

Sistema dm_v3_chain_explorer

Submissão de case Data Master

Nome: Marco Aurelio Reis Lima Menezes

Cargo: Engenheiro de Dados Sênior

Badge Atual: Data Advanced na carreira Engenharia de dados

Objetivo: Obter badge Data Expert

1. Objetivo do case

O objetivo final desse trabalho é sua submissão para o programa Data Master, e posterior apresentação do mesmo à banca de Data Experts. Nessa apresentação serão avaliados conceitos e técnicas de engenharia de dados, entre outros campos, aplicados na construção prática deste sistema intitulado **dm_v3_chain_explorer**.

Para alcançar tal objetivo final e, dados os requisitos do case, especificados pela organização do programa, para a construção desse sistema foram definidos objetivos específicos, categorizados em objetivos de negócio e objetivos técnicos.

1.1. Objetivos de negócio

A solução apresentada aqui tem como objetivo de negócio prover um sistema capaz de capturar, ingestar e armazenar dados com origem em **redes P2P do tipo blockchain**. Quando se fala em blockchain vale distinguir que o termo blockchain pode ser usado pra se referir a:

- **Estrutura de dados blockchain** que armazena blocos encadeados. Cada bloco contém um conjunto de transações efetuadas por usuários da rede, entre outros metadados. Um desses metadados é o hash do bloco anterior, o que garante a integridade da cadeia de blocos.
- **Rede de blockchain**, de topologia Peer-to-Peer onde nós validam transações e mineram novos blocos, estes construídos em uma estrutura de dados blockchain. Todos os nós da rede possuem uma cópia dessa estrutura de dados e são sincronizados entre si. Assim, novos blocos minerados, após consenso, são adicionados ao blockchain e sincronizados entre o restante dos nós para que a rede possa validar a integridade das transações contidas em todos os blocos.

1.1.1. Redes Blockchain Públicas

Blockchains públicas são redes P2P que armazenam uma estrutura de dados do tipo blockchain. Por serem públicas permitem que qualquer nó possa fazer parte na rede, aumentando a descentralização da mesma. Com isso uma pessoa comum desde que com requisitos de hardware e software e rede satisfeitos podem fazer parte dessa rede descentralizada.

1.1.2. Oportunidades em blockchains públicas

Atualmente existem inúmeras redes blockchain públicas onde circula uma quantidade significativa de capital e existe um ecossistema diverso de aplicações. Essas redes são usadas para, desde a transferência de tokens entre endereços, até a execução funções em contratos inteligentes para aplicações dos mais diversos fins. Entre as redes de blockchain públicas se destacam:

- **Bitcoin:** 1ª blockchain construída na qual seu token nativo, o BTC tem alto valor de mercado;
- **Ethereum:** 1ª blockchain com possibilidade deploy e interação com contratos inteligentes;
- **Polygon, arbitrum, Optimist, Base:** Blockchains de Layer 2 que rodam no topo da ethereum e são EVM (executam a Ethereum Virtual Machine).

Existem uma infinidade de blockchains públicas. Nesse trabalho o interesse é naquelas que possuem contratos inteligentes. Contratos inteligentes são desenvolvidos através de uma linguagem de

programação e toda uma pilha de software. Para a Ethereum e suas Layers 2 citadas, que possuem o maior market cap somadas (só a ethereum tem 65% de todo capital retido em contratos inteligentes) foi criada uma virtual machine chamada EVM. Por isso aqui o interesse se volta para essas redes plockchain públicas compatíveis com a Ethereum.

Vamos explorar aqui 2 possibilidades.

1. Em <https://defillama.com/chains> é possível ver o quanto de está alocado em cada uma dessas redes. A rede Ethereum, por exemplo, é a rede com maior capital preso no protocolo (TVL). É inegável que as instituições financeiras olham com interesse para oferecer como produto a venda desses tokens, tais como BTC e ETH. Mas qual seria o 1º passo para que uma instituição financeira, altamente regulada, possa oferecer um produto como esse? E prover mecanismos de segurança, para, por exemplo, monitorar e assegurar quem sansões de lavagem de dinheiro e financiamento ao terrorismo não estão sendo infringidas?

2. Nessas blockchains, além de transações de transferência de token nativo, são executadas funções em contratos inteligentes. Esses contratos são programas deployados em um bloco da rede com endereço próprio. Após deployados esses contratos passam a estar disponíveis para interação. Isso permite que aplicações descentralizadas (dApps) sejam criadas dentro do protocolo. Dessa forma, é possível criar aplicações financeiras descentralizadas (DeFi) que permitem empréstimos, trocas de tokens, entre outras funcionalidades. Em <https://defillama.com/> é possível ver uma lista de aplicações DeFi e o volume de capital aplicado em cada uma delas. Ao capturar as transações em tempo real, transações do tipo interação com contratos inteligentes, que possuem um campo de dados chamado `input`, é possível monitorar qual usuário está chamando qual função do contrato e com qual argumento. Não cabe aqui mensurar a vasta gama de oportunidades que se abrem ao capturar esses dados e processá-los.

