OPCODE CONSULTING
SMART CONTRACTS. SAFELY.

SHORT INTRODUCTION TO CODING NFTS

OVERVIEW OF MINTING PLATFORMS AND DEVELOPMENT TOOLS

matt@opcodeconsulting.com
twitter @CodingInLondon

NOVEMBER 2022

PURPOSE

The intention of this paper is to help NFT creators get up-to-speed with setting up a working environment to program and deploy NFTs. Because coding might not be necessary depending on your needs, the first section explores the capabilities of a subset of common minting platforms.

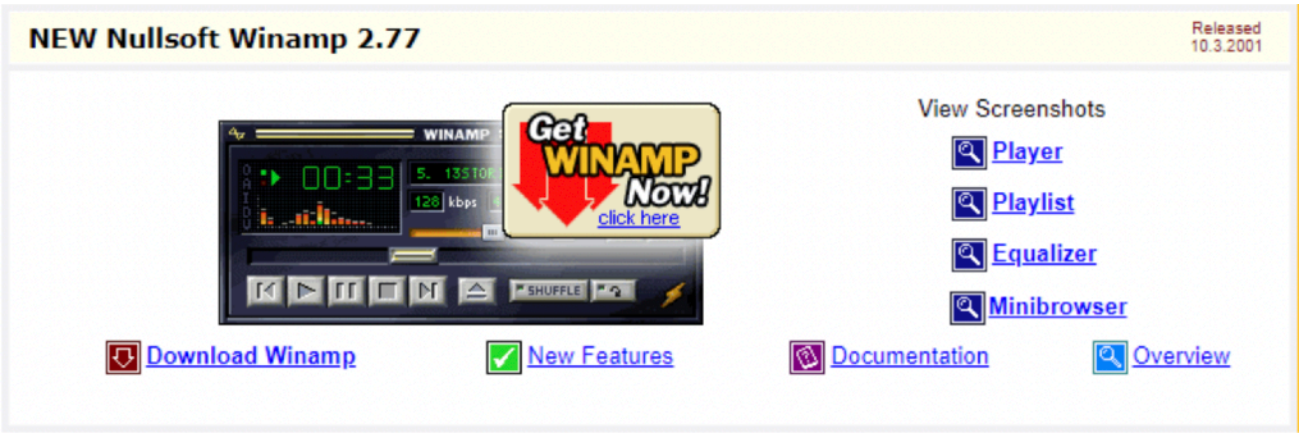
This is the first version of this paper and will evolve in the future with additional details.

INTRODUCTION

MORE THAN MONKEYS

NFTs address the problem of digital ownership. They allow you to claim “yep, this thing is mine” even if it’s digital.

Ever since the Internet has gone mainstream, the industry has been struggling with finding a technical solution to selling digital things. Remember how Microsoft was trying to introduce Digital Rights Management with their WMA format at the time when you could stream music for free with Pandora and use Winamp to play pirate mp3s downloaded overnight with Napster? There was no medium ground between clunky gatekeeping and full-on piracy.



Come on, you’re not that young, you **do** remember this.
Source: Web Design Museum

Then came Apple iTunes, Spotify and eventually Netflix that made the experience smoother and more legit. However something still isn’t quite right: with Kindle or Apple TV, when you buy a book or a movie you don’t really own the copy. You merely receive a right to download or stream something from a company that controls both your account and the content. This is not ownership as in owning a physical book, a DVD or an audio tape. It is more like *renting* rather than *owning*.

So what? Who cares as long as I can watch my Netflix?

In *Fahrenheit 451,* special squads had to go door to door with a flame-thrower. How tedious. Now that everything can be centrally modified, censored or deleted, burning books has become much more carbon neutral.

Bitcoin is this *zero to one* invention that makes it possible to natively store and move value on the Internet in a digital form at the protocol level, making the entire technology stack of the existing financial system obsolete overnight.

Ethereum generalises the idea by making the transfer of value easily programmable, making it very easy (too easy?) to create tokens and eventually NFTs.

As long as you are the only one to know the *private key* for your public address (whether the chain is Bitcoin, Ethereum, Tezos, Polygon, Avalanche, Algorand, Flow or Solana...) then you *own* everything that is on this address: coins, tokens and NFTs.

All you need is the private key. Any other type of secret (user-generated password, 2FA challenge, credit card number, expiry date, 3-digit pictogram thingy, phone number, SMS code, 3D authentication, postcode, passport number, birth date, maiden name of your Mum, name of your first hamster) is just inferior.

Because we have been conditioned to using things for free on the Internet for the past 30 years, we still struggle with the idea that we can own something online. We have not yet absorbed the concept intuitively. In 1994 we would have said “I downloaded a page of HTML markup over the HTTP protocol from a World Wide Web server”, today we just say “I found this online”. Similarly very soon instead of talking of “Non fungible tokens on blockchains” we will say “I own this online”.

Or even simply: “I own this”. The “online” bit will be implicit.

CONTENTS

OVERVIEW OF MINTING PLATFORMS

Taking the point of view of someone who wishes to create NFTs -as opposed to a collector or a buyer, I have briefly experimented with a small subset of NFT platforms. The takeaways are summarised in this chapter.

I don't have a gaming nor art background. The bias of this paper is technical. The intention is to grasp the capabilities of existing platforms in order to understand in which situations coding your own smart contract by hand makes sense.

SOME CLARIFICATIONS ABOUT GAS

Gas fees are a mechanism part of Ethereum design to prevent Denial of Service attacks against the network. Although Ethereum is distributed across thousands of machines, the whole system behaves as a single-core CPU running a single-threaded process. Just like Windows 3.1. Except it doesn't freeze up like Windows 3.1 used to.

If email had gas fees, there would be no such thing as spam or phishing because it would be too costly for spammers to spam.

- The price of Ethereum operations is measured in **gas units**. For instance a standard ETH transfer from one address to another costs 21000 gas units. Execution of smart contracts cost more.

- **gas price** is the price in ETH of a single gas unit. It is usually quoted in gwei (10^{-9} ETH or one billionth ETH). At the time of writing (10 July 2022), gas price is hovering around 10 gwei, which is historically cheap.

