Automate your Functions (Custom Logic Automation)

This tutorial shows you how to useChainlink Automation to automate your Chainlink Functions. Automation is essential when you want to trigger the same function regularly, such as fetching weather data daily or fetching an asset price on every block.

Read theAutomate your Functions (Time-based Automation) tutorial before you follow the steps in this example. This tutorial explains how to trigger your functions using aAutomation compatible contract .

After you deploy and set up your contract, Chainlink Automation triggers your function on every block.

caution

Chainlink Functions is still in BETA. The use of secrets in your requests is an experimental feature that may not operate as expected and is subject to change. Use of this feature is at your own risk and may result in unexpected errors, possible revealing of the secret as new versions are released, or other issues.

note

Chainlink Functions is a self-service solution. You must ensure that the data sources or APIs specified in requests are of sufficient quality and have the proper availability for your use case. You are responsible for complying with the licensing agreements for all data providers that you connect with through Chainlink Functions. Violations of data provider licensing agreements or theterms can result in suspension or termination of your Chainlink Functions account.

Prerequisites

note

You might skip these prerequisites if you have followed one of theseguides . You can check your subscription details (including the balance in LINK) in theChainlink Functions Subscription Manager . If your subscription runs out of LINK, follow theFund a Subscription guide.

Set up your environment

You must provide the private key from a testnet wallet to run the examples in this documentation. Install a Web3 wallet, configureNode.js , clone thesmartcontractkit/smart-contract-examples repository, and configure a.env.encfile with the required environment variables.

Install and configure your Web3 wallet for Polygon Mumbai:

1. Install Deno so you can compile and simulate your Functions source code on your local machine.
2. Install the MetaMask wallet or other Ethereum Web3 wallet.
3. 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.
4. Polygon Mumbai testnet and LINK token contract
5. Request testnet MATIC from thePolygon Faucet .
6. Request testnet LINK fromfaucets.chain.link/mumbai .

Install the required frameworks and dependencies:

1. Install the latest release of Node.js 20. Optionally, you can use thenvm package to switch between Node.js versions withnvm use 20.

Note: To ensure you are running the correct version in a terminal, typenode -v.

node-v$node-vv20.9.0 2. In a terminal, clone thesmart-contract examples repository and change directories. This example repository imports theChainlink Functions Toolkit NPM package . You can import this package to your own projects to enable them to work with Chainlink Functions.

gitclone https://github.com/smartcontractkit/smart-contract-examples.git&&\cd./smart-contract-examples/functions-examples/ 3. Runnpm installto install the dependencies.

npminstall 4. For higher security, the examples repository encrypts your environment variables at rest.

1. Set an encryption password for your environment variables.

npx env-enc set-pw 2. Runnpx env-enc setto configure a.env.encfile with the basic variables that you need to send your requests to the Polygon Mumbai network.

- POLYGON_MUMBAI_RPC_URL: Set a URL for the Polygon Mumbai testnet. You can sign up for a personal endpoint fromAlchemy ,Infura , or another node provider service.
- PRIVATE_KEY: Find the private key for your testnet wallet. If you use MetaMask, follow the instructions toExport a Private Key .Note: Your private key is needed to sign any transactions you make such as making requests.

npx env-encset

Configure your onchain resources

After you configure your local environment, configure some onchain resources to process your requests, receive the responses, and pay for the work done by the DON.

Deploy a Functions consumer contract onPolygon Mumbai

1. Open the FunctionsConsumerExample.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. Under theDeploysection, fill in the router address for your specific blockchain. You can find both of these addresses on theSupported Networks page. ForPolygon Mumbai, the router address is0x6E2dc0F9DB014aE19888F539E59285D2Ea04244C. 6. 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. 7. After you confirm the transaction, the contract address appears in theDeployed Contractslist. Copy the contract address.

Create a subscription

Follow theManaging Functions Subscriptions guide to accept the Chainlink Functions Terms of Service (ToS), create a subscription, fund it, then add your consumer contract address to it.

You can find the Chainlink Functions Subscription Manager atfunctions.chain.link .

Tutorial

This tutorial is configured to get the medianBTC/USDprice from multiple data sources on every block. Read theExamine the code section for a detailed explanation of the code example.

You can locate the scripts used in this tutorial in theexamples/10-automate-functionsdirectory .

1. Make sure to understand theAPI multiple calls guide.
2. Make sure your subscription has enough LINK to pay for your requests. Also, you must maintain a minimum balance to upload encrypted secrets to the DON (Read theminimum balance for uploading encrypted secrets section to learn more). You can check your subscription details (including the balance in LINK) in theChainlink Functions Subscription Manager . If your subscription runs out of LINK, follow theFund a Subscription guide. This guide recommends maintaining at least 2 LINK within your subscription.
3. Get a free API key fromCoinMarketCap and note your API key.
4. Runnpx env-enc setto add an encryptedCOINMARKETCAP_API_KEYto your.env.encfile.

npx env-encset

Deploy an Automated Functions Consumer contract

caution

When using Chainlink Automation, developers should be mindful of the risks of Automation attempting to perform upkeeps indefinitely if a previous upkeep fails to update due to reversion. To learn more, read theAutomation best practices .

