Create a Collection

Learn how to create a Myria collection.

> **INFO**
>
> Please note, in the `Staging` environment you can create five collections and 50,000 mint transactions per collection per month. If you want to create collections for your project on the `Production` environment, or you need to increase those limits, please contact our team. Be sure to include your project id in the request message.

Prerequisites

- Basic Javascript and Typescript knowledge
- Web IDE such as VS Code
- Latest NodeJs
- Web3 wallet address and Stark Key
- Typescript project created here or a new one with similar structure
- Registered Myria developer account and project

1. Deploy a contract

First, deploy an Ethereum smart contract that implements the `mintFor` function. If the deployment is successfull, you will get the associated contract address, which you will need to you create a new collection.

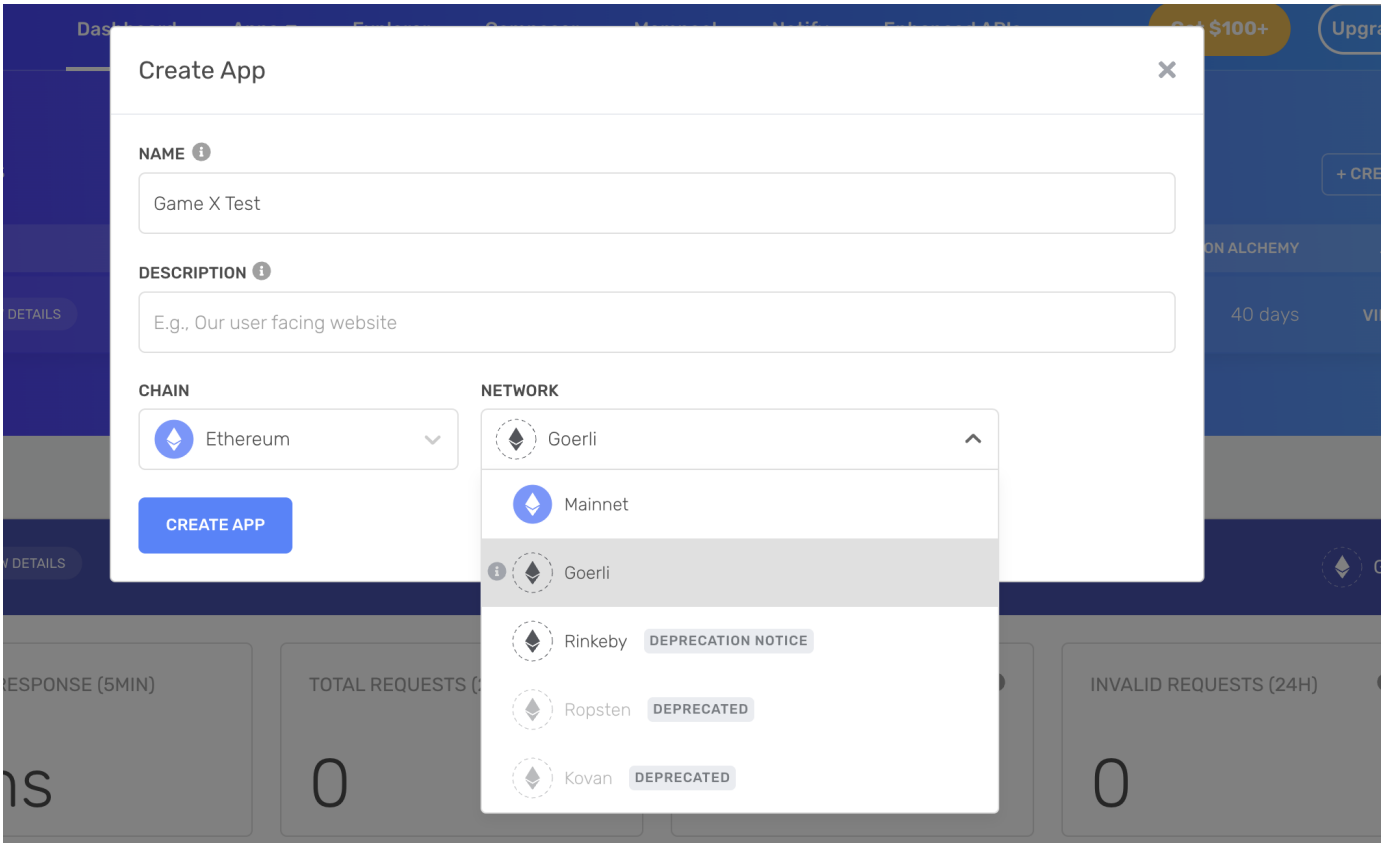
> **TIP**
>
> There's no need to write any Solidity code. The example provided below contains everything required to deploy your contract.

