Sugestão de respostas para o EE3 – 2024.2

A **NeoTech** é uma startup de tecnologia que desenvolveu uma plataforma SaaS para gestão de projetos colaborativos chamada **ColabFlow**. A plataforma teve um crescimento exponencial e agora precisa ser escalada para atender novos mercados, garantindo interoperabilidade com outros sistemas e modernização da arquitetura.

No entanto, a equipe de engenharia da NeoTech enfrenta diversos desafios, como a necessidade de refatorar partes do código legado, adotar uma abordagem orientada a serviços e implementar um modelo arquitetural que permita maior escalabilidade. Além disso, a equipe precisa decidir como evoluir a plataforma sem comprometer a experiência dos usuários existentes.

Os principais desafios são:

1. Começou como uma aplicação monolítica e agora precisa ser transformada em uma arquitetura mais modular e escalável;
2. Para garantir integração com ferramentas de terceiros, precisa projetar e documentar APIs seguindo boas práticas de design e versionamento;
3. Algumas partes do sistema foram desenvolvidas rapidamente e precisam passar por refatoração para garantir que continuem manuteníveis; e,
4. Como a base de usuários está crescendo, o time precisa garantir que as decisões arquiteturais levem em conta a centralidade dos dados e sua utilização eficiente.

Questão 1 (Fácil - 2,5 pontos) - Escolha do Design de API

O aluno pode escolher qualquer uma dessas arquiteturas, desde que justifique com clareza os benefícios, desafios e comparações.

REST

- **Justificativa:** REST é a escolha ideal para a API pública do ColabFlow, pois oferece simplicidade, escalabilidade e interoperabilidade. Como o ColabFlow será integrado a diferentes sistemas e clientes, uma API RESTful baseada em HTTP e JSON facilita a adoção por desenvolvedores externos.
- **Comparação:** Embora GraphQL seja uma alternativa interessante para consultas mais flexíveis, REST é mais simples de implementar e tem melhor suporte a cache e controle de versionamento.
- **Desafios de Implementação:** Gerenciar versionamento de endpoints pode ser um problema, além da necessidade de evitar sobrecarga com muitas chamadas HTTP.

GraphQL

- **Justificativa:** GraphQL é vantajoso pois permite que os clientes especifiquem exatamente quais dados desejam, reduzindo requisições desnecessárias e tornando as interações mais eficientes. Isso seria útil no ColabFlow, onde os usuários podem ter diferentes necessidades de consulta.
- **Comparação:** Diferente de REST, onde os endpoints retornam dados fixos, GraphQL evita over-fetching e under-fetching. No entanto, a complexidade da implementação e a necessidade de uma camada adicional de processamento podem ser desafios.
- **Desafios de Implementação:** GraphQL pode gerar carga excessiva no servidor se não for bem projetado e não possui suporte nativo a cache em HTTP.

SOAP

- **Justificativa:** SOAP seria uma escolha adequada se a NeoTech precisasse de transações seguras e alto nível de confiabilidade, como integrações com bancos ou setores regulamentados. Ele possui suporte embutido para WS-Security e transações ACID, tornando-o mais seguro que REST para certos casos.
- **Comparação:** REST é mais leve e mais fácil de integrar com aplicações modernas, mas não oferece os mesmos recursos robustos de segurança e transacionalidade que SOAP.
- **Desafios de Implementação:** SOAP é mais verboso e pesado, aumentando a latência da API, além de exigir mais esforço para adoção devido ao uso de XML e WSDL.

gRPC

- **Justificativa:** Se o ColabFlow precisasse de baixa latência e comunicação eficiente entre microserviços, gRPC seria uma excelente escolha. Ele usa protocol buffers (Protobuf) em vez de JSON, reduzindo o tamanho das mensagens e aumentando a performance.
- **Comparação:** Diferente de REST e GraphQL, que usam HTTP/JSON, gRPC usa HTTP/2, permitindo comunicação bidirecional e streaming. No entanto, gRPC não é ideal para APIs públicas, pois requer um cliente especializado para consumir os serviços.
- **Desafios de Implementação:** O suporte a navegadores web é limitado, e a curva de aprendizado para desenvolvedores pode ser maior devido ao uso do Protobuf.

WebSockets

- **Justificativa:** Se o ColabFlow exigisse comunicação em tempo real, como notificações instantâneas e colaboração ao vivo, WebSockets seria uma excelente escolha. Ele permite uma conexão persistente entre o cliente e o servidor, evitando a latência de múltiplas requisições HTTP.
- **Comparação:** REST e GraphQL não são ideais para comunicação em tempo real, pois exigem polling ou webhooks. No entanto, WebSockets pode ser mais difícil de escalar do que APIs RESTful convencionais.
- **Desafios de Implementação:** Gerenciar milhares de conexões simultâneas pode ser problemático, exigindo infraestrutura especializada e mecanismos como balanceamento de carga.

