Chainlink Functions Architecture

Prerequisites

Read the Chainlink Functionsintroduction to understand all the concepts discussed on this page.

Request and Receive Data

This model is similar to theBasic Request Model : The consumer contract initiates the cycle by sending a request to theFunctionsRouter contract . Oracle nodes watch for events emitted by theFunctionsCoordinator contract and run the computation offchain. Finally, oracle nodes use theChainlink OCR protocol to aggregate all the returned before passing the single aggregated response back to the consumer contract via a callback function.

The main actors and components are:

- Initiator (or end-user): initiates the request to Chainlink Functions. It can be anEOA (Externally Owned Account) or Chainlink Automation.
- Consumer contract: smart contract deployed by developers, which purpose is to interact with the FunctionsRouter to initiate a request.
- FunctionsRouter contract: manages subscriptions and is the entry point for consumers. The interface of the router is stable. Consumers call thesendRequestmethod to initiate a request.
- FunctionsCoordinator contracts: interface for theDecentralized Oracle Network . Oracle nodes listen to events emitted by the coordinator contract and interact with the coordinator to transmit the responses.
- DON: Chainlink Functions are powered by aDecentralized Oracle Network . The oracle nodes are independent of each other and are responsible for executing the request's source code. The nodes use theChainlink OCR protocol to aggregate all the nodes' responses. Finally, a DON's oracle sends the aggregate response to the consumer contract in a callback.
- Secrets endpoint: To transmit their secrets, users can encrypt them with the DON public key and then upload them to the secrets endpoint, a highly available service for securely sharing encrypted secrets with the nodes.Note: An alternative method involves self-hosting secrets. In this approach, users provide a publicly accessible HTTP(s) URL, allowing nodes to retrieve the encrypted secrets. Refer to thesecrets management page for detailed information on both methods.
- Serverless Environment: Every Oracle node accesses a distinct, sandboxed environment for computation. While the diagram illustrates an API request, the computation isn't restricted solely to this. You can perform any computation, from API calls to mathematical operations, using vanillaDeno code without module imports. Note: All nodes execute identical computations. If the target API has throttling limits, know that multiple simultaneous calls will occur since each DON node will independently run the request's source code.

Let's walk through the sequence of interactions among these components, as illustrated in the diagram:

1. If there are secrets, a user encrypts secrets with the public key linked to the DON master secret key (MSK) and then uploads the encrypted secrets to the secrets endpoint. The secrets endpoint pushes the encrypted secrets to the nodes part of the DON (The secrets capability depicted in this diagram is calledthreshold encryptionand is explained insecrets management).
2. AnEOA (Externally Owned Account) or Chainlink Automation initiates the request by calling the consumer contract.
3. The consumer contract should inherit theFunctionsClient contract. This ensures it will be able to receive responses from theFunctionsRouter contract via thehandleOracleFulfillmentcallback. The router contract starts the billing to estimate the fulfillment costs and block the amount in thereservation balance(To learn more, readCost simulation). Then it calls theFunctionsCoordinator contract .
4. The coordinator contract emits anOracleRequestevent containing information about the request.
5. On reception of the event by the DON, the DON's nodes decrypt the secrets using threshold decryption (The threshold encryption feature is explained insecrets management). Each DON's Oracle node executes the request's source code in a serverless environment.
6. The DON runs the Offchain Reporting protocol (OCR) to aggregate the values returned by each node's execution of the source code.
7. A DON's oracle node transmits the attested report (which includes the aggregated response) to the FunctionsCoordinator contract.
8. The FunctionsCoordinator contract calls the FunctionsRouter'sfulfillmethod to calculate the fulfillment costs and finalize the billing (To learn more, readCost calculation).
9. The FunctionsRouter contract calls the consumer contract's callback with the aggregated response.

Note: Chainlink Functions requests are not limited to API requests. The diagram depicts an example of API requests, but you can request the DON to run any computation.

Subscription management

Chainlink Functions do not require your consumer contracts to hold LINK tokens and send them to oracles when making

requests. Instead, you must create a subscription account and fund it to pay for your Chainlink Functions requests, so your consumer contracts don't need to hold LINK when calling Chainlink Functions.