1. Deploy a Functions consumer contract onPolygon Mumbai:

2. Open theCustomAutomatedFunctionsConsumerExample.sol in Remix.

Open in Remix What is Remix?1. Compile the contract. 2. Open MetaMask and select thePolygon Mumbainetwork. 3. In Remix under theDeploy & Run Transactionstab, selectInjected Provider - MetaMaskin theEnvironmentlist. Remix will use the MetaMask wallet to communicate withPolygon Mumbai. 4. Under theDeploysection, fill in the router address for your specific blockchain. You can find this address on theSupported Networks page. ForPolygon Mumbai, the router address is0x6E2dc0F9DB014aE19888F539E59285D2Ea04244C. 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 the Deployed Contracts list. Copy your contract address. 2. Create a Chainlink Functions subscription and add your contract as an approved consumer contract.Note: If you followed the previous tutorials, then you can reuse your existing subscription.

Configure your Automation Consumer contract

Configure the request details by calling theupdateRequestfunction. This step stores the encoded request (source code, reference to encrypted secrets if any, arguments), gas limit, subscription ID, and job ID in the contract storage (seeExamine the code). To do so, follow these steps:

1. On a terminal, go to theFunctions tutorialsdirectory .
2. OpenupdateRequest.js and replace the consumer contract address and the subscription ID with your own values:

constconsumerAddress="0x5abE77Ba2aE8918bfD96e2e382d5f213f10D39fA"// REPLACE this with your Functions consumer addressconstsubscriptionId=3// REPLACE this with your subscription ID 3. Run theupdateRequest.js script to update your Functions consumer contract's request details.

Configure Chainlink Automation

The consumer contract that you deployed is designed to be used with acustom logicautomation. Follow the instructions in theRegister a Custom Logic Upkeep guide to register your deployed contract using theChainlink Automation App . Use the following upkeep settings:

- Trigger: Custom logic
- Target contract address: The address of the Chainlink Functions consumer contract that you deployed
- Check data: Leave this field blank
- Gas limit:1000000
- Starting balance (LINK):1

You can leave the other settings at their default values for the example in this tutorial.

At this stage, your Functions consumer contract is configured to get the median Bitcoin price on every block.

Monitor your balances

There are two balances that you must monitor:

- Your subscription balance: Your balance will be charged each time your Chainlink Functions is fulfilled. If your balance is insufficient, your contract cannot send requests. Automating your Chainlink Functions means they will be regularly triggered, so monitor and fund your subscription account regularly. You can check your subscription details (including the balance in LINK) in theChainlink Functions Subscription Manager .
- Your upkeep balance: You can check this balance on theChainlink Automation App . The upkeep balance pays Chainlink Automation Network to send your requests according to your provided time interval. Chainlink Automation will not trigger your requests if your upkeep balance runs low.

Check Result

Go to theChainlink Automation App and connect to Polygon Mumbai. Your upkeep will be listed underMy upkeeps:

Click on your upkeep to fetch de details:

On your terminal, run thereadLatest to read the latest received response:

1. OpenreadLatest.jsand replace the consumer contract address with your own values:

constconsumerAddress="0x5abE77Ba2aE8918bfD96e2e382d5f213f10D39fA"// REPLACE this with your Functions consumer address 1. Run thereadLatest script.

nodeexamples/10-automate-functions/readLatest.js Example:

$ node examples/10-automate-functions/readLatest.js secp256k1 unavailable, reverting to browser version Last request ID is 0x310d57a7af34ae4ce565f5745ff46fe2706e96b25b3172ada60cc60f4603b38e ✅ Decoded response to uint256: 2625865n

Clean up

After you finish the guide, cancel your upkeep in theChainlink Automation App and withdraw the remaining funds. After you cancel the upkeep, there is a 50-block delay before you can withdraw the funds.

