Deploy a Smart Contract

The following guide deploys an ERC-721 contract on the Polygon Mumbai TestnetSuggest Edits

Smart Contract Platform provides an API for deploying, exploring, and interacting with smart contracts. The platform offers a powerful toolset for developers to build decentralized applications and for businesses to transition to web3.

This guide can also be followed to deploy smart contracts on the Ethereum network by changing the blockchain parameter in your request. Additionally, you can deploy to Mainnet by swapping out the Testnet API key for a Mainnet API key. See theTestnet vs Mainnet guide for more details.

OpenZeppelin

This tutorial will use OpenZeppelin's Contract Wizard to generate an ERC-721 smart contract. OpenZeppelin offers a secure smart contract development library - built on top of a foundation of community-vetted code.

The Contract Wizard includes:

1. Implementations of standards like ERC20 and ERC721.
2. Flexible role-based permissions scheme.
3. Reusable Solidity components to build custom contracts and complex decentralized systems.

Part One: Fund Your Wallet with Gas

If you already have a Circle developer-controlled wallet on Polygon Mumbai with Testnet MATIC, proceed topart two . If you haven't created a developer-controlled wallet, head to our quickstartCreate Your First Developer-Controlled Wallet guide. Once completed, jump into steps one and two below to acquire gas for deploying your first contract.

1. Acquire Tokens for Gas Fees

In Web3,gas fees refers to transaction fees that are always paid in the native currency of the blockchain network. For example, to deploy a contract on a contract on the Ethereum network, you must pay the transaction fee in Ether (ETH). Similarly, on the Polygon network, the fee would be in MATIC. To ensure smooth transactions, it is important to have a sufficient amount of the native token in your wallet.

To gather tokens for gas, we recommend using thePolygon Faucet . When you visit the faucet's website, you will be prompted to provide your wallet's address. To acquire it, make a request toGET /wallets as shown below noting down Wallet One's address wallets[0].address .

Node.js cURL // Import and configure the developer-controlled wallet SDK const { initiateDeveloperControlledWalletsClient } = require('@circle-fin/developer-controlled-wallets'); const circleDeveloperSdk = initiateDeveloperControlledWalletsClient({ apiKey: '', entitySecret: '' });

// Import and configure the smart contract SDK const { initiateSmartContractPlatformClient } = require('@circle-fin/smart-contract-platform'); const circleContractSdk = initiateSmartContractPlatformClient({ apiKey: '', entitySecret: '' });

const response = await circleDeveloperSdk.listWallets({}); curl --request GET \ --url 'https://api.circle.com/v1/w3s/wallets' \ --header 'accept: application/json' \ --header 'authorization: Bearer ' Response Body { "data": { "wallets": [{ "id": "17c411ba-7aef-4c5d-a4f8-951a60ce7fb1", "state": "LIVE", "walletSetId": "018b42d3-8934-7ad5-8ec7-a34179a7e6e5", "custodyType": "DEVELOPER", "address": "0x09e768def76316d8f127efb73047a046598fef7b", "blockchain": "MATIC-MUMBAI", "accountType": "EOA", "updateDate": "2023-10-18T12:48:45Z", "createDate": "2023-10-18T12:48:45Z" }, { "id": "8172d817-32b5-4f52-b071-4a8d789e08fc", "state": "LIVE", "walletSetId": "018b42d3-8934-7ad5-8ec7-a34179a7e6e5", "custodyType": "DEVELOPER", "address": "0x48f2d63f13f62ff1e3ab9dbda56f3e5c531557b2", "blockchain": "MATIC-MUMBAI", "accountType": "EOA", "updateDate": "2023-10-18T12:48:45Z", "createDate": "2023-10-18T12:48:45Z" }] } }

2. Check the Wallet's Balance

Before deploying the contract, double-check the MATIC tokens were received. This can be done by making a request toGET /wallets/{id}/balances providing the ID of Wallet One from the previous step.

