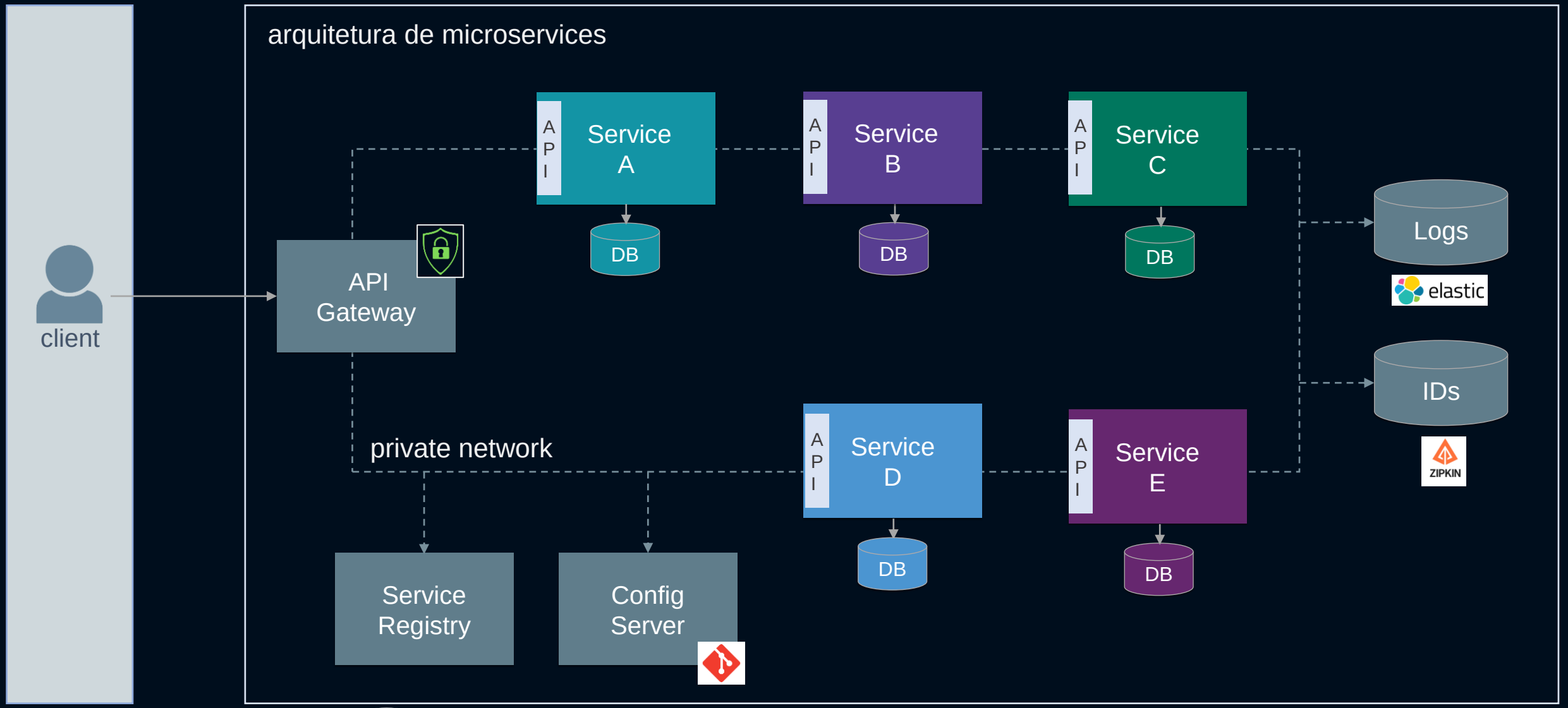
DECODER WEEK

Modelos de comunicações entre Microservices

DIA 2

Arquitetura de Microservices - Abstração



Base de Dados Compartilhada

ou

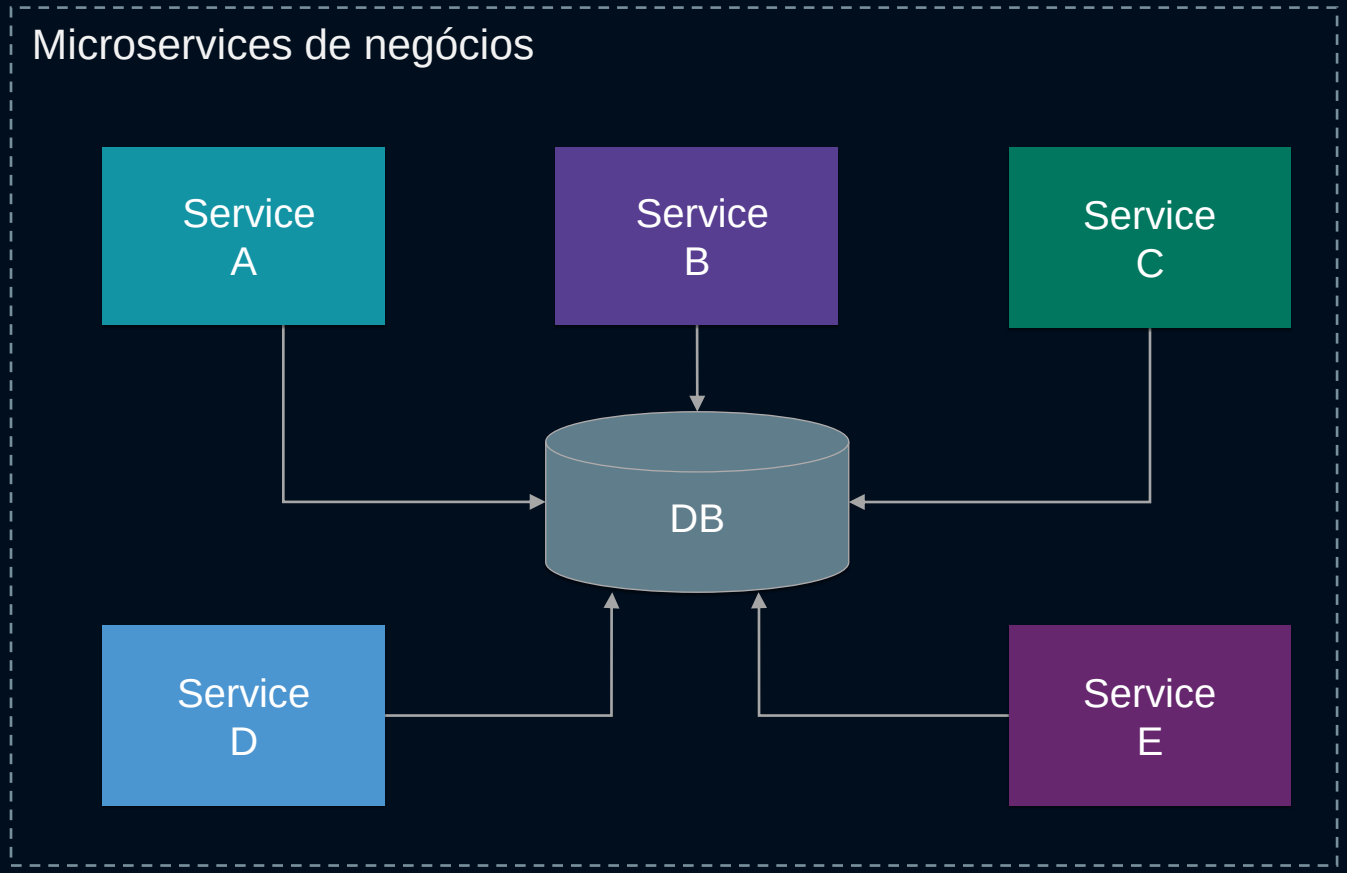
Base de Dados por Microservices

Base de Dados Compartilhada

Base de dados compartilhada garante forte consistência.

Base de dados compartilhada gera forte acoplamento, sem isolamento da modelagem de dados.