Examine the code

CustomAutomatedFunctionsConsumer.sol

// SPDX-License-Identifier: MITpragmasolidity0.8.19;import{FunctionsClient}from"@chainlink/contracts/src/v0.8/functions/v1_0_0/FunctionsClient.sol";import{AutomationCompatibleInterface}from"@chainlink/contracts/src/v0.8/auto *** @title Functions contract used for Automation. * @notice This contract is a demonstration of using Functions and Automation. * @notice NOT FOR PRODUCTION USE */contractCustomAutomatedFunctionsConsumerExampleisFunctionsClient,AutomationCompatibleInterface,ConfirmedOwner{uint256publiclastBlockNumber;bytespublicrequest;uint64pub {}**/ * @notice Checks if upkeep is needed based on the difference between the current and the last block number. * @dev This function checks if the current block number has incremented since the last recorded block number and returns a boolean indicating if upkeep is needed. * @return upkeepNeeded A boolean indicating if upkeep is needed (true if the current block number has incremented since the last recorded block number). * @return performData An empty bytes value since no additional data is needed for the upkeep in this implementation. */functioncheckUpkeep(bytescalldata/ checkData)*externalviewoverridereturns(boolupkeepNeeded,bytesmemoryperformData){upkeepNeeded=block.number-lastBlockNumber>0;// Check if the current block number has incremented since the last recorded block number// We don't use the checkData in this example. The checkData is defined when the Upkeep was registered.return(upkeepNeeded,"");// Return an empty bytes value for performData}/ * @notice Send a pre-encoded CBOR request if the current block number has incremented since the last recorded block number.* /functionperformUpkeep(bytescalldata/ *performData* /)externaloverride{if(block.number-lastBlockNumber>0){lastBlockNumber=block.number;s_upkeepCounter=s_upkeepCounter+1;try i_router.sendRequest(subscriptionId,request,FunctionsRequest.REQUEST_DATA_VERSION,gasLimit,donID)returns(bytes32requestId) {s_lastRequestId=requestId;s_requestCounter=s_requestCounter+1;emitRequestSent(requestId);}catchError(stringmemoryreason) {emitRequestRevertedWithErrorMsg(reason);}catch(bytesmemorydata){emitRequestRevertedWithoutErrorMsg(data);}}// We don't use the performData in this example. The performData is generated by the Automation Node's call to your checkUpkeep function}/// @notice Update the request settings/// @dev Only callable by the owner of the contract/// @param _request The new encoded CBOR request to be set. The request is encoded offchain/// @param _subscriptionId The new subscription ID to be set/// @param _gasLimit The new gas limit to be set/// @param _donID The new job ID to be setfunctionupdateRequest(bytesmemory_request,uint64_subscriptionId,uint32_gasLimit,bytes32_donID)externalonlyOwner{request=_request;subscriptionId=_subscriptionId;gasLimit=_gasLimit;donID=_ ** @notice Store latest result/error * @param requestId The request ID, returned by sendRequest() * @param response Aggregated response from the user code * @param err Aggregated error from the user code or from the execution pipeline * Either response or error parameter will be set, but never both*/functionfulfillRequest(bytes32requestId,bytesmemoryresponse,bytesmemoryerr)internaloverride{if(s_lastRequestId!=requestId) {revertUnexpectedRequestID(requestId);}s_lastResponse=response;s_lastError=err;s_responseCounter=s_responseCounter+1;emitResponse(requestId,s_lastResponse,s_lastError);}} Open in Remix What is Remix? * To write an automated Chainlink Functions consumer contract, your contract must importFunctionsClient.sol . You can read the API reference ofFunctionsClient .

The contract is available in an NPM package, so you can import it from within your project.

import {FunctionsClient} from "@chainlink/contracts/src/v0.8/functions/v1_0_0/FunctionsClient.sol"; * To write a compatible Automations contract, your contract must importAutomationCompatibleInterface.sol .

The contract is available in an NPM package, so you can import it from within your project.

import {AutomationCompatibleInterface} from "@chainlink/contracts/src/v0.8/automation/AutomationCompatible.sol"; * ThelastBlockNumberis stored in the contract. It represents the block number of the last timeperformUpkeepwas triggered. * The encodedrequest,subscriptionId,gasLimit, andjobIdare stored in the contract storage. The contract owner sets these variables by calling theupdateRequestfunction.Note: The request (source code, secrets, if any, and arguments) is encoded offchain. * 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, bytes response, bytes err); * We define two events that your smart contract emits when sending a request to Chainlink Functions fails:

event RequestRevertedWithErrorMsg(string reason); event RequestRevertedWithoutErrorMsg(bytes data); * Pass the router address for your network when you deploy the contract:

constructor(address router) FunctionsClient(router) * The three remaining functions are:

- checkUpkeepfor checking offchain ifperformUpkeepshould be executed.performUpkeepshould only be executed if the current block number is higher than the block number of the last execution.
- performUpkeep: Executed by Chainlink Automation whencheckUpkeepreturnstrue. This function sends the request (encoded inbytes) to the router by calling theFunctionsClientsendRequestfunction.Note: We usetry and catchto gracefully handle reverts ofi_router.sendRequestby emitting an event. We also update thelastBlockNumber, even wheni_router.sendRequestis unsuccessful.
- 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, then increments the response counters_responseCounterbefore emitting theResponseevent.

s_lastResponse = response; s_lastError = err; s_responseCounter = s_responseCounter + 1; emit Response(requestId, s_lastResponse, s_lastError);

source.js

The JavaScript code is similar to the one used in theCall Multiple Data Sources tutorial.

updateRequest.js

The JavaScript code is similar to the one used in theAutomate your Functions (Time-based Automation) tutorial.

readLatest.js

The JavaScript code is similar to the one used in theAutomate your Functions (Time-based Automation) tutorial.