

DIA 02

COMUNICAÇÕES ENTRE MICROSERVICES

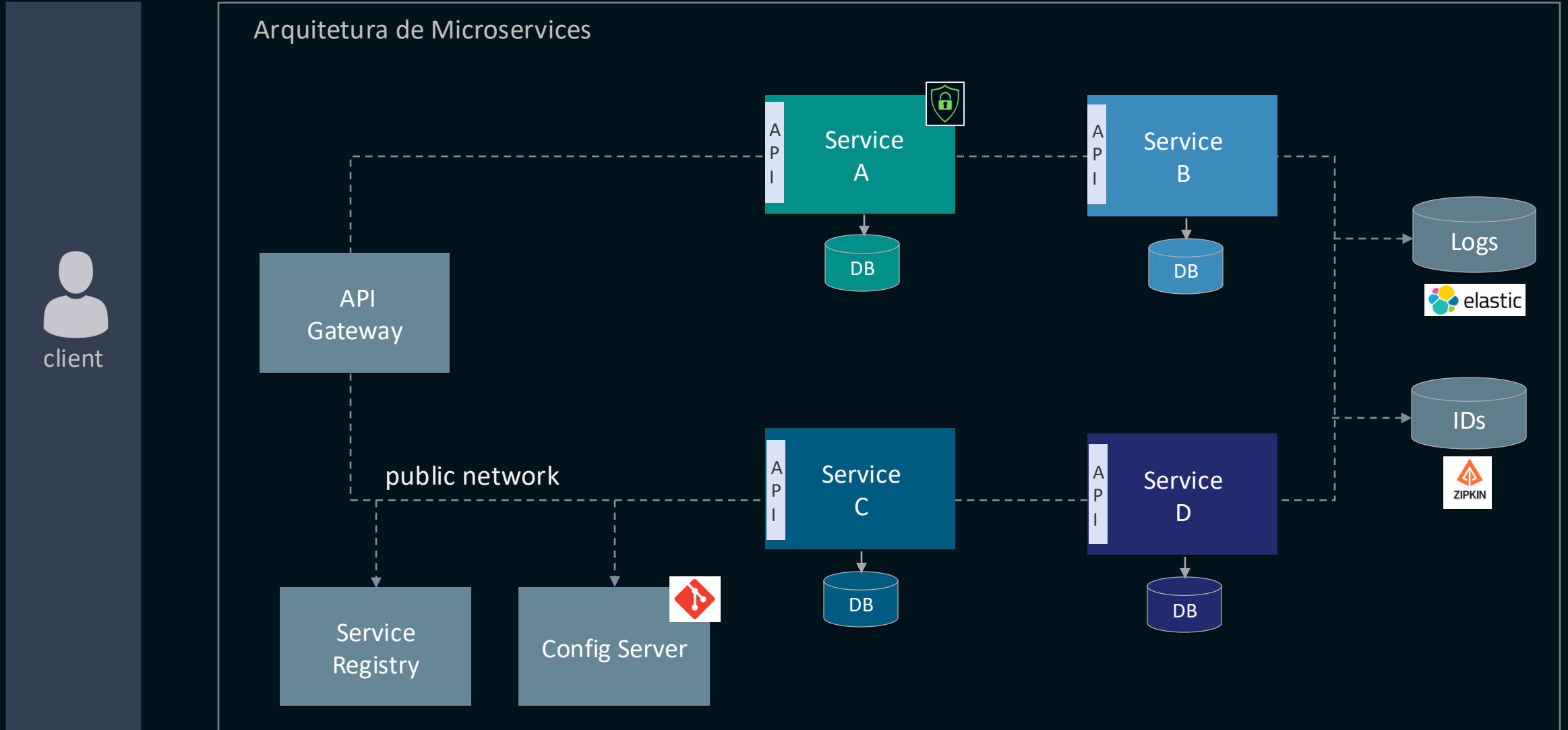


SEMANA
DECODER
com Michelli Brito

COMUNICAÇÕES ENTRE MICROSERVICES E GESTÃO DE DADOS DISTRIBUÍDOS

Dia 02

Arquitetura de Microservices - Abstração



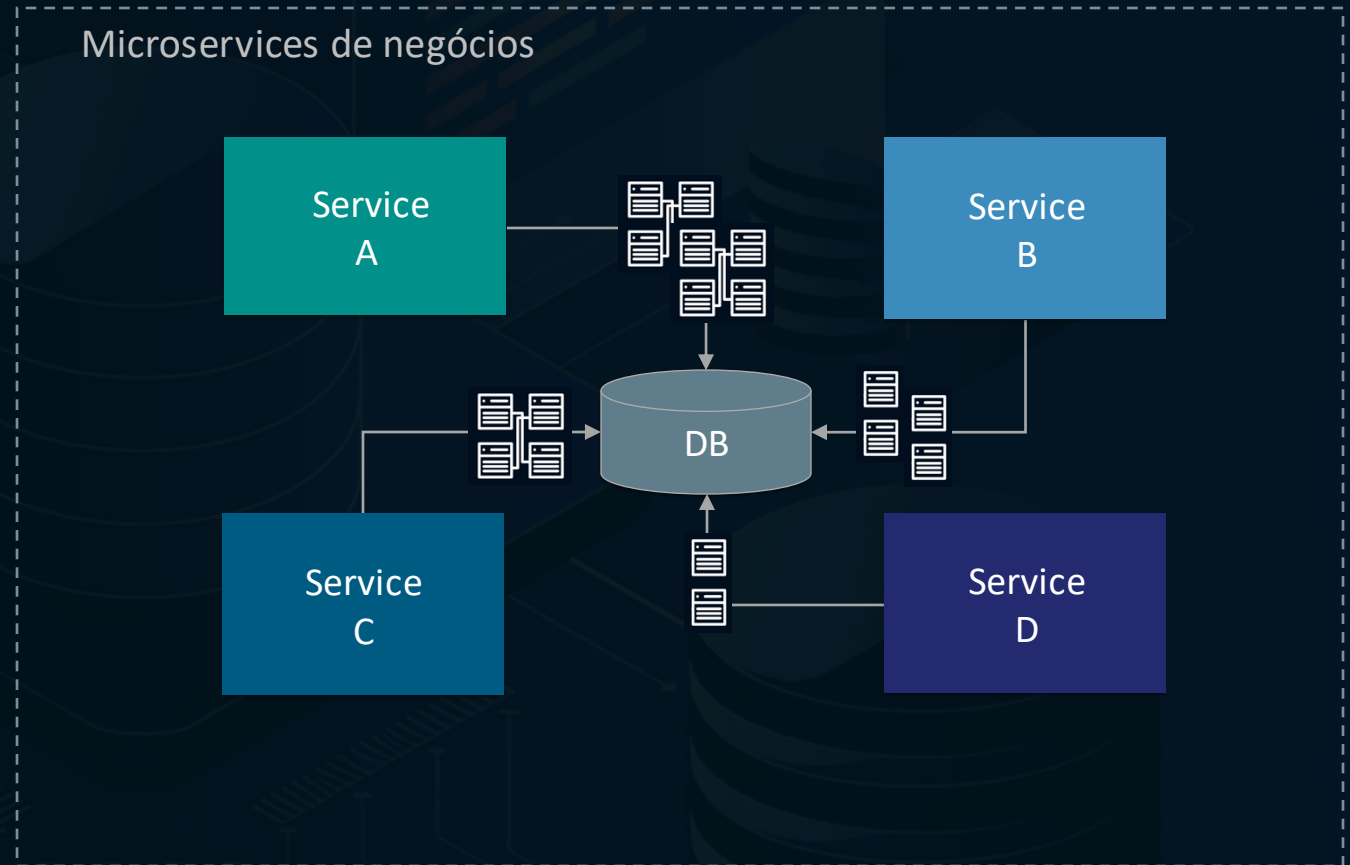
Base de Dados compartilhada ou Base de Dados por microservices

