

L16: Lab - Contract Interaction

Module B: Ethereum & Smart Contracts

Blockchain & Cryptocurrency Course

December 2025

By the end of this lab, you will be able to:

- Navigate and use Remix IDE for smart contract development
- Deploy a SimpleStorage contract in Remix VM
- Interact with deployed contracts (read and write functions)
- Connect MetaMask to Remix and deploy to Sepolia testnet
- Verify deployed contracts on Etherscan
- Debug transaction failures using Remix and Etherscan

What we'll build:

- Deploy a SimpleStorage contract that stores key-value pairs
- Test contract functionality locally
- Deploy to Sepolia testnet
- Verify contract on Etherscan
- Interact with contract via Etherscan UI

Tools Required:

- Remix IDE (remix.ethereum.org)
- MetaMask browser extension
- Sepolia testnet ETH (from faucet)
- Etherscan account (for verification)

Estimated Time: 45-60 minutes

Remix is a web-based Solidity IDE with:

- **File Explorer:** Create and manage Solidity files
- **Solidity Compiler:** Compile contracts to bytecode
- **Deploy & Run:** Deploy and interact with contracts
- **Plugin Manager:** Add tools (debugger, static analyzers, Etherscan verifier)
- **Terminal:** View transaction logs and errors

Key Features:

- No installation required (runs in browser)
- Multiple VM environments (JavaScript VM, Injected Provider, External HTTP)
- Built-in debugger with step-through execution
- Gas estimation and optimization hints
- Integration with MetaMask, Hardhat, Ganache

Step 1: Create SimpleStorage Contract

In Remix File Explorer, create SimpleStorage.sol:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract SimpleStorage {
    // State variable to store a number
    uint256 private storedData;

    // Event emitted when data is updated
    event DataUpdated(uint256 oldValue, uint256 newValue, address updatedBy);

    // Store a number
    function set(uint256 x) public {
        uint256 oldValue = storedData;
        storedData = x;
        emit DataUpdated(oldValue, x, msg.sender);
    }

    // Retrieve the stored number
    function get() public view returns (uint256) {
        return storedData;
    }

