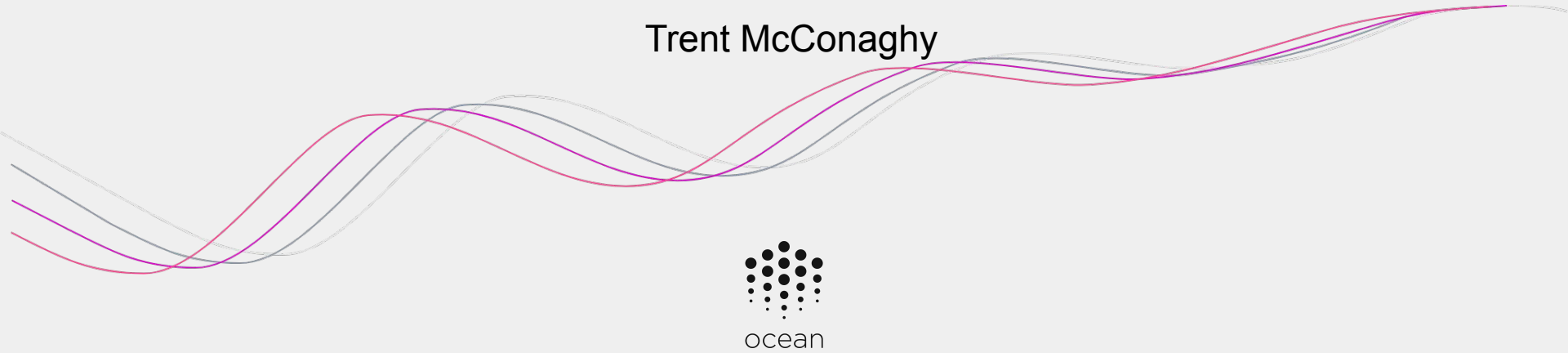
Composable DeSci Flows Via Token-Wrapped Data & Algorithms

Trent McConaghy



ocean

May 24, 2022

oceanprotocol.com
@trentmc0

Science * Data

Science is about making models to predict unseen events, in a reproducible way.
It's data all the way down.

Data can be

- raw measurements
- cleaned data
- features
- scientific models
- predictions

Data can also be algorithms to build the models.
Data can be dynamically changing.

Tokens for Composable Data Flows

Goal: composable, reproducible data flows.

How: tokenize data & algorithms

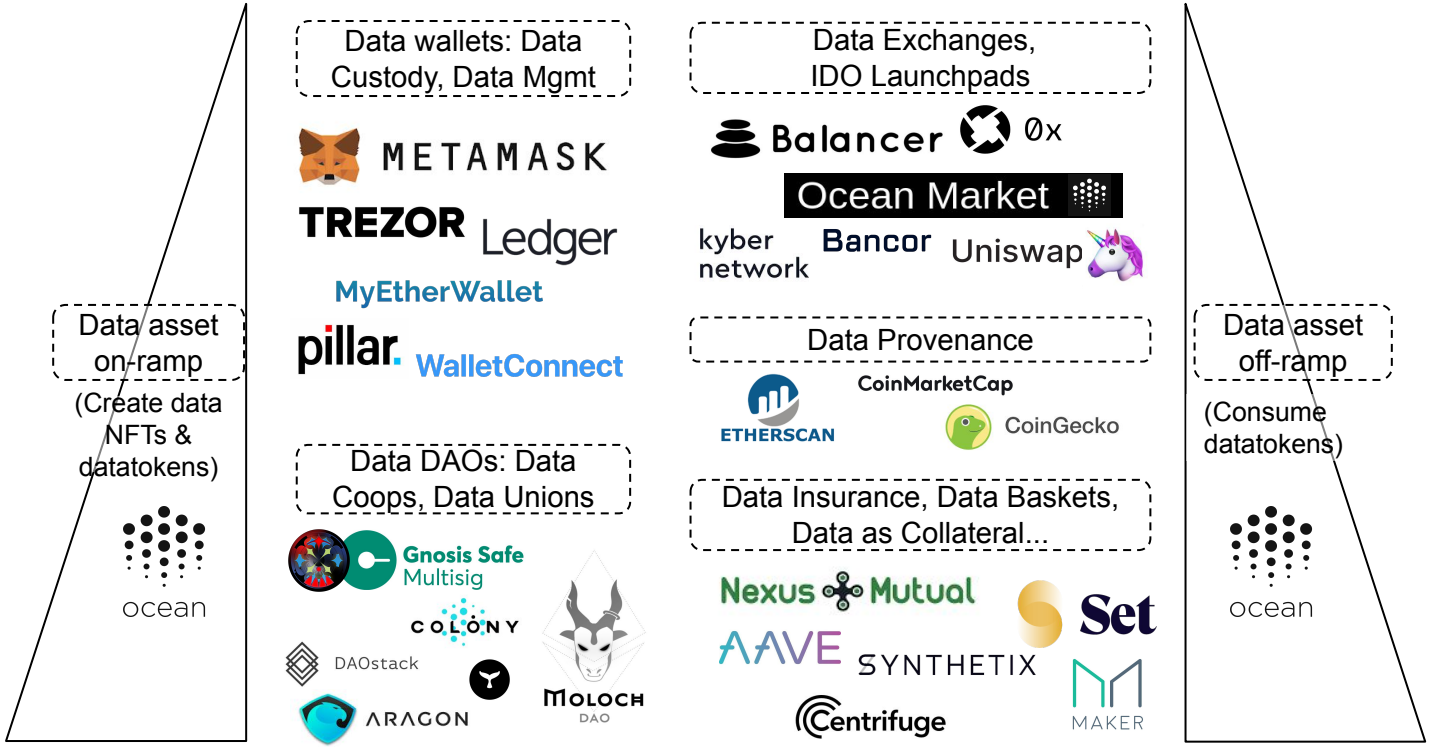
- **Small data: store on-chain inside ERC721 data NFT**
- **Large data: store off-chain. Access control via ERC20 datatokens**
  - Compute-to-Data flows preserve privacy and control.
  - Data & algorithms can use Web3 storage or compute.

Details: Where to store data <> How to share it

Where to store	Where to store: specific medium	How to share (access control)
Off-chain	Any web2 or web3 service. Eg S3, Filecoin	• Fully open, don't need provenance: just use http • Fully open, want provenance: Ocean datatokens, with free dispense • Share if paid: Ocean datatokens, with fixed-price, AMM, etc. • Fully private, only seen by algorithms: Ocean datatokens + Compute-to-Data
On-chain (small data)	Key-value pairs in data NFTs	• Fully open: store value plaintext • Open to marketplaces etc: encrypt value, share symmetric key liberally • Open sparingly: encrypt value, share symmetric key sparingly

Tokenize Data & Algorithms

Data on-ramp: deploy and mint ERC721 data NFTs & ERC20 datatokens.
Data off-ramp: consume datatokens
Enables data assets * Web3 wallets, exchanges, and DAOs



ocean

Atomic → Higher Level Building Blocks

Atomic building blocks: Data NFTs and datatokens

Higher level blocks. The atomic blocks naturally interoperate with

- Web3 wallets
- DAOs
- DEXes
- NFT marketplaces
- etc

Higher level yet. From this, we can construct many DeSci flows:

- AI-training provenance
- scientific model commons
- algorithm marketplaces
- and more

ocean

What I'll cover in detail

- On-chain data → data NFTs
- On-chain data with privacy → data NFTs with encryption
- Off-chain data → datatokens
- Off-chain data with privacy → datatokens with Compute-to-Data

ocean

On-chain data: Data NFTs

ocean

On-chain data (small): Ocean Data NFTs

```
erc721_nft = ocean.create_erc721_nft('NFTToken1', 'NFT1', alice_wallet)
```

```
#Key-value pair
key = "fav_color"
value = "blue"

#prep key for setter
key_hash = ocean.web3.keccak(text=key) #Contract/ERC725 requires keccak256 hash

#prep value for setter
value_hex = value.encode('utf-8').hex() #set_new_data() needs hex

#set
erc721_nft.set_new_data(key_hash, value_hex, alice_wallet)
```

```
value2_hex = erc721_nft.get_data(key_hash)
value2 = value2_hex.decode('ascii')
print(f"Found that {key} = {value2}")
```

ocean

On-chain data with privacy: Data NFTs with encryption

ocean

On-chain data with privacy: Data NFTs with encryption

Basic idea: symmetric key to encrypt & decrypt; share it by encrypting with consumer's pub key