  Ethereum mining nodes prioritise transactions based on *fee tips*, a bit like a bouncer in a VIP club that would allow you to jump the queue if you paid something on top of the entrance ticket. The gas price also depends on network congestion (Ethereum block usage), which determines the *base fee*. Therefore the total cost to enter the club depends on two components:

  1. how many people are queuing (*base fee*)

2. how keen you are to jump the queue (*fee tip*)

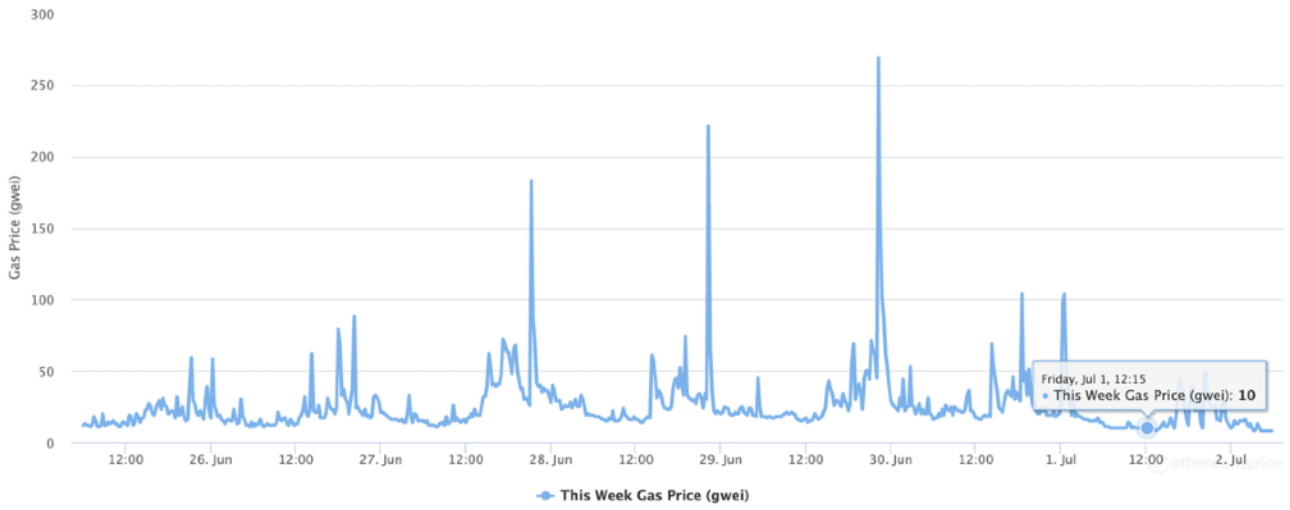
In other words:

gas price = base fee + fee tip

The *fee tip* is within your control. Most wallets usually suggest priority levels (low, standard, high). More on this topic here.

- **transaction fees** are calculated by multiplying the cost of one operation in gas units by the gas price.
  For instance a standard ETH transfer has a fixed cost of 21000 gas units. As of 10 July 2022, gas price is 10 gwei. Therefore an ETH transfer costs 21000 * 10 = 210000 gwei. Assuming an ETH price of USD1187, this is about USD0.25.

Ethereum gas price varies with the time of day. For some reason it tends to be low around European lunchtime:



Source: https://ethereumprice.org/gas/

MAIN TAKEAWAY

Before diving into the details of the research, some observations:

- Most platforms require a crypto wallet installed in your browser to authenticate, sign transactions and sometimes pay for gas fees. The most common wallet is Metamask. The wallet may be different depending on the chain you are targeting and usually requires you to own some crypto in order to create anything on mainnet. Only a few rare platforms accept credit card payments instead of crypto.

- The set of chains supported by a platform is an important factor. Any platform that supports Ethereum only and nothing else will incur significant gas fees.

- Using another chain than Ethereum or Polygon means buyers will not be able to use Metamask and might have to install a wallet specific to the targeted chain, which may add friction.

- **Polygon** has the advantage of full Metamask support and low gas fees. From the creator's point of view, Polygon appears as a completely different chain distinct from Ethereum. Under the hood it's actually a Proof-of-Stake sidechain attached to Ethereum. Does this mean it is less secure than Ethereum? Will NFTs minted on Polygon last as long as the Ethereum blockchain? This is still unclear. However if long-term survival of the NFT is not a priority, then it's a very good option because transaction fees paid in MATIC are very cheap.

WHAT ABOUT BITCOIN?

ERC721 and ERC1155 were specified within the Ethereum environment. Most of the trading volume takes place on Ethereum. However, Bitcoin is still the most secure, time-tested and future-proof chain. Therefore if priority is not to sell the NFT but instead maximise the chances that it will still be around in one hundred years, then it makes sense to consider what can be done on Bitcoin. This aspect is missing from the current paper but I plan on experimenting with this in the future.

- for more digging, see RSK, Stacks, Taro, Liquid, Slashtags and RGB protocol.

- here is an example of marketplace for Stacks.

- discussion about NFTs at Surfin Bitcoin 2022 (in French).

PLATFORMS SUMMARY

Platform features keep evolving fast so take the details in this table with a grain of salt. It will be out of date by the time I finish typing this sentence.

	Genre	Takeaway	Chains	Features
Rarible	**Art**	- No barrier of entry, you can create NFTs simply after signing in with your crypto wallet. Filling out a profile is optional. Verification is optional. - free minting on Ethereum - Requires specific wallets for Tezos	Ethereum Polygon Tezos Solana Flow	- Supports multiple copies with ERC1155 - File formats: pics and video - Storage: sends pictures/videos to IPFS by default - Unlockable content (on Ethereum)
SuperRare	**Art**	- Caters for professional artists, access is invite-only - works with Metamask but also requires to create an account with username and email - you are not supposed to upload content available elsewhere	Ethereum only	ECR721 and custom smart contracts for special 'series'
OpenSea	**Generic (collectibles, art, ...)**	- No barrier of entry. Login with metamask is enough - Very large reach - Mint for free on Polygon - Because cloud storage is used by default instead of permanent storage, this is ideal for experimenting	Ethereum Polygon Klaytn Solana and others	- Supports multiple copies with ECR1155 - File formats: pics, videos and 3D models - Storage: By default files are stored on a cloud controlled by OpenSea. You have to explicitly 'freeze' the NFT to send your picture to IPFS instead. - Supports lazy minting
Foundation	**Art**	No barrier of entry but before being able to create you must verify your Twitter or Instagram account	Ethereum only	- Deploy your own smart contract - Storage: IPFS
The Sandbox	**Game**	Game assets created with VoxEdit can be minted as NFTs if you are a whitelisted creator on the platform	Ethereum Polygon	
Decentraland	**Game**	Creations are limited to wearables to be carried around by the in-game avatars	Ethereum Polygon	
NFT Worlds	**Game**	- 10000 unique voxel-based worlds are available for sale - if you own a world you can customise it	Ethereum	Worlds were originally created to run on top of the Minecraft engine.

PLATFORMS DETAILS

RARIBLE

- Supports both **single copies** (ERC721) and **multiple copies** (ERC1155). The interface gives you a clear choice between the two. More about standards in the next section.

- supports **Polygon**, **Tezos** and **Flow** in addition to Ethereum.

    - To mint on Flow you will need to install a wallet called Blocto.

- Uses **IPFS by default** (unlike OpenSea where IPFS is optional). The picture you upload will be automatically stored on IPFS during the minting process. The process is transparent and you don't have to deal with CIDs or any specifics of IPFS. It is unclear whether Rarible also uses a mechanism to guarantee the permanence of the hosting (FileCoin? Pinata?).

- **Free minting** is available on Ethereum (fees will be paid by the buyer upon first sale). However you still have to pay gas fees if you create a collection.

- Creating a collection/minting NFTs involves gas fees, even on Polygon. Although gas fees will be extremely small on Polygon, you still have to make sure that your address has some MATIC before being able to create.

