ecosteer.

DOOF Components installation and configuration

1 Components configuration 2
2 Database Tables Installation 2
2.1 Authentication table (for dex_change_password) 3
3 Smart contract compilation and installation 3
3.1 Solidity compiler 3
3.2 Alastria Deployment 4
4 Configuration and Running of Components 4
4.1 NGINX Configuration 4
4.2 Gateway 6
4.2.1 For plain HTTP (default: behind NGINX) 6
4.2.2 For HTTPS (without NGINX) 7
4.3 Worker 7
4.4 Monitor 10
4.5 Python Client 11
4.5.1 Default configuration: connect via HTTP to gateway behind NGINX 12
4.5.2 Alternative configuration 12
5 Python client APIs usage 12
5.1.1 Start a session (default when using provided NGINX configuration) 13
5.1.2 Enable identities (set up accounts) 13
5.1.3 Use of variables 14
5.1.4 Account information 14
5.1.5 Product creation 14
5.1.6 Product information 14
5.1.7 Purpose of usage creation and information 15
5.1.8 Subscriptions list for product 15
5.1.9 Subscribe 15
5.1.10 Subscription info 15
5.1.11 Grant and revoke 16
5.1.12 Product configuration file 16
5.1.13 Subscription configuration file 16

5.1.14 Unsubscribe 17
5.1.15 Operations related to advertisements 17
5.1.16 Close connection 17
5.1.17 Login (JWT authentication) 17
6 Pilot: front-end deployment and usage 17
7 DOOF Platform usage 18

1 Components configuration

Example configuration files for the components are provided in the repository under the specific component's folder. Copy them to a folder outside the repository to change them and use other values without interfering with the files that git is tracking.

```
cd ~/
mkdir conf && cd conf
mkdir worker
mkdir monitor
mkdir gateway
mkdir clients
```

Copy the configuration files to these folders

```
cp ~/NGI-TRUSTCHAIN/DOOF/components/worker/worker_config.template
/home/ecosteer/conf/worker
```

2 Database Tables Installation

The worker requires a specific database model. Initialize it with the following operations:

```
cd ~/NGI-TRUSTCHAIN/DOOF/installation/database
./postgres_init.sh ecosteer ecosteer ecosteer
./create_tables.sh ecosteer ecosteer ecosteer
```

Check if tables were correctly installed

```
sudo -u ecosteer psql
\c
\l
\q
```

Create account for admin:

```
cd ~/NGI-TRUSTCHAIN/DOOF/installation/database
./dep_admin_init.sh ecosteer ecosteer admin@ecosteer.com
```

2.1 Authentication table (for dex_change_password)

Please note that authentication service is not part of the DOOF operations. The change password processor requires a database having a table named ext_account_repo for the operation of changing the password. This simulates the integration of DOOF with an external system of the client, which already has everything in place for the authentication service. You can easily set up a basic account table as in the following:

```
sudo -u ecosteer psql
\c ecosteer
CREATE TABLE IF NOT EXISTS ext_eccount_repo (
username TEXT PRIMARY KEY,
password TEXT,
is_admin BOOLEAN);
```

Before enabling a user on the DOOF, in case an external authentication service is used, or if the change of password functionality needs to be available, it must first of all be inserted in the ext_account_repo. For the pilot, the users were first of all inserted manually inside this table, with a pre-computed password hash, and then enabled on the DOOF platform via the client's admin APIs.

In order to insert a user, choose a password and compute its sha256 checksum. You can use sha256sum for this purpose. Pay attention to any whitespace leading or trailing characters.

e.g.

```
echo -n "password" | sha256sum
```

You will see this output (copy the hexadecimal string without spaces nor dash):

```
5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8  -
```

The following steps can be used to create a new user::

```
sudo -u ecosteer psql -d ecosteer
```

Then execute the following query with the username and password for the account you have chosen (for users that are not administrators, set the id_admin flag to False):

```
INSERT INTO ext_account_repo(username, password, is_admin) VALUES
('admin@ecosteer.com',
'5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8' ,True);
```

3 Smart contract compilation and installation

The smart contract was written in Solidity and is compatible with Hyperledger Besu and Alastria Network.