- Use cases: "login with Web3", "soul-bound tokens"
- https://github.com/oceanprotocol/ocean.py/blob/v4main/READMEs/profile-nfts-flow.md

```
erc721_nft = ocean.create_erc721_nft('NFTToken1', 'NFT1', alice_wallet)
```

```
#Prep value for setter
profiledata_val_encr_hex = Fernet(symkey).encrypt(profiledata_val.encode('utf-8')).hex()

#set
erc721_nft.set_new_data(profiledata_name_hash, profiledata_val_encr_hex, alice_wallet)
```

```
profiledata_val_encr_hex2 = erc721_nft.get_data(profiledata_name_hash)
profiledata_val2_bytes = Fernet(symkey).decrypt(profiledata_val_encr_hex2)
```

ocean

Off-chain data: Datatokens

ocean

Off-chain data: Ocean datatokens

```
erc721_nft = ocean.create_erc721_nft('NFTToken1', 'NFT1', alice_wallet)
```

```
erc20_token = erc721_nft.create_datatoken(
    template_index=1, # default value
    name="ERC20DT1",  # name for ERC20 token
    symbol="ERC20DT1Symbol",  # symbol for ERC20 token
    minter=alice_wallet.address,  # minter address
```

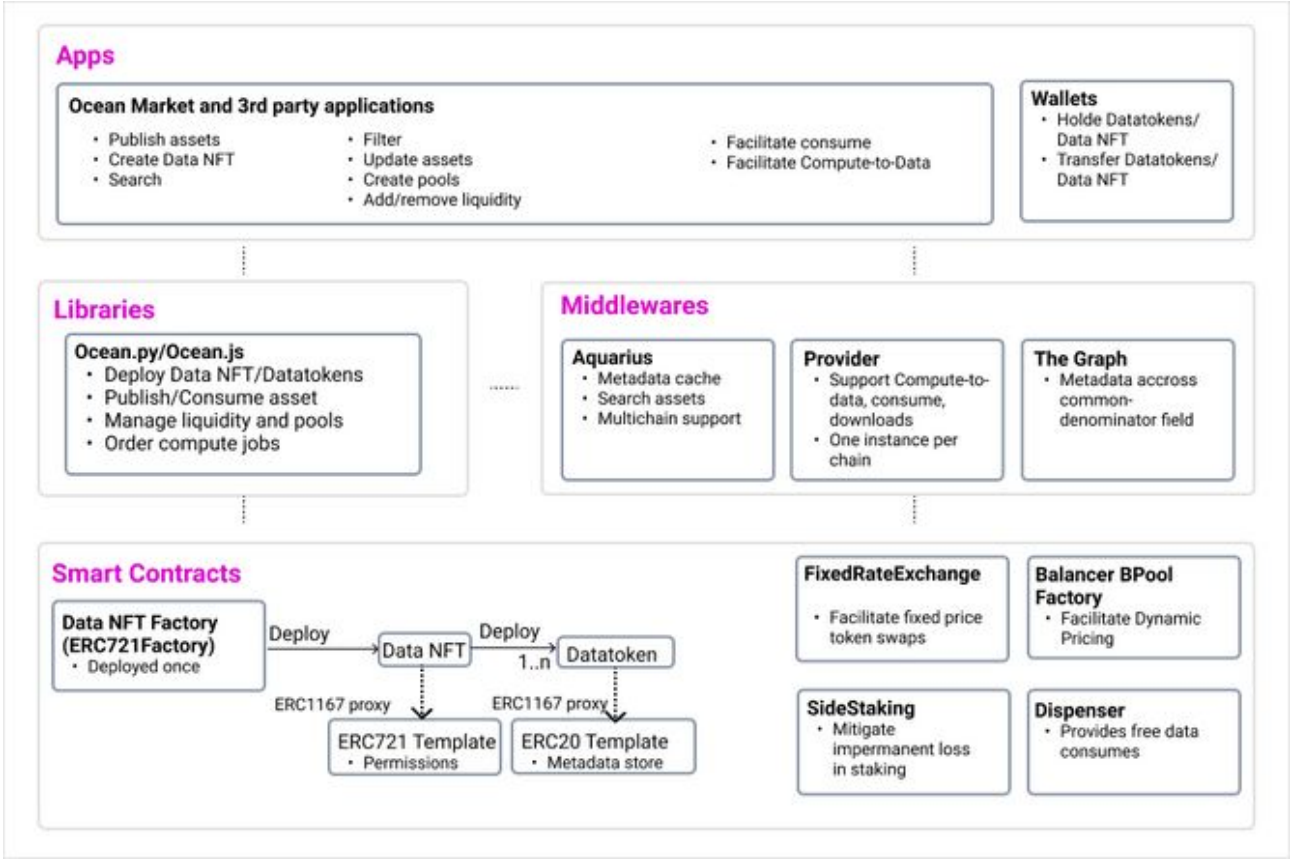
<here: Bob gets 1.0 datatokens>

```
# Bob sends his datatoken to the service
order_tx_id = ocean.assets.pay_for_access_service(
    asset,
    service,
```

```
# Bob downloads. If the connection breaks, Bob can request again by
file_path = ocean.assets.download_asset(
    asset=asset,
    service=service,
    consumer_wallet=bob_wallet,
    destination='./',
    order_tx_id=order_tx_id
)
```