- Unlockable content is available on Ethereum and coming soon for Polygon (as of June 2022).

Rarible getting started guide.

To experiment with Rarible, use their testnet version, with a faucet.

SUPERRARE

- SuperRare is rather exclusive: the platform is invite-only and involves an application (as of June 2022). There is no guarantee the application will be accepted.

    - supports only Ethereum chain and ERC-721. Each NFT can exist in one copy only. It makes sense to bet on longevity with Ethereum. Imagine you sell a creation that becomes a classic; in 30 years, it might still be sold and bring you royalties.

    - artwork created here cannot be sold anywhere else.

- Much more curated than Opensea, aimed at serious artists

- Works with Metamask

- Also requires the creation of an account with username and email

- Customisation of smart contracts is possible

Moralis has a good blog post on SuperRare.

OPENSEA

- Has the lowest barrier of entry. No invite, no application, anyone can apply and mint immediately.

- OpenSea supports **Polygon** in addition to Ethereum, which is a good way to save on gas fees.

- Supports lazy minting, which means the on-chain transfer of the NFT to Ethereum will effectively take place upon first buy and the associated gas fees will be paid by the buyer.

- OpenSea only requires wallet signing to log-in. No other info is required. i.e. you can login with Metamask only, without having to provide an email address.

- Among the supported formats, it's interesting to note OpenSea accepts **3D Models** on top of the usual picture / video formats.

- Storage: by default when you mint an item, the *name*, *media* and *description* are stored centrally on OpenSea's own storage. This means you can still edit your item after you created it. To go fully decentralised, you must freeze your listing, which

will upload the item's data to **IPFS**, after which you can't change the *name*, *media*, *description* any more.

- Depending on the chain you pick, it is possible to get paid in something else than ETH (for instance DAI and USDC are available with Ethereum and Polygon).

- You can test out OpenSea with testnets.

THE SANDBOX

This is a game metaverse with a funny lego-like graphic style. The in-game currency is SAND which is an ERC-20 token on Ethereum.

- It is possible to create **game assets** within Sandbox. They are called voxel assets and are created with VoxEdit. By the way, a 'voxel' is a pixel in 3 dimensions. It is a 3D representation used in GIS and medical imaging. The Sandbox has guidelines with regards to how to create voxel assets. As of June 2022 TheSandbox offers a creators fund where asset designers can apply for grants.

- Optionally, these game assets can be minted as NFTs. You have to be a whitelisted creator to be allowed to mint NFTs.

- NFTs minted with Sandbox are based on the ERC-1155 standard (therefore multiple copies of the same asset are supported).

DECENTRALAND

Decentraland is Ethereum-based / NFT-based from the ground up and controlled with a DAO. The intention of the developers of Decentraland is to eventually withdraw from the project and let it run autonomously. The in-game currency is MANA, an ERC-20 token that exists both on Ethereum and Polygon.

Do not expect *World Of Warcraft* levels of graphics here. Everything looks more like the Dire Straits *Money for Nothing* video from 1985.

- Fundamentally NFT-based: each LAND token represents a block of land in the game. LAND is an ERC-721 token on Ethereum. You buy LAND with MANA on the Decentraland marketplace.

- You can **create wearables** (wearables are NFTs on the Polygon chain). That is done with a 3D modelling tool called Blender. Wearables can then be sold on the Decentraland marketplace on the Polygon chain with no transaction cost (The Polygon transaction costs are covered by the Decentraland Foundation).

- There is a builder tool to create **scenes** (which are for instance the art galleries and buildings you will find on a plot of land). However as far as I understood scenes are not NFTs. They only exist within Decentraland and can be deployed either to a plot of land you own or into a scene pool to be used by other players. The Decentraland SDK makes it possible to code finer game dynamics within a scene.

NFT WORLDS

Part of the info in this section is obsolete as NFT Worlds is migrating away from Minecraft as of December 2022.

Allows easy building of metaverse worlds: voxel-based square terrains with different combinations of topological traits (hill, ocean, beach, forest, river, sunken ship...). The in-game ERC-20 token is called $WRLD.

- There are currently 10000 such worlds listed on OpenSea and available for purchase. They are used as a starting point for a world to be built by the owner. You can explore any of those worlds already with the viewer.

  For instance, try out world #3186 in the viewer. It's a little island.
  Use the <spacebar> go up, <shift> to go down, click and drag to turn around.

- World owners can make their world evolve. This is done either with the NFT Worlds client (in alpha as of June 2022) or with Minecraft Java Edition, which you have to pay for to download (USD27). For instance the Heroes Motors NFT project owns world #3186 and is planning to build a racing circuit on it.

- NFT Worlds is very new. It was released in October 2021. Yet the Minecraft ecosystem has been around for 10 years, which means a lot of support with regards to platforms (Windows, Mac, Unix, Xbox, Playstation, Xbox, Nintendo, phones) and tooling.

- How does NFT Worlds compare to Decentraland or Sandbox? There is support for a larger number of platforms, possibility for modification of the world within the game and direct access to smart contracts from within the game.

STANDARDS

An NFT is a smart contract that presents a specific interface. Two main standards exist with regards to this interface: ERC721 and ERC1155.

ERC721 says: “This smart contract defines a class of unique gizmos. gizmo #123 belongs to Starsky, gizmo #124 belongs to Hutch”

ERC1155 says: “This smart contract defines a class of gizmos where each gizmo can exist in multiple copies. Starsky owns 8 copies of gizmo #123 and Hutch owns 7. Hutch also owns 5 gizmo #124”

The OpenSea blog has a better description of those standards here.

ERC721

Check out the official Ethereum specification.

Creating an NFT means deploying a contract that implements the interface below.

```
FUNCTIONS
balanceOf(owner)
ownerOf(tokenId)
safeTransferFrom(from, to, tokenId, data)
safeTransferFrom(from, to, tokenId)
transferFrom(from, to, tokenId)
approve(to, tokenId)
setApprovalForAll(operator, _approved)
getApproved(tokenId)
isApprovedForAll(owner, operator)
```

ERC721 specification from OpenZeppelin’s API reference

The implementation is completely up to you, however the interface must be respected so that your NFT works with all platforms (and in particular the marketplaces).

The contract identifies each unique NFT with a *tokenId*.

balanceOf allows anyone to check how many tokens an address owns for this particular class of NFTs. If an address owns more than one NFT, each NFT will have a distinct *tokenId*.

ownerOf gives you the address of the owner of a specific NFT, identified by its tokenId.

If you are the owner of an NFT you can transfer it to someone else by calling *safeTransferFrom*. This works only if you call *safeTransferFrom* from an address that owns the *tokenId*. It is also possible to delegate the permission to transfer to another address, even though this other address does not own the NFT. You do that with the *approve* function.

All ERC721 NFTs do more or less the same basic things so the best practice is to use the OpenZeppelin default implementation as a starting point, to which you can add your own customisation.

