DESAFIO

INTRODUÇÃO

Como o mundo está se enraizando na nova era de serviços em nuvem, um desafio fundamental que assombra todos os arquitetos é identificar a solução tecnológica correta para o sistema. O grande número de alternativas e suas combinações é suficiente para deixar qualquer profissional com dúvidas.

Os desafios de hoje não se limitam à identificação de uma boa tecnologia ou ferramenta para a tarefa. Não estamos escolhendo entre uma boa ferramenta e uma má. A escolha é identificar um conjunto apropriado dentro de uma vasta opção de ferramentas e tecnologias disponíveis. Precisamos encontrar o subconjunto certo de tecnologias, que possam se complementar e funcionar bem em conjunto.

A arquitetura pretende ser facilitadora no processo de desenvolvimento de sistemas web, visando agilizar não somente o atendimento ao público, mas também gerando poucos erros em produção a cada ciclo de release.

Por que a Arquitetura de software é importante.

A arquitetura de software é importante, pois ajuda a garantir que o software seja de alta qualidade, fácil de manter, escalável, flexível e de desempenho otimizado, o que é crítico para o sucesso de projetos de software a longo prazo. Podemos observar em detalhes cada ponto abordado:

1. Qualidade do software: Uma boa arquitetura de software ajuda a garantir que a aplicação seja de alta qualidade, mantendo-o escalável, flexível, confiável e fácil de manter.
2. Facilidade de manutenção: Uma arquitetura clara e bem projetada torna o software mais fácil de manter e corrigir, pois os componentes são organizados de maneira lógica e fácil de entender.
3. Flexibilidade: Uma boa arquitetura de software deve ser projetada para permitir mudanças no futuro, sem prejudicar o funcionamento do software. Isso permite que o software seja adaptado às mudanças no ambiente de negócios.
4. Melhor desempenho: Uma arquitetura de software bem projetada pode ajudar a garantir que o software tenha desempenho otimizado, gerenciando de maneira eficiente recursos como memória, processamento e largura de banda de rede.
5. Melhor tomada de decisão: A arquitetura de software é um importante ponto de partida para tomar decisões de design, implantação e melhoria de software, pois permite visualizar e avaliar a estrutura geral do software.

Ou seja, a arquitetura de software é inerente ao processo de desenvolvimento de software. Uma boa arquitetura irá sustentar sistemas e soluções ao longo da vida do projeto, desde a concepção, passando pelo desenvolvimento e monitoramento, até o seu descomissionamento.

O que torna a arquitetura em nuvem tão especial

A arquitetura de nuvem (*#cloud* *#architecture*) vem ganhando espaço nas empresas conforme a necessidade em definir soluções de arquitetura para softwares distribuídos, pois oferece:

1. Escalabilidade: Ambientes em nuvem permitem que a capacidade de computação seja facilmente escalada para atender às demandas crescentes, sem a necessidade de adquirir hardware adicional.
2. Flexibilidade: A arquitetura de nuvem permite que os recursos sejam provisionados dinamicamente de acordo com as necessidades do software, o que torna o software mais flexível e escalável.
3. Agilidade: Permite que as soluções de software sejam implementadas rapidamente, sem a necessidade de esperar pela aquisição e instalação de hardware.
4. Redução de custos: A nuvem permite que os custos sejam reduzidos, pois é possível pagar apenas pelos recursos que são realmente utilizados, sem a necessidade de investir em hardware caro e manutenção.
5. Alta disponibilidade: A arquitetura de nuvem é projetada para oferecer alta disponibilidade, com recursos de tolerância a falhas e redundância de dados, para garantir que o software seja acessível a todos, mesmo em caso de falha.
6. Segurança: A nuvem oferece recursos de segurança robustos, como criptografia de dados, autenticação de usuários e controle de acesso, para proteger o software e os dados dos usuários.

Análise do Contexto

Os **fundos de investimento** são uma opção capaz de agradar diversos perfis de investidores. De um lado, eles são simples – e, por isso, adequados para quem está querendo sair da poupança e começar a fazer aplicações mais rentáveis. De outro, há fundos sofisticados, para quem já está habituado ao mercado financeiro.

No país, os fundos somam patrimônio de quase R$ 5,2 trilhões, segundo dados da Anbima (Associação Brasileira das Entidades dos Mercados Financeiro e de

Capitais) levantados em agosto de 2019.

O que são fundos de Investimento

Fundos são uma espécie de “condomínio” de investidores. Eles reúnem os recursos de diversas pessoas, para que sejam aplicados em conjunto no mercado financeiro e de capitais. Os ganhos obtidos com as aplicações são divididos entre os participantes, na proporção do valor depositado por cada um.

A soma do dinheiro dos investidores forma o patrimônio do fundo, que é aplicado por uma instituição ou profissional (gestor).

A cota de um fundo de investimento é uma fração do patrimônio desse fundo. Quando um investidor aplica seus recursos em um fundo, recebe um número de cotas proporcional ao valor que ele investiu.

O investidor no fundo não será dono diretamente dos ativos, mas das cotas, que representam uma fração dos ativos que estão na carteira.

A rentabilidade obtida pelo investidor também é calculada a partir do valor das cotas.

Como calcular o valor de uma Cota de um Fundo de Investimentos?

Primeiro, vale lembrar que o **valor da cota de um fundo de investimento é atualizado diariamente**. Ou seja, se naquele dia os ativos nos quais o fundo investe se valorizarem, isso será refletido no valor da cota.

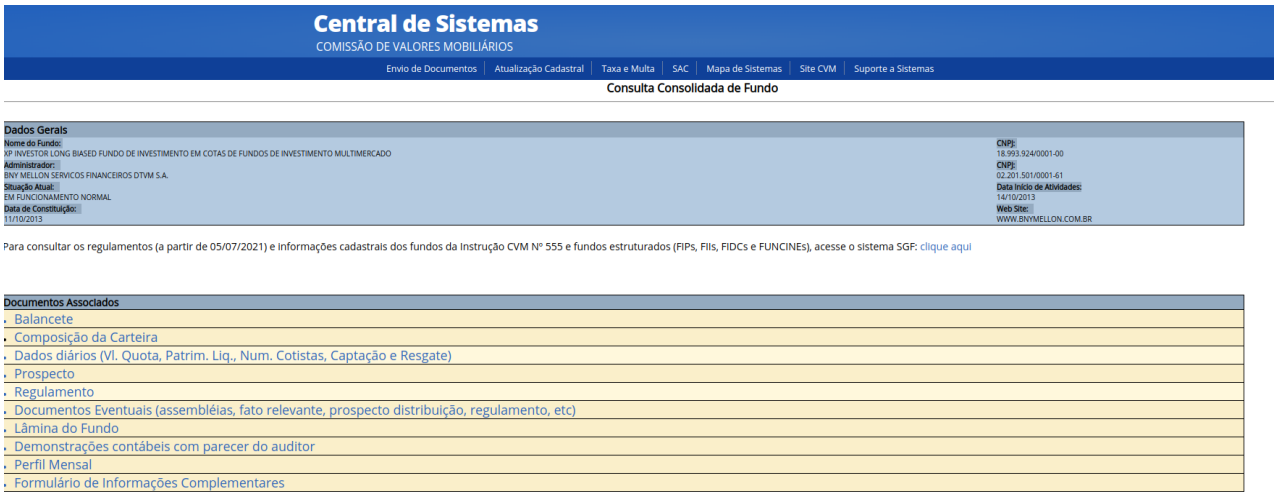
Portanto, no dia seguinte, se um investidor quiser aplicar mais recursos no fundo, isso será feito considerando o novo valor da cota.
É importante fazer uma observação aqui: o administrador do fundo é o responsável perante a Comissão de Valores Mobiliários - **CVM** pela atualização do valor dos ativos da carteira.

O Desafio

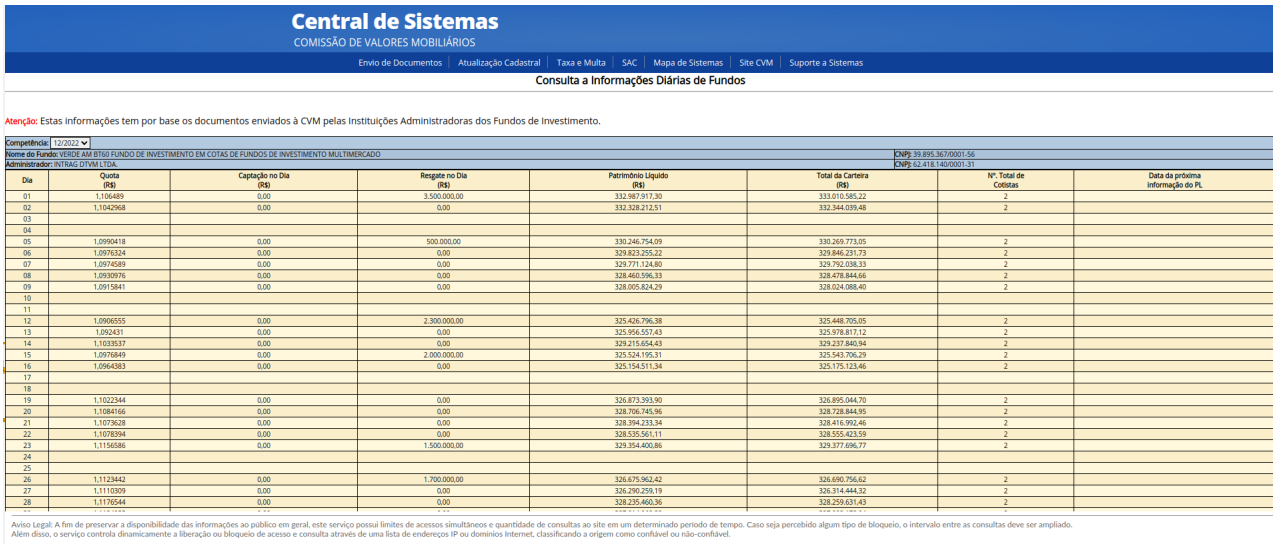
Caso alguém deseje consultar o valor de uma cota de um fundo de investimento, ou mesmo, acompanhar a performance de um fundo de investimento, será necessário acessar o site da CVM onde esse dados está disponível.

https://cvmweb.cvm.gov.br/swb/default.asp?sg_sistema=fundosreg

Com o site aberto deverá inserir o CNPJ ou nome do fundo que deseja consultar.



