

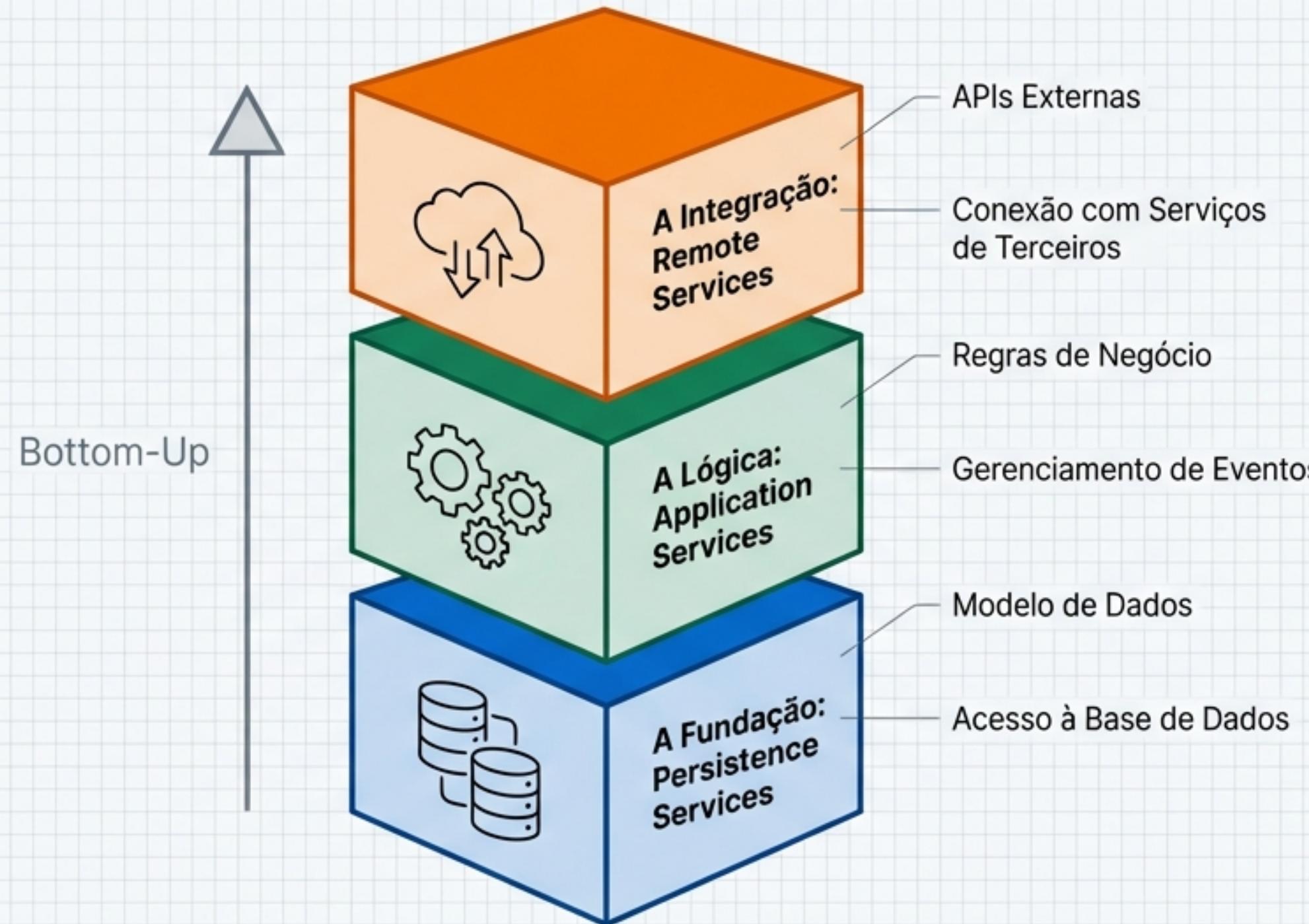
Construindo Aplicações Cloud-Native com CAP Java