ocean

Detail: Ocean architecture

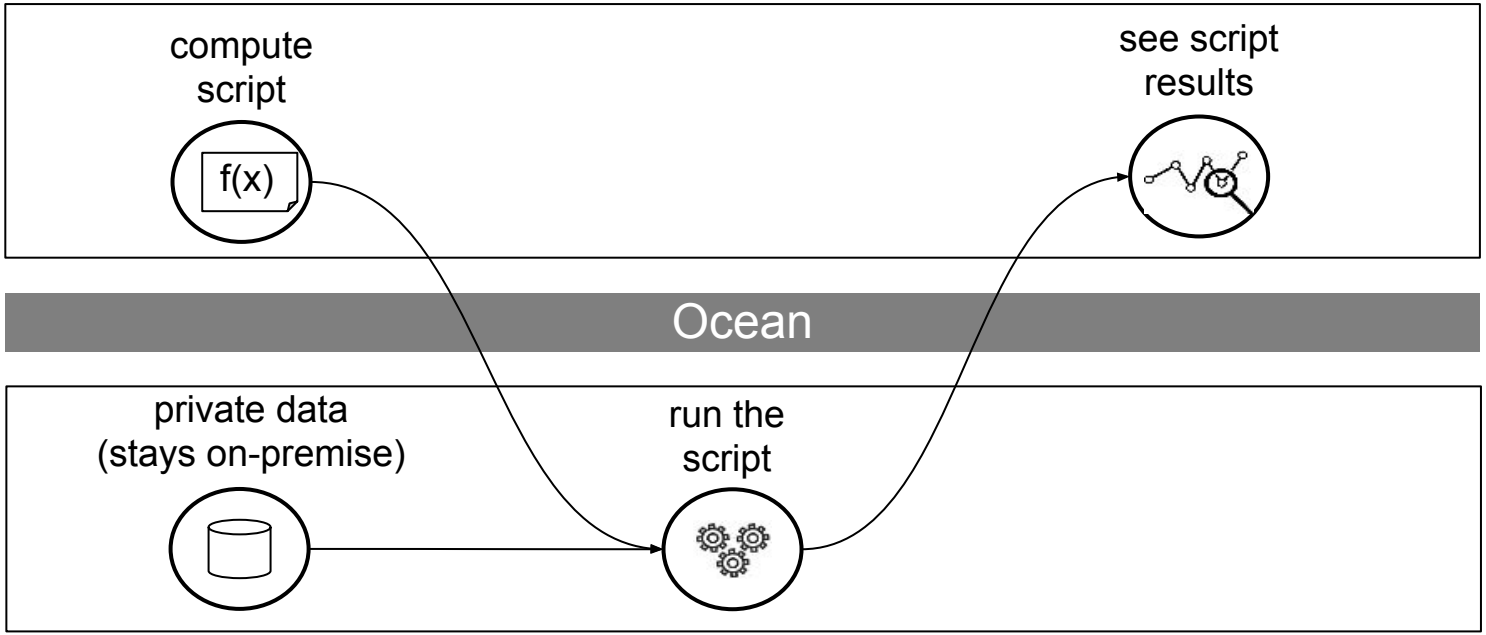


ocean

Off-chain data with privacy: Datatokens + Compute-to-Data

ocean

Off-chain data with privacy:
Ocean datatokens with Compute-to-Data



ocean

C2D Quickstart via Ocean.py: Overview

github.com/oceanprotocol/ocean.py/blob/main/READMEs/c2d-flow.md

🏄 Quickstart

Simple Flow

This stripped-down flow shows the essence of Ocean: simply cre

Go to simple flow

Marketplace flow

In this flow, a data asset is posted for sale in a marketplace, and pool.

Go to marketplace flow

Compute-to-Data flow

This flow uses Ocean Compute-to-Data (c2d) to compute results

Go to c2d flow

3. Alice publishes algorithm

For this step, there are some prerequisites needed. If you want to replace the sample algo need to do some dependency management. You can use one of the standard Ocean algo Use the image name and tag in the `container` part of the algorithm metadata. This dock dependency installation e.g. in the case of Python, OS-level library installations, pip instal more about docker image publishing.

In the same Python console:

```
# Publish ALG datatoken
ALG_datatoken = ocean.create_data_token('ALG1', 'ALG1', alice_wallet, blob=
ALG_datatoken.mint(alice_wallet.address, to_wei(100), alice_wallet)
print(f"ALG_datatoken.address = '{ALG_datatoken.address}'")

# Specify metadata and service attributes, for "GPR" algorithm script.
# In same location as Branin test dataset. GPR = Gaussian Process Regressior
ALG_metadata =  {
    "main": {
        "type": "algorithm",
```

You can use the result however you like. For the purpose of this example, let's plot it.

```
import numpy
from matplotlib import pyplot

X0_vec = numpy.linspace(-5., 10., 15)
X1_vec = numpy.linspace(0., 15., 15)
X0, X1 = numpy.meshgrid(X0_vec, X1_vec)
b, c, t = 0.1291845091439806, 1.5915494309189535, 0.039788735772973836
u = X1 - b*X0**2 + c*X0 - 6
r = 10.*(1. - t) * numpy.cos(X0) + 10
Z = u**2 + r

fig, ax = pyplot.subplots(subplot_kw={"projection": "3d"})
ax.scatter(X0, X1, model, c="r", label="model")
pyplot.title("Data + model")
pyplot.show() # or pyplot.savefig("test.png") to save the plot as a .png f
```

Here are the steps:

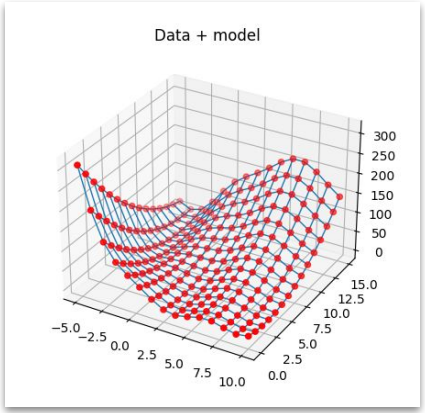
1. Setup
2. Alice publishes data asset
3. Alice publishes algorithm
4. Alice allows the algorithm for C2D for that data asset
5. Bob acquires datatokens for data and algorithm
6. Bob starts a compute job
7. Bob monitors logs / algorithm output

6. Bob starts a compute job

Only inputs needed: DATA_did, ALG_did. Everything else can get computed as ne

In the same Python console:

```
DATA_did = DATA_ddo.did  # for convenience
ALG_did = ALG_ddo.did
DATA_DDO = ocean.assets.resolve(DATA_did)  # make sure we operate on t
ALG_DDO = ocean.assets.resolve(ALG_did)

compute_service = DATA_DDO.get_service('compute')
algo_service = ALG_DDO.get_service('access')

from ocean_lib.web3_internal.constants import ZERO_ADDRESS
from ocean_lib.models.compute_input import ComputeInput

# order & pay for dataset
```



C2D Quickstart via Ocean.py: Where to find

github.com/oceanprotocol/ocean.py/blob/main/READMEs/c2d-flow.md

🏄 Quickstart

Here are flows to try out, from simple to specific detailed variants.

- **Simple flow** - the essence of Ocean - creating a data NFT & datatoken.
- **Publish flow** - a dataset is published.
- **Consume flow** - a published dataset is consumed (downloaded).
- **Marketplace flow** - a data asset is posted for sale in a datatoken pool, then purchased. Includes metadata.
- **Fixed rate exchange flow** - a data asset is posted for sale at fixed rate, then purchased.
- **Dispenser flow** - here, a datatoken dispenser is created and datatokens are dispensed for free.
- **Compute-to-data flow** - uses C2D to build an AI model a dataset that never leaves the premises.
- **Key-value database** - use data NFTs to store arbitrary key-value pairs on-chain.
- **Profile NFTs** - enable "login with Web3" where Dapp can access private user profile data.

C2D Quickstart: Steps

github.com/oceanprotocol/ocean.py/blob/main/READMEs/c2d-flow.md

1. Setup
2. Alice publishes data asset
3. Alice publishes algorithm
4. Alice allows the algorithm for C2D for that data asset
5. Bob acquires datatokens for data and algorithm
6. Bob starts a compute job
7. Bob monitors logs / algorithm output

C2D Quickstart: Step 2: Publish dataset

github.com/oceanprotocol/ocean.py/blob/main/READMEs/c2d-flow.md

```
# Publish DATA datatoken, mint tokens
from ocean_lib.web3_internal.currency import to_wei

DATA_datatoken = ocean.create_data_token('DATA1', 'DATA1', alice_wallet, blob=ocean.config.metadata_cache_uri)
DATA_datatoken.mint(alice_wallet.address, to_wei(100), alice_wallet)
print(f"DATA_datatoken.address = '{DATA_datatoken.address}'")

# Specify metadata & service attributes for Branin test dataset.
# It's specified using _local_ DDO metadata format; Aquarius will convert it to remote
# by removing `url` and adding `encryptedFiles` field.
DATA_metadata = {
    "main": {
        "type": "dataset",
        "files": [
          {
            "url": "https://raw.githubusercontent.com/trentmc/branin/main/branin.arff",
            "index": 0,
            "contentType": "text/text"
          }
        ],
```

C2D Quickstart: Step 3: Publish algorithm

github.com/oceanprotocol/ocean.py/blob/main/READMEs/c2d-flow.md

```
# Publish ALG datatoken
ALG_datatoken = ocean.create_data_token('ALG1', 'ALG1', alice_wallet, blob=ocean.config.metadata_cache_uri)
ALG_datatoken.mint(alice_wallet.address, to_wei(100), alice_wallet)
print(f"ALG_datatoken.address = '{ALG_datatoken.address}'")

# Specify metadata and service attributes, for "GPR" algorithm script.
# In same location as Branin test dataset. GPR = Gaussian Process Regression.
ALG_metadata =  {
    "main": {
        "type": "algorithm",
        "algorithm": {
            "language": "python",
            "format": "docker-image",
            "version": "0.1",
            "container": {
              "entrypoint": "python $ALGO",
              "image": "oceanprotocol/algo_dockers",
              "tag": "python-branin"
            }
        },
        "files": [
          {
            "url": "https://raw.githubusercontent.com/trentmc/branin/main/gpr.py",
            "index": 0,
```