Base de Dados Compartilhada

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

Base de dados compartilhada garante forte consistência.

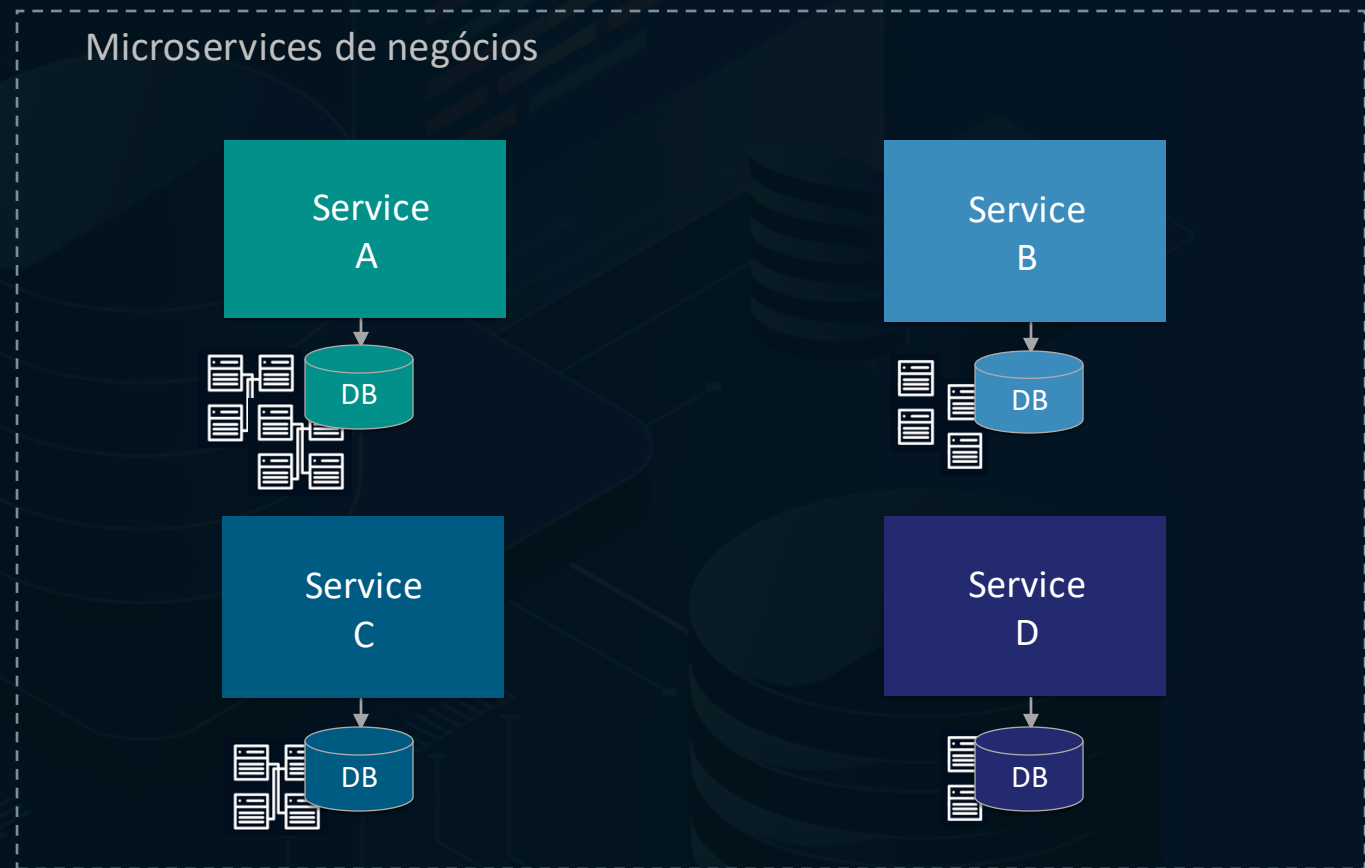
Base de dados compartilhada gera forte acoplamento, sem isolamento da modelagem de dados.



Base de Dados por Microservices (Distribuída)

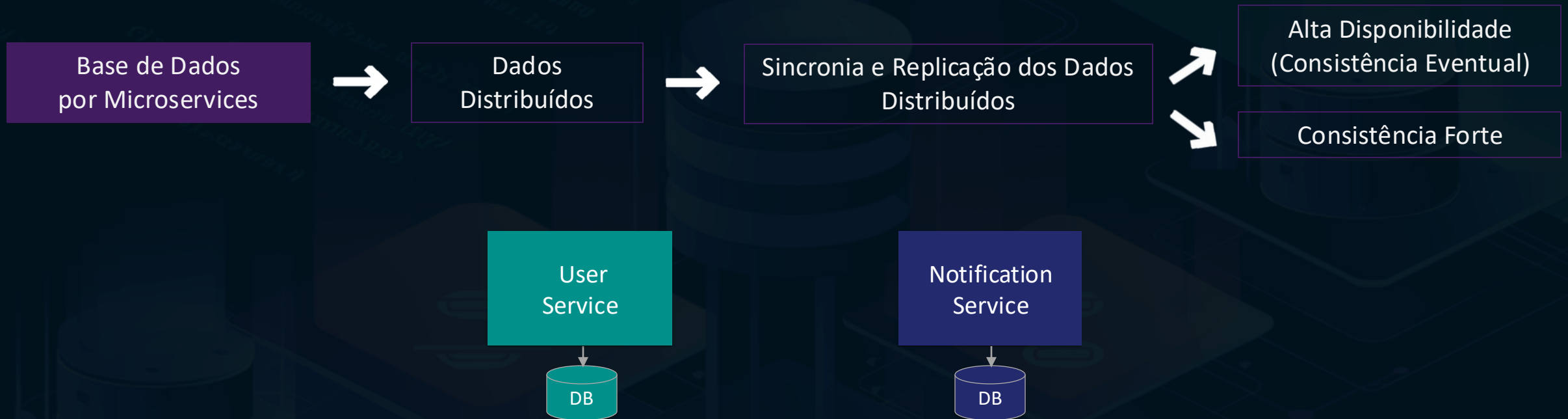
Base de dados por microservices geram menor acoplamento na arquitetura e maior isolamento da modelagem de dados.

