

Arquitetura para Sistema de Fluxo de Caixa

Janeiro de 2025

Por Sergio Figueiredo

Contextualização do problema

Um comerciante precisa controlar o seu fluxo de caixa diário com os lançamentos (débitos e créditos), também precisa de um relatório que disponibilize o saldo diário consolidado.

Para resolver este desafio, será preciso desenvolver uma arquitetura que integre processos e sistemas de forma eficiente, garantindo a entrega de valor para a organização. Isso inclui a definição de contextos, capacidades de negócio e domínios funcionais, escalabilidade das soluções para garantir alta disponibilidade, segurança e desempenho, a comunicação eficaz entre áreas e serviços, a seleção adequada de padrões arquiteturais, integração de tecnologias e frameworks, além de otimização de requisitos não-funcionais. Deve abranger a capacidade analítica, a visão sistêmica e a habilidade de criar soluções flexíveis e reutilizáveis.

Domínios Funcionais

1. Gestão de Lançamentos

Esse domínio é responsável por registrar os lançamentos financeiros de débito ou crédito no sistema.

- **Objetivo:** Registrar e processar as transações financeiras do comerciante.
- **Funções principais:**
 - Registrar transações de débito e crédito.
 - validar dados de transações.
 - publicar eventos para atualização de saldo.

2. Gestão de Saldos

Esse domínio cuida do cálculo, atualização e armazenamento dos saldos após cada lançamento.

- **Objetivo:** Calcular e manter os saldos consolidados diários.
- **Funções principais:**
 - Processar lançamentos em tempo real assincronamente.
 - consolidar saldos diários.
 - Armazenar saldos no banco de dados.

3. Geração de Relatórios

Este domínio é responsável por gerar os relatórios financeiros solicitados pelo usuário.

- **Objetivo:** Criar relatórios financeiros em diferentes formatos (CSV ou PDF) para consulta do cliente.
- **Funções principais:**
 - Gerar relatórios de saldo de acordo com um período.
 - Verificar e validação da existência de relatórios pré-gerados.
 - Armazenar e recuperar relatórios na nuvem (Cloud Storage).

4. Autenticação e Autorização

Esse domínio gerencia a autenticação de usuários e controle de acesso no sistema.

- **Objetivo:** Garantir que somente usuários autenticados possam acessar os serviços.
- **Funções principais:**
 - Autenticar usuários.
 - Gerir permissões e acesso aos serviços do sistema.

5. Integração com APIs Externas

Esse domínio gerencia a comunicação com sistemas externos, como o Apigee para roteamento de requisições e o Google Pub/Sub para eventos.

- **Objetivo:** Facilitar a integração com outros sistemas e serviços externos.
- **Funções principais:**
 - rotear requisições via Apigee.
 - Publicar e consumir eventos do Google Pub/Sub.

6. Monitoramento e Logging

Esse domínio cuida do monitoramento e da geração de logs do sistema para rastreabilidade e depuração.

- **Objetivo:** Registrar e monitorar todas as operações do sistema para garantir rastreabilidade.
- **Funções principais:**
 - registrar eventos e transações no Google Cloud Logging.
 - Monitorar falhas e eventos no sistema.

7. Gerenciamento de Infraestrutura

Esse domínio abrange as operações de infraestrutura, incluindo o gerenciamento de microsserviços, containerização e orquestração.

- **Objetivo:** Garantir a escalabilidade e a resiliência do sistema.
- **Funções principais:**
 - Gerenciar microsserviços com Docker e Kubernetes.
 - Provisionar recursos na Google Cloud.
 - Escalar e balancear carga para alta disponibilidade.

Capacidades de Negócio

1. Capacidades de Gestão de Lançamentos

- **Registrar transações financeiras:** Permite que o cliente registre lançamentos de débito ou crédito no sistema, especificando o valor, tipo e data da transação.
- **Validar transações:** Verifica a conformidade dos dados de transações (tipo, valor, etc.) antes de registrá-las.
- **Publicar eventos de lançamento:** Envia eventos sobre a transação (via Google Pub/Sub) para que outros sistemas, como o serviço de atualização de saldo, possam processá-la.
- **Confirmar a execução do lançamento:** Retorna para o front-end com o status de sucesso ou falha do lançamento realizado.
- **Armazenar transações no banco de dados:** Persiste as transações financeiras no banco SQL Server para controle histórico e auditoria.

2. Capacidades de Gestão de Saldos

- **Atualizar saldo em tempo real:** Processa os lançamentos financeiros registrados e atualiza os saldos consolidados de forma dinâmica e contínua.
- **Consolidar saldos diários:** Calcula o saldo final diário a partir dos lançamentos e atualiza o valor no banco de dados.
- **Armazenar saldo diário:** Grava o saldo diário consolidado no banco SQL Server para que esteja disponível para futuras consultas e relatórios.
- **Processar eventos de lançamento de saldo:** Consome os eventos publicados no Google Pub/Sub e os utiliza para calcular e atualizar os saldos de forma automática e assíncrona.
- **Garantir tolerância a falhas:** Em caso de falha no processo de atualização de saldo, o sistema deve permitir que o serviço de lançamento continue funcionando sem interrupções.

