Introdução ao problema das configurações

Pensemos em uma organização que começa a crescer em escala, portanto, começa a desenvolver microsserviços atômicos que suportam os diferentes processos de negócios. Esses microsserviços que fornecem infraestrutura, processamento e suporte a dados precisam de diferentes tipos de configurações para funcionar, por exemplo:

- Localização de outros microsserviços.
- Paths banco de dados e endpoints dos serviços a serem implantados.
- Endereço IP de servidores de armazenamento, como FTP ou S3.
- Tipo de logging informações por Log4j (INFO, ERRO, DEBUG).
- Senhas e/ou segredos (na prática, a segurança adicional é aplicada neste caso).
- Variáveis ambientais.

Mas… por que essas informações precisam de uma configuração específica e não podem ser constantes dentro de uma aplicação? Veremos quais são as razões.

Desacoplar as informações de configuração do código

Um desenvolvedor não precisa saber a URL do banco de dados de produção, senhas ou outras configurações gerais que são geralmente tratadas pela infraestrutura ou DevOps. Vamos imaginar que a equipe de infraestrutura tenha que alterar uma URL interna. Não faria sentido pedir a cada equipe de desenvolvimento para reimplantar, pois os sistemas devem conseguir reagir a essas alterações de configuração.

Separação de ambientes (produção, QA, desenvolvimento)

Normalmente, existem pelo menos três ambientes:

- **Produção** (onde o código que o usuário final vê é executado)
- **QA** (idem, mas para a equipe de QA)
- **dev** (para a equipe de desenvolvimento)

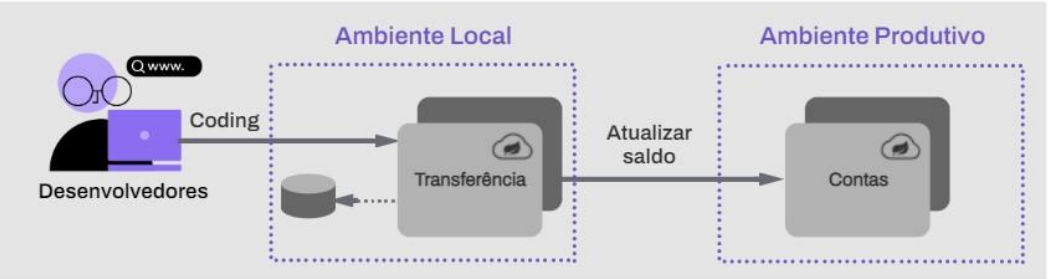
Cada ambiente tratará de sua própria configuração.
Embora isso possa ser tratado com arquivos (e, na prática, e até certo nível, isso é), para determinadas configurações (várias instâncias) e determinado número de microsserviços, é necessário centralizar as configurações.

Impedir deploys devido a alterações de configuração

Suponhamos um cenário onde várias configurações estão incorporadas no código. Existem configurações mantidas por anos, outras por meses e outras por semanas. A diferença entre ter a configuração embutida no código ou separada é que a segunda não precisa de deploys, apenas reinicia. Se um processo de implantação completo leva 6 minutos por instância e uma reinicialização leva 2 minutos, a vantagem é óbvia.

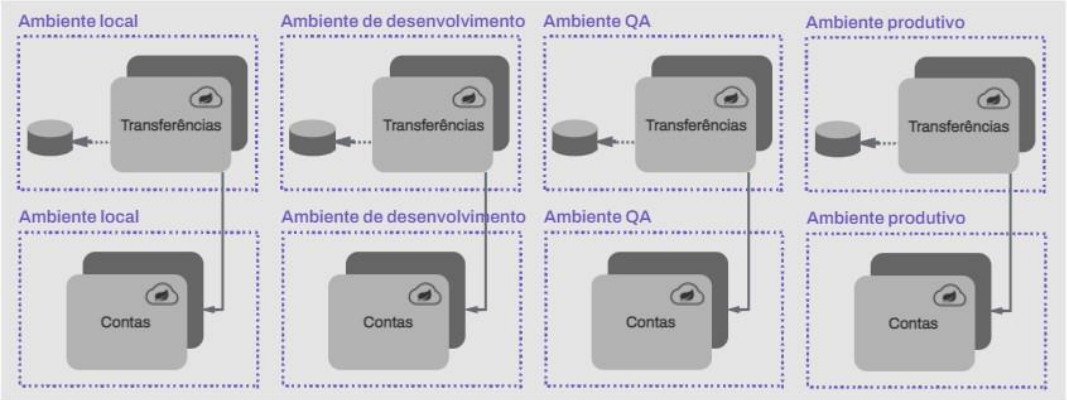
Conheça um exemplo →

Vamos imaginar que estamos desenvolvendo a funcionalidade de transferências entre contas em nossa máquina e precisamos atualizar o saldo das contas afetadas usando o serviço desenvolvido para esse fim:



Precisamos ter um serviço de conta estável que garanta que a operação seja realizada e depois verifique por um teste os saldos atualizados. Se tivéssemos apenas microsserviços em um ambiente real, acabaríamos impactando usuários finais, ou seria um erro grave.

