



Desarrollo de Contratos Inteligentes Seguros en Blockchain con Solidity

—— Manual para desarrolladores ——

Contenido

Introducción.....	5
¿Qué son los Contratos Inteligentes?	6
Importancia de la Seguridad en Contratos Inteligentes	6
Objetivo del manual	6
Fundamentos de Contratos Inteligentes.....	6
Blockchain.....	6
Lenguaje de Programación	7
Solidity	7
HERRAMIENTAS Y FRAMEWORKS DE DESARROLLO.....	8
Lenguajes de Desarrollo	8
Solidity	8
Vyper.....	9
Compiladores.....	9
Herramientas y Librerías	10
Node.js.....	10
Frameworks	14
Truffle	14
Drizzle	14
Embark	16
Brownie	17
Hardhat	17
OpenZeppelin	17
Entorno de Desarrollo Integrado (IDE)	18
Visual Studio Code.....	18
Remix IDE.....	19
Configuración del Entorno de Desarrollo.....	21
Instalación de Herramientas	21
Instalación de Node.js y npm	21
Instalación de Hardhat	21
Estructura del Proyecto	22
Desarrollo de un Contrato Inteligente	23

1. Crear un Nuevo Contrato	23
2. Compilación del Contrato.....	24
3. Despliegue del Contrato	24
Interactuar con el Contrato	26
Crear un Script de Interacción	27
Pruebas del Contrato	28
Crear Pruebas con Hardhat	28
Ejecutar las Pruebas.....	29

Introducción

Los contratos inteligentes han emergido como una herramienta poderosa y transformadora en la automatización de acuerdos y transacciones. Construidos sobre la tecnología blockchain, estos contratos permiten ejecutar automáticamente las condiciones acordadas entre las partes, eliminando la necesidad de intermediarios y aumentando la eficiencia y seguridad en diversas aplicaciones, desde finanzas descentralizadas (DeFi) hasta la gestión de cadenas de suministro. Sin embargo, a pesar de sus ventajas, los contratos inteligentes no están exentos de riesgos y vulnerabilidades. Un error en su codificación o un malentendido en su lógica puede llevar a consecuencias graves, incluyendo la pérdida de activos valiosos y la explotación por actores maliciosos.

Este manual se enfoca en la validación y verificación de contratos inteligentes, áreas cruciales en el desarrollo de estos contratos para garantizar su seguridad y fiabilidad. La validación se refiere a la comprobación de que el contrato cumple con los requisitos especificados y que su lógica es coherente y correcta. Por otro lado, la verificación se centra en asegurar que el contrato funcione como se espera en diferentes escenarios, incluyendo condiciones límite y situaciones excepcionales.

A lo largo de este manual, se abordarán diversas técnicas y herramientas que permiten a los desarrolladores no solo escribir código seguro, sino también validarlo y verificarlo rigurosamente antes de su implementación en una red blockchain. Se explorarán métodos como el análisis estático, pruebas unitarias, verificación formal, y análisis dinámico, cada uno de los cuales juega un papel fundamental en la creación de contratos inteligentes robustos.

Además, se discutirán las mejores prácticas de seguridad que deben adoptarse durante el desarrollo para mitigar riesgos comunes como la reentrancia, los desbordamientos (overflow), y el acceso no autorizado. También se incluirán estudios de caso que ilustran tanto contratos exitosos como aquellos que han fallado, ofreciendo lecciones valiosas sobre lo que se debe y no se debe hacer.

Este manual está diseñado para ser una guía práctica y accesible, dirigida a desarrolladores y profesionales del campo, así como a cualquier persona interesada en el desarrollo seguro de contratos inteligentes. Al final de este recorrido, el lector estará equipado con el conocimiento necesario para validar y verificar contratos inteligentes de manera efectiva, asegurando que cumplan con los más altos estándares de seguridad y funcionalidad.

¿Qué son los Contratos Inteligentes?

Los contratos inteligentes son programas informáticos que se ejecutan automáticamente cuando se cumplen ciertas condiciones predeterminadas. Funcionan en la blockchain, proporcionando transparencia, inmutabilidad y seguridad a las transacciones y acuerdos digitales.

El contrato inteligente es un conjunto de instrucciones desarrollado en Solidity (podría ser otro lenguaje de programación, pero en este libro solo usaremos Solidity). El lenguaje de programación Solidity se basa en IFTTT (que es la lógica IF-THIS-THEN-THAT: ejecuta el código si se cumple alguna condición). Dado que el contrato inteligente se ejecuta en EVM, no puede acceder a la red, al sistema de archivos ni a otros procesos que se ejecutan en EVM. El contrato inteligente puede acceder a datos externos a través de un Oracle (se trata de un elemento ofrecido por terceros que brinda información externa y que permite que los contratos inteligentes tomen decisiones si es necesario). Generalmente, el contrato inteligente se puede implementar con base en dos tipos de sistema: 1. Máquina virtual (VM): Ethereum. 2. Docker: Fabric. Todo el contenido de este libro se basa en los contratos inteligentes sobre Ethereum.

Importancia de la Seguridad en Contratos Inteligentes

La naturaleza inmutable de la blockchain implica que cualquier error o vulnerabilidad en un contrato inteligente puede tener consecuencias graves y permanentes. Por lo tanto, es crucial garantizar que los contratos inteligentes sean rigurosamente validados y verificados antes de su implementación.

Objetivo del manual

Este manual está diseñado para guiarte en el desarrollo de contratos inteligentes con Solidity, desde los conceptos básicos hasta ejemplos prácticos de implementación.

Fundamentos de Contratos Inteligentes

Blockchain

La blockchain es una tecnología de registro distribuido que permite transacciones seguras y transparentes. Cada bloque en una cadena contiene un conjunto de transacciones verificadas y es inmutable una vez añadido a la cadena, lo que garantiza la integridad de los datos.

Lenguaje de Programación

Solidity

El lenguaje más popular para desarrollar contratos inteligentes en Ethereum. Solidity es el lenguaje más popular para el desarrollo de contratos inteligentes en la blockchain de Ethereum. Es un lenguaje de programación de alto nivel. Tiene una sintaxis similar a JavaScript. Admite tipos estáticos, integración, bibliotecas y tipo compuesto definido por el usuario. Se puede compilar en el ensamblaje EVM y, por lo tanto, se puede ejecutar en todos los nodos de Ethereum. Hay otros lenguajes de programación de contratos inteligentes: Vyper, Yul, Rust y JavaScript. Sin lugar a duda, Solidity es el lenguaje de programación más popular para contratos inteligentes. EVM es un entorno limitado de tiempo de ejecución. Por lo tanto, todos los contratos inteligentes que descansan en Ethereum están segregados del entorno circundante. Como resultado, el contrato inteligente en EVM no puede acceder a la red, el sistema de archivos u otros procesos en Ethereum.

Características:

1. Tipado estático.
2. Soporte para herencia y bibliotecas.
3. Gestión de errores y excepciones.
4. Permite definir estructuras de datos complejas.



