Cómo crear un Smart Contract desde SSH, Leer y escribir desde un VPS.

Scolcoin Wei Chain 2023
Por el equipo de Accolombia 4.0 S.A.S y Blockchain Technology
https://scolcoin.com

Creación Scolcoin Nativo: Enero 3, 2018
Creación Scolcoin Wei Chain: marzo 13, 2023

La Blockchain de Colombia descentralizada gobernada democráticamente por personas, el activo digital más antiguo del mercado nacional y de código abierto. La tecnología Blockchain de Scolcoin Wei Chain convierte a nuestro

Verifique su transacción en Explorer:

PASOS:

- Primer paso:
  Debes tener instalado un servidor VPS en tu VPS y tenerlo desbloqueado
- Segundo Paso:
  Debes tener instalado las siguientes librerías:
  - ✓ Node.js v8.9.4 LTS or later
  - ✓ Git
  - ✓ NPM
  - ✓ Geth Scolcoin Wei Chain

Inicializar nuestro proyecto:

```
mkdir hello-world
cd hello-world
```

Ahora que estamos dentro de la carpeta de nuestro proyecto, usaremos npm init para inicializar el proyecto

```
npm init # (or npm init --yes)
```

Realmente no importa cómo responda las preguntas de instalación.

Descargar

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

```
npm install --save-dev @nomiclabs/hardhat-waffle ethereum-waffle chai
@nomiclabs/hardhat-ethers ethers
```

Consulte esta página para obtener más detalles sobre las instrucciones de instalación

Crear proyecto Hardhat

```
npx hardhat
```

Luego debería ver un mensaje de bienvenida y una opción para seleccionar lo que desea hacer. Seleccione "crear un hardhat.config.js vacío":

Esto generará un archivo para nosotros, que es donde especificaremos toda la configuración de nuestro proyecto

Agregar carpetas de proyectos

```
mkdir contracts
mkdir scripts
dir
```

- **contracts/** es donde mantendremos nuestro archivo de código de contrato inteligente.
- **scripts/** es donde mantendremos los scripts para implementar e interactuar con nuestro contrato.
- **test/:** directorio de archivos de prueba para probar su aplicación y contratos
- **hardhat-config.js**: archivo de configuración del hardhat

Redactar nuestro contrato

```
nano contracts/HelloWorld.sol
```

ingresamos el Smart contract:

```
pragma solidity >=0.7.3;

contract HelloWorld {

    event UpdatedMessages(string oldStr, string newStr);

    string public message;

   constructor(string memory initMessage) {
   message = initMessage;
   }

  function update(string memory newMessage) public {
    string memory oldMsg = message;
    message = newMessage;
    emit UpdatedMessages(oldMsg, newMessage);
  }
}
```

Ctrl X le damos que Y y enter para almacenar el Smart contract HelloWorld.sol

Este es un contrato inteligente súper simple que almacena un mensaje en el momento de la creación y se puede actualizar llamando a la updatefunción

Instale Ethers.js

Ethers.js es una biblioteca que facilita la interacción y la realización de solicitudes a Ethereum al combinar los métodos JSON-RPC estándar con métodos más fáciles de usar.

Hardhat hace que sea muy fácil integrar complementos para herramientas adicionales y funcionalidades extendidas. Aprovecharemos el complemento Ethers para la implementación de contratos (Ethers.js tiene algunos métodos de implementación de contratos súper limpios).

En el directorio de tu proyecto escribe:

```
npm install --save-dev @nomiclabs/hardhat-ethers "ethers@^5.0.0"
```

También necesitaremos Scolcoin Wei Chain en nuestro próximo paso con el archivo hardhat.config.js

Actualizar hardhat.config.js

```
nano hardhat.config.js
```

borramos todo y ponemos lo siguiente:

```
require("@nomiclabs/hardhat-ethers");

module.exports = {
  solidity: "0.7.3",
  defaultNetwork: "mainnet",
  networks: {
   localhost: {
     url: "http://127.0.0.1:3545"
   },
     hardhat: {},
    testnet: {
        url: "https://testnet-rpc.scolcoin.com",
```

```
      chainId: 6552,
      gasPrice: 20000000000,
    },
    mainnet: {
      url: "https://mainnet-rpc.scolcoin.com",
      chainId: 65450,
      gasPrice: 20000000000,
    }
  },
  solidity: {
  version: "0.7.3",
  settings: {
    optimizer: {
      enabled: true
    }
  }
  },
  paths: {
    sources: "./contracts",
    tests: "./test",
    cache: "./cache",
    artifacts: "./artifacts"
  },
  mocha: {
    timeout: 20000
  }

};
```

Ctrl X le damos que Y y enter para almacenar la configuración de hardhat.config.js

Compilar nuestro contrato

npx hardhat compile

Es posible que reciba una advertencia sobre , pero no debe preocuparse por eso. ¡Esperemos que todo lo demás se vea bien!
SPDX license identifier not provided in source file

Escriba nuestro script de implementación

nano /scripts/deploy.js

```
async function main() {
  const HelloWorld = await ethers.getContractFactory("HelloWorld");

  // Start deployment, returning a promise that resolves to a contract object
  const hello_world = await HelloWorld.deploy("Hello World!");
  console.log("Contract deployed to address:", hello_world.address);
}

main()
 .then(() => process.exit(0))
 .catch(error => {
   console.error(error);
   process.exit(1);
 });
```

Ctrl X le damos que Y y enter para almacenar el archivo deploy.js

Implementar nuestro contrato

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

Entonces deberías ver algo como:
Contract deployed to address: 0xCAFBf889bef0617d9209Cf96f18c850e901A6D61

Copie y pegue esta dirección para guardarla en algún lugar, ya que usaremos esta dirección para interactuar con nuestro Smart contract.

