Proof of Concept

Extracto DAO

Table of Contents

1.RESUMO EXECUTIVO

Extracto S.A., through its research division ExtractoDAO Labs, is a Brazilian DeepTech company dedicated to transforming frontier scientific research into proprietary technologies with commercial applications. The company develops scientific computing, quantitative technology and computational physics platforms originally created for computational cosmology, astrophysics, applied mathematics and high-performance numerical simulation. This Proof of Concept documents an important cloud-native infrastructure milestone, validating Kubernetes orchestration, Python backend services, secure API integration, GraphQL data access and NoSQL persistence capabilities that support the company's broader proprietary technology ecosystem.

For improved communication, below are the contacts of all professionals who will be involved in the PoC process:

Extracto DAO			
Joel	j.almeida@extractodao.com	(41)98792-2340	CEO
Eduardo Rodrigues		(41)999674-1007	It Head
HUAWEI CLOUD			
Zhu Wan Jing	zhu.wan.jing@huawei.com>	(11)93378-2246	Account Manager
			Technical Account Manager
Conrad Peres	Conrad.marques.peres@huawei.com	(11)992858222	Cloud Solutions Architect
Duan Bin	binro.duan@huawei.com	(11)981399251	Brazil Enterprise Cloud CTO

2.A APPLICATION

2.1 OVERVIEW

Extracto DAO Contract Query Microservice.

Conduct a proof of concept for contract querying on the blockchain in the Kubernetes cluster environment deployed in the Huawei Cloud Data Center in the São Paulo region. ExtractoDAO is structuring itself in the cloud and requires a POC for using the **CCE service, authenticating with an API on the Polygon blockchain. The digital certificate will need to be imported into the ELB for JWT authentication on the blockchain, allowing contract querying per trader, indexing on the backend, and persisting unstructured data in the MongoDB DDS. Subsequently, these data will be exposed for querying via GraphQL."**

The objectives of this environment using API services are::

1. Authentication → The Extractor Consult microservice needs to query the smart contract on the blockchain containing contract information per trader, thus it needs to authenticate via JWT on the APIIndexação e consulta de contratos →
2. Once authenticated with the filled token, the Extractor Consult microservice makes a second API call to retrieve the contracts of the trader passed as a parameter.
3. Contract Persistence → Once the Extractor Consult microservice has constructed the JSON structure that relates traders to contracts, it should persist them in the MongoDB DDS database.

3.POC OBJECTIVES

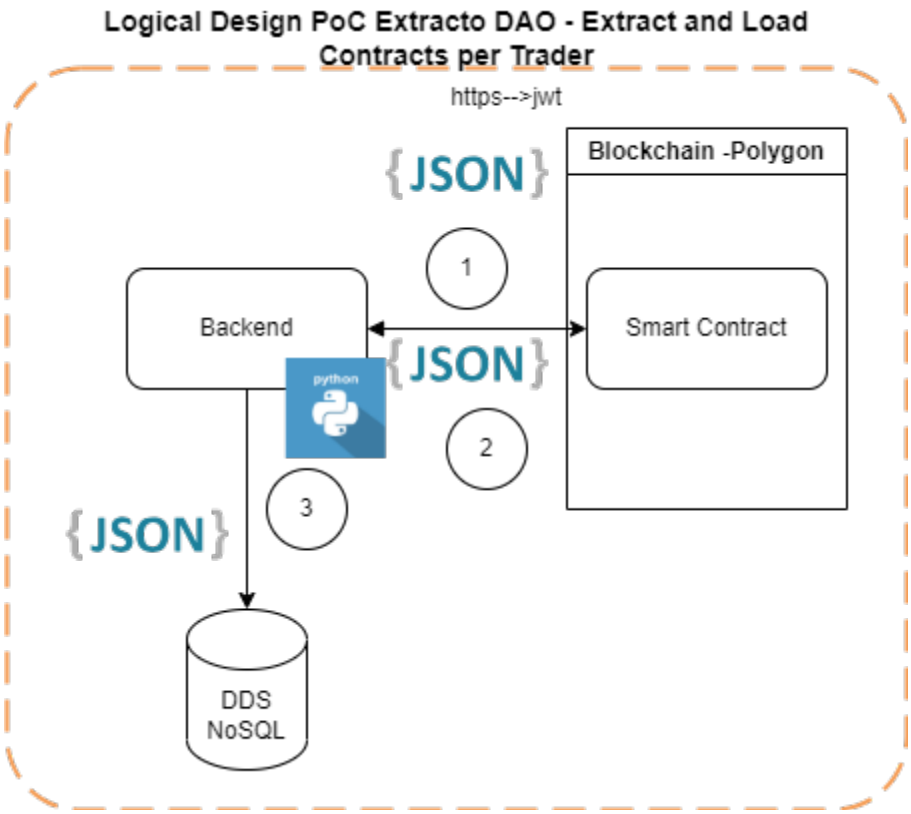
a) Perform an Application Test on the Extractor Consult microservice by calling GraphQL to retrieve the contracts for persistence. How many persisted contracts are required?