Concepts

- Terms of service (ToS): Before interacting with Chainlink Functions, users must agree to the terms of service. Once signed, the accounts that can manage subscriptions are added to theallowedSendersin theToS allow list contract .
- Chainlink Functions Subscription Manager: A user interface that allows users to agree to the sign Terms of service and interact with the FunctionsRouter to manage subscriptions.
- Subscription account: An account that holds LINK tokens and makes them available to fund requests to Chainlink DON. ASubscription IDuniquely identifies each account.
- Subscription ID: 64-bit unsigned integer representing the unique identifier of theSubscription account.
- Subscription owner: The wallet address that creates and manages aSubscription account. Any account can add LINK tokens to the subscription balance. Still, only the owner can add consumer contracts to the subscription, remove consumer contracts from the subscription, withdraw funds, or transfer a subscription. Only the subscription owner can generate encrypted secrets for requests that use theirSubscription ID.
- Subscription balance: The amount of LINK maintained on yourSubscription account. Requests from consumer contracts are funded as long as sufficient funds are in the balance, so be sure to maintain sufficient funds in yourSubscription balanceto pay for the requests and keep your applications running.
- Subscription reservation: The amount of LINK blocked on theSubscription balance. It corresponds to the total LINK amount to be paid by in-flight requests.
- Effective balance: The amount of LINK available on yourSubscription account.Effective balance = Subscription balance - Subscription reservation.
- Subscription consumers: Consumer contracts are approved to use funding from yourSubscription accountwhile making Chainlink Functions requests. The consumers receive response data in a callback.

Accept ToS

To ensure compliance and governance, Chainlink Functions mandates that any account that manages a subscription must first accept the platform's Terms of Service (ToS). The acceptance is verified by cross-referencing the account with theallowedSendersregistry contained within theTermsOfServiceAllowList contract .

The acceptance process is initiated via the Chainlink Functions Subscription Manager. After a user accepts the ToS by generating the required signature with their externally owned account (EOA), they transmit proof of acceptance to theTermsOfServiceAllowList contract . Upon successful validation of the proof, the EOA is added to theallowedSendersregistry, permitting it to manage subscriptions.

Create subscription

After the ToS is accepted, EOAs can create subscriptions. Upon creation, theFunctionsRouter assigns a unique identifier,Subscription ID.

Note: EOAs can directly interact with the FunctionsRouter contract using their preferred web3 library, such as web3.js or ethers.js.

Fund subscription

You must fund your subscription accounts with enough LINK tokens:

1. Connect your EOA to the Chainlink Functions Subscription Manager.
2. Fund your subscription account. The Chainlink Functions Subscription Manager abstracts the following:1. CalltransferAndCallon the LINK token contract, transferring LINK tokens along with theSubscription IDin the payload.
3. The FunctionsRouter contract implementsonTokenTransfer: It parses theSubscription IDfrom the payload and funds the subscription account with the transferred LINK amount.

Note: EOAs can directly interact with the LinkToken contract using their preferred web3 library, such as web3.js or ethers.js.

Add consumer

You must allowlist your consumers' contracts on your subscription account before they can make Chainlink Functions requests:

1. Connect your EOA to the Chainlink Functions Subscription Manager.
2. Add the address of the consumer contract to the subscription account.
3. The Chainlink Functions Subscription Manager interacts with the FunctionsRouter contract to register the consumer contract address to the subscription account.

Note: EOAs can directly interact with the FunctionsRouter contract using a web3 library, such as web3.js or ethers.js.

Remove consumer

To prevent further Chainlink Functions requests from a given consumer contract, you must remove it from your subscription account:

1. Connect your EOA to the Chainlink Functions Subscription Manager.
2. Remove the address of the consumer contract from the subscription account.
3. The Chainlink Functions Subscription Manager communicates with the FunctionsRouter contract to remove the consumer contract address from the subscription account.

Note: You can still remove consumers from your subscription even if in-flight requests exist. The consumer contract will still receive a callback, and yourSubscription Accountwill be charged.

Note: EOAs can directly interact with the FunctionsRouter contract using a web3 library, such as web3.js or ethers.js.

Cancel subscription

To cancel a subscription:

1. Connect your EOA to the Chainlink Functions Subscription Manager.
2. Cancel your subscription, providing theSubscription Balancereceiver account address. The Chainlink Functions Subscription Manager handles the following processes:1. Invokes thecancelSubscriptionfunction on the FunctionsRouter contract, deleting theSubscription IDand removing all associated consumers.
3. Transfers the remainingSubscription Balanceto the specified receiver account.

Note: EOAs can directly interact with the FunctionsRouter contract using their preferred web3 library, such as web3.js or ethers.js.

Note: Subscriptions cannot be canceled while there are in-flight requests. Furthermore, any expired requests (requests that have yet to receive a response within 5 minutes) must be timed out before cancellation.

Transferring ownership of a Subscription

Transferring ownership is currently only supported using theFunctions Hardhat Starter kit or theFunctions Toolkit NPM package :

1. Use thefunctions-sub-transfercommand to initiate the transfer of ownership by specifying the new owner's address.
2. As a prerequisite, the prospective owner must be part of theallowedSendersregistry within theTermsOfServiceAllowList contract . This verifies their acceptance of the Chainlink Functions' Terms of Service (ToS).
3. The prospective owner should use theFunctions Hardhat Starter kit and run thefunctions-sub-acceptcommand to confirm the ownership transfer.

Note: This guide will be updated as soon as the Chainlink Functions Subscription Manager supports ownership transfers.