1.1 Get the Alchemy key

Web3 developers consume data via RPC calls from one of the nodes on the network. Hosting nodes is expensive, that's why it's common to use third-party services like Alchemy or Infura.

This guide will use Alchemy as example. Once you set up log into your account, create a new app and choose the network to work with as follows:

Goerli Testnet **Ethereum Mainnet**



After you set up the app, reveal your Alchemy key by selecting `VIEW KEY` near the name of your app. Copy and save the `HTTPS` key. You'll need to use it in further steps.

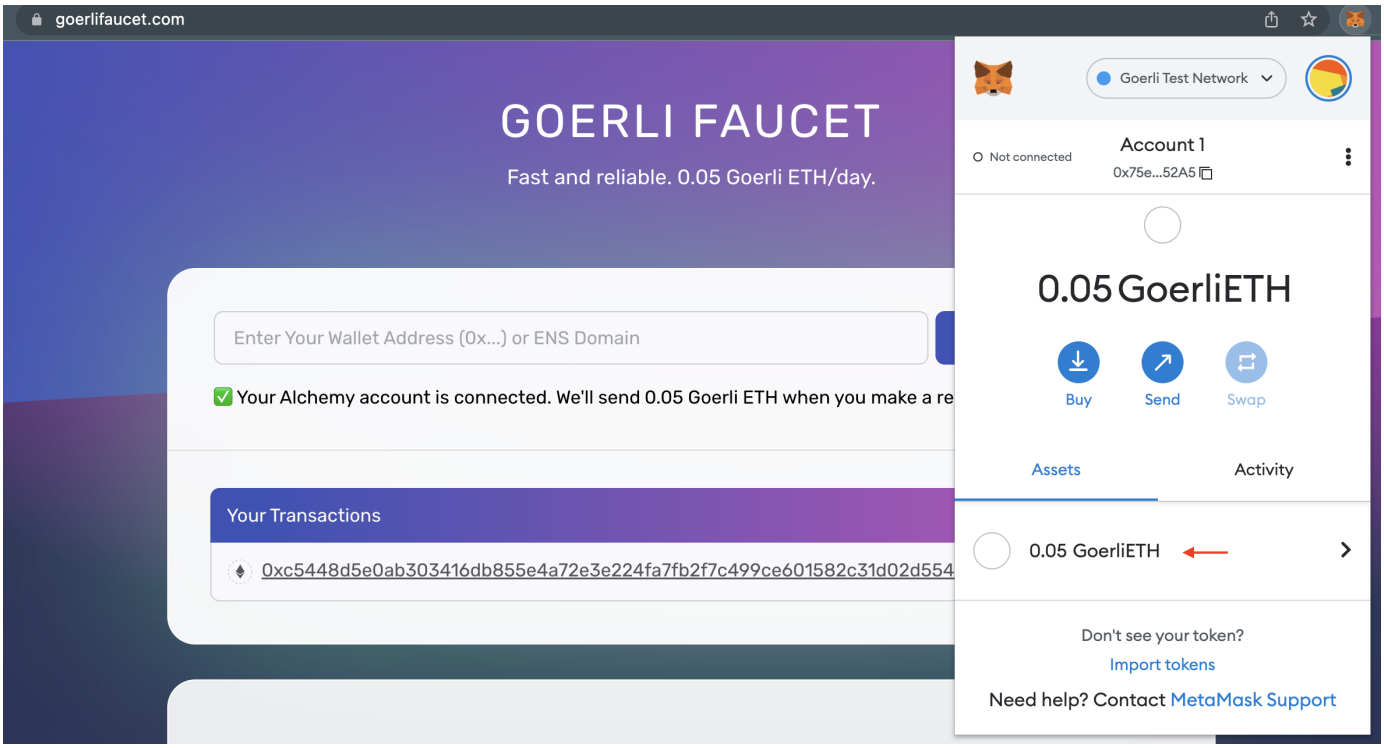
1.2 Fund your Web wallet

Code deployments to the Ethereum blockchain require ETH to pay assciated fees.

