

[Network](#) ▾[Nodes](#) ▾[APIs](#) ▾[Tools](#) ▾[ACPs](#) ▾[Step 1: Setting up Core](#) ▾[Avalanche l1s](#) > [Add utility](#)

Deploy a Smart Contract

Deploy a smart contract on your Avalanche L1.



Solidity Compiler EVM Version

Avalanche C-Chain and Subnet-EVM currently support the **Cancun** EVM version and do not yet support newer hardforks like **Pectra**. Since Solidity v0.8.30 changed its default target to Pectra, you must explicitly configure your compiler to target `cancun`.

In Remix: Open the Solidity Compiler panel → expand Advanced Configurations → set EVM Version to `cancun`.

For Hardhat or Foundry configurations, see the [contract verification guide](#).

This tutorial assumes that:

- an Avalanche L1 and EVM blockchain has been created
- Your Node is currently validating your target Avalanche L1
- Your wallet has a balance of the Avalanche L1 Native Token(Specified under *alloc* in your Genesis File).

Step 1: Setting up Core

EVM Avalanche L1 Settings: (EVM Core Tutorial)

- `Network Name`: Custom Subnet-EVM

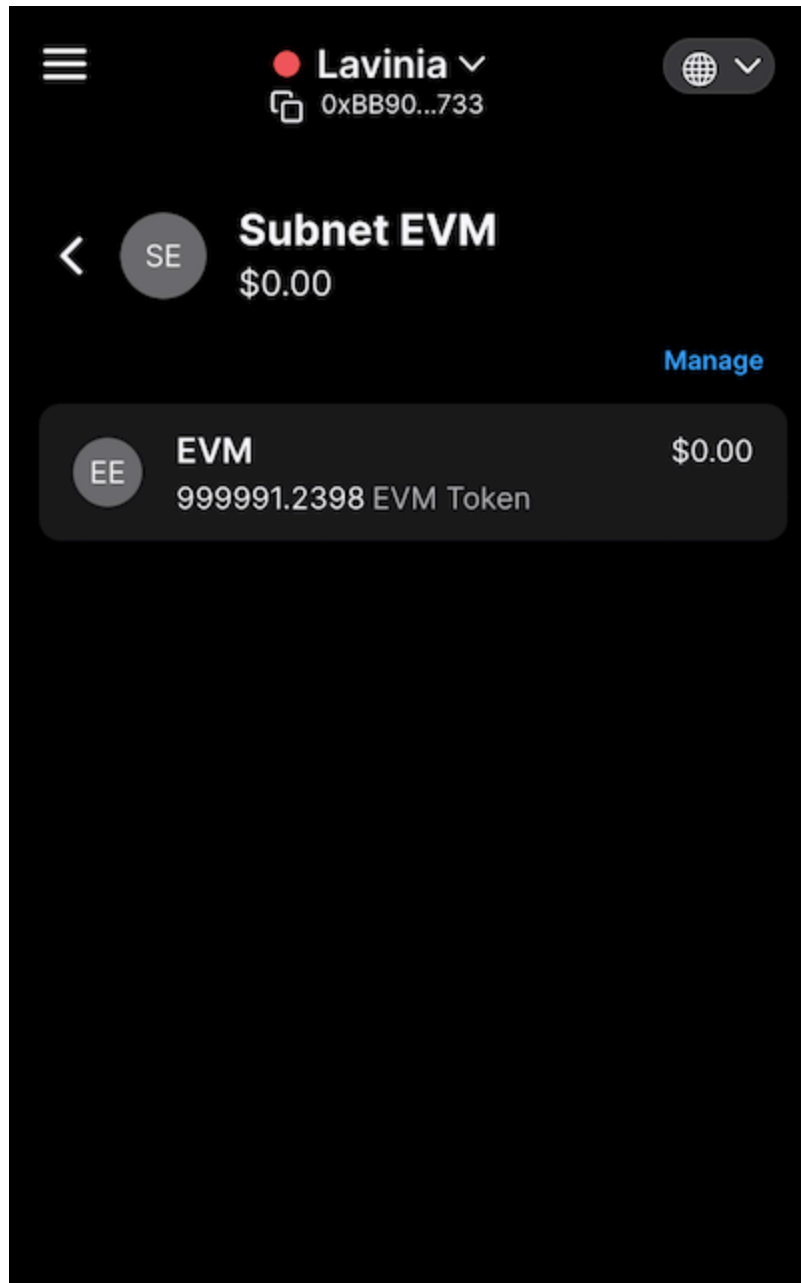


[Network](#) ▾[Nodes](#) ▾[APIs](#) ▾[Tools](#) ▾[ACPs](#) ▾

• **Symbol** : Subnet-EVM Token Symbol

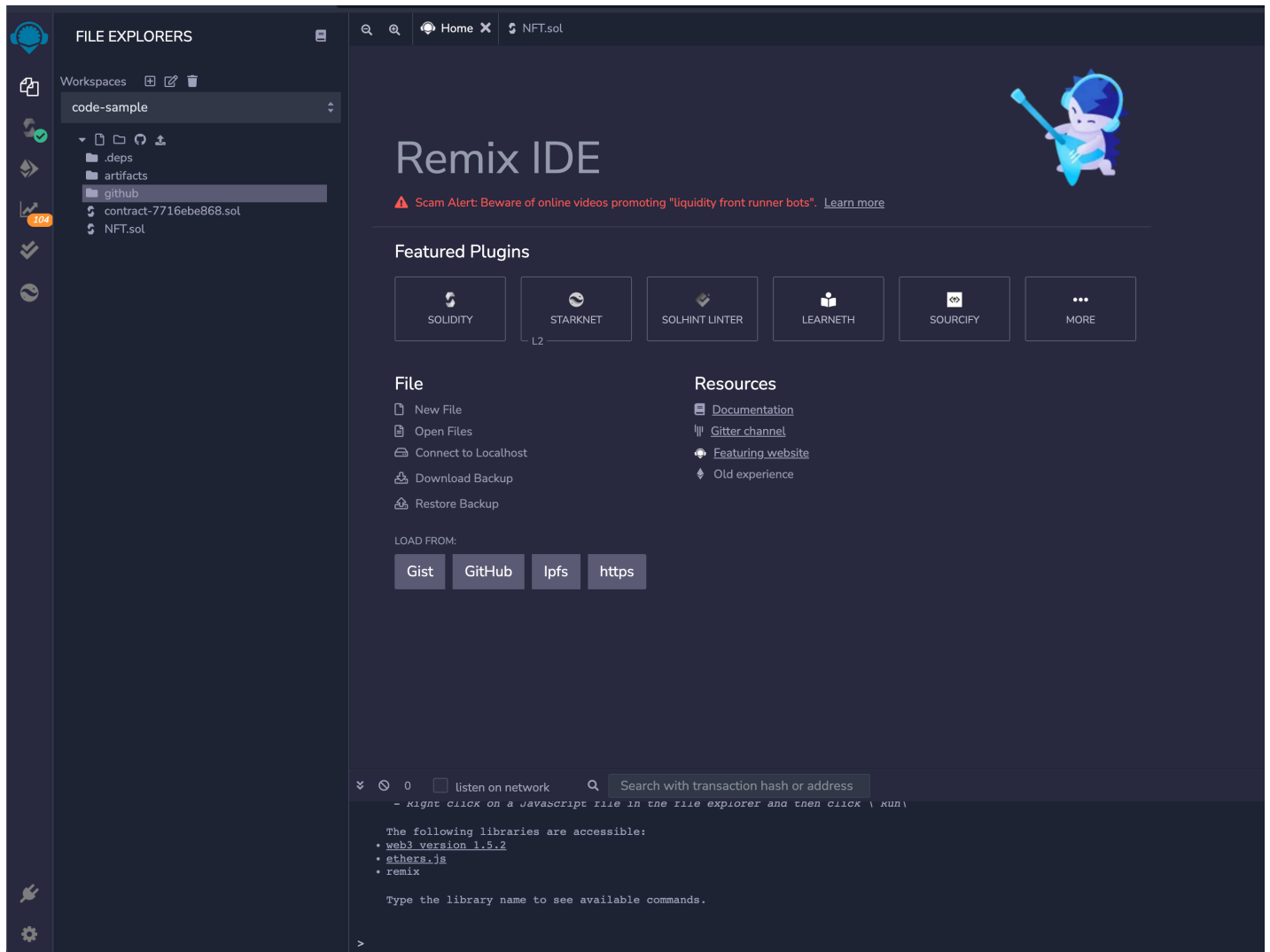
• **Explorer** : N/A

You should see a balance of your Avalanche L1's Native Token in Core.



[Network](#) ▾[Nodes](#) ▾[APIs](#) ▾[Tools](#) ▾[ACPs](#) ▾