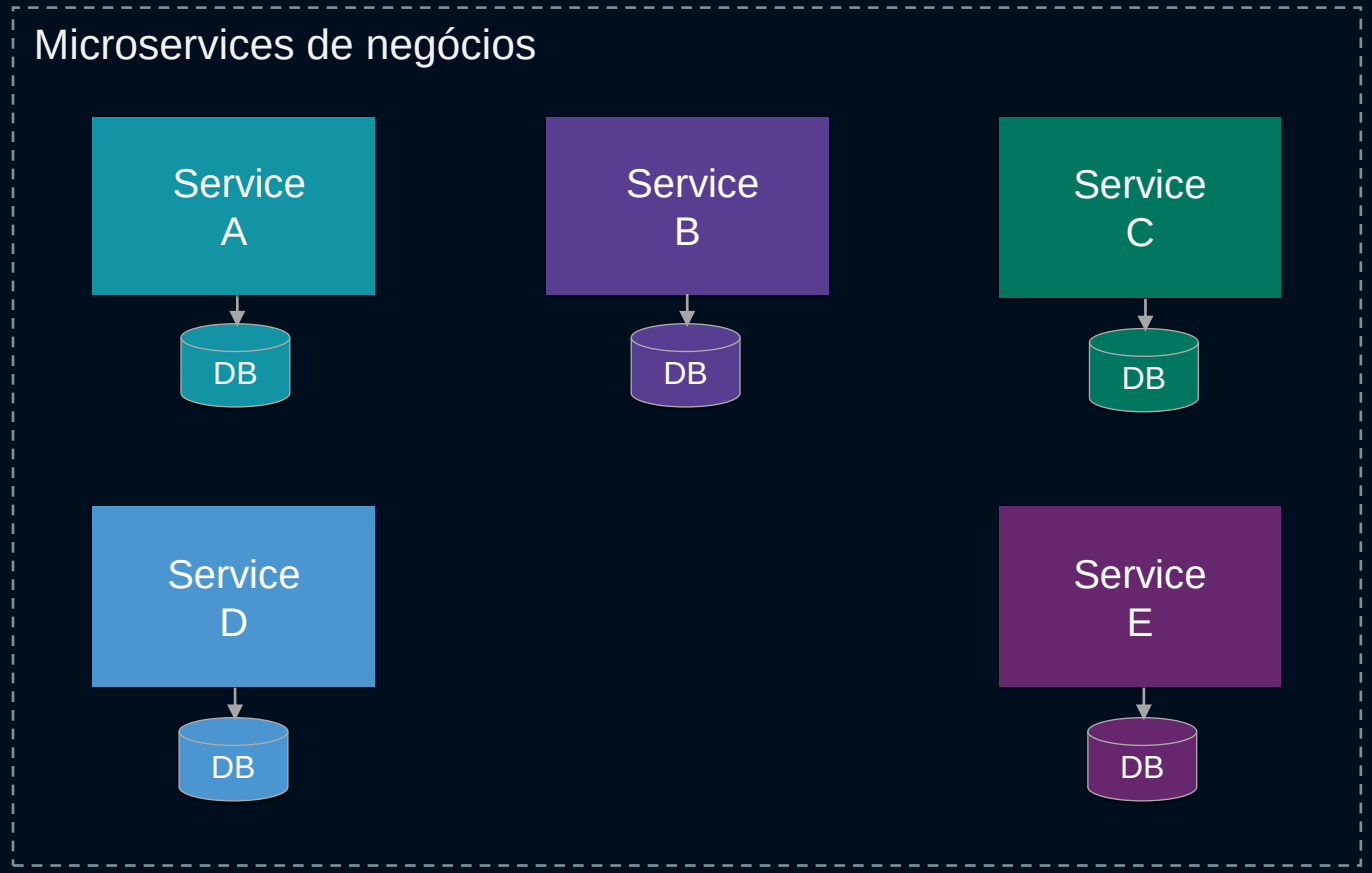
Na migração de Monolítico para Microservices o uso de base de dados compartilhada é comum no inicio.



Base de Dados por Microservices (Distribuída)

Base de dados por microservice geram menor acoplamento na arquitetura e maior isolamento da modelagem de dados.

Com base de dados por microservices temos que lidar com a sincronia e replicação dos dados distribuídos.



Gestão dos Dados Distribuídos em Microservices

Dados Distribuídos entre Microservices

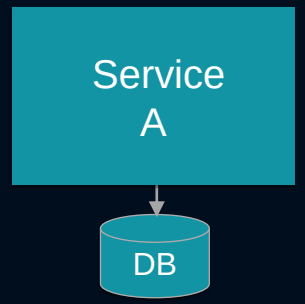
Base de Dados por Microservices

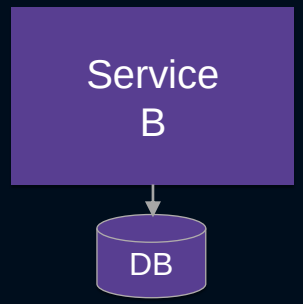
Dados Distribuídos

Sincronia e Replicação dos Dados Distribuídos

Consistência Forte

Alta Disponibilidade (Consistência Eventual)





Dados Distribuídos entre Microservices

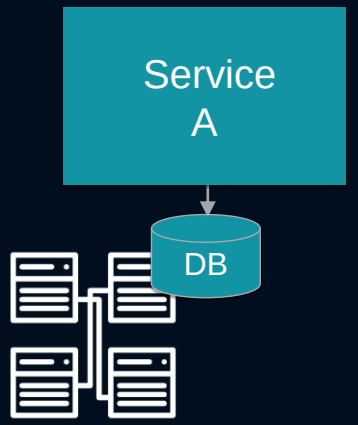
Base de Dados por Microservices

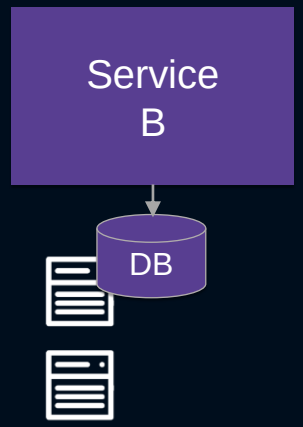
Dados Distribuídos

Sincronia e Replicação dos Dados Distribuídos

Consistência Forte

Alta Disponibilidade (Consistência Eventual)





