

Dominando Aplicações CAP Java

Um Playbook de Otimização e Observabilidade

Guia prático para desenvolvedores,
DevOps e SREs no ecossistema SAP.

Seu Guia Completo para Aplicações de Alta Performance



Otimização Profunda

Mergulhe fundo no desempenho da sua aplicação. Realize análises cirúrgicas para identificar e eliminar gargalos de CPU e memória.



Observabilidade Holística

Obtenha visibilidade 360°. Monitore a saúde, o comportamento e as interações da sua aplicação em tempo real, desde logs detalhados até traces distribuídos.



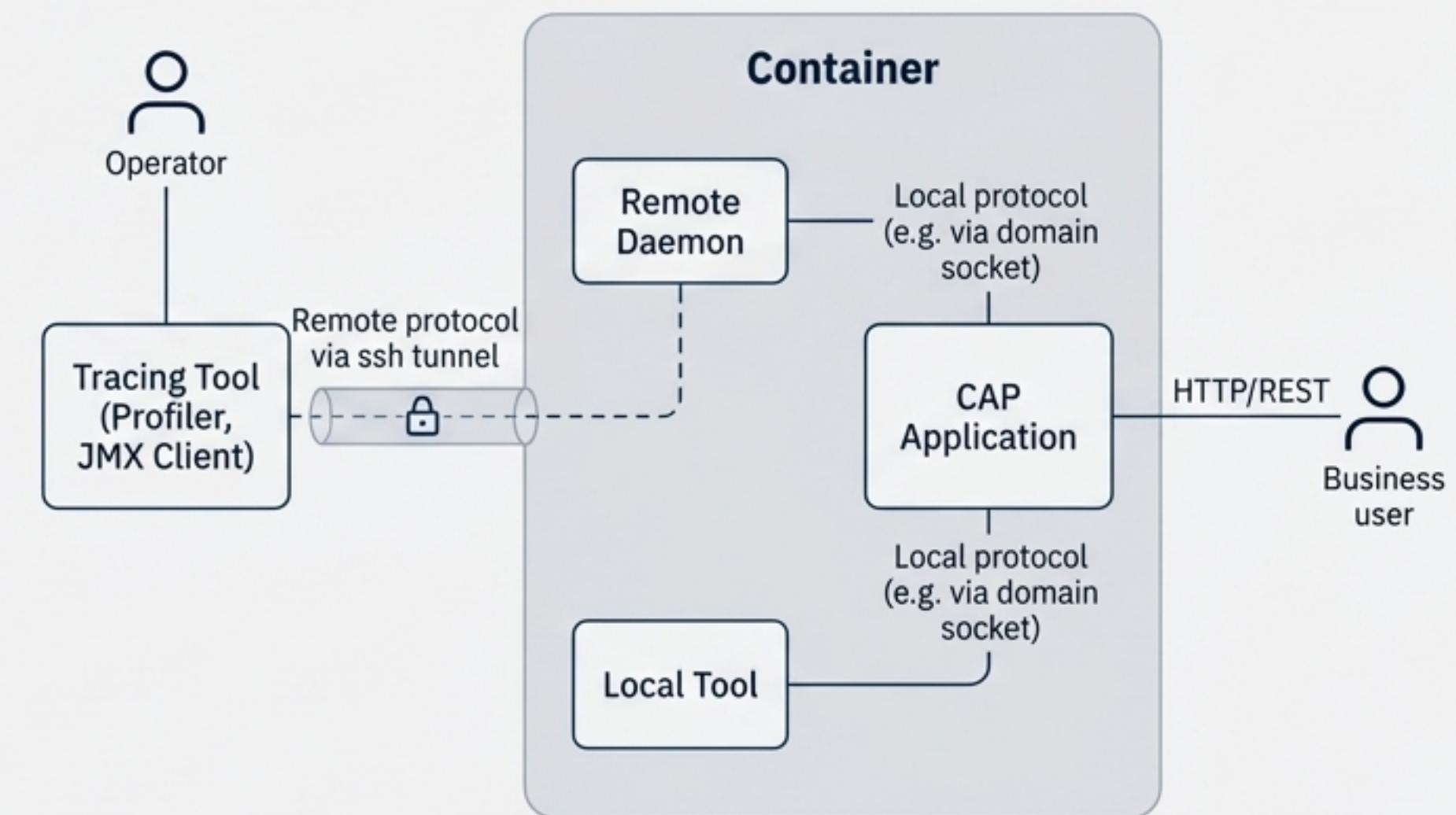
Experiência do Desenvolvedor

Acelere seu fluxo de trabalho. Utilize ferramentas que centralizam o controle e a visibilidade durante o ciclo de desenvolvimento.