Uma Jornada Arquitetural: Da Persistência de
Dados à Integração de Serviços

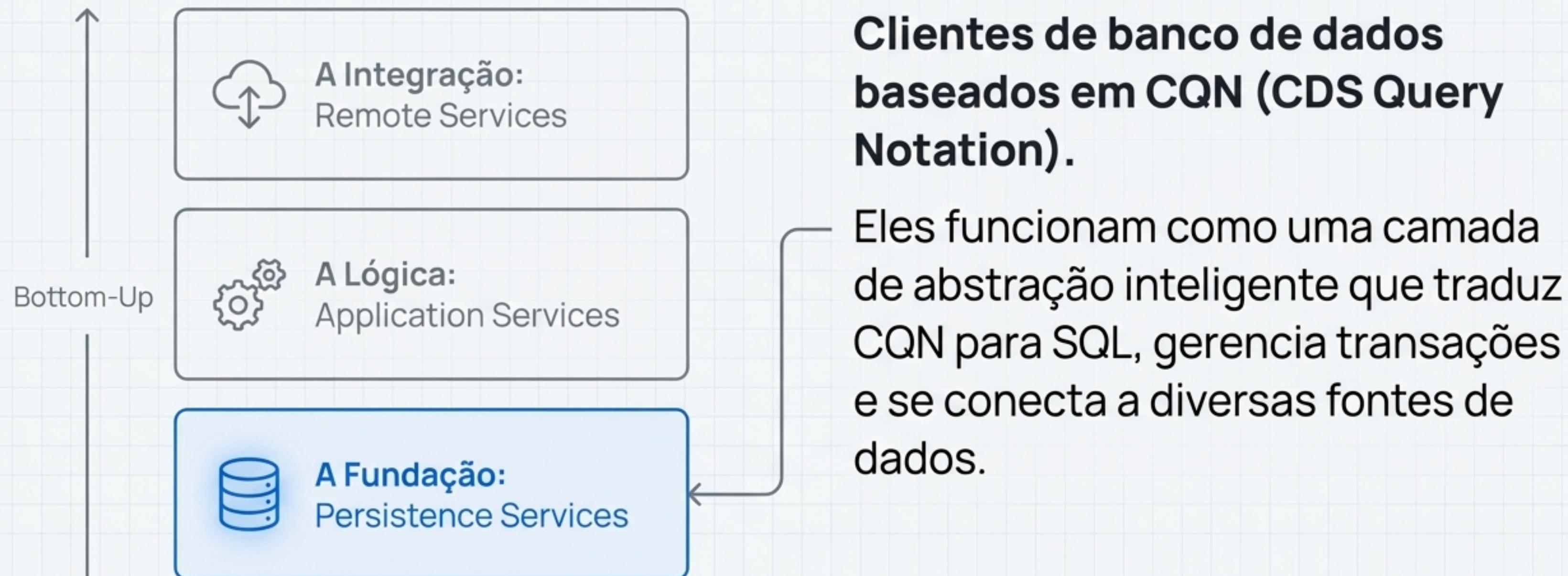


A Anatomia de uma Aplicação CAP Java

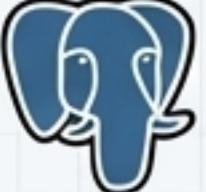
Uma aplicação CAP Java é construída sobre três camadas de serviços distintas e coesas. Vamos explorar cada uma delas, construindo nosso conhecimento da fundação até o topo.



