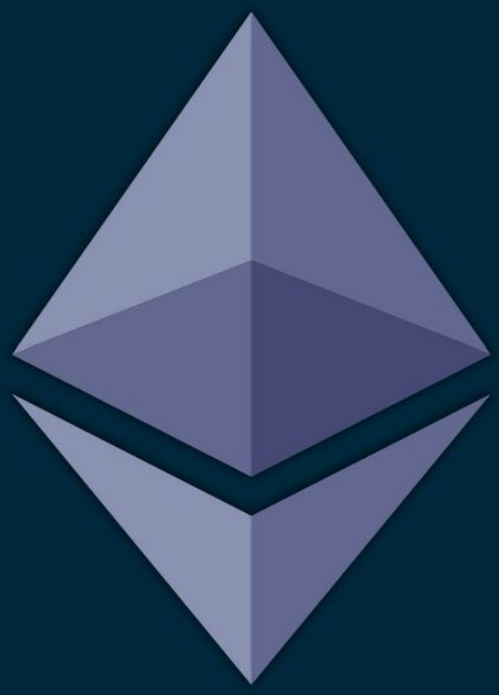
GET STARTED WITH

ETHEREUM



VSCODE

TRUFFLE

GANACHE

PINATA

ENS

FLEEK

OPENZEPPELIN

CHAINLINK

IPFS

INFURA

METAMASK

OPENSEA

BLOCKCHAIN SERIES

DAVI BAUER

How to Get Started With Ethereum

About this Book

Get Started With Ethereum

This book is a step by step guide for everyone who wants to start as an Ethereum developer. It was designed for those who have never programmed anything in blockchain and want to get started. I will cover everything from installing basic requirements to writing, testing and deploying smart contracts. I will also cover topics like IPFS, ENS, Chainlink, Truffle, Ganache, OpenZeppelin, Pinata, Fleek, Infura, Metamask, Opensea, among others.

You will receive free updates as new content becomes available.

Enjoy!

About the Author

Davi Pedro Bauer

I have 20+ years of experience in the IT market with experience in analysis and systems development. I've been working with agile methods since 2009, where I participated in agile adoption programs in multidisciplinary teams, supporting the implementation of processes and practices such as Scrum and Kanban, as well as the launch of new digital products for web and mobile platforms. Since 2016, I have been studying on topics related to Blockchain, such as cryptocurrencies, asset tokenization, smart contracts and distributed applications (DApps) and since 2019 I've been working with DevSecOps from code to infrastructure.

Follow me on Linkedin
Follow me on Github

https://leanpub.com/u/davibauer

Pre requirements

Install Blockchain Dev Kit Extension on VS Code

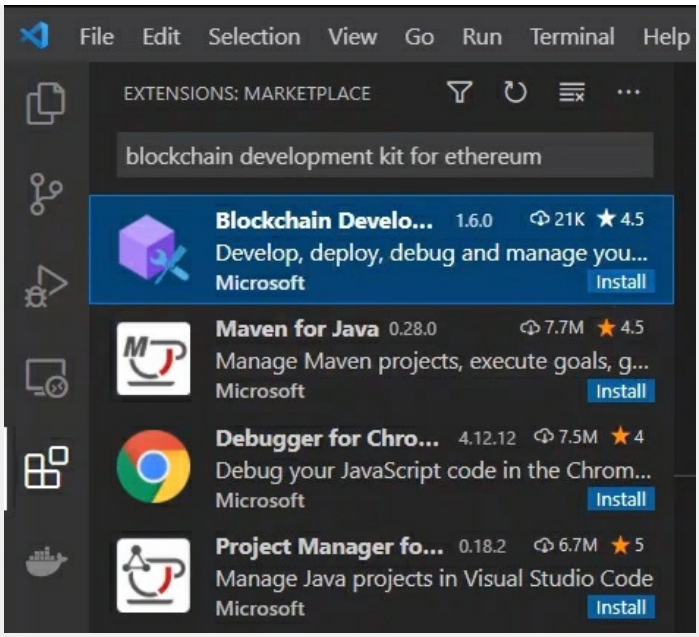
The Blockchain Developer Kit was built for both new users to Ethereum, but not get in the way for those familiar with the process. One of the primary goals is to help users create a project structure for these smart contracts, help in the compilation and building of these assets, deployment of these assets to blockchain endpoints as well as debugging of these contracts.

View on YouTube

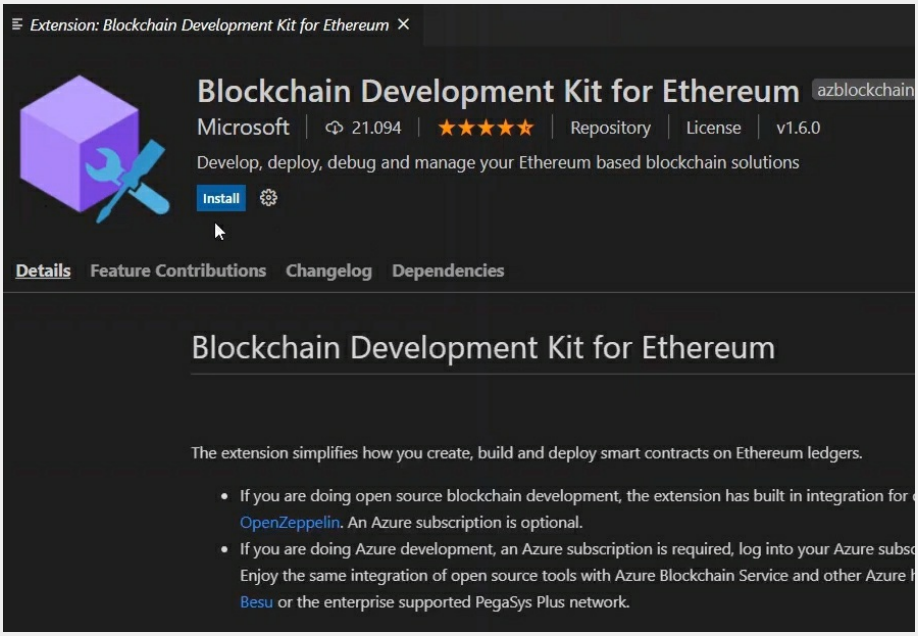
ESTIMATED TIME	1 minute

Installing the extension

Go to Extensions.
Search for “blockchain development kit for ethereum”.
Click on the extension created by Microsoft, usually will be the first one.



Click on “Install”.



Wait for the installation complete
That’s done!

Install Truffle

Truffle is a development environment, testing framework and asset pipeline for Ethereum, aiming to make life as an Ethereum developer easier.

View on YouTube

ESTIMATED TIME	2 minutes

Installing Truffle

Go to the terminal windows

VS CODE - BASH TERMINAL

TERMINAL
`$ npm install -g truffle`

Checking Truffle installation

Check if the installation was done successfully

VS CODE - BASH TERMINAL

TERMINAL
`$ truffle`

```
davi@DAVI-LATITUDE MINGW64 /c/blockchain/getstarted (master)
$ truffle
Truffle v5.2.6 - a development framework for Ethereum

Usage: truffle <command> [options]

Commands:
  build     Execute build pipeline (if configuration present)
  compile   Compile contract source files
  config    Set user-level configuration options
  console   Run a console with contract abstractions and commands available
  create    Helper to create new contracts, migrations and tests
  db        Database interface commands
  debug     Interactively debug any transaction on the blockchain
  deploy    (alias for migrate)
  develop   Open a console with a local development blockchain
  exec      Execute a JS module within this Truffle environment
  help      List all commands or provide information about a specific command
  init      Initialize new and empty Ethereum project
  install   Install a package from the Ethereum Package Registry
  migrate   Run migrations to deploy contracts
  networks  Show addresses for deployed contracts on each network
  obtain    Fetch and cache a specified compiler
  opcode    Print the compiled opcodes for a given contract
  publish   Publish a package to the Ethereum Package Registry
  run       Run a third-party command
  test      Run JavaScript and Solidity tests
  unbox     Download a Truffle Box, a pre-built Truffle project
  version   Show version number and exit
  watch     Watch filesystem for changes and rebuild the project automatically

See more at http://trufflesuite.com/docs
```

Install Ganache CLI

Ganache is a personal blockchain for rapid Ethereum and Corda distributed application development. You can use Ganache across the entire development cycle; enabling you to develop, deploy, and test your dApps in a safe and deterministic environment.

View on YouTube

ESTIMATED TIME	2 minutes

COMMAND OVERVIEW
npm install
This command installs a package and any packages that it depends on.
ganache-cli
Start the ganache development blockchain on 127.0.0.1:8545.

Installing Ganache

Go to the terminal windows

VS CODE - BASH TERMINAL

TERMINAL

```
npm install -g ganache-cli
```

Starting Ganache locally

Start Ganache CLI on 127.0.0.1:8545

VS CODE - BASH TERMINAL

TERMINAL

```
ganache-cli
```

It will also generate 10 accounts with their respectives public and private keys so that you can use it for test purposes.

```
davi@DAVI-LATITUDE MINGW64 /c/blockchain/getstarted (master)
$ ganache-cli
Ganache CLI v6.12.2 (ganache-core: 2.13.2)

Available Accounts
==================
(0) 0x6561f53E2c87E556b6f09BFf4799B3b2ea52Df3C (100 ETH)
(1) 0x4dD054592aB56a02F556801d9d25f4c21a79375a (100 ETH)
(2) 0x5D6F7D893DD4cd3f210bE411A8a080806CD497Ba (100 ETH)
(3) 0x919E866EdF19CA91abeC4Fd64A5b234180364ECD (100 ETH)
(4) 0x164b46F6ED93D3aaAb02258FbF45d00FDB388CB7 (100 ETH)
(5) 0x1ed6CB8Dff561C7124687A8e208041ec7dA326aA (100 ETH)
(6) 0x5e574182b073B1e071852eDC5d29c4591A14564E (100 ETH)
(7) 0xC080e81AB25f6790a3D0bFfab17e86B557e74d41 (100 ETH)
(8) 0x78759E4c5E0b8803D77fC46FBA0e15Ad85D00B42 (100 ETH)
(9) 0x91B49E68614E36CC6FdcDc22bA1d3207c0e9cf52 (100 ETH)
```

MetaMask

Install and Setup MetaMask Wallet

MetaMask is an extension for accessing Ethereum enabled distributed applications, or "Dapps" in your browser. The extension injects the Ethereum web3 API into every website's javascript context, so that dApps can read from the blockchain. MetaMask also lets the user create and manage their own identities (via private keys, local client wallet and hardware wallets like Trezor), so when a Dapp wants to perform a transaction and write to the blockchain, the user gets a secure interface to review the transaction, before approving or rejecting it.

View on YouTube

ESTIMATED TIME	2 minutes

Installing the wallet

Go to metamask.io
Click on "Install MetaMask"
Click on "Add to Brave" or your browser name
Click on "Add extension"
Click on "Get Started"

Configuring the wallet

Click on "Create a Wallet"

Click on "No Thanks" (but if you prefer, click on "I Agree" instead)
Define the password that you will use to open your wallet
Confirm the password
Agree to the terms of use
Click on "Create"
Now you can backup your secret backup phrase (you can also do it later)
For now, click on "Remind me later"
Your wallet is done!

Accessing your wallet

Click on extensions and pin MetaMask to your bar
Click on MetaMask icon, your wallet will be shown

Discovering your wallet address

Click on the three dots on the upper right side and then on "Account details"
You can see your wallet address in hash format and in a QR code format
You can also copy your wallet address by clicking on account name
That's all!