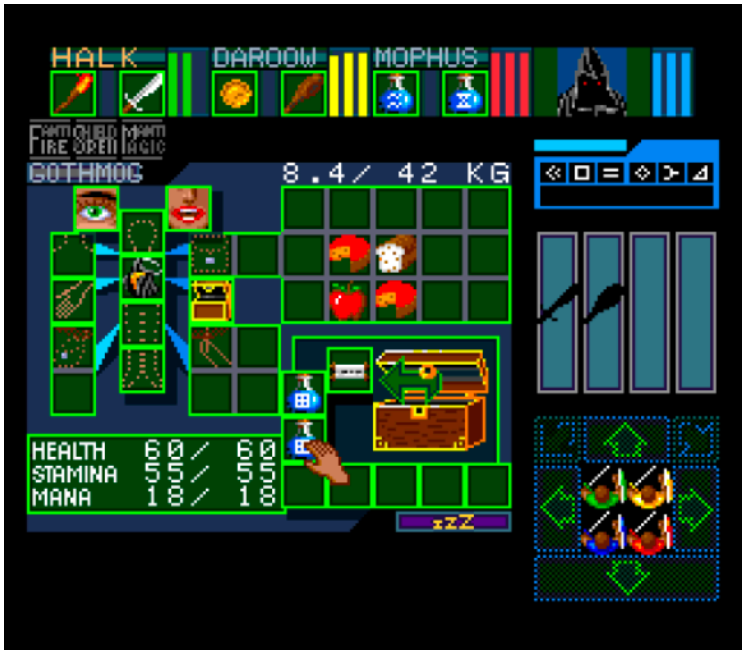
For a better explanation of how ERC721 is implemented, check out the OpenZeppelin documentation.

ERC1155

This standard takes the more pragmatic approach of allowing a *tokenId* to exist in multiple copies.

ERC1155 has its origin in gaming. I'm not a gamer but I used to play *Dungeon Master* on Atari STF in the 1990's and I remember I was happy to store more than one health potion in my bag whenever I found one in a tunnel.

Well if a health potion was an NFT there would be no reason to identify each health potion individually, would there? However it is important to record how many health potions a particular player owns.



Dungeon Master screenshot: content of a player's bag

This is what the ERC1155 interface looks like:

```
FUNCTIONS

balanceOf(account, id)

balanceOfBatch(accounts, ids)

setApprovalForAll(operator, approved)

isApprovedForAll(account, operator)

safeTransferFrom(from, to, id, amount, data)

safeBatchTransferFrom(from, to, ids, amounts, data)
```

ERC1155 specification from OpenZeppelin's API Reference

balanceOf tells you how many tokenIds an owner has (how many health potions do I have in my bag).

As in ERC721, *safeTransferFrom* allows an owner of a *tokenId* to transfer to another address. However there is now an *amount* parameter to specify how many *tokenId* are moved.

You can still delegate the right to transfer your NFTs to someone else with *setApprovalForAll*.

It's interesting to note that ERC1155 can do everything that ERC721 does. It just adds extra flexibility.

balanceOfBatch makes it easy to get the balance information for multiple owners, without having to do multiple calls (Each call has a gas cost, which is especially significant if you are using Ethereum).

Same goes for *safeBatchTransferFrom*.

UNDER THE HOOD

What happens when you mint an NFT with...

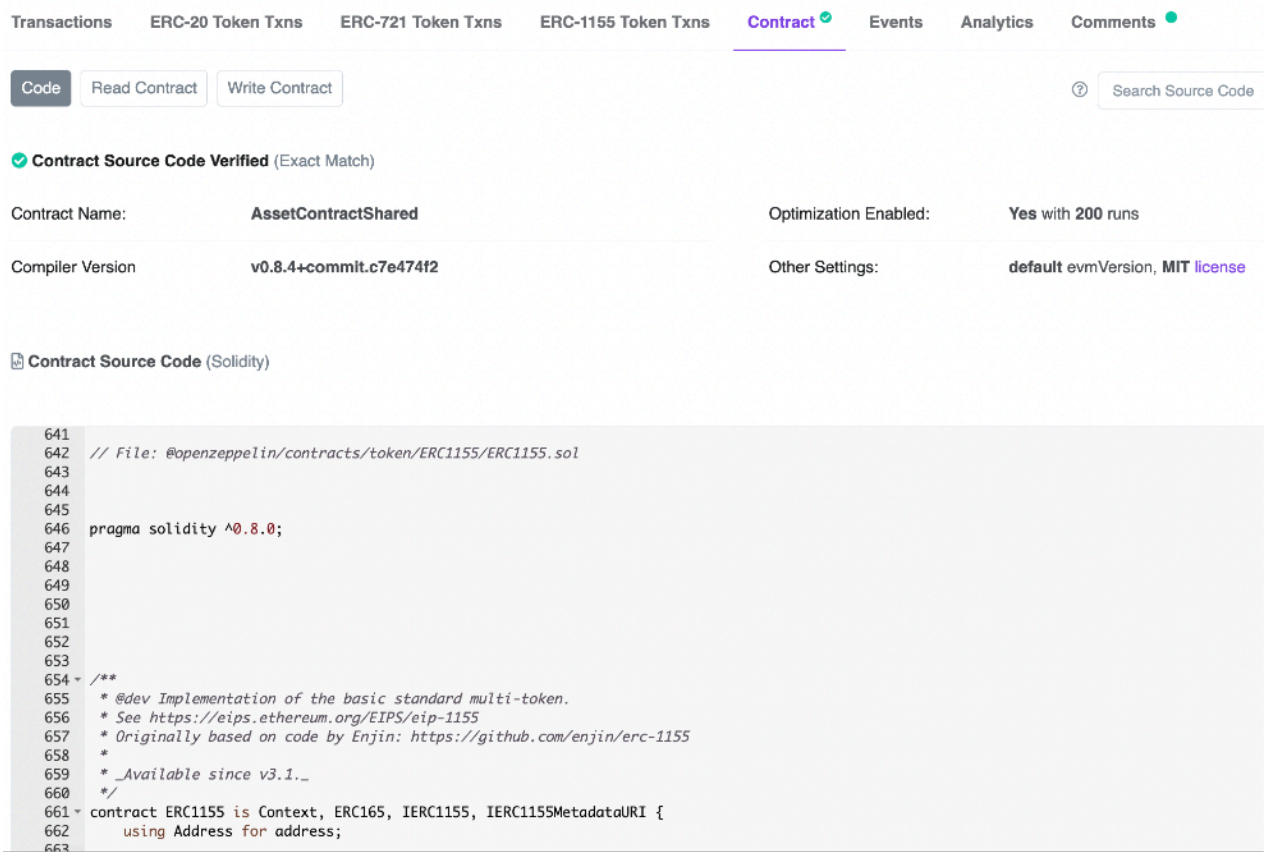
WITH OPENSEA

When creating an item on OpenSea, ERC1155 is used by default. Even if you specify that you want a single copy of your NFT, ERC1155 is systematically used in place of ERC721. Actually OpenSea uses one single smart contract for all users.