1.2. Objetivos técnicos

Para alcançar os objetivos de negócio propostos é preciso implementar um sistema capaz de capturar, ingestar, processar, persistir e utilizar dados da origem mencionada. Para isso, foram definidos os seguintes objetivos técnicos:

- Criar sistema de captura de dados brutos de redes de blockchain públicas.
- Criar um sistema de captura de dados de estado de contratos inteligentes.
- Criar um sistema de captura agnóstico à rede de blockchain, prém restrito a redes do tipo EVM (Ethereum Virtual Machine).
- Criar uma arquitetura de solução que permita a ingestão lambda.
- Minimizar latência e números de requisições, e maximizar a disponibilidade do sistema.
- Criar um ambiente reproduzível e escalável com serviços necessários à execução de papéis necessários ao sistema.
- Armazenar e consumir dados pertinentes a operação e análises em bancos analíticos e transacionais.
- Implementar ferramentas monitorar o sistema (dados de infraestrutura, logs, etc).

1.3. Observação sobre o tema escolhido

Dado que a tecnologia blockchain não é assunto trivial e também não é um requisito especificado no case, no corpo principal desse trabalho evitou-se detalhar o funcionamento de contratos inteligentes e aplicações DeFi. Porém, é entendido pelo autor desse trabalho que, apesar de não ser um requisito especificado no case, inúmeros conceitos aqui abordados exploram com profundidade campos como:

- Estruturas de dados complexas (o próprio blockchain);
- Arquiteturas de sistemas distribuídos e descentralizados;
- Conceitos relacionados a finanças.

Portanto, a escolha desse tema para case é uma oportunidade de aprendizado e de aplicação de conhecimentos de engenharia de dados, arquitetura de sistemas, segurança da informação, entre outros. Caso o leitor deseje se aprofundar mais nesse tema, a ****seção Appendice**** desse documento é um ótimo ponto de partida.

2. Arquitetura do case

Nesse tópico está detalhada a arquitetura de solução e técnica do **dm_v3_chain_explorer**.

2.1. Arquitetura de solução

Para detalhar a arquitetura de solução desse trabalho, é preciso extrair dos objetivos técnicos decisões tomadas e requisitos para o sistema.

