Ocean Protocol Documentation

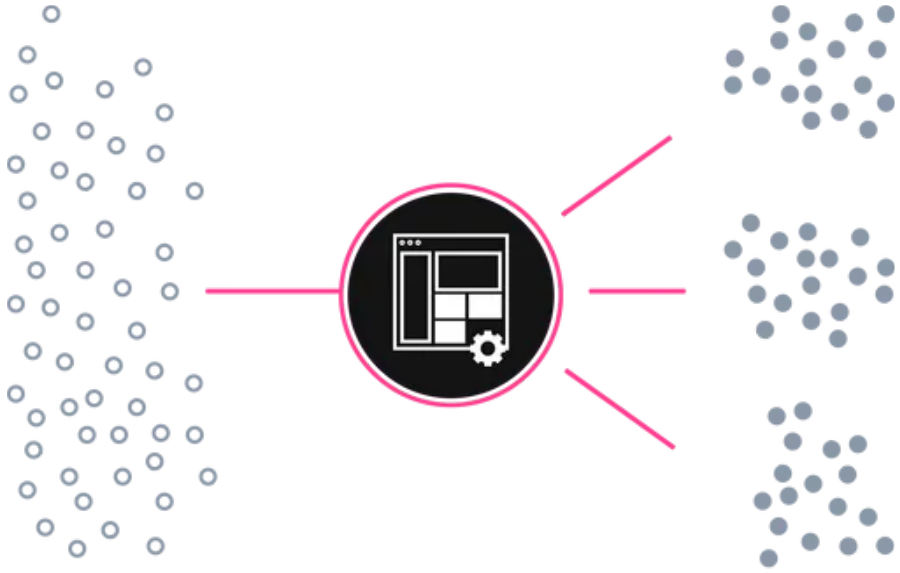
Orientation

Ocean Protocol - Tools for the Web3 Data Economy

What is Ocean?

Ocean provides the next generation of tools to unlock data at a large scale. Ocean makes it easy to publish and consume data services.

Ocean uses Data NFTs (ERC721) and datatokens (ERC20) as the interface to connect data assets with blockchain and DeFi tools. Crypto wallets become data wallets, crypto exchanges become data marketplaces, DAOs for data co-ops, and more via DeFi composability.



Creating a New Data Economy

The following guides are a greate place to start if you are new to Ocean:

- Architecture Overview
- Data NFTs and Datatokens
- Publish a data asset
- Download a data asset

What is our Mission?

To unlock data, for more equitable outcomes for users of data, using a thoughtful application of both technology and governance.

Society is becoming increasingly reliant on data, especially with the advent of AI. However, a small handful of organizations with both massive data assets and AI capabilities attained worrying levels of control which is a danger to a free and open society.

Our team and community is committed to kick-starting a New Data Economy that reaches every single person, company and device, giving power back to data owners and enabling people to capture value from data to better our world.

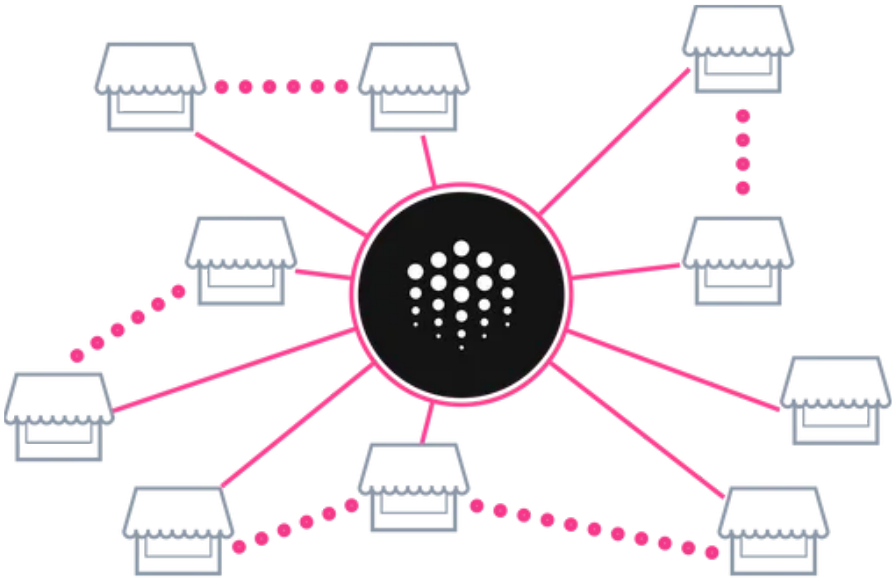
Find out more about the people building Ocean on our site.

What can you do with Ocean?

Buy or Sell Data

Use Ocean Market to publish and sell data, or browse and buy data. Data is published as interoperable ERC721 data NFTs & ERC20 datatokens. It's a decentralized exchange (DEX), tuned for data. The acts of publishing data, purchasing data, and consuming data are all recorded on the blockchain to make a tamper-proof audit trail.

As a data scientist or AI practitioner, you can benefit from access to more data (including private data), crypto-secured provenance in data & AI training, and income opportunities for selling data and curating data.



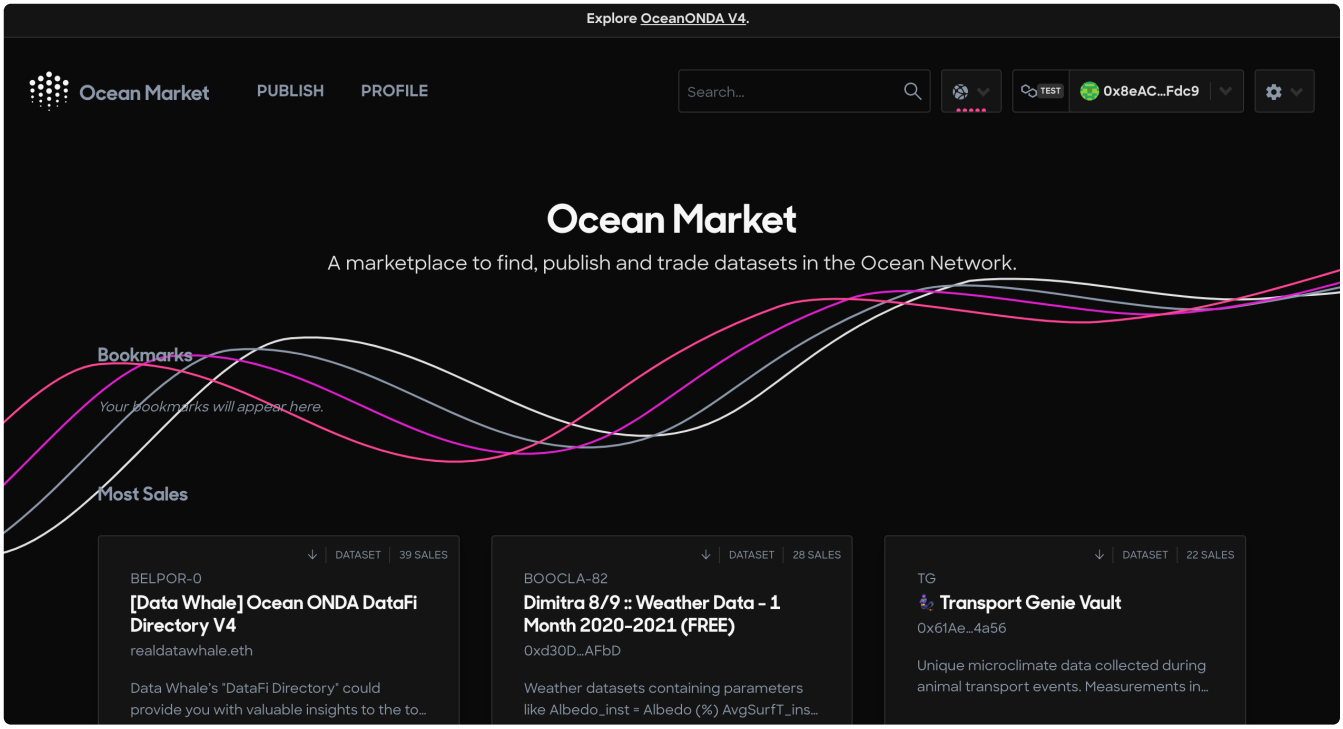
Decentralized Exchange Marketplaces

The following guides will help you get started with buying and selling data:

- Publish a data asset
- Download a data asset
- Publishing with hosting services

Build Your Own Data Market

Use Ocean Protocol software tools to build your own data marketplace, by either forking Ocean Market code or building up with Ocean components.



Ocean Market Homepage

If you're interested in starting your own marketplace checkout the following guides:

- Forking Ocean Market
- Customising your market
- Deploying your market

Manage datatokens and data NFTs for use in DeFi

Ocean makes it easy to publish data services (deploy ERC721 data NFTs and ERC20 datatokens), and to consume data services (spend datatokens). Crypto wallets, exchanges, and DAOs become data wallets, exchanges, and DAOs.

Use Ocean JavaScript or Python drivers to manage data NFTs and datatokens:

Ocean-based apps make data asset on-ramps and off-ramps easy for end users. Ocean smart contracts and libraries make this easy for developers. The data itself does not need to be on-chain, just the access control.

New Data on-ramp and off-ramp

Data NFTs are ERC721 tokens representing the unique asset and datatokens are ERC20 tokens to access data services. Each data service gets its own data NFT and one or more type of datatokens.

To access the dataset, you send 1.0 datatokens to the data provider (running Ocean Provider). To give access to someone else, send them 1.0 datatokens. That's it.

Since datatokens are ERC20, and live on Ethereum mainnet, there's a whole ecosystem to leverage.

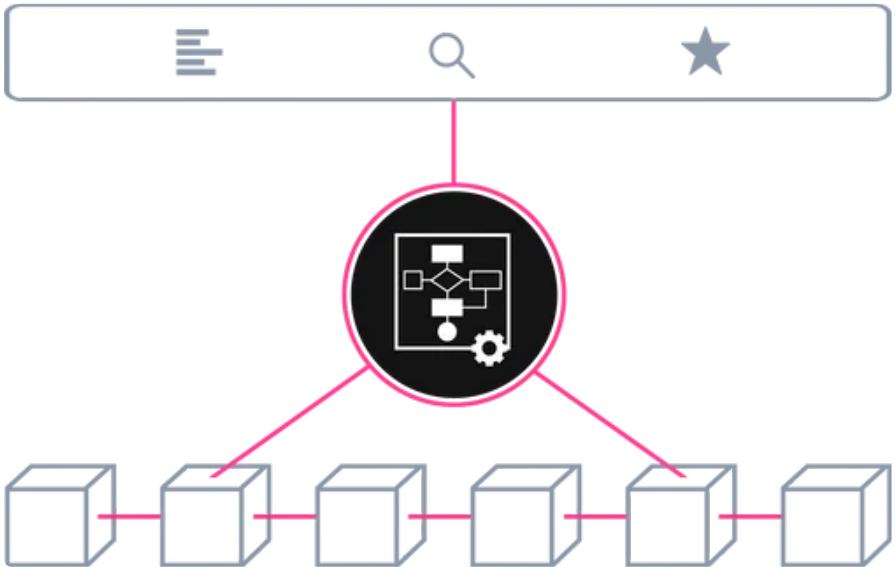
- *Publish and access data services:* downloadable files or compute-to-data. Use Ocean to deploy a new ERC721 and ERC20 datatoken contract for each data service, then mint datatokens.
- *Transfer datatokens* to another owner (or approve & transferFrom).
- *And more.* Use ERC20 support in web3.js, web3.py and Solidity to connect datatokens with crypto wallets and other DeFi services.

Compute-to-Data

Ocean's "Compute-to-Data" feature enables private data to be bought & sold. You can sell compute access to privately-held data, which never leaves the data owner’s premises. Ocean-based marketplaces enable the monetization of private data while preserving privacy.

Compute-to-data resolves the tradeoff between the benefits of using private data, and the risks of exposing it. It lets the data stay on-premise, yet allows 3rd parties to run specific compute jobs on it to get useful compute results like averaging or building an AI model.

The most valuable data is private data — using it can improve research and business outcomes. But concerns over privacy and control make it hard to access. With Compute-to-Data, private data isn’t directly shared but rather specific access to it is granted.



Compute-to-data

It can be used for data sharing in science or technology contexts, or in marketplaces for selling private data while preserving privacy, as an opportunity for companies to monetize their data assets.

Private data can help research, leading to life-altering innovations in science and technology. For example, more data improves the predictive accuracy of modern Artificial Intelligence (AI) models. Private data is often considered the most valuable data because it's so hard to get at, and using it can lead to potentially big payoffs.

Checkout these guides if you are aiming to get a deeper understanding on how compute-to-data works:

- Architecture
- Datasets & Algorithms
- Minikube Environment
- Writing Algorithms
- Private docker registry

How does it work?

In Ocean Protocol, each asset gets its own ERC721 **data NFT** and one(or more) ERC20 **datatokens**. This enables data wallets, data exchanges, and data co-ops by directly leveraging crypto wallets, exchanges, and more.

Ocean Protocol provides tools for developers to *build data markets*, and to *manage data NFTs and datatokens* for use in DeFi.

If you are new to web3 and blockchain technologies then we suggest you first read these introductory guides:

- Wallet Basics
- Set Up MetaMask Wallet
- Manage Your OCEAN Tokens

If ou are looking to get to grips with the inner workings of Ocean, then you'll be interested in the following guides:

- Architecture Overview
- Data NFTs and Datatokens
- Networks
- Fees
- Asset pricing
- DID & DDO
- Roles
- Set Up a Marketplace
- Compute-to-Data
- Deploying components
- Contributing

Supporters

GitBook is a supporter of this open source project by providing hosting for this documentation.

Wallet Basics

Ocean users need an ERC-20 compatible wallet to manage their ETH and OCEAN tokens.

Recommendations

- **Easiest:** Use MetaMask browser plug-in.
- **Still easy, but more secure:** Get a Trezor or Ledger hardware wallet, and use MetaMask to interact with it.
- The OCEAN Token page at oceanprotocol.com lists some other possible wallets.

The Meaning of "Wallet"

A wallet usually means "a thing that stores private keys (and maybe signs transactions)" (explained below). Examples include MetaMask, Trezor, and Ledger wallets.

A wallet can sometimes mean (web3) *software* for interacting with a thing that stores private keys. Examples include MetaMask, MyEtherWallet, and MyCrypto.

Note how MetaMask is in both lists!

Related Terminology

When you set up a new wallet, it might generate a **seed phrase** for you. Store that seed phrase somewhere secure and non-digital (e.g. on paper in a safe). It's extremely secret and sensitive. Anyone with your wallet's seed phrase could spend all the Ether and Ocean Tokens in all the accounts in your wallet.

Once your wallet is set up, it will have one or more **accounts**.

Each account has several **balances**, e.g. an Ether balance, an Ocean Token balance, and maybe other balances. All balances start at zero.

An account's Ether balance might be 7.1 ETH in the Ethereum Mainnet, 2.39 ETH in Görli testnet. You can move ETH from one network to another only with a specially setup exchange or bridge. Also, you can't transfer tokens from networks holding value such as Ethereum mainnet to networks not holding value, i.e., testnets like Görli. The same is true of OCEAN token balances.

Each account has one **private key** and one **address**. The address can be calculated from the private key. You must keep the private key secret because it's what's needed to spend/transfer ETH and OCEAN (or to sign transactions of any kind). You can share the address with others. In fact, if you want someone to send some ETH or OCEAN to an account, you give them the account's address.

Note that unlike traditional pocket wallets, crypto wallets don't actually store ETH or OCEAN. They store private keys.

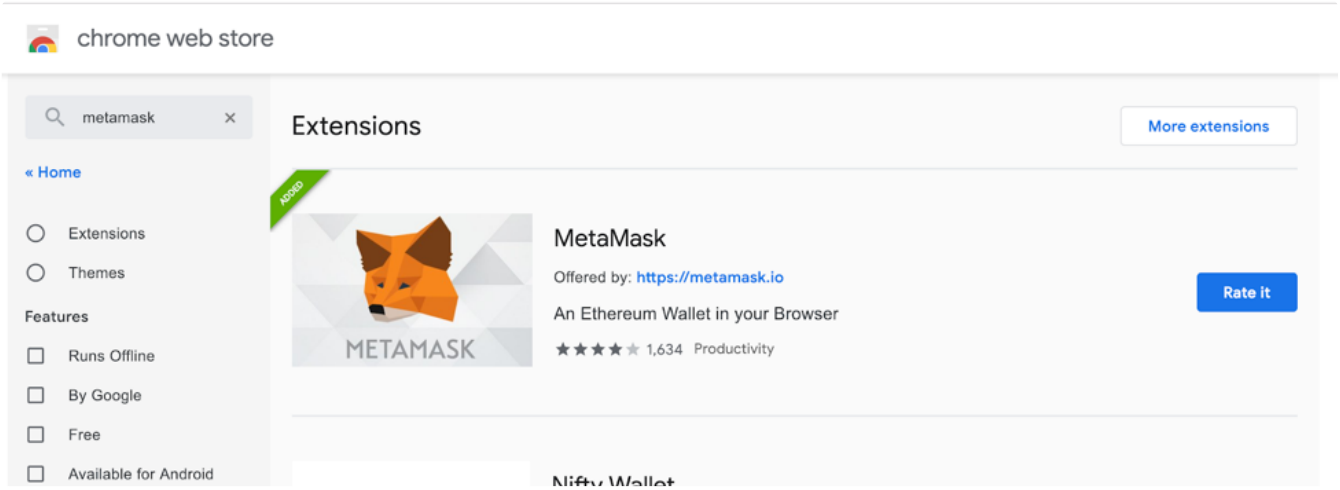
Set Up MetaMask Wallet

Tutorial about how to set up MetaMask for Chrome.

> MetaMask can also be used with a TREZOR or Ledger hardware wallet but we don't cover those options below; see the MetaMask documentation.

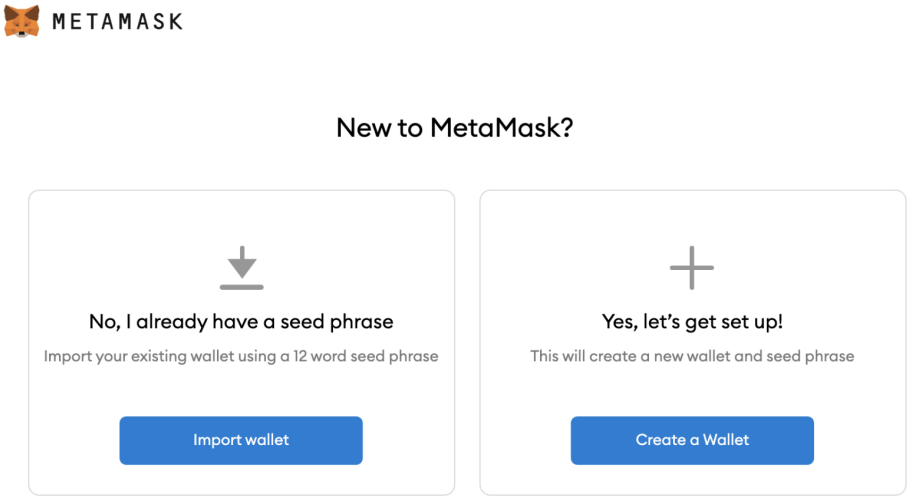
MetaMask Set Up Steps

1. Go to the Chrome Web Store for extensions and search for MetaMask.



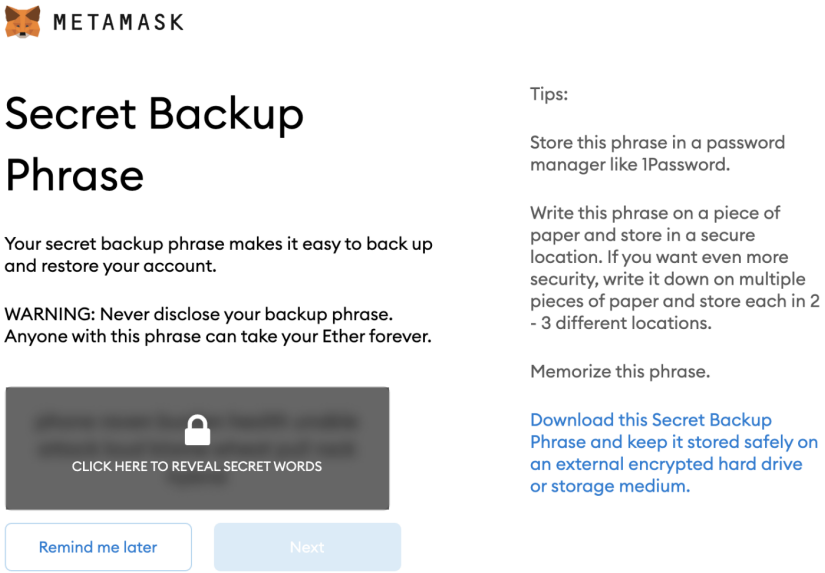
metamask-chrome-store

- Install MetaMask. The wallet provides a friendly user interface that will help you through each step. MetaMask gives you two options: importing an existing wallet or creating a new one. Choose to `Create a Wallet`:



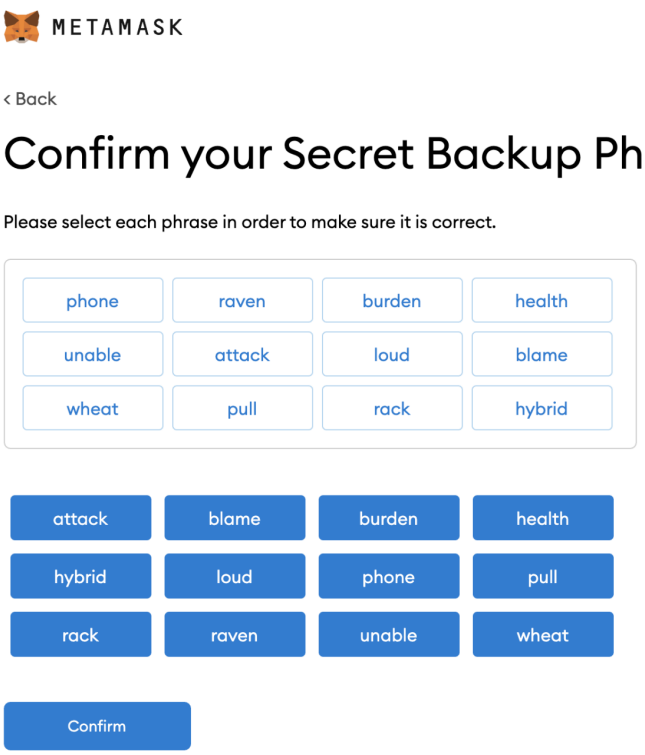
Create a wallet

- In the next step create a new password for your wallet. Read through and accept the terms and conditions. After that, MetaMask will generate Secret Backup Phrase for you. Write it down and store it in a safe place.



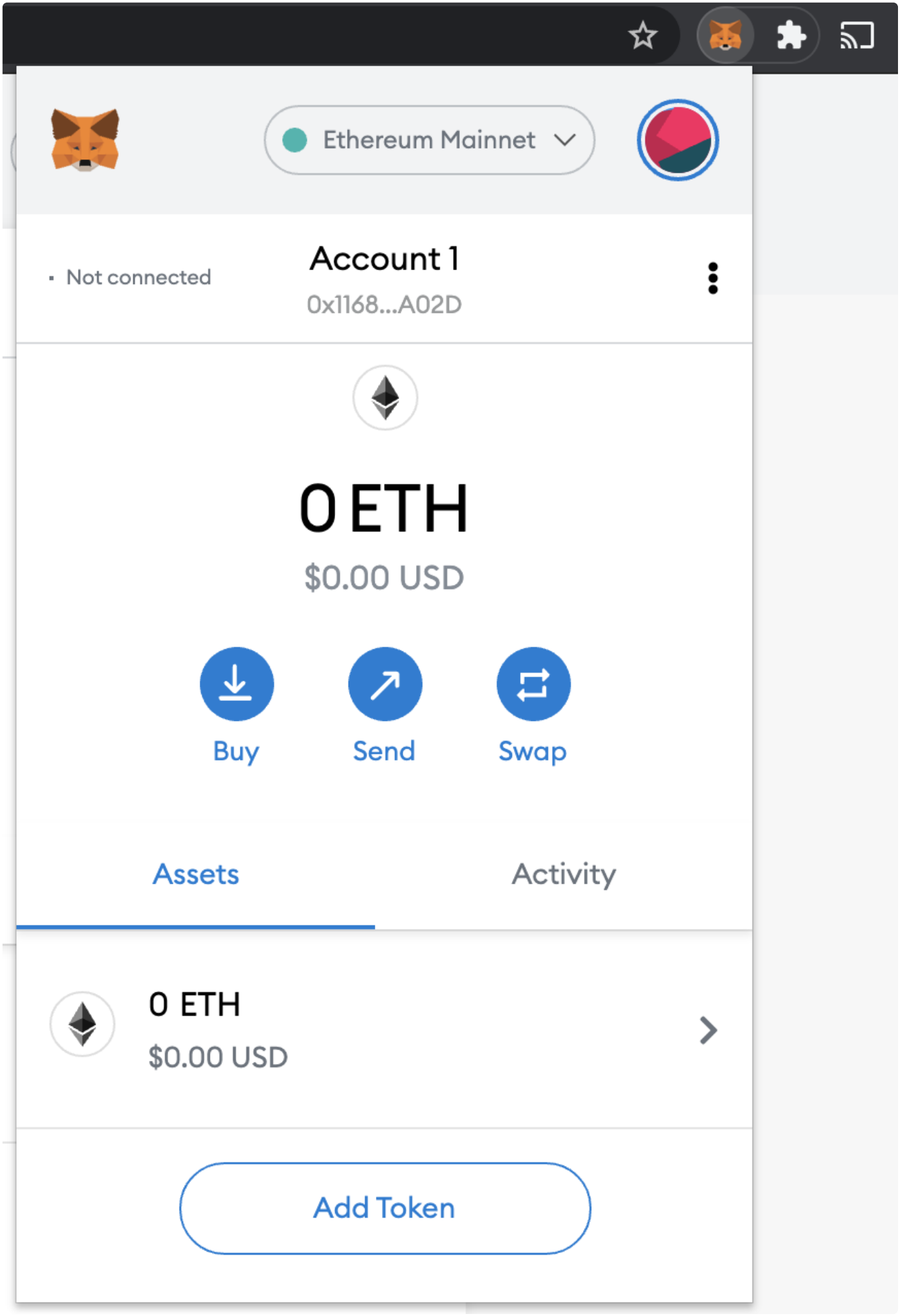
Secret Backup Phrase

- Continue forward. On the next page, MetaMask will ask you to confirm the backup phrase. Select the words in the correct sequence:



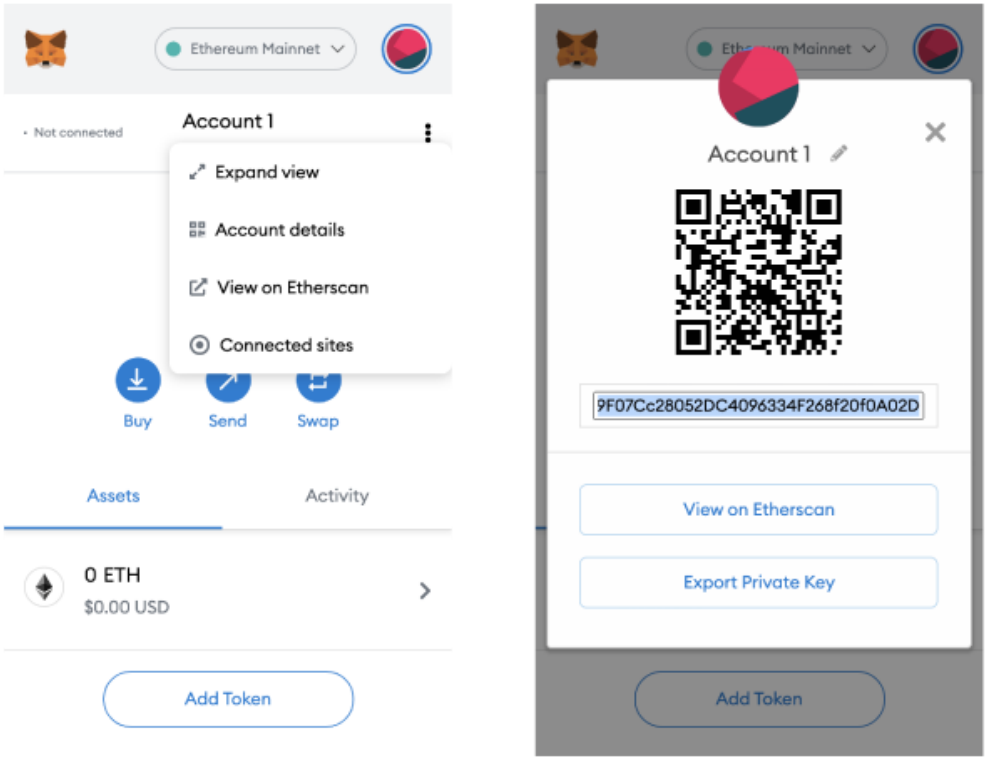
Confirm secret backup phrase

- Voila! Your account is now created. You can access MetaMask via the browser extension in the top right corner of your browser.



MetaMask browser extension

- You can now manage Ether and Ocean Tokens with your wallet. You can copy your account address to the clipboard from the options. When you want someone to send Ether or Ocean Tokens to you, you will have to give them that address. It's not a secret.



Manage tokens

You can also watch our tutorial video snippets if you want more help setting up MetaMask.

Set Up Custom Network

Sometimes it is required to use custom or external networks in MetaMask. We can add a new one through MetaMask's Settings.

Open the Settings menu and find the `Networks` option. When you open it, you'll be able to see all available networks your MetaMask wallet currently use. Click the `Add Network` button.

Ethereum Mainnet

Networks

activity. Only add custom networks you trust.

Network Name

New RPC URL

Chain ID

Currency Symbol (optional)

Block Explorer URL (optional)

Add custom/external network

There are a few empty inputs we need to fill:

- **Network Name:** this is the name that MetaMask is going to use to differentiate your network from the rest.
- **New RPC URL:** to operate with a network we need an endpoint (RPC). This can be a public or private URL.
- **Chain Id:** each chain has an Id
- **Currency Symbol:** it's the currency symbol MetaMask uses for your network
- **Block Explorer URL:** MetaMask uses this to provide a direct link to the network block explorer when a new transaction happens

When all the inputs are filled just click `Save`. MetaMask will automatically switch to the new network.

Manage Your OCEAN Tokens

How to use crypto wallet software to check your Ocean Token balance and to send Ocean Tokens to others.

If you don't see any Ocean Tokens in your crypto wallet software (e.g. MetaMask or MyEtherWallet), don't worry! It might not know how to manage Ocean Tokens yet.

Token Information

Almost all ERC-20 wallets require these values for adding a custom token:

Mainnet

- Contract Address: `0x967da4048cD07aB37855c090aAF366e4ce1b9F48`
- Symbol: `OCEAN`
- Decimals: `18`

Polygon Mainnet (previously Matic)

- Contract Address: `0x282d8efCe846A88B159800bd4130ad77443Fa1A1`
- Symbol: `mOCEAN`
- Decimals: `18`

Binance Smart Chain (BSC)

- Contract Address: `0xdce07662ca8ebc241316a15b611c89711414dd1a`
- Symbol: `OCEAN`
- Decimals: `18`

Görli

- Contract Address: `0xCfDdA22C9837aE76E0faA845354f33C62E03653a`
- Symbol: `OCEAN`
- Decimals: `18`

Mumbai

- Contract Address: `0xd8992Ed72C445c35Cb4A2be468568Ed1079357c8`
- Symbol: `OCEAN`
- Decimals: `18`

The OCEAN Token page at oceanprotocol.com has further details.

MetaMask

1. Make sure MetaMask is connected to the Ethereum Mainnet.
2. Select the account you want to manage.
3. Scroll down until the `Add Token` link is visible, then click on it.
4. Click on `Custom Token`.
5. Paste the Ocean Token contract address listed above into the *Token Contract Address* field. The other two fields should auto-fill. If not, add `OCEAN` for the symbol and `18` for the precision.
6. Click `Next`.
7. Click `Add Tokens`.

MetaMask should now show your Ocean Token (OCEAN) balance, and when you're looking at that, there should be a `Send` button to send Ocean Tokens to others. For help with that, see the MetaMask docs about how to send tokens.

Other Wallet Software

Do a web search to find out how to add a custom ERC-20 token to the wallet software you're using.

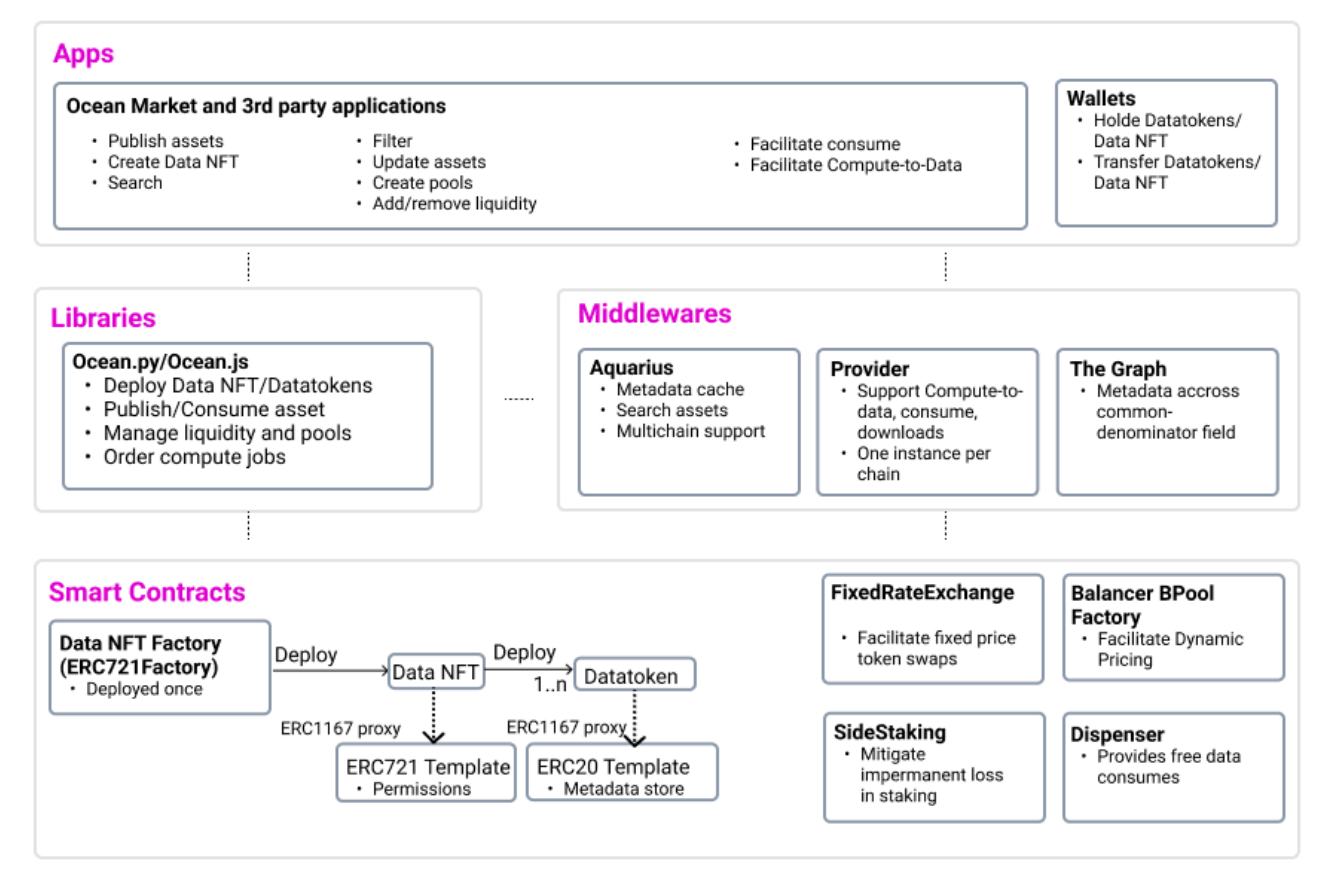
Core Concepts

Architecture Overview

Data NFTs and datatokens architecture

Overview

Here is the Ocean architecture.



Ocean Protocol tools architecture

Here's an overview of the figure.

- The top layer is **applications** like Ocean Market. With these apps, users can onboard services like data, algorithms, compute-to-data into crypto (publish and mint data NFTs and datatokens), hold datatokens as assets (data wallets), discover assets, and buy/sell datatokens for a fixed or auto-determined price (data marketplaces), and use data services (spend datatokens).
- Below are **libraries** used by the applications: Ocean.js (JavaScript library) and Ocean.py (Python library). This also includes middleware to assist discovery:
  - **Aquarius**: Provides metadata cache for faster search by caching on-chain data into elasticsearch
  - **Provider**: Facilitates downloading assets, DDO encryption, and communicating with `operator-service` for Compute-to-Data jobs.
  - **The Graph**: It is a 3rd party tool that developers can utilize the libraries to build their custom applications and marketplaces.

- The lowest level has the **smart contracts**. The smart contracts are deployed on the Ethereum mainnet and other compatible networks. Libraries encapsulate the calls to these smart contracts and provide features like publishing new assets, facilitating consumption, managing pricing, and much more. To see the supported networks click here.

Data NFTs, Datatokens and Access Control Tools

Data NFTs are based on ERC721 standard. The publisher can use Marketplace or client libraries to deploy a new data NFT contract. To save gas fees, it uses ERC1167 proxy approach on the **ERC721 template**. Publisher can then assign manager role to other Ethereum addresses who can deploy new datatoken contracts and even mint them. Each datatoken contract is associated with one data NFT contract. Click here to further read about data NFTs and datatokens.

ERC721 data NFTs represent holding copyright/base IP of a data asset, and ERC20 datatokens represent licenses to access the asset by downloading the content or running Compute-to-Data jobs.

Datatoken represents the asset that the publisher wants to monetize. The asset can be a dataset or an algorithm. The publisher actor holds the asset in Google Drive, Dropbox, AWS S3, on their phone, on their home server, etc. The publisher can optionally use IPFS for a content-addressable URL. Or instead of a file, the publisher may run a compute-to-data service.

In the **publish** step, the publisher invokes **Ocean Datatoken Factory** to deploy a new datatoken to the chain. To save gas fees, it uses ERC1167 proxy approach on the **ERC20 datatoken template**. The publisher then mints datatokens.

The publisher runs their own **Ocean Provider** or can use one deployed by Ocean Protocol. In the **download** step or while running C2D job, Provider software needs to retrieve the data service URL given a datatoken address. One approach would be for the publisher to run a database. However, this adds another dependency. To avoid this, the Provider encrypts the URL, which then gets published on-chain.

To initiate the **download** step, the data buyer sends 1.0 datatokens to the Provider wallet. Then they make a service request to the Provider. The Provider loads the encrypted URL, decrypts it, and provisions the requested service (send static data, or enable a compute-to-data job).

Instead of running a Provider themselves, the publisher can have a 3rd party like Ocean Market to run it. While more convenient, it means that the 3rd party has custody of the private encryption/decryption key (more centralized). Ocean will support more service types and URL custody options in the future.

Ocean JavaScript and Python libraries act as drivers for the lower-level contracts. Each library integrates with Ocean Provider to provision & access data services, and Ocean Aquarius for metadata.

Market Tools

Once someone has generated datatokens, they can be used in any ERC20 exchange, centralized or decentralized. In addition, Ocean provides a convenient default marketplace that is tuned for data: **Ocean Market**. It's a vendor-neutral reference data marketplace for use by the Ocean community.

The marketplaces are decentralized (no single owner or controller), and non-custodial (only the data owner holds the keys for the datatokens).

Ocean Market supports fixed pricing or free pricing. For more detials on pricing schema refer this guide.

Complementary to Ocean Market, Ocean has reference code to ease building **third-party data marketplaces**, such as for logistics (dexFreight data marketplace) or mobility (Daimler).

This post elaborates on Ocean marketplace tools.

Metadata Tools

Marketplaces use the Metadata of the asset for discovery. Metadata consists of information like the type of asset, name of the asset, creation date, license, etc. Each data asset can have a decentralized identifier (DID) that resolves to a DID document (DDO) for associated metadata. The DDO is essentially JSON filling in metadata fields. For more details on working with OCEAN DIDs check out the DID concept documentation. The DDO Metadata documentation goes into more depth regarding metadata structure.

OEP8 specifies Ocean metadata schema, including fields that must be filled. It's based on the public DataSet schema from schema.org.

Ocean uses the Ethereum mainnet and other compatible networks as an **on-chain metadata store**, i.e. to store both DID and DDO. This means that once the transaction fee is paid, there are no further expenses or devops work needed to ensure metadata availability into the future, aiding in the discoverability of data assets. It also simplifies integration with the rest of the Ocean system, which is Ethereum-based. Storage cost on Ethereum mainnet is not negligible, but not prohibitive and the other benefits are currently worth the trade-off compared to alternatives.

Due to the permissionless, decentralized nature of data on the Ethereum mainnet, any last mile tool can access metadata. **Ocean Aquarius** supports different metadata fields for each different Ocean-based marketplace. Developers could also use The Graph to see metadata fields that are common across all marketplaces.

Third-Party ERC20 Apps & Tools

The ERC20 nature of datatokens eases composability with other Ethereum tools and apps, including **MetaMask** and **Trezor** as data wallets, DEXes as data exchanges, and more. This post has details.

Actor Identities

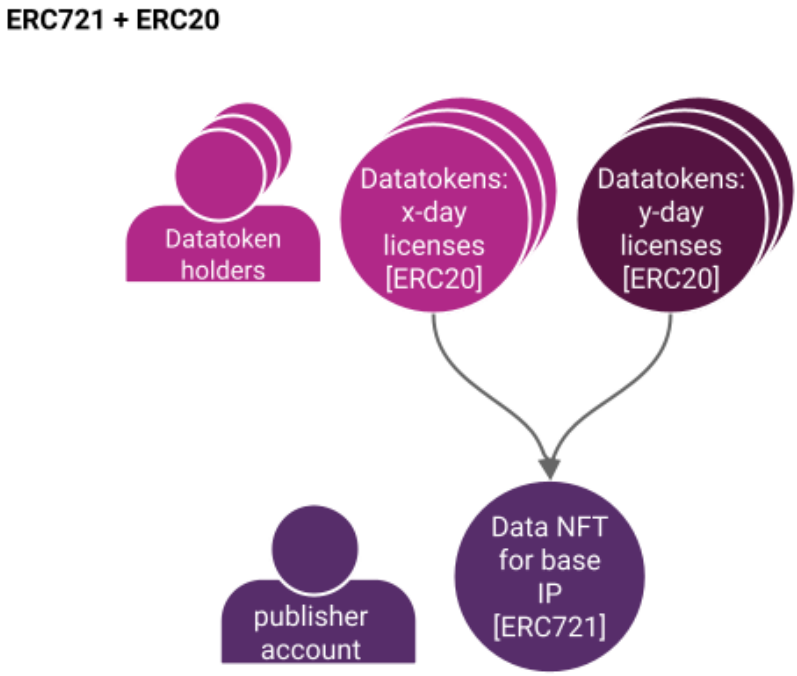
Actors like data providers and buyers have Ethereum addresses, aka web3 accounts. These are managed by crypto wallets, as one would expect. For most use cases, this is all that's needed. There are cases where the Ocean community could layer on protocols like Verifiable Credentials or tools like 3Box.

Data NFTs and Datatokens

In Ocean Protocol, ERC721 data NFTs represent holding copyright/base IP of a data asset, and ERC20 datatokens represent licenses to access the assets.

A non-fungible token stored on the blockchain represents a unique asset. NFTs can represent images, videos, digital art, or any piece of information. NFTs can be traded, and allow transfer of copyright/base IP. EIP-721 defines an interface for handling NFTs on EVM-compatible blockchains. The creator of the NFT can deploy a new contract on Ethereum or any Blockchain supporting NFT related interface and also, transfer the ownership of copyright/base IP through transfer transactions.

Fungible tokens represent fungible assets. If you have 5 ETH and Alice has 5 ETH, you and Alice could swap your ETH and your final holdings remain the same. They're apples-to-apples. Licenses (contracts) to access a copyrighted asset are naturally fungible - they can be swapped with each other.



Data NFT and datatoken

What is a Data NFT?

A data NFT represents the copyright (or exclusive license against copyright) for a data asset on the blockchain — we call this the “base IP”. When a user publishes a dataset in OceanOnda V4, they create a new NFT as part of the process. This data NFT is proof of your claim of base IP. Assuming a valid claim, you are entitled to the revenue from that asset, just like a title deed gives you the right to receive rent.

The data NFT smart contract holds metadata about the data asset, stores roles like “who can mint datatokens” or “who controls fees”, and an open-ended key-value store to enable custom fields.

If you have the private key that controls the NFT, you are the owner of that NFT. The owner has the claim on

the base IP and is the default recipient of any revenue. They can also assign another account to receive revenue. This enables the publisher to sell their base IP and the revenues that come with it. When the Data NFT is transferred to another user, all the information about roles and where the revenue should be sent is reset. The default recipient of the revenue is the new owner of the data NFT.

Data NFTs Open Up New Possibilities

With data NFTs, you are able to take advantage of the wider NFT ecosystem and all the tools and possibilities that come with it. As a first example, many leading crypto wallets have first-class support for NFTs, allowing you to manage data NFTs from those wallets. Or, you can post your data NFT for sale on a popular NFT marketplace like OpenSea or Rarible. As a final example, we're excited to see data NFTs linked to physical items via WiseKey chips.

High-Level Architecture

The image above describes how ERC721 data NFTs and ERC20 datatokens relate.

- Bottom: The publisher deploys an ERC721 data NFT contract representing the base IP for the data asset. They are now the manager of the data NFT.
- Top: The manager then deploys an ERC20 datatoken contract against the data NFT. The ERC20 represents a license with specific terms like "can download for the next 3 days". They could even publish further ERC20 datatoken contracts, to represent different license terms or for compute-to-data.

Terminology

- **Base IP** means the artifact being copyrighted. Represented by the {ERC721 address, tokenId} from the publish transactions.
- **Base IP holder** means the holder of the Base IP. Represented as the actor that did the initial "publish" action.
- **Sub-licensee** is the holder of the sub-license. Represented as the entity that controls address ERC721._owners[tokenId=x].
- **To Publish**: Claim copyright or exclusive base license.
- **To Sub-license**: Transfer one (of many) sub-licenses to new licensee: ERC20.transfer(to=licensee, value=1.0).

Implementation in Ocean Protocol

We have implemented data NFTs using the ERC721 standard. Ocean Protocol defines the ERC721Factory contract, allowing **Base IP holders** to create their ERC721 contract instances on any supported networks. The deployed contract stores Metadata, ownership, sub-license information, permissions. The contract creator can also create and mint ERC20 token instances for sub-licensing the **Base IP**.

ERC721 tokens are non-fungible, thus cannot be used for automatic price discovery like ERC20 tokens.

ERC721 and ERC20 combined together can be used for sub-licensing. Ocean Protocol's ERC721Template solves this problem by using ERC721 for tokenizing the **Base IP** and tokenizing sub-licenses by using ERC20.

Our implementation has been built on top of the battle-tested OpenZeppelin contract library. However, there are a bunch of interesting parts of our implementation that go a bit beyond an out-of-the-box NFT.

OceanOnda V4's data NFT factory can deploy different types of data NFTs based on a variety of templates. Some templates could be tuned for data unions, others for DeFi, and others yet for enterprise use cases.

Something else that we're super excited about in our data NFTs is a cutting-edge standard called ERC725 being driven by our friends at Lukso. The ERC725y feature enables the NFT owner (or a user with the "store updater" role) to input and update information in a key-value store. These values can be viewed externally by anyone.

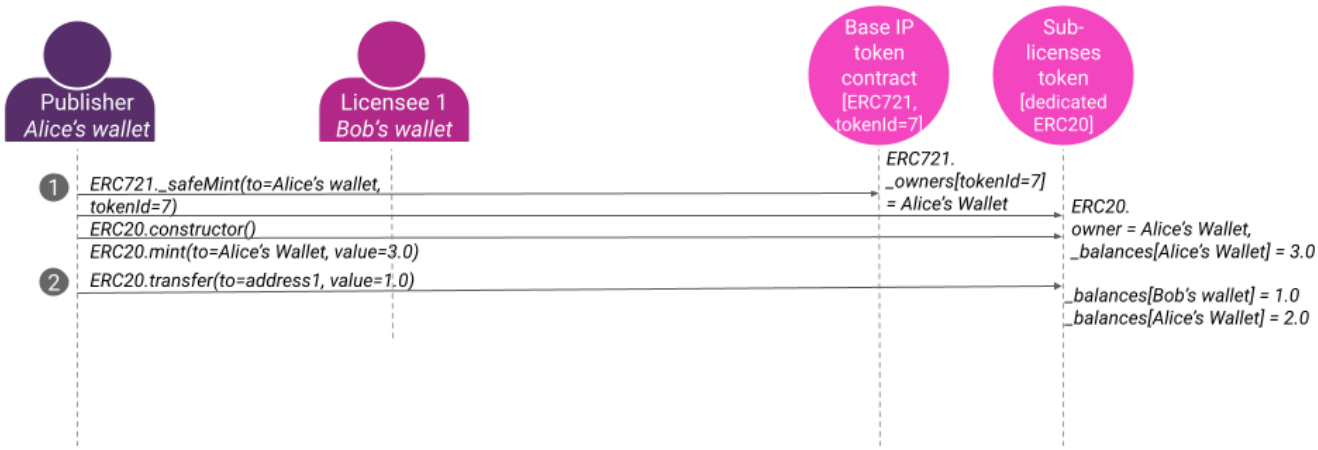
ERC725y is incredibly flexible and can be used to store any string; you could use it for anything from additional metadata to encrypted values. This helps future-proof the data NFTs and ensure that they are suitable for a wide range of projects that have not been launched yet. As you can imagine, the inclusion of ERC725y has huge potential and we look forward to seeing the different ways people end up using it. If you're interested in using this, take a look at EIP725.

Continuing the theme of flexibility, for a given data NFT, you can have one or more ERC20 datatoken contracts. Here's the main idea: 1.0 datatokens allows you to consume the corresponding dataset. Put another way, it's a sub-license from the base IP to be able to use the dataset according to the license terms (when you send it to the publisher). License terms can be set from a "good default", or by the Data NFT owner. ERC20 fungible token standard is a natural choice for datatokens, because licenses themselves are fungible: one license can be exchanged 1:1 another. Using the ERC20 standard enables interoperability of datatokens with ERC20-based wallets, DEXes, DAOs, and more. Datatokens can be given (simply transferred), purchased on a marketplace / exchange, airdropped, etc.

You can publish a data NFT initially with no ERC20 datatoken contracts. This means you simply aren't ready to grant access to your data asset yet (sub-license it). Then, you can publish one or more ERC20 datatoken contracts against the data NFT. One datatoken contract might grant consume rights for 1 day, another for 1 week, etc. Each different datatoken contract is for different license terms.

Ocean provides convenient methods to list ERC20 datatokens for sale, with fixed-price (atomic swap), or for free. Like any ERC20 token, datatokens may be listed in many decentralised exchanges (DEXes), centralised exchanges (CEXes), over-the-counter, or otherwise.

High-Level Behavior



Flow

Here's an example.

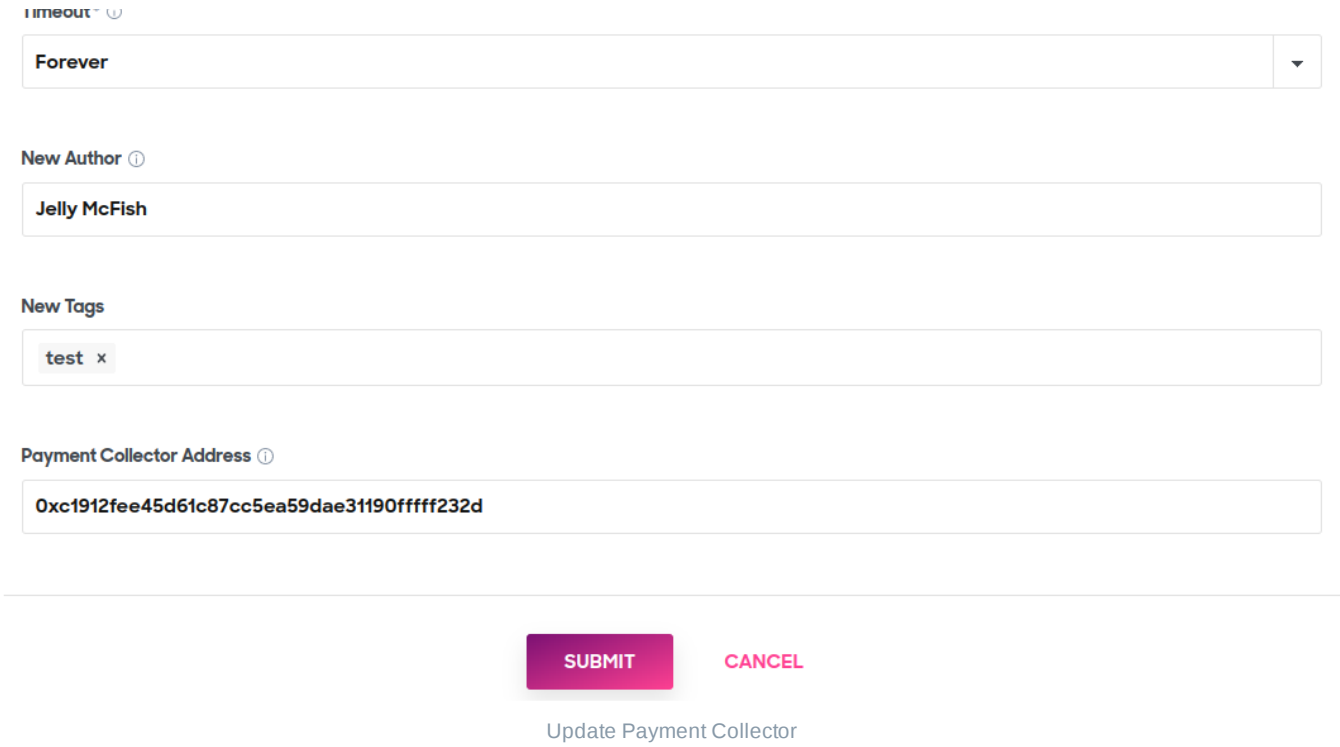
- In step 1, Alice **publishes** her dataset with Ocean: this means deploying an ERC721 data NFT contract (claiming copyright/base IP), then an ERC20 datatoken contract (license against base IP).
- In step 2, she **mints** some ERC20 datatokens and **transfers** 1.0 of them to Bob's wallet; now he has a license to be able to download that dataset.

Revenue

By default revenue is sent to the owner of the data NFT and this automatically updates when the data NFT is sent to a new owner. Owning a data NFT therefore has an explicit value - the revenue stream associated with selling the digital asset.

In some situations, you may want the revenue to be sent to another account rather than the owner. This can be done by setting a new **payment collector**. Changing the payment collector can be particularly useful when the data NFT is owned by an organization or enterprise, rather than an individual.

In order to set a new payment collector, you need to visit the asset detail page and then click on “Edit Asset” and then scroll down to the field call “Payment Collector Address”. Add the new Ethereum address in this field and then click “Submit“. Finally, you will then need to sign two transactions to finalize the update.



TemplateIds

Each data NFT or a datatoken is cloned from pre-defined template contracts. The *templateId* parameter refers to the template from which a data NFT or datatoken is created. The templateId can be set while creating data NFT/datatoken. The templateId is stored in the code of the smart contract and can be retrived using `getId()` function. Currently, Ocean protocol supports 1 template type for data NFT and 2 template variants for datatokens, namely: **regular template** and **enterprise template**. Each template supports the same interfaces but differs in the underlying implementation and can have additional features.

The only data NFT template currently available has templateId `1` and the source code is available here.

The details regarding currently supported datatoken templates are as follows:

- **Regular template**: The regular template allows users to buy/sell/hold datatokens. The datatokens can be minted by the address having a `MINTER` role, making the supply of datatoken variable. This template is assigned templateID `1` and the source code is available here.
- **Enterprise template**: The enterprise template has additional functions apart from methods in the ERC20 interface. This additional feature allows access to the service by paying in the basetoken instead of datatoken. Internally, the smart contract handles conversion of basetoken to datatoken, initiating an order to access the service, and minting/burning the datatoken. The total supply of the datatoken effectively remains 0 in the case of the enterprise template. This template is assigned templateID `2` and the source code is available here.

NOTE: Ocean Protocol might support additional variations of data NFT/datatoken by adding new templates.

Fractional Ownership

Fractional ownership is an exciting sub-niche of Web3, at the intersection of NFTs and DeFi. IT allows co-

ownership of data IP.

Ocean provides two approaches to fractional ownership:

- Sharded holding of ERC20 datatokens, where each ERC20 holder has the usual datatoken rights as described above, e.g. 1.0 datatokens to consume an asset. This comes out-of-the-box with Ocean.
- Sharding ERC721 data NFT, where each co-holder has right to some earnings against base IP, and co-controls the data NFT. For example, there's a DAO with the sole purpose to hold the data NFT; this DAO has its own ERC20 token; DAO members vote with tokens to update data NFT roles or deploy ERC20 datatokens against the ERC721.

Note: For (2), one might consider doing sharding with something like Niftex. But then there are questions: what rights do the shard-holders get exactly? It could be zero; for example, Amazon shareholders don't have the right to walk the hallways of the Amazon offices just because they hold shares. Secondly, how do the shard-holders control the data NFT? These questions get resolved by using a tokenized DAO, as described above.

Data DAOs are a cool use case whenever you have a group of people that wish to co-manage data, or bundle up data for larger collective bargaining power. The DAO may be a union, co-op, or trust.

Consider the following mobile app example. You install the app; it has a built-in crypto wallet; you give the app permission to see your location data; the app gets the DAO to sell your (anonymized) location data on your behalf; the DAO sells your data bundled along with thousands of other DAO members; as a DAO member you get a cut of the profits.

This has several variants. Each member's data feed could be its own data NFT with associated datatokens. Or, there's simply one data NFT aggregating datafeeds across all members into a single feed, and the feed is fractionalized by sharded holding of ERC20 tokens (1 above) or sharding the ERC721 data NFT (2 above). If you're interested in starting a data union then we recommend getting in touch with our friends at Data Union.

Other References

- Data & NFTs 1: Practical Connections of ERC721 with Intellectual Property
- Data & NFTs 2: Leveraging ERC20 Fungibility
- Data & NFTs 3: Combining ERC721 & ERC20
- Fungibility sightings in NFTs

Roles

The permissions stored on chain in the contracts control the access to the data NFT (ERC721) and datatoken (ERC20) smart contract functions.

The permissions are stored in the data NFT (ERC721) smart contract. The data NFT (ERC721) and datatoken (ERC20) smart contracts both use this information to restrict access to the smart contract functions. The tables below list restricted actions that are accessible only to the allowed users.

What Roles Can The Data NFT Owner Assign?

The data NFT is the base IP for the asset and all the datatokens are therefore linked to the data NFT smart contract — this has enabled us to do a bunch of cool new things around role administration. We've introduced a host of useful roles which give you flexibility in how you manage your project. This can be a big help for enterprises and startups who are ready to scale up and introduce a level of administration.

NFT Owner

The NFT owner is the owner of the base-IP and is therefore at the highest level. The NFT owner can perform any action or assign any role but crucially, the NFT owner is the only one who can assign the manager role. Upon deployment or transfer of the data NFT, the NFT owner is automatically added as a manager. The NFT owner is also the only role that can't be assigned to multiple users — the only way to share this role is via multi-sig or a DAO.

Manager

The manager can assign or revoke three main roles (deployer, metadata updater, store updater). The manager is also able to interact with the ERC725 data.

ERC20 Deployer

The Deployer has a bunch of privileges at the ERC20 datatoken level. They can deploy new datatokens with fixed price exchange, or free pricing. They can also update the ERC725Y key-value store and assign roles the ERC20 level.

Metadata Updater

There is also a specific role for updating the metadata. The Metadata updater has the ability to update the information about the data asset (title, description, sample data etc) that is displayed to the user on the asset detail page within the market.

Store Updater

The store updater can store, remove or update any arbitrary key value using the ERC725Y implementation (at the ERC721 level). The use case for this role depends a lot on what data is being stored in the ERC725Y key-value pair — as mentioned above, this is highly flexible.

Minter

The Minter has the ability to mint new datatokens, provided the limit has not been exceeded. In most cases, this role will not be used as the alternative is for the datatokens to be minted by the side-staking bot which has many advantages. We highly recommend taking a read of this article if you're interested in learning more about safer staking and one-sided staking.

Fee Manager

Finally, we also have a fee manager which has the ability to set a new fee collector — this is the account that will receive the datatokens when a data asset is consumed. If no fee collector account has been set, the datatokens will be sent by default to the NFT Owner. The applicable fees (market and community fees) are automatically deducted from the datatokens that are received.

Roles in data NFT (ERC721) smart contract

Action ↓ / Role →	NFT Owner	Manager	ERC20 Deployer	Store Updater	Metadata Updater
Set token URI					
Add manager	✓				
Remove manager	✓				
Clean permissions	✓				
Set base URI	✓				
Set Metadata state					✓
Set Metadata					✓
Create new datatoken			✓		
Executes any other smart contract		✓			
Set new key-value in store				✓	

Roles in datatoken (ERC20) smart contract

Action ↓ / Role →	ERC20 Deployer	Minter	NFT owner	Fee manage
Create Fixed Rate exchange	✓			
Create Dispenser	✓			
Add minter	✓			
Remove minter	✓			
Add fee manager	✓			
Remove fee manager	✓			
Set data	✓			
Clean permissions			✓	
Mint		✓		
Set fee collector				✓

Networks

All the public networks the Ocean Protocol contracts are deployed to, and additional core components deployed to them.

Ocean Protocol contracts are deployed on multiple public networks. You can always find the most up-to-date deployment addresses for all individual contracts in the address.json.

In each network, you'll need ETH to pay for gas, and OCEAN for certain Ocean actions. Because the Ethereum mainnet is a network for production settings, ETH and OCEAN tokens have real value on there. The ETH and OCEAN tokens in each test network don't have real value and are used for testing-purposes only. They can be obtained with *faucets* to dole out ETH and OCEAN.

The universal Aquarius Endpoint is `https://v4.aquarius.oceanprotocol.com`.

Ethereum Mainnet

Ethereum mainnet is a production network. In MetaMask and other ERC20 wallets, click on the network name dropdown, then select *Ethereum mainnet*.

Tokens

- Mainnet ETH:
  - Native token to pay transaction fees
- Mainnet OCEAN:
  - Address: 0x967da4048cD07aB37855c090aAF366e4ce1b9F48

Additional Components

What	URL
Explorer	https://etherscan.io
Ocean Market	Point wallet to Ethereum Mainnet network, at https://v4.market.oceanprotocol.com/
Provider	`https://v4.provider.mainnet.oceanprotocol.com`
Subgraph	`https://v4.subgraph.mainnet.oceanprotocol.com`

Polygon Mainnet

Ocean is deployed to Polygon Mainnet, another production network. Polygon's native token is MATIC. If you don't find Polygon as a predefined network in your wallet, you can connect to it manually via Polygon's guide here.

Tokens

- Matic:
  - Native token to pay transaction fees
- Matic OCEAN:
  - Address: 0x282d8efCe846A88B159800bd4130ad77443Fa1A1

Additional Components

What	URL
Explorer	https://polygonscan.com
Ocean Market	Point wallet to Ploygon Mainnet network, at https://v4.market.oceanprotocol.com/
Provider	`https://v4.provider.polygon.oceanprotocol.com/`
Subgraph	`https://v4.subgraph.polygon.oceanprotocol.com/`

Bridge

Check our Polygon Bridge guide to learn how you can deposit, withdraw and send tokens.

Binance Smart Chain

Ocean is deployed to Binance Smart Chain (BSC), another production network. BSC's native token is BNB - the Binance token.

If you don't find BSC as a predefined network in your wallet, you can connect to it manually via Binance's guide here.

Tokens

- BSC BNB:
  - Native token to pay transaction fees.
- BSC OCEAN:
  - Address: 0xdce07662ca8ebc241316a15b611c89711414dd1a

Additional Components

What	URL
Explorer	https://bscscan.com/
Ocean Market	Point wallet to Binance Smart Chain network, at https://v4.market.oceanprotocol.com/
Provider	`https://v4.provider.bsc.oceanprotocol.com`
Subgraph	`https://v4.subgraph.bsc.oceanprotocol.com`

Bridge

Check our BSC Bridge guide to learn how you can deposit, withdraw and send tokens.

Energy Web Chain

Ocean is deployed to Energy Web Chain, another production network. Energy Web's native token is EWT.

If you don't find Energy Web Chain as a predefined network in your wallet, you can connect to it using the guide here.

Tokens

- Energy Web Chain EWT:
  - Native token to pay transaction fees.
- Energy Web Chain OCEAN:
  - Address: 0x593122aae80a6fc3183b2ac0c4ab3336debee528

Additional Components

What	URL
Explorer	https://explorer.energyweb.org/
Ocean Market	Point wallet to Energy Web Chain network, at https://v4.market.oceanprotocol.com/
Provider	`https://v4.provider.energyweb.oceanprotocol.com/`
Subgraph	`https://v4.subgraph.energyweb.oceanprotocol.com`

Bridge

Use the link here to bridge the assets between EWC and Ethereum mainnet.

Moonriver

Ocean is deployed to Moonriver, another production network. Moonriver's native token is MOVR.