    // Increment by a given amount
    function increment(uint256 amount) public {
        uint256 oldValue = storedData;
        storedData += amount;
        emit DataUpdated(oldValue, storedData, msg.sender);
    }
}
```

Step 2: Compile the Contract

Using Solidity Compiler tab:

- ➊ Click **Solidity Compiler** icon (left sidebar)
- ➋ Select compiler version: **0.8.0 or higher**
- ➌ Enable **Auto compile** (optional but convenient)
- ➍ Click **Compile SimpleStorage.sol**
- ➎ Check for compilation errors in terminal
 - Green checkmark: Compilation successful
 - Red cross: Syntax or type errors
- ➏ Review warnings (if any):
 - Unused variables
 - Shadowing warnings
 - Gas optimization hints

Successful Compilation: You'll see ABI and Bytecode available for deployment

Step 3: Deploy in Remix VM

Using Deploy & Run Transactions tab:

- ➊ Click **Deploy & Run Transactions** icon
- ➋ Select **Environment: Remix VM (Shanghai)**
 - In-memory blockchain for testing
 - Pre-funded accounts with 100 ETH each
 - Fast, no real ETH required
- ➌ Select **Account:** One of the pre-funded accounts
- ➍ Set **Gas Limit:** 3000000 (default is fine)
- ➎ Select **Contract:** SimpleStorage
- ➏ Click **Deploy**
- ➐ Check terminal for transaction receipt:
 - Transaction hash
 - Contract address
 - Gas used

Step 4: Interact with Deployed Contract

In Deployed Contracts section:

Read Functions (view, free):

- 1 Click **get** button
- 2 Should return 0 (initial value)
- 3 No gas cost (call, not transaction)

Write Functions (state-changing, costs gas):

- 1 Enter 42 in **set** input field
- 2 Click **set** button
- 3 Check terminal for transaction details:
 - Gas used: approximately 43,000-45,000
 - Event emitted: DataUpdated(0, 42, your_address)
- 4 Click **get** again
- 5 Should now return 42
- 6 Try **increment(10)**
- 7 Click **get** - should return 52

Step 5: Inspect Events

Events are logged in transaction receipts:

- ❶ In terminal, click on a transaction hash
- ❷ Expand transaction details:
 - **status:** 0x1 (success) or 0x0 (failure)
 - **from:** Your account address
 - **to:** Contract address
 - **gas:** Total gas used
 - **logs:** Array of emitted events
- ❸ Expand **logs[0]**:
 - **event:** DataUpdated
 - **args:** [oldValue: 0, newValue: 42, updatedBy: 0x5B38...]

Use Case: Events are indexed and searchable off-chain (e.g., by front-end dApps)

Step 6: MetaMask Setup

Prepare MetaMask for testnet deployment:

- ❶ Install MetaMask browser extension (metamask.io)
- ❷ Create new wallet or import existing one
- ❸ Switch to **Sepolia Test Network**:
 - Click network dropdown (top of MetaMask)
 - Enable **Show test networks** in settings
 - Select **Sepolia**
- ❹ Get test ETH from faucet:
 - Visit sepoliafaucet.com or faucets.chain.link
 - Enter your address
 - Request 0.5 ETH (typically arrives in 30-60 seconds)
- ❺ Verify balance in MetaMask shows test ETH

Important: Never send real ETH to test networks! Test ETH has no value.

Step 7: Deploy to Sepolia Testnet

Connect Remix to MetaMask:

- ❶ In Remix Deploy tab, change **Environment** to **Injected Provider - MetaMask**
- ❷ MetaMask popup appears - click **Connect**
- ❸ Select account with test ETH
- ❹ Verify Remix shows:
 - Environment: Custom (11155111) network [Sepolia chain ID]
 - Account: Your MetaMask address
 - Balance: Your test ETH amount
- ❺ Select **SimpleStorage** contract
- ❻ Click **Deploy**
- ❼ MetaMask popup for transaction confirmation:
 - Review gas fee (typically 0.0005-0.002 ETH)
 - Click **Confirm**
- ❽ Wait for transaction confirmation (12-15 seconds on Sepolia)
- ❾ Copy contract address from Remix console

Step 8: Verify Contract on Etherscan

Make source code publicly verifiable:

- ➊ Go to **sepolia.etherscan.io**
- ➋ Paste contract address in search bar
- ➌ Click **Contract** tab
- ➍ Click **Verify and Publish**
- ➎ Fill in form:
 - Compiler Type: Solidity (Single file)
 - Compiler Version: Match Remix (e.g., v0.8.20)
 - License Type: MIT
- ➏ Paste entire Solidity source code
- ➐ Click **Verify and Publish**
- ➑ Wait for verification (10-30 seconds)
- ➒ Green checkmark appears: "Contract Source Code Verified"

Step 9: Interact via Etherscan

Use Etherscan's Read/Write UI:

Read Contract (no wallet needed):

- 1 Click **Read Contract** tab
- 2 Click **get** - shows current stored value
- 3 No transaction required, instant response

Write Contract (requires MetaMask):

- 1 Click **Write Contract** tab
- 2 Click **Connect to Web3** - connects MetaMask
- 3 Enter value in **set** input (e.g., 100)
- 4 Click **Write**
- 5 MetaMask popup - confirm transaction
- 6 Wait for confirmation
- 7 Go to **Read Contract** - verify new value

Advantage: No need for custom UI, direct blockchain interaction

Step 10: View Transaction History

Analyze all interactions with your contract:

- ① On Etherscan contract page, click **Transactions** tab
- ② See list of all transactions:
 - Contract deployment (labeled “Contract Creation”)
 - All `set()` and `increment()` calls
- ③ Click on any transaction hash to see:
 - Block number and timestamp
 - From address (who called function)
 - Gas used and fee paid
 - Input data (encoded function call)
 - Logs (emitted events)
- ④ Click **Logs** tab in transaction details:
 - See decoded DataUpdated events
 - Indexed parameters are searchable

Error 1: Transaction Reverted

- **Cause:** `require()` or `assert()` failed, out of gas
- **Debug:** Check Remix debugger, view revert reason in Etherscan
- **Example:** “Insufficient balance” error in token transfer

Error 2: Out of Gas

- **Cause:** Gas limit too low for operation
- **Fix:** Increase gas limit in MetaMask advanced settings
- **Prevention:** Use Remix gas estimation before deployment

Error 3: Nonce Too Low

- **Cause:** Pending transaction with same or higher nonce
- **Fix:** Wait for pending transaction, or cancel it in MetaMask

Error 4: Insufficient Funds

- **Cause:** Not enough ETH for gas fees
- **Fix:** Get more test ETH from faucet

Step-by-step transaction debugging:

- ① After transaction executes, click **Debug** button in terminal
- ② Remix Debugger opens with:
 - **Instructions:** EVM opcodes executed
 - **Solidity Locals:** Variable values at each step
 - **Step Detail:** Gas used, stack, memory
- ③ Use controls to navigate:
 - Step over (next instruction)
 - Step into (function call)
 - Step out (return from function)
 - Jump to breakpoint
- ④ Inspect state changes:
 - Hover over variables to see values
 - Check storage changes in real-time
 - Identify where gas is consumed

Use Case: Understand why transaction failed or optimize gas usage

- ➊ **Remix IDE:** Powerful web-based environment for Solidity development with integrated compiler, debugger, and deployment tools
- ➋ **Local Testing:** Remix VM provides fast, free testing environment before deploying to real networks
- ➌ **MetaMask Integration:** Injected Provider connects Remix to testnets and mainnet via MetaMask
- ➍ **Testnet Deployment:** Sepolia testnet allows realistic testing with free test ETH from faucets
- ➎ **Etherscan Verification:** Publishing source code enables transparency and direct contract interaction via Etherscan UI
- ➏ **Debugging:** Remix debugger and Etherscan transaction logs are essential for troubleshooting

- ❶ Why is it important to test contracts on Remix VM before deploying to testnets?
- ❷ What are the security implications of verifying your contract source code on Etherscan?
- ❸ How would you estimate the gas cost of a complex function before deploying to mainnet?
- ❹ What are the advantages of using events over storing data in state variables for historical records?
- ❺ How would you handle a situation where your contract is deployed but has a critical bug?

Coming up next:

- Understanding the ERC-20 fungible token standard
- ERC-20 interface: `totalSupply`, `balanceOf`, `transfer`, `approve`, `transferFrom`
- Implementing ERC-20 from scratch
- Using OpenZeppelin's ERC-20 implementation
- Token economics: minting, burning, supply management
- Real-world examples: USDC, DAI, LINK

Preparation:

- Keep your Remix IDE and MetaMask setup
- Review events and mappings from L15
- Browse existing ERC-20 tokens on Etherscan (e.g., USDC)