- For instance, as of July 2022, Ethereum OpenSea contract is on address 0x495f947276749Ce646f68AC8c248420045cb7b5e

- Polygon OpenSea contract is on address 0x2953399124F0cBB46d2CbACD8A89cF0599974963

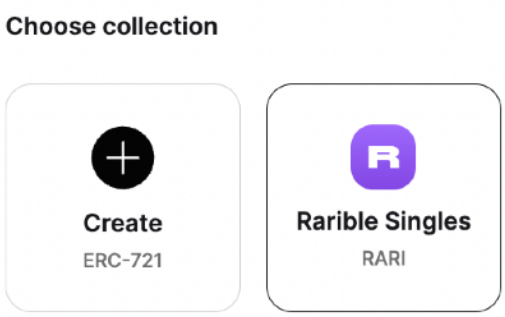
Interestingly if you look more closely at the Polygon contract, you can see that it has been **verified**, i.e the source code has been linked with the contract address for reference.



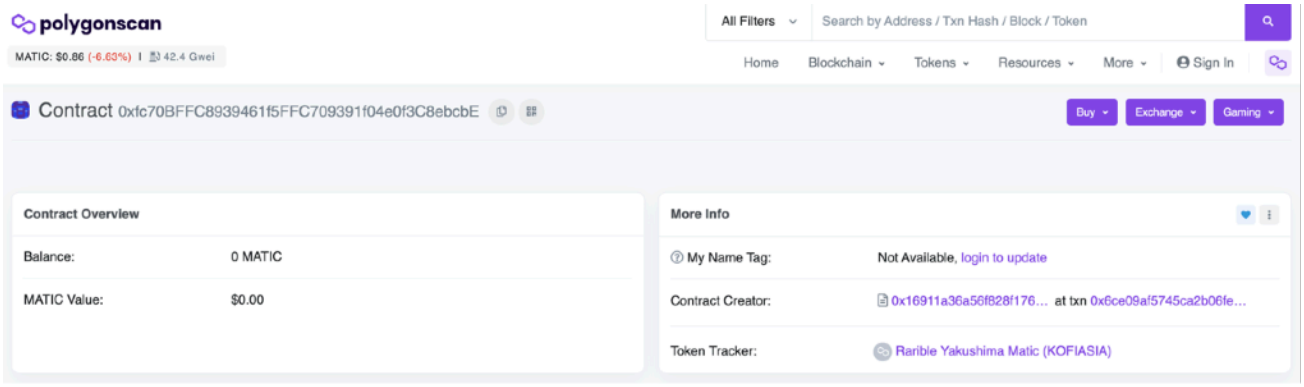
The OpenSea smart contract on Polygon, showing references to OpenZeppelin library

WITH RARIBLE

On Rarible you have the option to create your own ERC721 or ERC1155 collection.



Creating a “collection” results in the creation of a new smart contract with the token identifier you chose.



Rarible created contract 0xfc70BFFC8939461f5FFC709391f04e0f3C8ebcbE with token tracker KOFIASIA

However if you pick “Rarible Singles”, your NFT will leave inside Rarible’s Polygon smart contract on address 0x35f8aee672cdE8e5FD09C93D2BfE4FF5a9cF0756.

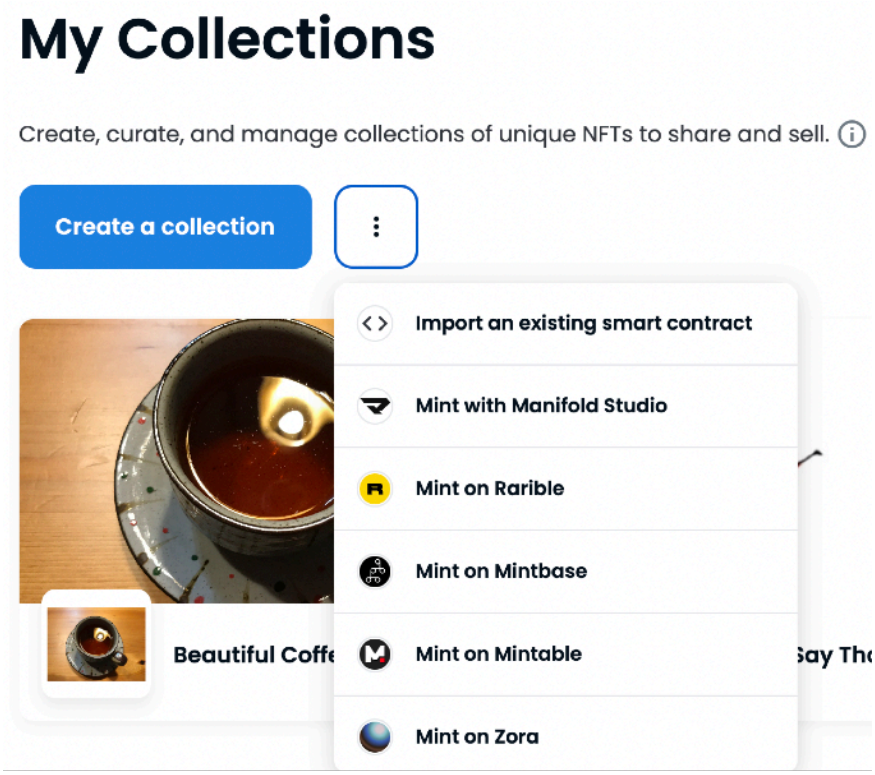
INTEROPERABILITY BETWEEN PLATFORMS

- If you create your own NFT smart contract, can you list it on any marketplace?

Yes for some marketplaces. For instance both OpenSea and Rarible have an option to import an existing smart contract given its contract address. See below.

- If you mint an NFT on Rarible, can you show it on the OpenSea marketplace?

Yes! As long as you know the contract address of your NFT and it follows ERC721 or ERC1155, you can import it into OpenSea. Go to *My Collections*, pick the *import existing contract* from the menu and specify the *network* and contract address to import from.



Your Rarible NFT will then be listed in OpenSea as any other NFT. The only way to tell that it was not minted on OpenSea will be the contract address which will not be the standard OpenSea address.

- If you mint an NFT on OpenSea and OpenSea shuts down, what happens to your NFT?

I am guessing that your OpenSea profile and listings would become unavailable.

However the NFT smart contract itself -whether it is the OpenSea smart contract, a contract created on another platform or a contract that you coded yourself- would still exist on-chain (Ethereum, Polygon, Solana or Klaytn), and would still contain an entry showing that your account address owns a specific token id.

If you had decided to freeze your metadata (IPFS), then your jpeg would still be available on decentralised storage.

However if you did not freeze your metadata then your jpeg, which is stored on an OpenSea cloud, would be unavailable.

DEVELOPING YOUR OWN NFT SMART CONTRACT

WHY?

Designing your own customised NFT logic involves writing smart contracts with Solidity code and deploying them to an EVM-compatible chain.