Infura

Create an Account on Infura

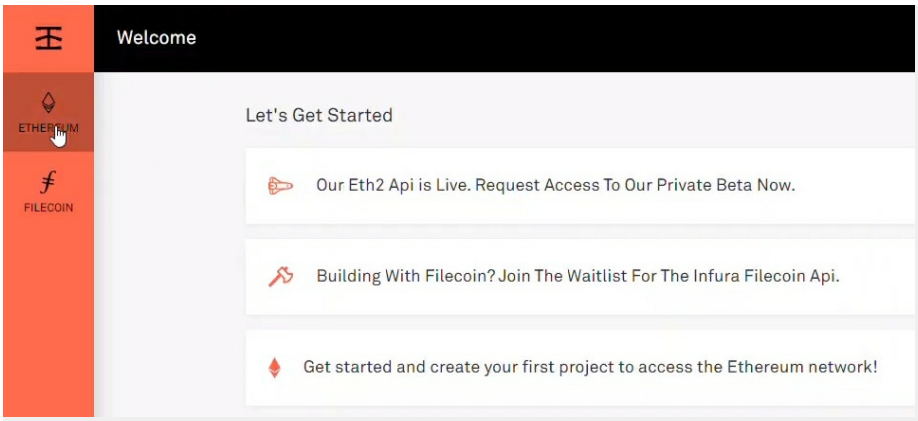
Infura provides the tools and infrastructure that allow developers to easily take their blockchain application from testing to scaled deployment - with simple, reliable access to Ethereum and IPFS.

View on YouTube

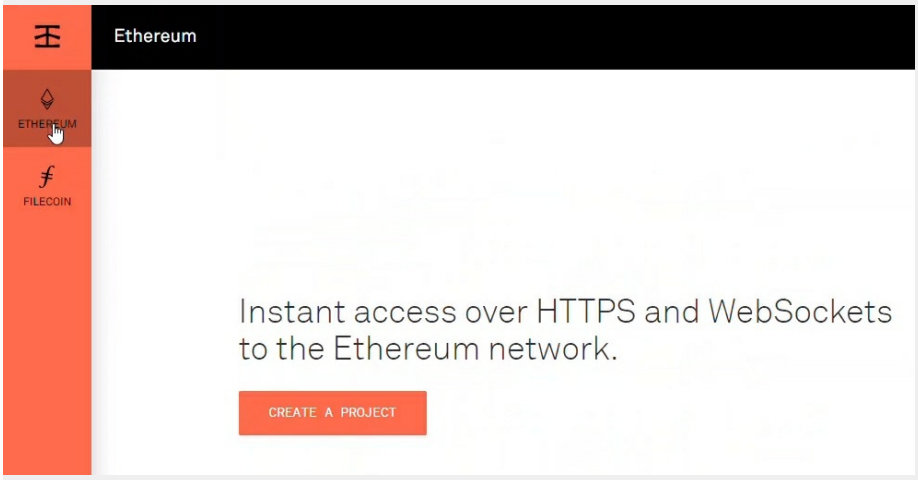
ESTIMATED TIME	2 minutes

Creating a new account

Go to infura.io
Click on “Get Started for Free”.
Enter your email and password
Click on “Sign Up”
A verification email will be send to your address
Check your email and confirm your account by clicking the verification link
After that, you will be redirected to your dashboard
Click on “Ethereum” tab on the left side menu



Now your account was created and you can start setting up a new project.



Setting up your infura project

Go to infura.io
Access your dashboard
Click on "ethereum"
Click on "create a project"
Define the project name
You can connect with different testnets and also to the mainnet
Save changes

PROJECT DETAILS

NAME*

MyCoin

SAVE CHANGES

KEYS

PROJECT ID

7f7261824605444497585504fbfde984

PROJECT SECRET

ENDPOINTS Mainnet

https://mainnet.infura.io/v3/7f7261824605444497585504fbfde984

wss://mainnet.infura.io/ws/v3/7f7261824605444497585504fbfde984

Solidity

Get Started With Solidity Project on VS Code

Smart contracts are most commonly used with Ethereum. Ethereum is the world's first programmable blockchain. It allows smart contracts to be defined to help facilitate the transfer of digital assets, like ether.

The language you'll use to write contracts is Solidity. Solidity is Turing-complete, which means that you can write complicated contracts in a clearly defined and coded way.

View on YouTube

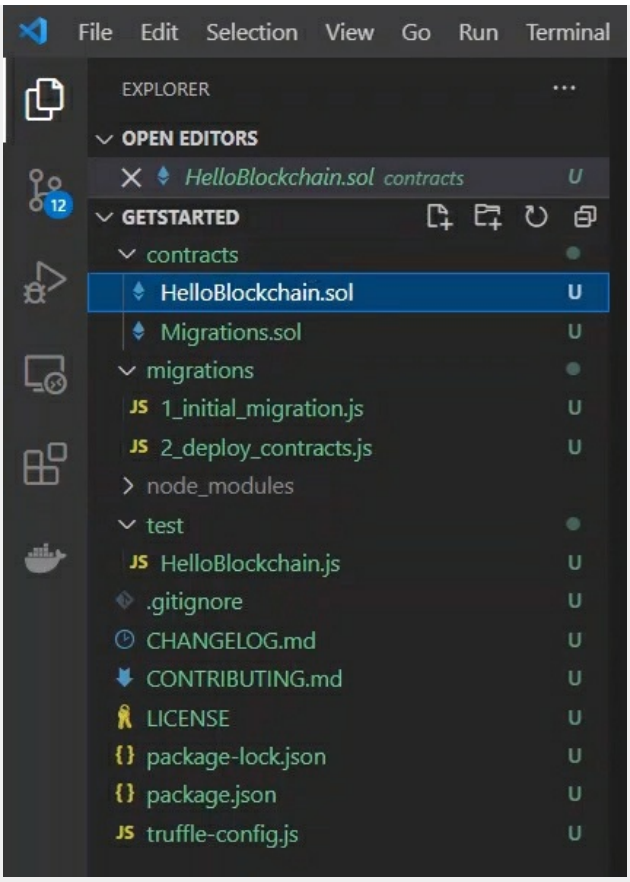
ESTIMATED TIME	4 minutes

Creating a new project

Click on “View > Command Palette...”.
Click on “Blockchain: New Solidity Project”.
Click on “Create basic project”
Select a folder where the project will be scaffolding.
Wait for the project to be created.

Compiling the project

Right click on HelloBlockchain.sol file



Select “Build Contracts”.
Wait for contracts to be built.

Deploying to development blockchain

Right click on HelloBlockchain.sol file.
Select “Deploy Contracts”.
Select “development 127.0.0.1:8545”.
Wait for the contracts to be deployed to the blockchain development network.
That’s done!

ERC20 - Fungible Tokens

Write a Simple ERC20 Token Using OpenZeppelin

Let’s create a simple ERC-20 ethereum smart contract, with the help of Truffle, and then import the OpenZeppelin contracts library.

Tokens can represent virtually anything in Ethereum:

- Reputation points in an online platform
- Skills of a character in a game
- Lottery tickets
- Financial assets like a share in a company
- A fiat currency like USD
- An ounce of gold

https://ethereum.org/en/developers/docs/standards/tokens/erc-20/

View on YouTube

ESTIMATED TIME	5 minutes

COMMAND OVERVIEW
truffle init
Initialize new and empty Ethereum project
npm init
Creates a package.json file for your project. This file contains information about the project’s packages and dependencies.

npm install

This command installs a package and any packages that it depends on.

Preparing the environment

VS CODE - BASH TERMINAL

TERMINAL

```
$ truffle init
$ npm init
$ npm install @openzeppelin/contracts
```

Writing the contract

Create a new file under contracts with the name ERC20MinerReward.sol
Add the license directive
Define the solidity minimum version
Import OpenZeppelin ERC-20 contract library
Define the contract class
Define the contract constructor
Define the contract name
Define the contract symbol

ERC20MinerReward.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^ 0.8.0 ;

import  "@openzeppelin/contracts/token/ERC20/ERC20.sol"

contract ERC20MinerReward is ERC20 {
    constructor () ERC20("MinerReward", "MRW")
}
```

8

Setting the Solidity compiler version

Copy the solidity version used in this contract
Open truffle-config.js
Uncomment the "solc" block
Set the solidity version pasting the copied value

Compiling the contract

Now is time to compile the contract

VS CODE - BASH TERMINAL

TERMINAL

```
$ truffle compile
```

The contract was compiled successfully!

Verifying the result

Noticed that a new folder build/contract was created
The brand new contract is there!

Deploy ERC20 Token to Ganache Development Blockchain

Ethereum Ganache is a local in-memory blockchain designed for development and testing. It simulates the features of a real Ethereum network, including the availability of a number of accounts funded with test Ether.

View on YouTube

ESTIMATED TIME	4 minutes

PRE REQUIREMENTS
Install Truffle

COMMAND OVERVIEW
ganache-cli
Start the ganache development blockchain on 127.0.0.1:8545.
truffle compile
Compile contract source file.
truffle migrate
Run migrations to deploy contracts.

Preparing migration

Create a new migration file
Add reference to smart contract
Add a export function to deploy the smart contract

Starting the blockchain

Open a new terminal

VS CODE - BASH TERMINAL

TERMINAL

```
$ ganache-cli
```

A new ganache blockchain is listening on 127.0.0.1:8545

Configuring the blockchain network

Open truffle-config.js
Uncomment development block from networks
Make sure host and port are corrects

Deploying the contract

VS CODE - BASH TERMINAL

TERMINAL

```
$ truffle compile
$ truffle migrate
```

The contract was deployed to ganache blockchain

A contract address was created

Adding the token to a wallet

Go the Brave browser (or any browser compatible with MetaMask)
Select the "Localhost 8585" network
Click on "Add Token"
Click on "Custom Token"
Copy the contract address
Paste on "Token Contract Address" field
The "Token Symbol" and "Decimals of Precision" is filled automatically
Click on "Next"
Click on "Add Token"
The token was added to MetaMask wallet
The token is there!

Create an ERC20 Token With Fixed Supply

ERC20 fixed supply tokens define the total tokens allowed in the smart contract. In this type of implementation, you cannot change it after being deployed on the blockchain.

https://openzeppelin.com
https://www.trufflesuite.com/truffle
https://www.trufflesuite.com/ganache
https://metamask.io

ESTIMATED TIME	12 minutes

VIDEO	YOUTUBE

SOURCE CODE	GITHUB

PRE REQUIREMENTS
Install Truffle

COMMAND OVERVIEW
truffle init
Initialize new and empty Ethereum project
npm init
Creates a package.json file for your project.
npm install
Installs a package and any packages that it depends on.

ganache-cli
Start the ganache development blockchain on 127.0.0.1:8545.
truffle compile
Compile contract source file.
truffle migrate
Run migrations to deploy contracts.

Creating the project

VS CODE - BASH TERMINAL

TERMINAL

```
$ truffle init
$ npm init
$ npm install @openzeppelin/contracts
```

Writing the contract

Create a new solidity file
Include the license declaration (it is mandatory)
Define the solidity minimum version
Import OpenZeppelin ERC-20 contract library
Define the fixed supply contract class inheriting from ERC20
Call the constructor passing the name and symbol
Assign the total supply to the sender address (who created the contract)
Override the decimals function
Set the number of decimals that this token will have

ERC20FixedSupply.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^ 0.8.0 ;

import  "@openzeppelin/contracts/token/ERC20/ERC20.sol"

contract  ERC20FixedSupply is ERC20 {
    constructor () ERC20("Fixed", "FIX"){
        _mint(msg.sender, 1000 );
    }

    function  decimals () public  view  virtual  override  returns
 (uint8){
        return  0 ;
    }
}
```

Go to truffle-config.js
Uncomment the "solc" block
Update the solidity version number

truffle-config.js

```
compilers: {
   solc: {
       version: "0.8.0" ,
       docker: true ,
       settings: {
           optimizer: {
               enabled: false ,
               runs: 200
           },
           evmVersion: "byzantium"
       }
   }
},
```

Under migrations folder, create a new file
Set the name to 2_deploy_contract.sol

Set the require method to your contract file
Export a function to deploy the contract

2_deploy_contract.sol

```
var  ERC20FixedSupply = artifacts.require( "./ERC20FixedSupply.sol" );

module .exports = function (deployer){
   deployer.deploy(ERC20FixedSupply);
}
```