If you don't find Moonriver as a predefined network in your wallet, you can connect to it using the guide here.

Tokens

- Moonriver MOVR:
  - Native token to pay transaction fees.
- Moonriver OCEAN:
  - Address: 0x99C409E5f62E4bd2AC142f17caFb6810B8F0BAAE

Additional Components

What	URL
Explorer	https://blockscout.moonriver.moonbeam.network
Ocean Market	Point wallet to Moonriver network, at https://v4.market.oceanprotocol.com/
Provider	`https://v4.provider.moonriver.oceanprotocol.com`
Subgraph	`https://v4.subgraph.moonriver.oceanprotocol.com`

Bridge

Use Anyswap to bridge between ETH Mainnet and Moonriver.

Görli

Görli is a test network.

In MetaMask and other ERC20 wallets, click on the network name dropdown, then select *Goerli*.

Tokens

- Görli ETH:
  - Native token to pay transaction fees
  - Faucet. You may find others by searching.
- Goerli OCEAN:
  - Address: 0xCfDdA22C9837aE76E0faA845354f33C62E03653a
  - Faucet

Additional Components

What	URL
Explorer	https://goerli.etherscan.io/
Ocean Market	Point wallet to Görli network, at https://market.oceanprotocol.com
Provider	`https://v4.provider.goerli.oceanprotocol.com`
Subgraph	`https://v4.subgraph.goerli.oceanprotocol.com`

Mumbai

Mumbai is a test network tuned for Matic / Polygon.

If you don't find Mumbai as a predefined network in your wallet, you can connect to it manually via Matic's guide.

Tokens

- Mumbai MATIC:
  - Native token to pay transaction fees
  - Faucet. You may find others by searching.
- Mumbai OCEAN:
  - Address: 0xd8992Ed72C445c35Cb4A2be468568Ed1079357c8
  - Faucet

Additional Components

What	URL
Explorer	https://mumbai.polygonscan.com
Ocean Market	Point wallet to Mumbai network, at https://market.oceanprotocol.com
Provider	`https://v4.provider.mumbai.oceanprotocol.com`
Subgraph	`https://v4.subgraph.mumbai.oceanprotocol.com`

Local / Ganache

The most straightforward way for local-only development is to use Barge, which runs Ganache, Aquarius, and Provider. It is used extensively by the Ocean core devs and for automated integration testing.

To connect to it from MetaMask, select the network called *Localhost 8545*.

Alternatively, you can run Ganache independently. Install it according to the Ganache docs. Then deploy Ocean contracts onto Ganache following docs in Ocean contracts repo. Ganache is at the RPC URL http://localhost:8545.

Tokens

- Ganache ETH:
  - Native token to pay transaction fees
  - By default, Ganache creates several Ethereum accounts at launch, gives each some ETH, and makes their private keys available in the logs. You can also instruct Ganache to give ETH to specific Ethereum addresses.
- Ganache OCEAN:
  - You can deploy an ERC20 token with label OCEAN. At a minimum, the token needs to be ERC20Detailed and ERC20Capped. You'll see examples in the quickstarts for the Ocean JavaScript and Python drivers.

Other

Some apps may need `network_id` and `chain_id`. Here's a list of values for major Ethereum networks.

Bridges

Binance Smart Chain (BSC) Bridge

Intro to BSC's Bridge

BSC provides several bridge options, including:

- withdraw crypto from Binance.com, and
- use Binance bridge.

The article How to Get Started with BSC by Binance Academy provides further details.

Links

- BSC Wallet Support. Includes MetaMask and Trust Wallet.
- BSC Bridge
- How to set up a custom network in MetaMask

Polygon (ex Matic) Bridge

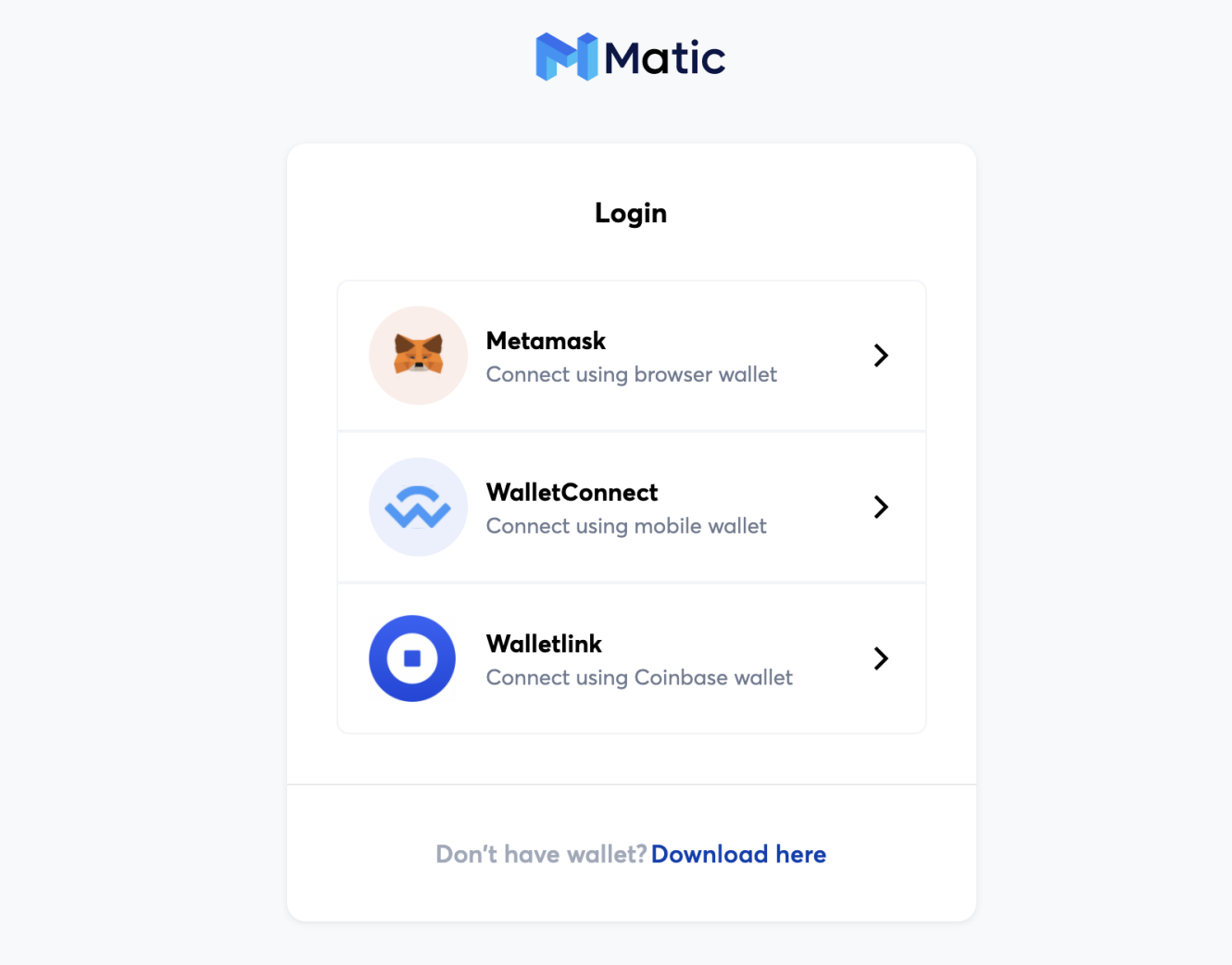
Links

- Matic Wallet
- Matic Bridge
- How to set up a custom network in MetaMask

Intro to Polygon's Bridge

The Polygon Network (ex Matic) provide us with a bridge (connecting Ethereum & Polygon blockchains), and a dedicated wallet that simplify the steps of transferring digital assets between the two networks. The wallet connects to your account via Metamask (or any of the other supported wallets).

When you open the wallet link, the wallet will ask to log in. Select your preferred way of connecting and confirm the action. In our guide we'll use Metamask.



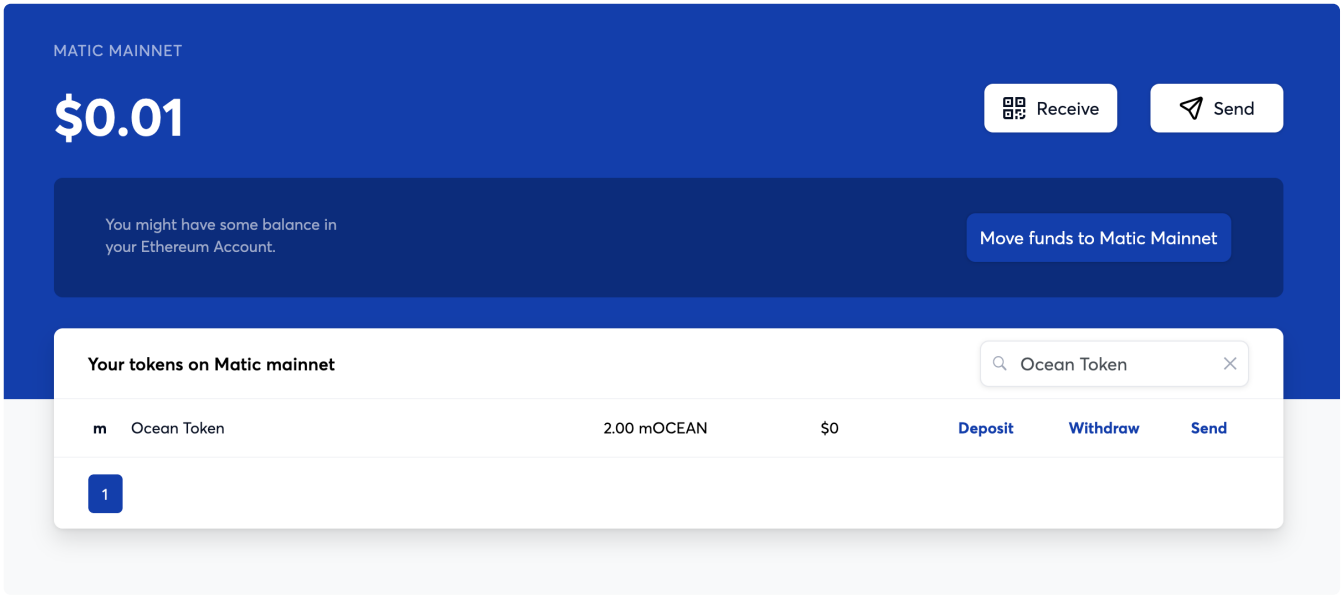
Login options

In some places, the Polygon Network is still using its old brand Matic. Either you run into Matic or Polygon - it's the exact same thing. For the purpose of this guide, we'll use Matic in the next few paragraphs since the interfaces you're going to use still use the old brand.

For details check the blog post.

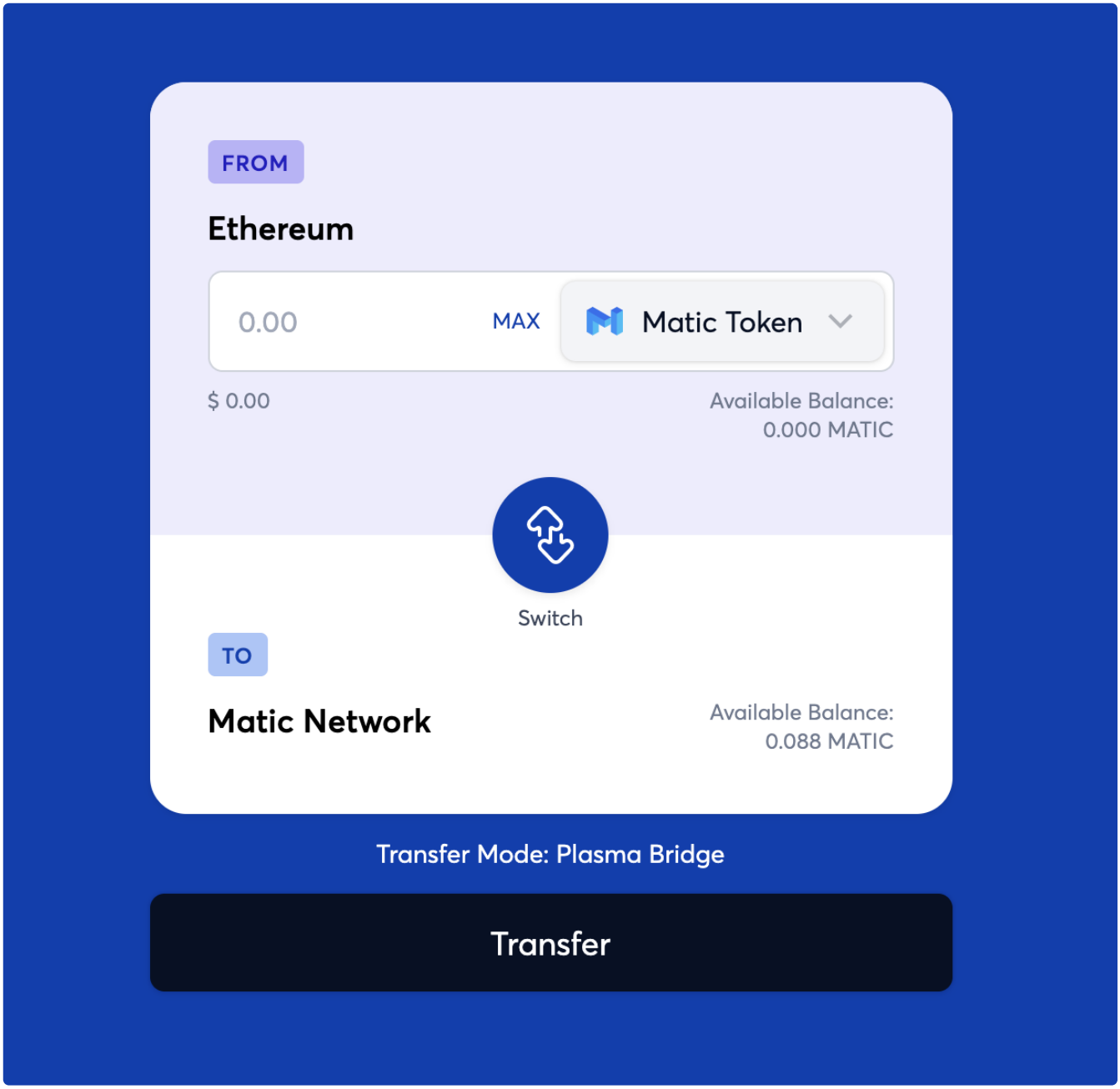
Deposit Tokens

On the main page of the wallet, you can see all tokens you own on the Matic Mainnet. To deposit tokens (transfer them from the Ethereum Mainnet) you can either use the “deposit” button for a selected token or use “Move funds to Matic Mainnet”.



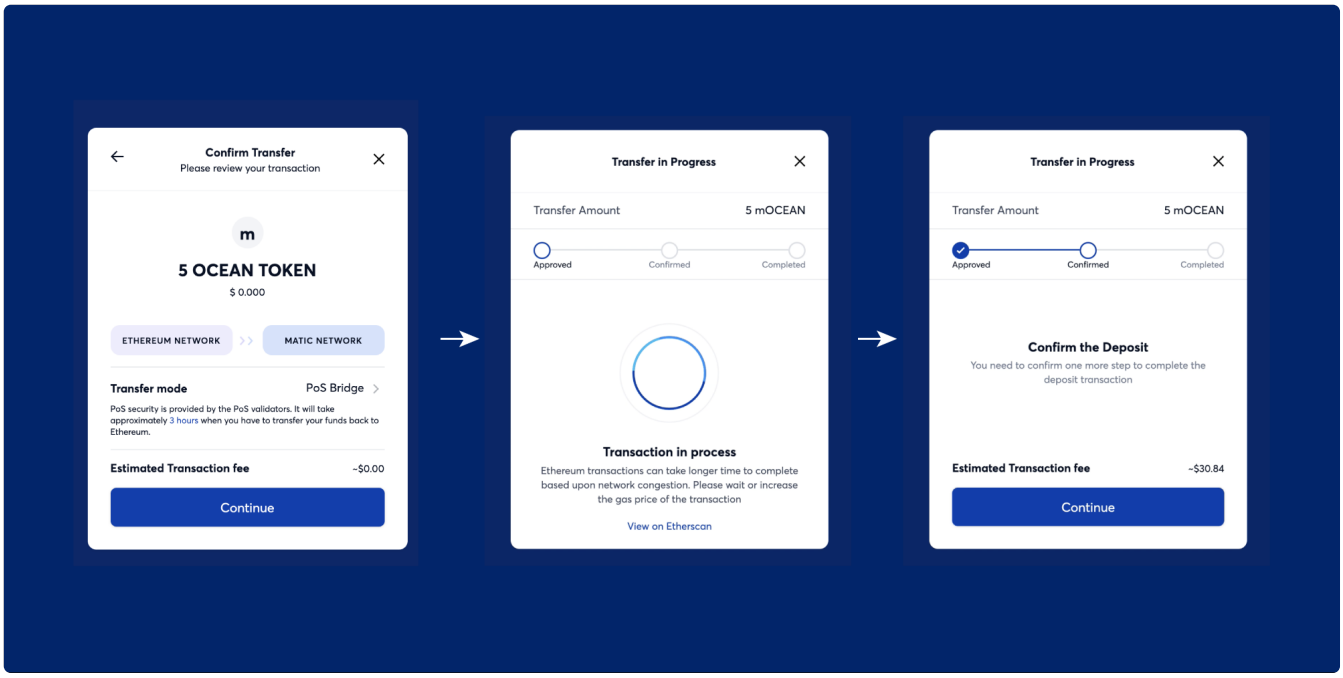
Main wallet page

Both options will redirect you to the bridge interface. In case you chose the second one, use the dropdown and select the token you want to transfer from the Ethereum Mainnet.



Bridge interface

Choose the amount to transfer and click the “Transfer” button. Matic’s bridge interface will guide you through the whole process and the different steps that will occur. You’ll need to sign two transactions on the Ethereum Mainnet. The first being the approval for your token to be traded on the Matic’s bridge and the second one being the deposit.



Transferring process

Withdraw Tokens

The withdrawing tokens process uses the same bridge interface. The only difference being that the withdraw happens from the Matic to the Ethereum Mainnet.

Again the bridge interface will guide you through the different steps.

For more in dept explanation of the deposit and withdraw actions check the official Polygon (ex Matic) docs.

Sending Tokens

While in the first two cases, the transactions are signed on the Ethereum Mainnet, transferring tokens between two Matic addresses happens on the Matic Mainnet. Thus it's required for you to connect to the Matic network to sign the transactions. You can use the following parameters to set the network in Metamask:

What	Value
Network Name	`Matic Mainnet`
RPC	`https://polygon-rpc.com/`
Chain Id	`137`
Currency Symbol	`MATIC`
Block Explorer URL	`https://explorer.matic.network/`

Follow our guide to learn how to use those values to set up a custom network in MetaMask.

Fees

The Ocean Protocol defines various fees for creating a sustainability loop.

Path to sustainability

Ocean Protocol achieves sustainability via the Web3 sustainability loop.

- The project grows and improves through the efforts of OceanDAO grant recipients.
- The OceanDAO votes to decide which proposals receive grants.
- Grant funds are sourced from the Ocean Protocol community treasury.
- The Ocean Protocol community collects fees when users interact with the protocol, thus completing the sustainability loop.

Fee types

Swap fee

Swap fees are collected whenever someone swaps a datatoken for base token (e.g., OCEAN) or base token for a datatoken. The swap can be conducted using a a fixed-rate exchange. These are the fees that are applied whenever a user swaps base token or datatoken:

- Publisher Marketplace swap fee
- Consumer Marketplace swap fee
- Provider Consumption Fees
- Ocean Community Fee

Publish fee

Publish fees can be charged to a publisher when they publish an asset.

Currently, the Ocean marketplace does not charge a publishing fee. Custom marketplaces can charge a publishing fee by adding an extra transaction in the publish flow.

Based on the use case of the marketplace, the marketplace owner can decide if this fee should be charged or not.

Consume fee

Consume fees (aka. Order fees) are charged when a user holding a datatoken exchanges it for the right to download an asset or to start a compute job that uses the asset.

These are the fees that are applied whenever a user pays to access an asset:

- Consume Market Consumption Fee
- Publisher Market Consumption Fee
- Provider Consumption Fees
- Ocean Community Fee

Ocean Community fee

Ocean's smart contracts collect **Ocean Community fees** during swap and order operations. These fees are reinvested in community projects via OceanDAO and other initiatives.

For swaps involving approved base tokens like OCEAN and H2O, the Ocean Community swap fee is 0.1%. For swaps involving other base tokens, the Ocean Community swap fee is 0.2%. The Ocean Community order fee is 0.03 DT per order operation.

These fees can be updated by the Ocean Protocol Foundation.

Provider fee

Provider is a component of Ocean Protocol's ecosystem that facilitates data consumption, starts compute jobs, encrypts DDOs, and decrypts DDOs. Provider also validates if the user can access a particular data asset or service. To learn more about Provider, click here.

Provider fees are paid to the individual or organization running their Provider instance when the user orders an asset. These fees can be set to an absolute amount, not as a percentage. The provider can also specify which token the fees must be paid in - they don't have to be the same token used in the consuming market.

Provider fees can also be used to charge for computing resources. Based on the compute resources needed to run an algorithm in the Compute-to-Data environment, a consumer can choose the amount to pay according to their needs.

These fees incentivize individuals and organizations to run their provider instances and charge consumers according to resource usage.

Fee values

The table is periodically updated. Users are advised to confirm new values through the contracts and the market.

Swap fees

Market/Type	Value in Ocean Market, using any Provider	Value in Other Markets
publishMarket: FixedRate	0%	Set in the market config, by the publishing market. Min = 0.001% Max = 50%
consumeMarket: FixedRate ERC20Template	0%	0%
consumeMarket: FixedRate EnterpriseTemplate	0%	Set in market config, by the consuming market.
Ocean Community: FixedRate OCEAN, H2O as base token	0.1%	0.1%
Ocean Community: FixedRate other base token	0.2%	0.2%

Publish fees

Market/Type	Value in Ocean Market, using any Provider	Value in Other Markets
-	0%	0%

Order fees (1 DT)

Market/Type	Value in Ocean Market, using any Provider	Value in Other Markets
publishMarket Absolute value, in any token. E.g. 5 USDT	0	Set in market config, by the publishing market.
consumeMarket Absolute value, in any token. E.g. 2 DAI	0	Set in market config, by the consuming market.
Ocean Community Fixed price in DT	0.03 DT	0.03 DT

Ocean Provider fees

Type	OPF Provider	3rd party Provider
Token in which fee is charged: `PROVIDER_FEE_TOKEN`	OCEAN	E.g. USDC
Download: `COST_PER_MB`	0	Set in Provider envvars.
Compute: `COST_PER_MIN` Environment: 1 CPU, 60 secs max	0	Set in OperatorEngine envvars
Compute: `COST_PER_MIN` Environment: 1 CPU, 1 hour max	1.0 OCEAN/min	Set in OperatorEngine envvars
Ocean Community	0% of the Provider fee	0% of the Provider fee

Further reading

- The Web3 Sustainability Loop

Asset Pricing

Choose the revenue model during asset publishing

Ocean Protocol offers two types of pricing options for asset monetization. The publisher can choose a pricing model which best suits their needs while publishing an asset. The pricing model selected cannot be changed once the asset is published.

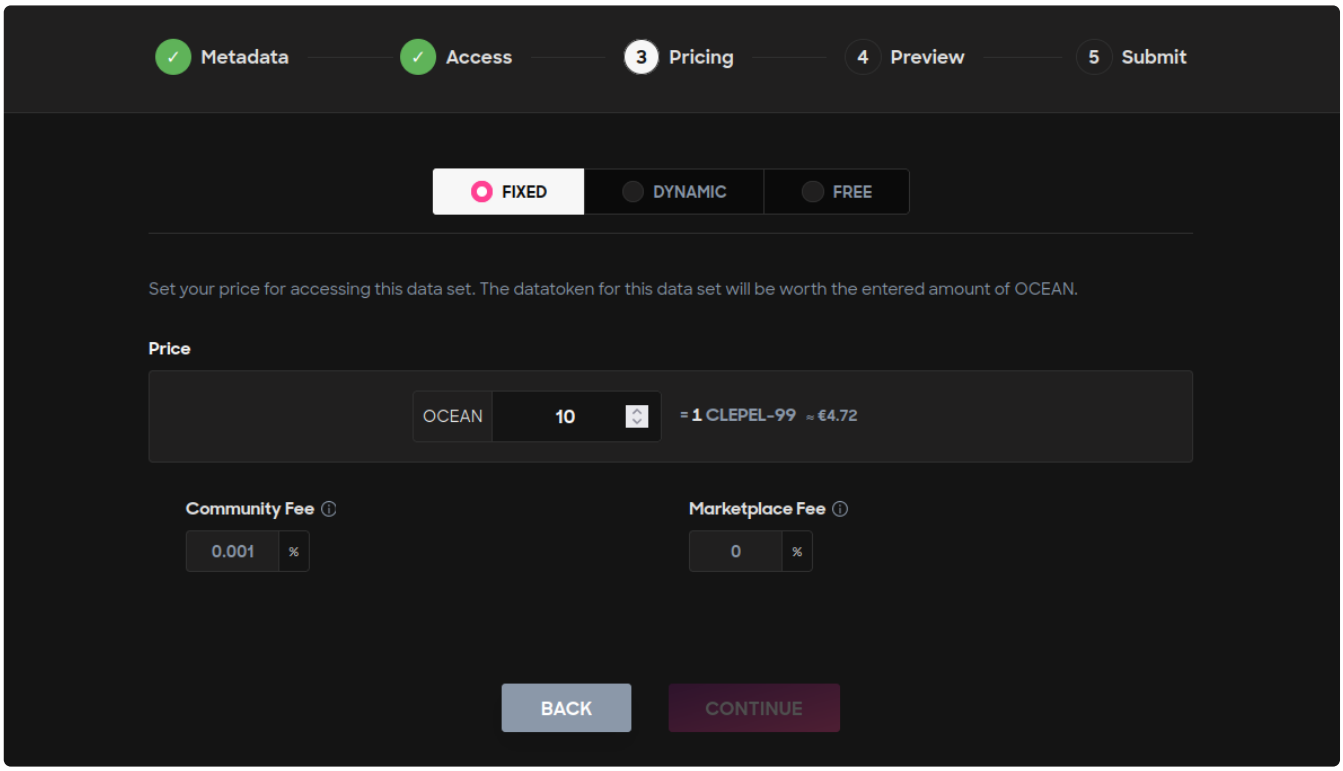
The price of an asset is determined by the number of Ocean tokens a buyer must pay to access the asset. When users pay the right amount of Ocean tokens, they get a *datatoken* in their wallets, a tokenized representation of the access right stored on the blockchain. To read more about datatoken and data NFT click here.

Fixed pricing

With the fixed price model, publishers set the price for the data in OCEAN. Ocean Market creates a datatoken in the background with a value equal to the dataset price in OCEAN so that buyers do not have to know about the datatoken. Buyers pay the amount specified in OCEAN for access. The publisher can update the price of the dataset later anytime.

A FixedRateExchange smart contract stores the information about the price of the assets published using this model.

The image below shows how to set the fixed pricing of an asset in the Ocean's Marketplace. Here, the price of the asset is set to 10 Ocean tokens.



fixed-asset-pricing

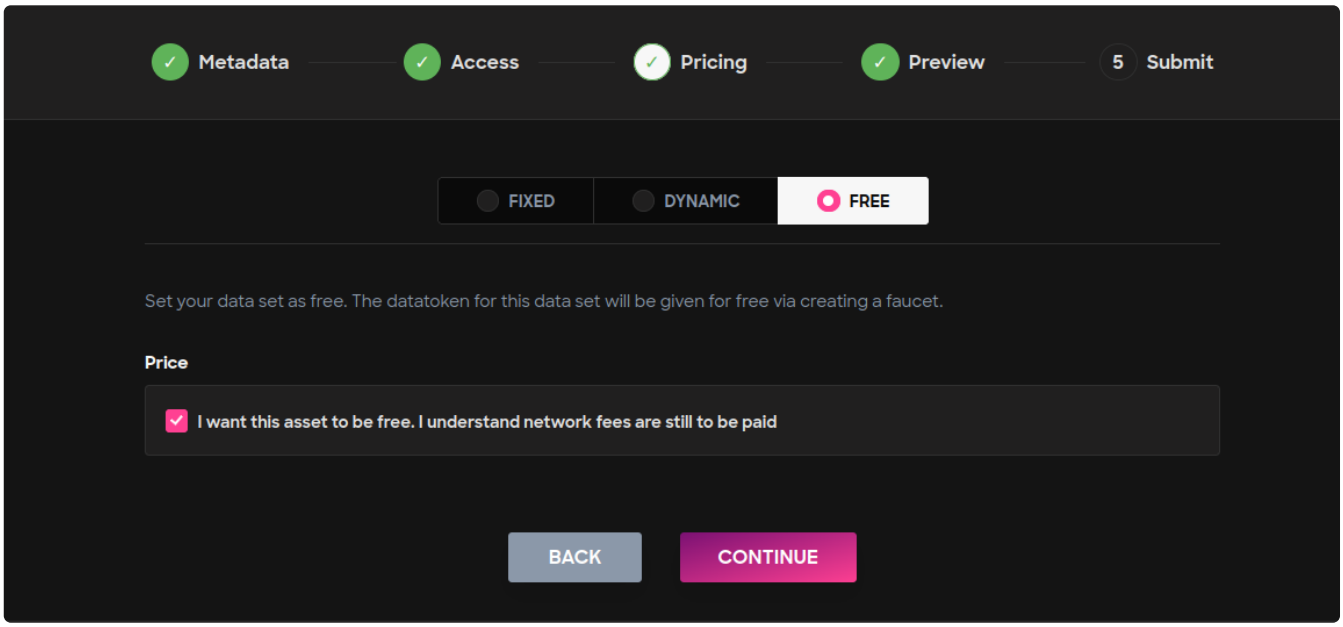
Free pricing

With the free pricing model, the buyers can access an asset without requiring them to pay for it except for the transaction fees.

With this pricing model, datatokens are allocated to the dispenser smart contract, which dispenses data tokens to users for free whenever they are accessing an asset.

Free pricing is suitable for individuals and organizations working in the public domain and want their datasets to be freely available. Publishers can also choose this model if they publish assets with licenses that require them to make them freely available.

The image below shows how to set free access to an asset in the Ocean's Marketplace.



free-asset-pricing

DID & DDO

Specification of decentralized identifiers for assets in Ocean Protocol using the DID & DDO standards.

v4.1.0

Overview

This document describes how Ocean assets follow the DID/DDO specification, such that Ocean assets can inherit DID/DDO benefits and enhance interoperability. DIDs and DDOs follow the specification defined by the World Wide Web Consortium (W3C).

Decentralized identifiers (DIDs) are a type of identifier that enable verifiable, decentralized digital identity. Each DID is associated with a unique entity, and DIDs may represent humans, objects, and more.

A DID Document (DDO) is a JSON blob that holds information about the DID. Given a DID, a *resolver* will return the DDO of that DID.

Rules for DID & DDO

An *asset* in Ocean represents a downloadable file, compute service, or similar. Each asset is a *resource* under the control of a *publisher*. The Ocean network itself does *not* store the actual resource (e.g. files).

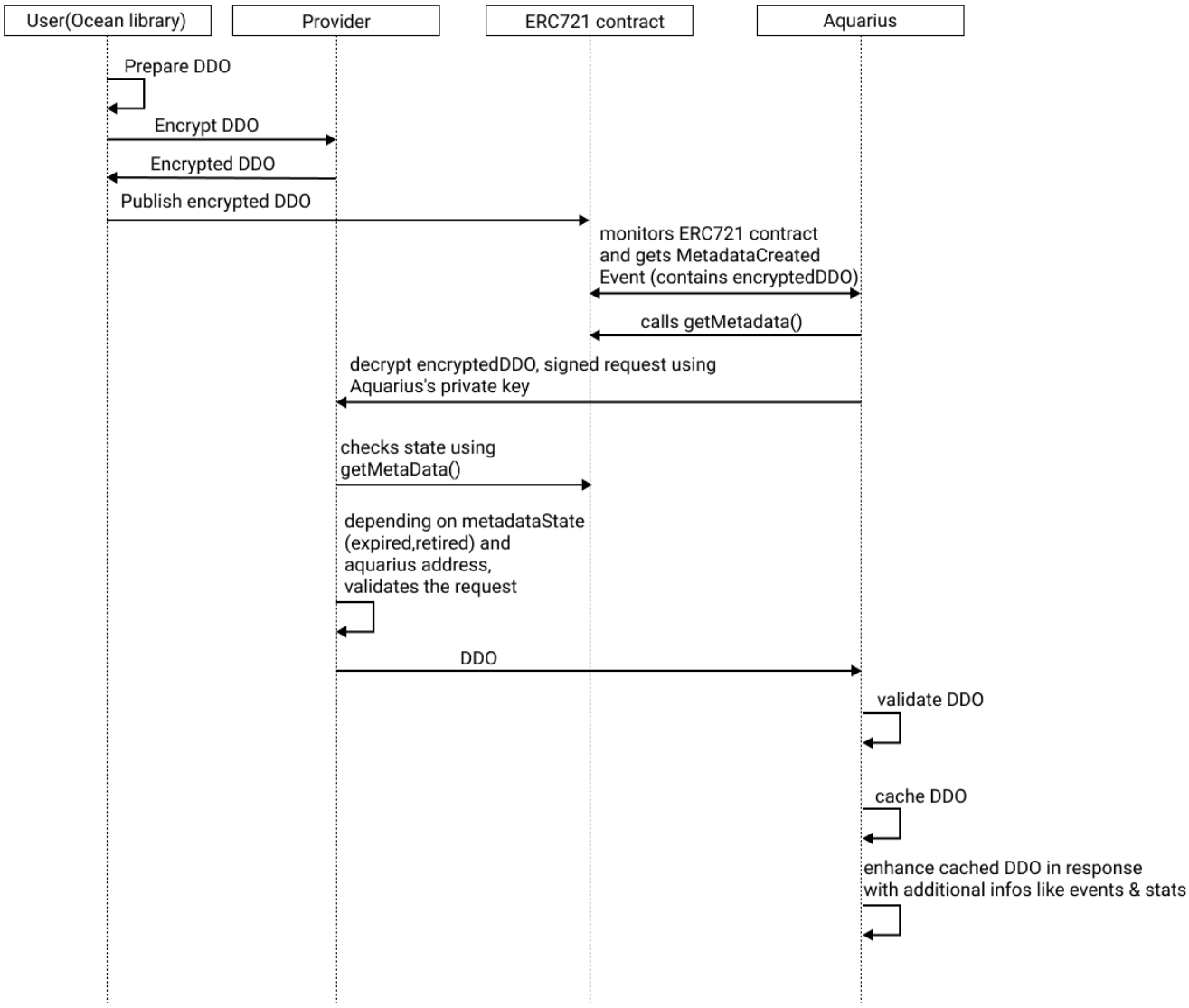
An *asset* has a DID and DDO. The DDO should include metadata about the asset, and define access in at least one service. Only *owners* or *delegated users* can modify the DDO.

All DDOs are stored on-chain in encrypted form to be fully GDPR-compatible. A metadata cache like *Aquarius* can help in reading, decrypting, and searching through encrypted DDO data from the chain. Because the file URLs are encrypted on top of the full DDO encryption, returning unencrypted DDOs e.g. via an API is safe to do as the file URLs will still stay encrypted.

Publishing & Retrieving DDOs

The DDO is stored on-chain as part of the NFT contract and stored in encrypted form using the private key of the *Provider*. To resolve it, a metadata cache like *Aquarius* must query the provider to decrypt the DDO.

Here is the flow:


User(Ocean library)
Provider
ERC721 contract
Aquarius
Prepare DDO
Encrypt DDO
Encrypted DDO
Publish encrypted DDO
monitors ERC721 contract
and gets MetadataCreated
Event (contains encryptedDDO)
calls getMetadata()
decrypt encryptedDDO, signed request using
Aquarius's private key
checks state using
getMetaData()
depending on metadataState
(expired,retired) and
aquarius address,
validates the request
DDO
validate DDO
cache DDO
enhance cached DDO in response
with additional infos like events & stats

DDO flow

UML source

```
title DDO flow

User(Ocean library) -> User(Ocean library): Prepare DDO
User(Ocean library) -> Provider: encrypt DDO
Provider -> User(Ocean library): encryptedDDO
User(Ocean library) -> ERC721 contract: publish encryptedDDO
Aquarius <-> ERC721 contract: monitors ERC721 contract and gets MetdadataCreated Even
Aquarius -> ERC721 contract: calls getMetaData()
Aquarius -> Provider: decrypt encryptedDDO, signed request using Aquarius's private k
Provider -> ERC721 contract: checks state using getMetaData()
Provider -> Provider: depending on metadataState (expired,retired) and aquarius addre
Provider -> Aquarius: DDO
Aquarius -> Aquarius : validate DDO
Aquarius -> Aquarius : cache DDO
Aquarius -> Aquarius : enhance cached DDO in response with additional infos like even
```

DID

In Ocean, a DID is a string that looks like this:

```
did:op:0ebed8226ada17fde24b6bf2b95d27f8f05fcce09139ff5cec31f6d81a7cd2ea
```

The part after `did:op:` is the ERC721 contract address(in checksum format) and the chainId (expressed as a decimal) the asset has been published to:

```
const checksum = sha256(ERC721 contract address + chainId)
console.log(checksum)
// 0ebed8226ada17fde24b6bf2b95d27f8f05fcce09139ff5cec31f6d81a7cd2ea
```

It follows the generic DID scheme.

DDO

A DDO in Ocean has these required attributes:

Attribute	Type	Description
`@context`	Array of `string`	Contexts used for validation.
`id`	`string`	Computed as `sha256(address of ERC721 contract + chainId)`.
`version`	`string`	Version information in SemVer notation referring to this DDO spec version, like `4.1.0`.
`chainId`	`number`	Stores chainId of the network th DDO was published to.
`nftAddress`	`string`	NFT contract linked to this asse
`metadata`	Metadata	Stores an object describing the asset.
`services`	Services	Stores an array of services defining access to the asset.
`credentials`	Credentials	Describes the credentials needed to access a dataset in addition to the `services` definition.

Metadata

This object holds information describing the actual asset.

Attribute	Type	Required	Description
`created`	`ISO date/time string`		Contains the date of the creation of the dataset content in ISO 8601 format preferabl with timezone designators, e.g. `2000-10-31T01:30:00Z`.
`updated`	`ISO date/time string`		Contains the date of last update of the dataset content in ISO 8601 format preferabl with timezone designators, e.g. `2000-10-31T01:30:00Z`.
`description`	`string`	✓	Details of what the resource is. For a dataset, this attribute explains what the dat represents and what i can be used for.
`copyrightHolder`	`string`		The party holding the legal copyright. Empt by default.
`name`	`string`	✓	Descriptive name or title of the asset.
`type`	`string`	✓	Asset type. Includes `"dataset"` (e.g. cs file), `"algorithm"` (e.g. Python script). Each type needs a different subset of metadata attributes.
`author`	`string`	✓	Name of the entity generating this data (e.g. Tfl, Disney Corp etc.).
			Short name referencing the licens

`license`	`string`	✓	of the asset (e.g. Pub Domain, CC-0, CC-B No License Specified etc.). If it's not specified, the followir value will be added: "No License Specified".
`links`	Array of `string`		Mapping of URL strings for data samples, or links to find out more information. Links ma be to either a URL or another asset.
`contentLanguage`	`string`		The language of the content. Use one of th language codes from the IETF BCP 47 standard
`tags`	Array of `string`		Array of keywords or tags used to describe this content. Empty by default.
`categories`	Array of `string`		Array of categories associated to the asset. Note: recommended to use `tags` instead of this
`additionalInformation`	Object		Stores additional information, this is customizable by publisher
`algorithm`	Algorithm Metadata	✓ (for algorithm assets only)	Information about ass of `type` `algorith`

Example:

```
{
  "metadata": {
    "created": "2020-11-15T12:27:48Z",
    "updated": "2021-05-17T21:58:02Z",
    "description": "Sample description",
    "name": "Sample asset",
    "type": "dataset",

    "author": "OPF",
    "license": "https://market.oceanprotocol.com/terms"
  }
}
```

Algorithm Metadata

An asset of type `algorithm` has additional attributes under `metadata.algorithm`, describing the algorithm and the Docker environment it is supposed to be run under.

Attribute	Type	Required	Description
`language`	`string`		Language used to implement the software.
`version`	`string`		Version of the softwar preferably in SemVer notation. E.g. `1.0.0`
`consumerParamet ers`	Consumer Parameters		An object that defines required consumer input before running the algorithm
`container`	`container`	✓	Object describing the Docker container image. See below

The `container` object has the following attributes defining the Docker image for running the algorithm:

Attribute	Type	Required	Description
`entrypoint`	`string`	✓	The command to execute, or script to ru inside the Docker image.
`image`	`string`	✓	Name of the Docker image.
`tag`	`string`	✓	Tag of the Docker image.
`checksum`	`string`	✓	Digest of the Docker image. (ie: sha256:xxxxx)

```
{
  "metadata": {
    "created": "2020-11-15T12:27:48Z",
    "updated": "2021-05-17T21:58:02Z",
    "description": "Sample description",
    "name": "Sample algorithm asset",
    "type": "algorithm",
    "author": "OPF",
    "license": "https://market.oceanprotocol.com/terms",
    "algorithm": {
      "language": "Node.js",
      "version": "1.0.0",
      "container": {
        "entrypoint": "node $ALGO",
        "image": "ubuntu",
        "tag": "latest",
        "checksum": "sha256:44e10daa6637893f4276bb8d7301eb35306ece50f61ca34dcab550"
      },
      "consumerParameters": {}
    }
  }
}
```

Services

Services define the access for an asset, and each service is represented by its respective datatoken.

An asset should have at least one service to be actually accessible, and can have as many services which make sense for a specific use case.

Attribute	Type	Required	Description
`id`	`string`	✓	Unique ID
`type`	`string`	✓	Type of service (`access`, `compute` `wss`, etc.
`name`	`string`		Service friendly name
`description`	`string`		Service description
`datatokenAddress`	`string`	✓	Datatoken address
`serviceEndpoint`	`string`	✓	Provider URL (schem + host)
`files`	Files	✓	Encrypted file URLs.
`timeout`	`number`	✓	Describing how long the service can be used after consumptio is initiated. A timeout `0` represents no time limit. Expressed in seconds.
`compute`	Compute	✓ (for compute assets only)	If service is of `type` `compute`, holds information about the compute-related privacy settings & resources.
`consumerParameters`	Consumer Parameters		An object the defines required consumer input before consuming the asset
`additionalInformation`	Object		Stores additional information, this is customizable by publisher

Files

The `files` field is returned as a `string` which holds the encrypted file URLs.

Example:

```
{
  "files": "0x044736da6dae39889ff570c34540f24e5e084f4e5bd81eff3691b729c2dd1465ae8292fc721e9d4
}
```

During the publish process, file URLs must be encrypted with a respective *Provider* API call before storing the DDO on-chain. For this, you need to send the following object to Provider:

```
{
  "datatokenAddress":"0x1",
  "nftAddress": "0x2",
  "files": [
    ...
  ]
}
```

where "files" contains one or more storage objects.

Type of objects supported:

`URL`

Static URLs.

Parameters:

- `url` - File url, required
- `method` - The HTTP method, required
- `headers` - Additional HTTP headers, optional

```
{
    "type": "url",
    "url": "https://url.com/file1.csv",
    "method": "GET",
    "headers":
    {
        "Authorization": "Bearer 123",
        "APIKEY": "124",
    }
}
```

`IPFS`

The Interplanetary File System (IPFS) is a distributed file storage protocol that allows computers all over the globe to store and serve files as part of a giant peer-to-peer network. Any computer, anywhere in the world, can download the IPFS software and start hosting and serving files.

Parameters:

- `hash` - The file hash

```
{
        "type": "ipfs",
        "hash": "XXX"
}
```

`GraphQL`

GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data.

Parameters:

- `url` - Server endpoint url, required
- `query` - The query to be executed, required
- `headers` - Additional HTTP headers, optional

```
{
        "type": "graphql",
        "url": "http://172.15.0.15:8000/subgraphs/name/oceanprotocol/ocean-subgraph",
        "headers":{
                "Authorization": "Bearer 123",
                "APIKEY": "124",
        },
        "query": """query{
            nfts(orderBy: createdTimestamp,orderDirection:desc){
                 id
                 symbol
                 createdTimestamp
            }
          }"""
}
```

`On-Chain`

Use a smart contract as data source.

Parameters:

- `chainId` - The chainId used to query the contract, required
- `address` - The smartcontract address, required
- `abi` - The function abi (NOT the entire contract abi), required

```
{
        "type": "smartcontract",
        "chainId": 1,
        "address": "0x8149276f275EEFAc110D74AFE8AFECEaeC7d1593",
        "abi": {
                        "inputs": [],
                        "name": "swapOceanFee",
                        "outputs": [{"internalType": "uint256", "name": "", "type": "uint256"}
                        "stateMutability": "view",
                        "type": "function",
                }
}
```

`Arweave`

Arweave is a decentralized data storage that allows to permanently store files over a distributed network of computers.

Parameters:

- `transactionId` - The transaction identifier

```
{
    {
    "type": "arweave",
    "transactionId": "a4qJoQZa1poIv5guEzkfgZYSAD0uYm7Vw4zm_tCswVQ",
  }
}
```

First class integrations supported in the future : `Filecoin` `Storj` `SQL`

A service can contain multiple files, using multiple storage types.

Example:

```
{
  "datatokenAddress":"0x1",
  "nftAddress": "0x2",
  "files": [
    {
      "type": "url",
      "url": "https://url.com/file1.csv",
      "method": "GET"
    },
    {
      "type": "ipfs",
      "hash": "XXXX"
    }
  ]
}
```

To get information about the files after encryption, the `/fileinfo` endpoint of *Provider* returns based on a passed DID an array of file metadata (based on the file type):

```
[
  {
    "type": "url",
    "contentLength": 100,
    "contentType": "application/json"
  },
  {
    "type": "ipfs",
    "contentLength": 130,
    "contentType": "application/text"
  }
]
```

This only concerns metadata about a file, but never the file URLs. The only way to decrypt them is to exchange at least 1 datatoken based on the respective service pricing scheme.

Compute Options

An asset with a service of `type` `compute` has the following additional attributes under the `compute` object. This object is required if the asset is of `type` `compute`, but can be omitted for `type` of `access`.

Attribute	Type	Required	Description
`allowRawAlgorithm`	`boolean`	✓	If `true`, any passed raw text will be allowe to run. Useful for an algorithm drag & drop use case, but increases risk of data escape through malicious user input. Should be `false` by default in all implementations.
`allowNetworkAccess`	`boolean`	✓	If `true`, the algorithm job will have network access.
`publisherTrustedAlgorithmPublishers`	Array of `string`	✓	If not defined, then an published algorithm is allowed. If empty arra then no algorithm is allowed. If not empty any algo published by the defined publisher is allowed.
`publisherTrustedAlgorithms`	Array of `publisherTrustedAlgorithms`	✓	If not defined, then an published algorithm is allowed. If empty arra then no algorithm is allowed. Otherwise only the algorithms defined in the array a allowed. (see below).

The `publisherTrustedAlgorithms` is an array of objects with the following structure:

Attribute	Type	Required	Description
`did`	`string`	✓	The DID of the algorithm which is trusted by the publisher.
`filesChecksum`	`string`	✓	Hash of algorithm's files (as `string`).
`containerSectionChecksum`	`string`	✓	Hash of algorithm's image details (as `string`).

To produce `filesChecksum`, call the Provider FileInfoEndpoint with parameter withChecksum = True. If algorithm has multiple files, `filesChecksum` is a concatenated string of all files checksums (ie: checksumFile1+checksumFile2 , etc)

To produce `containerSectionChecksum`:

```
sha256(algorithm_ddo.metadata.algorithm.container.entrypoint + algorithm_ddo.metadata.algorit
```

Example:

```
{
  "services": [
    {
      "id": "1",
      "type": "access",
      "files": "0x044736da6dae39889ff570c34540f24e5e084f...",
      "name": "Download service",
      "description": "Download service",
      "datatokenAddress": "0x123",
      "serviceEndpoint": "https://myprovider.com",
      "timeout": 0
    },
    {
      "id": "2",
      "type": "compute",
      "files": "0x6dd05e0edb460623c843a263291ebe757c1eb3...",
      "name": "Compute service",
      "description": "Compute service",
      "datatokenAddress": "0x124",
      "serviceEndpoint": "https://myprovider.com",
      "timeout": 0,
      "compute": {
        "allowRawAlgorithm": false,
        "allowNetworkAccess": true,
        "publisherTrustedAlgorithmPublishers": ["0x234", "0x235"],
        "publisherTrustedAlgorithms": [
          {
            "did": "did:op:123",
            "filesChecksum": "100",
            "containerSectionChecksum": "200"
          },
          {
            "did": "did:op:124",
            "filesChecksum": "110",
            "containerSectionChecksum": "210"
          }
        ]
      }
    }
  ]
}
```


Consumer Parameters

Sometimes, the asset needs additional input data before downloading or running a Compute-to-Data job. Examples:

- The publisher needs to know the sampling interval before the buyer downloads it. Suppose the dataset URL is `https://example.com/mydata`. The publisher defines a field called `sampling` and asks the buyer to enter a value. This parameter is then added to the URL of the published dataset as query parameters: `https://example.com/mydata?sampling=10`.

- An algorithm that needs to know the number of iterations it should perform. In this case, the algorithm publisher defines a field called `iterations`. The buyer needs to enter a value for the `iterations` parameter. Later, this value is stored in a specific location in the Compute-to-Data pod for the algorithm to read and use it.

The `consumerParameters` is an array of objects. Each object defines a field and has the following structure:

Attribute	Type	Required	Description
`name`	`string`	✓	The parameter name (this is sent as HTTP param or key towards algo)
`type`	`string`	✓	The field type (text, number, boolean, select)
`label`	`string`	✓	The field label which displayed
`required`	`boolean`	✓	If customer input for this field is mandatory
`description`	`string`	✓	The field description.
`default`	`string`, `number`, or `boolean`	✓	The field default value For select types, `string` key of default option.
`options`	Array of `option`		For select types, a list of options.

Each `option` is an `object` containing a single key:value pair where the key is the option name, and the value is the option value.

Example:

```
[
  {
    "name": "hometown",
    "type": "text",
    "label": "Hometown",
    "required": true,
    "description": "What is your hometown?",
    "default": "Nowhere"
  },
  {
    "name":"age",
    "type": "number",
    "label": "Age",
    "required": false,
    "description":"Please fill your age",
    "default": 0
  },
  {
    "name":"developer",
    "type": "boolean",
    "label": "Developer",
    "required": false,
    "description":"Are you a developer?",
    "default": false
  },
  {
    "name":"languagePreference",
    "type": "select",
    "label": "Language",
    "required": false,
    "description": "Do you like NodeJs or Python",
    "default": "nodejs",
    "options": [
      {
        "nodejs" : "I love NodeJs"
      },
      {
        "python" : "I love Python"
      }
    ]
  }
]
```

Algorithms will have access to a JSON file located at /data/inputs/algoCustomData.json, which contains the keys/values for input data required. Example:

```
{
  "hometown": "São Paulo",
  "age": 10,
  "developer": true,
  "languagePreference": "nodejs"

}
```

Credentials

By default, a consumer can access a resource if they have 1 datatoken. *Credentials* allow the publisher to optionally specify more fine-grained permissions.

Consider a medical data use case, where only a credentialed EU researcher can legally access a given dataset. Ocean supports this as follows: a consumer can only access the resource if they have 1 datatoken *and* one of the specified `"allow"` credentials.

This is like going to an R-rated movie, where you can only get in if you show both your movie ticket (datatoken) *and* some identification showing you're old enough (credential).

Only credentials that can be proven are supported. This includes Ethereum public addresses, and in the future W3C Verifiable Credentials and more.

Ocean also supports `"deny"` credentials: if a consumer has any of these credentials, they can not access the resource.

Here's an example object with both `"allow"` and `"deny"` entries:

```
{
  "credentials": {
    "allow": [
      {
        "type": "address",
        "values": ["0x123", "0x456"]
      }
    ],
    "deny": [
      {
        "type": "address",
        "values": ["0x2222", "0x333"]
      }
    ]
  }
}
```

DDO Checksum

In order to ensure the integrity of the DDO, a checksum is computed for each DDO:

```
const checksum = sha256(JSON.stringify(ddo))
```

The checksum hash is used when publishing/updating metadata using the `setMetaData` function in the ERC721 contract, and is stored in the event generated by the ERC721 contract:

```
event MetadataCreated(
  address indexed createdBy,
  uint8 state,
  string decryptorUrl,
  bytes flags,
  bytes data,
  bytes metaDataHash,
  uint256 timestamp,
  uint256 blockNumber
);

event MetadataUpdated(
  address indexed updatedBy,
  uint8 state,
  string decryptorUrl,
  bytes flags,
  bytes data,
  bytes metaDataHash,
  uint256 timestamp,
  uint256 blockNumber
);
```

Aquarius should always verify the checksum after data is decrypted via a *Provider* API call.

State

Each asset has a state, which is held by the NFT contract. The possible states are:

State	Description	Discoverable in Ocean Market	Ordering allowed	Listed under profile
`0`	Active	Yes	Yes	Yes
`1`	End-of-life	No	No	No
`2`	Deprecated (by another asset)	No	No	No
`3`	Revoked by publisher	No	No	No
`4`	Ordering is temporary disabled	Yes	No	Yes
`5`	Asset unlisted.	No	Yes	Yes

Aquarius Enhanced DDO Response

The following fields are added by *Aquarius* in its DDO response for convenience reasons, where an asset returned by *Aquarius* inherits the DDO fields stored on-chain.

These additional fields are never stored on-chain, and are never taken into consideration when hashing the DDO.

NFT

The `nft` object contains information about the ERC721 NFT contract which represents the intellectual property of the publisher.

Attribute	Type	Description
`address`	`string`	Contract address of the deployed ERC721 NFT contract.
`name`	`string`	Name of NFT set in contract.
`symbol`	`string`	Symbol of NFT set in contract.
`owner`	`string`	ETH account address of the NFT owner.
`state`	`number`	State of the asset reflecting the NFT contract value. See State
`created`	`ISO date/time string`	Contains the date of NFT creation.
`tokenURI`	`string`	tokenURI

Example:

```
{
  "nft": {
    "address": "0x000000",
    "name": "Ocean Protocol Asset v4",
    "symbol": "OCEAN-A-v4",
    "owner": "0x0000000",
    "state": 0,
    "created": "2000-10-31T01:30:00Z"
  }
}
```

Datatokens

The `datatokens` array contains information about the ERC20 datatokens attached to asset services.

Attribute	Type	Description
`address`	`string`	Contract address of the deployed ERC20 contract.
`name`	`string`	Name of NFT set in contract.
`symbol`	`string`	Symbol of NFT set in contract.
`serviceId`	`string`	ID of the service the datatoken attached to.

Example:

```
{
  "datatokens": [
    {
      "address": "0x000000",
      "name": "Datatoken 1",
      "symbol": "DT-1",
      "serviceId": "1"
    },
    {
      "address": "0x000001",
      "name": "Datatoken 2",
      "symbol": "DT-2",
      "serviceId": "2"
    }
  ]
}
```

Event

The `event` section contains information about the last transaction that created or updated the DDO.

Example:

```
{
  "event": {
    "tx": "0x8d127de58509be5dfac600792ad24cc9164921571d168bff2f123c7f1cb4b11c",
    "block": 12831214,
    "from": "0xAcca11dbeD4F863Bb3bC2336D3CE5BAC52aa1f83",
    "contract": "0x1a4b70d8c9DcA47cD6D0Fb3c52BB8634CA1C0Fdf",
    "datetime": "2000-10-31T01:30:00"
  }
}
```

Purgatory

Contains information about an asset's purgatory status defined in `list-purgatory`. Marketplace interfaces are encouraged to prevent certain user actions like adding liquidity on assets in purgatory.

Attribute	Type	Description
`state`	`boolean`	If `true`, asset is in purgatory.
`reason`	`string`	If asset is in purgatory, contains the reason for being there as defined in `list-purgatory`

Example:

```
{
  "purgatory": {
    "state": true,
    "reason": "Copyright violation"
  }
}
```

```
{
  "purgatory": {
    "state": false
  }
}
```

Statistics

The `stats` section contains different statistics fields.

Attribute	Type	Description
`orders`	`number`	How often an asset was ordered, meaning how often it was either downloaded or used as part of a compute job.

Example:

```
{
  "stats": {
    "orders": 4
  }
}
```

Full Enhanced DDO Example

```
{
  "@context": ["https://w3id.org/did/v1"],
  "id": "did:op:ACce67694eD2848dd683c651Dab7Af823b7dd123",
  "version": "4.1.0",
  "chainId": 1,
  "nftAddress": "0x123",
  "metadata": {
    "created": "2020-11-15T12:27:48Z",
    "updated": "2021-05-17T21:58:02Z",
    "description": "Sample description",
    "name": "Sample asset",
    "type": "dataset",
    "author": "OPF",
    "license": "https://market.oceanprotocol.com/terms"
  },
  "services": [
    {
      "id": "1",
      "type": "access",
      "files": "0x044736da6dae39889ff570c34540f24e5e084f4e5bd81eff3691b729c2dd1465ae8292fc721
      "name": "Download service",
      "description": "Download service",
      "datatokenAddress": "0x123",
      "serviceEndpoint": "https://myprovider.com",
      "timeout": 0,
      "consumerParameters": [
        {
          "name": "surname",
          "type": "text",
          "label": "Name",
          "required": true,
          "default": "NoName",
          "description": "Please fill your name"
        },
        {
          "name": "age",
          "type": "number",
          "label": "Age",
          "required": false,
          "default": 0,
          "description": "Please fill your age"
        }
      ]
    },
    {
      "id": "2",
      "type": "compute",
      "files": "0x044736da6dae39889ff570c34540f24e5e084f4e5bd81eff3691b729c2dd1465ae8292fc721
      "name": "Compute service",
      "description": "Compute service",
      "datatokenAddress": "0x124",
      "serviceEndpoint": "https://myprovider.com",
```

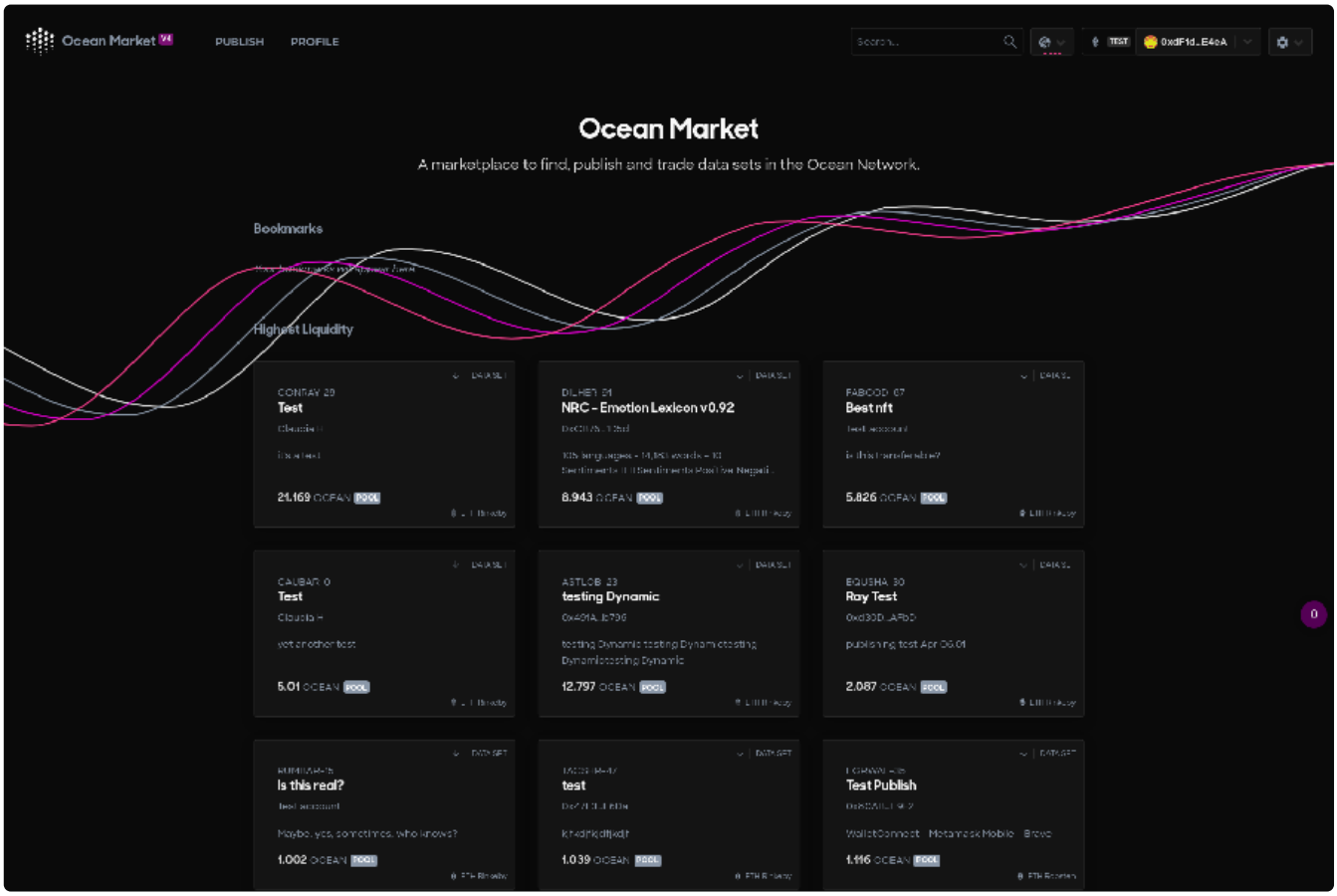
```
      "timeout": 3600,
      "compute": {
        "allowRawAlgorithm": false,
        "allowNetworkAccess": true,
        "publisherTrustedAlgorithmPublishers": ["0x234", "0x235"],
        "publisherTrustedAlgorithms": [
          {
            "did": "did:op:123",
            "filesChecksum": "100",
            "containerSectionChecksum": "200"
          },
          {
            "did": "did:op:124",
            "filesChecksum": "110",
            "containerSectionChecksum": "210"
          }
        ]
      }
    }
  ],
  "credentials": {
    "allow": [
      {
        "type": "address",
        "values": ["0x123", "0x456"]
      }
    ],
    "deny": [
      {
        "type": "address",
        "values": ["0x2222", "0x333"]
      }
    ]
  },

  "nft": {
    "address": "0x123",
    "name": "Ocean Protocol Asset v4",
    "symbol": "OCEAN-A-v4",
    "owner": "0x0000000",
    "state": 0,
    "created": "2000-10-31T01:30:00",
    "tokenURI": "xxx"
  },

  "datatokens": [
    {
      "address": "0x000000",
      "name": "Datatoken 1",
      "symbol": "DT-1",
      "serviceId": "1"
    },
    {
```

```
      "address": "0x000001",
      "name": "Datatoken 2",
      "symbol": "DT-2",
      "serviceId": "2"
    }
  ],

  "event": {
    "tx": "0x8d127de58509be5dfac600792ad24cc9164921571d168bff2f123c7f1cb4b11c",
    "block": 12831214,
    "from": "0xAcca11dbeD4F863Bb3bC2336D3CE5BAC52aa1f83",
    "contract": "0x1a4b70d8c9DcA47cD6D0Fb3c52BB8634CA1C0Fdf",
    "datetime": "2000-10-31T01:30:00"
  },

  "purgatory": {
    "state": false
  },

  "stats": {
    "orders": 4
  }
}
```

Using Ocean Market

About Ocean Market

- Ocean Market enables publishers to monetize their data and/or algorithms through blockchain technology.
- Consumers can purchase access to data, algorithms, compute services.



Ocean Market Landing Page

The following guides will help you get started with buying and selling data:

- Publish a data asset
- Download a data asset
- Publishing with hosting services

If you are new to web3 and blockchain technologies then we suggest you first read these introductory guides:

- Wallet Basics
- Set Up MetaMask Wallet
- Manage Your OCEAN Tokens

Removing Liquidity

The AMM pools, and dynamic pricing schema are no longer available on the Ocean Market. Refer this page on removing liquidity from the pool using Etherscan.