Neste ponto, para consultar o valor de uma cota, deverá acessar o link : *Dados diários (Vl Quota, Patrim Liq....)*:



Para cada mês que deseje a informação o usuário deve selecionar na caixa suspensa e navegar pela data que deseja, extraindo a informação que necessita.

Caso o trabalho envolva monitorar 2 ou mais fundos o processo manual de obtenção de dados passa a crescer exponencialmente, portanto é necessário desenvolver uma **aplicação que colete e armazene esse dados de forma automatizada**, aprimorando o andamento do fluxo de trabalho. Com a

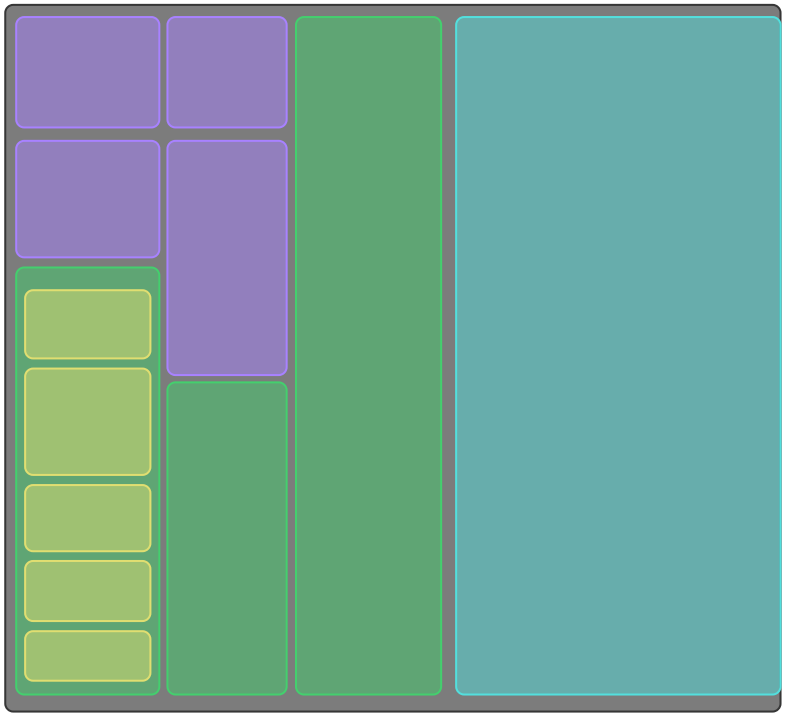
automatização, será possível reduzir custos, tempo, desperdícios, aumentando a produtividade, minimizando falhas e controle.

Para tanto, é necessário observar que, o desafio de extração de dados do mercado financeiro para consumo em uma aplicação envolve a coleta, processamento e armazenamento de dados, em tempo real, para serem utilizadas por vários usuários, em uma plataforma, em diversas aplicações e soluções. Diante do exposto, nos leva a definir alguns requisitos não funcionais, como:

1. Alto desempenho: é necessário garantir que a aplicação possa lidar com uma grande quantidade de dados financeiros e fornecer informações atualizadas rapidamente para os usuários.
2. Segurança dos dados: é crucial proteger os dados financeiros contra acesso não autorizado e garantir a privacidade dos usuários.
3. Múltiplos usuários: a aplicação deve ser projetada para lidar com vários usuários acessando grande massa de dados financeiros ao mesmo tempo.

Canvas do Projeto

CANVAS



MATRIZ CSD

- **CERTEZAS:** tudo que é considerado verdade absoluta
- **SUPOSIÇÕES:** normalmente, aqui ficam as opiniões divergentes e o que se acredita ser real, mas ainda não é possível provar

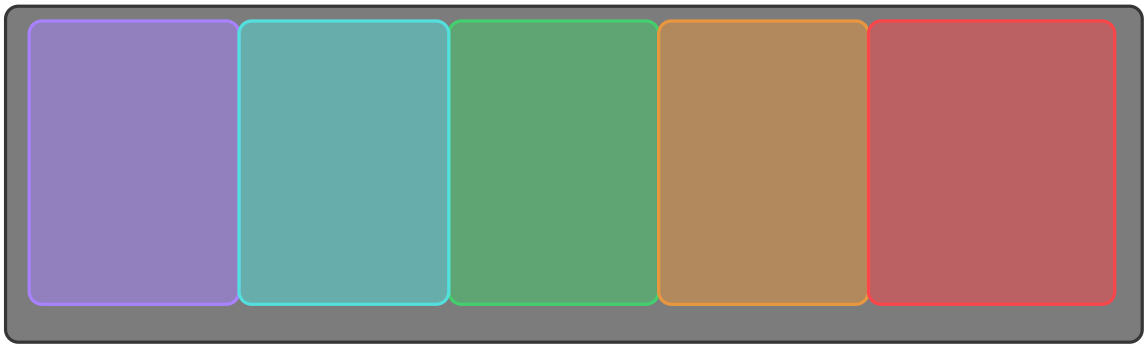
- **DÚVIDAS:** tudo o que é desconhecido sobre o projeto

CERTEZAS	SUPOSIÇÕES	DÚVIDAS
O usuário deseja consultar o valor das cotas dos fundos	Armazenamento local com containers	Verificar se a solução em containers do Redis já aborda TimeSeries, Mensageria, PubSub
O usuário deseja consultar o valor histórico das cotas para cada fundo	Verificar se a biblioteca BeautifulSoup manipula JS (mais leve)	Verificar requisitos de mensageria para o Redis
O usuário deseja consultar outros dados históricos (valor patrimonial, número de acionistas e cotas)	A biblioteca Selenium é mais pesada que as demais opções para extração dos dados, mas lida nativamente com JS	O usuário deseja ver os dados apresentados de qual forma? Gráfico, texto, de forma iterativa?
Para extrair os dados do site da CVM será necessário acessá-los e navegar pelos links	Redis possui solução para filas e Pubsub.	Como notificar o usuário que a extração dos dados assíncrona foi concluída?
O acesso via CNPJ dos fundos é único, evitando a procura em lista de resultados. → Definir acesso via CNPJ	Será necessário export dos drivers da biblioteca Selenium? Entretanto a mesma possui solução em containers	Permitir algum usuário excluir e recarregar os dados já salvos?
A extração dos dados dos site da CVM (I/O) é lenta, precisará ser tratado de forma assíncrona.	O Redis possui solução em containers	Como o Redis identifica as mensagens que foram tratadas corretamente e ainda não tratadas

CERTEZAS	SUPOSIÇÕES	DÚVIDAS
A biblioteca básica Requests não manipula JS	Mesmo ao usar o Redis em containers definir armazenamento local para testes.	
Criar a solução em containers possibilita o deploy em multiplataformas		
Ambientes financeiros demanda aplicações para lidar com múltiplos usuários, confiabilidade dos dados e segura		

MATRIZ POEMS

POEMS



Personas

Esta seção tem o objetivo de apresentar as pessoas envolvidas diretamente no problema apresentado, definindo suas características pessoais, sociais, intelectuais e profissionais, com possíveis detalhes levantados.

1. **Investidor iniciante**: essa persona que começando agora no mercado financeiro e e busca uma maneira fácil de acompanhar seus investimentos, a fim de, entender melhor como funcionam os

fundos de investimento. Suas motivações incluem a busca por informações claras e simples sobre seus investimentos e uma plataforma fácil de usar.

2. **Investidor experiente**: essa persona já possui conhecimento sobre o mercado financeiro e esta buscando uma plataforma mais avançada para controlar seus investimentos, com funcionalidades como análises de desempenho de fundos e gráficos avançados. Suas motivações podem incluir a busca por informações mais detalhadas e precisas sobre seus investimentos e uma plataforma que ofereça ferramentas avançadas de análise.
3. **Gestor de fundos**: essa persona é responsável por gerenciar os fundos de investimento e pode estar buscando uma plataforma que facilite o controle patrimonial dos fundos, permitindo uma visão clara do desempenho e dos investimentos. Suas motivações podem incluir a busca por uma ferramenta que permita uma gestão mais eficiente dos fundos e que ajude a identificar oportunidades de investimento.
4. **Consultor financeiro**: essa persona é responsável por aconselhar os investidores em suas decisões financeiras e pode estar buscando uma plataforma que ofereça ferramentas de análise e que permita um acompanhamento mais eficiente dos investimentos de seus clientes. Suas motivações podem incluir a busca por uma ferramenta que facilite seu trabalho e que ajude a identificar oportunidades de investimento para seus clientes.
5. **Pequeno investidor**: essa persona possui um patrimônio menor e pode estar buscando uma plataforma que permita investimentos em fundos com baixos valores de entrada e que ofereça uma visão clara do desempenho dos fundos. Suas motivações podem incluir a busca por uma plataforma que ofereça oportunidades de investimento acessíveis e que permita um acompanhamento fácil dos investimentos.
6. **Investidor institucional**: essa persona representa empresas e instituições que investem em fundos e pode estar buscando uma plataforma que ofereça uma visão completa do desempenho de seus investimentos e que permita um acompanhamento eficiente do patrimônio investido. Suas motivações podem incluir a busca por uma ferramenta que facilite o controle patrimonial dos fundos e que permita uma gestão mais eficiente dos investimentos.