Nota: felicidades creaste el Smart Contract llamado HelloWorld puedes validarlo en el explorador https://explorer.scolcoin.com

Vamos al siguiente paso a leer el contrato.

crea un archivo interact_read.js

nano /scripts/interact_read.js

ingresamos lo siguiente:

```
//es el contrato que copiaste y lo remplazas
const CONTRACT_ADDRESS = "0x00..0";

// ABI ubícalo en la carpeta hello-world/artifacts/contracts/HelloWorld.sol/
const contract = require("~/hello-
world/artifacts/contracts/HelloWorld.sol/HelloWorld.json");

//conexión:
   const url = "http://localhost:3545";
   const provider = new ethers.providers.JsonRpcProvider(url);
   const signer = provider.getSigner(0);

// solo lectura connection to the contract
   const helloWorldContract = new ethers.Contract(CONTRACT_ADDRESS, contract.abi,
provider);

// ...

async function main() {
  const message = await helloWorldContract.message();
  console.log("The message is: " + message);
}
main();
```

Ctrl X le damos que Y y enter para almacenar el archivo interact_read.js

- contrato ABI: se ubica en tu carpeta principal helloworld > artifacts > contracts > HelloWorld.scol > HelloWorld.json - Nuestro contrato ABI (Application Binary Interface) es la interfaz para interactuar con nuestro contrato inteligente. Hardhat (y Truffle) genera automáticamente una ABI para nosotros y la guarda en el archivo HelloWorld.json

Lea el mensaje de inicio

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

Después de ejecutar el archivo en la terminal, deberíamos ver esta respuesta:
The message is: Hello world!

¿Recuerdas cuando implementamos nuestro contrato? Ahora vamos a leer ese mensaje almacenado en nuestro contrato inteligente e imprimirlo en la consola.initMessage = "Hello world!"

Actualice el mensaje

¡Ahora, en lugar de solo leer el mensaje, también podemos actualizar el mensaje guardado en nuestro contrato inteligente usando la updatefunción! Bastante genial

nano scripts/interact_update.js

ingresamos el contenido:

```
//Contrato
const CONTRACT_ADDRESS = "0x00..0";

// ABI
const contract = require("/root/private/hello-world/artifacts/contracts/HelloWorld.sol/HelloWorld.json");

//conexion:
   const url = "http://localhost:3545";
   const provider = new ethers.providers.JsonRpcProvider(url);
   const signer = provider.getSigner(0);

// solo lectura
//    const helloWorldContract = new ethers.Contract(CONTRACT_ADDRESS, contract.abi, provider);

// escritura
    const helloWorldContract = new ethers.Contract(CONTRACT_ADDRESS, contract.abi, signer);

// ...

async function main() {
  const message = await helloWorldContract.message();
  console.log("The message is: " + message);

  console.log("Updating the message...");
```

```
  const tx = await helloWorldContract.update("Contrato exitoso.");
  await tx.wait();
}
main();
```

Ctrl X le damos que Y y enter para el archivo interact_update.js

Nota recuerda siempre tener el contrato que copiaste.

Tenga en cuenta que hacemos una llamada al objeto de transacción devuelto. Esto garantiza que nuestro script espere a que se extraiga la transacción en la cadena de bloques antes de continuar. Si dejara esta línea fuera, es posible que su secuencia de comandos no pueda ver el valor actualizado en su contrato..wait()message

Ejecutamos:
npx hardhat run scripts/interact_update.js --network localhost

Mientras ejecuta ese script, puede notar que el paso tarda un tiempo en cargarse antes de que se establezca el nuevo mensaje.Updating the message...

¡Eso se debe al proceso de minería!

Ahora verificamos con
npx hardhat run scripts/interact_read.js --network localhost

y tendremos el nuevo mensaje almacenado puedes hacer varias pruebas cambiando el archivo interact_update.js en la línea:
const tx = await helloWorldContract.update("Contrato exitoso.");
pon el mensaje que quieras lo grabas con Ctrl X y le das Y y luego Enter, y luego ejecutas:
npx hardhat run scripts/interact_update.js --network localhost
luego verificas el nuevo mensaje:
npx hardhat run scripts/interact_read.js --network localhost
y con esto aprendiste a interactuar con un contrato inteligente con un servidor VPS o local.

Ahora puedes hacer tus propios contratos y probar con ellos!