Profiling: O Diagnóstico Interno da Sua Aplicação Do Recm Bold

Métricas globais de monitoramento são insuficientes para diagnosticar problemas específicos. Ferramentas de profiling oferecem uma visão focada, mas devem ser usadas com cuidado para não interferir na operação. Os requisitos essenciais são:

- Serem ativáveis em tempo de execução.
- Utilizar um canal de comunicação seguro, não exposto a usuários não autorizados.
- Não interferir ou bloquear as requisições de negócio.



O Arsenal de Ferramentas para Análise de Performance

Ferramentas de Diagnóstico do JDK

- `jcmd`, `jinfo`, `jstack`, `jmap`: Para obter informações básicas do processo JVM, como stack traces, heap dumps e propriedades.
- **SAP JVM**: Ferramentas adicionais como `jvmmmon` e `jvmprof` para insights profundos sobre o consumo de recursos.

Introspecção com JMX (Java Management Extensions)

- Permite monitorar o estado interno da JVM via MBeans.
- O Spring Boot cria MBeans automaticamente.
- Ativação simples: `spring.jmx.enabled: true`
- Clientes comuns: JConsole (parte do JDK) e OpenJDK Mission Control.



Cuidado com a Exposição de JMX

Nunca exponha endpoints JMX/MBeans publicamente. Isso representa um sério risco de segurança. Para acesso remoto, utilize um túnel SSH seguro:

```
cf ssh -N -T -L <local-port>:localhost:<port> <app-name>
```

O Futuro da Performance: GraalVM Native Image

Compile suas aplicações Spring Boot em executáveis nativos para obter tempos de inicialização mais rápidos e menor consumo de memória. O CAP Java é compatível com essa tecnologia.

Condições Essenciais para Compilação Nativa



1. **Análise de Código Estática:** Apenas o código alcançável na análise estática é incluído. O classpath completo deve ser conhecido em tempo de build.



2. **Registro de Elementos Dinâmicos:** Usos de reflection, JDK proxies ou recursos precisam ser registrados previamente. O CAP Java automatiza isso para handlers e interfaces do modelo CDS.



3. **Configuração em Tempo de Build:** Definições de beans e service bindings (HANA, XSUAA) devem ser fornecidas durante o build, de forma similar aos descritores de deploy (**mta.yaml**).

Exemplo de Configuração (`native-build-env.json`)

```
{  
  "hana": [ { "name": "<hana-binding-name>" } ],  
  "xsuaa": [ { "name": "<xsuaa-binding-name>" } ]  
}
```

Logging: A Fundação da Visibilidade

Logs de aplicação são a primeira linha de defesa para reconstruir fluxos de execução e isolar falhas. O CAP Java utiliza a fachada SLF4J, que abstrai o framework de log (Logback, Log4j, etc.), permitindo flexibilidade no deploy.

