

Request Computation

This tutorial shows you how to run computations on the Chainlink Functions Decentralized Oracle Network (DON). The example code computes the [geometric mean](#) of numbers in a list. After [OCR](#) completes offchain computation and aggregation, it returns the result to your smart contract.

Prerequisites

note

You might skip these prerequisites if you have followed one of these [guides](#) . You can check your subscription details (including the balance in LINK) in the [Chainlink Functions Subscription Manager](#) . If your subscription runs out of LINK, follow the [Fund 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, configure [code.js](#), clone the [smartcontractkit/smart-contract-examples](#) repository, and configure `a.env` file 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 the [LINK Token Contracts](#) page.
4. [Polygon Mumbai testnet and LINK token contract](#)
5. Request testnet MATIC from the [Polygon Faucet](#) .
6. Request testnet LINK from [faucets.chain.link/mumbai](#) .

Install the required frameworks and dependencies:

1. [Install the latest release of Node.js 20](#). Optionally, you can use the [nvm package](#) to switch between Node.js versions with `nvm use 20`

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

node-v\$node-vv20.9.0 2. In a terminal, clone the [smart-contract examples](#) repository and change directories. This example repository imports the [Chainlink Functions Toolkit NPM package](#). You can import this package to your own projects to enable them to work with Chainlink Functions.

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

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

- ## 1. Set an encryption password for your environment variables

Run `npmx env-enc set-pw 2`. Run `npmx env-enc set` to configure a `.env.enc` file 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 from [Alchemy Infura](#), or another node provider service.
- **PRIVATE_KEY:** Find the private key for your testnet wallet. If you use MetaMask, follow the instructions to [Export 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 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. Under the Deploy section, fill in the router address for your specific blockchain. You can find both of these addresses on the [Supported Networks](#) page. For Polygon Mumbai, the router address is 0x6E2dc0F9BD014aE1988F539E59285D2Ea04244C. 6. 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. 7. After you confirm the transaction, the contract address appears in the Deployed Contracts list. Copy the contract address.

Create a subscription

Follow the [Managing 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 at functions.chain.link.

Tutorial

This tutorial is configured to get the average (geometric mean) from a list of numbers 1,2,3,4,5,6,7,8,9,10. Read the [Examine the code](#) section for a detailed description of the code example.

You can locate the scripts used in this tutorial in the [examples/1-simple-computation](#) directory.

To run the example:

1. Open the `filerequest.js`, which is located in the `1-simple-computation` folder.
2. Replace the consumer contract address and the subscription ID with your own values.

const consumerAddress="0x8dF78B7EE3128D00E90611FBED20A71397064D9"// REPLACE this with your Functions consumer address
const subscriptionId=3// REPLACE this with your subscription ID 3. Make a request;

`nodeexamples/1-simple-computation/request.js` The script runs your function in a sandbox environment before making an onchain transaction.

\$ node examples/1-simple-computation/request.js secp256k1 unavailable, reverting to browser version Start simulation... Performing simulation with the following versions: deno 1.36.3 (release, aarch64-apple-darwin) v8 11.6.189.12 typescript 5.1.6

[illegible]

Estimate request costs... Duplicate definition of Transfer (Transfer(address,address,uint256,bytes). Transfer(address,address,uint256)) Fulfillment cost estimated to 0.00000000000000215 LINK

Make request...

✓ Functions request sent! Transaction hash 0x662988579b77c0ab2ca78e40a3ece7d150bd6afb26bbaf42b2c3e5f13a4f895b. Waiting for a response... See your request in the explorer <https://mumbai.polygonscan.com/tx/0x662988579b77c0ab2ca78e40a3ece7d150bd6afb26bbaf42b2c3e5f13a4f895b>

[illegible]

✓ Decoded response to uint256: 453nThe output of the example gives you the following information:

- Your request is first run on a sandbox environment to ensure it is correctly configured.
- The fulfillment costs are estimated before making the request.
- Your request was successfully sent to Chainlink Functions. The transaction in this example is [is0x662988579b77c0ab2ca78e40a3ece7d150bd6afb26bbaf42b2c3e5f13a4f895b](#) and the request ID is [0x0xeddbdc53de4c796cb44e693a3a34600d616b76ce7ea3902849e687a20a3c8d86fe](#).

1. Definition of necessary identifiers:
2. routerAddress: Chainlink Functions router address on Polygon Mumbai.
3. donId: Identifier of the DON that will fulfill your requests on Polygon Mumbai.
4. explorerUrl: Block explorer url of Polygon Mumbai.
5. source: The source code must be a string object. That's why we use `readFileSync` to read `source.js` and then `callToString()` to get the content as a string object.
6. args: During the execution of your function, These arguments are passed to the source code.
7. gasLimit: Maximum gas that Chainlink Functions can use when transmitting the response to your contract.

8. Initialization of ethersigner and provider objects. The signer is used to make transactions on the blockchain, and the provider reads data from the blockchain.
9. Simulating your request in a local sandbox environment:
10. Use `simulateScript` from the Chainlink Functions NPM package.
11. Read the response of the simulation. If successful, use the Functions NPM package `decodeResult` function and `ReturnType` enum to decode the response to the expected returned type (`ReturnType.uint256` in this example).
12. Estimating the costs:
13. Initialize a `SubscriptionManager` from the Functions NPM package, then call the `estimateFunctionsRequestCost`.
14. The response is returned in Juels (1 LINK = 10^{18} Juels). Use the `ethers.utils.formatEther` utility function to convert the output to LINK.
15. Making a Chainlink Functions request:
16. Initialize your functions consumer contract using the contract address, abi, and ethers signer.
17. Call the `sendRequest` function of your consumer contract.
18. Waiting for the response:
19. Initialize a `ResponseListener` from the Functions NPM package and then call the `listenForResponseFromTransaction` function to wait for a response. By default, this function waits for five minutes.
20. Upon reception of the response, use the Functions NPM package `decodeResult` function and `ReturnType` enum to decode the response to the expected returned type (`ReturnType.uint256` in this example).