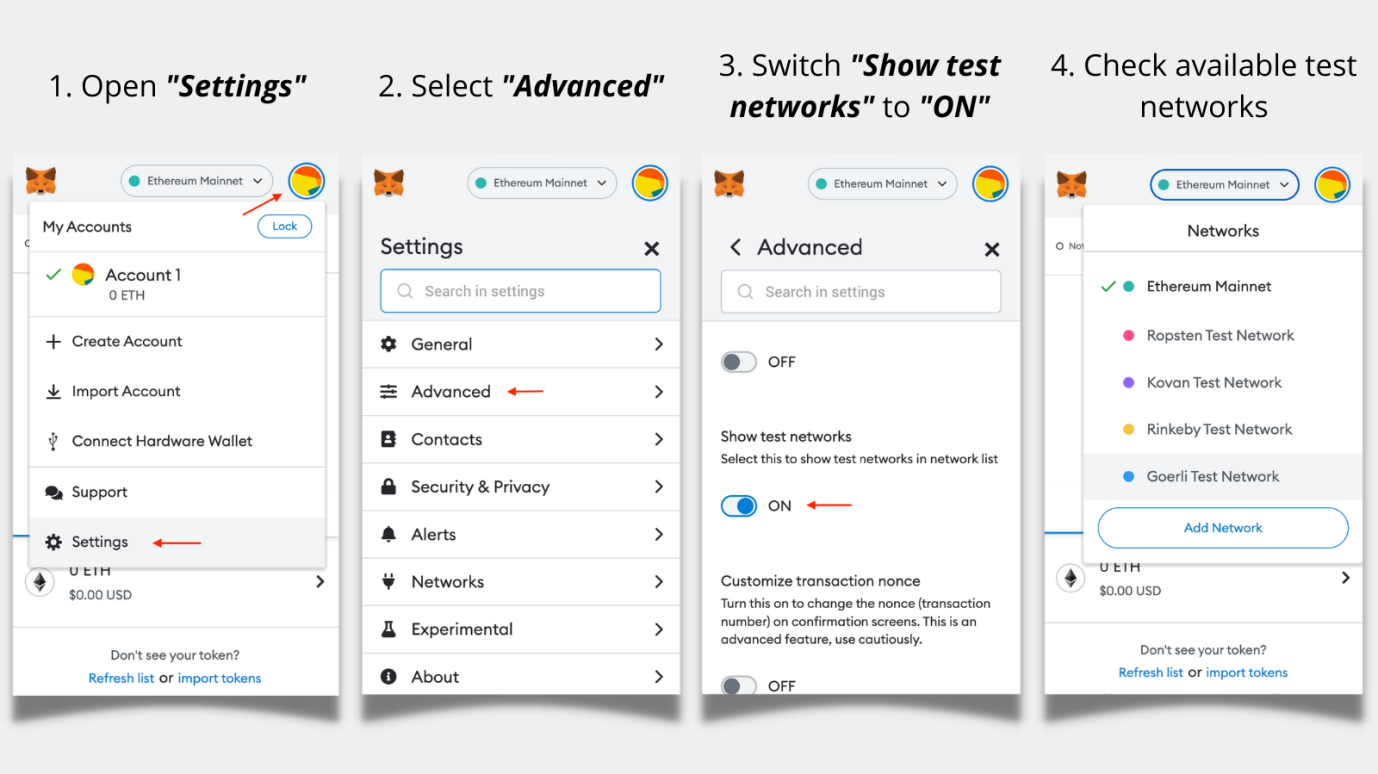
Goerli Testnet **Ethereum Mainnet**

The example below uses Goerli and requires test ETH on the Goerli testnet. You can request test funds from Goerli faucet by Alchemy.