Open a new terminal

VS CODE - BASH TERMINAL

TERMINAL

```
$ ganache-cli
```

Starting Ganache development blockchain

Split the terminal view

VS CODE - BASH TERMINAL

TERMINAL

```
$ ganache-cli
```

Go to truffle-config.js
Under networks, uncomment the development block

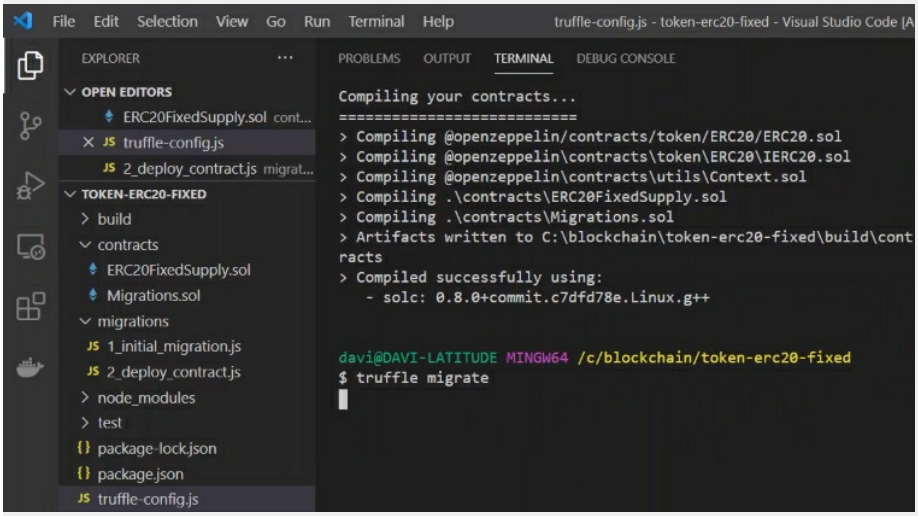
truffle-config.js

```
networks: {
   development: {
       host: "127.0.0.1" ,
       port: 8545 ,
       network_id: "*"
   },
```

Migrating the contract

VS CODE - BASH TERMINAL

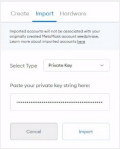
TERMINAL
`$ truffle migrate`



Copy the private key of the account that deployed the token

Configuring MetaMask

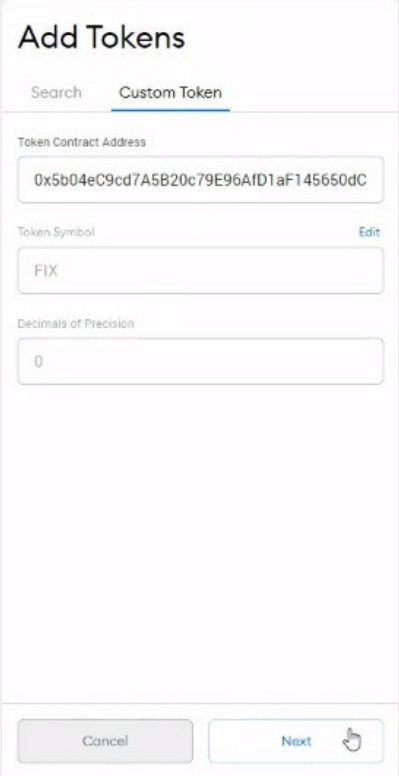
Open MetaMask
Click on your account and then "import account"
Paste the account private key
Click on "import"



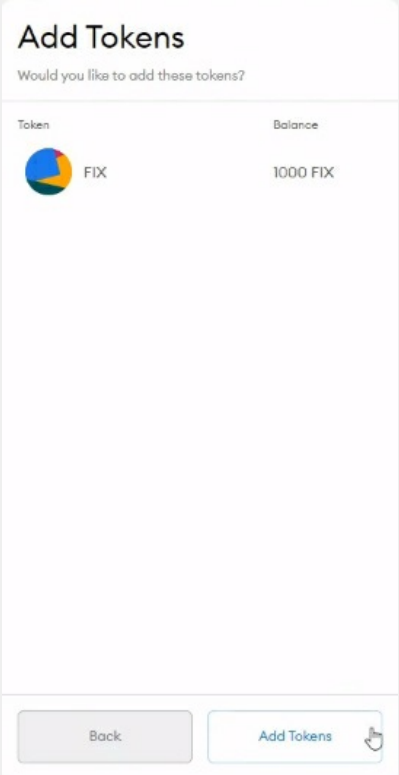
Click on networks list
Click on "Localhost:8545"

Adding the token

Click on "add token"
Select "custom token"
Paste the token contract address
Click on "next"



Click on "add tokens"



Go back to vs code
Copy another account private key
Go back to MetaMask
Repeat the steps you did for the first account including adding the token

Transferring tokens between accounts

Now change to the first imported account (the one that has all the tokens)
Click on "send"
Click on "transfer between my accounts"
Select the second created account
Input 115 FIX as the amount to transfer
Click on "next"
Click on "confirm"
The transaction was sent but it's in pending state
The transaction was confirmed

The total tokens was updated
Select the second imported account
Now this account has 115 FIX!

Deploy ERC20 Token to Testnet using Infura

You can use infura to deploy your smart contracts to test networks such as Ropsten, Kovan, Rinkeby, Gorli and also to the Mainnet. For testnet you will need to setup a new project on infura as well as access to the private key of the wallet that you will use to deploy the contracts. This wallet needs to have ether balance to complete the contract creation transaction.

https://infura.io

View on YouTube

ESTIMATED TIME	8 minutes

PRE REQUIREMENTS
Install Truffle
Create an Account on Infura.io
Get Test Ether From Faucet on Ropsten Network
Install and Setup MetaMask Wallet

COMMAND OVERVIEW
npm install fs
Install the fs package that provides a lot of very useful functionality to access and interact with the file system.
npm install @truffle/hdwallet-provider
Install the HD Wallet-enabled Web3 provider. This is used to sign transactions for addresses derived from a 12 or 24 word mnemonic.

```
truffle migrate --network ropsten
```

Run migrations to deploy contracts.

Installing the pre-requirements

Open a new terminal

VS CODE - BASH TERMINAL

```
TERMINAL
$ npm install fs
$ npm install @truffle/hdwallet-provider
```

Setting up your infura project

Go to infura.io
Access your dashboard
Click on "ethereum"
Click on "create a project"
Define the project name
You can connect with different testnets and also to the mainnet
Copy the project id
Save changes

Setting up your smart contract

Go to visual studio code
Open truffle-config.js
Uncomment the four constants: hdwalletprovider, infurakey, fs and mnemonic
Paste the project id as a value for infurakey constant
Uncomment the "ropsten" block

Make sure you are using the correct project id in the ropsten endpoint

Configuring the private key

Go to the browser and open your MetaMask wallet connected to infura network
Click on your account
Click on "settings"
Click on "security & privacy"
Click on "reveal seed phrase"
Enter your wallet password to continue
Copy the private key
Go back to visual studio code
Create a new file named ".secret"
Paste the private key

Deploying the smart contract

VS CODE - BASH TERMINAL

TERMINAL

```
$ truffle migrate --network ropsten
```

Checking your wallet balance

Go to your Metamask wallet again
Notice that your balance has been reduced

Verifying the smart contract on etherscan

Open a new window
Go to https://ropsten.etherscan.io
Copy the contract address that was created for your deploy
Paste the contract address in the search field
Click on the "find" button
The smart contract is there!
The tokens were created and transferred to the wallet that created the contract
Click on "Fixed (FIX)" token link
Here you can see the overview of your newly created token!

Unit Tests

Write Unit Tests for ERC20 Smart Contracts

Truffle comes standard with an automated testing framework to make testing your contracts much easier. This framework lets you write simple and manageable tests in different ways.

https://www.trufflesuite.com/truffle
https://www.trufflesuite.com/ganache

View on YouTube

ESTIMATED TIME	5 minutes

PRE REQUIREMENTS
Install Truffle
Install Ganache CLI

COMMAND OVERVIEW
truffle create test
Helper to create new contracts, migrations and tests.
truffle test --network development
Run JavaScript and Solidity tests.

Creating a new unit test file

Open a new terminal and execute the command.

TERMINAL

```
$ truffle create test erc20FixedSupply
```

Writing test for the contract total supply

Write a new test to assert that the contract was created with a fixed supply of 1000 coins.

ERC20FixedSupply.js

```
const erc20FixedSupply = artifacts.require( "erc20FixedSupply" );

contract( "erc20FixedSupply" , function () {
   it( "should assert true" , async function () {
        await erc20FixedSupply.deployed();
        return assert.isTrue( true );
   });

   it( "should return total supply of 1000" , async function () {
        const instance = await erc20FixedSupply.deployed();
        const totalSupply = await instance.totalSupply();

       assert.equal(totalSupply, 1000 );
   });
});
```

Test using the following command.

TERMINAL

```
$ truffle test --network development
```

The test will pass.

Writing test asserting for the contract balance

Add one more test to assert that the balance is correct after a new transfer is made between two accounts.

ERC20FixedSupply.js

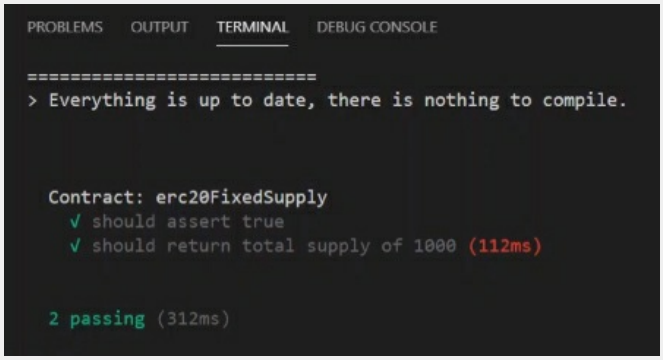
```
16     it( "should transfer 150 FIX" , async  function (){
17         const  instance = await  erc20FixedSupply.deployed();
18         await  instance.transfer( "account[1]" , 150 );
19
20         const  balanceAccount0 = await  instance.balanceOf(accounts[ 0
21 ]);
22         const  balanceAccount1 = await  instance.balanceOf(accounts[ 1
23 ]);
24
25        assert.equal(balanceAccount0.toNumber(), 850 );
          assert.equal(balanceAccount1.toNumber(), 150 );
    });
```

Execute the test again.

TERMINAL

```
$ truffle test --network development
```

All tests will pass.



Try to change some values like the account balance at line 23

and 24 and see how the results change from pass to fail.

