

GUIA PASO A PASO

Paso 1: Configuración inicial en OpenZeppelin Wizard

- Se configura el token con nombre **Coin** y símbolo **MTK**.
- Se establece un suministro inicial de **1,000,000** tokens.
- Se activan las opciones **Mintable**, **Burnable** y **Permit**.
- Se selecciona **Ownable** para control de permisos.
- El código generado incluye la definición del contrato y la función mint para crear tokens solo por el propietario.

The screenshot shows the OpenZeppelin Wizard interface for configuring an ERC20 token. The left sidebar contains settings for Name (Coin), Symbol (MTK), and Pre-mint (1000000). Under Features, Mintable, Burnable, and Permit are checked. Under Access Control, Ownable is selected. The main area displays the generated Solidity code, which includes imports for ERC20, ERC20Burnable, ERC20Permit, and Ownable, and defines the Coin contract with a constructor and a mint function. The code is as follows:

```
// SPDX-License-Identifier: MIT
// Compatible with OpenZeppelin Contracts ^5.0.0
pragma solidity ^0.8.27;

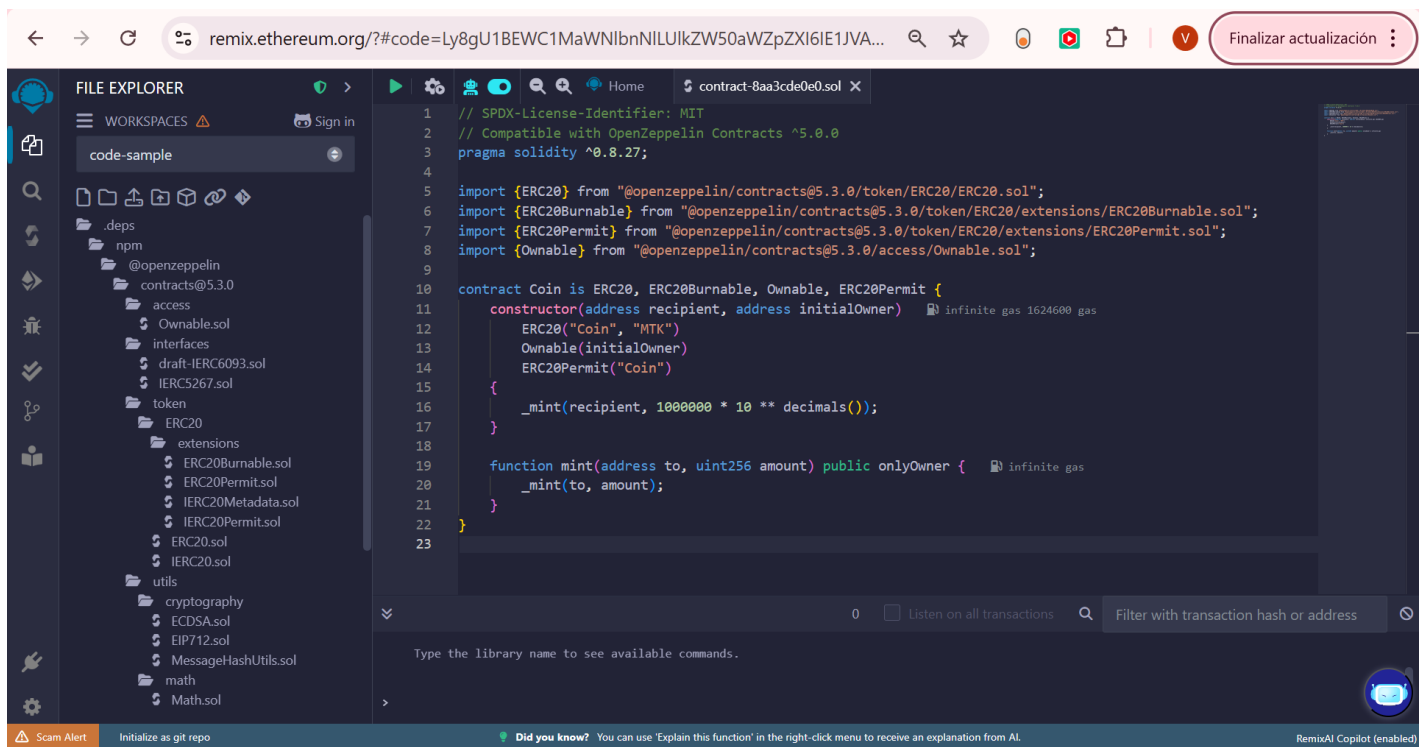
import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import {ERC20Burnable} from "@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol";
import {ERC20Permit} from "@openzeppelin/contracts/token/ERC20/extensions/ERC20Permit.sol";
import {Ownable} from "@openzeppelin/contracts/access/Ownable.sol";

contract Coin is ERC20, ERC20Burnable, Ownable, ERC20Permit {
    constructor(address recipient, address initialOwner)
        ERC20("Coin", "MTK")
        Ownable(initialOwner)
        ERC20Permit("Coin")
    {
        _mint(recipient, 1000000 * 10 ** decimals());
    }

    function mint(address to, uint256 amount) public onlyOwner {
        _mint(to, amount);
    }
}
```