2.1.1. Decisões tomadas

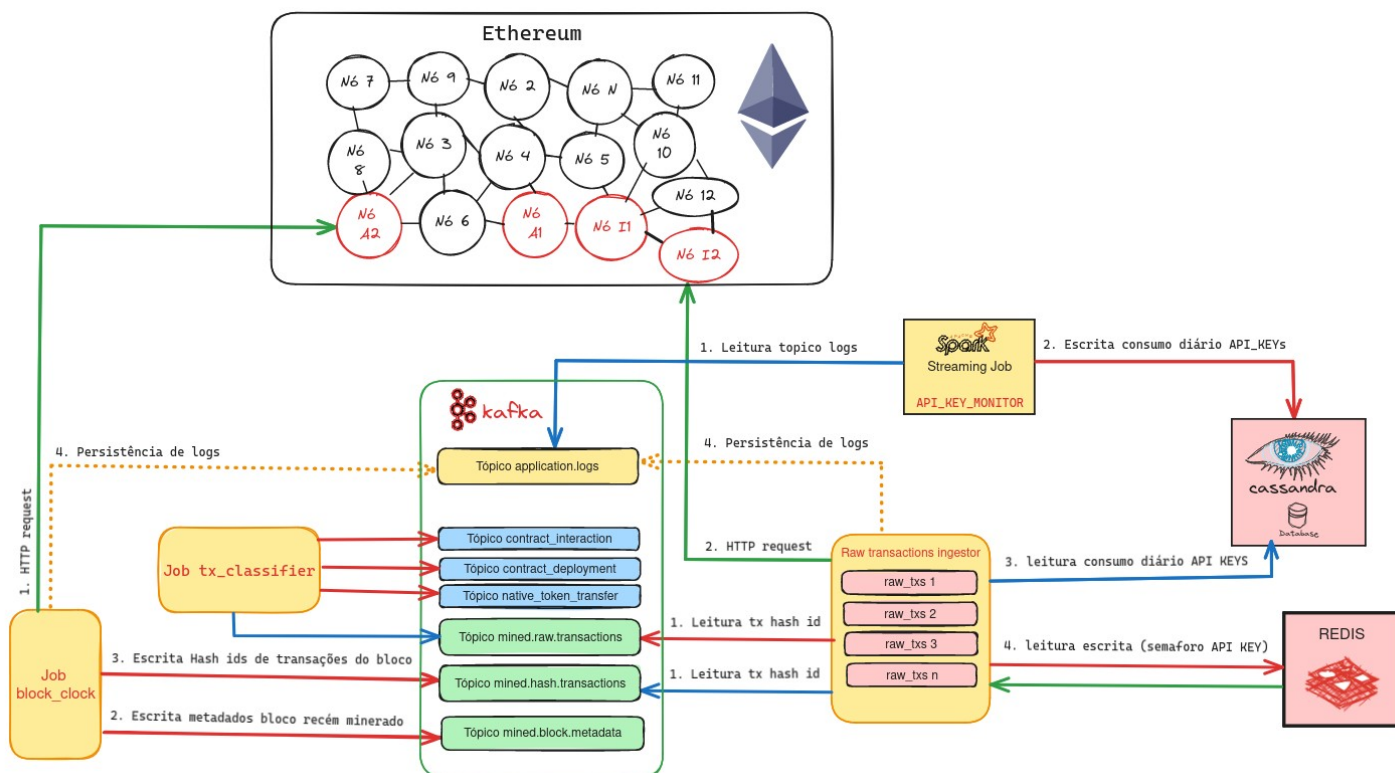
- Foi escolhido fazer ingestão de dados da rede Ethereum. Isso se justifica pelo fato de que a rede Ethereum é a rede com maior capital preso em tokens, sendo a melhor escolha para um requisito de negócio. Além disso, a rede Ethereum é uma rede do tipo EVM. Portanto, a solução proposta é agnóstica à rede de blockchain, mas restrita a redes do tipo EVM.
- Para capturar dados da rede é preciso ter um nó na rede Ethereum. Devido aos requisitos de hardware e software necessários para deploy de um nó on-premises ou em cloud, foi escolhido o uso de provedores de `Node-as-a-Service`. Esses provedores fornecem uma API para interação com um nó da rede Ethereum limitando o número de requisições por segundo e por dia usando uma API KEY.

2.1.2. Requisitos para o sistema

O requisito inicial do sistema é capturar e ingestar os dados brutos de transações da rede Ethereum com a menor latência possível. Dado que foi optado pelo uso de provedores de `Node-as-a-Service`, é preciso considerar os seguintes fatores:

- As requisições em nós disponibilizados por um provedor de `Node-as-a-Service` são limitadas. O provedor infura por exemplo, sua API KEY em **plano gratuito** oferece **10 requisições por segundo** e **100.000 requisições por dia**.
- Na rede Ethereum, um bloco é minerado a cada 8 segundos e contém cerca de 250 transações. Isso resulta em uma quantidade diária de mais de 2 milhões de transações. Portanto, para capturar esses dados, é necessário minimizar o número de requisições e manter um controle de uso das API KEYS.

Dados os pontos o desenho de solução para ingestão de dados da Ethereum em tempo real está ilustrado a seguir.



Nas seções seguintes é demonstrado como os diferentes serviços colaboram entre si para capturar os dados e colocá-los em tópicos do Kafka.

2.2. Arquitetura Técnica

A arquitetura técnica desse sistema é composta por diferentes camadas, cada uma com um conjunto de serviços que interagem entre si. As camadas são:

3. Explicação sobre o case desenvolvido

3.1. Histórico de desenvolvimento

Como foi o desenvolvimento desse case ao longo do tempo

3.2. Temática do case

Qual é a temática do case e por que a aplicabilidade desse case pode se tornar algo maior?

4. Aspectos técnicos desse trabalho

- **Docker:** A ferramenta docker foi utilizada para construção de imagens de serviços e orquestração de containers. As definições de imagens e serviços estão presentes no diretório *docker/* e *services/* na raiz do repositório desse trabalho.
- **Apache Kafka:** O Apache Kafka foi utilizado para captura e ingestão de dados brutos da rede Ethereum. A definição de serviços para o Kafka está presente no arquivo *services/cluster_dev_fast.yml*.

