NFT Contract Tutorial

Introduction

Non-Fungible Tokens (NFTs) are unique digital assets, each possessing distinct identities and attributes. Unlike fungible tokens created by the Token Factory, NFTs cannot be exchanged on a like-for-like basis.

Colony NFT Seiyan NFT This tutorial guides you through the creation and deployment of an NFT contract on Sei. By the end, you'll have deployed your own NFT contract. Select one of the tabs below to get started!

EVM CosmWasm In this section, we'll use Foundry to deploy an ERC-721 contract to the Ethereum network. ERC-721 is a standard for NFT contracts on Ethereum. Learm more about ERC-721here(opens in a new tab) .

Requirements

Before we start, ensure you have:

- Installed Foundry. Follow theinstallation guide(opens in a new tab)
- .
- A basic understanding of Solidity and smart contract development.
- A wallet with SEI tokens on devnet

You can obtain devnet tokens from one of the faucets listedhere .

Setting Up Your Project

1. Initialize a new Foundry project:
2. forge
3. init
4. my-nft-project
5. cd
6. my-nft-project
7. Install OpenZeppelin, a library for secure smart contract development.
8. forge
9. install
10. OpenZeppelin/openzeppelin-contracts
11. Create a new Solidity file undersrc/
12. for your NFT contract, e.g.,MyNFT.sol
13. .

Writing the ERC-721 Contract

1. Insrc/MyNFT.sol
2. , start by importing OpenZeppelin's ERC-721 implementation:
3. // contracts/MyNFT.sol
4. // SPDX-License-Identifier: MIT
5. pragma
6. solidity
7. ^0.8.20;
8. import
9. {
10. ERC721
11. }
12. from
13. "openzeppelin-contracts/contracts/token/ERC721/ERC721.sol"
14. ;
15. contract
16. MyNFT
17. is
18. ERC721
19. {
20. constructor
21. ()
22. ERC721
23. ("MyNFT", "MNFT") {
24. }

25. }

You may see an error in your IDE about importingopenzeppelin-contracts . To resolve this, run this command to createremapping.txt in the root of your project:

forge

remappings

remappings.txt 1. Add any additional functions or overrides necessary for your NFT.

Testing Your Contract

1. Write tests for your contract in thetest/
2. directory.
3. Run your tests with:
4. forge
5. test

Deploying Your Contract

1. Compile your contract:
2. forge
3. build
4. Deploy your contract to a local testnet (e.g., using Anvil, Foundry's local Ethereum node):
5. anvil
6. -a
7. 1
8. Flags
9.
    - -a
10.
    - : flag specifies the number of test accounts to create. This command will start a local Ethereum node and
11.
    - create one account. In the command output you will see created account and its private key.
12.
    - Use this private key to deploy your contract.
13. In a new terminal, deploy your contract:
14. forge
15. create
16. --rpc-url
17. http://localhost:8545
18. --private-key
19. <
20. test_account_private_ke
21. y
22.
23. src/MyNFT.sol:MyNFT
24. Flags
25.
    - --rpc-url
26.
    - : flag to specify the URL of the Ethereum node. In this case, it's the URL of the local Anvil node.
27.
    - --private-key
28.
    - : flag to specify the private key of the account that will deploy the contract.
29.
    - Anvil generated test account's private key is used in this case.
30. Deploy contract to the Sei devnet (EVM endpoint):
31. forge
32. create
33. --rpc-url
34. https://evm-rpc.arctic-1.seinetwork.io/
35. --private-key
36. <
37. your_private_ke

38. y
39.
40. src/MyNFT.sol:MyNFT
41. Flags
42.
    - --rpc-url
43.
    - : flag to specify the URL of the Ethereum node. In this case, it's the Sei EVM devnet URL.
44.
    - --private-key
45.
    - : flag to specify the private key of the account that will deploy the contract.
46.
    - The private key of your account on the Sei devnet is used in this case.

Make sure to have SEI in your account to cover the gas fees for contract deployment.

Interacting With Your NFT

- You can interact with your deployed NFT contract using Foundry'scast
- tool or through other tools like Ethers.js in a script or web application.

Create Pointer Contract

To enable seamless use of this NFT contract in CosmWasm environments, you can create a pointer contract. This process results in an CW721 token that can be imported and used in Sei wallets and applications.

seid

tx

wasm

instantiate

8

'{"erc721_address": "ERC721_TOKEN_ADDRESS"}'

--from=ACCOUNT

--label=LABEL

--chain-id=arctic-1

--broadcast-mode=block

--gas=250000

--fees=25000usei

--node=https://rpc-arctic-1.sei-apis.com/

--no-admin Parameters

- ERC721_TOKEN_ADDRESS
- : The contract address of the ERC721 token you want to create an CW721 pointer for.

Flags

- --from
- : The Sei address from which the deployment transaction is sent. This address must have enough balance to cover transaction fees.
- --label
- : A name for the contract instance used to identify the contract.
- --chain-id
- : Identifies the specific chain of the Sei network you're interacting with.arctic-1
- refers to the Sei devnet.
- --broadcast-mode

- : Determines how the transaction is broadcasted. Setting this toblock
- means the transaction will wait to be included in a block before returning a response.
- --gas
- : Specifies the maximum amount of gas that can be consumed by the transaction.
- --fees
- : Indicates the transaction fee.
- --node
- : Points to the specific Sei node RPC URL you're connecting to for transaction submission.
- --no-admin
- : Specifies that the contract should not have an admin. This flag is used to indicate that the contract, once deployed, cannot be upgraded or migrated.

Executing this command creates an CW721 NFT contract and outputs the contract address. This NFT contract is linked to the ERC721 NFT contract, meaning any activities involving CW721 NFTs will also reflect on the state of the ERC721 NFTs and vice versa.

Learn more about EVM interoperability and pointer contractshere .

Conclusion

Congratulations! You've successfully created and deployed an NFT contract on Sei.

If you encounter a bug while using the devnet, please submit it via the formhere . Last updated onMay 27, 2024Token Factory Pointer Contracts