Publish a Data Asset

Tutorial to publish assets using the Ocean Market

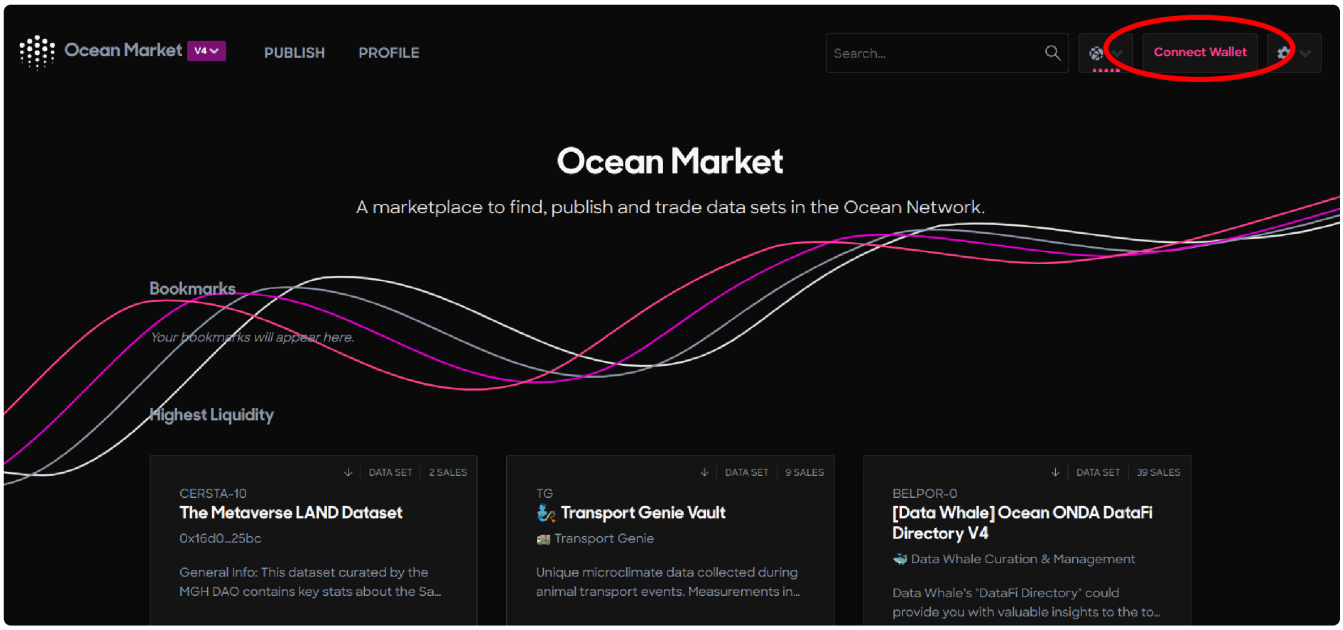
What can be published?

Ocean Market provides a convenient interface for individuals and organizations to publish their data. Datasets can be images, location information, audio, video, sales data, or combinations of all! There is no exhaustive list of what type of data can be published on the Market. Please note the Ocean Protocol team maintains a purgatory list here to block addresses and remove assets for any violations.

Tutorial

Connect wallet and navigate to the publish page

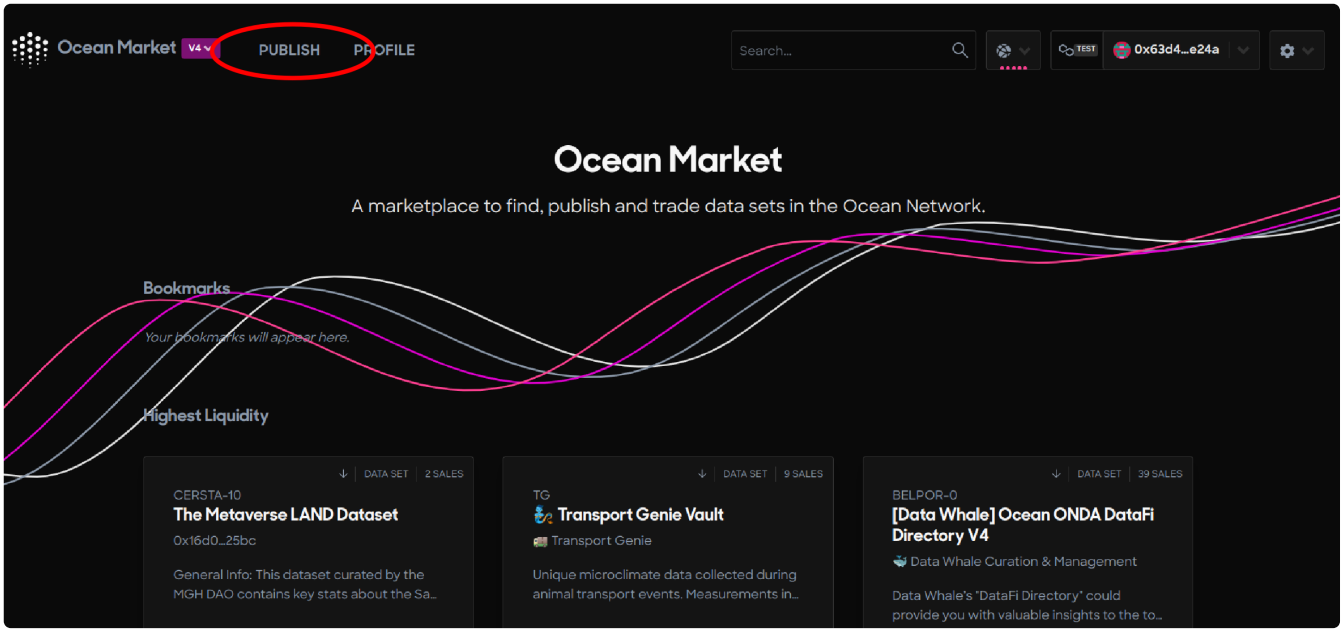
1. Go to Ocean Market
2. Connect wallet.



Connect wallet

In this tutorial, we will be using the Polygon Mumbai test network.

3. Go to the publish page.



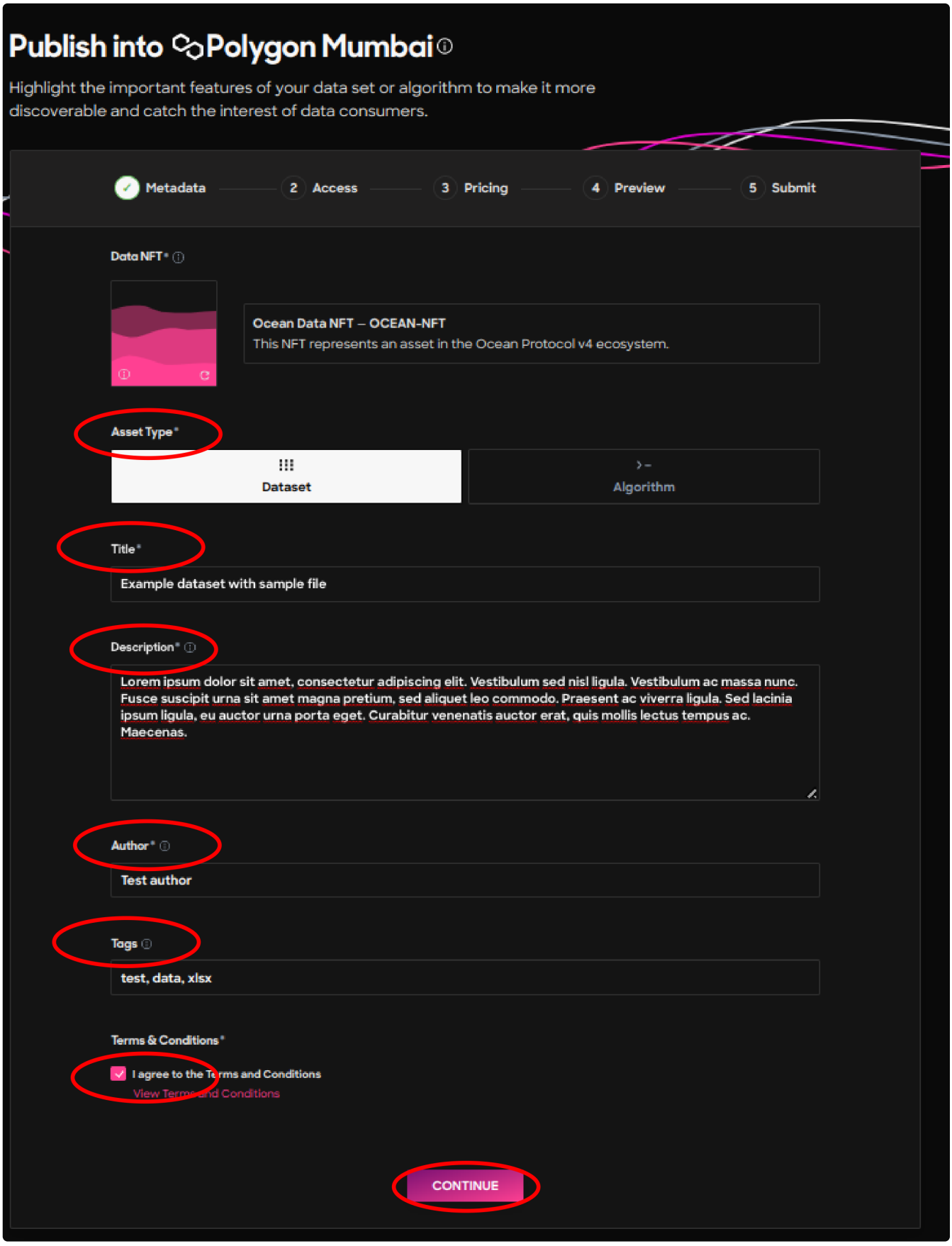
Publish page

Step 1 - Metadata

Fill in the metadata.

*Mandatory fields are marked with **

- **Asset type***
  An asset can be a *dataset* or an *algorithm*. The asset type cannot be changed after publication.
- **Title***
  The descriptive name of the asset. This field is editable after the asset publication.
- **Description***
  Description of the asset. Ocean Marketplace supports plain text and Markdown format for the description field. This field is editable after the asset publication.
- **Author***
  The author of the asset. The author can be an individual or an organization. This field is editable after the asset publication.
- **Tags**
  Tags help the asset to be discoverable. If not provided, the list of tags is empty by default.

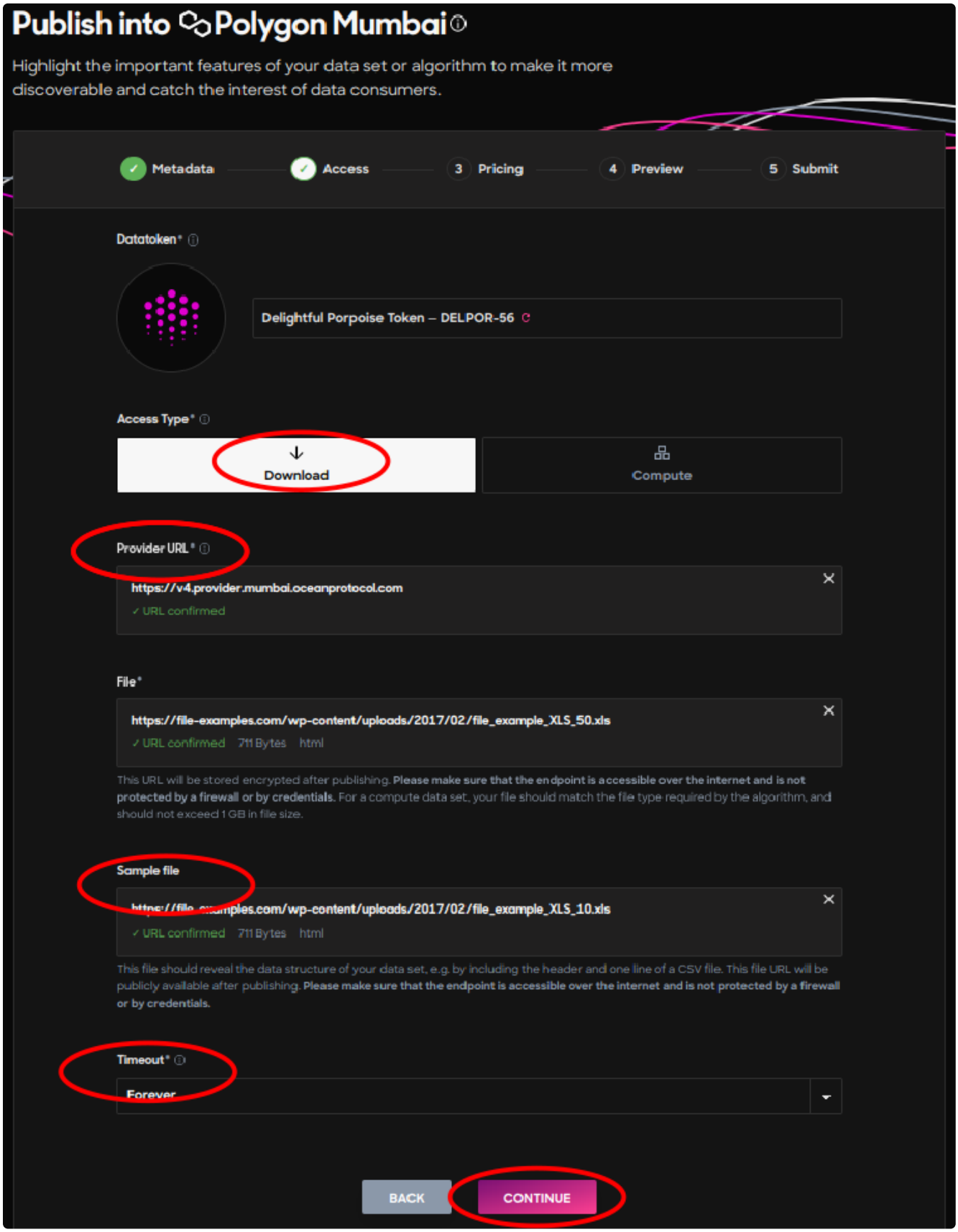


Asset metadata

Step 2 - Access details

*Mandatory fields are marked with **

- **Access Type***
  An asset can be a downloadable file or a compute service on which buyers can run their algorithm. Through **download**, buyers will be able to download the dataset. Through **compute**, buyers will be able to use the dataset in a compute-to-data environment.
- **Provider URL***
  Provider facilitates the asset download to buyers or for computing jobs and much more.
- **File***
  The direct URL of the dataset to be published. The file needs to be publicly accessible to be downloadable by buyers. If the file is hosted on services like Google Drive, the URL provided needs to point directly to the data asset file. Also, the file needs to have the proper permissions to be downloaded by anybody.
  Provider encrypts this field before publishing the asset on-chain.
- **Sample file**
  An optional field through which publishers provide a sample file of the dataset they want to publish. The buyers can access it before buying the dataset. This field is editable after the asset publication.
  Provider encrypts this field before publishing the asset on-chain.
- **Timeout***
  This field specifies how long the buyer can access the dataset after the dataset is purchased. This field is editable after the asset publication.



Access details

Step 3 - Pricing

The publisher needs to choose a pricing option for the asset before publishing the data asset. The pricing schema is not editable after the asset publication.

There are 2 pricing options for asset publication on Ocean Marketplace.

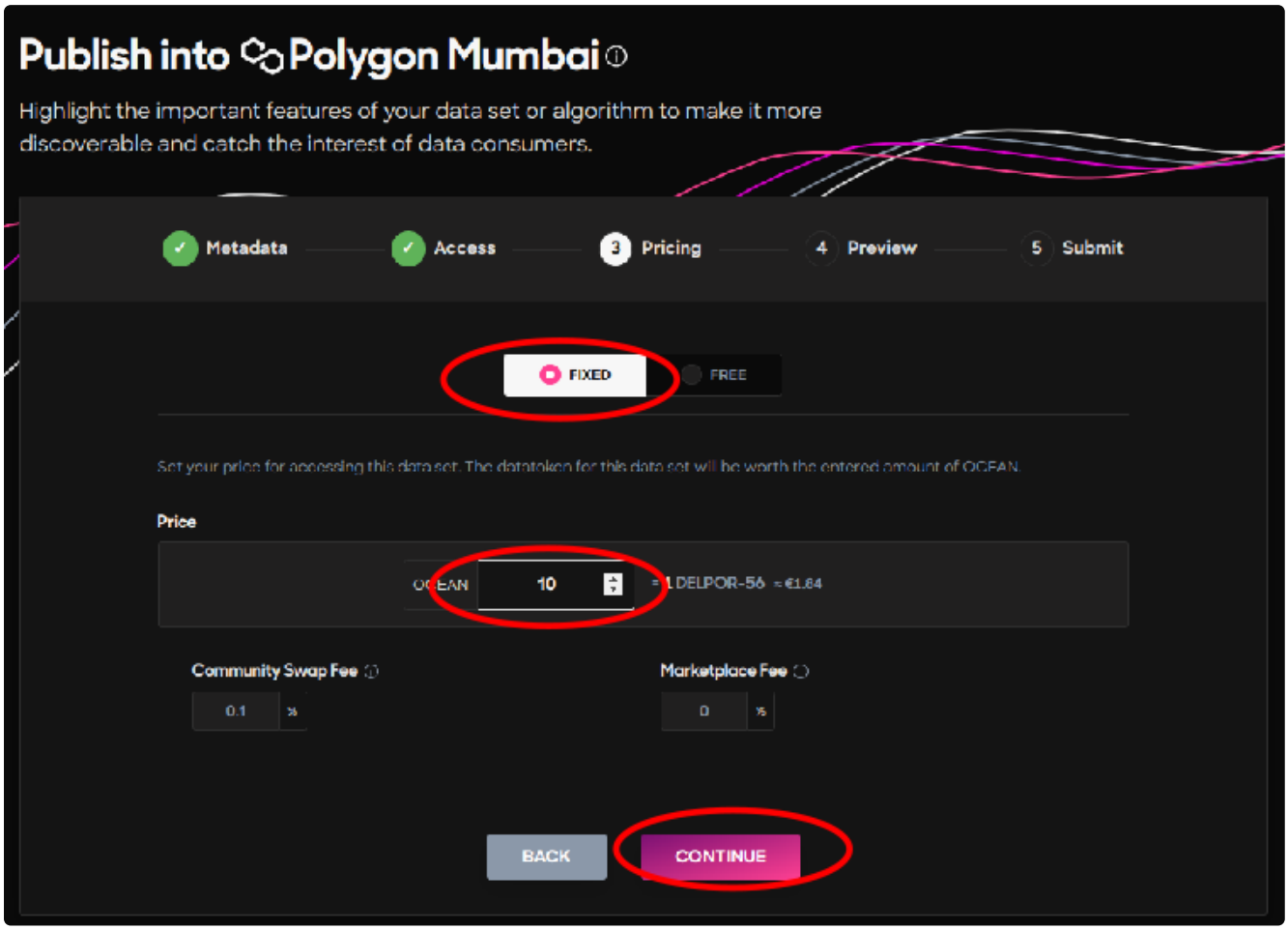
1. Fixed pricing
2. Free pricing

With the *fixed pricing* schema, the publisher sets the price that buyers will pay to download the data asset.

With the *free pricing* schema, the publisher provides an asset that is free to be downloaded by anyone.

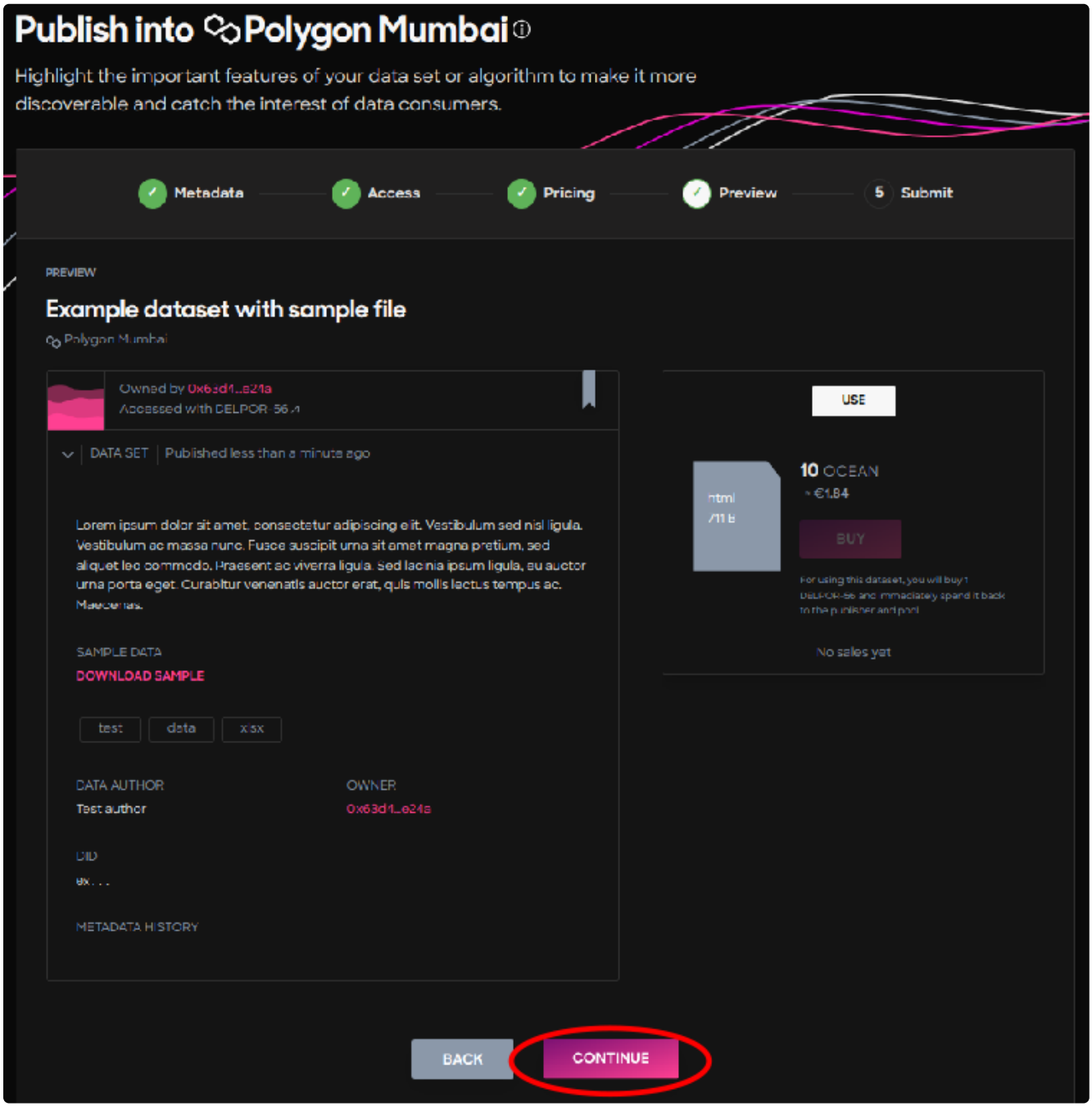
For more information on the pricing models, please refer this document.

For a deep dive into the fee structure, please refer to this document.



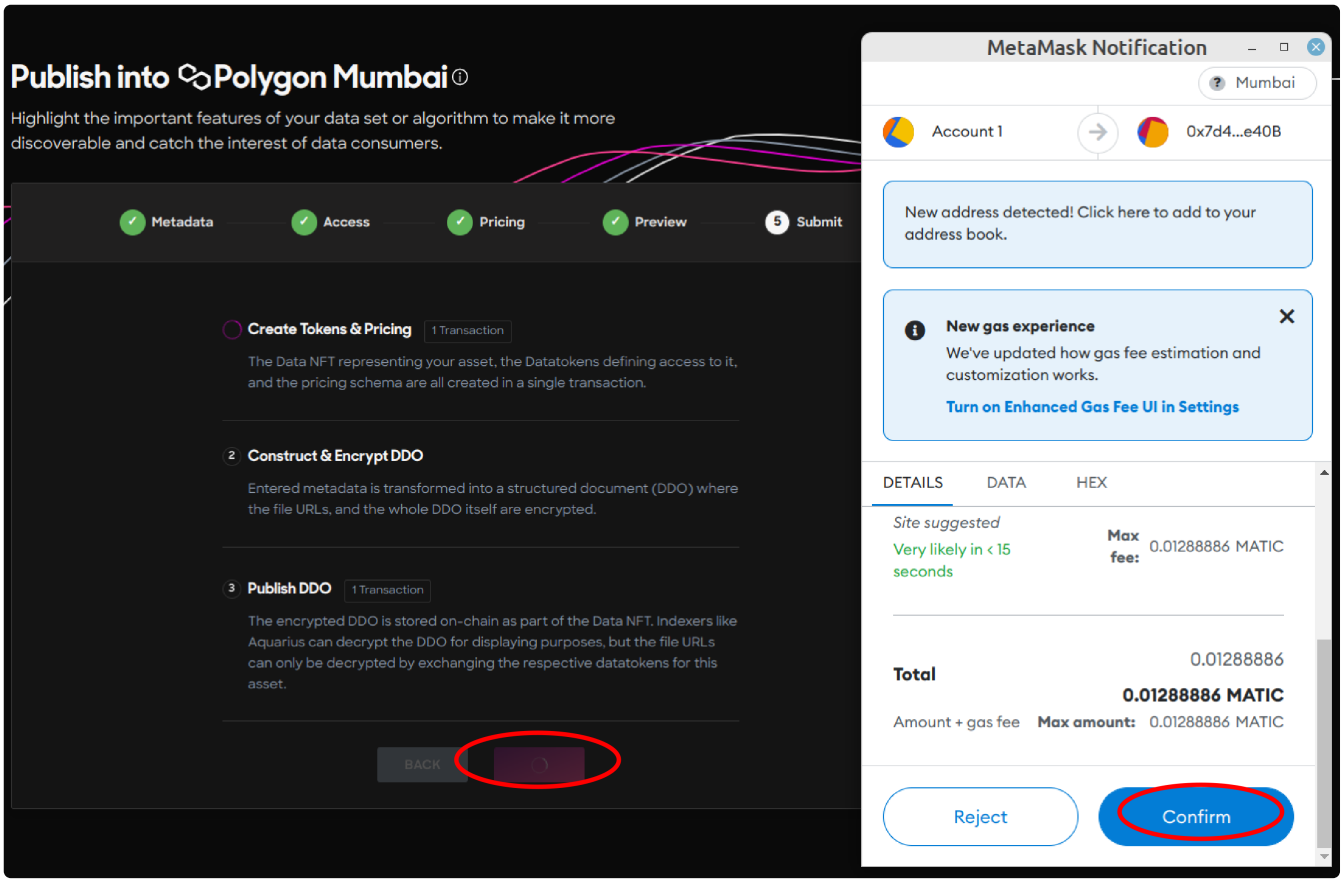
Asset pricing

Step 4 - Preview

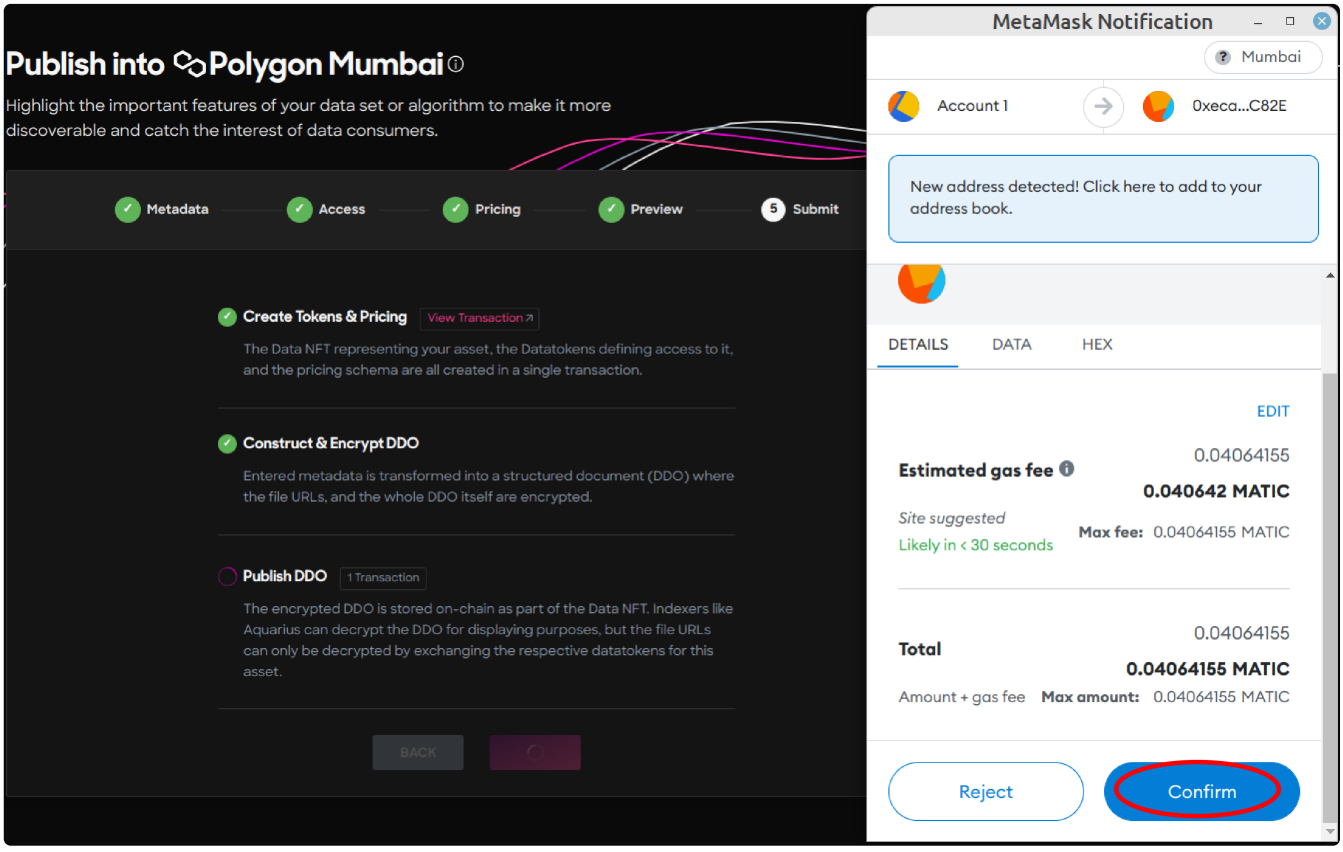


Preview

Step 5 - Blockchain transactions



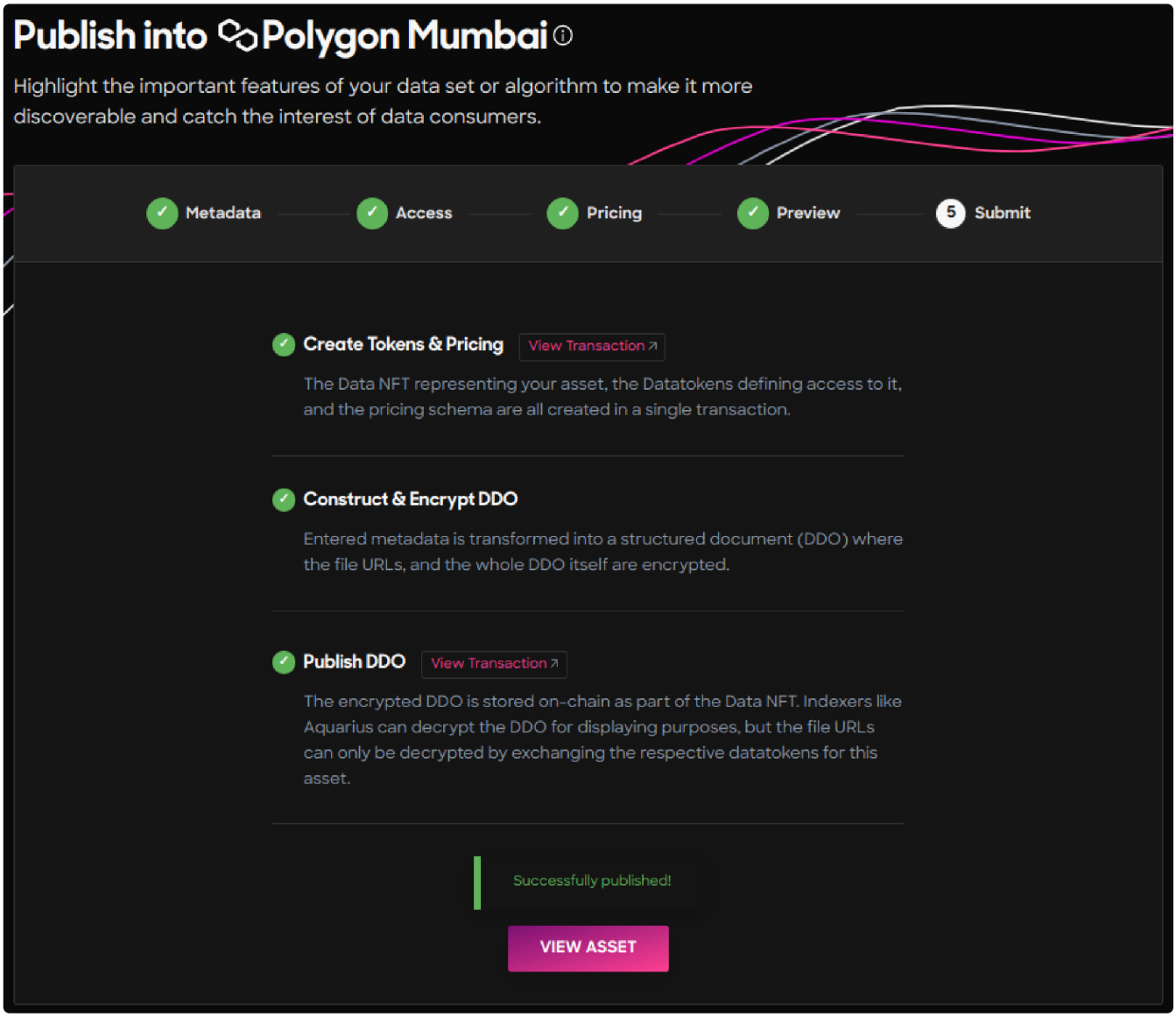
Transaction 1 - Deploy data NFT and datatoken



Transaction 2 - Deploy data NFT and datatoken

Confirmation

Now, the asset is successfully published and available in the Ocean Market.



Successful publish

On the profile page, the publisher has access to all his published assets.

Other Articles

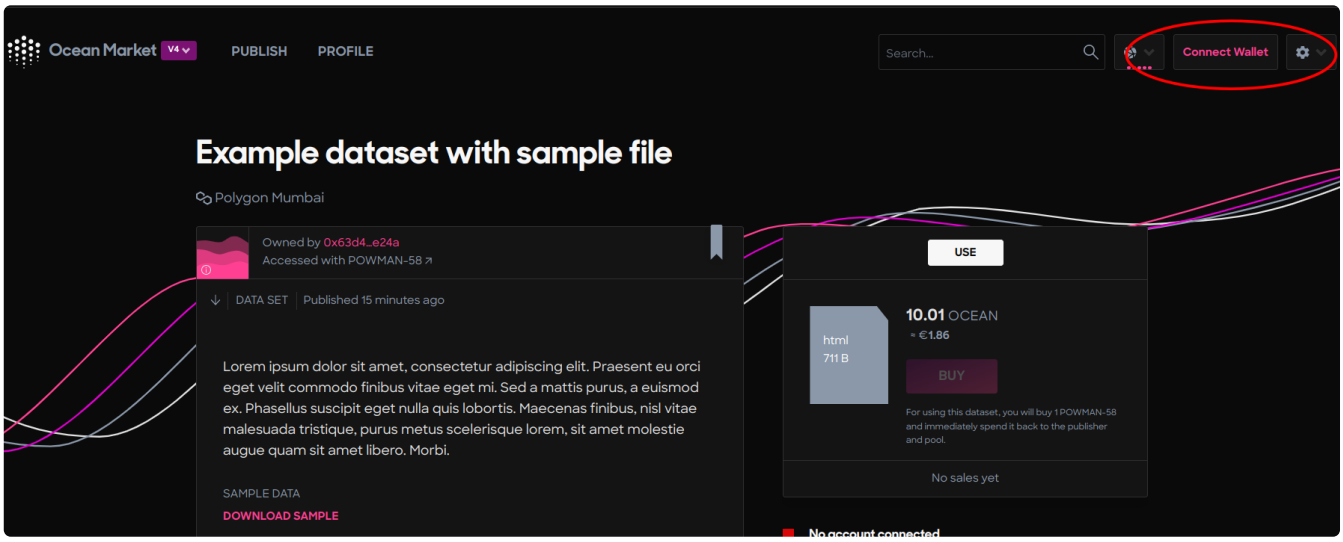
https://blog.oceanprotocol.com/on-selling-data-in-ocean-market-9afcfa1e6e43

Download a Data Asset

Tutorial to download assets using Ocean Market

Access marketplace

1. Go to Ocean Marketplace.
2. Search for the data asset. The Ocean Marketplace provides features to search the Data/Algorithms by text, and users can also sort the result by published date.
3. Connect wallet.



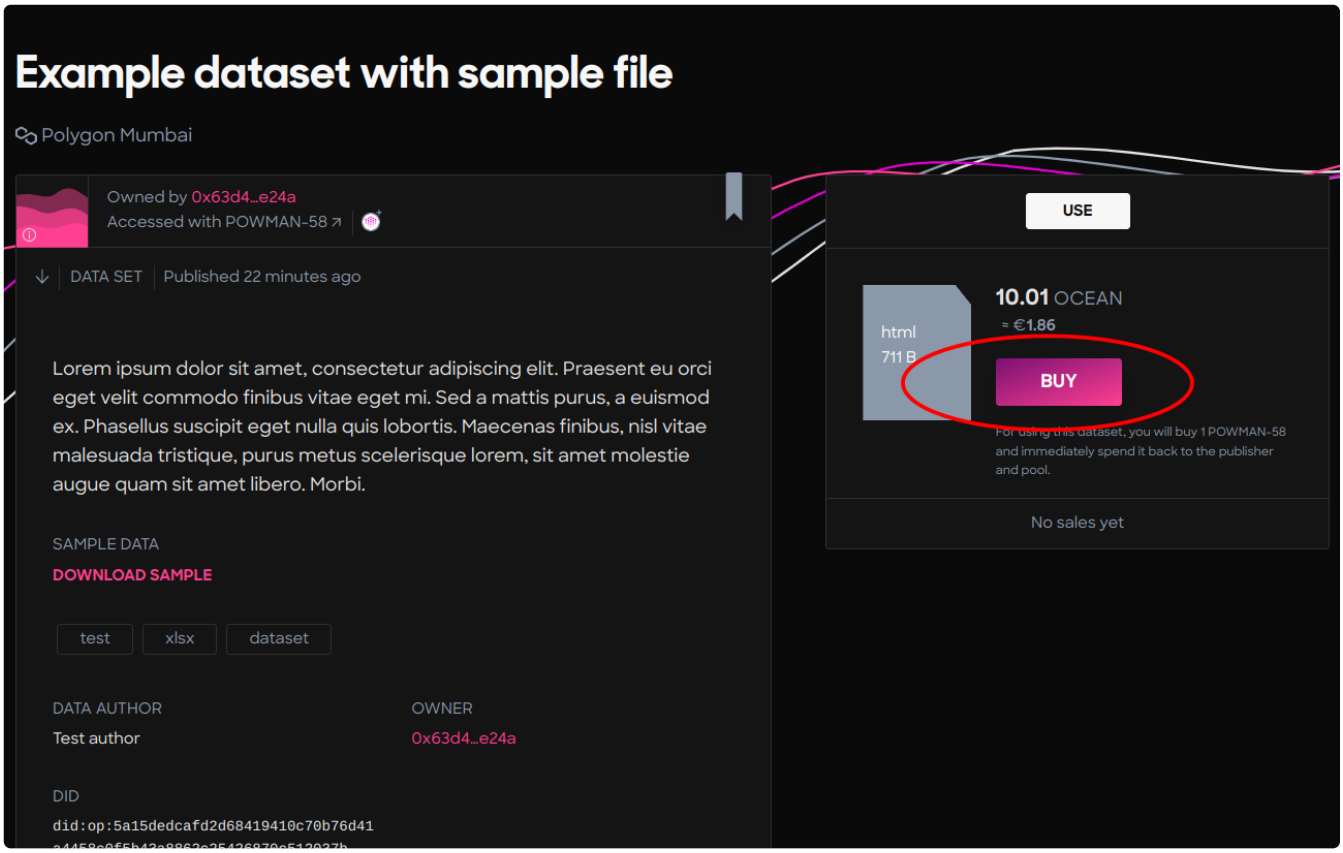
Connect wallet

```
In this tutorial, we will be using the Polygon Mumbai test network.
```

Tutorial

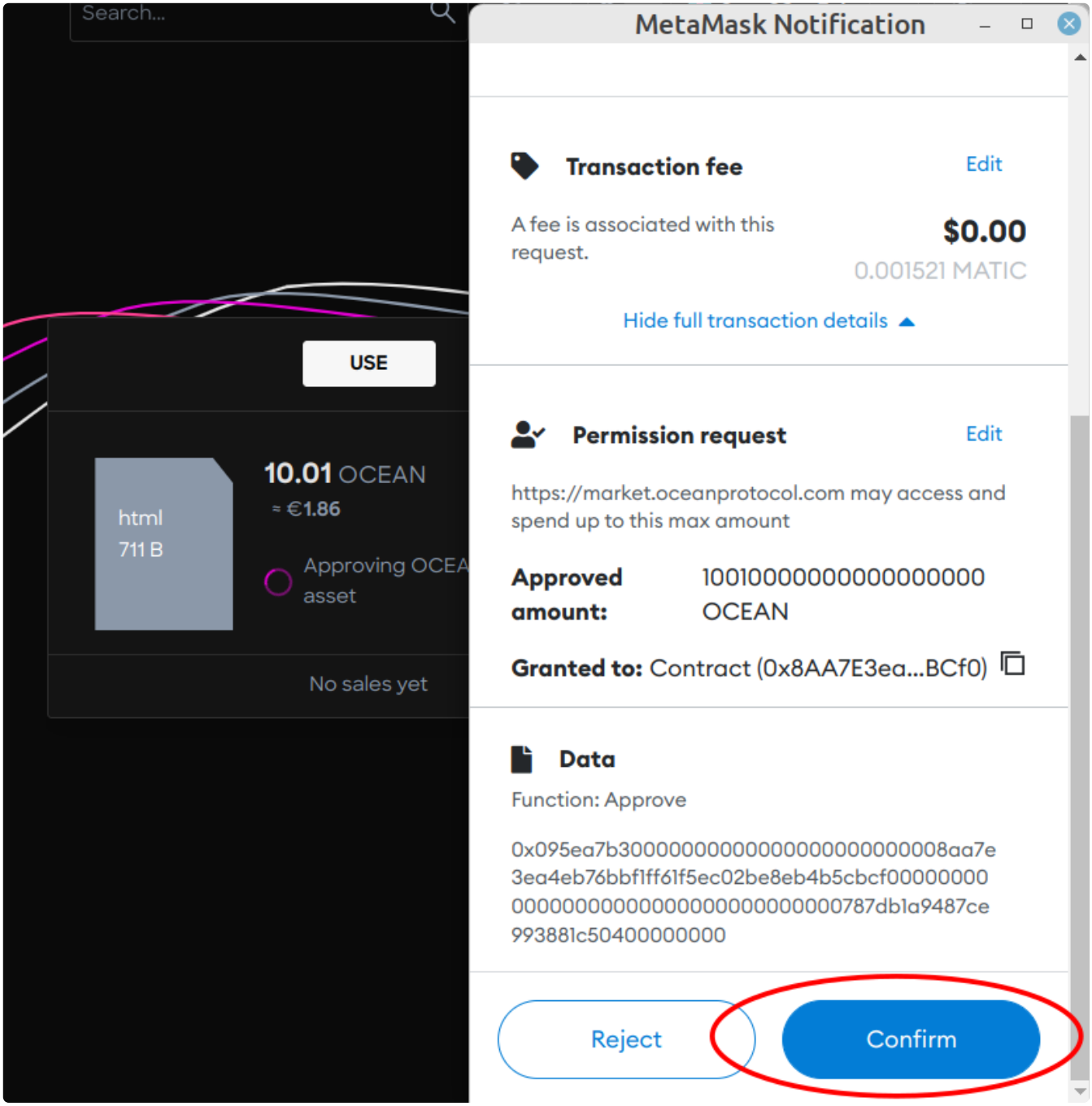
Step 1 - Click buy

The buy button is enabled only if the connected wallet address has enough OCEAN tokens to exchange them with 1 datatoken.



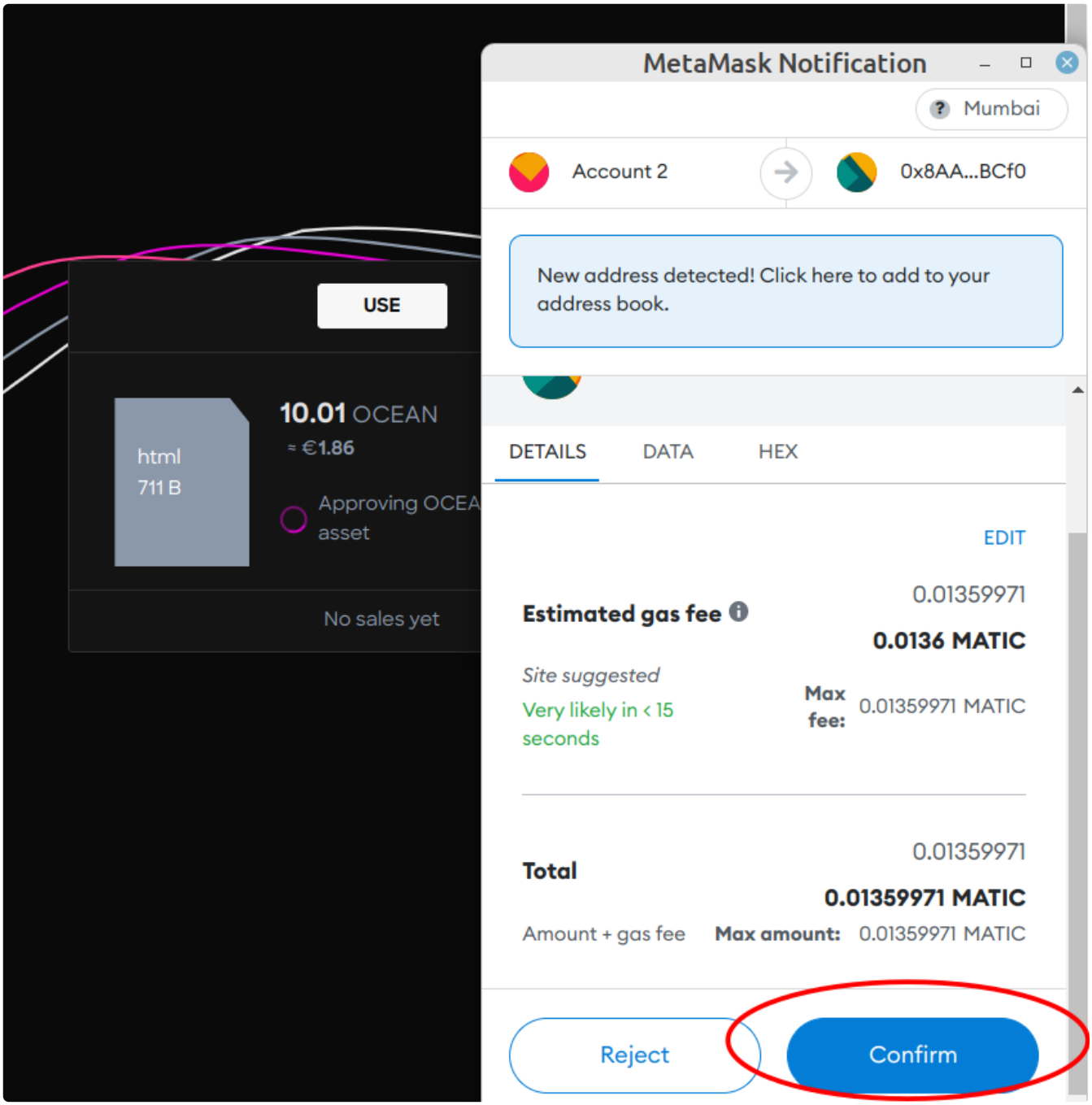
Buy

Step 2 - Allow access to OCEAN token(s)



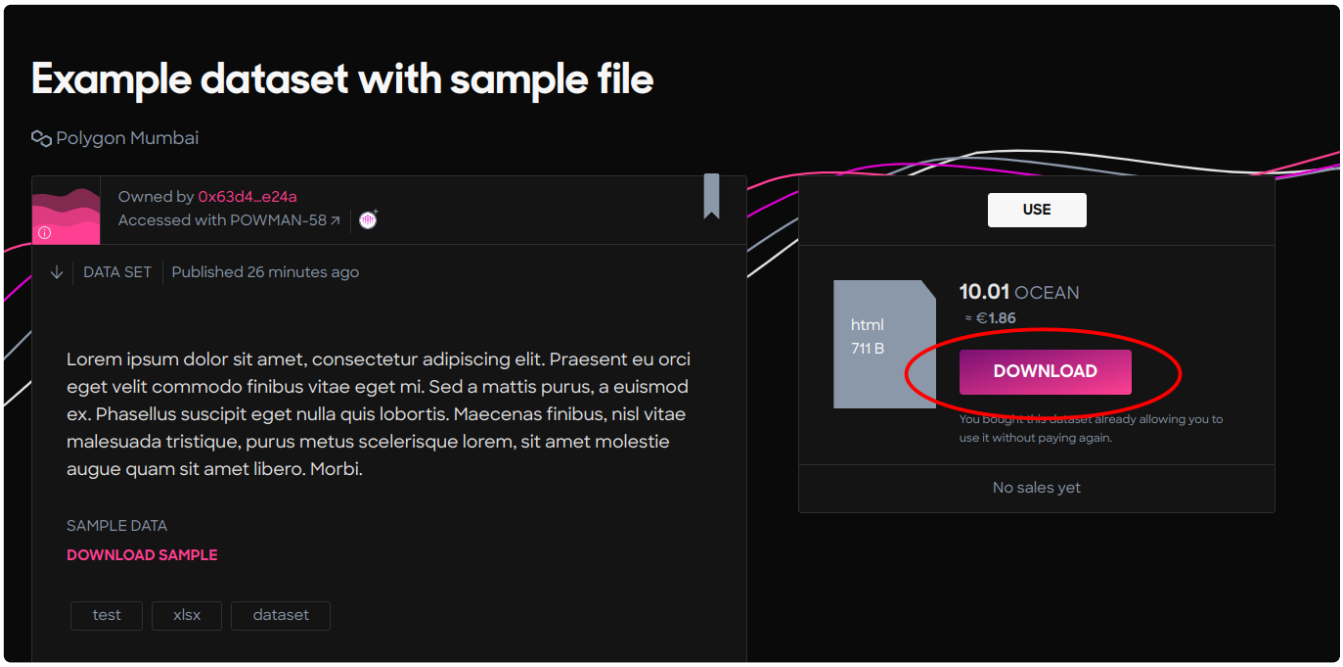
Transaction 1: Permissions to access OCEAN tokens

Step 3 - Buy a datatoken by exchanging it with OCEAN token(s)



Transaction 2: Buy datatoken

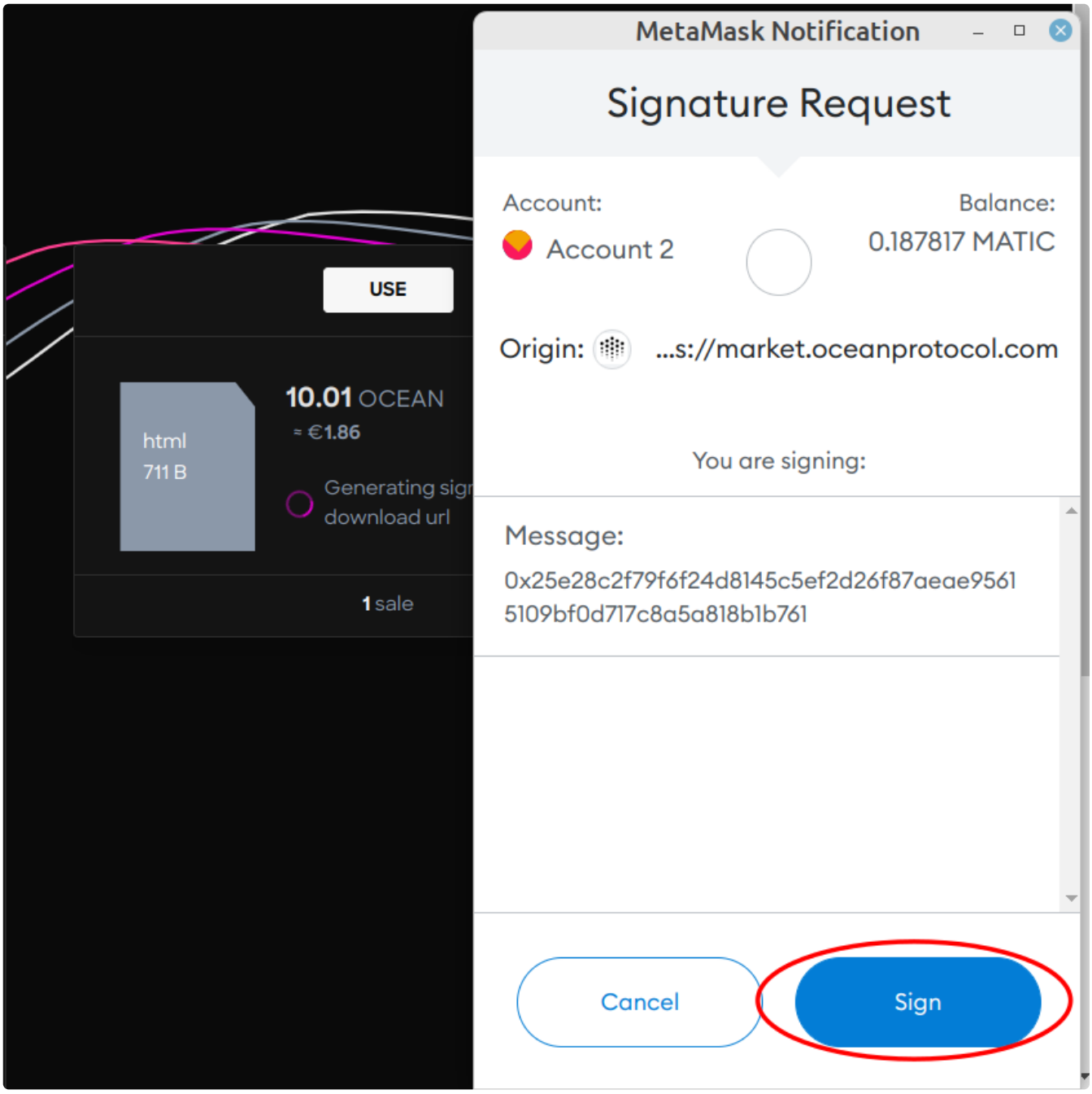
Step 4 - Click download



Download asset

Step 5 - Sign message

After signing the message, the file download will start.


MetaMask Notification
Signature Request
Account:
Account 2
Balance:
0.187817 MATIC
Origin:
...s://market.oceanprotocol.com
You are signing:
Message:
0x25e28c2f79f6f24d8145c5ef2d26f87aeae9561
5109bf0d717c8a5a818b1b761
Cancel
Sign
USE
10.01 OCEAN
≈ €1.86
html
711 B
Generating sig
download url
1 sale

Sign

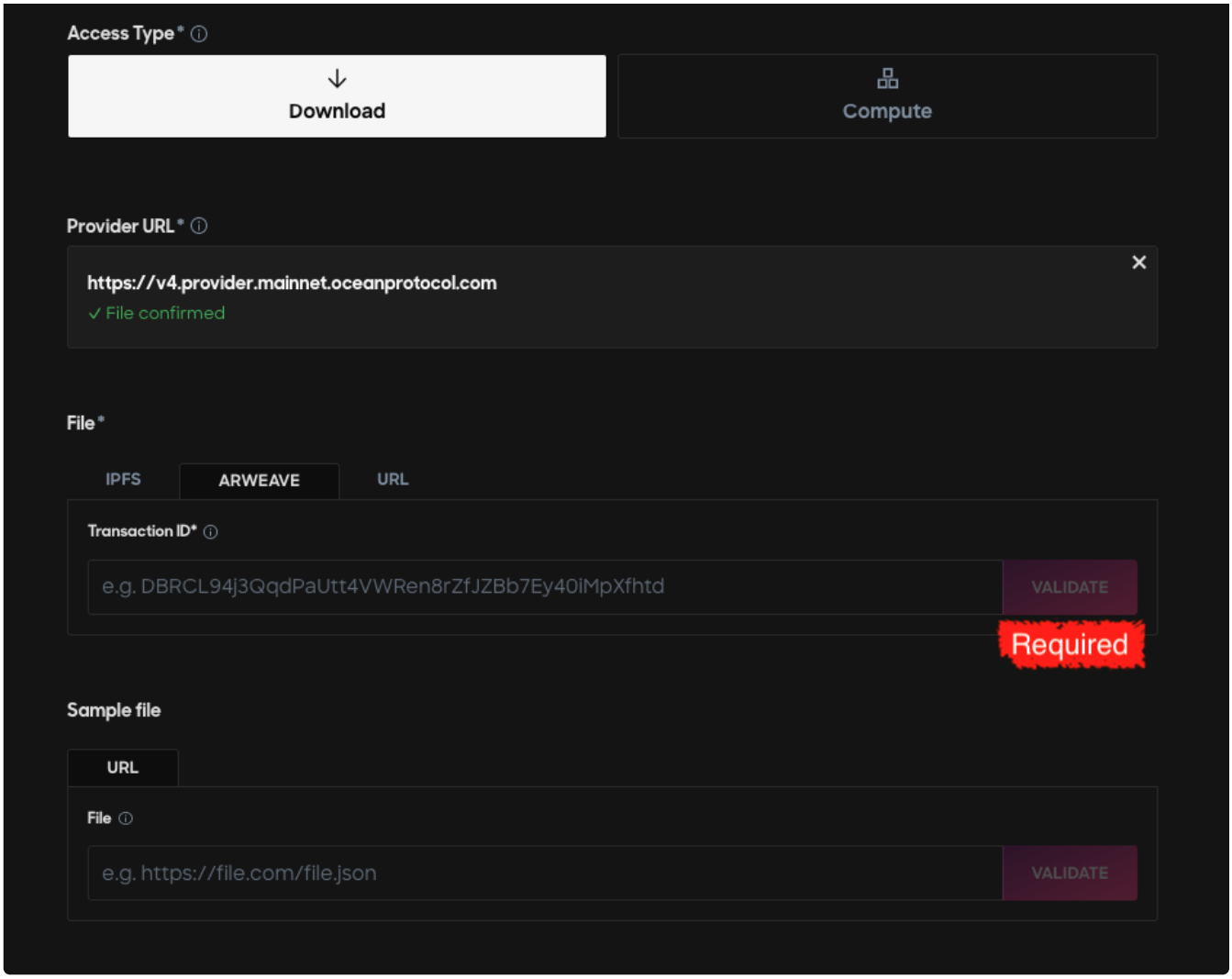
Publishing with Hosting Services

Tutorial to publish assets using hosting services like Arweave, AWS, and Azure.

Overview

To publish on the Ocean Marketplace, publishers must first host their assets. It is up to the asset publisher to decide where to host the asset. For example, a publisher can store the content on decentralized storage like Arweave or choose a centralized solution like their AWS server, private cloud server, or other third-party hosting services. Through publishing, the information required to access the asset is encrypted and stored as a part of DDO on the blockchain. Buyers don't have access directly to this information, but they interact with the Provider, which decrypts it and acts as a proxy to serve the asset. The DDO only stores the location of the file, which is accessed on-demand by the Provider. Implementing a security policy that allows only the Provider to access the file and blocks requests from other unauthorized actors is recommended. One of the possible ways to achieve this is to allow only the Provider's IP address to access the data. But, not all hosting services provide this feature. So, the publishers must consider the security features while choosing a hosting service.

On Ocean Marketplace, a publisher must provide the asset information during the publish step in the field shown in the below image. The information is a `link` for a classic URL, a `transaction ID` for a file stored on Arweave or a `CID` for an IPFS file.



Publish - File URL field

Publishers can choose any hosting service of their choice. The below section explains how to use commonly used hosting services with Ocean Marketplace.

⚠ Note **Please use a proper hosting solution to keep your files.** Systems like `Google Drive` are not specifically designed for this use case. They include various virus checks and rate limiters that prevent the `Provider` to download the asset once it was purchased.

Decentralized hosting

Arweave

Arweave is a global, permanent, and decentralized data storage layer that allows you to store documents and applications forever. Arweave is different from other decentralized storage solutions in that there is only one up-front cost to upload each file.

Step 1 - Get a new wallet and AR tokens

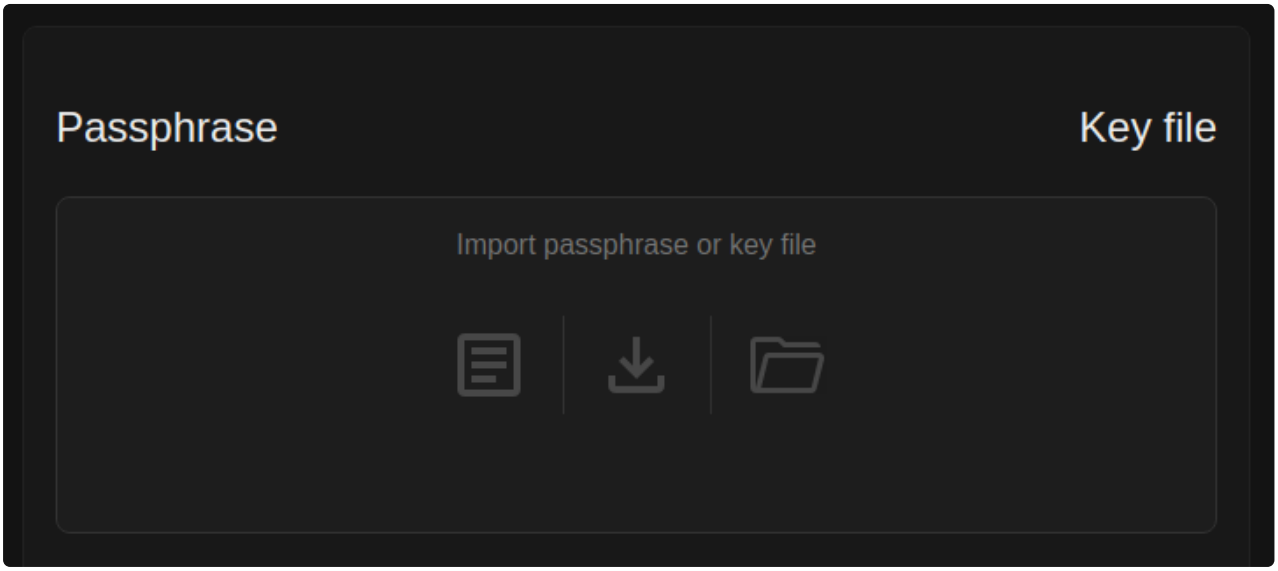
Download & save a new wallet (JSON key file) and receive a small amount of AR tokens for free using the Arweave faucet. If you already have an Arweave browser wallet, you can skip to Step 3.

At the time of writing, the faucet provides 0.02 AR which is more than enough to upload a file.

If at any point you need more AR tokens, you can fund your wallet from one of Arweave's supported exchanges.

Step 2 - Load the key file into the arweave.app web wallet

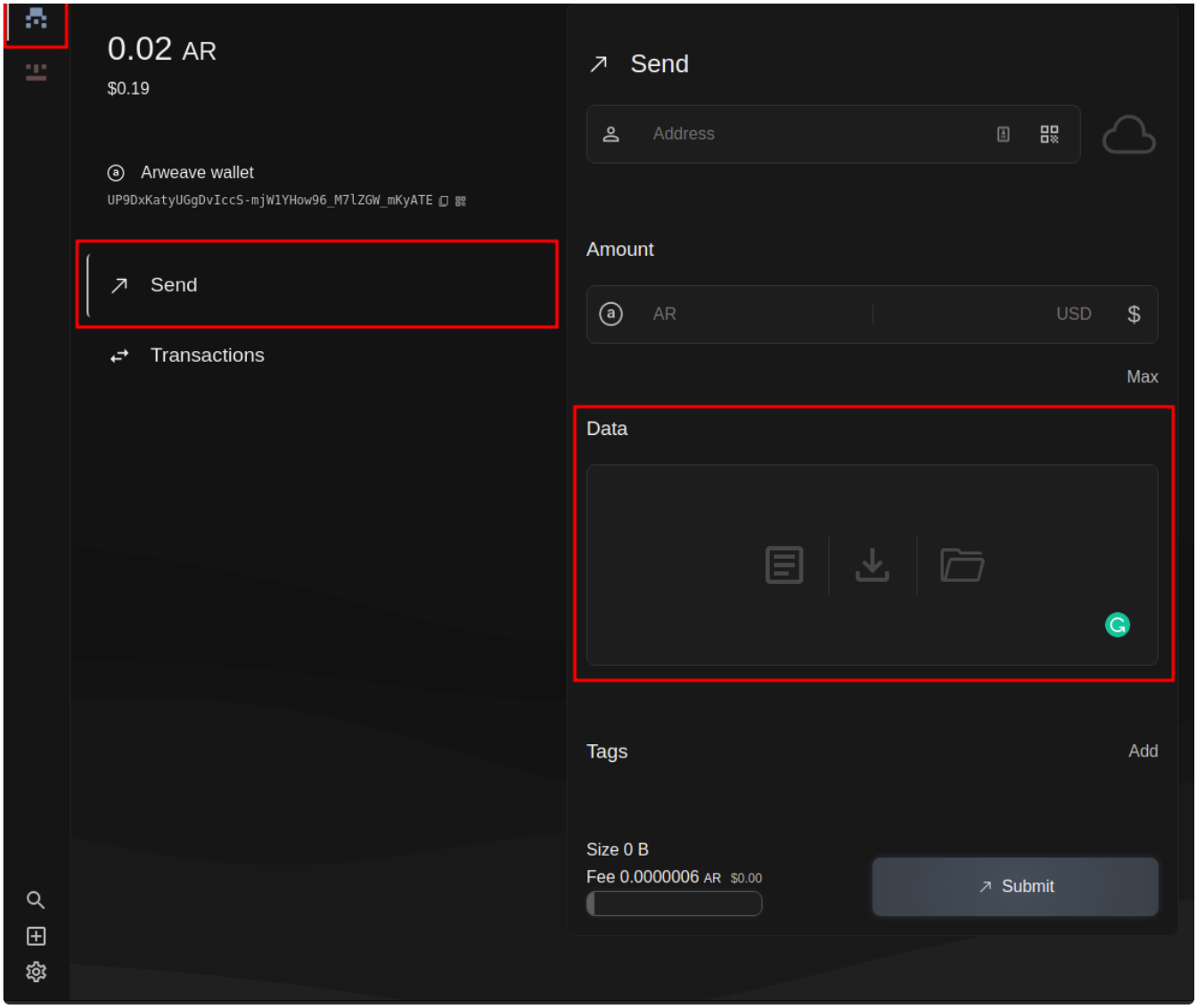
Open arweave.app in a browser. Select the '+' icon in the bottom left corner of the screen. Import the JSON key file from step 1.