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

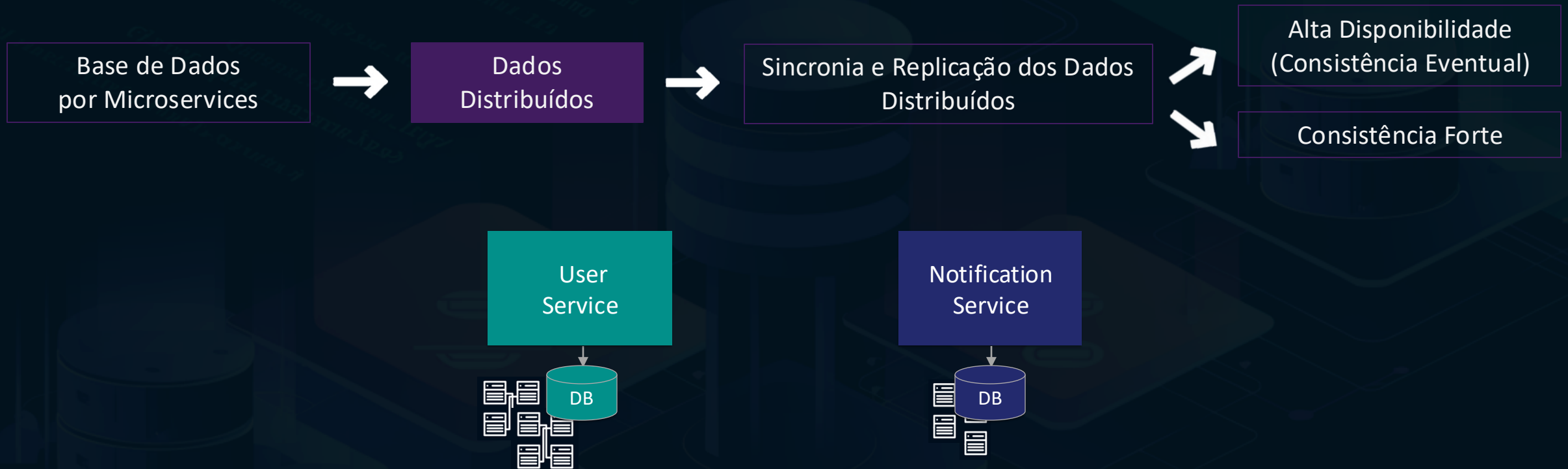


Gestão dos dados distribuídos em Microservices

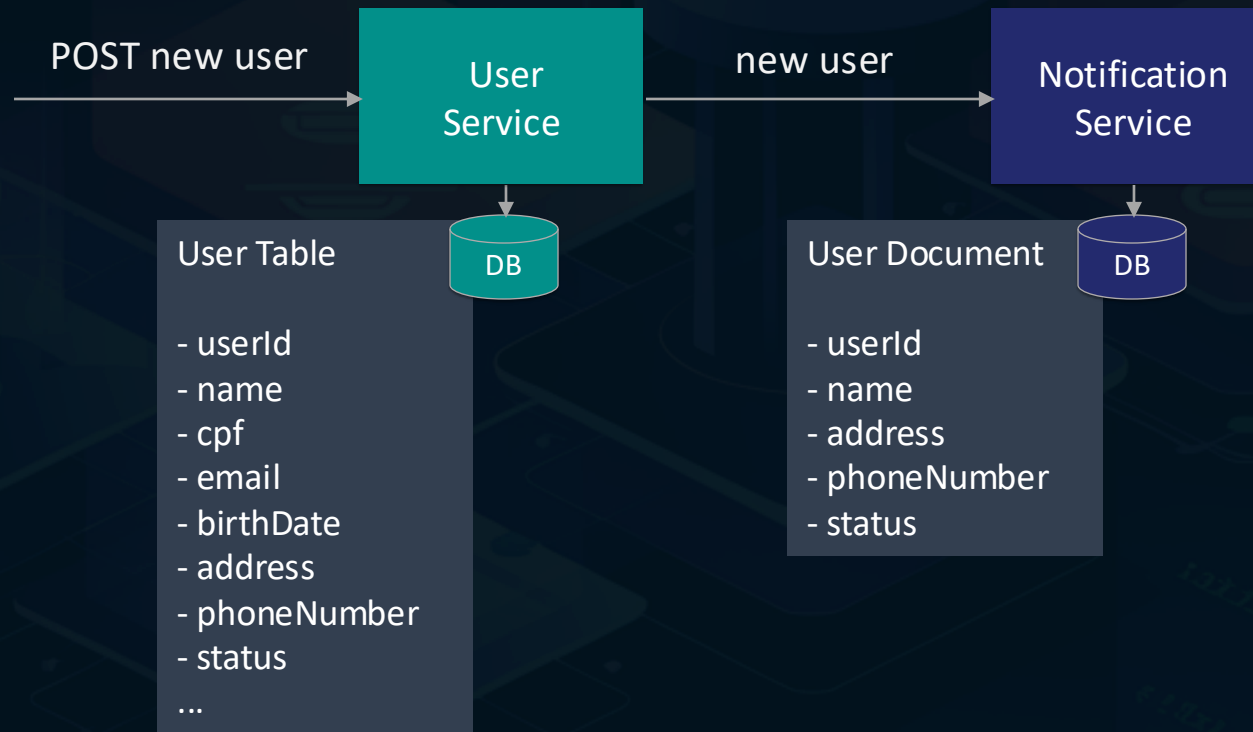
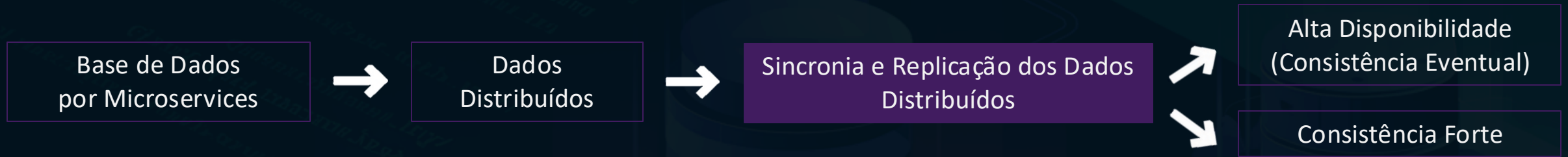
Dados Distribuídos entre Microservices



