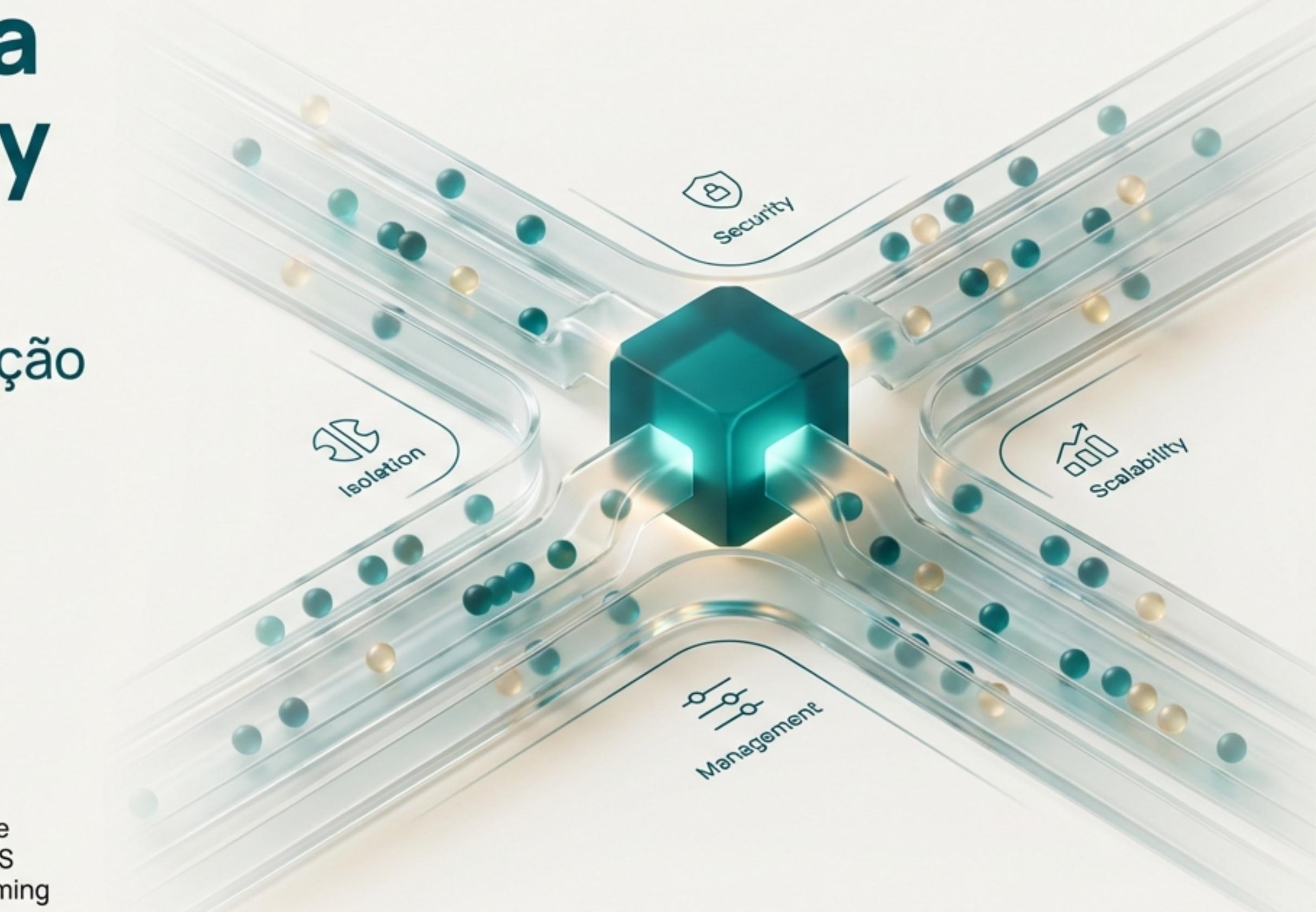


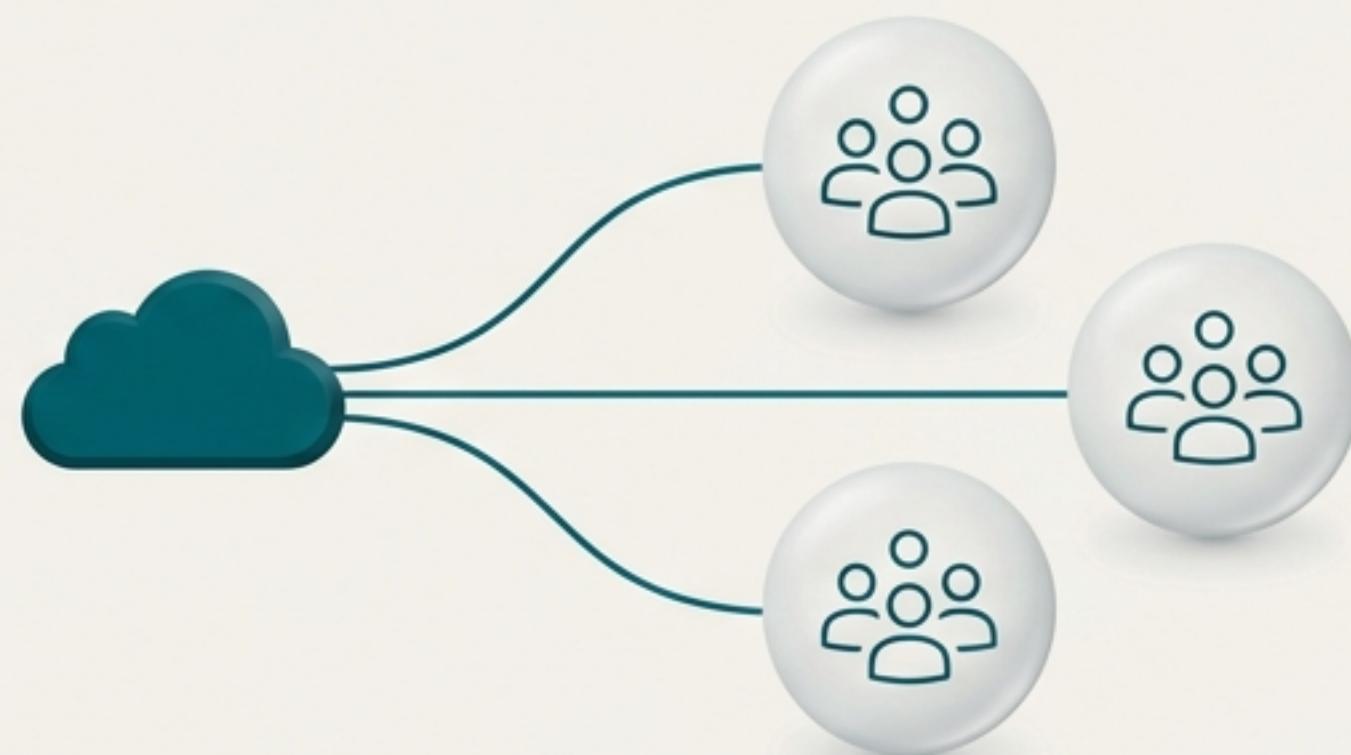
Dominando a Multitenancy no CAP Java

Da Arquitetura à Operação
em Produção



Um guia técnico visual para desenvolvedores que buscam implementar e gerenciar aplicações SaaS robustas com o SAP Cloud Application Programming Model.

O Fundamento: Aplicações CAP como Software as a Service (SaaS)



A multitenancy permite que uma única instância da sua aplicação sirva múltiplos clientes (tenants), cada um operando de forma completamente isolada, como se tivessem sua própria versão do software.



Isolamento de Dados

Cada tenant possui seu próprio container de banco de dados, garantindo a privacidade e a segurança.



Customização

Opcionalmente, cada tenant pode estender seu próprio modelo de dados (CDS), permitindo flexibilidade sem alterar o código base da aplicação.