Ilustración 1 Logo de Solidity

HERRAMIENTAS Y FRAMEWORKS DE DESARROLLO

Antes de comenzar a programar en Solidity, es crucial preparar el entorno de desarrollo y familiarizarse con las herramientas, lenguajes y frameworks que se utilizan en la creación de contratos inteligentes en Ethereum. Este capítulo proporciona una introducción a estos elementos, guiando paso a paso en la instalación y configuración de los programas necesarios. Ethereum ofrece una variedad de herramientas que facilitan el desarrollo, desde clientes hasta entornos de desarrollo integrados (IDEs) y frameworks. La figura 2.1 presenta una clasificación de estas herramientas, lo que te permitirá conocer las opciones disponibles y cómo se relacionan entre sí dentro del ecosistema de Ethereum.

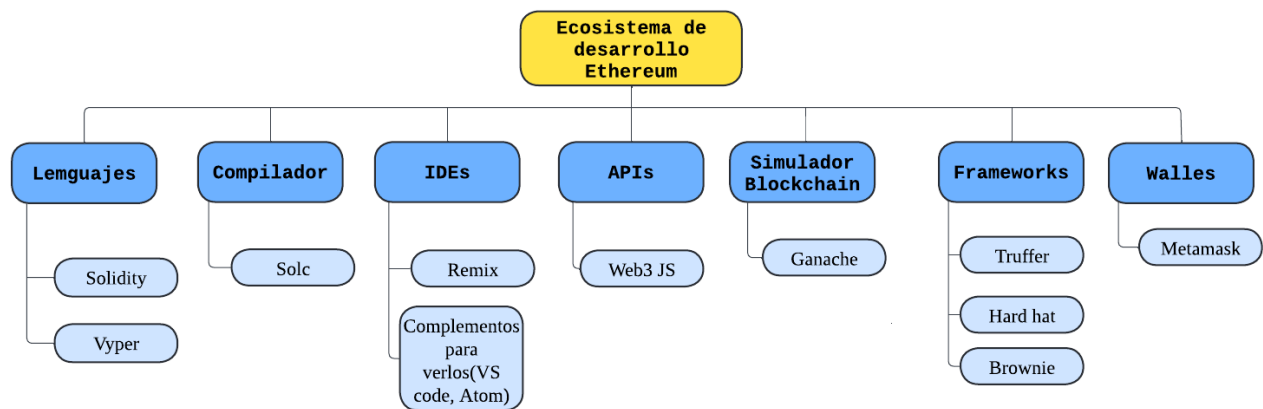


Ilustración 2 Diagrama de Ecosistema de desarrollo Ethereum

La clasificación anterior no abarca todos los frameworks y herramientas disponibles para el desarrollo en Ethereum, pero se enfoca en los más utilizados. Incluye algunas de las herramientas y frameworks que se emplearán en los ejemplos prácticos a lo largo de este manual.

Lenguajes de Desarrollo

Los contratos inteligentes en la blockchain de Ethereum pueden ser programados en varios lenguajes. Sin embargo, dos lenguajes principales destacan en la redacción de contratos.

Solidity

Este lenguaje se ha convertido prácticamente en un estándar para el desarrollo de contratos inteligentes en Ethereum. Será el enfoque principal de este manual, y se explorará en detalle en las secciones siguientes.

Vyper

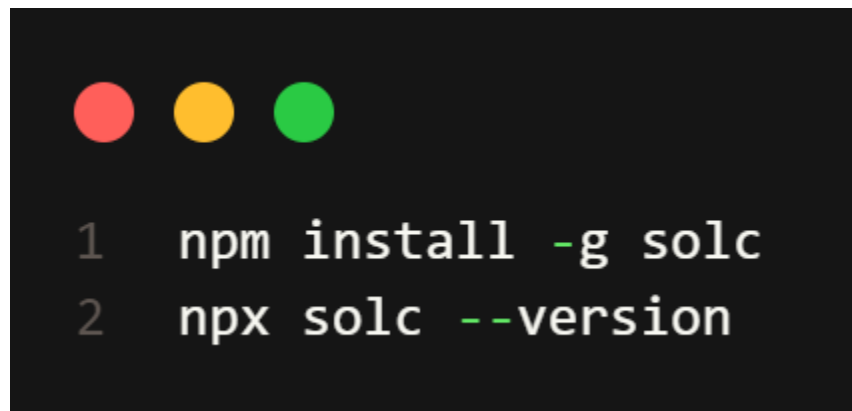
Es un lenguaje experimental inspirado en Python, diseñado para ofrecer mayor seguridad, simplicidad y facilidad de auditoría en el desarrollo de contratos inteligentes.

Dado que el código de Solidity debe ser transformado en bytecode para que pueda ser ejecutado en la blockchain de Ethereum, es necesario utilizar un compilador que realice esta conversión. En la próxima sección, presentaremos el compilador Solidity y explicaremos su funcionamiento.

Compiladores

Los compiladores son herramientas esenciales para transformar el código fuente de un contrato inteligente escrito en un lenguaje de alto nivel en un formato que pueda ser entendido y ejecutado por el entorno de ejecución de Ethereum, la Ethereum Virtual Machine (EVM). El compilador más comúnmente utilizado para Solidity es solc.

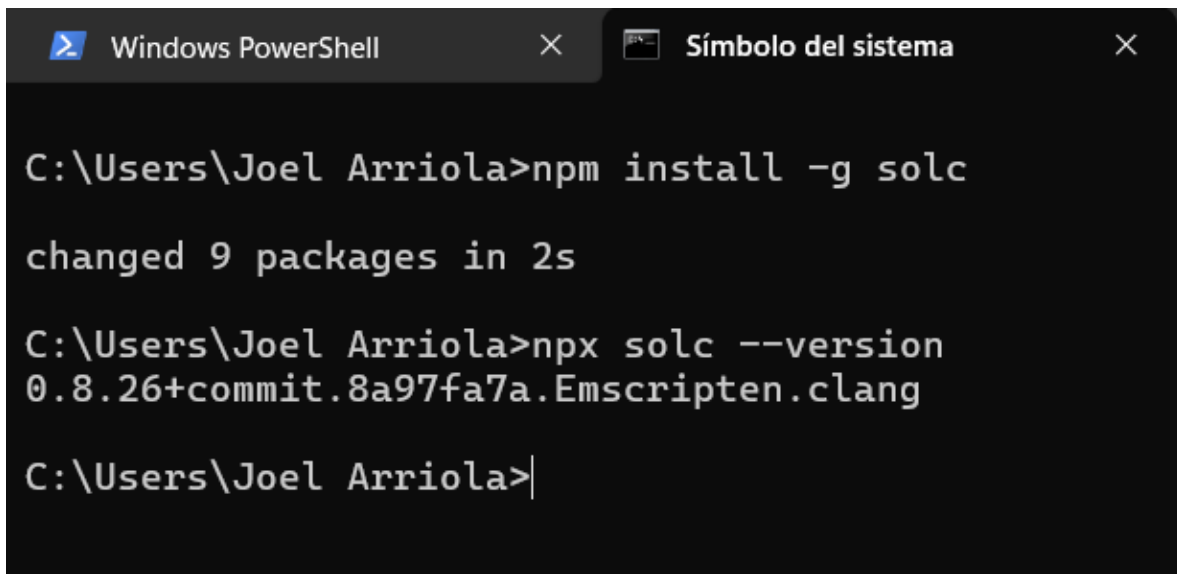
Solc convierte el código Solidity de alto nivel en bytecode, que es el formato que la EVM puede ejecutar directamente en la blockchain. Para poder utilizar solc, primero debes asegurarte de tener instaladas las dependencias necesarias, como Node.js. Una vez instaladas, puedes instalar y verificar el compilador solc utilizando el siguiente comando:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. It displays two numbered commands for installing and verifying the Solc compiler.

```
1  npm install -g solc
2  npx solc --version
```

