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 thesupported 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 theChainlink Functions Playground .
- Send a Chainlink Functions request to the DON. The JavaScript source code makes an API call to theStar 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 theFunctions playground .
2. Copy and paste the following source code into the playground's code block.

constcharacterId=args[0];constapiResponse=awaitFunctions.makeHttpRequest({url:https://swapi.info/api/people/${characterId}/,});if(apiResponse.error){throwError("Request failed");}const{data}=apiResponse;returnFunctions.encodeString(data.name); 3. UnderArgument, set the first argument to1. You are going to fetch the name of the first Star Wars character. 4. Click onRun code. UnderOutput, you should seeLuke 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 theLINK Token Contracts page.
3. Polygon Mumbai testnet and LINK token contract
4. Request testnet MATIC from thePolygon Faucet .
5. Request testnet LINK fromfaucets.chain.link/mumbai .

Deploy a Functions consumer contract onPolygon Mumbai

1. Open theGettingStartedFunctionsConsumer.sol contract in Remix.

Open in Remix What is Remix? 2. Compile the contract. 3. Open MetaMask and select thePolygon Mumbainetwork. 4. In Remix under theDeploy & Run Transactionstab, selectInjected Provider - MetaMaskin theEnvironmentlist. Remix will use the MetaMask wallet to communicate withPolygon Mumbai. 5. Click theDeploybutton to deploy the contract. MetaMask prompts you to confirm the transaction. Check the transaction details to make sure you are deploying the contract toPolygon Mumbai. 6. After you confirm the transaction, the contract address appears in theDeployed Contractslist. 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 tofunctions.chain.link .
2. ClickConnect wallet:
3. Read and accept the Chainlink Foundation Terms of Service. Then clickMetaMask.
4. Make sure your wallet is connected to thePolygon Mumbaitestnet. If not, click the network name in the top right corner of the page and selectPolygon Mumbai.
5. ClickCreate 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. ClickAdd funds:
2. For this example, add 2 LINK and clickAdd 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 clickAdd 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 atfunctions.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 theDeploy & Run Transactionstab, expand your contract in theDeployed Contractssection.
2. Expand thesendRequestfunction to display its parameters.
3. Fill in thesubscriptionIdwith your subscription ID andargswith[1]. You can find your subscription ID on the Chainlink Functions Subscription Manager atfunctions.chain.link . The[1]value forargsspecifies which argument in the response will be retrieved.
4. Click thetransactbutton.
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 isSuccess, check the character name. In Remix, under theDeploy & Run Transactionstab, click thecharacterfunction. 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 theTutorials 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: MITpragmasolidity0.8.19;import{FunctionsClient}from"@chainlink/contracts/src/v0.8/functions/dev/v1_0_0/FunctionsClient.sol";import{ConfirmedOwner}from"@chainlink/contracts/src/v0.8/shared/access/
*** 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.
*/contractGettingStartedFunctionsConsumerisFunctionsClient,ConfirmedOwner{usingFunctionsRequestforFunctionsRequest.Request;// State variables to store the last request ID, response, and errorbytes32publics_lastRequestId;bytespublics_lastResponse;bytespublics_lastError;// Custom error typeerrorUnexpectedRequestID(bytes32requestId);// Event to log responseseventResponse(bytes32indexedrequestId,stringcharacter,bytesresponse,byteserr);// Router address - Hardcoded for Mumbai// Check to get the router address for your supported network https://docs.chain.link/chainlink-functions/supported-networksaddressrouter=0x6E2dc0F9DB014aE19888F539E59285D2Ea04244C;// JavaScript source code// Fetch character name from the Star Wars API.// Documentation: https://swapi.info/peoplestringsource="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 limituint32gasLimit=300000;// donID - Hardcoded for Mumbai// Check to get the donID for your supported network https://docs.chain.link/chainlink-functions/supported-

*networksbytes32donID=0x66756e2d706f6c79676f6e2d6d756d6261692d3100000000000000000000000;// State variable to store the returned character informationstringpubliccharacter;/ * @notice Initializes the contract with the Chainlink router address and sets the contract owner* /constructor()FunctionsClient(router)ConfirmedOwner(msg.sender){}/ * **@notice Sends an HTTP request for character information * @param subscriptionId The ID for the Chainlink subscription * @param args The arguments to pass to the HTTP request * @return requestId The ID of the request */functionsendRequest(uint64subscriptionId,string[]calldataargs)externalonlyOwnerreturns(bytes32requestId) {FunctionsRequest.Requestmemoryreq;req.initializeRequestForInlineJavaScript(source);// Initialize the request with JS codeif(args.length>0)req.setArgs(args);// Set the arguments for the request// Send the request and store the request IDs_lastRequestId=_sendRequest(req.encodeCBOR(),subscriptionId,gasLimit,donID);returns_lastRequestId;}**/ * @notice Callback function for fulfilling a request * @param requestId The ID of the request to fulfill * @param response The HTTP response data * @param err Any errors from the Functions request */functionfulfillRequest(bytes32requestId,bytesmemoryresponse,bytesmemoryerr)internaloverride{if(s_lastRequestId!=requestId){revertUnexpectedRequestID(requestId);// Check if request IDs match}// Update the contract's state variables with the response and any errorss_lastResponse=response;character=string(response);s_lastError=err;// Emit an event to log the responseemitResponse(requestId,character,s_lastResponse,s_lastError);}} Open in Remix What is Remix? * To write a Chainlink Functions consumer contract, your contract must importFunctionsClient.sol andFunctionsRequest.sol . You can read the API referencesFunctionsClient andFunctionsRequest .