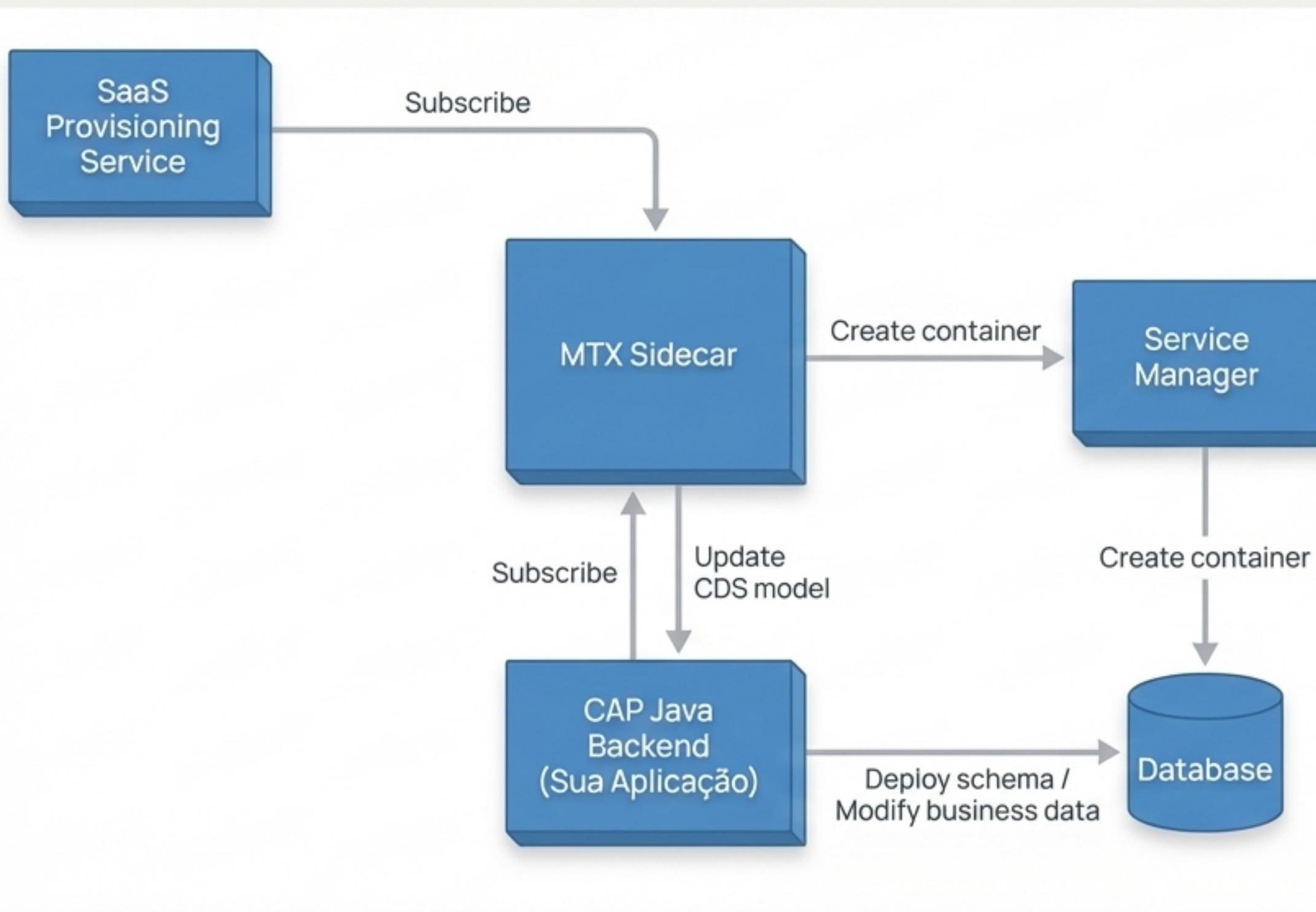


Eficiência

Um único backend gerenciado para múltiplos clientes.

Para viabilizar isso de forma transparente, o CAP Java se integra a um conjunto de serviços de multitenancy. Vamos ver a arquitetura.

A Arquitetura da Multitenancy no CAP Java



SaaS Provisioning Service:

A plataforma (ex: SAP BTP) que gerencia o ciclo de vida das assinaturas dos tenants. Ela dispara os eventos.

MTX Sidecar:

O "motor" da multitenancy. Suas responsabilidades principais são:

- Criar ou remover containers de banco de dados para os tenants.
- Gerenciar extensões de modelo CDS específicas de cada tenant.

CAP Java Backend:

Sua aplicação. É aqui que você implementa a lógica de negócios e, crucialmente, intercepta os eventos do ciclo de vida para adicionar comportamento personalizado.

Service Manager:

O serviço da plataforma responsável por provisionar recursos, como instâncias de banco de dados.

Seu Ponto de Controle: Os Eventos do `DeploymentService`

A plataforma (SaaS Provisioning) envia requisições de ciclo de vida (ex: novo tenant). O CAP Java traduz internamente essas requisições em eventos em um serviço técnico chamado `DeploymentService`.

Você só precisa registrar **event handlers** para estes eventos se precisar **sobrescrever** ou **estender o comportamento padrão**.



‘SUBSCRIBE’

Adicionar um novo tenant.



‘UNSUBSCRIBE’

Remover um tenant existente.



‘DEPENDENCIES’

Declarar dependências de serviços externos.



‘UPGRADE’

Disparar a atualização do modelo de dados.

O Nascimento do Tenant: Customizando o Evento `SUBSCRIBE`

Comportamento Padrão

Na fase `@On`, o CAP cria automaticamente um novo container de banco de dados para o tenant via Service Manager.

Quando Intervir?

- Para executar ações **antes** da criação do container (ex: logs, validações).
- **Cenário Crítico:** Quando há mais de uma instância de SAP HANA no seu space, você **precisa** especificar qual usar.

Nota Importante

Handlers customizados na fase `@On` não devem chamar `setCompleted()`, pois o framework gerencia o ciclo.

Como Intervir

```
// Handler para especificar o ID do banco de dados  
@Before  
public void beforeSubscription(SubscribeEventContext context) {  
    context.getOptions().put("provisioningParameters",  
        Collections.singletonMap("database_id", "<database ID>"));  
}
```

Execute sua lógica **antes** da fase principal do evento.

Aqui você injeta o `database_id` necessário para o provisionamento no ambiente correto.

