



Segurança em CAP Java: Do Fundamento à Produção

Um guia prático para construir aplicações seguras e robustas na SAP BTP.



Aviso Importante: Sem configuração de segurança, os serviços CDS são expostos publicamente. Uma configuração adequada de autenticação e autorização é essencial para proteger sua aplicação CAP.

Os Dois Pilares da Segurança de Aplicações

A segurança de aplicações se baseia em duas perguntas fundamentais. O CAP Java oferece mecanismos robustos para responder a ambas.



Autenticação

Quem é você?

Controla *quem* está usando o serviço. Envolve a verificação da identidade do usuário, do tenant e a validação de claims como papéis (roles) concedidos.



Autorização

O que você pode fazer?

Garante que o usuário tenha os privilégios necessários para acessar os recursos solicitados. Trata-se de controlar *o que* o usuário tem permissão para fazer.

Autenticação: Verificando a Identidade do Usuário



O CAP Java possui uma arquitetura modular que permite a configuração flexível de métodos de autenticação. Por padrão, ele suporta os serviços de identidade da plataforma BTP:

- SAP Cloud Identity Services - Identity Authentication (IAS): A solução preferencial, integrando endpoints entre sistemas SAP.
- SAP Authorization and Trust Management Service (XSUAA): A oferta anterior, com escopo em uma landscape BTP.

Como Habilitar

- Recomendação: Utilize os starters [cds-starter-cloudfoundry](#) ou [cds-starter-k8s](#). Eles incluem todas as dependências necessárias.
- Bindings de Serviço: A sua aplicação precisa estar vinculada às instâncias de serviço correspondentes, dependendo dos tokens que ela deve aceitar (somente XSUAA, somente IAS, ou ambos).

A Configuração de Segurança Automática do Spring Boot

Quando as dependências da biblioteca e um binding de serviço XSUAA/IAS estão presentes, o CAP Java SDK ativa uma configuração de segurança que impõe autenticação para todos os endpoints por padrão.

- Endpoints de adaptadores de protocolo (OData V4/V2, etc.).
- Endpoints customizados restantes (REST controllers, Spring Actuators).

Exemplo de Controle no Modelo CDS

Você pode abrir endpoints específicos para acesso público usando a pseudo-role `any`.

Observação: Só é possível tornar um endpoint público se o caminho completo do endpoint também for público.

```
1 // Este serviço e a entidade Books são
  // públicos.
2 service BooksService @(requires: 'any') {
3
4   @readonly
5   entity Books @(requires: 'any') {...}
6
7   // A entidade Reviews requer autenticação
    // por padrão.
8   entity Reviews {...}
9
10
11  // A entidade Orders requer o papel
    'Customer' (autenticação e autorização).
12  entity Orders @(requires: 'Customer')
13  {...}
14 }
```

Ajustando o Comportamento da Autenticação

A propriedade `cds.security.authentication.mode` controla a estratégia de autenticação para endpoints gerenciados por adaptadores de protocolo.

- `never`: Nenhum endpoint requer autenticação.
- `model-relaxed`: A autenticação é derivada das anotações `@requires` e `@restrict`. Se não houver anotação, o endpoint é autenticado.
- `model-strict` (Padrão): Todos os endpoints requerem autenticação, a menos que explicitamente marcados com `@requires: 'any'`. Isso reforça o princípio de "seguro por padrão".
- `always`: Todos os endpoints requerem autenticação, sem exceção.

Outras Propriedades Chave

- `authenticateUnknownEndpoints` (default: `true`) Impõe autenticação para endpoints não gerenciados por CAP (ex: custom REST).
- `authenticateMetadataEndpoints` (default: `true`) Impõe autenticação para endpoints `$metadata` do OData.



Dica: Para aplicações multitenant, é mandatório autenticar todos os endpoints, pois a informação do tenant é essencial para o processamento da requisição.

Customizando com Configurações Adicionais do Spring Security

Para casos de uso específicos, como um método de autenticação alternativo para certos endpoints, você pode adicionar sua própria configuração de segurança do Spring.

Regra de Ouro

Sua configuração customizada deve ter maior precedência que a do CAP. Utilize a anotação @Order com um valor numérico menor (ex: 1).