C2D Quickstart: Step 4: dataset allows algorithm

github.com/oceanprotocol/ocean.py/blob/main/READMEs/c2d-flow.md

```
from ocean_lib.assets.trusted_algorithms import add_publisher_trusted_algorithm
add_publisher_trusted_algorithm(DATA_ddo, ALG_ddo.did, config.metadata_cache_uri)
ocean.assets.update(DATA_ddo, publisher_wallet=alice_wallet)
```

C2D Quickstart: Step 5: get data & alg assets

github.com/oceanprotocol/ocean.py/blob/main/READMEs/c2d-flow.md

```
bob_wallet = Wallet(
    ocean.web3,
    os.getenv('TEST_PRIVATE_KEY2'),
    config.block_confirmations,
    config.transaction_timeout,
)
print(f"bob_wallet.address = '{bob_wallet.address}'")

# Alice shares access for both to Bob, as datatokens. Alternatively, Bob might have bought these in a market.
DATA_datatoken.transfer(bob_wallet.address, to_wei(5), from_wallet=alice_wallet)
ALG_datatoken.transfer(bob_wallet.address, to_wei(5), from_wallet=alice_wallet)
```

C2D Quickstart: Step 6: start compute

github.com/oceanprotocol/ocean.py/blob/main/READMEs/c2d-flow.md

```
# order & pay for dataset
dataset_order_requirements = ocean.assets.order(
    DATA_did, bob_wallet.address, service_type=compute_service.type
)
DATA_order_tx_id = ocean.assets.pay_for_service(
        ocean.web3,
        dataset_order_requirements.amount,
        dataset_order_requirements.data_token_address,
        DATA_did,
        compute_service.index,
        ZERO_ADDRESS,
        bob_wallet,
        dataset_order_requirements.computeAddress,
    )
```

```
# order & pay for algo
algo_order_requirements = ocean.assets.order(
    ALG_did, bob_wallet.address, service_type=algo_service.type
)
ALG_order_tx_id = ocean.assets.pay_for_service(
        ocean.web3,
        algo_order_requirements.amount,
        algo_order_requirements.data_token_address,
        ALG_did,
        algo_service.index,
        ZERO_ADDRESS,
        bob_wallet,
        algo_order_requirements.computeAddress,
)
```

```
compute_inputs = [ComputeInput(DATA_did, DATA_order_tx_id, compute_service.index)]
job_id = ocean.compute.start(
    compute_inputs,
    bob_wallet,
    algorithm_did=ALG_did,
    algorithm_tx_id=ALG_order_tx_id,
    algorithm_data_token=ALG_datatoken.address
)
print(f"Started compute job with id: {job_id}")
```

C2D Quickstart: step 7: see output

github.com/oceanprotocol/ocean.py/blob/main/READMEs/c2d-flow.md

In the same Python console, you can check the job status as many times as needed:

```
ocean.compute.status(DATA_did, job_id, bob_wallet)
```

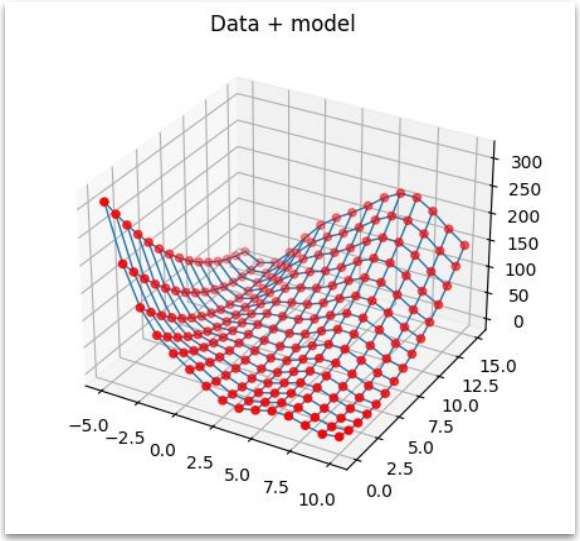
This will output the status of the current job. Here is a list of possible results: Operator Service Status description.

Once you get `{'ok': True, 'status': 70, 'statusText': 'Job finished'}`, Bob can check the result of the job.

```
result = ocean.compute.result_file(DATA_did, job_id, 0, bob_wallet)  # 0 index, means we retrieve the

import pickle
model = pickle.loads(result)  # the gaussian model result
```

You can use the result however you like. For the purpose of this example, let's plot it.

```
import numpy
from matplotlib import pyplot

X0_vec = numpy.linspace(-5., 10., 15)
X1_vec = numpy.linspace(0., 15., 15)
X0, X1 = numpy.meshgrid(X0_vec, X1_vec)
b, c, t = 0.12918450914398066, 1.5915494309189535, 0.039788735772973836
u = X1 - b*X0**2 + c*X0 - 6
r = 10.*(1. - t) * numpy.cos(X0) + 10
Z = u**2 + r

fig, ax = pyplot.subplots(subplot_kw={"projection": "3d"})
ax.scatter(X0, X1, model, c="r", label="model")
pyplot.title("Data + model")
pyplot.show() # or pyplot.savefig("test.png") to save the plot as a .png f
```



C2D Quickstart via Ocean.py: Recap

github.com/oceanprotocol/ocean.py/blob/main/READMEs/c2d-flow.md

🏄 Quickstart

Simple Flow

This stripped-down flow shows the essence of Ocean: simply cre

Go to simple flow

Marketplace flow

In this flow, a data asset is posted for sale in a marketplace, and pool.

Go to marketplace flow

Compute-to-Data flow

This flow uses Ocean Compute-to-Data (c2d) to compute results

Go to c2d flow

3. Alice publishes algorithm

For this step, there are some prerequisites needed. If you want to replace the sample algo need to do some dependency management. You can use one of the standard Ocean algo Use the image name and tag in the `container` part of the algorithm metadata. This dock dependency installation e.g. in the case of Python, OS-level library installations, pip instal more about docker image publishing.

In the same Python console:

```
# Publish ALG datatoken
ALG_datatoken = ocean.create_data_token('ALG1', 'ALG1', alice_wallet, blob=
ALG_datatoken.mint(alice_wallet.address, to_wei(100), alice_wallet)
print(f"ALG_datatoken.address = '{ALG_datatoken.address}'")

# Specify metadata and service attributes, for "GPR" algorithm script.
# In same location as Branin test dataset. GPR = Gaussian Process Regressior
ALG_metadata =  {
    "main": {
        "type": "algorithm",
```

You can use the result however you like. For the purpose of this example, let's plot it.

```
import numpy
from matplotlib import pyplot

X0_vec = numpy.linspace(-5., 10., 15)
X1_vec = numpy.linspace(0., 15., 15)
X0, X1 = numpy.meshgrid(X0_vec, X1_vec)
b, c, t = 0.1291845091439806, 1.5915494309189535, 0.039788735772973836
u = X1 - b*X0**2 + c*X0 - 6
r = 10.*(1. - t) * numpy.cos(X0) + 10
Z = u**2 + r

fig, ax = pyplot.subplots(subplot_kw={"projection": "3d"})
ax.scatter(X0, X1, model, c="r", label="model")
pyplot.title("Data + model")
pyplot.show() # or pyplot.savefig("test.png") to save the plot as a .png fi
```

Here are the steps:

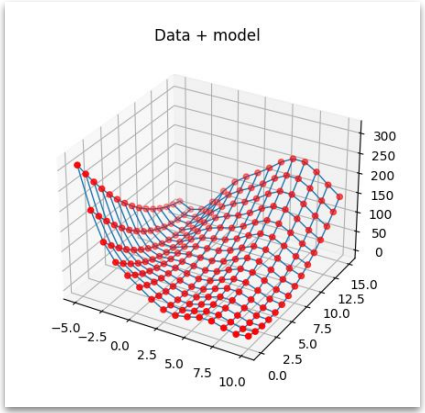
1. Setup
2. Alice publishes data asset
3. Alice publishes algorithm
4. Alice allows the algorithm for C2D for that data asset
5. Bob acquires datatokens for data and algorithm
6. Bob starts a compute job
7. Bob monitors logs / algorithm output

6. Bob starts a compute job

Only inputs needed: DATA_did, ALG_did. Everything else can get computed as ne

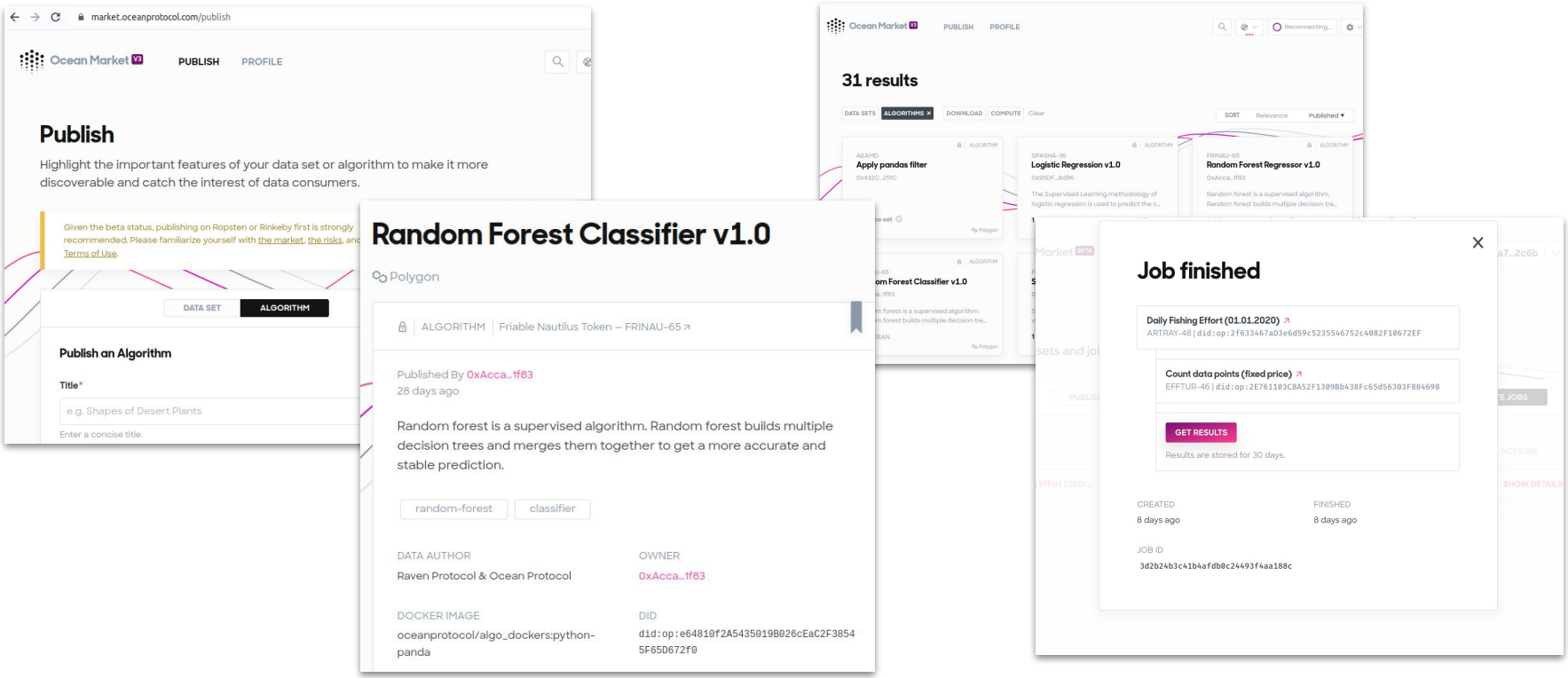
In the same Python console:

```
DATA_did = DATA_ddo.did  # for convenience
ALG_did = ALG_ddo.did
DATA_DDO = ocean.assets.resolve(DATA_did)  # make sure we operate on t
ALG_DDO = ocean.assets.resolve(ALG_did)

compute_service = DATA_DDO.get_service('compute')
algo_service = ALG_DDO.get_service('access')

from ocean_lib.web3_internal.constants import ZERO_ADDRESS
from ocean_lib.models.compute_input import ComputeInput

# order & pay for dataset
```



Compute-to-Data In Ocean Market

blog.oceanprotocol.com/compute-to-data-is-now-available-in-ocean-market-58868be52ef7