Node.js cURL const response = await circleDeveloperSdk.getWalletTokenBalance({ id: '17c411ba-7aef-4c5d-a4f8-951a60ce7fb1' }); curl --request GET \ --url 'https://api.circle.com/v1/w3s/wallets/{id}/balances' \ --header 'accept: application/json' \ --header 'authorization: Bearer ' Response Body { "data": { "tokenBalances": [{ "token": { "id": "e4f549f9-a910-59b1-b5cd-8f972871f5db", "blockchain": "MATIC-MUMBAI", "name": "Polygon-Mumbai", "symbol": "MATIC-MUMBAI", "decimals": 18, "isNative": true, "updateDate": "2023-06-29T02:37:14Z", "createDate": "2023-06-29T02:37:14Z" }, "amount": "0.2", "updateDate": "2023-08-03T22:22:07Z" }] } }

Part Two: Write and Deploy Smart Contract

3. Writing the Smart Contract

In the OpenZeppelinContract Wizard

1. Select the ERC721 template and the additional features you want to add to your contract.
2. SelectMintable
3. . By default, this selects theOwnable
4. access control, restricting the Mintable function to the wallet owner.
5. SelectOpen in Remix
6. button to create a new workspace inRemix
7. and compile the contract.
8. SelectCompile contract
9. to compile the Solidity code into bytecode for the Ethereum Virtual Machine (EVM). We recommend using the default compiler settings which are appropriate for most use cases.1. In Remix, you don’t need to update the code at all.
10. 1. If you don't see compile contract, ensure you are in theSolidity Compiler
11. 1. section. If not, select it in the left-hand navigation bar.
12. Once the compilation is complete, note down the ABI and bytecode. These values will be used in the next step.1. The compiler output is available underCompilation Details
13. 1. . For more information on the Solidity compiler’s outputs, seeusing the compiler
14. 1. .
15. 1. The Application Binary Interface (ABI) is the standard way to interact with contracts on an EVM from outside the blockchain and for contract-to-contract interaction.

4. Deploy the Smart Contract

To deploy a smart contract, we will use thePOST /contracts/deploy API. In the request body, you will provide the following values gathered from the prior steps:

1. entitySecretCiphertext
2. uniquely generatedEntity Secret Ciphertext
3. . To learn more about generating the ciphertext for this request, see the guideHow to Re-Encrypt the Entity Secret
4. .
5. walletId
6. acquired from step 1.
7. abiJSON
8. acquired from the Solidity Compiler in step 3.
9. bytecode
10. acquired from the Solidity Compiler in step 3.
11. constructorSignature
12. the signature of the constructor function. For this contract, it is
13. constructor(address initialOwner)
14. .
15. constructorParameters
16. the address to set as the initial owner of the contract. For this guide, it should be the address of your wallet, but it can be any address.

- abiJSON
- must be stringified i.e. quote escaped. To stringfy the JSON you can use theJsonFormatter's Stringfy
- tool.
- bytecode
- must be prefixed with
- 0x
- . Node.js cURL const response = await circleContractSdk.deployContract({ name: 'First Contract Name', description: 'First Contract Description', walletId: '046b6c7f-0b8a-43b9-b35d-6489e6daee91', blockchain: 'MATIC-MUMBAI', fee: {

