

README.md:

Welcome

Welcome to the technical documentation site for the Celo project! You can check out the live site at <https://docs.celo.org>.

Getting Started

This documentation site is built using Docusaurus v2.

You can run the site locally by cloning this repository, installing dependencies and starting docusaurus.

```
sh
git clone https://github.com/celo-org/docs.git
cd docs
yarn
yarn start
```

Edit files in the /docs directory to make updates to the documentation pages. Pages will be automatically updated when you are running the development server.

Files

- .gitignore - This specifies which files Git should ignore when committing and pushing to remote repositories.
- docusaurus.config.js - This is the configuration file for Docusaurus. You can manage the links in the header and footer, and site metadata here. A more in-depth introduction to this configuration file is available on the Docusaurus website and full documentation for the API is [here](#).
- package.json - This specifies the dependencies for the website and the commands that you can run with yarn.
- sidebars.js - This specifies the sidebars for your documentation. The full documentation for this file is available on the Docusaurus website.
- yarn.lock - This specifies the full dependencies for the website including the dependencies of the dependencies listed in package.json. Do not edit this file directly. Edit package.json instead, then run yarn as described above.

The .md files in the docs directory are the docs. See the Docusaurus website for the full documentation on how to create docs and to manage the metadata.

Contributing

We welcome contributions from the community. Please review our contribution guidelines for more information.

You can find many of the other main Celo project repositories at <https://github.com/celo-org>.

Internationalization

We are managing translations using Crowdin. You can find the Celo Docs Crowdin project [here](#). From here, click on the language that you would like to help translate the docs into.

Due to inconsistencies with managing the source code in 2 places (Github and Crowdin), only the English source code is stored in this repo. The translation sources are stored on Crowdin and are fetched when the site is built.

Some files are marked as high priority. In the following image you can see the up arrow on the right side of the file line, indicating the file is a higher priority than others. Please focus on files marked as high priority.

If you have any questions, please post in the project discussions tab [here](#).

Contact

Join us on Discord or file an issue.

Code of Conduct

Please review the code of conduct before contributing.

introducing-prosperity-passport.md:

title: Introducing Prosperity Passport - The First Soulbound Token-Powered Web3 Identity Solution for Celo Blockchain
description: Introducing Prosperity Passport - a groundbreaking Web3 identity solution for Celo blockchain. The first-ever soulbound token-powered platform to enable secure, decentralized identity verification.
authors:

- name: Kunal Dawar
title: Community Member, Celo Foundation
url: <https://github.com/developerkunal>
imageurl: <https://avatars.githubusercontent.com/u/35455566?v=4>

tags: ["beginner", "celosage", "celo"]

hidetableofcontents: false

slug: /tutorials/introducing-prosperity-passport-the-first-soulbound-token-powered-web3-identity-solution-for-celo-blockchain

!header

🔗 Introduction

The rise of Web3 technologies has created new opportunities for decentralized identity solutions that provide more privacy, security, and control for users. One such solution is Prosperity Passport, the first soulbound token-powered identity solution for the Celo blockchain.

What is Prosperity Passport?

Prosperity Passport is a decentralized identity solution that allows users to create and manage their own digital identity on the Celo blockchain. It leverages the power of soulbound tokens to secure and authenticate users' identities, making it more difficult for hackers and bad actors to steal or manipulate them.

Soulbound tokens are unique digital assets that are created when a user's identity is verified on the Celo blockchain. These tokens are then bound to the user's identity and cannot be transferred to anyone else. This means that only the rightful owner of the token can access their digital identity, making it more secure and tamper-proof.

How does Prosperity Passport work?

To use Prosperity Passport, users first need to verify their identity on the Celo blockchain. This can be done by providing some basic information, such as their name, address, and phone number. Once their identity is verified, a soulbound token is created and bound to their identity.

Users can then use their soulbound token to access various services and applications on the Celo blockchain. For example, they can use it to authenticate themselves on decentralized finance (DeFi) platforms, access decentralized marketplaces, and participate in decentralized governance.

What are the benefits of using Prosperity Passport?

There are several benefits to using Prosperity Passport as your decentralized identity solution:

- Increased privacy: With Prosperity Passport, users have more control over their personal data and can choose which information to share with which applications.
- Improved security: Soulbound tokens make it more difficult for bad actors to steal or manipulate users' identities, making Prosperity Passport a more secure solution.
- Seamless user experience: With Prosperity Passport, users only need to verify their identity once and can then access various services and applications on the Celo blockchain without having to repeat the process.

- Interoperability: Prosperity Passport is designed to be interoperable with other decentralized identity solutions, making it easier for users to switch between different solutions as needed.

How to Claim a .Celo Domain & Mint a Prosperity Passport

To get started with Prosperity Passport, users can mint their Prosperity Passport, which automatically creates their soulbound identity on Celo. Users can also mint a .celo domain, similar to an ENS domain, but on the Celo blockchain.

Here are the steps to claim a .Celo domain and mint a Prosperity Passport:

1. Visit app.prosperity.global

Connect your wallet and authenticate it on the Prosperity Passport dashboard. If you are not yet on Celo, add Celo Mainnet to your MetaMask and switch to Celo Mainnet.

!visitwebsite

2. You'll be prompted to create and mint your unique .celo domains (e.g. refi.celo; green.celo). Pick a domain name for your Prosperity Passport, and the time-frame you'd like to register it for. You can use any combination of numbers, letters and emojis.

!connect

!connect2

!getstarted

!dashboard

3. Five-character .celo domains and above are free – you just need to pay the Celo gas fee. The average gas fee is \$0.0005, according to Celo's website.

!domains

!searchdomains

4. Approve the transaction & success, that's it! Don't forget to share your newly minted .celo domain on social.

.Celo Domains

!register

!done

5. Once you create your Prosperity Passport, you can give it a domain name in the format of .celo, similar to an ENS. Up next, Masa will be working to integrate full .celo domain names within the Celo ecosystem

wallet infrastructure, so you can use your .celo as your human-readable address across Celo.

Feel free to mint as many .celo domains as you like. Use any creative combination of letters, numbers and emojis to create fun, unique, and rare .celo domains. Don't forget to share your newly minted .celo on social!

Rewards

Masa has some exciting rewards, perks, and partnerships lined up within the Celo Ecosystem in the near future. Get started by creating your Prosperity Passport at app.prosperity.passport.

Conclusion

Prosperity Passport is a promising decentralized identity solution that leverages the power of soulbound tokens to provide more privacy, security, and control for users on the Celo blockchain. As Web3 technologies continue to evolve, it's exciting to see innovative solutions like Prosperity Passport emerge to help users take back control of their digital identities.

About Author

Hi! My name is Kunal Dawar and I am a Full Stack web2/web3 Developer. I have participated in numerous hackathons and have been fortunate enough to win many of them.

One thing that I am truly passionate about is creating things that are reliable and don't break easily. I believe that creating high-quality products is important not only for the users but also for the overall growth and success of a business.

In my free time, I enjoy learning about new technologies and staying up-to-date with the latest trends in the field. I also love to share my knowledge with others and mentor those who are interested in pursuing a career in web development.

How-To-Build-A-Multi-Signature-Wallet-Contract.md:

title: How To Build A Multi Signature Wallet Contract That Requires Multiple Approvals For Transactions On Celo

description: In this tutorial, we will walk through the process of building a multi-signature wallet contract using Solidity and Remix IDE
authors:

- name: 4 Jonathan Iheme
url: <https://github.com/4undRaiser>
imageurl: <https://avatars.githubusercontent.com/u/87926451?s=96&v=4>
tags: [celosage, solidity, celo, intermediate]
hidetableofcontents: true
slug: "/tutorials/how-to-build-a-multi-signature-wallet-contract-that-requires-multiple-approvals-for-transactions-on-celo"

!header

Introduction

A multi-signature wallet contract is a type of smart contract that requires multiple approvals before executing a transaction. This can be useful for organizations or groups that want to maintain shared control over funds or resources. In this tutorial we'll create a simple multi-signature wallet contract written with Solidity:

Here's the github repo of our code. [source code](#)

Prerequisites

To follow this tutorial, you will need the following:

- Basic knowledge of Solidity programming language.
- A Development Environment Like Remix.
- The celo Extension Wallet.

SmartContract

Let's begin writing our smart contract in Remix IDE

The completed code Should look like this.

```
solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract MultiSigWallet {
    address[] public owners;
    uint256 public required;

    struct Transaction {
        address destination;
        uint256 value;
        bytes data;
        bool executed;
    }

    mapping(uint256 => Transaction) public transactions;
    mapping(uint256 => mapping(address => bool)) public confirmations;
    uint256 public transactionCount;

    modifier validRequirement(uint256 ownerCount, uint256 required) {
        require(required > 0, "Required should be greater than 0");
        require(ownerCount >= required, "Owners count should be greater
than or equal to required");
    }
}
```

```

    modifier ownerExists(address owner) {
        require(isOwner(owner), "Not an owner");
    }

    modifier notNull(address address) {
        require(address != address(0), "Address should not be null");
    }

    constructor(address[] memory owners, uint256 required)
    validRequirement(owners.length, required) {
        for (uint256 i = 0; i < owners.length; i++) {
            require(!isOwner(owners[i]), "Duplicate owner");
            owners.push(owners[i]);
        }
        required = required;
    }

    function isOwner(address owner) public view returns (bool) {
        for (uint256 i = 0; i < owners.length; i++) {
            if (owners[i] == owner) {
                return true;
            }
        }
        return false;
    }

    function submitTransaction(address destination, uint256 value, bytes
memory data)
    public
    ownerExists(msg.sender)
    notNull(destination)
    returns (uint256)
    {
        uint256 transactionId = addTransaction(destination, value, data);
        confirmTransaction(transactionId);
        return transactionId;
    }

    function confirmTransaction(uint256 transactionId) public
    ownerExists(msg.sender) {
        require(!confirmations[transactionId][msg.sender], "Transaction
already confirmed by this owner");
        confirmations[transactionId][msg.sender] = true;
        executeTransaction(transactionId);
    }

    function executeTransaction(uint256 transactionId) public {
        require(transactions[transactionId].executed == false,
"Transaction already executed");
        if (isConfirmed(transactionId)) {
            transactions[transactionId].executed = true;

```

```

        (bool success, ) =
transactions[transactionId].destination.call{value:
transactions[transactionId].value}{
    transactions[transactionId].data
};
    require(success, "Transaction execution failed");
}
}

function isConfirmed(uint256 transactionId) public view returns
(bool) {
    uint256 count = 0;
    for (uint256 i = 0; i < owners.length; i++) {
        if (confirmations[transactionId][owners[i]]) {
            count += 1;
        }
        if (count == required) {
            return true;
        }
    }
    return false;
}

function addTransaction(address destination, uint256 value, bytes
memory data)
    internal
    notNull(destination)
    returns (uint256)
{
    uint256 transactionId = transactionCount;
    transactions[transactionId] = Transaction({
        destination: destination,
        value: value,
        data: data,
        executed: false
    });
    transactionCount += 1;
    return transactionId;
}

function getOwners() public view returns (address[] memory) {
    return owners;
}

function getTransaction(uint256 transactionId) public view returns
(address destination, uint256 value, bytes memory data, bool executed) {
    Transaction memory transaction = transactions[transactionId];
    return (transaction.destination, transaction.value, transaction.data,
transaction.executed);
}

function getConfirmationCount(uint256 transactionId) public view returns
(uint256) {
    uint256 count = 0;

```



```

        for (uint256 i = 0; i < owners.length; i++) {
            if (confirmations[transactionId][owners[i]]) {
                count += 1;
            }
        }
        return count;
    }

function isConfirmedBy(uint256 transactionId, address owner) public view
returns (bool) {
    return confirmations[transactionId][owner];
}

receive() external payable {}
}

```

Breakdown

First, we declared our license and the solidity version.

```

solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

```

State Variables

The state variables of the contract are defined next:

```

solidity
contract MultiSigWallet {
    address[] public owners;
    uint256 public required;

    struct Transaction {
        address destination;
        uint256 value;
        bytes data;
        bool executed;
    }

    mapping(uint256 => Transaction) public transactions;
    mapping(uint256 => mapping(address => bool)) public confirmations;
    uint256 public transactionCount;
}

```

The owners variable is an array of addresses that represent the owners of the multi-signature wallet. The required variable represents the number of signatures required to execute a transaction.

The Transaction struct defines the properties of a transaction, including the destination address, value, data, and execution status.

The transactions mapping stores the transactions by their IDs. The confirmations mapping stores the confirmations for each transaction by the owner address. The transactionCount variable keeps track of the number of transactions.

Modifiers

```
solidity
    modifier validRequirement(uint256 ownerCount, uint256 required) {
        require(required > 0, "Required should be greater than 0");
        require(ownerCount >= required, "Owners count should be greater
than or equal to required");
        ;
    }

    modifier ownerExists(address owner) {
        require(isOwner(owner), "Not an owner");
        ;
    }

    modifier notNull(address address) {
        require(address != address(0), "Address should not be null");
        ;
    }
```

Modifiers are used to add conditions to functions. The validRequirement modifier checks if the number of owners and the required number of signatures are valid. The ownerExists modifier checks if the address passed is one of the owners. The notNull modifier checks if an address is not null.

Constructor

```
solidity
    constructor(address[] memory owners, uint256 required)
    validRequirement(owners.length, required) {
        for (uint256 i = 0; i < owners.length; i++) {
            require(!isOwner(owners[i]), "Duplicate owner");
            owners.push(owners[i]);
        }
        required = required;
    }
```

The constructor takes an array of addresses representing the owners and the required number of signatures. It calls the validRequirement modifier to check if the parameters are valid.

The constructor adds each owner to the owners array and sets the required number of signatures.

Functions

The contract defines several functions:

```
solidity
function isOwner(address owner) public view returns (bool) {
    for (uint256 i = 0; i < owners.length; i++) {
        if (owners[i] == owner) {
            return true;
        }
    }
    return false;
}
```

The isOwner function checks if the address passed is one of the owners.

```
solidity
function submitTransaction(address destination, uint256 value, bytes
memory data)
    public
    ownerExists(msg.sender)
    notNull(destination)
    returns (uint256)
{
    uint256 transactionId = addTransaction(destination, value, data);
    confirmTransaction(transactionId);
    return transactionId;
}
```

The submitTransaction function creates a new transaction and adds it to the transactions mapping using the addTransaction function. It then calls the confirmTransaction function to confirm the transaction.

```
solidity
function confirmTransaction(uint256 transactionId) public
ownerExists(msg.sender) {
    require(!confirmations[transactionId][msg.sender], "Transaction
already confirmed by this owner");
    confirmations[transactionId][msg.sender] = true;
    executeTransaction(transactionId);
}
```

The confirmTransaction function confirms a transaction by setting the confirmation for the transaction ID and the owner address to true. It then calls the executeTransaction function to execute the transaction if it has been confirmed by all required owners.

```
solidity
function executeTransaction(uint256 transactionId) public {
```

```

        require(transactions[transactionId].executed == false,
"Transaction already executed");
        if (isConfirmed(transactionId)) {
            transactions[transactionId].executed = true;
            (bool success, ) =
transactions[transactionId].destination.call{value:
transactions[transactionId].value}(
            transactions[transactionId].data
        );
            require(success, "Transaction execution failed");
        }
    }
}

```

The executeTransaction function executes a transaction if it has not been executed yet and if it has been confirmed by all required owners. It uses the call function to send the value and data to the destination address.

```

solidity
function isConfirmed(uint256 transactionId) public view returns (bool) {
    uint256 count = 0;
    for (uint256 i = 0; i < owners.length; i++) {
        if (confirmations[transactionId][owners[i]]) {
            count += 1;
        }
        if (count == required) {
            return true;
        }
    }
    return false;
}

```

The isConfirmed function checks if a transaction has been confirmed by all required owners.

```

solidity
function addTransaction(address destination, uint256 value, bytes
memory data)
    internal
    notNull(destination)
    returns (uint256)
{
    uint256 transactionId = transactionCount;
    transactions[transactionId] = Transaction({
        destination: destination,
        value: value,
        data: data,
        executed: false
    });
    transactionCount += 1;
    return transactionId;
}

```

The addTransaction function adds a new transaction to the transactions mapping and returns the transaction ID.

```
solidity
function getOwners() public view returns (address[] memory) {
    return owners;
}
```

The getOwners function returns the array of owner addresses.

```
solidity
function getTransaction(uint256 transactionId) public view returns
(address destination, uint256 value, bytes memory data, bool executed) {
    Transaction memory transaction = transactions[transactionId];
    return (transaction.destination, transaction.value, transaction.data,
transaction.executed);
}
```

The getTransaction function returns the properties of a transaction by its ID.

```
solidity
function getConfirmationCount(uint256 transactionId) public view returns
(uint256) {
    uint256 count = 0;
    for (uint256 i = 0; i < owners.length; i++) {
        if (confirmations[transactionId][owners[i]]) {
            count += 1;
        }
    }
    return count;
}
```

The getConfirmationCount function returns the number of confirmations for a transaction by its ID.

```
solidity
function isConfirmedBy(uint256 transactionId, address owner) public view
returns (bool) {
    return confirmations[transactionId][owner];
}
```

The isConfirmedBy function checks if a transaction has been confirmed by a specific owner.

```
solidity
receive() external payable {}
```

The receive function allows the contract to receive Ether.

Deployment

To deploy our smart contract successfully, we need the celo extension wallet which can be downloaded from [here](#)

Next, we need to fund our newly created wallet which can be done using the celo alfojares faucet [Here](#)

You can now fund your wallet and deploy your contract using the celo plugin in remix.

Conclusion

In this tutorial, we created a MultiSigWallet contract written in Solidity. We have covered the state variables, modifiers, constructor, and functions in detail. This contract is an example of how multi-signature wallets can be implemented in decentralized applications to ensure secure and transparent management of funds.

Next Steps

I hope you learned a lot from this tutorial. Here are some relevant links that would aid your learning further.

- [Celo Docs](#)
- [Solidity Docs](#)

About the author

I'm Jonathan Iheme, A full stack block-chain Developer from Nigeria.

Thank You!!

```
# contract-addresses.md:
```

```
---
title: Contract Addresses
id: contract-addresses
---
```

```
import YouTube from '@components/YouTube';
import PageRef from '@components/PageRef';
```

Core contract address proxies and implementations for the Celo network.

```
---
```

```
:::tip
```

View Celo smart contracts here [contract addresses](#) by searching testnet or mainnet on [celoscan.io](#) or [explorer.celo.org](#).

:::

Celo Mainnet

jsx

celocli network:contracts --node https://forno.celo.org

Contract	Proxy
Accounts	0x7d21685C17607338b313a7174bAb6620baD0aaB7
Attestations	0xdC553892cdeeeD9f575aa0FBA099e5847fd88D20
FederatedAttestations	0x0aD5b1d0C25ecF6266Dd951403723B2687d6aff2
OdisPayments	0xae6b29f31b96e61dddc792f45fda4e4f0356d0cb
BlockchainParameters	0x6E10a8864C65434A721d82e424d727326F9d5Bfa
DoubleSigningSlasher	0x50C100baCDe7E2b546371EB0Be1eACcf0A6772ec
DowntimeSlasher	0x71CAc3B31c138F3327C6cA14f9a1c8d752463fDd
Election	0x8D6677192144292870907E3Fa8A5527fE55A7ff6
EpochRewards	0x07F007d389883622Ef8D4d347b3f78007f28d8b7
Escrow	0xf4Fa51472Ca8d72AF678975D9F8795A504E7ada5
Exchange	0x67316300f17f063085Ca8bCa4bd3f7a5a3C66275
ExchangeEUR	0xE383394B913d7302c49F794C7d3243c429d53D1d
FeeCurrencyWhitelist	0xBB024E9cdCB2f9E34d893630D19611B8A5381b3c
Freezer	0x47a472F45057A9d79d62C6427367016409f4fF5A
FeeHandler	0xcD437749E43A154C07F3553504c68fBfD56B8778
GasPriceMinimum	0xDfca3a8d7699D8bAfe656823AD60C17cb8270ECC
GoldToken	0x471EcE3750Da237f93B8E339c536989b8978a438
Governance	0xD533Ca259b330c7A88f74E000a3FaEa2d63B7972
GrandaMento	0x03f6842B82DD2C9276931A17dd23D73C16454a49
LockedGold	0x6cC083Aed9e3ebe302A6336dBC7c921C9f03349E
MentoFeeHandlerSeller	0x4eFa274B7e33476C961065000D58ee09F7921A74
Random	0x22a4aAF42A50bFA7238182460E32f15859c93dfe
Registry	0x00000000000000000000000000000000000000ce10
Reserve	0x9380fA34Fd9e4Fd14c06305fd7B6199089eD4eb9
SortedOracles	0xefB84935239dAcdecF7c5bA76d8dE40b077B7b33
StableToken	0x765DE816845861e75A25fCA122bb6898B8B1282a
StableTokenEUR	0xD8763CBa276a3738E6DE85b4b3bF5FDed6D6cA73
TransferWhitelist	0xb49E4d6F0B7f8d0440F75697E6c8b37E09178BCF
UniswapFeeHandlerSeller	0xD3aeE28548Dbb65DF03981f0dC0713BfCBd10a97
Validators	0xaEb865bCa93DdC8F47b8e29F40C5399cE34d0C58

Alfajores Testnet

jsx

celocli network:contracts --node https://alfajores-forno.celo-testnet.org

Contract	Proxy
Accounts	0xed7f51A34B4e71fbE69B3091FcF879cD14bD73A9
Attestations	0xAD5E5722427d79Dff28a4Ab30249729d1F8B4cc0
FederatedAttestations	0x70F9314aF173c246669cFb0EEe79F9Cfd9C34ee3
OdisPayments	0x645170cdB6B5c1bc80847bb728dBa56C50a20a49

BlockchainParameters	0xE5aCbb07b4Eed078e39D50F66bF0c80cF1b93abe	
CeloDistributionSchedule	0x78Af211Ad79bCE6BF636640CE8c2C2b29e02365A	
DoubleSigningSlasher	0x88A4c203C488E8277f583942672E1aF77e2B5040	
DowntimeSlasher	0xf2224c1d7b447D9A43a98CBD82FCCC0eF1c11CC5	
Election	0x1c3eDf937CFc2F6F51784D20DEB1af1F9a8655fA	
EpochRewards	0xB10Ee11244526b94879e1956745bA2E35AE2bA20	
Escrow	0xb07E10c5837c282209c6B9B3DE0eDBeF16319a37	
Exchange	0x17bc3304F94c85618c46d0888aA937148007bD3C	
ExchangeBRL	0xf391DcaF77360d39e566b93c8c0ceb7128fa1A08	
ExchangeEUR	0x997B494F17D3c49E66Fafb50F37A972d8Db9325B	
FeeCurrencyWhitelist	0xB8641365dBe943Bc2fb6977e6FBc1630EF47dB5a	
FeeCurrencyDirectory	0x71FFbD48E34bdD5a87c3c683E866dc63b8B2a685	
Freezer	0xfe0Ada6E9a7b782f55750428CC1d8428Cd83C3F1	
FeeHandler	0xEAAff71AB67B5d0eF34ba62Ea06Ac3d3E2dAAA38	
GasPriceMinimum	0xd0Bf87a5936ee17014a057143a494Dc5C5d51E5e	
GoldToken	0xF194afDf50B03e69Bd7D057c1Aa9e10c9954E4C9	
Governance	0xAA963FC97281d9632d96700aB62A4D1340F9a28a	
GrandaMento	0xEcf09FCD57b0C8b1FD3DE92D59E234b88938485B	
LockedGold	0x6a4CC5693DC5BFA3799C699F3B941bA2Cb00c341	
MentoFeeHandlerSeller	0xae83C63B5FB50C353c8d23397bcC9dBf3a9837Ac	
Random	0xdd318EEF001BB0867Cd5c134496D6cF5Aa32311F	
Registry	0x00ce10	
Reserve	0xa7ed835288Aa4524bB6C73DD23c0bF4315D9Fe3e	
SortedOracles	0xFdd8bD58115FfBf04e47411c1d228eCC45E93075	
StableToken	0x874069Fa1Eb16D44d622F2e0Ca25eeA172369bC1	
StableTokenBRL	0xE4D517785D091D3c54818832dB6094bcc2744545	
StableTokenEUR	0x10c892A6EC43a53E45D0B916B4b7D383B1b78C0F	
TransferWhitelist	0x52449A99e3455acB831C0D580dCDAC8B290d5182	
UniswapFeeHandlerSeller	0xc7b6E77C3702666DDa8EB5b7F30234B020788b21	
Validators	0x9acF2A99914E083aD0d610672E93d14b0736BCC	

overview.md:

title: Celo Platform

description: Overview of the Celo platform and it's relationship to the Ethereum blockchain.

id: overview

Celo Platform

Overview of the Celo platform and it's relationship to the Ethereum blockchain.

What is the Celo Platform?

Celo is a complete stack of new blockchain software, core libraries that run on that blockchain, and end-user applications including a Wallet app that communicate with that logic.

Blockchain

A blockchain or cryptographic network is a broad term used to describe a database maintained by a distributed set of computers that do not share a trust relationship or common ownership. This arrangement is referred to as decentralized. The content of a blockchain's database, or ledger, is authenticated using cryptographic techniques, preventing its contents from being added to, edited or removed except according to a protocol operated by the network as a whole.

The code of the Celo Blockchain has shared ancestry with Ethereum, blockchain software for building general-purpose decentralized applications. Celo differs from Ethereum in several important areas as described in the following sections. However, it inherits a number of key concepts.

Smart Contracts

Ethereum applications are built using smart contracts. Smart contracts are programs written in languages like Solidity that produce bytecode for the Ethereum Virtual Machine or EVM, a runtime environment. Programs encoded in smart contracts receive messages and manipulate the blockchain ledger and are termed on-chain.

Cryptocurrency

Celo has a native unit of accounting, the cryptocurrency CELO, comparable to Ether on Ethereum. Celo's ledger consists of accounts, identified by an address. There are two types of accounts. Externally owned accounts have an associated CELO balance and are controlled by a user holding the associated public-private keypair. Contract accounts contain the code and data of a single smart contract which can be called and manipulate its own stored data.

ERC-20 is a standard interface for implementing cryptocurrencies or tokens as contracts, rather than via account balances. For additional information on this, see Celo for Ethereum Developers. In Celo, CELO has a duality as both the native currency and an ERC-20 compliant token on the Celo blockchain.

:::warning

Celo assets exist on an independent blockchain, and cannot be accessed through wallets that connect to the Ethereum network. Only use wallets designed to work with the Celo network. Do not send your Celo assets to your Ethereum wallet or send your Ethereum assets to your Celo wallet.

:::

Transactions

Users interact with the blockchain by creating signed transactions. These are requests to make a change to the ledger.

Transactions can complete the following actions

- Transfer value between accounts
- Execute a function in a smart contract and pass in arguments
- Create a new smart contract

Blocks

The blockchain is updated by a protocol that takes the current state of the ledger, applies a number of transactions in turn, each of which may execute code and result in updates to the global state. This creates a new block that consists of a header, identifying the previous block and other metadata, and a data structure that describes the new state.

Transaction Fees

To avoid Denial-of-Service attacks and ensure termination of calls to smart contract code, the account sending a transaction pays transaction fees for its execution steps using its own balance. Transactions specify a maximum gas which bounds the steps of execution before a transaction is reverted. A gas price determines the unit price for each step, and is used to prioritize which transactions the network applies. \ (In Celo transaction fees can be paid in ERC-20 currencies and gas pricing works differently from Ethereum\).

Learn more

For a more in depth explanation of Ethereum, see the Ethereum White Paper or documentation.

```
# token-addresses.md:
```

```
---
```

```
title: Token Addresses
```

```
id: token-addresses
```

```
---
```

```
import YouTube from '@components/YouTube';
```

```
import PageRef from '@components/PageRef';
```

Token addresses for Celo assets on Mainnet, Alfajores Testnet, and Baklava Testnet.

```
---
```

Mainnet

- CELO: 0x471EcE3750Da237f93B8E339c536989b8978a438
- cUSD: 0x765de816845861e75a25fca122bb6898b8b1282a
- cEUR: 0xd8763cba276a3738e6de85b4b3bf5fded6d6ca73
- cREAL: 0xe8537a3d056DA446677B9E9d6c5dB704EaAb4787

Alfajores Testnet

```
- CELO: 0xF194afDf50B03e69Bd7D057c1Aa9e10c9954E4C9
- cUSD: 0x874069fa1eb16d44d622f2e0ca25eea172369bc1
- cEUR: 0x10c892a6ec43a53e45d0b916b4b7d383b1b78c0f
- cREAL: 0xE4D517785D091D3c54818832dB6094bcc2744545
```

Baklava

```
- CELO: 0xdDc9bE57f553fe75752D61606B94CBD7e0264eF8
- cUSD: 0x62492A644A588FD904270BeD06ad52B9abfEA1aE
- cEUR: 0xf9ecE301247aD2CE21894941830A2470f4E774ca
```

welcome.md:

```
title: Welcome to Celo
description: Celo's mission is to build a financial system that creates
the conditions for prosperity—for everyone.
id: welcome
slug: /
---
```

```
import YouTube from '@components/YouTube';
import PageRef from '@components/PageRef';
```

Celo's mission is to build a financial system that creates the conditions for prosperity—for everyone.

Getting Started

Celo is a mobile-first blockchain that makes decentralized financial (DeFi) tools and services accessible to anyone with a mobile phone. It aims to break down barriers by bringing the powerful benefits of DeFi to the users of the 6 billion smartphones in circulation today. Use this documentation as your guide into the Celo ecosystem!

:::tip Celo ❤ Feedback

If you have any ideas to improve the docs please make an issue, discuss in the forum, or become a contributor.

:::

Celo Basics

Learn the Celo Basics to get an overview of our mission, vision, whitepapers, and resources.

<!-- <LinkCardsWrapper>

<LinkCard title="Database" description="A dedicated, scalable Postgres database" />

```
<LinkCard title="Auth" description="User management with Row Level
Security" />
<LinkCard title="File Storage" description="Store, organize, and serve
large files" />
<LinkCard title="Auto-generated APIs" description="Instantly generate
APIs for your database" />
</LinkCardsWrapper> -->
```

- What is Celo?
- Showcase
- Explorer
- Website
- GitHub

Use Celo

Get up and running as a Celo holder, validator, developer, using tools in the Celo ecosystem.

- Developers
- Validators
- Integrations
- Bridges
- Oracles

Explore Celo Data

Explore Celo data to view the reserve, network, dapps, and additional analytics.

- Celo Analytics
- Celo Reserve
- DappLooker
- Celo Stats
- The Celo

APIs & SDKs

Connect to Celo using a variety of APIs and SDKs for iOS, Java, React, and more.

- CLI
- IOS
- React
- Flutter
- Javascript

Join the Community

Connect with the community to stay up to date on the latest news, events, and updates.

- @CeloDevs
- @CeloOrg

- GitHub

:::tip Get Support 🗨

Need help with anything related to Celo? Find Celo on Discord, Forum, or Telegram.

:::

add-celo-testnet-to-metamask.md:

title: Add Celo Testnet to MetaMask

Follow this quick guide to add Alfajores to your MetaMask wallet and start playing around with it. You can get funds from the Alfajores Faucet.