Exemplo: Liberando um Endpoint `/public/**`

```
@Configuration  
@EnableWebSecurity  
public class AppSecurityConfig {  
    @Bean  
    @Order(1) // Prioridade maior que a configuração do CAP ←  
    public SecurityFilterChain appFilterChain(HttpSecurity http) throws Exception {  
        http  
            .securityMatcher(AntPathRequestMatcher.antMatcher("/public/**"))  
            .csrf(c -> c.disable())  
            .authorizeHttpRequests(r -> r.anyRequest().permitAll())  
            .build();  
    }  
}
```

 **Cuidado:** Seja cauteloso ao configurar a instância HttpSecurity. Certifique-se de que apenas os endpoints pretendidos sejam afetados para não criar brechas de segurança.

Tópico Avançado: Proof-of-Possession com mTLS

Proof-of-Possession (PoP) é uma técnica de segurança adicional onde um token JWT é vinculado a um cliente OAuth específico. Com IAS, isso é implementado através de um túnel mTLS (mutual TLS).

Como Funciona



1. O chamador (cliente) envia o token JWT emitido pelo IAS.
2. Além do token, o cliente envia um certificado de cliente, estabelecendo um canal mTLS.
3. O CAP Java valida tanto o token quanto o vínculo com o certificado.

Requisitos de Configuração

- **Cloud Foundry**: Expor a aplicação em uma rota que aceite certificados de cliente e os encaminhe no header `X-Forwarded-Client-Cert`.
- **AppRouter**: A destination que aponta para o backend CAP deve ter a propriedade `forwardAuthCertificates: true`.
- **Ativação**: O PoP é ativado por padrão ao usar autenticação IAS com a biblioteca SAP BTP Spring Security Client (versão 3.5.1 ou superior).



Autorização: Definindo o Que o Usuário Pode Fazer

O SDK do CAP Java oferece um serviço de autorização abrangente. Ao definir regras declarativamente em seu modelo CDS, o **runtime** impõe a autorização de forma genérica.

Dois Níveis de Autorização

1. Baseada em Papéis (Role-based)

Restringe o acesso a recursos com base nos papéis (roles) do usuário. Definido com `@requires`.

2. Baseada em Instância (Instance-based)

Leva a segurança a um nível mais granular. Um usuário pode ser restrito a instâncias que atendem a uma determinada condição (ex: `where: 'pais = $user.pais'`). Definido com `@restrict`.

Abordagem Recomendada

- Configure a autorização de forma declarativa no modelo CDS.
- Se necessário, implementações customizadas podem ser construídas sobre a API de Autorização.

Capacidades Avançadas de Autorização

Padrão desde CAP Java 4.0

O CAP Java oferece funcionalidades 'out of the box' para cenários de autorização complexos.

1. Deep Authorization



O quê

Queries são autorizadas não apenas na entidade de destino, mas também em todas as entidades associadas usadas na query (ex: em um `$expand`).

Impacto

Garante que o usuário não acesse dados de entidades relacionadas para as quais não tem permissão.

2. Forbidden em Seleção de Entidade Rejeitada



O quê

Quando um usuário não autorizado tenta acessar uma única instância, o sistema pode retornar `403 - Forbidden` em vez do padrão `404 - Not Found` para requisições `PATCH` e `DELETE`.

Impacto

Permite que a UI distinga entre 'não encontrado' e 'acesso negado', mas exige cuidado para não vazar a existência de dados.

Verificação de Autorização em Dados de Entrada



O quê

Dados de entrada em eventos `CREATE` e `UPDATE` são validados contra as condições de autorização baseadas em instância.

Impacto

Impede que um usuário crie ou modifique um registro com valores para os quais ele não teria permissão de acesso (ex: mudar um pedido para uma área contábil que ele não gerencia).



Acessando o Contexto do Usuário: A API UserInfo

Em handlers customizados, você pode precisar acessar as informações do usuário atual para lógicas de negócios complexas. O objeto **UserInfo** é a chave para isso.