3.1 Solidity compiler

The smart contract can be compiled with solc:

```
cd ~ && wget https://github.com/ethereum/solidity/releases/download/v0.5.0/solc-static-linux
sudo chmod +x solc-static-linux
```

```
cd ~/NGI-TRUSTCHAIN/DOOF/components/smart_contract
mkdir -p available
~/solc-static-linux -o target --overwrite --optimize --abi --bin Doof.sol
```

This should result in two files in the target directory, Doof.abi and Doof.bin. The worker component needs the path to the abi file.

3.2 Alastria Deployment

You can deploy the smart contract according to the specific deployment procedure of the selected blockchain infrastructure.

For Alastria, follow the given documentation: https://github.com/alastria/smart-contract-deployment . When deploying to Alastria, take note of the endpoint e.g.

https://red-b.alastria.io/v0/API-KEY, to be used in the configuration of the monitor and worker component, and of the blockchain address of the smart contract to be configured in the worker component.

4 Configuration and Running of Components

First of all, create the logs directory in the home directory of your user:

```
cd
mkdir logs
```

4.1 NGINX Configuration

The deployment of the Pilot requires a specific NGINX configuration, which is included in the repository. Please note that authentication endpoint (login-handler) is present but always returns 200. In a production scenario, an authentication server based e.g. on JWT for authentication can be used. The authentication server implementation and deployment is out of the scope of the current implementation and is to be deployed by any third party that uses DOOF.

```
cd ~/NGI-TRUSTCHAIN/DOOF/installation/nginx
sudo cp ecosteer.server.conf /etc/nginx/sites-available/
```

The default configuration file:

- Configures Nginx as an HTTP server.
- Indicates the location of front-end HTML files and the files themselves.

Create a symbolic link to use the ecosteer.server.conf file (the second and third lines are one command and have to be entered in the same command):

```
cd /etc/nginx/sites-available
sudo ln -s /etc/nginx/sites-available/ecosteer.server.conf /etc/nginx/sites-enabled/ecosteer.server.conf
```

Go to sites-enabled folder in the Nginx folder and remove the symbolic link with the default configuration file:

```
cd /etc/nginx/sites-enabled
sudo unlink /etc/nginx/sites-enabled/default
```

Head to nginx directory:

```
cd /etc/nginx
```

A good practice is to copy the default configuration file just in case:

```
sudo mv nginx.conf nginx_copy
```

Go to the nginx folder in the installation folder and copy the nginx.conf file from there:

```
cd ~/NGI-TRUSTCHAIN/DOOF/installation/nginx
sudo cp nginx.conf /etc/nginx/
```

Make sure that you copied the script correctly by running this command, which will check the syntax of your configuration file:

```
sudo nginx -t
```

Reload Nginx to make sure that the permissions have been applied correctly:

```
sudo systemctl reload nginx
```

Restart Nginx and check its status:

```
sudo systemctl restart nginx
sudo systemctl status nginx
```

Make sure that Nginx ports are open and that the default port 80 is closed by running these commands one by one:

```
nmap 127.0.0.1 -p 80
```

Please note, that configuration of NGINX to use https, will require an additional entry at the beginning of ecosteer.server.conf file:

```
listen 443 ssl;
server_name  yourdomain.org ;

# RSA certificate
ssl_certificate path/to/your/certs/fullchain.pem;
ssl_certificate_key path/to/your/certs/privkey.pem;
```

4.2 Gateway

The run file for the gateway starts the component on port 2783. You can change it manually in doof_gateway_run.sh script. If you change this port, please also change the ecosteer.server.conf configuration of section 3.1.

In NGI-TRUSTCHAIN/DOOF repo there is a conf/certs/ folder, containing a self-signed certificate cert.pem that can be used for the configuration of the Gateway to deploy in an HTTPS scenario. The certificate was created with openssl by giving the following contents when requested: IT – Bolzano – Bolzano - Ecosteer – Dev, with common name doof.