1. Open MetaMask

2. Go to settings

3. Select Networks

4. Add info about Celo L2 Alfajores Testnet

:::info

If you never added a new Network before then you need click on "Add a Network"

:::

Insert the following details:

- Network Name: Celo Alfajores Testnet
- New RPC URL: <https://alfajores-forno.celo-testnet.org>
- Chain ID: 44787
- Currency Symbol: CELO
- Block explorer URL: <https://celo-alfajores.blockscout.com/>

5. Click "Save"

build-on-socialconnect.md:

title: Introduction to SocialConnect

description: A beginner's guide to understanding and using SocialConnect

By the conclusion of this guide, you will have a basic understanding of SocialConnect and how to get started with it.

This document will cover:

- What is SocialConnect?
- Key Features
- Getting Started
- Further Reading

What is SocialConnect?

SocialConnect is an open-source protocol that maps off-chain personal identifiers (such as phone numbers, Twitter handles, etc.) to on-chain account addresses. This enables a convenient and interoperable user experience for various use cases, including:

- Payments: Send money directly to your friend's phone number.
- Social Discovery: Find someone's account based on their Twitter handle or other identity applications.

For a short demo of a payment from a Kaala wallet user to a Libera wallet user using only a phone number, check out the SocialConnect documentation.

Key Features

- Interoperability: Seamlessly map off-chain identifiers to on-chain addresses.
- Convenience: Simplify user interactions by using familiar identifiers like phone numbers and social media handles.
- Privacy: Ensure user privacy with features like phone number privacy and key hardening.

Getting Started

To start using SocialConnect, follow these steps:

1. Buy Odis Quota and Check Balance: Learn how to purchase Odis quota and check your balance.
2. Lookup Identifier: Find on-chain addresses using off-chain identifiers.
3. Register Identifier: Register new identifiers to map them to on-chain addresses.
4. Revoke Identifier: Remove mappings of identifiers when they are no longer needed.
5. Setup Issuer: Configure an issuer for managing identifiers.

For detailed guides and examples, refer to the SocialConnect documentation.

📖 Further Reading

To dive deeper into SocialConnect, explore the following sections in the documentation:

- Introduction to SocialConnect: Read more
- Guides: Step-by-step instructions for various tasks.
- Protocol: Detailed information about the protocol's architecture and components.
- Examples: Practical examples, including a Next.js example.
- Contracts: Information about the smart contracts used in SocialConnect.

index.md:

title: Building dApps on Celo

description: A guide for building on Celo.

import ColoredText from '/src/components/ColoredText';

Celo was created to enable real-world use cases on Ethereum.

Whether you're building your first dApp or looking to integrate an existing protocol onto Celo, we have all the resources and tools you need to get started.

Why Build on Celo?

- EVM Compatible: Celo is fully EVM-compatible, offering the same development experience as Ethereum with improved scalability and lower costs.
- Fast Transactions: After the migrations to an L2 Celo now has a 1 second block finality compared to formerly 5 seconds.
- Easy, Low-Cost Payments: Celo's seamless payment infrastructure, including Fee Abstraction, sub-cent fees, and native stablecoins, enables simple and affordable transactions.
- Global Reach: Celo supports 1,000+ projects in 150+ countries, providing a vibrant, global community that helps developers build, test, and scale their applications to millions of everyday users.

Getting Started

- [Quickstart with Celo Composer CLI](/build/quickstart)
- [Deploy a smart contract on Celo](/developer/dev-environments/overview)
- [Receive testnet funds](https://faucet.celo.org/alfajores)

- [Explore developer tooling](/developer)

quickstart.md:

title: Quickstart

To test out deploying a dApp on Celo, we recommend using Celo Composer, which allows you to quickly build, deploy, and iterate on decentralized applications using Celo. It provides a number of frameworks, examples, templates and Celo specific functionality to help you get started with your next dApp.

Prerequisites

- Node (v20 or higher)
- Git (v2.38 or higher)

How to use Celo Composer

The easiest way to start with Celo Composer is using @celo/celo-composer. This CLI tool lets you quickly start building dApps on Celo for multiple frameworks, including React (and rainbowkit). To get started, just run the following command, and follow the steps:

```
bash
npx @celo/celo-composer@latest create
```

Provide the Project Name:

You will be prompted to enter the name of your project

```
bash
What is your project name:
```

Choose a smart contract development environment:

You will be asked if you want to use Hardhat. Select Yes or No

```
bash
Do you want to use Hardhat? (Y/n)
```

Choose to Use a Pre-Built Template, highlighting Celo's unique features:

You will be asked if you want to use a template, check below for the options. Select Yes or No

```
bash
```


Do you want to use a template?

Select a Template:

If you chose to use a template, you will be prompted to select a template from the list provided

```
bash
built in frontend logic to use your dapp in MiniPay, pre-built example
functions for sign, transact and mint
- MiniPay
template built for easy Valora connectivity
- Valora
example project for matching your social Identifier to your wallet
address, using GitHub
- Social Connect
```

Provide the Project Owner's Name:

You will be asked to enter the project owner's name

```
bash
Project Owner name:
```

Wait for Project Creation:

The CLI will now create the project based on your inputs. This may take a few minutes.

Follow the instructions to start the project.

The same will be displayed on the console after the project is created

```
bash
🚀 Your starter project has been successfully created!
```

Before you start the project, please follow these steps:

1. Rename the file:
packages/react-app/.env.template
to
packages/react-app/.env
2. Open the newly renamed .env file and add all the required environment variables.

Once you've done that, you're all set to start your project!

Run the following commands from the packages/react-app folder to start the project:

```
yarn install
yarn dev
```

If you prefer npm, you can run:

```
npm install
npm run dev
```

Thank you for using Celo Composer! If you have any questions or need further assistance, please refer to the README or reach out to our team.

🔥Voila, you have a dApp ready to go. Start building your dApp on Celo.

Once your custom dApp has been created, just install dependencies, either with yarn or npm i, and run the respective script from the package.json file.

Supported Templates

MiniPay

- Pre-built template for creating a mini-payment application.
- Seamless integration with Celo blockchain for handling payments.

Checkout minipay docs to learn more about it.

Valora

- Template designed for Valora wallet integration.
- Facilitates easy wallet connectivity and transaction management.

Checkout valora docs to learn more about it.

Social Connect

- Template for building applications with social connectivity features.
- Supports various social login methods and user interactions.

Checkout social connect docs to learn more about it.

Support

Join the Celo Discord server. Reach out in the #general-dev channel with your questions and feedback.

support.md:

title: Support

description: A comprehensive guide on how to reach out for support

```
import ColoredText from '/src/components/ColoredText';
```

Support

This section explains how you can get support and stay up-to-date with Celo developer news.

:::tip

Help us improve by sharing your feedback on missing tools, unclear sections, or any challenges you encounter by creating an issue or by clicking on "Edit Page" on the bottom of each page.

:::

How to Get Support?

- For general questions, join our <ColoredText>Discord</ColoredText> and claim the developer role to ask questions in our #general-dev channel. Our moderators can provide guidance and answers.

- For advanced questions, create issues and participate in <ColoredText>GitHub Discussions</ColoredText>.

For the latest updates, sign up for the <ColoredText>Celo Signal Mailing List</ColoredText> and follow CeloDevs on <ColoredText>Twitter</ColoredText>.

add-cel2-testnet-to-metamask.md:

title: Add Cel2 Testnet to Metamask

description: Adding Cel2 testnet to your wallet

Follow this quick guide to add Alfajores to your MetaMask wallet and start playing around with it. You can get funds from the Alfajores Faucet.

1. Open Metamask

2. Go to settings

3. Select Networks

4. Add info about Celo L2 Alfajores Testnet

:::info

If you never added a new Network before then you need click on "Add a Network"
:::

Insert the following details:

- Network Name: Celo Alfajores Testnet
- New RPC URL: <https://alfajores-forno.celo-testnet.org>
- Chain ID: 44787
- Currency Symbol: CELO
- Block explorer URL: <https://explorer.celo.org/alfajores>

5. Click "Save"

cel2-architecture.md:

title: Architecture

Celo's architecture is a multi-layered system that includes a Layer 2 blockchain, core smart contracts, user applications, and a dynamic network topology, all optimized for scalability, security, and ease of use.

Introduction to the Celo Stack

The Celo stack consists of three main components that work together to deliver a seamless blockchain experience:

Celo Blockchain

The Celo blockchain operates as a Layer 2 (L2) solution using the OP Stack, with distinct layers for optimal performance and security:

- Execution Layer: EVM-compatible, allowing easy deployment of Ethereum smart contracts.
- Data Availability Layer: Utilizes EigenDA to ensure transaction data is accessible and cost-efficient.
- Settlement Layer: Leverages Ethereum to finalize transactions, benefiting from its security.

Celo Core Contracts

Celo Core Contracts are essential smart contracts on the Celo blockchain, managed through decentralized governance. Key contracts include:

- Attestations: Links users' phone numbers to their blockchain addresses for secure identity verification and Social Connect features.

- Governance: Allows community voting on protocol upgrades and changes.
- StableToken (e.g., cUSD, cEUR): Manages native stablecoin issuance and stability for seamless transactions.
- Exchange: Facilitates asset trading and liquidity within the Celo ecosystem.
- SortedOracles: Provides external data, like price feeds, critical for stability and DeFi applications.
- Validators: Manages validator operations and network security.

Applications

The Application Layer connects users directly to the blockchain. Developers can build user-friendly applications that are secure, transparent, and decentralized by utilizing Celo's blockchain.

Our Network Topology

The Celo network topology consists of various nodes running the Celo blockchain software in different configurations to support the decentralized infrastructure of the network.

Sequencers

Sequencers replace the traditional validator role in the L2 architecture. They are responsible for:

- Gathering transactions from other nodes
- Executing associated smart contracts to form new blocks
- Submitting these blocks to the Ethereum L1 for final settlement

Sequencers operate on a faster 2-second block time, improving transaction speed and throughput.

Full Nodes

Full nodes in the Celo L2 network serve multiple important functions:

- Relaying transactions and responding to queries from light clients
- Maintaining a copy of the L2 blockchain state
- Interacting with Ethereum L1 to read and verify L2 block data
- Optionally running an Ethereum node or using a third-party Ethereum node service

Full nodes can join or leave the network at any time, providing a decentralized infrastructure for the network.

Light Clients

Light clients, such as those running on mobile apps with limited data, continue to play a crucial role:

- Connecting to full nodes to request account and transaction data
- Signing and submitting new transactions
- Operating without maintaining a full copy of the blockchain state

Light clients benefit from the improved speed and lower costs of the L2 architecture while maintaining a similar user experience.

Data Availability Layer

Celo L2 incorporates EigenDA as its Data Availability (DA) layer:

- Ensures transaction data remains accessible and cost-efficient
- Operates separately from the execution and settlement layers
- Contributes to lower transaction costs and improved scalability

contract-addresses.md:

title: Contract Addresses

Holesky (L1)

| Contract Name | Address |
|-----------------------------|--|
| AddressManager | 0x26E35ab72a8a68fc116216145129989EaCc5c12a |
| AnchorStateRegistry | 0xEB4763277b4b92CF1DD3Fa8432fC3C21037c8253 |
| AnchorStateRegistryProxy | 0xedf4ed5230f9B1F54C33F381C1990129f30E9196 |
| CustomGasToken | 0xE11Dc89d9Fae9d5B6156F6B3b614bfda45D21e27 |
| CustomGasTokenProxy | 0xDEd08f6Ec0A57cE6Be62d1876d2CE92AF37edda0 |
| DelayedWETH | 0x916348362fcCdf3D548617f2ff8fC9fEBE44389b |
| DelayedWETHProxy | 0xb9034e29a6e5c94ae50F4eDBf8450492Bf6f636C |
| DisputeGameFactory | 0xB0C46509E24a0745d201114016fD666D6D1E3f8e |
| DisputeGameFactoryProxy | 0x831f39053688f05698ad0fB5f4DE7e56B2949c55 |
| L1CrossDomainMessenger | 0xd51013941d49631846fe52028C8D7a6EeBf06C98 |
| L1CrossDomainMessengerProxy | 0x0be65c09d5880143cb9C2D6E45474972eFC4C13B |
| L1ERC721Bridge | 0x7Fc64181a3629450C739d09131FFcE1Aae35176C |
| L1ERC721BridgeProxy | 0x856dD9501a509D3738aE2da7B79cDA1731d8E9b3 |
| L1StandardBridge | 0x65E8f629B13535f902020668Fe73aEc24e52F5D8 |

[illegible]

| | | | | |
|--|-------------------------|--|--|--|
| | StableToken | | 0x874069Fa1Eb16D44d622F2e0Ca25eeA172369bC1 | |
| | StableTokenBRL | | 0xE4D517785D091D3c54818832dB6094bcc2744545 | |
| | StableTokenEUR | | 0x10c892A6EC43a53E45D0B916B4b7D383B1b78C0F | |
| | TransferWhitelist | | 0x52449A99e3455acB831C0D580dCDac8B290d5182 | |
| | UniswapFeeHandlerSeller | | 0xc7b6E77C3702666DDa8EB5b7F30234B020788b21 | |
| | Validators | | 0x9acF2A99914E083aD0d610672E93d14b0736BBcc | |

decision-tree.md:

title: L2 Migration Decision Tree

description: Not sure what you need to do to migrate from Celo L1 to Celo L2?

<details>

<summary>I run a node</summary>

Follow the node upgrade instructions.

</details>

<details>

<summary>I use or need a node provider</summary>

The following node providers are available for Alfajores L2:

- Infura
- dRPC

If none of the above work for you, you can use cLabs best effort hosted node: Forno.

</details>

<details>

<summary>I use ContractKit or Fee Abstraction (ERC-20 gas tokens)</summary>

Required versions for SDK and libraries:

- @celo/connect >=6.1.0
- @celo/contractkit >=8.1.1
- @celo/celocli >=5.1.1
- @viem >=2

<details>

<summary>I need more information on ContractKit...</summary>

Have a look at the ContractKit Cel2 Guide

</details>

</details>

<details>

<summary>I need help!</summary>

Reach out to us!

- Technical question? celo-org discussions on GitHub
- Any type of question? Reach out on Discord in the #celo-L2-support channel

Forum:

Also check out these resources:

- Celo Docs
- Celo L2 Specifications
- Transaction types on Celo
- Celo Forum

</details>

deploy-contract.md:

title: Deploy Contract to Cel2 Alfajores Testnet

Deploying contract to Cel2 Alfajores Testnet is the same as deploying on any EVM chain, the only change you need to make is to the RPC endpoint.

In this tutorial, we will deploy the Counter.sol contract to Cel2 Alfajores Testnet using Foundry.

1. Initialize your project

```
bash
forge init
```

2. In the src folder you should already have a contract named Counter.sol which will deploy on the Cel2 Alfajores Testnet.

```
solidity
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

contract Counter {
    uint256 public number;

    function setNumber(uint256 newNumber) public {
        number = newNumber;
    }

    function increment() public {
```

```
        number++;
    }
}
```

3. Use the below command to deploy the contract on Cel2 Alfajores Testnet.

```
:::info
You will need Celo (not ETH) in your account to deploy the contract.
:::
```

```
bash
forge create src/Counter.sol:Counter --rpc-url https://alfajores-
forno.celo-testnet.org --private-key [PRIVATEKEY]
```

On successful deployment, you should see a similar output in your terminal.

```
!deployment-success
```

```
# faq.md:
```

```
---
title: Cel2 FAQ
description: Frequently Asked Questions about Cel2
---
```

How do I run a node or upgrade an existing node?

See the L2 Migration Guide.

For Alfajores, here are the related assets:

- Full migrated chaindata
- Rollup deploy config
- L1 contract addresses
- L2 allocs
- rollup.json
- Genesis used for snap syncing
- Container images:
 - op-geth
 - op-node
 - eigenda-proxy
- p2p peers:
 - op-geth bootnode/peers, to be used with op-geth --bootnodes flag:

```
text
```

```
enode://ac0f42fa46f8cc10bd02a103894d71d495537465133e7c442bc02dc76721a5f41
761cc2d8c69e7ba1b33e14e28f516436864d3e0836e2dcdaef032387f72447dd@34.83.164
.192:30303
```

enode://596002969b8b269a4fa34b4709b9600b64201e7d02e2f5f1350affd021b0cbda6ce2b913ebe24f0fb1edcf66b6c730a8a3b02cd940f4de995f73d3b290a0fc92@34.82.177.77:30303

enode://3619455064ef1ce667171bba1df80cfd4c097f018cf0205aaad496f0d509611b7c40396893d9e490ee390cd098888279e177a4d9bb09c58387bb0a6031d237f1@34.19.90.27:30303

enode://e3c54db6004a92d4ee87504f073f3234a25759b485274cc224037e3e5ee792f3b482c3f4fffcfb764af6e1859a1aea9710b71e1991e32c1dee7f40352124bb182@35.233.249.87:30303

enode://674410b34fd54c8406a4f945292b96111688d4bab49aecdc34b4f1b346891f4673dcb03ed44c38ab467ef7bec0b20f6031ad88aa1d35ce1333b343d00fa19fb1@34.168.43.76:30303

- op-node static peers, to be used with op-node --p2p.static flag:

text

/ip4/35.197.25.52/tcp/9222/p2p/16Uiu2HAmQEEdyLRSaVZDr5Sqbj1RnKmNDhtQJcEKmemrVxe4FxKwR

/ip4/34.105.22.4/tcp/9222/p2p/16Uiu2HAm1SZBDSugT5MMu7vBY8auDgfZFNhoDeXPLc9Me5FsAxwT

/ip4/34.83.209.168/tcp/9222/p2p/16Uiu2HAmGJAiUX6HLSo4nLh8T984qxxokwL23cVsYuNZy2SrK7C6

/ip4/34.83.214.149/tcp/9222/p2p/16Uiu2HAmAko2Kr3eAjM7tnshtEhYrxQYfKUvN2kwiyygeFoBAoi8S

/ip4/34.169.5.52/tcp/9222/p2p/16Uiu2HAmKc6YKHZygsjBDaj36uAufxpgZFgrzDqVBt6zTPwdhhJD

Can I use an RPC endpoint with Alfajores?

- Ethereum JSON-RPC endpoint: <https://alfajores-forno.celo-testnet.org> (op-geth kind)
- OP RPC endpoint: <https://op.alfajores.celo-testnet.org/> (op-node kind)

Is there an Alfajores faucet? Where? How do I get funds?

Faucet: <https://faucet.celo.org/alfajores>

Is there an Alfajores explorer?

Blockscout Explorer: <https://celo-alfajores.blockscout.com/>

How can I use the native bridge with Alfajores?

Through the Superbridge UI: <https://superbridge.app/celo-testnet> or <https://testnets.superbridge.app/celo-testnet>.

What's the difference between Dango and Alfajores?

Alfajores is the first testnet of the Celo L1 blockchain, launched in July 2019. It was upgraded to L2 in September 2024. For more details, refer to the Alfajores Testnet documentation.

Dango was the first testnet of the Celo L2 blockchain, launched in July 2024 and shut down in October 2024. It served as a test run for the Alfajores migration to L2. Dango diverged from Alfajores L1 at block 24940100.

Is there anything that used to work on Alfajores that doesn't anymore?

See L1→L2 Migration Changes.

Celo L2 setup

How is the Celo L2 different to Optimism?

See Celo L2 Specification.

What are the costs for L1 data and how are they paid?

For the testnet L1 data fees are covered by us.

I saw EigenDA mentioned, is it used?

Yes! See EigenDA.

What's the block time?

The block period is 1 second.

What's the throughput?

The gas limit per block is 30 million. The Alfajores testnet has a throughput of 30M gas/s.

For more details, see the Alfajores specifications.

What happened to these features?

- CELO token duality? Supported, see Token Duality
- Fee currencies? Supported, see Fee Abstraction
- Epoch rewards? Supported in Alfajores and Mainnet

Not sure what you need to do?

Check out the L2 Migration decision tree.

fee-currencies.md:

title: Fee Abstraction on Cel2

Fee Abstraction Addresses

Celo allows paying gas fees in currencies other than the native currency. The tokens that can be used to pay gas fees are controlled via governance and the list of tokens allowed is maintained in `FeeCurrencyWhitelist.sol` contract.

Alternate fee currency works with EOAs, no paymaster is required!

Get a list of whitelisted Fee Currencies

```
bash
celocli network:whitelist --node https://alfajores-forno.celo-testnet.org
```

Using Fee Abstraction with Celo CLI

Transfer 1 USDC using USDC as fee currency, with the `celocli` using the `--gasCurrency` flag

```
bash
celocli transfer:erc20 --erc20Address
0x2F25deB3848C207fc8E0c34035B3Ba7fC157602B --from
0x22ae7Cf4cD59773f058B685a7e6B7E0984C54966 --to
0xDF7d8B197EB130cF68809730b0D41999A830c4d7 --value 1000000 --gasCurrency
0x4822e58de6f5e485eF90df51C41CE01721331dC0 --privateKey [PRIVATEKEY]
```

| Symbol | Adapter | Token |
|--------|--|-------|
| cUSD | 0x874069Fa1Eb16D44d622F2e0Ca25eeA172369bC1 | |
| cEUR | 0x10c892A6EC43a53E45D0B916B4b7D383B1b78C0F | |
| cREAL | 0xE4D517785D091D3c54818832dB6094bcc2744545 | |
| cKES | 0x1E0433C1769271ECcF4CFF9FDdD515eefE6CdF92 | |
| USDC | 0x2F25deB3848C207fc8E0c34035B3Ba7fC157602B | |
| USD | 0x4822e58de6f5e485eF90df51C41CE01721331dC0 | |
| G\$ | 0x03d3daB843e6c03b3d271eff9178e6A96c28D25f | |

Using Fee Abstraction with Programmatically

You can use Fee Abstraction by specifying a token/adaptor address as a value for the `feeCurrency` property in the transaction object. The `feeCurrency` property in the transaction object is exclusive to Celo and allows paying gas fees using assets other than the native currency of the network.

:::info

Wallets will overwrite the `feeCurrency`, which is why this is recommended for wallet developers or backend developers.

:::

For using Fee Abstraction for a wallet or inside your dApp we recommend using `viem` or `wagmi`.

:::info

While we recommend `viem`, `web3.js` has added as of 4.13.1 support for `feeCurrency` via the usage of plugins. There is a celo-specific plugin for `web3@4` available on github.

:::warning

Currently, `ethers.js` doesn't support the `feeCurrency` property

:::

Sending transaction using Fee Abstraction

Sending transactions with fee currency other than the native currency of the network is pretty straightforward. All you need to do is set the `feeCurrency` property in the transaction object with the address of the token/adaptor you want to use to pay for gas fees.

The below code snippet demonstrates transferring 1 USDC using USDC as gas currency.

```
js
import { createWalletClient, http } from "viem";
import { celo } from "viem/chains";
import { privateKeyToAccount } from "viem/accounts";
import { stableTokenAbi } from "@celo/abis";

// Creating account from private key, you can choose to do it any other
way.
const account = privateKeyToAccount("0x432c...");

// WalletClient can perform transactions.
const client = createWalletClient({
  account,

  // Passing chain is how viem knows to try serializing tx as cip42.
  chain: celo,
  transport: http(),
});

const USDCADAPTERMAINNET = "0x2F25deB3848C207fc8E0c34035B3Ba7fC157602B";
```

```

const USDCMAINNET = "0xcebA9300f2b948710d2653dD7B07f33A8B32118C";

/
  The UI of the wallet should calculate the transaction fees, show it and
  consider the amount to not be part of the asset that the user i.e the
  amount corresponding to transaction fees should not be transferrable.
/
async function calculateTransactionFeesInUSDC(transaction) {
  // Implementation of getGasPriceInUSDC is in the above code snippet
  const gasPriceInUSDC = await getGasPriceInUSDC();

  // Implementation of estimateGasInUSDC is in the above code snippet
  const estimatedGasPrice = await estimateGasInUSDC(transaction);

  return gasPriceInUSDC  estimatedGasPrice;
}

async function send(amountInWei) {
  const to = USDCMAINNET;

  // Data to perform an ERC20 transfer
  const data = encodeFunctionData({
    abi: stableTokenAbi,
    functionName: "transfer",
    args: [
      "0xccc9576F841de93Cd32bEe7B98fE8B9BD3070e3D",
      // Different tokens can have different decimals, cUSD (18), USDC
(6)    amountInWei,
    ],
  });

  const transactionFee = await calculateTransactionFeesInUSDC({ to, data
});

  const tokenReceivedbyReceiver = parseEther("1") - transactionFee;

  /
    Now the data has to be encode again but with different transfer value
    because the receiver receives the amount minus the transaction fee.
  /
  const dataAfterFeeCalculation = encodeFunctionData({
    abi: stableTokenAbi,
    functionName: "transfer",
    args: [
      "0xccc9576F841de93Cd32bEe7B98fE8B9BD3070e3D",
      // Different tokens can have different decimals, cUSD (18), USDC
(6)    tokenReceivedbyReceiver,
    ],
  });

  // Transaction hash
  const hash = await client.sendTransaction({

```



```

        ...{ to, data: dataAfterFeeCalculation },
    /
    In case the transaction request does not include the feeCurrency
    property, the wallet can add it or change it to a different currency
    based on the user balance.

    Notice that we use the USDCADAPTERMAINNET address not the token
    address this is because at the protocol level only 18 decimals tokens are
    supported, but USDC is 6 decimals, the adapter acts a unit converter.
    /
    feeCurrency: USDCADAPTERMAINNET,
  });

  return hash;
}

```

index.md:

```

---
title: Overview
description: Overview
---

```

Celo is transitioning from a standalone EVM-compatible Layer 1 blockchain to an Ethereum Layer 2. This shift, proposed by cLabs in July 2023, aims to maintain the seamless user experience that Celo is known for—characterized by speed, low costs, and ease of use—while leveraging Ethereum’s security and ecosystem. As part of this transition, Celo is currently operating a Layer 2 testnet, Alfajores, which launched on September 26, 2024.

```

---
:::warning
While most applications should remain unaffected, node operators,
validators, and RPC providers must ensure their systems are prepared for
the transition to maintain seamless operations.
:::

```

What does this mean for our ecosystem?

Celo's evolution from an L1 EVM-compatible chain to an L2 solution marks a significant milestone in our ongoing relationship with the Ethereum ecosystem. As an L1 chain, Celo has always maintained close ties with Ethereum, sharing its commitment to decentralization, security, and innovation. By transitioning to an L2, Celo strengthens this bond, allowing our developers and protocols to immerse themselves even deeper into the vibrant, collaborative Ethereum community. This integration

enhances opportunities for open-source contributions, joint initiatives, and the development of public goods, ensuring that Celo's impact resonates widely across the blockchain space.

Technical Changes

From a technical standpoint, this shift brings substantial benefits. Native bridging between Celo and Ethereum, which was previously not possible, will now be a reality. This advancement significantly enhances the security of token transfers by reducing reliance on external bridges, which have often been a point of vulnerability. With native bridging, Celo can offer a more secure and streamlined experience for users, ensuring that transactions within our ecosystem are both safe and reliable. In essence, becoming an L2 not only aligns Celo more closely with Ethereum's expansive network but also empowers our community to innovate with greater confidence and reach.

Important Dates

:::note

These dates are subject to change. This documentation will be continuously updated as new dates become available.

:::

Early July, 2024: Dango L2 Testnet Launch

- The Dango Testnet announced on the 7th of July 2024, Celo's first L2 public test network, went live. Dango allowed developers and infrastructure providers to familiarize themselves with the L2 environment. It was shut down on the 9th of October 2024.

26th September, 2024: Alfajores L2 Testnet Launch

- The Celo L2 testnet, Alfajores, went live! This provides a testing environment for node operators and developers to ensure compatibility before the mainnet launch.

October 2024: Code Freeze and Audits

- The core dev team will freeze all feature development by October 14th. All further development will focus on bug fixes and partner support.

12th December 2024: Baklava L2 Launch

- Using the final audited release, the Celo validator community will perform a dry run of the L2 upgrade on the Baklava network.

Mid-January 2025: Celo L2 Mainnet Launch

- Following a successful Baklava upgrade, the Celo L2 mainnet will officially launch. All nodes must be updated by this time to avoid disruption. Further announcements will confirm the exact date.

Useful Links

- Layer 2 Specification
- L2 Migration Guide
- What's Changed?

- Cel2 Code
- FAQ

l2-operator-guide.md:

title: Celo L2 Migration Guide

description: How to migrate Celo nodes from L1 to L2

This guide is designed to help full node operators migrate from the Celo L1 to the Celo L2 client for the upcoming Layer 2 upgrades. Please note that the Celo mainnet instructions will be finalized after the Alfajores testnet upgrade occurs.

Alfajores migration overview

In the Celo L1 to L2 transition, we are migrating all historical Celo data into the Celo L2 node, ensuring that blocks, transactions, logs, and receipts are fully accessible within the Celo L2 environment.

:::info

The Alfajores network has migrated on block 26384000.

:::

Sometime before the transition, all Alfajores node operators must upgrade their existing nodes to the latest version and add a `--l2migrationblock` flag when restarting (see below). All Alfajores nodes that do this will stop adding blocks immediately before the specified block number.

When the block before `--l2migrationblock` is reached, node operators can start their L2 Alfajores nodes.

Those who do not need a full sync can start a Celo L2 node and use snap sync right away.

Those who do need a full sync have two options: download and use the migrated chaindata provided by cLabs or run a migration script on their own chaindata to convert it into a format compatible with the Celo L2 node.

To simplify the management of the Celo L2 node, no legacy execution logic is included in Celo L2.

However, RPC calls that require execution or state for pre-transition L1 blocks remain supported by proxying these requests from the Celo L2 node to a legacy Celo L1 node.

Therefore, operators looking to run full archive nodes or serve requests for historical state / execution now need to run both a Celo L1 node and a Celo L2 node. The Celo L2 node can be configured to redirect requests for historical state / execution to the Celo L1 node (see [Configuring a HistoricalRPCService](#)).

Since the Celo L2 node does still require the full pre-migration chain data, these operators will require approximately double the storage space as is currently needed.

Alfajores deployment resources

Here you can find the resources used for the Alfajores migration. Some of these are generated during the migration process if you decide to migrate your own chain data.

- Rollup deployment config
- L1 Contract Addresses
- L2 Allocs
- Genesis file
- Rollup config
- Migrated snapshot at migration block 26,384,000

Stopping an L1 node

If you're currently running an L1 Alfajores node, you can upgrade your Celo blockchain client to the v1.8.5 release before the migration, and include the `--l2migrationblock=26384000` flag when restarting. While this step is not mandatory for full nodes—since the network will stall if a quorum of elected validators has the flag set—it is recommended as a practice run for the upcoming Baklava and Mainnet migrations.

This will automatically prevent your node from processing blocks higher than `l2migrationblock - 1` (i.e.: 26383999).

Starting an L2 node

Node operators who wish to run an L2 node have three options, ranked by ease and the level of trust required:

1. Start an L2 node with snap sync. This option does not require running the migration script.
2. Start an L2 node with the provided migrated chaindata.
3. Migrate the L1 chaindata manually.

Snap sync provides a simpler and faster setup experience. However, it is not suitable if you plan to run an archive node. In that case, you'll need to either use an archive node snapshot or migrate your own archive data from an L1 node. While both options follow a similar process, there are some differences, particularly in how you prepare the chaindata from the L1. Nonetheless, the majority of the service configuration remains the same across all options.

:::info

If you plan to migrate your own chaindata, we recommend getting familiar with the instructions and running the pre-migration 1-3 days ahead of the full migration.