Open Remix → Select Solidity.



Create the smart contracts that we want to compile and deploy using Remix file explorer

Using GitHub

In Remix Home *Click* the GitHub button.



Network ▾

Nodes ▾

APIs ▾

Tools ▾

ACPs ▾

Gist

GitHub

Ipfs

https

Paste the link to the Smart Contract into the popup and *Click* import.

Import from Github



Enter the github URL you would like to load.

e.g

`https://github.com/0xcert/ethereum-erc721/src/contracts/tokens/nf-token-metadata.sol`

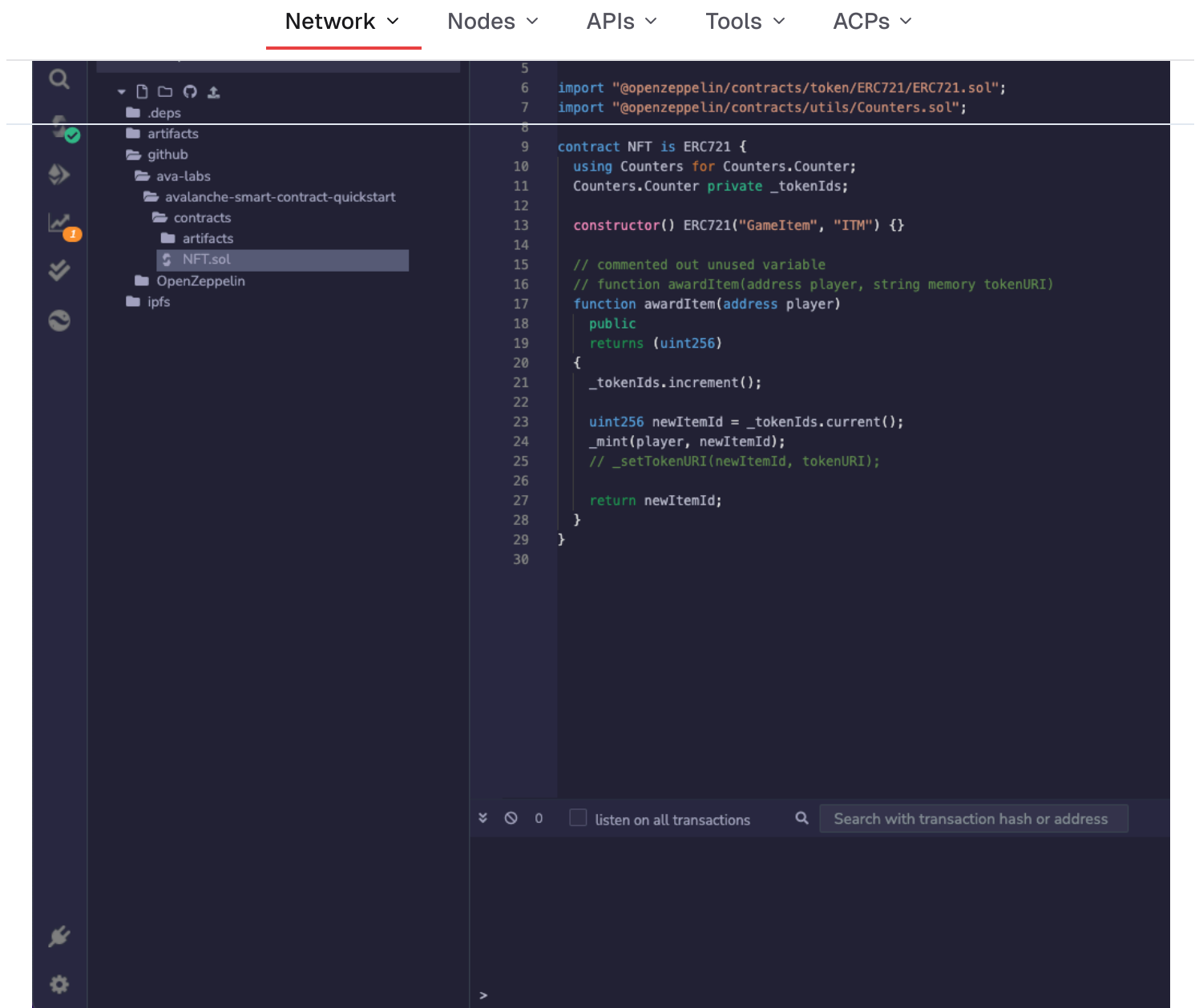
`https://github.com/OpenZeppelin/openzeppelin-solidity/blob/67bca857eedf99bf44a4b6a0fc5b5ed553135316/contracts/access/Roles.sol`