These contracts are available in an NPM package so that you can import them from within your project.

import {FunctionsClient} from "@chainlink/contracts/src/v0.8/functions/dev/v1_0_0/FunctionsClient.sol"; import {FunctionsRequest} from "@chainlink/contracts/src/v0.8/functions/dev/v1_0_0/libraries/FunctionsRequest.sol"; * Use the FunctionsRequest.sol library to get all the functions needed for building a Chainlink Functions request.

using FunctionsRequest for FunctionsRequest.Request; * The latest request ID, latest received response, and latest received error (if any) are defined as state variables:

bytes32 public s_lastRequestId; bytes public s_lastResponse; bytes public s_lastError; * We define theResponseevent that your smart contract will emit during the callback

event Response(bytes32 indexed requestId, string character, bytes response, bytes err); * The Chainlink Functions router address and donID are hardcoded for Polygon Mumbai. Check thesupported networks page to try the code sample on another testnet. * ThegasLimitis hardcoded to300000, the amount of gas that Chainlink Functions will use to fulfill your request. * The JavaScript source code is hardcoded in thesourcestate variable. For more explanation, read theJavaScript code section . * Pass the router address for your network when you deploy the contract:

constructor() FunctionsClient(router) * The two remaining functions are:

- sendRequestfor sending a request. It receives the subscription ID and list of arguments to pass to the source code. Then:
- It uses theFunctionsRequestlibrary to initialize the request and add the source code and arguments. You can read the API Reference forInitializing a request andadding arguments .

FunctionsRequest.Request memory req; req.initializeRequestForInlineJavaScript(source); if (args.length > 0) req.setArgs(args); * It sends the request to the router by calling theFunctionsClientsendRequestfunction. You can read the API reference forsending a request . Finally, it stores the request id ins_lastRequestIdand returns it.

s_lastRequestId = _sendRequest(req.encodeCBOR(), subscriptionId, gasLimit, jobId); return s_lastRequestId;Note:_sendRequestaccepts requests encoded inbytes. Therefore, you must encode it usingencodeCBOR . * fulfillRequestto be invoked during the callback. This function is defined inFunctionsClientasvirtual(readfulfillRequestAPI reference). So, your smart contract must override the function to implement the callback. The implementation of the callback is straightforward: the contract stores the latest response and error ins_lastResponseands_lastError, parses theresponsefrombytestostringto fetch the character name before emitting theResponseevent.

s_lastResponse = response; character = string(response); s_lastError = err; emit Response(requestId, s_lastResponse, s_lastError);

JavaScript code

constcharacterId=args[0];constapiResponse=awaitFunctions.makeHttpRequest({url:https://swapi.info/api/people/${characterId}/,});if(apiResponse.error){throwError("Request failed");}const{data}=apiResponse;returnFunctions.encodeString(data.name); This JavaScript source code usesFunctions.makeHttpRequest to make HTTP requests. The source code calls thehttps://swapi.info/API to request a Star Wars character name. If you read theFunctions.makeHttpRequest documentation and theStar Wars API documentation , you notice that URL has the following format where$characterIdis provided as parameter when making the HTTP request:

url: https://swapi.info/api/people/${characterId}/ To check the expected API response for the first character, you can directly paste the following URL in your browserhttps://swapi.info/api/people/1/or run thecurlcommand in your terminal:

curl-X'GET'\'https://swapi.info/api/people/1/'-H'accept: application/json' The response should be similar to the following example:

{"name":"Luke Skywalker","height":"172","mass":"77","hair_color":"blond","skin_color":"fair","eye_color":"blue","birth_year":"19BBY","gender":"male","homeworld":"https://swapi.info/api/planets/1/","films":["https://swapi.info/api/films/1/","https://swapi.info/api/films/2/","https://swapi.info/api/films/3/","https://swapi.info/api/films/6/"],"species":[],"vehicles":["https://swapi.info/api/vehicles/14/","https://swapi.info/api/vehicles/30/"],"starships":["https://swapi.info/api/starships/12/","https://swapi.info/api/starships/22/"],"created":"2014-12-09T13:50:51.644000Z","edited":"2014-12-20T21:17:56.891000Z","url":"https://swapi.info/api/people/1/"} Now that you understand the structure of the API. Let's delve into the JavaScript code. The main steps are:

- 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 abuffer using theFunctions.encodeStringhelper function. Because thenameis astring, we useencodeString. For other data types, you can use differentdata encoding functions .Note: Read thisarticle if you are new to Javascript Buffers and want to understand why they are important.