É obrigatório ter vários ambientes de trabalho, onde dispomos de toda a infraestrutura, serviços, base de dados e recursos necessários para garantir o **correto e integral** funcionamento do nosso sistema. Ou seja, se eu mover $ 100 da conta A para a conta B, no final da transação, a conta B deve ter mais $ 100 e a conta A - $ 100.
Poderíamos representar essa necessidade da seguinte forma:



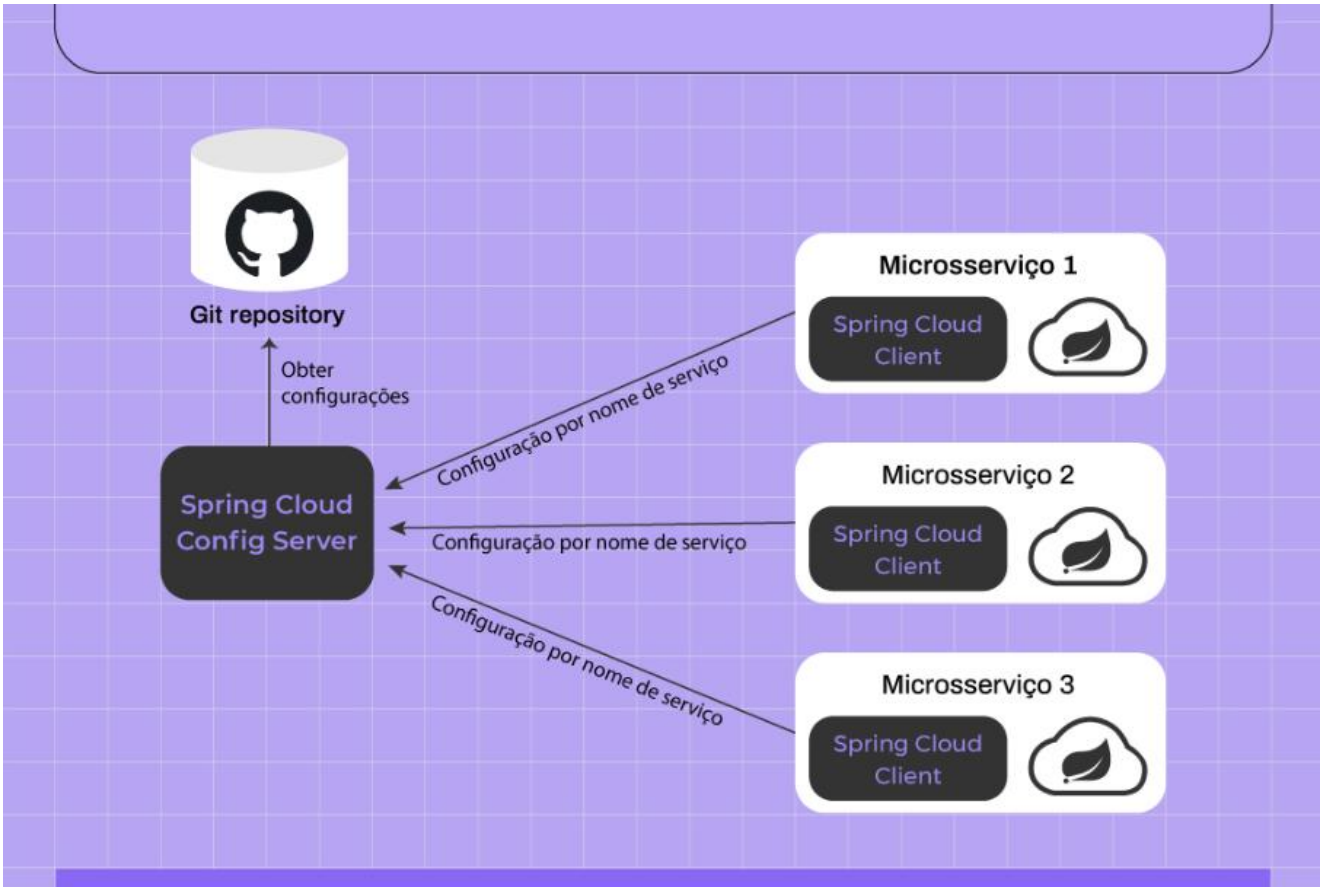
Analisando o diagrama, vemos que nosso microsserviço de transferência acessa um microsserviço de conta diferente (dependendo do ambiente onde está localizado), e essa capacidade não deve ser uma constante especificada no nível do código, pois é variável dependendo do ambiente e pode também sofrer alterações. No mesmo ambiente, ao alterar, por exemplo, o URL final.

Se essa configuração fosse programada e não configurada externamente, teríamos que recompilar o projeto do zero para cada deploy em cada ambiente, diminuindo abruptamente o TTM visto na primeira aula.

Solução para configurações distribuídas com Spring Cloud Config

A solução Spring Cloud nos fornece um framework chamado Spring Cloud Config que facilita a configuração dos microsserviços e ambientes mencionados anteriormente. Para isso, ele nos fornece dois componentes básicos: Config Server e Spring Cloud Client. Estes interagem entre si para cumprir sua missão, como vemos no diagrama a seguir:

Veremos agora como integrar o Spring Cloud Config ao Git para lidar com configurações distribuídas.



Conclusão

O Spring Cloud Config fornece uma excelente solução para as configurações que precisamos efetuar em sistemas distribuídos, realizando-as externamente ao próprio microsserviço e centralmente para todos os microsserviços.

Codificando juntos

Agora veremos como todos os conceitos que aprendemos nesta aula se reúnem no vídeo de live coding, onde mostraremos como implementar o Spring Cloud Config passo a passo. Em seguida, sugerimos que você faça um exercício para colocar em prática o que viu.