Ilustración 3 Comandos de Instalación de Solc Compilador

El resultado de salida se puede observar en la Ilustración 4 Resultado de Instalación Completa



```
Windows PowerShell
C:\Users\Joel Arriola>npm install -g solc
changed 9 packages in 2s
C:\Users\Joel Arriola>npx solc --version
0.8.26+commit.8a97fa7a.Emscripten.clang
C:\Users\Joel Arriola>
```

Ilustración 4 Resultado de Instalación Completa

Herramientas y Librerías

Ethereum cuenta con una amplia gama de herramientas y bibliotecas que facilitan el desarrollo de contratos inteligentes y aplicaciones descentralizadas. En esta sección, se analizarán algunas de las más utilizadas y se detallará cómo configurar el entorno de desarrollo.

Node.js

Uno de los primeros requisitos para el desarrollo en Ethereum es la instalación de Node.js. Node.js es un entorno de ejecución de código abierto que permite ejecutar JavaScript en un servidor, eliminando la necesidad de utilizar un navegador web. Este entorno es conocido por su capacidad para manejar una gran cantidad de conexiones simultáneas, lo que lo hace altamente escalable y eficiente.

Un aspecto clave de Node.js es su integración con NPM (Node Package Manager), una herramienta que viene preinstalada con Node.js. NPM permite la gestión de paquetes y la instalación de numerosos componentes desde un repositorio en línea, facilitando el desarrollo de aplicaciones complejas. Para quienes deseen profundizar en Node.js y NPM, se recomienda consultar la documentación oficial en [Nodejs.org](https://nodejs.org) y [Npmjs.com](https://npmjs.com).

Node.js facilita la creación de interfaces web utilizando JavaScript y permite interactuar de manera eficiente con la cadena de bloques. Dado que Node.js es un requisito fundamental para la mayoría de las herramientas y bibliotecas en el desarrollo de aplicaciones

descentralizadas (dApps), su instalación es un primer paso crucial. Para instalar Node.js, sigue estos pasos:

Descarga Node.js: Visita el sitio oficial de Node.js en nodejs.org y selecciona la versión recomendada para la mayoría de los usuarios, o la versión más reciente si necesitas características específicas.

Instalación en Windows/Mac: Ejecuta el instalador que descargaste y sigue las instrucciones en pantalla. El instalador también configurará NPM (Node Package Manager), una herramienta que se instala junto con Node.js y que te permitirá gestionar paquetes y bibliotecas.

Verificación de la instalación: Una vez completada la instalación, abre una terminal o línea de comandos y ejecuta los siguientes comandos para verificar que todo esté en orden:

Visita la página de descargas de Node.js:

Ingresa a <https://nodejs.org/en/download/package-manager>

Selecciona tu sistema operativo:

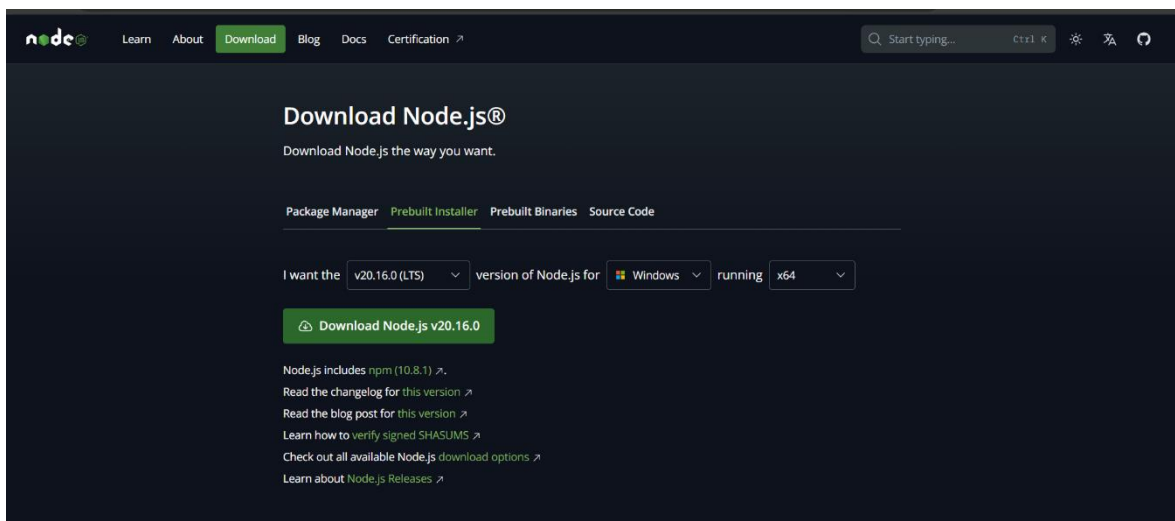


Ilustración 5 Página de Descarga de Node.js

En la página de descargas, verás opciones para diferentes sistemas operativos, como Windows, macOS, y Linux. Haz clic en el instalador correspondiente al sistema operativo que estás utilizando.

Descarga el instalador:

Dependiendo de tu sistema operativo, elige la versión de Node.js que desees instalar (generalmente se recomienda la LTS, que es la versión de soporte a largo plazo).

Instala Node.js:

Una vez descargado, ejecuta el instalador y sigue las instrucciones en pantalla para completar la instalación.