A Despedida do Tenant: Controlando o Evento 'UNSUBSCRIBE'

Comportamento Padrão

Na fase `@On`, o container de banco de dados do tenant é excluído.

Quando Intervir?

- Para enviar notificações ou executar lógicas de limpeza *antes* (`@Before`) ou *depois* (`@After`) da exclusão.
- **Cenário Avançado:** Para **impedir** a exclusão dos recursos, por exemplo, se um tenant tem múltiplas assinaturas ativas.



Ao acessar dados do tenant antes da exclusão, envolva o acesso em um `'ChangeSetContext'` ou transação para garantir que a transação seja concluída antes da remoção do container.

Como Intervir (Exemplos)

Exemplo 1: Notificação

```
@After  
public void afterUnsubscribe(UnsubscribeEventContext context) {  
    // Notificar que o offboarding foi concluído  
}
```

Exemplo 2: Impedir a Exclusão

```
@Before  
public void beforeUnsubscribe(UnsubscribeEventContext context) {  
    if (keepResources(context.getTenant())) {  
        context.setCompleted(); // Pula a fase @On (exclusão)  
    }  
}
```

Chamar `'setCompleted()'` encerra o processamento do evento, efetivamente pulando a lógica de exclusão padrão.



Mapeando o Ecossistema: O Evento `DEPENDENCIES`

A plataforma pode perguntar à sua aplicação: "De quais outros serviços você depende?" Isso é feito através do callback `getDependencies`, que dispara este evento.

Quando Implementar?

- **Obrigatório:** Se sua aplicação consome serviços de reuso da SAP (ex: um serviço de faturamento, logística, etc.). Você precisa retornar o `xsappname` desses serviços.
 - **Automático:** O CAP já adiciona automaticamente as dependências de serviços para os quais possui integrações dedicadas.
- ⓘ A localização exata do campo `xsappname` nas credenciais pode variar dependendo do serviço consumido.

Como Implementar

```
@On
public void onDependencies(DependenciesEventContext context) {
    List<Map<String, Object>> dependencies = new ArrayList<>();
    // 1. Encontra o service binding do serviço dependente
    Optional<ServiceBinding> service = cdsRuntime.getEnvironment()
        .getServiceBindings()...
    if (service.isPresent()) {
        // 2. Extrai o xsappname das credenciais
        String xsappname =
            extractXsappname(service.get().getCredentials());
        // 3. Adiciona à lista de resultados
        dependencies.add(SaaSRegistryDependency.create(xsappname));
    }
    context.setResult(dependencies);
}
```

Gerenciamento em Escala: Atualizando o Schema para Todos os Tenants

O Desafio

Ao lançar uma nova versão da sua aplicação com um modelo CDS atualizado, como garantir que o schema do banco de dados de **cada tenant inscrito** seja atualizado de forma consistente?



A Solução CAP

A atualização do schema precisa ser **disparada explicitamente**.

O Mecanismo

- Quando o processo de atualização é iniciado, o CAP emite o evento `UPGRADE`.
- **Comportamento Padrão:** Por default, o CAP Java SDK notifica o `MTX Sidecar` para realizar a atualização do schema, se necessário.

A seguir, veremos a ferramenta recomendada para disparar essa atualização para todos os tenants de forma segura, visando um deploy com zero downtime.

Ferramenta Operacional: O `Deploy Main Method`

Uma classe `main` (`com.sap.cds.framework.spring.utils.Deploy`) que pode ser executada pela linha de comando para atualizar todos os tenants (ou uma lista específica).

Estratégia de Uso: Execute esta ferramenta **com a aplicação Java parada**, antes de iniciar a nova versão. Isso evita que o novo código acesse um schema de banco de dados antigo.

Pré-requisitos

1. O `MTX Sidecar` deve estar em execução.
2. A propriedade `cds.multitenancy.component-scan` deve ser configurada com o package da sua aplicação para que todos os handlers sejam registrados corretamente.

Códigos de Saída

0 = Sucesso | 1 = Falha em pelo menos um tenant.

Como Executar

Localmente:

```
java -cp <jar-file> -Dloader.main=com.sap.cds.framework.spring.utils.Deploy [tenantId1] [tenantId2] ...
```

No SAP BTP (Cloud Foundry):

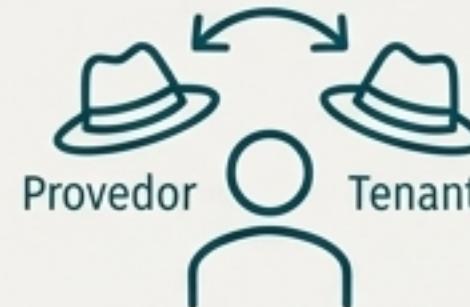
Recomenda-se usar **CF Tasks**. O comando precisa ser adaptado a partir do comando de start gerado pelo buildpack.

```
cf run-task <application_name> "<command>"
```

Técnica Avançada: Acessando Dados de Outros Tenants Programaticamente

O Poder: A API `RequestContextRunner` permite que você "troque de chapéu" e execute código no contexto de outro tenant.