:::

To set up your node, you'll need to run three services: op-node, op-geth, and eigenda-proxy. op-node serves as the consensus client for the L2 node, op-geth as the execution client, and eigenda-proxy acts as the interface between your L2 node and the data availability layer EigenDA.

If you plan to run multiple L2 nodes, you'll need separate instances of op-node and op-geth for each node, but you can share a single instance of eigenda-proxy. Additionally, you'll need an endpoint for the L1 RPC, which can either be a public node or your own L1 node. For Alfajores, the L1 used is Holesky.

- Running EigenDA Proxy
- Running op-geth
- Running op-node

Running EigenDA Proxy

:::info

We have deployed a public EigenDA proxy for Alfajores at <https://eigenda-proxy.alfajores.celo-testnet.org>. This instance has caching enabled, so all Alfajores blobs will be available for download, even if they have expired from EigenDA. You can configure your nodes to consume from this instance. Beware that for Mainnet, we are not planning to host such a proxy.

:::

:::info

These are brief instructions for running an eigenda-proxy instance. For more detailed instructions, please refer to the repository README.

:::

If you are using Kubernetes for this deployment, you can utilize our eigenda-proxy helm chart to simplify the process. Feel free to modify these instructions to better suit your specific needs.

1. You can use the official Docker image for the EigenDA Proxy. At the time of writing, we are using version 1.4.1 ([us-west1-docker.pkg.dev/devopsre/celo-blockchain-public/eigenda-proxy:v1.4.1](https://docker.pkg.dev/devopsre/celo-blockchain-public/eigenda-proxy/v1.4.1)). Alternatively, you can build it from source:

```
bash
git clone https://github.com/Layr-Labs/eigenda-proxy.git
cd eigenda-proxy
git checkout v1.2.0
make
```

Binary available at `./bin/eigenda-proxy`

2. You will need to download two files required for KZG verification. At the time of writing, these files are approximately 8GB in size, so please ensure you have enough space in the download directory. For example:

```
bash
```

```
EIGENDAKZGPROXYDIR=${HOME}/alfajores-l2/eigenda-proxy/kzg
mkdir -p ${EIGENDAKZGPROXYDIR}
```

```
[ ! -f ${EIGENDAKZGPROXYDIR}/g1.point ] && wget https://srs-
mainnet.s3.amazonaws.com/kzg/g1.point --output-
document=${EIGENDAKZGPROXYDIR}/g1.point
[ ! -f ${EIGENDAKZGPROXYDIR}/g2.point.powerOf2 ] && wget https://srs-
mainnet.s3.amazonaws.com/kzg/g2.point.powerOf2 --output-
document=${EIGENDAKZGPROXYDIR}/g2.point.powerOf2
```

```
wget https://raw.githubusercontent.com/Layr-Labs/eigenda-operator-
setup/master/resources/srsha256sums.txt --output-
document=${EIGENDAKZGPROXYDIR}/srsha256sums.txt
if (cd ${EIGENDAKZGPROXYDIR} && sha256sum -c srsha256sums.txt); then
    echo "Checksums match. Verification successful."
else
    echo "Error: Checksums do not match. Please delete this folder and try
again."
fi
```

3. Finally, we can run eigenda-proxy. Feel free to modify the --eigenda-eth-rpc flag to point to your own node or your preference:

```
bash
EIGENDAIMAGE=us-west1-docker.pkg.dev/devopsre/celo-blockchain-
public/eigenda-proxy:v1.4.1

docker run --platform linux/amd64 -d \
    --name eigenda-proxy \
    -v ${EIGENDAKZGPROXYDIR}:/data \
    --network=host \
    ${EIGENDAIMAGE} \
    /app/eigenda-proxy \
    --addr=0.0.0.0 \
    --port=4242 \
    --eigenda-disperser-rpc=disperser-holesky.eigenda.xyz:443 \
    --eigenda-eth-rpc=https://ethereum-holesky-rpc.publicnode.com \
    --eigenda-signer-private-key-hex=$(head -c 32 /dev/urandom | xxd -p
-c 32) \
    --eigenda-svc-manager-
addr=0xD4A7E1Bd8015057293f0D0A557088c286942e84b \
    --eigenda-status-query-timeout=45m \
    --eigenda-g1-path=/data/g1.point \
    --eigenda-g2-tau-path=/data/g2.point.powerOf2 \
    --eigenda-disable-tls=false \
    --eigenda-eth-confirmation-depth=1 \
    --eigenda-max-blob-length=300MiB
```

--eigenda-signer-private-key-hex is the proxy hex-encoded private key (without 0x). This account does not need to be associated with any existing ethereum account or holds any funds. You can create this key using your preferred tool. The key is used by the proxy solely for

dispersing data to eigenda, but since this proxy will only be retrieving data from eigenda, the key is not important, however in order to start the proxy, a key is still required.

You can check the logs running `docker logs -f eigenda-proxy` to ensure the service is running correctly.

Running op-geth

Now, let's move on to the op-geth execution client. It will be responsible for executing transactions and generating blocks. We recommend running op-geth on a machine with 8 modern cores (amd64 or arm64), at least 8GB of memory and 200GB of storage. Feel free to adjust these values based on your specific requirements.

:::info

You can use the official op-geth documentation as an additional reference.

:::

Although there are multiple ways to run op-geth, all options will share most of the same configuration. Depending on the option you choose, you may need to execute some steps before running op-geth to prepare the chaindata:

- Option 1: Snap sync
- Option 2: Download L2 chaindata
- Option 3: L1 chaindata migration

Option 1: Snap sync

With snap sync, you can start an L2 node without migrating or downloading the L1 chaindata. It is the easiest way to get started with an L2 node, but it does not support archive nodes. To start an L2 node with snap sync, you need to run op-geth with the `--syncmode=snap` flag.

Also, if you are using snap sync, you will need to init the chaindata dir using the provided genesis file. You should not init if you are starting your node with a migrated datadir. Run using the container or the binary according to your preference. You can use the following example as a reference.

```
bash
OPGETHDATADIR=${HOME}/alfajores-l2/op-geth-datadir
OPGETHIMAGE=us-west1-docker.pkg.dev/devopsre/celo-blockchain-public/op-geth:celo8
mkdir -p ${OPGETHDATADIR}
wget https://storage.googleapis.com/cel2-rollup-files/alfajores/genesis.json --output-document=${OPGETHDATADIR}/genesis.json

docker run -it \
  --name op-geth-init \
  -v ${OPGETHDATADIR}:/datadir \
```

```
${OPGETHIMAGE} \
  --datadir=/datadir \
  init /datadir/genesis.json
```

:::danger

If you plan to run your L2 node as an archive node (`--gcmode=archive`), geth will automatically default to `--state.scheme=hash`. So the geth init command above must be run with `--state.scheme=hash` also.

:::

Additionally when running `op-geth`, you need to provide `--syncmode=snap` flag. Please continue with executing `op-geth` instructions to start your L2 node.

Option 2: Download L2 chaindata

This option is best for nodes that need the full chain history (e.g. archive nodes). In case of an archive node, you can download the migrated chaindata from a L2 fullnode snapshot, and run `op-geth` with the `--gcmode=archive` flag (it will only keep archive state for L2 blocks). Also, with this option, you can either use `--syncmode=consensus-layer` or `--syncmode=snap` (you will need to have `op-node` peers to use `--syncmode=consensus-layer` and `op-geth` peers to use `--syncmode=snap`).

bash

```
OPGETHDATADIR=${HOME}/alfajores-l2/op-geth-datadir
```

```
mkdir -p ${OPGETHDATADIR}
```

```
wget https://storage.googleapis.com/cel2-rollup-
files/alfajores/alfajores-migrated-datadir.tar.zst
```

```
tar -xvf alfajores-migrated-datadir.tar.zst -C ${OPGETHDATADIR}
```

Please continue with executing `op-geth` instructions to start your L2 node.

Option 3: L1 chaindata migration

Migrating L1 chaindata is the most involved option, but you can use your own L1 chaindata, not trusting the provided chaindata. This option can be split into two steps: pre-migration and full migration. For the pre-migration, you can use the chaindata from a L1 fullnode you trust. This step can be used to prepare the chaindata for the full migration, reducing the time required for the full migration (and the downtime of the node during the migration).

For the full migration step, you will need to wait until Alfajores L1 has stopped producing blocks (that will happen at block 26384000).

Pre-migration

The migration script is provided as a docker image, but you can also review and build it from source. The docker image is available at `us-`

west1-docker.pkg.dev/devopsre/celo-blockchain-public/cel2-migration-tool:celo9. To build from source, you can follow the next steps:

```
bash
git clone https://github.com/celo-org/optimism.git
git checkout celo9
cd ops-chain-ops
make celo-migrate
```

24-72 hours before the migration block, node operators can run a pre-migration script. This script migrates the majority of the chaindata in advance so that the full migration can quickly complete the migration on top of the latest state when the migration block is reached.

1. Create a snapshot of the node's chaindata directory. This is a subdirectory of the node's datadir.
 - Do not run the migration script on a datadir that is actively being used by a node.
 - Make sure you have at least enough disk space to store double the size of the snapshot, as you will be storing both the old and new (migrated) chaindata.
2. Run the pre-migration script. Using the docker image, you can use the following command as a reference:

```
:::danger
Do not run the migration script on a datadir that is actively being used
by a node even if it has stopped adding blocks.
:::
```

```
bash
OPGETHDATADIR=${HOME}/alfajores-l2/op-geth-datadir
CEL2MIGRATIONIMAGE=us-west1-docker.pkg.dev/devopsre/celo-blockchain-
public/cel2-migration-tool:celo9

mkdir -p ${OPGETHDATADIR}
docker run --platform linux/amd64 -it --rm \
  -v /path/to/old/datadir/celo/chaindata:/old-db \
  -v ${OPGETHDATADIR}/geth/chaindata:/new-db \
  ${CEL2MIGRATIONIMAGE} \
  pre --old-db /old-db --new-db /new-db
```

Where:

- old-db should be the path to the chaindata snapshot.
- new-db should be the path where you want the L2 chaindata to be written.
- Please run the pre-migration script at least 24 hours before the migration block so that there is time to troubleshoot any issues.
- Ensure the pre-migration script completes successfully (this should be clear from the logs). If it does not, please reach out for assistance on discord

- Keep the new-db as is until the full migration script is run. If this data is corrupted or lost, the full migration may fail or take a very long time to complete.

Full migration

At block 26384000, the L1 chain will stop producing blocks. At this point, you can run the full migration script. For this step, you will need to pass in some additional files, and also configure paths to write the rollup config and genesis files to.

You can pull down the required deploy-config, l1-deployments, and l2-allocs files as follows.

```
bash
CEL2MIGRATIONDIR=${HOME}/alfajores-l2/cel2-migration-tool

mkdir -p ${CEL2MIGRATIONDIR}
wget -O ${CEL2MIGRATIONDIR}/config.json
https://storage.googleapis.com/cel2-rollup-files/alfajores/config.json
wget -O ${CEL2MIGRATIONDIR}/deployment-l1.json
https://storage.googleapis.com/cel2-rollup-files/alfajores/deployment-
l1.json
wget -O ${CEL2MIGRATIONDIR}/l2-allocs.json
https://storage.googleapis.com/cel2-rollup-files/alfajores/l2-allocs.json
```

Now we can run the migration script. Remember to stop your node (geth) before running it:

```
bash
OPGETHDATADIR=${HOME}/alfajores-l2/op-geth-datadir
CEL2MIGRATIONIMAGE=us-west1-docker.pkg.dev/devopsre/celo-blockchain-
public/cel2-migration-tool:celo9

mkdir -p ${OPGETHDATADIR}
docker run --platform linux/amd64 -it --rm \
  -v /path/to/old/datadir/celo/chaindata:/old-db \
  -v ${OPGETHDATADIR}/geth/chaindata:/new-db \
  -v ${CEL2MIGRATIONDIR}:/migration-files \
  ${CEL2MIGRATIONIMAGE} \
  full \
  --old-db /old-db \
  --new-db /new-db \
  --deploy-config /migration-files/config.json \
  --l1-deployments /migration-files/deployment-l1.json \
  --l2-allocs /migration-files/l2-allocs.json \
  --l1-rpc https://ethereum-holesky-rpc.publicnode.com \
  --outfile.rollup-config /migration-files/rollup.json \
  --outfile.genesis /migration-files/genesis.json \
  --migration-block-time=1727339320
```

- old-db must be the path to the chaindata snapshot or the chaindata of your stopped node.
- new-db must be the path where you want the L2 chaindata to be written. This must be the same path as in the pre-migration script, otherwise all the work done in the pre-migration will be lost.
- deploy-config must be the path to the JSON file that was used for the l1 contracts deployment. This will be distributed by cLabs.
- l1-deployments must be the path to the L1 deployments JSON file, which is the output of running the bedrock contracts deployment for the given 'deploy-config'. This will be distributed by cLabs.
- l1-rpc must be the RPC URL of the L1 node. For alfajores it must be a Holesky endpoint. Feel free to use any other endpoint that you trust.
- l2-allocs must be the path to the JSON file defining necessary state modifications that will be made during the full migration. This will be distributed by cLabs.
- outfile.rollup-config is the path where you want the rollup-config.json file to be written by the migration script. You will need to pass this file when starting the L2 node.
- outfile.genesis is the path where you want the genesis.json file to be written by the migration script. Any node wishing to snap sync on the L2 chain will need this file.
- migration-block-time Should be the unix timestamp of the first L2 block produced by the sequencer (1727339320). Not including this flag will cause the L2 block hash to not match the one posted below, which will prevent your node from syncing.

:::danger

Be sure to include the --migration-block-time flag when running the full migration, using the official timestamp posted below

:::

Ensure the full migration script completes successfully (this should be clear from the logs). If it does not, please reach out for assistance.

:::warning

Please check in your migration logs that the resulting migration block must look like this for Alfajores:

INFO [09-26|08:28:40.753] Wrote CeL2 migration block

height=26,384,000

root=0x38e32589e0300c46a5b98b037ea85abc5b28e3a592c36ce28d07efca8983283b

hash=0xe96cb39b59ebe02553e47424e7f57dbfbffca905c3ff350765985289754a00a3

timestamp=1,727,339,320

:::

Please continue with executing op-geth instructions to start your L2 node.

Executing op-geth

1. To run op-geth, you can use the container image: us-west1-

docker.pkg.dev/devopsre/celo-blockchain-public/op-geth:celo8.

Alternatively, you can clone the celo-org/op-geth repository and build op-geth from source:

```
bash
git clone https://github.com/celo-org/op-geth.git
cd op-geth
git checkout celo8
make geth
```

2. Now you can run the op-geth client. You can use the following example as a reference. If you wish to use full sync instead of snap sync you need to add the flag `--syncmode=full`.

```
bash
OPGETHIMAGE=us-west1-docker.pkg.dev/devopsre/celo-blockchain-public/op-
geth:celo8
```

```
docker run -d \
  --name op-geth \
  --network=host \
  -v ${OPGETHDATA}:/datadir \
  ${OPGETHIMAGE} \
  --datadir=/datadir \
  --networkid=44787 \
  --gcmode=full \
  --snapshot=true \
  --maxpeers=60 \
  --port=30303 \
  --rollup.sequencerhttp=https://sequencer.alfajores.celo-testnet.org \
  --rollup.disabletxpoolgossip=true \
  --authrpc.addr=127.0.0.1 \
  --authrpc.port=8551 \
  --authrpc.jwtsecret=/datadir/jwt.hex \
  --authrpc.vhosts='' \
  --http \
  --http.addr=127.0.0.1 \
  --http.port=8545 \
  --http.api=eth,net,web3,debug,txpool,engine \
  --http.vhosts='' \
  --http.corsdomain='' \
  --verbosity=3 \
  --
bootnodes=enode://ac0f42fa46f8cc10bd02a103894d71d495537465133e7c442bc02dc
76721a5f41761cc2d8c69e7ba1b33e14e28f516436864d3e0836e2dcdaf032387f72447dd
@34.83.164.192:30303,enode://596002969b8b269a4fa34b4709b9600b64201e7d02e2
f5f1350affd021b0cbda6ce2b913ebe24f0fb1edcf66b6c730a8a3b02cd940f4de995f73d
3b290a0fc92@34.82.177.77:30303,enode://3619455064ef1ce667171bba1df80cfd4c
097f018cf0205aaad496f0d509611b7c40396893d9e490ee390cd098888279e177a4d9bb0
9c58387bb0a6031d237f1@34.19.90.27:30303,enode://e3c54db6004a92d4ee87504f0
73f3234a25759b485274cc224037e3e5ee792f3b482c3f4fffcfb764af6e1859a1aea9710b
71e1991e32c1dee7f40352124bb182@35.233.249.87:30303,enode://674410b34fd54c
8406a4f945292b96111688d4bab49aecdc34b4f1b346891f4673dcb03ed44c38ab467ef7b
ec0b20f6031ad88aal35ce1333b343d00fa19fb1@34.168.43.76:30303
```

To see the logs, you can run `docker logs -f op-geth`.

Running op-node

Op-node is not a resource-demanding service. We recommend running it on any modern CPU (amd64 or arm64) with at least 2GB of memory. It is stateless, so it does not require any persistent storage.

1. To run op-node, you can use the container image: `us-west1-docker.pkg.dev/devopsre/celo-blockchain-public/op-node:celo9`. Alternatively, you can clone the `celo-org/optimism` repository and build op-node from source:

```
bash
git clone https://github.com/celo-org/optimism.git
cd optimism/op-node
git checkout celo9
make op-node
```

2. Make a directory for op node config and download the rollup config file:

```
bash
OPNODEDIR=${HOME}/alfajores-l2/op-node
mkdir -p ${OPNODEDIR}
wget https://storage.googleapis.com/cel2-rollup-files/alfajores/rollup.json --output-document=${OPNODEDIR}/rollup.json
```

3. Run the container (or the binary if preferred). You can use the following example as a reference. If you're using snap sync mode, you need to add the flag `--syncmode=execution-layer`.

```
bash
OPNODEIMAGE=us-west1-docker.pkg.dev/devopsre/celo-blockchain-public/op-node:celo9
```

```
cp ${OPGETHDATADIR}/jwt.hex ${OPNODEDIR}/jwt.hex
```

```
docker run --platform linux/amd64 -d \
  --name op-node \
  --network=host \
  --restart=always \
  -v ${OPNODEDIR}:/data \
  ${OPNODEIMAGE} \
  op-node \
  --l1.trustrpc=true \
  --l1=https://ethereum-holesky-rpc.publicnode.com \
  --l1.beacon=https://ethereum-holesky-beacon-api.publicnode.com \
  --l2=http://localhost:8551 \
  --l2.jwt-secret=/data/jwt.hex \
  --rollup.load-protocol-versions=true \
  --rollup.config=/data/rollup.json \
```

```

--verifier.l1-confs=4 \
--rpc.addr=127.0.0.1 \
--rpc.port=9545 \
--p2p.listen.tcp=9222 \
--p2p.listen.udp=9222 \
--p2p.priv.path=/data/op-nodep2ppriv.txt \
--
p2p.static=/ip4/35.197.25.52/tcp/9222/p2p/16Uiu2HAmQEEdyLRSaVZDr5SqbJ1RnKm
NDhtQJcEKmemrVxe4FxKwR,/ip4/34.105.22.4/tcp/9222/p2p/16Uiu2HAm1SZBDSugT5M
Mu7vBY8auDgfZFNhoDeXPLc9Me5FsAxwT,/ip4/34.83.209.168/tcp/9222/p2p/16Uiu2H
AmGJAiUX6HLSO4nLh8T984qXzokwL23cVsYuNZy2SrK7C6,/ip4/34.83.214.149/tcp/922
2/p2p/16Uiu2HAmAko2Kr3eAjM7tnshtEhYrxQYfKUvN2kwiYgeFoBAoi8S,/ip4/34.169.5
.52/tcp/9222/p2p/16Uiu2HAmKc6YKHzyGsjBDaj36uAufxpGZFgrzDqVBt6zTPwdhhJD \
--altda.enabled=true \
--altda.da-server=http://localhost:4242 \
--altda.da-service=true \
--altda.verify-on-read=false

```

You can check the logs running `docker logs -f op-node`. If you start `op-node` before `op-geth`, it will shut down after a few seconds if it cannot connect to its corresponding `op-geth` instance. This is normal behavior. It will run successfully once `op-geth` is running and it can connect to it.

Supporting historical execution

The Celo L2 node alone cannot serve RPC requests requiring historical state or execution from before the migration. To serve such requests, you must configure a `HistoricalRPCService` URL pointing to a Celo L1 node. The Celo L2 node will then proxy requests for historical state / execution out to the Celo L1 node.

To configure a `HistoricalRPCService`, add the following flag when starting `op-geth`

```

bash
--rollup.historicalrpc=<CELOL1NODEURL>

```

In case you do not have access to a Celo L1 node, we provide the following endpoint `<CELOL1NODEURL>=https://alfajores-forno.cell1.alfajores.celo-testnet.org` for the `HistoricalRPCService`.

The following rpc endpoints will forward historical requests.

```

- ethcall
- ethestimateGas
- ethgetBalance
- ethgetProof
- ethgetCode
- ethgetStorageAt
- ethcreateAccessList

```

- eth.getTransactionCount
- debug.traceBlockByNumber
- debug.traceBlockByHash
- debug.traceCall
- debug.traceTransaction

Mainnet migration

We haven't yet picked a migration date for upgrading the Celo mainnet to the Layer 2 codebase. Once a date is set, we will update this page with the mainnet migration instructions. The process will be similar to the Alfajores migration. For now, feel free to refer to the Alfajores migration instructions as a reference and stay tuned for updates on the Mainnet migration process!

network-information.md:

title: Cel2 Alfajores Testnet Network Information

description: Cel2 Alfajores Testnet Network Information

| Name | Value |
|---|---|
| ----- | ----- |
| ----- | ----- |
| ----- | ----- |
| ----- | ----- |
| Network Name | Cel2 Alfajores Testnet |
| Description | The Cel2 testnet |
| Chain ID | 44787 |
| Currency Symbol | CELO |
| RPC Endpoint (best effort) | https://alfajores-forno.celo-testnet.org |
|
 Note to developers: Forno is rate limited, as your usage increases consider options that can provide the desired level of support (SLA). | |
| Block Explorers | |
| https://explorer.celo.org/alfajores https://alfajores.celoexplorer.io | |

l1-l2.md:

Celo L1 → L2

Node operators

In the Celo L1 node, operators simply needed to run the celo-blockchain client, a single service that was a fork of go-ethereum. Moving to the Celo L2 node operators will need to run an op-geth instance for execution, an op-node instance for consensus and an eigenda-proxy for data availability. Instructions on operating nodes are [here](#).

Deprecated transactions

Sending these transaction types will no longer be supported, however you will still be able to retrieve any historical instances of these transactions.

Type 0 (0x0) Celo legacy transaction. These are type 0 transactions that had some combination of the following fields set ("feeCurrency", "gatewayFee", "gatewayFeeRecipient") and "ethCompatible" set to false.

Type 124 (0x7c) Celo dynamic fee transaction.

More details on supported transaction types [here](#).

Native bridge to Ethereum

An integral part of L2 design is that they have a native bridge to ethereum. Celo will become an ERC20 token on Ethereum and users will be able to use the native bridge to bridge between the Celo L2 and Ethereum. The Alfajores testnet bridge can be accessed [here](#).

Consensus

The BFT consensus protocol will be removed and replaced with a centralized sequencer. Although there will be no more need for validators the validator set, election mechanism and voting will remain, however validators will not need to run any infrastructure for the L2.

This is a temporary situation, after Mainnet launch we will be looking at re-introducing active roles for validators.

Validator fees and staking rewards

Transaction fees will now go to the sequencer.

Validator and staking rewards will remain, previously rewards were emitted on epoch blocks, Celo L2 does not have epoch blocks so reward emissions will be handled through a smart contract that can be called to periodically distribute rewards.

The amount of rewards to be distributed has not been decided, however during the time that validators are no longer required to run infrastructure rewards will likely be less than in the Celo L1.

Hardforks

See [here](#) for the list of hardforks that will be enabled in the first block of the L2.

Precompiled contracts

All Celo specific precompiles removed except for the transfer precompile which supports Celo token duality (the native asset CELO is also an ERC20 token)

Randomness

The random contract removed, if randomness is needed then the PREVRANDAO opcode can be used, more details [here](#).

Blocks

Block interval changed from 5s to 1s
Block gas changed from 50m to 30m

:::note

Note this results in a 300% increase in gas per second due to the shortened block time

:::

Added fields

withdrawals & withdrawalsRoot - these fields are inherited from Ethereum but not used by the op-stack or Celo, withdrawals will always be an empty list, withdrawalsRoot will always be the empty withdrawals root (0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cad001622fb5e363b421)

blobGasUsed & excessBlobGas - these fields are also inherited from Ethereum but not used by the op-stack or Celo, they will always be zero.
parentBeaconBlockRoot - set to the parentBeaconRoot of the L1 origin block.

Removed fields

randomness - not needed since the randomness feature is being removed
epochSnarkData - not needed since there will be no Plumo support in the Celo L2.

extraData - will have the BLS aggregated signature removed, this simplifies the L2 implementation and since we already trust those blocks and distributed rewards for them, the BLS signature is no longer required.

EIP1559 implementation

Previously our implementation used a smart contract ([here](#)) to calculate the base fee which allowed for governable parameters. Now we use the standard EIP1559 algorithm with the parameter values being defined in the chain config.

The parameters we are using are:

- Alfajores testnet:
 - EIP-1559 elasticity multiplier: 5
 - EIP-1559 denominator: 400
 - EIP-1559 floor: 25 Gwei (note that the floor is not part of the original EIP-1559 specification, but it did exist in the Celo L1 smart contract implementation)

RPC API

Pre-transition data

Old blocks, transactions, receipts and logs will still be accessible via the RPC API but they will differ a bit from the corresponding objects retrieved from the L1 RPC API.

In general the changes involve additional extra unset fields that have been added upstream but were not present on historical Celo L1 objects, and the removal of some unnecessarily set fields on Celo L1 objects.

For in depth details of what is changed see here - (TODO add section in specs covering this)

Pre-transition execution and state access

RPC API calls for pre-transition blocks that are performing execution or accessing state will not be directly supported by the new Celo L2 implementation, however if this is required you can configure your Celo L2 node to proxy to an archive Celo L1 node for these calls. See here

op-l2.md:

Optimism → Celo L2

Blocks

Celo L2 block times are 1s as opposed to 2s for optimism, the gas limit per block remains the same.

Native token

The native token is CELO as opposed to ETH. The native token is also an ERC20 token.

New transaction type

Type 123 (0x7b) transaction type allows paying for gas in currencies other than the native asset (CELO). It has an additional field feeCurrency which allows the sender to chose the currency they pay gas in a chosen fee currency. See here for details on using fee currencies.

The fee currencies available at mainnet launch will be:

- USDC (USDC)
- Tether USD (USD₮)
- PUSO (PUSO)
- ECO CFA (eXOF)
- Celo Kenyan Shilling (cKES)
- Celo Dollar (cUSD)
- Celo Euro (cEUR)
- Celo Brazilian Real (cREAL)

More details on supported transaction types [here](#).

L1 fees

In the Optimism model, an extra fee is added on to user transactions in order to cover the cost of the L1, this can be surprising to users since they have no visibility about what that fee may be since it is not included in the results of calling `ethestimateGas`.

The Celo L2 always keeps the L1 fee at zero. This doesn't however mean that we will not charge fees to cover the cost of the L1, just that we won't do it via the L1 fees mechanism. Instead we may raise or lower the base fee floor accordingly to match the L1 fees.

EIP1559 implementation

The Celo L2 adds a base fee floor, which imposes a lower limit on the base fee. This is currently configured through chain config.

The starting value is:

- Alfajores testnet:
 - base fee floor: 25 gwei

MaxCodeSize

The hardcoded protocol parameter `MaxCodeSize` is raised from 24576 to 65536.

Block receipt

Historically the Celo L1 generated a block receipts when system contract calls generated logs, although the Celo L2 doesn't produce block receipts, the pre-existing block receipts are retrievable via the RPC API `ethgetBlockReceipt` call passing the hash of the block in question.

Improved finality guarantees

The Celo L2 blocks only reference L1 blocks that are finalized, this means that the Celo L2 is protected from re-orgs that could occur due to L1 re-orgs, this is in contrast to the Optimism L2 blocks which reference L1 blocks 4 blocks behind the L1 chain head.

overview.md:

title: What's changed?

description: Changes from L1 to L2 and from op-stack to L2

What's changed

Celo is moving from being a POS (proof of stake) based L1 blockchain to an L2 built on the OP Stack. In Celo L1 both ordering and data availability were provided by the validators participating in the POS consensus mechanism, being an L2 means that Celo will instead rely on the the L1 (Ethereum) for ordering and on EigenDA for data availability. Outsourcing those components allows Celo to offer increased scalability while focussing on providing value for users. See below for more details about all the changes involved.

Details

Celo L1 → L2 changes

Optimism → Celo L2 changes

account.md:

```
celocli account
=====
```

Manage your account, keys, and metadata

```
celocli account:authorize
celocli account:balance ARG1
celocli account:claim-account ARG1
celocli account:claim-domain ARG1
celocli account:claim-keybase ARG1
celocli account:claim-name ARG1
celocli account:claim-storage ARG1
celocli account:create-metadata ARG1
celocli account:deauthorize
celocli account:delete-payment-delegation
celocli account:get-metadata ARG1
celocli account:get-payment-delegation
celocli account:list
celocli account:lock ARG1
celocli account:new
celocli account:offchain-read ARG1
celocli account:offchain-write
celocli account:proof-of-possession
celocli account:recover-old
celocli account:register
celocli account:register-data-encryption-key
celocli account:register-metadata
celocli account:set-name
celocli account:set-payment-delegation
celocli account:set-wallet
celocli account:show ARG1
celocli account:show-claimed-accounts ARG1
celocli account:show-metadata ARG1
celocli account:unlock ARG1
celocli account:verify-proof-of-possession
```

```
celocli account:authorize
```

Keep your locked Gold more secure by authorizing alternative keys to be used for signing attestations, voting, or validating. By doing so, you can continue to participate in the protocol while keeping the key with access to your locked Gold in cold storage. You must include a "proof-of-possession" of the key being authorized, which can be generated with the "account:proof-of-possession" command.

USAGE

```
$ celocli account:authorize --from <value> -r
vote|validator|attestation --signature
    <value> --signer <value> [--gasCurrency <value>] [--globalHelp] [--
blsKey <value>
    --blsPop <value>]
```

FLAGS

```
-r, --role=<option>
    (required) Role to delegate
    <options: vote|validator|attestation>

--blsKey=0x
    The BLS public key that the validator is using for consensus,
    should pass proof of
    possession. 96 bytes.

--blsPop=0x
    The BLS public key proof-of-possession, which consists of a
    signature on the account
    address. 48 bytes.

--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d
    (required) Account Address

--gasCurrency=0x1234567890123456789012345678901234567890
    Use a specific gas currency for transaction fees (defaults to CEL0
    if no gas
    currency is supplied). It must be a whitelisted token.

--globalHelp
    View all available global flags

--signature=0x
    (required) Signature (a.k.a proof-of-possession) of the signer key

--signer=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d
    (required) Account Address
```

DESCRIPTION

Keep your locked Gold more secure by authorizing alternative keys to be used for signing attestations, voting, or validating. By doing so, you can continue to participate in the protocol while keeping the key with access to your locked Gold in

cold storage. You must include a "proof-of-possession" of the key being authorized,
which can be generated with the "account:proof-of-possession" command.