type: 'level', config: { feeLevel: 'MEDIUM' } }, constructorSignature: 'constructor(address initialOwner)', constructorParameters: ['0x6F77e56DFA40190349E1078e97ac3Eb479E0Dac6'], entitySecretCiphertext: '0NtD3d3+nmgb4GqYQXzAjKF8h5Zq6sHM2k/...', abiJSON: '[\n\t{\n\t\t"inputs": [],\n\t\t"stateMutability": "nonpayable",\n\t\t"type": "constructor"\n\t},\n\t{\n\t\t"anonymous": false,', bytecode: '0x...' }); curl --request POST \ --url 'https://api.circle.com/v1/w3s/contracts/deploy' \ --header 'accept: application/json' \ --header 'content-type: application/json' \ --header 'authorization: Bearer ' \ --data ' { "idempotencyKey": "046b6c7f-0b8a-43b9-b35d-6489e6daee91", "name": "First Contract Name", "description": "First Contract Description", "walletId": "046b6c7f-0b8a-43b9-b35d-6489e6daee91", "blockchain": "MATIC-MUMBAI", "feeLevel": "MEDIUM", "constructorSignature": "constructor(address initialOwner)", "constructorParameters": ["0x6F77e56DFA40190349E1078e97ac3Eb479E0Dac6"], "entitySecretCiphertext": "0NtD3d3+nmgb4GqYQXzAjKF8h5Zq6sHM2k/...", "abiJSON": "[\n\t{\n\t\t\"inputs\": [],\n\t\t\"stateMutability\": \"nonpayable\",\n\t\t\"type\": \"constructor\"\n\t},\n\t{\n\t\t\"anonymous\": false,", "bytecode": "0x..." } ' If the call is successful, you will receive a transactionId and a contractId in the response.

Response Body { "data": { "contractId": "0189db12-4089-72eb-b4a8-2aee38cc38fe", "transactionId": "7b989c65-9678-56d8-a998-d295b8b04535" } } You can now call GET /contracts/{id} to check the status of your transaction. When the transaction has been confirmed on-chain, the contract response object will contain the contractAddress and have a status of PENDING . After 12 block confirmations, the contract will have a status of COMPLETE - as seen below.

Node.js cURL const response = await circleContractSdk.getContract({ id: '0189db84-72b7-7fcc-832b-5bf886b9a0ef' }); curl --request GET \ --url 'https://api.circle.com/v1/w3s/contracts/{id}' \ --header 'accept: application/json' \ --header 'authorization: Bearer ' JSON { "data": { "contract": { "id": "0189db84-72b7-7fcc-832b-5bf886b9a0ef", "deploymentTransactionId": "4f5bfa38-c598-56a6-932e-8b5bbd3d5fc9", "name": "First Contract Name", "description": "First Contract Description", "contractInputType": "BYTECODE", "createDate": "2023-08-09T18:17:17Z", "updateDate": "2023-08-09T18:17:17Z", "archived": false, "contractAddress": "0x1e124d7384cd34448ea5907bd0052a79355ab5eb", "blockchain": "MATIC-MUMBAI", "status": "COMPLETE", "deployerAddress": "0x1bf9ad0cc2ad298c69a2995aa806ee832788218c", "txHash": "0x241c4df6f08f9ed2b569c9f9b1cc48fb6074ffffaeee7552e716ce059161a743", "abiJSON": "[\n\t{\n\t\t\"inputs\": [],\n\t\t\"stateMutability\": \"nonpayable\",\n\t\t\"type\": \"constructor\"\n\t},\n\t{\n\t\t\"anonymous\": false,", "functions": [{ "name": "approve", "type": "function", "inputs": [{ "name": "to", "type": "address" }, { "name": "tokenId", "type": "uint256" }], "stateMutability": "nonpayable" }], "verificationStatus": "UNVERIFIED" } } } If you have configuredWebhooks , you will receive notifications on the status of the deployment transaction. Additionally, you can view your deployed contract on the Polygon Mumbai block explorerPolygonscan by running a search using contractAddress . Updated2 months ago

What’s Next Now that you have deployed a smart contract, you can interact with it and create your first NFT! *Interact with a Smart Contract * Table of Contents * * Part One: Fund Your Wallet with Gas* * * 1. Acquire Tokens for Gas Fees * * * 2. Check the Wallet's Balance * * Part Two: Write and Deploy Smart Contract * * * 3. Writing the Smart Contract * * * 4. Deploy the Smart Contract