ERC721- Non-Fungible Tokens

Create your art NFT using Ganache and OpenZeppelin

An NFT is a digital asset that represents real-world objects like art, music, in-game items and videos. In this video I'll show you how to create a NFT ERC 721 and deploy it to ethereum testnet network as well as how to add it to your metamask mobile wallet.

https://eips.ethereum.org/EIPS/eip-721

View on YouTube

ESTIMATED TIME	15 minutes

PRE REQUIREMENTS
Install Truffle
Create an Account on Infura.io
Get Test Ether From Faucet on Rinkeby Network
Install and Setup MetaMask Wallet

COMMAND OVERVIEW
truffle init
Initialize new and empty Ethereum project

npm install
This command installs a package and any packages that it depends on
ipfs daemon
Start IPFS local server on 127.0.0.1:5001
ipfs add <file>
Add file to your IPFS local node
ipfs pin remote add --service=pinata -name=<file> <hash>
Pin your file to the remote IPFS pinning service
truffle compile
Compile contract source file
truffle migrate
Run migrations to deploy contracts
truffle console
A basic interactive console connecting to an Ethereum client

Creating the project

Create a new project using truffle

TERMINAL
`$ truffle init`

Install open zeppelin contracts

TERMINAL

```
$ npm install @openzeppelin/contracts
```

Create a new solidity smart contract

TERMINAL

```
$ touch contracts/UniqueAsset.sol
```

Open the file
Import ERC721 URI Storage extension
Import Counters util
Create a new class extending ERC721URIStorage
Declare counters
Declare the constructor passing the coin name and the code
Create a new method for award item
Inside the new method, increment the token
Get the new token number
Mint a new item
Set the token URI

UniqueAsset.sol [View on GitHub]

```
// SPDX-License-Identifier: MIT
pragma solidity ^ 0.8.0 ;

import  "@openzeppelin/contracts/token/ERC721/extensions/ERC7
;
import  "@openzeppelin/contracts/utils/Counters.sol" ;

contract  UniqueAsset is  ERC721URIStorage {
    using  Counters for  Counters.Counter;
   Counters.Counter private _tokenIds;

    constructor () ERC721("UniqueAsset", "UNA") {}

    function  awardItem (address recipient, string memory meta
    public
    returns  (uint256)
   {
```

```
        _tokenIds.increment();
        uint256 newItemId = _tokenIds.current();
        _mint(recipient, newItemId);
        _setTokenURI(newItemId, metadata);
         return  newItemId;
    }
}
```

Create a new migration file

TERMINAL

```
$ touch migrations/2_deploy_contracts.js
```

Export the smart contract in the migration file

2_deploy_contracts.js | View on GitHub

```
const  UniqueAsset = artifacts.require( "UniqueAsset" );

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

Configuring the wallet

Install the file system “fs” package

TERMINAL

```
$ npm install fs
```

Install the wallet provider “hdwallet” package

TERMINAL

```
$ npm install @truffle/hdwallet-provider@1.2.3
```

Open truffle-config.js file
Uncomment the HDWalletProvider code section

truffle-config.js

```
const HDWalletProvider = require ( '@truffle/hdwallet-provider' );
const infuraKey = 'fj4jll3k.....' ;

const fs = require ( 'fs' );
const mnemonic = fs.readFileSync( ".secret" ).toString().trim();
```

Paste your Infura project id as a value for the variable infuraKey

Configuring the network

Uncomment the “ropsten” network section
Change “ropsten” to “rinkeby”
Change “ropsten” infura url to “rinkeby”
Change “YOU-PROJECT-ID” to “${infuraKey}”
Change the network_id to 42

truffle-config.js

```
rinkeby: {
   provider: () => new HDWalletProvider(mnemonic,
`https://rinkeby.infura.io/v3/${infuraKey}` ),
   network_id: 4 ,
   gas: 5500000 ,
   confirmations: 2 ,
   timeoutBlocks: 200 ,
   skipDryRun: true
},
```

Configuring the solidity compiler

Uncomment the “compilers” section
Change version to “0.8.0”

truffle-config.js

```
compilers: {
    solc: {
        version: "0.8.0" ,
        docker: true ,
        settings: {
            optimizer: {
                enabled: false ,
                runs: 200
            },
            evmVersion: "byzantinum"
        }
    }
},
```

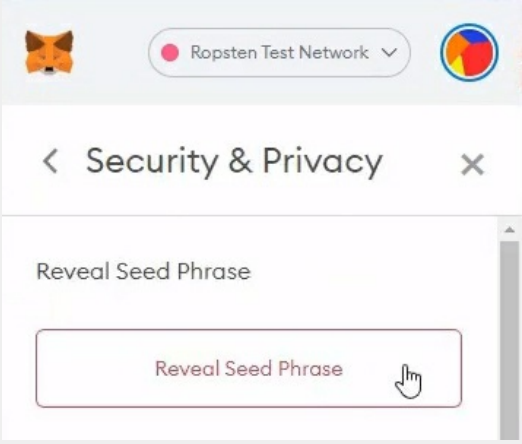
Configuring the private key

Create the secret file

TERMINAL

```
$ touch .secret
```

Go to the browser and open your MetaMask wallet connected to infura network
Click on your account
Click on "settings"
Click on "security & privacy"



Click on "reveal seed phrase"
Enter your wallet password to continue
Copy the private key
Paste the wallet mnemonic secret

Creating the badge image

Create the badge folder

TERMINAL

```
$ mkdir badge
```

Go to the badge root folder

TERMINAL

```
$ cd badge
```

Download the image that you will use as a badge from the internet. You can also copy and paste an existing image in this folder.

TERMINAL

```
curl
https://planouhost.z15.web.core.windows.net/badge.png >
```

```
badge-image.png
```

Adding the badge to your local IPFS

Initialize your local IPFS node

TERMINAL

```
$ ipfs daemon
```

Add your badge image to IPFS

TERMINAL

```
$ ipfs add badge-image.png
```

Running this command, you will receive a hash. This hash is your image address in IPFS.

```
$ ipfs add badge-image.png
 20.68 KiB / 20.68 KiB [=============================================
added QmZPxKJWqJTdudyaZUyf6uBzwwAT41QQyxhTHmMZWB9yx4 badge-image.png
 20.68 KiB / 20.68 KiB [=============================================
```

Pinning the badge to a remote IPFS node

Pin your badge using a remote IPFS service.

TERMINAL

```
ipfs pin remote add --service=pinata --name=badge-image.png
QmZPxKJWqJTdudyaZUyf6uBzwwAT41QQyxhTHmMZWB9yx4
```

You will get a response indicating the file was pinned successfully

```
$ ipfs pin remote add --service=pinata --name=badge-image.png QmZPxKJWqJTdudyaZUyf6uBzwwAT41QQyxhTHmMZWB9yx4
CID:    QmZPxKJWqJTdudyaZUyf6uBzwwAT41QQyxhTHmMZWB9yx4
Name:   badge-image.png
Status: pinned
```

Creating the badge metadata

Create the badge metadata json file

TERMINAL

```
touch badge-metadata.json
```

Open the file
Set the badge name
Set the badge description
Set the badge image address using a IPFS gateway

badge-metadata.json

```
{
    "name" : "My badge" ,
    "description" : "My badge description" ,
    "image" :
 "https://ipfs.io/ipfs/QmZPxKJWqJTdudyaZUyf6uBzwwAT41QQyxhTHmMZWB9yx4"
 }
```

Add your badge metadata to IPFS

TERMINAL

```
$ ipfs add badge-metadata.json
```

Pin your badge metadata using a remote IPFS service

TERMINAL

```
$ ipfs pin remote add --service=pinata --name=badge-
metadata.json
QmRzcwAtLWbeYqyaZUyf6uBzwwAT41QQyxhTHmMZWBfUTa
```

Compiling the smart contract

Compile the contract using truffle

TERMINAL

```
$ truffle compile
```

Migrating the smart contract

Migrate the contract to rinkeby network using truffle

TERMINAL

```
$ truffle migrate --network rinkeby
```

Instantiate the smart contract

Instantiate the contract using using truffle console

TERMINAL

```
$ truffle console --network rinkeby
truffle(rinkeby) let instance = await
UniqueAsset.deployed()
```

Awarding badge to a wallet

Call the method “getThePrice”
Enter your public ethereum address as a first parameter
Enter the IPFS address corresponding your badge metadata

TERMINAL

```
truffle(rinkeby) let result = await
instance.awardItem("0x62761466bB3A3Da83B408B5F5fE00ac7b2
```

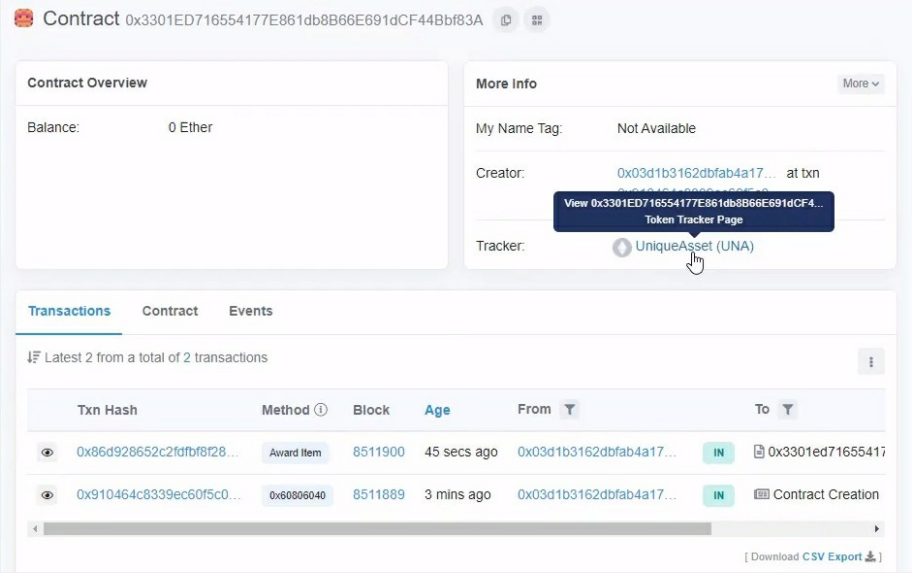
Checking badge on Etherscan

Copy the deployed contract address