Most NFT minting platforms use their own smart contracts that provide out-of-the-box features such as:

- Percentage of royalty sent to the NFT issuer for each transaction on the secondary market. OpenSea, Rarible, SuperRare all offer this.
- Unlockable content: file accessible to the first buyer. To be stored on a central server (cloud) or on decentralised storage (IPFS).

So why would you go through the pain of coding NFTs by yourself when you can easily mint NFTs with a few clicks?

1. Customisation

With a custom smart contract, you could do things such as:

- send a percentage of each sale to a charity
- reveal hidden content on a specific date
- access a storage system not supported by minting platforms, such as Arweave for instance. Most platforms are limited to IPFS.

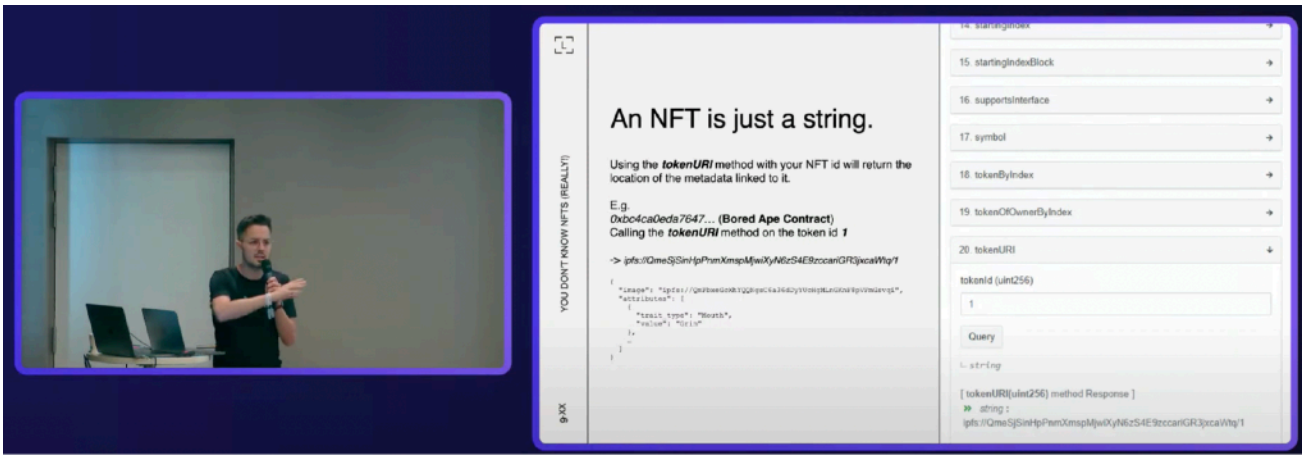
2. Security

If you use someone else's smart contract instead of providing your own, then you have to trust that the owner of the smart contract was honest at the time they created the code. **A smart contract owner has the ability to code any logic they want in the smart contract they provide.** For instance they could write logic that allows them to

transfer your NFT to someone else or change the image reference to another image in the metadata. Of course such a trick would be visible to all since smart contract code is visible in any chain explorer.

Some platforms (such as Rarible) provide smart contract addresses where the code is a proxy instead of a straight ERC721/ERC1155 contract. This is an architecture decision to make it possible to upgrade the code in case of a bug. However that means that **at any time the contract owner has the ability to change the code of the smart contract** storing your NFT.

For more information about those considerations check out Kevin Lambert talk at EthCC5.



An NFT is just a string. Kevin Lambert (Ledger).

RESOURCES

- NFT School complete end-to-end guide to mint NFTs from scratch with code using IPFS, FileCoin and Polygon.
- Eat the Blocks (YouTube), How to learn NFT Development for Beginners in 3 steps
- Moralis Web3 (YouTube) Ultimate NFT Programming Tutorial - FULL COURSE
- Moralis Academy - Build an NFT Marketplace (course)
- Alchemy's weekly coding challenges.
- Solidity:
  - Specification of the latest version of the Solidity language.
  - Moralis Academy courses on smart contracts 101, 201 and Security.
  - Openzeppelin has a guide for the full smart contracts development lifecycle.
  - Consensys Diligence security tooling guide.

ENVIRONMENT SETUP

Here are some set-up notes specific to an Ubuntu platform. I find it convenient to use Ubuntu under MacOs with a virtual machine, rather than polluting my MacOs install directly.

To run virtual machines under MacOs, I have tried Parallels (which is a subscription-based product) and VirtualBox (which is free).

The set-up assumes that you already have *Metamask* installed as an extension of your browser. I have been using Metamask extension for Firefox under Ubuntu.

Note that the *Brave* browser comes with a native wallet embedded in the browser instead of as a browser extension.

VISUAL STUDIO CODE

Linux install instructions are here.

1. To install Visual Studio Code under Ubuntu in a VM hosted on a MacBook Pro M1, you have to download the *arm64.deb* package. For other architectures try the *amd64.deb* package instead.

2. Then once you have downloaded the file (code_1.71.2-1663189619_arm64.deb for instance), open a terminal in the Download folder and type:

```
> sudo apt install ./code_1.71.2-1663189619_arm64.deb
```

This should have added the Visual Studio Code icon to your applications list.

While you're at it, why not install a vscode extension that makes Solidity code look better?

3. In vscode, click *Extensions* icon, search for "solidity" and select the *solidity* extension by Juan Blanco.

TRUFFLE

Truffle is a free command-line tool maintained by Consensys. It allows you to control compilation, test and deployment of smart contracts with scripts. You write unit-tests running against a local test version of Ethereum (Ganache) that works totally offline. Unit-tests are crucial in the context of smart contracts development. It's one of the many mitigation tools to reduce the risk of publishing code that could be exploited.

Another popular framework is Hardhat. I'm using Truffle in this section for no particular reason, only because it is the first framework the baby crocodile has seen when coming out of its egg. That said, I heard only good things about Hardhat.

- Step #1: install NVM and Node.js

Follow the instructions at trufflesuite.com to install nvm and Node.js under Ubuntu (similar instructions as MacOs under *MacOs > Node.JS - Use NVM*).

This is *nvm* not *npm*. nvm is the version manager for Node.js

The following commands worked for me:

```
> curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/
install.sh | bash

(restart the terminal)

> nvm install --lts
```

- Step #2: install Git

Find the instructions under Ubuntu > Git. It's actually just one command:

```
> sudo apt install git
```

- Step #3: install Truffle

Follow the instructions here. We're now using *npm*, not *nvm*.