b) Performance analysis of the new architecture running on Huawei's cloud. What is the expected response time for authentication?

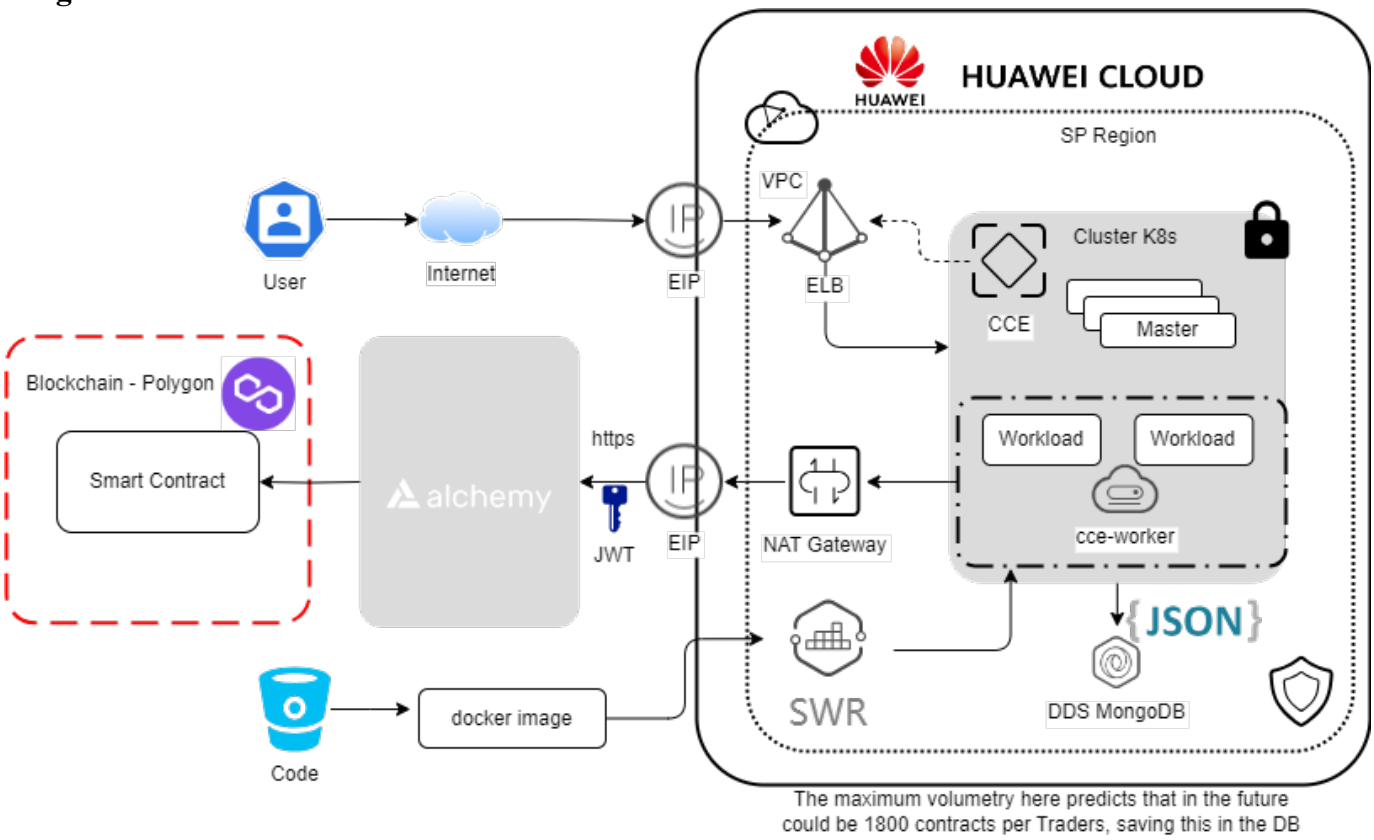
C) What is the expected response time for contract persistence?