Casos de Uso Comuns



- Acessar dados de configuração armazenados no **tenant provedor** enquanto processa uma requisição de um **tenant de negócio**.
- Acessar dados de um tenant específico em **jobs assíncronos**, onde não há um contexto de requisição inicial.



- **Performance:** Trocar de contexto é uma **operação cara**, pois pode exigir a busca de metadados do `MTX Sidecar`. Evite fazer isso em loops para todos os tenants.
- **Segurança:** A aplicação é **responsável por garantir a privacidade e o isolamento dos dados** ao desviar do comportamento padrão.

Como Fazer

Para o Tenant Provedor:

```
runtime.requestContext().systemUserProvider().run(context -> {  
    // Seu código executa como provedor  
});
```

Para um Tenant Específico:

```
runtime.requestContext().systemUser(tenant).run(context -> {  
    // Seu código executa no contexto do 'tenant' especificado  
});
```

Técnica Avançada: Enumerando os Tenants Inscritos

A Necessidade:

Em cenários de administração ou processamento em lote, pode ser necessário obter uma lista de todos os tenants ativos.

A Ferramenta:

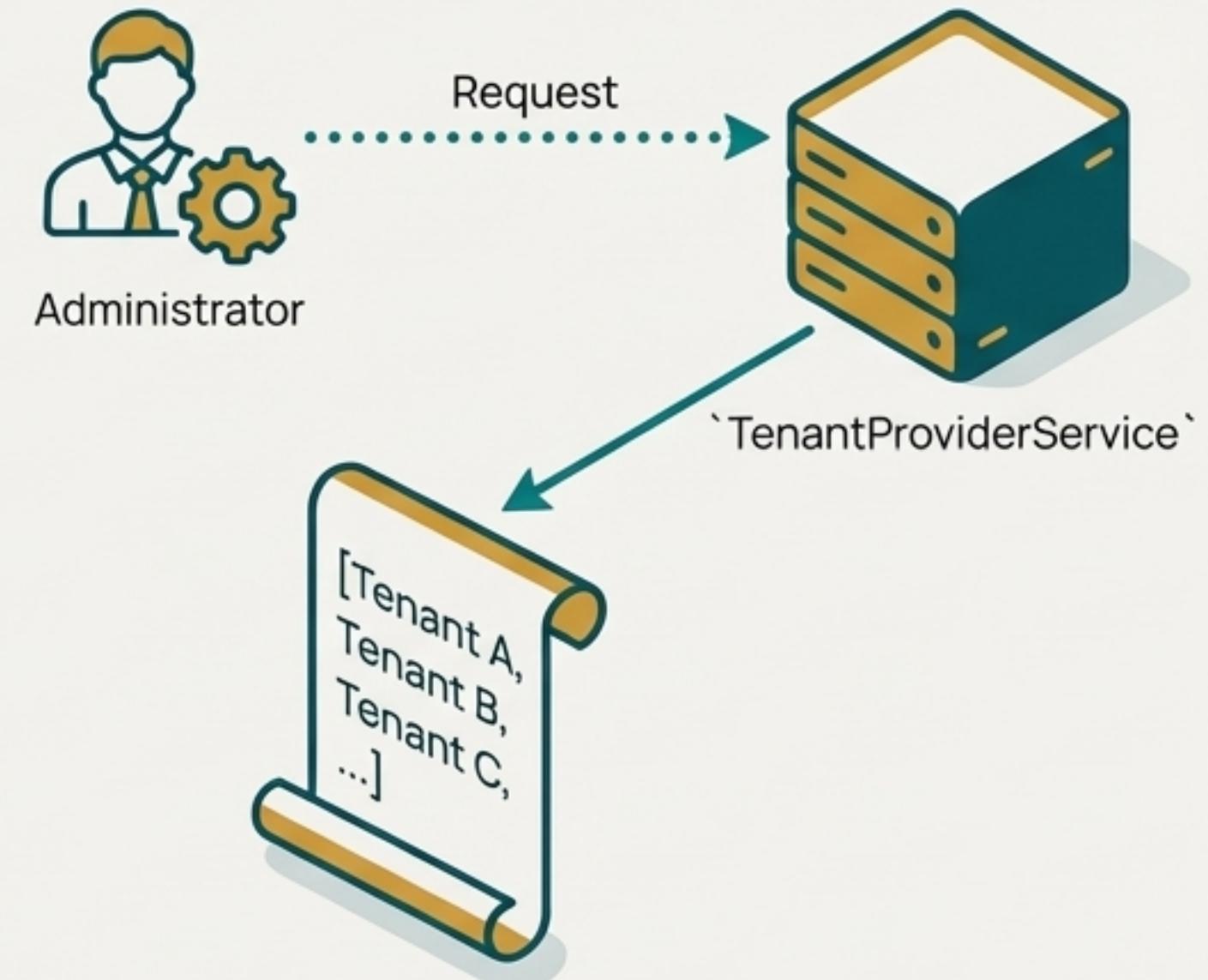
O `TenantProviderService` foi projetado para isso.