3. Capacidades de Geração de Relatórios

- **Gerar relatórios financeiros:** Cria relatórios de saldos consolidados para um período determinado (diário, mensal, etc.), no formato escolhido (CSV ou PDF).
- **Verificar existência de relatórios:** Antes de gerar um novo relatório, verifica se um relatório já existe para o período solicitado no Cloud Storage.
- **Armazenar relatórios gerados:** Armazena os relatórios gerados na Cloud Storage e retorna a URL do arquivo gerado para o front-end, facilitando o acesso do usuário.
- **Consultar dados históricos de saldos:** A API de relatórios consulta os dados do SQL Server para recuperar informações de saldo do período solicitado.
- **Facilitar o download de relatórios:** Permite que os usuários baixem os relatórios diretamente do front-end, fornecendo links para o arquivo gerado.

4. Capacidades de Autenticação e Autorização

- **Autenticar usuários:** Valida a identidade do usuário
- **Gerenciar permissões de acesso:** Controla o nível de acesso de cada usuário.
- **Monitorar sessões de usuários:** Rastreia e valida as sessões ativas dos usuários

5. Capacidades de Integração com APIs Externas

- **Roteamento de requisições via Apigee:** O Apigee serve como um gateway de API, roteando requisições do front-end para os microsserviços backend.
- **Publicação de eventos para sistemas externos:** Permite a comunicação assíncrona entre microsserviços via Google Pub/Sub, facilitando a integração entre sistemas de forma desacoplada.
- **Receber notificações de eventos:** Consumir eventos publicados no Google Pub/Sub e responder com as operações apropriadas (atualização de saldo, geração de relatório, etc.).

6. Capacidades de Monitoramento e Logging

- **Registrar eventos do sistema:** Toda operação crítica (como transações, atualizações de saldo e geração de relatórios) deve ser registrada no Google Cloud Logging para monitoramento e rastreabilidade.
- **Monitorar a performance do sistema:** Monitoramento da saúde dos microsserviços e do sistema como um todo, identificando gargalos ou falhas.
- **Detectar e alertar sobre falhas:** Detecta falhas e notifica os responsáveis para a correção rápida, com alertas configuráveis baseados em métricas do sistema.

7. Capacidades de Infraestrutura e Orquestração

- **Gerenciar microsserviços com Kubernetes:** Utiliza o Kubernetes para orquestrar os microsserviços, garantindo a escalabilidade e alta disponibilidade do sistema.
- **Containerizar microsserviços com Docker:** Cada microsserviço é containerizado para garantir que sejam independentes, portáteis e facilmente escaláveis.
- **Escalabilidade automática:** Implementa mecanismos de escalabilidade automática (auto-scaling) para lidar com picos de demanda, como quando o serviço de geração de relatórios recebe um grande número de requisições.
- **Balanceamento de carga:** Implementa algoritmos de balanceamento de carga para distribuir as requisições de forma eficiente entre os microsserviços, minimizando o risco de sobrecarga em qualquer parte do sistema.
- **Provisionamento de recursos na Google Cloud:** Gerencia a infraestrutura de nuvem, utilizando os recursos da Google Cloud para garantir disponibilidade, escalabilidade e segurança.

Requisitos Funcionais Refinados

1. Requisitos Funcionais de Lançamento de Valores

- **RF1. Registro de transações financeiras:** O sistema deve permitir que o usuário registre transações financeiras, especificando o tipo (débito ou crédito), valor e data do lançamento.
- **RF2. Validação de transações:** O sistema deve validar se os dados da transação (valor, tipo e data) são válidos antes de registrá-los.
- **RF3. Armazenamento de transações no banco de dados:** O sistema deve gravar as transações financeiras no banco de dados SQL Server, para garantir a persistência e integridade dos dados.
- **RF4. Publicação de eventos no Google Pub/Sub:** O sistema deve publicar um evento no Google Pub/Sub sempre que uma transação for registrada, para que o serviço de atualização de saldo possa ser acionado automaticamente.
- **RF5. Retorno de status para o front-end:** Após o lançamento da transação, o sistema deve retornar um status (sucesso ou falha) para o front-end para que o usuário saiba o resultado da operação.
- **RF6. Registro de eventos no Google Cloud Logging:** O sistema deve registrar todos os eventos de transações no Google Cloud Logging, para garantir rastreabilidade e auditoria.

2. Requisitos Funcionais de Atualização de Saldo

- **RF7. Processamento de eventos de lançamento:** O sistema deve consumir os eventos de lançamento (publicados no Google Pub/Sub) e processá-los para atualizar os saldos do comerciante.
- **RF8. Cálculo do saldo diário:** O sistema deve calcular o saldo diário com base nos lançamentos registrados, considerando os débitos e créditos.
- **RF9. Armazenamento de saldos no banco de dados:** O saldo diário consolidado deve ser armazenado no banco de dados SQL Server para garantir persistência e integridade.
- **RF10. Atualização em tempo real:** O sistema deve atualizar o saldo em tempo real assim que um lançamento for processado.
- **RF11. Garantir tolerância a falhas:** O sistema deve garantir que o serviço de lançamento de valores continue funcionando mesmo se o serviço de atualização de saldo estiver temporariamente fora de operação.
- **RF12. Registro de operações no Google Cloud Logging:** Todas as operações de atualização de saldo devem ser registradas no Google Cloud Logging para fins de rastreabilidade e auditoria.