Dados Distribuídos entre Microservices

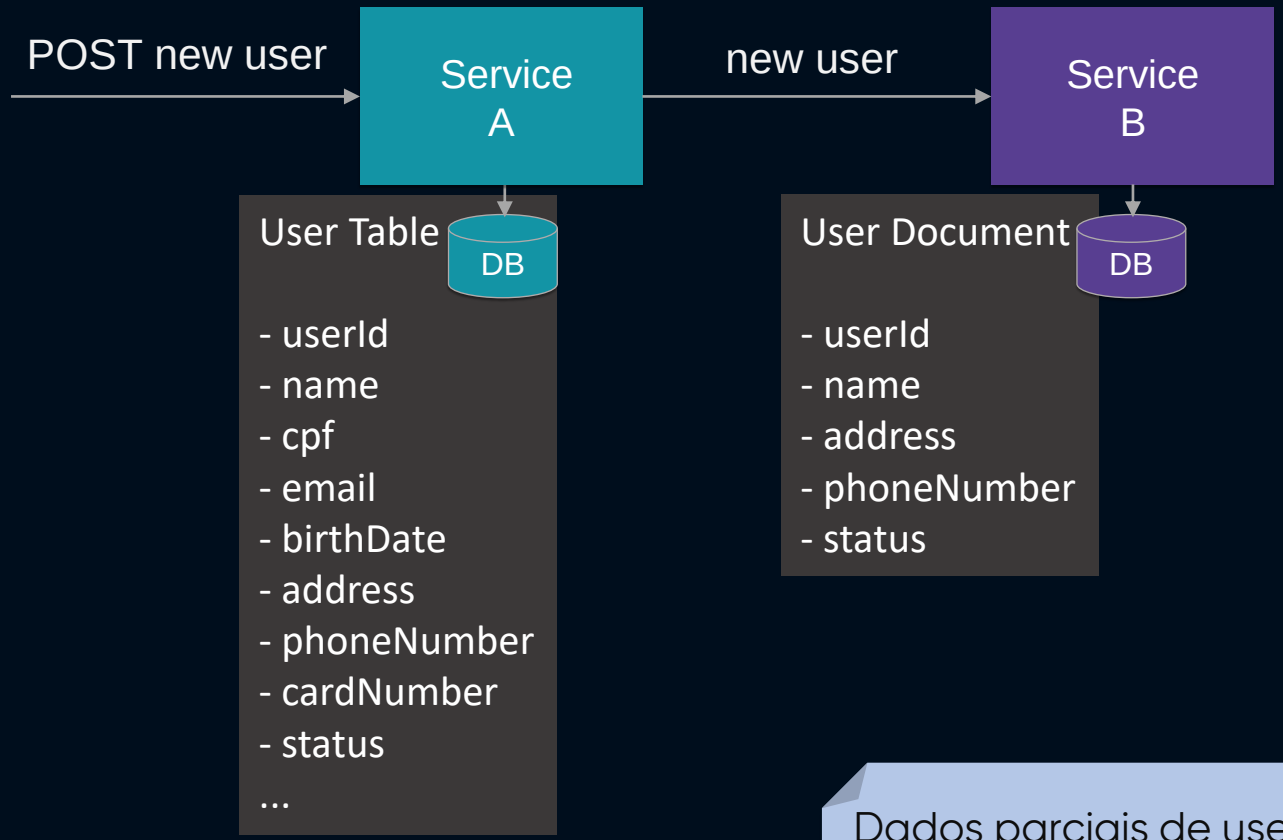
Base de Dados por Microservices

Dados Distribuídos

Sincronia e Replicação dos Dados Distribuídos

Consistência Forte

Alta Disponibilidade (Consistência Eventual)



Dados parciais de user replicados no Microservice B pela necessidade do negócio, buscando maior disponibilidade.

Dados Distribuídos entre Microservices

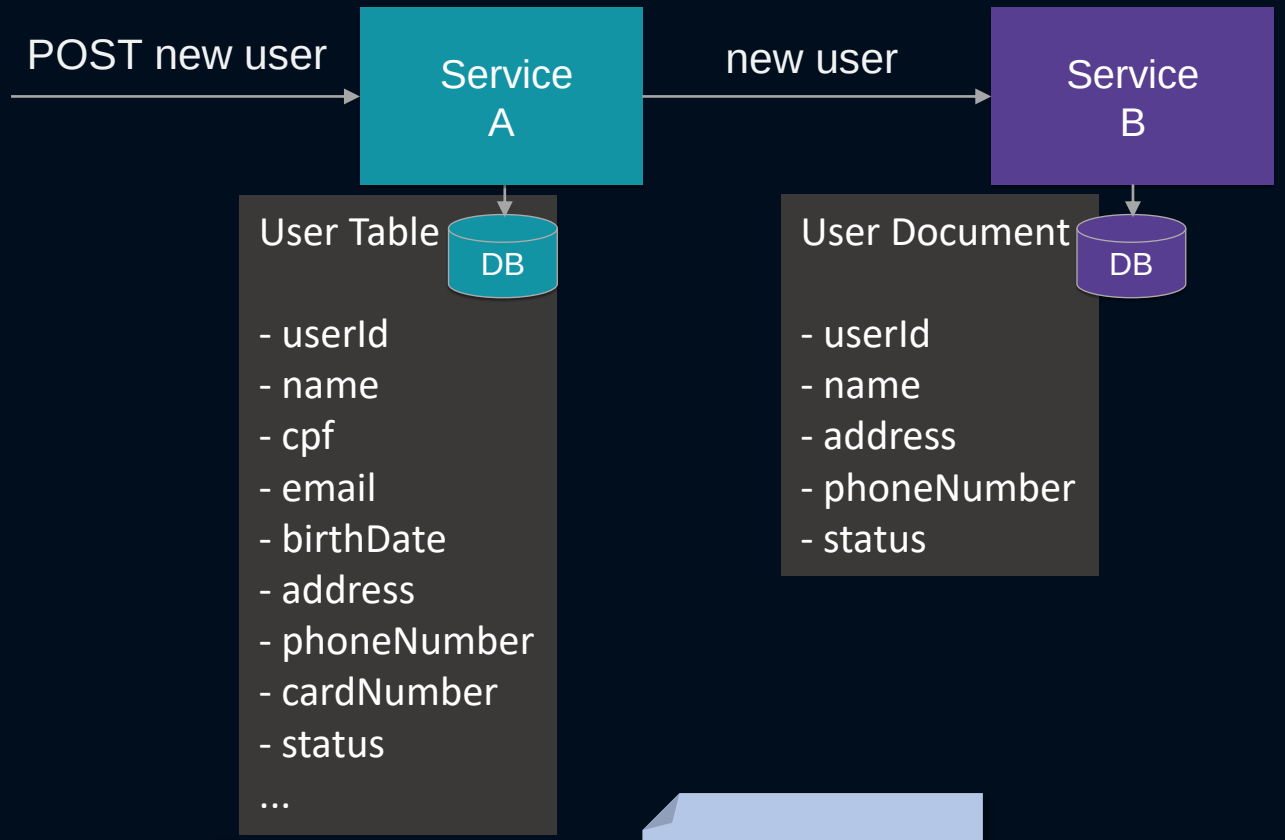
Base de Dados por Microservices

Dados Distribuídos

Sincronia e Replicação dos Dados Distribuídos

Consistência Forte

Alta Disponibilidade (Consistência Eventual)



Teorema CAP

Teorema CAP

Base de Dados por Microservices

↓

Dados Distribuídos

↓

Sincronia e Replicação dos Dados Distribuídos

↙ ↘

Consistência Forte

Alta Disponibilidade (Consistência Eventual)

