# GoodData Service Entity Management

Authors: dev-goodata

## Objectives

This document is the development specification of GoodData Service Entity Management in GoodData MainNet 1.0.

## Background

There are multiple entities in the GoodData Platform. Query Customer (QC), Multi-Party Computation Node (MPC) and Data Owner (DO).  As the management service of the GoodData Platform, the GoodData Service is responsible for the life cycle of these entities.

To support the vertical training in the GoodData Platform, another entity, the Computation Node, will be added into the platform. However, to simplify the implementation, the Computation Node will use the same binary as the MPC. We will use the MPC node to denote the Computation Node below.

## Detailed Design

### Entity Profile Management

#### GDS registration (nice to have)

1. GDS need to register to the chain before start.  ( Maybe no code is needed).
2. GDS can unregister to the chain. (One Api is needed).

#### QC profile management

QC’s profile contains username, email address, password, wallet address and public key. Everything except username can be changed. Also, because QC mostly contains user private information, QC profile updates do not go through the chain.

##### Use cases

1. QC can be registered with a unique username and email address.
    a. POST /qcs/register
2. QC can login using username, password.
    a. POST /qcs/login
3. QC can logout.
    a. DELETE /qcs/logout

4. QC can change email address, wallet address and the public key after logged in. Emails will be sent to the current (or previous in case of email address change) email address to notify users with the best efforts.
    a. PUT /qcs/{uuid}
5. QC can change the password before login with 2-step verification. An email will be sent to the registered email address with a random created url (ttl is 30 minutes). QC should be able to update the password using the random url if it is not ttl'ed.
    a. POST /qcs/updatepassword
        i. Payload should include an email address.
    b. GET /qcs/updatepassword/tempurl/{random_suffix}
6. TLS should be enabled for all QC communication.
7. QC always uses REST API to talk to GDS.

### DB schema

| Table name | query_customer_info | |
|---|---|---|
| Field name | Type | Comment |
| qc_uuid | VARCHAR | ID(primary key) |
| username | VARCHAR | Should be unique. Has an index. |
| password | VARCHAR | Encrypted password |
| email_address | VARCHAR | |
| wallet_address | VARCHAR | |
| public_key | VARCHAR(64) | |
| temp_password_url | VARCHAR | Temp url used to get back password, should be ttl'ed after 30 mins. |

## DO/MPC profile management

DO/MPC's profile contains some public information: wallet address and public key, docker image version, description and some private information: ip address, port, task slot.

### Use cases

1. DO/MPC can be registered.

   a. DO/MPC registration is a process instead of a single API call. DO/MPC will first be registered on the chain with the wallet address and the public key. GDS will monitor the MPC/DO_REGISTERED event on the chain, until then the DO/MPC registration RPC call with private information (ip address and port only, task slot will be taken care individually) will be handled. Before that, the DO/MPC server needs to send a DO/MPC registration RPC call every minute until it passes through or fails with a timeout.

```
rpc RegisterDO(RegisterDORequest) returns (RegisterDOResponse);
```

2. DO/MPC can be unregistered.
   a. GDS will monitor MPC/DO_UNREGISTERED events on the chain, DO and MPC will be marked last live time to be 0.
   b. Unregister status can be revoked by re-register RPC call (**Nice to have**).
3. DO/MPC IP address, port can be changed
   a. Private information like IP address and port can be changed through API calls.

```
rpc UpdateDO(UpdateDORequest) returns (UpdateDOResponse);
```

4. DO/MPC task slots can be changed.
   a. GDS will monitor MPC/DO_STAKE_IN events on the chain and get the number of staked in tokens. Task slots = staked_in_token / num_token_each_task.
   b. Note that num_token_each_task is different for DO, MPC. The number for DO will be larger because DO's tasks will be heavier in general.
5. TLS should be enabled for all DO/MPC communication.
6. DO/MPC always uses grpc to talk to the GDS.