EXAMPLES

```
authorize --from 0x5409ED021D9299bf6814279A6A1411A7e866A631 --role vote
--signer 0x6ecbelddb9ef729cbe972c83fb886247691fb6beb --signature
0x1b9fca4bbb5bfb1dbe69ef1cddb9b4202dcb6b134c5170611e1e36ecfa468d7b46c853
28d504934fce6c2a1571603a50ae224d2b32685e84d4d1aleebad8452eb
```

```
authorize --from 0x5409ED021D9299bf6814279A6A1411A7e866A631 --role
validator --signer 0x6ecbelddb9ef729cbe972c83fb886247691fb6beb --signature
0x1b9fca4bbb5bfb1dbe69ef1cddb9b4202dcb6b134c5170611e1e36ecfa468d7b46c853
28d504934fce6c2a1571603a50ae224d2b32685e84d4d1aleebad8452eb --blsKey
0x4fa3f67fc913878b068d1falcdcdc54913d3bf988dbe5a36a20fa888f20d4894c408a67
73f3d7bde11154f2a3076b700d345a42fd25a0e5e83f4db5586ac7979ac2053cd95d8f2ef
d3e959571ceccaa743e02cf4be3f5d7aaddb0b06fc9aff00 --blsPop
0xcd77255037eb68897cd487fdd85388cbda448f617f874449d4b11588b0b7ad8ddc20d9
bb450b513bb35664ea3923900
```

See code: `src/commands/account/authorize.ts`

```
celocli account:balance ARG1
```

View Celo Stables and CELO balances for an address

USAGE

```
$ celocli account:balance ARG1 [--gasCurrency <value>] [--globalHelp]
[--erc20Address <value>]
```

FLAGS

```
--erc20Address=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d
    Address of generic ERC-20 token to also check balance for

--gasCurrency=0x1234567890123456789012345678901234567890
    Use a specific gas currency for transaction fees (defaults to CELO
if no gas
    currency is supplied). It must be a whitelisted token.

--globalHelp
    View all available global flags
```

DESCRIPTION

View Celo Stables and CELO balances for an address

EXAMPLES

```
balance 0x5409ed021d9299bf6814279a6a1411a7e866a631

balance 0x5409ed021d9299bf6814279a6a1411a7e866a631 --erc20Address
0x765DE816845861e75A25fCA122bb6898B8B1282a
```

See code: src/commands/account/balance.ts

celocli account:claim-account ARG1

Claim another account, and optionally its public key, and add the claim to a local metadata file

USAGE

```
$ celocli account:claim-account ARG1 --from <value> --address <value>
[--gasCurrency <value>] [--globalHelp] [--publicKey <value>]
```

ARGUMENTS

ARG1 Path of the metadata file

FLAGS

| | |
|--|--------------|
| --address=<value> | (required) |
| The address of | |
| you want to | the account |
| | claim |
| --from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | (required) |
| Address of the | |
| set metadata for | account to |
| | or an |
| authorized signer for | the address |
| in the metadata | |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| --globalHelp | flags |
| available global | The public |
| | account that |
| --publicKey=<value> | |
| key of the | to send you |
| others may use | messages |
| encrypted | |

DESCRIPTION

Claim another account, and optionally its public key, and add the claim to a local

metadata file

EXAMPLES

```
claim-account /metadata.json --address
0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d --from
0x47e172F6CfB6c7D01C1574fa3E2Be7CC73269D95
```

See code: src/commands/account/claim-account.ts

celocli account:claim-domain ARG1

Claim a domain and add the claim to a local metadata file

USAGE

```
$ celocli account:claim-domain ARG1 --from <value> --domain <value> [--
gasCurrency
<value>] [--globalHelp]
```

ARGUMENTS

ARG1 Path of the metadata file

FLAGS

| | |
|--|---------------|
| --domain=<value> | (required) |
| The domain you | |
| --from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | want to claim |
| Address of the | (required) |
| | account to |
| set metadata for | |
| | or an |
| authorized signer for | |
| | the address |
| in the metadata | |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | |
| | for |
| transaction fees | |
| | (defaults to |
| CELO if no gas | |
| | currency is |
| supplied). It | |
| | must be a |
| whitelisted token. | |
| --globalHelp | View all |
| available global | |
| | flags |

DESCRIPTION

Claim a domain and add the claim to a local metadata file

EXAMPLES


```
claim-domain /metadata.json --domain example.com --from
0x47e172F6CfB6c7D01C1574fa3E2Be7CC73269D95
```

See code: src/commands/account/claim-domain.ts

```
celocli account:claim-keybase ARG1
```

Claim a keybase username and add the claim to a local metadata file

USAGE

```
$ celocli account:claim-keybase ARG1 --from <value> --username <value>
[--gasCurrency
  <value>] [--globalHelp]
```

ARGUMENTS

ARG1 Path of the metadata file

FLAGS

| | |
|---|--------------|
| <code>--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Address of the | |
| set metadata for | account to |
| authorized signer for | or an |
| in the metadata | the address |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| <code>--globalHelp</code> | flags |
| available global | (required) |
| <code>--username=<value></code> | username you |
| The keybase | |
| want to claim | |

DESCRIPTION

Claim a keybase username and add the claim to a local metadata file

EXAMPLES

```
claim-keybase /metadata.json --from
0x47e172F6CfB6c7D01C1574fa3E2Be7CC73269D95 --username myusername
```

See code: src/commands/account/claim-keybase.ts

celocli account:claim-name ARG1

Claim a name and add the claim to a local metadata file

USAGE

```
$ celocli account:claim-name ARG1 --from <value> --name <value> [--gasCurrency <value>] [--globalHelp]
```

ARGUMENTS

ARG1 Path of the metadata file

FLAGS

| | |
|--|--------------|
| --from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | (required) |
| Address of the | |
| set metadata for | account to |
| authorized signer for | or an |
| in the metadata | the address |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| --globalHelp | flags |
| available global | (required) |
| --name=<value> | |
| The name you want | to claim |

DESCRIPTION

Claim a name and add the claim to a local metadata file

EXAMPLES

```
claim-name /metadata.json --from
0x47e172F6CfB6c7D01C1574fa3E2Be7CC73269D95 --name myname
```

See code: src/commands/account/claim-name.ts

celocli account:claim-storage ARG1

Claim a storage root and add the claim to a local metadata file

USAGE

```
$ celocli account:claim-storage ARG1 --from <value> --url <value> [--gasCurrency <value>]
    [--globalHelp]
```

ARGUMENTS

ARG1 Path of the metadata file

FLAGS

| | |
|---|--------------|
| <code>--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Address of the | |
| set metadata for | account to |
| authorized signer for | or an |
| in the metadata | the address |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | |
| <code>--globalHelp</code> | View all |
| available global | flags |
| <code>--url=https://www.celo.org</code> | (required) |
| The URL of the | storage root |
| you want to | claim |

DESCRIPTION

Claim a storage root and add the claim to a local metadata file

EXAMPLES

```
claim-storage /metadata.json --url http://example.com/myurl --from
0x47e172F6CfB6c7D01C1574fa3E2Be7CC73269D95
```

See code: `src/commands/account/claim-storage.ts`

```
celocli account:create-metadata ARG1
```

Create an empty identity metadata file. Use this metadata file to store claims attesting to ownership of off-chain resources. Claims can be generated with the `account:claim-` commands.

USAGE

```
$ celocli account:create-metadata ARG1 --from <value> [--gasCurrency <value>]
[--globalHelp]
```

ARGUMENTS

ARG1 Path where the metadata should be saved

FLAGS

```
--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d (required)
Address of the account to
set metadata for or an
authorized signer for the address
in the metadata
--gasCurrency=0x1234567890123456789012345678901234567890 Use a
specific gas currency for
transaction fees (defaults to
CELO if no gas currency is
supplied). It must be a
whitelisted token. View all
--globalHelp available global flags
```

DESCRIPTION

Create an empty identity metadata file. Use this metadata file to store claims attesting to ownership of off-chain resources. Claims can be generated with the `account:claim-` commands.

EXAMPLES

```
create-metadata /metadata.json --from
0x47e172F6CfB6c7D01C1574fa3E2Be7CC73269D95
```

See code: `src/commands/account/create-metadata.ts`

```
celocli account:deauthorize
```

Remove an account's authorized attestation signer role.

USAGE

```
$ celocli account:deauthorize --from <value> -r attestation --signer <value>
```

[--gasCurrency <value>] [--globalHelp]

FLAGS

-r, --role=<option>
(required) Role to remove
<options: attestation>

--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d
(required) Account Address

--gasCurrency=0x1234567890123456789012345678901234567890
Use a specific gas currency for transaction fees (defaults to Celo
if no gas
currency is supplied). It must be a whitelisted token.

--globalHelp
View all available global flags

--signer=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d
(required) Account Address

DESCRIPTION

Remove an account's authorized attestation signer role.

EXAMPLES

deauthorize --from 0x5409ED021D9299bf6814279A6A1411A7e866A631 --role
attestation --signer 0x6ecbe1db9ef729cbe972c83fb886247691fb6beb

See code: `src/commands/account/deauthorize.ts`

`celocli account:delete-payment-delegation`

Removes a validator's payment delegation by setting beneficiary and
fraction to 0.

USAGE

\$ `celocli account:delete-payment-delegation --account <value> [--
gasCurrency <value>]`
[--globalHelp]

FLAGS

--account=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d (required)
Account Address

--gasCurrency=0x1234567890123456789012345678901234567890 Use a
specific gas currency for
transaction fees (defaults to
Celo if no gas
currency is
supplied). It

whitelisted token.

--globalHelp
available global

must be a

View all

flags

DESCRIPTION

Removes a validator's payment delegation by setting beneficiary and fraction to 0.

EXAMPLES

```
delete-payment-delegation --account  
0x5409ED021D9299bf6814279A6A1411A7e866A631
```

See code: `src/commands/account/delete-payment-delegation.ts`

`celocli account:get-metadata ARG1`

Show information about an address. Retrieves the metadata URL for an account from the on-chain, then fetches the metadata file off-chain and verifies proofs as able.

USAGE

```
$ celocli account:get-metadata ARG1 [--gasCurrency <value>] [--  
globalHelp] [--columns  
  <value> | -x] [--filter <value>] [--no-header | [--csv | --no-  
truncate]] [--output  
  csv|json|yaml | | ] [--sort <value>]
```

ARGUMENTS

ARG1 Address to get metadata for

FLAGS

-x, --extended
show extra columns

--columns=<value>
only show provided columns (comma-separated)

--csv
output is csv format [alias: --output=csv]

--filter=<value>
filter property by partial string matching, ex: name=foo

--gasCurrency=0x1234567890123456789012345678901234567890
Use a specific gas currency for transaction fees (defaults to CELO
if no gas
currency is supplied). It must be a whitelisted token.

--globalHelp
View all available global flags

--no-header
hide table header from output

--no-truncate
do not truncate output to fit screen

--output=<option>
output in a more machine friendly format
<options: csv|json|yaml>

--sort=<value>
property to sort by (prepend '-' for descending)

DESCRIPTION

Show information about an address. Retreives the metadata URL for an account from the on-chain, then fetches the metadata file off-chain and verifies proofs as able.

EXAMPLES

```
get-metadata 0x97f7333c51897469E8D98E7af8653aAb468050a3
```

See code: `src/commands/account/get-metadata.ts`

```
celocli account:get-payment-delegation
```

Get the payment delegation account beneficiary and fraction allocated from a validator's payment each epoch. The fraction cannot be greater than 1.

USAGE

```
$ celocli account:get-payment-delegation --account <value> [--
gasCurrency <value>] [--globalHelp]
  [--columns <value> | -x] [--filter <value>] [--no-header | [--csv | -
no-truncate]]
  [--output csv|json|yaml | | ] [--sort <value>]
```

FLAGS

-x, --extended
show extra columns

--account=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d
(required) Account Address

--columns=<value>
only show provided columns (comma-separated)

--csv
output is csv format [alias: --output=csv]

--filter=<value>

| | |
|--|--|
| <pre> --globalHelp available global --[no-]local show local and wallet accounts. to only show addresses.</pre> | <pre> View all flags If set, only hardware Use no-local keystore</pre> |
|--|--|

DESCRIPTION

List the addresses from the node and the local instance

See code: src/commands/account/list.ts

celocli account:lock ARG1

Lock an account which was previously unlocked

USAGE

```
$ celocli account:lock ARG1 [--gasCurrency <value>] [--globalHelp]
```

ARGUMENTS

ARG1 Account address

FLAGS

| | |
|---|---|
| <pre> --gasCurrency=0x1234567890123456789012345678901234567890 specific gas currency transaction fees CELO if no gas supplied). It whitelisted token. --globalHelp available global</pre> | <pre> Use a for (defaults to currency is must be a View all flags</pre> |
|---|---|

DESCRIPTION

Lock an account which was previously unlocked

EXAMPLES

```
lock 0x5409ed021d9299bf6814279a6a1411a7e866a631
```

See code: src/commands/account/lock.ts

celocli account:new

Creates a new account locally using the Celo Derivation Path (m/44'/52752'/0/changeIndex/addressIndex) and print out the key information. Save this information for local transaction signing or import into a Celo node. Ledger: this command has been tested swapping mnemonics with the Ledger successfully (only supports english)

USAGE

```
$ celocli account:new [--gasCurrency <value>] [--globalHelp] [--passphrasePath <value>] [--changeIndex <value>] [--addressIndex <value>] [--language chinesesimpli
```

```
fied|chinesetraditional|english|french|italian|japanese|korean|spanish]
[--mnemonicPath <value>] [--derivationPath <value>]
```

FLAGS

```
--addressIndex=<value>
    Choose the address index for the derivation path

--changeIndex=<value>
    Choose the change index for the derivation path

--derivationPath=<value>
    Choose a different derivation Path (Celo's default is "m/44'/52752'/0'"). Use "eth" as an alias of the Ethereum derivation path ("m/44'/60'/0'").
Recreating the same account requires knowledge of the mnemonic, passphrase (if any), and the derivation path

--gasCurrency=0x1234567890123456789012345678901234567890
    Use a specific gas currency for transaction fees (defaults to CELO if no gas currency is supplied). It must be a whitelisted token.

--globalHelp
    View all available global flags

--language=<option>
    [default: english] Language for the mnemonic words. WARNING, some hardware wallets don't support other languages
    <options:
chinesesimplified|chinesetraditional|english|french|italian|japanese|korean|spanish>

--mnemonicPath=<value>
    Instead of generating a new mnemonic (seed phrase), use the user-supplied mnemonic instead. Path to a file that contains all the mnemonic words separated by a space
```

(example: "word1 word2 word3 ... word24"). If the words are a language other than English, the --language flag must be used. Only BIP39 mnemonics are supported

--passphrasePath=<value>
Path to a file that contains the BIP39 passphrase to combine with the mnemonic specified using the mnemonicPath flag and the index specified using the addressIndex flag. Every passphrase generates a different private key and wallet address.

DESCRIPTION

Creates a new account locally using the Celo Derivation Path (m/44'/52752'/0/changeIndex/addressIndex) and print out the key information. Save this information for local transaction signing or import into a Celo node. Ledger: this command has been tested swapping mnemonics with the Ledger successfully (only supports english)

EXAMPLES

new

new --passphrasePath myFolder/mypassphrasefile

new --language spanish

new --passphrasePath somefolder/mypassphrasefile --language japanese --addressIndex 5

new --passphrasePath somefolder/mypassphrasefile --mnemonicPath somefolder/mymnemonicfile --addressIndex 5

See code: src/commands/account/new.ts

celocli account:offchain-read ARG1

DEV: Reads the name from offchain storage

USAGE

\$ celocli account:offchain-read ARG1 [--gasCurrency <value>] [--globalHelp] [--directory <value>] [--bucket <value> --provider git|aws|gcp] [--from <value>] [--privateDEK <value>]

FLAGS

--bucket=<value> If using a GCP or AWS

| | |
|--|--------------|
| bucket this | storage |
| required | parameter is |
| --directory=<value> | [default: .] |
| To which | directory |
| data should be | written |
| --from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | Account |
| Address | |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| --globalHelp | flags |
| available global | |
| --privateDEK=<value> | If the CLI |
| --provider=<option> | push to the |
| should attempt to | <options: |
| cloud | |
| git aws gcp> | |
| DESCRIPTION | |
| DEV: Reads the name from offchain storage | |
| EXAMPLES | |
| offchain-read 0x... | |
| offchain-read 0x... --from 0x... --privateKey 0x... | |
| See code: src/commands/account/offchain-read.ts | |
| celocli account:offchain-write | |
| DEV: Writes a name to offchain storage | |
| USAGE | |
| \$ celocli account:offchain-write --name <value> [--gasCurrency <value>] | |
| [--globalHelp] | |
| [--directory <value>] [--bucket <value> --provider git aws gcp] (-- | |
| privateDEK | |
| <value> --privateKey <value> --encryptTo <value>) | |

| | |
|--|--------------|
| FLAGS | |
| --bucket=<value> | If using a |
| GCP or AWS | storage |
| bucket this | parameter is |
| required | [default: .] |
| --directory=<value> | directory |
| To which | written |
| data should be | |
| --encryptTo=<value> | |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| --globalHelp | flags |
| available global | (required) |
| --name=<value> | (required) |
| --privateDEK=<value> | If the CLI |
| --privateKey=<value> | push to the |
| --provider=<option> | <options: |
| should attempt to | |
| cloud | |
| git aws gcp> | |

DESCRIPTION

DEV: Writes a name to offchain storage

EXAMPLES

```
offchain-write --name test-account --privateKey 0x...
```

```
offchain-write --name test-account --privateKey 0x... privateDEK 0x...
--encryptTo 0x...
```

See code: `src/commands/account/offchain-write.ts`

`celocli account:proof-of-possession`

Generate proof-of-possession to be used to authorize a signer. See the "account:authorize" command for more details.

USAGE

```
$ celocli account:proof-of-possession --signer <value> --account  
<value> [--gasCurrency  
  <value>] [--globalHelp]
```

FLAGS

| | |
|---|---------------|
| <code>--account=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Address of the | account that |
| needs to prove | possession of |
| the signer | key. |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| <code>--globalHelp</code> | flags |
| available global | (required) |
| <code>--signer=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | signer key to |
| Address of the | possession |
| prove | |
| of. | |

DESCRIPTION

Generate proof-of-possession to be used to authorize a signer. See the "account:authorize" command for more details.

EXAMPLES

```
proof-of-possession --account  
0x5409ed021d9299bf6814279a6a1411a7e866a631 --signer  
0x6ecbeldb9ef729cbe972c83fb886247691fb6beb
```

See code: `src/commands/account/proof-of-possession.ts`

```
celocli account:recover-old
```

Recovers the Valora old account and print out the key information. The old Valora app (in a beta state) generated the user address using a seed of 32 bytes, instead of 64 bytes. As the app fixed that, some old accounts were left with some funds. This command allows the user to recover those funds.

USAGE

```
$ celocli account:recover-old --mnemonicPath <value> [--gasCurrency
<value>]
  [--globalHelp] [--passphrasePath <value>] [--changeIndex <value>] [--
addressIndex
  <value>] [--language
chinesesimplified|chinesetraditional|english|french|italian|j
  apanese|korean|spanish] [--derivationPath <value>]
```

FLAGS

```
--addressIndex=<value>
  Choose the address index for the derivation path

--changeIndex=<value>
  Choose the change index for the derivation path

--derivationPath=<value>
  Choose a different derivation Path (Celo's default is
  "m/44'/52752'/0'"). Use "eth"
  as an alias of the Ethereum derivation path ("m/44'/60'/0'").
  Recreating the same
  account requires knowledge of the mnemonic, passphrase (if any),
  and the derivation
  path

--gasCurrency=0x1234567890123456789012345678901234567890
  Use a specific gas currency for transaction fees (defaults to CEL0
  if no gas
  currency is supplied). It must be a whitelisted token.

--globalHelp
  View all available global flags

--language=<option>
  [default: english] Language for the mnemonic words. WARNING, some
  hardware
  wallets don't support other languages
  <options:
  chinesesimplified|chinesetraditional|english|french|italian|japanese|kor
  ean|spanish>

--mnemonicPath=<value>
  (required) Path to a file that contains all the mnemonic words
  separated by a space
  (example: "word1 word2 word3 ... word24"). If the words are a
  language other than
  English, the --language flag must be used. Only BIP39 mnemonics are
  supported

--passphrasePath=<value>
  Path to a file that contains the BIP39 passphrase to combine with
  the mnemonic
```

specified using the mnemonicPath flag and the index specified using the addressIndex flag. Every passphrase generates a different private key and wallet address.

DESCRIPTION

Recovers the Valora old account and print out the key information. The old Valora app (in a beta state) generated the user address using a seed of 32 bytes, instead of 64 bytes. As the app fixed that, some old accounts were left with some funds. This command allows the user to recover those funds.

EXAMPLES

```
recover-old --mnemonicPath somefolder/mymnemonicfile

recover-old --mnemonicPath somefolder/mymnemonicfile --passphrasePath
myFolder/mypassphrasefile

recover-old --mnemonicPath somefolder/mymnemonicfile --language spanish

recover-old --mnemonicPath somefolder/mymnemonicfile --passphrasePath
somefolder/mypassphrasefile --language japanese --addressIndex 5

recover-old --mnemonicPath somefolder/mymnemonicfile --passphrasePath
somefolder/mypassphrasefile --addressIndex 5
```

See code: `src/commands/account/recover-old.ts`

`celocli account:register`

Register an account on-chain. This allows you to lock Gold, which is a pre-requisite for registering a Validator or Group, participating in Validator elections and on-chain Governance, and earning epoch rewards.

USAGE

```
$ celocli account:register --from <value> [--gasCurrency <value>] [--
globalHelp]
    [--name <value>]
```

FLAGS

| | |
|---|--------------|
| <code>--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Account Address | |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | |

| | |
|--------------------|-----------|
| whitelisted token. | must be a |
| --globalHelp | View all |
| available global | |
| | flags |
| --name=<value> | |

DESCRIPTION

Register an account on-chain. This allows you to lock Gold, which is a pre-requisite for registering a Validator or Group, participating in Validator elections and on-chain Governance, and earning epoch rewards.

EXAMPLES

```
register --from 0x5409ed021d9299bf6814279a6a1411a7e866a631
```

```
register --from 0x5409ed021d9299bf6814279a6a1411a7e866a631 --name test-account
```

See code: src/commands/account/register.ts

celocli account:register-data-encryption-key

Register a data encryption key for an account on chain. This key can be used to encrypt data to you such as offchain metadata or transaction comments

USAGE

```
$ celocli account:register-data-encryption-key --from <value> --publicKey <value> [--gasCurrency <value>] [--globalHelp]
```

FLAGS

| | |
|--|--------------|
| --from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | (required) |
| Address of the | |
| set the data | account to |
| key for | encryption |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | |
| --globalHelp | View all |
| available global | |

`--publicKey=<value>` flags
The public key (required)
you want to
register

DESCRIPTION

Register a data encryption key for an account on chain. This key can be used to encrypt data to you such as offchain metadata or transaction comments

EXAMPLES

```
register-data-encryption-key --publicKey 0x... --from  
0x47e172F6CfB6c7D01C1574fa3E2Be7CC73269D95
```

See code: `src/commands/account/register-data-encryption-key.ts`

```
celocli account:register-metadata
```

Register metadata URL for an account where users will be able to retrieve the metadata file and verify your claims

USAGE

```
$ celocli account:register-metadata --from <value> --url <value> [--  
gasCurrency <value>]  
  [--globalHelp] [--force] [--columns <value> | -x] [--filter <value>]  
  [--no-header |  
  [--csv | --no-truncate]] [--output csv|json|yaml | | ] [--sort  
<value>]
```

FLAGS

```
-x, --extended  
    show extra columns  
  
--columns=<value>  
    only show provided columns (comma-separated)  
  
--csv  
    output is csv format [alias: --output=csv]  
  
--filter=<value>  
    filter property by partial string matching, ex: name=foo  
  
--force  
    Ignore metadata validity checks  
  
--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d  
    (required) Address of the account to set metadata for  
  
--gasCurrency=0x1234567890123456789012345678901234567890  
    Use a specific gas currency for transaction fees (defaults to CELO  
if no gas
```

currency is supplied). It must be a whitelisted token.

--globalHelp

View all available global flags

--no-header

hide table header from output

--no-truncate

do not truncate output to fit screen

--output=<option>

output in a more machine friendly format

<options: csv|json|yaml>

--sort=<value>

property to sort by (prepend '-' for descending)

--url=<value>

(required) The url to the metadata you want to register

DESCRIPTION

Register metadata URL for an account where users will be able to retrieve the metadata file and verify your claims

EXAMPLES

```
register-metadata --url https://www.mywebsite.com/celo-metadata --from
0x47e172F6CfB6c7D01C1574fa3E2Be7CC73269D95
```

See code: src/commands/account/register-metadata.ts

celocli account:set-name

Sets the name of a registered account on-chain. An account's name is an optional human readable identifier

USAGE

```
$ celocli account:set-name --account <value> --name <value> [--
gasCurrency <value>]
[--globalHelp]
```

FLAGS

--account=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d (required)
Account Address

--gasCurrency=0x1234567890123456789012345678901234567890 Use a
specific gas currency

transaction fees for

(defaults to
CELO if no gas

supplied). It
whitelisted token.
--globalHelp
available global
--name=<value>

currency is
must be a
View all
flags
(required)

DESCRIPTION

Sets the name of a registered account on-chain. An account's name is an optional human readable identifier

EXAMPLES

set-name --account 0x5409ed021d9299bf6814279a6a1411a7e866a631 --name test-account

See code: src/commands/account/set-name.ts

celocli account:set-payment-delegation

Sets a payment delegation beneficiary, an account address to receive a fraction of the validator's payment every epoch. The fraction must not be greater than 1.

USAGE

\$ celocli account:set-payment-delegation --account <value> --beneficiary <value> --fraction <value> [--gasCurrency <value>] [--globalHelp]

FLAGS

--account=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d (required)
Account Address
--beneficiary=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d (required)
Account Address
--fraction=<value> (required)
--gasCurrency=0x1234567890123456789012345678901234567890 Use a specific gas currency
transaction fees
CELO if no gas
supplied). It
whitelisted token.
--globalHelp
available global

(defaults to
currency is
must be a
View all
flags

DESCRIPTION

Sets a payment delegation beneficiary, an account address to receive a fraction of the validator's payment every epoch. The fraction must not be greater than 1.

EXAMPLES

```
set-payment-delegation --account
0x5409ed021d9299bf6814279a6a1411a7e866a631 --beneficiary
0x6EcbelDB9EF729CBe972C83Fb886247691Fb6beb --fraction 0.1
```

See code: `src/commands/account/set-payment-delegation.ts`

`celocli account:set-wallet`

Sets the wallet of a registered account on-chain. An account's wallet is an optional wallet associated with an account. Can be set by the account or an account's signer.

USAGE

```
$ celocli account:set-wallet --account <value> --wallet <value> [--
gasCurrency
<value>] [--globalHelp] [--signature <value>] [--signer <value>]
```

FLAGS

| | |
|---|--------------|
| <code>--account=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Account Address | |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | |
| | for |
| transaction fees | |
| | (defaults to |
| CELO if no gas | |
| | currency is |
| supplied). It | |
| | must be a |
| whitelisted token. | |
| <code>--globalHelp</code> | View all |
| available global | |
| | flags |
| <code>--signature=0x</code> | Signature |
| (a.k.a. | |
| | proof-of- |
| possession) of the | |
| | signer key |
| <code>--signer=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | Address of |
| the signer key to | |
| | verify proof |
| of possession. | |
| <code>--wallet=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Account Address | |

DESCRIPTION

Sets the wallet of a registered account on-chain. An account's wallet is an optional wallet associated with an account. Can be set by the account or an account's signer.

EXAMPLES

```
set-wallet --account 0x5409ed021d9299bf6814279a6a1411a7e866a631 --
wallet 0x5409ed021d9299bf6814279a6a1411a7e866a631
```

```
set-wallet --account 0x5409ed021d9299bf6814279a6a1411a7e866a631 --
wallet 0x5409ed021d9299bf6814279a6a1411a7e866a631 --signer
0x0EdeDF7B1287f07db348997663EeEb283D70aBE7 --signature
0x1c5efaa1f7ca6484d49ccce76217e2fba0552c0b23462cff7ba646473bc2717ffc4ce45
be89bd5be9b5d23305e87fc2896808467c4081d9524a84c01b89ec91ca3
```

See code: `src/commands/account/set-wallet.ts`

`celocli account:show ARG1`

Show information for an account, including name, authorized vote, validator, and attestation signers, the URL at which account metadata is hosted, the address the account is using with the mobile wallet, and a public key that can be used to encrypt information for the account.

USAGE

```
$ celocli account:show ARG1 [--gasCurrency <value>] [--globalHelp]
```

FLAGS

```
--gasCurrency=0x1234567890123456789012345678901234567890 Use a
specific gas currency                                         for
transaction fees                                           (defaults to
CELO if no gas                                              currency is
supplied). It                                              must be a
whitelisted token.                                         View all
--globalHelp                                               available global flags
```

DESCRIPTION

Show information for an account, including name, authorized vote, validator, and attestation signers, the URL at which account metadata is hosted, the address the account is using with the mobile wallet, and a public key that can be used to encrypt information for the account.

EXAMPLES

```
show 0x5409ed021d9299bf6814279a6a1411a7e866a631
```

See code: `src/commands/account/show.ts`

```
celocli account:show-claimed-accounts ARG1
```

Show information about claimed accounts

USAGE

```
$ celocli account:show-claimed-accounts ARG1 [--gasCurrency <value>] [-  
-globalHelp]
```

FLAGS

```
--gasCurrency=0x1234567890123456789012345678901234567890 Use a  
specific gas currency                                     for  
transaction fees                                         (defaults to  
CELO if no gas                                           currency is  
supplied). It                                           must be a  
whitelisted token.                                       View all  
--globalHelp                                             available global  
                                                           flags
```

DESCRIPTION

Show information about claimed accounts

EXAMPLES

```
show-claimed-accounts 0x5409ed021d9299bf6814279a6a1411a7e866a631
```

See code: `src/commands/account/show-claimed-accounts.ts`

```
celocli account:show-metadata ARG1
```

Show the data in a local metadata file

USAGE

```
$ celocli account:show-metadata ARG1 [--gasCurrency <value>] [--  
globalHelp] [--columns  
  <value> | -x] [--filter <value>] [--no-header | [--csv | --no-  
truncate]] [--output  
  csv|json|yaml | | ] [--sort <value>]
```

ARGUMENTS

ARG1 Path of the metadata file

FLAGS

```

-x, --extended
    show extra columns

--columns=<value>
    only show provided columns (comma-separated)

--csv
    output is csv format [alias: --output=csv]

--filter=<value>
    filter property by partial string matching, ex: name=foo

--gasCurrency=0x1234567890123456789012345678901234567890
    Use a specific gas currency for transaction fees (defaults to CELO
if no gas
    currency is supplied). It must be a whitelisted token.

--globalHelp
    View all available global flags

--no-header
    hide table header from output

--no-truncate
    do not truncate output to fit screen

--output=<option>
    output in a more machine friendly format
    <options: csv|json|yaml>

--sort=<value>
    property to sort by (prepend '-' for descending)

```

DESCRIPTION

Show the data in a local metadata file

EXAMPLES

```
show-metadata /metadata.json
```

See code: `src/commands/account/show-metadata.ts`

```
celocli account:unlock ARG1
```

Unlock an account address to send transactions or validate blocks

USAGE

```
$ celocli account:unlock ARG1 [--gasCurrency <value>] [--globalHelp] [-
-password
    <value>] [--duration <value>]
```

ARGUMENTS

ARG1 Account address

FLAGS

| | |
|---|---------------|
| <code>--duration=<value></code> | Duration in |
| seconds to leave | the account |
| unlocked. | Unlocks until |
| the node exits | by default. |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| <code>--globalHelp</code> | flags |
| available global | Password used |
| <code>--password=<value></code> | account. If |
| to unlock the | you will be |
| not specified, | password. |
| prompted for a | |

DESCRIPTION

Unlock an account address to send transactions or validate blocks

EXAMPLES

```
unlock 0x5409ed021d9299bf6814279a6a1411a7e866a631
```

```
unlock 0x5409ed021d9299bf6814279a6a1411a7e866a631 --duration 600
```

See code: `src/commands/account/unlock.ts`

```
celocli account:verify-proof-of-possession
```

Verify a proof-of-possession. See the "account:proof-of-possession" command for more details.