Como Usar:

Injeção e Uso:

```
@Autowired TenantProviderService tenantProvider;  
...  
List<TenantInfo> tenantInfo = tenantProvider.readTenants();
```

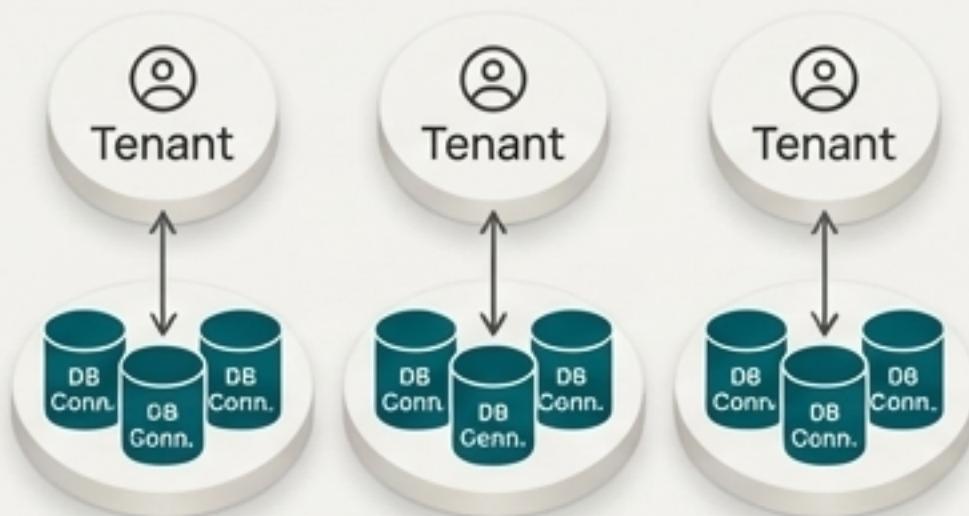
⚠ Recuperar a lista de todos os tenants é uma operação cara. É uma boa prática **cachear os resultados** se eles forem usados com frequência.



Otimização: O Dilema do Pooling de Conexões de Banco de Dados

Recursos (memória, conexões de DB) vs. Latência (tempo de resposta para o usuário).

Abordagem 1: Pool por Tenant (Padrão)

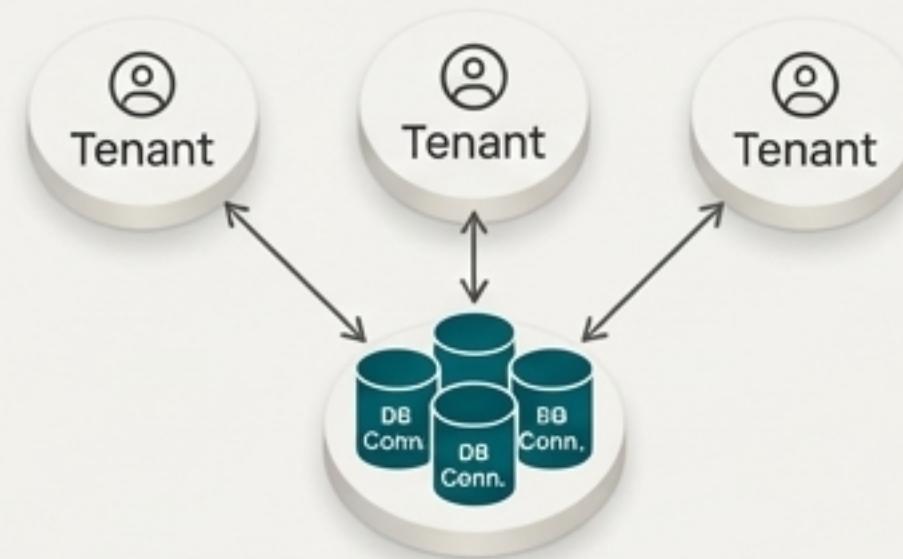


Como Funciona: Um pool de conexões dedicado e estático para cada tenant.

Vantagens: **Menor Latência.** As conexões estão sempre prontas para uso.

Desvantagens: **Maior Consumo de Recursos.** O número de conexões cresce linearmente com o número de tenants.

Abordagem 2: Pool Combinado



Como Funciona: Um único pool de conexões compartilhado por todos os tenants.

Vantagens: **Menor Consumo de Recursos.** O número de conexões é fixo.