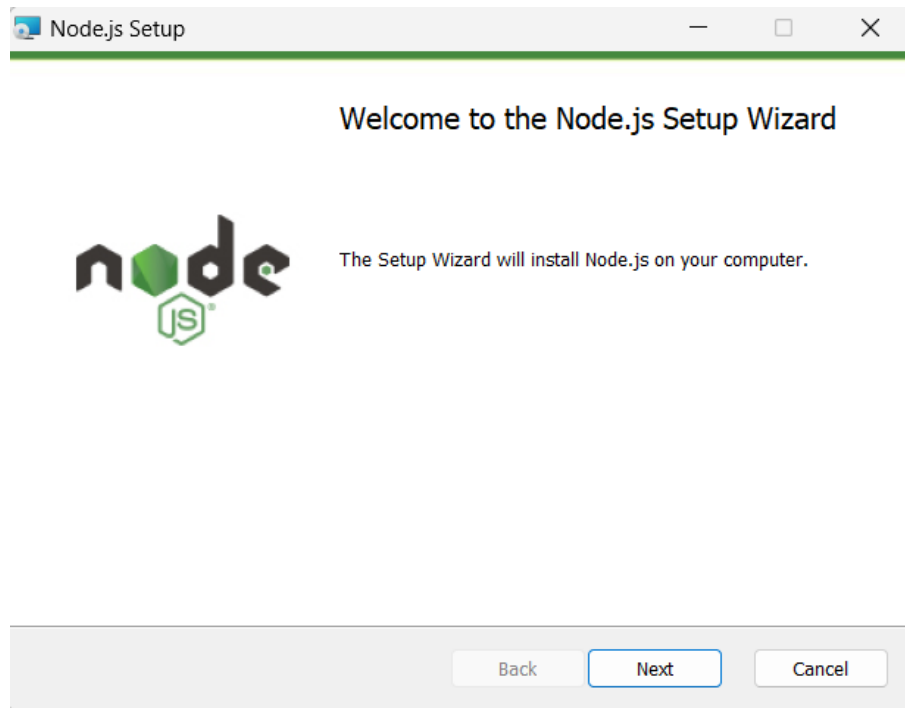


Ilustración 6 Ventana de inicio de instalación de node.js

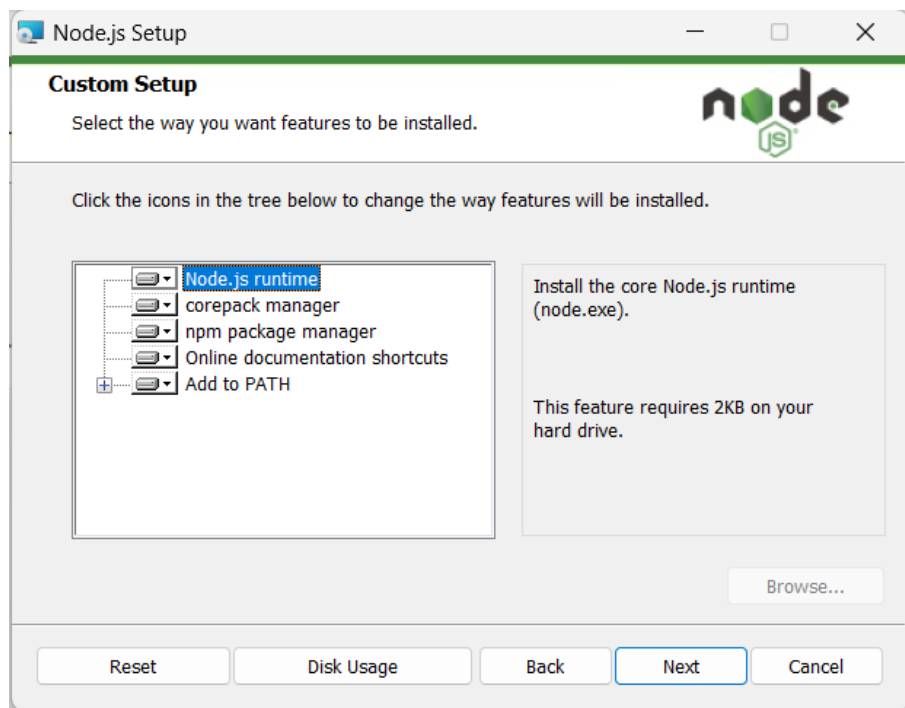


Ilustración 7 Ventana de instalación de node.js

Simplemente debemos avanzar en el proceso de instalación hasta llegar al apartado de instalar.

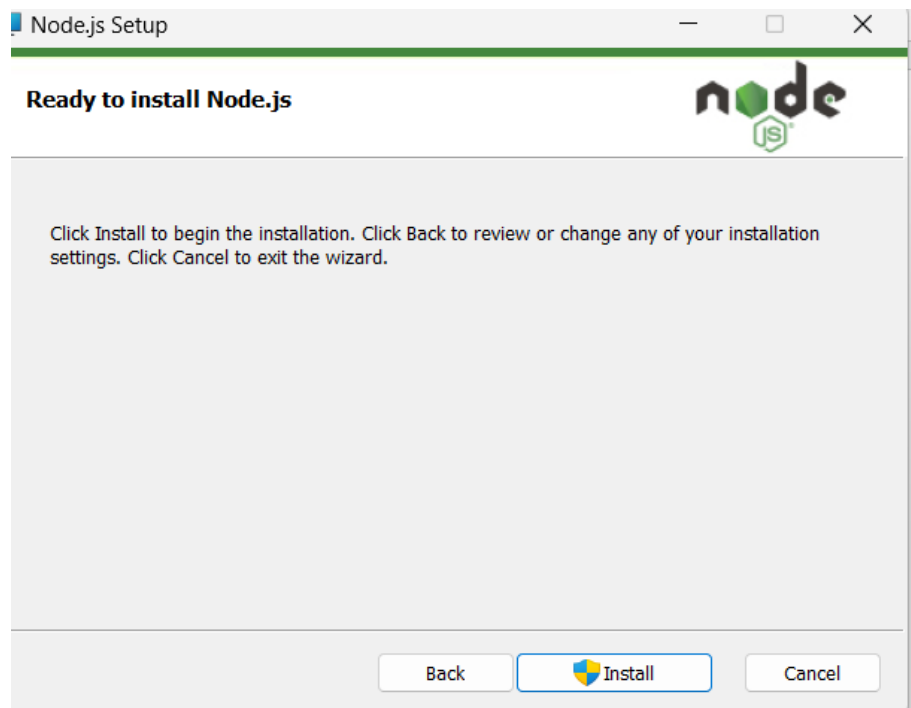


Ilustración 8 Ventana de instalación de Node.js Completada

Una vez descargado e instalado Node.js, es importante verificar que la instalación se haya realizado correctamente. Para ello, abrimos la ventana de comandos y escribimos `node -v`, seguido de la tecla Enter. Deberíamos ver la versión instalada de Node.js, que en mi caso es la 22.3.0. Para asegurarnos de que también se instaló el Node Package Manager (NPM), escribimos `npm -v` y presionamos Enter. En mi caso, la versión de NPM instalada es la 10.8.1.

```
C:\Users\Joel Arriola>node --version
v22.3.0

C:\Users\Joel Arriola>npm --version
10.8.1

C:\Users\Joel Arriola>
```

Ilustración 9 Verificación de Versión de Node.js

Frameworks

Hay varios frameworks disponibles para el desarrollo en Ethereum, y aunque es difícil cubrirlos todos, a continuación, se presenta una introducción a algunos de los más importantes.

Truffle

Truffle es un conjunto de herramientas diseñado para facilitar la programación de contratos inteligentes y el desarrollo de aplicaciones sostenibles y profesionales sobre la blockchain, utilizando la Máquina Virtual de Ethereum (EVM). Además, permite realizar diversas pruebas en un entorno de desarrollo integrado y amigable para el desarrollador.

La Máquina Virtual de Ethereum (EVM) es el entorno que posibilita a los desarrolladores crear contratos y aplicaciones inteligentes que la blockchain puede ejecutar y entender.

El principal objetivo de Truffle es proporcionar un entorno de desarrollo en la blockchain que simplifique el trabajo de los desarrolladores dedicados a la creación de aplicaciones descentralizadas (DApps) y contratos inteligentes en Ethereum.

Truffle ofrece un marco de prueba y una canalización de activos para Ethereum, lo que facilita que el proceso de desarrollo de aplicaciones para esta red sea más simple e intuitivo. A medida que evoluciona, el equipo de Truffle añade nuevas herramientas y características a este entorno, con el objetivo de que los desarrolladores dispongan de todo lo necesario dentro de un único espacio de trabajo para la creación, prueba, simulación, y otras tareas necesarias para perfeccionar las aplicaciones antes de lanzarlas a los usuarios finales.

