Deploy a Smart Contract From a Template

The following guide deploys a NFT Smart Contract from a template on the Polygon Mumbai TestnetSuggest Edits

Building smart contracts is made effortless with templates. Circle's Smart Contract Templates provide a quick and simple way to integrate smart contracts into your application. You can deploy contracts in just a few minutes without writing Solidity code. Our carefully curated templates, backed by Circle and audited by third parties, cater to popular on-chain use cases. Circle's templates can be deployed conveniently through either the console or APIs.

This guide walks through deploying an NFT on Polygon Mumbai Testnet using the NFT smart contract template and uploading an image to NFT.storage to mint as an NFT. This guide will provide you with a Web3 Services Console and API paths. Generally speaking, the Console is a great first option for learning to use SCP and breaking down all the functions a contract provides.

Many of the steps used here are transferable to other templates like Token and Multi-Tokens.

Evaluate Templates

To learn more about the NFT template or any other template, go to theWeb3 Services Console or theTemplates Glossary guide.

- Web3 Services Console:
- Viewing templates from the console provides a simple way to understand the contracts offered, what they are, the primary use cases each supports, the contracts' ABI functions, events, and code.
- Template Glossary:
- Outlines the templates offered, a summary of each one, and the template parameters provided when deploying via API.

Select Your Preferred Approach

1. Console Path:
2. Use the web3 services console and a console wallet to deploy a smart contract template and mint an NFT. This UI option is the preferred method for those new to using Smart Contracts. It is faster and has much fewer prerequisite steps.
3. API Path:
4. Use APIs to deploy a smart contract template and mint an NFT. This option is more technical and requires creating an API key and a Developer-Controlled wallet.

Console UI Path

Prerequisites

1. Create aDeveloper Account
2. .
3. Create anNFT.Storage
4. account. NFT.storage is a free service for uploading files to IPFS and requires an email or GitHub account.

1. Create a Console Wallet

To deploy a smart contract, you must provide a wallet to deploy from. If you do not already have a console wallet, follow the steps below to create one.

Console Wallets

Console Wallets are specifically designed to be used within the Console UI and cannot be utilized with the APIs. They are classified as Smart Contract Account (SCA) wallets within Web3 Services. Notably, W3S SCA Wallets leverage Circle's Gas Station product, eliminating the need to maintain the gas in the wallet to cover transaction fees. To learn more about the Gas Station feature, please refer to theGas Station overview. In theconsole :

1. Navigate to theConsole Wallets
2. tab.
3. SelectCreate Wallet

4. in the top right corner.
5. Provide a wallet name. In this case, you can use “Console Wallet 1”
6. SelectCreate
7. .

You should now see two wallets, one for Polygon and Ethereum, on the console wallets dashboard. Copy the address of the Polygon wallet. This will be used in the subsequent step.

Console Wallet Deploy Cost

Unlike Externally Owned Accounts (EOAs), SCAs cost gas to “deploy”. Note: With our process of lazy deployment, you won't have to pay the gas fee at the time of wallet creation. Instead, the fee will be charged when you initiate your first outbound transaction.

2. Deploy the Smart Contract

Now that you have a wallet let’s deploy the smart contract from a template.

In theconsole :

1. Navigate to theTemplates
2. tab via the left navigation.
3. SelectNFT
4. ERC-721.
5. Fill in the contract template deployment parameters:
6.
    1. Name
7.
    1. : The off-chain name of the contract. This will only be represented on Circle’s systems, such as the APIs and Web3 Services Console. In this case, you can use “My First NFT Contract” or any other name you prefer.
8.
    1. Contract Name:
9.
    1. The on-chain name for the contract. In this case, you can use “My First NFT Contract” or any other name you prefer.
10.
    1. Default Admin:
11.
    1. The address of the default admin. This address can execute permissions functions on the contract. This should be an address you control or the address of a user if deploying on their behalf. In this case, use the console wallet’s address from step one.
12.
    1.
        1. NOTE: The following roles will be given to the defaultAdmin:
13.
    1.
        1. DEFAULT_ADMIN_ROLE
14.
    1.
        1. ,
15.
    1.
        1. MINTER_ROLE
16.
    1.
        1. ,
17.
    1.
        1. TRANSFER_ROLE
18.
    1.
        1. . The
19.
    1.