For the component's configuration file you need to set the following macros:

MESSAGEQ_HOST – the host on which your message queue is running.

MESSAGEQ_PORT – the port which your message queue is listening to.

MQUSER – the message queue user.

MQPASSWORD – the message queue password.

Q_NAME - the name of the queue on which the output provider should publish imperatives (this needs to be the same on which the worker is listening to incoming imperatives).

Copy the configuration file template and name your new file in a way that corresponds to your configuration:

```
cp ~/NGI-TRUSTCHAIN/DOOF/components/gateway/gateway_config.template
~/conf/gateway/gateway.conf

cd ~/conf/gateway
```

Adjust this command with the correct host and port values and execute it:

```
cd ~/conf/gateway
sed -i 's/MESSAGEQ_HOST/127.0.0.1/g' gateway.conf && sed -i 's/MESSAGEQ_PORT/5672/g'
gateway.conf

sed -i 's/MQUSER/ecosteer/g' gateway.conf && sed -i 's/MQPASSWORD/ecosteer/g' gateway.conf
```

Set the correct queue name for this deployment:

```
sed -i 's/Q_NAME/imperatives/g' gateway.conf
```

4.2.1 For plain HTTP (default: behind NGINX)

Replace the template with your new configuration file in the execution file of the component:

```
cp ~/NGI-TRUSTCHAIN/DOOF/components/gateway/doof_gateway_run.sh ~/conf/gateway
cd ~/conf/gateway
```

```
sed -i 's/gateway_config.template/gateway.conf/g' doof_gateway_run.sh
```

Run:

```
./doof_gateway_run.sh
```

You can also use supervisor to start the component (ensure logs directory exists in home dir):

```
sudo cp ~/NGI-TRUSTCHAIN/DOOF/components/gateway/sup.gateway.conf /etc/supervisor/conf.d
sudo supervisorctl
reread
add http_gateway
```

You can monitor the output of the component in the logs directory:

```
cd ~/logs
sudo tail -f  http_gateway.stderr.log
```

4.2.2 For HTTPS (without NGINX)

```
cp -r ~/NGI-TRUSTCHAIN/DOOF/conf/certs ~/conf/
```

The run rile for HTTPS starts the component on port 5443 and uses the self-signed certificate that is in the conf/certs directory.

Replace the template with your new configuration file in the execution file of the component:

```
cp ~/NGI-TRUSTCHAIN/DOOF/components/gateway/s_doof_gateway_run.sh ~/conf/gateway
cd ~/conf/gateway

sed -i 's/gateway_config.template/gateway.conf/g' s_doof_gateway_run.sh
```

Run:

```
./s_doof_gateway_run.sh
```

Or you can use supervisor:

```
sudo cp ~/NGI-TRUSTCHAIN/DOOF/components/gateway/sup.s_gateway.conf /etc/supervisor/conf.d
sudo supervisorctl
reread
add secure_gateway
```

You can monitor the output of the component in the logs directory:

```
cd ~/logs
sudo tail -f  secure_gateway.stderr.log
```

4.3 Worker

The worker's configuration file has the following macros:

DATABASE_HOST – the host on which your database is running.

DATABASE_PORT – the port which your database is listening to.

DATABASE_NAME - the name of the database that this worker should use.

DATABASE_USER - the username of the database that this worker should use.

MESSAGEQ_HOST – the host on which your message queue is running.

MESSAGEQ_PORT – the port which your message queue is listening to.

MQUSER – the message queue user.

MQPASSWORD – the message queue password.

Q_NAME - the name of the queue on which the input provider receives imperatives

BROKER_HOST – the host on which your broker is running.

BROKER_PORT – the port which your broker is listening to.

