Projeto SSC0158 - Grupo 13 - Checkpoint 1

Integrantes

- Douglas Tallach
  9292708
  douglas.tallach@usp.br

- Juliano Fantozzi
  9791218
  juliano.fantozzi@usp.br

- Pedro Pastorello Fernandes
  10262502
  pedropastorf@usp.br

- Vinicius Leite Ribeiro
  10388200
  viniribeiro@usp.br

O projeto

Nós do grupo 13 propomos o desenvolvimento de um sistema simples de ETL (Extração, Transformação e Armazenamento) distribuído.

O sistema irá coletar dados públicos disponíveis na API do Twitter sobre mensagens e assuntos relacionados ao Covid-19 e a quarentena.

Após coletados, os dados serão processados e armazenados localmente para serem consultados e apresentados no momento de interesse.

Todos os componentes internos irão rodar como containers nas máquinas do LaSDPC destinadas ao nosso grupo.

A ideia é entregar um sistema simples que seja otimizado para rodar em ambientes com poucos recursos, porém que seja escalável o suficiente para processar grandes volumes de dados.

Componentes externos

- **API externa a ser consumida (Twitter):** Será a fonte de onde os dados serão coletados, no caso com chamadas de API.

- **Google Data Studio (SaaS):** Irá consumir os dados da nossa plataforma, sendo o destino final deles. Será a interface do usuário final, aonde ele poderá usar os dados para criar apresentações e dashboards.

Componentes internos

- **Extrator:** Serviço de extração de dados desenvolvido em Go.
  Coleta dados das fontes externas, pré-processa, agrupa e os envia para o serviço de processamento por meio do Kafka.

- **Apache Kafka:** Serviço de mensageria.
  Realizará o transporte dos dados do Extrator para o Transformador. Os dados serão agrupados em mensagens com esquemas definidos e em filas ordenadas, permitindo o desacoplamento dos serviços de transformação e extração.

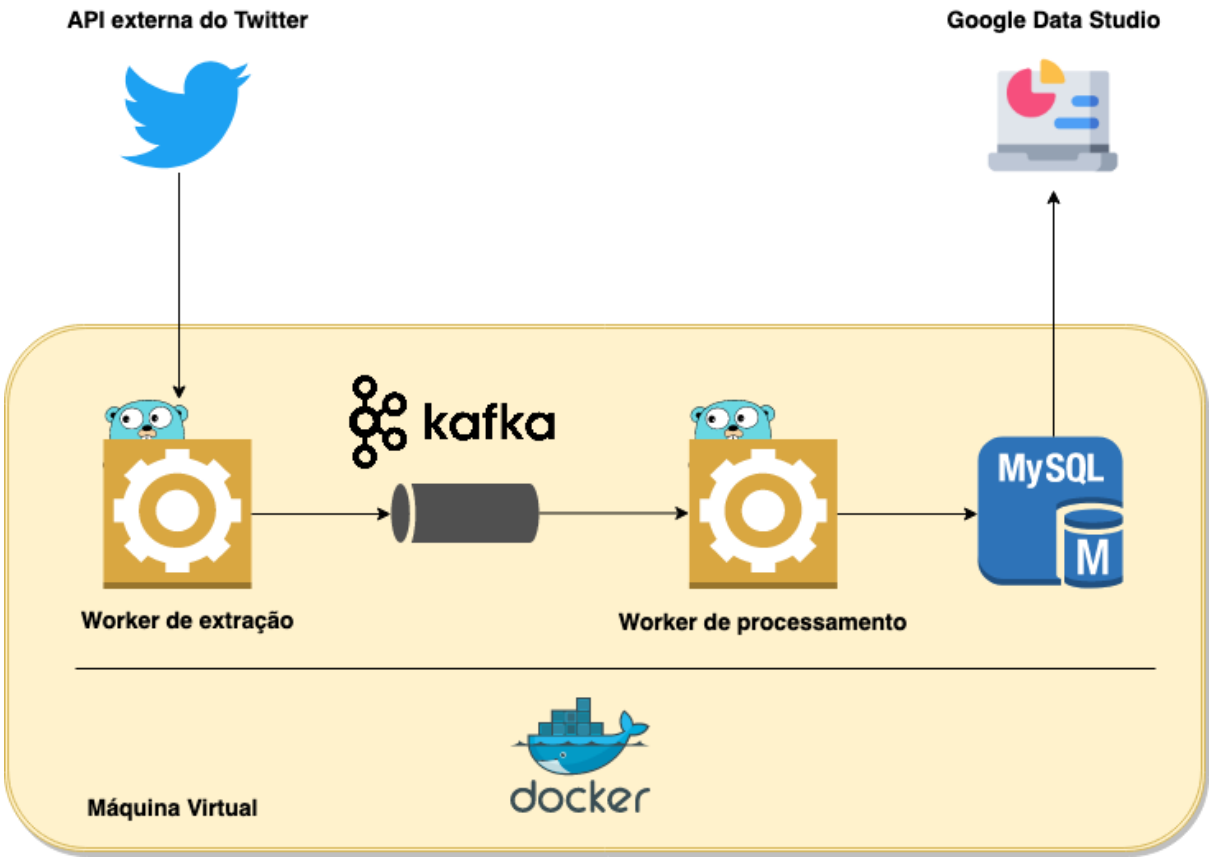
- **Transformador:** Serviço de processamento de dados desenvolvido em Go.
  Consome os dados pré-processados do Kafka, realiza todo o processamento principal neles e os insere no banco de dados.