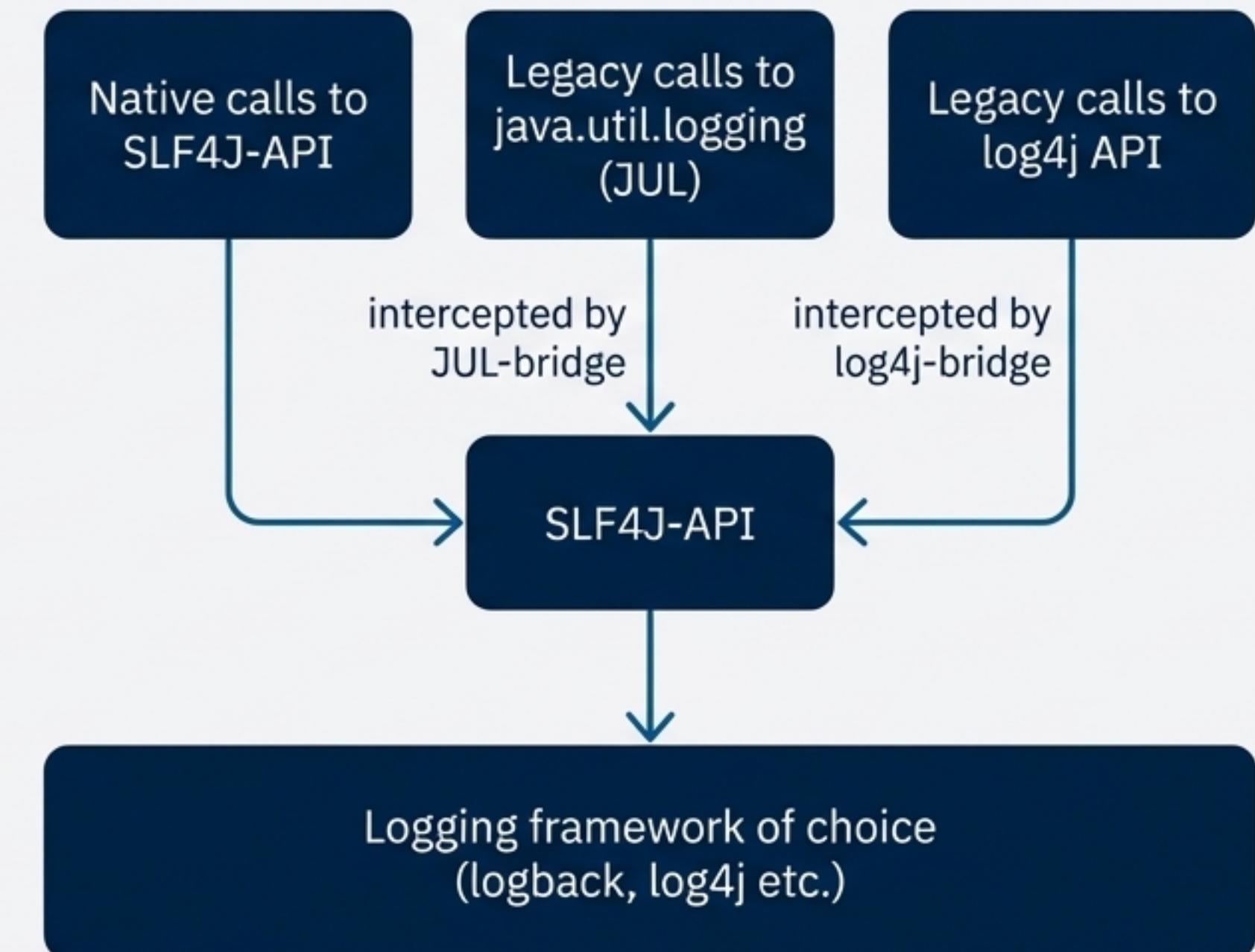
Dica de Mestre

Boas Práticas de Logging

Prefira logs parametrizados:

```
logger.debug("Consolidando pedido {}", order);
```

Concatenar strings (`"Consolidando pedido " +
order`) cria a `String` independentemente do nível
de log, impactando a performance.



Controle Dinâmico: Ajustando os Níveis de Log

TRACE

Fluxo detalhado da aplicação.

DEBUG

Mensagens de diagnóstico.

INFO

Fluxos importantes (padrão).

WARN

Cenários de erro em potencial.

ERROR

ERROS Erros e exceções.

Métodos de Configuração com Spring Boot

Em Tempo de Compilação:

```
logging:  
  level:  
    root: WARN  
    com.sap.cds.persistence.sql: DEBUG
```

Em Tempo de Execução (com Restart):

Usando variáveis de ambiente para sobrescrever a configuração.
Ex: `LOGGING_LEVEL_COM_SAP_CDS_SECURITY=DEBUG`

Em Tempo de Execução (sem Restart):

Usando Spring Actuators via requisições HTTP para alterar níveis dinamicamente. Ex:

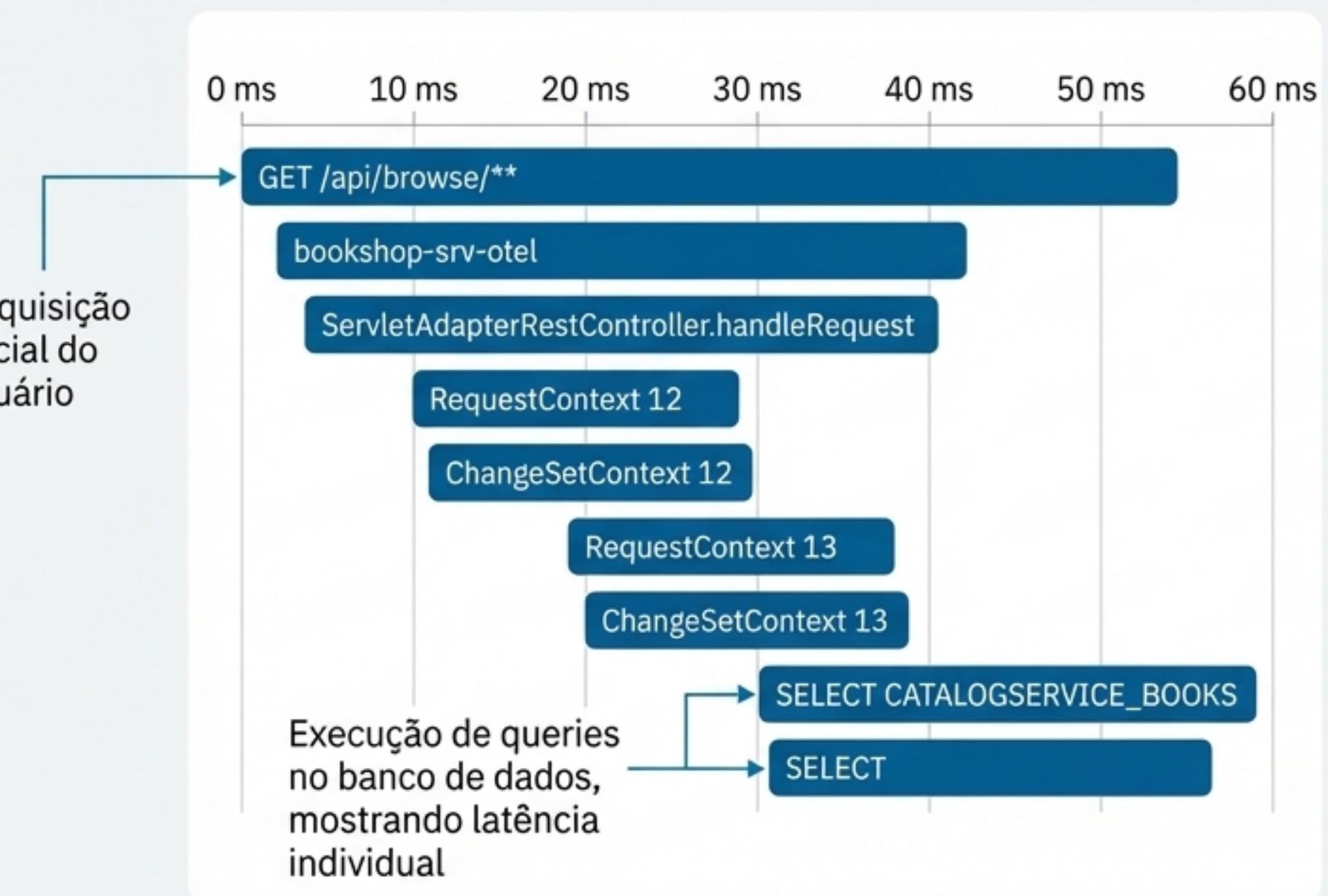
```
curl -X POST -H 'Content-Type: application/json' \  
-d '{"configuredLevel": "DEBUG"}' \  
https://<app-url>/actuator/loggers/com.sap.cds.security
```

A Visão Sistêmica: Monitoramento e Tracing com OpenTelemetry

OpenTelemetry é um framework open source que permite coletar e enviar sinais de observabilidade (traces, métricas e logs) para backends de análise como SAP Cloud Logging e Dynatrace.

O CAP Java oferece integração profunda, gerando telemetria automaticamente para:

- Requisições HTTP e eventos do ciclo de vida CAP.
- Execução de statements CQN.
- Requisições individuais em um `batch` OData.
- Métricas padrão da JVM (CPU, memória).