PROC_PROXY_ADDR – the proxy address that needs to be returned in the publisher and subscription configuration files by the worker

PROC_PROXY_PORT - the proxy port that needs to be returned in the publisher and subscription configuration files by the worker

BLOCKCHAIN_HOST – the IP or domain name of the host to which the worker connects to interact with the smart contract

BLOCKCHAIN_PORT – the port on the blockchain host to which the worker can connect to for interacting with smart contract

BLOCKCHAIN_CONTR_ADDR - this is found under folder 'available' after deployment, under ~/NGI-TRUSTCHAIN/DOOF/components/smart_contract

BLOCKCHAIN_OWNER_ADDR – the address of the marketplace owner, i.e. the account that will sign the transactions

BLOCKCHAIN_OWNER_PWD - the blockchain password of the marketplace owner, to be used to start and sign the transactions

Prerequisites: smart contract abi file must be present (compilation done)

```
cp ~/NGI-TRUSTCHAIN/DOOF/components/worker/worker_config.template
~/conf/worker/worker_config.json
cd ~/conf/worker
```

Adjust this command with the correct host, port, name, and user values:

```
sed -i 's/DATABASE_HOST/127.0.0.1/g' worker_config.json && sed -i 's/DATABASE_PORT/5432/g'
worker_config.json

sed -i 's/DATABASE_NAME/ecosteer/g' worker_config.json && sed -i 's/DATABASE_USER/ecosteer/g'
worker_config.json

sed -i 's/MESSAGEQ_HOST/127.0.0.1/g' worker_config.json && sed -i 's/MESSAGEQ_PORT/5672/g'
worker_config.json

sed -i 's/MQUSER/ecosteer/g' worker_config.json && sed -i 's/MQPASSWORD/ecosteer/g'
worker_config.json

sed -i 's/Q_NAME/imperatives/g' worker_config.json
```

```
sed -i 's/BROKER_HOST/127.0.0.1/g' worker_config.json && sed -i 's/BROKER_PORT/1883/g' worker_config.json
```

Adjust this command with the correct values for blockchain connection. You need to open the file and manually substitute 'http://BLOCKCHAIN_HOST:BLOCKCHAIN_PORT' with the complete endpoint API where you have deployed the smart contract. Use the smart contract address you have received from the deployment procedure, and the blockchain's owner address and password.

For a local EVM:

```
cd ~/conf/worker
sed -i 's/BLOCKCHAIN_HOST/127.0.0.1/g' worker_config.json && sed -i 's/BLOCKCHAIN_PORT/8545/g' worker_config.json

sed -i 's/BLOCKCHAIN_CONTR_ADDR/0x46D03Fde4178E54795322E3Ce7C856aD861F6D77/g' worker_config.json

sed -i 's/BLOCKCHAIN_OWNER_ADDR/0x3d5325e7c41cde9dbc3946405af30d9b64967197/g' worker_config.json && sed -i 's/BLOCKCHAIN_OWNER_PWD/ecosteer/g' worker_config.json
```

Configure the processor for dex_change_password:

```
sed -i 's/HOST_ACCOUNT_DB/127.0.0.1/g' worker_config.json && sed -i 's/PORT_ACCOUNT_DB/5432/g' worker_config.json

sed -i 's/ACCOUNT_DB_NAME/ecosteer/g' worker_config.json && sed -i 's/ACCOUNT_DB_USER/ecosteer/g' worker_config.json
```

Optional: configure publisher and subscriber configuration processors. If the processors are not configured, then the macros will be maintained and returned in the notification of the processor.

The following code snippet assumes the proxy runs on localhost.

```
cd ~/conf/worker

sed -i 's/PROC_PROXY_ADDR/127.0.0.1/g' worker_config.json && sed -i 's/PROC_PROXY_PORT/3000/g' worker_config.json
```

Replace the template with your new configuration file in the execution file of the component:

```
cp ~/NGI-TRUSTCHAIN/DOOF/components/worker/worker_run.sh ~/conf/worker
cd ~/conf/worker
sed -i 's/worker_config.template/worker_config.json/g' worker_run.sh
```

Run:

```
./worker_run.sh
```

Supervisor (ensure logs directory exists in home dir)

```
sudo cp ~/NGI-TRUSTCHAIN/DOOF/components/worker/conf/sup.worker.conf /etc/supervisor/conf.d/
sudo supervisorctl
reread
add doof_worker
```

You can monitor the output of the program in the logs directory:

```
cd ~/logs
sudo tail -f worker.stdout.log
```

4.4 Monitor

The monitor's configuration file has the following macros that need to be configured:

BLOCKCHAIN_HOST – the host on which your blockchain node is running.

