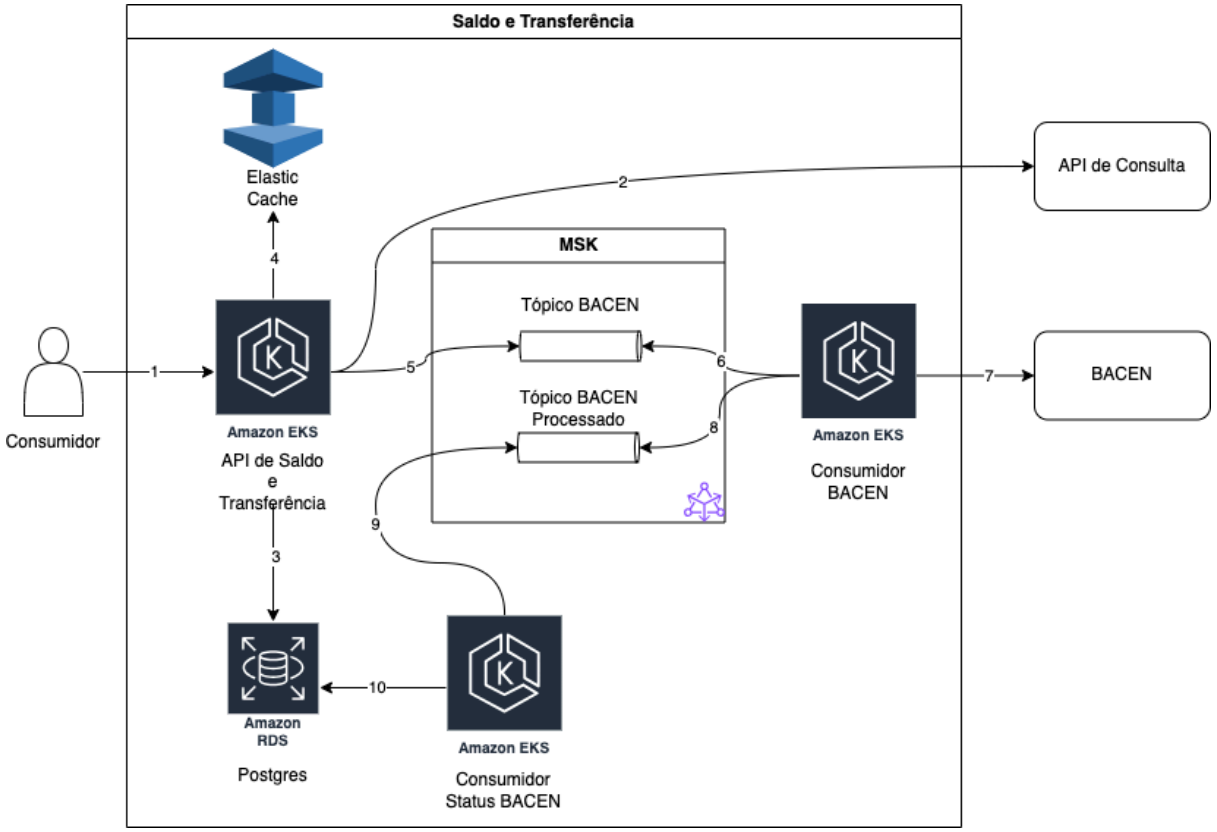
Desafio de Arquitetura

Foi escolhido seguir modelos arquiteturais focados em resiliência, escalabilidade e tolerância parcial do projeto do desafio.

Abaixo temos o desenho da arquitetura proposta com peças em sua maioria AWS.



Para conferência de Saldo

1. Request de consulta de saldo
2. Consulta o nome do Cliente
3. Consulta o Saldo no Banco de Dados
4. Retorno ao Consumidor

Para transferência

1. Request de solicitação de transferência
2. Consulta o nome do Cliente
3. Consulta o Saldo no Banco de Dados
4. Consulta o limite diário disponível
5. Envia evento de notificação ao BACEN
6. Consome o evento de notificação
7. Faz request síncrono ao Bacen
8. Envia evento de transferência processada pelo BACEN
9. Consome o evento de transferência processada

10. Atualiza base de dados da API de Saldo e Transferência

Motivação para escolha da Arquitetura

1. Foi escolhido utilização do EKS como container da aplicação por ser Kubernetes e em caso de oscilação de carga pode escalar horizontalmente o serviço que estamos propondo, tanto consumidores, quanto a própria API.
2. A parte de observabilidade não foi tão explorada para ter foco na solução propriamente dita, mas poderiam ser adicionados agentes do logstash para enviar os dados de log para uma solução ELK ou em caso de utilização de ferramentas terceiras como Dynatrace ou Datadog instalar agentes em todos os serviços
3. Foi escolhido um banco de dados relacional por se tratar de uma transação, então seria interessante termos garantido o ACID (Atomicidade. Consistência, Isolamento e Durabilidade). A escolha pelo Postgres se dá por ser open source, e apesar de uma indicação pela AWS, caso haja necessidade de construção em outra cloud pública ou em uma privada não invalida a solução.
4. A caching será utilizada para controlar o limite diário de transferências e justifica-se por ser uma informação simplista de consumo rápido com validade de um dia, podendo assim recorrer uma forma de storage um pouco mais volátil.
5. Para que o tempo de 100ms na transação seja respeitado vamos fazer uso do teorema de CAP (Consistency, Availability and Partial Tolerance), onde teremos a disponibilidade do Kafka e sua tolerância parcial como componentes importantes para responder ao cliente assim que o evento de envio para o BACEN for gerado, assim desacoplamos o tempo de resposta com a API do BACEN, utilização da cache para o limite de diário e podemos realizar em ordem de velocidade as verificações para caso alguma falhe a resposta seja mais rápida.
6. Para o menor impacto possível do cliente o ideal é realizar tudo de maneira assíncrona guardando o estado anterior, porém acredito que devamos manter síncrono a verificação do cliente pois em caso de falha de verificação o processo não deveria continuar, pois seria mais impactante para o cliente uma fraude do que o serviço de clientes fora do ar.
7. Para não depender da API do BACEN, única API externa aos domínios do banco creio que a solução de criar um tópico para notificação e um consumidor com o trabalho de enviar as notificações para o BACEN desacoplam as responsabilidade e fazemos uso da tolerância parcial.