Desvantagens: **Maior Latência.** Cada conexão precisa ser 'trocada' para o schema correto do tenant antes do uso.

Como Ativar o Pool Combinado: Defina a propriedade: `cds.multiTenancy.dataSource.combinePools.enabled = true`

⚠️ Com o pool combinado, um tenant pode esgotar todas as conexões (intencionalmente ou não). Considere implementar medidas de mitigação, como *rate-limiting*.

Ajuste Fino: Dimensionamento Dinâmico de Pools de Conexão

O Problema

Por padrão, os pools usam uma estratégia de dimensionamento estático, mantendo conexões abertas mesmo sem uso. Com muitos tenants, isso pode levar a problemas de recursos.

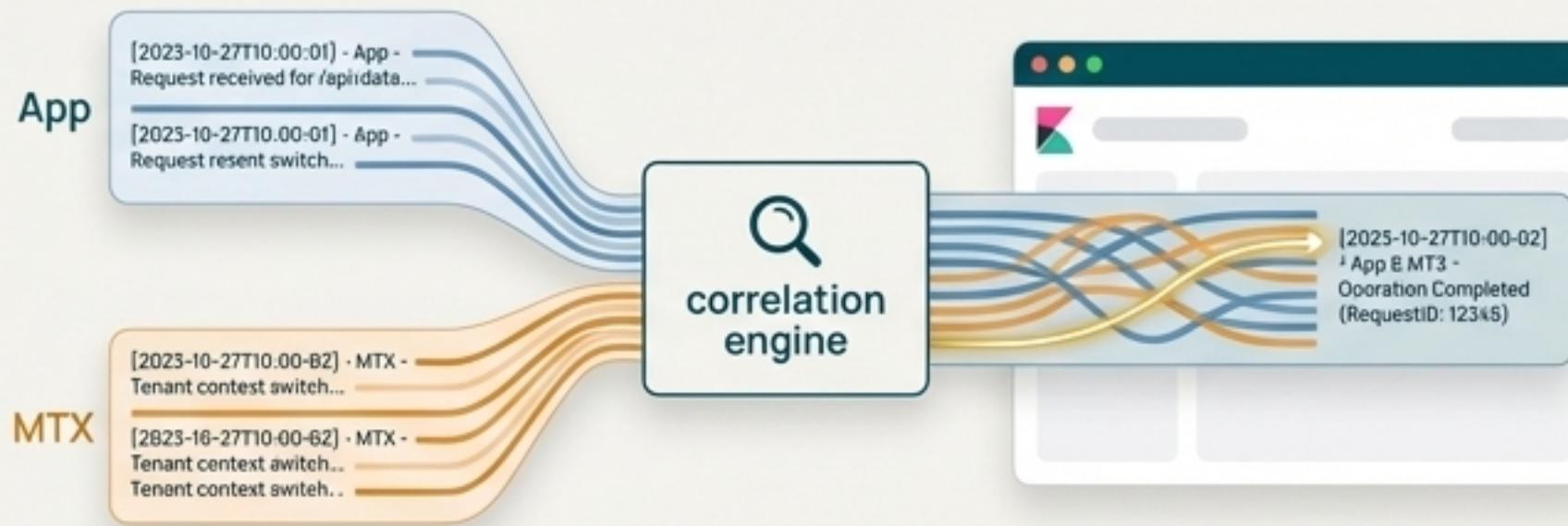
A Solução

Configure o HikariCP para um comportamento dinâmico, ajustando o tamanho do pool com base na carga real.

Parâmetros de Configuração Chave (`cds.dataSource.<service-instance>.hikari.*`):

minimum-idle	maximum-pool-size	idle-timeout
<p>Descrição: Número mínimo de conexões ociosas a serem mantidas.</p> <p>Recomendação: Para economizar recursos, mantenha este valor baixo (ex: 1).</p>	<p>Descrição: Número máximo de conexões no pool.</p> <p>Recomendação: Monitore sua aplicação sob carga para encontrar o valor ideal. O padrão (10) é um bom ponto de partida.</p>	<p>Descrição: Tempo após o qual uma conexão ociosa pode ser removida do pool (até atingir <code>minimum-idle</code>).</p> <p>Recomendação: Não defina um valor muito baixo para evitar a latência de abrir novas conexões constantemente.</p>

Operação em Produção: Logging e Configuração



Logging e Observabilidade

O Benefício: O suporte de logging do CAP permite que você tenha requisições **correlacionadas** através dos diferentes componentes (sua aplicação e o MTX Sidecar).