1. defaultAdmin
20.
    1.
        1. will also be set as the contract's owner. All of these roles can be updated by the defaultAdmin later.
21.
    1. Primary Sale Recipient:
22.
    1. The recipient's address for first-time sales. All first-time sale proceeds will go to this address. In this case, use the console wallet’s address from step one.
23.
    1. Royalty Recipient:
24.
    1. The recipient address for all royalties. Any proceeds from secondary sales will go to this address. In this case, use the console wallet’s address from step one.
25.
    1. Royalty Percent:
26.
    1. The percentage of secondary sales that go to the royalty recipient. There should be a value between 0 and 100. In this case, use “0”.
27.
    1. Network
28.
    1. : The network to deploy onto. In this case, select “Polygon Mumbai”.
29.
    1. Select Wallet:
30.
    1. The wallet to deploy the smart contract from. In this case, select “Console Wallet 1”
31.
    1. Deployment Speed
32.
    1. : A fee level setting (FAST, AVERAGE, SLOW). This affects how quickly the network will process your deployment transaction. In this case, you will use “AVERAGE”.
33. SelectDeploy
34. .

Once deployed, you will be placed back on theContracts dashboard. Here, you can check the deployment status of the contract. When you land on the page, expect the deployment status to bePending . Give it several seconds, and then refresh the page. At this point, the deployment status should be set toComplete .

3. Upload an NFT to IPFS

You’ll be usingNFT.Storage to upload your NFT’s metadata to IPFS. NFT.Storage is a free, decentralized storage and bandwidth provider for IPFS and Filecoin.Filecoin provides long-term storage for the data, ensuring that even if NFT.Storage is attacked or taken down, the NFT data will still be available.

NFT data stored by NFT.Storage can be accessed from the decentralized IPFS network from any peer with the content. This means you can use the CID of your data to find it through any IPFS provider. 1. Login toNFT.Storage 2. account and navigate toFiles 3. . 4. Using theUpload File 5. flow, upload your data, and you will receive the CID of your NFT metadata. You can make a full IPFS URL (ipfs://).

1. Grab the IPFS URL of your newly uploaded data. You should use IPFS URLs when minting your NFT in the next steps.

4. Mint an NFT

Now that your NFT has been successfully uploaded let's dive into the Console UI to explore the full range of capabilities it offers and mint an NFT. With the Console, you can seamlessly explore the NFT contract you deployed. Gain insights into its ABI functions, inspect their inputs, and easily execute functions. In this example, we will directly execute the mintTo function from the console - which mints an NFT to a user without a sale.

In theconsole :

1. Navigate to theContracts
2. tab.
3. Select theMy First NFT Contract
4. within the console.

5. SelectABI Functions
6. sub tab. This should be selected by default.
7. SelectWrite
8. .
9. SelectmintTo
10. .
11. On themintTo
12. ABI function, provide the following parameters:
13.
    1. _to:
14.
    1. The wallet address to which the NFT will be minted. In this case, use the wallet address of the wallet you deployed the contract from. Ultimately, this can be any wallet on-chain.
15.
    1. _uri:
16.
    1. A distinct Uniform Resource Identifier (URI) for a given asset. In this case, provide the IPFS URL copied from NFT.Storage.
17. SelectExecute Function
18. and ensureConsole Wallet 1
19. is selected.
20. SelectExecute.

Once executed, selectView Transaction History to access the Console Wallet's transaction dashboard. Here, you can easily monitor the progress and status of the transaction. Once the transaction state is marked asComplete , the NFT has been successfully transferred on-chain to the designated address mentioned above.

In the transaction history view, you will also notice an inbound transfer indicating that the NFT has been minted to the Console Wallet. Congratulations, you have deployed a smart contract and minted an NFT!

Next Steps

1. Learn how to mint NFTs via theAPI
2. .
3. Deploy a Token or Multi-Token contract.

API Path

Prerequisites

1. Create a Developer Account, API key, and set up a Developer-Controlled Wallet. We recommend using ourinteractive quickstart guide
2. to complete these requirements if you haven't already.
3. Create anNFT.Storage
4. account. NFT.storage is a free service that requires an email or GitHub account.

1. Deploy the Smart Contract

To deploy a smart contract, you must provide a wallet to deploy from. If you don't have the walletId , you can use theGET /wallets API to acquire the wallet ID. Ensure the wallet custody type is Developer-Controlled and the blockchain is Polygon Mumbai. We recommend the account type be SCA - removing the need for gas in your wallet for deployment. Lastly, note down the wallet's address. This will be used in the subsequent steps.

Deployment Request Parameters:

1. idempotencyKey
2. : A unique value generated by the client that the server uses to recognize subsequent retries of the same request.
3. name
4. : The off-chain name of the contract. This will only be represented on Circle's System such as the APIs and Web3 Services Console. In this case, you can use
5. My First NFT Contract
6. or any other name you prefer.
7. walletId
8. : The ID of the wallet to deploy from.
9. templateId
10. : The ID of the chosen template. In this case, you will use

11. 76b83278-50e2-4006-8b63-5b1a2a814533
12. as the ID for the NFT template. This
13. templateId
14. and other template-specific details are listed in the Templates Glossary
15. .
16. blockchain
17. : The network to deploy onto. In this case, you will use
18. MATIC-MUMBAI
19. .
20. entitySecretCiphertext
21. : The encrypted ciphertext of your entity secret. It must be re-encrypted for every new request. Go to How to Re-Encrypt the Entity Secret
22. to learn more about generating an entity secret ciphertext.
23. feeLevel
24. : A fee level setting (LOW, MEDIUM, HIGH). This affects how quickly the network will process your deployment transaction. In this case, you will use
25. MEDIUM
26. .
27. templateParameters
28. : A JSON object representing the template deployment parameters and arguments. This object is explained in more detail below.