Arweave.app import key file

Step 3 - Upload file

Select the newly imported wallet by clicking the "blockies" style icon in the top left corner of the screen. Select **Send.** Click the **Data** field and select the file you wish to upload.



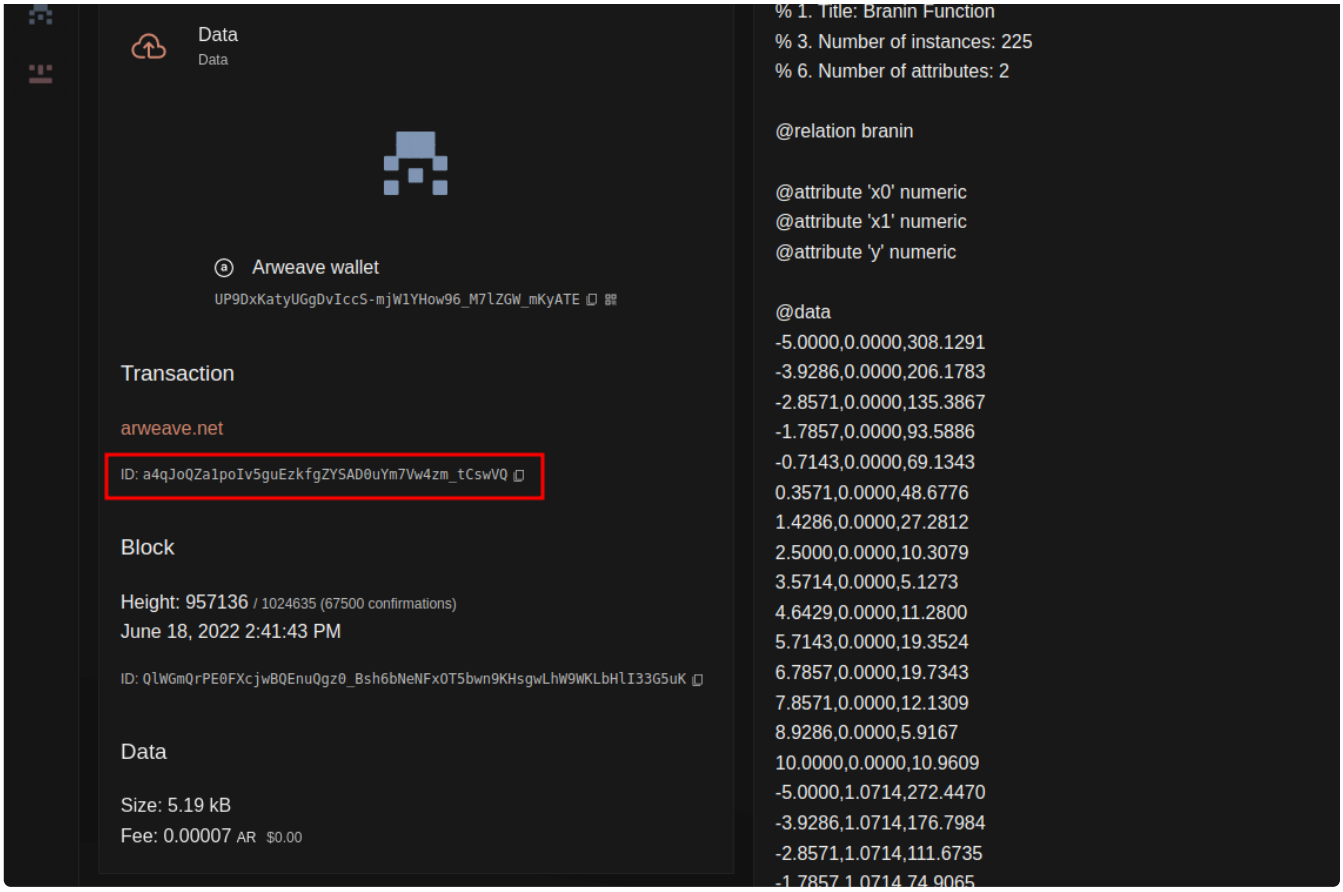
Arweave.app upload file

The fee in AR tokens will be calculated based on the size of the file and displayed near the bottom middle part of the screen. Select **Submit** to submit the transaction.

After submitting the transaction, select **Transactions** and wait until the transaction appears and eventually finalizes. This can take over 5 minutes so please be patient.

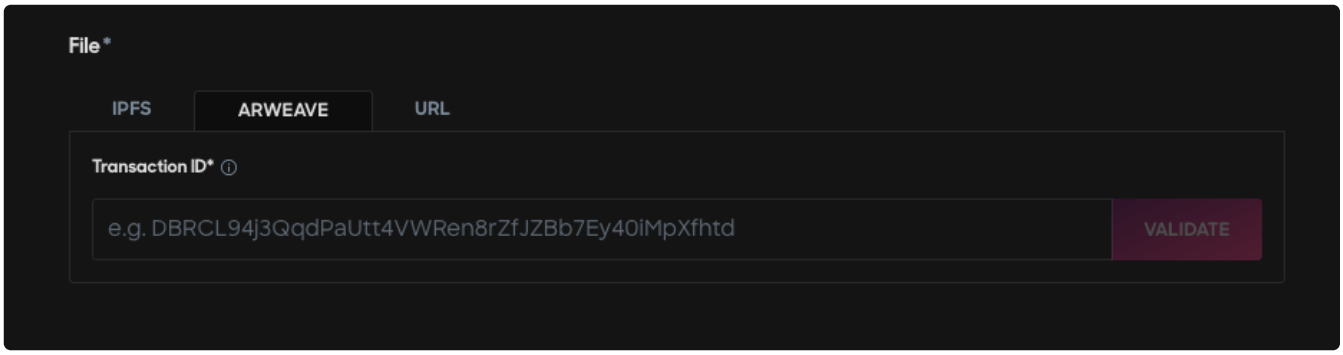
Step 4 - Copy the transaction ID

Once the transaction finalizes, select it, and copy the transaction ID.



Arweave.app transaction ID

Step 5 - Publish the asset with the transaction ID



Ocean Market - Publish with arweave transaction ID

Centralized hosting

AWS

AWS provides various options to host data and multiple configuration possibilities. Publishers are required to do their research and decide what would be the right choice. The below steps provide one of the possible ways to host data using an AWS S3 bucket and publish it on Ocean Marketplace.

Prerequisite

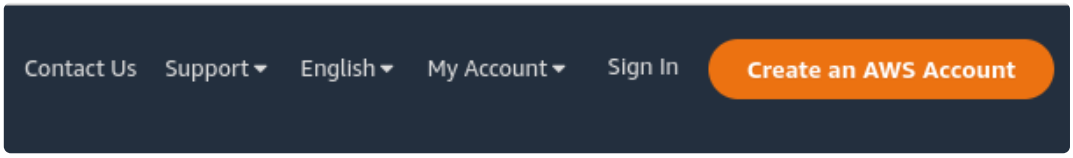
Create an account on AWS. Users might also be asked to provide payment details and billing addresses

that are out of this tutorial's scope.

Step 1 - Create a storage account

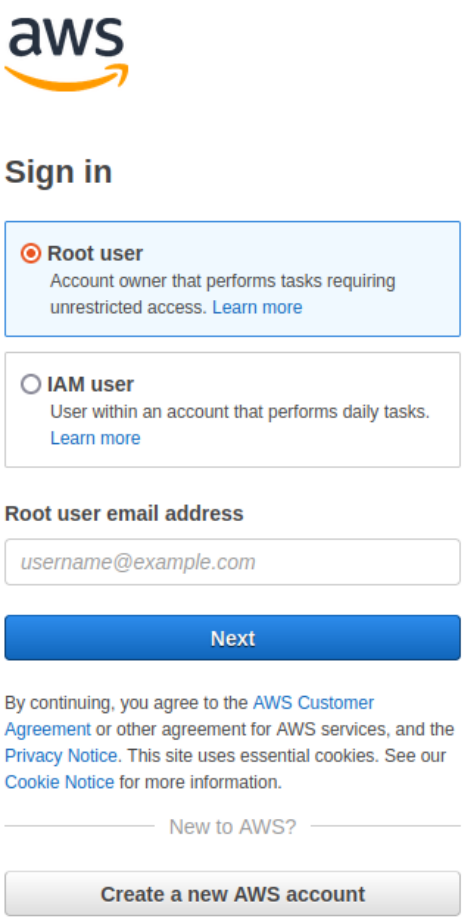
Go to AWS portal

Go to the AWS portal for S3: https://aws.amazon.com/s3/ and select from the upper right corner `Create an AWS account` as shown below.



Create an account - 1

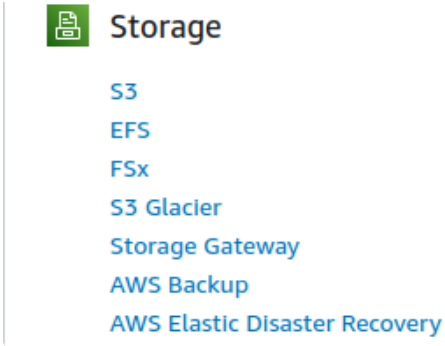
Fill in the details



)

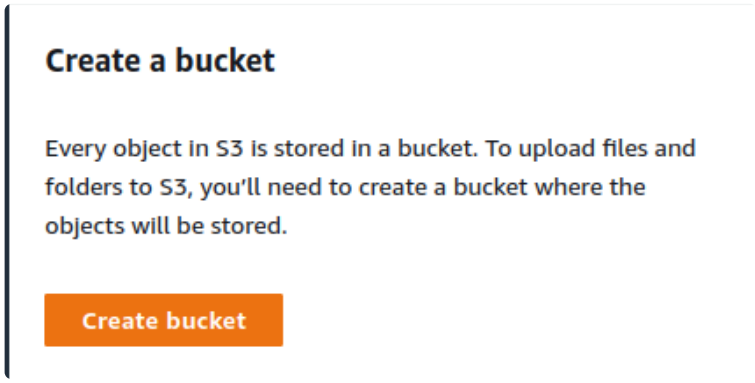
Create a bucket

After logging into the new account, search for the available services and select `S3` type of storage.



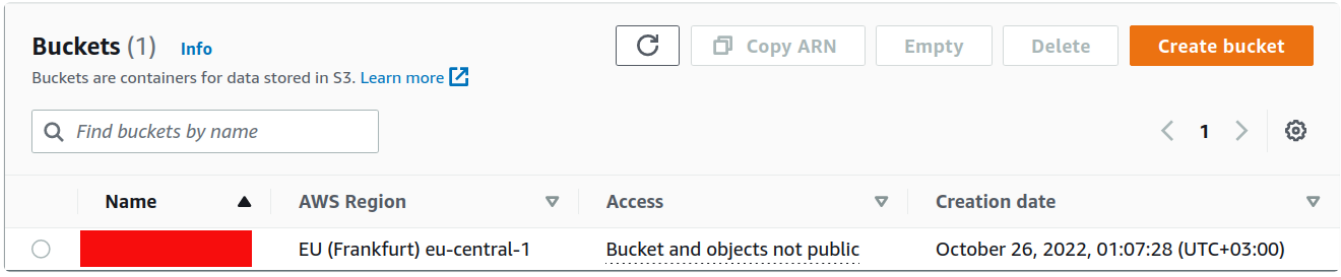
Create an account - 3

To create an S3 bucket, choose `Create bucket`.



Create an account - 4

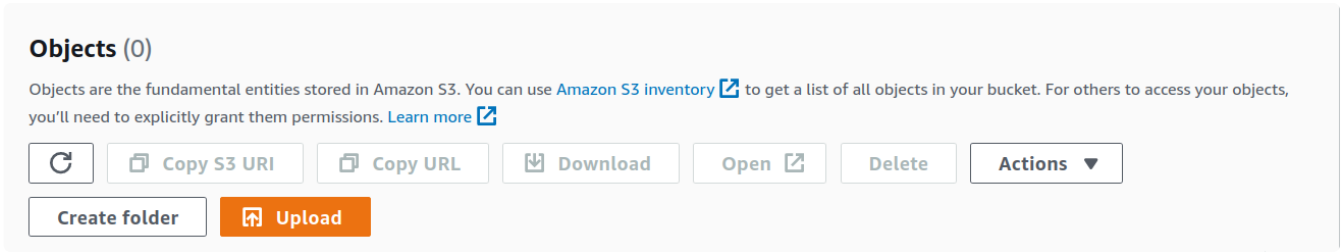
Fill in the form with the necessary information. Then, the bucket is up & running.



Create an account - 5

Step 2 - Upload asset on S3 bucket

Now, the asset can be uploaded by selecting the bucket name and choosing `Upload` in the `Objects` tab.

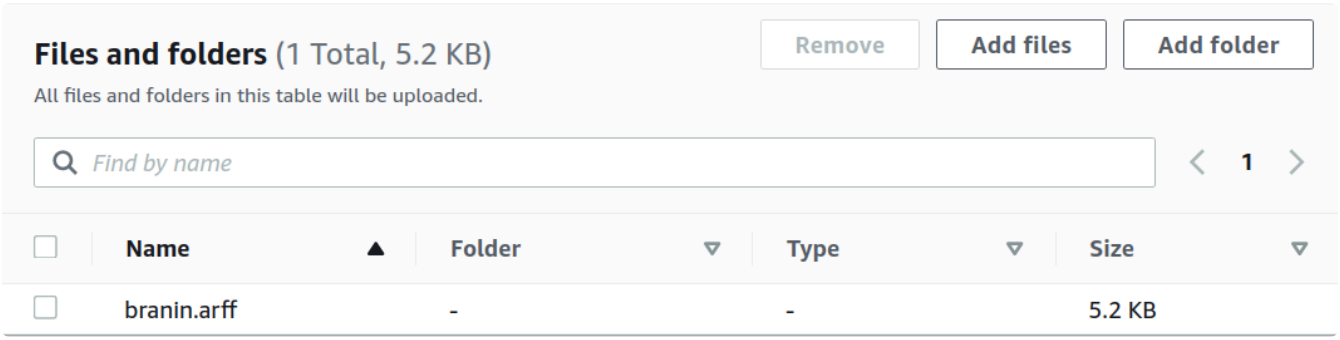


Upload asset on S3 bucket - 1

Add files to the bucket

Get the files and add them to the bucket.

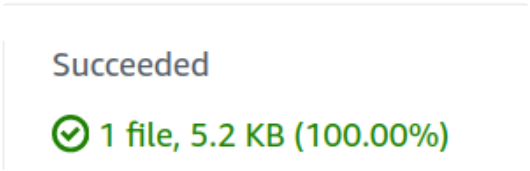
The file is an example used in multiple Ocean repositories, and it can be found here.



Upload asset on S3 bucket - 3

The permissions and properties can be set afterward, for the moment keep them as default.

After selecting `Upload`, make sure that the status is `Succeeded`.



Upload asset on S3 bucket - 4

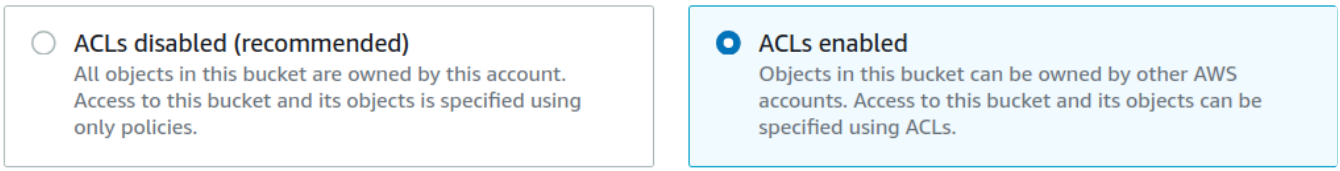
Step 3 - Access the Object URL on S3 Bucket

By default, the permissions of accessing the file from the S3 bucket are set to private. To publish an asset on the market, the S3 URL needs to be public. This step shows how to set up access control policies to grant permissions to others.

Editing permissions

Go to the `Permissions` tab and select `Edit` and then uncheck `Block all public access` boxes to give everyone read access to the object and click `Save`.

If editing the permissions is unavailable, modify the `Object Ownership` by enabling the ACLs as shown below.



Access the Object URL on S3 Bucket - 1

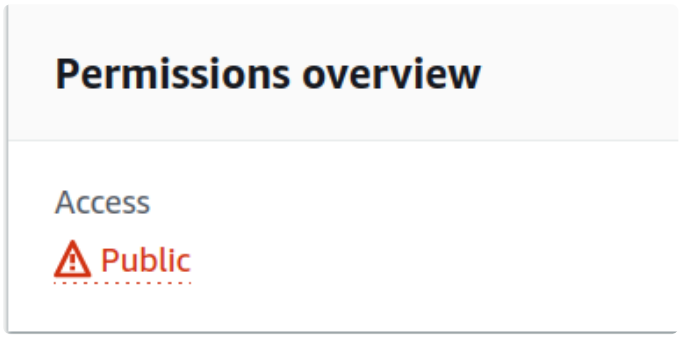
Modifying bucket policy

To have the bucket granted public access, its policy needs to be modified likewise.

Note that the `<BUCKET-NAME>` must be chosen from the personal buckets dashboard.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Public S3 Bucket",
      "Principal": "*",
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::<BUCKET-NAME>/*"
    }
  ]
}
```

After saving the changes, the bucket should appear as `Public` access.



Access the Object URL on S3 Bucket - 2

Verify the object URL on public access

Select the file from the bucket that needs verification and select `Open`. Now download the file on your system.

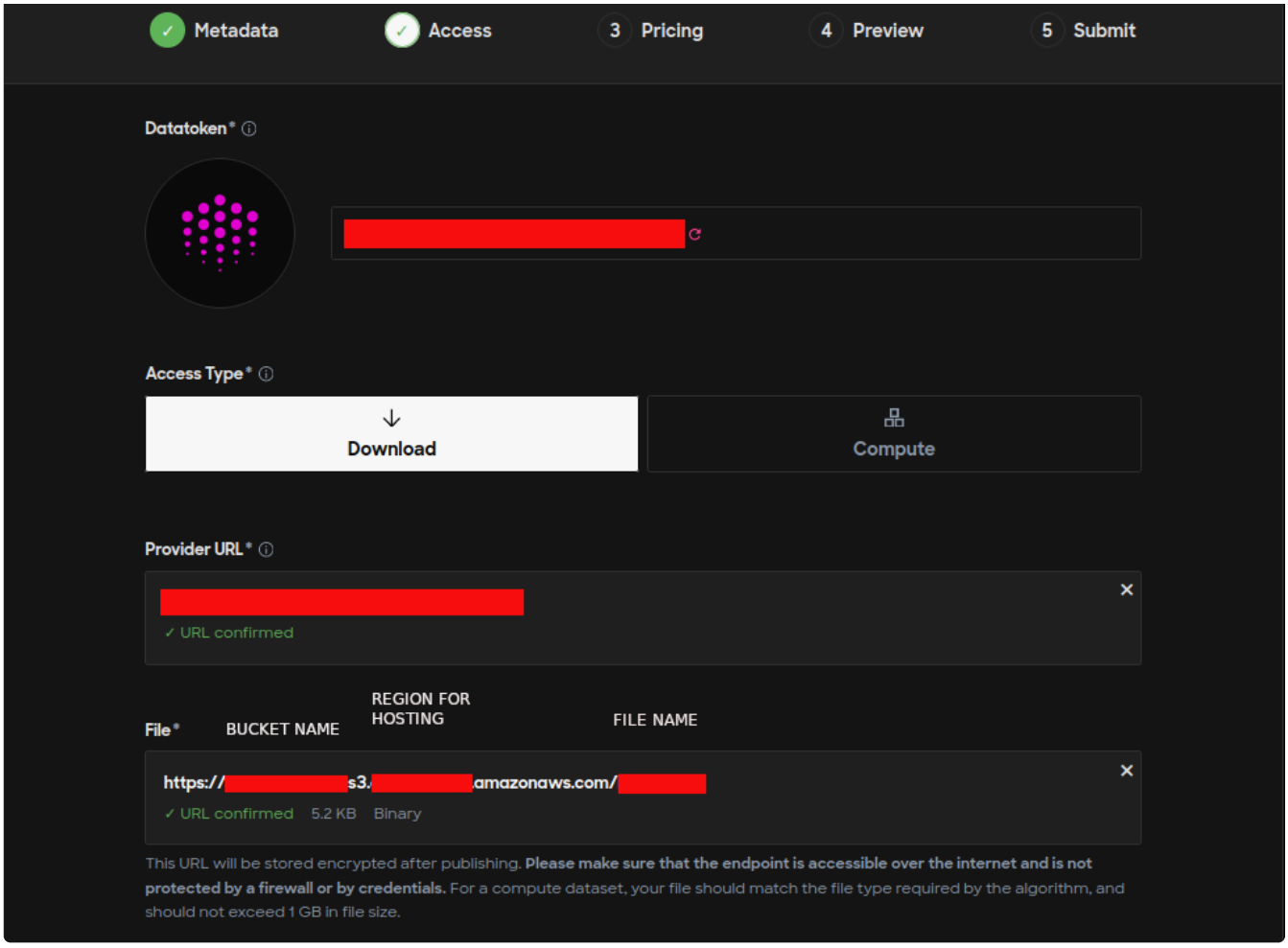
Access the Object URL on S3 Bucket - 3

Step 4 - Get the S3 Bucket Link & Publish Asset on Market

Now that the S3 endpoint has public access, the asset will be hosted successfully.

Go to Ocean Market to complete the form for asset creation.

Copy the `Object URL` that can be found at `Object Overview` from the AWS S3 bucket and paste it into the `File` field from the form found at step 2 as it is illustrated below.



Get the S3 Bucket Link & Publish Asset on Market - 1

Azure storage

Azure provides various options to host data and multiple configuration possibilities. Publishers are required to do their research and decide what would be the right choice. The below steps provide one of the possible ways to host data using Azure storage and publish it on Ocean Marketplace.

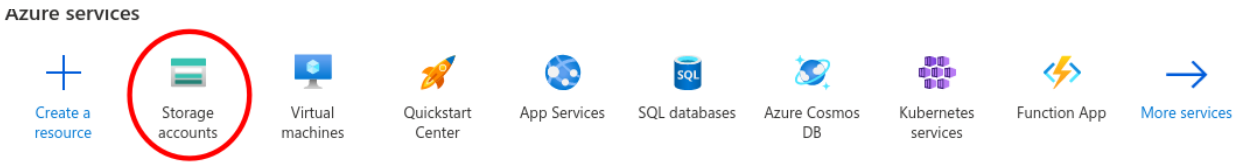
Prerequisite

Create an account on Azure. Users might also be asked to provide payment details and billing addresses that are out of this tutorial's scope.

Step 1 - Create a storage account

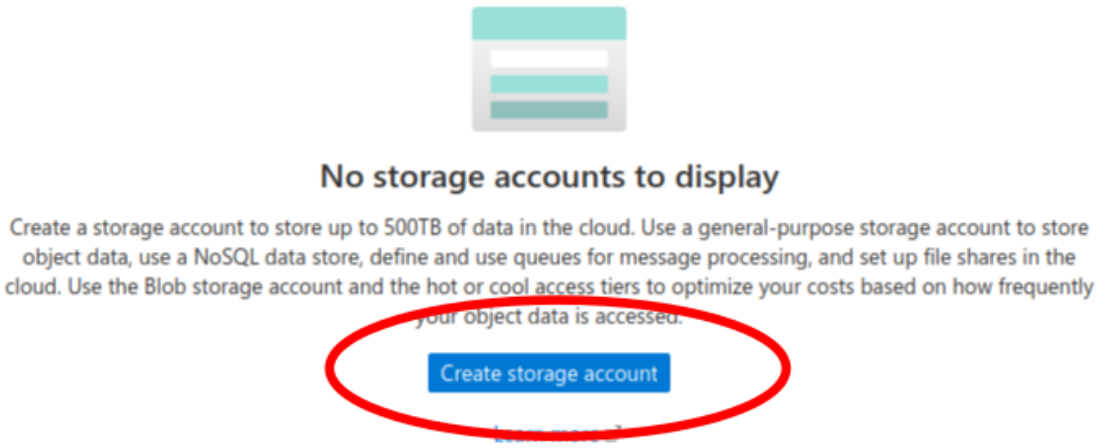
Go to Azure portal

Go to the Azure portal: https://portal.azure.com/#home and select `Storage accounts` as shown below.



Create a storage account - 1

Create a new storage account



Create a storage account - 2

Fill in the details

Home > Storage accounts >

Create a storage account ...

Basics | Advanced | Networking | Data protection | Encryption | Tags | Review + create

redundant. Azure Storage includes Azure Blobs (objects), Azure Data Lake Storage Gen2, Azure Files, Azure Queues, and Azure Tables. The cost of your storage account depends on the usage and the options you choose below. Learn more about Azure storage accounts

Project details

Select the subscription in which to create the new storage account. Choose a new or existing resource group to organize and manage your storage account together with other resources.

Subscription *

Resource group *

Create new

Instance details

If you need to create a legacy storage account type, please click here.

Storage account name * joypen

Region *

Performance *

- (•) **Standard:** Recommended for most scenarios (general-purpose v2 account)
- ( ) **Premium:** Recommended for scenarios that require low latency.

Redundancy * Geo-redundant storage (GRS)

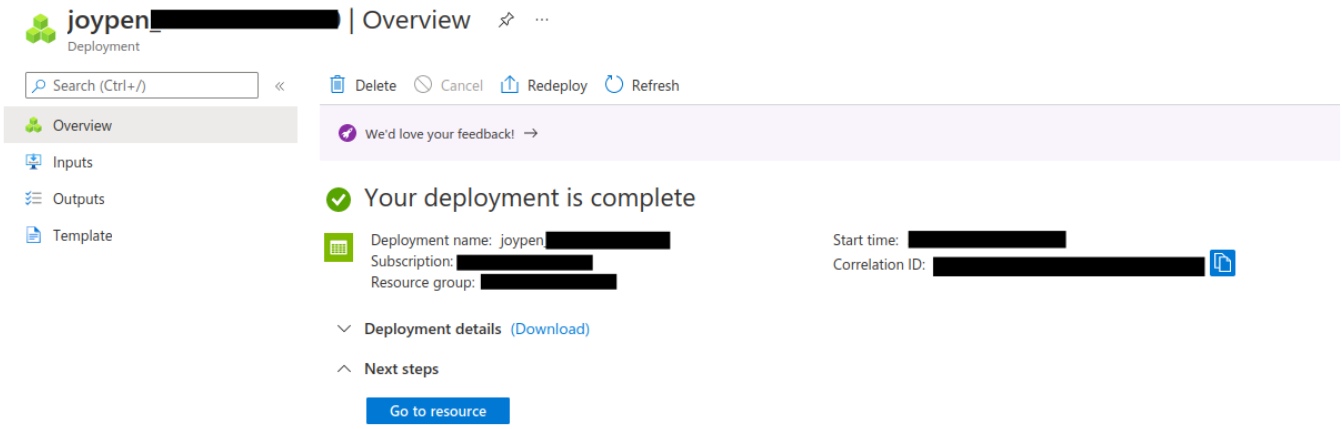
[x] Make read access to data available in the event of regional unavailability.

Review + create | < Previous | Next : Advanced >

Add details

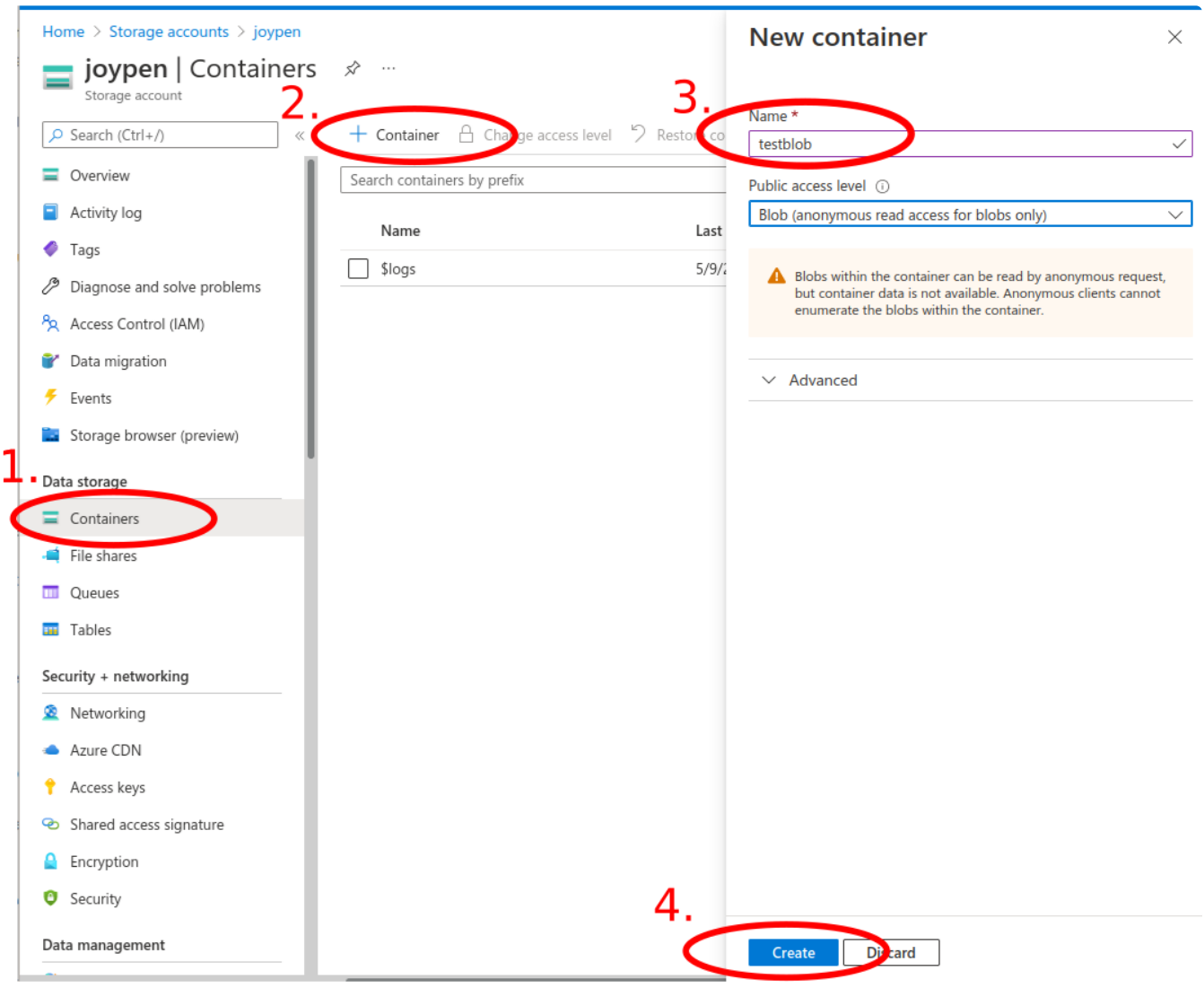
Storage account created

Home >



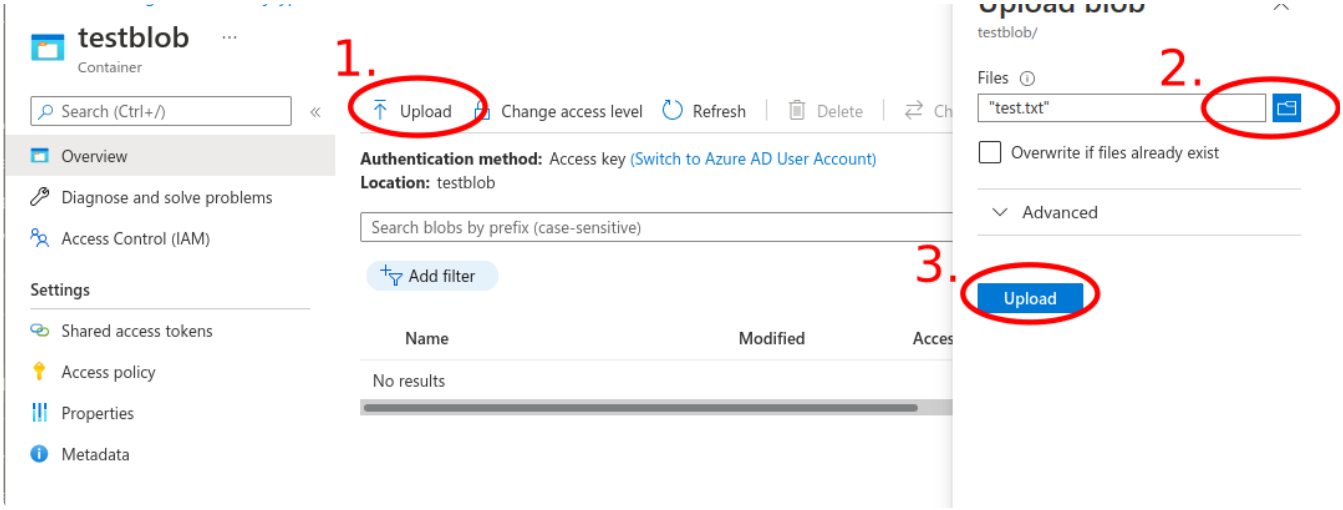
Storage account created

Step 2 - Create a blob container



Create a blob container

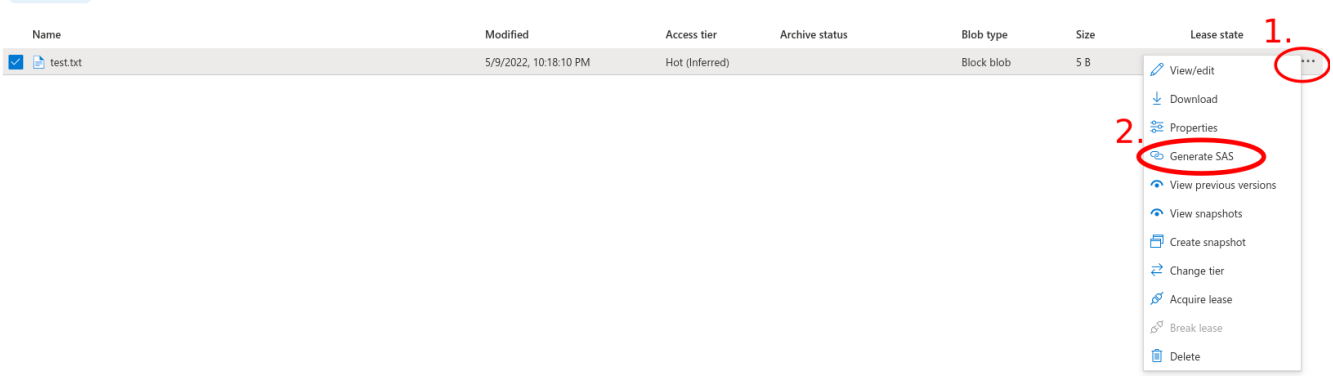
Step 3 - Upload a file



Upload a file

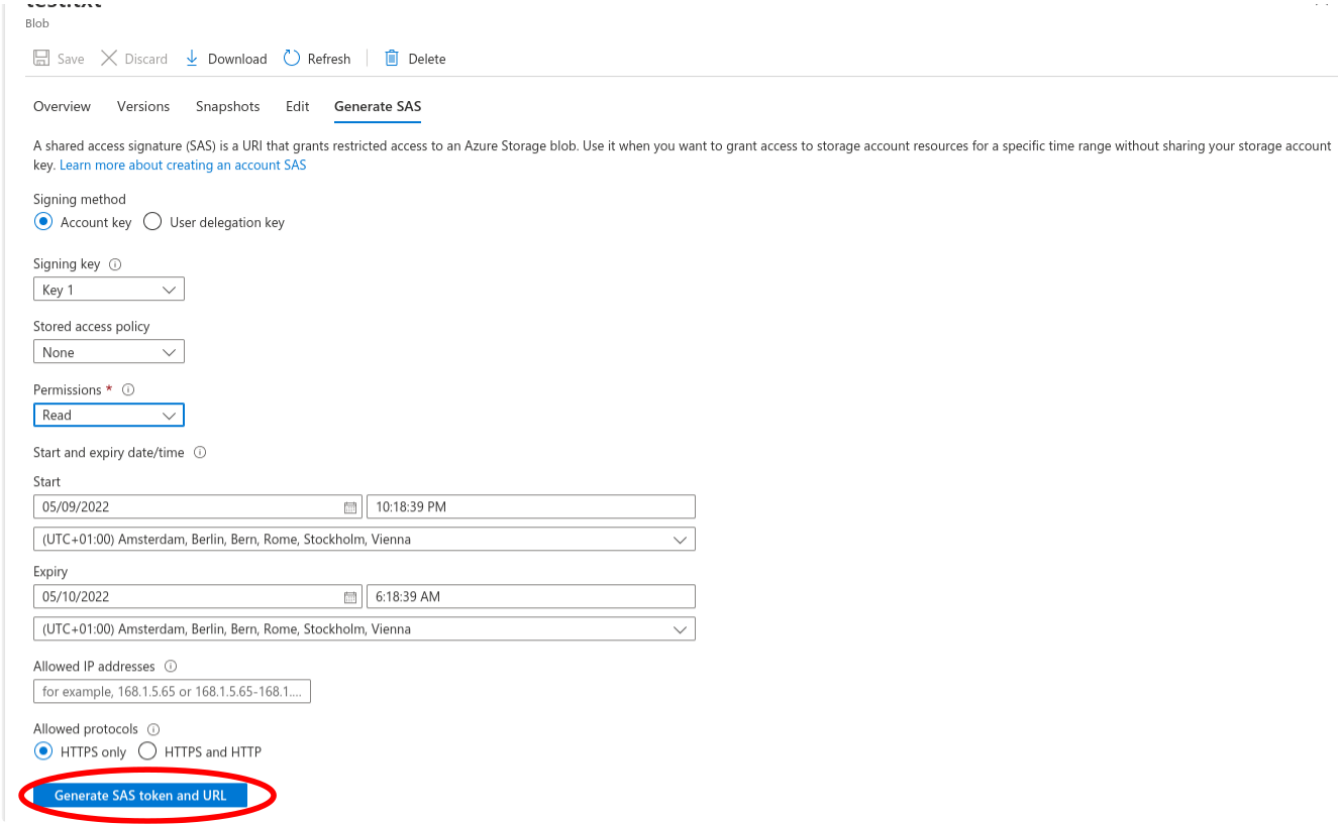
Step 4 - Share the file

Select the file to be published and click Generate SAS



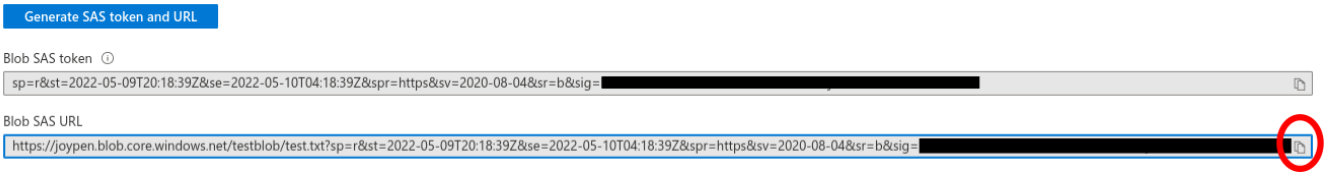
Click generate SAS

Configure the SAS details and click `Generate SAS token and URL`



Generate link to file

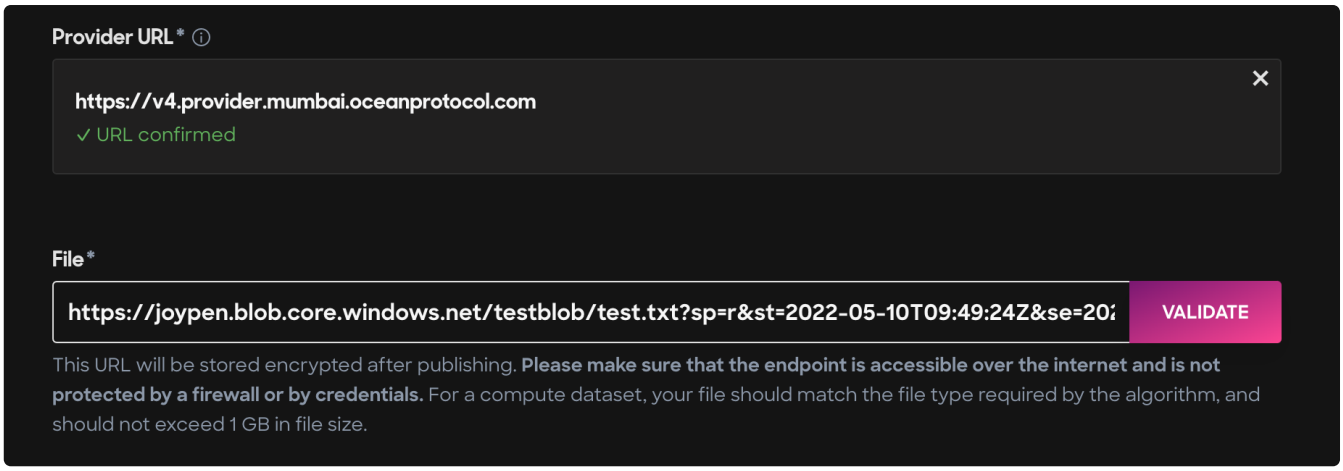
Copy the generated link



Copy the link

Step 5 - Publish the asset using the generated link

Now, copy and paste the link into the Publish page in the Ocean Marketplace.



Publish the file as an asset

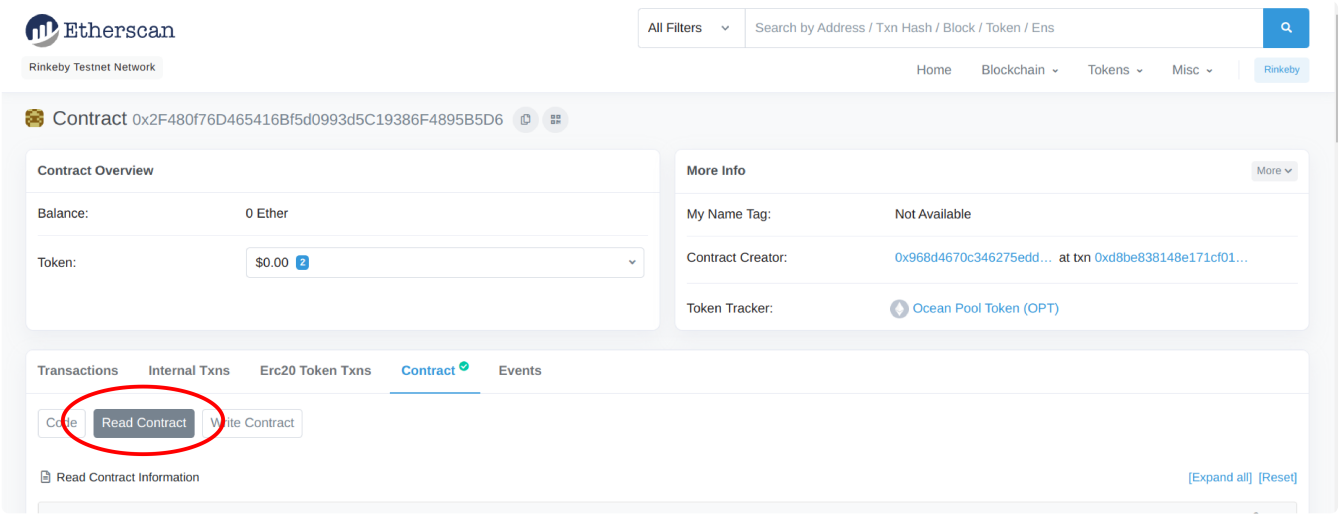
Liquidity Pools [deprecated]

In previous versions of Ocean liquidity pools and dynamic pricing were supported. These features have been deprecated and we now advise everyone to remove their liquidity from the remaining pools. It is no longer possible to do this via Ocean Market, so please follow this guide to remove your liquidity via etherscan.

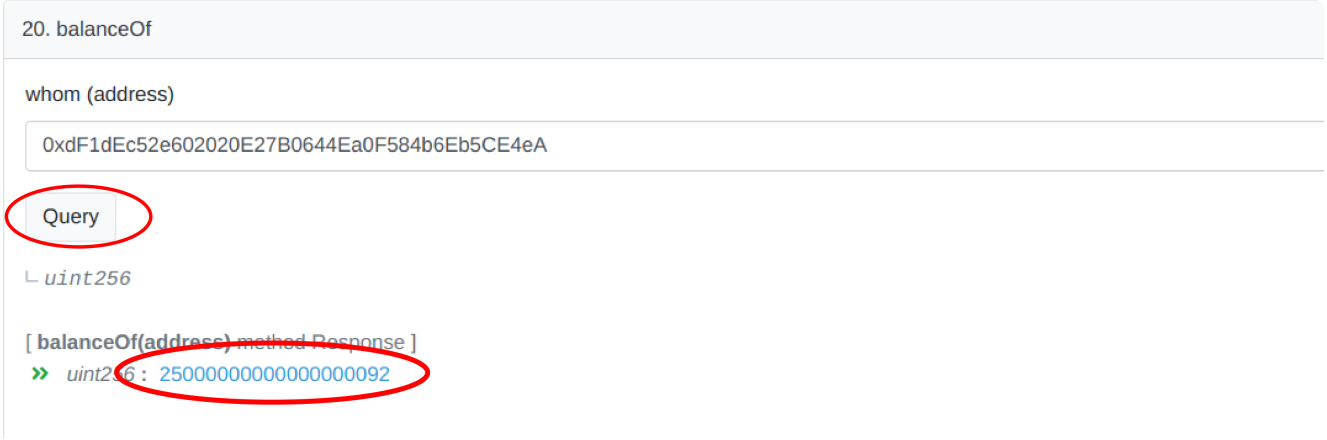
Remove liquidity using Etherscan

Get your balance of pool share tokens

1. Go to the pool's Etherscan/Polygonscan page. You can find it by inspecting your transactions on your account's Etherscan page under *Erc20 Token Txns*.
2. Click *View All* and look for Ocean Pool Token (OPT) transfers. Those transactions always come from the pool contract, which you can click on.
3. On the pool contract page, go to *Contract -> Read Contract*.

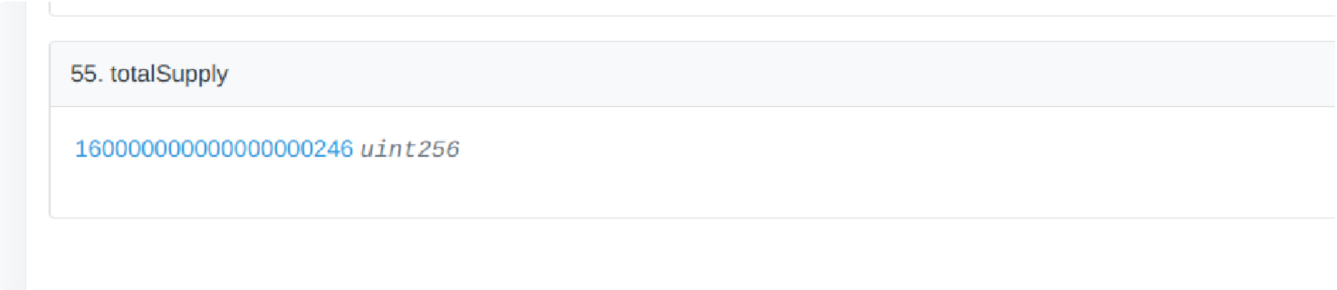


4. Go to field `20. balanceOf` and insert your ETH address. This will retrieve your pool share token balance in wei.



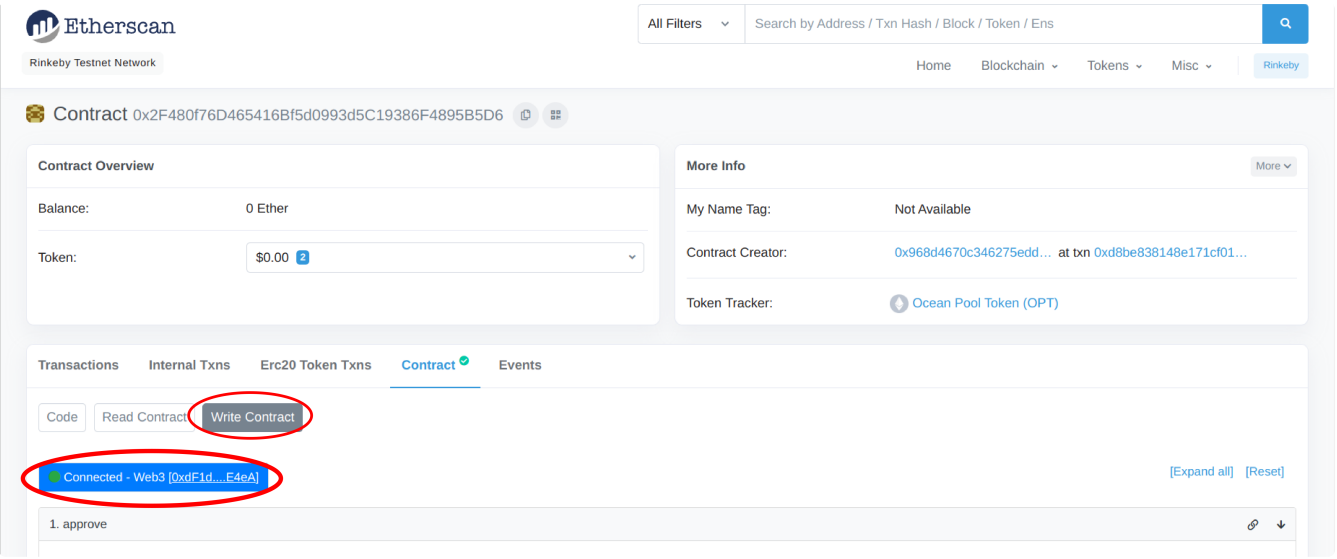
5. Copy this number as later you will use it as the `poolAmountIn` parameter.

6. Go to field `55. totalSupply` to get the total amount of pool shares, in wei.



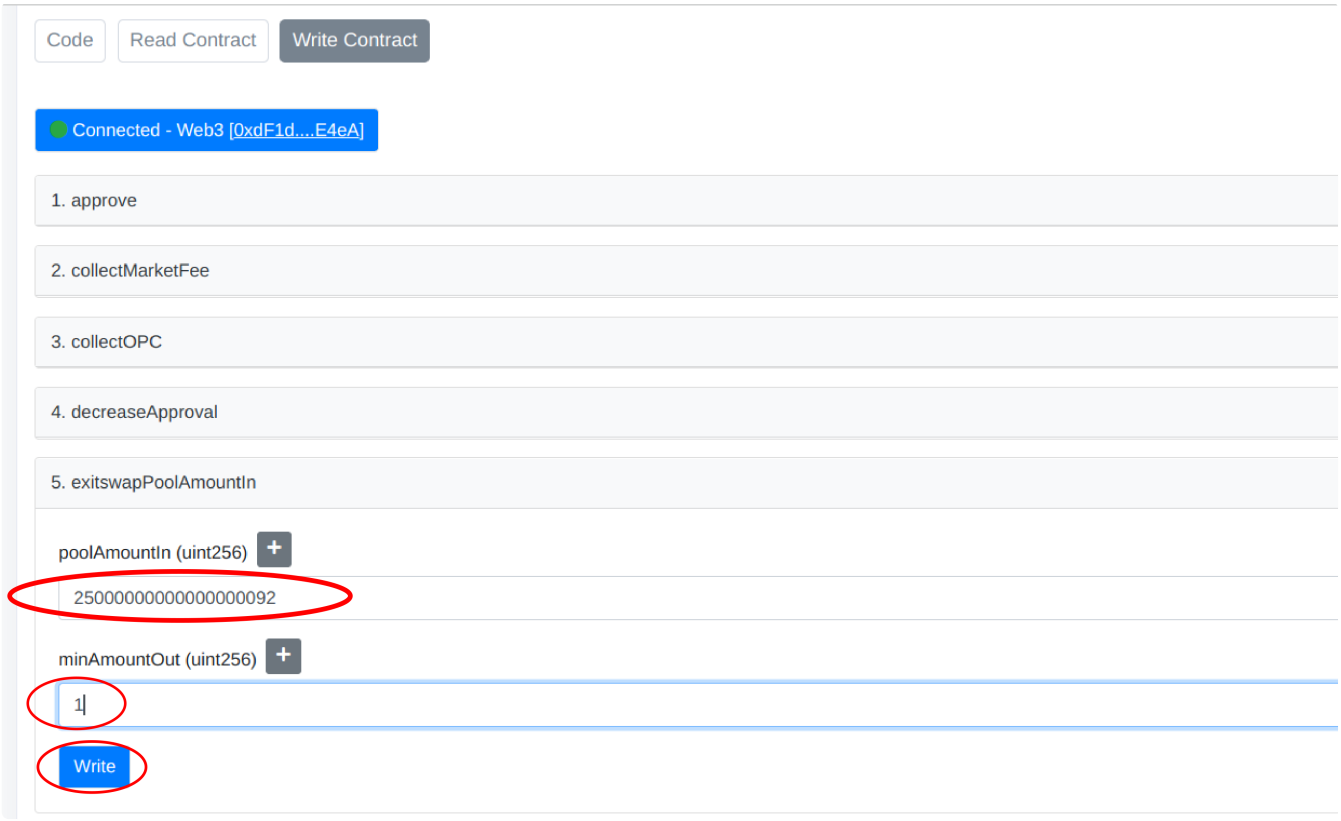
7. Divide the number by 2 to get the maximum of pool shares you can send in one pool exit transaction. If your number retrieved in former step is bigger, you have to send multiple transactions.

8. Go to *Contract -> Write Contract* and connect your wallet. Be sure to have your wallet connected to network of the pool.

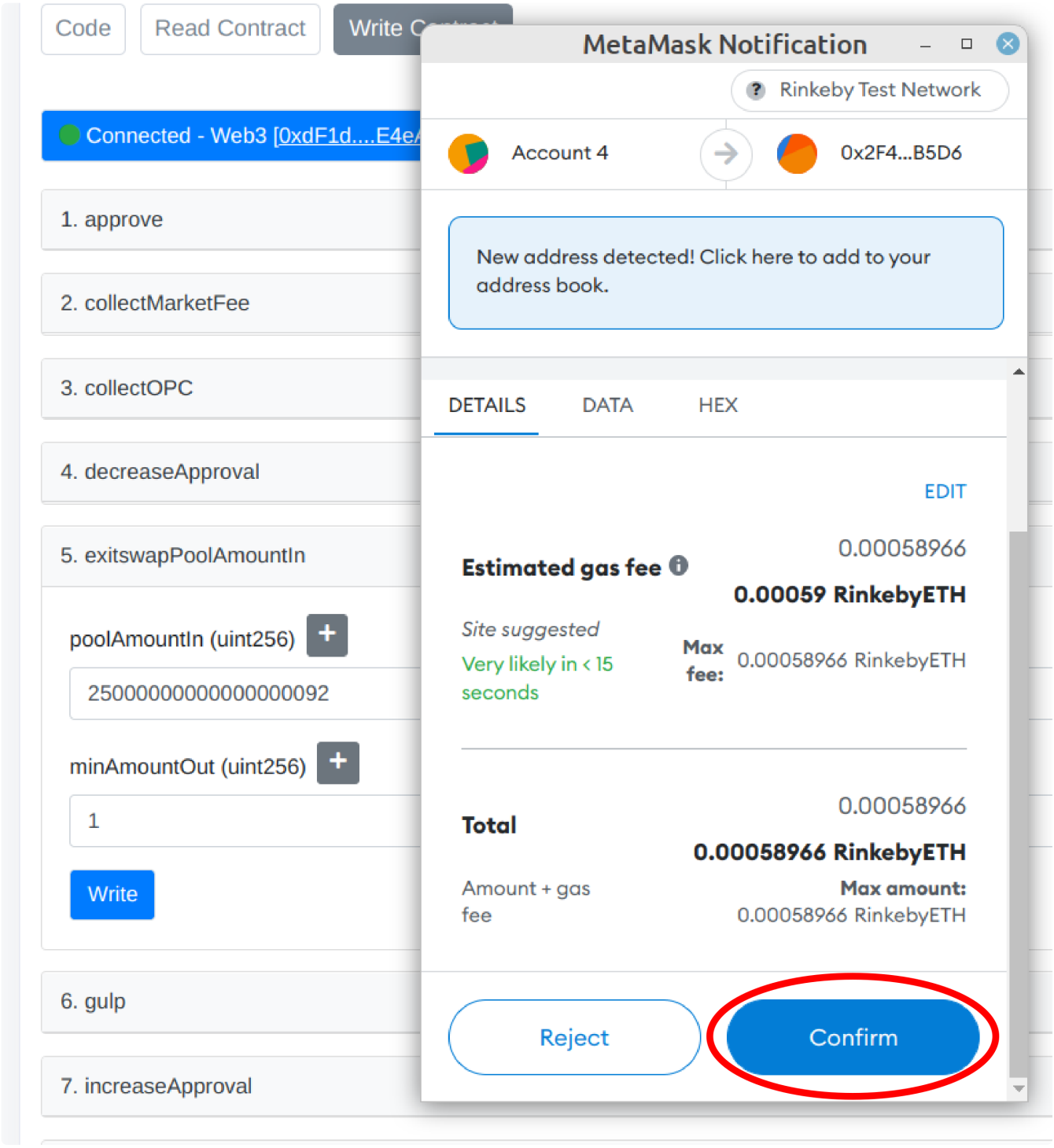


9. Go to the field `5. exitswapPoolAmountIn`

- For `poolAmountIn` add your pool shares in wei
- For `minAmountOut` use anything, like `1`
- Hit *Write*



10. Confirm transaction in Metamask


Code
Read Contract
Write Contract
Connected - Web3 [0xdF1d....E4e
1. approve
2. collectMarketFee
3. collectOPC
4. decreaseApproval
5. exitswapPoolAmountIn
poolAmountIn (uint256)
25000000000000000092
minAmountOut (uint256)
1
Write
6. gulp
7. increaseApproval
MetaMask Notification
Rinkeby Test Network
Account 4
0x2F4...B5D6
New address detected! Click here to add to your address book.
DETAILS
DATA
HEX
EDIT
Estimated gas fee
0.00058966
0.00059 RinkebyETH
Site suggested
Very likely in < 15 seconds
Max fee:
0.00058966 RinkebyETH
Total
0.00058966
0.00058966 RinkebyETH
Amount + gas fee
Max amount:
0.00058966 RinkebyETH
Reject
Confirm

Building with Ocean

Build a Marketplace

Forking and customizing Ocean Market (Frontend)

Outcome

Your own fully functioning customized fork of Ocean Market is deployed and working.

Introduction

Have you ever thought about monetizing digital assets over the blockchain? Your first instinct might be to head on over to a popular marketplace - but why not create your own marketplace? It's a lot easier than you might imagine. This guide will cover everything you need to start your own blockchain marketplace in less than an hour. The reason it's so easy is that we'll be starting with a fork of Ocean Market, which provides us with some pretty cool tech under-the-hood (if you're interested in blockchain, read on).

Using Ocean Market is already a big improvement on the alternatives that are out there, but it gets even better. Ocean Market is completely open-source and freely available under the Apache 2 license. This means that you can fork Ocean Market and set up your own marketplace in just a few steps. Ocean Market is primarily focused on monetizing data, but it can actually handle the sale of any digital asset. This guide will walk you through the process of forking Ocean Market and starting your own marketplace for selling photos; you'll be surprised how easy it is. No prior blockchain knowledge is required!

Content

The tutorial covers:

- Forking and running Ocean Market locally
- Customising your fork of Ocean market
- Quick deployment of Ocean Market

Preparation

Prior knowledge

If you're completely unfamiliar with Ocean Market or web3 applications in general, you will benefit from reading these guides first:

- To use your clone of Ocean Market, you'll need a wallet. We recommend getting set up with metamask.
- You'll also need some Ocean tokens on a testnet to use your marketplace.
- When you have the testnet tokens, have a go at publishing a data asset on Ocean Market.
- Run through the process of consuming a data asset on Ocean Market.

Required Prerequisites

- Git. Instructions for installing Git can be found here.
- Node.js can be downloaded from here (we're using version 16 in this guide)
- A decent code editor, such as Visual Studio Code.
- You'll need a Github account to fork Ocean market via Github.

Forking Ocean Market

Forking and running Ocean Market locally.

One of the best use cases for Ocean is running your own marketplace and monetizing your digital assets. With Ocean you can sell your data directly to your customers with no third party in-between, no need to speak to data with data brokers. Ocean makes this all super easy for you with some pretty cool tech under the hood. Furthermore, an Ocean powered market isn't just limited to selling data, you can use it for selling any type of digital asset!

Using Ocean Market is already a big improvement on the alternatives that are out there, but it gets even better. Ocean Market is completely open-source and made freely available under the Apache 2 license. This means that you can fork Ocean Market and set up your own data marketplace in just a few steps. This guide will walk you through the process, you'll be surprised how easy it is. No prior blockchain knowledge is required!

- Fork Ocean Market
- Clone the market locally
- Install the dependencies
- Run your Market fork for the first time

Fork Ocean Market

The first step is to log into Github and navigate to the Ocean Market repository, you'll need to log in or create a Github account. Now you need to click "Fork" in the top right-hand corner. If you are a member of an organization on Github, it will give you the option to clone it into either your personal account or the organization. Choose whichever is suitable for you.

Clone the market locally

Now we need to clone the market fork locally so that we can start making changes to the code. Upon forking Ocean Market, GitHub will take you to the repository page. Here, you should copy the URL of the repository. To do this, click on the green "Code" button and then click the copy icon to copy the HTTPS URL. Make sure that you have git installed and set up and installed on your computer before proceeding. See this guide if you're not familiar with git.

Install the dependencies

Installing the dependencies is a vital step for running the market. It's a super simple process, thanks to npm (node package manager). Make sure you have node.js installed, otherwise it will fail. In Ocean Market, we use node.js version 16 and it's highly recommended that you use the same.

Enter the following command to install the dependencies:

```
npm install
```

This command will take a few minutes to complete and you'll see some warnings as it runs (no need to worry about the warnings).

Run your Market fork for the first time

At this point, you are ready to run your data marketplace for the first time. This is another straightforward step that requires just one command:

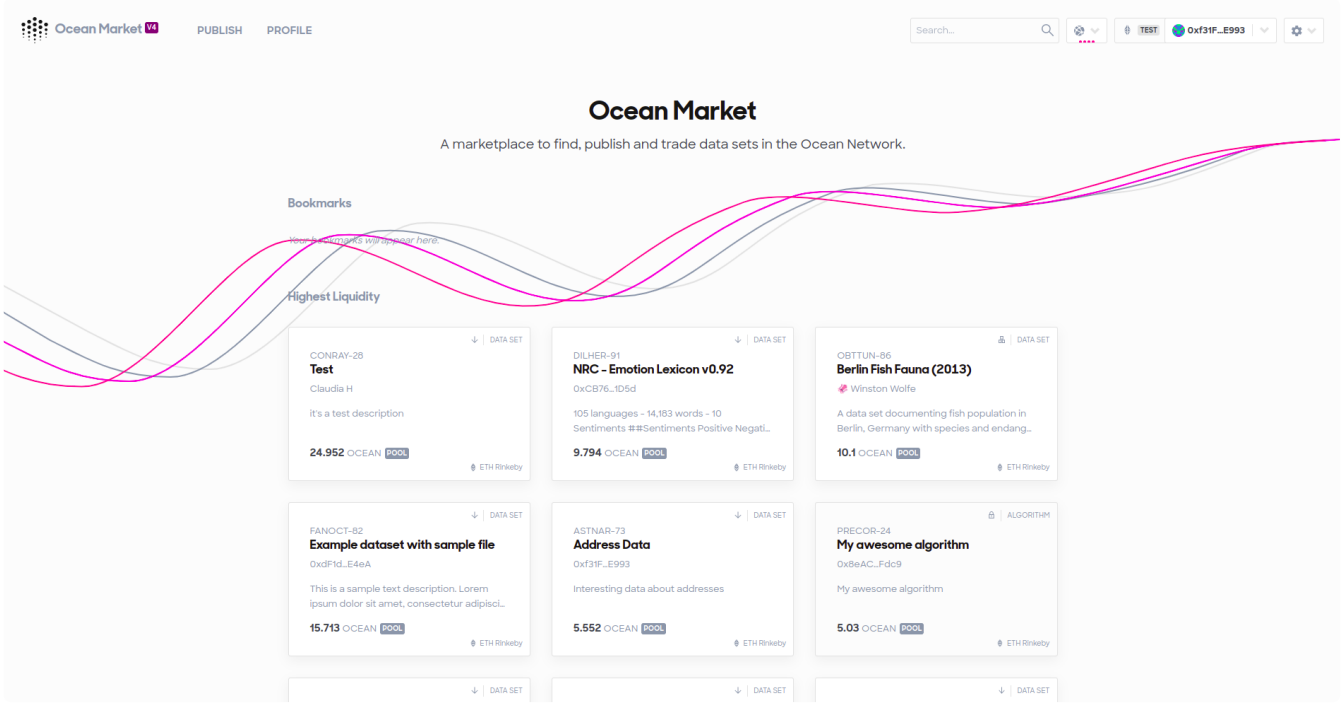
```
npm start
```

The above command will build the development bundle and run it locally.

Forking Ocean Market

Great news - your marketplace has successfully been built and is now running locally. Let's check it out! Open your browser and navigate to http://localhost:8000/. You'll see that you have a full-on clone of Ocean Market running locally. Give it a go and test out publishing and consuming assets - everything works!

That's all that's required to get a clone of Ocean market working. The whole process is made simple because your clone can happily use all the smart contracts and backend components that are maintained by Ocean Protocol Foundation.



Forking Ocean Market

So you've got a fully functioning marketplace at this point, which is pretty cool. But it doesn't really look like your marketplace. Right now, it's still just a clone of Ocean Market - the same branding, name, logo, etc. The next few steps focus on personalizing your marketplace.

Customising a Market

Step by step guide to customizing your fork of Ocean market

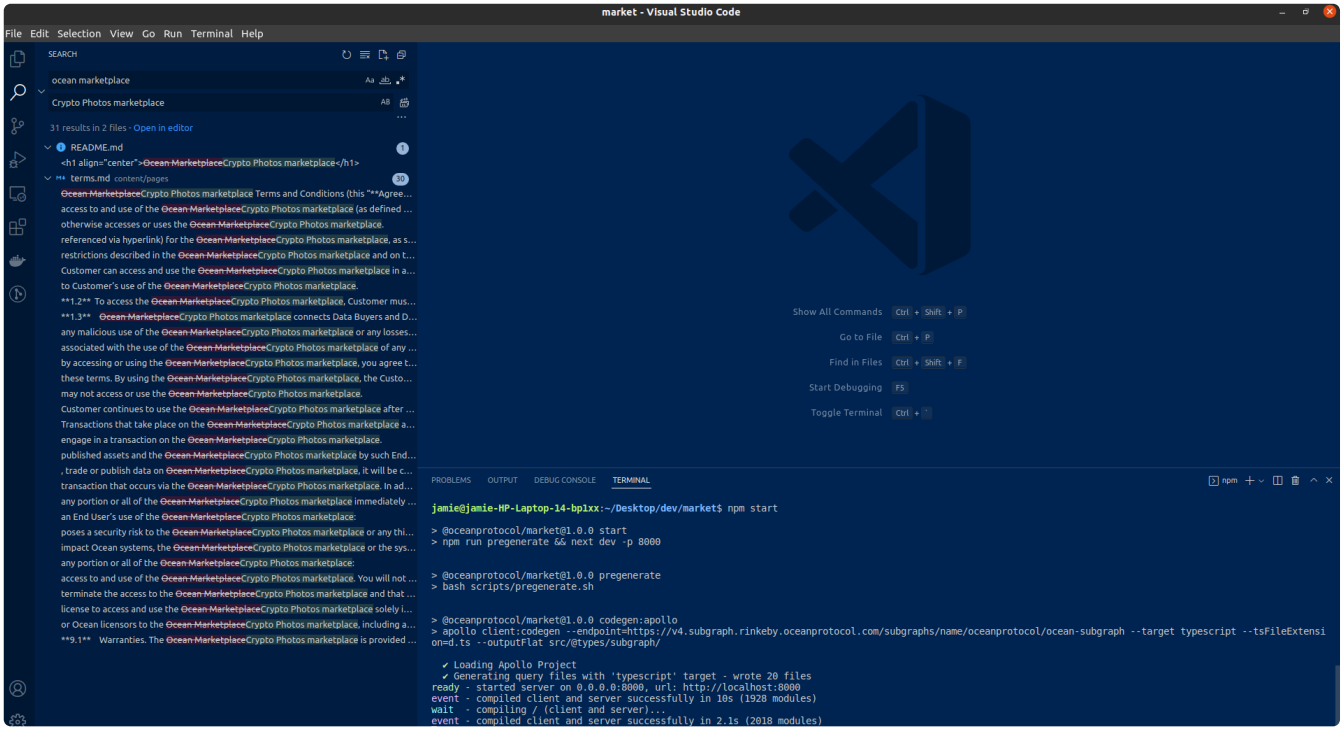
So you've got a fully functioning data marketplace at this point, which is pretty cool. But it doesn't really look like your data marketplace. Right now, it's still just a clone of Ocean Market — the same branding, name, logo, etc. The next few steps focus on personalizing your data marketplace.

- Change your Market Name
- Change the Logo
- Change the Styling
- Edit the Publish Form
- Change the Fee Address
- Build and host your Data Marketplace

Change your Market Name

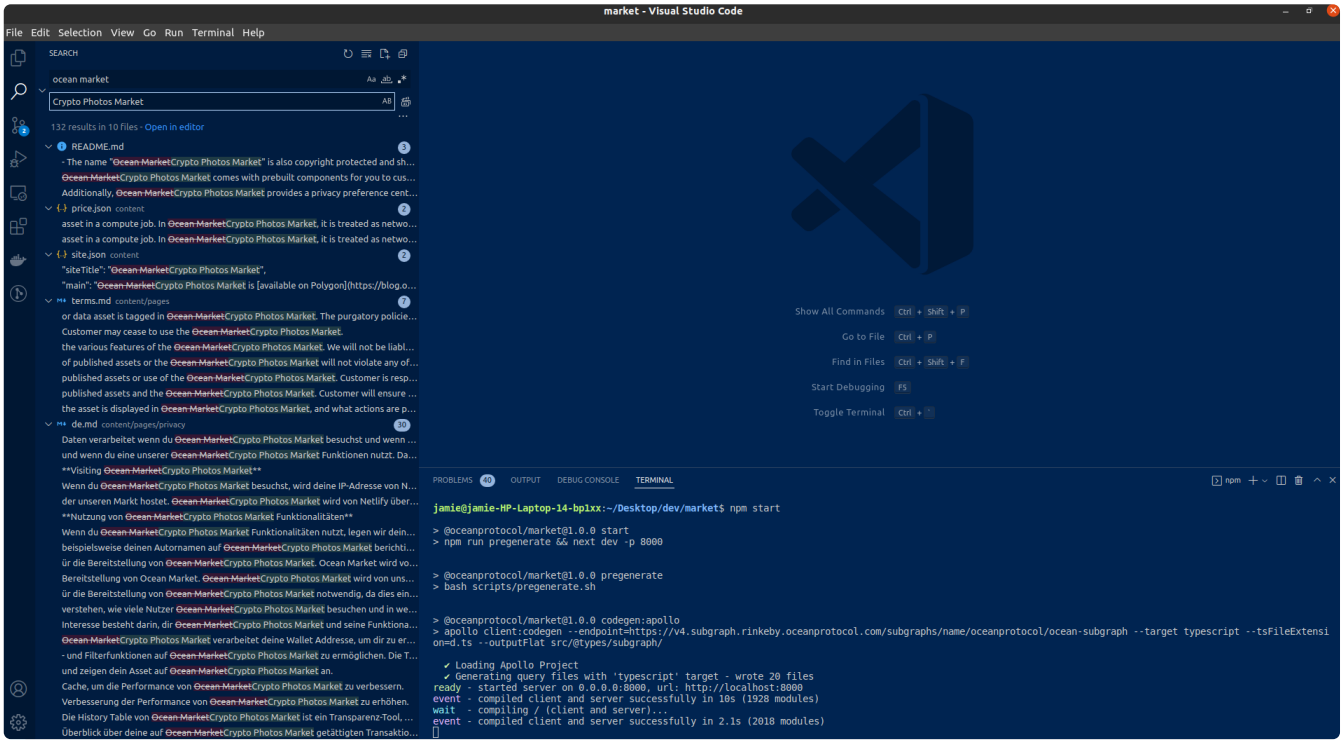
It's now time to open up your favorite code editor and start getting stuck into the code. The first thing we will be doing is changing the name of your marketplace. A decent code editor (such as VS Code) makes this incredibly simple by searching and replacing all the places where the name appears.

Let's start by searching and replacing "Ocean Marketplace". In VS Code there is a magnifying glass symbol in the left-hand panel (arrow 1 in the image below) that will open up the interface for searching and replacing text. Type "Ocean Marketplace" into the first textbox, and the name of your marketplace into the second textbox (arrow 2). To make things simple, there is a button to the right of the second textbox (arrow 3) that will replace all instances at once. You can take a moment to review all the text you're changing if you wish, and then click this button.

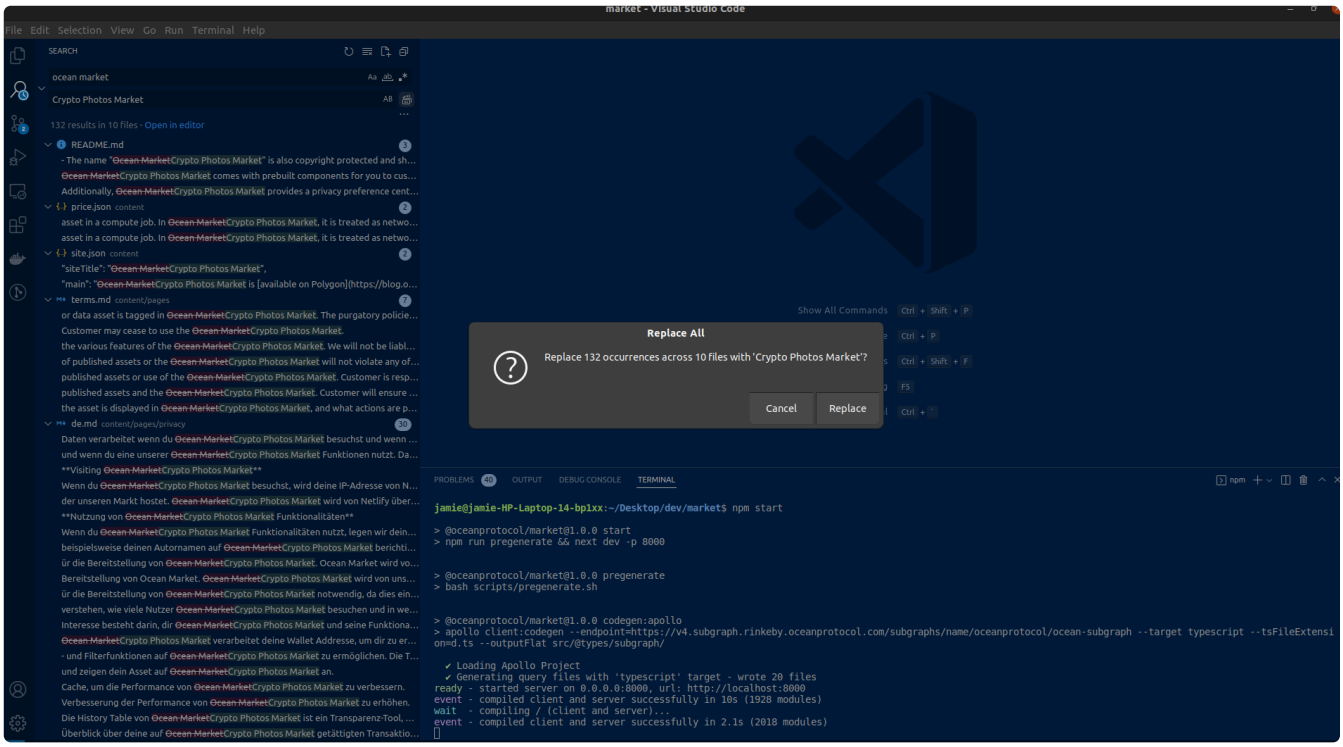


Market Customisation

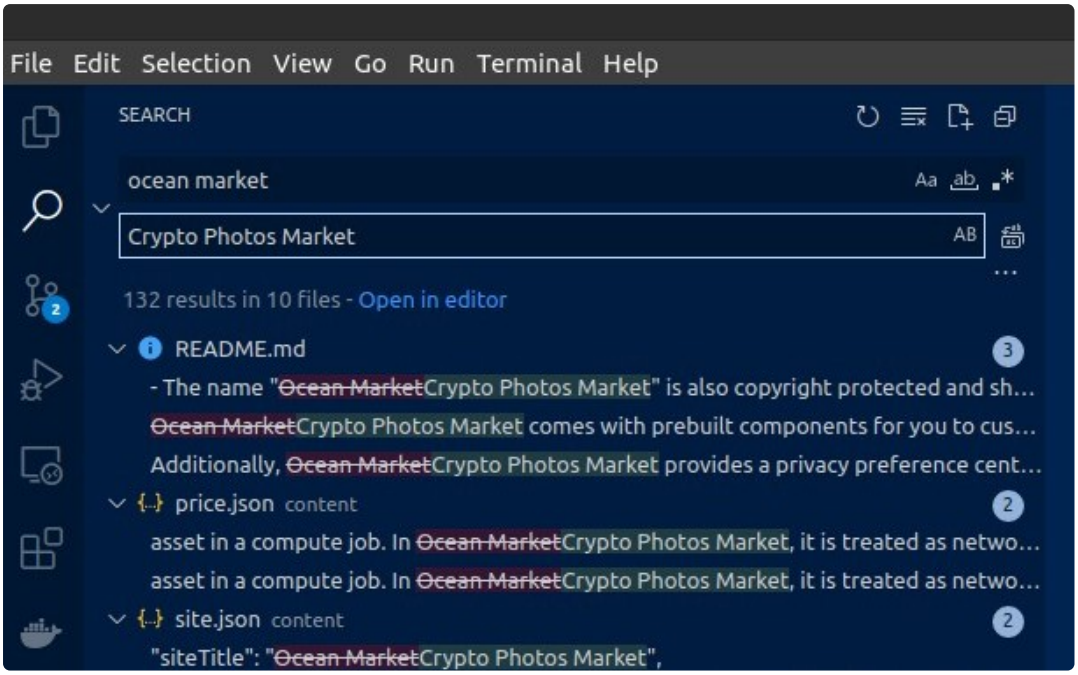
Next up, we need to repeat the process but this time we'll be searching and replacing "Ocean Market". As you can see in the screenshot below, we have called our fork "Crypto Photos Market".



Market Customisation

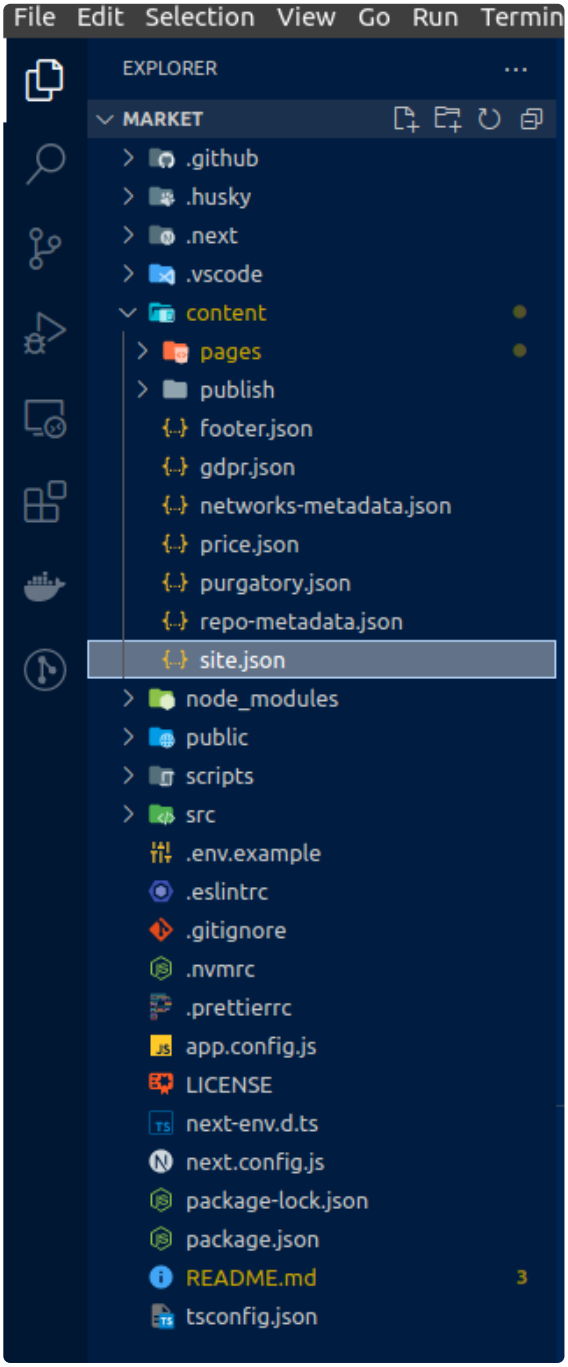


Market Customisation



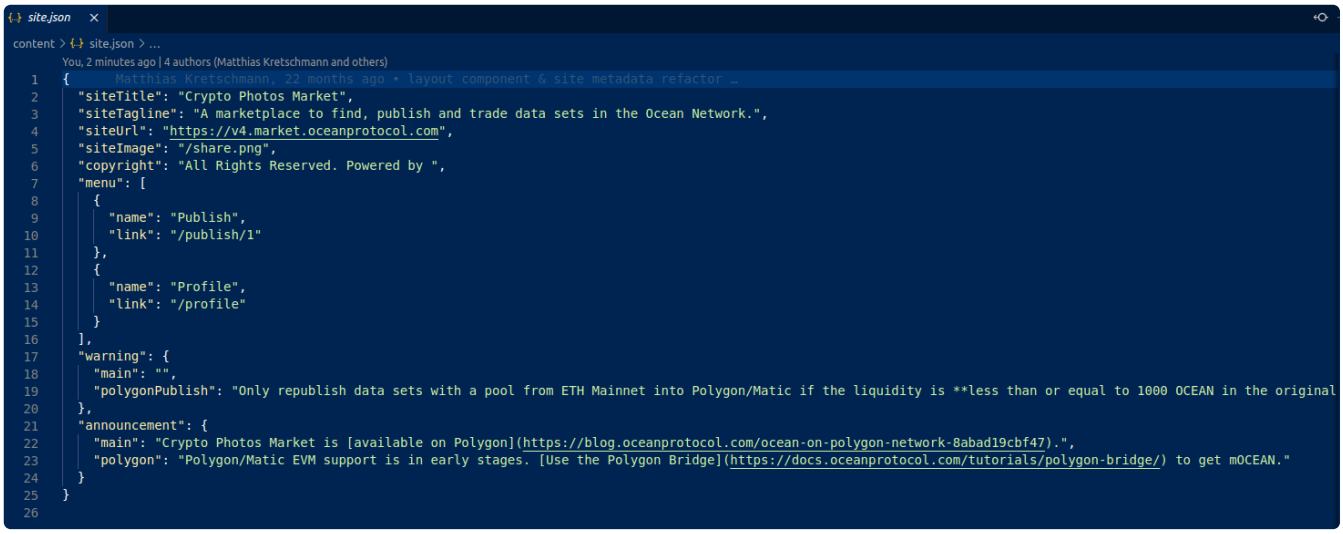
Market Customisation

Now let’s change the tagline of your site. Open up the folder called “content” and then open the file called “site.json”.



Market Customisation

On line 3 in this file you can enter the tagline that you want for your marketplace.



Market Customisation

Market Customisation

Change the Logo

The next important step to personalizing your marketplace is setting your own logo. We highly recommend using your logo in SVG format for this. The site logo is stored in the following location:

```
src/@images/logo.svg
```

Delete the “logo.svg” file from that folder and paste your own logo in the same folder. Then, if you rename your logo “logo.svg” everything will work without any problems.

At this point, it’s a good idea to check how things are looking. First check that you have saved all of your changes, then cancel the build that’s running in your terminal (Ctrl + C OR Cmnd + C) and start it again `npm start`. Once the build has finished, navigate to http://localhost:8000/ and see how things look.

Market Customisation

Awesome! Our logo is looking great!

Change the Styling

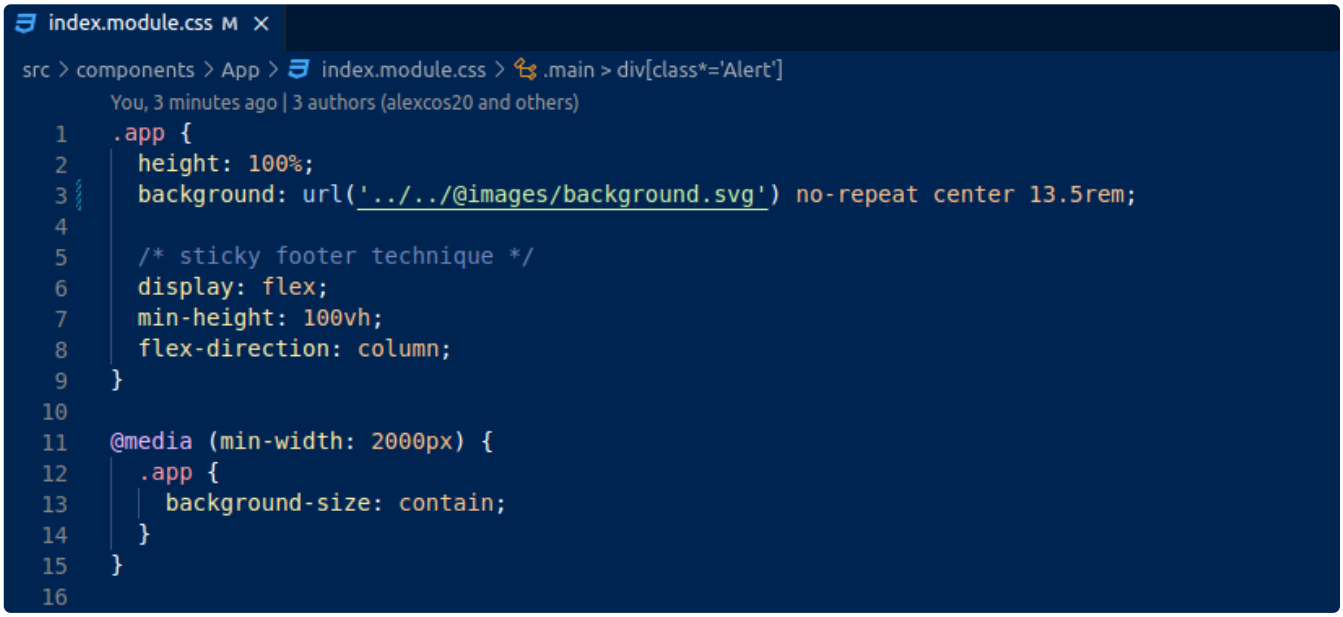
Hopefully, you like our pink and purple branding, but we don't expect you to keep it in your own marketplace. This step focuses on applying your own brand colors and styles.

Background

Let's start with the background. Open up the following CSS file:

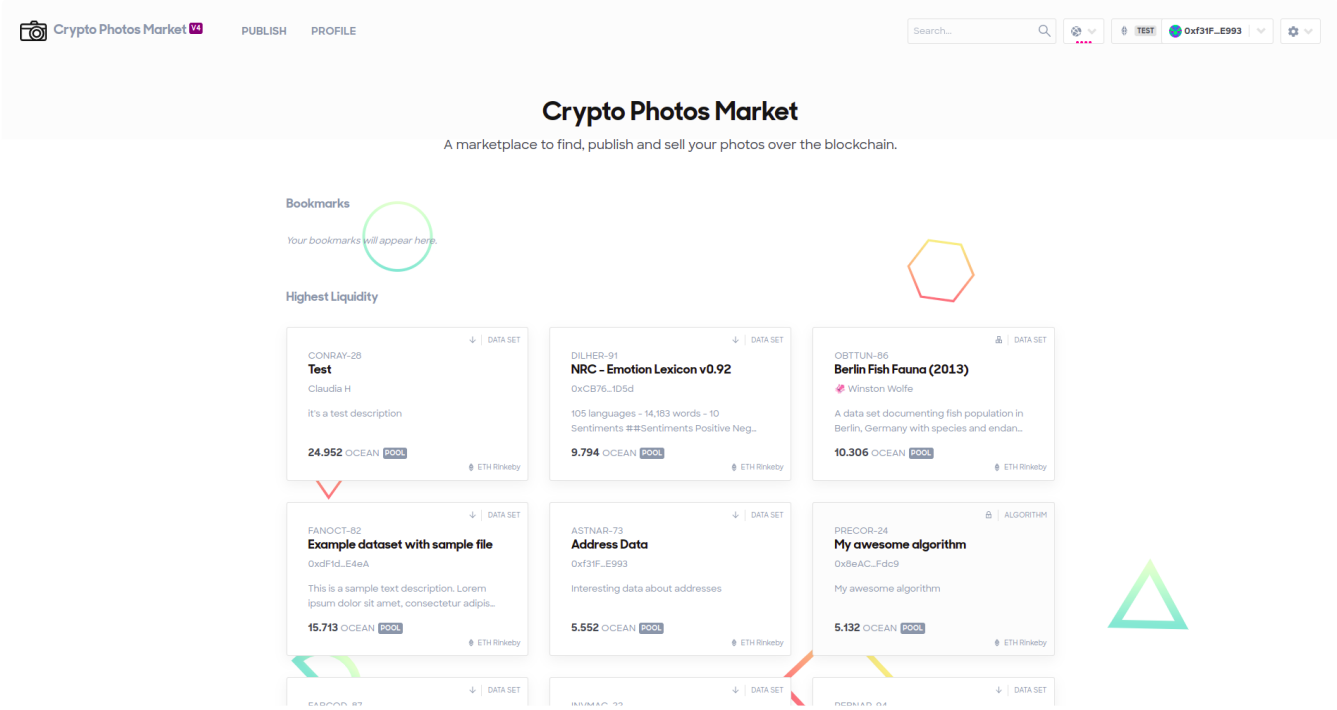
```
src/components/App/index.module.css
```

You'll notice in the screenshot above that we are setting our "wave" background on line 3. Here, you'll want to use your own background color or image. For this example, we'll use an SVG background from here. First, we save the new background image into the src/images/ folder (same folder as the logo), then we change the CSS to the file location of the new background (see line 3 in the image below).

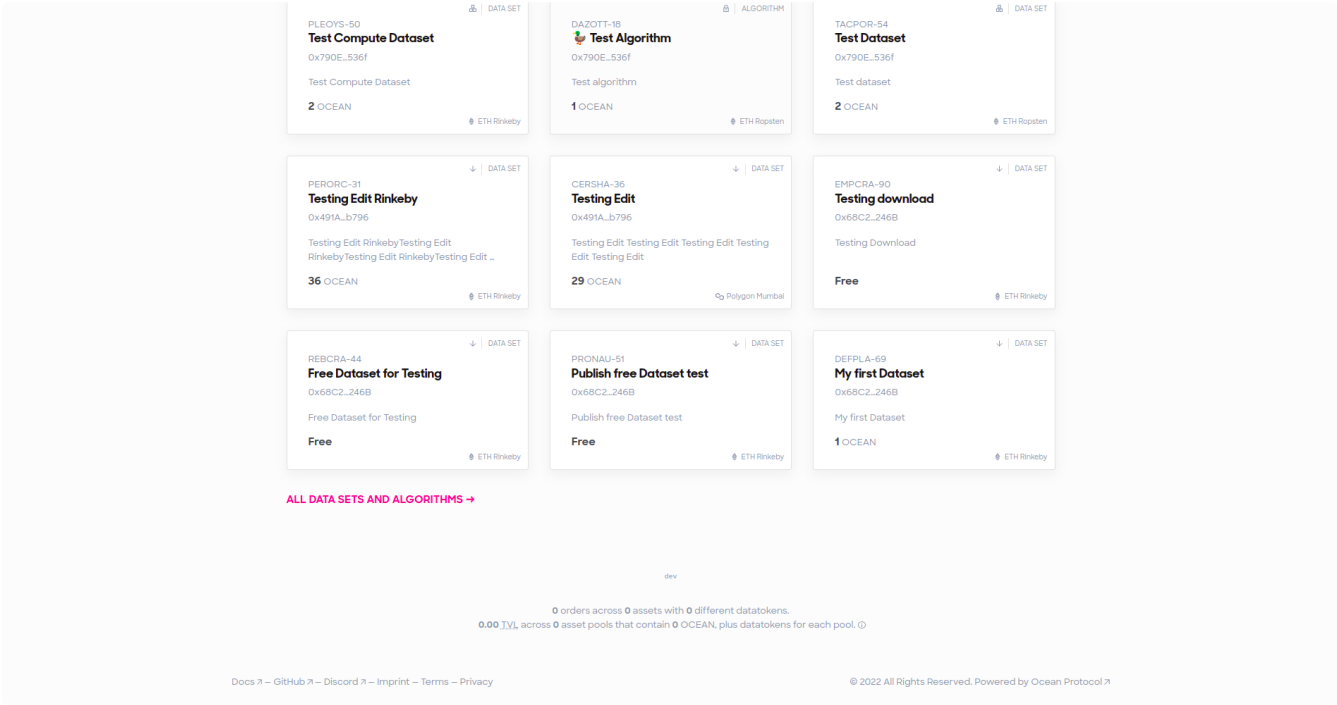


Market Customisation

If we save this file and view the site at this point, we get a white section at the top (see image below). And you'll also notice that the background doesn't fill all the way down to the bottom of the screen.



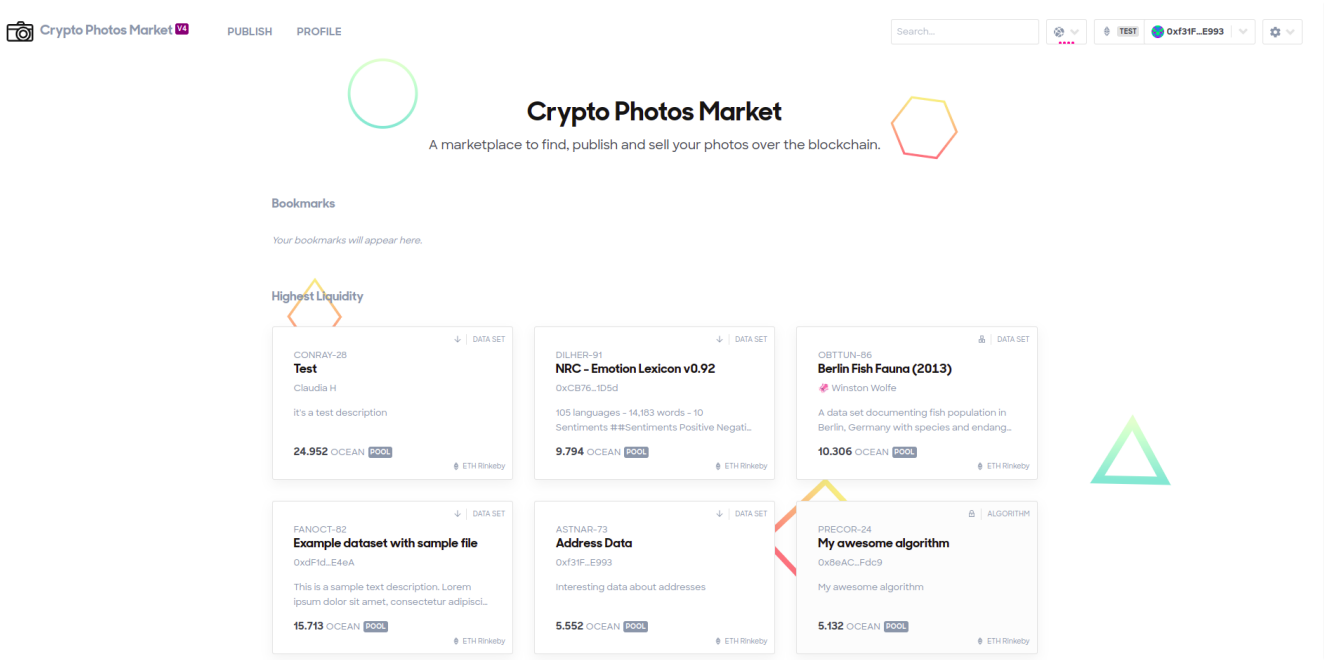
Market Customisation



Market Customisation

To fix this, we need to change the starting position of the background image and change it from no-repeat to repeat. We can do this on line 3.

When we view our marketplace, we can see that the new background starts at the top and fills the whole page. Perfect!

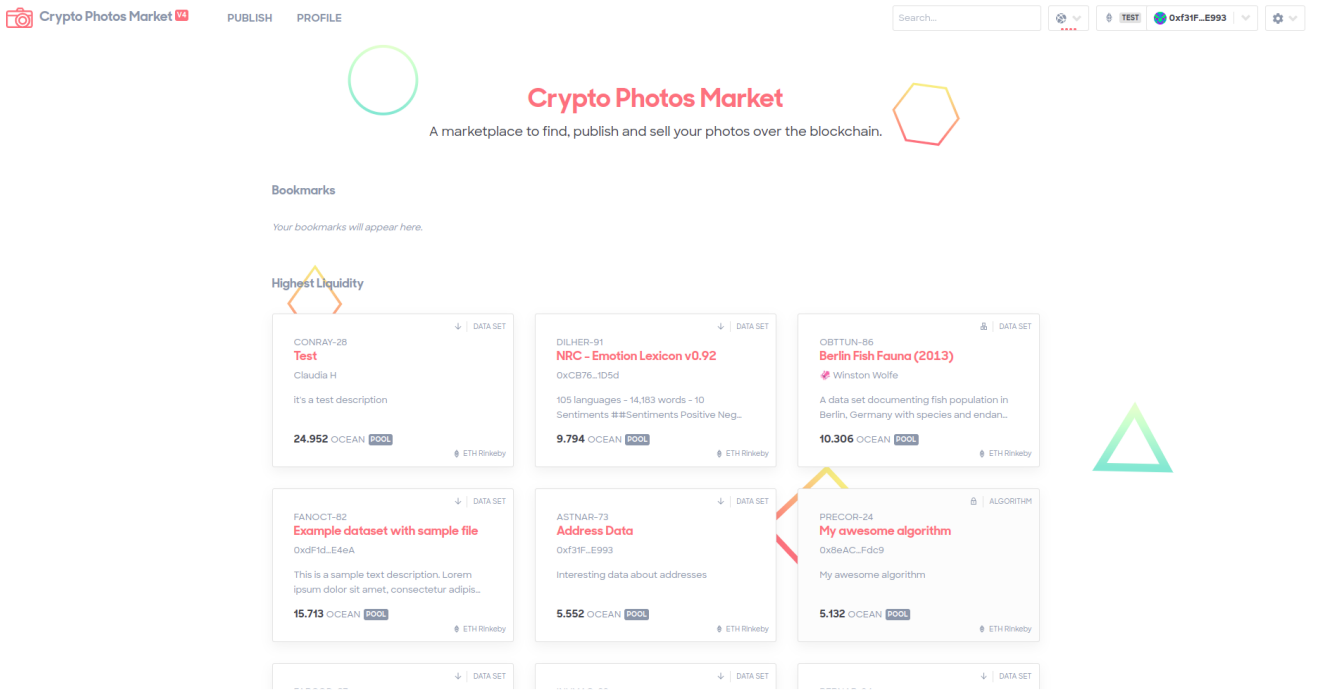


Market Customisation

Brand Colors

Next up, let's change the background colors to match your individual style. Open up the following file: src/global/_variables.css. Here you'll see the global style colors that are set. Now is the time to get creative, or consult your brand handbook (if you already have one).

You can change these colors as much as you wish until you're happy with how everything looks. Each time you save your changes, the site will immediately update so you can see how things look. You can see the styles chosen for this example in the image below.



Market Customisation

Change Fonts

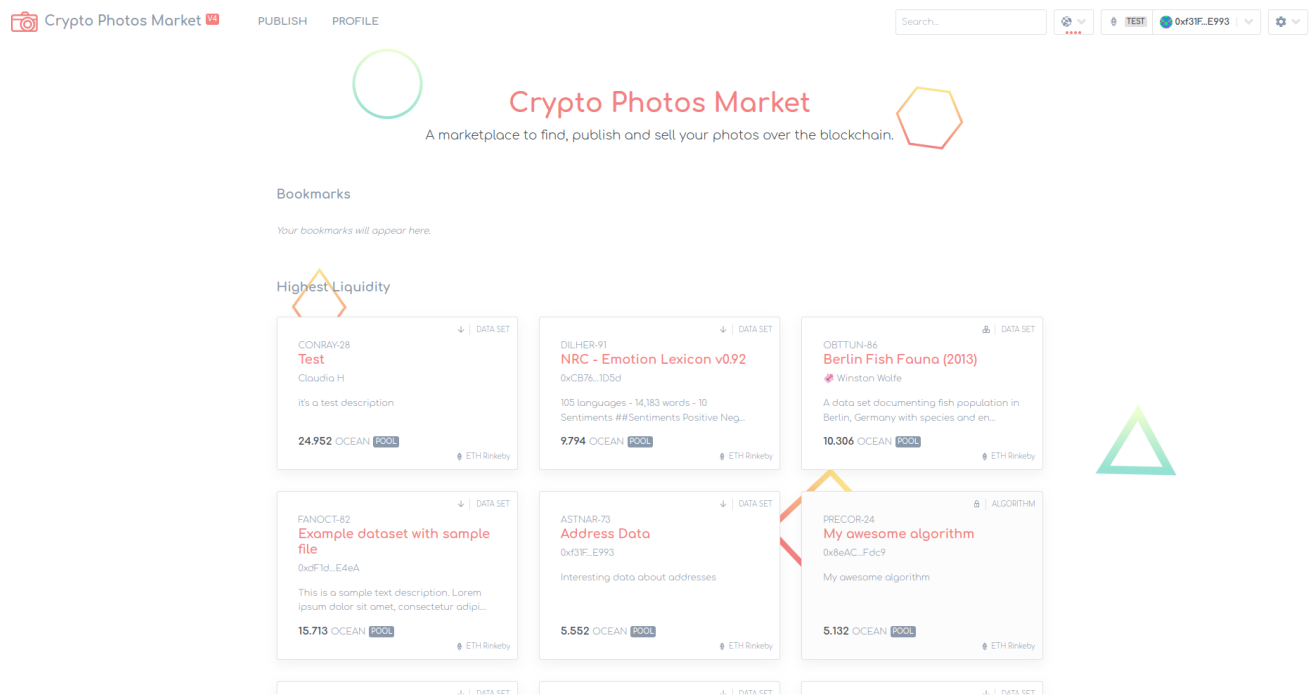
The final part of the styling that we'll alter in this guide is the fonts. This is an important step because the font used in Ocean Market is one of the few elements of the market that are copyright protected. If you want to use the same font you'll need to purchase a license. The other copyrighted elements are the logo and the name — which we have already changed.

If you don't already have a brand font, head over to Google Fonts to pick some fonts that suit the brand you're trying to create. Google makes it nice and easy to see how they'll look, and it's simple to import them into your project.

The global fonts are set in the same file as the colors, scroll down and you'll see them on lines 36 to 41.

If you are importing fonts, such as from Google fonts, you need to make sure that you include the import statement at the top of the _variables.css file.

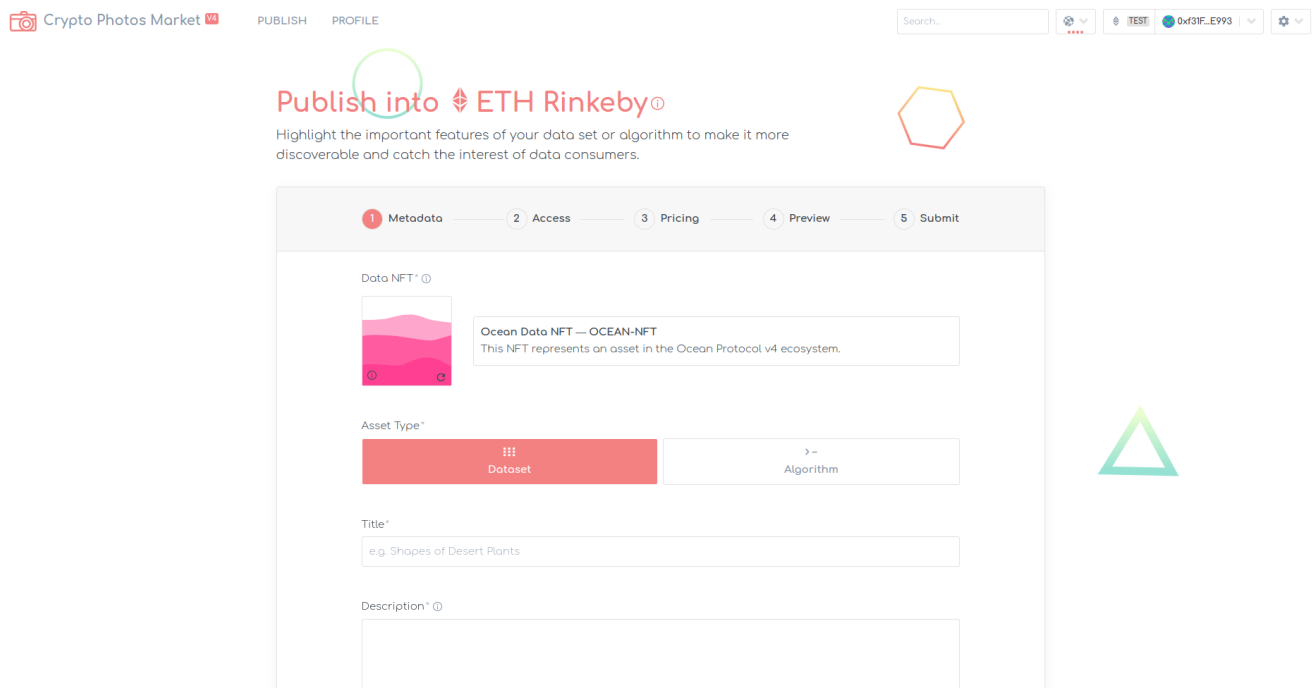
As with the color changes, it's a good idea to save the file with each change and check if the site is looking the way that you expected it to. You can see our eclectic choices below.



Market Customisation

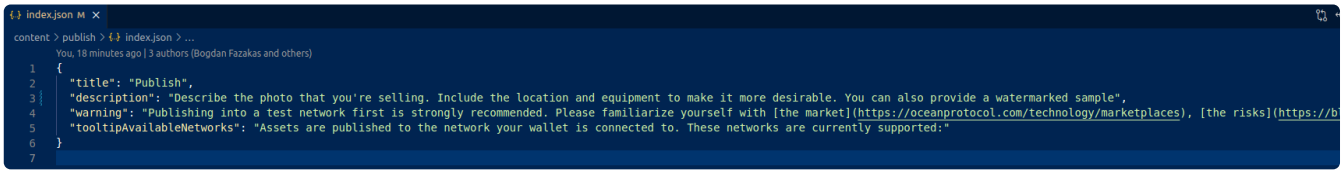
Customize the Publish Form

Let's head to the publish page to see what it looks like with our new styling - so far, so good. But there is one major issue, the publish form is still telling people to publish datasets. On our new marketplace, we want people to publish and sell their photos, so we're going to have to make some changes here.



Market Customisation

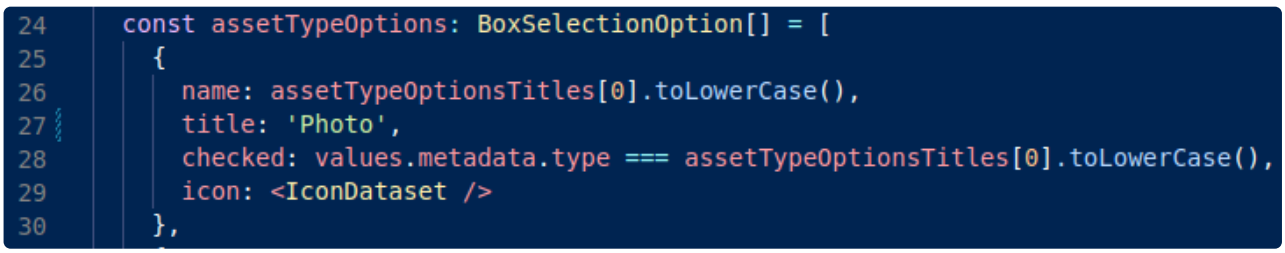
Open up the index.json file from content/publish/index.json - here we change the text to explain that this form is for publishing photos.



Market Customisation

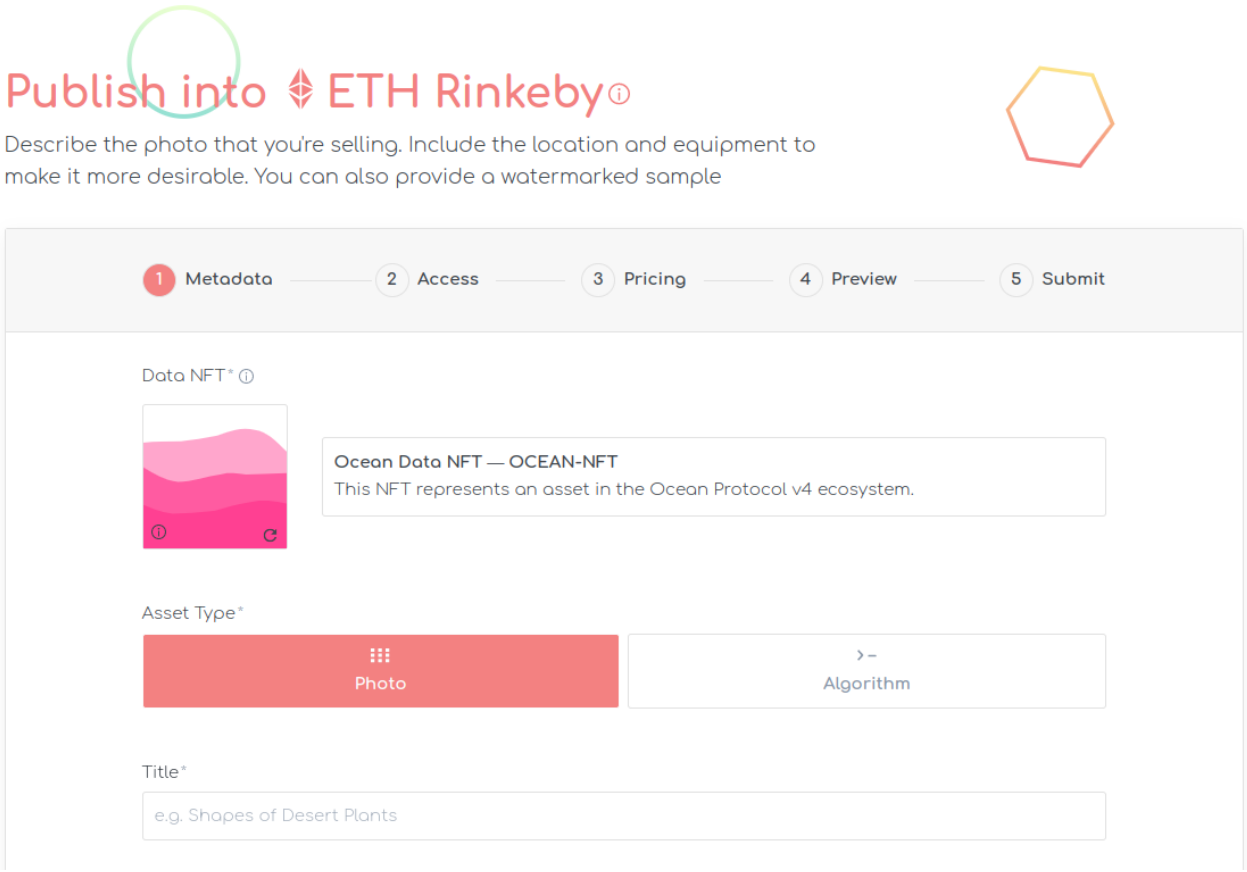
Additionally, the asset type current says dataset, and we need to change this so that it says photo. The simplest way to do this is to change the title of the asset type without changing anything else. Ocean can handle selling any digital asset that can be accessed via a URL, so no further changes are needed to accommodate selling photos.

Open up src/components/Publish/Metadata/index.tsx and change line 28 so that it says ‘Photo’



Market Customisation

Great, now our publish page explains that users should be publishing photos and photo is provided as an asset type option. We’ll also leave algorithm as an option in case some data scientists want to do some analysis or image transformation on the photos.



Market Customisation

There is one more thing that is fun to change before we move away from the publish form. You'll notice that Ocean Market V4 now has a cool SVG generation feature that creates the images for the Data NFT. It creates a series of pink waves. Let's change this so that it uses our brand colors in the waves!

Open up /src/@utils/SvgWaves.ts and have a look at lines 27 to 30 where the colors are specified. Currently, the pink color is the one used in the svg generator. You can replace this with your own brand color:

```
26
27  enum WaveColors {
28    Violet = '#F38181',
29    Pink = '#F38181',
30    Grey = '#95E1D3'
31  }
32
```

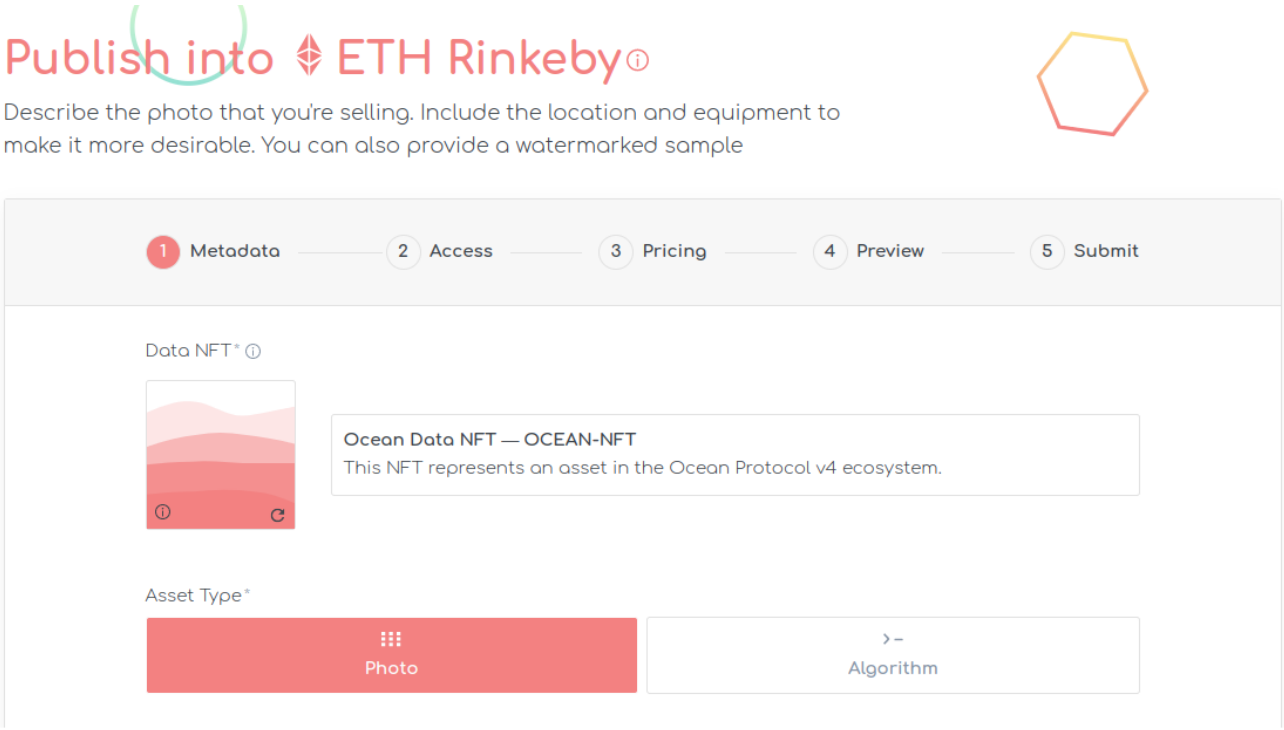
Market Customisation

If you're interested in doing some further customization, take a look at lines 53 to 64. You can change these properties to alter how the image looks. Feel free to play around with it. We've increased the number of layers from 4 to 5.

```
53    static getProps(): WaveProperties {
54      return {
55        width: 99,
56        height: 99,
57        viewBox: '0 0 99 99',
58        color: WaveColors.Pink,
59        fill: true,
60        layerCount: 5,
61        pointsPerLayer: randomIntFromInterval(3, 4),
62        variance: Math.random() * 0.2 + 0.5, // 0.5 - 0.7
63        maxOpacity: 255, // 0xff
64        opacitySteps: 68 // 0x44
65      }
66    }
67
```

Market Customisation

And now your customized publish page is ready for your customers:

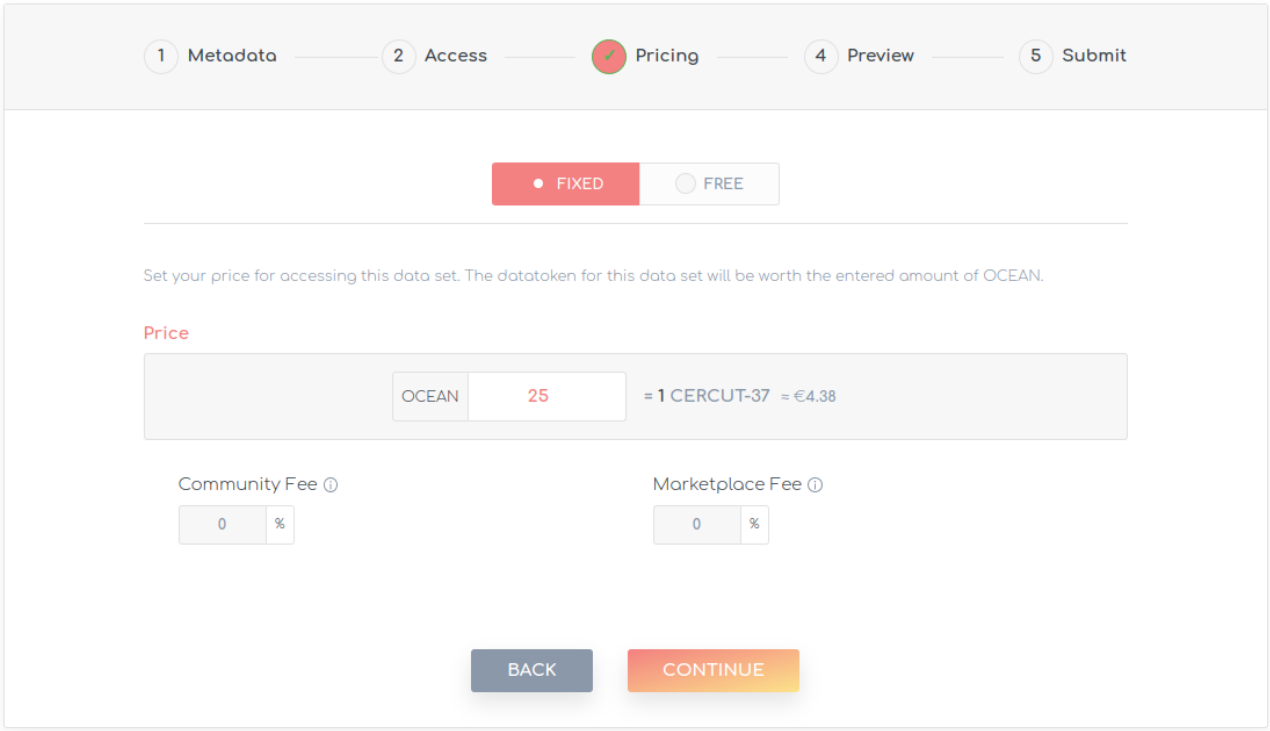


Market Customisation

Change the Fee Address

At this point, we have made a lot of changes and hopefully you're happy with the way that your marketplace is looking. Given that you now have your own awesome photo marketplace, it's about time we talked about monetizing it. Yup, that's right - you will earn a commission when people buy and sell photos in your marketplace. In Ocean V4, there are a whole host of new fees and customization options that you can use (read more about that here).

When someone sets the pricing for their photos in your marketplace, they are informed that a commission will be sent to the owner of the marketplace. You see that at the moment this fee is set to zero, so you'll want to increase that. And in order to receive the fees you'll you need to set the address that you want to recieve these fees in.



Market Customisation

This important step is the last thing that we will change in this guide. To set the marketplace fees and address, you'll need to save them as environmental variables. Create a new file called .env in the root of your repository.

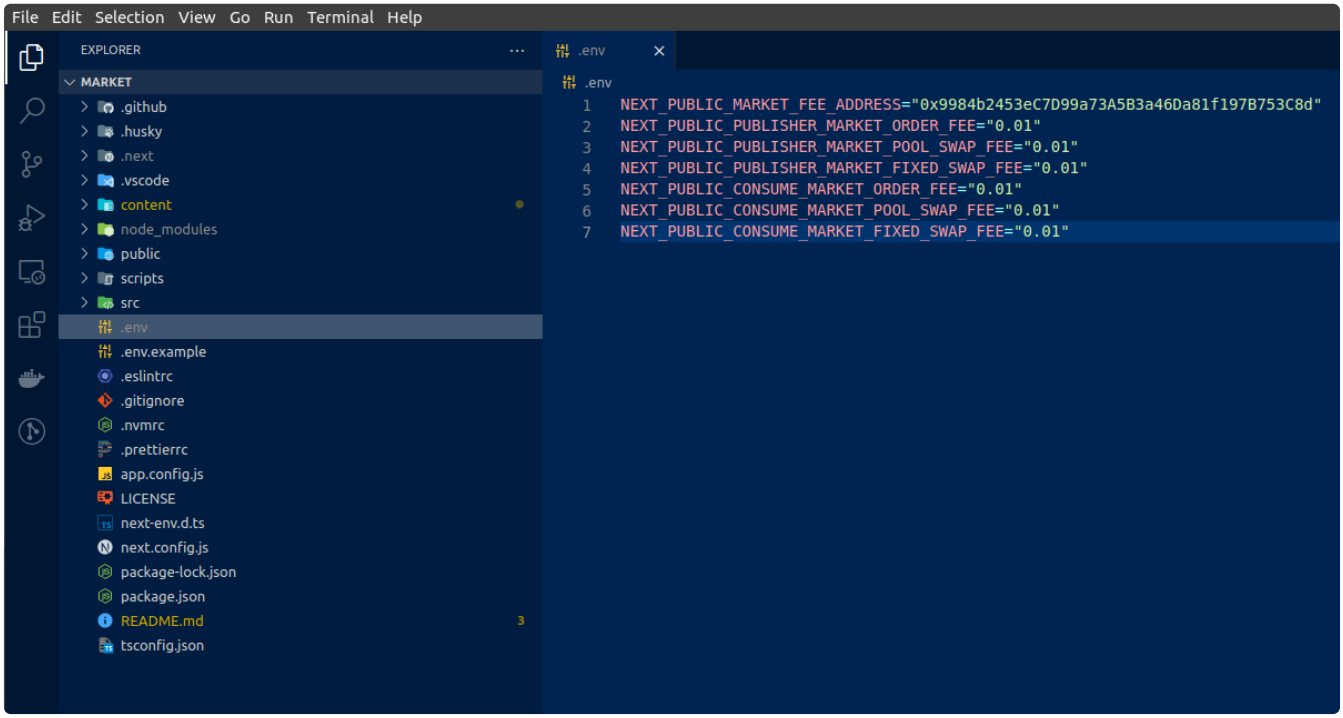
Copy and paste the following into the file:

```
NEXT_PUBLIC_MARKET_FEE_ADDRESS="0x123abc"
NEXT_PUBLIC_PUBLISHER_MARKET_ORDER_FEE="0.01"
NEXT_PUBLIC_PUBLISHER_MARKET_FIXED_SWAP_FEE="0.01"
NEXT_PUBLIC_CONSUME_MARKET_ORDER_FEE="0.01"
NEXT_PUBLIC_CONSUME_MARKET_FIXED_SWAP_FEE="0.01"
```

You need to replace "0x123abc" with your ethereum address (this is where the fees will be sent) and alter the fees to the levels that you intend them to be at. If you change you mind, these fees can always be altered later.

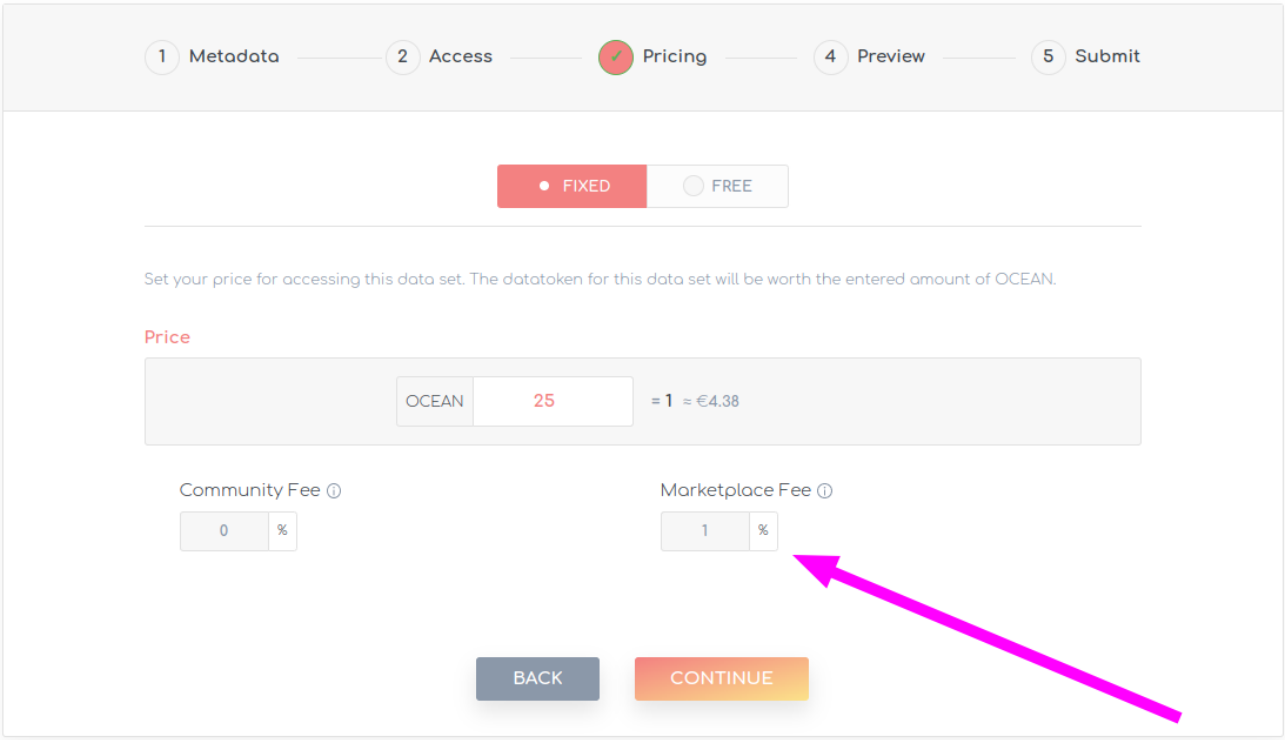
Go to Fees page to know more details about each type of fee and its relevance.

It is important that the file is saved in the right place at the root of your repository, your file structure should look the same as below.



Market Customisation

Now that's it; you now have a fully functioning photo marketplace that operates over the blockchain. Everytime someone uses it, you will receive revenue.



Market Customisation

Deploying a Market

Step by step guide to a quick deployment of Ocean Market

All that's left is for you to host your data marketplace and start sharing it with your future users.

Build and host your Data Marketplace

To host your data marketplace, you need to run the build command:

```
npm run build:static
```

This takes a few minutes to run. While this is running, you can get prepared to host your new data marketplace. You have many options for hosting your data marketplace (including AWS S3, Vercel, Netlify and many more). In this guide, we will demonstrate how to host it with surge, which is completely free and very easy to use.

Open up a new terminal window and run the following command to install surge:

```
npm install --global surge
```

When this is complete, navigate back to the terminal window that is building your finished data marketplace. Once the build is completed, enter the following commands to enter the public directory and host it:

```
cd out
```

```
surge
```

If this is your first time using surge, you will be prompted to enter an email address and password to create a free account. It will ask you to confirm the directory that it is about to publish, check that you are in the market/public/ directory and press enter to proceed. Now it gives you the option to choose the domain that you want your project to be available on. We have chosen https://crypto-photos.surge.sh which is a free option. You can also set a CNAME value in your DNS to make use of your own custom domain.

After a few minutes, your upload will be complete, and you're ready to share your data marketplace. You can view the version we created in this guide here.

Using Ocean Libraries

Ocean Protocol officially supports two client libraries:

- ocean.js
- ocean.py

ocean.js	ocean.py
Written in Javascript	Written in Python
Source code	Source code
Releases	Releases

The tutorials in this section will guide you how to setup the required configuration and interact with Ocean Protocol's smart contracts, Aquarius and Provider using the supported libraries.

Configuration

Obtaining API key for Ethereum node provider

Ocean Protocol's smart contracts are deployed on EVM-compatible networks. Using an API key provided by a third-party Ethereum node provider allows you to interact with the Ocean Protocol's smart contracts on the supported networks without requiring you to host a local node.

Choose any API provider of your choice. Some of the commonly used are:

- Infura
- Alchemy
- Moralis

The supported networks are listed here.

Create a directory

Let's start with creating a working directory where we store the environment variable file, configuration files and the scripts.