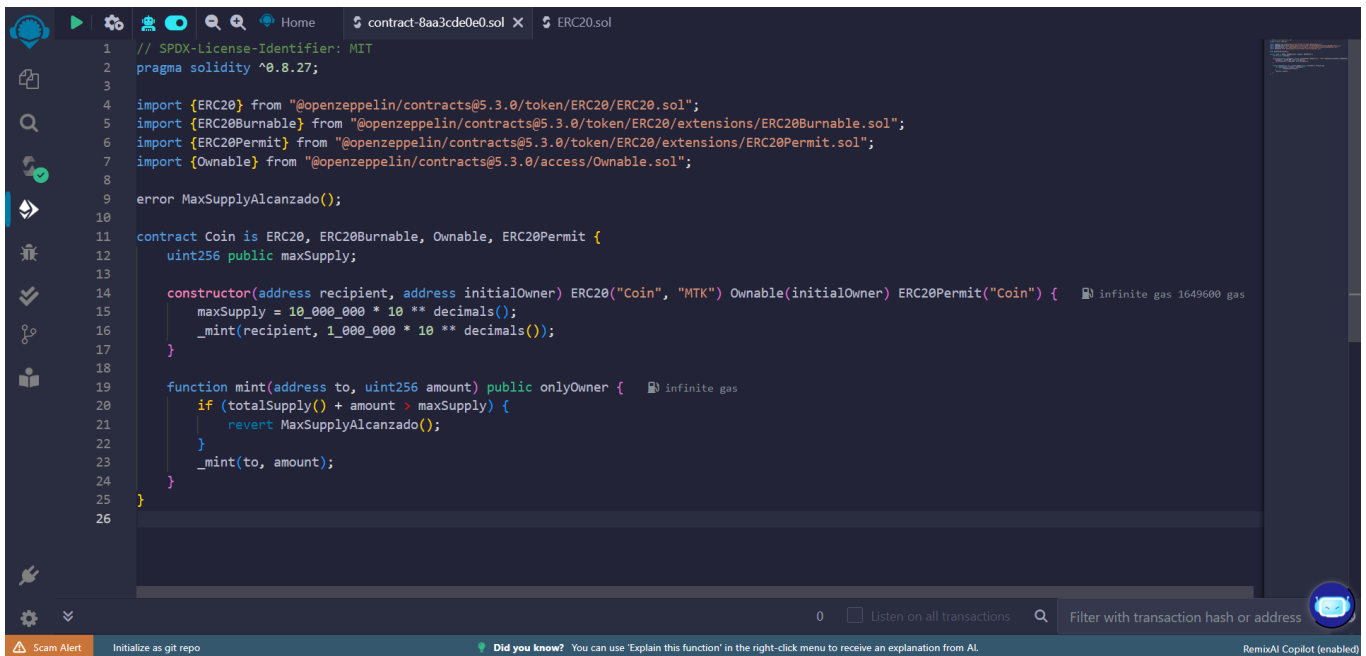
Paso 2: Abrir el código generado en Remix IDE

- Se cargó el **código** generado en **Remix**, un entorno en línea para desarrollar y probar **contratos Solidity**.
- Se observan las **importaciones** de **OpenZeppelin** necesarias para **ERC20**, **ERC20Burnable**, **ERC20Permit** y **Ownable**.
- Se muestra el contrato **Coin** con el **constructor** que:
 - Inicializa el token con **nombre "Coin"** y **símbolo "MTK"**.
 - Define al **propietario** con **Ownable**.
 - **Mintea** un **suministro inicial** de **1,000,000 tokens** a la dirección **recipient**.
- Está definida la función **mint** con restricción **onlyOwner** para permitir creación **controlada** de tokens.



