

Getting Started

Learn how to make requests to the Chainlink Functions Decentralized Oracle Network (DON) and make any computation or API calls offchain. Chainlink Functions is available on several blockchains (see the [supported networks page](#)), but this guide uses Polygon Mumbai to simplify access to testnet funds. Complete the following tasks to get started with Chainlink Functions:

- Set up your web3 wallet and fund it with testnet tokens.
- Simulate a Chainlink Functions on the [Chainlink Functions Playground](#).
- Send a Chainlink Functions request to the DON. The JavaScript source code makes an API call to the [Star Wars API](#) and fetches the name of a given character.
- Receive the response from Chainlink Functions and parse the result.

Simulation

Before making a Chainlink Functions request from your smart contract, it is always a good practice to simulate the source code offchain to make any adjustments or corrections.

1. Open the [Functions playground](#).
2. Copy and paste the following source code into the playground's code block.

```
const characterId = args[0]; const apiResponse = await Functions.makeHttpRequest({ url: 'https://swapi.info/api/people/${characterId}/', }); if (apiResponse.error) { throw Error("Request failed"); } const { data } = apiResponse; return Functions.encodeString(data.name);
```

3. Under Argument, set the first argument to 1. You are going to fetch the name of the first Star Wars character. 4. Click on Run code. Under Output, you should see Luke Skywalker.

Configure your resources

Configure your wallet

You will test on Polygon Mumbai, so you must have an Ethereum web3 wallet with enough MATIC and LINK tokens. MATIC is the native gas fee token on Polygon. You will use MATIC tokens to pay for gas whenever you make a transaction on Polygon Mumbai. On the other hand, you will use LINK tokens to pay the Chainlink Functions Decentralized Oracles Network (DON) for processing your request.

1. [Install the MetaMask wallet](#) or other Ethereum web3 wallet.
2. Set the network for your wallet to the Polygon Mumbai testnet. If you need to add Mumbai to your wallet, you can find the chain ID and the LINK token contract address on the [LINK Token Contracts](#) page.
3. [Polygon Mumbai testnet and LINK token contract](#)
4. Request testnet MATIC from the [Polygon Faucet](#).
5. Request testnet LINK from [faucets.chain.link/mumbai](#).

Deploy a Functions consumer contract on Polygon Mumbai

1. Open the [GettingStartedFunctionsConsumer.sol](#) contract in Remix.

[Open in Remix](#) [What is Remix?](#) 2. Compile the contract. 3. Open MetaMask and select the Polygon Mumbai network. 4. In Remix under the Deploy & Run Transaction tab, select Injected Provider - MetaMask in the Environment list. Remix will use the MetaMask wallet to communicate with Polygon Mumbai. 5. Click the Deploy button to deploy the contract. MetaMask prompts you to confirm the transaction. Check the transaction details to make sure you are deploying the contract to Polygon Mumbai. 6. After you confirm the transaction, the contract address appears in the Deployed Contracts list. Copy the contract address and save it for later. You will use this address with a Functions Subscription.

Create a subscription

You use a Chainlink Functions subscription to pay for, manage, and track Functions requests.

1. Go to [functions.chain.link](#).
2. Click Connect wallet:
3. Read and accept the Chainlink Foundation Terms of Service. Then click MetaMask.
4. Make sure your wallet is connected to the Polygon Mumbai testnet. If not, click the network name in the top right corner of the page and select Polygon Mumbai.
5. Click Create Subscription:
6. Provide an email address and an optional subscription name:
7. The first time you interact with the Subscription Manager using your EOA, you must accept the Terms of Service (ToS). A MetaMask popup appears and you are asked to accept the ToS:
8. After you approve the ToS, another MetaMask popup appears, and you are asked to approve the subscription creation:
9. After the subscription is created, MetaMask prompts you to sign a message that links the subscription name and email address to your subscription:

Fund your subscription

1. After the subscription is created, the Functions UI prompts you to fund your subscription. Click Add funds:
2. For this example, add 2 LINK and click Add funds:

Add a consumer to your subscription

1. After you fund your subscription, add your consumer to it. Specify the address for the consumer contract that you deployed earlier and click Add consumer. MetaMask prompts you to confirm the transaction.
2. Subscription creation and configuration is complete. You can always see the details of your subscription again at [functions.chain.link](#):

Run the example

The example is hardcoded to communicate with Chainlink Functions on Polygon Mumbai. After this example is run, you can examine the code and see a detailed description of all components.