- **MySQL:** Banco de dados.
  Será consultado pelo Google Data Studio com chamadas criptografadas. A inserção dos dados será feita sem criptografia pelo Transformador.

- **Docker:** Middleware de virtualização e orquestração.
  Plataforma que irá suportar a execução dos outros componentes.

Diagrama da arquitetura

O diagrama abaixo ilustra a arquitetura do sistema e o fluxo dos dados entre os seus componentes:



Protocolos de comunicação

Os protocolos de comunicação que serão usados, na ordem do fluxo de dados:

- **REST** para chamadas a APIs externas
- Apache Avro sobre HTTP para serialização das mensagens enviadas e recebidas pelo Kafka
- **SQL sobre HTTP** para inserção no banco de dados
- **SQL sobre HTTPS** para consulta ao banco de dados

Toda a comunicação interna do sistema será feita via HTTP em texto puro ou seja, sem criptografia. Estamos contando com uma ambiente de infraestrutura no qual podemos confiar nas requisições trafegadas internamente.

As únicas chamadas criptografadas (HTTPS) serão as feitas externamente para o banco de dados (Google Data Studio) e para isso um certificado TLS será gerado.

Em exceção a isso, apenas o acesso via SSH feito pelos desenvolvedores e pelas ferramentas de configuração para desenvolvimento e manutenção.

Tecnologias escolhidas

- Google Data Studio
  Camada de visualização dos dados (SaaS).

  Vai ser usado para fazer apresentações dos dados processados, consumidos diretamente do banco de dados por queries.

- Ansible
  Camada de configuração de infraestrutura.

  Será usado para configurar o ambiente e instalar dependências nas máquinas do lab via SSH.

- Terraform
  Camada de configuração de infraestrutura.

  Será usado para provisionar infraestrutura em algum provedor de nuvem, no nosso caso AWS (IaaS). Essa infraestrutura será usada para desenvolvimento.

- Docker
  Camada de virtualização.

  Utilizado para virtualizar, isolar e controlar o ciclo de vida das aplicações. Docker faz isso por meio de containers e uma API usada para interagir com os containers em execução.

  No geral, containers em execução são muito mais leves que Máquinas Virtuais e foi por isso que escolhemos a tecnologia. Há mais informações sobre isso no final do documento.

- Docker Compose
  Camada de orquestração.

  Utilizado para orquestrar os containers em execução e definir a infraestrutura usando a API do Docker.

  Será usado para controlar o consumo de recursos, configurar a rede e o roteamento, gerenciar volumes de armazenamento e monitorar a saúde dos serviços.
  Em seus arquivos de configuração são explicitados os requisitos e dependências dos serviços em relação à infraestrutura.

- Golang
  Linguagem do backend.

  É uma linguagem compilada otimizada para processamento assíncrono e distribuido.

  Não só por isso, Go é muito muito eficiente em termos de consumo de recursos e tempo de execução. É muito utilizada para processamento de dados e computação distribuída e em alguns grandes projetos de computação em nuvem, como o proprio Docker.

  As bibliotecas e frameworks a serem usadas serão definidas quando iniciarmos de

fato o desenvolvimento. No momento não podemos afimar nada com certeza em relação a isso.

- Apache Kafka
  Camada de transporte de dados.

  Será o serviço de mensageria usado, inicialmente, para carregar os dados pré processados vindos do serviço de extração para o serviço de transformação.

- MySQL
  Camada de armazenamento.

  Servidor de banco de dados relacional de código aberto. Foi o serviço de banco de dados escolhido por ser muito utilizado no mercado, tendo uma grande comunidade e integração com o Google Data Studio.

Nota sobre virtualização

Atualmente as duas formas de virtualização mais relevantes são Máquinas Virtuais (VMs) e Containers.

Decidimos que iremos usar containers para nosso projeto. Abaixo estão as considerações relevantes para essa decisão. Tudo isso foi levado em conta pensando nas nossas limitações de recursos (CPU, RAM, número de VMS) na infraestrutura que temos disponível no lab.

- ➔ **VMs:**
- ➔ virtualização de hardware, memória, rede, etc
  - ◆ custo alto de recursos no host que executa
    - CPU
    - RAM
  - ◆ isolamento de recursos
    - entre VMs
    - entre guests e o host
  - ◆ maior segurança
    - melhor para executar código de terceiros (como montar um provedor de nuvem)

- ➔ **Containers:**
- ➔ não há virtualização de hardware, apenas de software (user-space do SO)
  - ◆ imagens leves e flexíveis
    - maior agilidade no boot/shutdown
    - menos custo de recursos no host

- ➔ Kernel e hardware compartilhado
  - ◆ entre containers
  - ◆ entre guests e o host
  - ◆ menos seguros
    - ok para executar nosso próprio código em nossa infra
    - pode ser um problema ao vender containers como serviço - é preciso limitar o acesso dos containers ao Kernel/hardware que os executa

Nota sobre o ambiente na AWS

Não temos acesso de root às VMs do Lab, nem ao firewall que está associado a elas, nem a capacidade de subir novas instâncias de acordo com nossa necessidade sem ter que contatar o professor ou os monitores.

Para termos mais liberdade e agilidade no desenvolvimento, criamos um ambiente de VMs na AWS que irá replicar o ambiente do LaSDPC. Esse ambiente será usado para apenas desenvolver e testar o sistema.