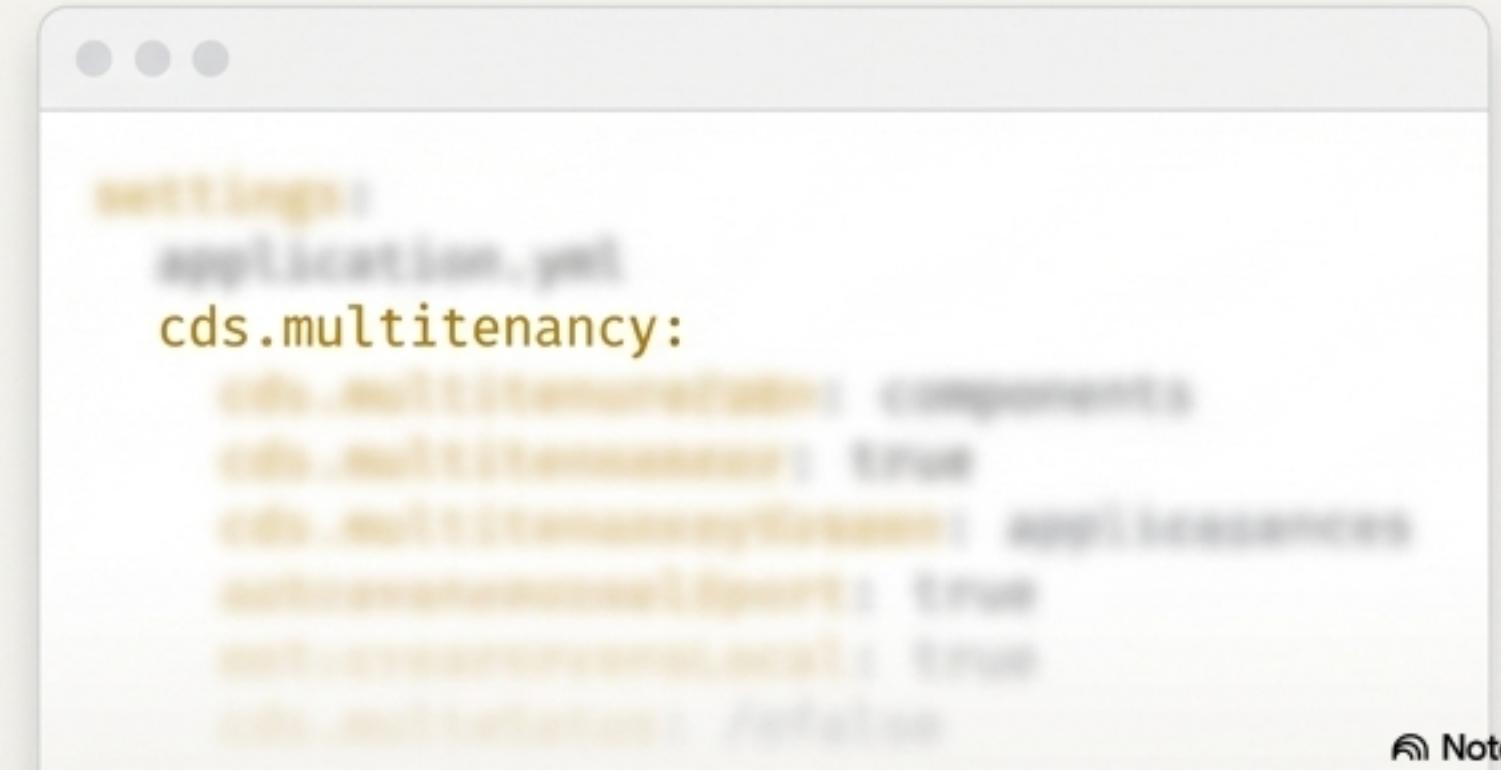
O Resultado: Em ferramentas como **Kibana**, **Cloud Logging Service** ou **Dynatrace**, você pode rastrear uma única operação de ponta a ponta, simplificando drasticamente o debug e a análise de performance.

Propriedades de Configuração

Ponto Central: Muitas configurações de multitenancy podem ser ajustadas através de propriedades de aplicação.

O Prefixo Chave: Procure por todas as configurações relevantes sob o prefixo: **cds.multitenancy**.

Exemplo: `cds.multitenancy.component-scan`, `cds.multiTenancy.dataSource.combinePools.enabled`, etc.



Resumo Estratégico: Seu Modelo Mental para Multitenancy

A Arquitetura

O MTX Sidecar é o motor operacional, gerenciando containers e modelos. Ele é orquestrado por eventos da plataforma.

Seu Controle

O DeploymentService e seus eventos (SUBSCRIBE, UNSUBSCRIBE, etc.) são suas alavancas para injetar lógica de negócio customizada no ciclo de vida do tenant.

A Operação

Use a ferramenta Deploy Main para atualizações de schema em massa e gerencie cuidadosamente o **pooling de conexões** para balancear latência e recursos, uma decisão crucial para a escalabilidade.

Consulte a documentação das propriedades sob `cds.multitenancy` para ajustar o comportamento do seu sistema e refiná-lo para as necessidades específicas da sua aplicação SaaS.