

Building & Deploying an NFT on Ethereum (Sepolia)

Foundry + OpenZeppelin + Alchemy — Complete Walkthrough Notes

What is this project about?

This project walks through the full process of building, deploying, and verifying a simple **ERC-721 NFT smart contract** on the Ethereum **Sepolia test network**.

The goal is not only to deploy a contract, but to understand:

- What NFTs are
- Why ERC-721 exists
- How professional tooling (Foundry) works
- How on-chain deployment actually happens

Why NFTs? (Conceptual Motivation)

Why do we use NFTs instead of normal tokens?

NFTs (Non-Fungible Tokens) represent **unique, indivisible assets**. Unlike ERC-20 tokens (which are fungible), each ERC-721 token has a unique `tokenId` and owner.

ERC-721 Use Cases

- Digital art and collectibles
- Identity or certificates
- Game items
- Proof of ownership

Key Concept Check

ERC-721 tokens are: unique and non-fungible (each token is distinct).

Step 1: Installing Foundry

What is Foundry?

Foundry is a professional Ethereum development framework used to:

- Compile Solidity contracts
- Run tests
- Deploy contracts to Ethereum networks

It replaces browser-based tools (e.g. Remix) with a local, scriptable workflow.

Install Foundry (macOS / Linux)

```
curl -L https://foundry.paradigm.xyz | bash  
foundryup
```

Why this matters

Using Foundry mirrors how smart contracts are developed in industry, not just tutorials.

Step 2: Setting Up a Foundry Project

Initialize a Foundry project

A Foundry project provides a standardized structure:

- `src/` for contracts
- `script/` for deployment scripts
- `test/` for testing

Create the project

```
forge init foundry-app  
cd foundry-app
```

Step 3: Installing OpenZeppelin Contracts

Why OpenZeppelin?

OpenZeppelin provides audited, production-grade implementations of:

- ERC-20
- ERC-721
- ERC-1155

Using OpenZeppelin avoids re-implementing security-critical logic.

Install OpenZeppelin with Foundry

```
forge install OpenZeppelin/openzeppelin-contracts  
forge remappings > remappings.txt
```

Step 4: Writing the NFT Contract

Contract Design

The contract:

- Inherits from ERC721
- Sets a name and symbol
- Mints one NFT to the deployer in the constructor

NFTVone.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;

import {ERC721} from
"openzeppelin-contracts/contracts/token/ERC721/ERC721.sol";

contract NFTVone is ERC721 {
    constructor() ERC721("NFTee", "ITM") {
        _mint(msg.sender, 1);
    }
}
```

Key Observation

`msg.sender` during deployment is the deployer's wallet address.

Step 5: Setting Up Alchemy (RPC Provider)

Why do we need an RPC provider?

Smart contracts are deployed via Ethereum nodes. Alchemy provides hosted nodes so we do not need to run our own.

Alchemy Setup

- Create a free Alchemy account
- Create an Ethereum Sepolia app
- Copy the HTTP RPC endpoint

Step 6: Environment Variables

Why use .env?

Sensitive data (RPC URLs, private keys) should never be hard-coded.

.env file

```
SEPOLIA_RPC_URL="https://eth-sepolia.g.alchemy.com/v2/XXXX"  
PRIVATE_KEY="0xYOUR_TESTNET_PRIVATE_KEY"
```

Security Note

Never use a wallet that holds real ETH. Never commit .env to GitHub.

Step 7: Deploying the Contract

Deployment Command

The following command:

- Compiles the contract
- Signs the transaction
- Broadcasts it to Sepolia

Deploy with Foundry

```
forge create \  
--rpc-url $SEPOLIA_RPC_URL \  
--private-key $PRIVATE_KEY \  
--broadcast \  
src/NFTVone.sol:NFTVone
```

Result & Verification

Deployment Result

- Deployer: 0xf47dE1c6...
- Contract Address: 0xC57DeA7AA24dD340bf5ead30D4c1b66F89D156F
- Transaction Hash: 0xf24e1d92...

How to Verify

- Open Sepolia Etherscan
- Paste the contract address or transaction hash
- View the **Transfer** event

The mint is shown as:

- **from:** 0x000...000
- **to:** deployer address
- **tokenId:** 1

Final Takeaways

- NFTs are unique on-chain assets
- OpenZeppelin prevents security mistakes
- Foundry enables professional workflows
- Deployment is an on-chain transaction

Reminder: Always separate development wallets from real-value wallets.