BLOCKCHAIN_PORT – the port which your blockchain node is listening to.

MESSAGEQ_HOST – the host on which your message queue is running.

MESSAGEQ_PORT – the port which your message queue is listening to.

MQUSER – the message queue user.

MQPASSWORD – the message queue password.

Q_NAME - the name of the queue on which the output provider should publish imperatives (the same on which the worker is listening to incoming imperatives).

Copy the configuration file template file and name your new file in a way that corresponds to your configuration:

```
cp ~/NGI-TRUSTCHAIN/DOOF/components/monitor/monitor_config.template ~/conf/monitor/monitor_local.conf

cd ~/conf/monitor
```

Adjust this command with the correct host and port values and execute it:

```
sed -i 's/MESSAGEQ_HOST/127.0.0.1/g' monitor_local.conf && sed -i 's/MESSAGEQ_PORT/5672/g' monitor_local.conf

sed -i 's/MQUSER/ecosteer/g' monitor_local.conf && sed -i 's/MQPASSWORD/ecosteer/g' monitor_local.conf
sed -i 's/Q_NAME/imperatives/g' monitor_local.conf

sed -i 's/BLOCKCHAIN_HOST/34.88.166.119/g' monitor_local.conf && sed -i 's/BLOCKCHAIN_PORT/8545/g' monitor_local.conf
```

Replace the template with your new configuration file in the execution file of the component:

```
cp ~/NGI-TRUSTCHAIN/DOOF/components/monitor/monitor_run.sh ~/conf/monitor
cd ~/conf/monitor
sed -i 's/monitor_config.template/monitor_local.conf/g' monitor_run.sh
```

Run:

```
./monitor_run.sh
```

With supervisor:

```
sudo cp ~/NGI-TRUSTCHAIN/DOOF/components/monitor/sup.monitor.conf /etc/supervisor/conf.d/
sudo supervisorctl
reread
add monitor
```

See if all components are up and running:

```
status
```

You can monitor the output of the program in the logs directory:

```
cd ~/logs
sudo tail -f monitor.stdout.log
```

4.5 Python Client

The Python client needs to be configured to use the correct DOOF Gateway endpoints and the correct broker.

The MACROS are:

BROKER_HOST – the host on which your broker is running.

BROKER_PORT – the port which your broker is listening to.

API_LOGIN – endpoint of the (optional) authentication server

GATEWAY_HOST – Ip address or domain name of host where DOOF Gateway can be reached

GATEWAY_PORT – Port where DOOF Gateway can be reached; it can be 80 if it is deployed behind NGINX as an HTTP server, 443 if NGINX + HTTPS is used, 2783 if you want to connect to the Gateway directly without passing through NGINX, 5443 for direct connection but with HTTPS support (these values can be found in the running scripts of the Gateway)

SSL_VALUE – False for port 80, 2783 (when no certificates are used); self_signed is the provided self-signed certificates are used; True is certificates provided by certificate authority are used

GATEWAY_API - /imperatives if DOOF gateway is reached directly, /dop/imperatives if the provided NGINX configuration is used

API_SESSION - /startsession if DOOF gateway is reached directly, /dop/startsession if the provided NGINX configuration is used

API_ADMIN - /sysadmin if DOOF gateway is reached directly,s /dop/sysadmin if the provided NGINX configuration is used

Configure the client:

```
cd ~/NGI-TRUSTCHAIN/DOOF/components/clients/python/conf
cp client_apis_conf.template client_apis.conf

sed -i 's/BROKER_HOST/127.0.0.1/g' client_apis.conf && sed -i 's/BROKER_PORT/1883/g'
client_apis.con
```

```
sed -i 's/API_LOGIN/\/login-handler/g'  client_apis.conf
```