Ato 1: A Fundação – Gerenciando Seus Dados com Persistence Services



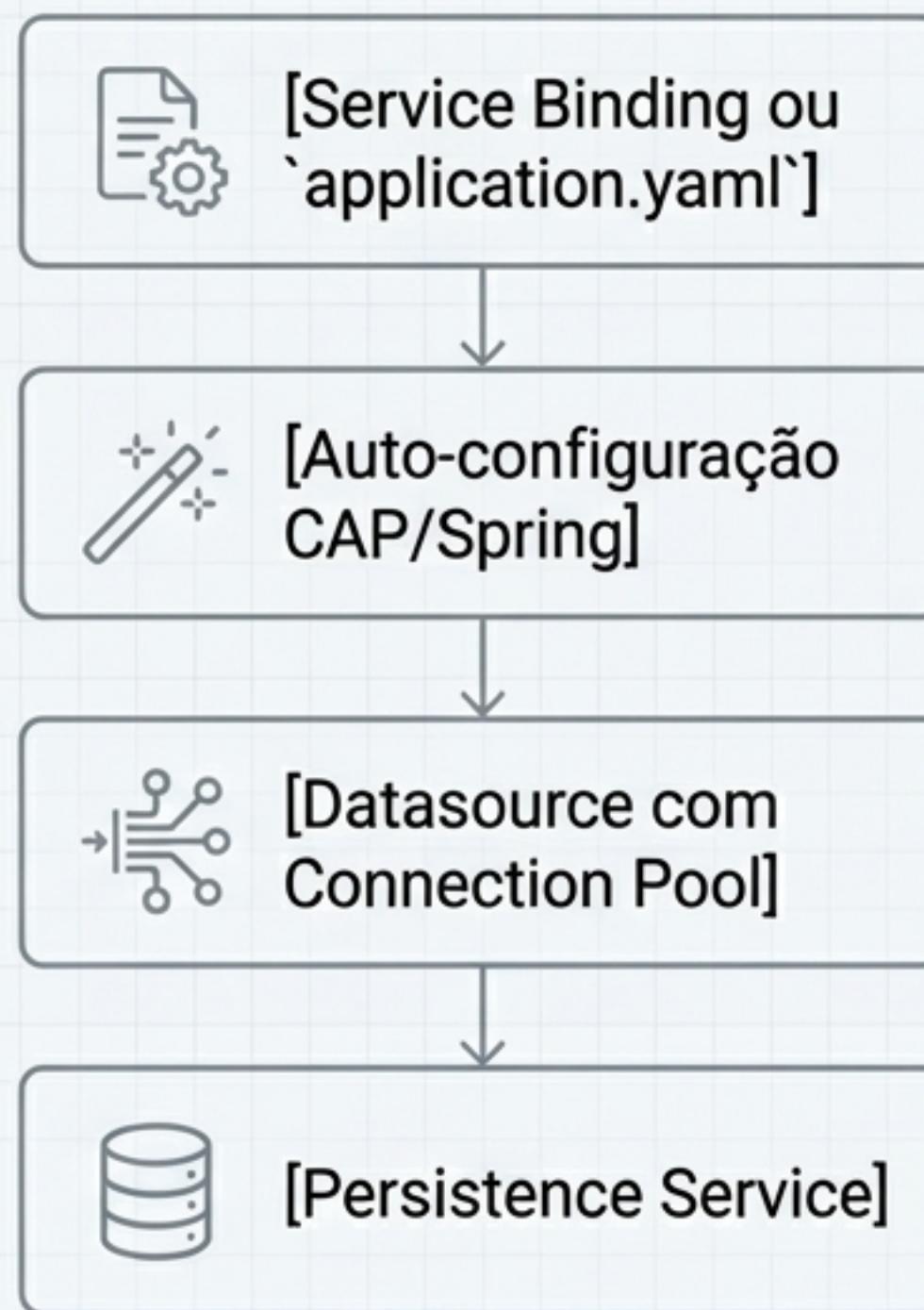
O Ecossistema de Bancos de Dados: Escolhendo a Ferramenta Certa

Banco de Dados	Uso Recomendado	Destaques e Limitações
 SAP HANA Cloud	Produção	Padrão ouro do CAP. Suporte completo a evolução de esquema e multitenancy.
 PostgreSQL	Produção (com ressalvas)	Limitações em ordenação por localidade (locale), multitenancy e extensibilidade.
 H2	Desenvolvimento & Testes	Não há suporte produtivo. Constraints referenciais não são geradas no DDL. Locking pessimista limitado.
 SQLite	Desenvolvimento & Testes	Acesso concorrente limitado (recomenda-se maximum-pool-size: 1). Sem locking pessimista ou streaming de LOBs.

SAP HANA Cloud é o banco de dados padrão e recomendado para uso produtivo com CAP Java.