Integrando com o Ecossistema SAP BTP

1 Passo 1: Habilitar o OpenTelemetry Java Agent

Adicionar a configuração `JBP_CONFIG_JAVA_OPTS` no `mta.yaml` para anexar o agente.

```
properties:  
  JBP_CONFIG_JAVA_OPTS:  
    java_opts: >  
      -javaagent:METACONF/.sap_java_buildpack/otel_agent/opentelemetry-javaagent.jar  
      -Dotel.javaagent.extensions=METACONF/.sap_java_buildpack/otel_agent_extension/sap-otel-java-agent-extension.jar
```

2 Passo 2: Configurar o Exportador de Sinais

Para SAP Cloud Logging:

Vincular o serviço `cloud-logging` (com `ingest_otlp`) e definir as variáveis de ambiente.

```
properties:  
  OTEL_METRICS_EXPORTER: cloud-logging  
  OTEL_TRACES_EXPORTER: cloud-logging
```

Para Dynatrace:

Conectar ao Dynatrace OneAgent e exportar apenas métricas via OpenTelemetry para evitar duplicação de traces.

```
properties:  
  OTEL_METRICS_EXPORTER: dynatrace  
  OTEL_TRACES_EXPORTER: none
```

Métricas e Saúde: O Pulso da Aplicação

Spring Boot Actuators fornecem endpoints prontos para uso que expõem informações operacionais críticas.

Endpoint de Saúde (/actuator/health)

- Oferece uma verificação de disponibilidade essencial.
- Agrega o status de componentes cruciais (banco de dados, serviços externos).
- CAP Java adiciona indicadores como modelProvider para o status do MTX sidecar.

Configuração Segura Recomendada

Expor apenas o endpoint de saúde publicamente, sem detalhes internos.

```
management:  
  endpoints:  
    web:  
      exposure:  
        include: health # Expor apenas /health  
    endpoint:  
      health:  
        show-components: always # Mostrar componentes no status agregado  
        show-details: never     # Não expor detalhes sensíveis
```