3. Requisitos Funcionais de Geração de Relatórios

- **RF13. Geração de relatórios financeiros:** O sistema deve permitir que o usuário gere relatórios financeiros para um período especificado, com base nos saldos consolidados.
- **RF14. Formatos de relatórios (CSV/PDF):** O sistema deve permitir que o usuário escolha o formato do relatório a ser gerado, podendo ser CSV ou PDF.

- **RF15. Verificação de relatórios existentes:** O sistema deve verificar se um relatório já foi gerado para o período solicitado, consultando a Cloud Storage.
- **RF16. Armazenamento de relatórios na Cloud Storage:** O relatório gerado deve ser armazenado na Cloud Storage, e a URL do arquivo gerado deve ser retornada para o front-end.
- **RF17. Retorno da URL do relatório para o front-end:** O sistema deve retornar a URL do relatório gerado para que o usuário possa visualizá-lo e/ou baixá-lo.
- **RF18. Registro de geração de relatório no Google Cloud Logging:** Cada geração de relatório deve ser registrada no Google Cloud Logging para rastreabilidade.

4. Requisitos Funcionais de Autenticação e Autorização

- **RF19. Autenticação de usuários:** O sistema deve permitir que os usuários se autenticem por meio de um mecanismo seguro (como token JWT ou OAuth).
- **RF20. Controle de permissões:** O sistema deve controlar as permissões de acesso dos usuários, garantindo que apenas usuários autorizados possam realizar lançamentos financeiros e acessar relatórios.
- **RF21. Gestão de sessões de usuários:** O sistema deve gerenciar as sessões de usuários autenticados, garantindo a segurança no acesso aos dados.

5. Requisitos Funcionais de Integração com APIs Externas

- **RF22. Roteamento de requisições via Apigee:** O sistema deve usar o Apigee para rotear as requisições do front-end para os microserviços backend, cuidando de autenticação, segurança e tráfego.
- **RF23. Publicação e consumo de eventos via Google Pub/Sub:** O sistema deve permitir a publicação de eventos no Google Pub/Sub para integração entre microserviços, como a atualização de saldo e a geração de relatórios.
- **RF24. Comunicação assíncrona entre microserviços:** O sistema deve garantir a comunicação assíncrona entre os microserviços, utilizando o Google Pub/Sub para processar eventos de forma desacoplada.

6. Requisitos Funcionais de Monitoramento e Logging

- **RF25. Registro de eventos do sistema no Google Cloud Logging:** O sistema deve registrar eventos importantes (como lançamentos, atualizações de saldo e geração de relatórios) no Google Cloud Logging para garantir rastreabilidade e depuração.
- **RF26. Monitoramento de falhas:** O sistema deve monitorar a ocorrência de falhas em qualquer parte da operação e registrar eventos de erro para diagnóstico e correção.
- **RF27. Alerta sobre falhas críticas:** O sistema deve gerar alertas em caso de falhas críticas que possam afetar o funcionamento do sistema (por exemplo, falha no cálculo de saldo ou geração de relatório).

7. Requisitos Funcionais de Infraestrutura e Orquestração

- **RF28. Containerização de microserviços:** O sistema deve usar Docker para containerizar todos os microserviços, garantindo portabilidade e independência entre os serviços.

- **RF29. Orquestração de microsserviços com Kubernetes:** O sistema deve usar o Kubernetes para orquestrar os microsserviços, permitindo escalabilidade, alta disponibilidade e gerenciamento eficiente.
- **RF30. Escalabilidade automática:** O sistema deve ser capaz de escalar automaticamente os microsserviços conforme a demanda, para garantir a disponibilidade em picos de requisição.
- **RF31. Balanceamento de carga:** O sistema deve garantir que o tráfego seja distribuído de forma equilibrada entre os microsserviços, utilizando um mecanismo de balanceamento de carga eficiente.

Requisitos não funcionais Refinados

1. Desempenho

- **RNF1. Tempo de resposta das transações:** O sistema deve garantir que o tempo de resposta para o lançamento de valores não ultrapasse 2 segundos, mesmo em alta carga de requisições.
- **RNF2. Capacidade de processamento de requisições:** O sistema deve ser capaz de processar até 50 requisições por segundo durante os picos de geração de relatórios, com no máximo 5% de perda de requisições.
- **RNF3. Latência mínima para atualização de saldo:** O serviço de atualização de saldo deve processar e refletir as transações em tempo real, com latência inferior a 1 segundo após o envio do evento.