Configurando o Acesso: Datasources e Connection Pools

Conceito



Exemplo Prático

```
# srv/src/main/resources/application.yaml
```

```
cds:
```

```
    dataSource:
```

```
        my-service-instance:
```

```
            hikari: _____
```

```
                maximum-pool-size: 20
```

```
                data-source-properties:
```

```
                    packetSize: 300000 _____
```

Nome do service binding

Tipo de pool (hikari, tomcat, dbcp2)

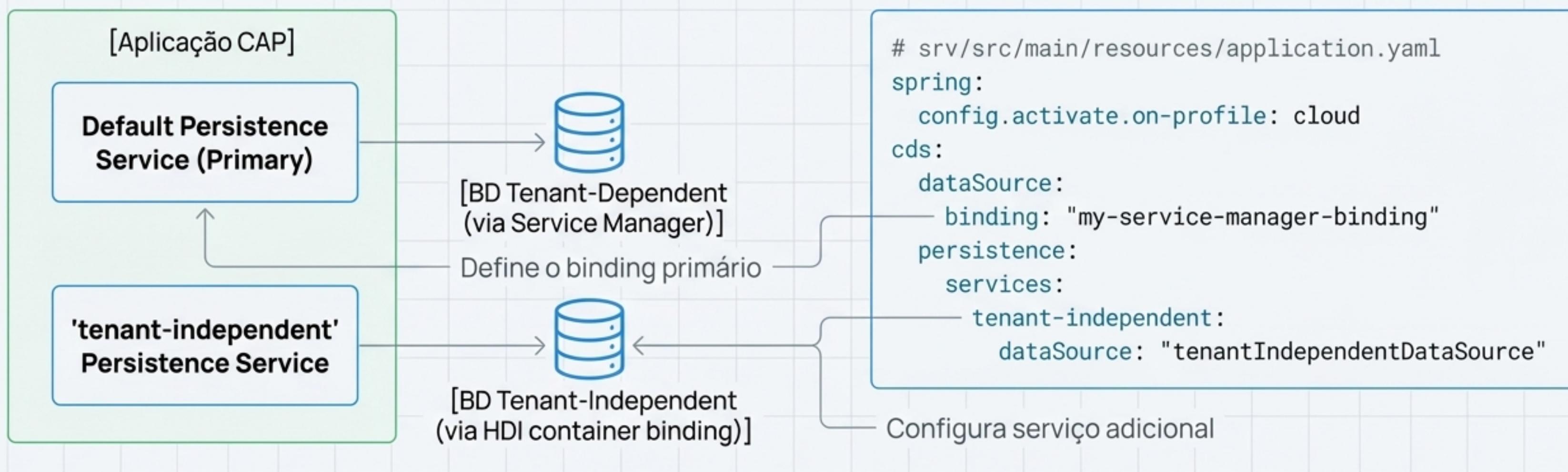
Propriedades específicas do driver JDBC

O CAP Java auto-configura `java.sql.DataSource` a partir de service bindings ou do `application.yaml`. A configuração explícita sempre tem precedência.