`https://github.com/avalanche-smart-contract-quickstart/blob/main/contracts/NFT.sol`

Import

For this example, we will deploy an ERC721 contract from the Avalanche Smart Contract Quickstart Repository.





Navigate to Deploy Tab → Open the "ENVIRONMENT" drop-down and select Injected Web3 (make sure Core is loaded).



6/11

Network ▾ Nodes ▾ APIs ▾ Tools ▾ ACPs ▾

Include nightly builds

LANGUAGE

Solidity

EVM VERSION

default

COMPILER CONFIGURATION

Auto compile

Enable optimization 200

Hide warnings

Compile NFT.sol

Compile and Run script

CONTRACT

NFT (NFT.sol)

Publish on Ipfs

Publish on Swarm

Compilation Details

ABI Bytecode

```
5
6 import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
7 import "@openzeppelin/contracts/utils/Counters.sol";
8
9 contract NFT is ERC721 {
10     using Counters for Counters.Counter;
11     Counters.Counter private _tokenIds;
12
13     constructor() ERC721("GameItem", "ITM") {}
14
15     // commented out unused variable
16     // function awardItem(address player, string memory tokenURI)
17     function awardItem(address player)
18         public
19         returns (uint256)
20     {
21         _tokenIds.increment();
22
23         uint256 newItemId = _tokenIds.current();
24         _mint(player, newItemId);
25         // _setTokenURI(newItemId, tokenURI);
26
27         return newItemId;
28     }
29 }
30
```

0

☐ listen on all transactions

Search with transaction hash or address

Now, the smart contract is compiled, Core is injected, and we are ready to deploy our ERC721. Click "Deploy."

<https://build.avax.network/docs/avalanche-11s/add-utility/deploy-smart-contract>

7/11

Network ▾ Nodes ▾ APIs ▾ Tools ▾ ACPs ▾

Custom (10920) network

ACCOUNT

0x845...B1aB3 (999992.3913716)