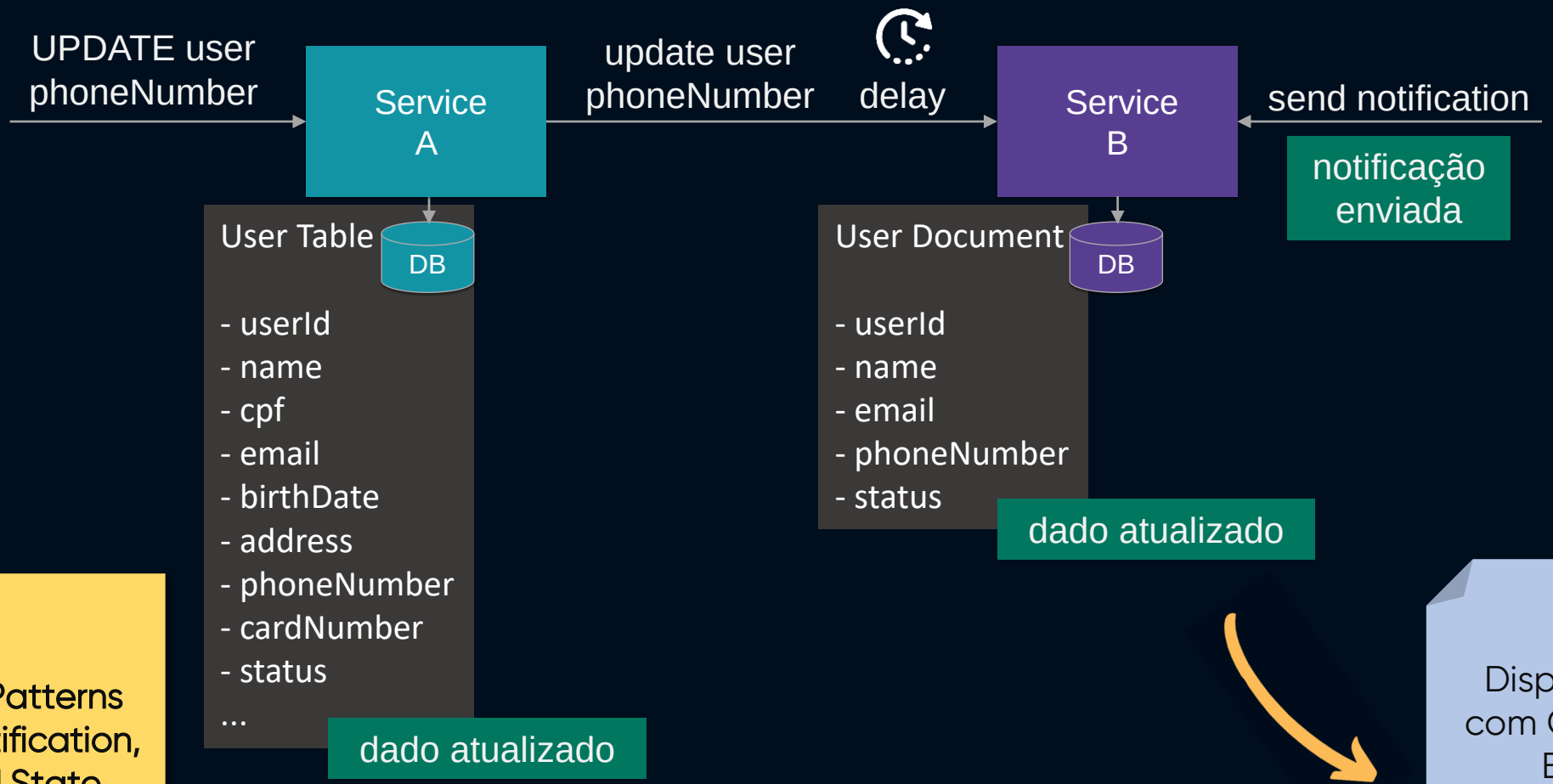
DESCOMPLICANDO

#decoderweek

Teorema CAP
Alta Disponibilidade *versus* Consistência Forte

Não há como garantir alta disponibilidade e consistência forte ao mesmo tempo em uma arquitetura de Microservices com dados distribuídos, ou seja, em favor da alta disponibilidade consequentemente temos a Consistência Eventual.

Alta Disponibilidade e Consistência Eventual



DESCOMPLICANDO

A busca é sempre pela maior disponibilidade possível...

Mesmo não existindo na realidade uma garantia de disponibilidade de 100%.
Por isso é comum vermos casos como disponibilidade de 99,99 ou 99,9999...

Disponibilidade em Microservices

Não criamos Microservices para que qualquer um possa parar em algum momento sem afetar os demais, mas sim para que alguns possam parar eventualmente e o sistema continuar disponível.

E isso já é muito melhor do que se nenhum pudesse parar.

UUIDs – Identificadores Distribuídos

IDs Sequenciais	UUIDs Temporais
20	280c6870-a5f0-4408-b355-731e1ad20258

IDs do tipo UUID são identificadores temporais universalmente exclusivos e essenciais para a sincronia e replicação de dados distribuídos.

- Podem ser gerados em qualquer lugar
- Garantem maior manutenibilidade
- Facilitam a replicação de dados
- Únicos em qualquer base de dados