El entorno de trabajo de Truffle se compone de tres componentes principales:

Truffle: Proporciona una herramienta de desarrollo con la capacidad de probar e implementar proyectos. Esta herramienta ha ganado notable popularidad, reflejada en el aumento de las descargas desde los repositorios respectivos.

Drizzle

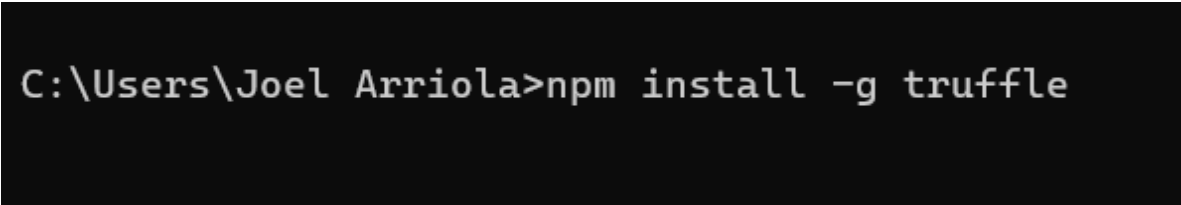
Drizzle es una biblioteca de desarrollo FrontEnd que se puede conectar directamente a los datos de un contrato inteligente. Esta herramienta permite a los desarrolladores crear interfaces de usuario que interactúan con contratos inteligentes, facilitando la construcción de aplicaciones descentralizadas (DApps) con Ethereum.

El conjunto de herramientas proporcionado por Truffle, que incluye Drizzle, forma la Suite de Truffle. Con esta suite, los desarrolladores pueden realizar diversas operaciones como:

- Soporte integrado para compilar, implementar y vincular contratos inteligentes.
- Pruebas automatizadas de contratos, lo que simplifica la verificación de su correcto funcionamiento.

- Compatibilidad tanto con aplicaciones de consola como con aplicaciones web, lo que facilita su integración en diferentes tipos de proyectos.
- Gestión de redes y paquetes, esencial para manejar las distintas configuraciones y dependencias en un proyecto Ethereum.
- Consola Truffle, que permite comunicarse directamente con contratos inteligentes, ofreciendo una forma interactiva de trabajar con ellos.
- Ejecutor de scripts externos, que permite ejecutar scripts dentro del entorno Truffle, ampliando las capacidades de automatización y desarrollo.
- Truffle se encuentra disponible en Truffle Suite.

Antes de proceder con la instalación de Truffle, es importante asegurarse de que Node.js esté correctamente instalado en el sistema. Puedes verificar la instalación de Node.js como se mencionó anteriormente.




```
C:\Users\Joel Arriola>npm install -g truffle
```

Ilustración 10 Comando de instalación de truffle

La instalación de Truffle es un proceso simple que se puede realizar fácilmente utilizando Node Package Manager (npm). Solo necesitas ejecutar el siguiente comando en la terminal:

Este comando instalará Truffle de manera global en tu sistema. El proceso tomará unos minutos y, una vez finalizado, podrás verificar que Truffle se haya instalado correctamente. Para hacerlo, utiliza el comando `truffle` en la terminal. Esto debería mostrarte una lista de opciones de ayuda, confirmando que Truffle está listo para ser utilizado:



```
C:\Users\Joel Arriola>truffle
```

Ilustración 11 Confirmación que truffle está listo para ser utilizado

Este comando te mostrará la información necesaria para comenzar a utilizar Truffle en tus proyectos de desarrollo en Ethereum.

```
Truffle v5.11.5 - a development framework for Ethereum

Usage: truffle <command> [options]

Commands:
  truffle build      Execute build pipeline (if configuration present)
  truffle call       Call read-only contract function with arguments
  truffle compile    Compile contract source files
  truffle config     Set user-level configuration options
  truffle console    Run a console with contract abstractions and commands
                    available
  truffle create     Helper to create new contracts, migrations and tests
  truffle dashboard  Start Truffle Dashboard to sign development transactions
                    using browser wallet
  truffle db         Database interface commands
  truffle debug      Interactively debug any transaction on the blockchain
                    (alias for migrate)
  truffle deploy     Open a console with a local development blockchain
  truffle develop    Execute a JS module within this Truffle environment
  truffle exec       List all commands or provide information about a specific
                    command
  truffle init       Initialize new and empty Ethereum project
  truffle migrate    Run migrations to deploy contracts
  truffle networks   Show addresses for deployed contracts on each network
  truffle obtain     Fetch and cache a specified compiler
  truffle opcode     Print the compiled opcodes for a given contract
  truffle preserve   Save data to decentralized storage platforms like IPFS and
                    Filecoin
  truffle run        Run a third-party command
  truffle test       Run JavaScript and Solidity tests
  truffle unbox      Download a Truffle Box, a pre-built Truffle project
  truffle version    Show version number and exit
  truffle watch      Watch filesystem for changes and rebuild the project
                    automatically

Options:
  --help    Show help                                [boolean]
  --version Show version number                        [boolean]

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

Ilustración 12 Información de Truffle

Más adelante, utilizaremos Truffle para probar e implementar contratos inteligentes en la blockchain de Ethereum. Sin embargo, antes de adentrarnos en su uso, continuaremos explorando otros frameworks que son comúnmente utilizados en el desarrollo de aplicaciones para la cadena de bloques de Ethereum. Esto nos permitirá tener una visión más completa de las herramientas disponibles y cómo pueden integrarse en nuestros proyectos de desarrollo.

Embark

Embark es una plataforma de desarrollo integral y robusta para la creación e implementación de aplicaciones descentralizadas (DApps). Facilita todo el ciclo de vida del desarrollo de contratos inteligentes, desde la configuración hasta las pruebas y el despliegue. Además, Embark se integra con tecnologías como Swarm, IPFS, y Whisper, lo que permite un desarrollo más completo y escalable de aplicaciones descentralizadas.

Una característica destacada de Embark es Cockpit, una interfaz web que proporciona un entorno de desarrollo integrado (IDE). Esta herramienta facilita enormemente el desarrollo y

la depuración de DApps, permitiendo a los desarrolladores trabajar de manera más eficiente y centrada.

Para instalar Embark, se puede utilizar el siguiente comando a través de Node Package Manager (npm):