4.5.1 Default configuration: connect via HTTP to gateway behind NGINX

If Gateway is reachable on port 80 (e.g. behind NGINX):

```
sed -i 's/GATEWAY_HOST/127.0.0.1/g' client_apis.conf && sed -i 's/GATEWAY_PORT/80/g' client_apis.conf

sed -i 's/SSL_VALUE/False/g' client_apis.conf
```

If the provided NGINX configuration is used, please set the following endpoints:

```
sed -i 's/GATEWAY_API/\/dop\/imperatives/g'  client_apis.conf
sed -i 's/API_SESSION/\/dop\/startsession/g'  client_apis.conf
sed -i 's/API_ADMIN/\/dop\/sysadmin/g'  client_apis.conf
```

4.5.2 Alternative configuration

If the DOOF gateway has a domain name is reachable on port 443 please use those values:

```
sed -i 's/GATEWAY_HOST/gateway_domain/g' client_apis.conf && sed -i 's/GATEWAY_PORT/443/g'
```

If self-signed certificates are used:

```
sed -i 's/SSL_VALUE/self_signed/g' client_apis.conf
```

If a certificate by certificate authority is used:

```
sed -i 's/SSL_VALUE/True/g' client_apis.conf
```

If the client connects directly to the DOOF gateway, without being proxied through NGINX, use the following endpoints (no /dop prepended):

```
sed -i 's/GATEWAY_API/\/imperatives/g' client_apis.conf
sed -i 's/API_SESSION/\/startsession/g'  client_apis.conf
sed -i 's/API_ADMIN/\/sysadmin/g'  client_apis.conf
```

5 Python client APIs usage

This section shows how to use the Python client APIs from a command line. You can integrate the same APIs into custom programs.

Source the virtual environment:

```
source /home/ecosteer/virtualenv/dop/bin/activate

cd ~/NGI-TRUSTCHAIN/DOOF/components/clients/python
source env.sh
```

Invoke the console with the configuration file:

```
python client_console.py conf/client_apis.conf
```

Then you can invoke the APIs as indicated in D3 final apis. 'apis' is the object that offers all the functionalities. The console initializes the apis.

In a development scenario without authentication service, you can set up an unauthenticated session to start using the apis. Start with the admin account, as it is the unique account existing at the beginning:

5.1.1 Start a session (default when using provided NGINX configuration)

```
>>> apis.setup_unauth_session("admin@ecosteer.com")
```

After this, you can send all the events that the client and worker support. Functions have the same names of the events they send.

5.1.2 Enable identities (set up accounts)

The admin account exists as a user but it was not enabled on the blockchain, so you may want first of all to enable it:

```
subject = "admin@ecosteer.com"
apis.dop_enable_identity(subject, "admin")
```

You may want to create also an account for the data owner and one for the data recipient.

After this, please open two terminals, one where you can log in as data owner and one for the data recipient, so that you can perform the operations listed in the next subsections more easily.

```
apis.dop_enable_identity("data_owner@email.com", "data owner")
```

```
apis.dop_enable_identity("data_recipient@email.com", "data recipient")
```

Please also manually add the username and a custom password to the authentication table as well (see section 2.1) for the usage of change password functionality.

5.1.2.1 Login as data owner

Open a new terminal, source the virtual environment, start the client console, and then login as the newly enabled data owner:

```
>>> apis.setup_unauth_session(“data_owner@email.com”)
```

5.1.2.2 Login as data recipient

Open a new terminal, source the virtual environment, start the client console, and then login as the newly enabled data recipient:

```
>>> apis.setup_unauth_session(“data_recipient@email.com”)
```

5.1.3 Use of variables

You can set variables and use them in the API calls. It is recommended to save at least product_id and subscription_id as variables during the interaction with the back-end.

```
product_id = ""
subscription_id = ""
purpose_id = ""

product_label="product from cli"
price=0
period=0

purpose_label="Purpose of usage"
purpose_url="INSERT HERE A URL OF A DOCUMENT"
```