5. Reprodução da arquitetura e do case

Um dos requisitos deste trabalho é que a solução proposta seja reproduzível. Essa característica da reprodutibilidade é importante pelos seguintes motivos:

- A reprodução do trabalho permite os avaliadores executarem o sistema e entenderem como ele funciona.
- Esse trabalho é um sistema complexo, tendo diversos serviços interagindo com aplicações para que sua finalidade seja alcançada, como exposto nas seções anteriores. Provêr um passo-a-passo para o leitor reproduzi-lo em seu ambiente local dá a ele a oportunidade de entendê-lo em partes e como um todo. E extrair partes úteis para um projeto pessoal, após entendimento.
- A reprodutibilidade é alcançada em várias camadas. Desde a definição de imagens docker, passando pela orquestração de containers, até a execução de scripts para deployar serviços em ambiente distribuído. Esses mesmos passos podem ser usados para deployar serviços em ambientes de cloud, como AWS, Azure.

1. Considerações

1. O passo-a-passo a seguir foi testado em ambiente Linux, OS ubuntu 22.04, com docker instalado e configurado com engine na versão 26.1.3.
2. Esse passo a passo indica como deployar os serviços em um ambiente local, single node com docker-compose. Ao final é apresentado como executá-lo em um ambiente distribuído, multi-node com docker swarm.
3. Um fator crucial para execução desse sistema e que deve ser considerado ao se tentar reproduzi-lo, são as API Keys usadas para interagir com a rede blockchain por meio de um provedor Node-as-a-Service.

5.2. Pré-requisitos

Seguem abaixo os pré-requisitos para reprodução desse sistema em ambiente local.

5.2.1. Requisitos de hardware

Para execução desse sistema em ambiente local, é recomendado possuir memória RAM de no mínimo 16 GB e processador com 4 núcleos.

5.2.2. Docker

Como abordado em tópicos anteriores, o docker possibilita o encapsulamento de serviços em containers, que são executados em ambientes isolados e por isso foi usado na construção desse sistema local. Assim sendo, para reproduzir esse sistema em ambiente local, é necessário ter o docker instalado e configurado.

Para verificar se o docker está instalado e configurado adequadamente, execute os comandos abaixo.

```
dadaia@dadaia-desktop:~/workspace/projects/dm_structure$ docker version
Client: Docker Engine - Community
Version:      26.1.3
API version:  1.45
Go version:   go1.21.10
Git commit:   b72abbb
Built:        Thu May 16 08:33:29 2024
OS/Arch:      linux/amd64
Context:      default
```

Caso não esteja instalado, siga as instruções de instalação no site oficial do docker <https://docs.docker.com/engine/install/>.

`Docker ps`

5.2.3. Docker Compose e Docker Swarm

As ferramentas de orquestração de containers `Docker Compose` e `Docker Swarm` são necessárias para deployar os serviços em ambiente local e distribuído, respectivamente. Porém elas são instaladas junto com o docker.

Para verificar se estão instaladas adequadamente, execute os comandos abaixo.

`docker-compose`

`docker swarm`

Com as ferramentas citadas corretamente instaladas e configuradas, é possível prosseguir com o passo-a-passo para reprodução do sistema.

5.3. Automação de comandos e Makefile

Conforme mencionado acima, as ferramentas instaladas já são suficientes para executar esse sistema localmente. Porém, a sequência de comandos necessários para buildar, deployar e monitorar os serviços aqui presente é extensa.

Para simplificar esse processo foi definido um arquivo chamado **Makefile** na raiz desse projeto. Nesse arquivo estão definidos comandos que por de trás dos panos executaram comandos docker. Para saber mais sobre o makefile visite <https://www.gnu.org/software/make/manual/make.html>.

Essa ferramenta está previamente instalada em sistemas Linux. Caso não possua o make instalado e tenha dificuldades para instalá-lo, abrir o arquivo e executar os comandos docker manualmente é uma opção.

5.4. Passo-a-passo para reprodução do sistema

5.4.1. Clonagem do repositório

O primeiro passo para reprodução desse sistema é clona-lo em um diretório local. Para isso, execute o comando abaixo e em seguida navegue para o diretório do projeto.

`git clone git@github.com:marcoareliomenezes/dm_v3_chain_explorer.git`

5.4.2. Pull e Build das imagens docker

As funcionalidade de **pull** e **build** de imagens do docker estão intrinsecamente ligadas. Ao se tentar fazer o build de uma imagem, caso essa não seja encontrada localmente, ou ainda, tem dentro da definição do Dockerfile de uma imagem a instrução **FROM** aponta para uma imagem que não está localmente, o docker tenta fazer o pull da imagem do repositório remoto, no caso o docker hub.

Portanto é certo que na primeira execução do comando de build abaixo, dependendo da velocidade da conexão com a internet, o processo pode demorar um pouco.