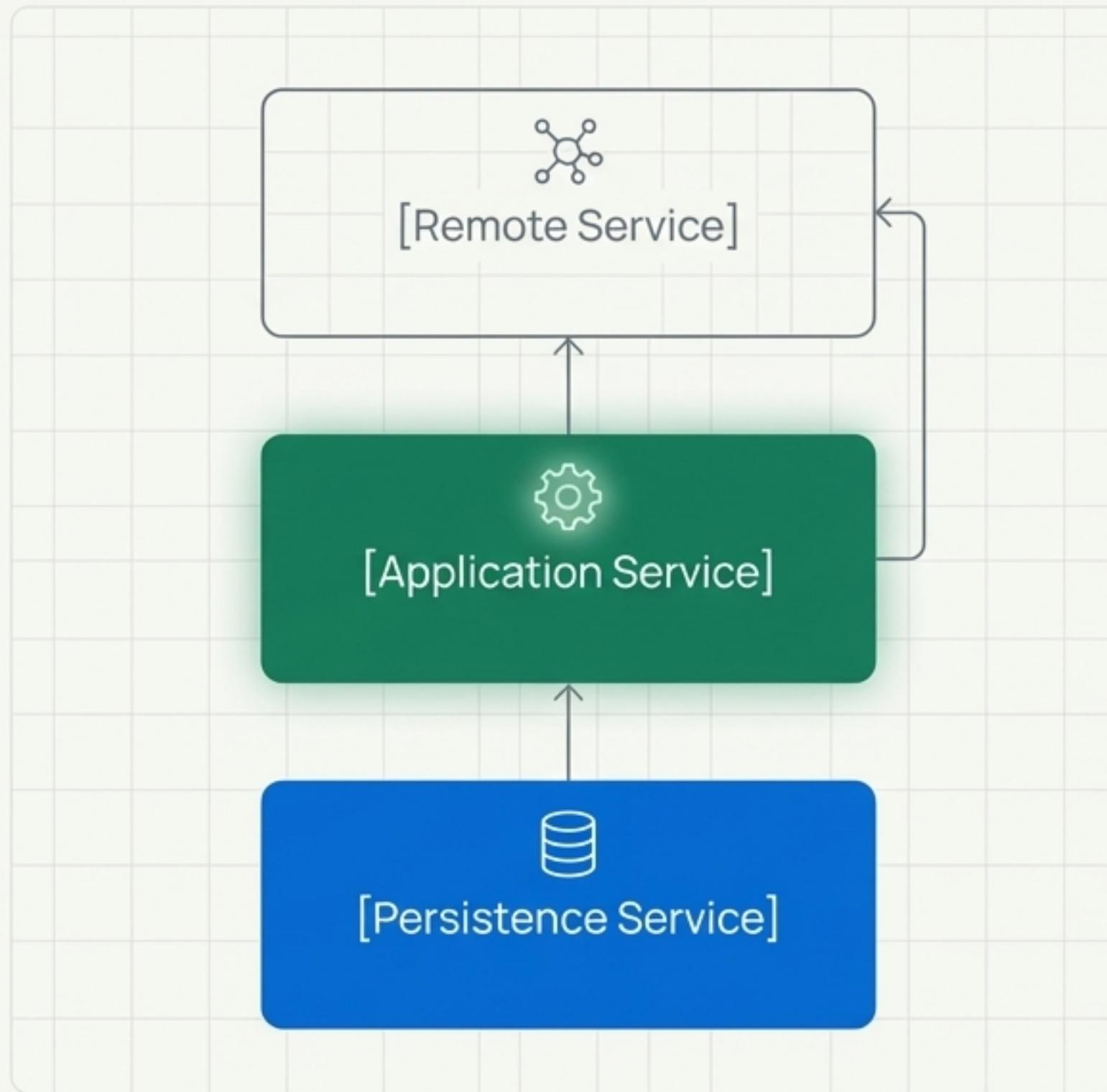
Arquiteturas Avançadas: Gerenciando Múltiplos Persistence Services

Quando múltiplos `DataSource` existem, um deve ser definido como o `Default Persistence Service` (usado pelos handlers genéricos). Os demais geram `Persistence Services` adicionais, nomeados a partir do service binding.

Caso de Uso: Aplicação Multitenant com Dados Tenant-Independent



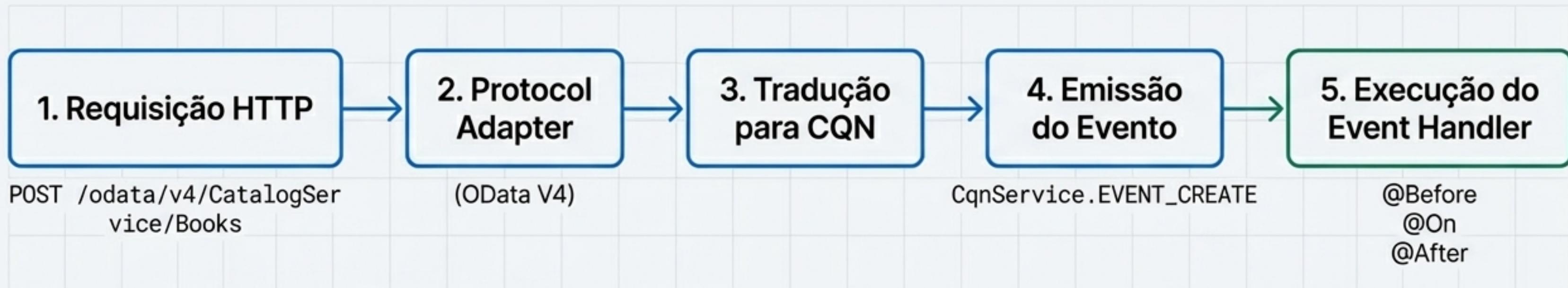
Ato 2: A Lógica – Construindo APIs com Application Services



Definindo as APIs e a Lógica de Negócios.

Application Services definem as APIs que a sua aplicação expõe, comumente via OData. É nesta camada que você implementa a lógica de negócios customizada, estendendo o comportamento padrão dos eventos CRUD e adicionando ações e funções.