USAGE

```
$ celocli account:verify-proof-of-possession --signer <value> --account  
<value> --signature <value>  
  [--gasCurrency <value>] [--globalHelp]
```

FLAGS

| | |
|---|---------------|
| <code>--account=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Address of the | account that |
| needs to prove | possession of |
| the signer | key. |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| <code>--globalHelp</code> | flags |
| available global | (required) |
| <code>--signature=0x</code> | proof-of- |
| Signature (a.k.a. | signer key |
| possession) of the | (required) |
| <code>--signer=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | signer key to |
| Address of the | of |
| verify proof | |
| possession. | |

DESCRIPTION

Verify a proof-of-possession. See the "account:proof-of-possession" command for more details.

EXAMPLES

```
verify-proof-of-possession --account
0x199eDF79ABCa29A2Fa4014882d3C13dC191A5B58 --signer
0x0EdeDF7B1287f07db348997663EeEb283D70aBE7 --signature
0x1c5efaa1f7ca6484d49ccce76217e2fba0552c0b23462cff7ba646473bc2717ffc4ce45
be89bd5be9b5d23305e87fc2896808467c4081d9524a84c01b89ec91ca3
```

See code: `src/commands/account/verify-proof-of-possession.ts`

autocomplete.md:

```
celocli autocomplete
=====
```

Display autocomplete installation instructions.

```
[celocli autocomplete [SHELL]](#celocli-autocomplete-shell)
```

```
celocli autocomplete [SHELL] {#celocli-autocomplete-shell}
```

Display autocomplete installation instructions.

USAGE

```
$ celocli autocomplete [SHELL] [-r]
```

ARGUMENTS

```
SHELL (zsh|bash|powershell) Shell type
```

FLAGS

```
-r, --refresh-cache Refresh cache (ignores displaying instructions)
```

DESCRIPTION

Display autocomplete installation instructions.

EXAMPLES

```
$ celocli autocomplete
```

```
$ celocli autocomplete bash
```

```
$ celocli autocomplete zsh
```

```
$ celocli autocomplete powershell
```

```
$ celocli autocomplete --refresh-cache
```

See code: @oclif/plugin-autocomplete

commands.md:

```
celocli commands
```

```
=====
```

list all the commands

```
celocli commands
```

```
celocli commands {#celocli-commands}
```

list all the commands

USAGE

```
$ celocli commands [--json] [--deprecated] [-h] [--hidden] [--tree]
  [--columns <value> | -x] [--filter <value>] [--no-header | [--csv | -
-no-truncate]]
  [--output csv|json|yaml | | ] [--sort <value>]
```

FLAGS

| | |
|-------------------|--|
| -h, --help | Show CLI help. |
| -x, --extended | show extra columns |
| --columns=<value> | only show provided columns (comma-separated) |
| --csv | output is csv format [alias: --output=csv] |
| --deprecated | show deprecated commands |
| --filter=<value> | filter property by partial string matching, ex:
name=foo |
| --hidden | show hidden commands |
| --no-header | hide table header from output |
| --no-truncate | do not truncate output to fit screen |
| --output=<option> | output in a more machine friendly format
<options: csv json yaml> |
| --sort=<value> | property to sort by (prepend '-' for descending) |
| --tree | show tree of commands |

GLOBAL FLAGS

--json Format output as json.

DESCRIPTION

list all the commands

See code: @oclif/plugin-commands

config.md:

celocli config
=====

Configure CLI options which persist across commands

celocli config:get
celocli config:set

celocli config:get {#celocli-configget}

Output network node configuration

USAGE

\$ celocli config:get [--gasCurrency <value>] [--globalHelp]

FLAGS

| | |
|--|-----------------------------|
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a specific gas currency |
| | for transaction fees |
| | (defaults to CELO if no gas |
| | currency is supplied). It |

whitelisted token.

 --globalHelp
available global

must be a

View all

flags

DESCRIPTION

Output network node configuration

See code: src/commands/config/get.ts

celocli config:set {#celocli-configset}

Configure running node information for propagating transactions to network

USAGE

\$ celocli config:set [-n <value>] [--gasCurrency <value>] [--globalHelp]

FLAGS

 -n, --node=<value>
 URL of the node to run commands against (defaults to 'http://localhost:8545')

 --gasCurrency=0x1234567890123456789012345678901234567890
 Use a specific gas currency for transaction fees (defaults to Celo if no gas currency is supplied). It must be a whitelisted token.

 --globalHelp
 View all available global flags

DESCRIPTION

Configure running node information for propagating transactions to network

EXAMPLES

set --node mainnet alias for forno

set --node forno alias for https://forno.celo.org

set --node baklava alias for https://baklava-forno.celo-testnet.org

set --node alfajores alias for https://alfajores-forno.celo-testnet.org

set --node localhost alias for local

set --node local alias for http://localhost:8545

set --node ws://localhost:2500

```
set --node <geth-location>/geth.ipc
```

See code: src/commands/config/set.ts

```
# dkg.md:
```

```
celocli dkg
=====
```

Publish your locally computed DKG results to the blockchain

```
celocli dkg:allowlist
celocli dkg:deploy
celocli dkg:get
celocli dkg:publish
celocli dkg:register
celocli dkg:start
```

```
celocli dkg:allowlist {#celocli-dkgallowlist}
```

Allowlist an address in the DKG

USAGE

```
$ celocli dkg:allowlist --participantAddress <value> --address <value>
--from
  <value> [--gasCurrency <value>] [--globalHelp]
```

FLAGS

| | |
|--|--------------|
| --address=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | (required) |
| DKG Contract | Address |
| --from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | (required) |
| Address of the | sender |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| --globalHelp | flags |
| available global | (required) |
| --participantAddress=<value> | participant |
| Address of the | |
| to allowlist | |

DESCRIPTION

Allowlist an address in the DKG

See code: `src/commands/dkg/allowlist.ts`

`celocli dkg:deploy {#celocli-dkgdeploy}`

Deploys the DKG smart contract

USAGE

```
$ celocli dkg:deploy --phaseDuration <value> --threshold <value> --from  
  <value> [--gasCurrency <value>] [--globalHelp]
```

FLAGS

| | |
|---|--------------|
| <code>--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Address of the | sender |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| <code>--globalHelp</code> | flags |
| available global | (required) |
| <code>--phaseDuration=<value></code> | DKG phase in |
| Duration of each | (required) |
| blocks | use for the |
| <code>--threshold=<value></code> | |
| The threshold to | |
| DKG | |

DESCRIPTION

Deploys the DKG smart contract

See code: `src/commands/dkg/deploy.ts`

`celocli dkg:get {#celocli-dkgget}`

Gets data from the contract to run the next phase

USAGE

```
$ celocli dkg:get --method
  shares|responses|justifications|participants|phase|group --address
<value>
  [--gasCurrency <value>] [--globalHelp]
```

FLAGS

| | |
|---|--------------|
| <code>--address=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| DKG Contract | Address |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| <code>--globalHelp</code> | flags |
| available global | (required) |
| <code>--method=<option></code> | call |
| Getter method to | <options: |
| shares responses j | |
| ustifications participants p | |
| | hase group> |

DESCRIPTION

Gets data from the contract to run the next phase

See code: `src/commands/dkg/get.ts`

```
celocli dkg:publish {#celocli-dkgpublish}
```

Publishes data for each phase of the DKG

USAGE

```
$ celocli dkg:publish --data <value> --address <value> --from <value>
  [--gasCurrency <value>] [--globalHelp]
```

FLAGS

| | |
|---|------------|
| <code>--address=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| DKG Contract | Address |
| <code>--data=<value></code> | (required) |
| Path to the data | being |
| published | |

| | |
|---|--------------|
| <code>--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Address of the | |
| | sender |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | |
| | for |
| transaction fees | |
| | (defaults to |
| CELO if no gas | |
| | currency is |
| supplied). It | |
| | must be a |
| whitelisted token. | |
| <code>--globalHelp</code> | View all |
| available global | |
| | flags |

DESCRIPTION

Publishes data for each phase of the DKG

See code: `src/commands/dkg/publish.ts`

`celocli dkg:register {#celocli-dkgregister}`

Register a public key in the DKG

USAGE

```
$ celocli dkg:register --blsKey <value> --address <value> --from
<value>
  [--gasCurrency <value>] [--globalHelp]
```

FLAGS

| | |
|---|--------------|
| <code>--address=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| DKG Contract | |
| | Address |
| <code>--blsKey=<value></code> | (required) |
| <code>--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Address of the | |
| | sender |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | |
| | for |
| transaction fees | |
| | (defaults to |
| CELO if no gas | |
| | currency is |
| supplied). It | |
| | must be a |
| whitelisted token. | |
| <code>--globalHelp</code> | View all |
| available global | |
| | flags |

DESCRIPTION

Register a public key in the DKG

See code: src/commands/dkg/register.ts

celocli dkg:start {#celocli-dkgstart}

Starts the DKG

USAGE

```
$ celocli dkg:start --address <value> --from <value> [--gasCurrency <value>]
  [--globalHelp]
```

FLAGS

| | |
|--|--------------------|
| --address=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | (required) |
| DKG Contract | |
| --from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | Address (required) |
| Address of the | |
| --gasCurrency=0x1234567890123456789012345678901234567890 | sender |
| specific gas currency | Use a |
| transaction fees | for |
| CELO if no gas | (defaults to |
| supplied). It | currency is |
| whitelisted token. | must be a |
| --globalHelp | View all |
| available global | flags |

DESCRIPTION

Starts the DKG

See code: src/commands/dkg/start.ts

election.md:

celocli election
=====

Participate in and view the state of Validator Elections

```
celocli election:activate
celocli election:current
```

```
celocli election:list
celocli election:revoke
celocli election:run
celocli election:show ARG1
celocli election:vote
```

```
celocli election:activate {#celocli-electionactivate}
```

Activate pending votes in validator elections to begin earning rewards. To earn rewards as a voter, it is required to activate your pending votes at some point after the end of the epoch in which they were made.

USAGE

```
$ celocli election:activate --from <value> [--gasCurrency <value>] [--globalHelp]
    [--for <value>] [--wait]
```

FLAGS

| | |
|--|---------------|
| --for=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | Optional: use |
| this to | activate |
| votes for another | address |
| --from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | (required) |
| Address sending | transaction |
| (and voter's | address if -- |
| for not | specified) |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| --globalHelp | flags |
| available global | Wait until |
| --wait | can be |
| all pending votes | |
| activated | |

DESCRIPTION

Activate pending votes in validator elections to begin earning rewards. To earn

rewards as a voter, it is required to activate your pending votes at some point after

the end of the epoch in which they were made.

EXAMPLES

```
activate --from 0x4443d0349e8b3075cba511a0a87796597602a0f1

activate --from 0x4443d0349e8b3075cba511a0a87796597602a0f1 --for
0x5409ed021d9299bf6814279a6a1411a7e866a631

activate --from 0x4443d0349e8b3075cba511a0a87796597602a0f1 --wait
```

See code: `src/commands/election/activate.ts`

```
celocli election:current {#celocli-electioncurrent}
```

Outputs the set of validators currently participating in BFT to create blocks. An election is run to select the validator set at the end of every epoch.

USAGE

```
$ celocli election:current [--gasCurrency <value>] [--globalHelp] [--
valset]
  [--columns <value> | -x] [--filter <value>] [--no-header | [--csv | -
-no-truncate]]
  [--output csv|json|yaml | | ] [--sort <value>]
```

FLAGS

```
-x, --extended
  show extra columns

--columns=<value>
  only show provided columns (comma-separated)

--csv
  output is csv format [alias: --output=csv]

--filter=<value>
  filter property by partial string matching, ex: name=foo

--gasCurrency=0x1234567890123456789012345678901234567890
  Use a specific gas currency for transaction fees (defaults to CELO
if no gas
  currency is supplied). It must be a whitelisted token.

--globalHelp
  View all available global flags

--no-header
  hide table header from output

--no-truncate
  do not truncate output to fit screen
```

--output=<option>
output in a more machine friendly format
<options: csv|json|yaml>

--sort=<value>
property to sort by (prepend '-' for descending)

--valset
Show currently used signers from valset (by default the authorized validator signers are shown). Useful for checking if keys have been rotated.

DESCRIPTION

Outputs the set of validators currently participating in BFT to create blocks. An election is run to select the validator set at the end of every epoch.

See code: src/commands/election/current.ts

celocli election:list {#celocli-electionlist}

Prints the list of validator groups, the number of votes they have received, the number of additional votes they are able to receive, and whether or not they are eligible to elect validators.

USAGE

```
$ celocli election:list [--gasCurrency <value>] [--globalHelp] [--columns <value>]
  | -x] [--filter <value>] [--no-header | [--csv | --no-truncate]] [--output
  csv|json|yaml | | ] [--sort <value>]
```

FLAGS

-x, --extended
show extra columns

--columns=<value>
only show provided columns (comma-separated)

--csv
output is csv format [alias: --output=csv]

--filter=<value>
filter property by partial string matching, ex: name=foo

--gasCurrency=0x1234567890123456789012345678901234567890
Use a specific gas currency for transaction fees (defaults to CELO if no gas currency is supplied). It must be a whitelisted token.

--globalHelp
View all available global flags

```

--no-header
    hide table header from output

--no-truncate
    do not truncate output to fit screen

--output=<option>
    output in a more machine friendly format
    <options: csv|json|yaml>

--sort=<value>
    property to sort by (prepend '-' for descending)

```

DESCRIPTION

Prints the list of validator groups, the number of votes they have received, the number of additional votes they are able to receive, and whether or not they are eligible to elect validators.

EXAMPLES

```
list
```

See code: src/commands/election/list.ts

```
celocli election:revoke {#celocli-electionrevoke}
```

Revoke votes for a Validator Group in validator elections.

USAGE

```
$ celocli election:revoke --from <value> --for <value> --value <value>
  [--gasCurrency <value>] [--globalHelp]
```

FLAGS

| | |
|--|--------------|
| --for=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | (required) |
| ValidatorGroup's | |
| | address |
| --from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | (required) |
| Voter's address | |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | |
| | for |
| transaction fees | |
| | (defaults to |
| CELO if no gas | |
| | currency is |
| supplied). It | |
| | must be a |
| whitelisted token. | |
| --globalHelp | View all |
| available global | |

| | |
|------------------------------------|------------|
| <code>--value=<value></code> | flags |
| Value of votes to | (required) |
| | revoke |

DESCRIPTION

Revoke votes for a Validator Group in validator elections.

EXAMPLES

```
revoke --from 0x4443d0349e8b3075cba511a0a87796597602a0f1 --for
0x932fee04521f5fcb21949041bf161917da3f588b, --value 1000000
```

See code: `src/commands/election/revoke.ts`

```
celocli election:run {#celocli-electionrun}
```

Runs a "mock" election and prints out the validators that would be elected if the epoch ended right now.

USAGE

```
$ celocli election:run [--gasCurrency <value>] [--globalHelp] [--
columns <value>
| -x] [--filter <value>] [--no-header | [--csv | --no-truncate]] [--
output
csv|json|yaml | | ] [--sort <value>]
```

FLAGS

```
-x, --extended
    show extra columns

--columns=<value>
    only show provided columns (comma-separated)

--csv
    output is csv format [alias: --output=csv]

--filter=<value>
    filter property by partial string matching, ex: name=foo

--gasCurrency=0x1234567890123456789012345678901234567890
    Use a specific gas currency for transaction fees (defaults to CELO
if no gas
    currency is supplied). It must be a whitelisted token.

--globalHelp
    View all available global flags

--no-header
    hide table header from output

--no-truncate
    do not truncate output to fit screen
```

```
--output=<option>
    output in a more machine friendly format
    <options: csv|json|yaml>

--sort=<value>
    property to sort by (prepend '-' for descending)
```

DESCRIPTION

Runs a "mock" election and prints out the validators that would be elected if the epoch ended right now.

See code: `src/commands/election/run.ts`

```
celocli election:show ARG1 {#celocli-electionshow-arg1}
```

Show election information about a voter or registered Validator Group

USAGE

```
$ celocli election:show ARG1 [--gasCurrency <value>] [--globalHelp] [--voter |
    --group]
```

ARGUMENTS

ARG1 Voter or Validator Groups's address

FLAGS

| | |
|--|-----------------------------|
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a specific gas currency |
| | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| --globalHelp | flags |
| available global | Show |
| --group | group running |
| information about a | |
| in Validator | elections |
| --voter | Show |
| information about an | account |
| voting in Validator | elections |

DESCRIPTION

Show election information about a voter or registered Validator Group

EXAMPLES

```
show 0x97f7333c51897469E8D98E7af8653aAb468050a3 --voter
```

```
show 0x97f7333c51897469E8D98E7af8653aAb468050a3 --group
```

See code: src/commands/election/show.ts

```
celocli election:vote {#celocli-electionvote}
```

Vote for a Validator Group in validator elections.

USAGE

```
$ celocli election:vote --from <value> --for <value> --value <value>
[--gasCurrency <value>] [--globalHelp]
```

FLAGS

| | |
|---|--------------|
| <code>--for=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| ValidatorGroup's | |
| <code>--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | address |
| Voter's address | (required) |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | |
| transaction fees | for |
| CELO if no gas | (defaults to |
| supplied). It | currency is |
| whitelisted token. | must be a |
| <code>--globalHelp</code> | View all |
| available global | |
| <code>--value=<value></code> | flags |
| Amount of Gold | (required) |
| for group | used to vote |

DESCRIPTION

Vote for a Validator Group in validator elections.

EXAMPLES

```
vote --from 0x4443d0349e8b3075cba511a0a87796597602a0f1 --for
0x932fee04521f5fcb21949041bf161917da3f588b, --value 1000000
```

See code: src/commands/election/vote.ts

exchange.md:

celocli exchange
=====

Exchange Celo Dollars and CELO via Mento

celocli exchange:celo
celocli exchange:dollars
celocli exchange:euros
celocli exchange:reals
celocli exchange:show
celocli exchange:stable

celocli exchange:celo {#celocli-exchangecelo}

Exchange CELO for StableTokens via Mento. (Note: this is the equivalent of the old exchange:gold)

USAGE

\$ celocli exchange:celo --from <value> --value <value> [--gasCurrency <value>]
[--globalHelp] [--forAtLeast <value>] [--stableToken
cUSD|cusd|cEUR|ceur|cREAL|creal]

FLAGS

| | |
|--|---------------|
| --forAtLeast=10000000000000000000000 | [default: 0] |
| Optional, the | minimum value |
| of | StableTokens |
| to receive in | return |
| --from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | (required) |
| The address with | CELO to |
| exchange | |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| --globalHelp | flags |
| available global | [default: |
| --stableToken=<option> | |
| cusd] Name of the | |

| | |
|--------------------------------|-------------|
| receive | stable to |
| cUSD cusd cEUR ceu | <options: |
| r cREAL creal> | |
| --value=1000000000000000000000 | (required) |
| The value of CELO | to exchange |
| for a | StableToken |

DESCRIPTION

Exchange CELO for StableTokens via Mento. (Note: this is the equivalent of the old exchange:gold)

EXAMPLES

```
celo --value 500000000000000000000 --from
0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d
```

```
celo --value 500000000000000000000 --forAtLeast 1000000000000000000000 --from
0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d --stableToken
cStableTokenSymbol
```

See code: src/commands/exchange/celo.ts

```
celocli exchange:dollars {#celocli-exchangedollars}
```

Exchange Celo Dollars for CELO via Mento

USAGE

```
$ celocli exchange:dollars --from <value> --value <value> [--
gasCurrency <value>]
[--globalHelp] [--forAtLeast <value>]
```

FLAGS

| | |
|--|---------------|
| --forAtLeast=1000000000000000000000 | [default: 0] |
| Optional, the | minimum value |
| of CELO to | receive in |
| return | (required) |
| --from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | Celo Dollars |
| The address with | Use a |
| to exchange | for |
| --gasCurrency=0x1234567890123456789012345678901234567890 | |
| specific gas currency | |
| transaction fees | |

CELO if no gas
supplied). It
whitelisted token.
--globalHelp
available global

--value=100000000000000000000000
The value of Celo
exchange for CELO

DESCRIPTION
Exchange Celo Dollars for CELO via Mento

EXAMPLES
dollars --value 100000000000000000000000 --from
0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d

dollars --value 100000000000000000000000 --forAtLeast 500000000000000000000000 --from
0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d

See code: src/commands/exchange/dollars.ts
celocli exchange:euros {#celocli-exchangeeuros}
Exchange Celo Euros for CELO via Mento

USAGE
\$ celocli exchange:euros --from <value> --value <value> [--gasCurrency
<value>]
[--globalHelp] [--forAtLeast <value>]

FLAGS
--forAtLeast=100000000000000000000000 [default: 0]
Optional, the
of CELO to
return
--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d (required)
The address with
exchange
--gasCurrency=0x1234567890123456789012345678901234567890 Use a
specific gas currency
transaction fees
CELO if no gas

(defaults to
currency is
must be a
View all
flags
(required)
Dollars to
minimum value
receive in
(required)
Celo Euros to
for
(defaults to

supplied). It
whitelisted token.
--globalHelp
available global

--value=1000000000000000000000
The value of Celo
exchange for CELO

DESCRIPTION
Exchange Celo Euros for CELO via Mento

EXAMPLES
euros --value 1000000000000000000000 --from
0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d

euros --value 1000000000000000000000 --forAtLeast 5000000000000000000000 --from
0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d

See code: src/commands/exchange/euros.ts

celocli exchange:reals {#celocli-exchangereals}

Exchange Celo Brazilian Real (cREAL) for CELO via Mento

USAGE
\$ celocli exchange:reals --from <value> --value <value> [--gasCurrency
<value>]
[--globalHelp] [--forAtLeast <value>]

FLAGS
--forAtLeast=1000000000000000000000000
Optional, the

of CELO to

return
--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d
The address with

Brazilian Real to

--gasCurrency=0x1234567890123456789012345678901234567890
specific gas currency

transaction fees

CELO if no gas

currency is
must be a

View all

flags
(required)

Euros to

[default: 0]

minimum value

receive in

(required)

Celo

exchange

Use a

for

(defaults to

supplied). It
whitelisted token.
--globalHelp
available global

--value=1000000000000000000000
The value of Celo
Real to exchange

currency is
must be a
View all
flags
(required)
Brazilian
for CELO

DESCRIPTION

Exchange Celo Brazilian Real (cREAL) for CELO via Mento

EXAMPLES

reals --value 1000000000000000000000 --from
0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d

reals --value 1000000000000000000000 --forAtLeast 5000000000000000000000 --from
0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d

See code: src/commands/exchange/reals.ts

celocli exchange:show {#celocli-exchangeshow}

Show the current exchange rates offered by the Broker

USAGE

\$ celocli exchange:show [--gasCurrency <value>] [--globalHelp] [--
amount <value>]

FLAGS

--amount=<value> [default:
1000000000000000000000] Amount
being exchanged of the token
rates for to report
--gasCurrency=0x1234567890123456789012345678901234567890 Use a
specific gas currency for
transaction fees (defaults to
CELO if no gas currency is
supplied). It must be a
whitelisted token.

`--globalHelp`
available global

View all
flags

DESCRIPTION

Show the current exchange rates offered by the Broker

EXAMPLES

`list`

See code: `src/commands/exchange/show.ts`

`celocli exchange:stable {#celocli-exchangestable}`

Exchange Stable Token for CELO via Mento

USAGE

`$ celocli exchange:stable --from <value> --value <value> [--gasCurrency <value>]`
`[--globalHelp] [--forAtLeast <value>] [--stableToken`
`cUSD|cusd|cEUR|ceur|cREAL|creal]`

FLAGS

`--forAtLeast=10000000000000000000000` [default: 0]
Optional, the minimum value
of CELO to receive in
return (required)
`--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d` (required)
The address with the Stable
Token to exchange
`--gasCurrency=0x1234567890123456789012345678901234567890` Use a
specific gas currency for
transaction fees (defaults to
CELO if no gas currency is
supplied). It must be a
whitelisted token. View all
`--globalHelp` flags
available global Name of the
`--stableToken=<option>` be transfered
stable token to <options:
cUSD|cusd|cEUR|ceu

```
r|cREAL|creal> --value=10000000000000000000000      (required)
The value of                                           Stable Tokens
to exchange                                          for CELo
```

| DESCRIPTION |
|--|
| Exchange Stable Token for CELo via Mento |

EXAMPLES

```
stable --value 10000000000000000 --from
0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d --stableToken
cStableTokenSymbol
```

```
stable --value 10000000000000000 --forAtLeast 5000000000000000 --from
0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d --stableToken
cStableTokenSymbol
```

See code: `src/commands/exchange/stable.ts`

```
# governance.md:
```

celocli governance
=====

Interact with on-chain governance proposals and hotfixes

```
celocli governance:approvehotfix
celocli governance:build-proposal
celocli governance:dequeue
celocli governance:execute
celocli governance:executehotfix
celocli governance:hashhotfix
celocli governance:list
celocli governance:preparehotfix
celocli governance:propose
celocli governance:revokeupvote
celocli governance:show
celocli governance:showaccount
celocli governance:showhotfix
celocli governance:upvote
celocli governance:view
celocli governance:viewaccount
celocli governance:viewhotfix
celocli governance:vote
celocli governance:votePartially
celocli governance:whitelisthotfix
celocli governance:withdraw
```

```
celocli governance:approvehotfix {#celocli-governanceapprovehotfix}
```


Approve a dequeued governance proposal (or hotfix)

USAGE

```
$ celocli governance:approvehotfix --from <value> [--gasCurrency
<value>] [--globalHelp]
  [--proposalID <value> | --hotfix <value>] [--useMultiSig]
```

FLAGS

| | |
|---|--------------|
| <code>--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Approver's | address |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| <code>--globalHelp</code> | flags |
| available global | Hash of |
| <code>--hotfix=<value></code> | UUID of |
| hotfix proposal | True means |
| <code>--proposalID=<value></code> | be sent |
| proposal to approve | |
| <code>--useMultiSig</code> | |
| the request will | |
| through multisig. | |

DESCRIPTION

Approve a dequeued governance proposal (or hotfix)

ALIASES

```
$ celocli governance:approve
$ celocli governance:approvehotfix
```

EXAMPLES

```
approve --proposalID 99 --from
0x5409ed021d9299bf6814279a6a1411a7e866a631
```

```
approve --proposalID 99 --from
0x5409ed021d9299bf6814279a6a1411a7e866a631 --useMultiSig
```

```
approve --hotfix
0xfcfc98ec3db7c56f0866a7149e811bf7f9e30c9d40008b0def497fcc6fe90649 --from
0xCc50EaC48bA71343dC76852FAE1892c6Bd2971DA --useMultiSig
```

```
celocli governance:build-proposal {#celocli-governancebuild-proposal}
```

Interactively build a governance proposal

USAGE

```
$ celocli governance:build-proposal [--gasCurrency <value>] [--globalHelp] [--output <value>]
  [--afterExecutingProposal <value> | --afterExecutingID <value>]
```

FLAGS

| | |
|---|--------------|
| <code>--afterExecutingID=<value></code> | Governance |
| proposal | identifier |
| which will be | executed |
| prior to proposal | being built |
| <code>--afterExecutingProposal=<value></code> | Path to |
| proposal which will | be executed |
| prior to | proposal |
| being built | |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | |
| <code>--globalHelp</code> | View all |
| available global | flags |
| <code>--output=<value></code> | [default: |
| proposalTransactions.json] | |
| output | Path to |

DESCRIPTION

Interactively build a governance proposal

EXAMPLES

```
build-proposal --output ./transactions.json
```

See code: src/commands/governance/build-proposal.ts

```
celocli governance:dequeue {#celocli-governancedequeue}
```

Try to dequeue governance proposal

USAGE

```
$ celocli governance:dequeue --from <value> [--gasCurrency <value>] [--globalHelp]
```

FLAGS

| | |
|---|--------------|
| <code>--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| From address | |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| <code>--globalHelp</code> | flags |
| available global | |

DESCRIPTION

Try to dequeue governance proposal

EXAMPLES

```
dequeue --from 0x5409ed021d9299bf6814279a6a1411a7e866a631
```

See code: `src/commands/governance/dequeue.ts`

```
celocli governance:execute {#celocli-governanceexecute}
```

Execute a passing governance proposal

USAGE

```
$ celocli governance:execute --proposalID <value> --from <value> [--gasCurrency <value>] [--globalHelp]
```

FLAGS

| | |
|---|--------------|
| <code>--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Executor's | address |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | |

supplied). It
whitelisted token.
--globalHelp
available global

--proposalID=<value>
UUID of proposal

currency is
must be a
View all
flags
(required)
to execute

DESCRIPTION

Execute a passing governance proposal

EXAMPLES

execute --proposalID 99 --from
0x5409ed021d9299bf6814279a6a1411a7e866a631

See code: src/commands/governance/execute.ts

celocli governance:executehotfix {#celocli-governanceexecutehotfix}

Execute a governance hotfix prepared for the current epoch

USAGE

\$ celocli governance:executehotfix --from <value> --jsonTransactions
<value> --salt <value>
[--gasCurrency <value>] [--globalHelp]

FLAGS

--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d (required)
Executors's
address
--gasCurrency=0x1234567890123456789012345678901234567890 Use a
specific gas currency
for
transaction fees
(defaults to
CELO if no gas
currency is
supplied). It
must be a
whitelisted token.
View all
--globalHelp
available global
flags
(required)
--jsonTransactions=<value>
Path to json
transactions
(required)
--salt=<value>
Secret salt

with hotfix associated

DESCRIPTION

Execute a governance hotfix prepared for the current epoch

EXAMPLES

executehotfix --jsonTransactions ./transactions.json --salt
0x614dcc5ac13cba47c2430bdee7829bb8c8f3603a8ace22e7680d317b39e3658 --from
0x5409ed021d9299bf6814279a6a1411a7e866a631

See code: src/commands/governance/executehotfix.ts

celocli governance:hashhotfix {#celocli-governancehashhotfix}

Hash a governance hotfix specified by JSON and a salt

USAGE

\$ celocli governance:hashhotfix --jsonTransactions <value> --salt
<value> [--gasCurrency
 <value>] [--globalHelp] [--force]