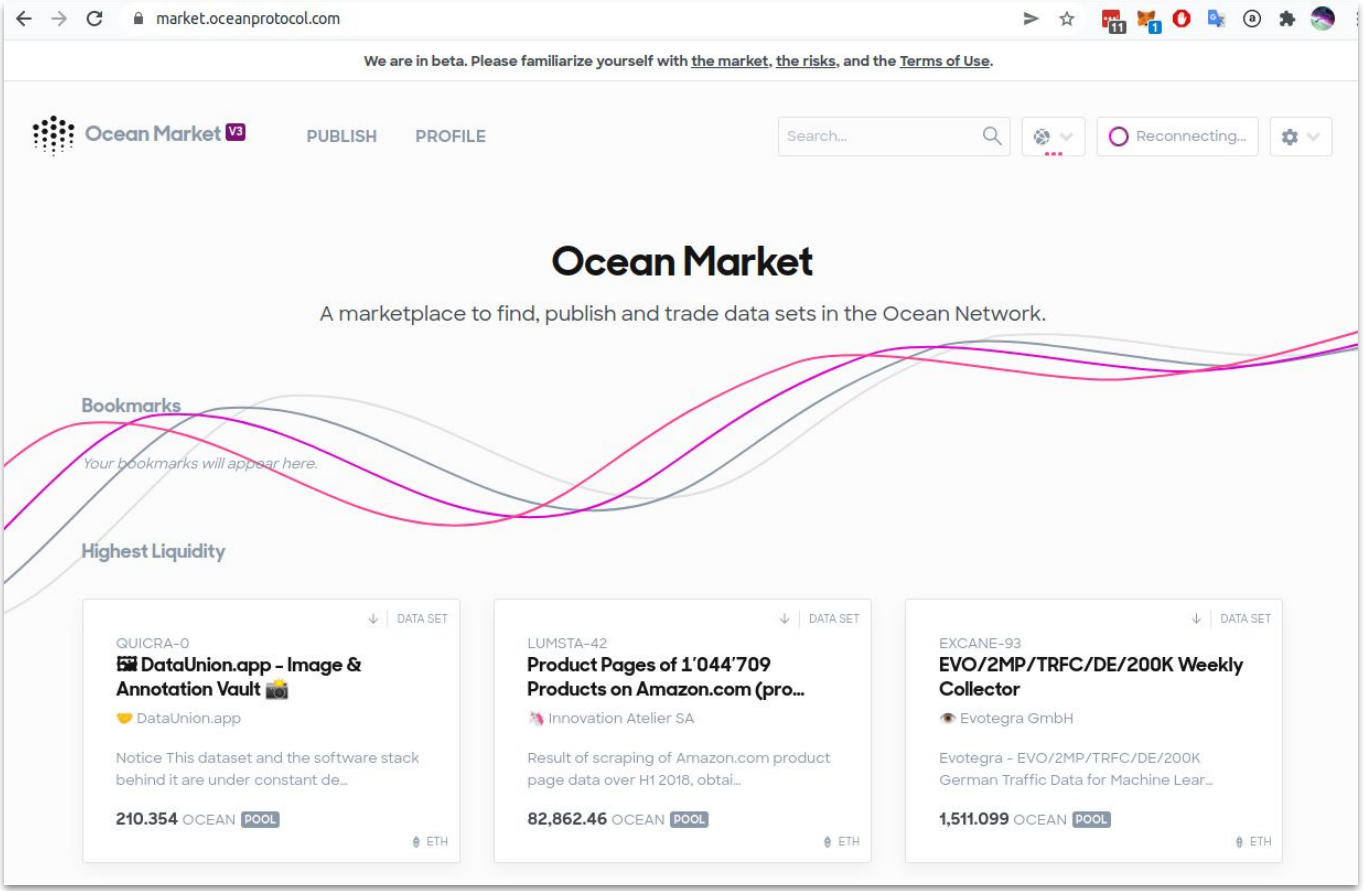


ocean

Ocean Market:
Decentralized data market for algorithms + data

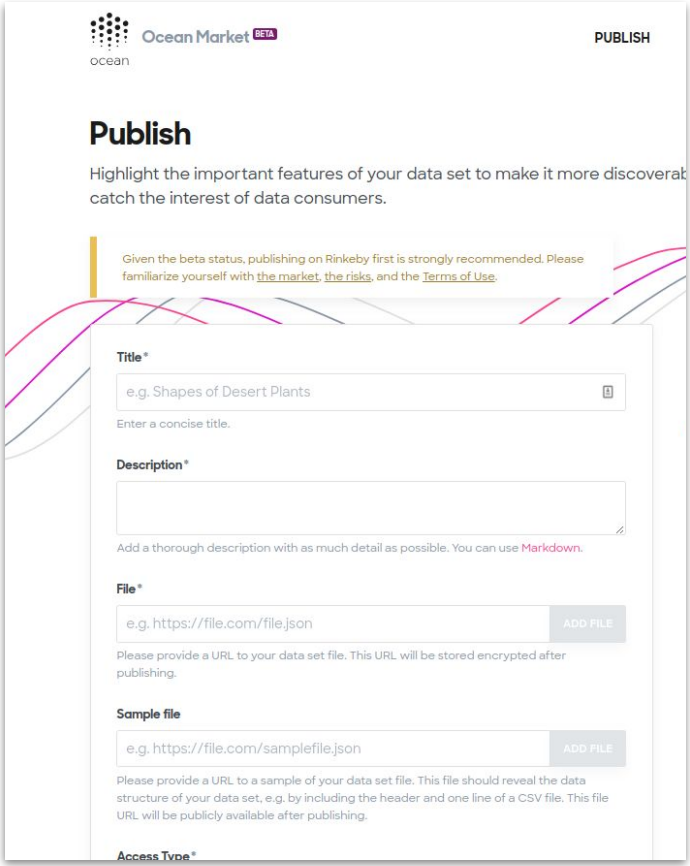
ocean

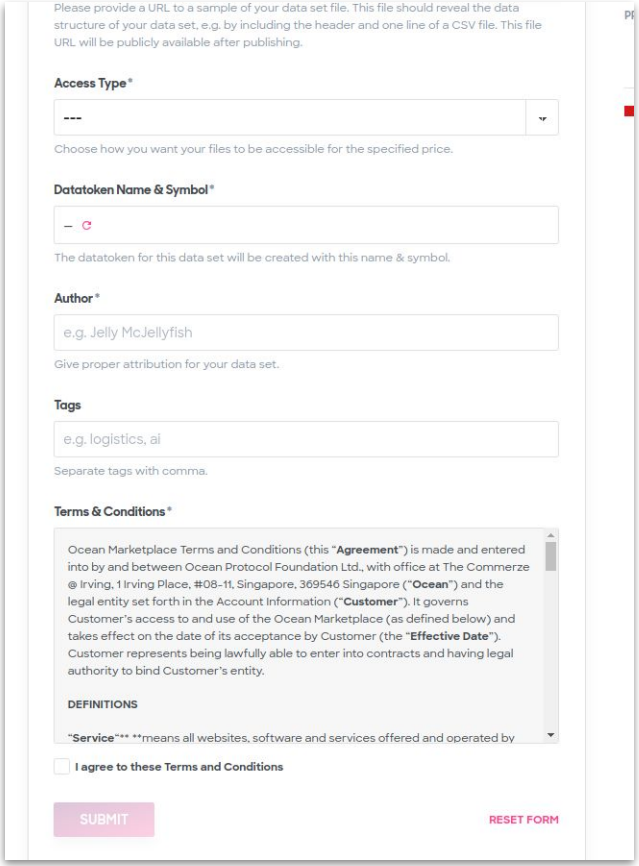
Ocean Market: Splash Page



ocean

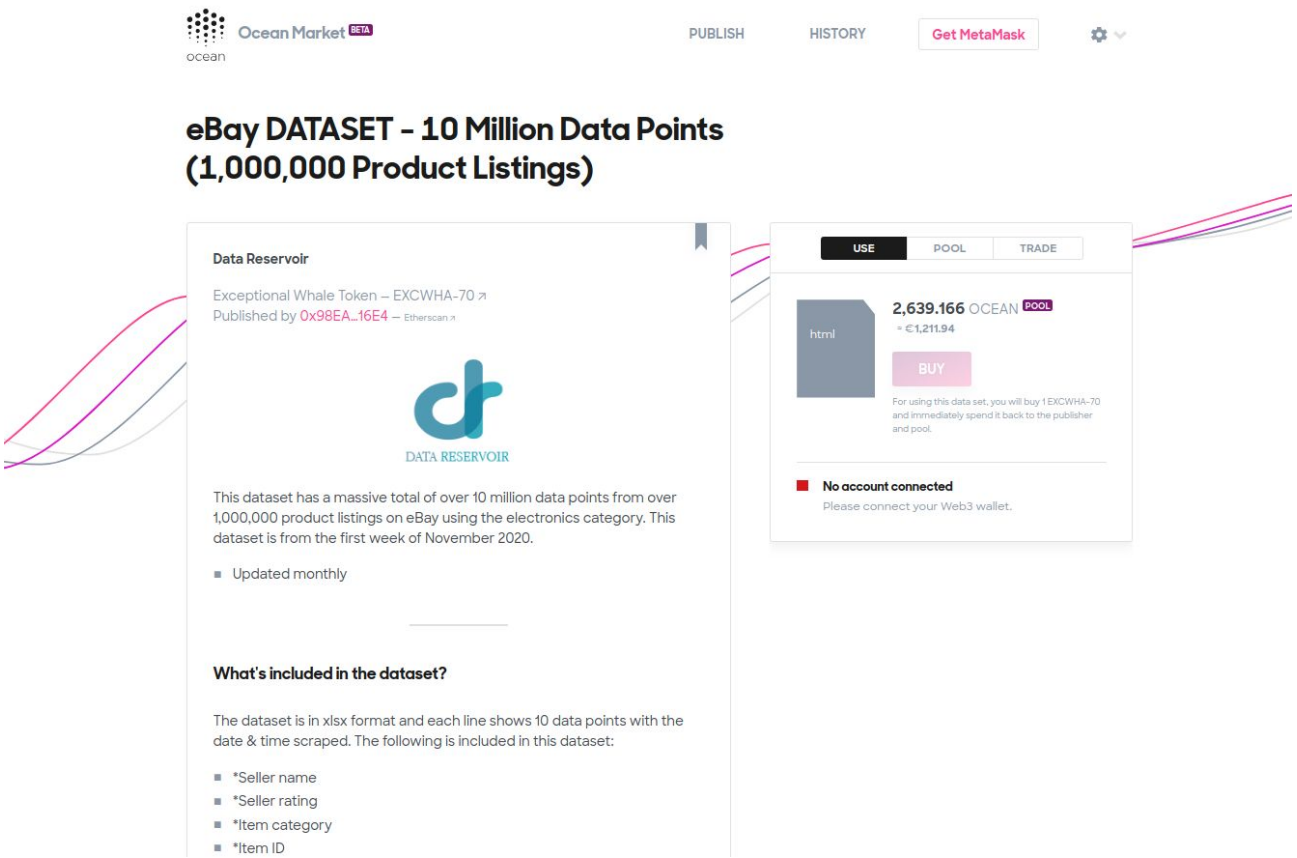
Ocean Market: Publish Flow, for an Initial Data Offering