Área de Experimentação

SOLUÇÃO

Boas práticas de arquitetura de software visam contribuir para o atendimento dos objetivos, respeitando restrições, atingindo atributos de qualidade.

Dividir para Conquistar

Como já explanado anteriormente, os requisitos não funcionais da nossa solução será o alto desempenho, flexibilidade e escalabilidade irão impactar diretamente a solução a ser adotada.

Utilizando a estratégia "Divide and Conquer" é possível abordar o problema e dividir sua complexidade em subproblemas menores de fácil entendimento e, assim, endereçar soluções de forma eficiente, definindo blocos de responsabilidades, tornando a software mais fácil de entender, aprender, fazer futuras expansões e manutenções.

A aplicação deste padrão envolve três etapas principais:

1. Análise do problema: O primeiro passo é identificar o problema complexo que precisa ser resolvido e analisá-lo para identificar subproblemas menores e independentes.
2. Solução de subproblemas: Em seguida, cada subproblema é solucionado individualmente. Esta etapa pode envolver a aplicação de técnicas de resolução de problemas semelhantes ou o uso de algoritmos específicos para solucionar cada subproblema.
3. Combinação de soluções: Por fim, as soluções para cada subproblema são combinadas para obter a solução final para o problema original.

Dessas três etapas é possível se aprofundar:

1. O usuário deseja consultar os dados histórico da cota dos fundos como valor, patrimônio líquido e número total de cotistas;
2. A forma mais simples de acessar o fundo é pelo CNPJ (id único do fundo);
3. Para consultar os dados histórico será necessário acessar o site da CVM sobre o respectivo fundo;
4. Para acessar o site será necessário ler os objetos carregados na request (HTML) , na maior parte das vezes serão textos, outras vezes texto em formato de tabelas, mas será necessário também manipular scripts JS da aplicação;
5. O site da CVM não trás todos os dados de uma vez, será necessário percorrer cada mês disponível, manipulando o js script da caixa de seleção (mm/aaaa);
6. É necessário identificar quais meses estão disponível para consulta;
7. Após a leitura dos dados será necessário armazená-los;
8. Como são dados temporais será necessário ordená-los;
9. Como dados históricos não se alteraram, caso os dados já estejam na base de dados da aplicação será necessário apenas atualizar os mais recentes;

Diante dos subproblemas está claro que a base da aplicação será um web crawler, ou seja, um programa de computador desenvolvido com a finalidade de rastrear e analisar o código de um website em busca de informações, e depois armazená-la. O exemplo clássico dessa aplicação é o bot construído pelo Google para indexar websites no seu motor de buscas, o *googlebot*

Em relação ao ponto 5, devido a forma como foi construído o site da CVM, será necessário percorrer as tabelas, consultando mês a mês as informações, esse processo envolve carregar o JS e executá-lo, a cada consulta. Essa carga de informações tornará nossa aplicação lenta, portanto o retorno para o usuário irá levar certo tempo, descartando assim uma aplicação síncrona.

Ao olhar para esses subproblemas, aliado aos requisitos não funcionais de uma aplicação para o mercado financeiro (segurança, escalabilidade e desempenho) um dos design patterns já salta aos olhos que é o de mensageria:

Design Pattern

> O Design Pattern de Mensageria é um padrão de projeto de software que define uma abordagem para comunicação entre componentes de software distribuídos. Ele se concentra em soluções para problemas comuns enfrentados na criação de aplicativos distribuídos, como integração de sistemas, escalabilidade, alta disponibilidade e tolerância a falhas.
>
> O Design Pattern de Mensageria envolve o uso de mensagens para a comunicação entre componentes, em vez de invocar diretamente outros componentes. Cada componente pode ser pensado como um "produtor" ou "consumidor" de mensagens. O produtor envia mensagens para o sistema, enquanto o consumidor as recebe e as processa.
>
> Os Design Patterns de Mensageria incluem vários padrões, como o padrão Publish/Subscribe, o padrão Request/Reply e o padrão Message Queue. Cada padrão define uma abordagem específica para a comunicação de mensagens entre componentes distribuídos.
>
> Ao utilizar o Design Pattern de Mensageria, é possível criar soluções de software distribuídas que são escaláveis, flexíveis, tolerantes a falhas e de fácil manutenção. Além disso, ele ajuda a garantir que o software seja integrado de maneira eficiente, sem prejudicar o desempenho ou a disponibilidade do sistema.
>
> ### Pub/Sub

O padrão Publish/Subscribe (também conhecido como "Pub/Sub") é um modelo de comunicação em que os componentes de software se registram como interessados ou assinantes em receber mensagens de determinados tipos. Quando uma mensagem é publicada, ela é entregue a todos os componentes registrados como assinantes nesse tipo de mensagem.

O produtor dessa mensagem é um outro componente isolado, que pode ou não se importar com o resultado final

No padrão Pub/Sub, existe um intermediário (ou "broker") que gerencia a publicação e a entrega das mensagens. Este intermediário é responsável por armazenar mensagens enviadas por produtores e entregá-las aos consumidores correspondentes.

Message Queue

O padrão Message Queue (fila de mensagens) é um modelo de comunicação em que as mensagens são armazenadas em uma fila antes de serem processadas pelos consumidores. Neste padrão, o intermediário (ou "broker") é responsável por garantir que as mensagens sejam entregues em ordem, evitando que mensagens sejam perdidas ou processadas duplicadamente.

A principal diferença entre os dois padrões é que, no padrão Pub/Sub, as mensagens são entregues a todos os componentes interessados, enquanto no padrão Message Queue, as mensagens são entregues a um único componente, na ordem em que foram adicionadas na fila.

Ambos os padrões são amplamente utilizados em aplicativos distribuídos, especialmente em sistemas que precisam lidar com grandes volumes de mensagens ou processamento em lote. Eles ajudam a garantir a escalabilidade, a alta disponibilidade e a tolerância a falhas de sistemas distribuídos.

Detalhes Técnicos

https://zguide.zeromq.org/docs/chapter2/#Messaging-Patterns

Os workers conectam-se a montante ao publisher, e a jusante ao funil. Isto significa que você pode adicionar workers arbitrariamente. Se os workers se conectarem a seus pontos finais, você precisará (a) mais pontos finais e (b) modificar o publisher e/ou o funil cada vez que você adicionar um worker. Dizemos que o publisher e o funil são partes estáveis de nossa arquitetura e os workers são partes dinâmicas da mesma.

> Os padrões de mensageria são implementados por pares de soquetes com tipos correspondentes. Em outras palavras, para entender os padrões é necessário entender os tipos de soquetes e como eles funcionam em conjunto. Na maioria das vezes, isto só requer estudo;
>
> **Request-reply**: Resposta à solicitação, que conecta um conjunto de clientes a um conjunto de serviços. Este é um padrão de chamada de procedimento remoto e distribuição de tarefas.
>
> **Pub-sub**: que conecta um conjunto de editores a um conjunto de assinantes. Este é um padrão de distribuição de dados.
>
> **Pipeline**: que conecta os nós em um padrão de fan-out/fan-in que pode ter vários passos e loops. Este é um padrão paralelo de distribuição de tarefas e coleta de dados.
>
> **Exclusive pair**: par exclusivo, que conecta dois soquetes exclusivamente. Este é um padrão para conectar dois fios em um processo, não confundir com pares "normais" de soquetes.
>
> ### Etapas de um sistema de mensageria
>
> 1. Conectar o publisher a uma fila / tópico
> 2. Criar a mensagem
> 3. Enviar para a fila / tópico
> 4. Conectar o assinante a uma fila / tópico
> 5. Receber a mensagem
> 6. Acessar o conteúdo da mensagem
> 7. Processar o conteúdo
>
> *#design_patterns* *#mensageria* *#pubsub*

Padrão arquitetural

Aliado ao padrão de mensageria podemos pensar no modelo de arquitetura em Micro serviços (*#microservices*) .
A API ficará após o Load Balance, esse API será responsável por validar a requisição feita pelo usuário, deverá verificar se o CNPJ existe cadastrado na base da CVM, quais meses possuem dados para o fundo solicitado e escrever as mensagens na fila. Quando os dados do fundo já estiverem no banco de dados deverá devolver ao usuário.
Nessa API também ficará o limitador de chamadas e o processo de autenticação a serem desenvolvidos futuramente.

Micro serviços são serviços com interfaces públicas enxutas e extremamente coesas.

A interface pública de um serviço habilita a “entrada e saída” de informações, tanto de forma síncrona – em um modelo baseado em *request/response* – quanto assíncrona – através de troca de mensagens com comandos ou eventos.
Interfaces pública enxutas e coesas, bem delimitadas, fazem com que micro serviços sejam mais fáceis de entender, tanto em sua lógica interna quanto em demandas de integração/cooperação.