To request a Goerli faucet, switch to the Goerli network in your MetaMask, copy and paste your wallet address and click `Send me ETH`. As a result, you should see tokens in your wallet as follows:



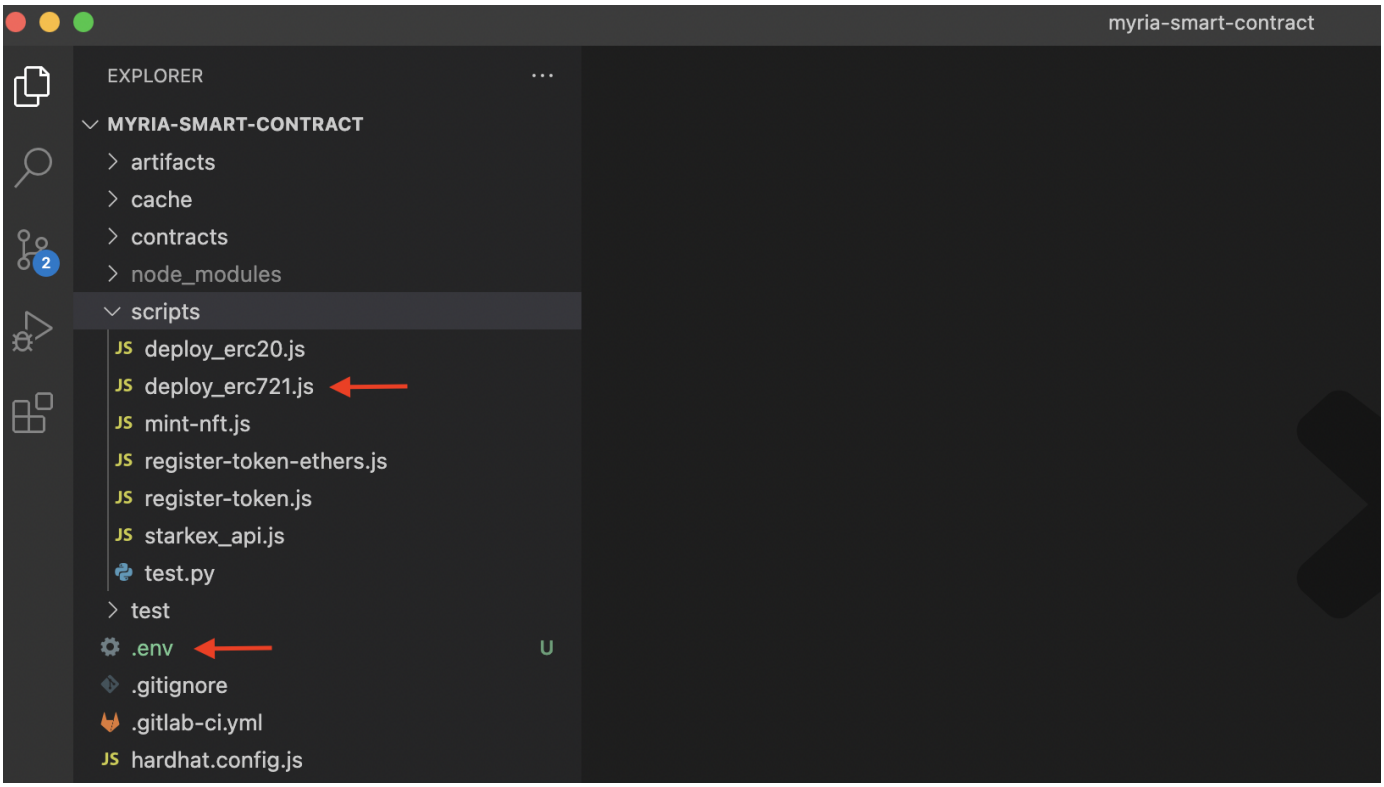
When you install MetaMask the first time, it may have test networks disabled. To enable those, use the following steps:



1.3 Clone the repository

```
git clone https://github.com/MyriaPlatform/myria-smart-contract.git
```

1.4 Open your contract in Visual Studio Code



1.5 Setup the project

To set up the project, first install the dependencies:

```
yarn
```

Then, rename the `.env.example` into `.env` and change the variables as follows:

- `API_URL` - the RPC URL on the corresponding network (testnet or mainnet)
- `PRIVATE_KEY` - the private key of your Web3 wallet
- `PUBLIC_KEY` - the public key of your Web3 wallet
- `STARKEX_ADDRESS` - keep the existing value

1.6 Deploy the contract

Finally, run the `deploy.js` script to deploy the contract:

```
yarn deploy-erc721
```

1.7 Get your contract address

If the deployment is successful, you will get the contract address and transaction hash to check transaction details. Copy and save this address somewhere. You'll need to use it later.