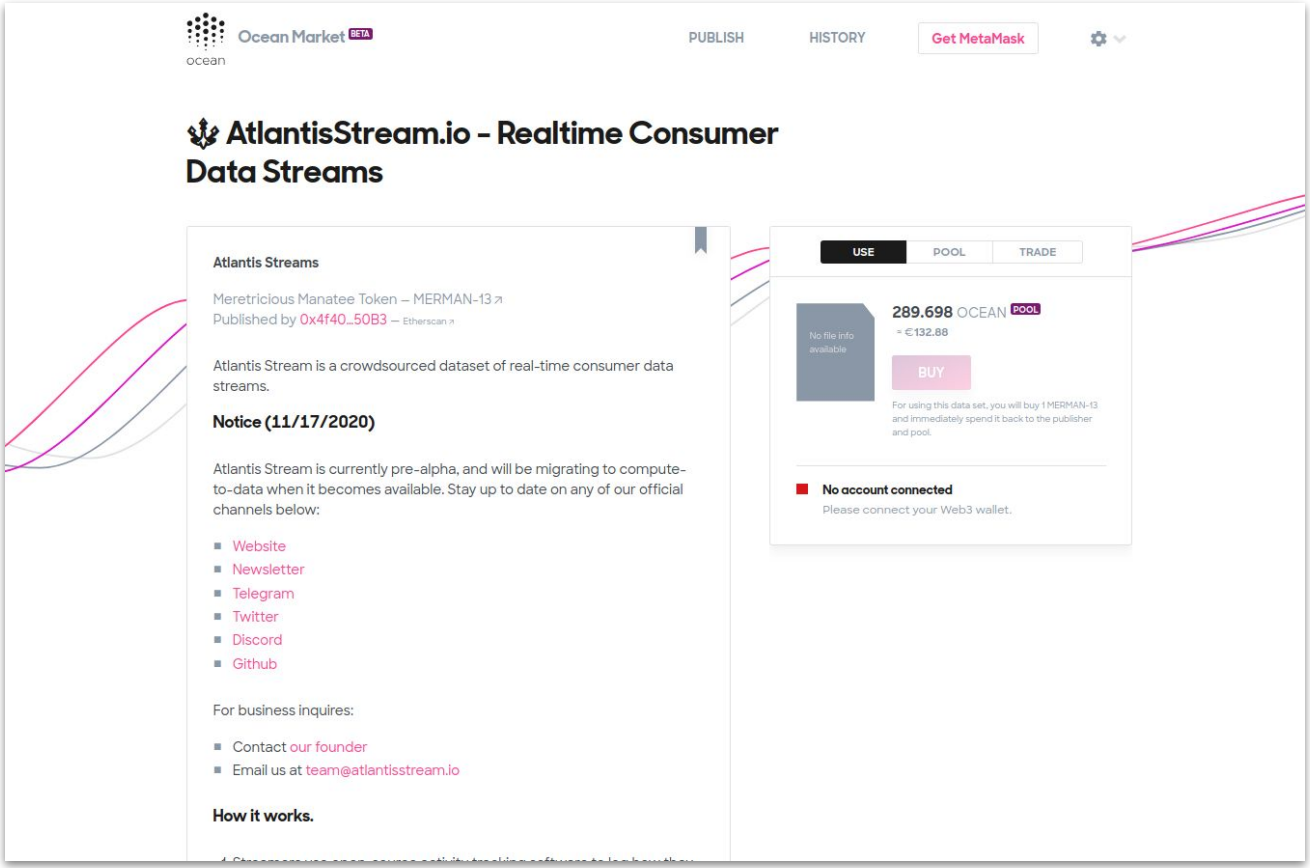
ocean

Example Data Asset, for Fixed Price



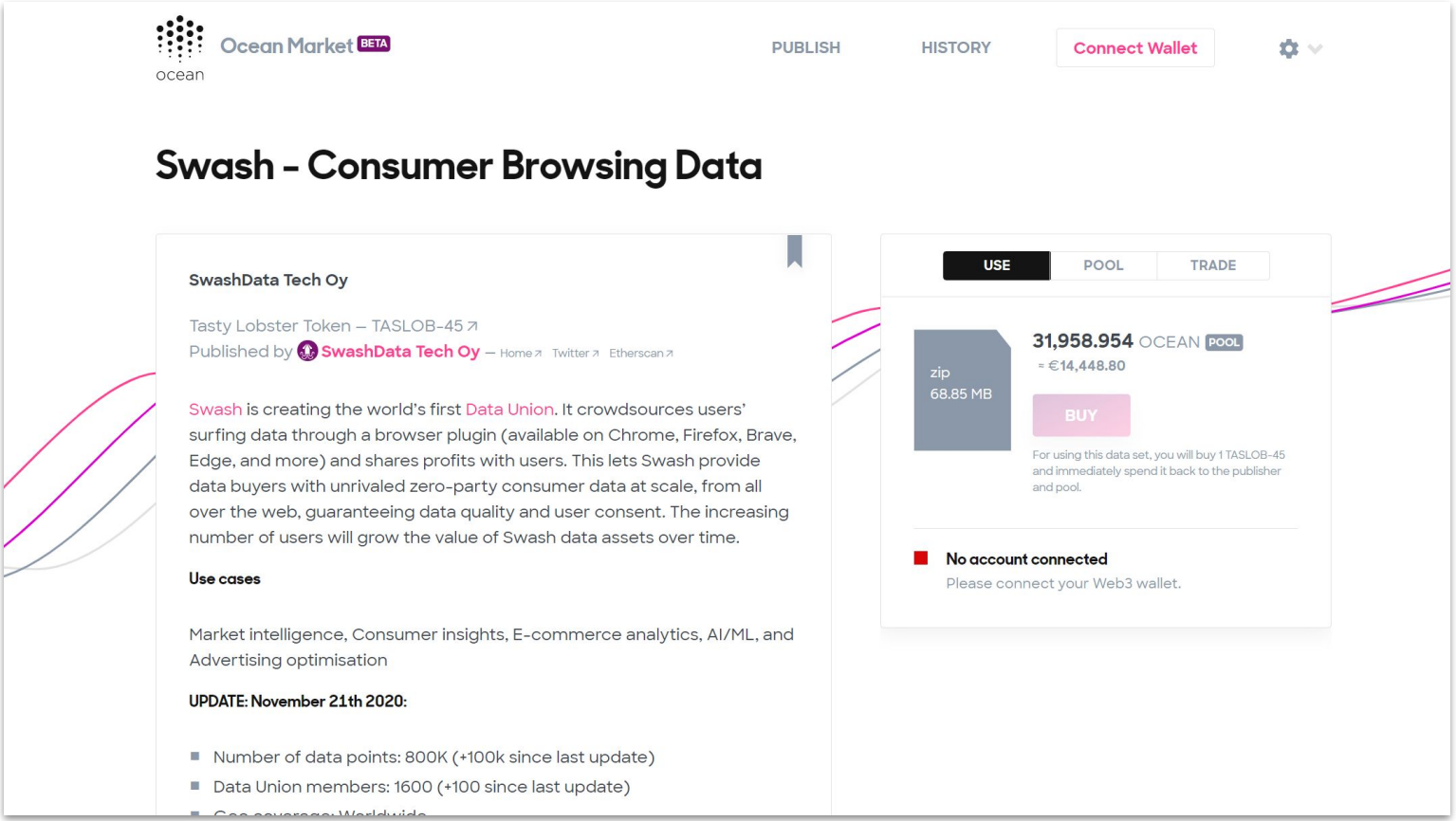
ocean

Example Data Asset, with Automatic Price Discovery (via AMM)