Atenção:
"O trade-off negativo da reutilização é o *#acoplamento*. Quando um arquiteto projeta um sistema favorecendo o reuso, ele também favorece ao acoplamento para permitir esse reuso, seja por herança ou composição."
Neil Ford

Oito falácias da computação distribuída

Em 1994, Peter Deutsch, que trabalhou na Sun Microsystems, consolidou uma lista de 7 falácias frequentemente associadas a sistemas distribuídos. Mais tarde, em 1997, James Gosling adicionou um item a essa lista, criando a comumente conhecida relação das oito falácias da computação distribuída. São elas:

- A rede é confiável
- A latência é zero
- A largura de banda é infinita
- A rede é segura
- A topologia da rede nunca muda
- Existe um administrador
- Custo de transferência de dados é zero
- A rede é homogênea

O Padrão de Projetos - Micro Serviços

O padrão de projeto Microservices é um modelo de arquitetura de software que permite a criação de aplicativos complexos como uma coleção de pequenos serviços independentes e altamente especializados. Cada serviço é projetado para realizar uma tarefa

específica e é responsável por sua própria funcionalidade, dados e escalabilidade.

Este padrão se baseia na ideia de que grandes aplicativos monolíticos são difíceis de manter e escalar, e que é mais fácil trabalhar com aplicativos menores e mais focados. Ao utilizar o padrão Microservices, é possível criar aplicativos altamente escaláveis, resilientes e flexíveis, que são mais fáceis de manter e atualizar.

As empresas estão percebendo cada vez mais a economia de custo ao se
resolver problemas de implantação e melhorar as operações de produção e de DevOps
usando os containers. Essa soluções usando containers auxiliam as empresas a
criar e implantar aplicativos com a velocidade e a escala da nuvem, seja qual for a
escolha de plataformas ou de ferramentas.

O Docker está se tornando o verdadeiro padrão no setor de containers, com
suporte dos fornecedores mais significativos nos ecossistemas do Windows e do Linux. No
futuro, o Docker provavelmente estará onipresente em qualquer datacenter na nuvem
ou no local.As empresas estão percebendo cada vez mais a economia de custo ao se
resolver problemas de implantação e melhorar as operações de produção e de DevOps
usando os containers.

A aplicação deste padrão envolve três etapas principais:

1. Identificação de serviços: O primeiro passo é identificar as funcionalidades do aplicativo e dividi-las em serviços independentes e altamente especializados.
2. Desenvolvimento de serviços: Em seguida, cada serviço é desenvolvido de forma independente, utilizando tecnologias e ferramentas apropriadas.
3. Integração de serviços: Por fim, os serviços são integrados para formar o aplicativo final. Esta integração pode ser realizada através de mensagens, APIs ou outras técnicas de comunicação.

#architecture *#microservices* *#docker*

Arquitetura inicial Proposta

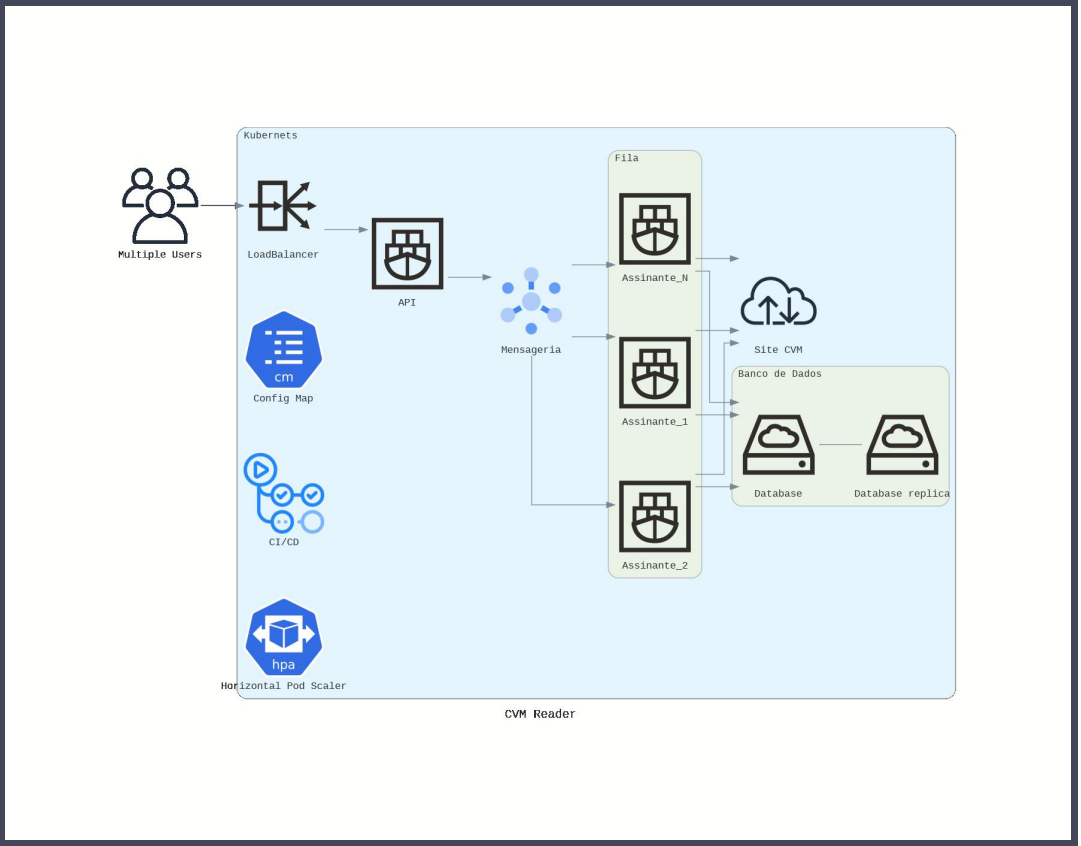
Linguagem definida para construção da solução: Python versão 3.10

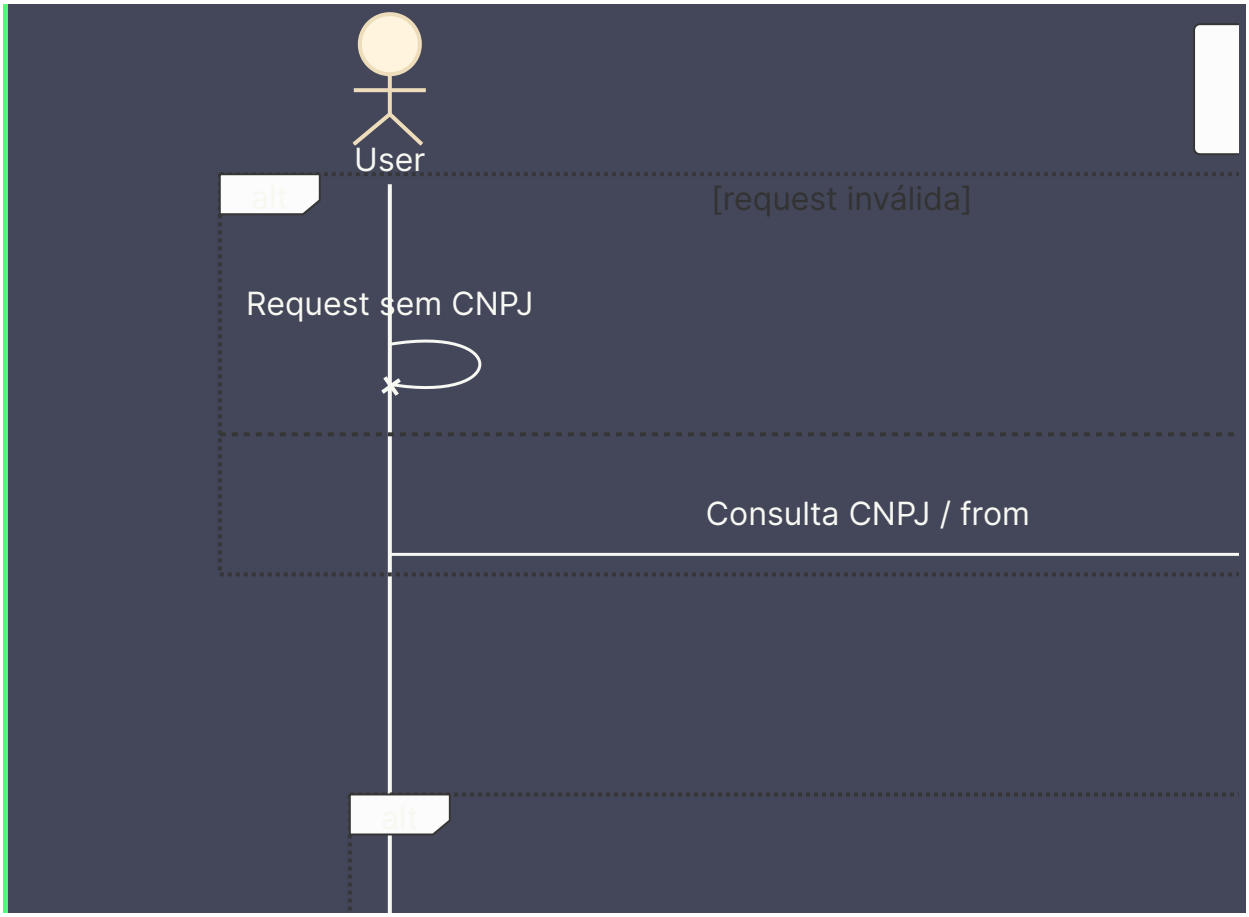
https://colab.research.google.com/drive/1HFNup6hsPtCMYuEKKjAHrKDRxiLjLJh7?usp=sharing