```
2_deploy_contracts.js
=====================

   Deploying 'UniqueAsset'
   -----------------------
   > transaction hash:    0x910464c8339ec60f5c03eb1ab154c9135a9ee44e62a5fc206837912cf0df8ec0
   > Blocks: 1            Seconds: 9
   > contract address:    0x3301ED716554177E861db8B66E691dCF44Bbf83A
   > block number:        8511889
   > block timestamp:     1619913002
   > account:             0x03d1b3162DBFaB4A175038eAa4EA4b39423d5A6F
   > balance:             2.677347324555972908
   > gas used:            2465040 (0x259d10)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.0493008 ETH
```

Go to rinkeby.etherscan.io
Paste the contract address in the search bar

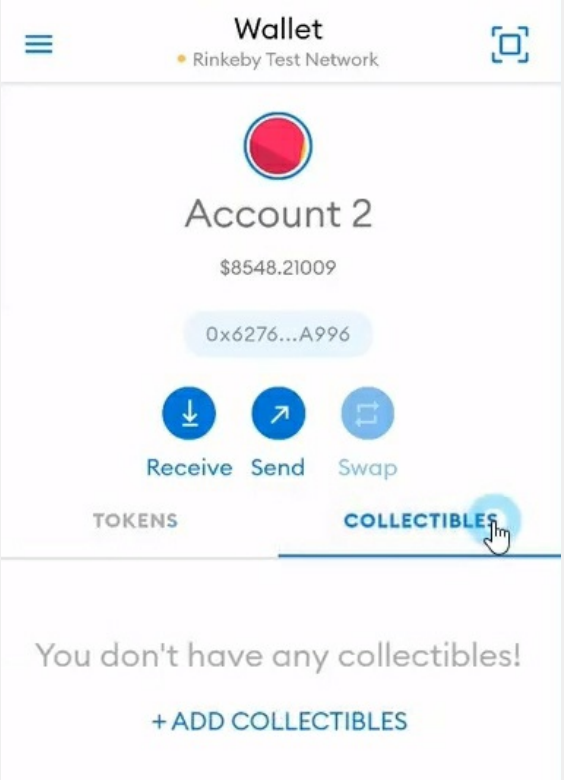
Click on search icon
Now you can see that the contract was deployed successfully



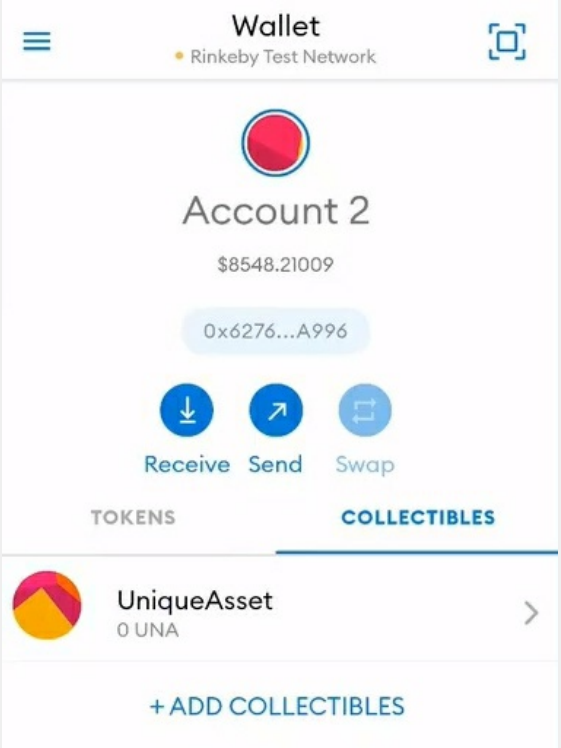
You can also realize that the last transaction made was for award a new item

Adding the NFT token to your wallet

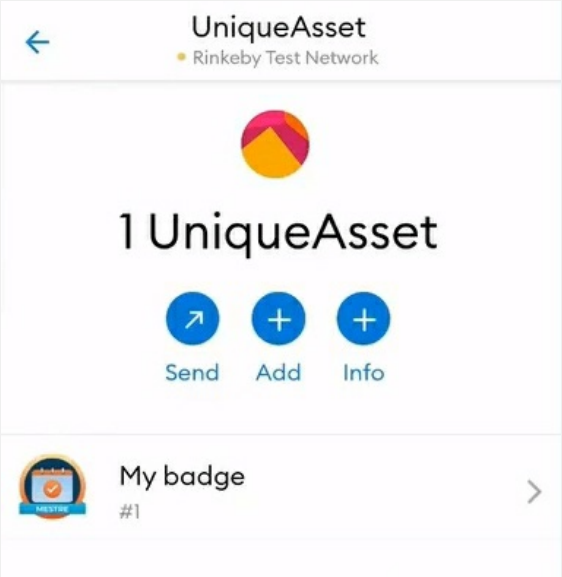
Open your MetaMask wallet on your mobile phone (collectibles are only available on mobile version)
Click on “collectibles”



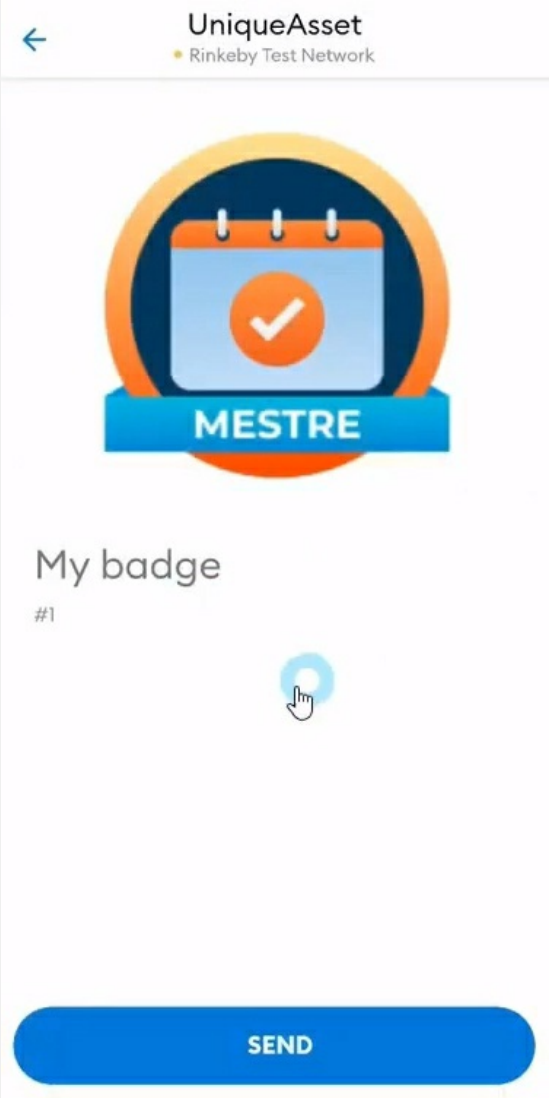
Click on "add collectible"
Paste the token contract address here (the same that you copied in the last section)
Enter the token ID, as it is the first token you will enter 1 here
Click on "add"
Wait for a few seconds
Now the NFT token was added!



Click on “UniqueAsset”
Now you can see all the badges that you earn



Click on “My Badge”
You can see the badge details now!
Also, you have a send button so that you can send it to another wallet



That’s it, you just created your first NFT token!

Ether Faucets

Get Test Ether From Faucet on Ropsten Network

This ethereum test faucet drips 1 ether every 5 seconds. You can use it to test smart contracts on the Ropsten network with no need to use real ether for this purpose (even because eth from the mainnet is not valid in ropsten and vice versa). You have a request limit of 1 eth for every 24 hours in order to avoid network spam. Ether in a test network has no real value except for testing purposes in smart contracts development.

https://faucet.ropsten.be

View on YouTube

ESTIMATED TIME	2 minutes

PRE REQUIREMENTS
Install and Setup MetaMask Wallet

Accessing the faucet

Go to https://faucet.ropsten.be
Copy your wallet address (make sure the network selected is Ropsten)
Paste your contract address in the form field
Click on "send me test ether"

Waiting for the transaction

Click on the "transaction hash" (open in a new window)
Wait for the transaction be completed
The transaction was successfully completed
Go to your MetaMask wallet, you have now 1 ether!

Get Test Ether From Faucet on Rinkeby Testnet

This Ether faucet is running on the Rinkeby network. To prevent malicious actors from exhausting all available funds or accumulating enough Ether to mount long running spam attacks, requests are tied to common 3rd party social network accounts. Anyone having a Twitter or Facebook account may request funds within the permitted limits.

https://faucet.rinkeby.io

View on YouTube

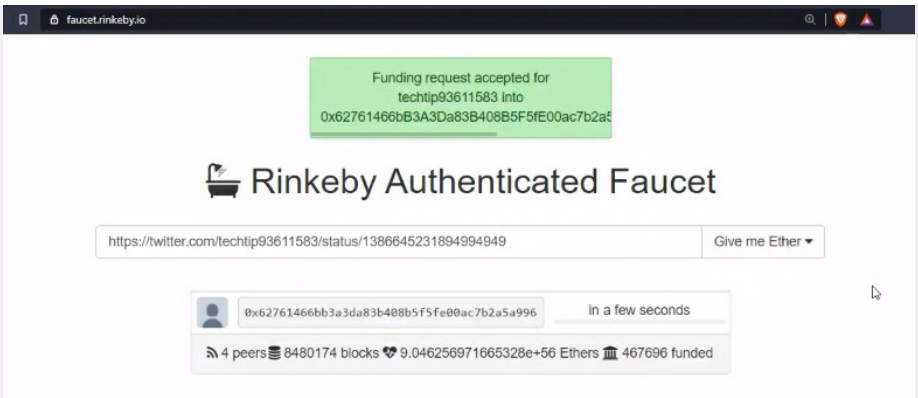
ESTIMATED TIME	2 minutes

Preparing for funding

Open your Metamask wallet
Copy you wallet address to clipboard
Go to your twitter account
Paste your wallet address
Click on your tweet
Copy your tweet address (the url in the address bar)

Funding your wallet

Go to https://faucet.rinkeby.io
Paste your tweet address in the text field
Click on “Give me Ether”
Select on of the options available (for example: 3 Ethers / 8 hours)
The request will be funding in a few seconds



Checking your wallet

Wait a few moments and check your Metamask wallet
You will get 3 ether in your wallet account!

IPFS - InterPlanetary File System

Create Your IPFS Node

The InterPlanetary File System (IPFS) is a protocol and peer-to-peer network for storing and sharing data in a distributed file system. IPFS uses content-addressing to uniquely identify each file in a global namespace connecting all computing devices.

https://ipfs.io

View on YouTube

ESTIMATED TIME	3 minutes

COMMAND OVERVIEW
choco install go-ipfs
Install the IPFS implementation in go.
ipfs init
Initialize the local IPFS repository.
ipfs daemon
Start IPFS local server on 127.0.0.1:5001.
ipfs cat <hash>
View IPFS file content.
ipfs swarm peers

Show the peers who are directly connected to your node.

Installing the node

VS CODE - BASH TERMINAL

TERMINAL

```
$ choco install go-ipfs
```

Configuring the node

VS CODE - BASH TERMINAL

TERMINAL

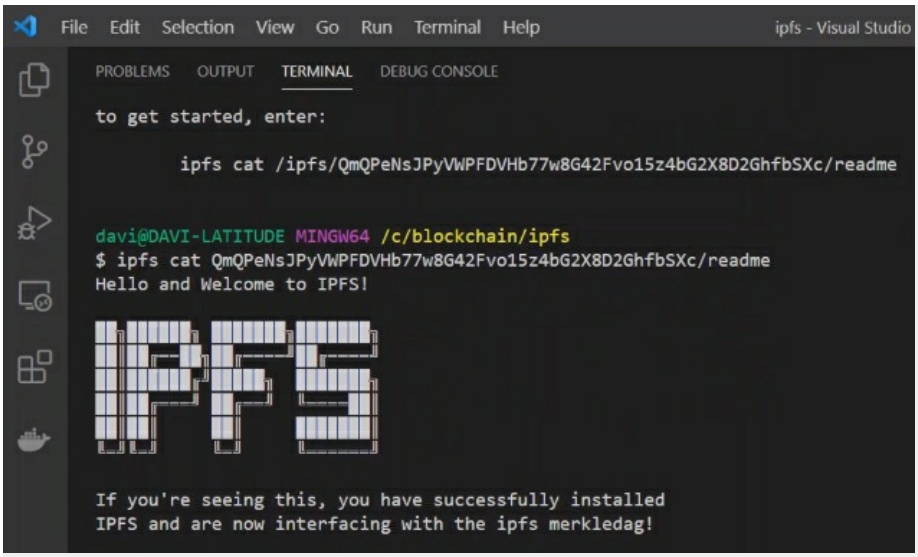
```
$ ipfs init
$ ipfs daemon
```

Testing the node

VS CODE - BASH TERMINAL

TERMINAL

```
$ ipfs cat "hash"
$ ipfs swarm peers
```



Exploring your IPFS node

Copy the WebUI link
Go to browser and paste the link
Your node is connected to IPFS!
Click on "files", no files here yet!
Click on "explore"
Click on "peers"
This are the peers that you are connected
Click on "settings"
You can see your node settings here

Add Files to IPFS

A computer running IPFS can ask all the peers it's connected to whether they have a file with a particular hash and, if one of them does, they send back the whole file. Without a short, unique identifier like a cryptographic hash, that wouldn't be possible.

https://ipfs.io

View on YouTube

ESTIMATED TIME	2 minutes

PRE REQUIREMENTS
Create Your IPFS Node

COMMAND OVERVIEW
ipfs daemon
Start IPFS local server on 127.0.0.1:5001.
echo "<text>" > <file>
Output a given text to a new file.
ipfs add <file>
Add file to your IPFS local node.
ipfs cat <hash>
View IPFS file content.

Adding the file

VS CODE - BASH TERMINAL

TERMINAL

```
$ ipfs daemon
$ echo "test"  hello.txt
$ ipfs add hello.txt
```

The file was added to ipfs resulting in a hash identifier

Viewing the file content on the console

VS CODE - BASH TERMINAL

TERMINAL

```
$ ipfs cat "hash"
```

You will see the file content be displayed

Checking the file in the web ui

Got to http://127.0.0.1:5001/webui
Click on "files"
Click on "pins"
Copy the hash
Find by this hash, you can see that this hash exists

Viewing the file content in the browser

Open a new tab
ipfs://"hash"

Now you can see your file content in the browser

Setup IPFS Browser Extension

IPFS Companion allows you locally running IPFS node directly inside your favorite browser, enabling support for ipfs:// addresses, automatic IPFS gateway loading of websites and file paths, easy IPFS file import and sharing, and more.

https://ipfs.io

View on YouTube

ESTIMATED TIME	3 minutes

PRE REQUIREMENTS
Create Your IPFS Node

COMMAND OVERVIEW
ipfs daemon
Start IPFS local server on 127.0.0.1:5001.

Installing the browser extension

Go to [IPFS Companion Extension]
Click on “Add to Brave” or the name of your browser
Click on “Add extension”
Click on extensions icon and pin “IPFS Companion” to extensions bar

Configuring the node type

Click on “IPFS Companion” icon
Click on “gear” icon
On “IPFS Node Type” select “External”

Starting an external node

Go to Visual Studio Code
Open a new terminal

VS CODE - BASH TERMINAL

TERMINAL