Modelos de Comunicações entre Microservices

MODELOS DE COMUNICAÇÕES VIA APIS

MODELOS DE COMUNICAÇÕES VIA MENSAGERIA

MODELO DE COMUNICAÇÃO HÍBRIDA

MODELOS DE COMUNICAÇÕES VIA APIS

- Comunicação Síncrona via APIs

– Comunicação Assíncrona via APIs

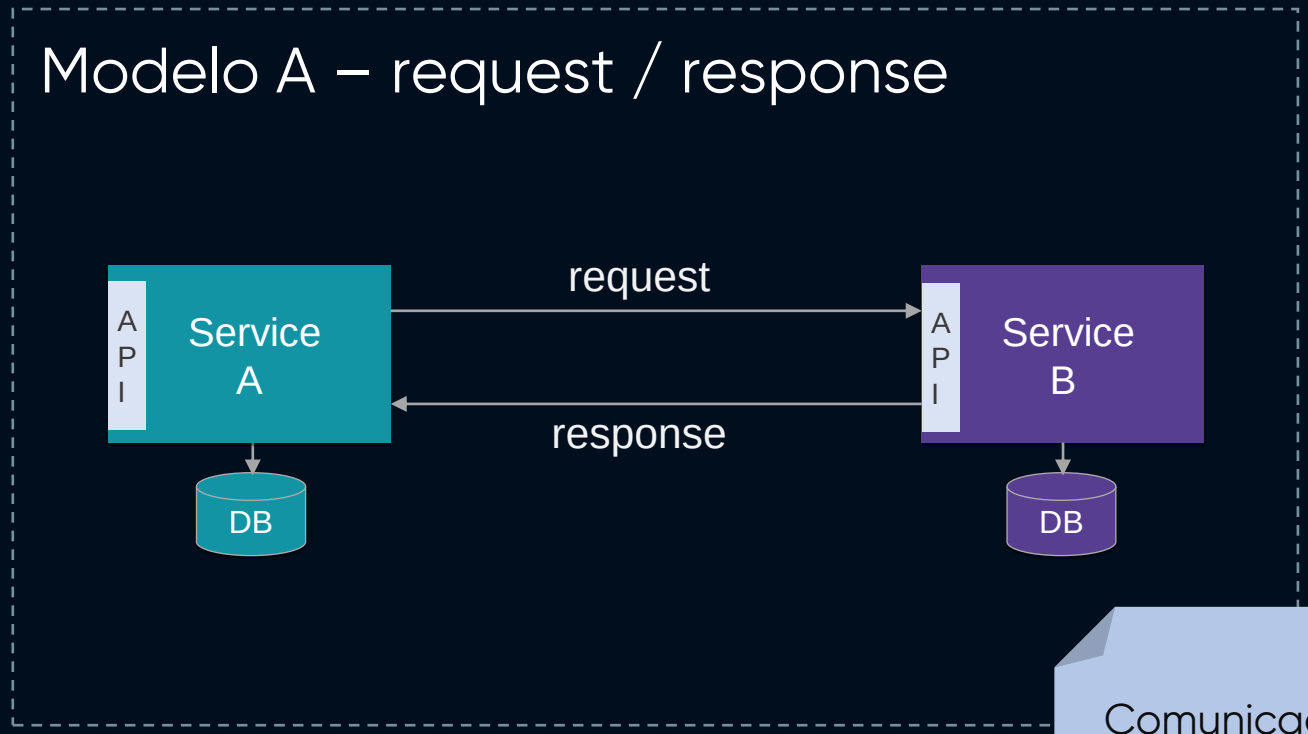
Comunicação Síncrona via APIs

Vantagens

- Forte consistência

Pontos de atenção

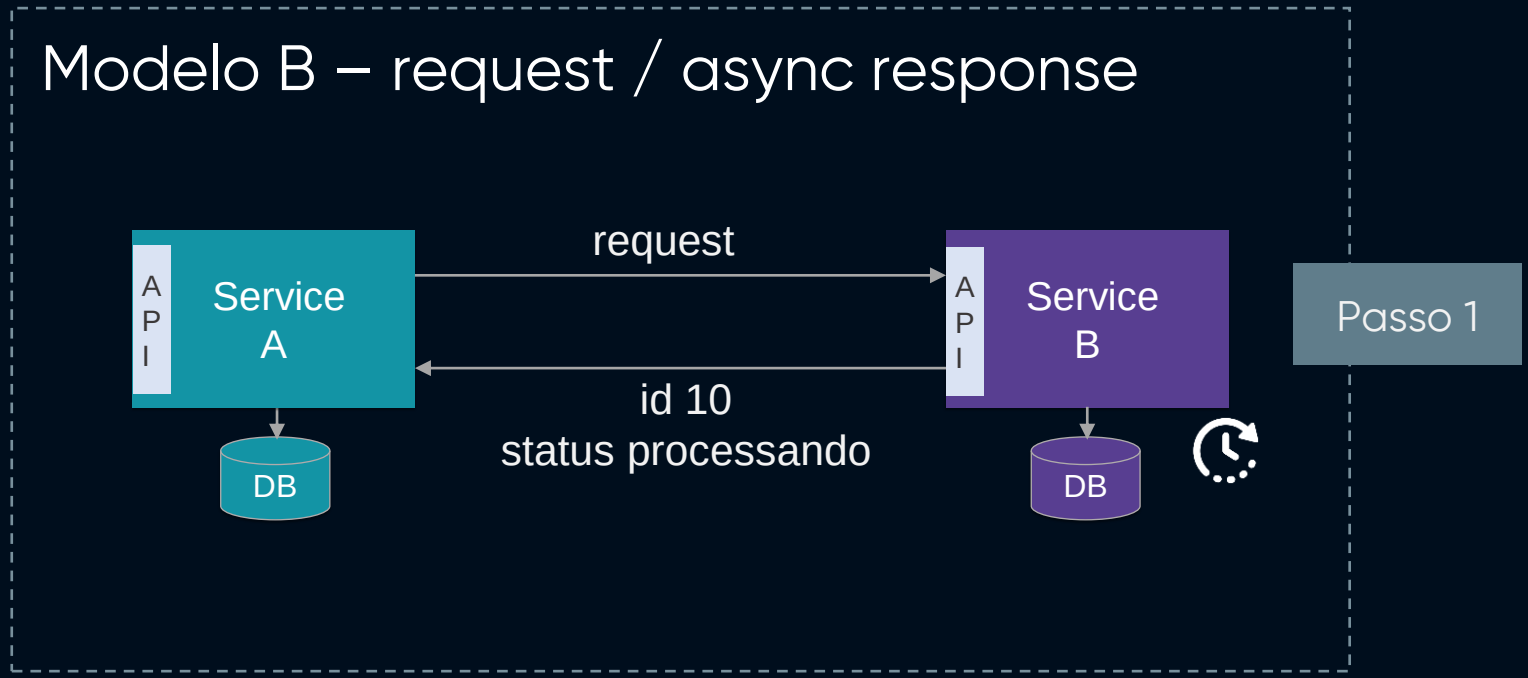
- Disponibilidade comprometida
- Comunicação bloqueante
- Aumento acoplamento entre os Microservices



Comunicação síncrona utilizando de métodos HTTP.

Métodos GET, POST, PUT, DELETE...

Comunicação Assíncrona via APIs



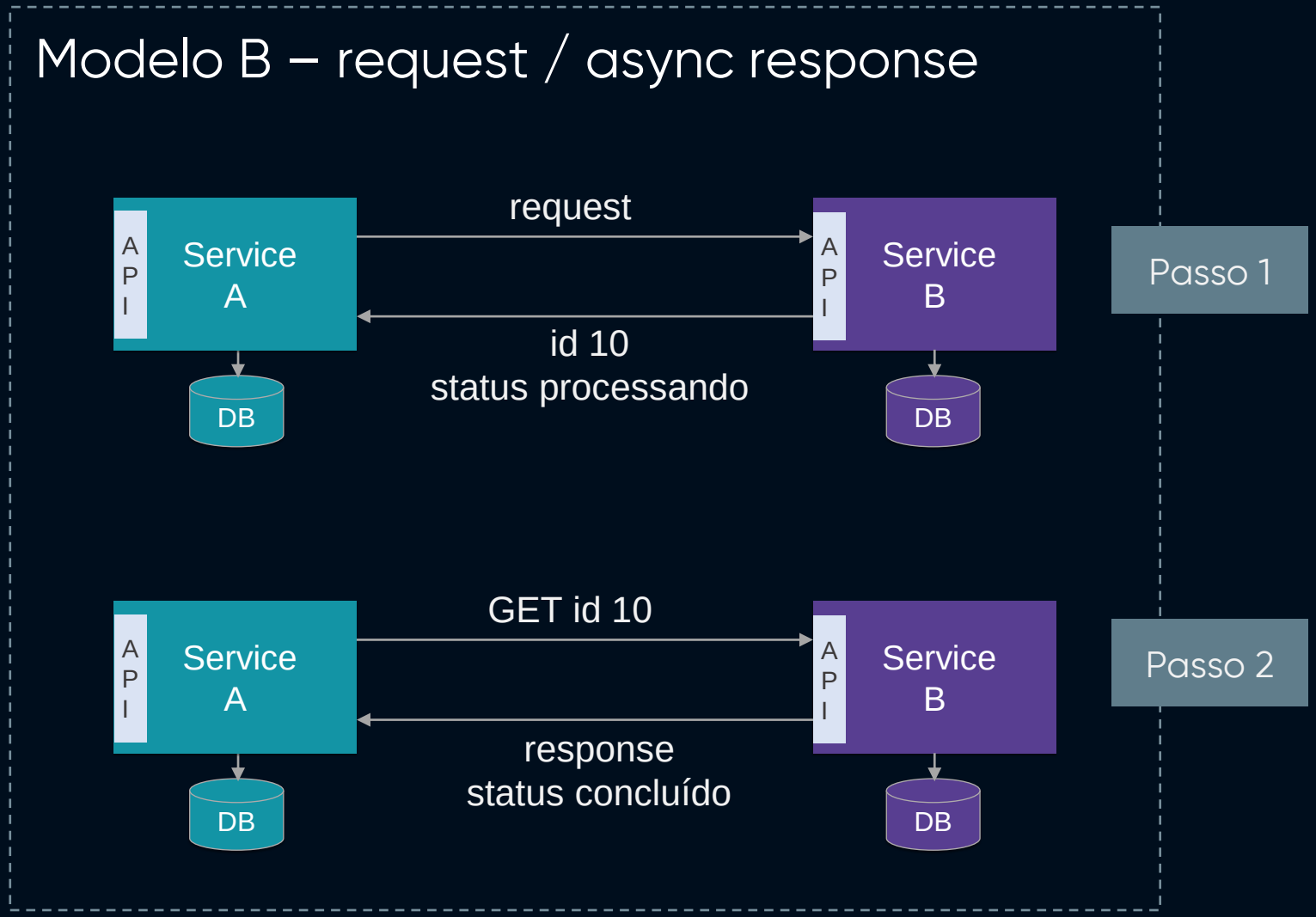
Comunicação Assíncrona via APIs

Vantagens

- Maior disponibilidade
- Comunicação não bloqueante

Pontos de atenção

- Aumento acoplamento entre os Microservices

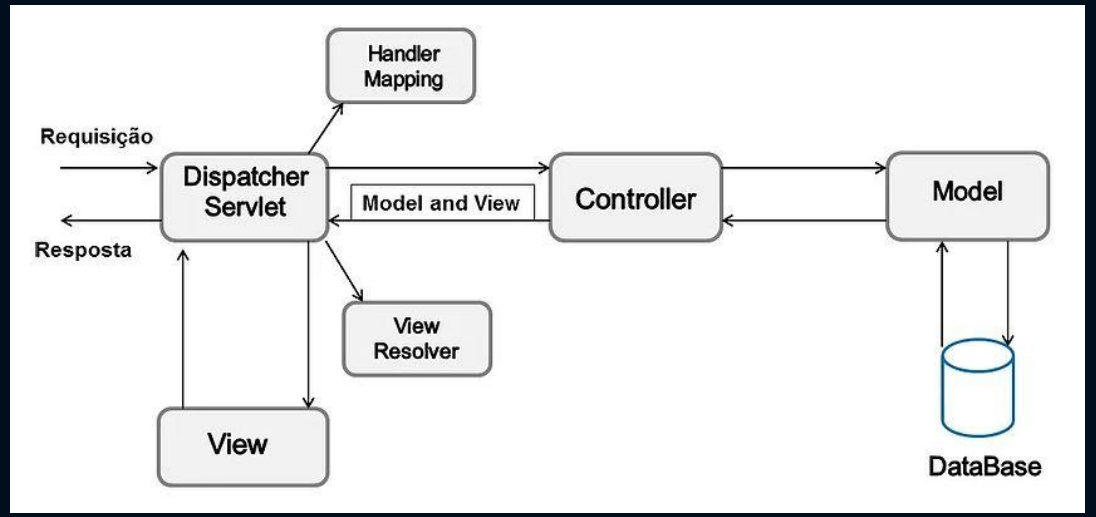


COMPLEMENTANDO

APIs com Spring Web MVC

- Conhecido como Spring MVC
- Módulo fonte: spring-webmvc
- Utiliza de um controlador frontal: Dispatcher Servlet
- Servidor Tomcat