If you want more details about the contract, wallet, or blockchain transactions, you can use one of the blockchain explorers such as Etherscan.

Here's an example of deployed contract address on the Goerli testnet: 0x5845f81c315cdbb86747c44abd68454638e7520f.

Transaction Details

Overview Logs (2) State

[This is a Goerli **Testnet** transaction only]

Transaction Hash:	0x4ad0dbd48a62902bb497cbac58825f9dda1d9754fa28b52b6d2078b638d3d6c4
Status:	Success
Block:	7221636 25810 Block Confirmations
Timestamp:	4 days 12 hrs ago (Jul-13-2022 09:24:26 PM +UTC)
From:	0x75e0d99fa416823f4c07cdafc86a3b0ca37b52a5
To:	[Contract 0x5845f81c315cdbb86747c44abd68454638e7520f Created]
Value:	0 Ether ($0.00)
Transaction Fee:	0.006356716050853728 Ether ($0.00)
Gas Price:	0.000000002000000016 Ether (2.000000016 Gwei)

2. Create the Metadata API URL

Each collection requires a metadataApiUrl that defines the metadata schema for containing assets. The URL should return a JSON object and have a predefined structure. There are three ways to create your `metadataApiUrl`. The below steps show Pinata as an example.

2.1 Create a Pinata account

You can create an account through the official Pinata website.

2.2 Upload media files

After you create the account, upload your media files to Pinata. When preparing media files for your collection, consider these rules:

1. If you have all the media files ready, upload the entire folder with your files. Pinata will generate a link containing those files for you. Note that you won't be able to add or remove items from that folder so use this option only if your collection will remain the same. Here's an example:

Index of /ipfs/QmNkE2c8ECx3aFhfhWorSyzx2N7m5E3FU1vMowp3XkEAnP
QmNkE2c8ECx3aFhfhWorSyzx2N7m5E3FU1vMowp3XkEAnP

..	
.DS_Store	QmUR...wZXH
01.png	Qmae...MLPt
02.png	QmRY...PhAs
03.png	Qmej...dKzj
04.png	QmWU...41Qx
05.png	QmP1...dvwn

2. If you don't have all your media files ready or you plan to expand your collection gradually, then upload individual files to the root Pinata folder as follows:

03.png 7/12/2022 484 B	QmejRSJuiNiQmM74uzow11z3aBX2cQHDTYWSp7SLsgdKzj	False	More
02.png 7/12/2022 2.34 KB	QmRYwBhvzii9HSND82p5sLv4R3fzmQiyhaBs42uTB4PhAs	False	More
01.png 7/12/2022 2.17 KB	Qmae3dbyXos21g7oSeK3g4C7GyhSV8pxXrkWFmekgaMLPt	False	More

Files or folders in Pinata are identified by a unique `CID` and have the following URL structure:

```
https://gateway.pinata.cloud/ipfs/CID
```

For example, `https://gateway.pinata.cloud/ipfs/Qmae3dbyXos21g7oSeK3g4C7GyhSV8pxXrkWFmekgaMLPt`.