4.ARCHITECTURE

Target Logical Architecture



Target Cloud Architecture



Sizing the Reference
We need a solution capable of storing up to 1800 contracts per trader.
Number of traders? 100 * 331,200 KB = 323.4375 MB of storage in the database
What is the amount in KB/MB per trader? 184 bytes * 1800 = 331,200 bytes = 331.200 KB
What is the amount in KB/MB per contract? 323.4375 MB of storage in the database
How many calls will the trader make to query their contracts? 100 traders corresponding to calls that will return 1000 in total, but per trader. Each trader will make only one call.
What is the endpoint of the smart contract on the Polygon blockchain? https://polygon-mumbai.g.alchemy.com/v2/lHqgHZ2VBzJlJNhiUvHi6UVY_oCsDtHM API KEY lHqgHZ2VBzJlJNhiUvHi6UVY_oCsDtHM (exemplo da conta de Alchemy de Conrad)
Can you provide the digital certificate of the secure server to be imported into our Elastic Load Balancer?

Huawei Cloud Architecture
Utilize the following technologies:
Domain Driven Design
Container Orchestration Architecture with k8s
Non-Structured Data Persistence Architecture with MongoDB.

5.ACTIVITIES

Schedule
Start Date: 19/06/23
End Date: 30/06/23

Activities to be carried out:

1. Deploy the CCE Cluster of the expected minimum size.
    a. Upload the source code to the Extracto DAO code repository.
    b. Buildar the code;
    c. Upload the image for SWR;
    d. Apply the image to a container to be replicated by the orchestrator.;
2. Create a MongoDB DDS Cluster of the expected minimum size.;
    a. Create a user for the application;
    b. Create a table.;
3. Create a NAT Gateway for the use of the CCE Cluster;
4. Import the digital certificate into the ELB (Elastic Load Balancer);
5. Test connectivity.
6. Test the integration as a whole;
7. Test data;
8. Stress testing;

6.EXPECTED OUTCOME

During the execution of the PoC, we intend to record:

- Authentication response times need to be between …. ;
- Query and persistence response times need to be between....;
- Communication from ELB to the NAT Gateway going to the internet to reach the Polygon blockchain API should be secure using HTTPS and authenticating using JWT tokens, encrypting the data in transit.

7. PoC Results:

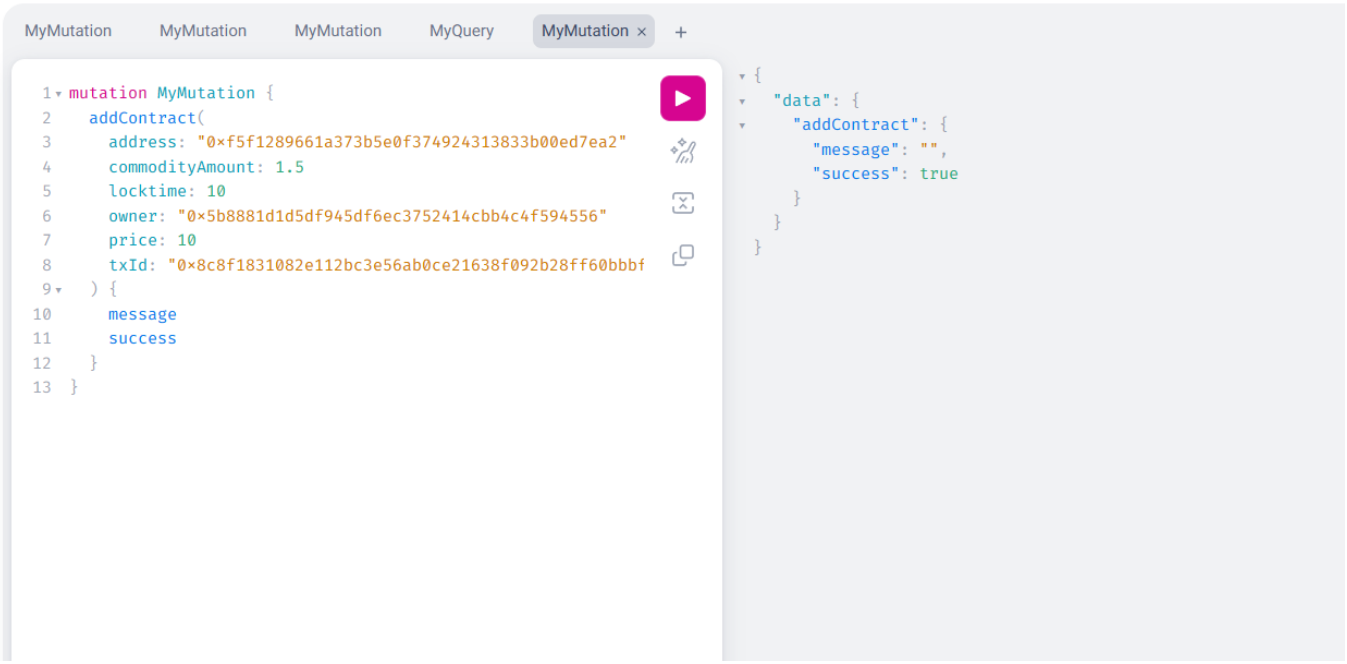
After querying the Polygon blockchain for all contracts owned by the owner: 0x5b8881d1d5df945df6ec3752414cbb4c4f594556, the GraphQL deployed on CCE was able to perform operations on the MongoDB deployed on DDS