Como Obter o UserInfo

- Via **EventContext**: `UserInfo user = context.getUserInfo();`
- Via **Injeção de Dependência**: `@Autowired UserInfo user;`

Métodos Úteis do UserInfo

Método	Descrição
<code>getName()</code>	Retorna o nome de logon único do usuário.
<code>getTenant()</code>	Retorna o tenant do usuário.
<code>isAuthenticated()</code>	Retorna true se o usuário foi autenticado.
<code>hasRole(String role)</code>	Verifica se o usuário possui um determinado papel.
<code>getAttributeValues(String attribute)</code>	Retorna a lista de valores de um atributo de usuário (referenciado por <code>\$user.<attribute></code>).
<code>isPrivileged()</code>	Retorna true se o usuário está em modo privilegiado (bypass de autorização).

Executando Lógica em Modo Privilegiado

```
cdsRuntime.requestContext().privilegedUser().run(privilegedContext -> {
    // Código aqui dentro bypassa as checagens de autorização.
    assert privilegedContext.getUserInfo().isPrivileged();
});
```

A Abordagem Moderna: Autorização IAS via AMS

Aplicações CAP que utilizam o **Identity Authentication Service (IAS)** para autenticação devem gerenciar a autorização através do **Authorization Management Service (AMS)**.

- O CAP oferece uma integração simplificada com o AMS, permitindo o gerenciamento centralizado de políticas de acesso em nível de negócio.
- Essa integração funciona como um plugin de fácil consumo para aplicações CAP.