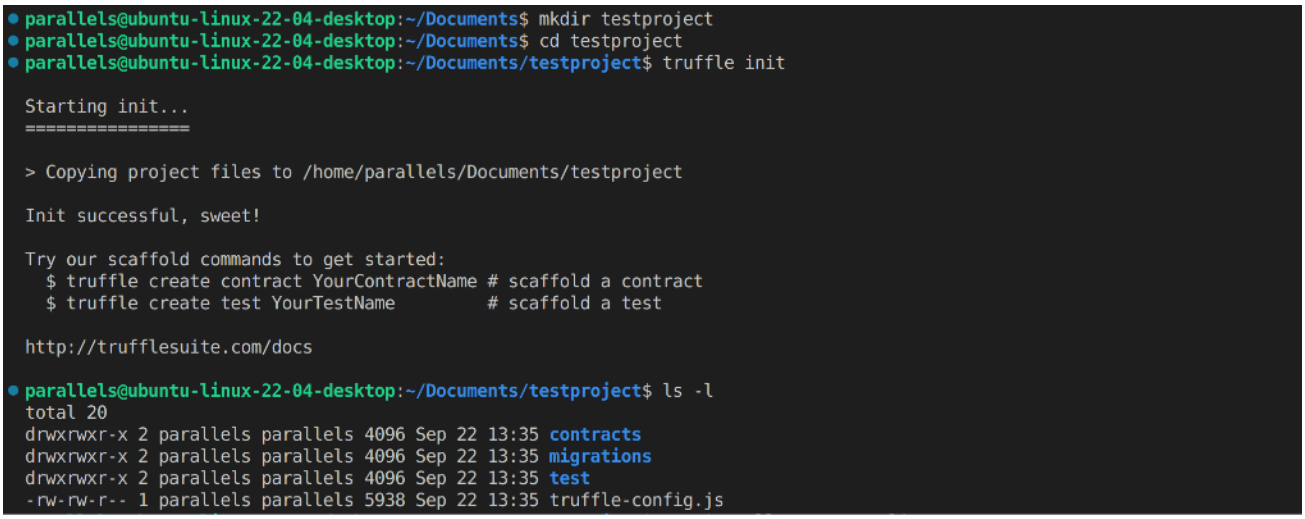
```
> npm i -g truffle
> npm i -g ganache
```

- Step #4: initialise a project directory

1. Create a folder for your project. Inside the folder project run

   ```
   > truffle init
   ```

   This creates 3 distinct folders reflecting your development lifecycle: contracts, migration, tests.



2. Update the solc compiler version in truffle-config.js if it's not already up-to-date. What version of the Solidity compiler you use is entirely up to you. It will have to be compatible with whatever libraries you import (the Openzeppelin libraries for instance). You can find the latest version number here.

3. Dry-run your environment with

   ```
   > truffle compile
   ```

   This will download the version of solc you specified in truffle-config.js. Your contracts folder is empty therefore there is nothing to compile and everything should work fine.

Go to the Truffle's doc for more help.

OPENZEPPELIN

When it comes to smart contracts, the least code you write yourself, the better. Find giants and stand on their shoulders.

Openzeppelin is a library that offers well-tested code for a number of situations, including default implementations for tokens ERC20, ERC721 and ERC1155.

By the way, this is not just to save time: re-using code that has been audited and has been running in production for years minimises the risk of exploits.

Install instructions here. In your project folder type:

```
> npm install @openzeppelin/contracts
```

When you open your project folder under Visual Studio Code, you will see the Openzeppelin files installed under *node_modules/@openzeppelin/contracts*.

Take a look under the folder *token* for interface code and default implementations of ERC20, ERC721 and ERC1155.

DEVELOPMENT PROCESS

This section helps you build the simplest NFT possible (re-using Openzeppelin's standard ERC721 implementation) and go through all steps to eventually deploy it to a real testnet.

The NFT itself is completely useless but the purpose of this section is to give you a feel of the tooling involved in publishing your own smart contract logic and how to install an operational development environment.

COMPILE

Add a basic My721 contract deriving from OpenZeppelin's ERC721. Save it under the *contracts* folder as My721.sol.

```
// SPDX-License-Identifier: MIT
```

```
pragma solidity ^0.8.0;

import "../node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol";

contract My721 is ERC721{

    constructor() ERC721("MyNFTContract","MYT"){
    }

}
```

Navigate through node_modules/@openzeppelin and take a look at ERC721.sol and IERC721.sol. To summarise:

- *IERC721.sol* has the interface that makes the smart contract interoperable with any app that expects a standard NFT (OpenSea, etherscan.io, etc...)
- *ERC721.sol* has standard default implementation for the minimum functionality one would expect from an NFT (such as mapping a token id to an owner's address for instance).
- *My721.sol* specifies the name and symbol of your class of NFTs. Many NFTs can exist with the same name/symbol. They will be identified by their token id.

DEPLOY LOCALLY

Before deploying to an actual network (testnet or mainnet), you normally deploy to a local blockchain emulator where most of the testing is done. No waiting time, no network connectivity needed, no ETH needed.

Deployment to mainnet requires access to an actual Ethereum node, which you can get from a service or set-up yourself.

Migrations require a bit of javascript code that tells the Truffle framework which contract to migrate.

1. Under the *migrations* folder, add a file called *2_migrate_my721.js* with the following code:

```
const Mig = artifacts.require("My721");

module.exports = function (deployer) {
  deployer.deploy(Mig);
};
```

When run, this javascript code creates and broadcasts an Ethereum transaction that creates your contract at a new Ethereum address. More about migrations files here.

The Truffle framework creates javascript wrappers around your Solidity contracts. Those wrappers ("abstractions") allow you to interact directly with the public functions of the contract once it is deployed on a chain.

You will eventually do this in mainnet (but at that point you'll get only one shot).

2. From the project folder type

```
> truffle develop
> migrate
```

This compiles the contracts then deploys them. It also creates 10 fresh ETH dummy accounts along with their private keys, that you can use for your testing.

3. Interact with the deployed contracts like this:

```
> var instance = await My721.deployed()
> instance.name()
> instance.symbol()
```

The first line gets an instance to the javascript wrapper around your My721 contract. You can then call all its public functions directly.

- **How do you tell truffle which chain to migrate to?** This is done in the *truffle-config.js* file located in your project folder.

If you are using the local blockchain created by truffle migrate then you don't need to change anything in *truffle-config.js*.

If however you are using any other local blockchain (such as the one started from the *ganache* client for instance) then you will need to specify in *truffle-config.js* the RPC host/port displayed in ganache:

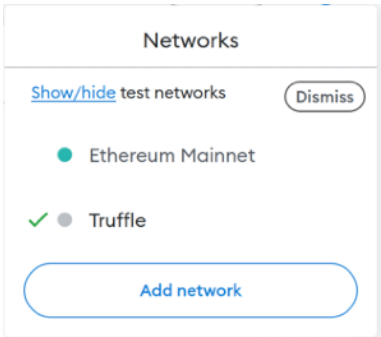
```
networks: {
  // Useful for testing. The `development` name is special - truffle uses it by default
  // if it's defined here and no other network is specified at the command line.
  // You should run a client (like ganache-cli, geth or parity) in a separate terminal
  // tab if you use this network and you must also set the `host`, `port` and `network_id`
  // options below to some value.
  //
  development: {
   host: "127.0.0.1",     // Localhost (default: none)
   port: 8545,            // Standard Ethereum port (default: none)
   network_id: "*",       // Any network (default: none)
  }
```

- **What is the difference between truffle develop and ganache?** truffle develop automatically starts a local blockchain emulator then accepts *compile* and *migrate* commands. The ganache CLI is a separate client that starts a similar local blockchain emulator. There is also a ganache GUI (but I have not tried installing it under Ubuntu).