2. Escalabilidade

- **RNF4. Escalabilidade horizontal:** O sistema deve ser capaz de escalar horizontalmente para suportar picos de requisições, especialmente durante o processo de geração de relatórios.
- **RNF5. Escalabilidade de microsserviços:** Cada microsserviço deve ser escalável independentemente, utilizando a infraestrutura do Kubernetes para dimensionar automaticamente os recursos necessários com base na demanda.

3. Disponibilidade e Alta Disponibilidade

- **RNF6. Alta disponibilidade:** O sistema deve garantir uma disponibilidade mínima de 99,9% (três noves), incluindo todas as APIs e microsserviços críticos.
- **RNF7. Tolerância a falhas:** Em caso de falha em algum dos microsserviços, o sistema deve ser resiliente, garantindo que os serviços críticos (como lançamento de valores) continuem funcionando sem interrupções.
- **RNF8. Recuperação automática de falhas:** O sistema deve ser capaz de se recuperar automaticamente de falhas (como falha de container ou microsserviço) utilizando o Kubernetes, minimizando o impacto no serviço.

4. Segurança

- **RNF9. Autenticação segura:** O sistema deve utilizar mecanismos seguros de autenticação para garantir que apenas usuários autenticados possam acessar funcionalidades sensíveis.
- **RNF10. Autorização e controle de acesso:** O sistema deve garantir que os usuários possuam permissões adequadas para acessar recursos específicos, como geração de relatórios ou lançamento de valores.
- **RNF11. Criptografia de dados sensíveis:** Todos os dados sensíveis (como transações financeiras) devem ser criptografados em repouso e durante a transmissão, utilizando protocolos de segurança como TLS.
- **RNF12. Proteção contra ataques:** O sistema deve ser projetado para resistir a ataques comuns, e ter monitoramento para identificar atividades suspeitas.

5. Manutenibilidade e Monitoramento

- **RNF13. Facilidade de manutenção:** O sistema deve ser modular, com microserviços independentes, permitindo que novas funcionalidades ou correções sejam implementadas sem afetar outros componentes.
- **RNF14. Registro detalhado de logs:** Todos os eventos críticos (como lançamentos, erros e falhas) devem ser registrados no Google Cloud Logging para facilitar a análise, diagnóstico e correção de problemas.
- **RNF15. Monitoramento de desempenho:** O sistema deve implementar métricas e alertas para monitorar o desempenho de cada microserviço e garantir que o sistema esteja funcionando de forma otimizada.
- **RNF16. Alertas e notificações:** O sistema deve ser capaz de enviar alertas (via e-mail, SMS ou outros canais) quando ocorrerem falhas críticas ou problemas de desempenho.

6. Usabilidade

- **RNF17. Interface intuitiva:** A interface do usuário (front-end) deve ser fácil de usar, permitindo que os usuários executem tarefas como lançamentos financeiros, consulta de saldos e geração de relatórios sem dificuldades.
- **RNF18. Feedback imediato para o usuário:** O sistema deve fornecer feedback imediato para o usuário em casos de sucesso ou erro nas operações de lançamento de valores, geração de relatórios e outros processos.

7. Portabilidade

- **RNF19. Compatibilidade com múltiplos ambientes de nuvem:** O sistema deve ser implementado de forma a permitir sua execução em diferentes ambientes de nuvem, com foco na Google Cloud, mas também podendo ser facilmente adaptado para outras plataformas, se necessário.
- **RNF20. Independência de plataforma:** Os microserviços e containers (Docker) devem ser portáteis, podendo ser executados em diferentes máquinas ou clusters sem modificações significativas no código.

8. Compliance e Auditoria

- **RNF21. Conformidade regulatória:** O sistema deve estar em conformidade com as regulamentações locais sobre dados financeiros, como a Lei Geral de Proteção de Dados (LGPD) no Brasil ou o Regulamento Geral de Proteção de Dados (GDPR) na Europa, garantindo a privacidade e proteção dos dados dos usuários.
- **RNF22. Auditoria e rastreabilidade:** O sistema deve permitir auditoria completa, com registros de todas as ações realizadas (como lançamentos financeiros, atualizações de saldo e gerações de relatórios), para que seja possível rastrear e revisar qualquer operação, caso necessário.

9. Custo e Eficiência Operacional

- **RNF23. Eficiência de uso de recursos:** O sistema deve ser eficiente no uso de recursos computacionais (como CPU e memória), evitando desperdício de recursos na Google Cloud.

- **RNF24. Otimização de custos na nuvem:** A infraestrutura na Google Cloud deve ser configurada de forma a otimizar os custos operacionais, utilizando práticas como escalabilidade automática e escolha adequada de serviços.

10. CI/CD (Integração Contínua e Entrega Contínua)

- **RNF25. Automação de testes e deploy:** O sistema deve implementar pipelines de CI/CD para garantir que novas versões dos microsserviços sejam testadas automaticamente e implantadas sem intervenção manual, assegurando que o código em produção esteja sempre atualizado e livre de erros.
- **RNF26. Testes automatizados:** O sistema deve ser integrado com ferramentas de testes automatizados para garantir a qualidade e estabilidade do software, com cobertura para testes unitários, de integração e de performance.