```
mkdir my-ocean-project
cd my-ocean-project
```

Create a `.env` file

In the working directory create a `.env` file. The content of this file will store the values for following variables:

Variable name	Description	Required
OCEAN_NETWORK	Name of the network where the Ocean Protocol's smart contracts are deployed.	Yes
OCEAN_NETWORK_URL	The URL of the Ethereum node (along with API key for non-local networks)	Yes
PRIVATE_KEY	The private key of the account which you want to use. A private key is made up of 64 hex characters. Make sure you have sufficient balance to pay for the transaction fees.	Yes
AQUARIUS_URL	The URL of the Aquarius. This value is needed when reading an asset from off-chain store.	No
PROVIDER_URL	The URL of the Provider. This value is needed when publishing a new asset or update an existing asset.	No

> ⓘ Treat this file as a secret and do not commit this file to git or share the content publicly. If you are using git, then include this file name in `.gitignore` file.

The below tabs show partially filled `.env` file content for some of the supported networks.

Mainnet

.env

```
# Mandatory environment variables

OCEAN_NETWORK=mainnet
OCEAN_NETWORK_URL=<replace this>
PRIVATE_KEY=<secret>

# Optional environment variables

AQUARIUS_URL=https://v4.aquarius.oceanprotocol.com/
PROVIDER_URL=https://v4.provider.mainnet.oceanprotocol.com
```

Polygon

.env

```
# Mandatory environment variables

OCEAN_NETWORK=polygon
OCEAN_NETWORK_URL=<replace this>
PRIVATE_KEY=<secret>

# Optional environment variables

AQUARIUS_URL=https://v4.aquarius.oceanprotocol.com/
PROVIDER_URL=https://v4.provider.polygon.oceanprotocol.com
```

Local (using Barge)

.env

```
# Mandatory environment variables
OCEAN_NETWORK=development
OCEAN_NETWORK_URL=http://172.15.0.3:8545/
AQUARIUS_URL=http://172.15.0.5:5000
PROVIDER_URL=http://172.15.0.4:8030

# Replace PRIVATE_KEY if needed
PRIVATE_KEY=0xc594c6e5def4bab63ac29eed19a134c130388f74f019bc74b8f4389df2837a58
```

*NOTE: If using ocean.py, additionally specify **ADDRESS_FILE** variable in the `.env` file. Copy the content of this link locally and set the **ADDRESS_FILE** so that its value is a correct file path.*

Setup dependencies

In this step the required dependencies will be installed.

ocean.js

```
npm init
npm install @oceanprotocol/lib@latest dotenv web3 @truffle/hdwallet-provider
```

ocean.py

```
python3 -m venv venv
source venv/bin/activate
pip install wheel

# Install Ocean library. Allow pre-releases to get the latest v4 version.
pip install ocean-lib
```

Create a configuration file

A configuration file will read the content of the `.env` file and initialize the required configuration objects which will be used in the further tutorials. The below scripts creates a Web3 wallet instance and a Ocean's configuration object.

Create the configuration file in the working directory i.e. at the same path where the `.env` is located.

ocean.js

config.js

```
// Import dependencies
require('dotenv').config();
const HDWalletProvider = require('@truffle/hdwallet-provider');
const fs = require('fs');

const { homedir } = require('os');
const { ConfigHelper } = require('@oceanprotocol/lib');

// Get configuration for the given network
let oceanConfig = new ConfigHelper().getConfig(process.env.OCEAN_NETWORK);

// If using local development environment, read the addresses from local file.
// The local deployment address file can be generated using barge.
if (process.env.OCEAN_NETWORK === 'development') {
  const addressData = JSON.parse(
    fs.readFileSync(
      process.env.ADDRESS_FILE
        || `${homedir}/.ocean/ocean-contracts/artifacts/address.json`,
      'utf8'
    )
  );
  const addresses = addressData[process.env.OCEAN_NETWORK];

  oceanConfig = {
    ...oceanConfig,
    oceanTokenAddress: addresses.Ocean,
    poolTemplateAddress: addresses.poolTemplate,
    fixedRateExchangeAddress: addresses.FixedPrice,
    dispenserAddress: addresses.Dispenser,
    erc721FactoryAddress: addresses.ERC721Factory,
    sideStakingAddress: addresses.Staking,
    opfCommunityFeeCollector: addresses.OPFCommunityFeeCollector
  };
}

oceanConfig = {
  ...oceanConfig,
  nodeUri: process.env.OCEAN_NETWORK_URL,
  // Set optional properties - Provider URL and Aquarius URL
  metadataCacheUri: process.env.AQUARIUS_URL || oceanConfig.metadataCacheUri,
  providerUri: process.env.PROVIDER_URL || oceanConfig.providerUri
};

const web3Provider = new HDWalletProvider(
  process.env.PRIVATE_KEY,
  oceanConfig.nodeUri
```

```
);

module.exports = {
  web3Provider,
  oceanConfig
};
```

ocean.py

config.py

```
import os
from dotenv import load_dotenv
from ocean_lib.example_config import get_config_dict
from ocean_lib.web3_internal.utils import connect_to_network
from ocean_lib.ocean.ocean import Ocean

load_dotenv()

# Create Ocean instance

network_name = os.getenv("OCEAN_NETWORK")
connect_to_network(network_name)

config = get_config_dict(network_name)
ocean = Ocean(config)

from brownie.network import accounts
accounts.clear()
user_private_key = os.getenv('PRIVATE_KEY')
wallet = accounts.add(user_private_key)
```

Now, all the dependencies are ready and you can proceed with interacting with Ocean infrastructure using Ocean libraries.

Creating a data NFT

This tutorial guides you through the process of creating your own data NFT using Ocean libraries. To know more about data NFT please refer this page.

Prerequisites

- Obtain an API key
- Set up the .env file
- Install the dependencies
- Create a configuration file

Create a script to deploy dataNFT

Create a new file in the same working directory where configuration file (`config.py` / `config.js`) and `.env` files are present, and copy the code as listed below.

ocean.js

create_dataNFT.js

```
// Import dependencies
const { NftFactory } = require('@oceanprotocol/lib');
const Web3 = require('web3');

// Note: Make sure .env file and config.js are created and setup correctly
const { web3Provider, oceanConfig } = require('./config');

const web3 = new Web3(web3Provider);

// Deinfe a function which will create a dataNFT using Ocean.js library
const createDataNFT = async () => {

  // Create a NFTFactory
  const Factory = new NftFactory(oceanConfig.erc721FactoryAddress, web3);

  const accounts = await web3.eth.getAccounts();
  const publisherAccount = accounts[0];

  // Define dataNFT parameters
  const nftParams = {
    name: '72120Bundle',
    symbol: '72Bundle',
    // Optional parameters
    templateIndex: 1,
    tokenURI: 'https://example.com',
    transferable: true,
    owner: publisherAccount
  };

  // Call a Factory.createNFT(...) which will create a new dataNFT
  const erc721Address = await Factory.createNFT(
    publisherAccount,
    nftParams
  );

  return {
    erc721Address
  };
};

// Call the create createDataNFT() function
createDataNFT()
  .then(({ erc721Address }) => {
    console.log(`DataNft address ${erc721Address}`);
    process.exit();
  })
```

```
  .catch((err) => {
    console.error(err);
    process.exit(1);
  });
```

Executing script

```
node create_dataNFT.js
```

ocean.py

create_dataNFT.py

```
# Note: Ensure that .env and config.py are correctly setup
from config import web3_wallet, ocean

data_nft = ocean.create_data_nft(
    name="NFTToken1",
    symbol="NFT1",
    from_wallet=web3_wallet,
    # Optional parameters
    token_uri="https://example.com",
    template_index=1,
    transferable=True,
    owner=web3_wallet.address,
)
print(f"Created dataNFT. Its address is {data_nft.address}")
```

Executing script

```
python create_dataNFT.py
```

Publish with Fixed Pricing

This tutorial guides you through the process of creating your own data NFT and a datatoken with fixed pricing, using Ocean libraries. To know more about data NFTs and datatokens please refer this page. Ocean Protocol supports different pricing schemes which can be set while publishing an asset. Please refer this page for more details on pricing schemes.

Prerequisites

- Obtain an API key
- Set up the .env file
- Install the dependencies
- Create a configuration file

Create a script to deploy data NFT and datatoken with fixed pricing.

Create a new file in the same working directory where configuration file (`config.py` / `config.js`) and `.env` files are present, and copy the code as listed below.

> ⓘ **Fees**: The code snippets below define fees related parameters. Please refer fees page for more details

ocean.js

create_datatoken_with_fre.js

```
// Import dependencies
const { NftFactory, Datatoken } = require('@oceanprotocol/lib');
const Web3 = require('web3');
const { web3Provider, oceanConfig } = require('./config');

// Create a web3 instance
const web3 = new Web3(web3Provider);

// Define a function createFRE()
const createFRE = async () => {
  const Factory = new NftFactory(oceanConfig.erc721FactoryAddress, web3);

  // Get accounts from web3 instance
  const accounts = await web3.eth.getAccounts();
  const publisherAccount = accounts[0];

  // data NFT parameters: name, symbol, templateIndex, etc.
  const nftParams = {
    name: '72120Bundle',
    symbol: '72Bundle',
    templateIndex: 1,
    tokenURI: 'https://example.com',
    transferable: true,
    owner: publisherAccount
  };

  // datatoken parameters: name, symbol, templateIndex, etc.
  const erc20Params = {
    name: "Sample datatoken",
    symbol: "SDT",
    templateIndex: 1,
    cap: '100000',
    feeAmount: '0',
    // paymentCollector is the address
    paymentCollector: '0x0000000000000000000000000000000000000000',
    feeToken: '0x0000000000000000000000000000000000000000',
    minter: publisherAccount,
    mpFeeAddress: '0x0000000000000000000000000000000000000000'
  };

  const fixedPriceParams = {
    fixedRateAddress: oceanConfig.fixedRateExchangeAddress,
    baseTokenAddress: oceanConfig.oceanTokenAddress,
    owner: publisherAccount,
    marketFeeCollector: publisherAccount,
    baseTokenDecimals: 18,
```

```
    datatokenDecimals: 18,
    fixedRate: '1',
    marketFee: '0',
    // Optional parameters
    // allowedConsumer: publisherAccount,  //  only account that consume the exhcha
    withMint: false // add FixedPriced contract as minter if withMint == true
  }

  // Create data NFT and a datatoken with Fixed Rate exchange
  const result = await Factory.createNftErc20WithFixedRate(
    publisherAccount,
    nftParams,
    erc20Params,
    fixedPriceParams
  );

  // Get the data NFT address and datatoken address from the result
  const erc721Address = result.events.NFTCreated.returnValues[0];
  const datatokenAddress = result.events.TokenCreated.returnValues[0];

  return {
    erc721Address,
    datatokenAddress
  };
};

// Call the createFRE() function
createFRE()
  .then(({ erc721Address, datatokenAddress }) => {
    console.log(`DataNft address ${erc721Address}`);
    console.log(`Datatoken address ${datatokenAddress}`);
    process.exit(1);

  })
  .catch((err) => {
    console.error(err);
    process.exit(1);
  });
```

Execute script

```
node create_datatoken_with_fre.js
```

ocean.py

create_datatoken_with_fre.py

```
# Note: Ensure that .env and config.py are correctly setup
from config import web3_wallet, ocean

data_nft = ocean.create_data_nft(
```

```
    name="NFTToken1",
    symbol="NFT1",
    from_wallet=web3_wallet,
    # Optional parameters
    token_uri="https://example.com",
    template_index=1,
    transferable=True,
    owner=web3_wallet.address,
)
print(f"Created dataNFT. Its address is {data_nft.address}")

# replace the addresses here
fee_manager = "0x0000000000000000000000000000000000000000"
publish_market_order_fee_address = "0x0000000000000000000000000000000000000000"
publish_market_order_fee_token = "0x0000000000000000000000000000000000000000"
minter = web3_wallet.address

# replace the fee amount
publish_market_order_fee_amount = 0

datatoken = data_nft.create_datatoken(
    name="Datatoken 1",
    symbol="DT1",
    datatoken_cap="100000",
    from_wallet=web3_wallet,
    # Ootional parameters below
    template_index=1,
    fee_manager=fee_manager,
    publish_market_order_fee_token=publish_market_order_fee_token,
    publish_market_order_fee_amount=publish_market_order_fee_amount,
    minter=minter,
    publish_market_order_fee_address=publish_market_order_fee_address,
)
print(f"Created datatoken. Its address is {datatoken.address}")


exchange_id = ocean.create_fixed_rate(
    datatoken=datatoken,
    base_token=ocean.OCEAN_token,
    amount=ocean.to_wei(100),
    from_wallet=web3_wallet,
)

print(f"Created fixed rate exchange with ID {exchange_id.hex()}")
```

Execute script

```
python create_datatoken_with_fre.py
```

Mint Datatokens

This tutorial guides you through the process of minting datatokens and sending them to a receiver address. The tutorial assumes that you already have the address of the datatoken contract which is owned by you.

Prerequisites

- Obtain an API key
- Set up the .env file
- Install the dependencies
- Create a configuration file

Create a script to mint datatokens

Create a new file in the same working directory where configuration file (`config.py` / `config.js`) and `.env` files are present, and copy the code as listed below.

ocean.js

mint_datatoken.js

```
// Import dependencies
const { NftFactory, Datatoken } = require('@oceanprotocol/lib');
const Web3 = require('web3');
const { web3Provider, oceanConfig } = require('./config');

// Create a web3 instance
const web3 = new Web3(web3Provider);

// Change this
const datatokenAddress = "0xD3542e5F56655fb818F9118CE219e1D10751BC82"
const receiverAddress = "0xBE5449a6A97aD46c8558A3356267Ee5D2731ab5e"

// Create a function which will take `datatokenAddress` and `receiverAddress` as pa
const mintDatatoken = async (datatokenAddress, receiverAddress) => {
  const accounts = await web3.eth.getAccounts();
  const publisherAccount = accounts[0];

  // Create datatoken instance
  const datatoken = new Datatoken(web3);

  // Get current datatoken balance of receiver
  let receiverBalance = await datatoken.balance(
    datatokenAddress,
    receiverAddress
  );
  console.log(`Receiver balance before mint: ${receiverBalance}`);

  // Mint datatoken
  await datatoken.mint(
    datatokenAddress,
    publisherAccount,
    '1',
    receiverAddress
  );

  // Get new datatoken balance of receiver
  receiverBalance = await datatoken.balance(
    datatokenAddress,
    receiverAddress
  );
  console.log(`Receiver balance after mint: ${receiverBalance}`);
};

// Call mintDatatoken(...) function defined above
mintDatatoken(datatokenAddress, receiverAddress)
  .then(() => {
```

```
    process.exit(1);
  });
})
.catch((err) => {
  console.error(err);
  process.exit(1);
});
```

Execute script

```
node mint_datatoken.js
```

ocean.py

mint_datatoken.py

```
# Note: Ensure that .env and config.py are correctly setup
from config import web3_wallet, ocean

# Change this
datatoken_address = "0xD3542e5F56655fb818F9118CE219e1D10751BC82"
receiver_address = "0xBE5449a6A97aD46c8558A3356267Ee5D2731ab5e"

datatoken = ocean.get_datatoken(datatoken_address)

print(f"Balance before mint: {datatoken.balanceOf(receiver_address)}")

# Mint datatokens
datatoken.mint(
    account_address=receiver_address,
    value=ocean.to_wei("1"),
    from_wallet=web3_wallet,
)

print(f"Balance after mint: {datatoken.balanceOf(receiver_address)}")
nt_d
```

Execute script

```
python mint_datatoken.py
```

Update Metadata

This tutorial will guide you to update an existing asset published on-chain using Ocean libraries. The tutorial assumes that you already have the `did` of the asset which needs to be updated. In this tutorial, we will update the name, description, tags of the data NFT. Please refer the page on DDO to know more about additional the fields which can be updated.

Prerequisites

- Obtain an API key
- Set up the .env file
- Install the dependencies
- Create a configuration file

> ⓘ The variable **AQUARIUS_URL** and **PROVIDER_URL** should be set correctly in `.env` file

Create a script to update the metadata

Create a new file in the same working directory where configuration file (`config.py` / `config.js`) and `.env` files are present, and copy the code as listed below.

ocean.js

updateMetadata.js

```
// Import dependencies
const {
  Nft,
  ProviderInstance,
  getHash,
  Aquarius
} = require('@oceanprotocol/lib');
const { SHA256 } = require('crypto-js');
const Web3 = require('web3');
const { web3Provider, oceanConfig } = require('./config');

// Create a web3 instance
const web3 = new Web3(web3Provider);

// Create Aquarius instance
const aquarius = new Aquarius(oceanConfig.metadataCacheUri);
const nft = new Nft(web3);
const providerUrl = oceanConfig.providerUri;

// replace the did here
const did = "did:op:a419f07306d71f3357f8df74807d5d12bddd6bcd738eb0b461470c64859d6f0

// This function takes did as a parameter and updates the data NFT information
const setMetadata = async (did) => {
  const accounts = await web3.eth.getAccounts();
  const publisherAccount = accounts[0];

  // Fetch ddo from Aquarius
  const ddo = await aquarius.resolve(did);

  // update the ddo here
  ddo.metadata.name = "Sample dataset v2";
  ddo.metadata.description = "Lorem ipsum dolor sit amet, consetetur sadipscing eli
  ddo.metadata.tags = ["new tag1", "new tag2"];

  providerResponse = await ProviderInstance.encrypt(ddo, providerUrl);
  const encryptedResponse = await providerResponse;
  const metadataHash = getHash(JSON.stringify(ddo));

  // Update the data NFT metadata
  await nft.setMetadata(
    ddo.nftAddress,
    publisherAccount,
    0,
    providerUrl,
    '',
```

```
    '0x2',
    encryptedResponse,
    `0x${metadataHash}`
  );

  // Check if ddo is correctly udpated in Aquarius
  await aquarius.waitForAqua(ddo.id);

  console.log(`Resolved asset did [${ddo.id}]from aquarius.`);
  console.log(`Updated name: [${ddo.metadata.name}].`);
  console.log(`Updated description: [${ddo.metadata.description}].`);
  console.log(`Updated tags: [${ddo.metadata.tags}].`);

};

// Call setMetadata(...) function defined above
setMetadata(did).then(() => {
  process.exit();
}).catch((err) => {
  console.error(err);
  process.exit(1);
});
```

Execute the script

```
node updateMetadata.js
```

Compute-to-Data

Providing access to data in a privacy-preserving fashion

Quick Start

- Compute-to-Data example

Motivation

The most basic scenario for a Publisher is to provide access to the datasets they own or manage. However, a Publisher may offer a service to execute some computation on top of their data. This has some benefits:

- The data **never** leaves the Publisher enclave.
- It's not necessary to move the data; the algorithm is sent to the data.
- Having only one copy of the data and not moving it makes it easier to be compliant with data protection regulations.

This page elaborates on the benefits.

Further Reading

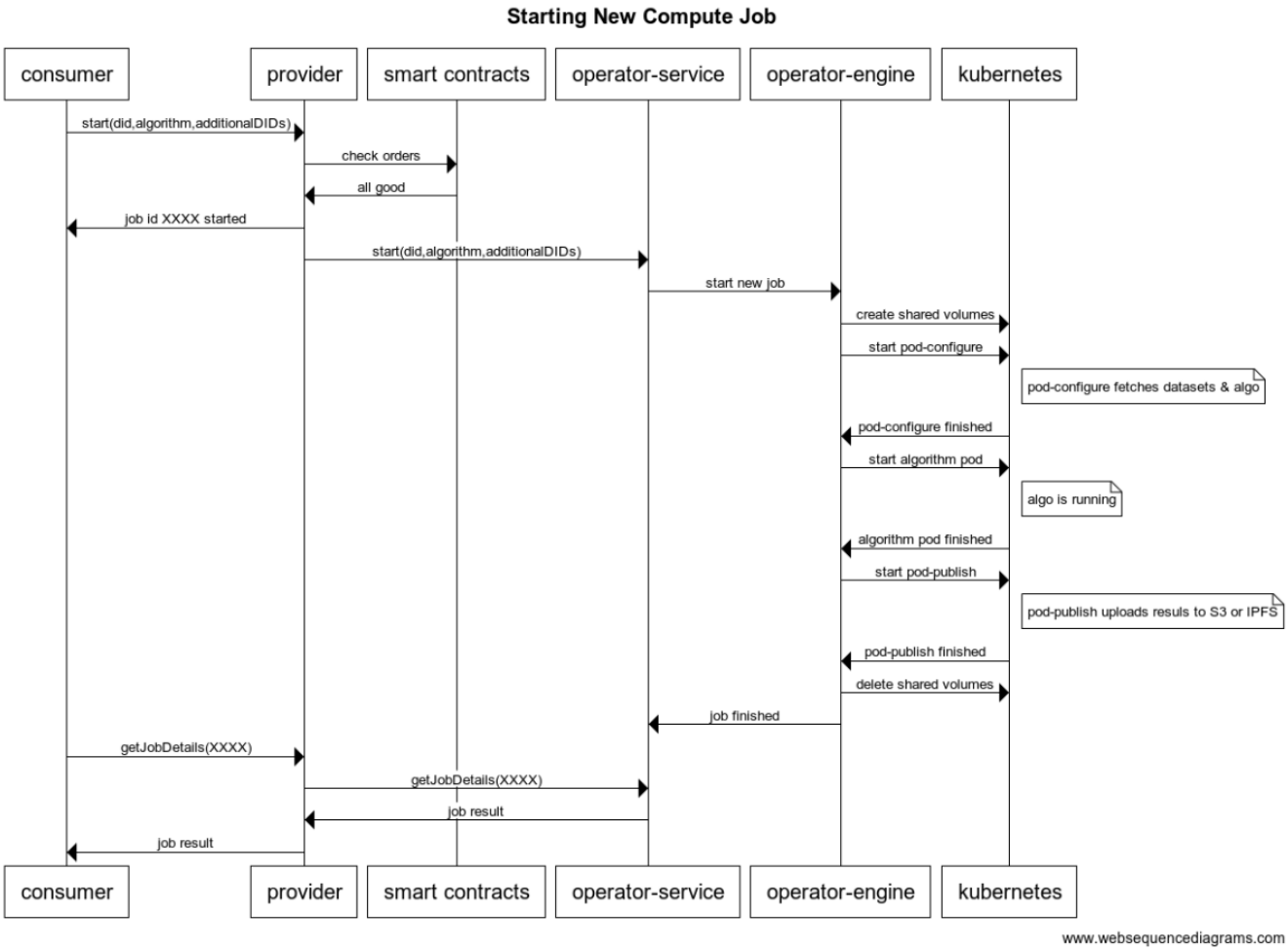
- Compute-to-Data architecture
- Tutorial: Writing Algorithms
- Tutorial: Set Up a Compute-to-Data Environment
- Use Compute-to-Data in Ocean Market
- Build ML models via Ocean Market or Python
- Compute-to-Data Python Quickstart
- (Old) Compute-to-Data specs (OEP12)

Architecture

Architecture overview

Architecture Overview

Here's the sequence diagram for starting a new compute job.



Sequence Diagram for computing services

The Consumer calls the Provider with `start(did, algorithm, additionalDIDs)`. It returns job id `XXXX`. The Provider oversees the rest of the work. At any point, the Consumer can query the Provider for the job status via `getJobDetails(XXXX)`.

Here's how Provider works. First, it ensures that the Consumer has sent the appropriate datatokens to get access. Then, it calls asks the Operator-Service (a microservice) to start the job, which passes on the request to Operator-Engine (the actual compute system). Operator-Engine runs Kubernetes compute jobs etc as needed. Operator-Engine reports when to Operator-Service when the job has finished.

Here's the actors/components:

- Consumers - The end users who need to use some computing services offered by the same Publisher as the data Publisher.
- Operator-Service - Micro-service that is handling the compute requests.
- Operator-Engine - The computing systems where the compute will be executed.
- Kubernetes - a K8 cluster

Before the flow can begin, these pre-conditions must be met:

- The Asset DDO has a `compute` service.
- The Asset DDO compute service must permit algorithms to run on it.
- The Asset DDO must specify an Ocean Provider endpoint exposed by the Publisher.

Access Control using Ocean Provider

As with the `access` service, the `compute` service requires the **Ocean Provider** as a component handled by Publishers. Ocean Provider is in charge of interacting with users and managing the basics of a Publisher's infrastructure to integrate this infrastructure into Ocean Protocol. The direct interaction with the infrastructure where the data resides happens through this component only.

Ocean Provider includes the credentials to interact with the infrastructure (initially in cloud providers, but it could be on-premise).

Compute-to-Data Environment

Operator Service

The **Operator Service** is a micro-service in charge of managing the workflow executing requests.

The main responsibilities are:

- Expose an HTTP API allowing for the execution of data access and compute endpoints.
- Interact with the infrastructure (cloud/on-premise) using the Publisher's credentials.
- Start/stop/execute computing instances with the algorithms provided by users.
- Retrieve the logs generated during executions.

Typically the Operator Service is integrated from Ocean Provider, but can be called independently of it.

The Operator Service is in charge of establishing the communication with the K8s cluster, allowing it to:

- Register new compute jobs
- List the current compute jobs
- Get a detailed result for a given job
- Stop a running job

The Operator Service doesn't provide any storage capability, all the state is stored directly in the K8s cluster.

Operator Engine

The **Operator Engine** is in charge of orchestrating the compute infrastructure using Kubernetes as backend where each compute job runs in an isolated Kubernetes Pod. Typically the Operator Engine retrieves the workflows created by the Operator Service in Kubernetes, and manage the infrastructure necessary to complete the execution of the compute workflows.

The Operator Engine is in charge of retrieving all the workflows registered in a K8s cluster, allowing to:

- Orchestrate the flow of the execution
- Start the configuration pod in charge of download the workflow dependencies (datasets and algorithms)
- Start the pod including the algorithm to execute
- Start the publishing pod that publish the new assets created in the Ocean Protocol network.
- The Operator Engine doesn't provide any storage capability, all the state is stored directly in the K8s cluster.

Pod: Configuration

Pod: Publishing

Datasets & Algorithms

Datasets and Algorithms

Datasets & Algorithms

With Compute-to-Data, datasets are not allowed to leave the premises of the data holder, only algorithms can be permitted to run on them under certain conditions within an isolated and secure environment. Algorithms are an asset type just like datasets and can be priced in the same way.

Algorithms can be public or private by setting `"attributes.main.type"` value in DDO as follows:

- `"access"` - public. The algorithm can be downloaded, given appropriate datatoken.
- `"compute"` - private. The algorithm is only available to use as part of a compute job without any way to download it. The Algorithm must be published on the same Ocean Provider as the dataset it's targeted to run on.

For each dataset, publishers can choose to allow various permission levels for algorithms to run:

- allow selected algorithms, referenced by their DID
- allow all algorithms published within a network or marketplace
- allow raw algorithms, for advanced use cases circumventing algorithm as an asset type, but most prone to data escape

All implementations should set permissions to private by default: upon publishing a compute dataset, no algorithms should be allowed to run on it. This is to prevent data escape by a rogue algorithm being written in a way to extract all data from a dataset.

Minikube Environment

Requirements

- functioning internet-accessable provider service
- machine capable of running compute (e.g. we used a machine with 8 CPUs, 16 GB Ram, 100GB SSD and fast internet connection)
- Ubuntu 20.04

Install Docker and Git

```
sudo apt update
sudo apt install git docker.io
sudo usermod -aG docker $USER && newgrp docker
```

Install Minikube

```
wget -q --show-progress https://github.com/kubernetes/minikube/releases/download/v1.22.0/minil
sudo dpkg -i minikube_1.22.0-0_amd64.deb
```

Start Minikube

First command is imporant, and solves a PersistentVolumeClaims problem.

```
minikube config set kubernetes-version v1.16.0
minikube start --cni=calico --driver=docker --container-runtime=docker
```

Depending on the number of available CPUs, RAM, and the required resources for running the job, consider adding options `--cpu`, `--memory`, and `--disk-size` to avoid runtime issues.

For other options to run minikube refer to this link

Install kubectl

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/li
curl -LO "https://dl.k8s.io/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd6
echo "$(<kubectl.sha256) kubectl" | sha256sum --check

sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

Wait untill all the defaults are running (1/1).

```
watch kubectl get pods --all-namespaces
```

Run IPFS host

```
export ipfs_staging=~/ipfs_staging
export ipfs_data=~/ipfs_data

docker run -d --name ipfs_host -v $ipfs_staging:/export -v $ipfs_data:/data/ipfs -p 4001:4001

sudo /bin/sh -c 'echo "127.0.0.1    youripfsserver" >> /etc/hosts'
```

Storage class (Optional)

For minikube, you can use the default 'standard' class.

For AWS, please make sure that your class allocates volumes in the same region and zone in which you are running your pods.

We created our own 'standard' class in AWS:

```
kubectl get storageclass standard -o yaml
```

```
allowedTopologies:
- matchLabelExpressions:
    - key: failure-domain.beta.kubernetes.io/zone
          values:
          - us-east-1a
apiVersion: storage.k8s.io/v1
kind: StorageClass
parameters:
    fsType: ext4
    type: gp2
provisioner: kubernetes.io/aws-ebs
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

For more information, please visit
`https://kubernetes.io/docs/concepts/storage/storage-classes/`

Download and Configure Operator Service

Open new terminal and run the command below.

```
git clone https://github.com/oceanprotocol/operator-service.git
```

Edit `operator-service/kubernetes/postgres-configmap.yaml`. Change `POSTGRES_PASSWORD` to nice long random password.

Edit `operator-service/kubernetes/deployment.yaml`. Optionally change:

- `ALGO_POD_TIMEOUT`
- add `requests_cpu`
- add `requests_memory`
- add `limits_cpu`
- add `limits_memory`

```
...
    spec:
      containers:
      - env:
        - name: requests_cpu
          value: "4"
        - name: requests_memory
          value: "8Gi"
        - name: limits_cpu
          value: "8"
        - name: limits_memory
          value: "15Gi"
        - name: ALGO_POD_TIMEOUT
          value: "3600"
...
```

Download and Configure Operator Engine

```
git clone https://github.com/oceanprotocol/operator-engine.git
```

Check the README section of operator engine to customize your deployment.

At a minimum you should add your IPFS URLs or AWS settings, and add (or remove) notification URLs.

Create namespaces

```
kubectl create ns ocean-operator
kubectl create ns ocean-compute
```

Deploy Operator Service

```
kubectl config set-context --current --namespace ocean-operator
kubectl create -f operator-service/kubernetes/postgres-configmap.yaml
kubectl create -f operator-service/kubernetes/postgres-storage.yaml
kubectl create -f operator-service/kubernetes/postgres-deployment.yaml
kubectl create -f operator-service/kubernetes/postgresql-service.yaml
kubectl apply  -f operator-service/kubernetes/deployment.yaml
```

Deploy Operator Engine

```
kubectl config set-context --current --namespace ocean-compute
kubectl apply  -f operator-engine/kubernetes/sa.yml
kubectl apply  -f operator-engine/kubernetes/binding.yml
kubectl apply  -f operator-engine/kubernetes/operator.yml
kubectl create -f operator-service/kubernetes/postgres-configmap.yaml
```

Optional: For production enviroments, it's safer to block access to metadata. To do so run the below command:

```
kubectl -n ocean-compute apply -f /ocean/operator-engine/kubernetes/egress.yaml
```

Expose Operator Service

```
kubectl expose deployment operator-api --namespace=ocean-operator --port=8050
```

Run a port forward or create your ingress service and setup DNS and certificates (not covered here):

```
kubectl -n ocean-operator port-forward svc/operator-api 8050
```

Alternatively you could use another method to communicate between the C2D Environment and the provider, such as an SSH tunnel.

Initialize database

If your minikube is running on compute.example.com:

```
curl -X POST "https://compute.example.com/api/v1/operator/pgsqlinit" -H  "accept: application/
```

Update Provider

Update your provider service by updating the `operator_service.url` value in `config.ini`

```
operator_service.url = https://compute.example.com/
```

Restart your provider service.

Watch the explanatory video for more details

Writing Algorithms

Learn how to write algorithms for use in Ocean Protocol's Compute-to-Data feature.

Overview

An algorithm in the Ocean Protocol stack is another asset type, in addition to data sets. An algorithm for Compute to Data is composed of the following:

- an algorithm code
- a Docker image (base image + tag)
- an entry point

Environment

When creating an algorithm asset in Ocean Protocol, the additional `algorithm` object needs to be included in its metadata service to define the Docker container environment:

```
{
  "algorithm": {
    "container": {
      "entrypoint": "node $ALGO",
      "image": "node",
      "tag": "latest"
    }
  }
}
```

Variable	Usage
`image`	The Docker image name the algorithm will run wit
`tag`	The Docker image tag that you are going to use.
`entrypoint`	The Docker entrypoint. `$ALGO` is a macro that gets replaced inside the compute job, depending where your algorithm code is downloaded.

Define your entrypoint according to your dependencies. E.g. if you have multiple versions of python installed, use the appropriate command `python3.6 $ALGO`.

What Docker container should I use?

There are plenty of Docker containers that work out-of-the-box. However, if you have custom dependencies,

you may want to configure your own Docker Image. To do so, create a Dockerfile with the appropriate instructions for dependency management and publish the container, e.g. using Dockerhub.

We also collect some example images which you can also view in Dockerhub.

When publishing an algorithm through the Ocean Market, these properties can be set via the publish UI.

Environment Examples

Run an algorithm written in JavaScript/Node.js, based on Node.js v14:

```
{
  "algorithm": {
    "container": {
      "entrypoint": "node $ALGO",
      "image": "node",
      "tag": "14"
    }
  }
}
```

Run an algorithm written in Python, based on Python v3.9:

```
{
  "algorithm": {
    "container": {
      "entrypoint": "python3.9 $ALGO",
      "image": "python",
      "tag": "3.9.4-alpine3.13"
    }
  }
}
```

Data Storage

As part of a compute job, every algorithm runs in a K8s pod with these volumes mounted:

Path	Permissions	Usage
`/data/inputs`	read	Storage for input data sets, accessible only to the algorithm running in the pod. Contents will be the files themselves, inside indexed folders e.g. `/data/inputs/{did}/{service_id}`.
`/data/ddos`	read	Storage for all DDOs involved in a compute job (input data set + algorithm). Contents will be json files containing the DDO structure.
`/data/outputs`	read/write	Storage for all of the algorithm's output files. They are uploaded on some form of cloud storage, and URLs are sent back to the consumer.
`/data/logs/`	read/write	All algorithm output (such as `print`, `console.log`, etc.) is stored in a file located in this folder. They are stored and sent to the consumer as well.

Please note that when using local Providers or Metatata Caches, the ddos might not be correctly transferred into c2d, but inputs are still available. If your algorithm relies on contents from the DDO json structure, make sure to use a public Provider and Metadata Cache (Aquarius instance).

Environment variables available to algorithms

For every algorithm pod, the Compute to Data environment provides the following environment variables:

Variable	Usage
`DIDS`	An array of DID strings containing the input datasets.
`TRANSFORMATION_DID`	The DID of the algorithm.

Example: JavaScript/Node.js

The following is a simple JavaScript/Node.js algorithm, doing a line count for ALL input datasets. The

algorithm is not using any environment variables, but instead it's scanning the `/data/inputs` folder.

```
const fs = require('fs')

const inputFolder = '/data/inputs'
const outputFolder = '/data/outputs'

async function countrows(file) {
  console.log('Start counting for ' + file)
  const fileBuffer = fs.readFileSync(file)
  const toString = fileBuffer.toString()
  const splitLines = toString.split('\n')
  const rows = splitLines.length - 1
  fs.appendFileSync(outputFolder + '/output.log', file + ',' + rows + '\r\n')
  console.log('Finished. We have ' + rows + ' lines')
}

async function processfolder(folder) {
  const files = fs.readdirSync(folder)

  for (const i = 0; i < files.length; i++) {
    const file = files[i]
    const fullpath = folder + '/' + file
    if (fs.statSync(fullpath).isDirectory()) {
      await processfolder(fullpath)
    } else {
      await countrows(fullpath)
    }
  }
}

processfolder(inputFolder)
```

This snippet will create and expose the following files as compute job results to the consumer:

- `/data/outputs/output.log`
- `/data/logs/algo.log`

To run this, use the following container object:

```
{
  "algorithm": {
    "container": {
      "entrypoint": "node $ALGO",
      "image": "node",
      "tag": "12"
    }
  }
}
```

Example: Python

A more advanced line counting in Python, which relies on environment variables and constructs a job object, containing all the input files & DDOs

```
import pandas as pd
import numpy as np
import os
import time
import json

def get_job_details():
    """Reads in metadata information about assets used by the algo"""
    job = dict()
    job['dids'] = json.loads(os.getenv('DIDS', None))
    job['metadata'] = dict()
    job['files'] = dict()
    job['algo'] = dict()
    job['secret'] = os.getenv('secret', None)
    algo_did = os.getenv('TRANSFORMATION_DID', None)
    if job['dids'] is not None:
        for did in job['dids']:
            # get the ddo from disk
            filename = '/data/ddos/' + did
            print(f'Reading json from {filename}')
            with open(filename) as json_file:
                ddo = json.load(json_file)
                # search for metadata service
                for service in ddo['service']:
                    if service['type'] == 'metadata':
                        job['files'][did] = list()
                        index = 0
                        for file in service['attributes']['main']['files']:
                            job['files'][did].append(
                                '/data/inputs/' + did + '/' + str(index))
                            index = index + 1
    if algo_did is not None:
        job['algo']['did'] = algo_did
        job['algo']['ddo_path'] = '/data/ddos/' + algo_did
    return job



def line_counter(job_details):
    """Executes the line counter based on inputs"""
    print('Starting compute job with the following input information:')
    print(json.dumps(job_details, sort_keys=True, indent=4))

    """ Now, count the lines of the first file in first did """
    first_did = job_details['dids'][0]
    filename = job_details['files'][first_did][0]
    non_blank_count = 0
    with open(filename) as infp:
        for line in infp:
            if line.strip():
                non_blank_count += 1
    print ('number of non-blank lines found %d' % non_blank_count)

    """ Print that number to output to generate algo output"""
    f = open("/data/outputs/result", "w")
```

```
    f.write(str(non_blank_count))
    f.close()


if __name__ == '__main__':
    line_counter(get_job_details())
```

To run this algorithm, use the following `container` object:

```
{
  "algorithm": {
    "container": {
      "entrypoint": "python3.6 $ALGO",
      "image": "oceanprotocol/algo_dockers",
      "tag": "python-sql"
    }
  }
}
```

Private Docker Registry

Learn how to setup your own docker registry and push images for running algorithms in a C2D environment.

The document is intended for a production setup. The tutorial provides the steps to setup a private docker registry on the server for the following scenarios:

- Allow registry access only to the C2D environment.
- Anyone can pull the image from the registry but, only authenticated users will push images to the registry.

Setup 1: Allow registry access only to the C2D environment

To implement this use case, 1 domain will be required:

- **example.com**: This domain will allow only image pull operations

Note: Please change the domain names to your application-specific domain names.

1.1 Prerequisites

- Running docker environment on the linux server.
- Docker compose is installed.
- C2D environment is running.
- The domain names is mapped to the server hosting the registry.

1.2 Generate certificates

```
# install certbot: https://certbot.eff.org/
sudo certbot certonly --standalone --cert-name example.com -d example.com
```

Note: Do check the access right of the files/directories where certificates are stored. Usually, they are at `/etc/letsencrypt/`.

1.3 Generate password file

Replace content in `<>` with appropriate content.

```
docker run \
  --entrypoint htpasswd \
  httpd:2 -Bbn <username> <password> > <path>/auth/htpasswd
```

1.4 Docker compose template file for registry

Copy the below yml content to `docker-compose.yml` file and replace content in `<>`.

```
version: '3'

services:
  registry:
    restart: always
    container_name: my-docker-registry
    image: registry:2
    ports:
      - 5050:5000
    environment:
      REGISTRY_AUTH: htpasswd
      REGISTRY_AUTH_HTPASSWD_PATH: /auth/htpasswd
      REGISTRY_AUTH_HTPASSWD_REALM: Registry Realm
      REGISTRY_HTTP_SECRET: <secret>
    volumes:
      - <path>/data:/var/lib/registry
      - <path>/auth:/auth
  nginx:
        image: nginx:latest
        container_name: nginx
        volumes:
            -  <path>/nginx/logs:/app/logs/
            -  nginx.conf:/etc/nginx/nginx.conf
            -  /etc/letsencrypt/:/etc/letsencrypt/
        ports:
            - 80:80
            - 443:443
        depends_on:
            - registry
```

1.5 Nginx configuration

Copy the below nginx configuration to a `nginx.conf` file.

```
events {}
http {
  access_log /app/logs/access.log;
  error_log /app/logs/error.log;

        server {
        client_max_body_size 4096M;
                listen 80 default_server;
                server_name _;
                return 301 https://$host$request_uri;
        }

  server {
          # Allowed request size should be large enough to allow pull operations
          client_max_body_size 4096M;
          listen 443 ssl;
          server_name example.com;
          ssl_certificate /etc/letsencrypt/live/example.com/fullchain.pem;
          ssl_certificate_key /etc/letsencrypt/live/example.com/privkey.pem;
          location / {
                  proxy_connect_timeout 75s;
                  proxy_pass http://registry-read-only:5000;
          }
  }
}
```

1.6 Create kubernetes secret in C2D server

Login into Compute-to-data enviroment and run the following command with appropriate credentials:

```
kubectl create secret docker-registry regcred --docker-server=example.com --docker-username=<
```

1.7 Update operator-engine configuration

Add `PULL_SECRET` property with value `regcred` in the operator.yml file of operator-engine configuration. For more detials on operator-engine properties refer this link

Apply updated operator-engine configuration.

```
kubectl config set-context --current --namespace ocean-compute
kubectl apply  -f operator-engine/kubernetes/operator.yml
```

Steup 2: Allow anyonymous `pull` operations

To implement this use case, 2 domains will be required:

- **example.com**: This domain will allow image push/pull operations only to the authenticated users.
- **readonly.example.com**: This domain will allow only image pull operations

Note: Please change the domain names to your application-specific domain names.

2.1 Prerequisites

- Running docker environment on the linux server.
- Docker compose is installed.
- 2 domain names is mapped to the same server IP address.

2.2 Generate certificates

```
# install certbot: https://certbot.eff.org/
sudo certbot certonly --standalone --cert-name example.com -d example.com
sudo certbot certonly --standalone --cert-name readonly.example.com -d readonly.example.com
```

Note: Do check the access right of the files/directories where certificates are stored. Usually, they are at `/etc/letsencrypt/`.

2.3 Generate password file

Replace content in `<>` with appropriate content.

```
docker run \
  --entrypoint htpasswd \
  httpd:2 -Bbn <username> <password> > <path>/auth/htpasswd
```

2.4 Docker compose template file for registry

Copy the below yml content to `docker-compose.yml` file and replace content in `<>`. Here, we will be creating two services of the docker registry so that anyone can `pull` the images from the registry but, only authenticated users can `push` the images.

```
version: '3'

services:
  registry:
    restart: always
    container_name: my-docker-registry
    image: registry:2
    ports:
      - 5050:5000
    environment:
      REGISTRY_AUTH: htpasswd
      REGISTRY_AUTH_HTPASSWD_PATH: /auth/htpasswd
      REGISTRY_AUTH_HTPASSWD_REALM: Registry Realm
      REGISTRY_HTTP_SECRET: <secret>
    volumes:
      - <path>/data:/var/lib/registry
      - <path>/auth:/auth
  registry-read-only:
      restart: always
      container_name: my-registry-read-only
      image: registry:2
      read_only: true
      ports:
        - 5051:5000
      environment:
        REGISTRY_HTTP_SECRET: ${REGISTRY_HTTP_SECRET}
      volumes:
        - <path>/docker-registry/data:/var/lib/registry:ro
      depends_on:
        - registry
  nginx:
        image: nginx:latest
        container_name: nginx
        volumes:
            -  <path>/nginx/logs:/app/logs/
            -  nginx.conf:/etc/nginx/nginx.conf
            -  /etc/letsencrypt/:/etc/letsencrypt/
        ports:
            - 80:80
            - 443:443
        depends_on:
            - registry-read-only
```

2.5 Nginx configuration

Copy the below nginx configuration to a `nginx.conf` file.

```
events {}
http {
  access_log /app/logs/access.log;
  error_log /app/logs/error.log;

        server {
        client_max_body_size 4096M;
                listen 80 default_server;
                server_name _;
                return 301 https://$host$request_uri;
        }

  server {
           # Allowed request size should be large enough to allow push operations
           client_max_body_size 4096M;
           listen 443 ssl;
           server_name readonly.example.com;
           ssl_certificate /etc/letsencrypt/live/readonly.example.com/fullchain.pem;
           ssl_certificate_key /etc/letsencrypt/live/readonly.example.com/privkey.pem;
           location / {
                   proxy_connect_timeout 75s;
                   proxy_pass http://registry:5000;
           }
  }

  server {
          # Allowed request size should be large enough to allow pull operations
          client_max_body_size 4096M;
          listen 443 ssl;
          server_name example.com;
          ssl_certificate /etc/letsencrypt/live/example.com/fullchain.pem;
          ssl_certificate_key /etc/letsencrypt/live/example.com/privkey.pem;
          location / {
                  proxy_connect_timeout 75s;
                  proxy_pass http://registry-read-only:5000;
          }
  }
}
```

Start the registry

```
docker-compose -f docker-compose.yml up
```

Working with registry

Login to registry

```
docker login example.com -u <username> -p <password>
```

Build and push an image to the registry

Use the commands below to build an image from a `Dockerfile` and push it to your private registry.

```
docker build . -t example.com/my-algo:latest
docker image push example.com/my-algo:latest
```

List images in the registry

```
curl -X GET -u <username>:<password> https://example.com/v2/_catalog
```

Pull an image from the registry

Use the commands below to build an image from a `Dockerfile` and push it to your private registry.

```
# requires login
docker image pull example.com/my-algo:latest

# allows anonymous pull if 2nd setup scenario is implemented
docker image pull readonly.example.com/my-algo:latest
```

Next step

You can publish an algorithm asset with the metadata containing registry URL, image, and tag information to enable users to run C2D jobs.

Further references

- Setup Compute-to-Data environment
- Writing algorithms
- C2D example

User defined parameters

Learn how to define and use custom parameters while downloading assets or using dataset in Compute-to-data environment

Overview

Ocean Protocol allows dataset buyers to provide custom parameters that can be used to fetch the downloaded data in a specific format, download a different type of data or pass some additional input to the algorithms in the Compute-to-Data job.

There 2 are types of parameters that asset publishers can support:

- User defined parameters
- Algorithm custom parameters

Publish a dataset that uses custom parameters

The dataset publisher can support these parameters to allow filtering or querying of the published dataset. The additional parameters that facilitate this are called `User defined parameters`. The Provider combines the original asset URL and the entered parameter values into a new URL and then streams the response from the modified URL back to the buyer.

Use case for user defined parameters

For example, if the publisher has published an URL `https://example.com` which serves large size historical weather data from all over the world, the publisher could allow buyers to filter the data based on location, type of data, etc. It is possible to do this using user defined parameters.

Suppose the publisher defines the following 2 parameters:

- `location`: A string indicating region code
- `type`: A string indicating the type of weather data. It can be temperature/humidity/pressure.

Suppose the buyer wants to download the temperature data in the region code `XYZ`. While downloading the data, the buyer enters the desired parameter values using ocean.py.

The provider will decrypt the URL from the DDO published on-chain, construct a new URL with the additional parameters and finally stream data to the buyer.

Internally, the new URL will be of the format `https://example.com/?location=XYZ&type=temperature`. The server hosting the data has to read these parameters and serve the appropriate data.

The following steps will specify how the publisher can support additional parameters.

Step 1: Create a service

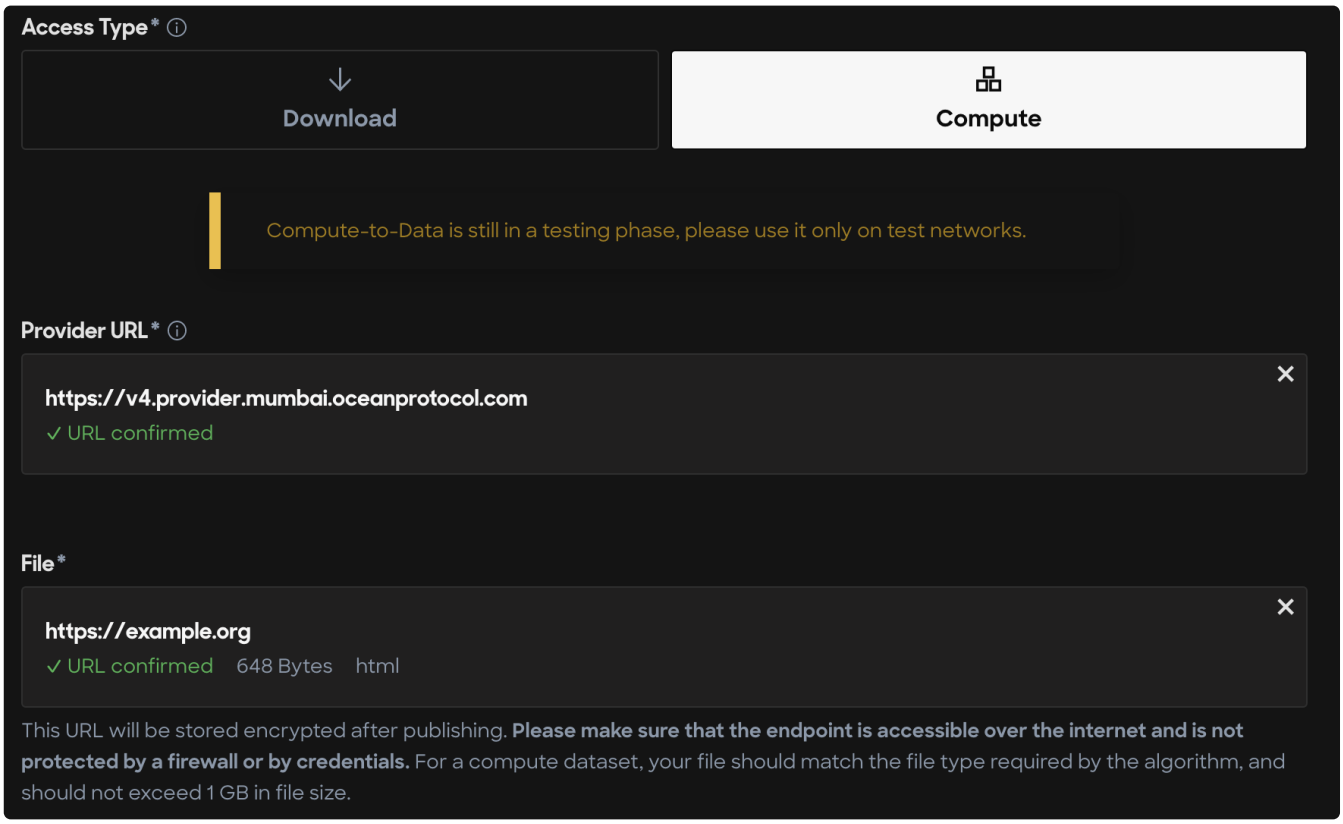
The below python script exposes a REST endpoint that takes two parameters, namely: `location` and `type`. Let's assume that the dataset publisher hosts the service at domain `example.com` along with HTTPS support. The publisher must ensure that the URL is accessible to Provider.

The code snippet is only for demo purposes and not for production use.

```
from flask import Flask, request
def get_data(data_type: str, location: str):
    '''
        Add some business logic here to get
        the required data with given parameters
    '''
    return {}
@app.route('/', methods=['GET'])
def serve_content():
    args = request.args
    data_type = args.get('type')
    location = args.get('location')
    result = get_data(data_type, location)
    return result
```

Step 2: Publish dataset asset with compute service

The publisher now must provide the file URL as `https://example.org` while publishing the asset, as shown in the below image.



For a complete tutorial on publishing asset using Ocean Marketplace read our guide on publishing with Ocean Market.

Publish an algorithm that uses custom parameters

Use case for algorithm custom parameters

For example, if the algorithm publisher has published an URL `https://example.org` which serves python script to analyze historical weather data published in the previous section. If the algorithm publisher wants buyers to specify the number of iterations the algorithm must perform over the data, it is possible to do so using algorithm custom parameters.

Suppose the algorithm publisher defines a parameter called `iterations` and expects the buyer to give this input before running the algorithm in a Compute-to-Data environment. The buyer can enter the desired parameter value using ocean.py or ocean.js.

The provider passes the entered parameters and saves them in a specific path in the Compute-to-Data environment. The algorithm can later read this value and perform required computations.

The following steps will specify how the algorithm publisher can support additional algorithm custom parameters.

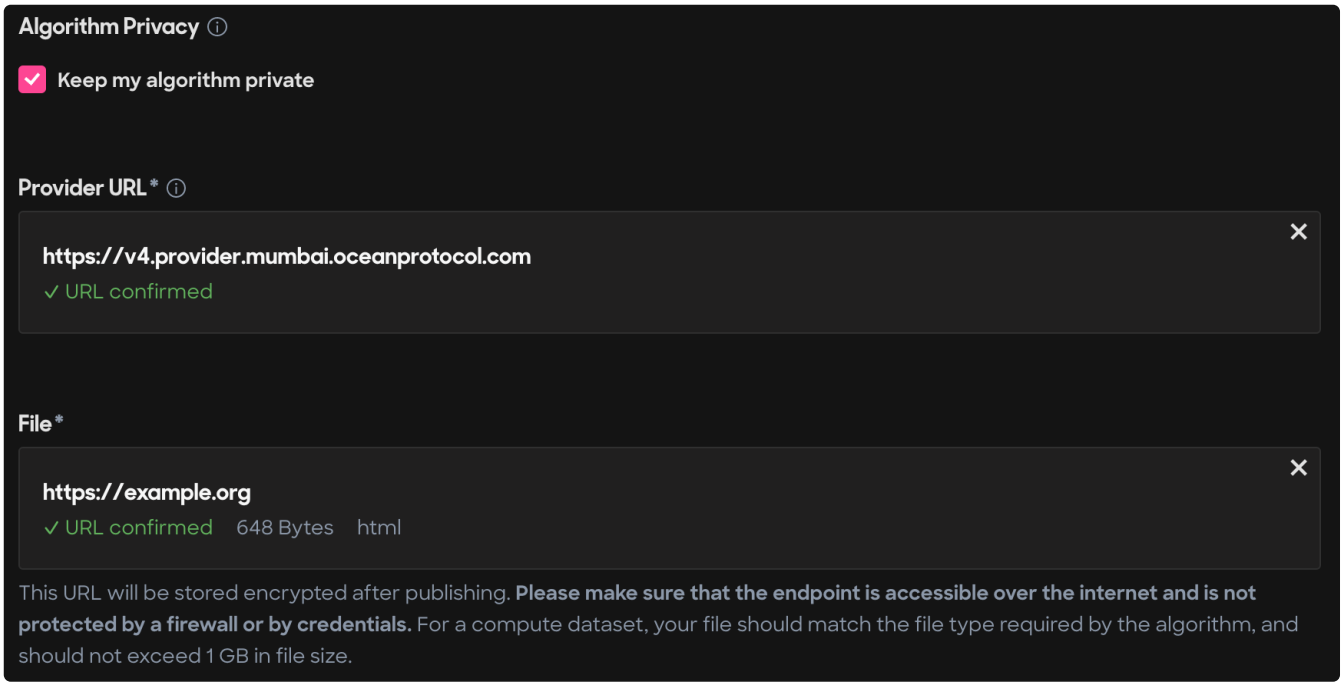
Step 1: Create an algorithm

The code snippet is only for demo purposes and not for production use.

```
def run_algorithm(i: int):
    pass
def read_algorithm_custom_input():
    parameters_file = os.path.join(os.sep, "data", "inputs", "algoCustomData.json")
    with open(parameters_file, "r") as file:
        return json.load(file)
algorithm_inputs = read_algorithm_custom_input()
iterations = algorithm_inputs["iterations"]
for i in range(iterations):
    # Run some machine learning algorithm
    print(f"Running iteration {i}")
    result = run_algorithm(i)
output_dir = os.path.join(os.sep, "data", "outputs")
with open(os.path.join(output_dir, "result"), "w") as f:
    f.write(result)
```

Step 2: Publish algorithm asset

The publisher now must provide the file URL as `https://example.org` while publishing the algorithm asset, as shown in the below image.



For a complete tutorial on publishing asset using Ocean Marketplace read our guide on publishing with Ocean Market.

Starting compute job with custom parameters

In this example, the buyer wants to run the algorithm with certain parameters on a selected dataset. The code snippet below shows how the buyer can start the compute job with custom parameter values. Before embarking on this tutorial you should familiarize yourself with how to:

- Search for a dataset using Ocean market or Aquarius API
- Allow an algorithm to run on the dataset
- Buy datatokens using Ocean market or ocean.py
- Set up ocean.py

For configuring ocean.py/ocean.js, please refer this guide. Copy the below code snippet to a file locally after completing required configurations and execute the script.

Python

start_compute.py

```
# Import dependencies
from config import web3_wallet, ocean, config, web3_wallet
from ocean_lib.models.compute_input import ComputeInput
# Replace theses variables with the appropriate did values
dataset_did = "did:op:<>"
algorithm_did = "did:op:<>"

# Define algorithm input
algorithm_input = {
        "iterations": 1000
}

# Define dataset parameters
dataset_input = {
    "type": "temperature",
    "location": "XYZ"
}

# Resolve assets using Aquarius
aquarius = Aquarius.get_instance(config.metadata_cache_uri)
DATA_asset = aquarius.wait_for_asset(dataset_did)
ALGO_asset = aquarius.wait_for_asset(algorithm_did)

compute_service = DATA_asset.services[0]
algo_service = ALGO_asset.services[0]
free_c2d_env = ocean.compute.get_free_c2d_environment(compute_service.service_endpo

DATA_compute_input = ComputeInput(DATA_asset, compute_service, userdata=dataset_inp
ALGO_compute_input = ComputeInput(ALGO_asset, algo_service)

# Pay for the compute job
datasets, algorithm = ocean.assets.pay_for_compute_service(
    datasets=[DATA_compute_input],
    algorithm_data=ALGO_compute_input,
    consume_market_order_fee_address=web3_wallet.address,
    wallet=web3_wallet,
    compute_environment=free_c2d_env["id"],
    valid_until=int((datetime.utcnow() + timedelta(days=1)).timestamp()),
    consumer_address=free_c2d_env["consumerAddress"],
)

assert datasets, "pay for dataset unsuccessful"
assert algorithm, "pay for algorithm unsuccessful"

# Start compute job
job_id = ocean.compute.start(
```

```
    consumer_wallet=web3_wallet,
    dataset=datasets[0],
    compute_environment=free_c2d_env["id"],
    algorithm=algorithm,
    algorithm_algocustomdata=algorithm_input
)
# Printing the job id here. Use this job_id to retrive the result of the compute jo
print("job_id", job_id)
```

Execute script

```
python start_compute.py
```

Deploying Components

Setup a Server

The following tutorial shows how to create a server ready for hosting Ocean Protocol's components.

Using hosting services

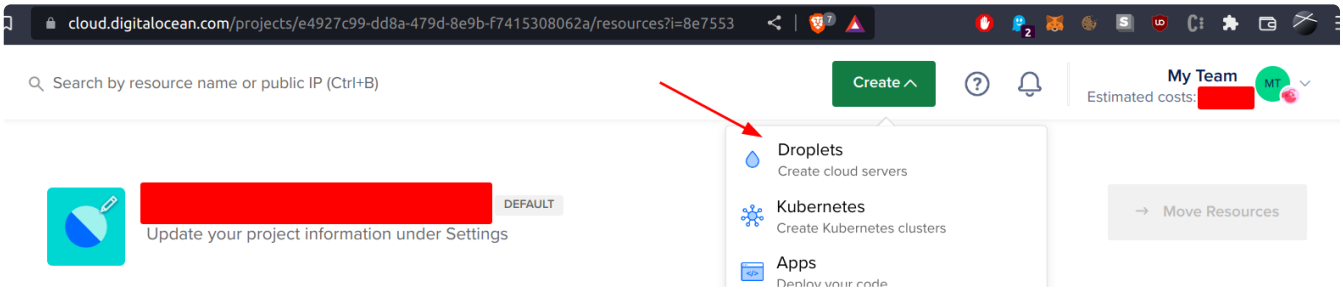
Ocean Protocol's components can be hosted on any infrastructure providers like AWS, Azure, Heroku, Digitalocean, and many others. The tutorial here explains how to create a server using Digitalocean and installing docker which will be required to host Ocean Protocol's components. Apart from steps for create a server, the remaining part of the tutorial will be same for all hosting providers.

Creating account and setting billing

Go to https://www.digitalocean.com/ and create an account. Provide the appropriate information for billing and accounting.

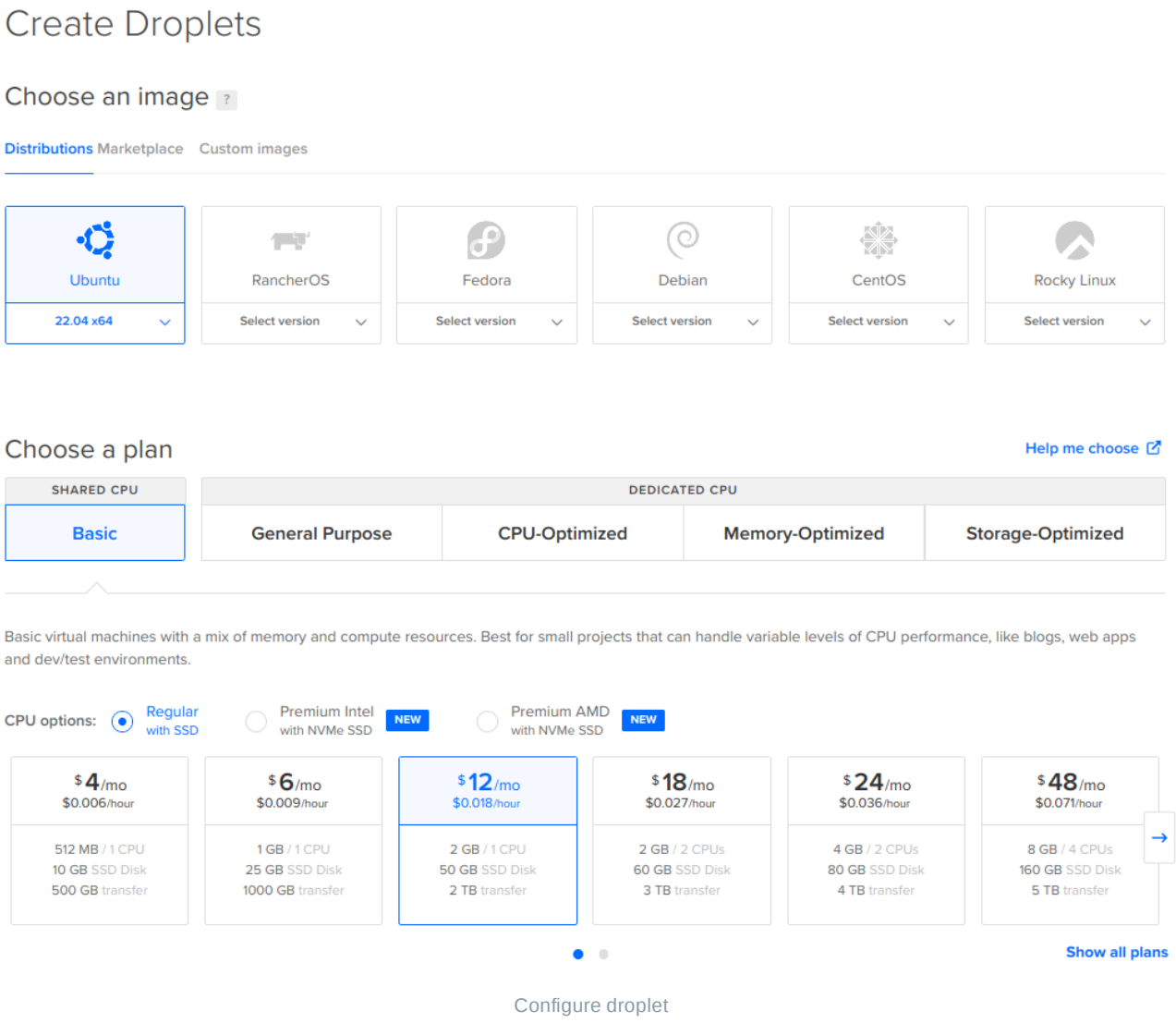
Create a droplet

Click on `Create` button and choose `Droplets` options from dropdown.