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```



COMPLEMENTANDO

APIs Reativas com Spring Web Webflux

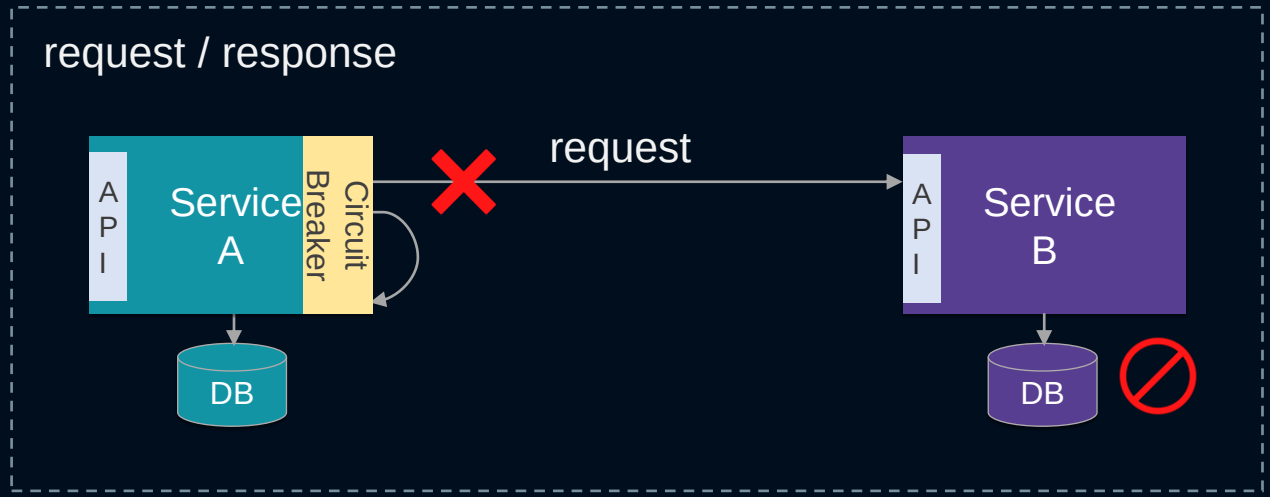
- Inserido na versão 5 do Spring Framework
- Baseado no projeto Reactor
- Criação de APIs Reativas
- Não bloqueante
- Servidor Netty
- Trabalha com os tipos Flux e Mono

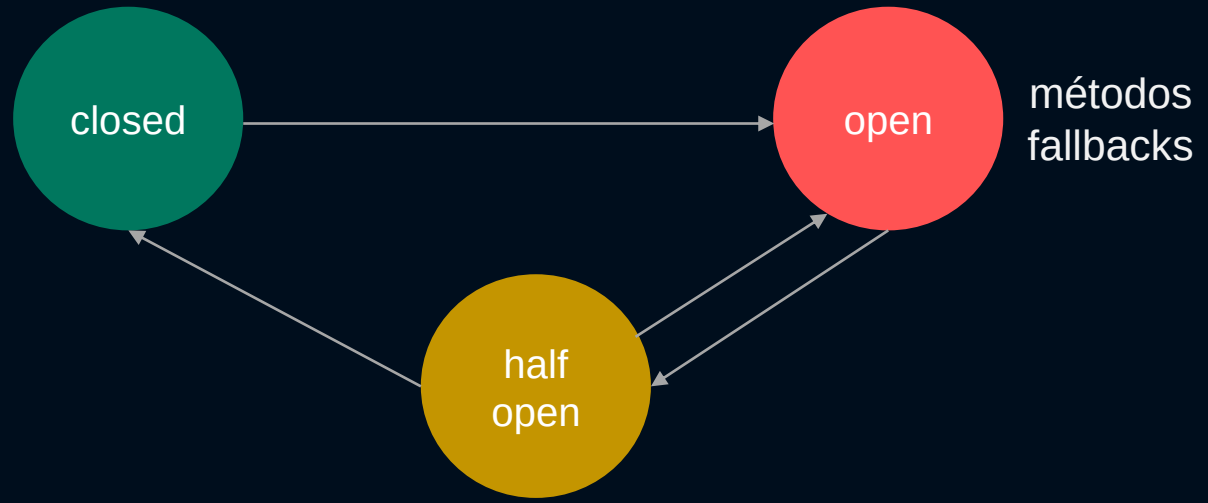
```xml
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-webflux</artifactId>
</dependency>
```

Spring oferece suporte para aplicação não bloqueante de ponta a ponta.

Resiliência com Circuit Breaker

- Disjuntor Circuit Breaker
- Estados: CLOSED, OPEN e HALF_OPEN
- Métodos fallbacks e fluxos alternativos
- Políticas de retentativas: Retry