```
C:\Users\Joel Arriola>npm install -g embark
```

Ilustración 14 Comando de instalación de embark

Brownie

Brownie es un framework basado en Python diseñado para el desarrollo y prueba de contratos inteligentes en la blockchain de Ethereum. Es compatible tanto con Solidity como con Vyper, y proporciona herramientas relevantes para la prueba y depuración de contratos inteligentes. Brownie es ideal para los desarrolladores que prefieren trabajar en un entorno Python y buscan un framework que les permita integrar pruebas y depuración de manera eficiente. Puedes encontrar más información sobre Brownie en su documentación oficial: <https://eth-brownie.readthedocs.io/en/stable/>.

Hardhat

Hardhat es un entorno de desarrollo completo para Ethereum que facilita la compilación, implementación, prueba y depuración de software en la blockchain. Hardhat permite a los desarrolladores gestionar y automatizar las tareas repetitivas que son parte integral del proceso de creación de contratos inteligentes y DApps. Además, proporciona un flujo de trabajo flexible que permite extender sus funcionalidades, haciendo que la compilación, ejecución y prueba de contratos inteligentes sea más eficiente. Para más información sobre Hardhat, puedes visitar su sitio web oficial en <https://hardhat.org/>.

OpenZeppelin

OpenZeppelin es una suite de productos de seguridad diseñada para ofrecer soluciones integrales en la creación, gestión e inspección de todos los aspectos del desarrollo y las operaciones de software en proyectos de Ethereum. OpenZeppelin se ha consolidado como un estándar en la industria para garantizar la seguridad y confiabilidad en contratos inteligentes y aplicaciones descentralizadas. Puedes encontrar más información sobre sus productos y servicios en <https://openzeppelin.com/sdk/>.

En esta sección, hemos repasado algunos de los frameworks más importantes que se utilizan en el ecosistema Ethereum para el desarrollo de contratos inteligentes y aplicaciones

descentralizadas. En la siguiente sección, profundizaremos en las herramientas disponibles para escribir e implementar estos contratos, proporcionando una visión clara de cómo integrarlas en tu flujo de trabajo de desarrollo.

Entorno de Desarrollo Integrado (IDE)

Un Entorno de Desarrollo Integrado (IDE) es una herramienta fundamental para los desarrolladores, ya que combina en una sola aplicación todo lo necesario para escribir, compilar, depurar y desplegar software. Un IDE típico incluye un editor de código fuente, herramientas para la automatización de tareas de construcción y un depurador que facilita la identificación y corrección de errores en el código. Además, muchos IDEs modernos ofrecen funciones como autocompletado inteligente (IntelliSense), que sugiere código a medida que escribes, agilizando el proceso de desarrollo.

Visual Studio Code

Visual Studio Code se ha vuelto bastante popular y se usa comúnmente para el desarrollo de Solidity. En el contexto del desarrollo de contratos inteligentes con Solidity, el código se puede escribir tanto en modo offline, utilizando editores de código como Visual Studio Code, como en modo online, utilizando herramientas como Remix. En Visual Studio Code, por ejemplo, existe una extensión específica para Solidity que proporciona resaltado de sintaxis, formato y autocompletado, haciendo que el proceso de escritura y edición del código sea más eficiente y menos propenso a errores. Esta extensión puede instalarse fácilmente desde la tienda de extensiones de Visual Studio Code, mejorando significativamente la experiencia de desarrollo en Solidity.



```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3
4 contract Almacenamiento {
5     uint256 private numero;
6
7     // Evento que se emite cuando se actualiza el número
8     event NumeroActualizado(uint256 nuevoNumero);
9
10    // Función para almacenar un número
11    function almacenarNumero(uint256 _numero) public {
12        numero = _numero;
13        emit NumeroActualizado(_numero);
14    }
15
16    // Función para recuperar el número almacenado
17    function recuperarNumero() public view returns (uint256) {
18        return numero;
19    }
20 }
```

Ilustración 15 Visual Studio Code Solidity

Remix IDE

Remix IDE es una herramienta de desarrollo de contratos inteligentes que se ofrece tanto en versión web como de escritorio y es de código abierto. Este entorno de desarrollo facilita un ciclo de desarrollo ágil y está equipado con una variedad de complementos, lo que proporciona una interfaz intuitiva y rica en funcionalidades. Remix IDE se utiliza no solo para todo el proceso de desarrollo de contratos inteligentes, sino también como una plataforma educativa para aprender y enseñar sobre Ethereum.

Con Remix, puedes compilar y ejecutar contratos inteligentes escritos en Solidity de manera rápida y eficiente. Su entorno permite experimentar con contratos y realizar pruebas sin necesidad de configurar un entorno de desarrollo complejo.

Para comenzar a usar Remix IDE y probar un contrato inteligente básico, como HelloWorld.sol, sigue estos pasos:

Abre Remix IDE en tu navegador visitando Remix Ethereum IDE.
<https://remix.ethereum.org/>

En la interfaz de Remix, selecciona Nuevo Archivo (New File).

Elige el entorno de desarrollo para Solidity, creando un nuevo archivo con extensión .sol.

Con estos pasos, estarás listo para comenzar a desarrollar y probar contratos inteligentes en Remix IDE.

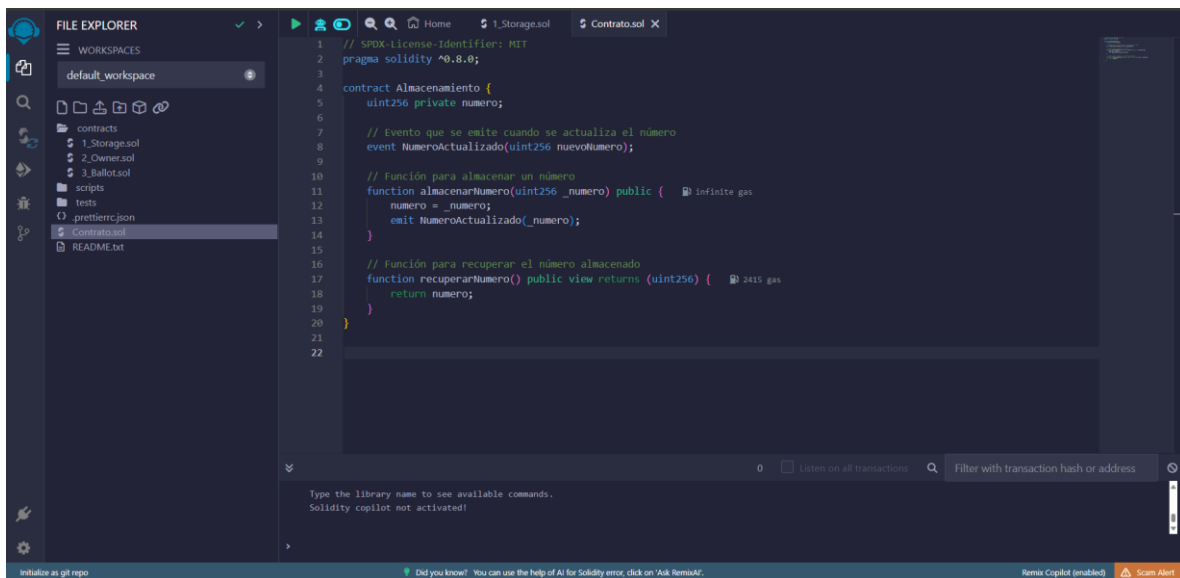


Ilustración 16 Solidity Remix IDE

Configuración del Entorno de Desarrollo

Instalación de Herramientas

Instalación de Node.js y npm

Descargar Node.js: Ve a <https://nodejs.org/> en y descarga el instalador para tu sistema operativo.

Instalación: Ejecuta el instalador y sigue las instrucciones para instalar Node.js y npm (Node Package Manager).

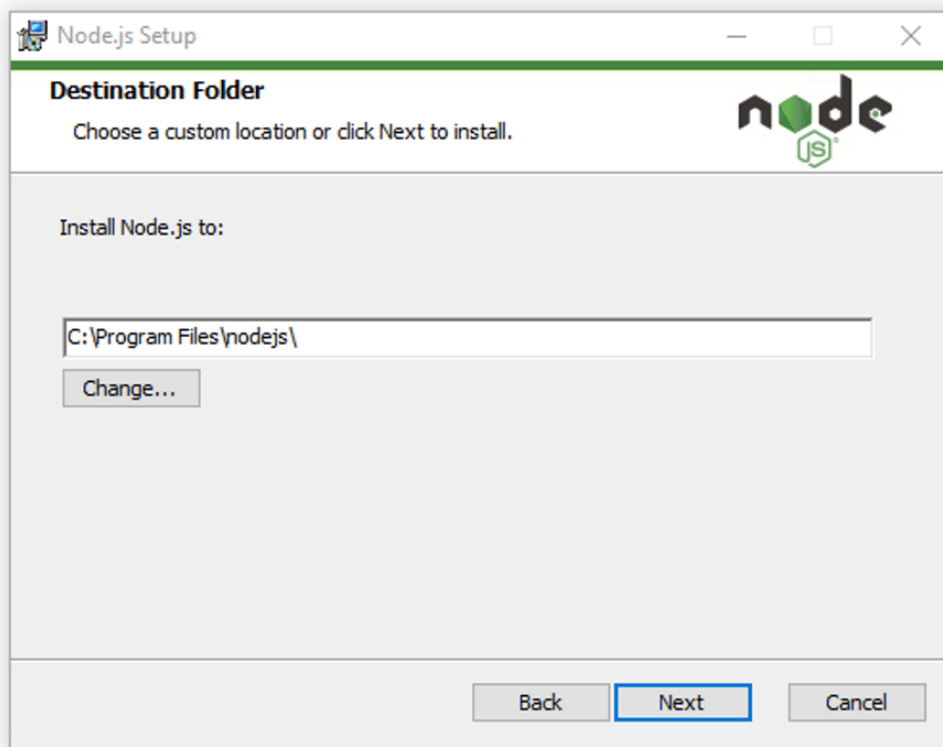


Ilustración 17 Instalación de node.js

Instalación de Hardhat

Hardhat es un entorno de desarrollo para Ethereum que facilita la creación, despliegue y prueba de contratos inteligentes.

1. Crear un Directorio para tu Proyecto:

```
mkdir my-smart-contracts  
  
cd my-smart-contracts
```

2. Inicializar un Proyecto de Node.js:

```
npm init -y
```

3. Instalar Hardhat:

```
npm install --save-dev hardhat
```

4. Configurar Hardhat:

```
npx hardhat
```

archivos básicos y Selecciona "Create a basic sample project" y sigue las instrucciones para configurar tu proyecto Hardhat. Esto generará una estructura de directorios.

Estructura del Proyecto

Tu proyecto debería tener la siguiente estructura después de configurar Hardhat:

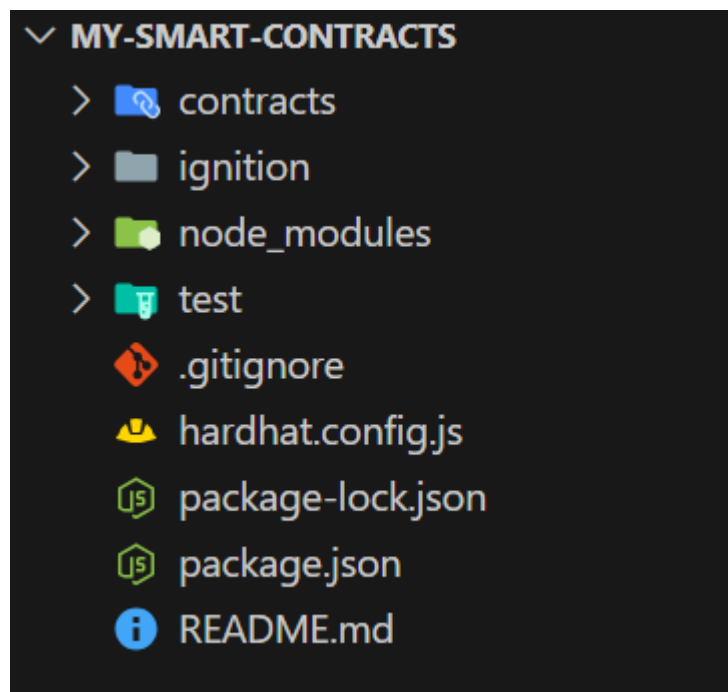


Ilustración 18 Estructura del proyecto

Desarrollo de un Contrato Inteligente

1. Crear un Nuevo Contrato

Ejemplo: Contrato de Almacenamiento Básico

1. Crear un Archivo del Contrato

Navega al directorio `contracts` y crea un archivo llamado `Storage.sol`:

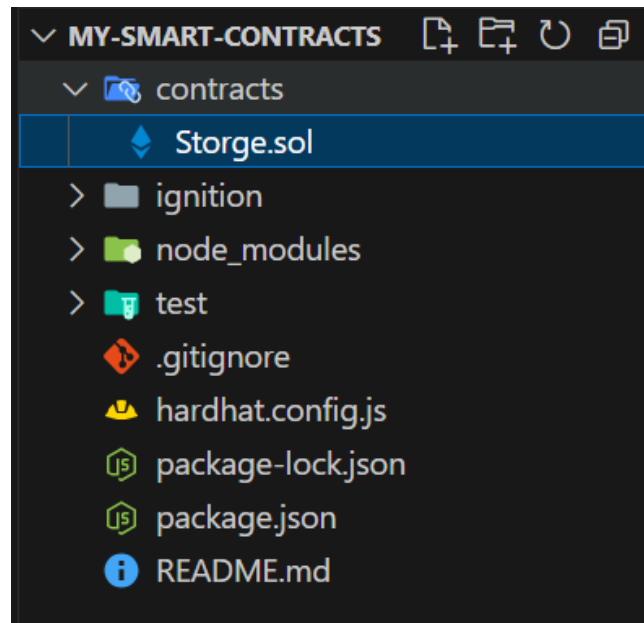


Ilustración 19 Creación de Archivo sol

2. Escribir el Código del Contrato

Abre `Storage.sol` en tu editor de código y añade el siguiente código:



```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3
4 contract Storage {
5     uint256 private storedNumber;
6
7     // Función para almacenar un número
8     function store(uint256 _number) public {
9         storedNumber = _number;
10    }
11
12    // Función para recuperar el número almacenado
13    function retrieve() public view returns (uint256) {
14        return storedNumber;
15    }
16 }
```

Ilustración 20 Sintaxis Código sol

Explicación del Código

- `// SPDX-License-Identifier: MIT`: Licencia del contrato.
- `pragma solidity ^0.8.0;`: Define la versión del compilador Solidity.
- `contract Storage`: Declara el contrato Storage.
- `uint256 private storedNumber;`: Variable privada para almacenar el número.
- `function store(uint256 _number) public`: Función pública para almacenar un número.
- `function retrieve() public view returns (uint256)`: Función pública de solo lectura para recuperar el número almacenado.

2. Compilación del Contrato

Para compilar el contrato, ejecuta:

`npx hardhat compile` "Esto generará los archivos ABI y bytecode necesarios en el directorio `artifacts`".

3. Despliegue del Contrato

1. Crear el Archivo de Despliegue

Crea el directorio `scripts` y crea un archivo llamado `deploy.js`:

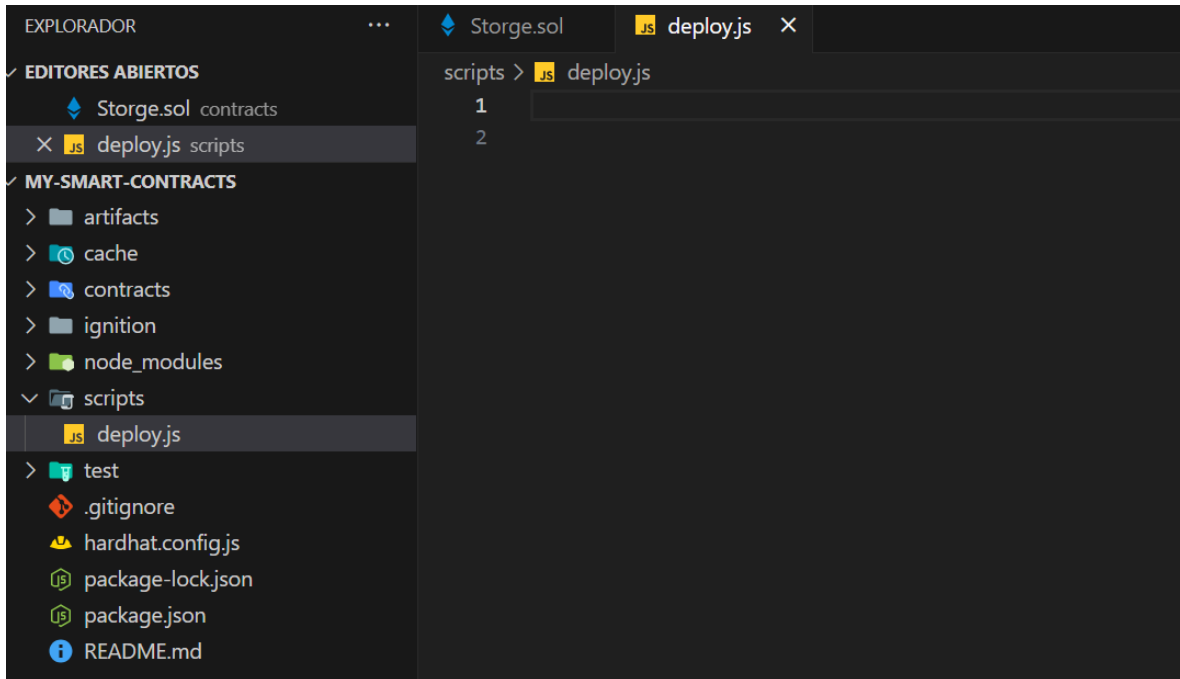


Ilustración 21 Scripts archivo deploy.js

2. Escribir el Código del Script de Despliegue

Abre `deploy.js` en tu editor de código y añade el siguiente código:

```
1  async function main() {
2    // Obtén la cuenta que desplegará el contrato
3    const [deployer] = await ethers.getSigners();
4    console.log("Desplegando contratos con la cuenta:", deployer.address);
5
6    // Obtén el contrato Storage
7    const Storage = await ethers.getContractFactory("Storage");
8    // Despliega el contrato
9    const storage = await Storage.deploy();
10   await storage.deployed();
11
12   console.log("Contrato desplegado a:", storage.address);
13 }
14
15 main().catch((error) => {
16   console.error(error);
17   process.exitCode = 1;
18 });
```

Explicación del Código

- `const [deployer] = await ethers.getSigners();` :: Obtiene la cuenta para desplegar el contrato.
- `const Storage = await ethers.getContractFactory("Storage");` :: Obtiene el contrato `Storage`.
- `const storage = await Storage.deploy();` :: Despliega el contrato.
- `console.log("Contrato desplegado a:", storage.address);` :: Muestra la dirección del contrato desplegado.

Ejecutar el Script de Despliegue

Para desplegar el contrato, ejecuta:

```
npx hardhat run scripts/deploy.js --network localhost
```

Asegúrate de que tengas un nodo Ethereum local en ejecución.

Interactuar con el Contrato

Crear un Script de Interacción

1. Crear el Archivo de Interacción

Navega al directorio `scripts` y crea un archivo llamado `interact.js`:

Escribir el Código del Script de Interacción

Abre `interact.js` en tu editor de código y añade el siguiente código:

```
1  async function main() {
2    const [deployer] = await ethers.getSigners();
3    const storageAddress = "0xYourContractAddress"; // Reemplaza con la dirección de tu contrato
4
5    const Storage = await ethers.getContractFactory("Storage");
6    const storage = Storage.attach(storageAddress);
7
8    // Almacenar un número
9    const tx = await storage.store(42);
10   await tx.wait();
11
12   // Recuperar el número
13   const storedNumber = await storage.retrieve();
14   console.log("Número almacenado:", storedNumber.toString());
15 }
16
17 main().catch((error) => {
18   console.error(error);
19   process.exitCode = 1;
20 });
```

Explicación del Código

- `const storageAddress = "0xYourContractAddress";` : Dirección del contrato desplegado.
- `const storage = Storage.attach(storageAddress);` : Conecta el script al contrato desplegado.
- `const tx = await storage.store(42);` : Llama a la función `store` para almacenar el número 42.
- `const storedNumber = await storage.retrieve();` : Llama a la función `retrieve` para obtener el número almacenado.

Ejecutar el Script de Interacción: Para interactuar con el contrato, ejecuta:

```
npx hardhat run scripts/interact.js --network localhost
```

Pruebas del Contrato

Crear Pruebas con Hardhat

1. Crear el Archivo de Pruebas

Navega al directorio `test` y crea un archivo llamado `storage-test.js`:

2. Escribir el Código de Pruebas

Abre `storage-test.js` en tu editor de código y añade el siguiente código:

```
1  const { expect } = require("chai");
2
3  describe("Storage Contract", function () {
4      it("Should store and retrieve a number", async function () {
5          const [owner] = await ethers.getSigners();
6
7          const Storage = await ethers.getContractFactory("Storage");
8          const storage = await Storage.deploy();
9          await storage.deployed();
10
11          // Almacena un número
12          await storage.store(100);
13
14          // Recupera el número
15          expect(await storage.retrieve()).to.equal(100);
16      });
17  });
```

Explicación del Código

- `const { expect } = require("chai");` :: Importa la biblioteca de aserciones.
- `const Storage = await ethers.getContractFactory("Storage");` :: Obtiene el contrato `Storage`.
- `await storage.store(100);` :: Llama a la función `store` para almacenar el número 100.
- `expect(await storage.retrieve()).to.equal(100);` :: Verifica que el número almacenado sea 100.

Ejecutar las Pruebas

Para ejecutar las pruebas, ejecuta:

```
npx hardhat test
```