Dados Distribuídos entre Microservices



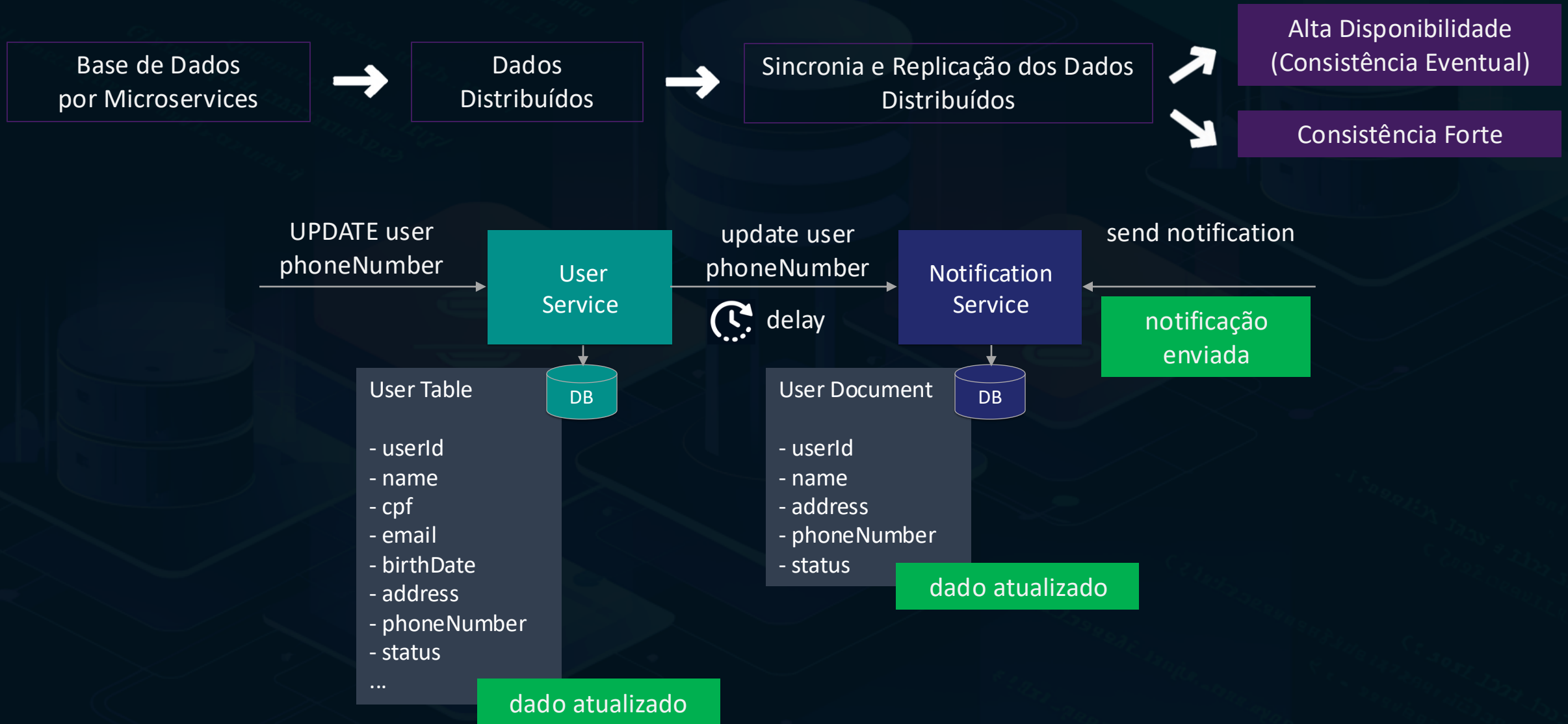
Dados Distribuídos entre Microservices



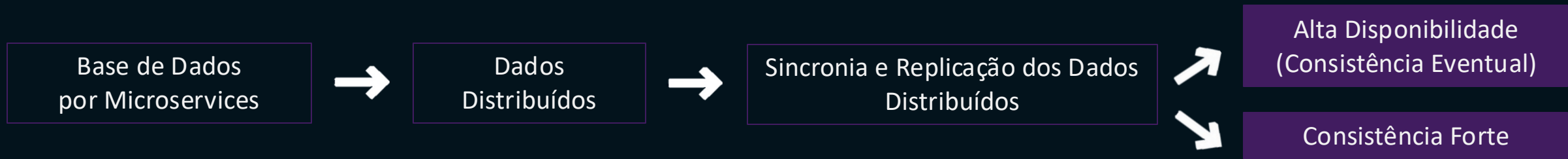
Dados parciais de user replicados no Microservice B pela necessidade do negócio, buscando maior disponibilidade.

Podemos utilizar do Event Carried State Transfer Pattern para realizar a replicação dos dados pela arquitetura.

Dados Distribuídos entre Microservices



Teorema CAP



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**.