2.3 Create metadata files

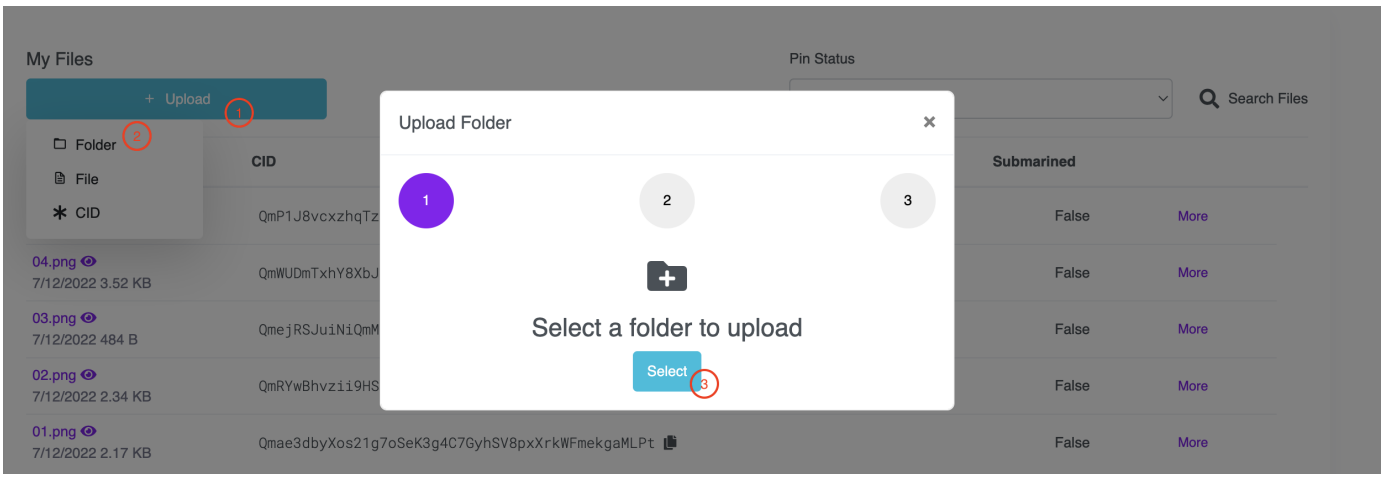
Next, create metadata files for all assets within a collection. Each file represents a JSON object that defines asset's metadata schema. Those files will give a unique identifier to each of the assets, also known as Token ID. You can create files in any IDE as follows:

Make sure to follow these rules for creating the metadata files:

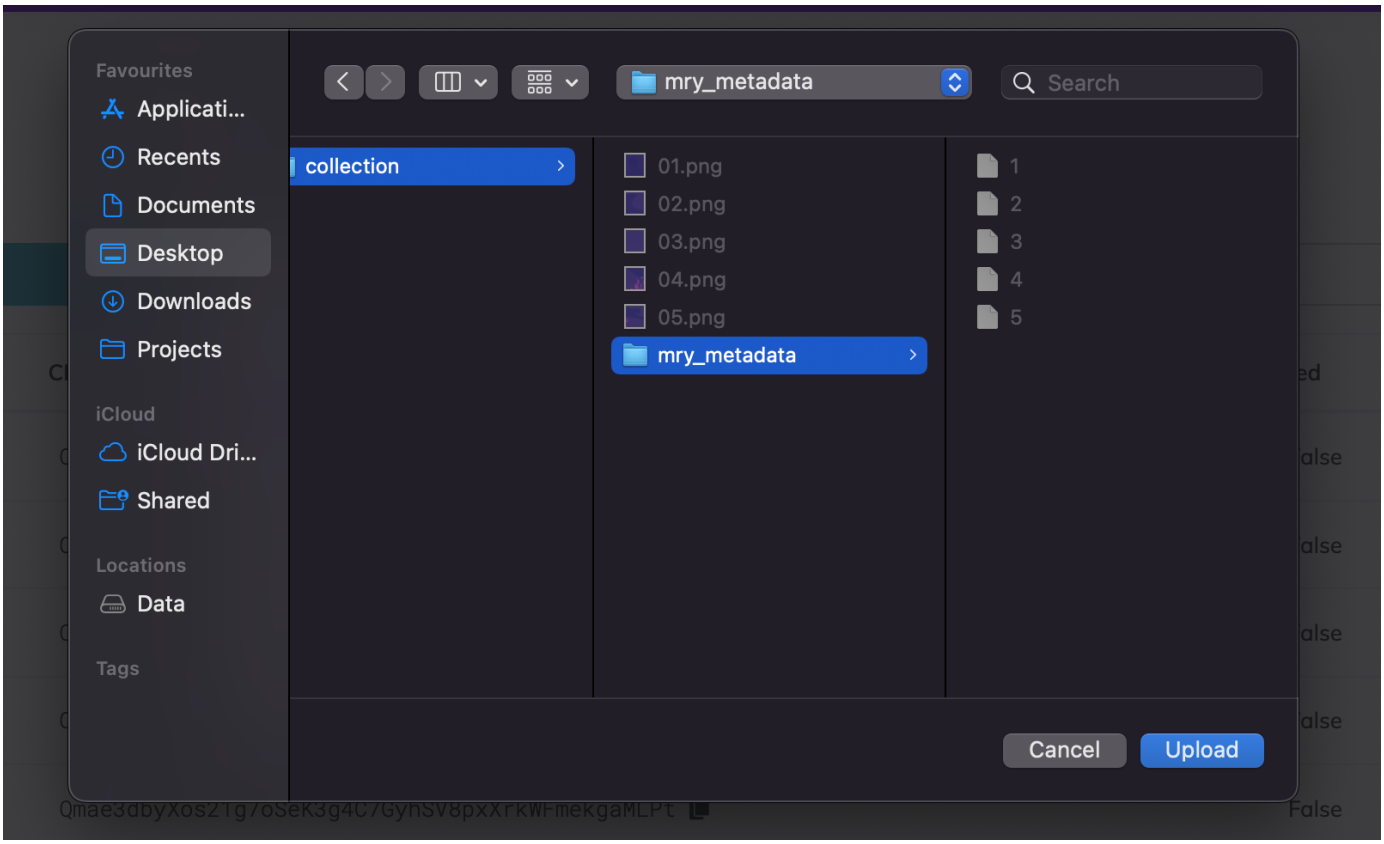
- File names should not have any extension and should have plain numbers as their names
- All files should follow the same metadata schema
- The `name` property should be unique for each file
- The `image` property should point to the corresponding image on Pinata
- The `external_url` property should point to the corresponding image URL of the collection's website