1. In Remix under the Deploy & Run Transaction tab, expand your contract in the Deployed Contracts section.
2. Expand the sendRequest function to display its parameters.
3. Fill in the subscriptionId with your subscription ID and args with [1]. You can find your subscription ID on the Chainlink Functions Subscription Manager [functions.chain.link](#). The [1] value for arg specifies which argument in the response will be retrieved.
4. Click the transaction button.
5. Wait for the request to be fulfilled. You can monitor the status of your request on the Chainlink Functions Subscription Manager.
6. Refresh the Functions UI to get the latest request status.
7. After the status is Success, check the character name. In Remix, under the Deploy & Run Transaction tab, click the character function. If the transaction and request ran correctly, you will see the name of your character in the response.

Chainlink Functions is capable of much more than just retrieving data. Try one of the [tutorials](#) to see examples that can GET and POST to public APIs, securely handle API secrets, handle custom responses, and query multiple APIs.

Examine the code

Solidity code

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.19;
import {FunctionsClient} from "@chainlink/contracts/src/v0.8/functions/v1_0_0/FunctionsClient.sol";
import {ConfirmedOwner} from "@chainlink/contracts/src/v0.8/shared/access/ConfirmedOwner.sol";

* Request testnet LINK and ETH here: https://faucets.chain.link/ * Find information on LINK Token Contracts and get the latest ETH and LINK faucets here: https://docs.chain.link/resources/link-token-contracts/ * // * @title GettingStartedFunctionsConsumer * @notice This is an example contract to show how to make HTTP requests using Chainlink * @dev This contract uses hardcoded values and should not be used in production.

contract GettingStartedFunctionsConsumer is FunctionsClient, ConfirmedOwner {
    using FunctionsRequest for FunctionsRequest;
    Request; // State variables to store the last request ID, response, and error
    bytes32 public _lastRequestId;
    bytes public _lastResponse;
    bytes public _lastError; // Custom error type errorUnexpectedRequestID(bytes32 requestId); // Event to log response
    event Response(bytes32 indexed requestId, string character, bytes response, bytes err); // Router address - Hardcoded for Mumbai // Check to get the router address for your supported network
    https://docs.chain.link/chainlink-functions/supported-networks address router = 0x6E2dc0F9DB014aE19888F539E59285D2Ea04244C; // JavaScript source code // Fetch character name from the Star Wars API // Documentation: https://swapi.info/people?string=source="const characterId = args[0];"const apiResponse = await Functions.makeHttpRequest({url: https://swapi.info/api/people/${characterId}/});"if (apiResponse.error) {throw Error("Request failed");}"const { data } = apiResponse;"return Functions.encodeString(data.name);"// Callback gas limit
    uint32 gasLimit = 300000; // donID - Hardcoded for Mumbai // Check to get the donID for your supported network https://docs.chain.link/chainlink-functions/supported-
```

- FetchcharacterIdfromargs. Args is an array. ThecharacterIdis located in the first element.
- Make the HTTP call usingFunctions.makeHttpRequestand store the response inapiResponse.
- Throw an error if the call is not successful.
- The API response is located atdata.
- Read the name from the API responsedata.nameand return the result as [a buffer](#) using theFunctions.encodeStringhelper function. Because the nameis a string, we useencodeString. For other data types, you can use different[data encoding functions](#). Note: Read this[article](#) if you are new to Javascript Buffers and want to understand why they are important.