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
 30	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

Comunicações
via APIs

Comunicações
via Mensageria

Comunicação
Híbrida

Comunicação Síncrona via APIs

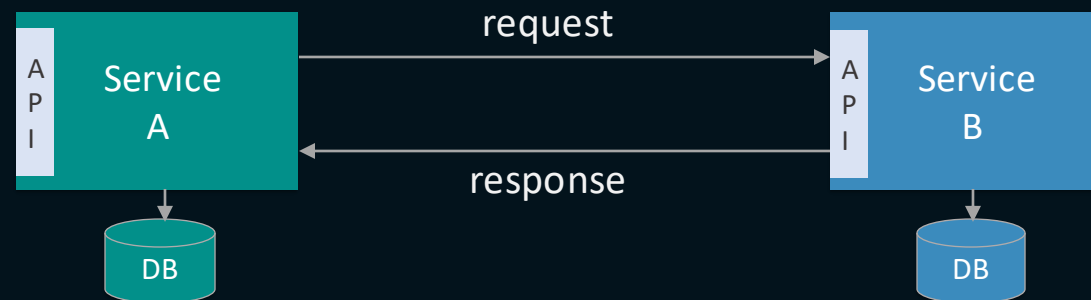
Pontos positivos

- Forte consistência

Pontos de atenção

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

Modelo A – request / response

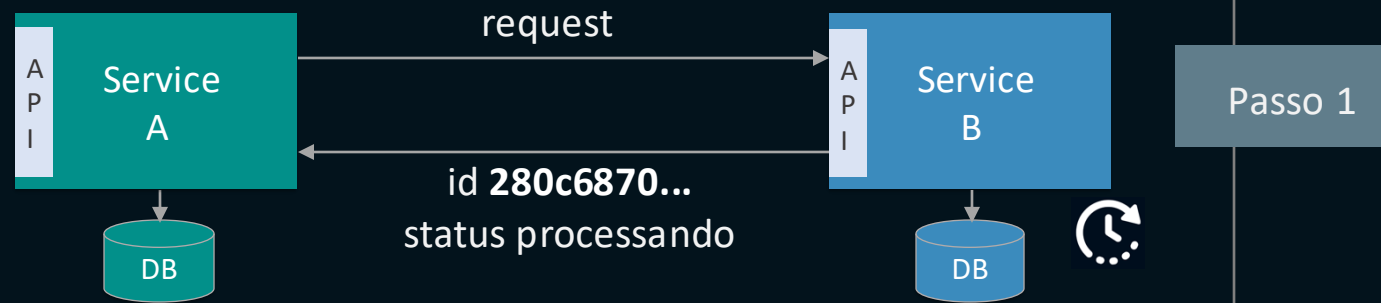


Comunicação síncrona utilizando de métodos HTTP: GET, POST, PUT, DELETE...

Modelo que requer o Registry Discovery Pattern, Distributed Tracing Pattern, Circuit Breaker Pattern...