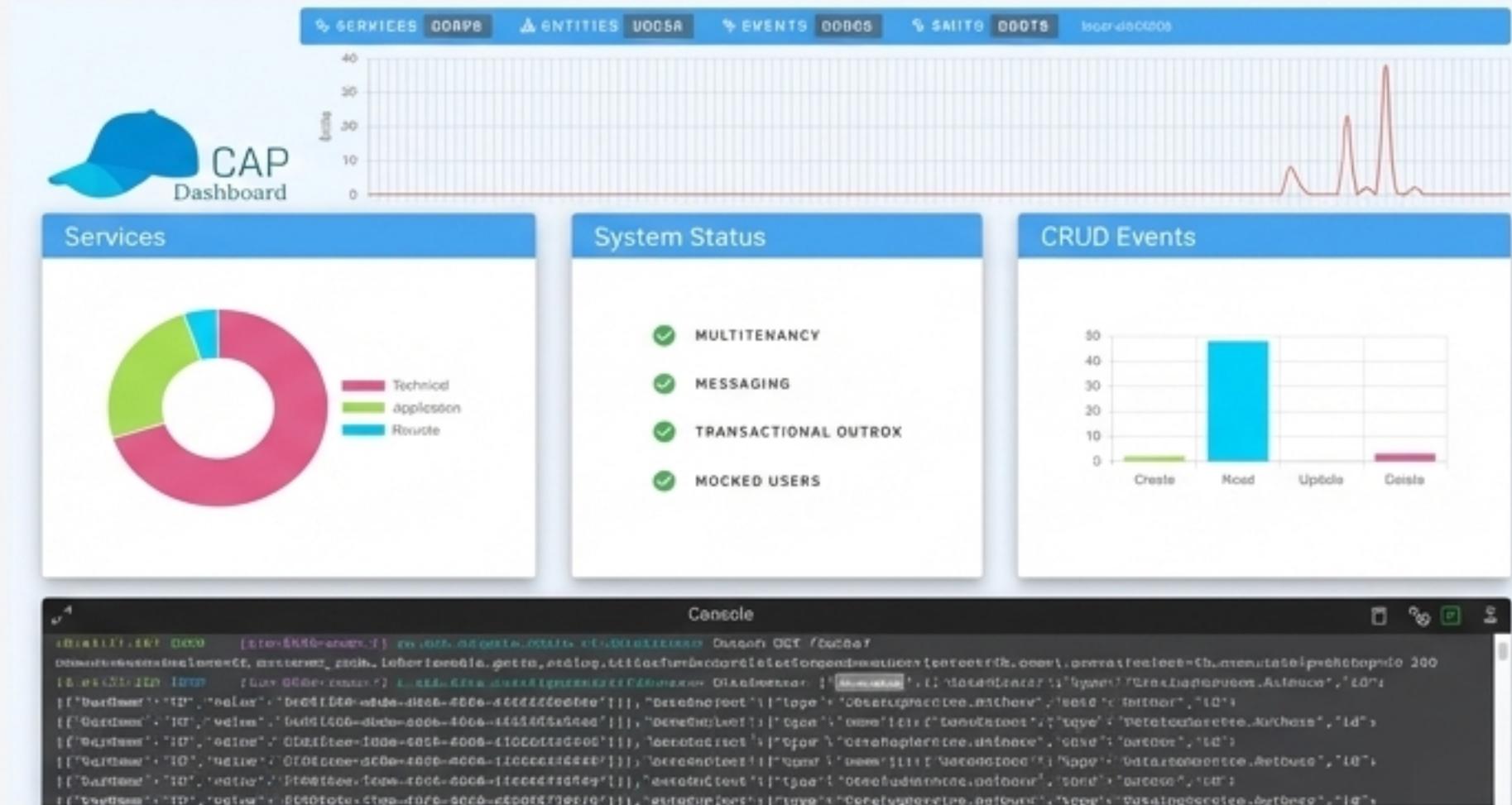
Segurança em Primeiro Lugar

Endpoints de actuator podem expor dados sensíveis. Utilize JMX para acesso local à maioria dos actuadores ('metrics', 'env', 'beans') e exponha via web apenas o que for estritamente necessário, como o '/health'.

Módulo 3: A Experiência do Desenvolvedor

Apresentando o CAP Developer Dashboard

Uma central de controle unificada para monitorar e gerenciar sua aplicação CAP durante o desenvolvimento. Simplifique a identificação de problemas e a integração de componentes como mensageria, resiliência e multitenancy.



AVISO: O dashboard deve ser usado **estritamente em ambientes de desenvolvimento**. Seu uso em produção é proibido, pois concede acesso a dados sensíveis e representa um risco de segurança.

Configuração e Acesso em Ambiente Local

Passo 1: Adicionar a Dependência Maven



Incluir a feature `cds-feature-dev-dashboard` no `pom.xml`.

```
<dependency>
  <groupId>com.sap.cds</groupId>
  <artifactId>cds-feature-dev-dashboard</artifactId>
</dependency>
```

Passo 2: Configurar Autorização



O acesso ao dashboard requer a role `cds.Developer`. A configuração padrão de mock users já inclui um usuário `developer` com essa role. Para usuários customizados, adicione a role explicitamente no `application.yaml`.

```
cds:
  security:
    mock:
      users:
        - name: myUser
          password: myPass
          roles:
            - cds.Developer
```

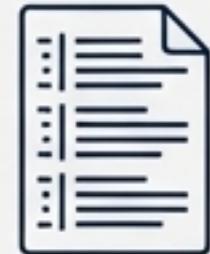
Para cenários de desenvolvimento na nuvem (Cloud Foundry), é necessário desativar o perfil de produção e ajustar o `xs-security.json` e `xs-app.json` para incluir a role e as rotas do dashboard.

Seu Toolkit Completo para Aplicações CAP Java



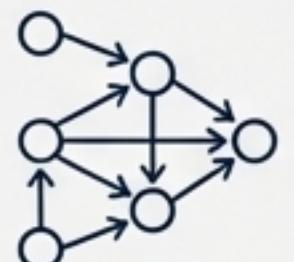
Profiling (JMX, jcmand)

Análise Profunda de Performance
(Identificar gargalos de CPU/Memória)



Logging (SLF4J, Logback)

Histórico de Eventos
(Rastrear e depurar fluxos de execução)



Tracing (OpenTelemetry)

Visualização de Fluxo Distribuído
(Entender a jornada de uma requisição entre serviços)



Métricas (Actuators)

Monitoramento de Saúde
(Verificar o ‘pulso’ da aplicação e de suas dependências)



Developer Dashboard

Aceleração do Desenvolvimento
(Centralizar controle e visibilidade no ambiente local)