5.1.4 Account information

Get information about the account:

```
apis.dop_account_info()
```

5.1.5 Product creation

Login as the data owner and create a product:

```
product_label="product from cli"
price=0
period=0

apis.dop_product_create(product_label, price, period)
# save the product id you will see in the JSON reply in a variable
product_id=”69cb7f2f-a3ae-46e5-bccf-0b16d33d34f9”
```

5.1.6 Product information

As a data owner you can get the list of published products:

```
apis.dop_products_list(query_type="published")
```

Anyone can get detailed information about a product by specifying the product_id:

```
apis.dop_products_list(filer={"id":product_id})
```

5.1.7 Purpose of usage creation and information

Login as a data recipient and create a purpose of usage:

```
purpose_label="Purpose of usage"
purpose_url="INSERT HERE A URL OF A DOCUMENT"
apis.dop_purpose_create(purpose_label, purpose_url)
# save the purpose id in a variable, e.g. purpose_id
```

List the purposes of usage inserted by the current user:

```
apis.dop_purpose_list()
```

5.1.8 Subscriptions list for product

Check the subscriptions to the new product (dop_product_subscriptions operation can be performed by anyone):

```
apis.dop_product_subscriptions(product_id, query_type="any")
```

5.1.9 Subscribe

After having inserted a purpose of usage, a data recipient can subscribe to a product:

```
apis.dop_product_subscribe(product_id, purpose_id)
#save the subscription id in a variable
subscription_id=”ed0dea1d-d688-4331-bdea-8c0678d8ecee”
```

5.1.10 Subscription info

You can check the subscription info, both as a data owner and as a data user:

```
apis.dop_subscription_info(subscription_id)
```

For the data recipient, the product will be in the list of subscribed products:

```
apis.dop_products_list(query_type="subscribed")
```

You can also see all the subscriptions to a given product of the data recipient currently logged in:

```
apis.dop_product_subscriptions(product_id, query_type="sub")
```

5.1.11 Grant and revoke

As a data owner, you can change the visibility status of a subscription:

```
apis.dop_subscription_grant(subscription_id)
apis.dop_subscription_revoke(subscription_id)
```

5.1.12 Product configuration file

If you want to aquire the product configuration file you need to log in as data owner and request it to the worker:

```
apis.dop_pub_configuration(product_id)
```

The output will contain a JSON event as the following:

```
dop_python_apis - dop_pub_configuration_handler
events/f13dcd3a-f435-4dad-8bcd-3707901317b8
@REPLY{"task": "", "event": "dop_pub_configuration", "params": {"err": 0, "phase": 1, "product_id":
"799CC9ED-8529-4BBF-9D06-C29F9F9FDDA7", "config":
"eyJjaGFubmVsIjogIjc5OUNDOUVELTg1MjktNEJCRi05RDA2LUMyOUY5RjlGRERBNyIsICJwdWJsaXNoZX
IiOiB7InBhc3N3b3JkIjogIjYwYmEzMWE1MWExYzYwMWJjNjVjIiwgInNlY3JldCI6ICJzZWNyZXQiLCAicHJv
ZHVjdCI6ICJkYWZlYmE5MC02ZjNkLTQ1Y2UtYTNhNS1hOTM4MTNmMWYyZTIifSwgInNlcnZlciI6IFt7Im
FkZHJlc3MiOiAiMTI3LjAuMC4xIiwgInBvcnQiOiAiMzAwMCJ9XSwgImdyYWNlX3BlcmlvZCI6IDQwMDAsI
CJrZXlfbWF4YWdlIjogMzAsICJrZXlfbGVuIjogNDgsICJrZXlfaW52YWxpZF9tYXhjb3VudCI6IDE1LCAibG9vc
F9pbnRlcnZhbCI6IDUwMDAsICJzdGFydF93aXRoX2dldCI6IHRydWUsICJsb2dfbGV2ZWwiOiAzLCAia3Nf
d2F0Y2hkb2dfbWF4dGltZSI6IDQwMDAsICJrZ193YXRjaGRvZ19tYXh0aW1lIjogNDAwMH0="}}
```

Decode the base64 encoded content of the params.config key to get a JSON payload that you can use to configure the data origin's PET capabilities.