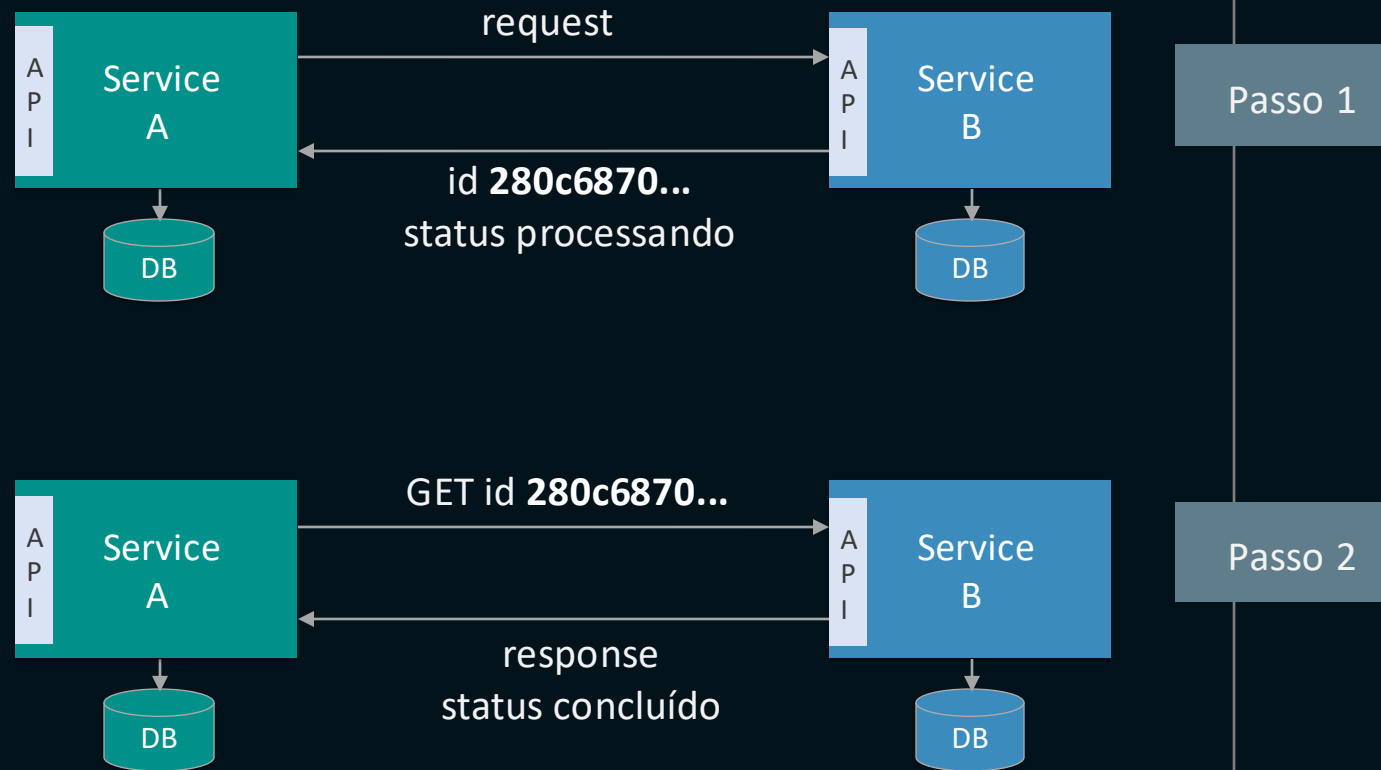
Comunicação Assíncrona via APIs

Modelo B – request / async response



Comunicação Assíncrona via APIs

Modelo B – request / async response



Pontos positivos

- Maior disponibilidade
- Comunicação não bloqueante

Pontos de atenção

- Aumento acoplamento entre os Microservices



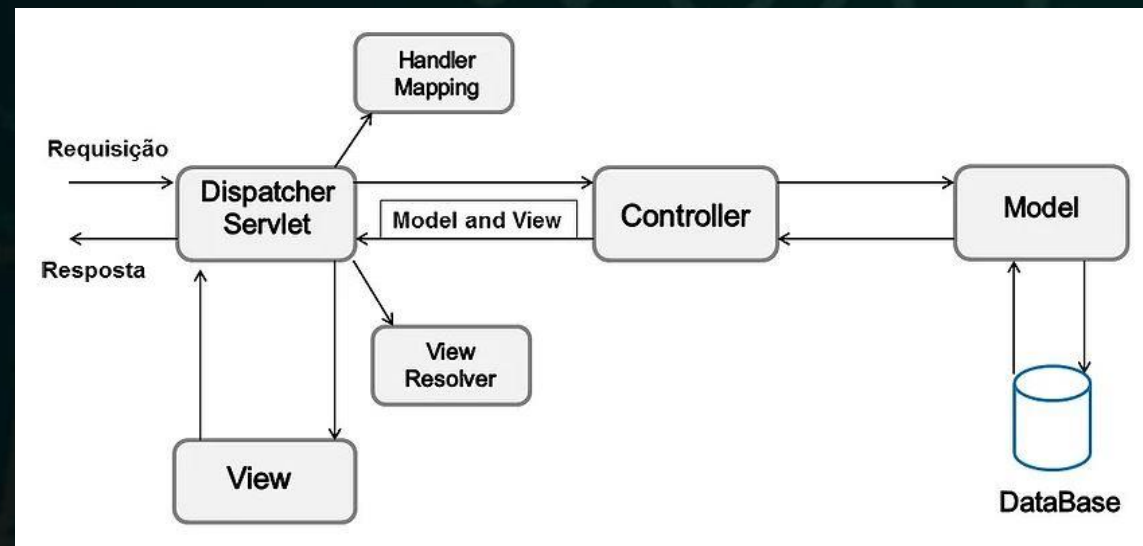
APIs com Spring Web MVC

Conhecido como Spring MVC

Módulo fonte: spring-webmvc

Utiliza de um controlador frontal:
Dispatcher Servlet

Servidor Tomcat





APIs Reativas com Spring Web Webflux

Inserido na versão 5 do Spring Framework

Não bloqueante

Baseado no projeto Reactor

Servidor Netty

Criação de APIs Reativas

Trabalha com os tipos Flux e Mono

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

APIs com ecossistema Spring

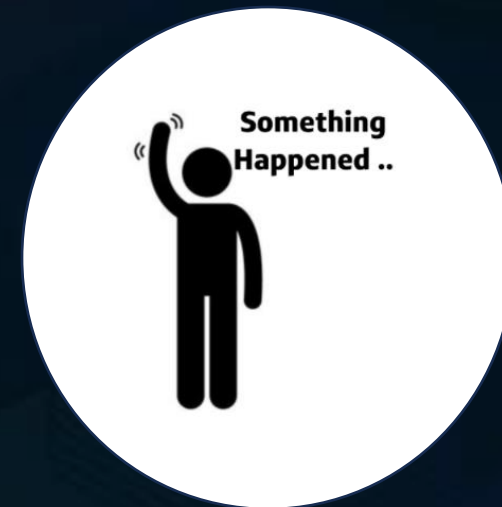


Mensageria via Comandos e Eventos

Command Message



Event Message

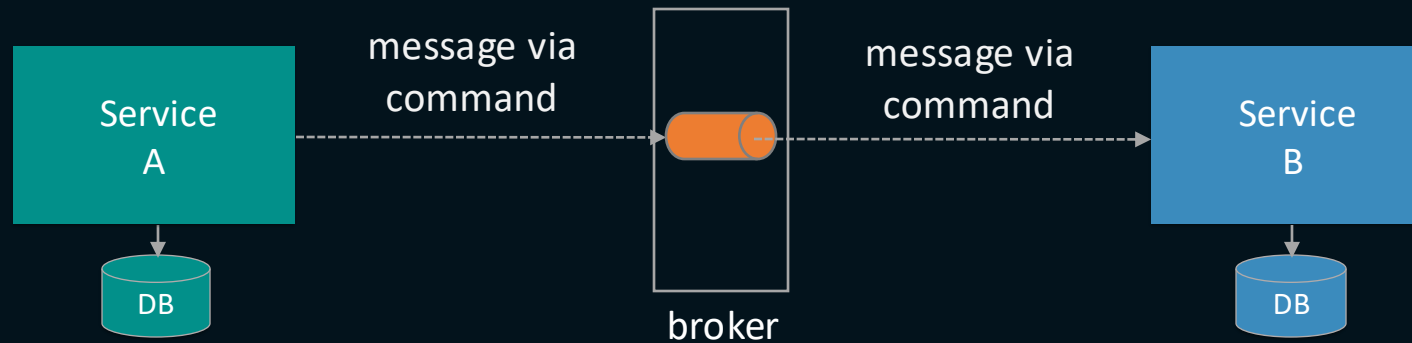


Comunicação Assíncrona via Mensageria Comandos

Pontos positivos

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

Modelo A - One way



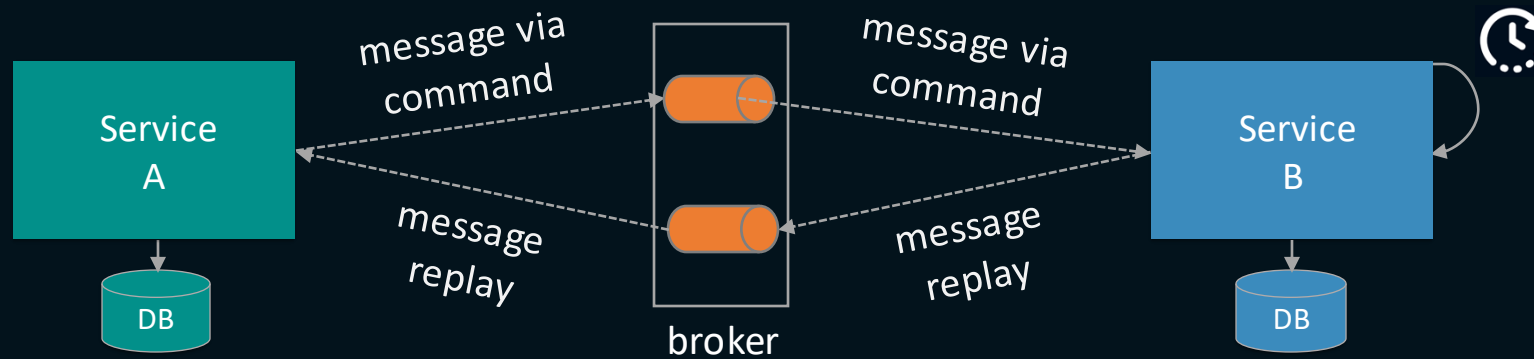
Modelo utilizado em Broker Pattern e Mediator Pattern, por exemplo.

Dia 3

Microservices Patterns e suas aplicabilidades.