- Querying all contracts;
- Contract creation;

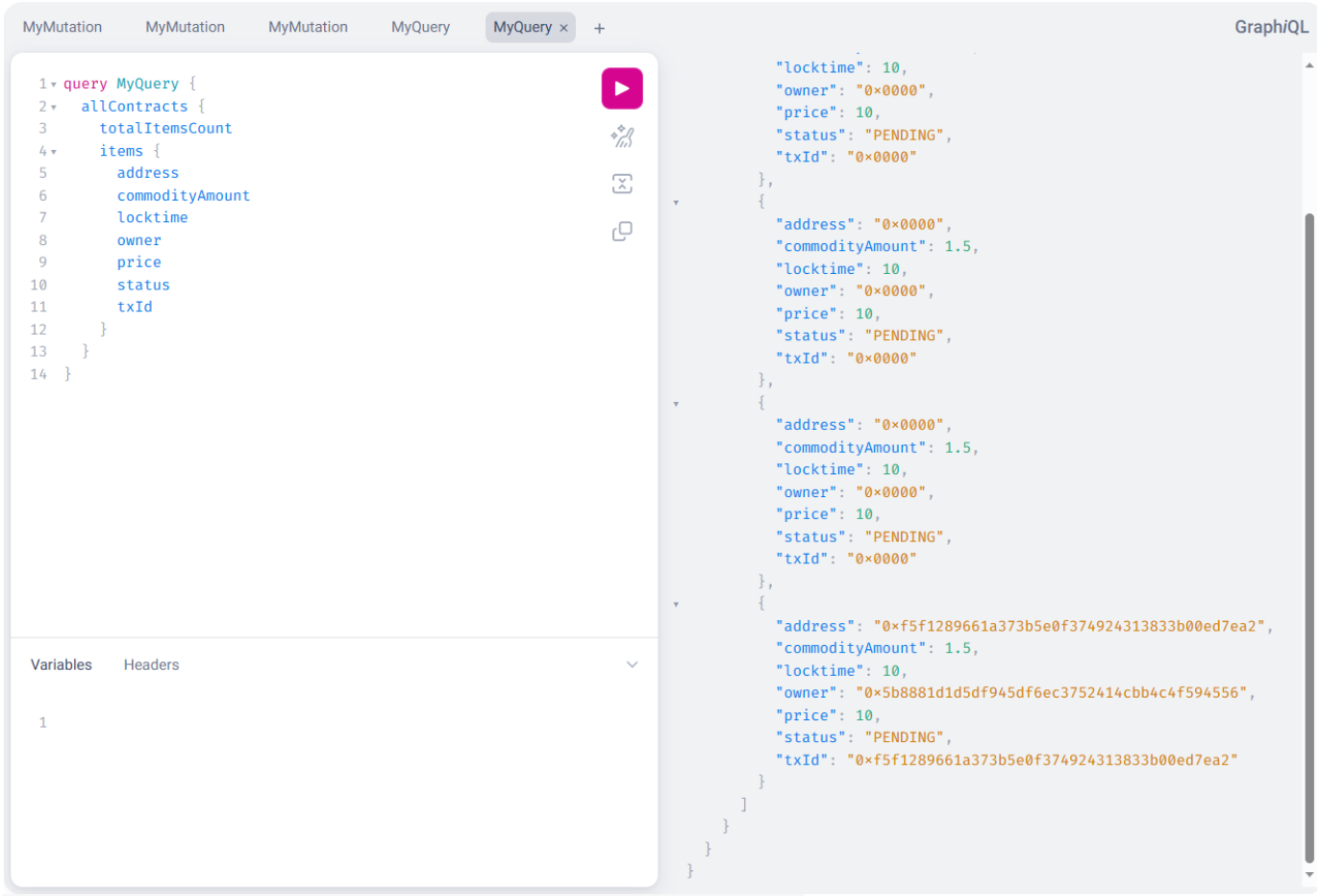
The first log provides evidence of contract creation via GraphQL Mutation in the contracts collection within MongoDB:

```
INFO:     172.18.0.1:43603 - "POST /graphql HTTP/1.1" 200 OK
Create Contract!
INFO:     172.18.0.1:43603 - "POST /graphql HTTP/1.1" 200 OK
Create Contract!
INFO:     172.18.0.1:14947 - "POST /graphql HTTP/1.1" 200 OK
Create Contract!
Connected successfully to MongoDB.
INFO:     172.18.0.1:61484 - "POST /graphql HTTP/1.1" 200 OK
INFO:     172.18.0.1:61484 - "GET /graphql HTTP/1.1" 200 OK
Connected successfully to MongoDB.
INFO:     Uvicorn running on http://0.0.0.0:80 (Press CTRL+C to quit)
INFO:     Application startup complete.
INFO:     Waiting for application startup.
INFO:     Started server process [10]
INFO:     Uvicorn running on http://0.0.0.0:80 (Press CTRL+C to quit)
INFO:     Application startup complete.
INFO:     Waiting for application startup.
INFO:     Started server process [9]
```