O Ciclo de Vida de uma Requisição: Tratando Eventos CRUD



Verbo HTTP	Evento CRUD CAP
POST	CREATE
GET	READ
PATCH / PUT	UPDATE
DELETE	DELETE

Exemplo de Event Handler:

```
@Before(event = CqnService.EVENT_CREATE,  
entity = Books_.CDS_NAME)  
public void validateData(List<Books> books) {  
    // ... sua lógica de validação aqui ...  
}
```

Lógica de Negócios Sob Demanda: Implementando Ações e Funções

Phase 1: Modelar

Modelagem em CDS

```
service CatalogService {  
    entity Books { ... } actions {  
        action review(stars: Integer) returns Reviews;  
    };  
}
```

Phase 2: Implementar

Implementação em Java

```
@On(event = "review", entity = Books_.CDS_NAME)  
public void onReview(ReviewEventContext context) {  
    // Lógica para criar a review com base nos 'stars'  
    Integer stars = context.getStars();  
    Reviews review = ...; // Cria a review  
    context setResult(review);  
}
```

Key Insight: O `cds-maven-plugin` gera interfaces `EventContext` específicas para cada ação, provendo acesso type-safe aos parâmetros de entrada e ao resultado.

Controlando a Exposição: Configurando Rotas e Protocolos de Serviço

A URL final de um serviço é a combinação do `base_path` do protocolo com o `path` relativo do serviço.

Essa configuração pode ser feita diretamente no modelo CDS ou no `application.yaml`.

Configuração via Anotação CDS

```
    ↘ Define o caminho relativo do serviço  
@path : 'browse'  
@protocols: [ 'odata-v4', 'odata-v2' ]  
service CatalogService { ... }  
  
↑  
Especifica os protocolos suportados
```

Configuração via `application.yaml`

```
cds.application.services.CatalogService.serve:  
  path: 'browse' ← Define o caminho relativo  
  protocols: ←  
    - 'odata-v4'  
    - 'odata-v2'  
  
Listas de protocolos habilitados
```

[/odata/v4]

base path

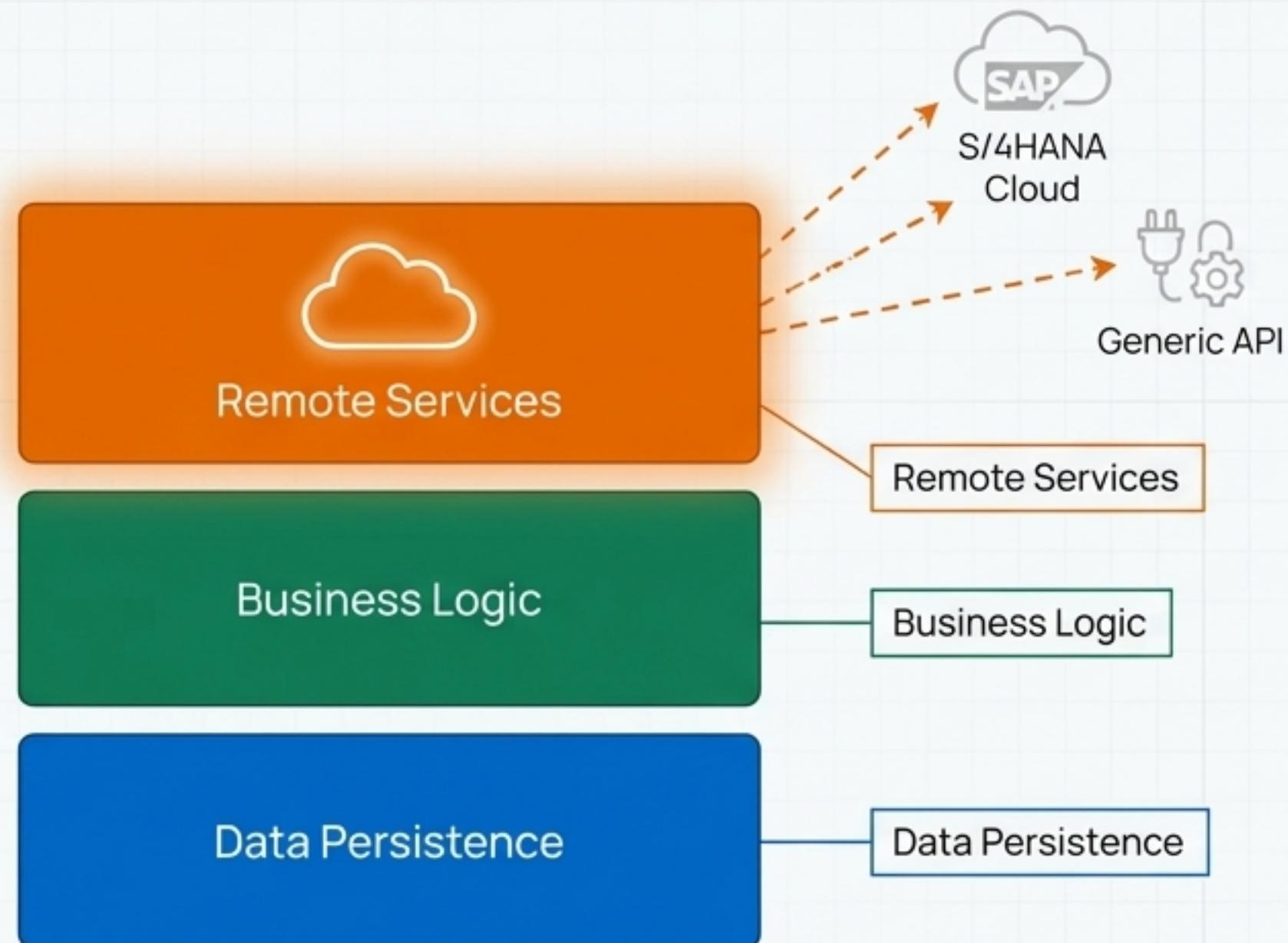
[/browse]

