



HUAWEI

Proof of Concept

Extracto DAO

Table of Contents



HUAWEI CLOUD

1. RESUMO EXECUTIVO	3
2. APLICAÇÃO	4
2.1 VISÃO GERAL	Erro! Indicador não definido.
3. OBJETIVOS DA POC	4
4. ARQUITETURA	Erro! Indicador não definido.
5. Cronograma e ATIVIDADES	Erro! Indicador não definido.
6. RESULTADO ESPERADO.....	Erro! Indicador não definido.

- EXECUTIVE SUMMARY 3
- APPLICATION 4 2.1. OVERVIEW 4
- POC OBJECTIVES 4
- ARCHITECTURE 5
- SCHEDULE AND ACTIVITIES 5
- EXPECTED OUTCOME 6



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 API *Indexação 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.

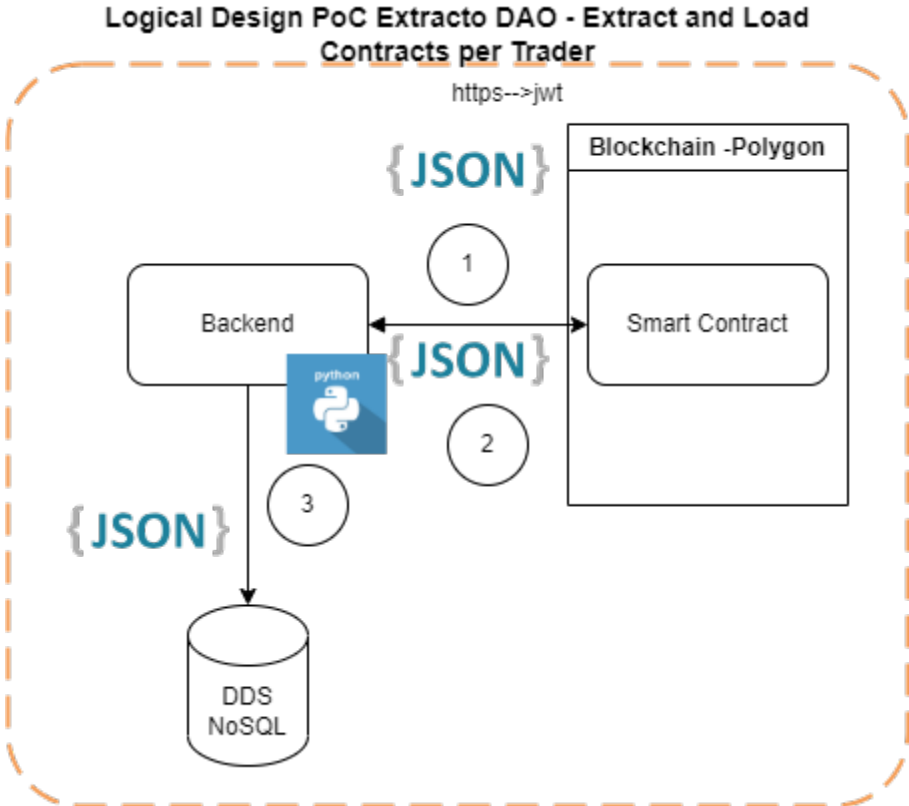
3.POC OBJECTIVES

- 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?