MODELOS DE COMUNICAÇÕES VIA MENSAGERIA

- Comunicação Assíncrona via Mensageria utilizando Comandos

– Comunicação Assíncrona via Mensageria utilizando de Eventos

Mensagem

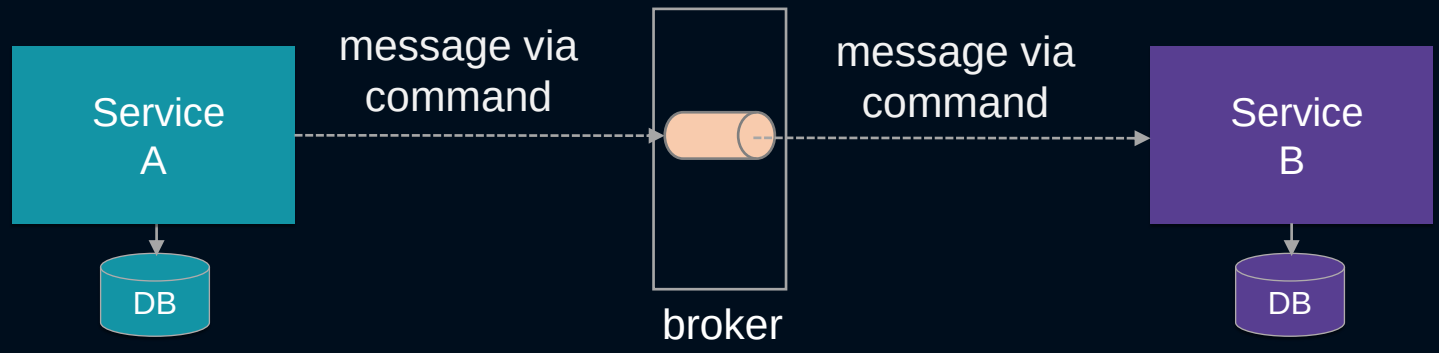
Comando

vs

Evento

Comunicação Assíncrona via Mensageria Comandos

Modelo A - One way

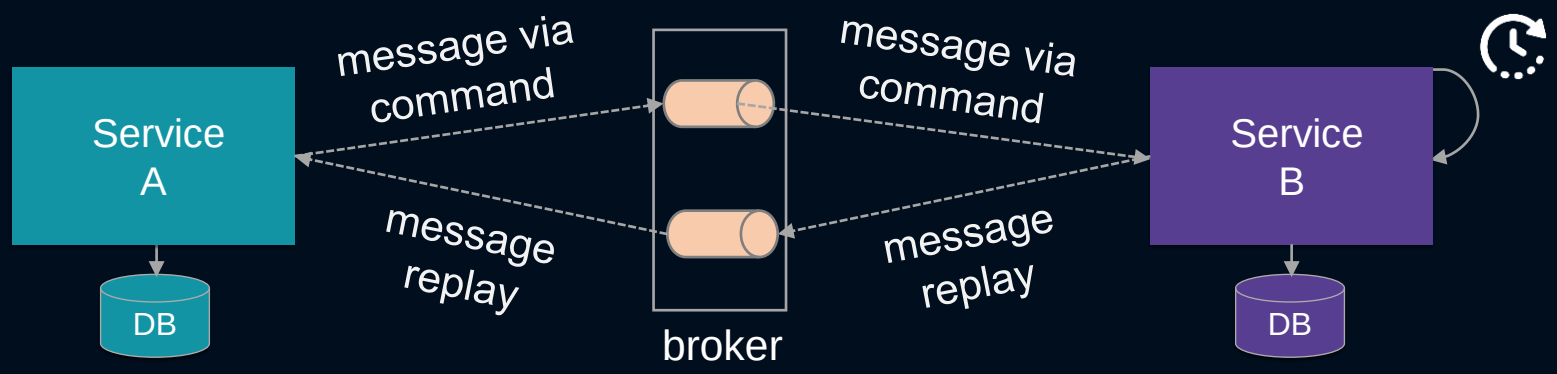


Vantagens

- Maior disponibilidade
- Menor acoplamento
- Em casos de falhas, preserva-se a mensagem (tratativas de filas DLQ, reprocessamento...)