Paso 3: Edición del código en Remix para añadir límite de suministro

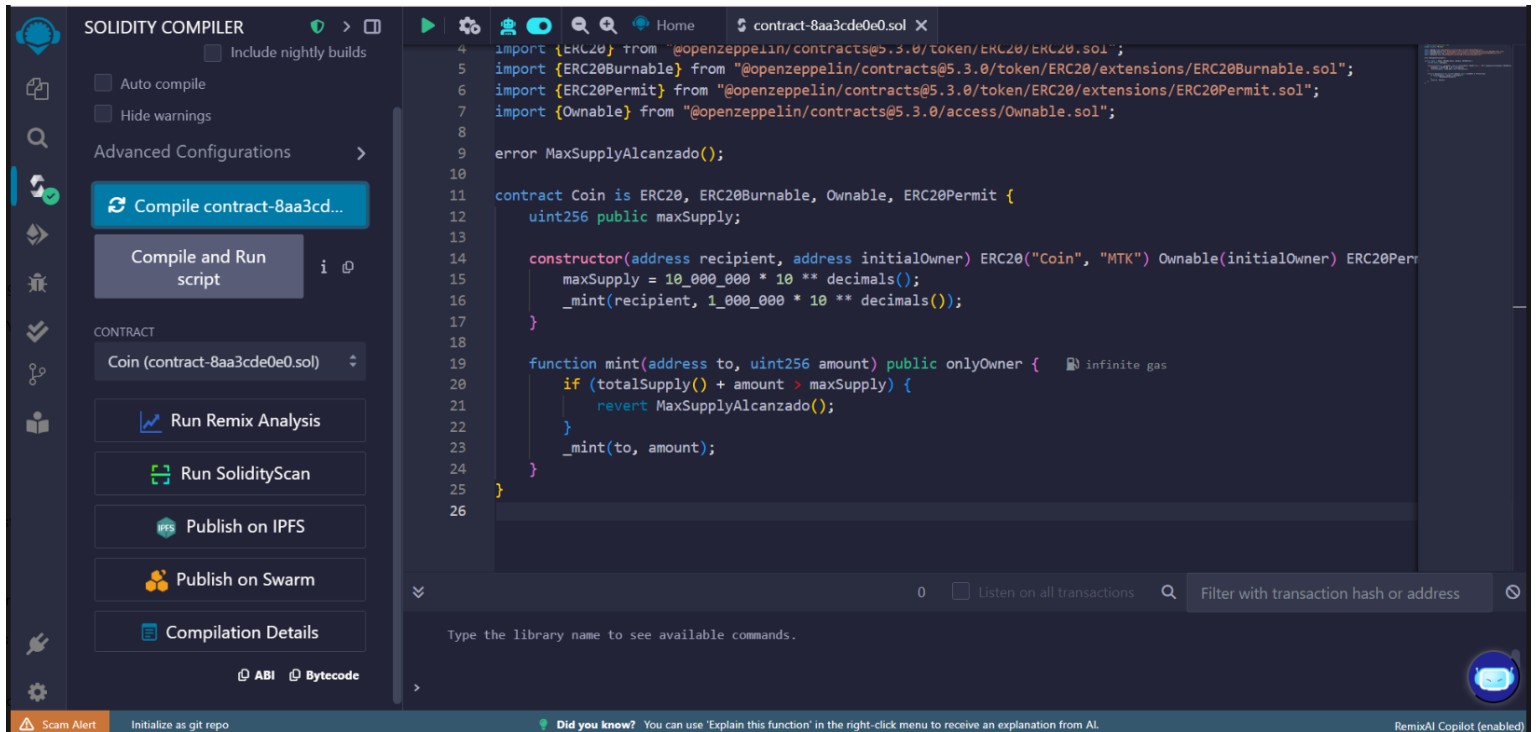
- Se abre el contrato en Remix y se observa la importación de los contratos de **OpenZeppelin** necesarios.
- Se declara el error personalizado **MaxSupplyAlcanzado** para controlar intentos de exceder el límite.
- Se agrega la variable pública **maxSupply** que representa el límite máximo de tokens.
- En el **constructor**, se inicializa **maxSupply** con 10 millones de tokens (considerando decimales).
- Se mintean 1 millón de tokens iniciales a la dirección **recipient**.
- Se modifica la función **mint** para:
 - Verificar que la suma de **totalSupply()** más la cantidad a mintear no supere **maxSupply**.
 - Si se excede, la función revierte con el error **MaxSupplyAlcanzado**.
 - Si no, se ejecuta la acuñación con **_mint**.
- Esto asegura que el contrato cumple con la condición de límite máximo de emisión establecida en el proyecto.



```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.27;
3
4 import {ERC20} from "@openzeppelin/contracts@5.3.0/token/ERC20/ERC20.sol";
5 import {ERC20Burnable} from "@openzeppelin/contracts@5.3.0/token/ERC20/extensions/ERC20Burnable.sol";
6 import {ERC20Permit} from "@openzeppelin/contracts@5.3.0/token/ERC20/extensions/ERC20Permit.sol";
7 import {Ownable} from "@openzeppelin/contracts@5.3.0/access/Ownable.sol";
8
9 error MaxSupplyAlcanzado();
10
11 contract Coin is ERC20, ERC20Burnable, Ownable, ERC20Permit {
12     uint256 public maxSupply;
13
14     constructor(address recipient, address initialOwner) ERC20("Coin", "MTK") Ownable(initialOwner) ERC20Permit("Coin") {
15         maxSupply = 10_000_000 * 10 ** decimals();
16         _mint(recipient, 1_000_000 * 10 ** decimals());
17     }
18
19     function mint(address to, uint256 amount) public onlyOwner {
20         if (totalSupply() + amount > maxSupply) {
21             revert MaxSupplyAlcanzado();
22         }
23         _mint(to, amount);
24     }
25 }
26
```