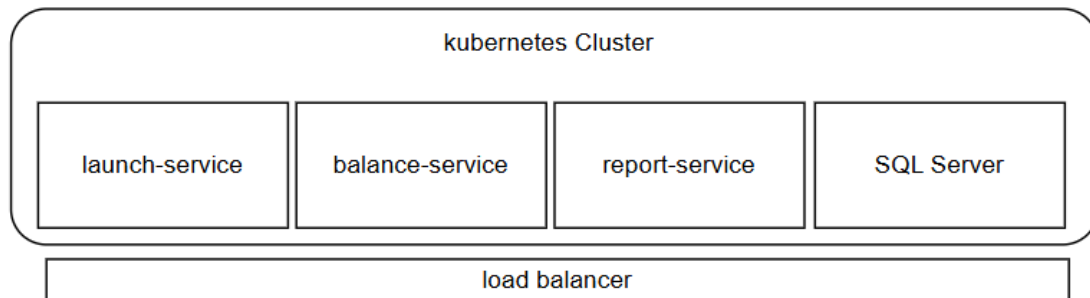
Padrões Arquiteturais

O padrão arquitetural dominante desta solução é arquitetura **baseada em microserviços** por apresentar microserviços independentes que se comunicam entre si por meio de APIs. Entretanto existem outros de padrões presentes nessa solução como por exemplo **arquitetura de nuvem** (Cloud-Native), já que toda solução de back-end está hospedado na nuvem do Google e utiliza de componentes nativos como Apigee, Cloud Storage, Cloud Logging e Pub/Sub. As camadas de apresentação, Aplicação e banco se encontram bem separadas e divididas no Desenho de Solução o que também caracteriza uma **arquitetura de camadas**.

Quadro comparativo de vantagens e desvantagens:

Padrão Arquitetural	Vantagens	Desvantagens
Arquitetura de Mcrosserviços	Alta escalabilidade; baixo acoplamento; resiliência; independência de linguagens; facilidade de manutenção evolução	Em caso de múltiplos serviços complexidade na gestão; em caso de dependências externas os testes podem se tornar complexos;
Arquitetura de camadas	Separação de responsabilidades; fácil manutenção; facilidade para testes unitários	Sistemas grandes podem ser complexos nessa arquitetura; complexidade para mudanças na medica que o sistema cresce;
Arquitetura de nuvem	Escalabilidade automática, resiliência, alta disponibilidade, flexibilidade de recursos e facilidade de integração	Maior esforço para previsibilidade de custos já que é sob demanda; dependência do provedor da nuvem; complexidade de gestão proporcional a utilização dos serviços

O Kubernetes Cluster da Solução

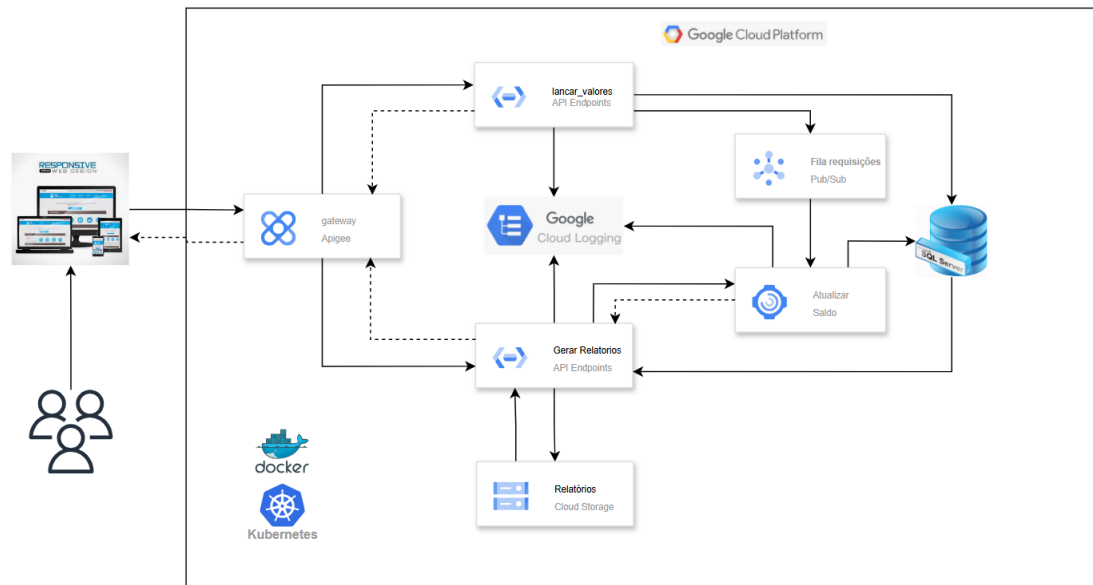


A solução proposta para a arquitetura de microsserviços no Kubernetes contém um **cluster único** que hospeda múltiplos **pods**, cada um responsável por um serviço distinto. A solução inclui **quatro pods** principais: o **launch-service-pod**, responsável pela gestão dos lançamentos de valores e comunicação com o banco de dados; o **balance-service-pod**, encarregado de consumir eventos do Google Pub/Sub e atualizar os saldos em tempo real; o **report-service-pod**, que gera relatórios a partir das transações e dados consolidados armazenados no banco de dados; e o **database-pod**, que contém o SQL Server para armazenar as transações e saldos de forma persistente. Além disso, um **load balancer** é configurado para distribuir as requisições externas entre os diferentes serviços, garantindo alta disponibilidade e balanceamento de carga eficiente, direcionando as chamadas para o pod correspondente, seja para lançar valores, atualizar saldos ou gerar relatórios. O Kubernetes orquestra a criação, escalabilidade e disponibilidade dos pods dentro do cluster, permitindo uma comunicação fluida e segura entre os serviços.