Habilitando a Integração com a CDS CLI

```
cds add ams
```

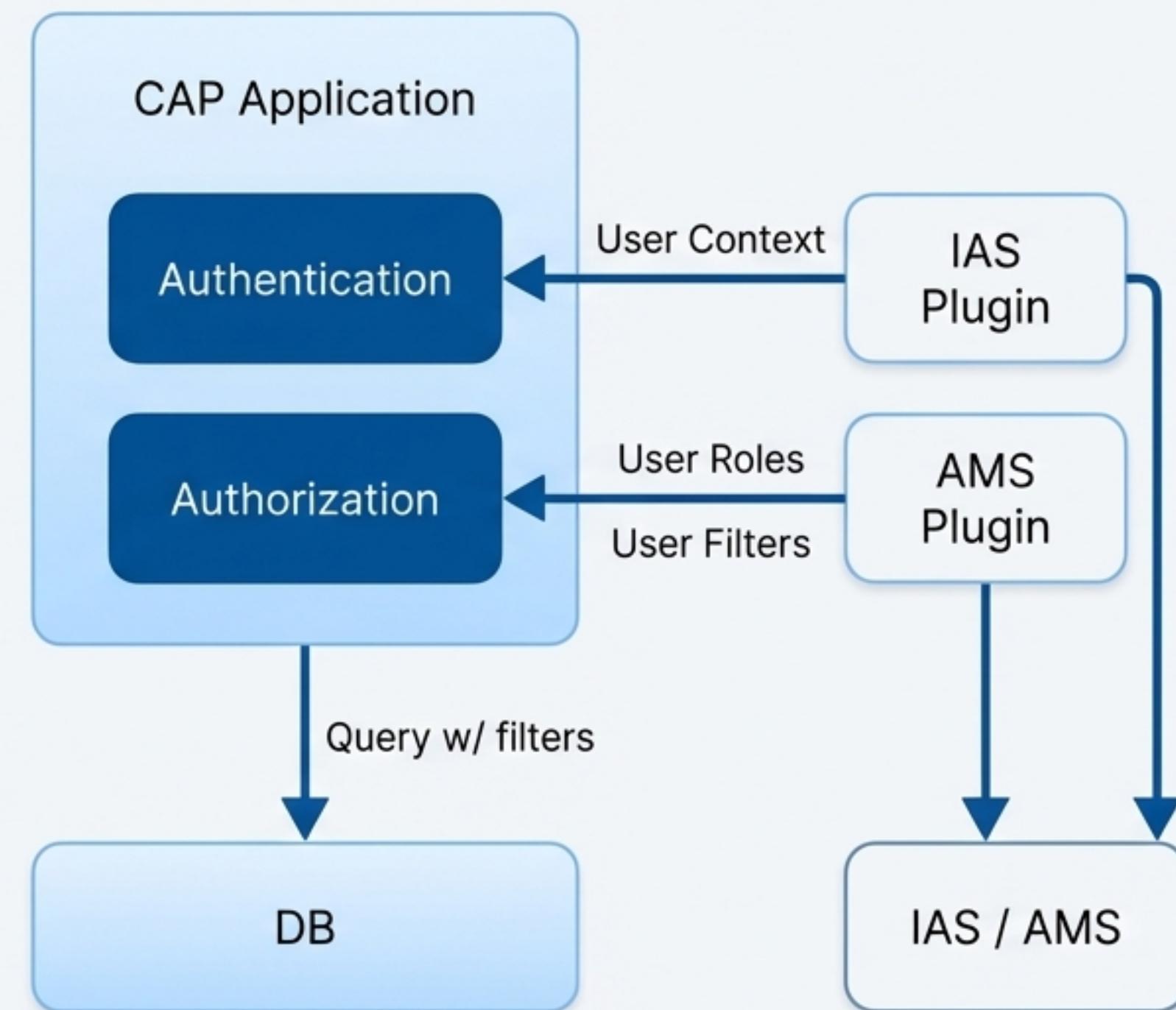
Vantagem Principal

O modelo CDS e as regras de autorização técnica na aplicação são totalmente desacoplados das políticas de negócio definidas no AMS.

Como a Integração CAP + IAS/AMS Funciona

A interação entre a aplicação CAP e o AMS (via plugin) ocorre de forma transparente:

- 1. Autenticação:** O IAS Plugin realiza a autenticação como um pré-requisito. O contexto do usuário (User Context) é estabelecido.
- 2. Injeção de Claims:** O AMS Plugin injeta os papéis (User Roles) e filtros de instância (User Filters) no contexto, com base nas políticas do AMS atribuídas ao usuário.
- 3. Autorização CAP:** O CAP executa a autorização normalmente, usando as regras do modelo CDS (`@requires`, `@restrict`) e os claims injetados pelo AMS.



Do Modelo CDS às Políticas de Acesso no AMS

O plugin do AMS (@sap/ams) automatiza a geração de políticas a partir do seu modelo CDS.



1. Definição no CDS

Você define um papel necessário para acessar uma entidade.

```
entity Books @requires: 'Reader' {...}
```



2. Geração da Política (Build)

Durante o build (`mvn clean install`), o plugin gera um arquivo `basePolicies.dcl` que traduz a regra do CDS para a Data Control Language (DCL) do AMS.

```
POLICY Reader {  
    ASSIGN ROLE Reader;  
}
```

O arquivo gerado fica em `srv/src/main/resources/ams/cap/basePolicies.dcl`.



3. Implantação (Deploy)

O comando `cds add ams` também configura um 'deployer' que fará o upload dessas políticas base para o servidor AMS durante o deploy da aplicação. Após o deploy, as políticas podem ser atribuídas a usuários no painel de administração do IAS/AMS.

Protegendo a Comunicação de Saída (Outbound)