- **Can you view the dummy ETH accounts in Metamask?** Yes. Metamask can control any ETH account as long as it knows the network and the private key.

To connect to your local blockchain from Metamask,

1. click *Add Network*.



2. Then specify:

- *RPC URL* displayed by ganache/truffle at startup. For instance: http://127.0.0.1:9545/

- Chain Id: 1337 (displayed in ganache)

3. Import a dummy account from ganache/truffle with the *Import account* menu in Metamask. Copy and paste the private key provided by ganache/truffle.

Ganache accounts come pre-loaded with 1000 fake ETH. You can move funds between the dummy accounts from inside Metamask.

TEST

You can test the contract by interacting with it manually from the console.

But in a real project, you want to automate those tests so that they can be repeated systematically whenever you change something in the code. Non-regression is useful.

To write a test, simply add a javascript file under the *test* folder. A typical test file would look like this:

```
const Nft = artifacts.require("My721");


contract("My721 tests", accounts => {
    it("the name should be correct", async() =>{

        let nft = await Nft.deployed();
        let returnedName = await nft.name();

        assert(returnedName == 'MyNFTContract', "name() returned wrong value " + returnedName);

    })


    it("the symbol should be correct", async() =>{

        let nft = await Nft.deployed();
        let returnedSymbol = await nft.symbol();

        assert(returnedSymbol == 'MYT', "symbol() returned wrong value " + returnedSymbol);

    })
});
```

Sounds tedious? Remember, your code will handle money belonging to other people and you can't fix issues in production. Do not cut corners.

Resources:

- truffle documentation on writing unit tests in javascript.
- Is is also possible to write unit tests in Solidity for more control.

DEPLOY TO TESTNET

To deploy to testnet or mainnet, you have to connect over RPC to a real Ethereum node (not a local blockchain emulation).

Alchemy and Infura are examples of node providers that offer access to Ethereum nodes as a service, which means you have to sign up and create an account with them. They both have free plans.

- For more options of node providers, check out the Ethereum documentation.
- Also check out this EatTheBlocks video.

Another option is to set-up a node yourself, which you can do with NiceNode or DappNode.

Truffle projects usually deploy to nodes managed by *Infura*, while *Alchemy* documentation describes projects based on *Hardhat*.

I thought of mixing things up a little so the following steps use the *Truffle* environment with a node from *Alchemy*. However I heard good things about Hardhat so I encourage you to give it a try.

Resources:

- Alchemy Hello World project.

- HDWalletProvider documentation. HDWalletProvider is used by the truffle-config.js file.

1. Sign-up for a free account at Alchemy

2. In the Alchemy dashboard, create an app for the Goerli network. Make a note of the API key URL. More info here.

3. Create an account in Metamask, connect it to the Goerli network and fund it from a faucet. More details here.

4. Install dotenv, create an .env file in the project folder. Export a private key from your Metamask test account. Store the private key and Alchemy API key URL inside the .env file. More details here.

5. Modify truffle-config.js to add dotenv support, import the keys and pass them to the HDWalletProvider within the Goerli network configuration. My truffle-config.js file looks like this:

```
require('dotenv').config();
const { PRIVATE_KEY, API_URL } = process.env;

const HDWalletProvider = require('@truffle/hdwallet-provider');

module.exports = {

  networks: {
    goerli: {
      provider: () => new HDWalletProvider(PRIVATE_KEY, API_URL),
      network_id: 5,       // Goerli's id
      confirmations: 2,    // # of confirmations to wait between
deployments. (default: 0)
      timeoutBlocks: 200,  // # of blocks before a deployment times out
(minimum/default: 50)
      skipDryRun: true     // Skip dry run before migrations? (default:
false for public nets )
    },
  },

  // Configure your compilers
  compilers: {
    solc: {
      version: "0.8.17",      // Fetch exact version from solc-bin
(default: truffle's version)
    }
  },

};
```

6. In order to deploy, use truffle migrate (not truffle develop)

```
> truffle migrate --network goerli
```

The command should confirm deployment and show you the new contract address. You should be able to see the contract address in Goerli etherscan and confirm a transaction just took place.

7. Confirm you can interact with your contract with truffle console:

```
> truffle console --network goerli
truffle(goerli)> var instance = await My721.deployed()
undefined
truffle(goerli)> instance.name()
'MyNFTContract'
truffle(goerli)> instance.symbol()
'MYT'
```

PUBLISH THE CODE FOR DOCUMENTATION

Keep those good habits going. How to make sure your code will appear on etherscan.io?

Coming soon...

STORAGE

This section will be enriched in a future version of the paper.

IPFS

IPFS is the protocol for decentralised storage. There is a good presentation given at EthAmsterdam that explains IPFS in details and how it relates to FileCoin and Pinata.

FileCoin is a mechanism that gives incentives to node owners to store files over IPFS, to guarantee that there is at least one machine to host a particular file. The user who wants to ensure a file is persisted in IPFS makes a deal on Filecoin. Filecoin provides cryptographic proof that a file is actually stored. The storage mechanism is not necessarily IPFS and can be any cloud storage.

Pinata is a paid-for service that guarantees your files are stored on IPFS (as an alternative to FileCoin).

Unstoppable domains allows you to buy a domain name pointing to an IPFS file.

Resources:

- Filecoin & IPFS 101 (Moralis Web3). Contains a list of tools to do IPFS hosting.
- Protoschool tutorials on IPFS and Filecoin.

HOW TO DEPLOY TO IPFS?

Coming soon...

ARWEAVE

Coming soon...

CONCLUSION

I hope that this short guide gave you some tools and starting points for developing your own NFT project, at least from a technical point of view. Applications of NFTs are endless and what you do with them is entirely up to you.

The tech is evolving very quickly and some of the minting platforms or chains mentioned here might disappear overnight.

From a distance it looks like Ethereum is a safe bet for developing NFTs. However it makes sense to also keep an eye on what is happening on top of Bitcoin. After all if an NFT is a certificate of digital property, you would obviously prefer it to outlive all sorts of geo-political events, state attacks and censorship.

This is the first version of this guide and it will evolve in the future.

LEGAL

www.opcodeconsulting.com

Limit of Liability/Disclaimer of Warranty: While the authors have used their best efforts in preparing this work, they make no representations or warranties with respect to the accuracy or completeness of the contents of this work and specifically disclaim all warranties, including without limitation any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives, written sales materials or promotional statements for this work. The fact that an organisation, website, or product is referred to in this work as a citation and/or potential source of further information does not mean that the authors endorse the information or services the organisation, website, or product may provide or recommendations it may make. This work is made available with the understanding that the authors are not engaged in rendering professional services. The advice and strategies contained herein may not be suitable for your situation. You should consult with a specialist where appropriate. Further, readers should be aware that websites listed in this work may have changed or disappeared between when this work was written and when it is read. Authors shall not be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

OPCODE CONSULTING

SMART CONTRACTS. SAFELY.