MQTT

- **Justificativa:** Se o ColabFlow tivesse integração com dispositivos IoT, como sensores ou dispositivos conectados, MQTT seria a melhor escolha. Ele é um protocolo leve e eficiente, projetado para comunicação assíncrona em redes instáveis.
- **Comparação:** WebSockets poderia ser usado para comunicação em tempo real, mas MQTT é mais eficiente para ambientes de baixo consumo de energia e alta latência, como IoT.
- **Desafios de Implementação:** MQTT exige infraestrutura de um broker (como Mosquitto ou EMQX), e a compatibilidade com navegadores é limitada.

Questão 2 (Intermediária - 3,5 pontos) - DDSD e Arquitetura Baseada em Dados

A adoção do Data-Driven Software Development (DDSD) na NeoTech envolve desafios significativos, especialmente porque o ColabFlow precisa processar grandes volumes de dados de maneira eficiente. Os principais desafios arquiteturais incluem:

Escalabilidade da Persistência de Dados:

- O DDSD exige a coleta e o armazenamento contínuo de métricas e logs do sistema. Bancos de dados relacionais tradicionais podem ter dificuldades para lidar com essa carga crescente.
- **Possível solução**: Utilização de bancos NoSQL (como MongoDB, Cassandra) ou Data Lakes para armazenamento distribuído e escalável.

Processamento de Dados em Tempo Real vs. Batch

- Algumas análises exigem dados em tempo real, como a detecção de padrões de uso ou alertas personalizados, enquanto outras podem ser processadas em modo batch. A escolha errada pode levar a gargalos ou a informações desatualizadas.
- **Possível solução**: Implementação de uma arquitetura híbrida combinando Apache Kafka ou Apache Flink para processamento em tempo real e Hadoop/Spark para processamento batch.

Governança e Qualidade dos Dados

- Garantir que os dados coletados sejam precisos, completos e consistentes é essencial para evitar insights errôneos. Além disso, a empresa deve se preocupar com LGPD e compliance na gestão dessas informações.
- **Possível solução**: Adoção de pipelines de qualidade de dados, validação automática e ferramentas de catalogação como Apache Atlas ou DataHub.

Questão 3 (Difícil - 4,0 pontos) - Escolha do Estilo Arquitetural

Nesta questão mais de uma resposta pode ser considerada correta, desde que a escolha seja bem justificada e alinhada ao contexto do ColabFlow. A modernização do sistema busca escalabilidade, modularidade e atualização independente, então algumas arquiteturas são mais adequadas que outras, mas isso não significa que só há uma resposta certa.

Estilo Arquitetural	Pode ser uma resposta válida?	Justificativa
Arquitetura em Camadas	Não recomendada	Mantém um acoplamento forte entre módulos e não facilita escalabilidade horizontal.
Pipeline	Pouco adequado	Mais útil para processamento contínuo de dados (ex.: ETL), mas não resolve modularidade e escalabilidade do sistema.
Arquitetura Microkernel	Pouco adequado	Útil para sistemas com core mínimo e plugins (ex.: IDEs e ERPs), mas não resolve bem escalabilidade e modularidade do ColabFlow.
Service-Based Architecture	Válida	Divide o sistema em grupos de serviços, reduzindo acoplamento sem a complexidade total de microsserviços. Boa opção para uma transição gradual.
Event-Driven Architecture	Válida	Permite baixo acoplamento e escalabilidade, pois os serviços reagem a eventos. Útil se o ColabFlow precisar de processamento assíncrono (ex.: notificações, fluxos de atividades).
Arquitetura Space-Based	Válida	Boa para altas cargas e escalabilidade horizontal, eliminando gargalos de banco de dados. Útil se o ColabFlow crescer exponencialmente.
Arquitetura Orientada a Serviços Baseada em Orquestração	Válida	Permite modularidade com controle centralizado sobre a interação entre serviços. Boa opção para empresas que precisam de governança forte.
Microservices Architecture	Válida	Ideal para modularidade e escalabilidade, mas aumenta a complexidade. Seria recomendada se a NeoTech tiver maturidade para lidar com microsserviços.

Se o aluno optar pela escolha de um estilo inadequado, mas justificada com um cenário muito específico, a resposta pode não ser totalmente zerada, mas deve perder pontos pela falta de alinhamento com os desafios da NeoTech.

