15084 --- [ntainer#0-0-C-1] o.e.k.c.c.internals.ConsumerCoordinator : [Consumer clientId=consumer-order-group-1, groupId=order-group] Successfully synced  
2024-11-06T16:51:55.126-03:00 INFO 15084 --- [ntainer#0-0-C-1] k.c.c.i.ConsumerRebalanceListenerInvoker : [Consumer clientId=consumer-order-group-1, groupId=order-group] Notifying assignor  
2024-11-06T16:51:55.127-03:00 INFO 15084 --- [ntainer#0-0-C-1] o.e.k.c.c.internals.ConsumerUtils : [Consumer clientId=consumer-order-group-1, groupId=order-group] Adding newly assigned partitions to the committed offset fetcher  
2024-11-06T16:51:55.127-03:00 INFO 15084 --- [ntainer#0-0-C-1] o.e.k.l.KafkaMessageListenerContainer : Setting offset for partition nome-do-topico-produtoA-0 to the committed offset fetcher  
2024-11-06T16:51:55.127-03:00 INFO 15084 --- [ntainer#0-0-C-1] o.e.k.l.KafkaMessageListenerContainer : order-group: partitions assigned: [nome-do-topico-produtoA-0]  
Produto recebido via Kafka: ProdutoRequest(id=4, nome=teste, preco=100.0, quantidade=1)  
Produto recebido via Kafka: ProdutoRequest(id=1, nome=TESTE, preco=110.0, quantidade=10)
```

Teste de validações:

```
2024-11-06T18:38:13.164-03:00 INFO 13000 --- [azure.com:10260] org.mongodb.driver.cluster : Discovering new topology  
Produto recebido via Kafka: ProdutoRequest(id=3, nome=TESTE, preco=110.0, quantidade=10)  
Produto recebido via Kafka: ProdutoRequest(id=3, nome=TESTE, preco=110.0, quantidade=10)  
Produto 3 já existe na base de dados, não será salvo novamente.  
Produto recebido via Kafka: ProdutoRequest(id=3, nome=TESTE, preco=110.0, quantidade=10)
```

```
2024-11-06T18:44:08.204-03:00 INFO 14732 --- [azure.com:10260] org.mongodb.driver.cluster : Monitor thread successfully started  
2024-11-06T18:44:08.206-03:00 INFO 14732 --- [azure.com:10260] org.mongodb.driver.cluster : Discovered cluster type of S  
Produto recebido via Kafka: ProdutoRequest(id=3, nome=TESTE, preco=110.0, quantidade=10)  
Produto 3 já existe na base de dados, não será salvo novamente.  
Produto recebido via Kafka: ProdutoRequest(id=4, nome=TESTE, preco=110.0, quantidade=10)  
Produto 4 foi gravado na base de dados na nuvem Azure.  
Produto recebido via Kafka: ProdutoRequest(id=5, nome=produto chocolate, preco=6546.0, quantidade=10)  
Produto 5 foi gravado na base de dados na nuvem Azure.  
Produto recebido via Kafka: ProdutoRequest(id=7, nome=produto biscoito, preco=6546.0, quantidade=10)  
Produto 7 foi gravado na base de dados na nuvem Azure.
```