## DB schema

| Table name | data_owner/mpc_info | |
|---|---|---|
| Field name | Type | Comment |
| wallet_address | VARCHAR | ID(primary key) |
| public_key | VARCHAR | |
| description | JSON | |
| docker_image_version | VARCHAR | |
| client_address | VARCHAR | Ip:port or dns_name |
| last_live_time | INT64 | Need index.<br><br>If it is set as 0, nothing can change it again unless re- |

|  |  |  |
|---|---|---|
|  |  | register API call. |
| reputation | INT | Nice to have |
| MPC only |  |  |
| CPU left | INT | Need index |
| GPU left | INT | Need index |
| RAM left | INT | Need index |
| task_slot_total | INT | Need index |
| task_slot_taken | INT |  |
| task_slot_left | INT | = task_slot_total - <br><br>task_slot_taken, Need index or create a view with index. |
| running_vertical_training_jobs, running_horizontal_training_jobs, running_vertical_prediction_jobs | All INTs |  |

## DO/MPC to GDS authentication

Even though GDS and DO/MPC always use TLS RPC connection, we cannot make sure DO/MPC will not modify other entities' information and send API pretending to be other entities. That said, it is required to make sure that DO/MPC claimed in the RPC request is the DO/MPC who is sending the request.

Each RPC request should contain the following fields in the payload.

```
message AuthenticationToken {
  string signatured_token = 1;
  string salt = 2;
  string entity_key = 3;
}
```

Where signature_token = encrypt(entity_key + salt, private_key).
In the beginning of each RPC request, GDS will authorize incoming changes by doing decrypt(esignature_token, public_key) =  entity_key + salt.

## DO Dataset Management

DO is able to upload training datasets for QC to choose. The training data will be stored in DO's local database. While its metadata will be stored in GDS. All of the events need to push to the chain.

### Use cases

1. DO adds/updates dataset schema and the ask price.
    a. DO will first call GDS to update the metadata and get the dataset uuid.
    b. GDS will update the dataset in DB and return to DO.
    c. DO then upload the actual data. DO is responsible to fully upload the data by itself. GDS will start to show updated data as soon as the metadata is updated.
    ```
    rpc UpdateDataset(UpdateDatasetRequest) returns
    (UpdateDatasetResponse);
    ```

**Why does metadata need to be updated first?** Because GDS needs to change the schema page as soon as there is an intention to change the schema. Stale information will increase the chance for the QC to place invalid orders.

2. DO can remove a dataset.
    a. DO will first call GDS to update the metadata and then remove data locally.
    ```
    rpc RemoveDataset(RemoveDatasetRequest) returns
    (RemoveDatasetResponse);
    ```
3. QC should be able to search by some fields in the dataset.
    a. GET /datasets/search?fields=...
4. QC should be able to search by do_wallet_address
    a. GET /datasets/search?do=...
5. QC should be able to search by keywords/hashtags
    a. GET /datasets/search?tags=...

### DB schema

| Table name | dataset_schema_info | |
|---|---|---|
| Field name | Type | Comment |
| dataset_uuid | VARCHAR | ID(primary key) |
| do_wallet_address | VARCHAR | Need to have index if use |

|  |  |  |
|---|---|---|
|  |  | case 4 is needed |
| dataset_size | INT64 |  |
| description | VARCHAR |  |
| ask_price | INT256 |  |
| schema_info | JSON | Needed if use case 3 is not needed. |

Dataset_schema_details table is needed for use case 3.

| Table name | dataset_schema_details |  |
|---|---|---|
| Field name | Type | Comment |
| dataset_field_name | VARCHAR | Need to have index |
| dataset_field_type | VARCHAR | Need to have index |
| dataset_uuid | VARCHAR |  |

The Primary key contains all fields.

Dataset_tags table is needed for use case 5.

| Table name | dataset_schema_details |  |
|---|---|---|
| Field name | Type | Comment |
| dataset_tag | VARCHAR | Need to have index |
| dataset_uuid | VARCHAR |  |

The Primary key contains all fields.

## Node Management

### Heartbeat

DO/MPC need to send the heartbeat request to the chain every x minutes. GDS will monitor ENTITY_HEARTBEAT events and update last_live_time in DB.

Have a thread to check if last_live_time > timeout, then claim the entity is dead to send to the chain to update the status. Chain can trigger the timeout process to do the refund. Service will failover all running jobs and asynchronously get node unregister signal (See detailed design in query management spec).

Heartbeat will also include and record CPU usage, GPU usage, RAM usage.

### Remote docker image update (TBD)

## Test/Rollout Plan

All of the use cases need to have unit tests. And use local DB for testing.
DO/MPC, schema life cycle integration test is needed.