Comunicação Assíncrona via Mensageria Comandos

Modelo B - request / async response

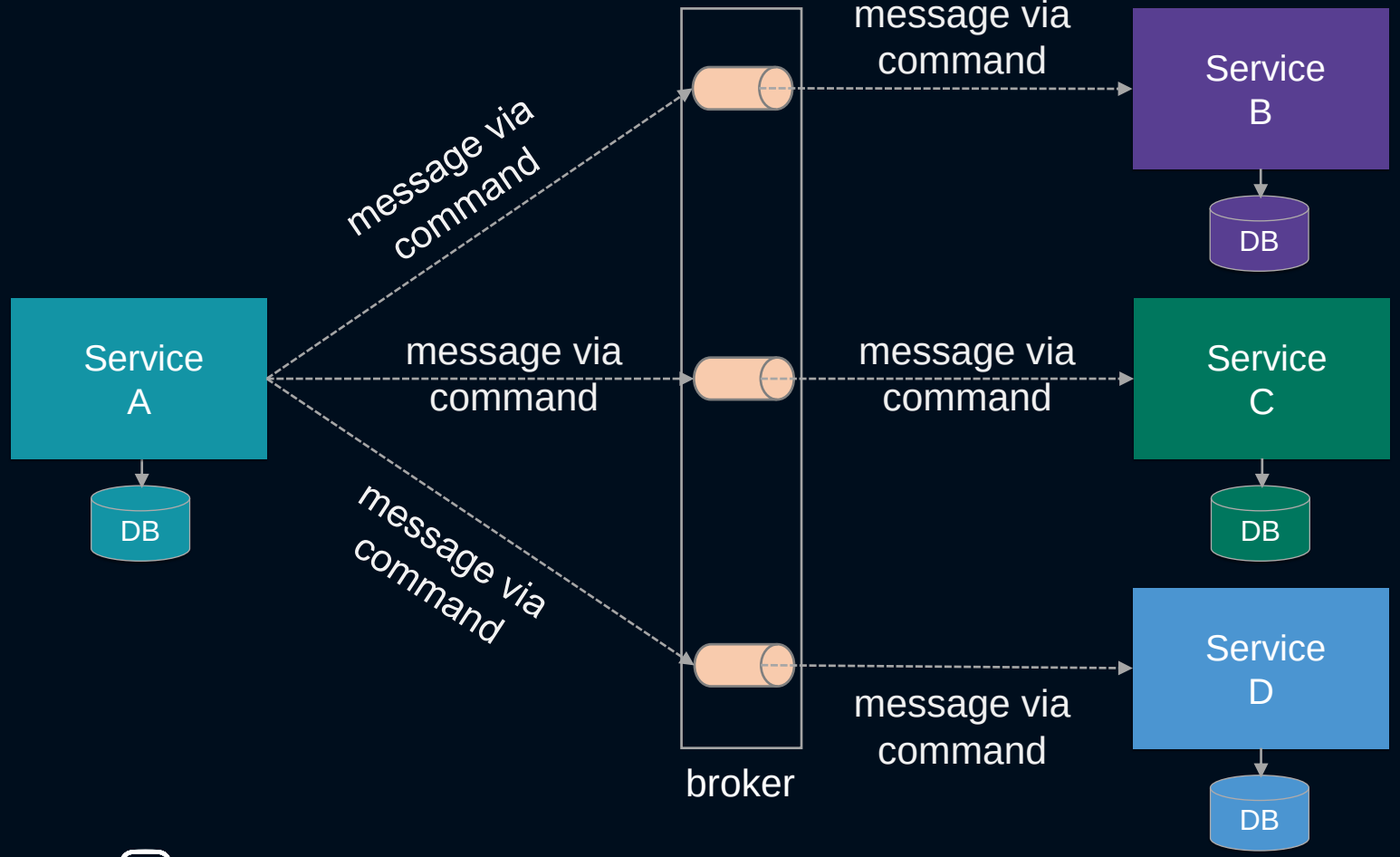


Vantagens

- Maior disponibilidade
- Comunicação não bloqueante

Comunicação Assíncrona via Mensageria Comandos

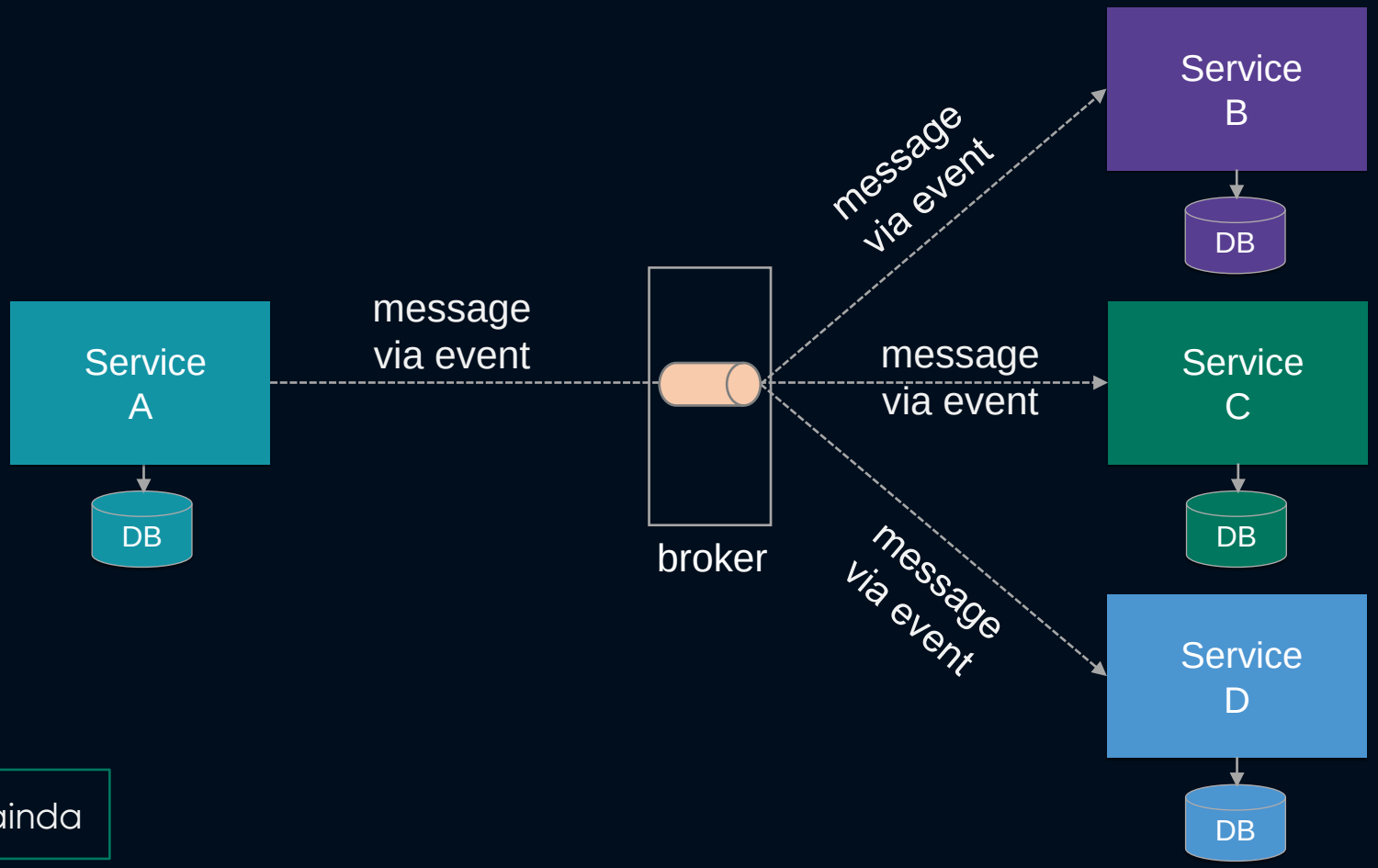
Modelo C - producer / consumers



Modelo muito utilizado em Orquestração com Saga.

Comunicação Assíncrona via Mensageria Eventos

Modelo D - publish / subscribers



Vantagens

- Menor acoplamento ainda

Modelo muito utilizado em Coreografia com Saga e replicação de dados

COMPLEMENTANDO

Mensageria com Spring AMQP

Suporte a mensagens utilizando protocolo AMQP

Enviar e receber mensagens

Definir automaticamente queues, exchanges e routing keys

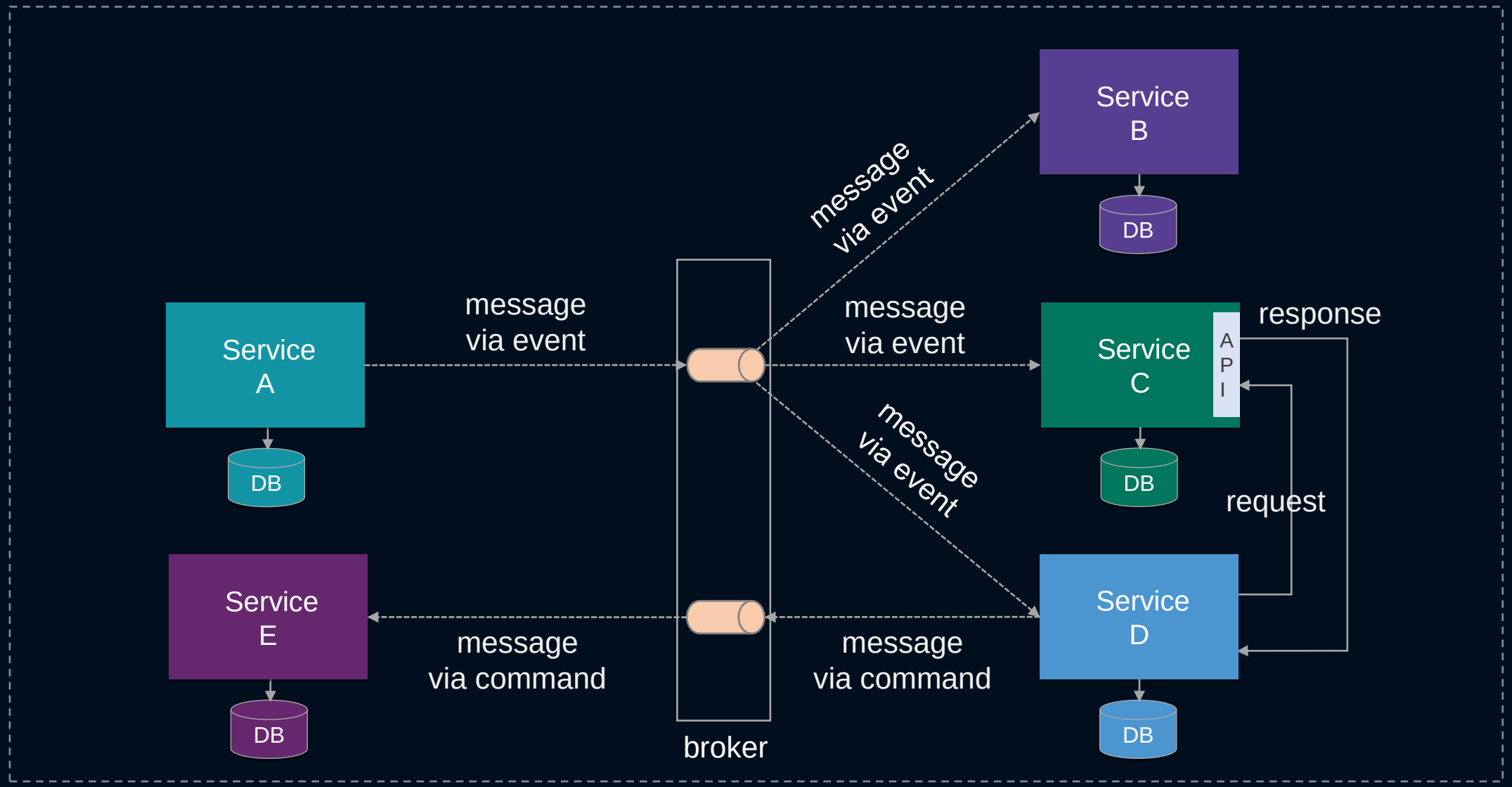
Projeto consiste em duas partes:
base spring-amqp e spring-rabbit

```xml
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-amqp</artifactId>
</dependency>
```

Exemplos de Brokers

MODELO DE COMUNICAÇÃO HÍBRIDA VIA API E MENSAGERIA

Comunicação Híbrida via API e Mensageria



MODELOS DE COMUNICAÇÕES VIA APIS

- Comunicação Síncrona via APIs
- Comunicação Assíncrona via APIs
- Resiliência com Circuit Breaker
- Projetos Spring Web MVC e Webflux

MODELOS DE COMUNICAÇÕES VIA MENSAGERIA

- Comandos versus Eventos
- Comunicação Assíncrona via Mensageria utilizando de Comandos
- Comunicação Assíncrona via Mensageria utilizando de Eventos
- Projeto Spring AMQP
- Exemplos de Brokers

MODELO DE COMUNICAÇÃO HÍBRIDA

- Comunicação híbrida via APIs e Mensageria com Comandos e Eventos

Todo o conteúdo dos slides serão disponibilizados no grupo whatsapp =)

Próximo dia da Decoder Week...

- SAGA PATTERN
- COREOGRAFIA
- ORQUESTRAÇÃO
- API COMPOSITION PATTERN
- MEDIATOR PATTERN
- EVENT DRIVEN
- EVENT NOTIFICATION PATTERN
- EVENT CARRIED STATE TRANSFER
- BROKER PATTERN
- ARQUITETURA HEXAGONAL

E MAIS....