FLAGS

| | |
|--|--------------|
| --force | Skip |
| execution check | |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | |
| transaction fees | for |
| CELO if no gas | (defaults to |
| supplied). It | currency is |
| whitelisted token. | must be a |
| --globalHelp | View all |
| available global | |
| --jsonTransactions=<value> | flags |
| Path to json | (required) |
| of the hotfix | transactions |
| --salt=<value> | (required) |
| Secret salt | |
| with hotfix | associated |

DESCRIPTION

Hash a governance hotfix specified by JSON and a salt

EXAMPLES

```
hashhotfix --jsonTransactions ./transactions.json --salt
0x614dcc5ac13cba47c2430bdee7829bb8c8f3603a8ace22e7680d317b39e3658
```

See code: src/commands/governance/hashhotfix.ts

```
celocli governance:list {#celocli-governancelist}
```

List live governance proposals (queued and ongoing)

USAGE

```
$ celocli governance:list [--gasCurrency <value>] [--globalHelp] [--
columns <value>
  | -x] [--filter <value>] [--no-header | [--csv | --no-truncate]] [--
output
  csv|json|yaml | | ] [--sort <value>]
```

FLAGS

```
-x, --extended
  show extra columns

--columns=<value>
  only show provided columns (comma-separated)

--csv
  output is csv format [alias: --output=csv]

--filter=<value>
  filter property by partial string matching, ex: name=foo

--gasCurrency=0x1234567890123456789012345678901234567890
  Use a specific gas currency for transaction fees (defaults to CELO
if no gas
  currency is supplied). It must be a whitelisted token.

--globalHelp
  View all available global flags

--no-header
  hide table header from output

--no-truncate
  do not truncate output to fit screen

--output=<option>
  output in a more machine friendly format
  <options: csv|json|yaml>

--sort=<value>
  property to sort by (prepend '-' for descending)
```

DESCRIPTION

List live governance proposals (queued and ongoing)

EXAMPLES

list

See code: src/commands/governance/list.ts

```
celocli governance:preparehotfix {#celocli-governancepreparehotfix}
```

Prepare a governance hotfix for execution in the current epoch

USAGE

```
$ celocli governance:preparehotfix --from <value> --hash <value> [--  
gasCurrency <value>]  
[--globalHelp]
```

FLAGS

| | |
|---|--------------|
| <code>--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Preparer's | address |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| <code>--globalHelp</code> | flags |
| available global | (required) |
| <code>--hash=<value></code> | transactions |
| Hash of hotfix | |

DESCRIPTION

Prepare a governance hotfix for execution in the current epoch

EXAMPLES

```
preparehotfix --hash  
0x614dcc5ac13cba47c2430bdee7829bb8c8f3603a8ace22e7680d317b39e3658 --from  
0x5409ed021d9299bf6814279a6a1411a7e866a631
```

See code: src/commands/governance/preparehotfix.ts

```
celocli governance:propose {#celocli-governancepropose}
```

Submit a governance proposal

USAGE

```
$ celocli governance:propose --jsonTransactions <value> --deposit  
<value> --from  
  <value> --descriptionURL <value> [--gasCurrency <value>] [--  
globalHelp] [--for  
  <value> --useMultiSig] [--force] [--noInfo] [--afterExecutingProposal  
<value> |  
  --afterExecutingID <value>]
```

FLAGS

| | |
|--|--------------|
| --afterExecutingID=<value> | Governance |
| proposal | identifier |
| which will be | executed |
| prior to proposal | |
| --afterExecutingProposal=<value> | Path to |
| proposal which will | be executed |
| prior to | proposal |
| --deposit=<value> | (required) |
| Amount of Celo to | attach to |
| proposal | |
| --descriptionURL=<value> | (required) A |
| URL where | further |
| information about | the proposal |
| can be viewed | |
| --for=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | Address of |
| the multi-sig | contract |
| --force | Skip |
| execution check | |
| --from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | (required) |
| Proposer's | address |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | |
| --globalHelp | View all |
| available global | flags |
| --jsonTransactions=<value> | (required) |
| Path to json | |

| | |
|----------------------------|---------------|
| <code>--noInfo</code> | transactions |
| the proposal | Skip printing |
| <code>--useMultiSig</code> | info |
| the request will | True means |
| through multisig. | be sent |

DESCRIPTION

Submit a governance proposal

EXAMPLES

```
propose --jsonTransactions ./transactions.json --deposit 10000e18 --
from 0x5409ed021d9299bf6814279a6a1411a7e866a631 --descriptionURL
https://gist.github.com/yorhodes/46430each8ed2f73f7bf79bef9d58a33
```

```
propose --jsonTransactions ./transactions.json --deposit 10000e18 --
from 0x5409ed021d9299bf6814279a6a1411a7e866a631 --useMultiSig --for
0x6c3dDFB1A9e73B5F49eDD46624F4954Bf66CAe93 --descriptionURL
https://gist.github.com/yorhodes/46430each8ed2f73f7bf79bef9d58a33
```

See code: `src/commands/governance/propose.ts`

```
celocli governance:revokeupvote {#celocli-governancerevokeupvote}
```

Revoke upvotes for queued governance proposals

USAGE

```
$ celocli governance:revokeupvote --from <value> [--gasCurrency
<value>]
[--globalHelp]
```

FLAGS

| | |
|---|--------------|
| <code>--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Upvoter's address | |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| <code>--globalHelp</code> | flags |
| available global | |

DESCRIPTION

Revoke upvotes for queued governance proposals

EXAMPLES

```
revokeupvote --from 0x5409ed021d9299bf6814279a6a1411a7e866a631
```

See code: `src/commands/governance/revokeupvote.ts`

```
celocli governance:show {#celocli-governanceshow}
```

Show information about a governance proposal, hotfix, or account.

USAGE

```
$ celocli governance:show [--gasCurrency <value>] [--globalHelp] [--raw]
  [--jsonTransactions <value>] [--notwhitelisted] [--whitelisters | --nonwhitelisters
  | | [--proposalID <value> | --account <value> | --hotfix <value>]]
  [--afterExecutingProposal <value> | --afterExecutingID <value>]
```

FLAGS

| | |
|---|---|
| <code>--account=<value></code> | Address of account or voter |
| <code>--afterExecutingID=<value></code> | Governance proposal identifier which will be executed prior to proposal |
| <code>--afterExecutingProposal=<value></code> | Path to proposal which will be executed prior to proposal |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a specific gas currency for transaction fees (defaults to CELO if no gas currency is supplied). It must be a whitelisted token. |
| <code>--globalHelp</code> | View all available global flags |
| <code>--hotfix=<value></code> | Hash of hotfix proposal |
| <code>--jsonTransactions=<value></code> | Output proposal JSON to provided file |
| <code>--nonwhitelisters</code> | If set, displays validators |

| | |
|-------------------------|---------------|
| whitelisted | that have not |
| --notwhitelisted | the hotfix. |
| validators who have not | List |
| the specified | whitelisted |
| --proposalID=<value> | hotfix |
| proposal to view | UUID of |
| --raw | Display |
| proposal in raw | bytes format |
| --whitelisters | If set, |
| displays validators | that have |
| whitelisted the | hotfix. |

DESCRIPTION

Show information about a governance proposal, hotfix, or account.

ALIASES

```
$ celocli governance:show
$ celocli governance:showhotfix
$ celocli governance:showaccount
$ celocli governance:view
$ celocli governance:viewhotfix
$ celocli governance:viewaccount
```

EXAMPLES

```
show --proposalID 99

show --proposalID 99 --raw

show --hotfix
0x614dcc5ac13cba47c2430bdee7829bb8c8f3603a8ace22e7680d317b39e3658

show --hotfix
0x614dcc5ac13cba47c2430bdee7829bb8c8f3603a8ace22e7680d317b39e3658 --
whitelisters

show --hotfix
0x614dcc5ac13cba47c2430bdee7829bb8c8f3603a8ace22e7680d317b39e3658 --
nonwhitelisters

show --account 0x47e172f6cfb6c7d01c1574fa3e2be7cc73269d95
```

See code: `src/commands/governance/show.ts`

```
celocli governance:showaccount {#celocli-governanceshowaccount}
```

Show information about a governance proposal, hotfix, or account.

USAGE

```
$ celocli governance:showaccount [--gasCurrency <value>] [--globalHelp]
[--raw]
  [--jsonTransactions <value>] [--notwhitelisted] [--whitelisters | --
nonwhitelisters
  | | [--proposalID <value> | --account <value> | --hotfix <value>]]
  [--afterExecutingProposal <value> | --afterExecutingID <value>]
```

FLAGS

| | |
|---|---------------|
| <code>--account=<value></code> | Address of |
| account or voter | |
| <code>--afterExecutingID=<value></code> | Governance |
| proposal | identifier |
| which will be | executed |
| prior to proposal | |
| <code>--afterExecutingProposal=<value></code> | Path to |
| proposal which will | be executed |
| prior to | proposal |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| <code>--globalHelp</code> | flags |
| available global | Hash of |
| <code>--hotfix=<value></code> | Output |
| hotfix proposal | provided file |
| <code>--jsonTransactions=<value></code> | If set, |
| proposal JSON to | that have not |
| <code>--nonwhitelisters</code> | the hotfix. |
| displays validators | List |
| whitelisted | whitelisted |
| <code>--notwhitelisted</code> | hotfix |
| validators who have not | UUID of |
| the specified | |
| <code>--proposalID=<value></code> | |
| proposal to view | |

| | |
|----------------------------------|--------------|
| <code>--raw</code> | Display |
| <code>proposal in raw</code> | |
| | bytes format |
| <code>--whitelisters</code> | If set, |
| <code>displays validators</code> | |
| | that have |
| <code>whitelisted the</code> | |
| | hotfix. |

DESCRIPTION

Show information about a governance proposal, hotfix, or account.

ALIASES

```
$ celocli governance:show
$ celocli governance:showhotfix
$ celocli governance:showaccount
$ celocli governance:view
$ celocli governance:viewhotfix
$ celocli governance:viewaccount
```

EXAMPLES

```
show --proposalID 99
```

```
show --proposalID 99 --raw
```

```
show --hotfix
```

```
0x614dcc5ac13cba47c2430bdee7829bb8c8f3603a8ace22e7680d317b39e3658
```

```
show --hotfix
```

```
0x614dcc5ac13cba47c2430bdee7829bb8c8f3603a8ace22e7680d317b39e3658 --
whitelisters
```

```
show --hotfix
```

```
0x614dcc5ac13cba47c2430bdee7829bb8c8f3603a8ace22e7680d317b39e3658 --
nonwhitelisters
```

```
show --account 0x47e172f6cfb6c7d01c1574fa3e2be7cc73269d95
```

```
celocli governance:showhotfix {#celocli-governanceshowhotfix}
```

Show information about a governance proposal, hotfix, or account.

USAGE

```
$ celocli governance:showhotfix [--gasCurrency <value>] [--globalHelp]
[--raw]
  [--jsonTransactions <value>] [--notwhitelisted] [--whitelisters | --
nonwhitelisters
  | | [--proposalID <value> | --account <value> | --hotfix <value>]]
  [--afterExecutingProposal <value> | --afterExecutingID <value>]
```

FLAGS

| | |
|--|---|
| <pre> --account=<value> account or voter --afterExecutingID=<value> proposal which will be prior to proposal --afterExecutingProposal=<value> proposal which will prior to --gasCurrency=0x1234567890123456789012345678901234567890 specific gas currency transaction fees CELO if no gas supplied). It whitelisted token. --globalHelp available global --hotfix=<value> hotfix proposal --jsonTransactions=<value> proposal JSON to --nonwhitelisters displays validators whitelisted --notwhitelisted validators who have not the specified --proposalID=<value> proposal to view --raw proposal in raw --whitelisters displays validators whitelisted the </pre> | <pre> Address of Governance identifier executed Path to be executed proposal Use a for (defaults to currency is must be a View all flags Hash of Output provided file If set, that have not the hotfix. List whitelisted hotfix UUID of Display bytes format If set, that have hotfix. </pre> |
|--|---|

DESCRIPTION

Show information about a governance proposal, hotfix, or account.

ALIASES

```
$ celocli governance:show
$ celocli governance:showhotfix
$ celocli governance:showaccount
$ celocli governance:view
$ celocli governance:viewhotfix
$ celocli governance:viewaccount
```

EXAMPLES

```
show --proposalID 99

show --proposalID 99 --raw

show --hotfix
0x614dcc5ac13cba47c2430bdee7829bb8c8f3603a8ace22e7680d317b39e3658

show --hotfix
0x614dcc5ac13cba47c2430bdee7829bb8c8f3603a8ace22e7680d317b39e3658 --
whitelisters

show --hotfix
0x614dcc5ac13cba47c2430bdee7829bb8c8f3603a8ace22e7680d317b39e3658 --
nonwhitelisters

show --account 0x47e172f6cfb6c7d01c1574fa3e2be7cc73269d95
```

```
celocli governance:upvote {#celocli-governanceupvote}
```

Upvote a queued governance proposal

USAGE

```
$ celocli governance:upvote --proposalID <value> --from <value> [--
gasCurrency
<value>] [--globalHelp]
```

FLAGS

| | |
|--|------------------|
| --from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | (required) |
| Upvoter's address | |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| --globalHelp | available global |
| | flags |

--proposalID=<value> (required)
UUID of proposal
to upvote

DESCRIPTION

Upvote a queued governance proposal

EXAMPLES

upvote --proposalID 99 --from
0x5409ed021d9299bf6814279a6a1411a7e866a631

See code: src/commands/governance/upvote.ts

celocli governance:view {#celocli-governanceview}

Show information about a governance proposal, hotfix, or account.

USAGE

```
$ celocli governance:view [--gasCurrency <value>] [--globalHelp] [--raw]
    [--jsonTransactions <value>] [--notwhitelisted] [--whitelisters | --nonwhitelisters
    | | [--proposalID <value> | --account <value> | --hotfix <value>]]
    [--afterExecutingProposal <value> | --afterExecutingID <value>]
```

FLAGS

| | |
|--|--------------|
| --account=<value> | Address of |
| account or voter | |
| --afterExecutingID=<value> | Governance |
| proposal | identifier |
| which will be | executed |
| prior to proposal | |
| --afterExecutingProposal=<value> | Path to |
| proposal which will | be executed |
| prior to | proposal |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| --globalHelp | flags |
| available global | |

| | |
|---|---------------|
| <code>--hotfix=<value></code> | Hash of |
| <code>hotfix proposal</code> | |
| <code>--jsonTransactions=<value></code> | Output |
| <code>proposal JSON to</code> | |
| <code>--nonwhitelisters</code> | provided file |
| <code>displays validators</code> | If set, |
| | that have not |
| <code>whitelisted</code> | |
| | the hotfix. |
| <code>--notwhitelisted</code> | List |
| <code>validators who have not</code> | |
| | whitelisted |
| <code>the specified</code> | |
| <code>--proposalID=<value></code> | hotfix |
| <code>proposal to view</code> | UUID of |
| <code>--raw</code> | |
| <code>proposal in raw</code> | Display |
| | |
| <code>--whitelisters</code> | bytes format |
| <code>displays validators</code> | If set, |
| | that have |
| <code>whitelisted the</code> | |
| | hotfix. |

DESCRIPTION

Show information about a governance proposal, hotfix, or account.

ALIASES

```
$ celocli governance:show
$ celocli governance:showhotfix
$ celocli governance:showaccount
$ celocli governance:view
$ celocli governance:viewhotfix
$ celocli governance:viewaccount
```

EXAMPLES

```
show --proposalID 99

show --proposalID 99 --raw

show --hotfix
0x614dcc5ac13cba47c2430bdee7829bb8c8f3603a8ace22e7680d317b39e3658

show --hotfix
0x614dcc5ac13cba47c2430bdee7829bb8c8f3603a8ace22e7680d317b39e3658 --
whitelisters

show --hotfix
0x614dcc5ac13cba47c2430bdee7829bb8c8f3603a8ace22e7680d317b39e3658 --
nonwhitelisters

show --account 0x47e172f6cfb6c7d01c1574fa3e2be7cc73269d95
```

```
celocli governance:viewaccount {#celocli-governanceviewaccount}
```

Show information about a governance proposal, hotfix, or account.

USAGE

```
$ celocli governance:viewaccount [--gasCurrency <value>] [--globalHelp]
[--raw]
  [--jsonTransactions <value>] [--notwhitelisted] [--whitelisters | --
nonwhitelisters
  | | [--proposalID <value> | --account <value> | --hotfix <value>]]
  [--afterExecutingProposal <value> | --afterExecutingID <value>]
```

FLAGS

| | |
|--|---------------|
| --account=<value> | Address of |
| account or voter | |
| --afterExecutingID=<value> | Governance |
| proposal | identifier |
| which will be | executed |
| prior to proposal | |
| --afterExecutingProposal=<value> | Path to |
| proposal which will | be executed |
| prior to | proposal |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| --globalHelp | flags |
| available global | Hash of |
| --hotfix=<value> | Output |
| hotfix proposal | |
| --jsonTransactions=<value> | provided file |
| proposal JSON to | If set, |
| --nonwhitelisters | that have not |
| displays validators | the hotfix. |
| whitelisted | List |
| --notwhitelisted | |
| validators who have not | |

| | |
|----------------------|--------------|
| the specified | whitelisted |
| --proposalID=<value> | hotfix |
| proposal to view | UUID of |
| --raw | Display |
| proposal in raw | bytes format |
| --whitelisters | If set, |
| displays validators | that have |
| whitelisted the | hotfix. |

DESCRIPTION

Show information about a governance proposal, hotfix, or account.

ALIASES

```
$ celocli governance:show
$ celocli governance:showhotfix
$ celocli governance:showaccount
$ celocli governance:view
$ celocli governance:viewhotfix
$ celocli governance:viewaccount
```

EXAMPLES

```
show --proposalID 99

show --proposalID 99 --raw

show --hotfix
0x614dcc5ac13cba47c2430bdee7829bb8c8f3603a8ace22e7680d317b39e3658

show --hotfix
0x614dcc5ac13cba47c2430bdee7829bb8c8f3603a8ace22e7680d317b39e3658 --
whitelisters

show --hotfix
0x614dcc5ac13cba47c2430bdee7829bb8c8f3603a8ace22e7680d317b39e3658 --
nonwhitelisters

show --account 0x47e172f6cfb6c7d01c1574fa3e2be7cc73269d95
```

```
celocli governance:viewhotfix {#celocli-governanceviewhotfix}
```

Show information about a governance proposal, hotfix, or account.

USAGE

```
$ celocli governance:viewhotfix [--gasCurrency <value>] [--globalHelp]
[--raw]
[--jsonTransactions <value>] [--notwhitelisted] [--whitelisters | --
nonwhitelisters]
```

```
| | [--proposalID <value> | --account <value> | --hotfix <value>]]
[--afterExecutingProposal <value> | --afterExecutingID <value>]
```

FLAGS

| | |
|---|---------------|
| --account=<value>
account or voter | Address of |
| --afterExecutingID=<value>
proposal | Governance |
| | identifier |
| which will be | executed |
| prior to proposal | |
| --afterExecutingProposal=<value>
proposal which will | Path to |
| | be executed |
| prior to | proposal |
| --gasCurrency=0x1234567890123456789012345678901234567890
specific gas currency | Use a |
| | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| --globalHelp
available global | flags |
| --hotfix=<value>
hotfix proposal | Hash of |
| --jsonTransactions=<value>
proposal JSON to | Output |
| --nonwhitelisters
displays validators | provided file |
| | If set, |
| whitelisted | that have not |
| | the hotfix. |
| --notwhitelisted
validators who have not | List |
| | whitelisted |
| the specified | hotfix |
| --proposalID=<value>
proposal to view | UUID of |
| --raw
proposal in raw | Display |
| | bytes format |
| --whitelisters
displays validators | If set, |
| | that have |
| whitelisted the | |

hotfix.

DESCRIPTION

Show information about a governance proposal, hotfix, or account.

ALIASES

```
$ celocli governance:show
$ celocli governance:showhotfix
$ celocli governance:showaccount
$ celocli governance:view
$ celocli governance:viewhotfix
$ celocli governance:viewaccount
```

EXAMPLES

```
show --proposalID 99
```

```
show --proposalID 99 --raw
```

```
show --hotfix
```

```
0x614dcc5ac13c4a47c2430bdee7829bb8c8f3603a8ace22e7680d317b39e3658
```

```
show --hotfix
```

```
0x614dcc5ac13c4a47c2430bdee7829bb8c8f3603a8ace22e7680d317b39e3658 --
whitelisters
```

```
show --hotfix
```

```
0x614dcc5ac13c4a47c2430bdee7829bb8c8f3603a8ace22e7680d317b39e3658 --
nonwhitelisters
```

```
show --account 0x47e172f6cfb6c7d01c1574fa3e2be7cc73269d95
```

```
celocli governance:vote {#celocli-governancevote}
```

Vote on an approved governance proposal

USAGE

```
$ celocli governance:vote --proposalID <value> --value Abstain|No|Yes -
-from
  <value> [--gasCurrency <value>] [--globalHelp]
```

FLAGS

```
--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d (required)
Voter's address
--gasCurrency=0x1234567890123456789012345678901234567890 Use a
specific gas currency for
transaction fees (defaults to
CELO if no gas currency is
supplied). It
```

| | |
|---|---------------|
| whitelisted token. | must be a |
| --globalHelp | View all |
| available global | |
| --proposalID=<value> | flags |
| UUID of proposal | (required) |
| --value=<option> | to vote on |
| Vote | (required) |
| | <options: |
| Abstain No Yes> | |
| DESCRIPTION | |
| Vote on an approved governance proposal | |
| EXAMPLES | |
| vote --proposalID 99 --value Yes --from | |
| 0x5409ed021d9299bf6814279a6a1411a7e866a631 | |
| See code: src/commands/governance/vote.ts | |
| celocli governance:votePartially {#celocli-governancevotepartially} | |
| Vote partially on an approved governance proposal | |
| USAGE | |
| \$ celocli governance:votePartially --proposalID <value> --from <value> | |
| [--gasCurrency | |
| <value>] [--globalHelp] [--yes <value>] [--no <value>] [--abstain | |
| <value>] | |
| FLAGS | |
| --abstain=<value> | Abstain votes |
| --from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | (required) |
| Voter's address | |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | |
| | for |
| transaction fees | |
| | (defaults to |
| CELO if no gas | |
| | currency is |
| supplied). It | |
| | must be a |
| whitelisted token. | |
| --globalHelp | View all |
| available global | |
| --no=<value> | flags |
| --proposalID=<value> | No votes |
| UUID of proposal | (required) |

| | |
|--------------------------------|---------------------------------|
| <pre>--yes=<value></pre> | <pre>to vote on Yes votes</pre> |
|--------------------------------|---------------------------------|

DESCRIPTION

Vote partially on an approved governance proposal

EXAMPLES

```
vote-partially --proposalID 99 --yes 10 --no 20 --from
0x5409ed021d9299bf6814279a6a1411a7e866a631
```

See code: src/commands/governance/votePartially.ts

```
celocli governance:whitelisthotfix {#celocli-governancewhitelisthotfix}
```

Whitelist a governance hotfix

USAGE

```
$ celocli governance:whitelisthotfix --from <value> --hash <value> [--
gasCurrency <value>]
[--globalHelp]
```

FLAGS

| | |
|---|-----------------------------|
| <pre>--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d Whitelister's</pre> | <pre>(required)</pre> |
| <pre>--gasCurrency=0x1234567890123456789012345678901234567890 specific gas currency</pre> | <pre>address Use a</pre> |
| <pre>transaction fees</pre> | <pre>for</pre> |
| <pre>CELO if no gas</pre> | <pre>(defaults to</pre> |
| <pre>supplied). It</pre> | <pre>currency is</pre> |
| <pre>whitelisted token.</pre> | <pre>must be a</pre> |
| <pre>--globalHelp available global</pre> | <pre>View all</pre> |
| <pre>--hash=<value> Hash of hotfix</pre> | <pre>flags (required)</pre> |
| | <pre>transactions</pre> |

DESCRIPTION

Whitelist a governance hotfix

EXAMPLES

```
whitelisthotfix --hash
0x614dcc5ac13cba47c2430bdee7829bb8c8f3603a8ace22e7680d317b39e3658 --from
0x5409ed021d9299bf6814279a6a1411a7e866a631
```

See code: src/commands/governance/whitelisthotfix.ts

```
celocli governance:withdraw {#celocli-governancewithdraw}
```

Withdraw refunded governance proposal deposits.

USAGE

```
$ celocli governance:withdraw --from <value> [--gasCurrency <value>] [-  
-globalHelp]
```

FLAGS

| | |
|---|--------------|
| <code>--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Proposer's | address |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| <code>--globalHelp</code> | flags |
| available global | |

DESCRIPTION

Withdraw refunded governance proposal deposits.

EXAMPLES

```
withdraw --from 0x5409ed021d9299bf6814279a6a1411a7e866a631
```

See code: `src/commands/governance/withdraw.ts`

grandamento.md:

```
celocli grandamento  
=====
```

Cancels a Granda Mento exchange proposal

```
celocli grandamento:cancel  
celocli grandamento:execute  
celocli grandamento:get-buy-amount  
celocli grandamento:list  
celocli grandamento:propose  
celocli grandamento:show
```

```
celocli grandamento:cancel {#celocli-grandamentocancel}
```


Cancels a Granda Mento exchange proposal

USAGE

```
$ celocli grandamento:cancel --from <value> (--proposalID <value> | |  
)  
    [--gasCurrency <value>] [--globalHelp]
```

FLAGS

| | |
|---|--------------|
| <code>--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| The address | |
| cancel the | allowed to |
| | proposal |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| <code>--globalHelp</code> | flags |
| available global | (required) |
| <code>--proposalID=<value></code> | to view |
| UUID of proposal | |

DESCRIPTION

Cancels a Granda Mento exchange proposal

```
celocli grandamento:execute {#celocli-grandamentoexecute}
```

Executes a Granda Mento exchange proposal

USAGE

```
$ celocli grandamento:execute --from <value> --proposalID <value> [--  
gasCurrency  
    <value>] [--globalHelp]
```

FLAGS

| | |
|---|-------------|
| <code>--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| The address to | execute the |
| exchange | proposal |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | |

| | |
|----------------------|--------------|
| transaction fees | for |
| CELO if no gas | (defaults to |
| supplied). It | currency is |
| whitelisted token. | must be a |
| --globalHelp | View all |
| available global | flags |
| --proposalID=<value> | (required) |
| UUID of proposal | to view |

DESCRIPTION

Executes a Granda Mento exchange proposal

celocli grandamento:get-buy-amount {#celocli-grandamentoget-buy-amount}

Gets the buy amount for a prospective Granda Mento exchange

USAGE

```
$ celocli grandamento:get-buy-amount --value <value> --stableToken
  cUSD|cusd|cEUR|ceur|cREAL|creal --sellCelo [--gasCurrency <value>] [-
  -globalHelp]
```

FLAGS

| | |
|--|--------------|
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| --globalHelp | flags |
| available global | (required) |
| --sellCelo | (required) |
| Sell or buy CELO | Name of the |
| --stableToken=<option> | receive or |
| [default: cusd] | <options: |
| stable to | |
| send | |
| cUSD cusd cEUR ceu | |

```
r|cREAL|creal>
--value=1000000000000000000000          (required)
The value of the
tokens to
exchange
```

```
celocli grandamento:list {#celocli-grandamentolist}
```

```

USAGE
  $ celocli grandamento:list [--gasCurrency <value>] [--globalHelp]

```

| DESCRIPTION |
|---|
| List current active Granda Mento exchange proposals |

Proposes a Granda Mento exchange

```

FLAGS
  --from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d      (required)
The address with
tokens to
exchange

```

```
--gasCurrency=0x1234567890123456789012345678901234567890    Use a specific gas currency  
  
transaction fees                                                for CELO if no gas supplied). It must be a whitelisted token.  
--globalHelp                                                    View all available global flags (required)  
--sellCelo                                                       Sell or buy CELOR  
--stableToken=<option>                                           (required) [default: cUSD] Name of the stable coin to receive or send <options:  
cUSD|cusd|cEUR|ceu  
  
r|creal|creal>  
--value=100000000000000000000                                (required)  
The value of the tokens to exchange
```

| | |
|-------------|----------------------------------|
| DESCRIPTION | Proposes a Granda Mento exchange |
|-------------|----------------------------------|

`celocli grandamento:show {#celocli-grandamentoshow}`

Shows details of a Granda Mento exchange proposal

```

USAGE
$ celocli grandamento:show --proposalID <value> [--gasCurrency <value>]
  [--globalHelp]

```

[illegible]

| | |
|---|------------|
| <code>--globalHelp</code> | View all |
| available global | |
| <code>--proposalID=<value></code> | flags |
| UUID of proposal | (required) |
| | to view |

DESCRIPTION

Shows details of a Granda Mento exchange proposal

help.md:

```
celocli help
=====
```

Display help for celocli.

```
[celocli help [COMMANDS]](#celocli-help-commands)
```

```
celocli help [COMMANDS] {#celocli-help-commands}
```

Display help for celocli.

USAGE

```
$ celocli help [COMMANDS] [-n]
```

ARGUMENTS

COMMANDS Command to show help for.

FLAGS

-n, --nested-commands Include all nested commands in the output.

DESCRIPTION

Display help for celocli.

See code: @oclif/plugin-help

identity.md:

```
celocli identity
=====
```

Interact with ODIS and the attestations service

```
celocli identity:get-attestations
```

```
celocli identity:identifier
```

```
celocli identity:withdraw-attestation-rewards
```

```
celocli identity:get-attestations {#celocli-identityget-attestations}
```

Looks up attestations associated with the provided phone number. If a pepper is not provided, it uses the --from account's balance to query the pepper.

USAGE

```
$ celocli identity:get-attestations [--gasCurrency <value>] [--globalHelp] [--phoneNumber <value>] [--from <value>] [--pepper <value>] [--identifier <value>] [--network <value>]
```

FLAGS

| | |
|--|---------------|
| --from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | Account whose |
| balance to use | |
| | for querying |
| ODIS for the | |
| | pepper lookup |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | |
| | for |
| transaction fees | |
| | (defaults to |
| CELO if no gas | |
| | currency is |
| supplied). It | |
| | must be a |
| whitelisted token. | |
| --globalHelp | View all |
| available global | |
| | flags |
| --identifier=<value> | On-chain |
| identifier | |
| --network=<value> | The ODIS |
| service to hit: | |
| | mainnet, |
| alfajores, | |
| alfajoresstaging | |
| --pepper=<value> | ODIS phone |
| number pepper | |
| --phoneNumber=<value> | Phone number |
| to check | |
| | attestations |
| for | |

DESCRIPTION

Looks up attestations associated with the provided phone number. If a pepper is not provided, it uses the --from account's balance to query the pepper.

EXAMPLES

```
get-attestations --phoneNumber +15555555555 --from
0x47e172F6CfB6c7D01C1574fa3E2Be7CC73269D95
```

```
get-attestations --phoneNumber +15555555555 --pepper XgnKVpplZc0p1
```

```
get-attestations --identifier
0x4952c9db9c283a62721b13f56c4b5e84a438e2569af3de21cb3440efa8840872
```

See code: src/commands/identity/get-attestations.ts

```
celocli identity:identifier {#celocli-identityidentifier}
```

Queries ODIS for the on-chain identifier and pepper corresponding to a given phone number.