2.4 Upload metadata folder to Pinata

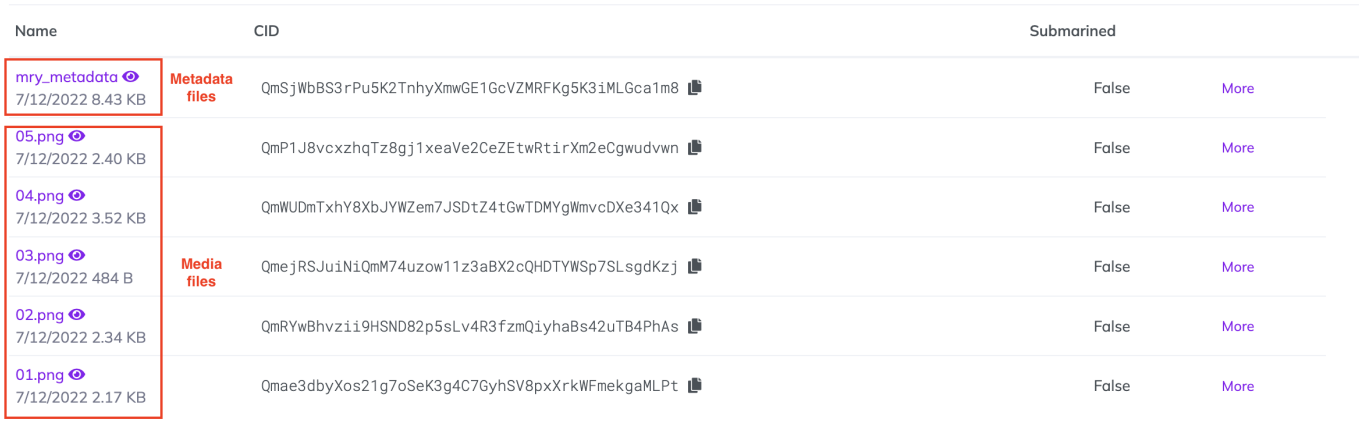
Once you have your metadata files ready, upload the entire folder to your Pinata account:



The contents of the metadata folder should have all of the files created earlier in VS Code:

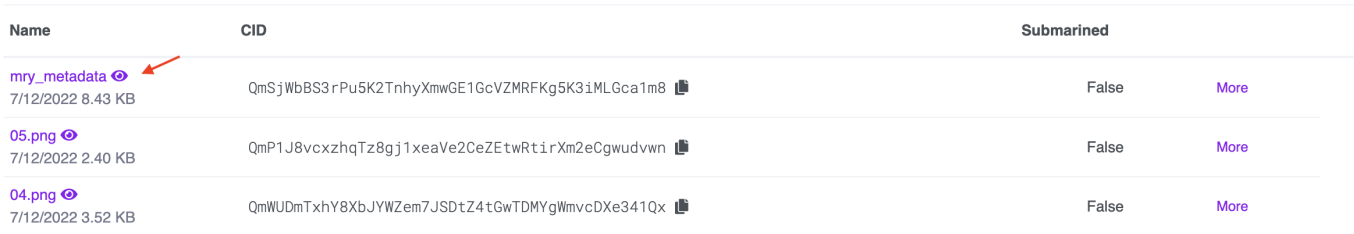


As a result, your Pinata account will have the structure that looks as follows:



2.5 Get your Metadata API URL

Finally, click the 👁 icon on your newly uploaded folder and copy the URL.



The copied link will be your `metadataApiUrl`. To verify that you set up everything correctly, open your browser at `metadataApiUrl` followed by a TOKEN_ID as a number. It should return

a JSON object that represents the metadata of that particular asset. The example below returns metadata of the first asset in a given collection:

gateway.pinata.cloud/ipfs/QmSjWbBS3rPu5K2TnhyXmwGE1GcVZMRFKg5K3iMLGca1m8/1

metadataApiUrl TOKEN_ID


```
{
  "description": "MT Original 1",
  "external_url": "",
  "image": "https://gateway.pinata.cloud/ipfs/Qmae3dbyXos21g7oSeK3g4C7GyhSV8pxXrkWFmekgaMLPt",
  "name": "mto1",
  "attributes": [
    {
      "trait_type": "Category",
      "value": "Silver"
    },
    {
      "trait_type": "Edition",
      "value": 1
    },
    {
      "trait_type": "Level",
      "value": 1
    }
  ]
}
```


3. Create a Collection

You can create a new Myria collection as follows:

1. Open a Typescript project created here

2. Create the `create-collection.ts` file in the above project and paste this code:

Typescript

```
import { CollectionManager, CreateCollectionParams,
CreateCollectionResponse, EnvTypes } from "myria-core-sdk";

(async (): Promise<void> => {
  // define the environment: STAGING or PRODUCTION
  const env = EnvTypes.STAGING;

  // get access to the `CollectionManager`
  const collectionManager: CollectionManager = new
CollectionManager(env);

  // define params
  const params: CreateCollectionParams = {
    name: "COLLECTION_NAME",
    description: "COLLECTION_DESCRIPTION",
```

```
    contractAddress: "CONTRACT_ADDRESS",
    metadataApiUrl: "METADATA_API_URL",
    ownerPublicKey: "OWNER_PUBLIC_KEY",
    projectId: "PROJECT_ID",
    starkKey: "STARK_KEY"
  };

  // create a collection
  const collectionResponse: CreateCollectionResponse | undefined =
    await collectionManager.createCollection(params);

  // log the result
  console.log(JSON.stringify(collectionResponse, null, 2));
})();
```

Replace the `params` values as follows:

- `COLLECTION_NAME` - collection name
- `COLLECTION_DESCRIPTION` - collection description
- `CONTRACT_ADDRESS` - contract address used to withdraw assets to the Ethereum network
- `METADATA_API_URL` - API URL that will store collection metadata
- `PROJECT_ID` - project id to which the collection will belong
- `STARK_KEY` - Stark Key, has to start with `0x`

3. Add a script to load the `create-collection.ts` file in `package.json`:

```
{
  "scripts": {
    "create-project": "ts-node create-project.ts",
    "create-collection": "ts-node create-collection.ts",
  },
}
```

4. Run the `create-collection` script:

```
npm run create-collection
```

After a collection is created, you will see the response that looks as follows:

▸ CreateCollectionResponse

Next steps

Now that you have a Myria collection, you can mint assets into that collection.