service path

/odata/v4/browse

URL Final

Ato 3: A Integração – Consumindo APIs Externas com Remote Services



Estendendo o modelo de programação para sistemas externos.

Nenhuma aplicação é uma ilha. Remote Services são clientes baseados em CQN para consumir APIs OData V2/V4. Eles abstraem a complexidade de chamadas HTTP e autenticação, e potencializam o SAP Cloud SDK.

Conectividade por Configuração: Definindo um Remote Service



```
# srv/src/main/resources/application.yaml
cds:
  remote.services:
    API_BUSINESS_PARTNER:
      type: "odata-v2"
      destination:
        name: "s4-business-partner-api"
```

Nome lógico do serviço remoto
Protocolo da API
Nome do destino no BTP

A configuração define **O QUÊ** (serviço CDS), **ONDE** (destino ou binding) e **COMO** (protocolo), enquanto o framework cuida da execução.

Uma Linguagem para Dominar Todas: CQN para Persistência e Integração

Consultando o Banco de Dados Local

```
// Injetando o Persistence Service  
@Autowired  
PersistenceService db;  
  
// Query local  
Select<Books_> query = Select.from(Books_.class)  
    .columns(b -> b.title());  
Result result = db.run(query);
```

Dependency
Injection

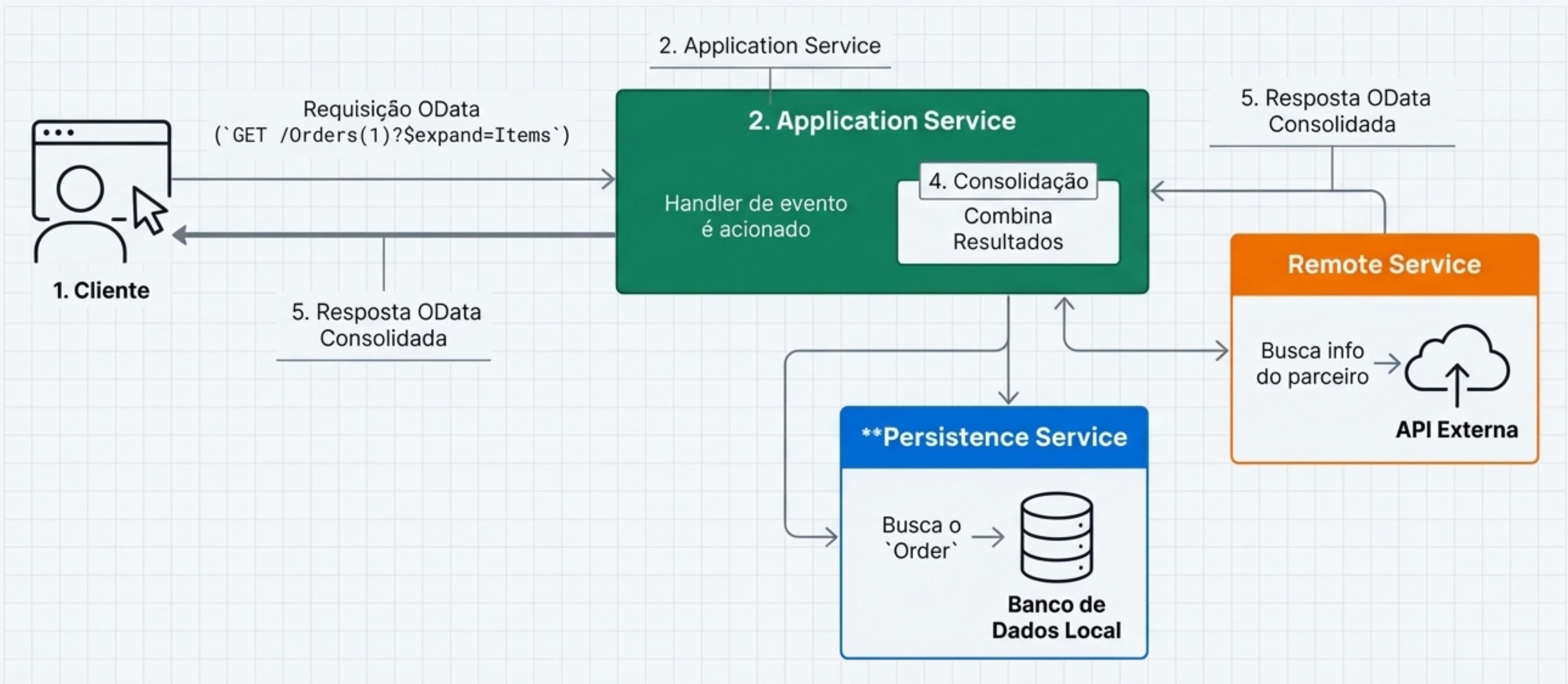
Consumindo um Serviço Remoto

```
// Injetando o Remote Service  
@Autowired  