```
$ ipfs daemon
```

Importing a file

Click on “IPFS Companion” icon
Click on “Import”
Click on “Pick a file”
Select a file from your local disc
The file will be stored in your ipfs node

Pin and Unpin IPFS Files on Local Node

To ensure that data persists on IPFS, and is not deleted during garbage collection, data can be pinned to one or more IPFS nodes. Pinning gives you control over disk space and data retention. As such, you should use that control to pin any content you wish to keep on IPFS indefinitely. The default behavior for IPFS is to pin files to your local IPFS node.

https://ipfs.io

View on YouTube

ESTIMATED TIME	3 minutes

PRE REQUIREMENTS
Create Your IPFS Node

COMMAND OVERVIEW
ipfs daemon
Start IPFS local server on 127.0.0.1:5001.
echo "<text>" > <file>
Output a given text to a new file.
ipfs add <file>
Add file to your IPFS local node.
ipfs cat <hash>

View IPFS file content.

Starting your local node

VS CODE - BASH TERMINAL

TERMINAL

```
$ ipfs daemon
```

Adding file to your node

VS CODE - BASH TERMINAL

TERMINAL

```
$ echo "world"  > hello.txt
$ ipfs add hello.txt
```

When you add a file, this is automatically pinned to your local node.

Checking the file was added

VS CODE - BASH TERMINAL

TERMINAL

```
$ ipfs cat your_file_hash
```

Verifying your file was pinned

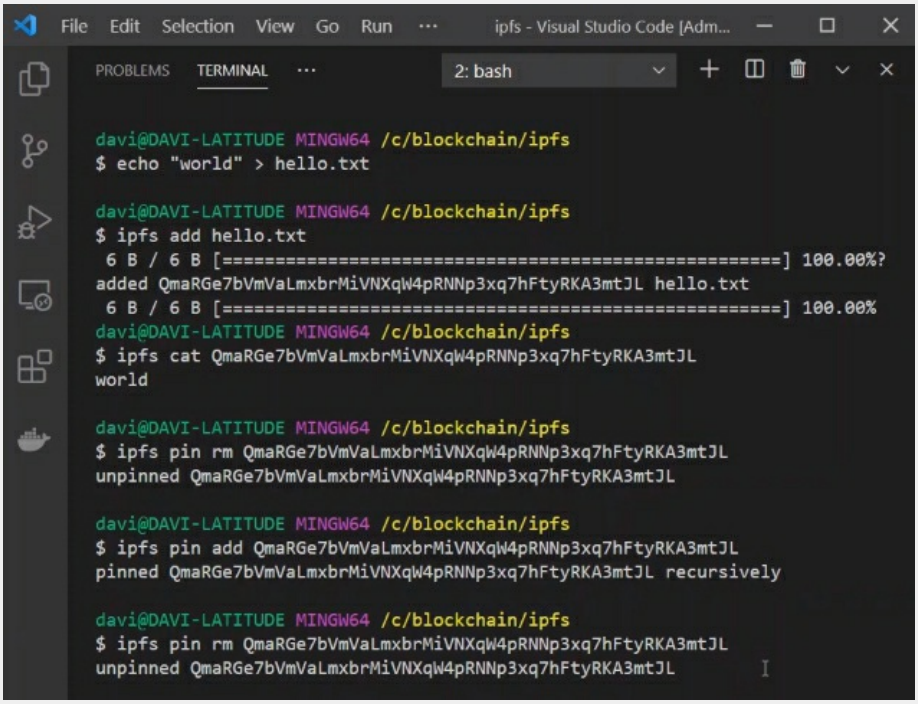
Go to http://127.0.0.1:5001/webui
Click on “Files”
Click on “Pins”
Your file is there!

Unpinning your file

VS CODE - BASH TERMINAL

TERMINAL

```
$ ipfs pin rm your_file_hash
```



Pinning your file manually

VS CODE - BASH TERMINAL

TERMINAL

```
$ ipfs pin add your_file_hash
```

Your file was pinned again

Pin and Unpin Files on Remote Node using Pinata

It's also possible to pin your files to a remote pinning service. These third-party services give you the opportunity to pin files not to your own local node, but to nodes that they operate. You don't need to worry about your own node's available disk space or uptime.

While you can use a remote pinning service's own GUI, CLI, or other dev tools to manage IPFS files pinned to their service, you can also work directly with pinning services using your local IPFS installation — meaning that you don't need to learn a pinning service's unique API or other tooling.

https://ipfs.io
https://pinata.cloud

View on YouTube

ESTIMATED TIME	4 minutes

PRE REQUIREMENTS
Create Your IPFS Node

COMMAND OVERVIEW
ipfs pin remote service add pinata <endpoint> <jwt_key>
Add a new remote pinning service.
ipfs pin remote service ls
List existing pinning services added.

ipfs add <file>
Add file to your IPFS local node.
ipfs cat <hash>
View IPFS file content.
ipfs pin remote add --service=pinata -name=<file> <hash>
Pin your file to the remote IPFS pinning service.
ipfs pin remote rm --service=pinata -name=<file> <hash>
Unpin your file from the remote IPFS pinning service.

Setting up API Keys on Pinata

Log in on your Pinata account
Go to “API Keys”
Click on “New Key”
Check “Admin”
Use “admin-cli” as your “Key Name”
Click on “Create Key”
A new key will be generated to you
Copy the “JWT” value from this window

Setting up Pinata as a remove service on your terminal

Add Pinata as a pinning remote service

VS CODE - BASH TERMINAL

TERMINAL

```
$ ipfs pin remote service add pinata https://api.pinata.cloud/psa your_jwt_key
```

List all existing remote services and check that Pinata is there

TERMINAL

```
$ ipfs pin remote service ls
```

Adding a new file to your local IPFS node

Add Pinata as a pinning remote service

VS CODE - BASH TERMINAL

TERMINAL

```
$ echo "world"  > hello.txt
$ ipfs add hello.txt
$ ipfs cat your_file_hash
```

Pinning your file to the remote IPFS node

TERMINAL

```
$ ipfs pin remote add --service=pinata -name=hello.txt your_file_hash
```

Go back to Pinata website
Click on “Pin Manager”
Your file will appear on this page!

Unpinning your file from the remote IPFS node

TERMINAL

```
$ ipfs pin remote rm --service=pinata -name=hello.txt
your_file_hash
```

Go back to Pinata website
Click on “Pin Manager”
Your file will no longer appear on this page, that means your file was unpinned.

Host Your Site on IPFS Using Fleek

Fleek allows you to build on Open Web protocols and a base layer infrastructure powered by them. Build and host your sites, apps, Dapps, and other services on trustless, permissionless, and open technologies focused on creating user-controlled, encrypted, private, p2p experiences.

https://fleek.co

View on YouTube

ESTIMATED TIME	5 minutes

COMMAND OVERVIEW
git clone
Clones a repository into a newly created directory.
npm install
Install the dependencies in the local node_modules folder.
fleek login
Login in into your Fleek account.
fleek site:init
Initialize a Fleek site in your local directory.
fleek site:deploy
Deploy changes in your publish directory.

Login on Fleek

Go to https://fleek.co
Log in with your account
Go to your VS Code

Cloning your existing repository

Clone a existing repository with sample code

VS CODE - BASH TERMINAL

```
TERMINAL
$ git clone https: //github.com/johnnymatthews/random-planet-facts
```

Installing Fleek

Install Fleek command line

VS CODE - BASH TERMINAL

```
TERMINAL
$ npm install -g @fleekhq/fleek-cli
```

Login into Fleek (you will be prompted to complete the flow in your browser)

VS CODE - BASH TERMINAL

```
TERMINAL
$ fleek login
```

Initializing Fleek

Init the Fleek site in your current directory

VS CODE - BASH TERMINAL

```
TERMINAL
$ fleek site:init
```

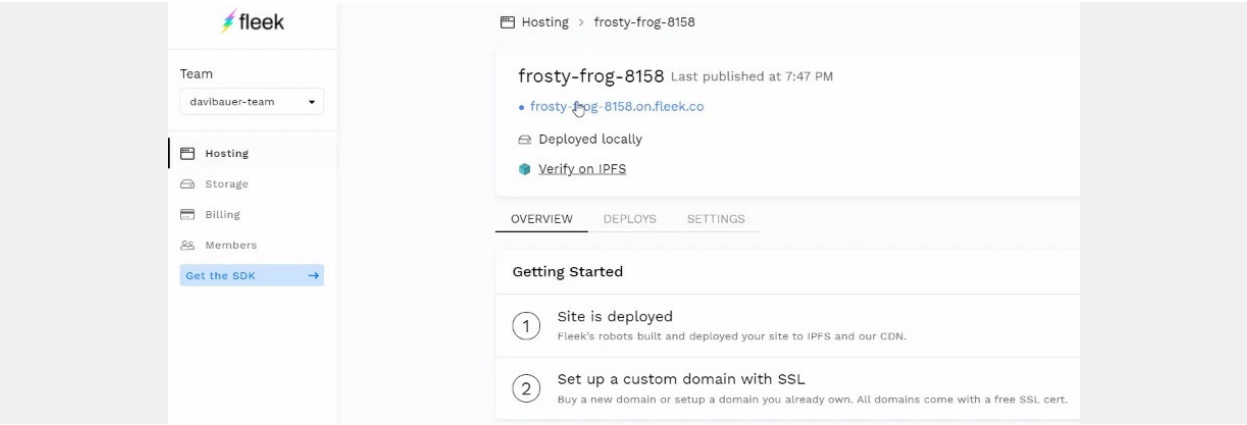
Select which team you wanna use (use the arrows key for select).
Select which site you wanna use (use the arrows key for select).
Select the public directory for deployment.

Deploying your site

VS CODE - BASH TERMINAL