`make build`

Para melhor compreensão das imagens buildadas, basta abrir o arquivo `Makefile` na raiz do repositório desse trabalho e verificar as instruções de build.

5.4.3. Inicialização de ambiente de Desenvolvimento

Os comandos para deploy de serviços foram divididos de acordo com as camadas mencionadas na seção 1.2. Dessa forma há maior espaço para que máquinas com recursos mais escassos possam executar o sistema em partes.

Vale ressaltar nesse caso também, que caso imagens utilizadas não estejam disponíveis localmente, o docker tentará fazer o pull das imagens do repositório remoto, podendo demorar um pouco. Para deployar os serviços em ambiente local, execute os comandos abaixo.

1. Deploy de serviços da camada Fast

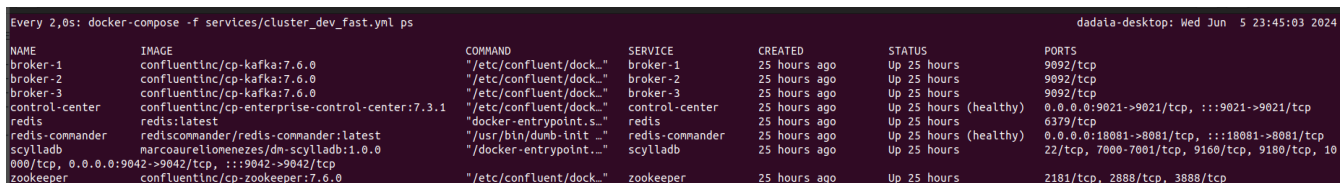
Para deployar os containers de serviços da camada fast, execute o comando abaixo.

```
make deploy_dev_fast
```

```
make watch_dev_fast
```

```
make stop_dev_fast
```

Após o deploy dos serviços da camada fast, o comando watch retorna o status dos containers, como mostrado na imagem abaixo. É possível haver conflitos de portas, caso isso ocorra, será preciso alterar as portas expostas ao localhost no arquivo `service/cluster_dev_fast.yml`. O mesmo vale para o deploy de serviços nas outras camadas.



NAME	IMAGE	COMMAND	SERVICE	CREATED	STATUS	PORTS
broker-1	confluentinc/cp-kafka:7.6.0	"/etc/confluent/dock..."	broker-1	25 hours ago	Up 25 hours	9092/tcp
broker-2	confluentinc/cp-kafka:7.6.0	"/etc/confluent/dock..."	broker-2	25 hours ago	Up 25 hours	9092/tcp
broker-3	confluentinc/cp-kafka:7.6.0	"/etc/confluent/dock..."	broker-3	25 hours ago	Up 25 hours	9092/tcp
control-center	confluentinc/cp-enterprise-control-center:7.3.1	"/etc/confluent/dock..."	control-center	25 hours ago	Up 25 hours (healthy)	0.0.0.0:9021->9021/tcp, :::9021->9021/tcp
redis	redis:latest	"docker-entrypoint.s..."	redis	25 hours ago	Up 25 hours	6379/tcp
redis-commander	rediscommander/redis-commander:latest	"/usr/bin/dumb-init -..."	redis-commander	25 hours ago	Up 25 hours (healthy)	0.0.0.0:18081->8081/tcp, :::18081->8081/tcp
scylladb	marcoaurilionnezes/dn-scylladb:1.0.0	"/docker-entrypoint..."	scylladb	25 hours ago	Up 25 hours	22/tcp, 7000-7001/tcp, 9160/tcp, 9180/tcp, 10000/tcp, 0.0.0.0:9042->9042/tcp, :::9042->9042/tcp
zookeeper	confluentinc/cp-zookeeper:7.6.0	"/etc/confluent/dock..."	zookeeper	25 hours ago	Up 25 hours	2181/tcp, 2888/tcp, 3888/tcp

Com isso, os seguintes endpoints passarão a estar disponíveis:

- **Confluent Control-Center:** <http://localhost:9021>.
- **Redis Commander:** <http://localhost:18081>.

Para visualizar os dados do **ScyllaDB**, a partir de queries, um cliente como DBeaver pode ser usado. De uma forma mais simplificada, o **cqlsh** pode ser usado para visualizar os dados.

```
docker exec -it scylladb cqlsh
```

2. Deploy de serviços da camada de Batch

Para deployar os containers de serviços da camada de aplicação, execute o comando abaixo.

```
make deploy_dev_batch
```

```
make stop_dev_batch
```

3. Deploy de serviços da camada de Aplicação

Para deployar os containers de serviços da camada de aplicação, execute o comando abaixo.

```
make deploy_dev_app
```

```
make stop_dev_app
```