@Qualifier(ApiBusinessPartner_.CDS_NAME)  
CqnService bupa;  
  
// Query remota  
Select<ABusinessPartner_> query =  
Select.from(ABusinessPartner_.class)  
    .columns(p -> p.BusinessPartnerFullName());  
Result result = bupa.run(query);
```

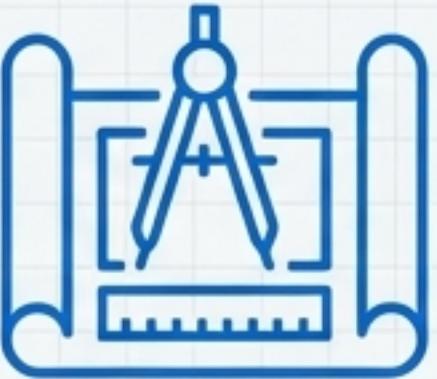
Service
Identifier

A mesma API CQN, type-safe e fluente, unifica o acesso a dados locais e remotos.
Isso simplifica radicalmente o código e promove a consistência arquitetural.

A Arquitetura Completa: A Jornada de uma Requisição



Por que CAP Java? Uma Arquitetura Coesa para a Nuvem



Modelo Consistente e Produtivo

- Definição declarativa de dados e serviços com CDS.
- Linguagem de query unificada (CQN) para fontes locais e remotas.
- Geração de código type-safe que previne erros em tempo de compilação.



Foco na Lógica de Negócios

- Abstração de complexidades (transações, HTTP, autenticação).
- Framework orientado a eventos para uma lógica de negócios limpa e desacoplada.
- Handlers genéricos que fornecem funcionalidades CRUD prontas para uso.



Construído para o Ecossistema SAP

- Integração nativa com serviços do SAP BTP (Destination, Service Binding, XSUAA).
- Suporte robusto para cenários de multitenancy e microserviços.
- Potencializado pelo SAP Cloud SDK para conectividade empresarial.