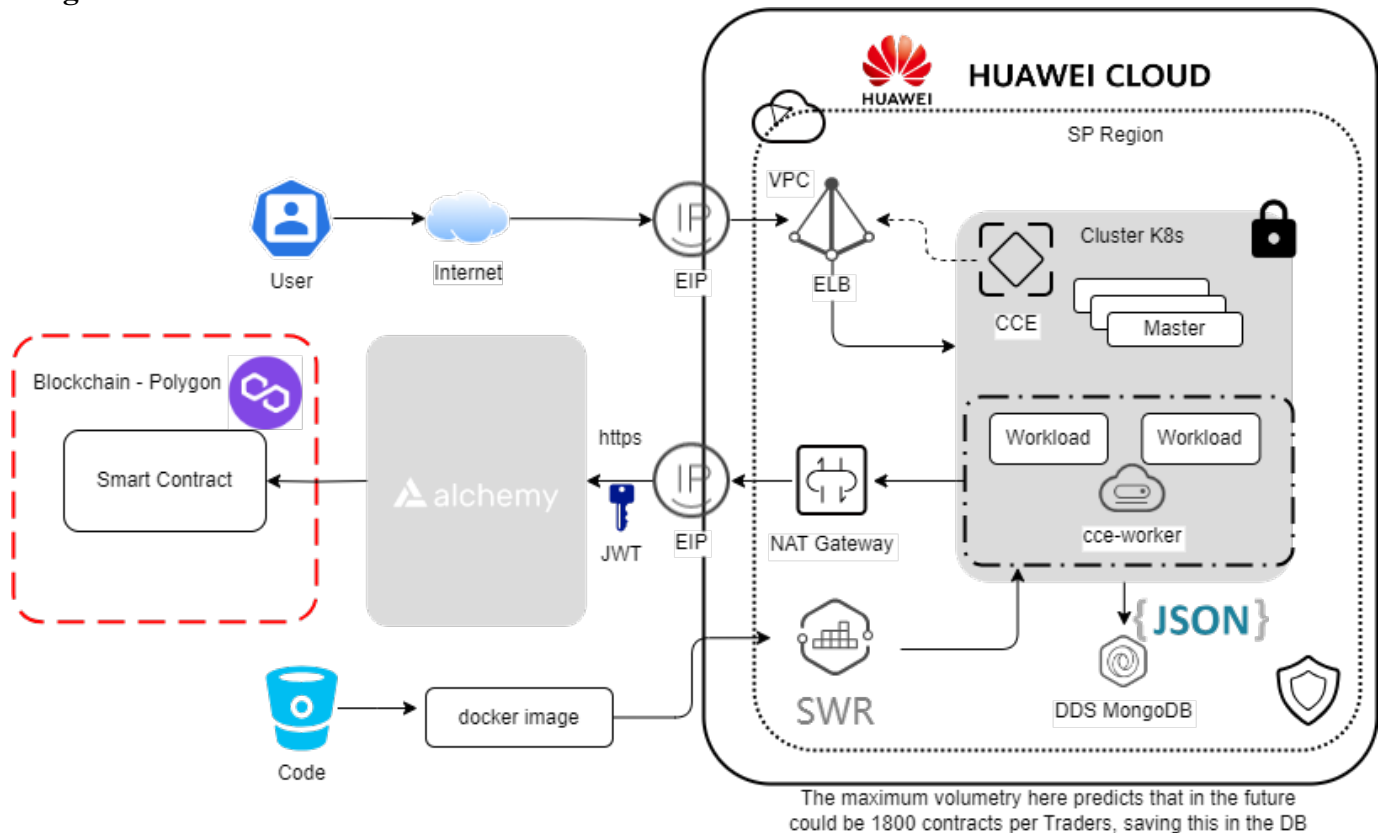


4.ARCHITECTURE

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? $100 * 331,200 \text{ KB} = 33,120,000 \text{ KB}$ of storage in the database

What is the amount in KB/MB per trader? $184 \text{ bytes} * 1800 = 331,200 \text{ bytes} = 331.200 \text{ KB}$

What is the amount in KB/MB per contract? $33,120,000 \text{ KB}$ 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/IHqgHZ2VBzJIJNhiUvHi6UVY_oCsDtHM API KEY

IHqgHZ2VBzJIJNhiUvHi6UVY_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:

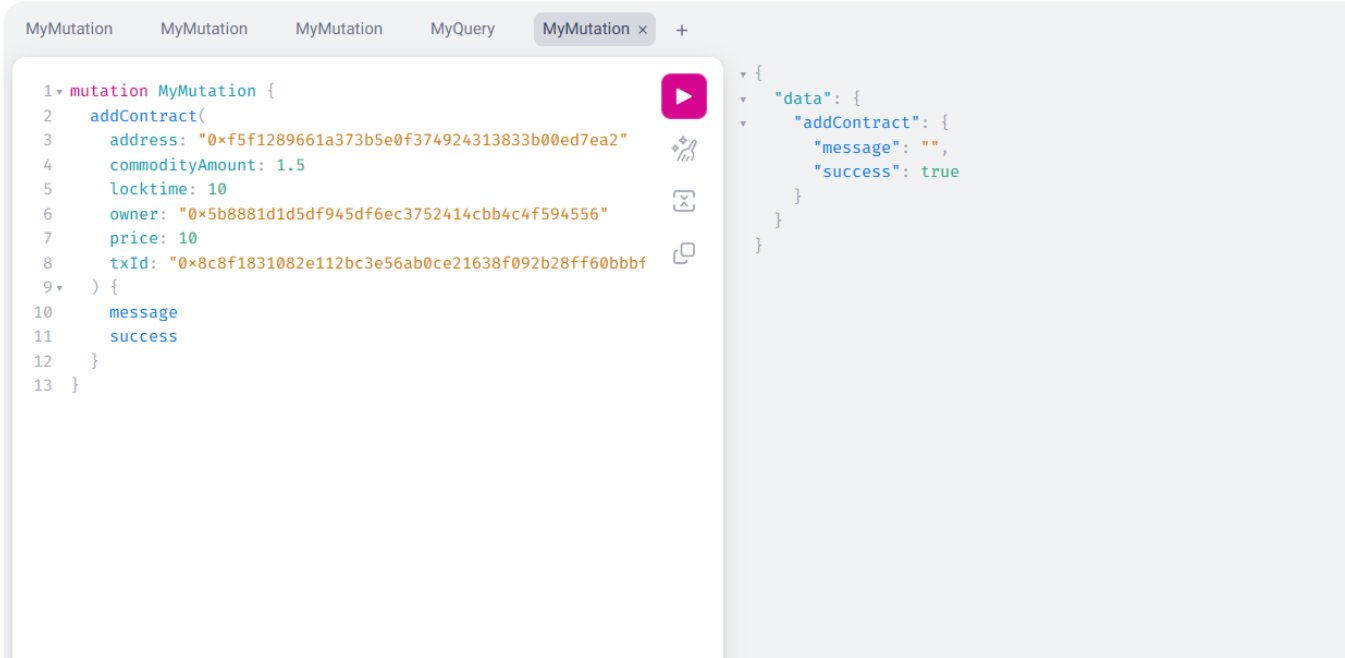
After querying the Polygon blockchain for all contracts owned by the owner: 0x5b8881d1d5df945df6ec3752414cbb4c4f594556, the GraphQL deployed on CCE was able to perform operations on the MongoDB deployed on DDS

- 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:



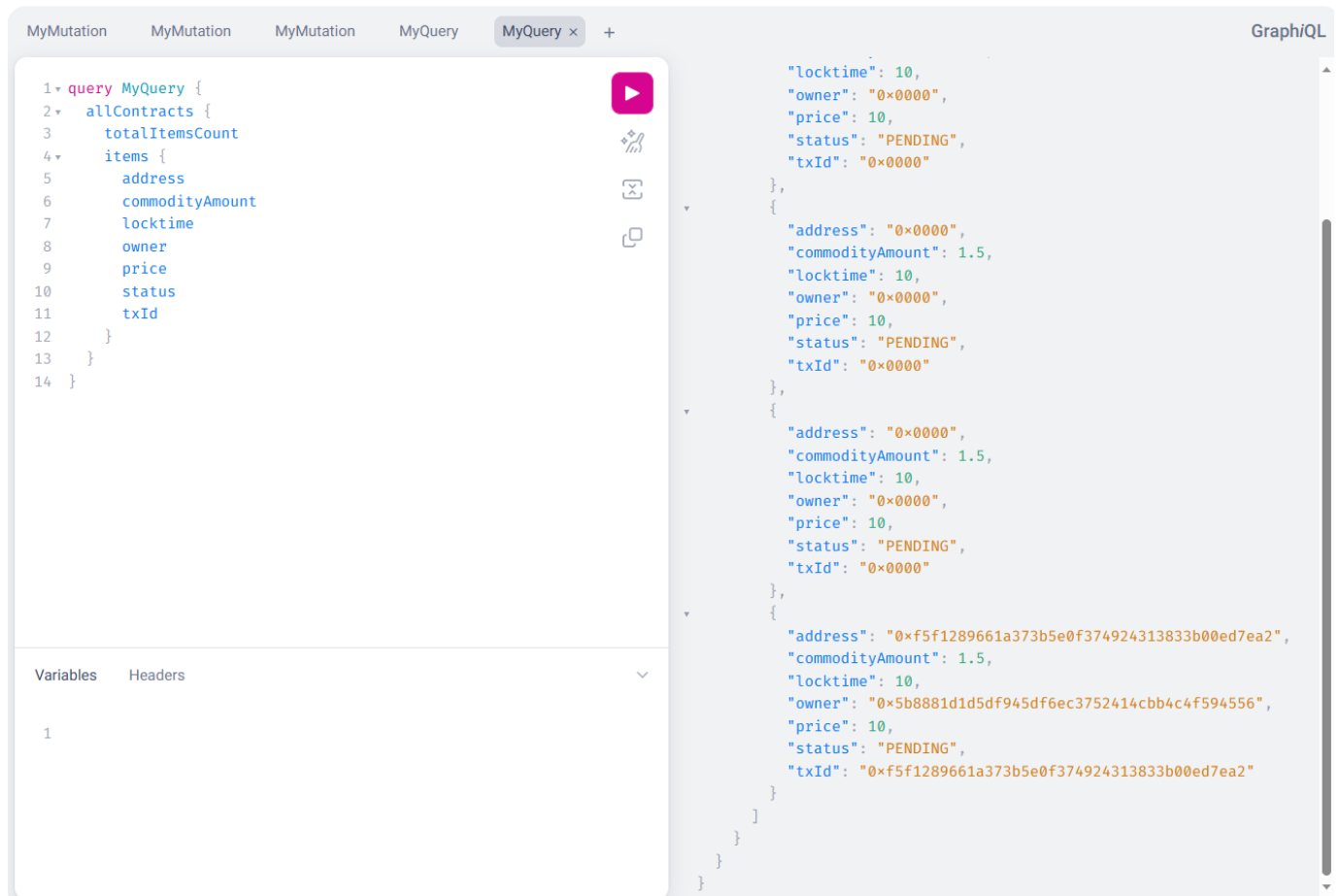
The image shows a code editor with a tab labeled 'MyMutation'. The editor contains a GraphQL mutation and its corresponding JSON response. The mutation is as follows:

```
1 mutation MyMutation {  
2   addContract(  
3     address: "0xf5f1289661a373b5e0f374924313833b00ed7ea2"  
4     commodityAmount: 1.5  
5     locktime: 10  
6     owner: "0x5b8881d1d5df945df6ec3752414cbb4c4f594556"  
7     price: 10  
8     txId: "0x8c8f1831082e112bc3e56ab0ce21638f092b28ff60bbbf"  
9   ) {  
10    message  
11    success  
12  }  
13 }
```

The JSON response is as follows:

```
{  
  "data": {  
    "addContract": {  
      "message": "",  
      "success": true  
    }  
  }  
}
```