```
TERMINAL
$ fleek site:deploy
```

Go back to Fleek site
Click on “Hosting”
Click on “Verify on IPFS”
This is your site hosted on IPFS
You can see now your deployed site online
Go back to “Hosting”
Click on “your-site.on.fleek.co”
This is your site host friendly address.


fleek
Team
davibauer-team
Hosting
Storage
Billing
Members
Get the SDK
Hosting > frosty-frog-8158
frosty-frog-8158 Last published at 7:47 PM
frosty-frog-8158.on.fleek.co
Deployed locally
Verify on IPFS
OVERVIEW
DEPLOYS
SETTINGS
Getting Started
1
Site is deployed
Fleek's robots built and deployed your site to IPFS and our CDN.
2
Set up a custom domain with SSL
Buy a new domain or setup a domain you already own. All domains come with a free SSL cert.

ENS - Ethereum Name Service

Register your ENS to Receive any Crypto, Token or NFT on Your Wallet

Ethereum Name Service (ENS) is a service that makes it easy to send and receive cryptocurrency and access special websites by simple names rather than by long, complex strings of letters and numbers.

https://ens.domains

View on YouTube

ESTIMATED TIME	3 minutes

PRE REQUIREMENTS
Install and Setup MetaMask Wallet

Searching your domain name

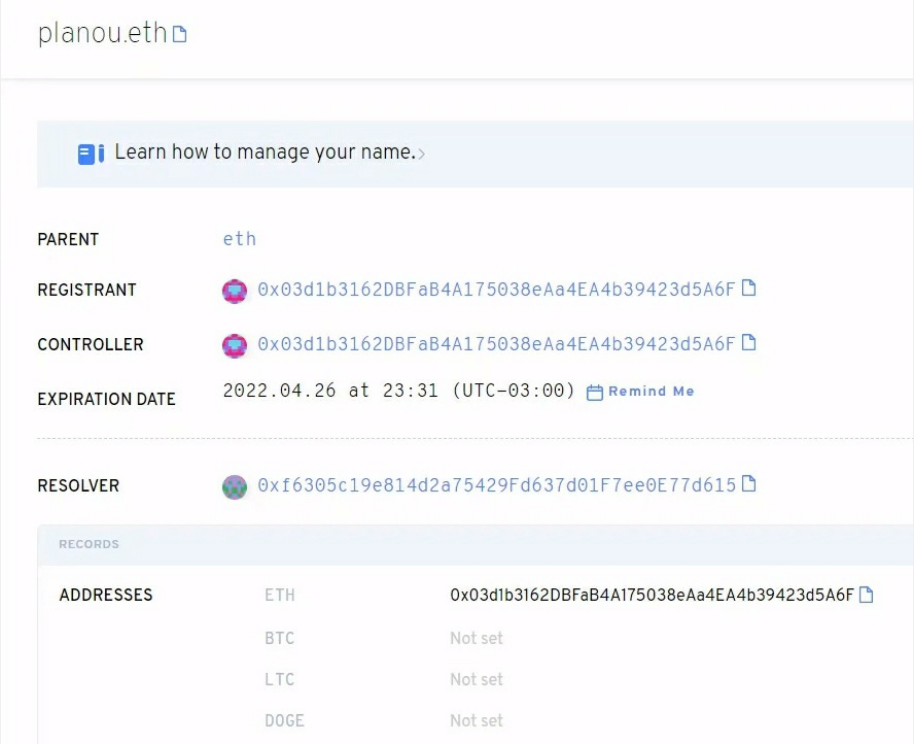
Go to ens.domains
Click on “Launch App”
Search for the domain name that you want to register (for example: planou.eth)
Check the registration period (minimum 1 year)
Check the registration price

Request to register

Click on “Request To Register”
The MetaMask notification will be open in order to confirm the transaction
Click on “Confirm”
Wait for the transaction to be confirmed on Blockchain.
Click on “Register”
A new MetaMask notification will be shown again
Click on “Confirm”
Wait for the transaction to be confirmed on Blockchain

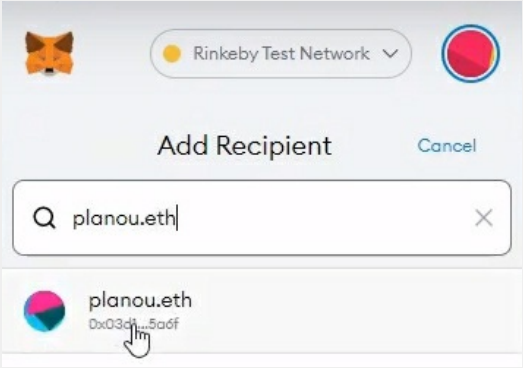
Managing your registration name

Click on “Manage name”
Scroll down to "Addresses"
Realize that the ETH address is set to the wallet that created the domain (your wallet)



Checking the name resolution

Click on your MetaMask wallet
Click on “Send”
Type your ENS name (for example: planou.eth)
Note that the name has resolved to a wallet address



Now you can use the ENS name as a recipient instead of using the wallet hash address for your transactions

Chainlink

Get Crypto Prices Inside Smart Contracts using Chainlink Oracles

Chainlink is a decentralized network of nodes that provide data and information from off-blockchain sources to on-blockchain smart contracts via oracles. In this video you will learn how to get the latest price of cryptocurrencies inside smart contracts, using the ETH/USD Price Feed on the Kovan testnet.

https://chain.link

View on YouTube

ESTIMATED TIME	10 minutes

PRE REQUIREMENTS
Install Truffle
Create an Account on Infura.io
Get Test Ether From Faucet on Kovan Network
Install and Setup MetaMask Wallet

COMMAND OVERVIEW
npm install fs
Install the fs package that provides a lot of very useful functionality to access and interact with the file system.

npm install @truffle/hdwallet-provider
Install the HD Wallet-enabled Web3 provider. This is used to sign transactions for addresses derived from a 12 or 24 word mnemonic.
truffle migrate --network kovan
Run migrations to deploy contracts.

Creating the project

Go a Terminal and click on “New Terminal”
Initialize a new truffle project

TERMINAL
`$ truffle init`

Install Chainlink contracts package

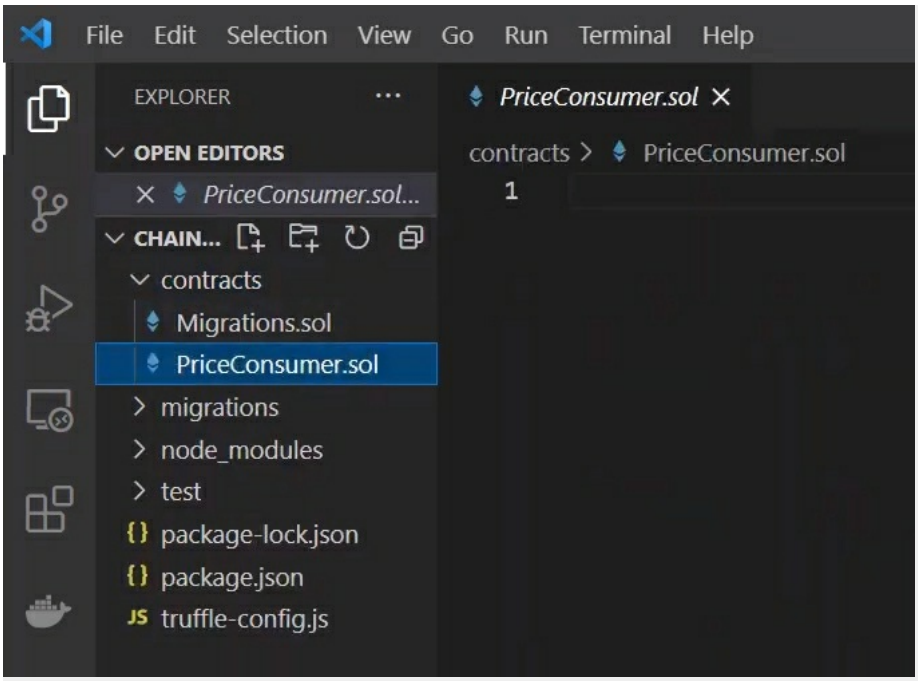
TERMINAL
`$ npm install @chainlink/contracts`

Creating the smart contract

Create a new smart contract for price consumption

TERMINAL
`$ touch contracts/PriceConsumer.sol`

Open the file “PriceConsumer.sol”



Define the Solidity version
Import chainlink contract interface
Define the contract name
Define the contract constructor

PriceConsumer.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^ 0.8.0 ;

import  "@chainlink/contracts/src/v0.6/interfaces/AggregatorV3Interface.

contract  PriceConsumer {

   aggregatorV3Interface internal  priceFeed;

    constructor (){
       priceFeed = AggregatorV3Interface()
   }
}
```

Go to docs.chain.link/docs/ethereum-addresses
Scroll down to “Kovan” section
Copy the “Proxy” address on the line “ETH/USD”

USING PRICE FEEDS

- Introduction to Price Feeds
- Get the Latest Price
- Historical Price Data
- API Reference
- Contract Addresses
  - ENS
  - Ethereum Price Feeds
  - Binance Smart Chain Price Feeds
  - Polygon (Matic) Price Feeds
  - xDai Price Feeds
  - Huobi Eco Chain Price Feeds

USING RANDOMNESS

- Introduction to Chainlink VRF
- Get a Random Number

BTC / USD	8	0x6135b13325bfC4B00278B4abC5e20bbce2D6580e
BUSD / ETH	18	0xbF7A18ea5DE0501f7559144e702b29c55b055CcB
BZRX / ETH	18	0x9aa9da35DC44F93D90436BfE256f465f720c3Ae5
CHF / USD	8	0xed0616BeF04D374969f302a34AE4A63882490A8C
COMP / USD	8	0xECF93D14d25E02bA2C13698eeDca9aA98348EFb6
CVI	18	0x0BD102ef50a6a133B38Bf3Bd3d40cE36cc1aB5A8
DAI / ETH	18	0x22B58f1EbEDfCA50feF632bD73368b2FdA96D541
DAI / USD	8	0x777A68032a88E5A84678A77Af2CD65A7b3c0775a
ENJ / ETH	18	0xfaDbe2ee798889F02d1d39eDaD98Eff4c7fe95D4
ETH / USDT RSI 4h	18	0x10900f50d1bC46b4Ed796C50A4Cc63791CaF7501
ETH / USD	8	0x9326BFA02ADD2366b30bacB125260Af641031331
EUR / USD	8	0x0c15Ab9A0DB086e062194c273CC79f41597Bbf13
Ferrari F12 TDF / USD	8	0x22a2D07993A1A18b3b86E56F960fa735b6D6cFD9

Paste the address on AggregatorV3Interface constructor
Create the function to get the price

PriceConsumer.sol

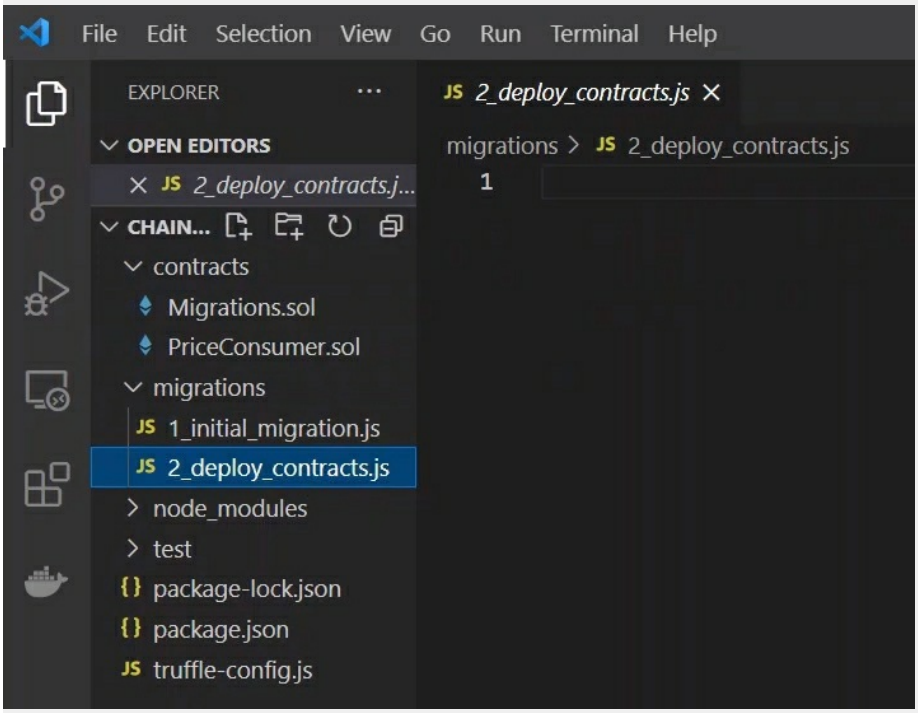
```
// SPDX-License-Identifier: MIT
pragma solidity ^ 0.8.0 ;

import  "@chainlink/contracts/src/v0.6/interfaces/AggregatorV3Interface.

contract  PriceConsumer {

   aggregatorV3Interface internal  priceFeed;
    constructor (){
       priceFeed =
AggregatorV3Interface(        0x9326BFA02ADD2366b30bacB125260Af641031331
)
   }

    function  getThePrice () public  view  returns  (int){
       (
           uint80 roundID,
           int price,
           uint startedAt,
           uint timeStamp,
           uint80 answeredInRound,
       ) = priceFeed.latestRoundDate();
        return  price;
   }
}
```