GAS LIMIT

3000000

VALUE

0 Wei

CONTRACT

NFT - github/ava-labs/avalanche-smart-contract

Deploy

☒ Publish to IPFS

OR

At Address

Load contract from Address

Transactions recorded 10

Deployed Contracts

Currently you have no contract instances to interact with.

```
5
6 import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
7 import "@openzeppelin/contracts/utils/Counters.sol";
8
9 contract NFT is ERC721 {
10     using Counters for Counters.Counter;
11     Counters.Counter private _tokenIds;
12
13     constructor() ERC721("GameItem", "ITM") {}
14
15     // commented out unused variable
16     // function awardItem(address player, string memory tokenURI)
17     function awardItem(address player)
18         public
19         returns (uint256)
20     {
21         _tokenIds.increment();
22
23         uint256 newItemId = _tokenIds.current();
24         _mint(player, newItemId);
25         // _setTokenURI(newItemId, tokenURI);
26
27         return newItemId;
28     }
29 }
30
```

☒ listen on all transactions

Search with transaction hash or address

Confirm the transaction on the Core pop up.



Network ▾

Nodes ▾

APIs ▾

Tools ▾

ACPs ▾

Transaction Details 

Account

Lavinia

Network

Avalanche (C-Chain)

Transaction Summary

Transaction Type

Contract deployment

Network Fee 

Normal
26

Fast
28

Instant
29

Custom
53

Fee Amount

0 EVM
\$0

Reject

Approve

Our contract is successfully deployed!



Network ▾

Nodes ▾

APIs ▾

Tools ▾

ACPs ▾

The screenshot displays the Avalanche Builder Hub interface. On the left, the 'Custom (10920) network' is selected. The 'ACCOUNT' section shows an address starting with '0x845...B1aB3' and a balance of '999991.3297990'. The 'GAS LIMIT' is set to '3000000' and the 'VALUE' is '0' in 'Wei'. The 'CONTRACT' section shows the path 'NFT - github/ava-labs/avalanche-smart-contract'. A 'Deploy' button is visible, along with a checkbox for 'Publish to IPFS'. Below this, there are options to 'At Address' or 'Load contract from Address'. The 'Transactions recorded' section shows '12' transactions. The 'Deployed Contracts' section shows a list of contracts, including 'NFT AT 0X15E...E558F (BLOCKCHAIN)'. On the right, the smart contract code is displayed, showing the 'NFT' contract. The code includes imports for '@openzeppelin/contracts' and '@openzeppelin/contracts/token/ERC721/ERC721.sol', and defines the 'NFT' contract with a constructor and an 'awardItem' function. The bottom of the interface shows a search bar and a status message 'creation of NFT pending...'.

Now, we can expand it by selecting it from the "Deployed Contracts" tab and test it out.



Network ▾

Nodes ▾

APIs ▾

Tools ▾

ACPs ▾

Transactions recorded 12

Deployed Contracts

▼ NFT AT 0X15E...E558F (BLOCKCHAIN)

- approve address to, uint256 tokenId
- awardItem address player
- safeTransferFr... address from, address to, uint256 tokenId
- safeTransferFr... address from, address to, uint256 tokenId, byte
- setApprovalFo... address operator, bool approved
- transferFrom address from, address to, uint256 tokenId
- balanceOf address owner
- getApproved uint256 tokenId
- isApprovedFor... address owner, address operator
- name
- ownerOf uint256 tokenId
- supportsInterf... bytes4 interfaced
- symbol
- tokenURI uint256 tokenId

Low level interactions

CALLDATA

Transact

```
6 import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
7 import "@openzeppelin/contracts/utils/Counters.sol";
8
9 contract NFT is ERC721 {
10     using Counters for Counters.Counter;
11     Counters.Counter private _tokenIds;
12
13     constructor() ERC721("GameItem", "ITM") {}
14
15     // commented out unused variable
16     // function awardItem(address player, string memory tokenURI)
17     function awardItem(address player)
18     public
19     returns (uint256)
20     {
21         _tokenIds.increment();
22
23         uint256 newItemId = _tokenIds.current();
24         _mint(player, newItemId);
25         // _setTokenURI(newItemId, tokenURI);
26
27         return newItemId;
28     }
29 }
30
```

0 ☐ listen on all transactions Search with transaction hash or address

creation of NFT pending...

Is this guide helpful?

👍 Yes

👎 No

📄 Copy Markdown

