Running oracles

Oracles are a fundamental piece for Mento, the stability protocol behind Celo stable assets. Their purpose is to forward to the blockchain the price of CELO/USD, CELO/EUR and CELO/BRL.

Getting started

Oracles work by running a client that fetches the price from centralized exchanges (CEX) and pushes those prices on-chain by callingSortedOracles.report(address token, uint256 value, address lesserKey, address greaterKey) . SortedOracles is aCelo Core Contract .

Areference implementation of such client is written in TypeScript and would be used for this guide. Releases for this client can be foundhere .

Requirements

- One VM dedicated for each oracle is recommended, but it is acceptable that they run multiple instances in the case they are for different stables.
- A dedicated full node running in its own VM. Minimal hardware requirements and instructions on how to run a full node can be foundhere
- .
- The private key of an address on Celo, which can be stored on a private key file, on a HMS or hosted in the full nodes itself. More information about each can be found below.

It is not strictly required but it is recommended to have theCelo CLI available at least in your local environment, and ideally in each VM. It could be especially useful to respond to on-call.

Setting up the environment

Find the latest stable Docker Image for the oracle in the oracle releases here .

From the oracle VM, make sure you can access your node. This can be done via the Celo CLI with this command:

celocli node:synced --node YOUR_NODE_HOSTNAME:YOUR_NODE_PORT

Also make sure your node is accesible via WS, usually full nodes listen in port8546 .

warning Using Forno or other public full node providers to run the oracles in production is strongly discouraged. Oracles doing so wouldn't be elegible for rewards. The oracle is configured by passing individual environment variables or an env file when starting the Docker container. You'll need to create an env file named.env.prod in your oracle VM. A template env file in a format accepted by Docker can be found in theGithub repository . A list of all the available, as well as required variables, can be foundhere .

Running with HSM

Using HSM is the recommended way to store the keys for the oracles. Currently supported HSM are Azure and AWS. If you're have already configured HSM, the relevant variables to add to your.env.prod are:

AWS:

- WALLET_TYPE=AWS_HSM
- AWS_KEY_REGION

Azure:

- WALLET_TYPE=AZURE_HSM
- AZURE_KEY_VAULT_NAME

Using a private key

warning This method is not recommended in production as the private key remains unencrypted in the VM. You can create a new private key with:

celocli account:new

The output field ofprivateKey should be stored to a file and its path should be set in the env variablePRIVATE_KEY_PATH . AditionallyWALLET_TYPE should also be set toPRIVATE_KEY . This private key should have some CELO balance used for

gas to sign the report transactions.

Setting up your keys in the node

Instructions to generate an account and store it in the node can be foundhere .

Recommended configuration

warning WARNING: it is encouraged that before running the oracles in production, they should run for at least a week in one of the Celo Public testnets. The configuration currently run by cLabs in production can be foundhere for each stable token. It is strongly advised not to modify the recommended values, especially the exchange sources, unless there is good data to support it.

The only variable that is not set in the env file isPRICE_SOURCES . This sets what exchanges and prices shall be used to report. It is recommended to store this in a file calledprice_sources and the export the content to a new env variable withcat .

export PRICE_SOURCES=(cat price_sources)

An example of such file for CELO/USD is:

[[{ exchange: 'BITTREX', symbol: 'CELOUSD', toInvert: false }], [{ exchange: 'COINBASE', symbol: 'CELOUSD', toInvert: false }], [{ exchange: 'OKCOIN', symbol: 'CELOUSD', toInvert: false }], [{ exchange: 'BINANCE', symbol: 'CELOBUSD', toInvert: false }, { exchange: 'COINBASE', symbol: 'BUSDUSD', toInvert: false },],] tip Note that this example configuration is using three direct pairs, and the last one is an implicit pair calculated using two exchanges. This is useful in the case extra liquidity is required to calculate the price.

Existing exchange connectos

Available connectors are, in alphabetical order:

- Binance
- Bitso
- Bittrex
- Coinbase
- Novadax
- OkCoin

Running the node

Once all the environment variables are set in the VM, an oracle can be started with:

docker run --name celo-oracle -it --restart unless-stopped --env-file .env.prod -e PRICE_SOURCES=PRICE_SOURCES us-west1-docker.pkg.dev/celo-testnet-production/celo-oracle/celo-oracle:1.0.0-rc2

If your oracle is not yet enabled by governance, you'll see these messages in the terminal:

Account 0x... is not whitelisted as an oracle for CURRENCY_PAIR

As soon as governance enables it, the node should start reporting automatically.

Governance

The last step to run an oracle is to enable their addresses on-chain using theCelo Governance Process . Only addresses allowed by governance are allowed to report. Thus, the first step to spin up a new oracle is creating a governance proposal and submit on-chain for community voting. An example of such proposal can be foundhere .

Using kubernetes

ReferenceHelm Charts configuration used by cLabs can be found in thecelo-monorepo repository .

Metrics

The oracle client supports metrics suitable for Prometheus. Available metrics and their configuration can be found in thetechnical documentation .

Monitoring

There are two public dashboards deployed where the community can watch how individual oracle is performing on mainnet:

1. One with high sampling but short timeframe
2. One with low sampling but longer timeframe

Building from source

Instructions can be found in the development documentation . Edit this page Previous Using RedStone oracles Next Contract Addresses