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



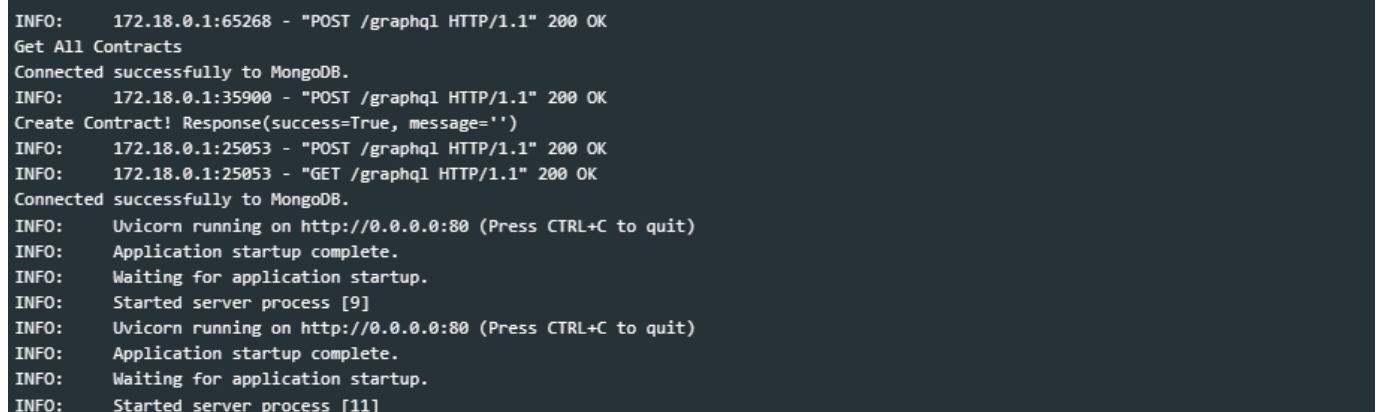
The screenshot shows a GraphQL IDE interface with a query editor on the left and a response viewer on the right. The query is as follows:

```
1 query MyQuery {
2   allContracts {
3     totalItemsCount
4     items {
5       address
6       commodityAmount
7       locktime
8       owner
9       price
10      status
11      txId
12    }
13  }
14 }
```

The response viewer shows a JSON array of contract objects:

```
{
  "locktime": 10,
  "owner": "0x0000",
  "price": 10,
  "status": "PENDING",
  "txId": "0x0000"
},
{
  "address": "0x0000",
  "commodityAmount": 1.5,
  "locktime": 10,
  "owner": "0x0000",
  "price": 10,
  "status": "PENDING",
  "txId": "0x0000"
},
{
  "address": "0x0000",
  "commodityAmount": 1.5,
  "locktime": 10,
  "owner": "0x0000",
  "price": 10,
  "status": "PENDING",
  "txId": "0x0000"
},
{
  "address": "0xf5f1289661a373b5e0f374924313833b00ed7ea2",
  "commodityAmount": 1.5,
  "locktime": 10,
  "owner": "0x5b8881d1d5df945df6ec3752414cbb4c4f594556",
  "price": 10,
  "status": "PENDING",
  "txId": "0xf5f1289661a373b5e0f374924313833b00ed7ea2"
}
]
```

The log confirms that the operation was indeed performed.:



The terminal log shows the following output:

```
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.

Log content for exact or fuzzy search (case-sensitive):