Justificativa da Escolha Arquitetural utilizada

A escolha dessa arquitetura baseia-se na necessidade de escalabilidade, resiliência e alta disponibilidade do sistema. Utilizando microsserviços, garantimos que cada componente seja independente e escalável, permitindo ajustes de desempenho conforme a demanda. A utilização do Kubernetes e Docker assegura que os serviços possam ser facilmente gerenciados e distribuídos, enquanto o Google Cloud Pub/Sub garante comunicação assíncrona eficiente entre os serviços, sem bloqueios. O Google Cloud Storage oferece solução gerenciada e segura para armazenamento de dados e relatórios, enquanto o Apigee facilita a segurança e o gerenciamento das APIs. Essa arquitetura permite que o sistema seja robusto, seguro e capaz de lidar com picos de tráfego sem comprometer a performance ou disponibilidade.

Arquitetura Alvo

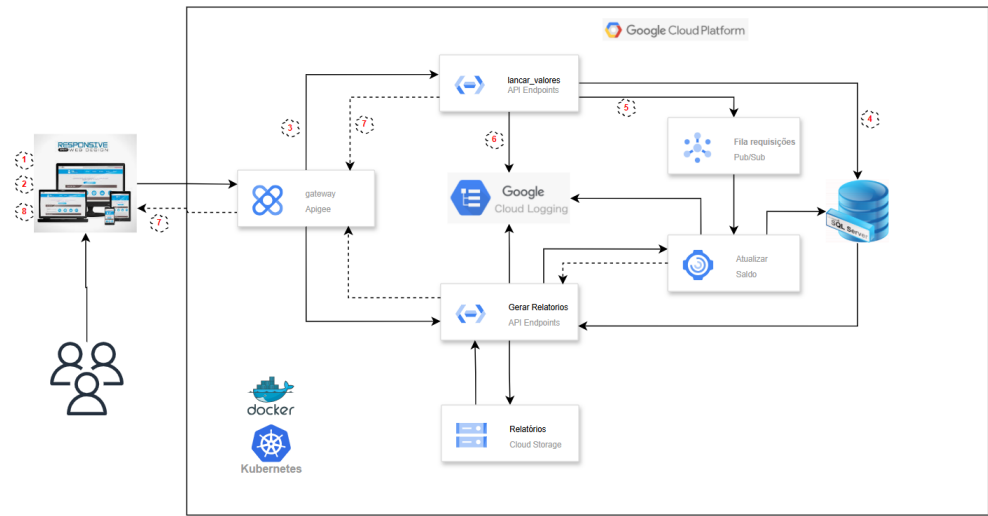


A arquitetura alvo é baseada em microsserviços, hospedada na Google Cloud, utilizando Docker e Kubernetes para orquestração. A solução inclui os seguintes componentes principais:

1. **Frontend:** Interface do usuário acessada via navegador ou aplicativo móvel.
2. **Apigee:** Gateway de API que gerencia autenticação, segurança e roteamento das requisições para os microsserviços.
3. **Microsserviços:**
 - **Lançar_Valores:** Recebe e processa os lançamentos financeiros, gravando no SQL Server e publicando eventos no Google Pub/Sub.
 - **Atualizar_Saldo:** Consome eventos do Pub/Sub e atualiza o saldo no SQL Server.
 - **Gerar_Relatórios:** Gera relatórios consolidados a partir do saldo e armazena no Google Cloud Storage.
4. **Google Cloud Pub/Sub:** Mensageria assíncrona para comunicação entre os microsserviços.
5. **Google Cloud SQL:** Banco de dados SQL Server para armazenar transações e saldos.
6. **Kubernetes:** Orquestra a implantação, escalabilidade e resiliência dos microsserviços.
7. **Google Cloud Logging:** Monitoramento e registro de todas as operações.
8. **CI/CD:** Pipelines automatizadas para testes e implantação contínua.
9. **Google Cloud Load Balancer:** Balanceamento de carga para distribuir requisições entre as instâncias.