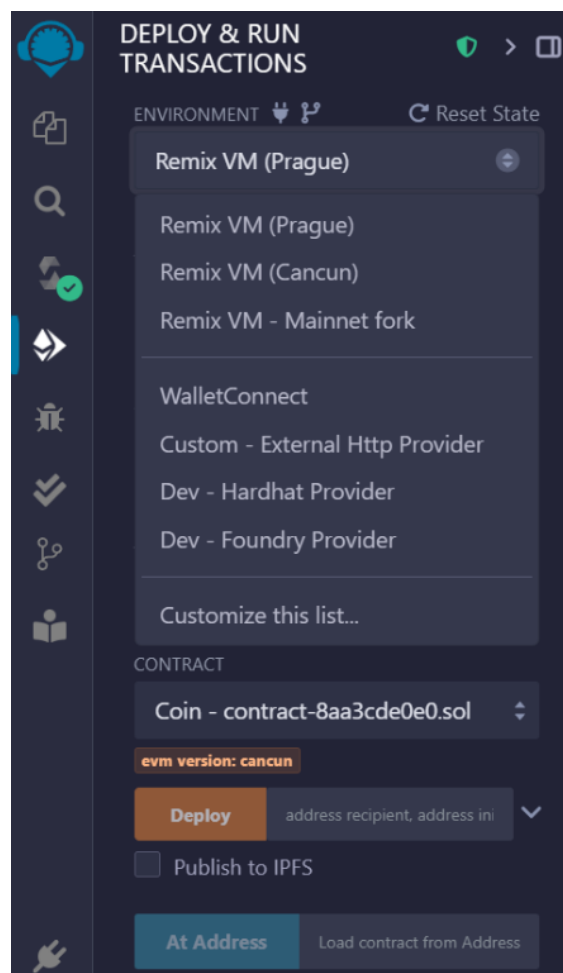
Paso 4: Compilar el contrato en Remix

- Se utiliza el compilador **Solidity** en Remix para compilar el contrato Coin.
- Se selecciona la versión del compilador **0.8.27** compatible con el código.
- Se pulsa el botón **Compile contract-8aa3cde0e0.sol** para generar el bytecode.
- La compilación sin errores confirma que el contrato está listo para desplegarse.
- Remix muestra los detalles de la compilación, incluyendo el bytecode y ABI.