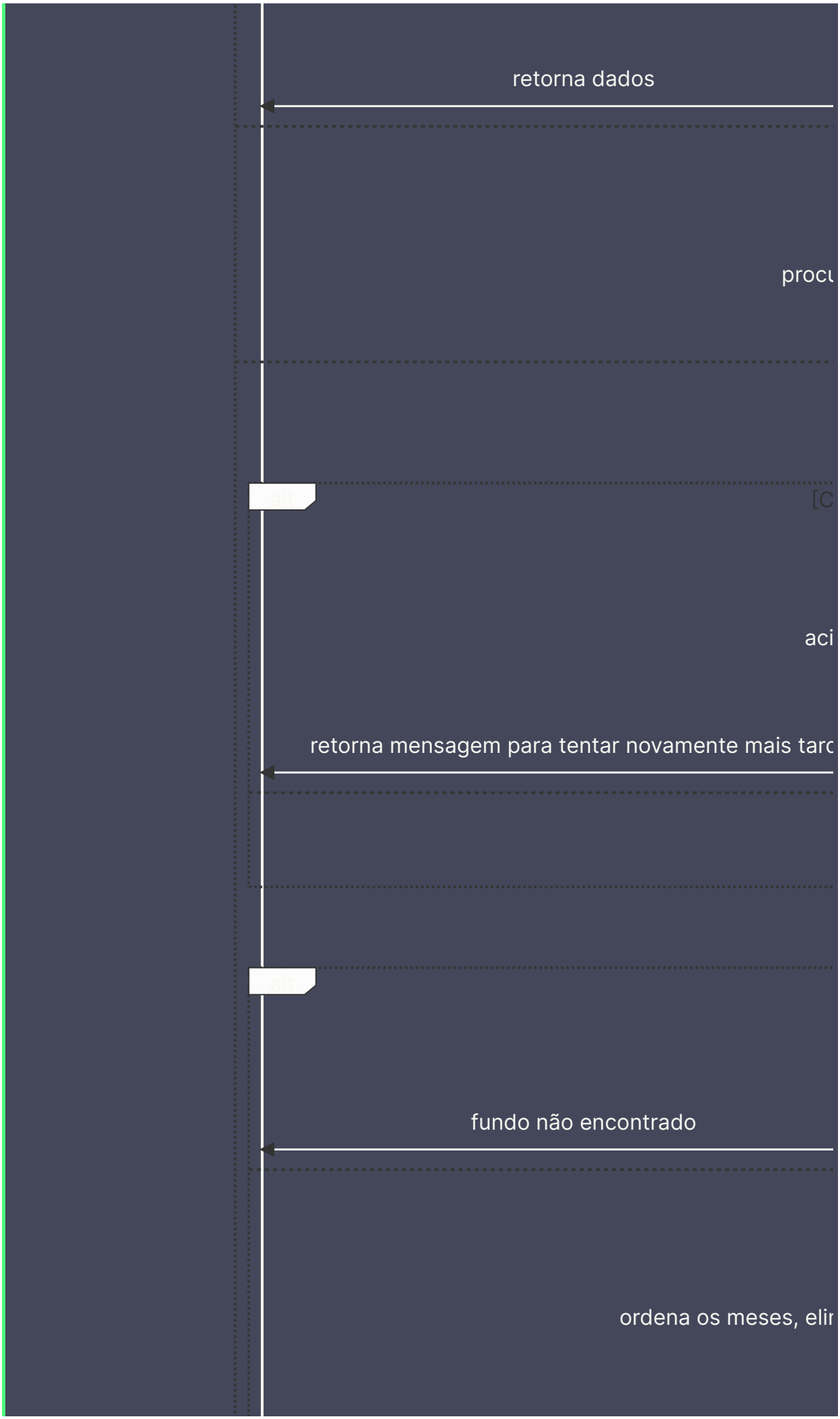
#python #diagram #draw

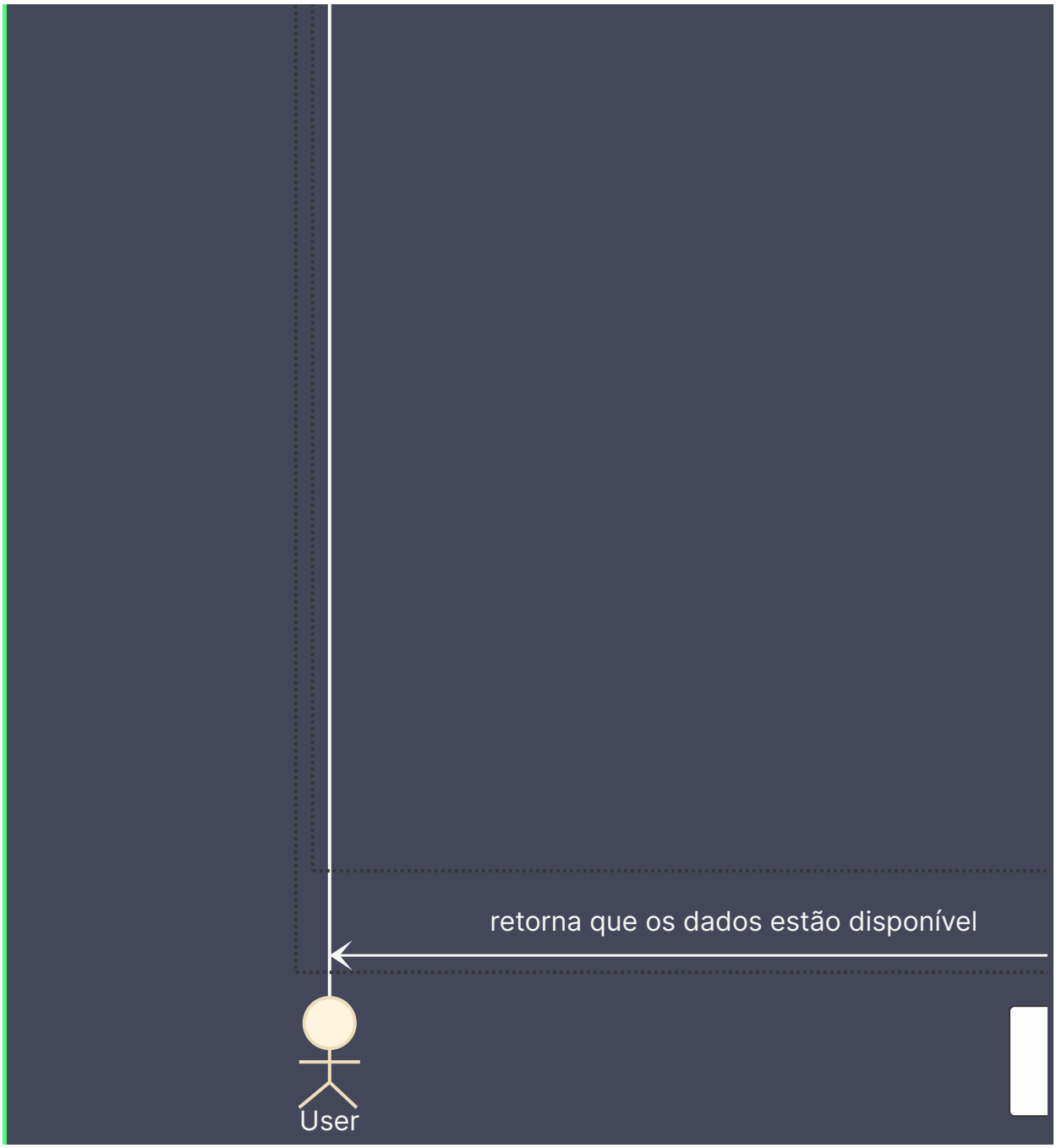
```
!pip install diagrams
from diagrams import Diagram, Cluster

graph_attr = {
"bgcolor": "transparent"
}

from diagrams.onprem.client import Users
from diagrams.oci.storage import StorageGateway as st
from diagrams.oci.network import LoadBalancer as lb
from diagrams.oci.compute import OKE
from diagrams.gcp.analytics import PubSub
from diagrams.onprem.network import Internet as Web
from diagrams.onprem.ci import GithubActions
from diagrams.k8s.clusterconfig import HPA
from diagrams.k8s.podconfig import CM

# filename="arch", outformat="jpg"
with Diagram("CVM Reader", show=False, graph_attr=graph_attr,
filename="arch", outformat="jpg") as diag:
        user = Users("Multiple Users")

        with Cluster("Kubernets"):

                ep = lb("LoadBalancer")
                api = OKE("API")
                msg = PubSub("Mensageria")
                cvm = Web("Site CVM")
                cm = CM("Config Map")
                act = GithubActions("CI/CD")
                hor_scaler = HPA("Horizontal Pod Scaler")

                with Cluster("Fila"):
                        queue = [OKE("Assinante1"),
                        OKE("Assinante1"),
                        OKE("AssinanteN")]

                with Cluster("Banco de Dados"):
                        db = st("Database")
```

```
                    db - [st("Database replica")]

        user >> ep >> api >> msg >> queue >> cvm
        queue >> db
diag
```



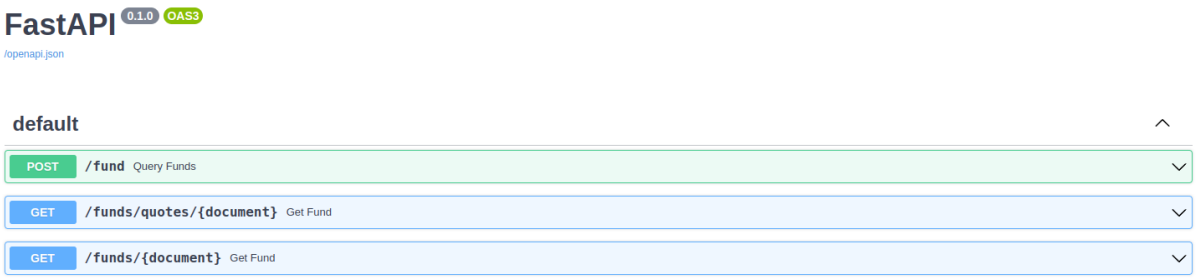



retorna dados
procu
alt
[C
aci
retorna mensagem para tentar novamente mais tarc
alt
fundo não encontrado
ordena os meses, elir



A solução proposta é um modelo arquitetural em micro serviços, divido em uma API Rest, com possui endpoints específicos e concentrando as regras de negócio.