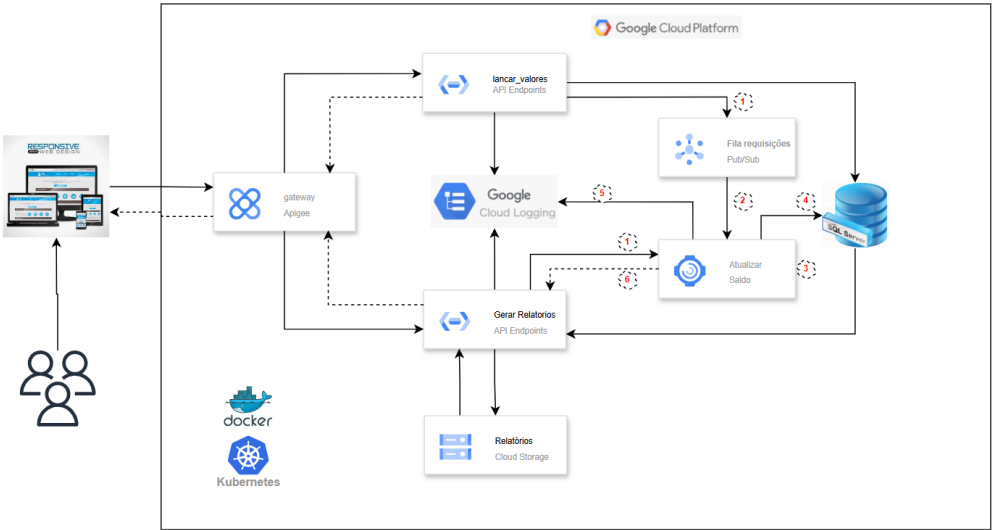
Fluxos de Solução

1. Fluxo para Lançamento de valores



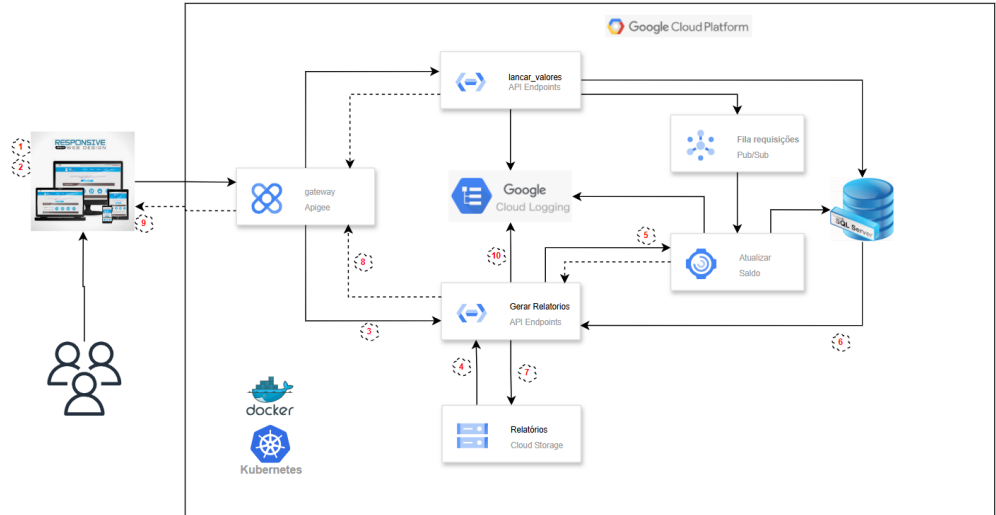
1	Fluxo inicia com o usuário já autenticado, acessando a tela de lançamento de valores
2	Usuário lança o valor que será inserido no sistema e escolhe o tipo (débito/crédito)
3	Aplicação chama Apigee que encaminha requisição para o serviço Lançar_valores
4	O serviço Lançar_Valores grava uma transação no banco SQL Server
5	O serviço Lançar_valores publica um evento na fila do Google Pub/Sub com os dados de lançamento para futura atualização do saldo
6	O Serviço Lançar_Valores registra o evento e o status da transação no Google Cloud Logging
7	O serviço Lançar_valores retorna para a Apigee e consequentemente para a aplicação front-end o status da transação
8	A aplicação exibe o resultado da operação e o fluxo se encerra

2. Fluxo para atualização de saldo



1	O fluxo se inicia automaticamente quando um evento for publicado na fila do Google Pub/Sub ou quando a API Gerar_Relatorio solicitar atualização
2	O serviço Atualizar_Saldo recebe os eventos do Google Pub/sub e processa os lançamentos em tempo real
3	O serviço Atualizar Saldo consolida todos os lançamentos e calcula o saldo atualizado
4	O Saldo é gravado no SQL Server
5	O Serviço Atualizar_Saldo registra a operação no Google Cloud Logging
6	Caso tenha sido acionado pela API gerar_relatório e serviço retorna o status da operação para a Apigee que retorna para o fron-end