The decoded content will look like the following:

```
{"channel": "799CC9ED-8529-4BBF-9D06-C29F9F9FDDA7", "publisher": {"password":
"60ba31a51a1c601bc65c", "secret": "secret", "product": "dafeba90-6f3d-45ce-a3a5-a93813f1f2e2"},
"server": [{"address": "127.0.0.1", "port": "3000"}], "grace_period": 4000, "key_maxage": 30,
"key_len": 48, "key_invalid_maxcount": 15, "loop_interval": 5000, "start_with_get": true, "log_level":
3, "ks_watchdog_maxtime": 4000, "kg_watchdog_maxtime": 4000}
```

The processor for dop_pub_configuration can be configured to return the information needed by the selected PET's data origin integration point.

5.1.13 Subscription configuration file

For the subscription configuration file, you need to log in as data recipient, request it to the worker and decode it similarly to the product's configuration file:

```
apis.dop_sub_configuration(subscription_id)
```

Use the JSON payload to configure the data recipient's PET capabilities.

As for the processor for dop_pub_configuration, the processor for dop_sub_configuration can be configured to return the information needed by the selected PET's data sink integration point.

5.1.14 Unsubscribe

You can also unsubscribe from the product when logged in as data recipient:

```
apis.dop_product_unsubscribe(subscription_id)
```

5.1.15 Operations related to advertisements

Data recipients can add advertisements to which data owners can adhere in order to receive a subscription from the data recipient. Rif_advertisement_list operation was tailored for the front-end pilot for this purpose, and only returns those advertisments to which users can adhere to.

An example of usage of rif_advertisement_create:

```
apis.rif_advertisement_create("secret", " … I offer this service in exchange of data visibility for this purpose …", purpose_id, "UNIQUE ID ON RECIPIENT SIDE")
```

A data owner can request the advertisement with:

```
apis.rif_advertisement_list()
```

5.1.16 Close connection

After having performed the operations, you can close the connection:

```
apis.close()
```

Use CTRL-D to exit the console.

5.1.17 Login (JWT authentication)

In a production scenario where an authentication server is used and the login-handler returns an authentication header containing a JWT, use setup_new_session for initiating a new session, supply username and password:

```
>>> apis.setup_new_session("admin@ecosteer.com", "ecosteer")
```

6 Pilot: front-end deployment and usage

Configure the front-end to connect to the services running on the chosen hosts.

```
cd ~/NGI-TRUSTCHAIN/DOOF/pilot
cp index_config.template index_config.json
```

Replace the MACROS with your chosen values, the ones related to the deployment of the third-party services and NGINX:

```
sed -i 's/BROKER_HOST/$IP/g' index_config.json && sed -i 's/BROKER_PORT/8083/g' index_config.json && sed -i 's/BROKER_TOPIC/events/g' index_config.json

sed -i 's/MLE_ID/2/g' index_config.json && sed -i 's/MLE_CIPHER/none/g' index_config.json && sed -i 's/MLE_MODE/ECB/g' index_config.json

sed -i 's/AUTH_TYPE/none/g' index_config.json

sed -i 's/NGINX_HOST/$IP/g' index_config.json && sed -i 's/NGINX_PORT/80/g' index_config.json

sed -i 's/OWNER_APP_HOST/$IP/g' index_config.json && sed -i 's/OWNER_APP_PORT/80/g' index_config.json
```

Under the Pilot, under the section Useful Info > Manuals, you will find a video showcasing the Qubee and the procedure for its configuration.

7 DOOF Platform usage

As a test, you can add users via the Python APIs, login with these users via the Python APIs, and then create products. Then you can login into the web application with the accounts of those users, and see the products you inserted previously.

The front-end does not have the capability of adding new products as in the pilot the users and their assets were provisioned before giving them access to the front-end itself. The same applies to subscriptions, which in the pilot were registered via command line operations through the Python clients.

The front-end allows data owners to manage information about their accounts, their products and the subscriptions to their products, and to see notifications and messages regarding subscriptions.