Comunicação Assíncrona via Mensageria Comandos

Modelo B - request / async response

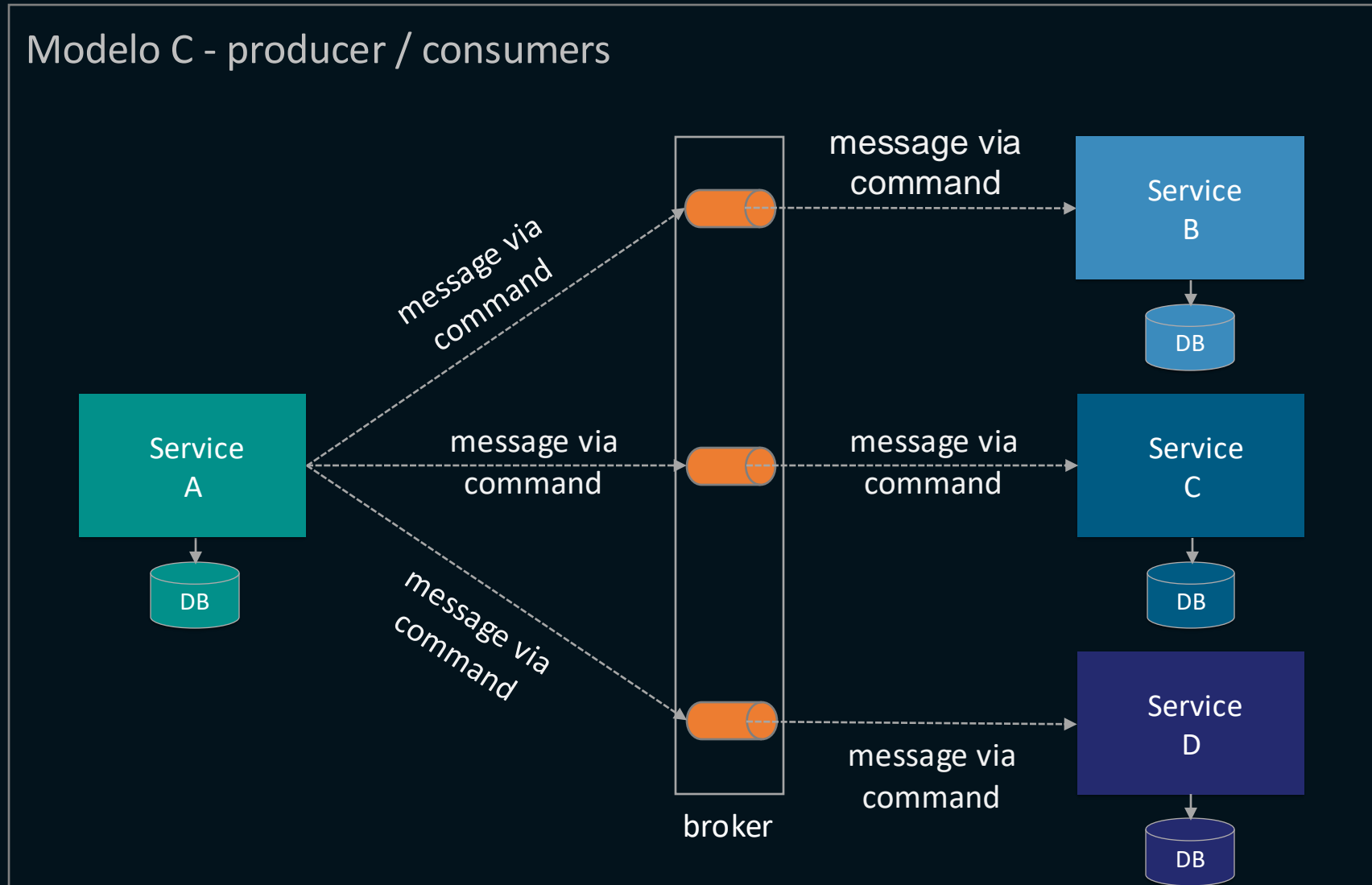


Pontos positivos

- Maior disponibilidade
- Comunicação não bloqueante

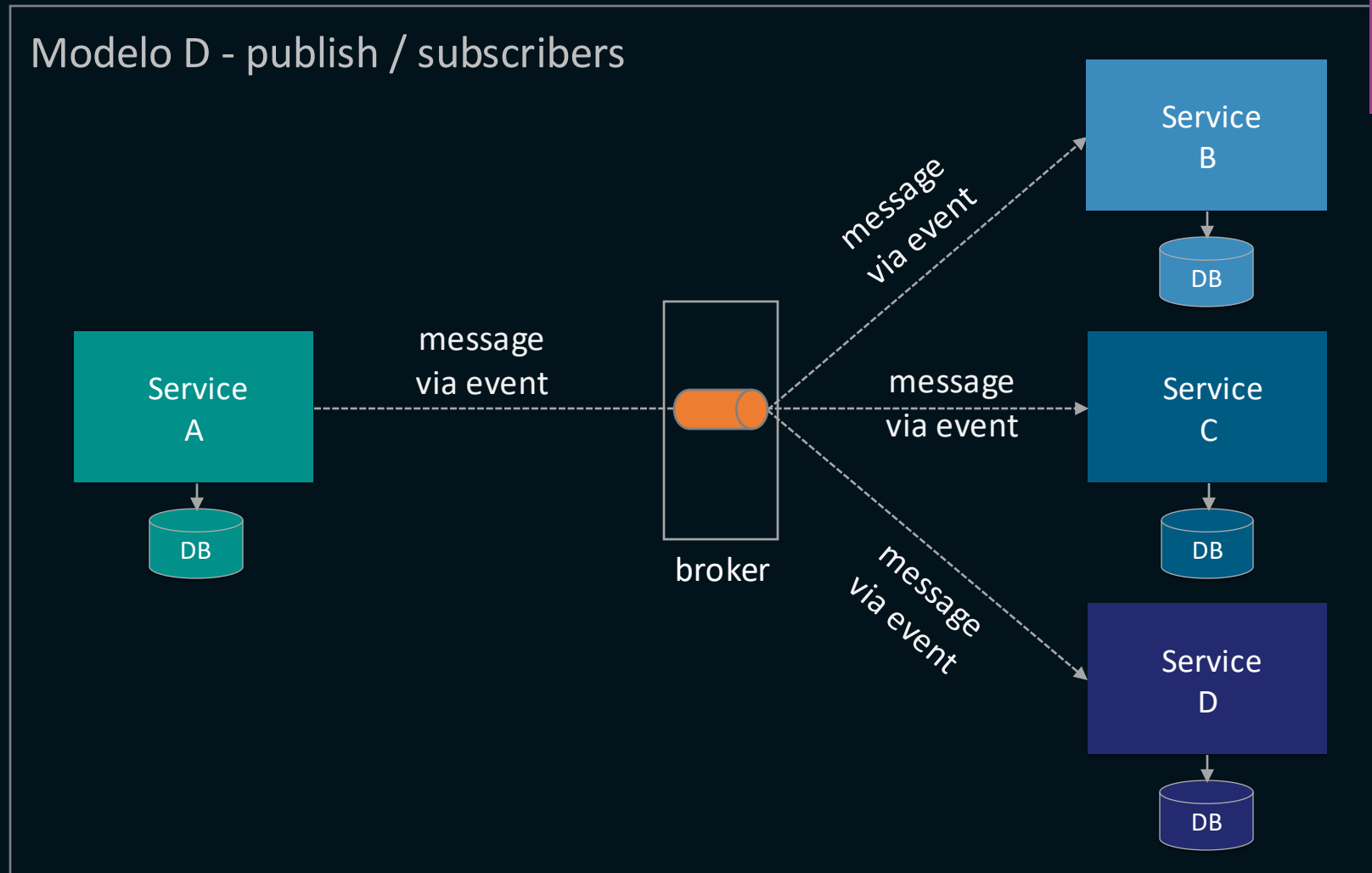
Modelo utilizado em Saga Pattern com Orquestração, por exemplo.