USAGE

```
$ celocli identity:identifier --from <value> --phoneNumber <value> [--
gasCurrency
  <value>] [--globalHelp] [--context <value>]
```

FLAGS

| | |
|--|---------------|
| --context=<value> | mainnet |
| (default), | alfajores, or |
| alfajoresstaging | |
| --from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | (required) |
| The address from | which to |
| perform the query | |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| --globalHelp | flags |
| available global | (required) |
| --phoneNumber=+14152223333 | for which to |
| The phone number | identifier. |
| query the | e164 format |
| Should be in | code. |
| with country | |

DESCRIPTION

Queries ODIS for the on-chain identifier and pepper corresponding to a given phone number.

EXAMPLES

```
identifier --phoneNumber +14151231234 --from
0x5409ed021d9299bf6814279a6a1411a7e866a631 --context alfajores
```

See code: `src/commands/identity/identifier.ts`

```
celocli identity:withdraw-attestation-rewards {#celocli-identitywithdraw-attestation-rewards}
```

Withdraw accumulated attestation rewards for a given currency

USAGE

```
$ celocli identity:withdraw-attestation-rewards --from <value> [--
gasCurrency <value>] [--globalHelp]
[--tokenAddress <value>]
```

FLAGS

```
--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d
    (required) Address to withdraw from. Can be the attestation signer
address or the
    underlying account address

--gasCurrency=0x1234567890123456789012345678901234567890
    Use a specific gas currency for transaction fees (defaults to CEL0
if no gas
    currency is supplied). It must be a whitelisted token.

--globalHelp
    View all available global flags

--tokenAddress=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d
    The address of the token that will be withdrawn. Defaults to cUSD
```

DESCRIPTION

Withdraw accumulated attestation rewards for a given currency

See code: `src/commands/identity/withdraw-attestation-rewards.ts`

index.md:

title: Celo CLI

description: The Command Line Interface allows users to interact with the Celo Protocol smart contracts.

sidebarposition: 1

Celo CLI

Introduction to the Celo Command Line Interface and installation instructions.

What is the Celo CLI {#what-is-the-celo-cli}

The Command Line Interface allows users to interact with the Celo Protocol smart contracts.

It's a command-line interface around the ContractKit. It allows you to interact with the Celo Protocol and smart contracts using command-line tools rather than writing JavaScript. It provides modules for interacting with modules on the ContractKit and is an excellent code reference when defining your own modules. Some common features you may want to consider are helping users participate in elections or in on-chain governance, voting for validators, or helping users interact with multi-sig contracts.

NPM Package {#npm-package}

The Celo CLI is published as a node module on NPM. Assuming you have npm and git both installed, you can install the Celo CLI using the following command:

```
bash
npm install -g @celo/celocli
```

:::info

If you have trouble installing globally \ (i.e. with the -g flag\), try installing to a local directory instead with yarn add @celo/celocli and run with yarn celocli.

:::

Commands {#commands}

The tool is broken down into modules and commands with the following pattern:

```
text
celocli <module>:<command> <...args> <...flags?>
```

The celocli tool assumes that users are running a node which they have access to signing transactions on, or have another mechanism for signing transactions (such as a Ledger wallet or supplying the private key as an argument to the command). See the documentation on the config module for information about how to set which node commands are sent to.

:::info

All balances of CELO or Celo Dollars are expressed in units of 10^{-18} .

:::

You can find the Celo CLI package on NPM here.

To see all available commands, run `celocli commands`.

To see all available flags for a command, add the flag `--globalHelp` to the command.

Optional: Run a Full Node {#optional-run-a-full-node}

Commands need to connect to a Celo node to execute most functionality. You can either use Forno (this is the easiest way) or run your own full node if you prefer. See the Running a Full Node instructions for more details on running a full node.

The easiest way to connect `celocli` to the Celo network is by running the following command in your terminal with `celocli` installed:

```
bash
celocli config:set --node=https://forno.celo.org
```

You can verify that `celocli` is connected by running

```
bash
celocli config:get
```

Import Accounts {#import-accounts}

If you are connecting to a remote node (like Forno), Celo CLI will need to sign transactions locally before sending them. To do this, Celo CLI needs access to a private key. There are a couple ways to sign transactions using Celo CLI.

Import Private Key (less secure) {#import-private-key-less-secure}

Add the `--privateKey` flag followed by the private key associated with the sending account. For example:

```
shell
celocli transfer:celo --from <accountAddress> --to <addressOfChoice> --
value <valueInCeloWei> --privateKey <privateKey> --node
https://forno.celo.org
```

Or you can use a Ledger hardware wallet. (preferred, see below)

Using a Ledger Wallet {#using-a-ledger-wallet}

The Celo CLI supports using a Ledger hardware wallet to sign transactions. Just add the `--useLedger` flag to a command that requires a signature.

You can specify the number of addresses to get for local signing with the `--ledgerAddresses` flag.

You can specify an array of index addresses for local signing. Example `--ledgerCustomAddresses "[4,99]"`.

For example:

```
shell
celocli transfer:celo --to <addressOfChoice> --value 1000000 --from
<accountAddress> --useLedger
```

Plugins {#plugins}

Additional plugins can be installed which make the CLI experience smoother. Currently, `celocli` only supports installing plugins published on NPM within the `@celo/` and `@clabs/` scopes.

:::danger

Installing a 3rd party plugin can be dangerous! Please always be sure that you trust the plugin provider.

:::

The autocomplete plugin adds an interactive autocomplete for bash and zsh shells. To enable the autocomplete plugin, follow the instructions provided at:

```
text
celocli autocomplete
```

The update warning plugin notifies the user if they are using an outdated version of the CLI. This plugin is enabled by default.

lockedgold.md:

```
celocli lockedgold
=====
```

View and manage locked CELO

```
celocli lockedgold:delegate
celocli lockedgold:delegate-info
celocli lockedgold:lock
```

```
celocli lockedgold:max-delegatees-count
celocli lockedgold:revoke-delegate
celocli lockedgold:show ARG1
celocli lockedgold:unlock
celocli lockedgold:update-delegated-amount
celocli lockedgold:withdraw
```

```
celocli lockedgold:delegate {#celocli-lockedgoldddelegate}
```

Delegate locked celo.

USAGE

```
$ celocli lockedgold:delegate --from <value> --to <value> --percent
<value>
  [--gasCurrency <value>] [--globalHelp]
```

FLAGS

```
--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d      (required)
Account Address
--gasCurrency=0x1234567890123456789012345678901234567890 Use a
specific gas currency                                     for
transaction fees                                         (defaults to
CELO if no gas                                           currency is
supplied). It                                           must be a
whitelisted token.                                       View all
--globalHelp                                             flags
available global                                         (required) 1-
--percent=<value>                                         celo to be
100% of locked                                           (required)
delegated
--to=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d
Account Address
```

DESCRIPTION

Delegate locked celo.

EXAMPLES

```
delegate --from 0x47e172F6CfB6c7D01C1574fa3E2Be7CC73269D95 --to
0xc0ffee254729296a45a3885639AC7E10F9d54979 --percent 100
```

See code: src/commands/lockedgold/delegate.ts

```
celocli lockedgold:delegate-info {#celocli-lockedgoldddelegate-info}
```

Delegate info about account.

```
$ celocli lockedgold:delegate-info --account <value> [--gasCurrency
<value>]
[--globalHelp]
```

| | |
|--|---|
| --account=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFFf232d | (required) |
| Account Address | |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a specific gas currency |
| | for transaction fees |
| | (defaults to CELO if no gas supplied). It |
| | must be a whitelisted token. |
| --globalHelp | View all available global flags |

Delegate info about account.

```
delegate-info --account 0x47e172F6CfB6c7D01C1574fa3E2Be7CC73269D95
```

```
celocli lockedgold:lock {#celocli-lockedgoldlock}
```

USAGE

```
$ celocli lockedgold:lock --from <value> --value <value> [--gasCurrency
<value>]
    [--globalHelp]
```

[illegible]

whitelisted token.

```
--globalHelp
available global
```

```
--value=<value>
```

The unit amount

of CELO

DESCRIPTION

Locks CELO to be used in governance and validator elections.

EXAMPLES

```
lock --from 0x47e172f6cfB6c7D01C1574fa3E2Be7CC73269D95 --value  
1000000000000000000000
```

See code: `src/commands/lockedgold/lock.ts`

```
celocli lockedgold:max-delegates-count {#celocli-lockedgoldmax-
delegates-count}
```

Returns the maximum number of delegates allowed per account.

USAGE

```
$ celocli lockedgold:max-delegates-count [--gasCurrency <value>] [--globalHelp]
```

FLAGS

```
--gasCurrency=0x123456789012345678901234567890 Use a
specific gas currency                                for
transaction fees                                    (defaults to
CELO if no gas                                       currency is
supplied). It                                        must be a
whitelisted token.                                  View all
  --globalHelp                                       available global
available global                                     flags
```

DESCRIPTION

Returns the maximum number of delegates allowed per account.

EXAMPLES

max-delegates-count

See code: `src/commands/lockedgold/max-delegates-count.ts`

```
celocli lockedgold:revoke-delegate {#celocli-lockedgoldrevoke-delegate}
```

Revoke delegated locked celo.

USAGE

```
$ celocli lockedgold:revoke-delegate --from <value> --to <value> --percent <value>
    [--gasCurrency <value>] [--globalHelp]
```

FLAGS

| | |
|---|---|
| <code>--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Account Address | |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a specific gas currency for transaction fees (defaults to CELO if no gas supplied). It must be a whitelisted token. |
| <code>--globalHelp</code> | View all available global flags |
| <code>--percent=<value></code> | (required) 1-100% of locked celo to be revoked from currently delegated amount |
| <code>--to=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Account Address | |

DESCRIPTION

Revoke delegated locked celo.

EXAMPLES

```
revoke-delegate --from 0x47e172F6CfB6c7D01C1574fa3E2Be7CC73269D95 --to 0xc0ffee254729296a45a3885639AC7E10F9d54979 --percent 100
```

See code: `src/commands/lockedgold/revoke-delegate.ts`

```
celocli lockedgold:show ARG1 {#celocli-lockedgoldshow-arg1}
```

Show Locked Gold information for a given account. This includes the total amount of locked celo, the amount being used for voting in Validator Elections, the Locked Gold balance this account is required to maintain due to a registered Validator or Validator Group, and any pending withdrawals that have been initiated via "lockedgold:unlock".

USAGE

| | |
|--------------------|--------------|
| CELO if no gas | (defaults to |
| supplied). It | currency is |
| whitelisted token. | must be a |
| --globalHelp | View all |
| available global | flags |
| --value=<value> | (required) |
| The unit amount | of CELO |

DESCRIPTION

Unlocks CELO, which can be withdrawn after the unlocking period. Unlocked celo will appear as a "pending withdrawal" until the unlocking period is over, after which it can be withdrawn via "lockedgold:withdraw".

EXAMPLES

```
unlock --from 0x47e172F6CfB6c7D01C1574fa3E2Be7CC73269D95 --value 500000000
```

See code: `src/commands/lockedgold/unlock.ts`

```
celocli lockedgold:update-delegated-amount {#celocli-lockedgoldupdate-delegated-amount}
```

Updates the amount of delegated locked celo. There might be discrepancy between the amount of locked celo and the amount of delegated locked celo because of received rewards.

USAGE

```
$ celocli lockedgold:update-delegated-amount --from <value> --to <value> [--gasCurrency <value>] [--globalHelp]
```

FLAGS

| | |
|--|--------------|
| --from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | (required) |
| Account Address | |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | |

--globalHelp View all
available global
--to=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d flags
Account Address (required)

DESCRIPTION

Updates the amount of delegated locked celo. There might be discrepancy between the amount of locked celo and the amount of delegated locked celo because of received rewards.

EXAMPLES

```
update-delegated-amount --from  
0x47e172F6CfB6c7D01C1574fa3E2Be7CC73269D95 --to  
0xc0ffee254729296a45a3885639AC7E10F9d54979
```

See code: src/commands/lockedgold/update-delegated-amount.ts

```
celocli lockedgold:withdraw {#celocli-lockedgoldwithdraw}
```

Withdraw any pending withdrawals created via "lockedgold:unlock" that have become available.

USAGE

```
$ celocli lockedgold:withdraw --from <value> [--gasCurrency <value>] [-  
-globalHelp]
```

FLAGS

--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d (required)
Account Address
--gasCurrency=0x1234567890123456789012345678901234567890 Use a
specific gas currency for
transaction fees (defaults to
CELO if no gas currency is
supplied). It must be a
whitelisted token. View all
--globalHelp flags
available global

DESCRIPTION

Withdraw any pending withdrawals created via "lockedgold:unlock" that have become available.

EXAMPLES

```
withdraw --from 0x47e172F6CfB6c7D01C1574fa3E2Be7CC73269D95
```

See code: `src/commands/lockedgold/withdraw.ts`

multisig.md:

```
celocli multisig
=====
```

Approves an existing transaction on a multi-sig contract

```
celocli multisig:approve
celocli multisig:show ARG1
celocli multisig:transfer ARG1
```

```
celocli multisig:approve {#celocli-multisigapprove}
```

Approves an existing transaction on a multi-sig contract

USAGE

```
$ celocli multisig:approve --from <value> --for <value> --tx <value> [-
-gasCurrency
  <value>] [--globalHelp]
```

FLAGS

| | |
|---|---------------|
| <code>--for=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Address of the | |
| contract | multi-sig |
| <code>--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Account approving | |
| transaction | the multi-sig |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | |
| transaction fees | for |
| CELO if no gas | (defaults to |
| supplied). It | currency is |
| whitelisted token. | must be a |
| <code>--globalHelp</code> | View all |
| available global | |
| <code>--tx=<value></code> | flags |
| Transaction to | (required) |
| | approve |

DESCRIPTION

Approves an existing transaction on a multi-sig contract

EXAMPLES

```
approve --from 0x6ecbe1db9ef729cbe972c83fb886247691fb6beb --for
0x5409ed021d9299bf6814279a6a1411a7e866a631 --tx 3
```

See code: src/commands/multisig/approve.ts

```
celocli multisig:show ARG1 {#celocli-multisigshow-arg1}
```

Shows information about multi-sig contract

USAGE

```
$ celocli multisig:show ARG1 [--gasCurrency <value>] [--globalHelp] [--
tx
    <value>] [--all] [--raw]
```

FLAGS

| | |
|--|---------------|
| --all | Show info |
| about all | transactions |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| --globalHelp | flags |
| available global | Do not |
| --raw | transactions |
| attempt to parse | Show info for |
| --tx=<value> | |
| a transaction | |

DESCRIPTION

Shows information about multi-sig contract

EXAMPLES

```
show 0x5409ed021d9299bf6814279a6a1411a7e866a631

show 0x5409ed021d9299bf6814279a6a1411a7e866a631 --tx 3

show 0x5409ed021d9299bf6814279a6a1411a7e866a631 --all --raw
```

See code: src/commands/multisig/show.ts

```
celocli multisig:transfer ARG1 {#celocli-multisigtransfer-arg1}
```

Ability to approve CELO transfers to and from multisig. Submit transaction or approve a matching existing transaction

USAGE

```
$ celocli multisig:transfer ARG1 --to <value> --amount <value> --from <value>
  [--gasCurrency <value>] [--globalHelp] [--transferFrom] [--sender <value>]
```

FLAGS

| | |
|---|--------------|
| <code>--amount=<value></code> | (required) |
| Amount to | |
| | transfer, |
| e.g. 10e18 | |
| <code>--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Account | |
| | transferring |
| value to the | |
| | recipient |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | |
| | for |
| transaction fees | |
| | (defaults to |
| CELO if no gas | |
| | currency is |
| supplied). It | |
| | must be a |
| whitelisted token. | |
| <code>--globalHelp</code> | View all |
| available global | |
| | flags |
| <code>--sender=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | Identify |
| sender if | |
| | performing |
| <code>transferFrom</code> | |
| <code>--to=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Recipient of | |
| | transfer |
| <code>--transferFrom</code> | Perform |
| <code>transferFrom</code> instead | |
| | of transfer |
| in the ERC-20 | |
| | interface |

DESCRIPTION

Ability to approve CELO transfers to and from multisig. Submit transaction or approve a matching existing transaction

EXAMPLES

```
transfer <multiSigAddr> --to 0x5409ed021d9299bf6814279a6a1411a7e866a631
--amount 200000e18 --from 0x123abc
```

```
transfer <multiSigAddr> --transferFrom --sender 0x123abc --to
0x5409ed021d9299bf6814279a6a1411a7e866a631 --amount 200000e18 --from
0x123abc
```

See code: `src/commands/multisig/transfer.ts`

network.md:

```
celocli network
=====
```

View details about the network, like contracts and parameters

```
celocli network:contracts
celocli network:info
celocli network:parameters
celocli network:whitelist
```

```
celocli network:contracts {#celocli-networkcontracts}
```

Lists Celo core contracts and their addresses.

USAGE

```
$ celocli network:contracts [--gasCurrency <value>] [--globalHelp] [--
columns <value>
  | -x] [--filter <value>] [--no-header | [--csv | --no-truncate]] [--
output
  csv|json|yaml | | ] [--sort <value>]
```

FLAGS

```
-x, --extended
  show extra columns

--columns=<value>
  only show provided columns (comma-separated)

--csv
  output is csv format [alias: --output=csv]

--filter=<value>
  filter property by partial string matching, ex: name=foo

--gasCurrency=0x1234567890123456789012345678901234567890
  Use a specific gas currency for transaction fees (defaults to CEL0
if no gas
  currency is supplied). It must be a whitelisted token.
```

--globalHelp
View all available global flags

--no-header
hide table header from output

--no-truncate
do not truncate output to fit screen

--output=<option>
output in a more machine friendly format
<options: csv|json|yaml>

--sort=<value>
property to sort by (prepend '-' for descending)

DESCRIPTION

Lists Celo core contracts and their addresses.

See code: src/commands/network/contracts.ts

celocli network:info {#celocli-networkinfo}

View general network information such as the current block number

USAGE

\$ celocli network:info [--gasCurrency <value>] [--globalHelp] [-n <value>]

FLAGS

-n, --lastN=<value>
[default: 1] Fetch info about the last n epochs

--gasCurrency=0x1234567890123456789012345678901234567890
Use a specific gas currency for transaction fees (defaults to CELO if no gas currency is supplied). It must be a whitelisted token.

--globalHelp
View all available global flags

DESCRIPTION

View general network information such as the current block number

See code: src/commands/network/info.ts

celocli network:parameters {#celocli-networkparameters}

View parameters of the network, including but not limited to configuration for the various Celo core smart contracts.

USAGE

```
$ celocli network:parameters [--gasCurrency <value>] [--globalHelp] [--raw]
```

FLAGS

| | |
|---|---|
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a specific gas currency |
| <code>--raw</code> | Display raw numerical configuration |
| <code>--globalHelp</code> | View all available global flags |
| <code>--gasCurrency=0x123456789012345678901234567890</code> | Use a specific gas currency for transaction fees (defaults to CELO if no gas currency is supplied). It must be a whitelisted token. |

DESCRIPTION

View parameters of the network, including but not limited to configuration for the various Celo core smart contracts.

See code: `src/commands/network/parameters.ts`

```
celocli network:whitelist {#celocli-networkwhitelist}
```

List the whitelisted fee currencies

USAGE

```
$ celocli network:whitelist [--gasCurrency <value>] [--globalHelp]
```

FLAGS

| | |
|---|---|
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a specific gas currency for transaction fees (defaults to CELO if no gas currency is supplied). It must be a whitelisted token. |
| <code>--globalHelp</code> | View all available global flags |

DESCRIPTION

List the whitelisted fee currencies

EXAMPLES

whitelist

See code: src/commands/network/whitelist.ts

node.md:

celocli node
=====

Manage your Celo node

celocli node:accounts
celocli node:synced

celocli node:accounts {#celocli-nodeaccounts}

List the addresses that this node has the private keys for.

USAGE

\$ celocli node:accounts [--gasCurrency <value>] [--globalHelp]

FLAGS

| | |
|--|-----------------------------|
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a specific gas currency |
| | for transaction fees |
| | (defaults to CELO if no gas |
| | supplied). It must be a |
| | whitelisted token. |
| --globalHelp | View all available global |
| | flags |

DESCRIPTION

List the addresses that this node has the private keys for.

See code: src/commands/node/accounts.ts

celocli node:synced {#celocli-nodesynced}

Check if the node is synced

USAGE

```
$ celocli node:syncd [--gasCurrency <value>] [--globalHelp] [--verbose]
```

FLAGS

```
--gasCurrency=0x1234567890123456789012345678901234567890 Use a
specific gas currency                                         for
transaction fees                                             (defaults to
CELO if no gas                                               currency is
supplied). It                                                must be a
whitelisted token.                                          View all
--globalHelp                                                 flags
available global                                             output the
--verbose                                                    syncing
full status if
```

DESCRIPTION

Check if the node is synced

See code: src/commands/node/synced.ts

oracle.md:

```
celocli oracle
=====
```

List oracle addresses for a given token

```
celocli oracle:list ARG1
celocli oracle:remove-expired-reports ARG1
celocli oracle:report ARG1
celocli oracle:reports ARG1
```

```
celocli oracle:list ARG1 {#celocli-oraclelist-arg1}
```

List oracle addresses for a given token

USAGE

```
$ celocli oracle:list ARG1 [--gasCurrency <value>] [--globalHelp]
```

ARGUMENTS

ARG1 [default: StableToken] Token to list the oracles for

FLAGS

`--gasCurrency=0x1234567890123456789012345678901234567890` Use a specific gas currency for transaction fees (defaults to CELO if no gas currency is supplied). It must be a whitelisted token. View all available global flags

DESCRIPTION

List oracle addresses for a given token

EXAMPLES

```
list StableToken

list

list StableTokenEUR
```

See code: `src/commands/oracle/list.ts`

```
celocli oracle:remove-expired-reports ARG1 {#celocli-oracleremove-expired-reports-arg1}
```

Remove expired oracle reports for a specified token

USAGE

```
$ celocli oracle:remove-expired-reports ARG1 --from <value> [--gasCurrency <value>]
  [--globalHelp]
```

ARGUMENTS

`ARG1` [default: `StableToken`] Token to remove expired reports for

FLAGS

`--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d` (required)
Address of the account removing oracle reports
`--gasCurrency=0x1234567890123456789012345678901234567890` Use a specific gas currency for transaction fees (defaults to CELO if no gas

supplied). It
whitelisted token.
--globalHelp
available global

currency is
must be a
View all
flags

DESCRIPTION

Remove expired oracle reports for a specified token

EXAMPLES

remove-expired-reports StableToken --from
0x8c349AAc7065a35B7166f2659d6C35D75A3893C1

remove-expired-reports --from
0x8c349AAc7065a35B7166f2659d6C35D75A3893C1

remove-expired-reports StableTokenEUR --from
0x8c349AAc7065a35B7166f2659d6C35D75A3893C1

See code: src/commands/oracle/remove-expired-reports.ts

celocli oracle:report ARG1 {#celocli-oraclereport-arg1}

Report the price of CELO in a specified token

USAGE

\$ celocli oracle:report ARG1 --from <value> --value <value> [--
gasCurrency
<value>] [--globalHelp]

ARGUMENTS

ARG1 [default: StableToken] Token to report on

FLAGS

--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d (required)
Address of the

account oracle

--gasCurrency=0x1234567890123456789012345678901234567890 Use a
specific gas currency

transaction fees for

CELO if no gas (defaults to

supplied). It currency is

whitelisted token. must be a

--globalHelp View all
available global

`--value=<value>` flags
Amount of the (required)
token equal to 1 specified
CELO

DESCRIPTION

Report the price of CELO in a specified token

EXAMPLES

```
report StableToken --value 1.02 --from  
0x8c349AAc7065a35B7166f2659d6C35D75A3893C1
```

```
report --value 0.99 --from 0x8c349AAc7065a35B7166f2659d6C35D75A3893C1
```

```
report StableTokenEUR --value 1.02 --from  
0x8c349AAc7065a35B7166f2659d6C35D75A3893C1
```

See code: `src/commands/oracle/report.ts`

```
celocli oracle:reports ARG1 {#celocli-oraclereports-arg1}
```

List oracle reports for a given token

USAGE

```
$ celocli oracle:reports ARG1 [--gasCurrency <value>] [--globalHelp] [-  
-columns  
  <value> | -x] [--filter <value>] [--no-header | [--csv | --no-  
truncate]] [--output  
  csv|json|yaml | | ] [--sort <value>]
```

ARGUMENTS

ARG1 [default: StableToken] Token to list the reports for

FLAGS

```
-x, --extended  
  show extra columns  
  
--columns=<value>  
  only show provided columns (comma-separated)  
  
--csv  
  output is csv format [alias: --output=csv]  
  
--filter=<value>  
  filter property by partial string matching, ex: name=foo  
  
--gasCurrency=0x1234567890123456789012345678901234567890  
  Use a specific gas currency for transaction fees (defaults to CELO  
if no gas  
  currency is supplied). It must be a whitelisted token.
```

```
--globalHelp
    View all available global flags

--no-header
    hide table header from output

--no-truncate
    do not truncate output to fit screen

--output=<option>
    output in a more machine friendly format
    <options: csv|json|yaml>

--sort=<value>
    property to sort by (prepend '-' for descending)
```

DESCRIPTION

List oracle reports for a given token

EXAMPLES

```
reports StableToken

reports

reports StableTokenEUR
```

See code: src/commands/oracle/reports.ts

plugins.md:

```
celocli plugins
=====
```

List installed plugins.

```
celocli plugins
celocli plugins:install PLUGIN...
celocli plugins:inspect PLUGIN...
celocli plugins:install PLUGIN...
celocli plugins:link PLUGIN
celocli plugins:uninstall PLUGIN...
celocli plugins:reset
celocli plugins:uninstall PLUGIN...
celocli plugins:uninstall PLUGIN...
celocli plugins:update
```

```
celocli plugins {#celocli-plugins}
```

List installed plugins.

USAGE

```
$ celocli plugins [--json] [--core]
```

FLAGS

```
--core  Show core plugins.
```

GLOBAL FLAGS

```
--json  Format output as json.
```

DESCRIPTION

List installed plugins.

EXAMPLES

```
$ celocli plugins
```

See code: @oclif/plugin-plugins

```
celocli plugins:install PLUGIN... {#celocli-pluginsinstall-plugin}
```

Installs a plugin into the CLI.

USAGE

```
$ celocli plugins:add plugins:install PLUGIN...
```

ARGUMENTS

```
PLUGIN  Plugin to install.
```

FLAGS

```
-f, --force  Run yarn install with force flag.  
-h, --help   Show CLI help.  
-s, --silent Silences yarn output.  
-v, --verbose Show verbose yarn output.
```

GLOBAL FLAGS

```
--json  Format output as json.
```

DESCRIPTION

Installs a plugin into the CLI.
Can be installed from npm or a git url.

Installation of a user-installed plugin will override a core plugin.

e.g. If you have a core plugin that has a 'hello' command, installing a user-installed

plugin with a 'hello' command will override the core plugin implementation. This is

useful if a user needs to update core plugin functionality in the CLI without the need

to patch and update the whole CLI.

ALIASES

```
$ celocli plugins:add
```

EXAMPLES

```
$ celocli plugins:add myplugin
```

```
$ celocli plugins:add https://github.com/someuser/someplugin
```

```
$ celocli plugins:add someuser/someplugin
```

```
celocli plugins:inspect PLUGIN... {#celocli-pluginsinspect-plugin}
```

Displays installation properties of a plugin.

USAGE

```
$ celocli plugins:inspect PLUGIN...
```

ARGUMENTS

```
PLUGIN [default: .] Plugin to inspect.
```

FLAGS

```
-h, --help      Show CLI help.
```

```
-v, --verbose
```

GLOBAL FLAGS

```
--json  Format output as json.
```

DESCRIPTION

Displays installation properties of a plugin.

EXAMPLES

```
$ celocli plugins:inspect myplugin
```

See code: @oclif/plugin-plugins

```
celocli plugins:install PLUGIN... {#celocli-pluginsinstall-plugin-1}
```

Installs a plugin into the CLI.

USAGE

```
$ celocli plugins:install PLUGIN...
```

ARGUMENTS

```
PLUGIN  Plugin to install.
```

FLAGS

```
-f, --force  Run yarn install with force flag.
```

```
-h, --help  Show CLI help.
```

```
-s, --silent  Silences yarn output.
```

```
-v, --verbose  Show verbose yarn output.
```


GLOBAL FLAGS

`--json` Format output as json.

DESCRIPTION

Installs a plugin into the CLI.
Can be installed from npm or a git url.

Installation of a user-installed plugin will override a core plugin.

e.g. If you have a core plugin that has a 'hello' command, installing a user-installed

plugin with a 'hello' command will override the core plugin implementation. This is

useful if a user needs to update core plugin functionality in the CLI without the need

to patch and update the whole CLI.

ALIASES

`$ celocli plugins:add`

EXAMPLES

`$ celocli plugins:install myplugin`

`$ celocli plugins:install https://github.com/someuser/someplugin`

`$ celocli plugins:install someuser/someplugin`

See code: @oclif/plugin-plugins

`celocli plugins:link PLUGIN {#celocli-pluginslink-plugin}`

Links a plugin into the CLI for development.

USAGE

`$ celocli plugins:link PLUGIN`

ARGUMENTS

`PATH` [default: .] path to plugin

FLAGS

`-h, --help` Show CLI help.

`-v, --verbose`

`--[no-]install` Install dependencies after linking the plugin.

DESCRIPTION

Links a plugin into the CLI for development.

Installation of a linked plugin will override a user-installed or core plugin.

e.g. If you have a user-installed or core plugin that has a 'hello' command,

installing a linked plugin with a 'hello' command will override the user-installed or core plugin implementation. This is useful for development work.

EXAMPLES

```
$ celocli plugins:link myplugin
```

See code: @oclif/plugin-plugins

```
celocli plugins:uninstall PLUGIN... {#celocli-pluginsuninstall-plugin}
```

Removes a plugin from the CLI.

USAGE

```
$ celocli plugins:remove plugins:uninstall PLUGIN...
```

ARGUMENTS

PLUGIN plugin to uninstall

FLAGS

```
-h, --help      Show CLI help.
-v, --verbose
```

DESCRIPTION

Removes a plugin from the CLI.

ALIASES

```
$ celocli plugins:unlink
$ celocli plugins:remove
```

EXAMPLES

```
$ celocli plugins:remove myplugin
```

```
celocli plugins:reset {#celocli-pluginsreset}
```

Remove all user-installed and linked plugins.

USAGE

```
$ celocli plugins:reset
```

See code: @oclif/plugin-plugins

```
celocli plugins:uninstall PLUGIN... {#celocli-pluginsuninstall-plugin-1}
```

Removes a plugin from the CLI.

USAGE

```
$ celocli plugins:uninstall PLUGIN...
```

ARGUMENTS

```
PLUGIN  plugin to uninstall
```

FLAGS

```
-h, --help      Show CLI help.  
-v, --verbose
```

DESCRIPTION

Removes a plugin from the CLI.

ALIASES

```
$ celocli plugins:unlink  
$ celocli plugins:remove
```

EXAMPLES

```
$ celocli plugins:uninstall myplugin
```

See code: @oclif/plugin-plugins

```
celocli plugins:uninstall PLUGIN... {#celocli-pluginsuninstall-plugin-2}
```

Removes a plugin from the CLI.