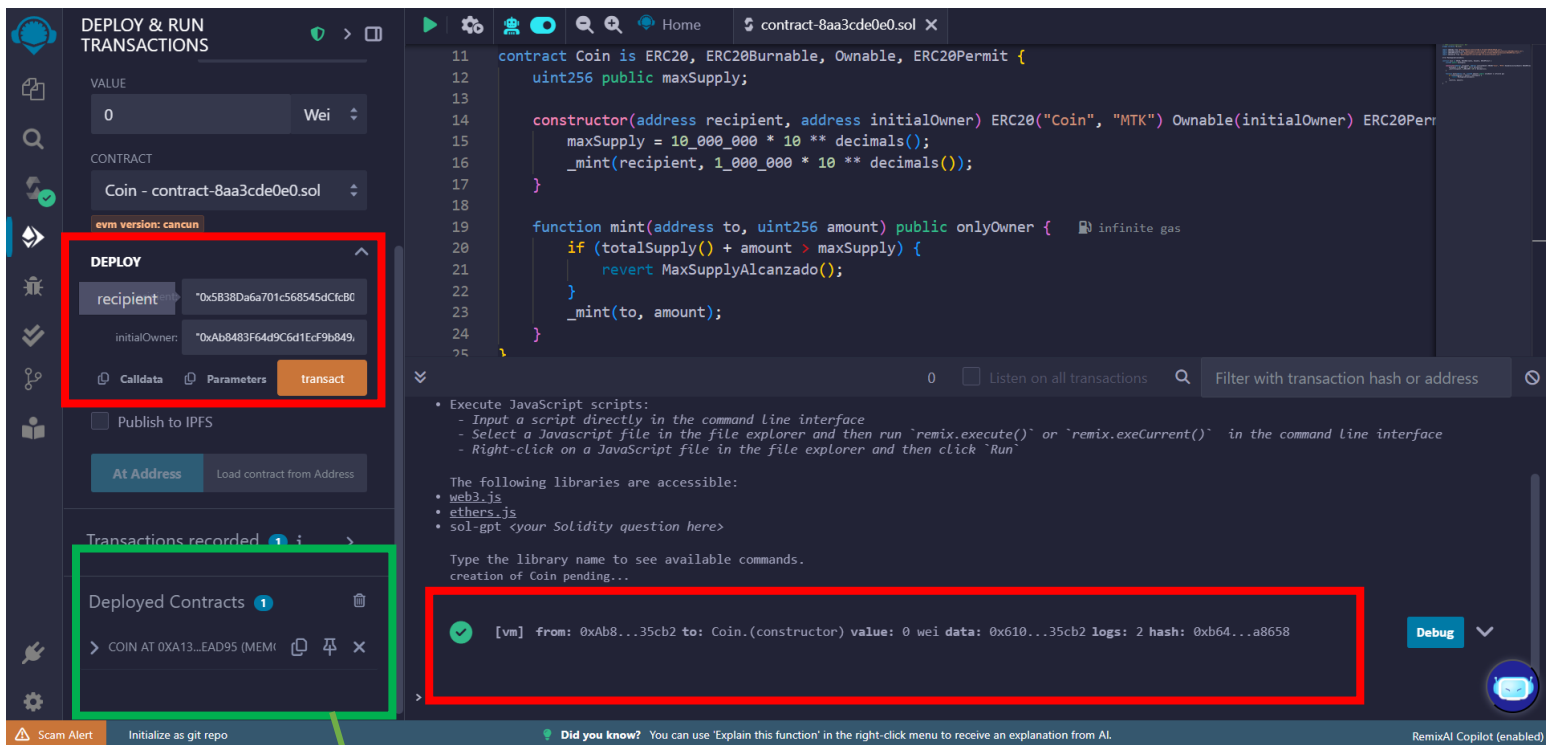
Paso 5: Selección del entorno y preparación para desplegar

- En el panel **Deploy & Run Transactions** de Remix, se selecciona el entorno (**Environment**).
- Se escoge el entorno **Remix VM (Prague)** para simular la red Ethereum localmente.
- En **Contract**, se selecciona el contrato compilado **Coin - contract-8aa3cde0e0.sol**.
- El botón **Deploy** está listo para desplegar el contrato en la red seleccionada.
- Debajo del botón **Deploy** aparecen los parámetros del constructor donde se deben ingresar las direcciones recipient y initialOwner.
- Opcionalmente, se puede activar la casilla **Publish to IPFS** para publicar el código en la red IPFS.



Paso 6: Desplegar el contrato

- En el panel **Deploy & Run Transactions**, se ingresan los parámetros del constructor:
 - **recipient:** dirección que recibirá el suministro inicial.
 - **initialOwner:** dirección que será asignada como dueño del contrato.
- Se verifica que el contrato seleccionado sea **Coin - contract-8aa3cde0e0.sol**.
- Se presiona el botón **Deploy** para desplegar el contrato en la red simulada (**Remix VM**).
- En la consola se muestra el resultado del despliegue con el hash de la transacción y confirmación exitosa.
- Aparece el contrato desplegado en la sección **Deployed Contracts** listo para interactuar.



Contrato desplegado exitosamente