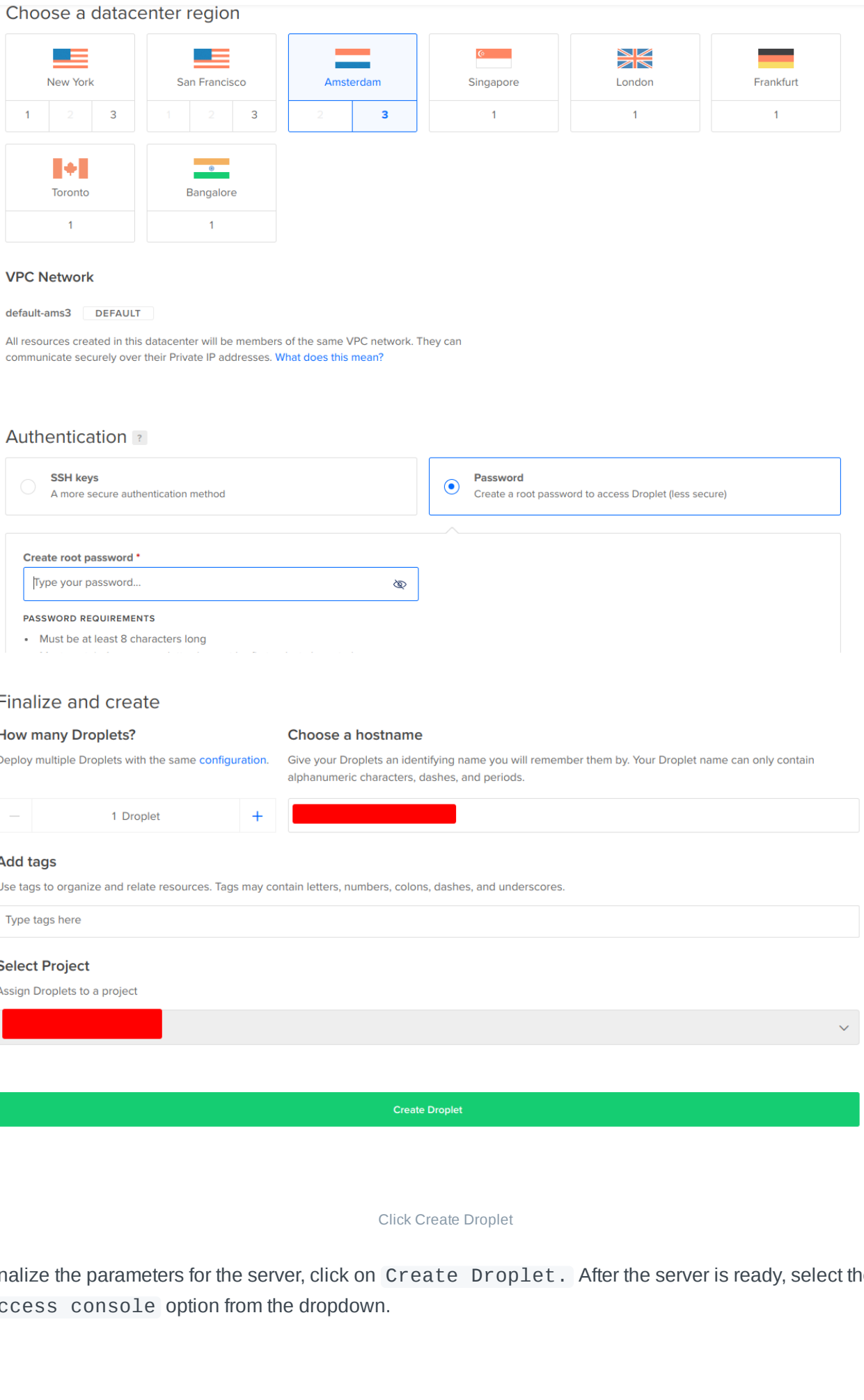


Configure droplet

Select Ubuntu OS and choose a plan. The required CPU, Memory depends on the number of requests Aquarius is expected to serve.

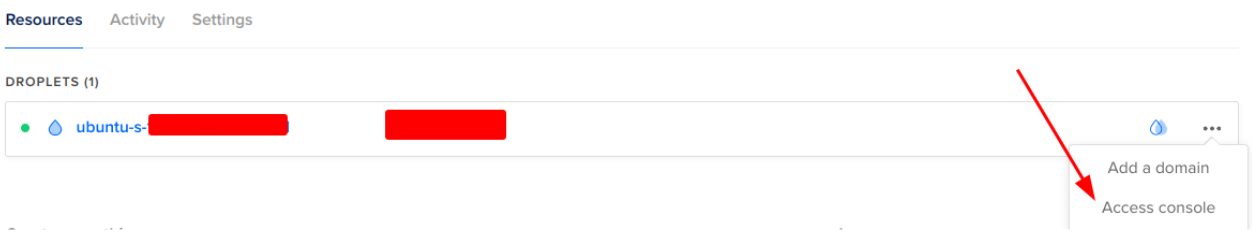


Also, select the region where you want Aquarius to be hosted and a root password.

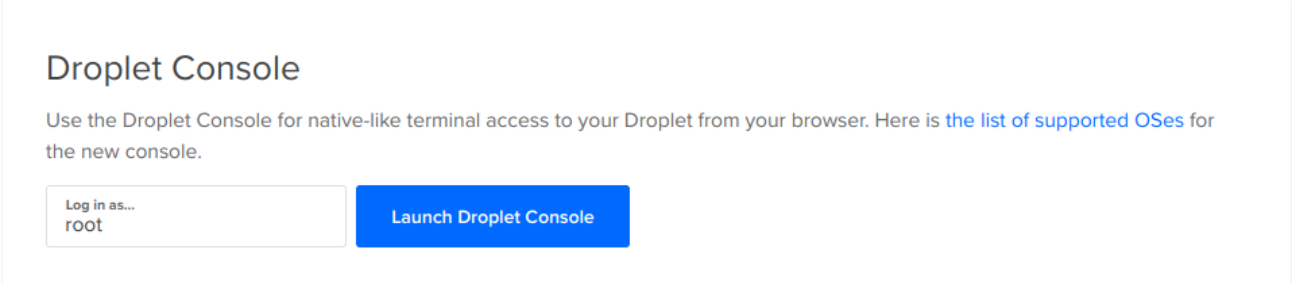


Click Create Droplet

Finalize the parameters for the server, click on `Create Droplet.` After the server is ready, select the `Access console` option from the dropdown.



Click Access Console



Click Launch Droplet Console

A window will open with a terminal session. Now, the required infrastructure is ready for hosting Aquarius, Provider or the Subgraph. Let's install docker and docker-compose on the server. Follow the installation guide here.

The below commands shows the commands executed by following the guide.

```
sudo apt-get update
sudo apt-get install ca-certificates curl gnupg lsb-release
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyri
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://dowr
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin

# Now install docker-compose
sudo apt-get update
sudo apt-get install docker-compose-plugin
```

Now that, the server is ready with all the required dependencies are installed for hosting Ocean Components, follow the instructions given in Component specific guide.

- Deploying Marketplace
- Deploying Aquarius

Deploying Marketplace

Prerequisites

- A server for hosting Ocean Marketplace. See this guide on creating a server.

Create a directory

```
mkdir my-marketplace
cd my-marketplace
```

Create file with name `.env`

Copy the below content into the `.env` file.

.env

```
# Update this value if Market should using custom Aquarius
NEXT_PUBLIC_METADATACACHE_URI=https://v4.aquarius.oceanprotocol.com

#NEXT_PUBLIC_INFURA_PROJECT_ID="xxx"
#NEXT_PUBLIC_MARKET_FEE_ADDRESS="0xxx"
#NEXT_PUBLIC_PUBLISHER_MARKET_ORDER_FEE="1"
#NEXT_PUBLIC_CONSUME_MARKET_ORDER_FEE="1"
#NEXT_PUBLIC_CONSUME_MARKET_FIXED_SWAP_FEE="1"

#
# ADVANCED SETTINGS
#

# Toggle pricing options presented during price creation
#NEXT_PUBLIC_ALLOW_FIXED_PRICING="true"
#NEXT_PUBLIC_ALLOW_FREE_PRICING="true"

# Privacy Preference Center
#NEXT_PUBLIC_PRIVACY_PREFERENCE_CENTER="true"
```

Create a `Dockerfile` file and copy the below content into it.

Dockerfile

```
FROM node:16
RUN git clone https://github.com/oceanprotocol/market.git /usr/app/market
WORKDIR /usr/app/market
RUN npm ci --legacy-peer-deps
RUN npm run build

EXPOSE 3000
CMD ["npx", "next", "start"]
```

Build a docker image

```
docker build . -f Dockerfile -t market:latest
```

Start the marketplace

Deploying Aquarius

About Aquarius

Aquarius is an off-chain component with caches the asset metadata published on-chain. By deploying own Aquarius, developers can control which assets are visible in their marketplace. For example, having a custom Aquarius instance allows assets only from specific addresses to be visible on the marketplace. This tutorial will provide the steps to deploy Aquarius. Ocean Protocol provides Aquarius docker images which can be viewed here. Visit this page to view Aquarius source code.

Aquarius consists of two parts:
- **API:** The Aquarius API offers a convenient way to access the medatata without scanning the chain yourself.
- **Event monitor:** Aquarius continually monitors the chains for MetadataCreated and MetadataUpdated events, processes these events and adds them to the database.

Prerequisites

- A server for hosting Aquarius. See this guide on creating a server.
- Docker and Docker compose are installed. Click here to view guide on installing docker.
- Obtain an API key

Create a working directory

```
mkdir Aquarius
cd Aquarius
```

Create a `.env` file

Copy the below content into the `.env` file and edit the values as needed.

.env

```
# check the available versions: https://hub.docker.com/repository/docker/oceanprotocol/aquariu
AQUARIUS_VERSION=latest
ALLOWED_PUBLISHERS='[""]'
# Elastic search credentials
DB_USERNAME=username
DB_PASSWORD=password

# Replace below value with the API provider of your choice
EVENTS_RPC_POLYGON=<polygon-key>
EVENTS_RPC_MAINNET=<mainnet-key>
```

Create docker-compose file

docker-compose.yml

```
version: '3'
services:
   elasticsearch:
    image: elasticsearch:6.8.17
    container_name: elasticsearch
    restart: on-failure
    environment:
      ES_JAVA_OPTS: "-Xms512m -Xmx512m"
      MAX_MAP_COUNT: "64000"
      discovery.type: "single-node"
    volumes:
      - data:/usr/share/elasticsearch/data
    ports:
      - 9200:9200
    networks:
      - ocean_backend
   aquarius:
    image: oceanprotocol/aquarius:${AQUARIUS_VERSION}
    container_name: aquarius
    restart: on-failure
    ports:
      - 5000:5000
    networks:
      - ocean_backend
    depends_on:
      - elasticsearch
    environment:
      DB_MODULE: elasticsearch
      DB_HOSTNAME: elasticsearch
      DB_PORT: 9200
      DB_USERNAME: ${DB_USERNAME}
      DB_PASSWORD: ${DB_PASSWORD}
      DB_NAME: aquarius
      DB_SCHEME: http
      DB_SSL : "false"
      LOG_LEVEL: "DEBUG"
      AQUARIUS_BIND_URL : "http://0.0.0.0:5000"
      AQUARIUS_WORKERS : "8"
      RUN_AQUARIUS_SERVER: "1"
      AQUARIUS_CONFIG_FILE: "config.ini"
      EVENTS_ALLOW: 0
      RUN_EVENTS_MONITOR: 0
      ALLOWED_PUBLISHERS: ${ALLOWED_PUBLISHERS}
volumes:
  data:
    driver: local
networks:
  ocean_backend:
    driver: bridge
```

Create events monitor docker compose file

Events monitor - Mainnet

docker-compose-events-mainnet.yml

```
version: '3'
services:
  aquarius-events-mainnet:
    image: oceanprotocol/aquarius:${AQUARIUS_VERSION}
    container_name: aquarius-events-mainnet
    restart: on-failure
    networks:
      - ocean_backend
    depends_on:
      - elasticsearch
    environment:
      DB_MODULE: elasticsearch
      DB_HOSTNAME: elasticsearch
      DB_PORT: 9200
      DB_USERNAME: ${DB_USERNAME}
      DB_PASSWORD: ${DB_PASSWORD}
      DB_NAME: aquarius
      DB_SCHEME: http
      DB_SSL : "false"
      LOG_LEVEL: "DEBUG"
      AQUARIUS_BIND_URL: "http://0.0.0.0:5000"
      AQUARIUS_WORKERS : "1"
      RUN_AQUARIUS_SERVER : "0"
      AQUARIUS_CONFIG_FILE: "config.ini"
      NETWORK_NAME: "mainnet"
      EVENTS_RPC: ${EVENTS_RPC_MAINNET}
      METADATA_UPDATE_ALL : "0"
      OCEAN_ADDRESS :  "0x967da4048cD07aB37855c090aAF366e4ce1b9F48"
      EVENTS_ALLOW: 0
      RUN_EVENTS_MONITOR: 1
      BLOCKS_CHUNK_SIZE: "5000"
volumes:
  data:
    driver: local
networks:
  ocean_backend:
    driver: bridge
```

Events monitor - Polygon

docker-compose-events-ploygon.yml

```
version: '3'
services:
```

```
  aquarius-events-polygon:
    image: oceanprotocol/aquarius:${AQUARIUS_VERSION}
    container_name: aquarius-events-polygon
    restart: on-failure
    networks:
      - ocean_backend
    depends_on:
      - elasticsearch
    environment:
      DB_MODULE: elasticsearch
      DB_HOSTNAME: elasticsearch
      DB_PORT: 9200
      DB_USERNAME: ${DB_USERNAME}
      DB_PASSWORD: ${DB_PASSWORD}
      DB_NAME: aquarius
      DB_SCHEME: http
      DB_SSL : "false"
      LOG_LEVEL: "DEBUG"
      AQUARIUS_BIND_URL: "http://0.0.0.0:5000"
      AQUARIUS_WORKERS : "1"
      RUN_AQUARIUS_SERVER : "0"
      AQUARIUS_CONFIG_FILE: "config.ini"
      NETWORK_NAME: "polygon"
      EVENTS_RPC: ${EVENTS_RPC_POLYGON}
      METADATA_UPDATE_ALL: "0"
      OCEAN_ADDRESS: "0x282d8efCe846A88B159800bd4130ad77443Fa1A1"
      EVENTS_ALLOW: 0
      RUN_EVENTS_MONITOR: 1
      METADATA_CONTRACT_ADDRESS: "0x80E63f73cAc60c1662f27D2DFd2EA834acddBaa8"
      BLOCKS_CHUNK_SIZE: "5000"
volumes:
  data:
    driver: local
networks:
  ocean_backend:
    driver: bridge
```

Start Aquarius

```
docker-compose \
-f docker-compose.yml \
-f docker-compose-events-mainnet.yml \
-f docker-compose-events-polygon.yml \
--env-file .env \
-d \
up
```

After pulling all the asset metadata from the blockchain, Aquarius can be used to query the assets using Elasticsearch query. Aquarius REST API are documented here.

Deploying Provider

About Provider

Provider encrypts the URL and metadata during publish and decrypts the URL when the dataset is downloaded or a compute job is started. It enables the access to data assets by streaming data (and never the URL). It performs checks on chain for buyer permissions and payments. It also Provides compute services (connects to C2D environment). The source code of Provider can be access from here.

Prerequisites

- Docker and Docker compose are installed. Click here to view guide on installing docker.
- Obtain an API key

Create a working directory

```
mkdir Provider
cd Provider
```

Create a `.env` file

Copy the below content into the `.env` file and edit the values as needed.

.env

```
# Mandatory variables

# Update the value to the appropriate tag from here: https://hub.docker.com/r/oceanprotocol/pr
PROVIDER_VERSION=latest
PROVIDER_PRIVATE_KEY=<private-key>
NETWORK_URL=<api-key>
AQUARIUS_URL=<aquarius-url>
```

Create docker-compose file

> Set the value of OCEAN_PROVIDER_WORKERS to 2 or more to avoid a race condition when provider checks whether it should call a remote provider or not.

docker-compose.provider.yml

```
version: '3'
services:
  provider:
    image: oceanprotocol/provider-py:v1.0.20
    container_name: provider
    ports:
      - 8030:8030
    networks:
      - ocean_backend
    environment:
      NETWORK_URL: ${NETWORK_URL}
      PROVIDER_PRIVATE_KEY: ${PROVIDER_PRIVATE_KEY}
      LOG_LEVEL: DEBUG
      OCEAN_PROVIDER_URL: "http://0.0.0.0:8030"
      OCEAN_PROVIDER_WORKERS: "2"
      OCEAN_PROVIDER_TIMEOUT: "9000"
      # Defining OPERATOR_SERVICE_URL is optional. Set the value only if Provider should suppo
      OPERATOR_SERVICE_URL: "<operator-service-url>"
      # Defining IPFS_GATEWAY is optional. Set the value if Provider should support resolving
      IPFS_GATEWAY: "<ipfs-url>"
      AQUARIUS_URL: ${AQUARIUS_URL}
volumes:
  data:
    driver: local
networks:
  ocean_backend:
    driver: bridge
```

Start Provider

```
docker-compose \
-f docker-compose.provider.yml
--env-file .env \
-d \
up
```

Deploying Ocean Subgraph

About Ocean subgraph

Ocean subgraph allows querying the datatoken, dataNFT, and all event information using GraphQL. Hosting the Ocean subgraph saves the cost and time required in querying the data directly from the blockchain. The steps in this tutorial will explain how to host Ocean subgraph for the EVM compatible chains supported by Ocean Protocol.

Prerequisites

- A server for hosting Ocean subgraph. See this guide on creating a server.
- Docker and Docker compose are installed. Click here to view guide on installing docker.
- Obtain an API key

Create a working directory

```
mkdir ocean-subgraph
cd ocean-subgraph
```

Create a `.env` file

Copy the below content into the `.env` file and edit the values as needed.

.env

```
ETHEREUM_NODE_PROVIDER_API='mumbai:https://polygon-mumbai.infura.io/v3/${INFURA_PROJECT_ID}'
```

Create docker-compose file

docker-compose.yml

```
version: '3'
services:
  graph-node:
    image: graphprotocol/graph-node:v0.26.0
    ports:
      - '9000:8000'
      - '8001:8001'
      - '8020:8020'
      - '8030:8030'
      - '8040:8040'
    depends_on:
      - ipfs

      - postgres
    environment:
      postgres_host: postgres
      postgres_user: graph-node
      postgres_pass: let-me-in
      postgres_db: graph-node
      ipfs: 'ipfs:5001'
      ethereum: ${ETHEREUM_NODE_PROVIDER_API}
      RUST_LOG: info
  ipfs:
    image: ipfs/go-ipfs:v0.4.23
    ports:
      - '5001:5001'
    volumes:
      - ./data/ipfs:/data/ipfs
  postgres:
    image: postgres
    ports:
      - '5432:5432'
    command: ['postgres', '-cshared_preload_libraries=pg_stat_statements']
    environment:
      POSTGRES_USER: graph-node
      POSTGRES_PASSWORD: let-me-in
      POSTGRES_DB: graph-node
    volumes:
      - ./data/postgres:/var/lib/postgresql/data
```

Start Ocean subgraph

```
docker-compose \
-f docker-compose.yml
--env-file .env \
-d \
up
```

Using Ocean Subgraph

Ocean subgraph is an off-chain service that provides information about datatokens, users, and balances using GraphQL. The Ocean subgraph provides a faster way to fetch data rather than an on-chain query. The data from Ocean subgraph can be queried using GraphQL.

You can use the Subgraph instances hosted by Ocean Protocol or host your instance. The page on Deploying Ocean Subgraph provides a guide on hosting your own Ocean Subgraph instance.

For each supported network, an individual Ocean subgraph is deployed. The information about supported networks are available on this page.

Ocean Subgraph GraphiQL

The below table provides the link to GraphiQL for the support networks. The Graphql queries can be directly executed using the GraphiQL interface without the need of any setup.

Network and link
Ethereum Mainnet
Polygon Mainnet
Binance Smart Chain
Energy Web Chain
Moonriver
Mumbai
Görli

List data NFTs

The result of following GraphQL query returns the information about data nfts.

> Copy the query in the GraphiQL interface to fetch the results from the mainnet. For other networks use this table.

Query

```
{
  nfts (skip:0, first: 10, subgraphError:deny){
    id
    name
    symbol
    owner
    address
    assetState
    tx
    block
    transferable
 }
}
```

Code snippets

Python

The python script below can be used to run the the query. If you wish to change the network, then replace the value of variable `base_url` as needed.

Create script

list_dataNFTs.py

```
import requests
import json


query = """
{
  nfts (skip:0, first: 10, subgraphError:deny){
    id
    name
    symbol
    owner
    address
    assetState
    tx
    block
    transferable
 }
}"""


base_url = "https://v4.subgraph.mainnet.oceanprotocol.com"
route = "/subgraphs/name/oceanprotocol/ocean-subgraph"

url = base_url + route

headers = {"Content-Type": "application/json"}
payload = json.dumps({"query": query})
response = requests.request("POST", url, headers=headers, data=payload)
result = json.loads(response.text)

print(json.dumps(result, indent=4, sort_keys=True))
```

Execute script

```
python list_dataNFTs.py
```

Javascript

The javascript below can be used to run the the query. If you wish to change the network, then replace the value of variable `baseUrl` as needed.

Create script

listDatatoken.js

```
var axios = require('axios');

const query = `{
    nfts (skip:0, first: 10, subgraphError:deny){
      id
      name
      symbol
      owner
      address
      assetState
      tx
      block
      transferable
   }
}`

const baseUrl = "https://v4.subgraph.mainnet.oceanprotocol.com"
const route = "/subgraphs/name/oceanprotocol/ocean-subgraph"


const url = `${baseUrl}${route}`

var config = {
    method: 'post',
    url: url,
    headers: { "Content-Type": "application/json" },
    data: JSON.stringify({ "query": query })
};

axios(config)
    .then(function (response) {
        console.log(JSON.stringify(response.data));
    })
    .catch(function (error) {
        console.log(error);
    });
```

Execute script

```
node listDatatoken.js
```

Response

Sample response


```
{

  "data": {
    "nfts": [
      {
        "address": "0x1c161d721e6d99f58d47f709cdc77025056c544c",
        "assetState": 0,
        "block": 15185270,
        "id": "0x1c161d721e6d99f58d47f709cdc77025056c544c",
        "name": "Ocean Data NFT",
        "owner": "0xd30dd83132f2227f114db8b90f565bca2832afbd",
        "symbol": "OCEAN-NFT",
        "transferable": true,
        "tx": "0x327a9da0d2e9df945fd2f8e10b1caa77acf98e803c5a2f588597172a0bcbb93a"
      },
      {
        "address": "0x1e06501660623aa973474e3c59efb8ba542cb083",
        "assetState": 0,
        "block": 15185119,
        "id": "0x1e06501660623aa973474e3c59efb8ba542cb083",
        "name": "Ocean Data NFT",
        "owner": "0xd30dd83132f2227f114db8b90f565bca2832afbd",
        "symbol": "OCEAN-NFT",
        "transferable": true,
        "tx": "0xd351ccee22b505d811c29fa524db920815936672b20b8f3a09485e389902fd27"
      },
      {
        "address": "0x2eaa55236f799c6ebec72e77a1a6296ea2e704b1",
        "assetState": 0,
        "block": 15185009,
        "id": "0x2eaa55236f799c6ebec72e77a1a6296ea2e704b1",
        "name": "Ocean Data NFT",
        "owner": "0xd30dd83132f2227f114db8b90f565bca2832afbd",
        "symbol": "OCEAN-NFT",
        "transferable": true,
        "tx": "0xf6d55306ab4dc339dc1655a2d119af468a79a70fa62ea11de78879da61e89e7b"
      },
      {
        "address": "0x2fbe924f6c92825929dc7785fe05d15e35f2612b",
        "assetState": 0,
        "block": 15185235,
        "id": "0x2fbe924f6c92825929dc7785fe05d15e35f2612b",
        "name": "Ocean Data NFT",
        "owner": "0xd30dd83132f2227f114db8b90f565bca2832afbd",
        "symbol": "OCEAN-NFT",
        "transferable": true,
        "tx": "0xa9ff9d461b4b7344ea181de32fa6412c7ea8e21f485ab4d8a7b9cfcdb68d9d51"
      },
      {
```

```
    "address": "0x4c04433bb1760a66be7713884bb6370e9c567cef",
    "assetState": 0,
    "block": 15185169,
    "id": "0x4c04433bb1760a66be7713884bb6370e9c567cef",
    "name": "Ocean Data NFT",
    "owner": "0xd30dd83132f2227f114db8b90f565bca2832afbd",

    "symbol": "OCEAN-NFT",
    "transferable": true,
    "tx": "0x54c5463e8988b5fa4e4cfe71ee391505801931abe9e94bf1588dd538ec3aa4c9"
  },
  {
    "address": "0x619c500dcb0251b31cd480030db2dcc19866c0c3",
    "assetState": 0,
    "block": 15236619,
    "id": "0x619c500dcb0251b31cd480030db2dcc19866c0c3",
    "name": "abc",
    "owner": "0x12fe650c86cd4346933ef1bcab21a1979d4c6786",
    "symbol": "GOAL-9956",
    "transferable": true,
    "tx": "0x6178b03589cda98573ff52a1afbcc07b14a2fddacc0132595949e9d8a0ed1b32"
  },
  {
    "address": "0x6d45a5b38a122a6dbc042601236d6ecc5c8e343e",
    "assetState": 0,
    "block": 15109853,
    "id": "0x6d45a5b38a122a6dbc042601236d6ecc5c8e343e",
    "name": "Ocean Data NFT",
    "owner": "0xbbd33afa85539fa65cc08a2e61a001876d2f13fe",
    "symbol": "OCEAN-NFT",
    "transferable": true,
    "tx": "0x27aa77a0bf3f7878910dc7bfe2116d9271138c222b3d898381a5c72eefefe624"
  },
  {
    "address": "0x7400078c5d4fd7704afca45a928d9fc97cbea744",
    "assetState": 0,
    "block": 15185056,
    "id": "0x7400078c5d4fd7704afca45a928d9fc97cbea744",
    "name": "Ocean Data NFT",
    "owner": "0xd30dd83132f2227f114db8b90f565bca2832afbd",
    "symbol": "OCEAN-NFT",
    "transferable": true,
    "tx": "0x2025374cd347e25e2651feec2f2faa2feb26664698eaea42b5dad1a31eda57f8"
  },
  {
    "address": "0x81decdb59dce5b4323e683a76f8fa8dd0eabc560",
    "assetState": 0,
    "block": 15185003,
    "id": "0x81decdb59dce5b4323e683a76f8fa8dd0eabc560",
    "name": "Ocean Data NFT",
    "owner": "0xd30dd83132f2227f114db8b90f565bca2832afbd",
    "symbol": "OCEAN-NFT",
    "transferable": true,
```

```
      "tx": "0x6ad6ec2ce86bb70e077590a64c886d72975374bd2e993f143d9da8edcaace82b"
    },
    {
      "address": "0x8684119ecf77c5be41f01760ad466725ffd9b960",
      "assetState": 0,
      "block": 14933034,
      "id": "0x8684119ecf77c5be41f01760ad466725ffd9b960",
      "name": "Ocean Data NFT",
      "owner": "0x87b5606fba13529e1812319d25c6c2cd5c3f3cbc",
      "symbol": "OCEAN-NFT",
      "transferable": true,
      "tx": "0x55ba746cd8e8fb4c739b8544a9034848082b627500b854cb8db0802dd7beb172"
    }
  ]
 }
}
```

List all Tokens

The result of following GraphQL query returns the information about datatokens.

> Copy the query in the GraphiQL interface to fetch the results from the mainnet. For other networks use this table.

Query

```
{
  tokens(skip:0, first: 2, subgraphError: deny){
    id
    symbol
    nft {
      name
      symbol
      address
    }
    name
    symbol
    cap
    isDatatoken
    holderCount
    orderCount
    orders(skip:0,first:1){
      amount
      serviceIndex
      payer {
        id
      }
      consumer{
        id
      }
      estimatedUSDValue
      lastPriceToken
      lastPriceValue
    }
  }
}
```

Code

Python

The python script below can be used to run the the query. If you wish to change the network, then replace the value of variable `base_url` as needed.

Create script

list_all_tokens.py

```
import requests
import json

query = """
{{
        tokens(skip:0, first: 2, subgraphError: deny){{
    id
    symbol
    nft {{
      name
      symbol
      address
    }}
    name
    symbol
    cap
    isDatatoken
    holderCount
    orderCount
    orders(skip:0,first:1){{
      amount
      serviceIndex
      payer {{
        id
      }}
      consumer{{
        id
      }}
      estimatedUSDValue
      lastPriceToken
      lastPriceValue
    }}


  }}
}}"""

base_url = "https://v4.subgraph.mainnet.oceanprotocol.com"
route = "/subgraphs/name/oceanprotocol/ocean-subgraph"

url = base_url + route
```

```
headers = {"Content-Type": "application/json"}
payload = json.dumps({"query": query})
response = requests.request("POST", url, headers=headers, data=payload)
result = json.loads(response.text)

print(json.dumps(result, indent=4, sort_keys=True))
```

Execute script

```
python list_all_tokens.py
```

Javascript

The javascript below can be used to run the the query. If you wish to change the network, then replace the value of variable `baseUrl` as needed.

Create script

listAllTokens.js

```
var axios = require('axios');

const query = `{
    tokens(skip:0, first: 2, subgraphError: deny){
      id
      symbol
      nft {
        name
        symbol
        address
      }
      name
      symbol
      cap
      isDatatoken
      holderCount
      orderCount
      orders(skip:0,first:1){
        amount
        serviceIndex
        payer {
          id
        }
        consumer{
          id
        }
        estimatedUSDValue
        lastPriceToken
        lastPriceValue
      }
    }
}`
```

```
const baseUrl = "https://v4.subgraph.mainnet.oceanprotocol.com"
const route = "/subgraphs/name/oceanprotocol/ocean-subgraph"

const url = `${baseUrl}${route}`

var config = {
    method: 'post',
    url: url,
    headers: { "Content-Type": "application/json" },
    data: JSON.stringify({ "query": query })
};

axios(config)
    .then(function (response) {
        console.log(JSON.stringify(response.data));
    })
    .catch(function (error) {
        console.log(error);
    });
```

Execute script

```
node listAllTokens.js
```

Response

Sample Response


```
{
  "data": {
    "tokens": [
      {
        "cap": null,
        "holderCount": "0",
        "id": "0x0642026e7f0b6ccac5925b4e7fa61384250e1701",
        "isDatatoken": false,
        "name": "H2O",
        "nft": null,
        "orderCount": "0",
        "orders": [],
        "symbol": "H2O"
      },
      {
        "cap": "115792089237316195423570985008687900000000000000000000000000",
        "holderCount": "0",
        "id": "0x122d10d543bc600967b4db0f45f80cb1ddee43eb",
        "isDatatoken": true,
        "name": "Brave Lobster Token",
        "nft": {

          "address": "0xea615374949a2405c3ee555053eca4d74ec4c2f0",
          "name": "Ocean Data NFT",
          "symbol": "OCEAN-NFT"
        },
        "orderCount": "0",
        "orders": [],
        "symbol": "BRALOB-11"
      }
    ]
  }
}
```

Get Data NFT Information

The result of following GraphQL query returns the information about a particular datatoken. Here, `0x1c161d721e6d99f58d47f709cdc77025056c544c` is the address of the dataNFT.

> ⓘ Copy the query in the GraphiQL interface to fetch the results from the mainnet. For other networks use this table.

Query

```
{
  nft (id:"0x1c161d721e6d99f58d47f709cdc77025056c544c", subgraphError:deny){
    id
    name
    symbol
    owner
    address
    assetState
    tx
    block
    transferable
    creator
    createdTimestamp
    providerUrl
    managerRole
    erc20DeployerRole
    storeUpdateRole
    metadataRole
    tokenUri
    template
    orderCount
 }
}
```

Code

Python

The python script below can be used to run the the query. If you wish to change the network, then replace the value of variable `base_url` as needed. Change the value of the variable dataNFT_address with the address of the datatoken of your choice.

Create script

dataNFT_information.py

```
import requests
import json

dataNFT_address = "0x1c161d721e6d99f58d47f709cdc77025056c544c"
query = """
{{
  nft (id:"{0}", subgraphError:deny){{
    id
    name
    symbol
    owner
    address
    assetState
    tx
    block
    transferable
    creator
    createdTimestamp
    providerUrl
    managerRole
    erc20DeployerRole
    storeUpdateRole
    metadataRole
    tokenUri
    template
    orderCount
 }}
}}""".format(
    dataNFT_address
)

base_url = "https://v4.subgraph.mainnet.oceanprotocol.com"
route = "/subgraphs/name/oceanprotocol/ocean-subgraph"

url = base_url + route

headers = {"Content-Type": "application/json"}
payload = json.dumps({"query": query})
response = requests.request("POST", url, headers=headers, data=payload)
result = json.loads(response.text)
```

```
print(json.dumps(result, indent=4, sort_keys=True))
```

Execute script

```
python dataNFT_information.py
```

Javascript

The javascript below can be used to run the the query. If you wish to change the network, then replace the value of variable `baseUrl` as needed. Change the value of the variable `datanftAddress` with the address of the datatoken of your choice.

Create script

dataNFTInfo.js

```
var axios = require('axios');

const datanftAddress = "0x1c161d721e6d99f58d47f709cdc77025056c544c";

const query = `{
  nft (id:"${datanftAddress}", subgraphError:deny){
    id
    name
    symbol
    owner
    address
    assetState
    tx
    block
    transferable
    creator
    createdTimestamp
    providerUrl
    managerRole
    erc20DeployerRole
    storeUpdateRole
    metadataRole
    tokenUri
    template
    orderCount
 }
}`

const baseUrl = "https://v4.subgraph.mainnet.oceanprotocol.com"
const route = "/subgraphs/name/oceanprotocol/ocean-subgraph"

const url = `${baseUrl}${route}`

var config = {
  method: 'post',
```

```
  url: url,
  headers:’{ "Content-Type": "application/json" },
  data: JSON.stringify({ "query": query })
};

axios(config)
  .then(function (response) {
    console.log(JSON.stringify(response.data));
  })
  .catch(function (error) {
    console.log(error);
  });
```

Execute script

```
node dataNFTInfo.js
```

Response

Sample response


```
{
  "data": {
    "nft": {
      "address": "0x1c161d721e6d99f58d47f709cdc77025056c544c",
      "assetState": 0,
      "block": 15185270,
      "createdTimestamp": 1658397870,
      "creator": "0xd30dd83132f2227f114db8b90f565bca2832afbd",
      "erc20DeployerRole": [
        "0x1706df1f2d93558d1d77bed49ccdb8b88fafc306"
      ],
      "id": "0x1c161d721e6d99f58d47f709cdc77025056c544c",
      "managerRole": [
        "0xd30dd83132f2227f114db8b90f565bca2832afbd"
      ],
      "metadataRole": null,
      "name": "Ocean Data NFT",
      "orderCount": "1",
      "owner": "0xd30dd83132f2227f114db8b90f565bca2832afbd",
      "providerUrl": "https://v4.provider.mainnet.oceanprotocol.com",
      "storeUpdateRole": null,
      "symbol": "OCEAN-NFT",
      "template": "",
      "tokenUri": "<removed>",
      "transferable": true,
      "tx": "0x327a9da0d2e9df945fd2f8e10b1caa77acf98e803c5a2f588597172a0bcbb93a"
    }
  }
}
```

Get Datatoken Information

The result of following GraphQL query returns the information about a particular datatoken. Here, `0x122d10d543bc600967b4db0f45f80cb1ddee43eb` is the address of the datatoken.

> ⓘ Copy the query in the GraphiQL interface to fetch the results from the mainnet. For other networks use this table.

Query

```
{
  token(id:"0x122d10d543bc600967b4db0f45f80cb1ddee43eb", subgraphError: deny){
    id
    symbol
    nft {
      name
      symbol
      address
    }
    name
    symbol
    cap
    isDatatoken
    holderCount
    orderCount
    orders(skip:0,first:1){
      amount
      serviceIndex
      payer {
        id
      }
      consumer{
        id
      }
      estimatedUSDValue
      lastPriceToken
      lastPriceValue
    }
  }
  fixedRateExchanges(subgraphError:deny){
    id
    price
    active
  }
}
```

Code

Python

The python script below can be used to run the the query. If you wish to change the network, then replace the value of variable `base_url` as needed. Change the value of the variable `datatoken_address` with the address of the datatoken of your choice.

Create script

datatoken_information.py

```
import requests
import json

datatoken_address = "0x122d10d543bc600967b4db0f45f80cb1ddee43eb"
query = """
{{
  token(id:"{0}", subgraphError: deny){{
    id
    symbol
    nft {{
      name
      symbol
      address
    }}
    name
    symbol
    cap
    isDatatoken
    holderCount
    orderCount
    orders(skip:0,first:1){{
      amount
      serviceIndex
      payer {{
        id
      }}
      consumer{{
        id
      }}
      estimatedUSDValue
      lastPriceToken
      lastPriceValue
    }}
  }}
  fixedRateExchanges(subgraphError:deny){{
    id
    price
    active
  }}
}}""".format(
    datatoken_address
```

```
)

base_url = "https://v4.subgraph.mainnet.oceanprotocol.com/"
route = "/subgraphs/name/oceanprotocol/ocean-subgraph"

url = base_url + route

headers = {"Content-Type": "application/json"}
payload = json.dumps({"query": query})
response = requests.request("POST", url, headers=headers, data=payload)
result = json.loads(response.text)

print(json.dumps(result, indent=4, sort_keys=True))
```

Execute script

```
python datatoken_information.py
```

Javascript

The javascript below can be used to run the the query. If you wish to change the network, then replace the value of variable `baseUrl` as needed. Change the value of the variable `datatokenAddress` with the address of the datatoken of your choice.

Create script

datatokenInfo.js

```
var axios = require('axios');

const datatokenAddress = "0x122d10d543bc600967b4db0f45f80cb1ddee43eb";

const query = `{
  token(id:"${datatokenAddress}", subgraphError: deny){
    id
    symbol
    nft {
      name
      symbol
      address
    }
    name
    symbol
    cap
    isDatatoken
    holderCount
    orderCount
    orders(skip:0,first:1){

      amount
      serviceIndex
      payer {
```

```
      id
    }
    consumer{
      id
    }
    estimatedUSDValue
    lastPriceToken
    lastPriceValue
  }
}
fixedRateExchanges(subgraphError:deny){
  id
  price
  active
}
}`

const baseUrl = "https://v4.subgraph.mainnet.oceanprotocol.com"
const route = "/subgraphs/name/oceanprotocol/ocean-subgraph"

const url = `${baseUrl}${route}`

var config = {
  method: 'post',
  url: url,
  headers: { "Content-Type": "application/json" },
  data: JSON.stringify({ "query": query })
};

axios(config)
  .then(function (response) {
    console.log(JSON.stringify(response.data));
  })
  .catch(function (error) {
    console.log(error);
  });
```

Execute script

```
node datatokenInfo.js
```

Response

Sample response


```
{
  "data": {
    "fixedRateExchanges": [
      {
        "active": true,
        "id": "0xfa48673a7c36a2a768f89ac1ee8c355d5c367b02-0x06284c39b48afe5f01a04d56f
        "price": "33"
      },
      {
        "active": true,
        "id": "0xfa48673a7c36a2a768f89ac1ee8c355d5c367b02-0x2719862ebc4ed253f09088c87
        "price": "35"
      },
      {
        "active": true,
        "id": "0xfa48673a7c36a2a768f89ac1ee8c355d5c367b02-0x2dccaa373e4b65d5ec153c150
        "price": "33"
      }
    ],
    "token": {
      "cap": "115792089237316195423570985008687900000000000000000000000000",
      "holderCount": "0",
      "id": "0x122d10d543bc600967b4db0f45f80cb1ddee43eb",
      "isDatatoken": true,
      "name": "Brave Lobster Token",
      "nft": {
        "address": "0xea615374949a2405c3ee555053eca4d74ec4c2f0",
        "name": "Ocean Data NFT",
        "symbol": "OCEAN-NFT"
      },
      "orderCount": "0",
      "orders": [],
      "symbol": "BRALOB-11"
    }
  }
}
```

List Fixed Rate Exchanges

The result of following GraphQL query returns the information about the Fixed Rate Exchanges.

> ⓘ Copy the query in the GraphiQL interface to fetch the results from the mainnet. For other networks use this table.

Query

```
{
  fixedRateExchanges(skip:0, first:2, subgraphError:deny){
    id
    contract
    exchangeId
    owner{id}
    datatoken{
      id
      name
      symbol
    }
    price
    datatokenBalance
    active
    totalSwapValue
    swaps(skip:0, first:1){
      tx
      by {
        id
      }
      baseTokenAmount
      dataTokenAmount
      createdTimestamp
    }
    updates(skip:0, first:1){
      oldPrice
      newPrice
      newActive
      createdTimestamp
      tx
    }
  }
}
```

Code

Python

The python script below can be used to run the the query. If you wish to change the network, then replace the value of variable `base_url` as needed.

Create script

list_fixed_rate_exchanges.py

```
import requests
import json


query = """
{
  fixedRateExchanges(skip:0, first:2, subgraphError:deny){
    id
    contract
    exchangeId
    owner{id}
    datatoken{
      id
      name
      symbol
    }
    price
    datatokenBalance
    active
    totalSwapValue
    swaps(skip:0, first:1){
      tx
      by {
        id
      }
      baseTokenAmount
      dataTokenAmount
      createdTimestamp
    }
    updates(skip:0, first:1){
      oldPrice
      newPrice
      newActive
      createdTimestamp
      tx
    }
  }
}"""


base_url = "https://v4.subgraph.mainnet.oceanprotocol.com"
route = "/subgraphs/name/oceanprotocol/ocean-subgraph"
```

```
url = base_url + route

headers = {"Content-Type": "application/json"}
payload = json.dumps({"query": query})
response = requests.request("POST", url, headers=headers, data=payload)
result = json.loads(response.text)

print(json.dumps(result, indent=4, sort_keys=True))
```

Execute script

```
python list_fixed_rate_exchanges.py
```

Javascript

The javascript below can be used to run the the query. If you wish to change the network, then replace the value of variable `baseUrl` as needed.

Create script

listFRE.js

```
var axios = require('axios');

const query = `{
  fixedRateExchanges(skip:0, first:2, subgraphError:deny){
    id
    contract
    exchangeId
    owner{id}
    datatoken{
      id
      name
      symbol
    }
    price
    datatokenBalance
    active
    totalSwapValue
    swaps(skip:0, first:1){
      tx
      by {
        id
      }
      baseTokenAmount
      dataTokenAmount
      createdTimestamp
    }
    updates(skip:0, first:1){
      oldPrice
      newPrice
```

```
      newActive
      createdTimestamp
      tx
    }
  }
}`

const baseUrl = "https://v4.subgraph.mainnet.oceanprotocol.com"
const route = "/subgraphs/name/oceanprotocol/ocean-subgraph"

const url = `${baseUrl}${route}`

var config = {
  method: 'post',
  url: url,
  headers: { "Content-Type": "application/json" },
  data: JSON.stringify({ "query": query })
};

axios(config)
  .then(function (response) {
    console.log(JSON.stringify(response.data));
  })
  .catch(function (error) {
    console.log(error);
  });
```

Execute script

```
node listFRE.js
```

Response

Sample response


```
{
  "data": {
    "fixedRateExchanges": [
      {
        "active": true,
        "contract": "0xfa48673a7c36a2a768f89ac1ee8c355d5c367b02",
        "datatoken": {
          "id": "0x9b39a17cc72c8be4813d890172eff746470994ac",
          "name": "Delightful Pelican Token",
          "symbol": "DELPEL-79"
        },
        "datatokenBalance": "0",
        "exchangeId": "0x06284c39b48afe5f01a04d56f1aae45dbb29793b190ee11e93a4a7721538
        "id": "0xfa48673a7c36a2a768f89ac1ee8c355d5c367b02-0x06284c39b48afe5f01a04d56f
        "owner": {
          "id": "0x03ef3f422d429bcbd4ee5f77da2917a699f237ed"
        },
        "price": "33",
        "swaps": [
          {
            "baseTokenAmount": "33.033",
            "by": {
              "id": "0x9b39a17cc72c8be4813d890172eff746470994ac"
            },
            "createdTimestamp": 1656563684,
            "dataTokenAmount": "1",
            "tx": "0x0b55482f69169c103563062e109f9d71afa01d18f201c425b24b1c74d3c282a3
          }
        ],
        "totalSwapValue": "0",
        "updates": []
      },
      {
        "active": true,
        "contract": "0xfa48673a7c36a2a768f89ac1ee8c355d5c367b02",
        "datatoken": {
          "id": "0x2cf074e36a802241f2f8ddb35f4a4557b8f1179b",
          "name": "Arcadian Eel Token",
          "symbol": "ARCEEL-17"
        },
        "datatokenBalance": "0",
        "exchangeId": "0x2719862ebc4ed253f09088c878e00ef8ee2a792e1c5c765fac35dc18d7ef
        "id": "0xfa48673a7c36a2a768f89ac1ee8c355d5c367b02-0x2719862ebc4ed253f09088c87
        "owner": {
          "id": "0x87b5606fba13529e1812319d25c6c2cd5c3f3cbc"
        },
        "price": "35",
        "swaps": [],
        "totalSwapValue": "0",
```

```
        "updates": []
      }
    ]
  }
}
```

Contributing

Help to improve and develop Ocean core software.

Report a bug

To report a bug that isn't a vulnerability, go to the relevant GitHub repository, click on the *Issues* tab and select *Bug report*.

Before reporting a bug, search existing open and closed issues and PRs to see if something has already been reported. If not, then go ahead and create a new bug report, following the structure suggested in the issue template.

Report Vulnerabilities

You may be able to earn a bounty for reporting vulnerabilities in sensitive parts of our code. Check our page on Immunify for the latest bug bounties available. You can also responsibly disclose flaws by emailing us at security@oceanprotocol.com.

Suggest a new feature

Use the *Issues* section of each repository and select *Feature request* to suggest and discuss any features you would like to see added.

As with bug reports, search existing open and closed issues and PRs to see if something has already been reported.

Fix or improve core software

We'd love to have you contribute to any repository within the `oceanprotocol` GitHub organization!

Before you start coding right away, please follow those basic guidelines:

- If no issue for your case is present, open one first before starting to work on something, so it can be discussed.
- Make yourself familiar with eventual repository-specific contribution requirements and code style requirements.
- Because of the weird world of intellectual property, we need you to follow the legal requirements for contributing code.
- Be excellent to each other, as outlined in our Contributor Code of Conduct.

Workflow

A typical code contribution in any Ocean Protocol repository would go as follows:

1. As an external developer, fork the respective repo and push to your own fork. Ocean core developers push directly on the repo under `oceanprotocol` org.

2. You should create a new branch for your changes. The naming convention for branches is: `issue-001-short-feature-description`. The issue number `issue-001` needs to reference the GitHub issue that you are trying to fix. The short feature description helps to quickly distinguish your branch among the other branches in play.
3. To get visibility and Continuous Integration feedback as early as possible, open your Pull Request as a `Draft`.
4. Give it a meaningful title, and at least link to the respective issue in the Pull Request description, like `Fixes #23`. Describe your changes, mention things for reviewers to look out for, and for UI changes screenshots and videos are helpful.
5. Once your Pull Request is ready, mark it as `Ready for Review`, in most repositories code owners are automatically notified and asked for review.
6. Get all CI checks green and address eventual change requests.
7. If your PR stays open for longer and merge conflicts are detected, merge or rebase your branch against the current `main` branch.
8. Once a Pull Request is approved, you can merge it.

Depending on the release management of each repository, your contribution will be either included in a next release, or is put live automatically.

Except for GitHub, you can find most Ocean Protocol core developers in Discord if you have further development questions.

Develop an app or integration on top of Ocean Protocol

Create an app with one of Ocean Protocol's interface points:

Ocean documentation will help. And... you're here:)

Improve these docs

These docs can always be improved. Every content page has an edit link at its end linking you to the content source on GitHub for simple copy editing.

If you found a technical bug or have an improvement suggestion, head over to the repo's *Issues* section:

Apply for a developer job

Really love building on Ocean and want to dive deeper? Consider joining us full time. Our openings are listed at https://github.com/oceanprotocol/devjobs.

Get Funding

Funding can be for contributing to the core software, building apps, doing integrations, fixing bugs, community outreach, and more. Checkout our active funding programs for more information:

- **Ocean DAO** (grants curated by the community).
- **Shipyard** (Ocean curated grants).
- **Data Bounties** (rewards for publishing algorithms and datasets).

Other ways to get involved

Please go to the Ocean Community Page for more ideas on how to get involved.

Contributor Code of Conduct

Be excellent to each other.

As contributors and maintainers of this project, and in the interest of fostering an open and welcoming community, we pledge to respect all people who contribute to the project.

We are committed to making participation in this project a harassment-free experience for everyone, regardless of level of experience, gender, gender identity and expression, sexual orientation, disability, personal appearance, body size, race, ethnicity, age, religion, nationality, or species.

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery
- Personal attacks
- Trolling or insulting/derogatory comments
- Public or private harassment
- Publishing other's private information, such as physical or electronic addresses, without explicit permission
- Deliberate intimidation
- Other unethical or unprofessional conduct

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

By adopting this Code of Conduct, project maintainers commit themselves to fairly and consistently applying these principles to every aspect of managing this project. Project maintainers who do not follow or enforce the Code of Conduct may be permanently removed from the project team.

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community.

Instances of abusive, harassing, or otherwise unacceptable behavior directed at yourself or another community member may be reported by contacting a project maintainer at conduct@oceanprotocol.com. All complaints will be reviewed and investigated and will result in a response that is appropriate to the circumstances. Maintainers are obligated to maintain confidentiality with regard to the reporter of an incident.

This Code of Conduct is adapted from the Contributor Covenant, version 1.3.0, available at contributor-covenant.org/version/1/3/0/

Legal Requirements

How to make sure your code contributions can be included in the Ocean Protocol codebase.

Ocean Protocol Software Licensing

All Ocean Protocol code (software) is licensed under an Apache 2.0 license. This page describes the Ocean Protocol policy to ensure that all contributions to the Ocean Protocol code are also licensed under the Apache 2.0 license (and that the contributor has the right to license it as such).

If you are:

- contributing code to complete a *currently-open* Ocean Protocol bounty or
- a *current* employee of BigchainDB GmbH

then there is nothing extra for you to do: licensing is already handled.

Otherwise you are an "external contributor" and you must do the following:

1. Make sure that every file you modified or created contains a copyright notice comment like the following (at the top of the file):
   ```
   # Copyright Ocean Protocol contributors
   # SPDX-License-Identifier: Apache-2.0
   ```
   - If a copyright notice is not present, then add one.
   - If the first line of the file is a line beginning with `#!` (e.g. `#!/usr/bin/python3`) then leave that as the first line and add the copyright notice afterwards.
   - If a copyright notice is present but it says something like `Copyright 2018 Ocean Protocol Foundation` then please change it to say the above.
   - Make sure you're using the correct syntax for comments (which varies from language to language). The example shown above is for a Python file.
2. Read the Developer Certificate of Origin, Version 1.1.
3. You will be asked to include a Signed-off-by line in all your commit messages. (Instructions are given in the next step.) Make sure you understand that including a Signed-off-by line in your commits certifies that you can make the statements in the Developer Certificate of Origin. If you have questions about this, then please ask on Discord or elsewhere. Do not continue until you fully understand.
4. Make sure that all your commit messages include a Signed-off-by line of the form:
   ```
   Signed-off-by: Random J Developer <random@developer.example.org>
   ```
   with your real name and your real email address. Sorry, no pseudonyms or anonymous contributions. Tip: You can tell Git to include a Signed-off-by line in a commit message by using `git commit --signoff` or `git commit -s`.

Credits

The Developer Certificate of Origin was developed by the Linux community and has since been adopted by other projects, including many under the Linux Foundation umbrella (e.g. Hyperledger Fabric). The process described above (with the Signed-off-by line in Git commits) is also based on the process used by the Linux community.

The Future

In the future, the Ocean Protocol Foundation will dissolve and the policy will probably change to work more like the Linux Kernel, where *every* contributor must include a Signed-off-by line in all Git commits.

Partners & Collaborators

We work with many partners who understand the value of Ocean

We work closely with our collaborators and service partners to iterate on our underlying technology, and to deliver world-class Web3 experiences built on top of Ocean Protocol. An up to date list of our partners and collaborators is maintained on our main site.

Other Useful Information

OceanDAO projects

Ocean Pearl is a great way to browse 80+ Ocean projects that came through OceanDAO. These projects may be building on the Ocean stack, doing outreach, unlocking data, or more.

Using Ocean Market

Ocean Market is the best place to find projects that publish datasets. Simply go there and browse the datasets:)

Learning about Ocean

The Ocean Academy project is a great way to learn more about Ocean beyond oceanprotocol.com and docs.oceanprotocol.com.

Trading OCEAN

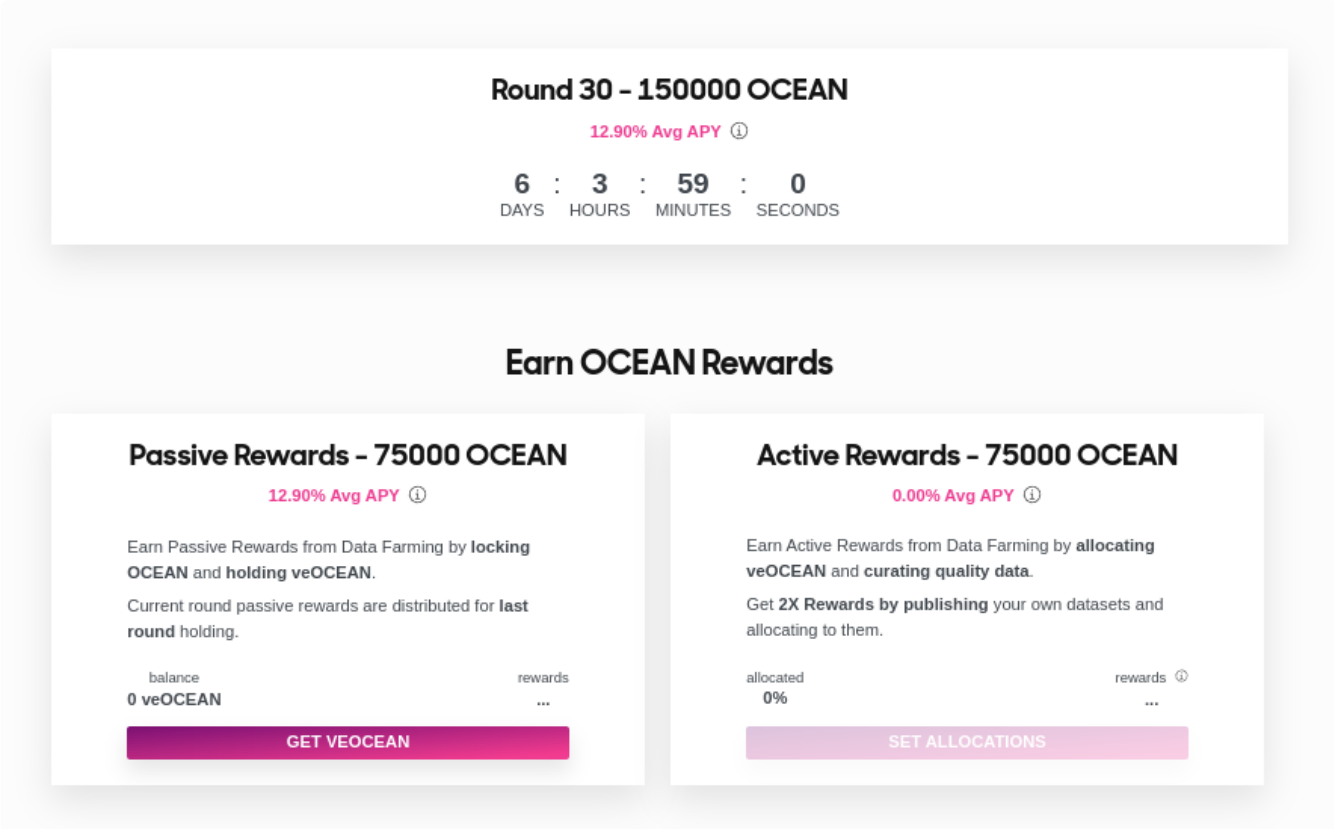
The Coingecko OCEAN markets page lists forums to exchange OCEAN. Many of them offer liquidity mining and other yield opportunities.

veOCEAN & Data Farming

An overview of Ocean Protocol's governance and incentives mechanisms

veOCEAN is a fork of veCRV, and enables you to become a governance participant, eligible to receive rewards and engage with different protocol mechanisms.

The following docs should provide you with sufficient intuition to access, utilize, and build upon the protocol's core incentive and reward system: Data Farming.



veOCEAN

Learning about veOCEAN will help you answer the question "What can I do with my veOCEAN?" and give you insights on how veOCEAN works. It will teach you everything you need to know about why it exists and how it works.

You will learn that by just holding veOCEAN passively, you are able to earn rewards.

Data Farming

Data Farming 101 will teach you about the different reward systems, how they work, and how to access them. By the end of it, you should be more familiar with how Data Farming works and able to take next steps to curate assets.

Data Farming Background will provide you with more intuitions about Data Farming, briefly explain the Reward Function, and how the program evolved over time.

Further Reading

Finally, if you want to continue expanding your knowledge on OCEAN token emissions, APY estimates, and get useful answers to some of the most common questions, you can read the following:

Emissions & APYs will provide you will information about how OCEAN will be released over time through the Data Farming program and provide you with APY studies.

Our FAQ answers many different questions about staking, chains, deployments, and other details that may be valuable to you.

Reference

All content within has been assembled via reference of the Ocean Data Farming Series, official Ocean Protocol github repositories, and v4 Whitepapers.

veOCEAN

An overview of the governance token, veOCEAN (vote-escrowed).

veOCEAN is a mechanism to align near-term incentives (maximize APY) with long-term incentives (long-term locking). It's a fork of veCRV contracts which have been battle-tested over years.

The amount of reward depends on how long the tokens are locked for. You must lock up your OCEAN into the vault for to obtain veOCEAN. Going forward, veOCEAN will be the main mechanism for staking OCEAN, and for curation of datasets.

After creating your lock you will be credited veOCEAN. We sometimes refer to veOCEAN as your “voting power”.

WARNING: You will not be able to retrieve your original OCEAN deposit until the lock ends.

What can I do with veOCEAN?

veOCEAN allows you to engage with different protocol mechanisms and benefit from the reward programs available.

There are 4 things you can do with veOCEAN.

1. **Hold it** veOCEAN pays **Passive Rewards** every week.
2. **Allocate it** veOCEAN pays **Active Rewards** every week to the top performing Datasets, Algorithms, dApps, and more.
3. **Delegate it** You can delegate veOCEAN to other Data Farmers who can curate Datasets for you. In return for their services, these farmers may charge you a fee for helping you receive APY on **Active Rewards**. The Delegate feature has just been recently released and enables veOCEAN holders to more easily access Active Rewards.
4. **2x Stake** If you are a publisher, allocating veOCEAN to your own Dataset gives your veOCEAN a 2x Bonus. This is an incentive for publishers to engage with their assets and benefit from from the protocol further.

What is time locking?

Users can lock their OCEAN for different lengths of time to gain voting power. Our app is configured to lock OCEAN for a minimum of 2 weeks and a maximum of four years for max benefit.

Users that lock their OCEAN for a longer period of time receive more veOCEAN to reflect their conviction in the system.

Year	Lock Multiplier	veOCEAN
1	0.25x	0.25
2	0.50x	0.50
3	0.75x	0.75
4	1.00x	1.00

The Lock Multiplier. Amount of veOCEAN received per OCEAN locked.

If you've locked OCEAN for 4 years, you will be unable to retrieve your deposit until this time expires.

After choosing your lock period and locking up your OCEAN into the vault, you will be credited with veOCEAN.

veOCEAN is non-transferable. You can't sell it or send it to other addresses.

Linear Decay

Your veOCEAN balance will slowly start declining as soon as you receive it.

veOCEAN balance decreases linearly over time until the Lock End Date. When your lock time has lapsed by 50%, you will have 50% of your original veOCEAN balance.

When your lock time ends your veOCEAN balance will hit 0, and your OCEAN tokens can be withdrawn.

If you lock 1.0 OCEAN for 4 years, you get 1.0 veOCEAN at the start.

Years Passed	veOCEAN Left
1 year	0.75
2 years	0.50
3 years	0.25
4 years	0.00

At the end of your 4 years, your OCEAN is unlocked.

Replenishing your veOCEAN

You can choose to update your lock and replenish your veOCEAN balance at any time.

To maximize rewards, participants would need to update their 4-year lock every week in order to maintain their veOCEAN balance as high as possible.

veOCEAN Earnings

All earnings for veOCEAN holders are claimable in Ethereum mainnet. (Data assets for DF may published in any network where Ocean's deployed in production: Eth mainnet, Polygon, etc.)

There's a new DF round every week; in line with this, there's a new ve distribution "epoch" every week. This affects when you can first claim rewards. Specifically, if you lock OCEAN on day x, you'll be able to claim rewards on the first ve epoch that begins after day x+7. Put another way, from the time you lock OCEAN, you must wait at least a week, and up to two weeks, to be able to claim rewards. (This behavior is inherited from veCRV. Here's the code.)

veOCEAN holders have earnings from two sources:

Earnings from Community Fees

Every transaction in Ocean Market and Ocean backend generates transaction fees, some of which go to the community. 50% of the community fees will go to veOCEAN holders; the rest will go to Ocean community-oriented traction programs.

All earnings here are passive.

Earnings from Data Farming

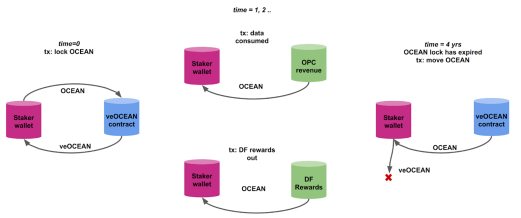
veOCEAN holders will each get weekly DF rewards allocation, except a small carveout for any Data Challenge initiatives that may run through DF ops.

veOCEAN holders can be passive, though they will earn more if active.

"Being active" means allocating veOCEAN to promising data assets (data NFTs). Then, rewards follow the usual DF formula: DCV * stake. Stake is the amount of veOCEAN allocated to the data asset. There is no liquidity locked inside a datatoken pool. (And this stake is safe: you can't lose your OCEAN as it is merely locked.)

Flow of Value

The image below illustrates the flow of value. On the left, at time 0, the staker locks their OCEAN into the veOCEAN contract, and receives veOCEAN. In the middle, the staker receives OCEAN rewards every time there's revenue to the Ocean Protocol Community (top), and also as part of Data Farming rewards (bottom). On the right, when the lock expires (e.g. 4 years) then the staker is able to move their OCEAN around again.



Flow of Value

The veOCEAN design is in accordance with the Web3 Sustainability Loop, which Ocean uses as its system-level design.

The veOCEAN code was forked from the veCRV code. veCRV parameters will be the starting point. To minimize risk, tweaks will be circumspect.

Security

veOCEAN core contracts use veCRV contracts with zero changes, on purpose: the veCRV contracts have been battle-tested for two years and have not had security issues. Nearly 500 million USD is locked across all forks of veCRV, with the leading DeFi protocols adopting this standard. veCRV contracts have been audited by Trail of Bits and Quantstamp.

We have built a new contract for users to point their veOCEAN towards given data assets (“allocate veOCEAN”). These new contracts do not control the veOCEAN core contracts at all. In the event of a breach, the only funds at risk would be the rewards distributed for a single week; and we would be able to redirect future funds to a different contract.

We have an ongoing bug bounty via Immunefi for Ocean software, including veOCEAN and DF components. If you identify an issue, please report it there and get rewarded.

Data Farming 101

An introduction to Data Farming and Ocean Protocol's key incentives.

Data Farming (DF) incentivizes for growth of Data Consume Volume (DCV) in the Ocean ecosystem.

It rewards OCEAN to liquidity providers (stakers) as a function of consume volume and liquidity. It's like DeFi liquidity mining, but tuned for data consumption. DF's aim is to achieve a minimum supply of data for network effects to kick in, and once the network flywheel is spinning, to increase growth rate.

Reward Categories

Rewards are paid in OCEAN and distributed every week on Thursday as follow:

Passive Rewards	Active Rewards
50%	50%

Active Rewards are governed and defined by the Reward Function.

Final Caveat: We reserve the right to make reasonable changes to these plans, if unforeseen circumstances emerge.

How to access DF and claim rewards

Please follow this tutorial to learn how the Ocean Protocol reward programs work, and how to access them.

Otherwise, go to the DF webapp at df.oceandao.org and explore Data Farming for yourself.

Where to claim?

All earnings for veOCEAN holders are claimable on the "Rewards" page inside the Data Farming webapp on Ethereum mainnet.

Data assets for DF may published in any network where Ocean's deployed in production: Eth mainnet, Polygon, etc.

When to claim?

There are fresh rewards available every Thursday. If you wish, you can wait for many weeks to accumulate before claiming. (It's all on-chain.)

When to do a first claim?

From the time you lock OCEAN, you must wait at least a week, and up to two weeks, to be able to claim rewards.

The nerdy version: if you lock OCEAN on day x, you'll be able to claim rewards on the first weekly ve "epoch" that begins after day x+7.

This behavior is inherited from veCRV; here's the code.

DF Main

DF Main started Mar 16, 2023 in DF Round 29. DF29 has 150K OCEAN rewards available (a 2x increase from DF28). As DF Main progresses, rewards will increase to 300K (another 2x), then 600K (another 2x), then beyond 1.1M OCEAN/week (near 2x) then decaying over time.

As of DF29 (Mar 16, 2023), wash consuming is not profitable. So, organically-generated Data Consume Volume is the main driver of active DF rewards.

Example APYs are 5–20% between Passive & Active rewards.

Full implementation of DF Main will be over many months, after which DF will be decentralized.

DF Main lasts for decades.

Reward Schedule

The table below cross-references DF Round Number, Start Date, Phase & Week, Sub-Phase & Week, and OCEAN Rewards/Week.