Endpoints:



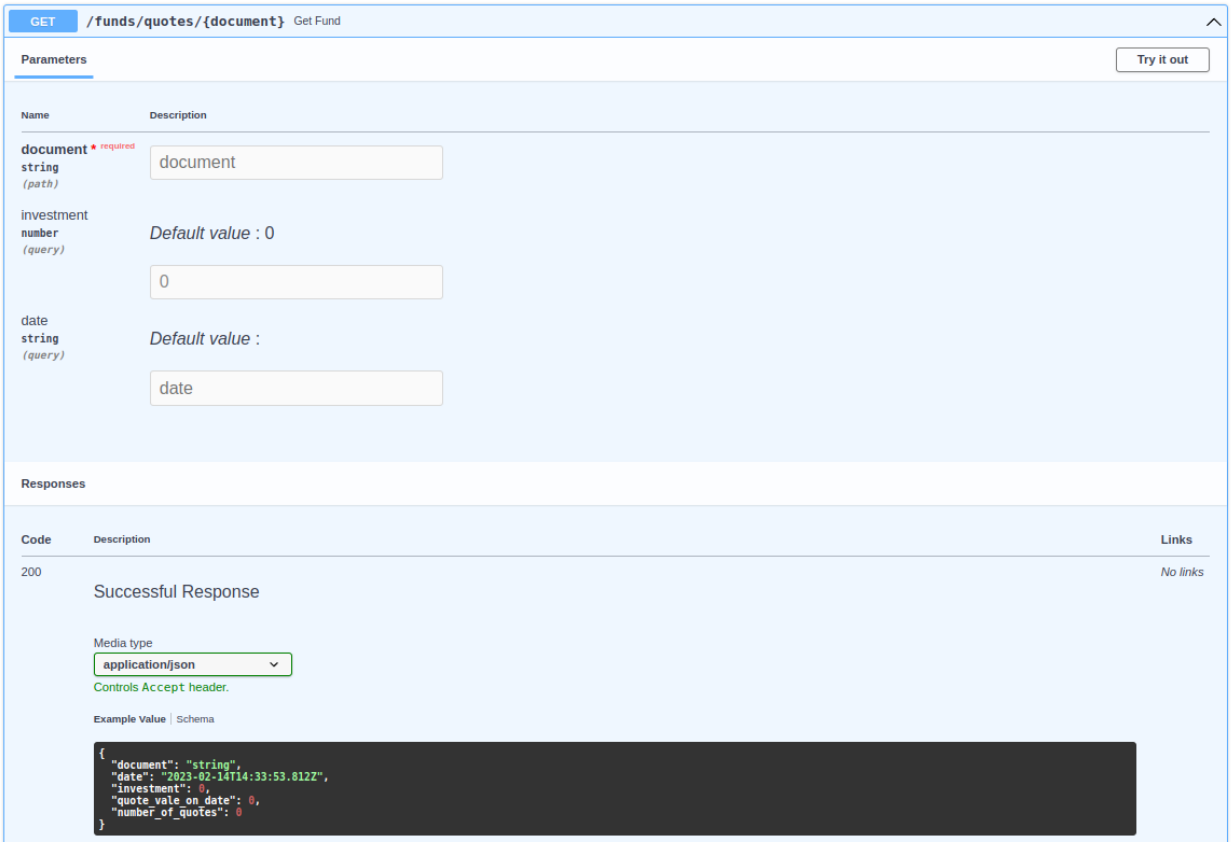
POST /fund → assíncrono e responsável por executar a carga dos dados.
requisição:

```
{
  "document": "string",
  "from_date": "string"
}
```

resposta:

```
{
  "document": "string",
  "from_date": "2023-02-14",
  "quotes": 0,
  "fund_name": "string",
  "fund_id": "string",
  "fund_released_on": "2023-02-14",
  "active": true,
  "timeseries": [
    {
      "timestamp": "2023-02-14T14:31:21.324Z",
      "value": 0,
      "owners": 0,
      "net_worth": 0
    }
  ]
}
```

GET /funds/quotes/(document)



principais regras de negócio da API:

- ao realizar uma request a API verifica e valida se todos os campos necessários foram passados e se os campos estão no formato esperado;
- após a validação da request verifica se o CNPJ já existe na base de dados:
  > - caso exista: vai verificar o último dia que houve atualização e confrontar com a data do servidor onde a aplicação esta rodando, caso os dados já sejam atuais irá apenas devolver os resultados a partir do banco de dados;
  > - a aplicação irá executar uma consulta no site da CVM, caso o CNPJ não seja encontrado será devolvido um erro pro usuário de CNPJ não encontrado.
  > - caso não exista na base de dados local ou necessite de atualização irá iniciar o fluxo de atualização. Cada mês necessário de atualização é publicado como uma mensagem independente para a fila do Pub/Sub.

Assinantes ou Workers

A essa aplicação cabe a responsabilidade de obter os dados diretamente do site da CVM. Como são uma aplicação independente são facilmente escalados horizontalmente pelo kubernetes.

Principais regras de negócio:

- Responsável por realiza a carga dos dados a partir do site da CVM, sendo essa aplicação uma assinante do fluxo de mensageria.
- para cada mensagem na fila a aplicação irá disponibilizar um thread (processador) livre que será responsável por fazer a chamada no site da CVM, realizando a carga dos dados, o processamento e ordenação dos dados obtidos para aquela mensagem (referente a um mês do fundo). Após a ordenação a aplicação ainda é responsável por executar a persistência desses dados em forma de TimeSeries.
- caso necessário a aplicação foi arquitetada para trabalhar de forma isolada, sendo possível escalonar horizontalmente esse etapa, criando um pool de assinantes, acelerando o processo consumo e processamento das mensagens na fila.

Escolha da Camada de Persistência de Dados

Principais soluções para persistência

1. PostgreSQL - é um dos principais bancos de dados relacionais de código aberto disponíveis hoje, com mais de 30 anos de história. É usado em uma variedade de aplicações e empresas em todo o mundo, desde sites de comércio eletrônico de grande porte até aplicativos de gerenciamento de conteúdo.

2. SQLite - é um sistema de gerenciamento de banco de dados gratuito, leve e de código aberto. É usado em muitos dispositivos, como smartphones, tablets, laptops e PCs. Ele foi criado para fornecer um meio de armazenar dados de forma rápida e segura para ser simples e fácil de usar.
3. MongoDB - é um banco de dados NoSQL orientado a documentos altamente escalável, flexível e de alto desempenho. É um dos mais populares do mundo, sendo usado por empresas de todos os tamanhos. Como usa documentos em formato JSON para armazenar dados, isso o torna ideal para aplicativos de alto volume que precisam de flexibilidade.
4. Redis - conjunto de estruturas versáteis de dados na memória que permite a fácil criação de várias aplicações personalizadas. Os principais casos de uso do Redis incluem cache, gerenciamento de sessões, PUB/SUB e classificações. É o armazenamento de chave-valor mais conhecido atualmente.