Template Parameters

Template parameters are the inputs for initializing and deploying the template on-chain. Each template will have its own set of parameters.

For the NFT template, the required properties are:

1. name:
2. The on-chain name for the contract. In this case, you can use
3. My First NFT Contract
4. or any other name you prefer.
5. defaultAdmin:
6. The address of the default admin. This address can execute permissioned functions on the contract. This should be an address you control or the address of a user if deploying on their behalf. In this case, use the developer-controlled wallet's address retrieved from the GET /wallets
7. API above.1. NOTE: The following roles will be given to the defaultAdmin:
8. 1. DEFAULT_ADMIN_ROLE
9. 1. ,
10. 1. MINTER_ROLE
11. 1. ,
12. 1. TRANSFER_ROLE
13. 1. . The
14. 1. defaultAdmin
15. 1. will also be set as the contract's owner. All of these roles can be updated by the defaultAdmin later.
16. primarySaleRecipient:
17. The recipient address for first-time sales. All first-time sale proceeds will go to this address. In this case, use the developer-controlled wallet's address retrieved from the GET /wallets
18. API above.
19. royaltyRecipient:
20. The recipient address for all royalties. Any proceeds from secondary sales will go to this address. In this case, use
21. 0
22. .
23. royaltyPercent:
24. The percentage of secondary sales that go to the royalty recipient. It should be a value between 0 and 100. In this case, use the developer-controlled wallet's address retrieved from the GET /wallets
25. API above.

There are additional properties and configurations available for more advanced use cases. Please see the Templates Glossary for more information on configuring the NFT Template and for extended use cases of all template types.

Now that you have all your request parameters in order, you can deploy the smart contract by making a request to the POST /templates/{id}/deploy API. You must also pass your entity secret ciphertext in the request body. The full request is as follows:

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 circleContractSdk.deployContractTemplate({ id: 'aea21da6-0aa2-4971-9a1a-5098842b1248', blockchain: 'MATIC-MUMBAI', name: 'My First NFT Contract', walletId: '', templateParameters: { name: 'My First NFT Contract', defaultAdmin: '

*', primarySaleRecipient: '
', royaltyRecipient: '
', royaltyPercent: 0 }, fee: { type: 'level', config: { feeLevel: 'MEDIUM' } } }); curl --request POST \ --url 'https://api.circle.com/v1/w3s/templates/aea21da6-0aa2-4971-9a1a-5098842b1248/deploy' \ --header 'accept: application/json' \ --header 'content-type: application/json' \ --header 'authorization: Bearer ' \ --data ' { "idempotencyKey": , "blockchain": "MATIC-MUMBAI", "name": "My First NFT Contract", "walletId": , "templateParameters": { "name": "My First NFT Contract", "defaultAdmin":
, "primarySaleRecipient":
, "royaltyRecipient":
, "royaltyPercent": 0 }, "feeLevel": "MEDIUM", "entitySecretCiphertext": " } ' JSON { "data": { "contractIds": ["b7c35372-ce69-4ccd-bfaa-504c14634f0d"], "transactionId": "601a0815-f749-41d8-b193-22cadd2a8977" } }*

A successful API response indicates that the deployment process has been initiated but does not guarantee that the deployment will be successful. The returned transactionId can be used to check the status of the deployment transaction usingget a transaction . To validate that your contract has deployed successfully, make a request toGET /transactions/{id} , providing the ID of the transaction received in the response above.

Node.js cURL const response = await circleDeveloperSdk.listTransactions({}); curl --request GET \ --url https://api.circle.com/v1/w3s/transactions/{id} \ --header 'accept: application/json' \ --header 'authorization: Bearer ' Response { "data": { "transaction": { "id": "601a0815-f749-41d8-b193-22cadd2a8977", "blockchain": "MATIC-MUMBAI", "state": "COMPLETE", ... [other fields truncated] } } } The returned contractId can be used to perform further management and interaction with the deployed smart contract, such as minting an NFT.

2. Upload an NFT to IPFS

You'll be usingNFT.Storage to upload your NFT's metadata to IPFS. NFT.Storage is a free, decentralized storage and bandwidth provider for IPFS and Filecoin.Filecoin provides long-term storage for the data, ensuring that even if NFT.Storage is attacked or taken down, the NFT data will still be available.