Possíveis respostas esperadas para cada escolha válida:

Service-Based Architecture
Justificativa: A Service-Based Architecture (SBA) é uma evolução do monólito tradicional, separando o sistema em grupos de serviços coesos, que se comunicam por APIs. Essa abordagem permite que a NeoTech modernize o ColabFlow de forma gradual, sem a complexidade total de uma migração para microsserviços.
Benefícios:

- Baixo acoplamento interno entre módulos, permitindo que equipes trabalhem em partes separadas do sistema.
- Facilidade de manutenção e atualização de componentes sem impacto generalizado.
- Menor complexidade que microsserviços, facilitando a transição arquitetural sem a necessidade de uma grande reestruturação.

Desafio e Mitigação: Gerenciamento da comunicação entre serviços: Se não for bem projetada, a arquitetura pode gerar dependências excessivas entre serviços, dificultando a escalabilidade.
Solução: Utilizar um API Gateway e definir contratos claros entre serviços para minimizar acoplamento desnecessário.

Event-Driven Architecture
Justificativa: A Event-Driven Architecture (EDA) é ideal para o ColabFlow, pois permite que componentes se comuniquem de forma assíncrona, reduzindo a dependência direta entre serviços. Isso melhora a escalabilidade e a capacidade do sistema de lidar com grandes volumes de eventos sem gargalos.
Benefícios:

- Baixo acoplamento: Os serviços respondem a eventos, em vez de dependerem diretamente uns dos outros.
- Alta escalabilidade: Permite a distribuição da carga de trabalho, evitando gargalos de processamento.
- Maior resiliência: Se um serviço falhar, os eventos podem ser processados assim que ele for restaurado.

Desafio e Mitigação: Gerenciamento da consistência dos dados: Como os eventos são processados de forma assíncrona, pode haver problemas de sincronização entre serviços.
Solução: Implementar **event sourcing e CQRS** para garantir que todos os serviços tenham acesso aos dados mais atualizados.

Arquitetura Space-Based
Justificativa: A Arquitetura Space-Based (SBA) é uma excelente opção para sistemas que precisam escalar horizontalmente e evitar gargalos de banco de dados, como o ColabFlow. Em vez de um banco centralizado, os dados são distribuídos em vários nós, permitindo maior resiliência e desempenho.
Benefícios:

- Alta escalabilidade horizontal, reduzindo a dependência de um único banco de dados.
- Melhor tolerância a falhas, pois os dados são replicados em diferentes nós do sistema.
- Desempenho otimizado, eliminando gargalos típicos de bancos de dados tradicionais.

Desafio e Mitigação: Gerenciamento da consistência dos dados: Como os dados são distribuídos, pode ser difícil garantir que todos os nós tenham informações atualizadas.
Solução: Implementar estratégias de replicação e sincronização, como sharding e caching distribuído.

Arquitetura Orientada a Serviços Baseada em Orquestração

Justificativa: A Arquitetura Orientada a Serviços Baseada em Orquestração é ideal para sistemas que precisam de governança e controle sobre a comunicação entre serviços. No caso do ColabFlow, isso garantiria que todas as interações entre módulos sigam um fluxo bem definido.
Benefícios:

- Governança centralizada, garantindo controle sobre os fluxos de comunicação.
- Facilidade de monitoramento e auditoria das interações entre serviços.
- Menos complexidade operacional do que uma abordagem totalmente distribuída como microsserviços.

Desafio e Mitigação: Ponto único de falha no orquestrador: Se o mecanismo de orquestração falhar, os serviços podem parar de se comunicar.
Solução: Implementar failover automático e replicação do mecanismo de orquestração para evitar falhas críticas.

Microservices Architecture
Justificativa: A Arquitetura de Microsserviços é uma escolha avançada para o ColabFlow, pois permite que cada funcionalidade seja desenvolvida e implantada independentemente. Essa abordagem favorece escalabilidade e inovação rápida, desde que a empresa tenha maturidade técnica para gerenciá-la.
Benefícios:

- Alta modularidade, permitindo que diferentes equipes trabalhem em serviços independentes.
- Escalabilidade sob demanda, onde cada microsserviço pode ser dimensionado individualmente.
- Maior resiliência, pois a falha de um serviço não derruba todo o sistema.

Desafio e Mitigação: Complexidade operacional: Gerenciar dezenas (ou centenas) de microsserviços pode ser desafiador.
Solução: Utilizar Service Mesh (ex.: Istio) para gerenciar a comunicação entre microsserviços e automação com CI/CD e Kubernetes.