The signature of the Mutation that adds a new contract to the collection:



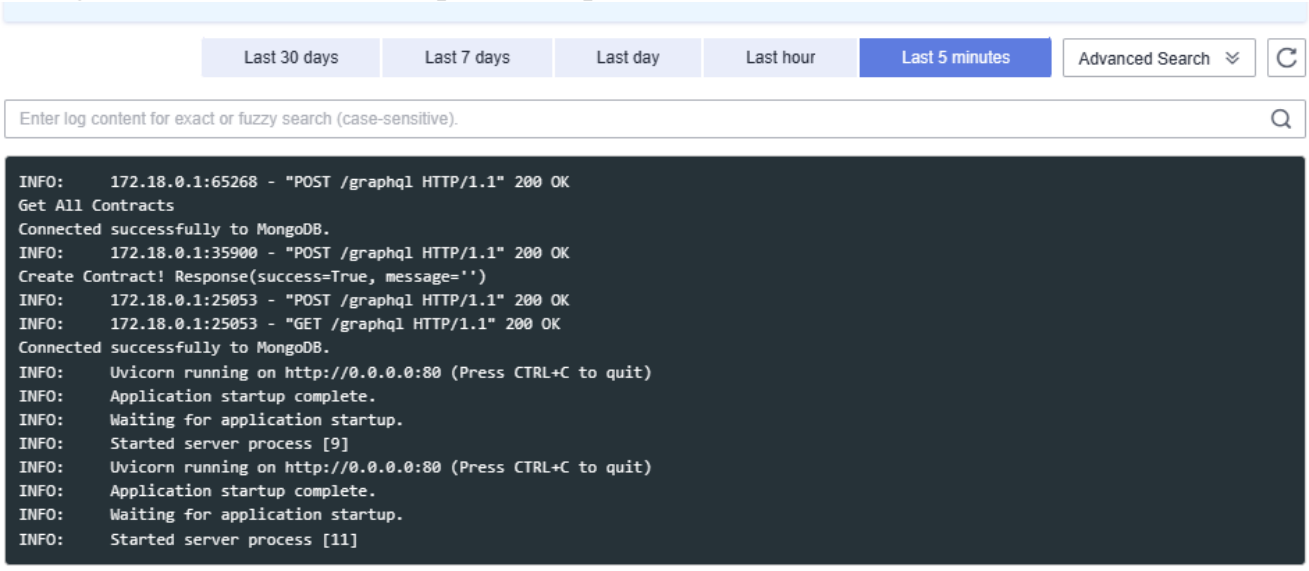
Next, the GraphQL allContracts query provides evidence of the contracts that were inserted into MongoDB:



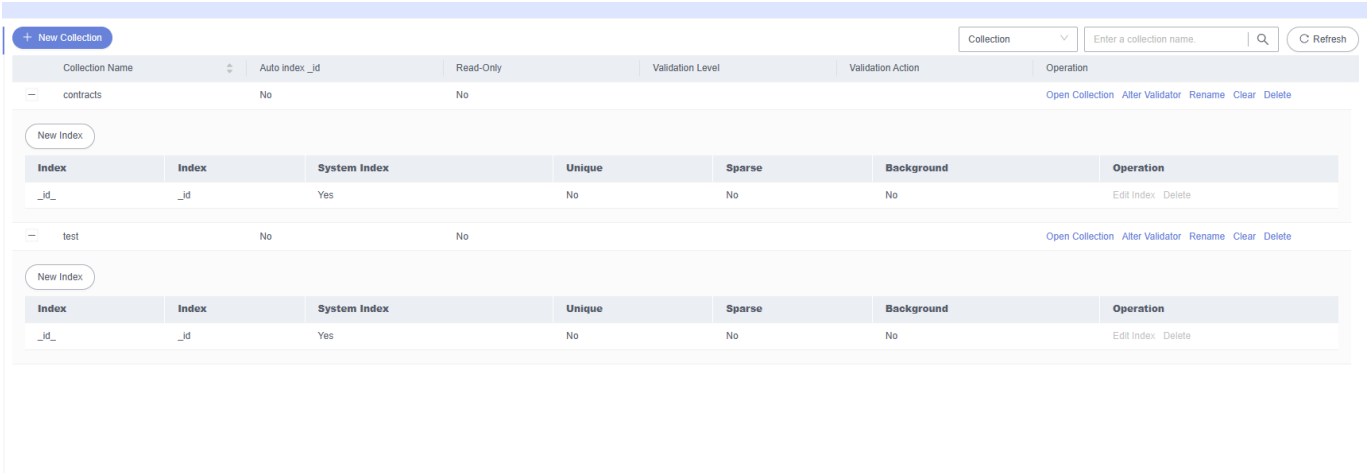
The log confirms that the operation was indeed performed.:

```
INFO:     172.18.0.1:65268 - "POST /graphql HTTP/1.1" 200 OK
Get All Contracts
Connected successfully to MongoDB.
INFO:     172.18.0.1:35900 - "POST /graphql HTTP/1.1" 200 OK
Create Contract! Response(success=True, message='')
INFO:     172.18.0.1:25053 - "POST /graphql HTTP/1.1" 200 OK
INFO:     172.18.0.1:25053 - "GET /graphql HTTP/1.1" 200 OK
Connected successfully to MongoDB.
INFO:     Uvicorn running on http://0.0.0.0:80 (Press CTRL+C to quit)
INFO:     Application startup complete.
INFO:     Waiting for application startup.
INFO:     Started server process [9]
INFO:     Uvicorn running on http://0.0.0.0:80 (Press CTRL+C to quit)
INFO:     Application startup complete.
INFO:     Waiting for application startup.
INFO:     Started server process [11]
```