Creating the migration

Create the migration file

TERMINAL

```
$ touch migrations/2_deploy_contracts.sol
```



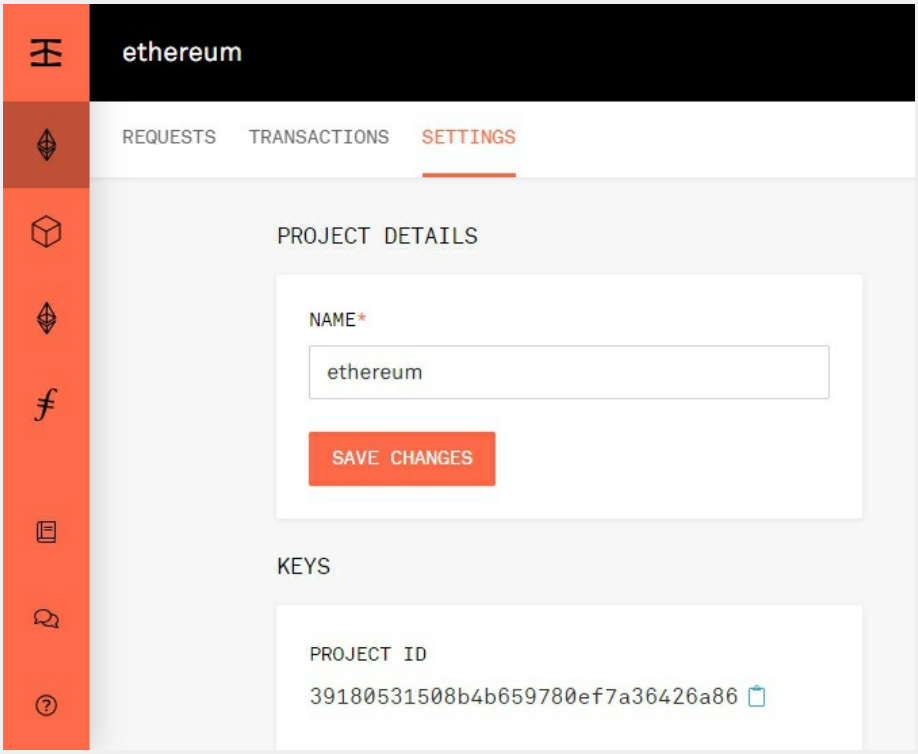
Write code to deploy the “PriceConsumer” smart contract

2_deploy_contracts.js

```
const  PriceConsumer = artifacts.require( "PriceConsumer" );

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

Setting up your infura project

Go to infura.io
Access your dashboard
Click on "ethereum"
Click on "create a project"
Define the project name
You can connect with different testnets and also to the mainnet
Copy the project id



Save changes

Configuring the wallet

Install the file system “fs” package

TERMINAL

```
$ npm install fs
```

Install the wallet provider “hdwallet” package

TERMINAL

```
$ npm install @truffle/hdwallet-provider@1.2.3
```

Open truffle-config.js file
Uncomment the HDWalletProvider code section

truffle-config.js

```
const HDWalletProvider = require ( '@truffle/hdwallet-provider' );
const infuraKey = 'fj4jll3k.....' ;

const fs = require ( 'fs' );
const mnemonic = fs.readFileSync( ".secret" ).toString().trim();
```

Paste your Infura project id as a value for the variable infuraKey

Configuring the network

Uncomment the “ropsten” network section
Change “ropsten” to “kovan”
Change “ropsten” infura url to “kovan”
Change “YOU-PROJECT-ID” to “${infuraKey}”
Change the network_id to 42

truffle-config.js

```
kovan: {
   provider: () => new  HDWalletProvider(mnemonic,
`https://kovan.infura.io/v3/${infuraKey}` ),
   network_id: 42 ,
   gas: 5500000 ,
   confirmations: 2 ,
   timeoutBlocks: 200 ,
   skipDryRun: true
},
```

Configuring the solidity compiler

Uncomment the “compilers” section
Change version to “0.8.0”

truffle-config.js

```
compilers: {
   solc: {
       version: "0.8.0" ,
       docker: true ,
       settings: {
           optimizer: {
               enabled: false ,
               runs: 200
           },
           evmVersion: "byzantinum"
       }
   }
},
```

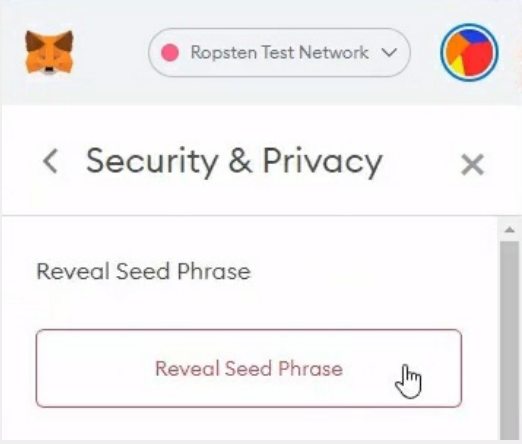
Configuring the private key

Create the secret file

TERMINAL

```
$ touch .secret
```

Go to the browser and open your MetaMask wallet connected to infura network
Click on your account
Click on "settings"
Click on "security & privacy"



Click on "reveal seed phrase"
Enter your wallet password to continue
Copy the private key
Paste the wallet mnemonic secret

Compiling the smart contract

Compile the contract using truffle

TERMINAL
`$ truffle compile`

Deploying the smart contract

Deploy the contract to kovan network using truffle

TERMINAL
`$ truffle migrate --network kovan`

Wait for the contract be deployed and the transactions be confirmed on blockchain
Confirm your contract address

```
PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

=====================

   Deploying 'PriceConsumer'
   -------------------------
   > transaction hash:    0x03e9d73d942d4f7b70f2376bc4346ebe5a8fa091173459b396362eca7f181bc7
   > Blocks: 2            Seconds: 6
   > contract address:    0xC4fBA00B693feC97cc8783768F701827e5C27B6b
   > block number:        24520560
   > block timestamp:     1619608172
   > account:             0x03d1b3162DBFaB4A175038eAa4EA4b39423d5A6F
   > balance:             1.97274432
   > gas used:            228505 (0x37c99)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.0045701 ETH

   Pausing for 2 confirmations...
   ------------------------------
   > confirmation number: 1 (block: 24520562)
   > confirmation number: 2 (block: 24520563)

   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:           0.0045701 ETH


Summary
=======
> Total deployments:   2
> Final cost:          0.00957478 ETH
```

Getting the price information from the smart contract

Instantiate the contract using using truffle console

TERMINAL

```
$ truffle console --network kovan
truffle(kovan) let instance = await
PriceConsumer.deployed()
```

Call the method “getThePrice”

TERMINAL

```
truffle(kovan) let price = await instance.getThePrice()
```

Output the result to number

TERMINAL

```
truffle(kovan) price.toNumber()
```

```
265499339990
```

That’s it, you just created a smart contract and consumed the chainlink price feed oracle!

Nethereum

Get Ether Balance using Nethereum

Nethereum is an open source .NET integration library for Ethereum, simplifying smart contract management and interaction with Ethereum nodes whether they are public or private.

https://nethereum.com

View on YouTube

ESTIMATED TIME	5 minutes

Creating the project

Go a Terminal and click on “New Terminal”
Create a new dotnet console project

TERMINAL
`$ dotnet new console -o sample`

Go to the project root directory

TERMINAL
`$ cd sample`

Installing web3

Install nethereum web3 package

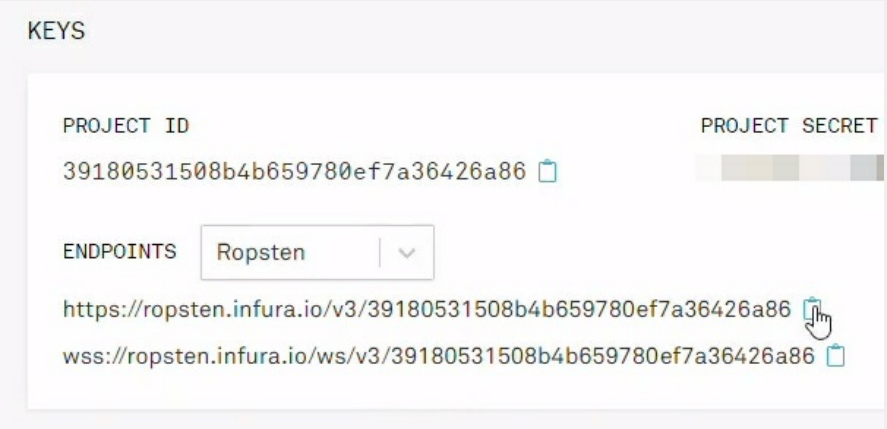
```
TERMINAL
$ dotnet add package Nethereum.Web3
```

Restore all the project packages

```
TERMINAL
$ dotnet restore
```

Creating the method

Open “Program.cs” file
Add reference for threading and web3
Add a new method for getting the account balance
Instantiate a new web3 object
Go to your infura project settings
Select “Ropsten” network
Copy Ropsten https endpoint



Use this endpoint as a parameter for web3 object constructor
Get the balance from web3
Use your wallet public address as a parameter
Write code to output the balance in Wei

Convert the Wei balance in Ether
Write code to output the balance in Ether
Now change your main method in order to call GetAccountBalance()

Program.cs

```
using System;
using System.Threading.Tasks;
using Nethereum.Web3;

namespace NethereumSample
{
    class  Program
   {
        static  void  Main(string[] args)
       {
           GetAccountBalance().Wait();
           Console.ReadLine();
       }

        static  async  Task GetAccountBalance()
       {
            var  web3 = new  Web3(
"https://ropsten.infura.io/v3/39180531508b4b659780ef7a36426a86" );
            var  balance = await  web3.Eth.GetBalance.SendRequestAsync(
"0x03d1b3162DBFaB4A175038eAa4EA4b39423d5A6F" );
           Console.WriteLine($ "Balance in Wei: {balance.Value}" );

            var  etherAmount = Web3.Convert.FromWei(balance.Value);
           Console.WriteLine($ "Balance in Ether: {etherAmount}" );
       }
   }
}
```

Getting the balance

Build the project

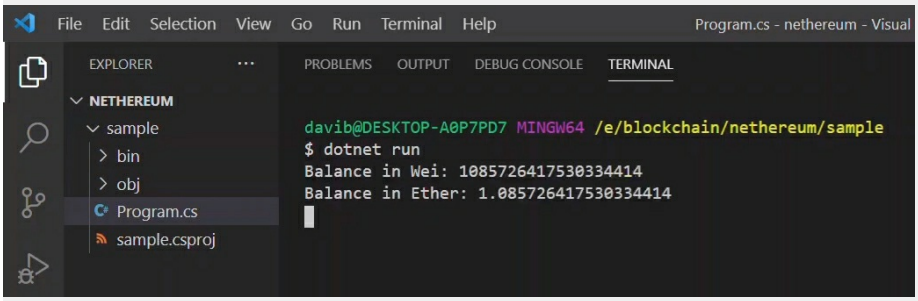
TERMINAL

```
$ dotnet build
```

Run the project

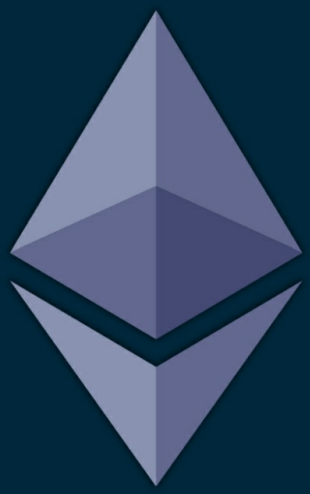
TERMINAL

```
$ dotnet run
```



GET STARTED WITH

ETHEREUM



VSCODE

TRUFFLE

GANACHE

PINATA

ENS

FLEEK

OPENZEPPELIN

CHAINLINK

IPFS

INFURA

METAMASK

OPENSEA

BLOCKCHAIN SERIES

DAVI BAUER