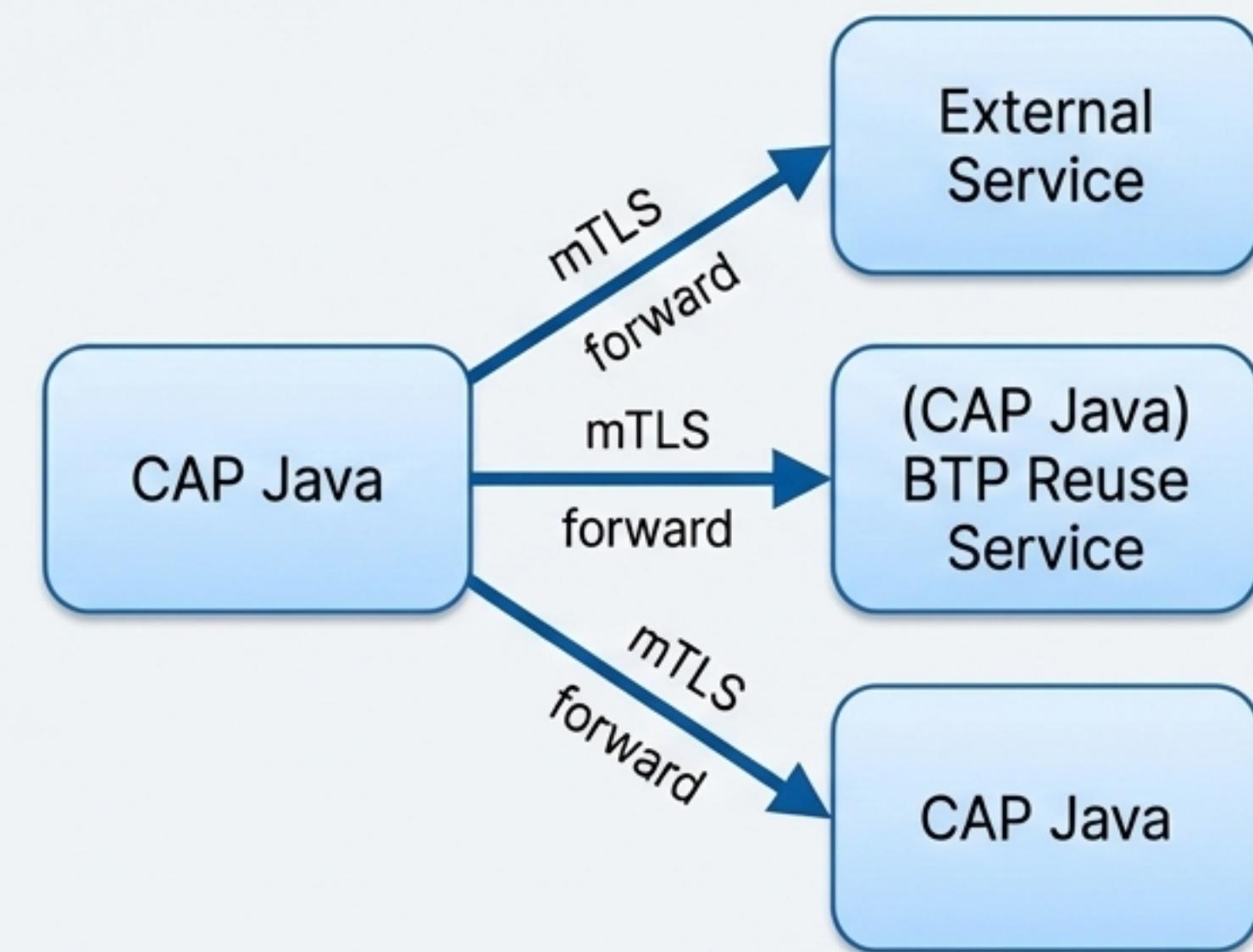
O CAP Java também simplifica a comunicação segura e a propagação de usuário para serviços remotos baseados em IAS. A comunicação service-to-service é protegida com mTLS automaticamente.

Tipos de Serviços Suportados

- **Serviços Internos:** Aplicações CAP na mesma instância de identidade.
- **Serviços Externos (IAS App-to-App):** Aplicações externas consumidas via Destination.
- **BTP Reuse Services:** Serviços BTP consumidos via service binding.

Fluxo de Comunicação

O CAP Java atua como cliente, estabelecendo túneis mTLS com outros serviços para encaminhar requisições de forma segura, propagando a identidade do usuário quando aplicável.



Resumo das Melhores Práticas de Segurança em CAP Java

- 1 Fundamentos Primeiro:** Sempre configure a segurança. Lembre-se que, por padrão, os serviços são públicos. Entenda a diferença clara entre Autenticação e Autorização.
- 2 Seguro por Padrão:** Utilize o modo de autenticação model-strict (padrão) e abra endpoints para o público de forma explícita e consciente com @requires: 'any' .
- 3 Seja Declarativo:** Defina suas regras de autorização diretamente no modelo CDS com @requires e @restrict. Isso torna a segurança visível, auditável e mais fácil de manter.
- 4 Adote o Padrão Moderno:** Para novas aplicações, utilize a autenticação IAS com autorização via AMS. A integração via cds add ams simplifica drasticamente a configuração.
- 5 Proteja Todas as Frentes:** Não se esqueça da comunicação de saída. O CAP Java simplifica a configuração de mTLS e propagação de usuário para chamadas a outros serviços.