DF #	Start date	Phase & week	Sub-phase & week	OCEAN / wk
DF1	2022-06-16	DF Alpha Week 1	DF Alpha - Week 1	10,000
…	…	…	…	…
DF4	2022-07-07	DF Alpha Week 4	DF Alpha - Week 1	10,000
DF5	2022-09-29	DF/VE Alpha Week 1	DF/VE Alpha - Week 1	10,000
…	…	…	…	…
DF8	2022-10-20	DF/VE Alpha Week 4	DF/VE Alpha - Week 4	10,000
DF9	2022-10-27	DF Beta Week 1	DF Beta 1 - Week 1	50,000
…	…	…	…	…
DF18	2022-12-29	DF Beta Week 10	DF Beta 1 - Week 10	50,000
DF19	2023-01-05	DF Beta Week 11	DF Beta 2 - Week 1	75,000
…	…	…	…	…
DF28	2023-03-09	DF Beta Week 20	DF Beta 2 - Week 10	75,000
DF29	2023-03-16	DF Main Week 1	DF Main 1 - Week 1	150,000
…	…	…	…	…
DF79	2024-03-07	DF Main Week 52	DF Main 1 - Week 52	150,000
DF80	2024-03-14	DF Main Week 53	DF Main 2 - Week 1	300,000
…	…	…	…	…
DF105	2024-09-05	DF Main Week 78	DF Main 2 - Week 26	300,000
DF106	2024-09-12	DF Main Week 79	DF Main 3 - Week 1	600,000
…	…	…	…	…
DF131	2025-03-06	DF Main Week 104	DF Main 3 - Week 26	600,000
DF132	2025-03-13	DF Main Week 105	DF Main 4 - Week 1	approx 1,100,000
…	…	…	…	*halven every 4 yrs*
DF1000+	21xx-xx-xx	DF Main Week 1000+	DF Main 4 - Week 1000+	

Ocean Reward Schedule for the next 20+ years

Ranked Rewards

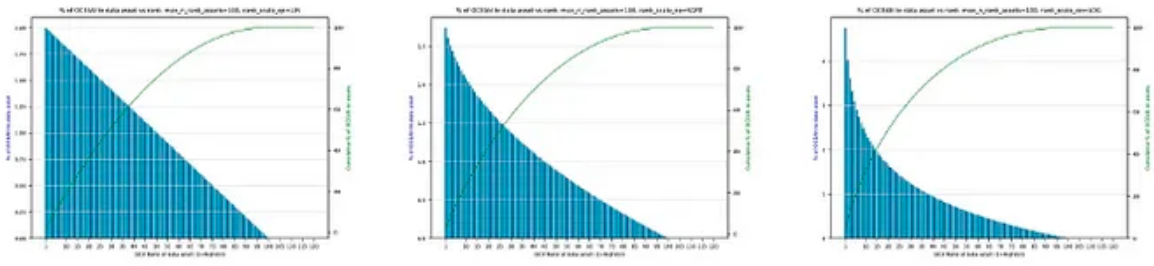
In DF23 Ranked Rewards were introduced and smooth the reward distribution by using a logarithmic function.

Since rewards are distributed across the Top 100 assets, all participants (Publishers & Curators) are now incentivized to support a broader range of assets rather than optimizing on a single asset.

At the top-end, this helps increase quality and diversification of inventory.

At the bottom-end, this eliminates some potential free-rider issues and smooths out the reward distribution.

The images below explore the effect of scaling (LIN vs SQRT vs LOG), for is when *max_n_rank_assets* = 100. Going from LIN to SQRT, the dropoff after the top-ranked asset is sharper. Going to LOG, it's sharper yet. In LIN, the top-ranked asset gets 2% of OCEAN reward; in SQRT it's about 2.7%; in LOG it's about 4.8%.



Effect of scaling: LIN, SQRT, and LOG. max_n_rank_assets = 100

You can read more about the implementation in this blog post and find the full study in these slides.

Publisher Rewards - 2x Stake

DF gives stronger incentives to publish data services, as follows.

All the veOCEAN a publisher has allocated to an asset they've published ("staked") is treated as 2x the stake for rewards calculation.

1. As a staker, due to their staked veOCEAN on their own assets (1x).
2. As a publisher, for having veOCEAN staked on their own asset(1x).

The final reward is then calculated and bundled together to be distributed.

You can read more about the implementation in this blog post.

Data Farming Background

Data Farming (DF) incentivizes for growth of Data Consume Volume (DCV) in the Ocean ecosystem.

It rewards OCEAN to stakers as a function of consume volume and liquidity. It's like DeFi liquidity mining, but tuned for data consumption. DF's aim is to achieve a minimum supply of data for network effects to kick in, and once the network flywheel is spinning, to increase growth rate.

Active Work to Drive APY

Data Farming is not a wholly passive activity. The name of the game is to drive Data Consume Volume (DCV). High APYs happen only when there is sufficiently high DCV. High DCV means publishing and consuming truly useful datasets (or algorithms).

Thus, if you really want to max out your APY:

1. Create & publish datasets (and make $ in selling them) — or work with people who can
2. Lock OCEAN and stake veOCEAN on them.
3. Buy the datasets (and use them to make $) — or work with people who can
4. Claim the rewards.

Driving DCV for publishing & consuming is your challenge. It will take real work. And then the reward is APY. It's incentives all the way down :)

Reward Function

The Reward Function (RF) governs how active rewards are allocated to stakers.

Rewards are calculated as follows:

1. Distribute OCEAN across each asset based on rank: highest-DCV asset gets most OCEAN, etc.
2. For each asset and each veOCEAN holder: – If the holder is a publisher, 2x the effective stake – Baseline rewards = (% stake in asset) * (OCEAN for asset) – Bound rewards to the asset by 125% APY – Bound rewards by asset's DCV * 0.1%. This prevents wash consume.

You can find this code inside calcrewards.py in the Ocean Protocol df-py repo

Data Assets that Qualify for DF

Data assets that have veOCEAN allocated towards them get DF rewards.

The data asset may be of any type — dataset (for static URIs), algorithm for Compute-to-Data, or any other Datatoken token-gated system. The data asset may be fixed price or free price. If fixed price, any token of exchange is alright (OCEAN, H2O, USDC, ..).

To qualify for DF, a data asset must also:

- Have been created by Ocean Smart contracts deployed by OPF to production networks
- Be visible on Ocean Market
- Can't be in purgatory

4 Phases of Data Farming

Data Farming has evolved over time and will continue to do so as the Emission Curve progresses. We are now in DF main, below are the previous phases and parameters incurred during the evolution of the Data Farming program.

DF Alpha - Rounds 1-4 (4 wks)
10K OCEAN rewards were budgeted per week. Counting started Thu June 16, 2022 and ended July 13, 2022. Rewards were distributed at the end of every week, for the activity of the previous week. It ran for 4 weeks. The aim was to test technology, learn, and onboard data publishers.

DF/VE Alpha - Rounds 5-8 (4 wks)
10K OCEAN rewards were budgeted per week. Counting started Thu Sep 29, 2022 and ended Oct 27, 2022. Rewards were distributed at the end of every week, for the activity of the previous week. It ran for 4 weeks. The aim was to resume Data Farming along with veOCEAN, test the technology, onboard data publishers, and keep learning.

DF Beta - Rounds 9-28 (20 wks)
Up to 100K OCEAN rewards were budget per week. Counting started Thu Oct 27, 2022 and ended March 15, 2023. It ran for 20 weeks. The aim was to test the effect of larger incentives, support ecosystem participation, while continue refining the underlying technology.

DF Main - Rounds 29-1000+
Immediately followed the release of DF Beta on Thu Mar 16, 2023. Rewards begin at 150k per week and go to 1.1M OCEAN per week. DF Main emits 503.4M OCEAN worth of rewards and lasts for decades. Expected APY is 125% over many months (once fully ramped), staying generous over the long term.

The amount of OCEAN released is determined by the emission schedule as defined by the Emission Curve, and perhaps more easily uderstood in the Reward Schedule

Emissions & APYs

Details on the emission curves and a study on estimated APYs

With veOCEAN, OceanDAO evolves to be more like CurveDAO:

- ve is at the heart with v = voting (in data asset curation) and e = escrowed (locked) OCEAN. The longer the lockup, the more voting and rewards, which reconciles near and long-term DAO incentives.
- The DAO has increased bias to automation, and to minimizing the governance attack surface.

The baseline emissions schedule determines the weekly OCEAN budget for this phase. The schedule is like Bitcoin, including a half-life of 4 years. Unlike Bitcoin, there is a burn-in period to ratchet up value-at-risk versus time:

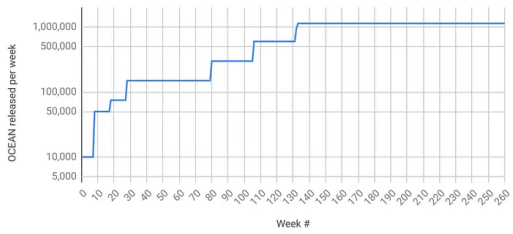
- The curve initially gets a multiplier of 10% for 12 months (DF Main 1)
- Then, it transitions to multiplier of 25% for 6 months (DF Main 2)
- Then, a multiplier of 50% for 6 months (DF Main 3)
- Finally, a multiplier of 100%. (DF Main 4)

We implement the first three phases as constants, because they are relatively short in duration. We implement the fourth phase as a Bitcoin-style exponential: constant, with the constant dividing by two (“halvening”) every four years.

Let’s visualize!

Emissions — first 5 years.

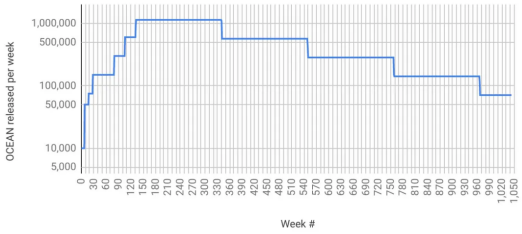
The image below shows the first 5 years. The y-axis is OCEAN released each week. It’s log-scaled to easily see the differences. The x-axis is time, measured in weeks. In weeks 0–29, we can see the distinct phases for DF Alpha (DF1 // week 0), DF/VE Alpha (DF5 // week 4), DF Beta (DF9 // week 8), DF Main 1 (DF29 // week 28), DF Main 2 (DF80 // week 79), DF Main 3 (DF106 // week 105), and DF Main 4 (DF132 // week 131).



OCEAN released to DF per week — first 5 years

Emissions — First 20 years.

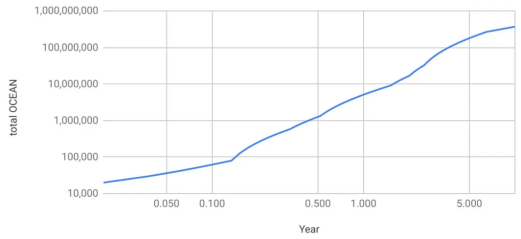
The image below is like the previous one: OCEAN released per week, but now for the first 20 years. Week 131 onwards is DF Main 4. We can see that the y-value divides by two (“halvens”) every four years.



OCEAN released to DF per week — first 20 years

Total OCEAN released.

The image below shows the total OCEAN released by DF for the first 20 years. The y-axis is log-scaled to capture both the small initial rewards and exponentially larger values later on. The x-axis is also log-scaled so that we can more readily see how the curve converges over time.

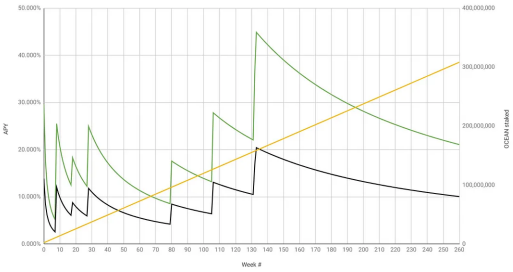


Total OCEAN released to DF — first 20 years

Example APYs

The plot below shows estimated APY over time. Green includes both passive and active rewards; black is just passive rewards. As of DF29, wash consume is no longer profitable, so we should expect a large drop in DCV and therefore in active rewards. So passive rewards (black) provides a great baseline with upside in active rewards (green).

APYs are an estimate because APY depends on OCEAN locked. OCEAN locked for future weeks is not known precisely; it must be estimated. The yellow line is the model for OCEAN locked. We modeled OCEAN locked by observing linear growth from week 5 (when OCEAN locking was introduced) to week 28 (now): OCEAN locked grew from 7.89M OCEAN to 34.98M OCEAN respectively, or 1.177M more OCEAN locked per week.



Green: estimated APYs (passive + active). Black: estimated APYs (just passive). Yellow: estimated staking
The plots are calculated from this Google Sheet.

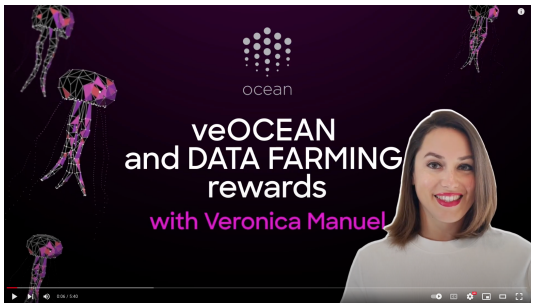
OCEAN lock time affects APY. The numbers above assume that all locked OCEAN is locked for 4 years, so that 1 OCEAN → 1 veOCEAN. But APY could be much worse or more if you lock for shorter durations. Here are approximate bounds.

If you lock for 4 years, and everyone else locks for 2, then multiply expected APY by 2. If you lock for 4 years and others for 1, then multiply by 4. Conversely, if you lock for 2 years and everyone else for 4, then divide your expected APY by 2. If you lock for 1 year and others for 4, then divide by 4. The numbers assume that you're actively allocating veOCEAN allocation towards high-DCV data assets. For passive locking or low-DCV data assets, divide APY by 2 (approximate).

Rewards Tutorial

Follow this step-by-step guide to get OCEAN rewards.

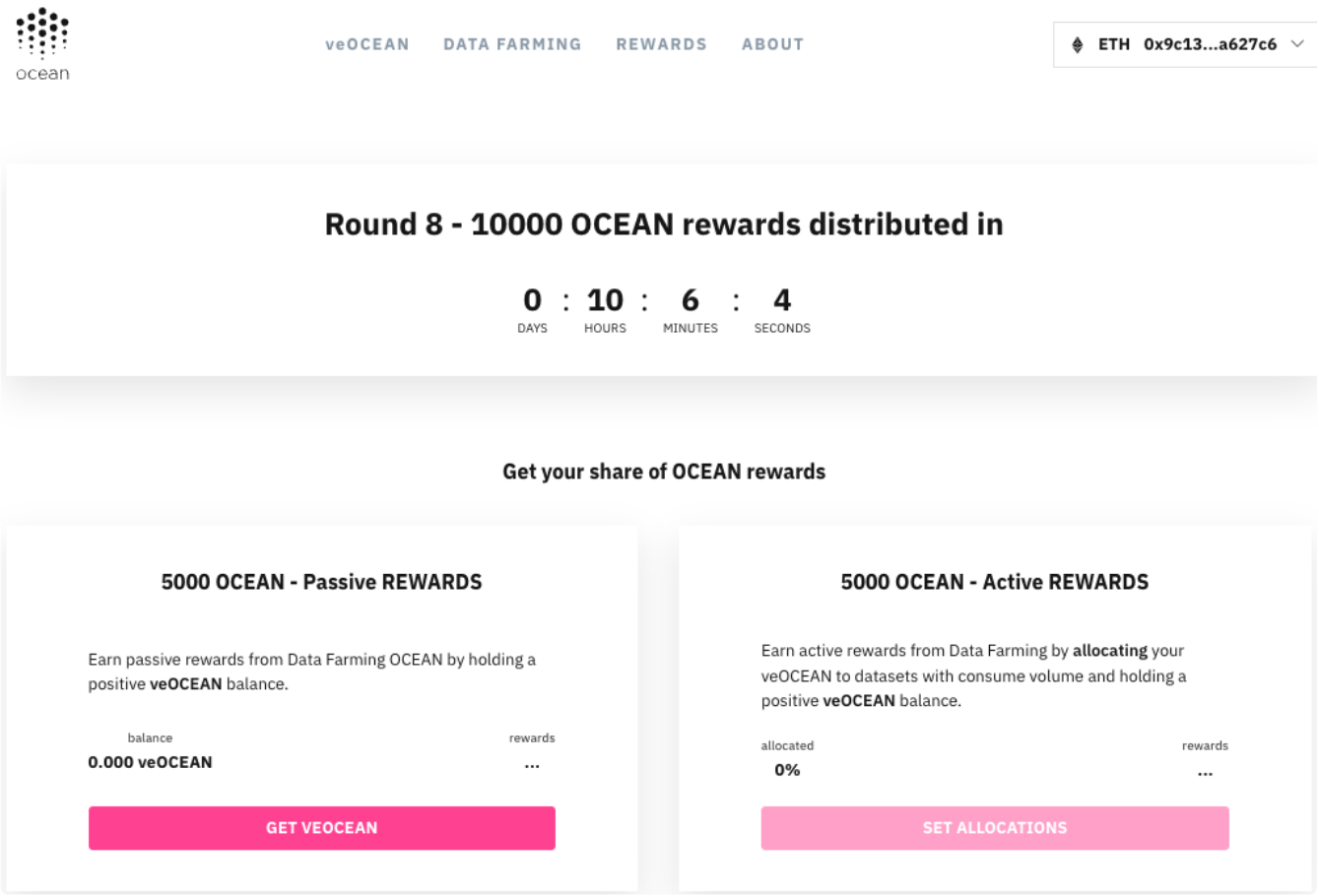
There are two types of OCEAN rewards: passive and active rewards. OCEAN token holders may generate passive OCEAN rewards by locking up OCEAN in exchange for veOCEAN tokens. veOCEAN tokens can then be allocated to Ocean Market datasets and algorithms to generate active OCEAN rewards.



How do I get rewards?

To generate rewards, start by navigating to df.oceandao.org. At the top of this page is the weekly round of OCEAN rewards and the quantity of OCEAN rewards to be distributed. The countdown timer shows the time until **each Thursday** when rewards are distributed. OCEAN rewards can be claimed every Thursday on the Rewards page.

- If you don't have veOCEAN tokens, then click the Get veOCEAN button on the Passive Rewards panel on the left side of the page to navigate to the veOCEAN tab.
- If you already have veOCEAN tokens, then on the Active Rewards panel on the right side of the page, click the Set Allocations button to navigate to the Data Farming tab.



veOCEAN

Passive Rewards

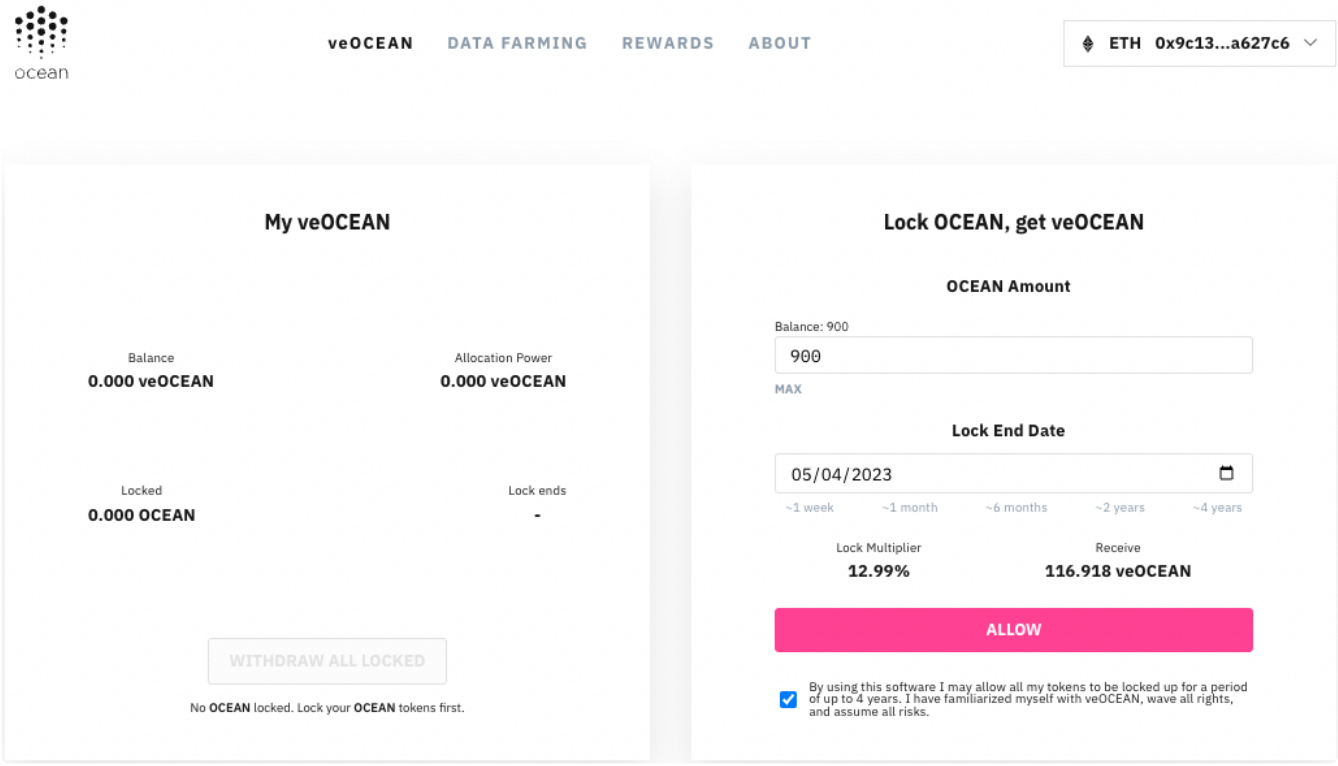
When a user locks their OCEAN tokens for a finite period of time, they get veOCEAN tokens in return. Based on the quantity of veOCEAN, the user accumulates weekly OCEAN rewards. Because rewards are generated without human intervention, these are called "passive" OCEAN rewards. OCEAN rewards are claimable every Thursday on the Rewards page.

How do I get veOCEAN?

After navigating to the veOCEAN page, you can generate passive OCEAN rewards by locking OCEAN tokens on the "Lock OCEAN, get veOCEAN" panel on the right side of the page. Connect a wallet to see the balance of OCEAN tokens update above the OCEAN Amount form field. Select a lock end date to see the Lock Multiplier and Receive veOcean fields update.

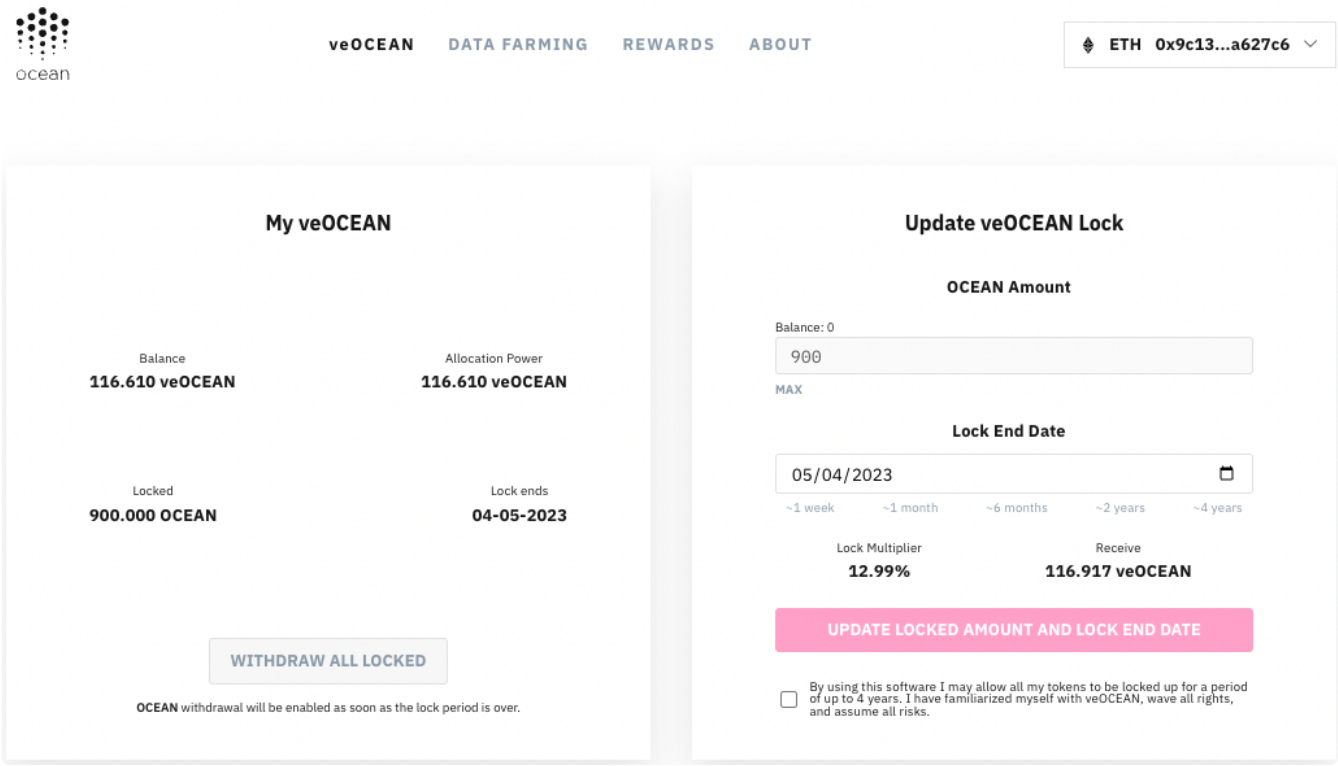
The more OCEAN tokens you lock or if you lock them for a longer period of time, then the more rewards you get!

Click the checkbox below the inactive pink ALLOW button, then click the activated pink ALLOW button. Sign the transaction with your wallet. Then, click the LOCK OCEAN button. Sign the transaction with your wallet. Note that all veOCEAN contracts are deployed on the Ethereum mainnet.



Now the OCEAN tokens are locked in exchange for veOCEAN. The left side panel called "My veOCEAN" shows the corresponding balances of OCEAN and veOCEAN. You can withdraw your OCEAN tokens on this panel when the lock time ends.

Note that your OCEAN tokens cannot be withdrawn until the Lock End Date!



Notice the right side panel is now titled "Update veOCEAN Lock". You can add OCEAN tokens to your lock or you can increase the Lock End Date, but you cannot shorten your Lock End Date.

Data Farming

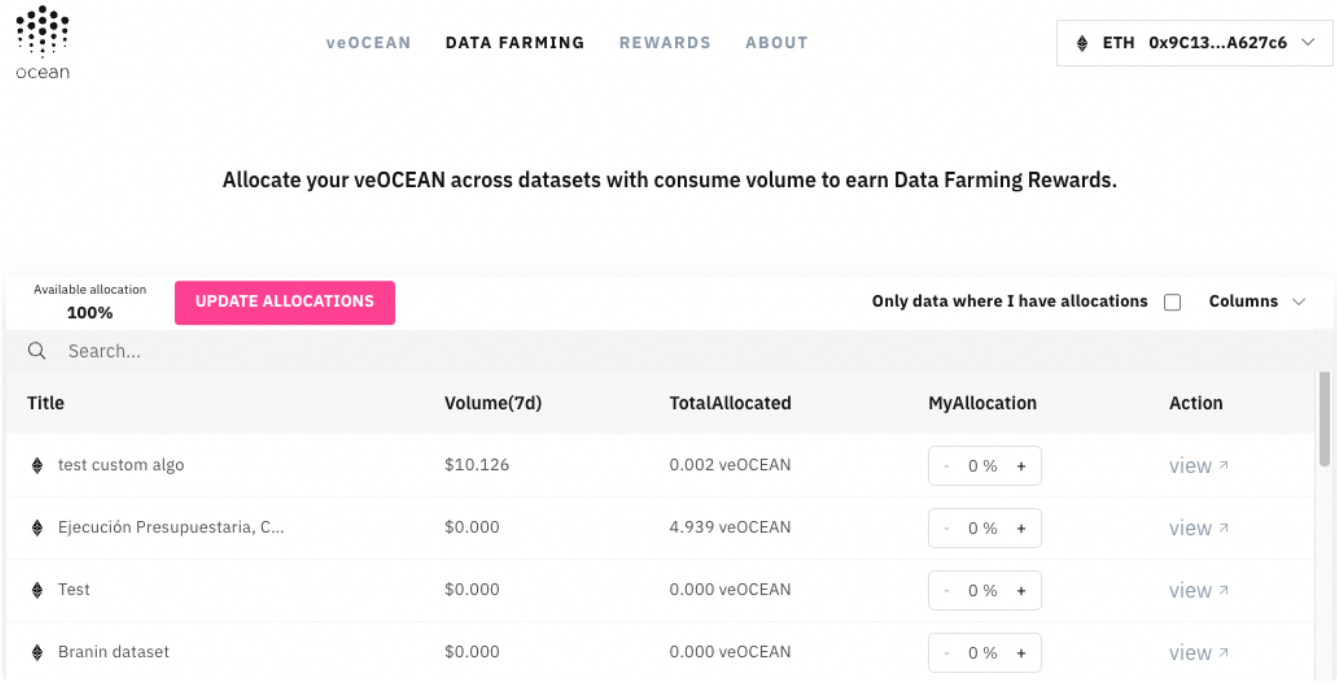
Active Rewards

When a user allocates veOCEAN tokens to Ocean Market projects, then weekly OCEAN rewards are given to a user based on the sales of those projects. Since these rewards depend on human intervention to decide the allocations, they are categorized as "active" rewards instead of passive rewards. OCEAN rewards are claimable every Thursday on the Rewards page.

How do I allocate veOCEAN?

You can generate active OCEAN rewards by allocating veOCEAN to various OCEAN Market projects to gain a portion of project sales.

Click on the Data Farming tab at the top of the page to navigate to the Data Farming page.



The OCEAN Market datasets and algorithms are listed in the grid. Each column is sortable, and there is a Search field on top of the grid to search for specific projects.

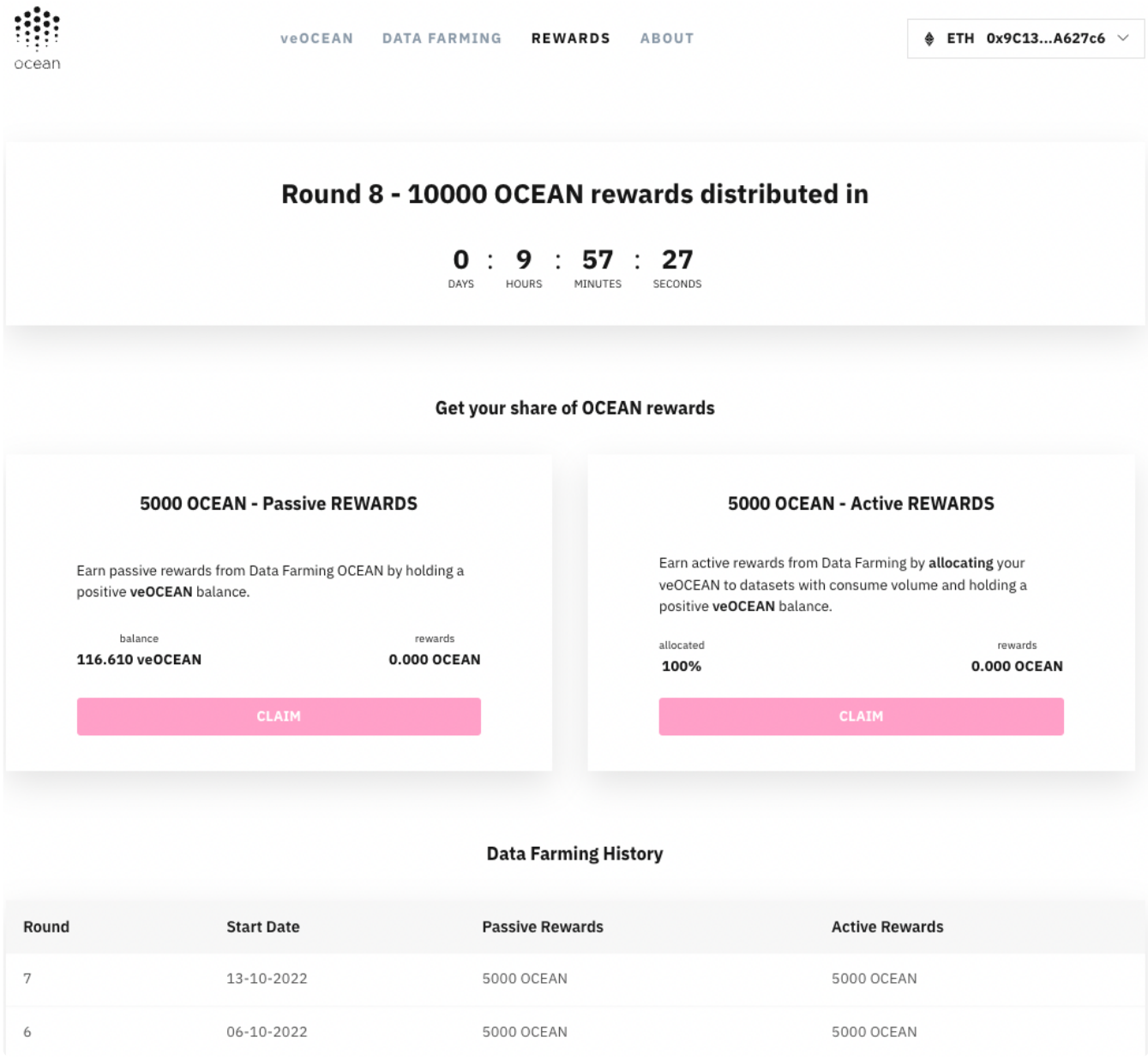
It is recommended to allocate all of your veOCEAN tokens to OCEAN Market projects to generate maximum rewards. When you allocate your veOCEAN to these datasets or algorithms, then you get a portion of those projects's sales. **If you allocate your veOCEAN to a project with many other allocators, then your portion of rewards will become diluted because there are more allocators to reward. You may be interested then in allocating veOCEAN to a project with fewer allocators to generate a greater percentage of rewards. However, if a project does not sell, then no rewards are generated.** Thus, it is important to allocate veOCEAN to projects you believe in, and do your research.

Once you allocate your percentage of veOCEAN to projects using the MyAllocation column, then click on the UPDATE ALLOCATIONS button and sign the transaction with your wallet. Note that as the OCEAN Market projects exist on different networks i.e. Ethereum, Polygon, etc. you can allocate your veOCEAN towards assets that are published on these different networks.

Claim Rewards

Click on the Rewards tab at the top of the page to come to the same page as at the beginning of this tutorial. Notice the balance of veOCEAN appears under the Passive Rewards panel on the left and the percentage allocated appears on the Active Rewards panel on the right.

All rewards are paid out in OCEAN tokens. On every Thursday the pink "Claim" buttons on this page become activated, and you can claim your weekly OCEAN rewards directly into your wallet by clicking on these active buttons.



Linear Decay

Your balance of veOCEAN may be less than the amount when you first locked your tokens because your veOCEAN balance decreases linearly over time until the Lock End Date when you can withdraw your OCEAN tokens. This is because rewards are designed to be paid out weekly in a decreasing amount until you unlock your OCEAN tokens entirely. The veOCEAN code is a fork of Curve's battle tested veCRV token code.

Withdrawl

After the Lock End Date, then you can withdraw your principal OCEAN tokens on the veOCEAN page on the left side panel.

Learn More

If you would like to find out more details about veOCEAN, Data Farming, and rewards calculations, then please visit the About tab to read a great blog post on this topic.

API References

Aquarius REST API

Assets

Get `/api/aquarius/assets/ddo/<did>`

- Description
  Get DDO of a particular asset.
- Parameters

name	description	type	in	required
`did`	DID of the asset	string	path	true

- Example

```
curl --location --request GET 'https://v4.aquarius.oceanprotocol.com/api/aquarius/assets/d
```

- Responses
  - 200
    - content-type: json
    - description: On successful operation returns DDO information.
  - 404
    - content-type: json
    - description: This asset DID is not in ES.
    - response body:

```
{
    "error": "Asset DID <did> not found in Elasticsearch."
}
```

GET `/api/aquarius/assets/metadata/<did>`

- Description
  Get metadata of a particular asset.
- Parameters

name	description	type	in	required
`did`	DID of the asset	string	path	true

- Example

```
curl --location --request GET 'https://v4.aquarius.oceanprotocol.com/api/aquarius/assets/m
```

- Responses
  - 200
    - content-type: json
    - description: successful operation.
  - 404
    - content-type: json
    - description: This asset DID is not in ES.
    - response body:

```
{
    "error": "Error encountered while retrieving metadata: NotFoundError(404, '{\"_
}
```

POST `/api/aquarius/assets/names`

- Description
  Get names of assets as specified in the payload.
- Parameters

name	description	type	in	required
`didList`	list of asset DIDs	list	body	true

- Example

```
curl --location --request POST 'https://v4.aquarius.oceanprotocol.com/api/aquarius/assets/
--header 'Content-Type: application/json' \
--data-raw '{
    "didList" : ["did:op:cd086344c275bc7c560e91d472be069a24921e73a2c3798fb2b8caadf8d245d6"
}'
```

- Responses
  - 200
    - content-type: json
    - description: successful operation.
    - response body:

      ```
      {"did:op:cd086344c275bc7c560e91d472be069a24921e73a2c3798fb2b8caadf8d245d6": "Ocean
      ```
  - 400
    - content-type: json
    - description: This asset DID is not in ES.
    - response body:

      ```
      {
      "error": "The requested didList can not be empty."
      }
      ```

POST `/api/aquarius/assets/query`

- Description
  Run a native ES query. Body must be a valid json object.
- Example

```
curl --location --request POST 'https://v4.aquarius.oceanprotocol.com/api/aquarius/assets/
--header 'Content-Type: application/json' \
--data-raw '{
    "query": {
        "match_all": {}
    }
}'
```

- Responses
  - 200
    - content-type: json
  - 500
    - description: elasticsearch exception

POST `/api/aquarius/assets/ddo/validate`

- Description
  Validate DDO content. Cosumes `application/octet-stream`
- Example

```
curl --location --request POST 'https://v4.aquarius.oceanprotocol.com/api/aquarius/assets/
--header 'Content-Type: application/json' \
--data-raw '<json_body>'
```

- Valid body

```
    {
        "@context": ["https://w3id.org/did/v1"],
        "id": "did:op:56c3d0ac76c02cc5cec98993be2b23c8a681800c08f2ff77d40c895907517280",
        "version": "4.1.0",
        "chainId": 1337,
        "nftAddress": "0xabc",
        "metadata": {
            "created": "2000-10-31T01:30:00.000-05:00Z",
            "updated": "2000-10-31T01:30:00.000-05:00",
            "name": "Ocean protocol white paper",
            "type": "dataset",
            "description": "Ocean protocol white paper -- description",
            "author": "Ocean Protocol Foundation Ltd.",
            "license": "CC-BY",
            "contentLanguage": "en-US",
            "tags": ["white-papers"],
            "additionalInformation": {"test-key": "test-value"},
            "links": [
                "http://data.ceda.ac.uk/badc/ukcp09/data/gridded-land-obs/gridded-land-obs
                "http://data.ceda.ac.uk/badc/ukcp09/data/gridded-land-obs/gridded-land-obs
                "http://data.ceda.ac.uk/badc/ukcp09/"
            ]
        },
        "services": [
            {
                "id": "test",
                "type": "access",
                "datatokenAddress": "0xC7EC1970B09224B317c52d92f37F5e1E4fF6B687",
                "name": "Download service",
                "description": "Download service",
                "serviceEndpoint": "http://172.15.0.4:8030/",
                "timeout": 0,
                "files": "encryptedFiles"
            }
        ]
    }
```

- Responses:
  - 200
    - description: successfully request.
  - 400
    -

description: Invalid DDO format

- 500
  - description: Error

POST `/api/aquarius/assets/triggerCaching`

- Description
  Manually triggers DDO caching based on a transacionId containing either MetadataCreated or MetadataUpdated event(s).
- Parameters

name	description	type	in	required
`transactionId`	DID of the asset	string	path	true
`logIndex`	custom log index for the transaction	int	path	false

- Example

```
curl --location --request POST 'https://v4.aquarius.oceanprotocol.com/api/aquarius/assets/
--header 'Content-Type: application/json' \
--data-raw '<json_body>'
```

- Valid body

```
    {
        "transactionId": "0x945596edf2a26d127514a78ed94fea86b199e68e9bed8b6f6d6c8bb24e451f
        "logIndex": 0
    }
```

- Responses:
  - 200
    - description: triggering successful, updated asset returned
  - 400
    - description: request issues: either log index not found, or neither of MetadataCreated, MetadataUpdated found in tx log
  - 500
    - description: Error

Chains

GET `/api/aquarius/chains/list`

- Description
  Get chains list
- Example

```
curl --location --request GET 'https://v4.aquarius.oceanprotocol.com/api/aquarius/chains/l
```

- Response
  - 200
    - Description: Successful request
    - Body

```
{   "246": true, "3": true, "137": true,
    "2021000": true, "4": true, "1": true,
    "56": true, "80001": true, "1287": true
}
```

GET `/api/aquarius/chains/status/{chain_id}`

- Description
  Get index status for a specific chain_id
- Example

```
curl --location --request GET 'https://v4.aquarius.oceanprotocol.com/api/aquarius/chains/s
```

- Response
  - 200
    - Description: Successful request
    - Body

```
{"last_block": 25198729}
```

Others

GET `/`

- Description
  Get version, plugin, and software information.
- Example

```
curl --location --request GET 'https://v4.aquarius.oceanprotocol.com/'
```

- Response
  - 200
    - Description: Successful request
    - Body

```
{
    "plugin": "elasticsearch",
    "software": "Aquarius",
    "version": "4.2.0"
}
```

GET `/health`

- Description
  Get health status
- Example

```
curl --location --request GET 'https://v4.aquarius.oceanprotocol.com/health'
```

- Response
  - 200
    - Description: Successful request
    - Body

```
Elasticsearch connected
```

GET /spec

- Description
  Get swagger spec
- Example

```
curl --location --request GET 'https://v4.aquarius.oceanprotocol.com/spec'
```

- Response
  - 200
    - Description: Successful request

Postman documentation

Click here to explore the documentation and more examples in postman.

Provider REST API

Ocean Provider Endpoints Specification

This document specifies the endpoints for Ocean Provider to be implemented by the core developers.

If you want to see the provider URLs for our supported networks, kindly check for `Provider` component on this page.

For inspecting the errors received from `Provider` and their reasons, please revise this document.

nonce endpoint

GET /api/services/nonce

Parameters

```
    userAddress: String object containing a user's ethereum address
```

Returns: Json object containing the last-used nonce value. The nonce endpoint is just informative, use the current UTC timestamp as a nonce, where required in other endpoints.

Example:

```
POST /api/services/nonce?userAddress=0x990922334
```

Response:

```
{
  "nonce": 23
}
```

Encrypt endpoint

GET /api/services/encrypt

Body: binary application/octet-stream

Returns: Bytes string containing the encrypted document.

Example:

```
POST /api/services/encrypt
body: b'\xfd7zXZ\x00\x00\x04\xe6\xd6\xb4F\ ... \x00\x04YZ'
```

Response:

```
b'0x04b2bfab1f4e...7ed0573'
```

Decrypt endpoint

POST /api/services/decrypt

Parameters

```
    decrypterAddress: String object containing the address of the decrypter (required)
    chainId: the chain id of the network the document is on (required)
    transactionId: the transaction id of the encrypted document (optional)
    dataNftAddress: the address of the data nft (optional)
    encryptedDocument: the encrypted document (optional)
    flags: the flags of the encrypted document (optional)
    documentHash: the hash of the encrypted document (optional)
    nonce: the nonce of the encrypted document (required)
    signature: the signature of the encrypted document (required)
```

Returns: Bytes string containing the decrypted document.

Example:

```
POST /api/services/decrypt
payload: {
    'decrypterAddress':'0xA78deb2Fa79463945C247991075E2a0e98Ba7A09'
    'chainId':8996
    'dataNftAddress':'0xBD558814eE914800EbfeF4a1cbE196F5161823d9'
    'encryptedDocument':'0xfd377a585a0...f07afef7dc214'
    'flags': 1
    'documentHash':'0x0cb38a7bba49758a86f8556642aff655d00e41da28240d5ea0f596b74094d91f'
    'nonce':'1644315615.24195'
    'signature':'0xd6f27047853203824ab9e5acef87d0a501a64aee93f33a83b6f91cbe8fb4489824defceacc
}
```

Response:

```
b'{"@context": ["https://w3id.org/did/v1"], "id": "did:op:0c184915b07b44c888d468be85a9b28253e8
```

File info endpoint

POST /api/services/fileinfo

Retrieves Content-Type and Content-Length from the given URL or asset.

Parameters

For published assets:

```
{
    did: String, DID of the dataset
    serviceId: String, ID of the service
}
```

For file objects,see https://docs.oceanprotocol.com/core-concepts/did-ddo#files

If checksum is requests, file size should be lower < MAX_CHECKSUM_LENGTH (see Provider ENVs) If file is larger, checksum WILL NOT be computed.

Returns: Json document file info object

Example:

```
POST /api/services/fileinfo
payload:
{
    "did":"0x1111",
    "serviceId": "0",
}
```

Response:

```
[
    {
        "contentLength":"1161",
        "contentType":"application/json",
        "index":0,
        "valid": true
    },...
]
```

Initial service request endpoint

GET /api/services/initialize

Parameters

```
    documentId: String object containing document id (e.g. a DID)
    serviceId: String, ID of the service the datatoken is attached to
    consumerAddress: String object containing consumer's address
    environment: String representing a compute environment offered by the provider
    validUntil: Integer, date of validity of the service (optional)
    fileIndex: Integer, the index of the file from the files list in the dataset. If set, prov
```

Returns: Json document with a quote for amount of tokens to transfer to the provider account.

Example:

```
GET /api/services/initialize
payload:
{
    "documentId":"0x1111",
    "serviceId": 0,
    "consumerAddress":"0x990922334",
}
payload (with optional parameters):
{
    "documentId":"0x1111",
    "serviceId": 0,
    "consumerAddress":"0x990922334",
    "validUntil": 1578004800,
    "fileIndex": 1
}
```

Response:

```
{
    "datatoken": "0x21fa3ea32892091...",
    "nonce": 23,
    "providerFee": {
        "providerFeeAddress": "0xabc123...",
        "providerFeeToken": "0xabc123...",
        "providerFeeAmount": "200",
        "providerData": "0xabc123...",
        "v": 27,
        "r": "0xabc123...",
        "s": "0xabc123...",
        "validUntil": 123456,
    },
    "computeAddress": "0x8123jdf8sdsa..."
}
```

Download endpoint

GET /api/services/download

Parameters

```
    documentId: String object containing document id (e.g. a DID)
    serviceId: String, representing the list of `file` objects that describe each file in the
    transferTxId: Hex string -- the id of on-chain transaction for approval of datatokens tra
    given to the provider's account
    fileIndex: integer, the index of the file from the files list in the dataset
    nonce: Nonce
    consumerAddress: String object containing consumer's address
    signature: String object containg user signature (signed message)
```

Returns: File stream. Retrieves the attached asset files.

Example:

```
GET /api/services/download
payload:
{
    "documentId":"0x1111",
    "serviceId": 0,
    "fileIndex": 0,
    "datatoken": "",
    "consumerAddress":"0x990922334",
    "signature":"0x00110011",
    "transferTxId": "0xa09fc23421345532e34829"
```

Response:

```
{
  "": ""
}
```

Compute endpoints

All compute endpoints respond with an Array of status objects, each object describing a compute job info.

Each status object will contain:

```
    owner:The owner of this compute job
    documentId: String object containing document id (e.g. a DID)
    jobId: String object containing workflowId
    dateCreated: Unix timestamp of job creation
    dateFinished: Unix timestamp when job finished (null if job not finished)
    status:  Int, see below for list
    statusText: String, see below
    algorithmLogUrl: URL to get the algo log (for user)
    resultsUrls: Array of URLs for algo outputs
    resultsDid: If published, the DID
```

Status description (`statusText`): (see Operator-Service for full status list)

status	Description
1	Warming up
10	Job started
20	Configuring volumes
30	Provisioning success
31	Data provisioning failed
32	Algorithm provisioning failed
40	Running algorith
50	Filtering results
60	Publishing results
70	Job completed

Create new job or restart an existing stopped job

POST /api/services/compute

Start a new job

Parameters

```
    signature: String object containg user signature (signed message) (required)
    consumerAddress: String object containing consumer's ethereum address (required)
    nonce: Integer, Nonce (required)
    environment: String representing a compute environment offered by the provider
    dataset: Json object containing dataset information
        dataset.documentId: String, object containing document id (e.g. a DID) (required)
        dataset.serviceId: String, ID of the service the datatoken is attached to (required)
        dataset.transferTxId: Hex string, the id of on-chain transaction for approval of datat
            given to the provider's account (required)
        dataset.userdata: Json, user-defined parameters passed to the dataset service (optiona
    algorithm: Json object, containing algorithm information
        algorithm.documentId: Hex string, the did of the algorithm to be executed (optional)
        algorithm.meta: Json object, defines the algorithm attributes and url or raw code (opt
        algorithm.serviceId: String, ID of the service to use to process the algorithm (optior
        algorithm.transferTxId: Hex string, the id of on-chain transaction of the order to use
        algorithm.userdata: Json, user-defined parameters passed to the algorithm running serv
        algorithm.algocustomdata: Json object, algorithm custom parameters (optional)
    additionalDatasets: Json object containing a list of dataset objects (optional)

    One of `algorithm.documentId` or `algorithm.meta` is required, `algorithm.meta` takes prec
```

Returns: Array of `status` objects as described above, in this case the array will have only one object

Example:

```
POST /api/compute
payload:
{
    "signature": "0x00110011",
    "consumerAddress": "0x123abc",
    "nonce": 1,
    "environment": "env",
    "dataset": {
        "documentId": "did:op:2222...",
        "serviceId": "compute",
        "transferTxId": "0x0232123..."
    }
}
```

Response:

```
[
    {
      "jobId": "0x1111:001",
      "status": 1,
      "statusText": "Job started",
      ...
    }
]
```

Status and Result

GET /api/services/compute

Get all jobs and corresponding stats

Parameters

```
    signature: String object containg user signature (signed message)
    documentId: String object containing document did  (optional)
    jobId: String object containing workflowID (optional)
    consumerAddress: String object containing consumer's address (optional)

    At least one parameter from documentId, jobId and owner is required (can be any of them)
```

Returns

Array of `status` objects as described above

Example:

```
GET /api/services/compute?signature=0x00110011&documentId=did:op:1111&jobId=012023
```

Response:

```
[
  {
    "owner": "0x1111",
    "documentId": "did:op:2222",
    "jobId": "3333",
    "dateCreated": "2020-10-01T01:00:00Z",
    "dateFinished": "2020-10-01T01:00:00Z",
    "status": 5,
    "statusText": "Job finished",
    "algorithmLogUrl": "http://example.net/logs/algo.log",
    "resultsUrls": [
      "http://example.net/logs/output/0",
      "http://example.net/logs/output/1"
    ],
    "resultsDid": "did:op:87bdaabb33354d2eb014af5091c604fb4b0f67dc6cca4d18a96547bffdc27bcf"
  },
  {
    "owner": "0x1111",
    "documentId": "did:op:2222",
    "jobId": "3334",
    "dateCreated": "2020-10-01T01:00:00Z",
    "dateFinished": "2020-10-01T01:00:00Z",
    "status": 5,
    "statusText": "Job finished",
    "algorithmLogUrl": "http://example.net/logs2/algo.log",

    "resultsUrls": [
      "http://example.net/logs2/output/0",
      "http://example.net/logs2/output/1"
    ],
    "resultsDid": ""
  }
]
```


GET /api/services/computeResult

Allows download of asset data file.

Parameters

```
    jobId: String object containing workflowId (optional)
    index: Integer, index of the result to download (optional)
    consumerAddress: String object containing consumer's address (optional)
    nonce: Integer, Nonce (required)
    signature: String object containg user signature (signed message)
```

Returns: Bytes string containing the compute result.

Example:

```
GET /api/services/computeResult?index=0&consumerAddress=0xA78deb2Fa79463945C247991075E2a0e98Ba
```

Response:

```
b'{"result": "0x0000000000000000000000000000000000000000000000000000000000000001"}'
```

Stop

PUT /api/services/compute

Stop a running compute job.

Parameters

```
    signature: String object containg user signature (signed message)
    documentId: String object containing document did (optional)
    jobId: String object containing workflowID (optional)
    consumerAddress: String object containing consumer's address (optional)

    At least one parameter from documentId,jobId and owner is required (can be any of them)
```

Returns

Array of `status` objects as described above

Example:

```
PUT /api/services/compute?signature=0x00110011&documentId=did:op:1111&jobId=012023
```

Response:

```
[
    {
      ...,
      "status": 7,
      "statusText": "Job stopped",
      ...
    }
]
```

Delete

DELETE /api/services/compute

Delete a compute job and all resources associated with the job. If job is running it will be stopped first.

Parameters

```
    signature: String object containg user signature (signed message)
    documentId: String object containing document did (optional)
    jobId: String object containing workflowId (optional)
    consumerAddress: String object containing consumer's address (optional)

    At least one parameter from documentId, jobId is required (can be any of them)
    in addition to consumerAddress and signature
```

Returns

Array of `status` objects as described above

Example:

```
DELETE /api/services/compute?signature=0x00110011&documentId=did:op:1111&jobId=012023
```

Response:

```
[
    {
      ...,
      "status": 8,
      "statusText": "Job deleted successfully",
      ...
    }
]
```

GET /api/services/computeEnvironments

Allows download of asset data file.

Parameters

```

```

Returns: List of compute environments.

Example:

```
GET /api/services/computeEnvironments
```

Response:

```
[
    {
        "cpuType":"AMD Ryzen 7 5800X 8-Core Processor",
        "currentJobs":0,
        "desc":"This is a mocked environment",
        "diskGB":2,
        "gpuType":"AMD RX570",
        "id":"ocean-compute",
        "maxJobs":10,
        "nCPU":2,
        "nGPU":0,
        "priceMin":2.3,
        "ramGB":1
    },
    ...
]
```


Authentication endpoints

Provider offers an alternative to signing each request, by allowing users to generate auth tokens. The generated auth token can be used until its expiration in all supported requests. Simply omit the signature parameter and add the AuthToken request header based on a created token.

Please note that if a signature parameter exists, it will take precedence over the AuthToken headers. All routes that support a signature parameter support the replacement, with the exception of auth-related ones (createAuthToken and deleteAuthToken need to be signed).

GET /api/services/createAuthToken

Allows the user to create an auth token.

Parameters

```
address: String object containing consumer's address (optional)
nonce: Integer, Nonce (required)
signature: String object containg user signature (signed message)
 The signature is based on hashing the following parameters:
   address + nonce
expiration: valid future UTC timestamp (required)
```

Returns: Created auth token.

Example:

```
GET /api/services/createAuthToken?address=<your_address>&&nonce=<your_nonce>&&expiration=<exp
```

Inside the angular brackets, the user should provide the valid values for the request.

Response:

```
{"token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE2NjAwNTMxMjksImFkZHJlc3MiOiIweEE3O
```

DELETE /api/services/deleteAuthToken

Allows the user to delete an existing auth token before it naturally expires.

Parameters

```
address: String object containing consumer's address (optional)
nonce: Integer, Nonce (required)
signature: String object containg user signature (signed message)
  The signature is based on hashing the following parameters:
  address + nonce
token: token to be expired
```

Returns: Success message if token is successfully deleted. If the token is not found or already expired, returns an error message.

Example:

```
DELETE /api/services/deleteAuthToken?address=<your_address>&&nonce=<your_nonce>&&token=<your_
```

Inside the angular brackets, the user should provide the valid values for the request.

Response:

```
{"success": "Token has been deactivated."}
```

FAQ

Frequently Asked Questions about Ocean Protocol

Have some questions about Ocean Protocol?

Hopefully you'll find the answers here! If not then please don't hesitate to reach out to us on discord - there are no stupid questions!

General Questions

What is a decentralized data marketplace?

A data marketplace allows providers to publish data and buyers to consume data. Unlike centralized data marketplaces, decentralized ones give users more control over their data, algorithms and analytics by minimizing custodianship and providing transparent and immutable records of every transaction. With features such as Compute-to-Data (C2D), data and algorithms can be ingested into secure Docker containers where escapes avoided, protecting both the data and algorithms.

What is needed to use a decentralized marketplace?

Users access decentralized marketplaces via Metamask. Metamask is an applet interface that manages unique IDs, generated and controlled fully by the user. These unique IDs (aka Ethereum address) are used to store digital assets such as cryptocurrency, datatokens, NFTs and other web3 native assets.

A Metamask wallet can easily be set up as a browser extension by going to your browser’s web store for extensions and search for “MetaMask”. For additional help setting up your MetaMask wallet, watch our short tutorial video and review these instructions on Ocean’s documentation page.

Once a user has Metamask installed and an Ethereum address, they can register, consume or stake on datasets on Ocean Market.

How is Ocean different from other data marketplaces?

Ocean Protocol is a decentralized data marketplace which gives users complete control of their data. The Ocean Protocol technology is built on smart contracts, decentralized computer scripts with no intermediary that are triggered by the users. The Ocean Market exposes the functionality of the smart contracts in a browser-friendly interface. Data providers and consumers can discover one another and transact in a peer-to-peer manner with the minimal amount of intermediary involvement.

How do I price my data?

Ocean gives you two different options for pricing your data - fixed price or free. You need to decide what your dataset is worth and how you want to price it. You can change the price but you can't change the price format (e.g. from fixed to free).

Is my data secure?

Yes. Ocean Protocol understands that some data is too sensitive to be shared — potentially due to GDPR or other reasons. For these types of datasets, we offer a unique service called compute-to-data. This enables you to monetise the dataset that sits behind a firewall without ever revealing the raw data to the consumer. For example, researchers and data scientists pay to run their algorithms on the data set and the computation is performed behind a firewall; all the researchers or data scientists receive is the results generated by their algorithm.

Where is my data stored?

Ocean does not provide data storage. Users have the choice to store their data on their own servers, cloud or decentralized storage. Users need only to provide a URL to the dataset, which is then encrypted as a means to protect the access to the dataset.

How do I control who accesses my data?

Ocean provides tools for access control, fine grained permissions, passlisting and blocklisting addresses. Data and AI services can be shared under the conditions set by the owner of data. There is no central intermediary, which ensures no one can interfere with the transaction and both the publisher and user have transparency.

Can I restrict who is able to access my dataset?

Yes - Ocean has implemented fine grained permissions. This means that you can create allow and deny lists that restrict access from certain individuals or limit access to particular organizations.

What is the reach of Ocean Market - how many data buyers can I sell to?

Hundreds of unique datasets are available that are sourced from private individuals, research institutions, commercial enterprises and government. Publishing data on Ocean offers data providers and algorithm owners an exciting new channel to connect with a rapidly growing community of Web3 enthusiasts and data science professionals around the world.

Technical Questions

Why does Ocean Protocol use the Blockchain?

For both providers and consumers of data, blockchain is a superior substrate for building applications.Blockchain allows business logic to be instantiated in a network and triggered by the users, without intermediaries. This innovation promises lower transaction costs, higher security, more control, less errors and more transparency & auditability.

The blockchain is public - does this mean that anyone can access my data?

No one is able to access data via the blockchain without purchasing access (with the datatoken) though the smart contract. Ocean smart contracts encrypt the URL to the dataset before it is published on the blockchain. This means that only the encrypted URL will be queryable in the public blockchain. Ocean technology facilitates data access to the consumer via a proxy (Ocean Provider) and the unencrypted url is never exposed.

What is a smart contract and why is it relevant?

The blockchain can do more than just store information - it can also run code. A smart contract is an executable script that runs on the blockchain, with no intermediary and is fully transparent and auditable by anyone. In Ocean, smart contracts facilitate access to data and AI if the access conditions set out by the publisher are fulfilled.

What is a datatoken?

A datatoken is an access token to datasets and services published in the Ocean ecosystem. Datatokens can be purchased via the Ocean Market or on a decentralized crypto exchange. . If a consumer wishes to access a dataset, they must acquire the datatoken and then exchange the datatoken for access to the dataset.

How do I acquire datatokens?

Datatokens can be acquired and traded in Ocean Market. There are several ways to acquire data tokens. Data publishers can acquire datatokens by publishing datasets and then receiving the generated datatokens.

Consumers can click "buy" on an asset in Ocean Market to buy and redeem a datatoken in exchange for access to a dataset.

Datatokens can also be sent from anyone who holds a datatoken for a particular asset.

Data Selling Questions

How are organizations leveraging data sharing?

For the most part organizations are leveraging data sharing to benefit from data monetization, however increasingly organizations are also sharing data in order to boost their progress on sustainability goals. For example, data aggregated from vehicles can not only bring new revenue streams to automotive firms but can also be used to battle pollution.

Does it pay to become a marketplace operator?

Yes. Marketplace operators benefit from earning commission on marketplace transactions related to data consumption. Ocean Market is primarily focussed on monetising data however it is also designed to handle the sale of any digital asset or service. As a result the total addressable market goes way beyond revenues from just selling data. Operating costs for an Ocean-powered marketplace are moderate and the base code is open source and available free of charge under the Apache 2 license.

Why Publish?

Publishing data, algorithms and other digital assets and services on an Ocean-powered marketplace offers numerous opportunities to earn on the future revenue streams connected to that data as well as build lucrative ecosystem that add value to the published asset. It also allows for the discovery and insights into new use cases and applications of the published asset.

What about the price fluctuation of Ocean?

Price fluctuation is mitigated through the use of the Ocean backed stable coin H2O.

Who pays for gas fees?

Gas fees for marketplace transactions are paid by the user initiating the transaction (for publishing, consuming, etc).

Where do the docker containers run?

Dockers containers can run anywhere. Ocean Market use a docker run by the Ocean Protocol Foundation OPF); limit: 1 CPU limit / 60 seconds max. NOTE: This means OPF technically has access to data. In the case of a forked Ocean-powered marketplace the owner of marketplace must set up computation environment. If individual users of the marketplace are concerned with security they should be prepared to host both the data and provide compute-to-data services on premise.

Who pays for the computation?

The marketplace owner.

What cryptocurrency do I need for transactions?

The type if cryptocurrencies needed for transactions on the marketplace depends on which network(s) the marketplace is running (Ethereum, Polygon, EWT, BSC, Moonriver, etc.). Regardless of network, users will need to have Ocean tokens as well as the corresponding network token, which is used to pay for gas.

Can I use the off the shelf CSS available in the repo?

Marketplace name, logo and typeface must be changed by the client. Slight modification would be enough for compliance. For more information consult the READ ME file on GitHub. https://github.com/oceanprotocol/market#-forking

What's to come with Ocean this year?

Checkout our roadmap to see what's we are currently working on. If you are interested in tracking our progress towards these goals then take a look at our github.

Data Farming FAQ

What assets are eligible for Data Farming?

The data asset may be of any type — dataset (for static URIs), algorithm for Compute-to-Data, or any other Datatoken token-gated system. The data asset may be fixed price or free price. You can find more details in the DF Background page

When exactly does counting start and finish, for a given week?

The counting starts at 12.01am on Thursday, and ends at 11.59pm on the following Wednesday.

I staked for just one day. What rewards might I expect?

At least 50 snapshots are randomly taken throughout the week. If you've staked just one day, and all else being equal, you should expect 1/7 the rewards compared to the full 7 days.

The datatoken price may change throughout the week. What price is taken in the DCV calculation?

The price is taken at the same time as each consume. E.g. if a data asset has three consumes, where price was 1 OCEAN when the first consume happened, and the price was 10 OCEAN when the other consumes happened, then the total DCV for the asset is 1 + 10 + 10 = 21.

Can the reward function change during a given week?

No. At the beginning of a new DF round (DF1, DF2, etc), rules are laid out, either implicitly if no change from previous round, or explicitly in a blog post if there are new rules. This is: reward function, bounds, etc. Then teams stake, buy data, consume, etc. And LPs are given DF rewards based on staking, DCV, etc at the end of the week. Overall cycle time is one week.

Caveat: it's no at least in theory! Sometimes there may be tweaks if there is community consensus, or a bug.

veOCEAN FAQ

What is the official formula for the Linear Decay?

The Linear Decay formula for veOCEAN can be expressed as follows in python.

```
FOUR_YEARS = 60 * 60 * 24 * 7 * 52

veOcean_balance = OCEAN_amount_locked * (your_unlock_timestamp – current_unix_timesta
```

To learn more about systems driving veOCEAN and Data Farming, please visit our df-py github repository.

Staking FAQs

What about passive stakers — people who just want to stake in one place and be done?

Earnings are passive by default

What about active stakers — people who want to do extra work and get rewarded?

Ot works. Half the DF revenue goes to veOCEAN stake that users can allocate. Allocate well → more $$

Pricing FAQs

In this scheme, can people stake on fixed-price datasets?

Yes. They allocate their veOCEAN to datasets. Then DF rewards follow the usual DF formula: DCV * veOCEAN stake.

In this scheme, can people stake on free datasets?

Yes. They allocate their veOCEAN to datasets. Then DF rewards follow the usual DF formula: DCV * veOCEAN stake. Except in this case although DCV is 0, the gas fees will still count towards calculating rewards.

Does this work for other pricing schemes?

Yes, from the get-go! It doesn't matter how data is priced, this works for all schemes.

Chains FAQ

Which chain is veOCEAN be deployed on?

veOCEAN & DF core contracts are deployed on Ethereum mainnet and allow users to allocate veOCEAN tokens to any asset, on any chain.

Which networks are eligible for Data Farming?

Data assets for DF may published in any network where Ocean's deployed in production: Eth Mainnet, Polygon, BSC, and more.

You can find a list of all supported chains here.

Where can I find the veOCEAN and DF contracts?

They are deployed on Ethereum mainnet, alongside other Ocean contract deployments. You can find the full list of contracts here.

What is the official veOCEAN epoch start_time?

veFeeDistributor has a start_time of 1663804800 (Thu Sep 22 2022 00:00:00)

Will the Market still need to be multi-chain?

Yes, Ocean Market still needs to be multi-chain: all the reasons that we went multi-chain for are as valid as ever.

Which chain supports Fixed Price Assets?

You can publish Fixed Price Assets to any chain that Ocean supports.

Community