Comunicação Assíncrona via Mensageria Comandos



Modelo utilizado em Saga Pattern com Orquestração, por exemplo.

Comunicação Assíncrona via Mensageria Eventos



Dia 3

Microservices Patterns
e suas aplicabilidades.

Pontos positivos

- Menor acoplamento
ainda

Modelo utilizado em Saga
Pattern com Coreografia,
Event Notification e Event
State Transfer...



Mensageria com Spring AMQP

Suporte a mensagens utilizando protocolo AMQP

Enviar e receber mensagens

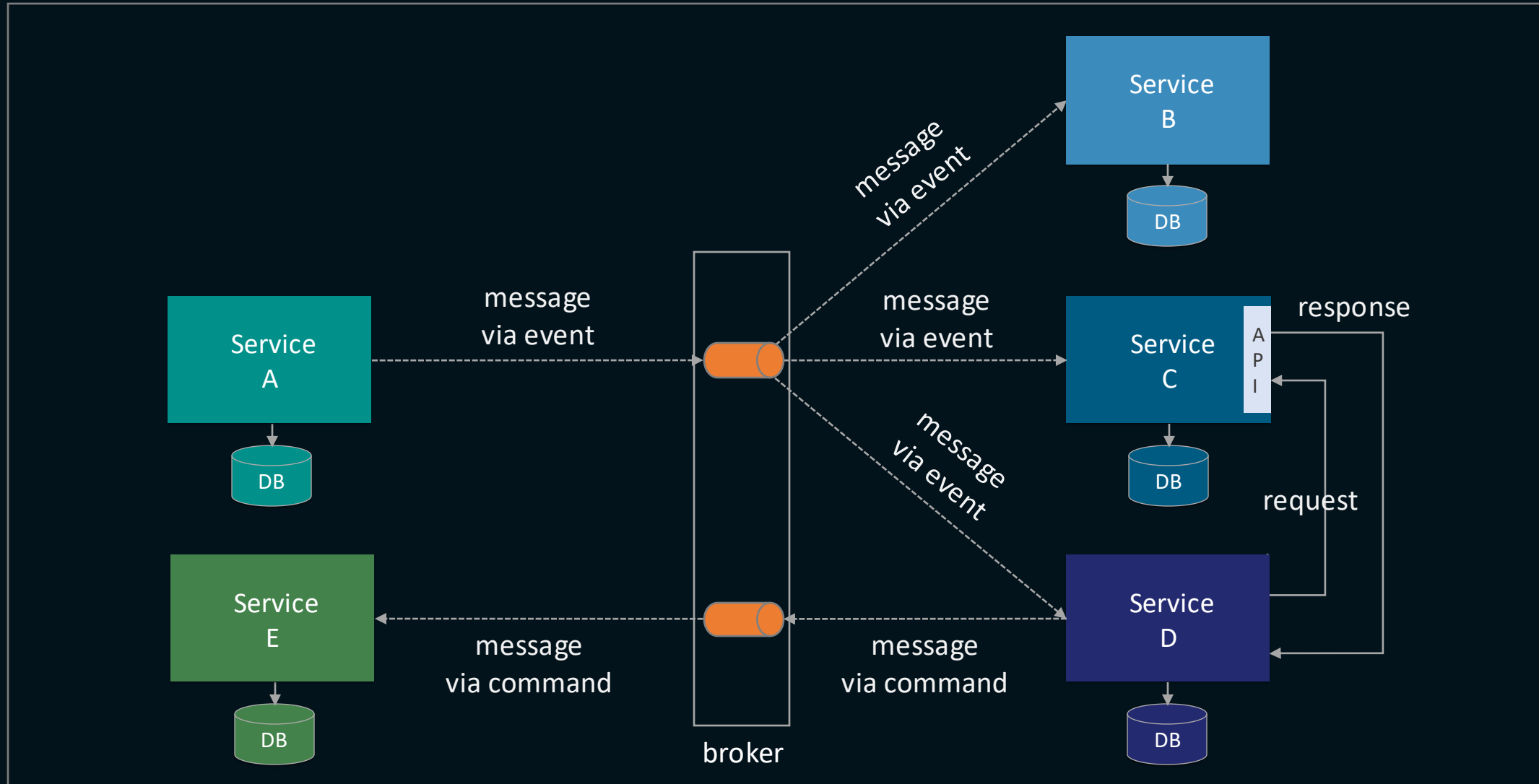
Definir automaticamente queues, exchanges e routing keys

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

Exemplos de Brokers



Comunicação Híbrida via API e Mensageria



Modelos de Comunicações via APIs

- Comunicação Síncrona via APIs
- Comunicação Assíncrona via APIs
- Projeto Spring Web MVC
- Projeto Spring Webflux

Modelos de Comunicações via Mensageria

- Comandos e 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 conteúdo dos slides
serão disponibilizados no
grupo whatsapp.

Prévia do próximo dia Semana Decoder

Broker Pattern e Mediator Pattern

API Composition Pattern

Event Notification Pattern

Registry Discovery Pattern

Event-Carried State Transfer Pattern

Circuit Breaker Pattern

Saga Pattern com Orquestração

Distributed Tracing Pattern

Saga Pattern com Coreografia

Ports and Adapters Pattern