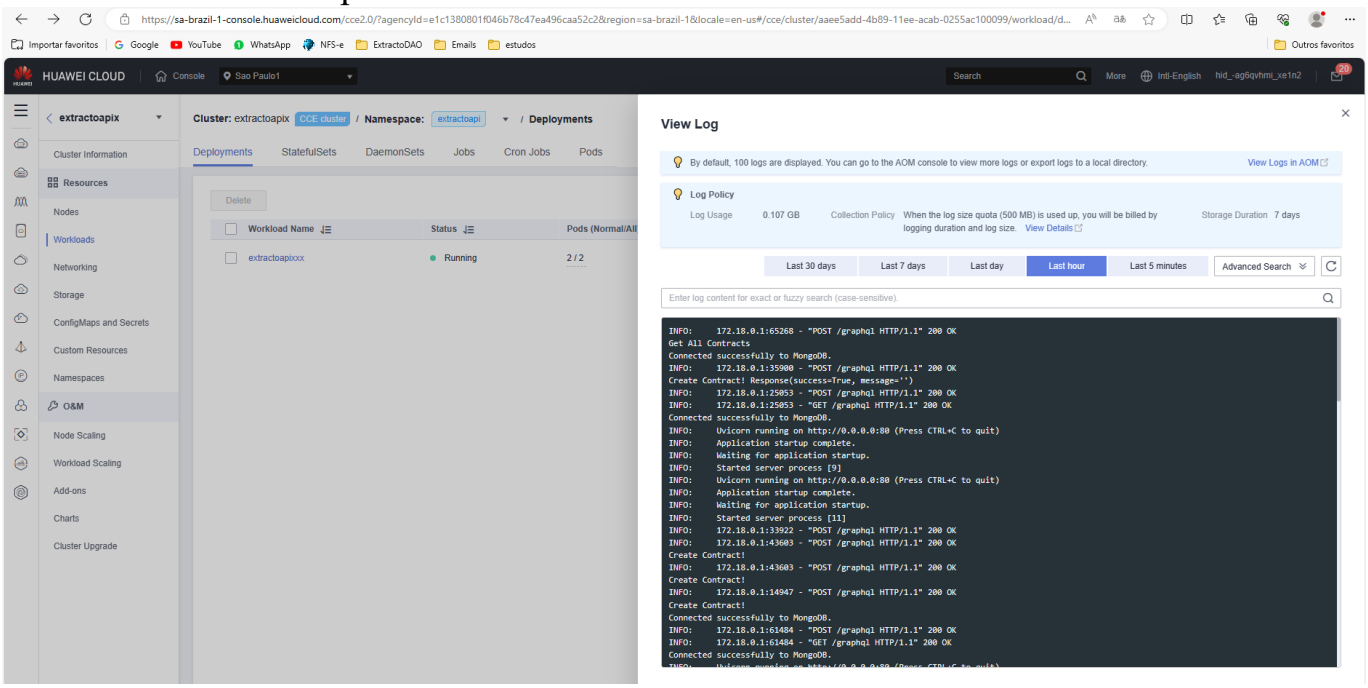
The log below shows the last operations performed in the last 5 minutes.



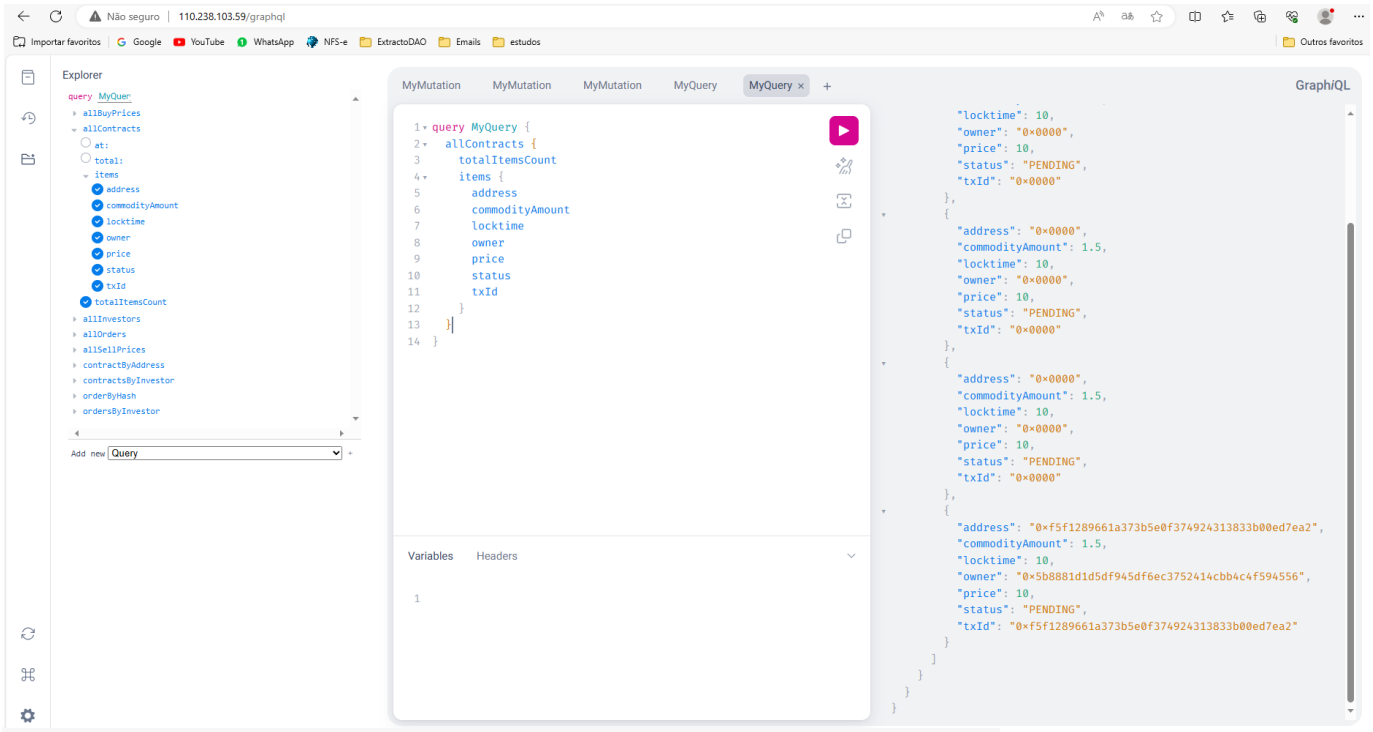
Below, the screenshot shows the smart contracts collection that was created to store contracts originating from the Polygon blockchain:



Next, the screenshot below demonstrates the Python workload created to perform query and contract creation operations from the blockchain:



This is the screenshot of the GraphQL interface of the "extractoapixxx" workload that allows you to manage the data of ExtractoDAO's agribusiness contracts on the Polygon network:



This is the log of the operation performed above, namely "allContracts:

```
INFO:     172.18.0.1:44373 - "POST /graphql HTTP/1.1" 200 OK
Get All Contracts [Contract(commodity_amount=1.5, status=0, locktime=10, address='0x0000', owner='0x0000', price=10, tx_id='0x0000'), Cont
ract(commodity_amount=1.5, status=0, locktime=10, address='0x0000', owner='0x0000', price=10, tx_id='0x0000'), Contract(commodity_amount=
1.5, status=0, locktime=10, address='0x0000', owner='0x0000', price=10, tx_id='0x0000'), Contract(commodity_amount=1.5, status=0, locktime
=10, address='0xf5f1289661a373b5e0f374924313833b00ed7ea2', owner='0x5b8881d1d5df945df6ec3752414cbb4c4f594556', price=10, tx_id='0xf5f12896
61a373b5e0f374924313833b00ed7ea2'), Contract(commodity_amount=1.5, status=0, locktime=10, address='0xf5f1289661a373b5e0f374924313833b00ed7
ea2', owner='0x5b8881d1d5df945df6ec3752414cbb4c4f594556', price=10, tx_id='0xf5f1289661a373b5e0f374924313833b00ed7ea2')]
INFO:     172.18.0.1:34010 - "GET / HTTP/1.1" 200 OK
Connected successfully to MongoDB.
INFO:     172.18.0.1:47788 - "POST /graphql HTTP/1.1" 200 OK
Create Contract! Response(success=True, message='')
INFO:     172.18.0.1:56779 - "POST /graphql HTTP/1.1" 200 OK
INFO:     172.18.0.1:56779 - "GET /graphql HTTP/1.1" 200 OK
Connected successfully to MongoDB.
INFO:     Uvicorn running on http://0.0.0.0:80 (Press CTRL+C to quit)
INFO:     Application startup complete.
INFO:     Waiting for application startup.
INFO:     Started server process [10]
INFO:     Uvicorn running on http://0.0.0.0:80 (Press CTRL+C to quit)
INFO:     Application startup complete.
INFO:     Waiting for application startup.
INFO:     Started server process [10]
INFO:     172.18.0.1:34943 - "GET /actuator/health HTTP/1.1" 404 Not Found
```

Considering this scope of the PoC, it was indeed successful..

At.te,

Conrad Marques Peres
Cloud Solutions Architect

E-mail: conrad.marques.peres@huawei.com
Mobile: +55 11 91404-1664