3. Fluxo de geração de relatórios



1	O usuário já autenticado no sistema acessa a tela de relatórios
2	O usuário seleciona a data de inicio e fim e o formato de arquivo (CSV ou PDF)
3	A Aplicação chama a API Gerar_relatorios via Apigee
4	a API Gerar_relatorios consulta a Cloud Storage para verificar se o relatório no período solicitado já existe identificando a URL do arquivo
4.1	Se o relatório existir, segue para o passo 8
4.1	Se não existir, segue para o passo 5
5	A API Gerar_relatorios solicita ao serviço atualizar_saldo a consolidação de valores pendentes
6	a API Gerar_relatorios lê do banco SQL Server todas as informações relativas ao saldo do período solicitado
7	a API Gerar_relatorios confecciona o relatório no formato solicitado e grava na Cloud Storage armazenando a URL do arquivo
8	a API Gerar_relatorios retorna a URL do relatório para a Apigee
9	O Apigee entrega a URL do relatório para a aplicação que exibe na tela para o usuário com opção de download
10	Toda a operação é registrada no Google Cloud Logging

Testes

Apresentaremos abaixo uma planilha contendo os principais casos de testes unitários e integrados para serem aplicados pelo time de QA a fim de garantir qualidade mínima na solução

Testes Unitários

ID do Teste	Componente	Descrição do Teste	Entrada	Resultado Esperado
T01	Serviço de Lançamento de Valores	Testar se o serviço registra corretamente uma transação de débito no banco de dados	<code>{"type": "debit", "amount": 100.50}</code>	Transação registrada com sucesso no banco, com ID gerado.
T02	Serviço de Lançamento de Valores	Testar se o serviço registra corretamente uma transação de crédito no banco de dados	<code>{"type": "credit", "amount": 200.00}</code>	Transação registrada com sucesso no banco, com ID gerado.
T03	Serviço de Atualização de Saldo	Testar se o saldo é atualizado corretamente após a transação de débito	Transação de débito no valor 100.50	Saldo ajustado corretamente no banco após transação.
T04	Serviço de Atualização de Saldo	Testar se o saldo é atualizado corretamente após a transação de crédito	Transação de crédito no valor 200.00	Saldo ajustado corretamente no banco após transação.
T05	Serviço de Geração de Relatórios	Testar se o relatório é gerado corretamente em formato CSV	Solicitação de relatório com datas de início e fim: 2025-01-01 e 2025-01-31, formato CSV	Relatório gerado e armazenado no Google Cloud Storage, URL retornada.
T06	Serviço de Geração de Relatórios	Testar se o relatório é gerado corretamente em formato PDF	Solicitação de relatório com datas de início e fim: 2025-01-01 e 2025-01-31, formato PDF	Relatório gerado e armazenado no Google Cloud Storage, URL retornada.

Testes Integrados

ID do Teste	Componente	Descrição do Teste	Entrada	Resultado Esperado
TI01	Lançamento de Valores + Atualização de Saldo	Testar o fluxo completo de lançamento de débito seguido de atualização do saldo	Transação de débito: {"type": "debit", "amount": 100.50}	Saldo atualizado corretamente após o lançamento de débito.
TI02	Lançamento de Valores + Atualização de Saldo	Testar o fluxo completo de lançamento de crédito seguido de atualização do saldo	Transação de crédito: {"type": "credit", "amount": 200.00}	Saldo atualizado corretamente após o lançamento de crédito.
TI03	Lançamento de Valores + Geração de Relatório	Testar o fluxo completo de lançamento de valores e geração de relatório no formato CSV	Lançamento de débito: {"type": "debit", "amount": 100.50}, Relatório para período 2025-01-01 a 2025-01-31, formato CSV	Relatório gerado corretamente e armazenado no Google Cloud Storage.
TI04	Lançamento de Valores + Geração de Relatório	Testar o fluxo completo de lançamento de valores e geração de relatório no formato PDF	Lançamento de crédito: {"type": "credit", "amount": 200.00}, Relatório para período 2025-01-01 a 2025-01-31, formato PDF	Relatório gerado corretamente e armazenado no Google Cloud Storage.
TI05	Lançamento de Valores + Atualização de Saldo + Geração de Relatório	Testar o fluxo completo de lançamento de valores, atualização de saldo e geração de relatório	Lançamento de débito: {"type": "debit", "amount": 100.50}, Geração de relatório no formato CSV	Relatório gerado corretamente com saldo atualizado.
TI06	Serviço de Lançamento de Valores + Google Pub/Sub	Testar se o evento do lançamento de valores é enviado corretamente para o Google Pub/Sub	Lançamento de valor: {"type": "debit", "amount": 100.50}	Evento de lançamento de valor publicado com sucesso no Google Pub/Sub.

Código e Documentação do Sistema

Todo Código está disponibilizado em um domínio público do GitHub no caminho:

https://github.com/srj-figueiredo/Fluxo_Caixa_GCP.git

Tecnologias Utilizadas

- **Python 3.x:** Linguagem de programação para desenvolvimento dos microserviços.
- **Flask:** Framework web para construção das APIs RESTful.
- **SQL Server:** Banco de dados relacional para armazenar transações e saldo.
- **Docker:** Containerização de cada serviço para facilitar o desenvolvimento e implantação.
- **Google Pub/Sub:** Sistema de mensageria para orquestrar eventos entre os serviços.
- **Google Cloud Storage:** Armazenamento de relatórios gerados em CSV ou PDF.
- **Docker Compose:** Orquestração local dos microserviços.

Estrutura do GitHub **Fluxo_Caixa_GCP**