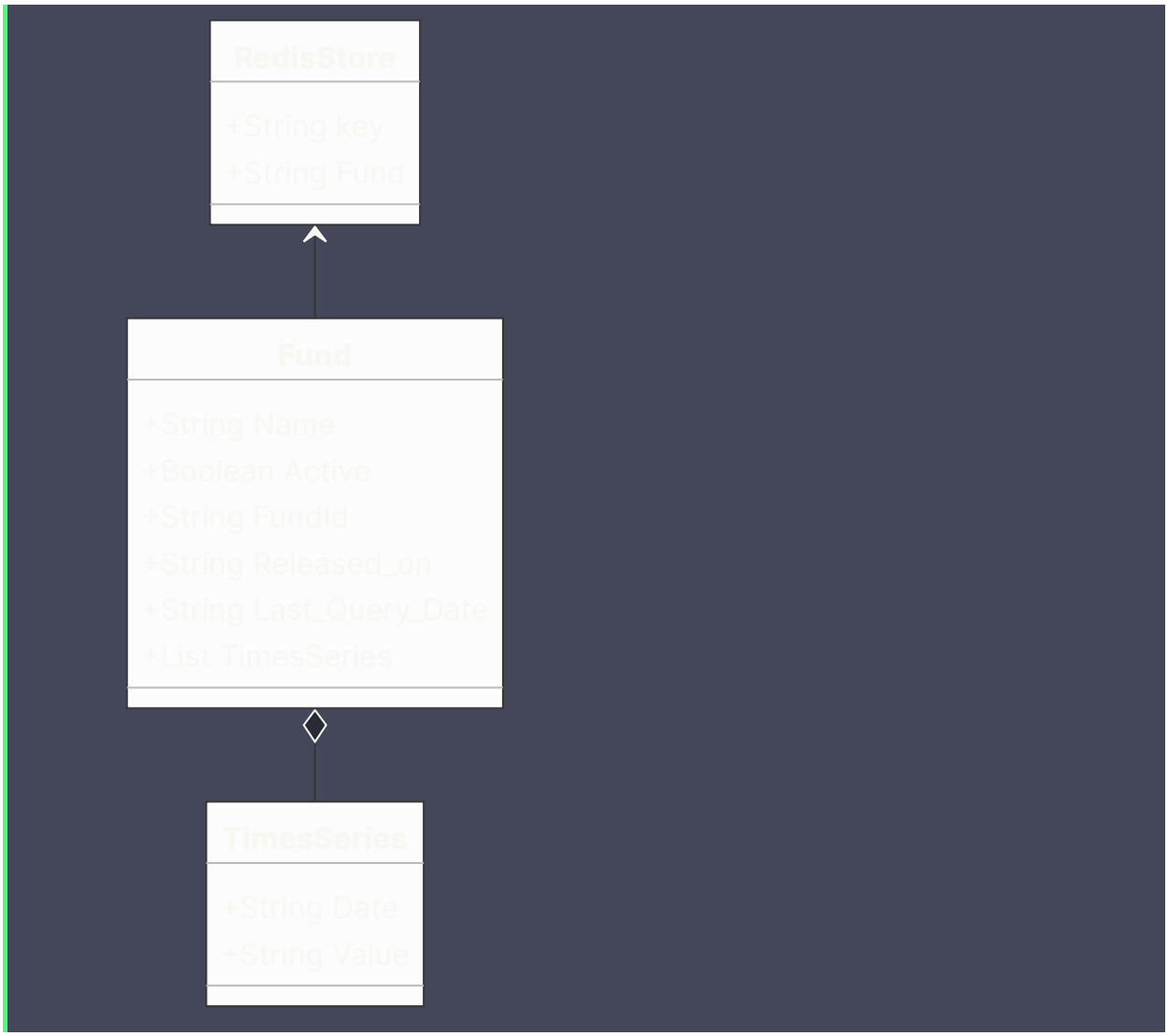
Justificativa para escolha

Conforme definido pelos requisitos não funcionais, um ambiente financeiro é baseado em três pilares confiabilidade, segurança e desempenho. O Redis, que significa Remote Dictionary Server, é um datastore de chave-valor rápido e de código aberto na memória. é um componente padrão e largamente utilizado em soluções na web para qualquer empreendimento significativo. o Redis é um dos mecanismos de código aberto mais usados, chamado de o banco de dados “preferido” pelo Stack Overflow por cinco anos consecutivos. Por causa de sua rápida performance, o Redis é uma opção popular para armazenamento em cache, gerenciamento de sessão, jogos, placares, análises em tempo real, dados geoespaciais, chamada de corrida, conversa/sistema de mensagens, transmissão de mídia e aplicações pub/sub.

Para tornar a vida do arquiteto mais simples, aliado a essa solução de alto desempenho a Redis INC criou um pacote completo de ferramentas que fornecem todos os principais componentes de uma arquitetura Microservice - o Redis Stream, Redis Pub Sub, Redis Search, Redis AI, Redis JSON, Redis Graph, Redis Time series, e muitas outras.

A stack Redis (Redis JSON, Redis Search, Redis Timeseries, Redis Graph, e Redis Bloom) reduz significativamente o caos tecnológico. Juntamente com estes, as estruturas de dados Pub-Sub e Streaming permitem uma comunicação assíncrona elegante entre seus micro serviços.

Modelo das entidades



Mensagens

Escolha do Framework para a API

Um framework é um conjunto de pacotes e módulos que tornam a programação mais fácil e rápida. Fornece uma série de ferramentas que são necessárias na hora de desenvolver uma aplicação.

No desenvolvimento web, existem certos procedimentos e elementos que são padronizados para qualquer tipo de site. Podemos citar operações de conexão cliente-servidor, autenticação de usuários, requisições a bancos de dados, etc.

Por que usar Frameworks?

Depois de entender porque utilizar framework, podemos listar algumas vantagens:

- Temos um aumento da segurança e da produtividade;
- Possibilita o trabalho em equipe;
- Ao utilizar um framework, você fará parte de uma comunidade de pessoas que também o utilizam, sendo possível tirar dúvidas e trocar experiências;
- Maior organização;

- O tempo de programação tem uma redução significativa.

Os principais Frameworks em Python

1. DJANGO - considerado por muitos o melhor disponível para desenvolvimento web, por ser bastante robusto e completo, Ele é gratuito e open source, podendo ser utilizado para construir qualquer tipo de site desejado.
2. FLASK - é um microframework Python voltado ao desenvolvimento web, baseado nas bibliotecas WSGI Werkzeug e Jinja2. Sua ideia é ser um modelo simples para a construção de sites, buscando facilitar tarefas e economizar tempo. Baseada no Web Server Gateway Interface (WSGI).
3. FALCON - é um framework Python que foi concebido com a intenção de ser confiável e apresentar uma performance de alto nível. Seu código base é otimizável e extensível. Ele é utilizado para construir Application Programming Interface (APIs), em português, Interface de Programação de Aplicativos. APIs são conjuntos de normas que possibilitam a integração entre plataformas.
4. FastAPI - é um framework Python focado no desenvolvimento de API's, tem como principais características ser moderno, rápido e simples. Tem como base o Asynchronous Server Gateway Interface (ASGI). É um framework relativamente novo, carregado com diversas utilidades das versões mais recentes do Python e recursos mais modernos, como documentação dos endpoints automático no Swagger UI, programação assíncrona e tem ganhado bastante popularidade.

Definição e escolha do Framework

Critério	Django	Flask	Falcon	FastAPI
Documentação	x	x		x
Foco em API		x	x	x
Performance			x	x
Facilidade de uso	x	x	x	x
Auto Documentação				x
Auto Validação da chamada	x			x
RESULTADO	3	3	3	6

mais detalhes podem ser vistos em *link*

Demais soluções embarcadas

Versionamento de Código e GIT

O controle de versão, também chamado de source control, é um sistema que rastreia as mudanças no código durante todo o ciclo de vida do desenvolvimento e mantém um registro de cada alteração. Estes sistemas também armazenarão detalhes tais como autoria e carimbo de data e hora junto com cada versão. Isto permite às equipes comparar diferenças ou determinar seções de código problemáticas, revendo o histórico de revisões.

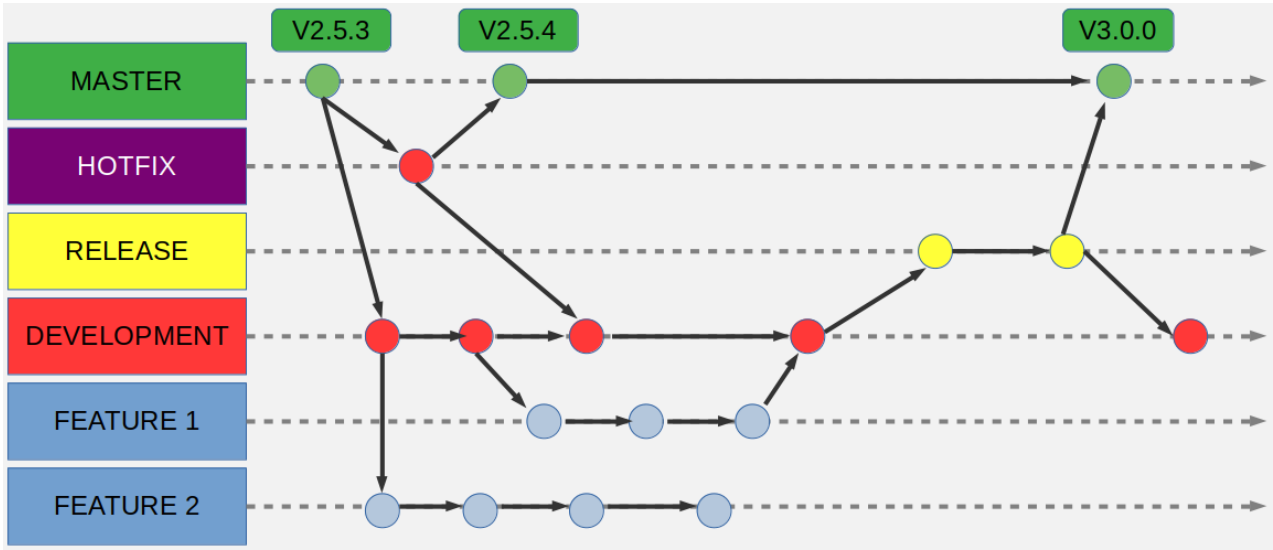
GIT

Git – é uma das, senão a mais utilizada ferramente de SCM (Source Control Management) presentes no mercado de TI e um ponto muito importante, é disponibilizado de forma 100% *Open Source*. O Git foi desenvolvido em 2005 por **Linus Torvalds** após a revogação da licença do BitKeeper, utilizado até então para manter o código fonte do Kernel *Linux*.

Atualmente é a principal ferramenta de gerenciamento de código fonte, sendo a base para as metodologias de desenvolvimento ágil, como: XP, Agile, Scrum. Não somente nessas metodologias, mas entrando também no universo *DevOps*, onde vemos as soluções de SCM como a base para as esteiras de desenvolvimento como o *Continuous Integration e Continuous Delivery*, conhecido como *Pipeline CI/CD*.

GITFlow

GitFlow é uma forma de **gerenciamento de branch**, ela propõe uma forma de configurar as branchs para que se tenha uma separação em Master, Release, Desenvolvimento, Features e Hotfix, sendo:



Master – Branch principal, que conteria seu código de produção.

Release – Branch para release, lançamento de versões de sua aplicação que posteriormente poderiam ser *mergiadas* na branch master.

Desenvolvimento – Branch para manter o código em atual desenvolvimento.