```
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]
```

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

Collection Name: contracts, Auto index _id: No, Read-Only: No, Validation Level: No, Validation Action: No, Operation: Open Collection, Alter Validator, Rename, Clear, Delete

Index	Index	System Index	Unique	Sparse	Background	Operation
id	_id	Yes	No	No	No	Edit Index, Delete

Collection Name: test, Auto index _id: No, Read-Only: No, Validation Level: No, Validation Action: No, Operation: Open Collection, Alter Validator, Rename, Clear, Delete

Index	Index	System Index	Unique	Sparse	Background	Operation
id	_id	Yes	No	No	No	Edit Index, Delete

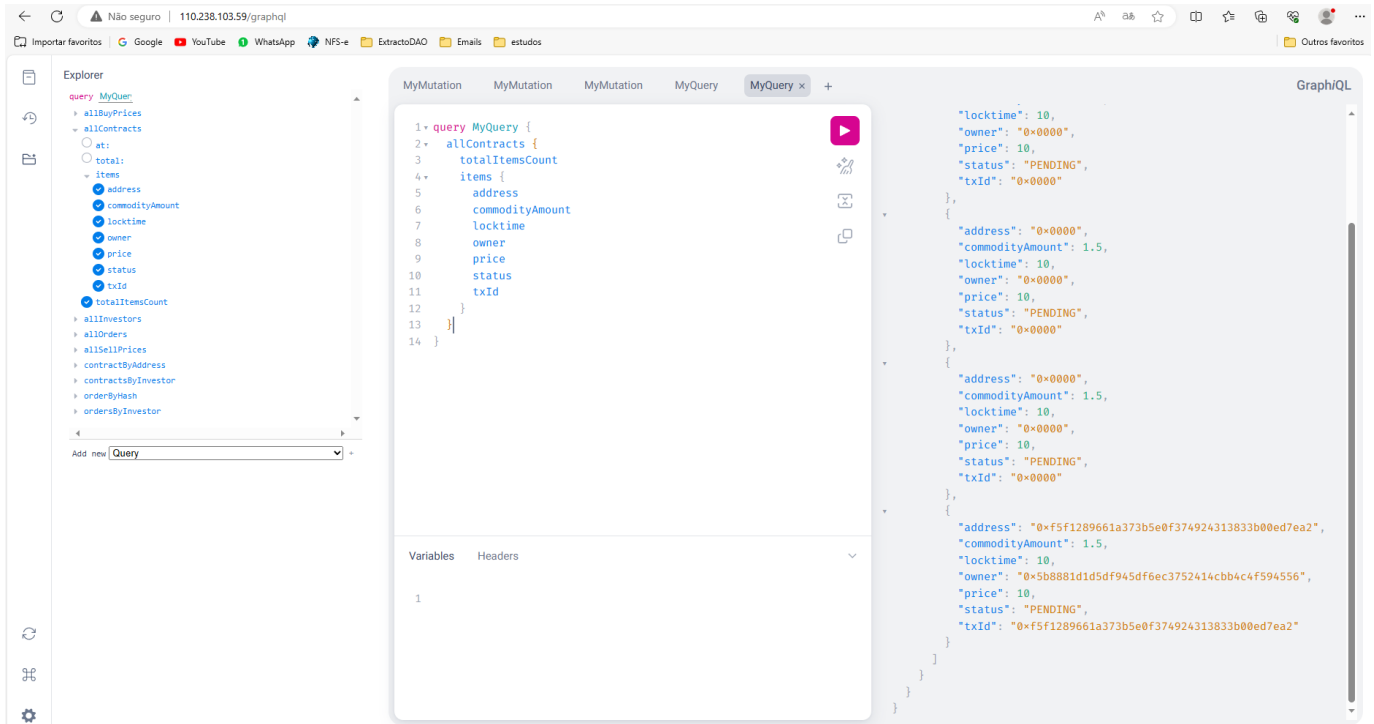
Next, the screenshot below demonstrates the Python workload created to perform query and contract creation operations from the blockchain:

The screenshot displays the Huawei Cloud console interface. On the left, a navigation menu shows 'Resources' with 'Workloads' selected. The main area shows the 'Cluster: extractoapix' and 'Namespace: extractoapi' with a 'Deployments' tab active. A table lists the deployment 'extractoapix' with a status of 'Running' and 2/2 pods.

The 'View Log' window is open, showing log entries for the deployment. The logs indicate successful connections to MongoDB and the execution of HTTP requests for contract creation and retrieval.

```
INFO: 172.18.0.1:65268 - "POST /graphql HTTP/1.1" 200 OK
Get All Contracts
Connected successfully to MongoDB.
INFO: 172.18.0.1:35980 - "POST /graphql HTTP/1.1" 200 OK
Create Contract! Response(success=True, messages=[])
INFO: 172.18.0.1:25953 - "POST /graphql HTTP/1.1" 200 OK
INFO: 172.18.0.1:25953 - "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]
INFO: 172.18.0.1:33922 - "POST /graphql HTTP/1.1" 200 OK
INFO: 172.18.0.1:43683 - "POST /graphql HTTP/1.1" 200 OK
Create Contract!
INFO: 172.18.0.1:43683 - "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.
```

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'), 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='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='0xf5f1289661a373b5e0f374924313833b00ed7ea2'), Contract(commodity_amount=1.5, status=0, locktime=10, address='0xf5f1289661a373b5e0f374924313833b00ed7ea2', 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