ocean

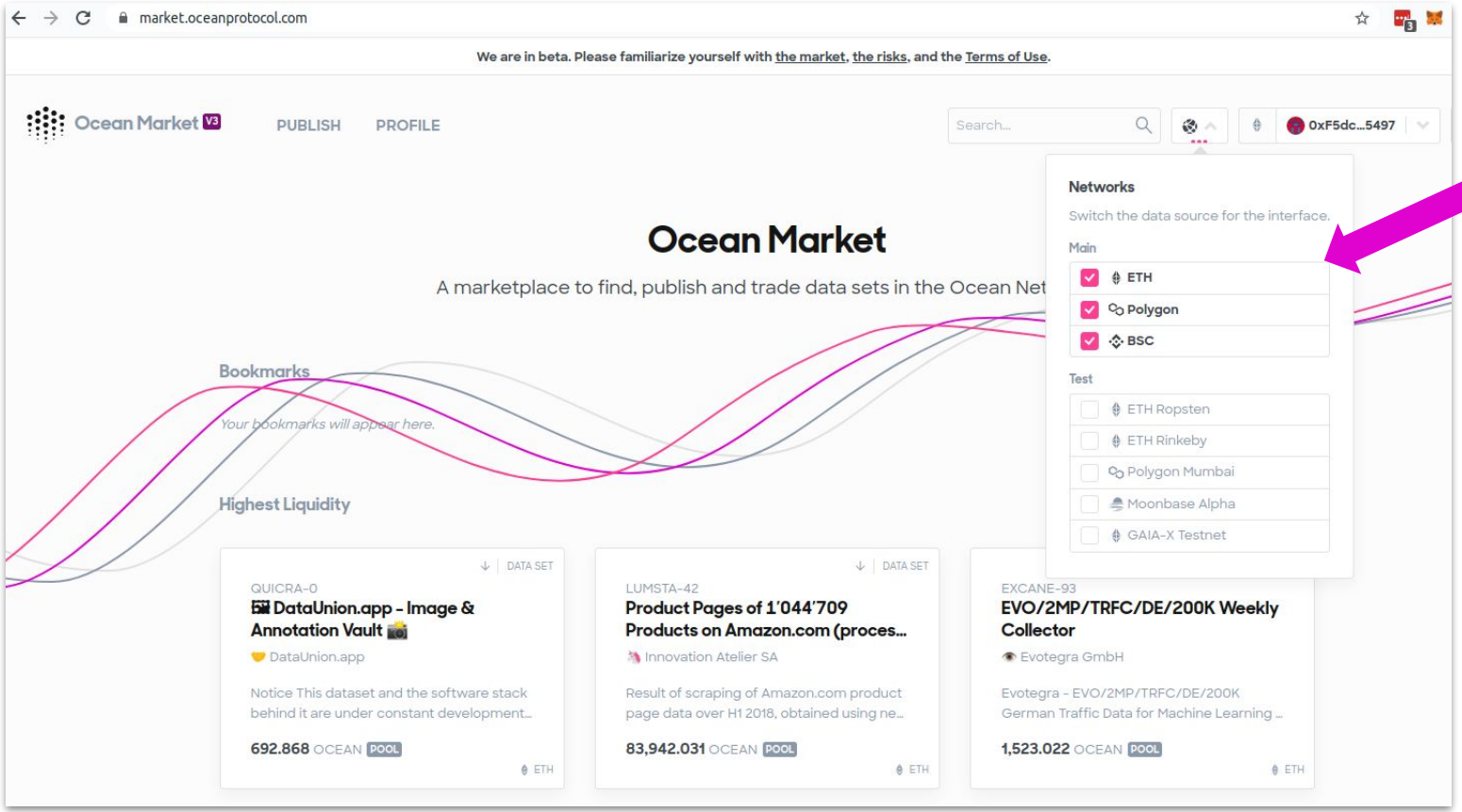
Example Data Asset: A Data Union



ocean

Ocean is multi-chain

blog.oceanprotocol.com/ocean-makes-multinetwork-even-simpler-c3ec6c0cbd50



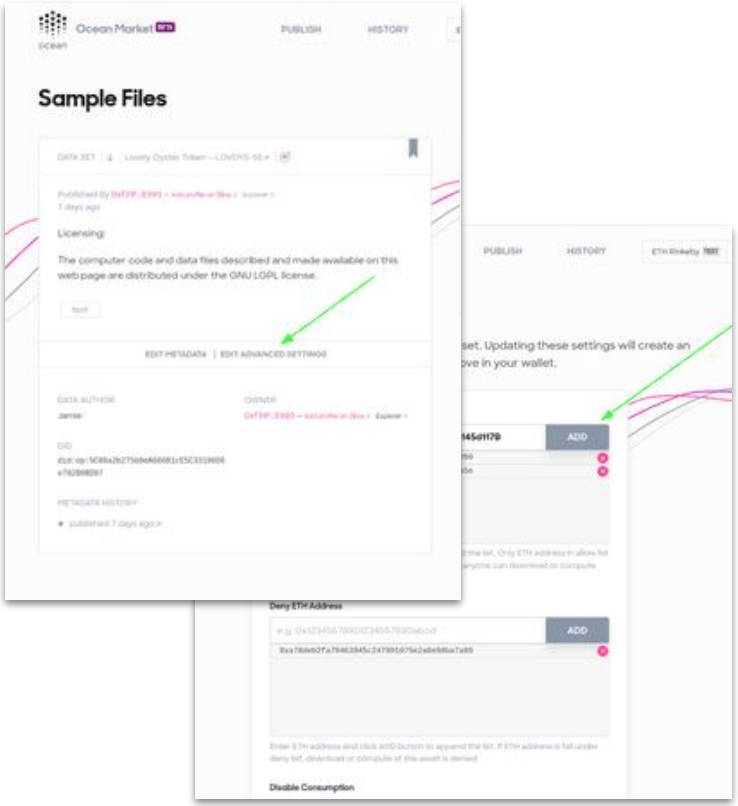
ocean

Fine-grained permissions

blog.oceanprotocol.com/fine-grained-permissions-now-supported-in-ocean-protocol-4fe434af24b9

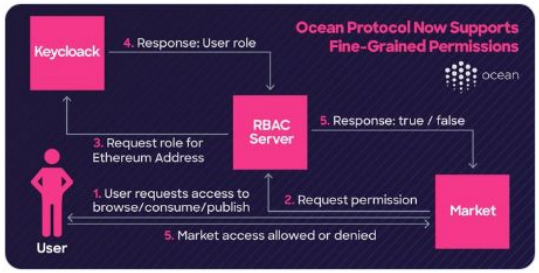
How to handle data exchange for
- 🏥Medical data only for credentialed EU researchers
- 🚗Selling automotive data within a consortium
- 🇩🇪🇸🇬Sharing data across offices in a multinational ?

@oceanprotocol fine-grained permissions handles this 🐬😀.



Ocean Protocol @oceanprotocol · Sep 22

🚨TECHNICAL UPDATE | Fine-Grained Permissions have now been launched on the Ocean Market that will offer the enterprises and other users more precise ways to specify and manage access. This new update addresses issues of access control along two main levels:



5 41 136

ocean

Conclusion

ocean

Science * Data

Science is about making models to predict unseen events, in a reproducible way.
It's data all the way down.

Data can be

- raw measurements
- cleaned data
- features
- scientific models
- predictions

Data can also be algorithms to build the models.
Data can be dynamically changing.

Tokens for Composable Data Flows

Goal: composable, reproducible data flows.

How: tokenize data & algorithms

- **Small data: store on-chain inside ERC721 data NFT**
- **Large data: store off-chain. Access control via ERC20 datatokens**
  - Compute-to-Data flows preserve privacy and control.
  - Data & algorithms can use Web3 storage or compute.

Tokens for Composable Data Flows

Try for yourself!

- https://github.com/oceanprotocol/ocean.py
- https://docs.oceanprotocol.com

ocean