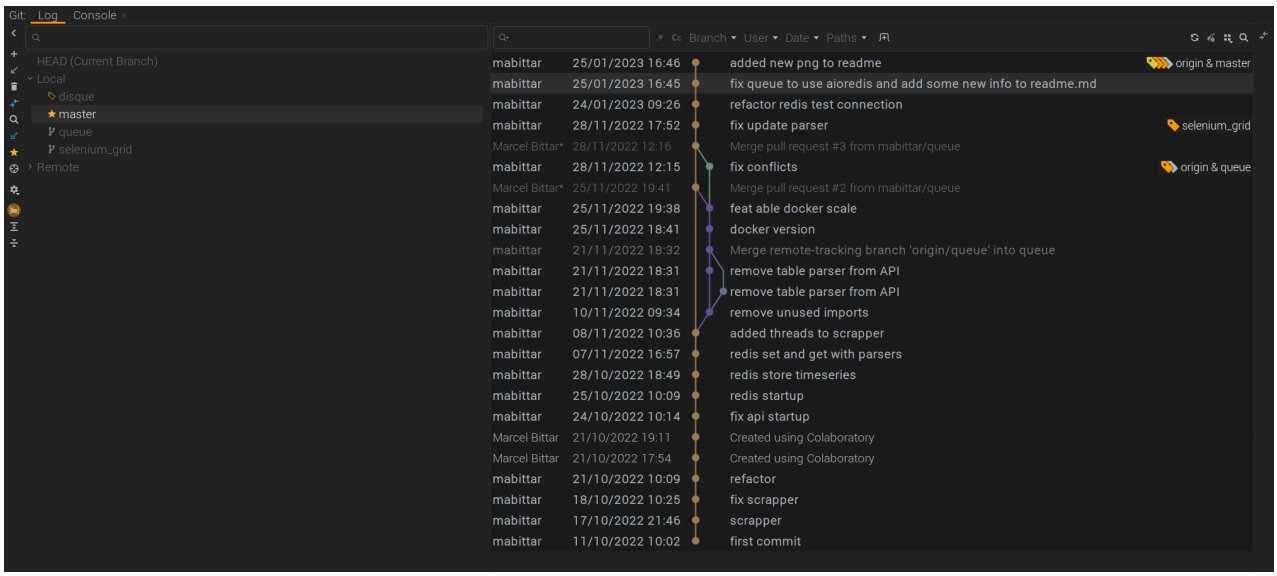
Features – Branchs que são copias de desenvolvimento onde justamente acontece a criação de uma nova funcionalidade do software.

Hotfix – Branch criada quando realiza-se alguma atualização de segurança que normalmente precisa entrar diretamente em produção.

GitHub do projeto

O repositório remoto pode ser acessado em:
https://github.com/mabittar/funds_pricer

Histórico do projeto até a data atual



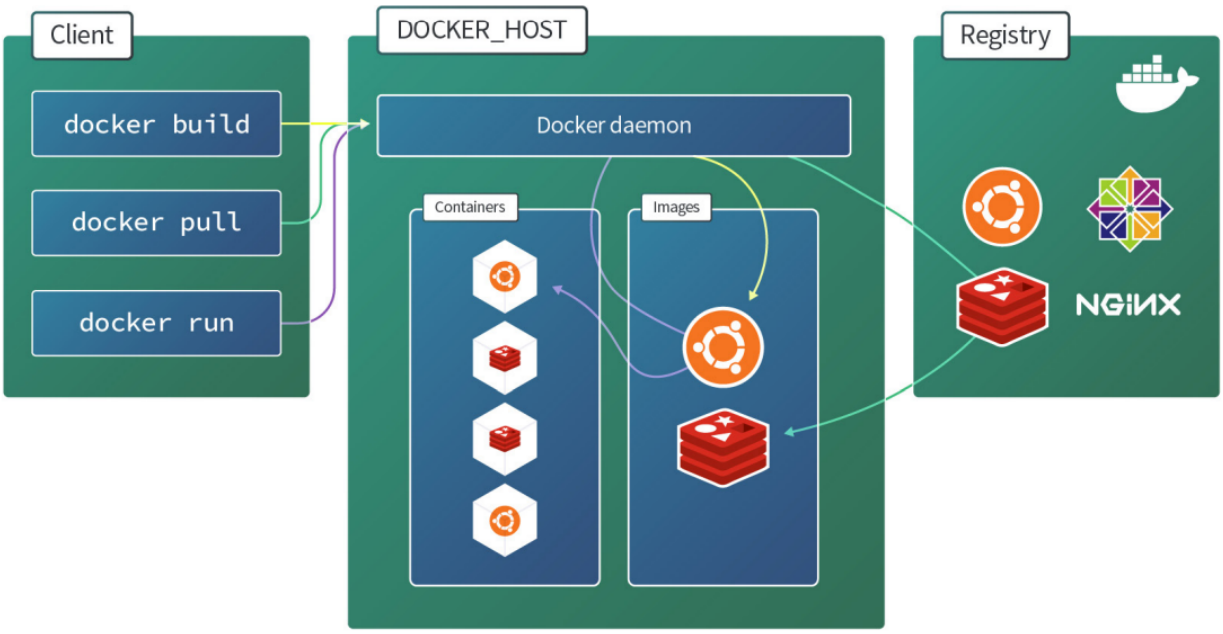
Docker

Docker é um software de código aberto, usado para desenvolver aplicativos dentro de containers virtuais, uma Plataform as a Service (PaaS). Desenvolver aplicativos em containers permite que vários aplicativos funcionem em ambientes totalmente isolados ou em conjunto.

Ao desenvolver um aplicativo utilizando Docker, economiza-se o esforço de solucionar problemas de incompatibilidade entre diferentes sistemas. Isso porque, o Docker é um software executado da mesma forma em todos os ambientes.

Como o Docker é um pacote de software com todas as dependências necessárias a sua execução, todas as configurações e instrução para iniciar ou parar o container estão prontas na imagem do Docker criado.

A figura a seguir ilustra os vários componentes da plataforma e como eles se interconectam.



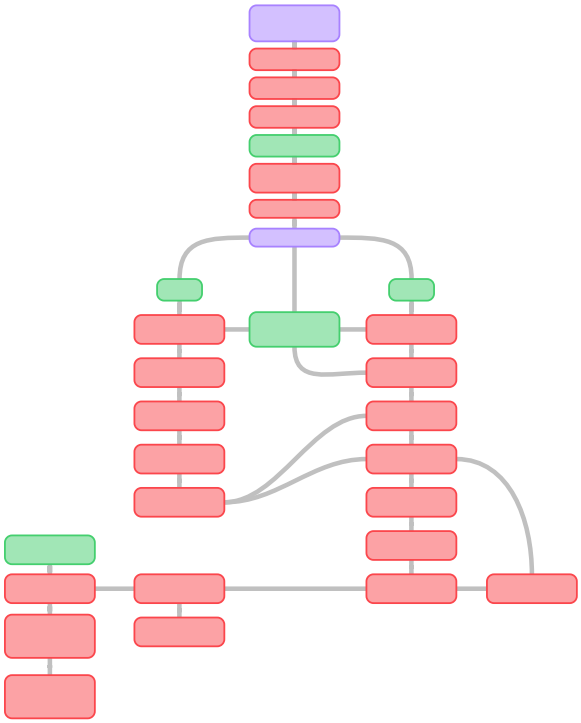
O cliente Docker, é o principal componente utilizado para criar, gerenciar e executar aplicativos em containers. Sendo o principal método de controle.
O servidor Docker ou deamon é que aguarda as instruções feitas pelo cliente e gerencia as imagens criadas
As imagens do Docker, são responsáveis por instruir o servidor Docker com os requisitos sobre como criar um container Docker. Essas imagens podem ser baixas ou criadas pelo desenvolvedor.
O Registro do Docker é uma aplicação que funciona em paralelo utilizado para hospedar e distribuir imagens do Docker. O registro é o responsável por armazenar as imagens localmente.

Quais as vantagens de trabalhar com Docker

- Portabilidade - esse é o principal atrativo do Docker, pois permite que os desenvolvedores criem ou instalem aplicativos em qualquer máquina e tenham certeza que irá funcionar.
- Automação - com a ajuda de rotinas (cron) os desenvolvedores podem automatizar facilmente seu trabalho
- Comunidade - Hoje existe mais de 9 milhões de imagens Docker. Diversas aplicações podem ser encontradas e utilizadas de forma isolada, independente da plataforma.

Planejamento e Desenvolvimento

RoadMap de Desenvolvimento



Sprint 1

Proof of Concept

Desenvolver script para verificar a viabilidade de extração dos dados da CVM. Avaliar quais bibliotecas serão necessárias para manipular os dados e navegar entre as diferentes datas.

Script para Extração dos Dados da CVM

Com o PoC em estágio avançado é possível começar o desenvolvimento para transformar o PoC em um script, definindo os modelos de representação das TimeSeries em runtime e no Banco de Dados.

Sprint 2

Divisão em Micro serviços

Com o avanço do script será possível investigar o contrato da Requisição da chamada a ser feita na API, bem como esboçar as regras de Negócio.

Ainda, a partir do script inicial, será possível definir quais dados deverão circular pelo ambiente de mensageria. E começar o desenvolvimento da fila para consumo das mensagens.

Para persistência dos dados, testes de conexão e dos demais módulos o Redis deverá estar funcionando no ambiente.

Ao desenvolver o micro serviço responsável pela requisição será necessário elaborar os middlewares de conexão com banco de dados, health check e logger das requisições e respostas com timer para observabilidade da aplicação

Sprint 3

Nessa sprint o desenvolvimento focará nas soluções de recebimento e validação das requisições através da API, validação das regras de negócio, consulta ao banco de dados, envio de mensagens ao serviço de PubSub e devolução das TS ao cliente no ambiente da API. Para o serviço de consumo das mensagens na fila, o foco será em persistir os dados e habilitar o serviço em multithreads para ganho de eficiência.