NFT data stored by NFT.Storage can be accessed from the decentralized IPFS network from any peer with the content. This means you can use the CID of your data to find it through any IPFS provider. 1. Login toNFT.Storage 2. account and navigate toFiles 3. . 4. Using theUpload File 5. flow, upload your data, and you will receive the CID of your NFT metadata. You can make a full IPFS URL (ipfs://).

1. *Grab the IPFS URL of your newly uploaded data. You should use IPFS URLs when minting your NFT in the next steps.*

3. Mint an NFT

Now that your NFT has been successfully uploaded let's dive into the Web3 Services Console to explore its full range of capabilities. With the Console, you can seamlessly explore the NFT contract you deployed. Gain insights into its ABI functions, inspect their inputs, and easily execute functions through an API call by conveniently copying the required code.

Alternatively, you can retrieve detailed information about the contract using the Get contract by ID response, which includes all the available functions and events.

In this example, we will use the code snippet generator to build the contract execution request for the mintTo function - which mints an NFT to a user without a sale. This can be done by selecting the contract within the Console, then in the ABI functions tab, selecting write, and selecting mintTo.

As you fill in the parameters for the mintTo function, the code to make a contract execution API request and mint the NFT is automatically generated. Make sure to select your preferred wallet type and code generated, in this case, select a Developer Controlled wallet and Shell to create a cURL request example.

mintTo function parameters:

1. *_to:*
2. *The wallet address that the NFT will be minted to. In this case, use the wallet address of the wallet you deployed the contract from. Ultimately, this can be any wallet on-chain.*
3. *_uri:*
4. *A distinct Uniform Resource Identifier (URI) for a given asset. In this case, provide the IPFS URL copied from NFT.Storage.*

As a result of plugging these parameters in, you will be provided with a code block similar to the one below. Once copied to your preferred API client, add your walletId and entitySecretCiphertext to the API request.

Node.js cURL const response = await circleDeveloperSdk.createContractExecutionTransaction({ walletId: '', abiFunctionSignature: 'mintTo(address,string)', abiParameters: ['0x6E5eAf34c73D1CD0be4e24f923b97CF38e10d1f3', 'ipfs://bafkreibdi6623n3xpf7ymk62ckb4bo75o3qemwkpfvp5i25j66itxvsoei'], contractAddress: '0xf7761519cf2d84c31f634e249790b4757c10a103', fee: { type: 'level', config: { feeLevel: 'MEDIUM' } } }); curl --request POST \ --url https://api.circle.com/v1/w3s/developer/transactions/contractExecution \ --header 'authorization: Bearer ' \ --header 'accept: application/json' \ --header 'content-type: application/json' \ --data '{ "abiFunctionSignature": "mintTo(address,string)", "abiParameters": ["0x6E5eAf34c73D1CD0be4e24f923b97CF38e10d1f3", "ipfs://bafkreibdi6623n3xpf7ymk62ckb4bo75o3qemwkpfvp5i25j66itxvsoei"], "idempotencyKey": "a8350bbe-98b1-4204-aadd-385865530c79", "contractAddress": "0xf7761519cf2d84c31f634e249790b4757c10a103", "feeLevel": "MEDIUM", "walletId": "", "entitySecretCiphertext": "" }' Note that the address of the walletId must have the MINTER_ROLE on the contract to mint an NFT. Make sure you use the appropriate walletId .

JSON { "data": { "id": "601a0815-f749-41d8-b193-22cadd2a8977", "state": "INITIATED" } }

A successful API response indicates that the mintTo transaction has been initiated but does not guarantee that the deployment will be successful. To validate that your contract has deployed successfully, make a request to GET /transactions/{id} providing the ID of the transaction received in the response above.

Node.js cURL const response = await circleDeveloperSdk.getTransaction({ id: '601a0815-f749-41d8-b193-22cadd2a8977' }); curl --request GET \ --url https://api.circle.com/v1/w3s/transactions/{id} \ --header 'accept: application/json' \ --header 'authorization: Bearer ' Response { "data": { "transaction": { "id": "601a0815-f749-41d8-b193-22cadd2a8977", "blockchain": "MATIC-MUMBAI", "state": "COMPLETE", ... [other fields truncated] } } } Once the transaction is complete, the receiver - the _to address that you previously specified - will now be able to use and interact with the minted NFT.

Congratulations, you have deployed a smart contract and minted an NFT!

Next Steps

1. *Learn how to mint NFTs via the Console*
2. *.*
3. *Deploy a Token or Multi-Token contract. Updated16 days ago*
4. *Table of Contents*
5.
    - *Evaluate Templates*
6.
    - *Select Your Preferred Approach*
7.