USAGE

```
$ celocli plugins:unlink plugins:uninstall PLUGIN...
```

ARGUMENTS

```
PLUGIN  plugin to uninstall
```

FLAGS

```
-h, --help      Show CLI help.  
-v, --verbose
```

DESCRIPTION

Removes a plugin from the CLI.

ALIASES

```
$ celocli plugins:unlink  
$ celocli plugins:remove
```

EXAMPLES

```
$ celocli plugins:unlink myplugin
```

```
celocli plugins:update {#celocli-pluginsupdate}
```

Update installed plugins.

USAGE

```
$ celocli plugins:update [-h] [-v]
```

FLAGS

```
-h, --help      Show CLI help.
-v, --verbose
```

DESCRIPTION

Update installed plugins.

See code: @oclif/plugin-plugins

releasecelo.md:

```
celocli releasecelo
=====
```

View and manage ReleaseGold contracts

```
celocli releasecelo:authorize
celocli releasecelo:create-account
celocli releasecelo:locked-gold
celocli releasecelo:refund-and-finalize
celocli releasecelo:revoke
celocli releasecelo:revoke-votes
celocli releasecelo:set-account
celocli releasecelo:set-account-wallet-address
celocli releasecelo:set-beneficiary
celocli releasecelo:set-can-expire
celocli releasecelo:set-liquidity-provision
celocli releasecelo:set-max-distribution
celocli releasecelo:show
celocli releasecelo:transfer-dollars
celocli releasecelo:withdraw
```

```
celocli releasecelo:authorize {#celocli-releaseceloauthorize}
```

Authorize an alternative key to be used for a given action (Vote, Validate, Attest) on behalf of the ReleaseGold instance contract.

USAGE

```
$ celocli releasecelo:authorize --contract <value> --role
vote|validator|attestation
    --signer <value> --signature <value> [--gasCurrency <value>] [--
globalHelp]
    [--blsKey <value> --blsPop <value>]
```

FLAGS

```
--blsKey=0x                                The BLS
public key that the                         validator is
using for
```

| | |
|---|---------------|
| should pass proof | consensus, |
| possession. 96 bytes. | of |
| --blsPop=0x | The BLS |
| public key | proof-of- |
| possession, which | consists of a |
| signature on | the account |
| address. 48 | bytes. |
| --contract=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | (required) |
| Address of the | ReleaseGold |
| Contract | |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| --globalHelp | flags |
| available global | (required) |
| --role=<option> | <options: |
| vote validator attestation> | |
| --signature=0x | (required) |
| Signature (a.k.a. | proof-of- |
| possession) of the | signer key |
| --signer=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | (required) |
| The signer key | that is to be |
| used for | voting |
| through the | ReleaseGold |
| instance | |
| DESCRIPTION | |
| Authorize an alternative key to be used for a given action (Vote, | |
| Validate, Attest) on | |
| behalf of the ReleaseGold instance contract. | |
| EXAMPLES | |

```
authorize --contract 0x5409ED021D9299bf6814279A6A1411A7e866A631 --role
vote --signer 0x6ecbe1db9ef729cbe972c83fb886247691fb6beb --signature
0x1b9fca4bbb5bfb1dbe69ef1cddb9b4202dcb6b134c5170611e1e36ecfa468d7b46c853
28d504934fce6c2a1571603a50ae224d2b32685e84d4d1aleebad8452eb
```

```
authorize --contract 0x5409ED021D9299bf6814279A6A1411A7e866A631 --role
validator --signer 0x6ecbe1db9ef729cbe972c83fb886247691fb6beb --signature
0x1b9fca4bbb5bfb1dbe69ef1cddb9b4202dcb6b134c5170611e1e36ecfa468d7b46c853
28d504934fce6c2a1571603a50ae224d2b32685e84d4d1aleebad8452eb --blsKey
0x4fa3f67fc913878b068d1falcdcdc54913d3bf988dbe5a36a20fa888f20d4894c408a67
73f3d7bde11154f2a3076b700d345a42fd25a0e5e83f4db5586ac7979ac2053cd95d8f2ef
d3e959571ceccaa743e02cf4be3f5d7aaddb0b06fc9aff00 --blsPop
0xcdb77255037eb68897cd487fdd85388cbda448f617f874449d4b11588b0b7ad8ddc20d9
bb450b513bb35664ea3923900
```

```
authorize --contract 0x5409ED021D9299bf6814279A6A1411A7e866A631 --role
attestation --signer 0x6ecbe1db9ef729cbe972c83fb886247691fb6beb --
signature
0x1b9fca4bbb5bfb1dbe69ef1cddb9b4202dcb6b134c5170611e1e36ecfa468d7b46c853
28d504934fce6c2a1571603a50ae224d2b32685e84d4d1aleebad8452eb
```

See code: `src/commands/releasecelo/authorize.ts`

```
celocli releasecelo:create-account {#celocli-releasecelo:create-account}
```

Creates a new account for the ReleaseGold instance

USAGE

```
$ celocli releasecelo:create-account --contract <value> [--gasCurrency
<value>]
[--globalHelp]
```

FLAGS

| | |
|---|--------------|
| <code>--contract=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Address of the | ReleaseGold |
| Contract | |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| <code>--globalHelp</code> | flags |
| available global | |

DESCRIPTION

Creates a new account for the ReleaseGold instance

EXAMPLES

```
create-account --contract 0x5409ED021D9299bf6814279A6A1411A7e866A631
```

See code: `src/commands/releasecelo/create-account.ts`

```
celocli releasecelo:locked-gold {#celocli-releasecelolocked-gold}
```

Perform actions [lock, unlock, withdraw] on CELO that has been locked via the provided ReleaseGold contract.

USAGE

```
$ celocli releasecelo:locked-gold --contract <value> -a
lock|unlock|withdraw --value
<value> [--gasCurrency <value>] [--globalHelp] [--yes]
```

FLAGS

[illegible]

DESCRIPTION

Perform actions [lock, unlock, withdraw] on CELO that has been locked via the provided ReleaseGold contract.

EXAMPLES

```
locked-gold --contract 0xCcc8a47BE435F1590809337BB14081b256Ae26A8 --  
action lock --value 1000000000000000000000
```

```
locked-gold --contract 0xCcc8a47BE435F1590809337BB14081b256Ae26A8 --  
action unlock --value 100000000000000000000
```

```
locked-gold --contract 0xCcc8a47BE435F1590809337BB14081b256Ae26A8 --
action withdraw --value 1000000000000000000000000
```

See code: `src/commands/releasecelo/locked-gold.ts`

```
celocli releasecelo:refund-and-finalize {#celocli-releasecelorefund-and-
finalize}
```

Refund the given contract's balance to the appropriate parties and destroy the contact. Can only be called by the release owner of revocable ReleaseGold instances.

USAGE

```
$ celocli releasecelo:refund-and-finalize --contract <value> [--
gasCurrency <value>]
[--globalHelp]
```

FLAGS

| | |
|---|--------------|
| <code>--contract=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Address of the | ReleaseGold |
| Contract | |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| <code>--globalHelp</code> | flags |
| available global | |

DESCRIPTION

Refund the given contract's balance to the appropriate parties and destroy the contact.

Can only be called by the release owner of revocable ReleaseGold instances.

EXAMPLES

```
refund-and-finalize --contract
0x5409ED021D9299bf6814279A6A1411A7e866A631
```

See code: `src/commands/releasecelo/refund-and-finalize.ts`

```
celocli releasecelo:revoke {#celocli-releasecelorevoke}
```


Revoke the given contract instance. Once revoked, any Locked Gold can be unlocked by the release owner. The beneficiary will then be able to withdraw any released Gold that had yet to be withdrawn, and the remainder can be transferred by the release owner to the refund address. Note that not all ReleaseGold instances are revokable.

USAGE

```
$ celocli releasecelo:revoke --contract <value> [--gasCurrency <value>]
[--globalHelp]
    [--yesreally]
```

FLAGS

| | |
|---|---------------|
| <code>--contract=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Address of the | ReleaseGold |
| Contract | |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| <code>--globalHelp</code> | flags |
| available global | Override |
| <code>--yesreally</code> | liquidity (be |
| prompt to set | |
| careful!) | |

DESCRIPTION

Revoke the given contract instance. Once revoked, any Locked Gold can be unlocked by the release owner. The beneficiary will then be able to withdraw any released Gold that had yet to be withdrawn, and the remainder can be transferred by the release owner to the refund address. Note that not all ReleaseGold instances are revokable.

EXAMPLES

```
revoke --contract 0x5409ED021D9299bf6814279A6A1411A7e866A631
```

See code: `src/commands/releasecelo/revoke.ts`

```
celocli releasecelo:revoke-votes {#celocli-releasecelorevoke-votes}
```

Revokes votes for the given contract's account from the given group's account

USAGE

```
$ celocli releasecelo:revoke-votes --contract <value> [--gasCurrency <value>] [--globalHelp]
    [--group <value> | --allGroups] [--votes <value> | --allVotes | ]
```

FLAGS

| | |
|---|---------------|
| <code>--allGroups</code> | Revoke all |
| votes from all | |
| | groups |
| <code>--allVotes</code> | Revoke all |
| votes | |
| <code>--contract=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Address of the | |
| | ReleaseGold |
| Contract | |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | |
| | for |
| transaction fees | |
| | (defaults to |
| CELO if no gas | |
| | currency is |
| supplied). It | |
| | must be a |
| whitelisted token. | |
| <code>--globalHelp</code> | View all |
| available global | |
| | flags |
| <code>--group=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | Address of |
| the group to | |
| | revoke votes |
| from | |
| <code>--votes=<value></code> | The number of |
| votes to | |
| | revoke |

DESCRIPTION

Revokes votes for the given contract's account from the given group's account

EXAMPLES

```
revoke-votes --contract 0x47e172F6CfB6c7D01C1574fa3E2Be7CC73269D95 --
group 0x5409ED021D9299bf6814279A6A1411A7e866A631 --votes 100
```

```
revoke-votes --contract 0x47e172F6CfB6c7D01C1574fa3E2Be7CC73269D95 --
allVotes --allGroups
```

See code: `src/commands/releasecelo/revoke-votes.ts`

```
celocli releasecelo:set-account {#celocli-releaseceloset-account}
```

Set account properties of the ReleaseGold instance account such as name, data encryption key, and the metadata URL

USAGE

```
$ celocli releasecelo:set-account --contract <value> -p  
name|dataEncryptionKey|metaURL -v  
    <value> [--gasCurrency <value>] [--globalHelp]
```

FLAGS

```
-p, --property=<option>  
    (required) Property type to set  
    <options: name|dataEncryptionKey|metaURL>  
  
-v, --value=<value>  
    (required) Property value to set  
  
--contract=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d  
    (required) Address of the ReleaseGold Contract  
  
--gasCurrency=0x1234567890123456789012345678901234567890  
    Use a specific gas currency for transaction fees (defaults to CEL0  
if no gas  
    currency is supplied). It must be a whitelisted token.  
  
--globalHelp  
    View all available global flags
```

DESCRIPTION

Set account properties of the ReleaseGold instance account such as name, data encryption key, and the metadata URL

EXAMPLES

```
set-account --contract 0x5719118266779B58D0f9519383A4A27aA7b829E5 --  
property name --value mywallet
```

```
set-account --contract 0x5719118266779B58D0f9519383A4A27aA7b829E5 --  
property dataEncryptionKey --value  
0x041bb96e35f9f4b71ca8de561fff55a249ddf9d13ab582bdd09a09e75da68ae4cd0ab70  
38030f41b237498b4d76387ae878dc8d98fd6f6db2c15362d1a3bf11216
```

```
set-account --contract 0x5719118266779B58D0f9519383A4A27aA7b829E5 --  
property metaURL --value www.example.com
```

See code: `src/commands/releasecelo/set-account.ts`

```
celocli releasecelo:set-account-wallet-address {#celocli-releaseceloset-  
account-wallet-address}
```

Set the ReleaseGold contract account's wallet address

USAGE

```
$ celocli releasecelo:set-account-wallet-address --contract <value> --
walletAddress <value> [--gasCurrency
<value>] [--globalHelp] [--pop <value>]
```

FLAGS

```
--contract=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d
    (required) Address of the ReleaseGold Contract

--gasCurrency=0x1234567890123456789012345678901234567890
    Use a specific gas currency for transaction fees (defaults to CELO
if no gas
    currency is supplied). It must be a whitelisted token.

--globalHelp
    View all available global flags

--pop=<value>
    ECDSA PoP for signer over contract's account

--walletAddress=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d
    (required) Address of wallet to set for contract's account and
signer of PoP. 0x0 if
    owner wants payers to contact them directly.
```

DESCRIPTION

Set the ReleaseGold contract account's wallet address

EXAMPLES

```
set-account-wallet-address --contract
0x5409ED021D9299bf6814279A6A1411A7e866A631 --walletAddress
0xE36Ea790bc9d7AB70C55260C66D52b1eca985f84 --pop
0x1b3e611d05e46753c43444cdc55c2cc3d95c54da0eba2464a8cc8cb01bd57ae8bb3d82a
0e293ca97e5813e7fb9b624127f42ef0871d025d8a56fe2f8f08117e25b
```

See code: `src/commands/releasecelo/set-account-wallet-address.ts`

```
celocli releasecelo:set-beneficiary {#celocli-releasecelaset-beneficiary}
```

Set the beneficiary of the ReleaseGold contract. This command is gated via a multi-sig, so this is expected to be called twice: once by the contract's beneficiary and once by the contract's releaseOwner. Once both addresses call this command with the same parameters, the tx will execute.

USAGE

```
$ celocli releasecelo:set-beneficiary --contract <value> --from <value>
--beneficiary <value>
    [--gasCurrency <value>] [--globalHelp] [--yesreally]
```

FLAGS

`--beneficiary=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d` (required)
Address of the new beneficiary
`--contract=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d` (required)
Address of the ReleaseGold Contract
`--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d` (required)
Address to submit multisig transaction from (one of the owners)
`--gasCurrency=0x1234567890123456789012345678901234567890` Use a specific gas currency for transaction fees (defaults to CELO if no gas currency is supplied). It must be a whitelisted token.
`--globalHelp` View all available global flags
`--yesreally` Override prompt to set new beneficiary (be careful!)

DESCRIPTION

Set the beneficiary of the ReleaseGold contract. This command is gated via a multi-sig, so this is expected to be called twice: once by the contract's beneficiary and once by the contract's releaseOwner. Once both addresses call this command with the same parameters, the tx will execute.

EXAMPLES

```
set-beneficiary --contract 0x5409ED021D9299bf6814279A6A1411A7e866A631 -  
-from 0xE36Ea790bc9d7AB70C55260C66D52b1eca985f84 --beneficiary  
0x6Ecbe1DB9EF729CBe972C83Fb886247691Fb6beb
```

See code: `src/commands/releasecelo/set-beneficiary.ts`

```
celocli releasecelo:set-can-expire {#celocli-releaseceloset-can-expire}
```

Set the canExpire flag for the given ReleaseGold contract

USAGE

```
$ celocli releasecelo:set-can-expire --contract <value> --value
true|false|True|False
[--gasCurrency <value>] [--globalHelp] [--yesreally]
```

FLAGS

```
--contract=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d (required)
Address of the ReleaseGold
Contract
--gasCurrency=0x1234567890123456789012345678901234567890 Use a
specific gas currency for
transaction fees (defaults to
CELO if no gas currency is
supplied). It must be a
whitelisted token. View all
--globalHelp flags
available global (required)
--value=<option> <options:
canExpire value
true|false|True|False>
--yesreally Override
prompt to set expiration
flag (be careful!)
```

DESCRIPTION

Set the canExpire flag for the given ReleaseGold contract

EXAMPLES

```
set-can-expire --contract 0x5409ED021D9299bf6814279A6A1411A7e866A631 --
value true
```

See code: `src/commands/releasecelo/set-can-expire.ts`

```
celocli releasecelo:set-liquidity-provision {#celocli-releasecelaset-
liquidity-provision}
```

Set the liquidity provision to true, allowing the beneficiary to withdraw released gold.

USAGE

```
$ celocli releasecelo:set-liquidity-provision --contract <value> [--gasCurrency <value>] [--globalHelp] [--yesreally]
```

FLAGS

```
--contract=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d (required)
Address of the
ReleaseGold
Contract
--gasCurrency=0x1234567890123456789012345678901234567890 Use a
specific gas currency
for
transaction fees
(defaults to
CELO if no gas
currency is
supplied). It
must be a
whitelisted token.
--globalHelp View all
available global
flags
--yesreally Override
prompt to set
liquidity (be
careful!)
```

DESCRIPTION

Set the liquidity provision to true, allowing the beneficiary to withdraw released gold.

EXAMPLES

```
set-liquidity-provision --contract
0x5409ED021D9299bf6814279A6A1411A7e866A631
```

See code: `src/commands/releasecelo/set-liquidity-provision.ts`

```
celocli releasecelo:set-max-distribution {#celocli-releaseceloset-max-
distribution}
```

Set the maximum distribution of celo for the given contract

USAGE

```
$ celocli releasecelo:set-max-distribution --contract <value> --distributionRatio <value>
[--gasCurrency <value>] [--globalHelp] [--yesreally]
```

FLAGS

```
--contract=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d (required)
Address of the
```

| | |
|--|---------------|
| Contract | ReleaseGold |
| --distributionRatio=<value> | (required) |
| Amount in range | [0, 1000] (3 |
| significant | figures) |
| indicating % of | total balance |
| available for | distribution. |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| --globalHelp | flags |
| available global | Override |
| --yesreally | maximum |
| prompt to set new | careful!) |
| distribution (be | |

DESCRIPTION

Set the maximum distribution of celo for the given contract

EXAMPLES

```
set-max-distribution --contract
0x5409ED021D9299bf6814279A6A1411A7e866A631 --distributionRatio 1000
```

See code: `src/commands/releasecelo/set-max-distribution.ts`

```
celocli releasecelo:show {#celocli-releaseceloshow}
```

Show info on a ReleaseGold instance contract.

USAGE

```
$ celocli releasecelo:show --contract <value> [--gasCurrency <value>]
[--globalHelp]
```

FLAGS

| | |
|---|-------------|
| --contract=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | (required) |
| Address of the | ReleaseGold |
| Contract | |


```
--gasCurrency=0x1234567890123456789012345678901234567890 Use a
specific gas currency                                     for
                                                         transaction fees
                                                         (defaults to
CELO if no gas                                          currency is
supplied). It                                           must be a
whitelisted token.                                     whitelisted token.
    --globalHelp                                         View all
available global                                        flags
```

DESCRIPTION

Show info on a ReleaseGold instance contract.

EXAMPLES

```
show --contract 0x47e172F6CfB6c7D01C1574fa3E2Be7CC73269D95
```

See code: `src/commands/releasecelo/show.ts`

```
celocli releasecelo:transfer-dollars {#celocli-releasecelotransfer-  
dollars}
```

Transfer Celo Dollars from the given contract address. Dollars may be accrued to the ReleaseGold contract via validator epoch rewards.

USAGE

```
$ celocli releasecelo:transfer-dollars --contract <value> --to <value>
--value <value>
    [--gasCurrency <value>] [--globalHelp]
```

FLAGS

| | |
|---|--------------|
| <code>--contract=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Address of the | |
| | ReleaseGold |
| Contract | |
| <code>--gasCurrency=0x12345678901234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | |
| | for |
| transaction fees | |
| | (defaults to |
| CELO if no gas | |
| | currency is |
| supplied). It | |
| | must be a |
| whitelisted token. | |
| <code>--globalHelp</code> | View all |
| available global | |
| | flags |

| | |
|--|--------------|
| <code>--to=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Address of the | |
| Celo Dollars | recipient of |
| | transfer |
| <code>--value=1000000000000000000000</code> | (required) |
| Value (in Wei) of | |
| to transfer | Celo Dollars |

DESCRIPTION

Transfer Celo Dollars from the given contract address. Dollars may be accrued to the ReleaseGold contract via validator epoch rewards.

EXAMPLES

```
transfer-dollars --contract 0x5409ED021D9299bf6814279A6A1411A7e866A631
--to 0x6Ecbe1DB9EF729CBe972C83Fb886247691Fb6beb --value
1000000000000000000000
```

See code: `src/commands/releasecelo/transfer-dollars.ts`

```
celocli releasecelo:withdraw {#celocli-releasecelowithdraw}
```

Withdraws value released celo to the beneficiary address. Fails if value worth of celo has not been released yet.

USAGE

```
$ celocli releasecelo:withdraw --contract <value> --value <value> [--
gasCurrency
  <value>] [--globalHelp]
```

FLAGS

| | |
|---|--------------|
| <code>--contract=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Address of the | |
| Contract | ReleaseGold |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | |
| <code>--globalHelp</code> | View all |
| available global | flags |
| <code>--value=1000000000000000000000</code> | (required) |
| Amount of | |

released celo

withdraw

Withdraws value released celo to the beneficiary address. Fails if value worth of celo has not been released yet.

```
withdraw --contract 0x5409ED021D9299bf6814279A6A1411A7e866A631 --value  
100000000000000000000
```

```
# reserve.md:
```

Shows information about reserve

```
celocli reserve:status
celocli reserve:transfergold
```

```
celocli reserve:status {#celocli-reservestatus}
```

Shows information about reserve

```
$ celocli reserve:status [--gasCurrency <value>] [--globalHelp]
```

```
--gasCurrency=0x1234567890123456789012345678901234567890  Use a
specific gas currency                                           for
transaction fees                                              (defaults to
CELO if no gas                                              currency is
supplied). It                                              must be a
whitelisted token.                                           whitelisted token.
  --globalHelp                                              View all
available global                                              flags
```

Shows information about reserve

EXAMPLES

status

See code: src/commands/reserve/status.ts

celocli reserve:transfergold {#celocli-reservetransfergold}

Transfers reserve celo to other reserve address

USAGE

```
$ celocli reserve:transfergold --value <value> --to <value> --from  
<value>  
  [--gasCurrency <value>] [--globalHelp] [--useMultiSig]
```

FLAGS

| | |
|--|--------------|
| --from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | (required) |
| Spender's address | |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| --globalHelp | flags |
| available global | (required) |
| --to=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | True means |
| Receiving address | be sent |
| --useMultiSig | (required) |
| the request will | of CELO |
| through multisig. | |
| --value=<value> | |
| The unit amount | |

DESCRIPTION

Transfers reserve celo to other reserve address

EXAMPLES

```
transfergold --value 9000 --to  
0x91c987bf62D25945dB517BDAA840A6c661374402 --from  
0x5409ed021d9299bf6814279a6a1411a7e866a631  
  
transfergold --value 9000 --to  
0x91c987bf62D25945dB517BDAA840A6c661374402 --from  
0x5409ed021d9299bf6814279a6a1411a7e866a631 --useMultiSig
```

See code: src/commands/reserve/transfergold.ts

rewards.md:

```
celocli rewards
=====
```

Show rewards information about a voter, registered Validator, or Validator Group

```
celocli rewards:show
```

```
celocli rewards:show {#celocli-rewardsshow}
```

Show rewards information about a voter, registered Validator, or Validator Group

USAGE

```
$ celocli rewards:show [--gasCurrency <value>] [--globalHelp] [--estimate]
  [--voter <value>] [--validator <value>] [--group <value>] [--slashing]
  [--epochs <value>] [--columns <value> | -x] [--filter <value>] [--no-header |
  [--csv |
  --no-truncate]] [--output csv|json|yaml |  | ] [--sort <value>]
```

FLAGS

```
-x, --extended
  show extra columns

--columns=<value>
  only show provided columns (comma-separated)

--csv
  output is csv format [alias: --output=csv]

--epochs=<value>
  [default: 1] Show results for the last N epochs

--estimate
  Estimate voter rewards from current votes

--filter=<value>
  filter property by partial string matching, ex: name=foo

--gasCurrency=0x1234567890123456789012345678901234567890
  Use a specific gas currency for transaction fees (defaults to CELO
if no gas
  currency is supplied). It must be a whitelisted token.

--globalHelp
  View all available global flags
```

```

--group=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d
    Validator Group to show rewards for

--no-header
    hide table header from output

--no-truncate
    do not truncate output to fit screen

--output=<option>
    output in a more machine friendly format
    <options: csv|json|yaml>

--slashing
    Show rewards for slashing

--sort=<value>
    property to sort by (prepend '-' for descending)

--validator=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d
    Validator to show rewards for

--voter=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d
    Voter to show rewards for

```

DESCRIPTION

Show rewards information about a voter, registered Validator, or Validator Group

EXAMPLES

```
show --voter 0x5409ed021d9299bf6814279a6a1411a7e866a631
```

See code: `src/commands/rewards/show.ts`

transfer.md:

```
celocli transfer
=====
```

Transfer CEL0 and Celo Dollars

```
celocli transfer:celo
celocli transfer:dollars
celocli transfer:erc20
celocli transfer:euros
celocli transfer:gold
celocli transfer:reals
celocli transfer:stable
```

```
celocli transfer:celo {#celocli-transfercelo}
```

Transfer CELO to a specified address. (Note: this is the equivalent of the old transfer:gold)

USAGE

```
$ celocli transfer:celo --from <value> --to <value> --value <value>
[--gasCurrency <value>] [--globalHelp] [--comment <value>]
```

FLAGS

| | |
|---|--------------|
| <code>--comment=<value></code> | Transfer |
| comment | |
| <code>--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Address of the | |
| | sender |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | |
| | for |
| transaction fees | |
| | (defaults to |
| CELO if no gas | |
| | currency is |
| supplied). It | |
| | must be a |
| whitelisted token. | |
| <code>--globalHelp</code> | View all |
| available global | |
| | flags |
| <code>--to=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Address of the | |
| | receiver |
| <code>--value=<value></code> | (required) |
| Amount to | |
| | transfer (in |
| wei) | |

DESCRIPTION

Transfer CELO to a specified address. (Note: this is the equivalent of the old transfer:gold)

EXAMPLES

```
celo --from 0xa0Af2E71cECc248f4a7fD606F203467B500Dd53B --to
0x5409ed021d9299bf6814279a6a1411a7e866a631 --value 10000000000000000000
```

See code: `src/commands/transfer/celo.ts`

```
celocli transfer:dollars {#celocli-transferdollars}
```

Transfer Celo Dollars (cUSD) to a specified address.

USAGE

```
$ celocli transfer:dollars --from <value> --to <value> --value <value>
```

[--gasCurrency <value>] [--globalHelp] [--comment <value>]

FLAGS

| | |
|--|--------------|
| --comment=<value> | Transfer |
| comment | |
| --from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | (required) |
| Address of the | |
| | sender |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | |
| | for |
| transaction fees | |
| | (defaults to |
| CELO if no gas | |
| | currency is |
| supplied). It | |
| | must be a |
| whitelisted token. | |
| --globalHelp | View all |
| available global | |
| | flags |
| --to=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | (required) |
| Address of the | |
| | receiver |
| --value=<value> | (required) |
| Amount to | |
| | transfer (in |
| wei) | |

DESCRIPTION

Transfer Celo Dollars (cUSD) to a specified address.

EXAMPLES

dollars --from 0xa0Af2E71cECc248f4a7fD606F203467B500Dd53B --to
0x5409ed021d9299bf6814279a6a1411a7e866a631 --value 1000000000000000000

See code: src/commands/transfer/dollars.ts

celocli transfer:erc20 {#celocli-transfererc20}

Transfer ERC20 to a specified address

USAGE

\$ celocli transfer:erc20 --erc20Address <value> --from <value> --to
<value>
--value <value> [--gasCurrency <value>] [--globalHelp]

FLAGS

--erc20Address=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d
(required) Custom erc20 to check it's balance too

--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d

(required) Address of the sender

--gasCurrency=0x1234567890123456789012345678901234567890
Use a specific gas currency for transaction fees (defaults to CELO
if no gas
currency is supplied). It must be a whitelisted token.

--globalHelp
View all available global flags

--to=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d
(required) Address of the receiver

--value=<value>
(required) Amount to transfer (in wei)

DESCRIPTION

Transfer ERC20 to a specified address

EXAMPLES

```
erc20 --erc20Address 0x765DE816845861e75A25fCA122bb6898B8B1282a --from  
0xa0Af2E71cECc248f4a7fD606F203467B500Dd53B --to  
0x5409ed021d9299bf6814279a6a1411a7e866a631 --value 10000000000000000000
```

See code: src/commands/transfer/erc20.ts

```
celocli transfer:euros {#celocli-transfereuros}
```

Transfer Celo Euros (cEUR) to a specified address.

USAGE

```
$ celocli transfer:euros --from <value> --to <value> --value <value>  
[--gasCurrency <value>] [--globalHelp] [--comment <value>]
```

FLAGS

| | |
|--|--------------|
| --comment=<value> | Transfer |
| comment | |
| --from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | (required) |
| Address of the | |
| | sender |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | |
| | for |
| transaction fees | |
| | (defaults to |
| CELO if no gas | |
| | currency is |
| supplied). It | |
| | must be a |
| whitelisted token. | |
| --globalHelp | View all |
| available global | |

| | |
|--|--------------|
| <code>--to=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | flags |
| Address of the | (required) |
| <code>--value=<value></code> | receiver |
| Amount to | (required) |
| wei) | transfer (in |

DESCRIPTION

Transfer Celo Euros (cEUR) to a specified address.

EXAMPLES

```
euros --from 0xa0Af2E71cECc248f4a7fD606F203467B500Dd53B --to
0x5409ed021d9299bf6814279a6a1411a7e866a631 --value 1000000000000000000
```

See code: `src/commands/transfer/euros.ts`

```
celocli transfer:gold {#celocli-transfergold}
```

Transfer CELO to a specified address. DEPRECATION WARNING Use the "transfer:celo" command instead

USAGE

```
$ celocli transfer:gold --from <value> --to <value> --value <value>
[--gasCurrency <value>] [--globalHelp] [--comment <value>]
```

FLAGS

| | |
|---|--------------|
| <code>--comment=<value></code> | Transfer |
| comment | |
| <code>--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Address of the | |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | sender |
| specific gas currency | Use a |
| transaction fees | for |
| CELO if no gas | (defaults to |
| supplied). It | currency is |
| whitelisted token. | must be a |
| <code>--globalHelp</code> | View all |
| available global | |
| <code>--to=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | flags |
| Address of the | (required) |
| <code>--value=<value></code> | receiver |
| Amount to | (required) |

transfer (in
wei)

DESCRIPTION

Transfer CELO to a specified address. DEPRECATION WARNING Use the
"transfer:celo"
command instead

EXAMPLES

```
gold --from 0xa0Af2E71cECc248f4a7fD606F203467B500Dd53B --to  
0x5409ed021d9299bf6814279a6a1411a7e866a631 --value 1000000000000000000
```

See code: src/commands/transfer/gold.ts

```
celocli transfer:reals {#celocli-transferreals}
```

Transfer Celo Brazilian Real (cREAL) to a specified address.

USAGE

```
$ celocli transfer:reals --from <value> --to <value> --value <value>  
[--gasCurrency <value>] [--globalHelp] [--comment <value>]
```

FLAGS

| | |
|--|--------------|
| --comment=<value> | Transfer |
| comment | |
| --from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | (required) |
| Address of the | |
| | sender |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | |
| | for |
| transaction fees | |
| | (defaults to |
| CELO if no gas | |
| | currency is |
| supplied). It | |
| | must be a |
| whitelisted token. | |
| --globalHelp | View all |
| available global | |
| | flags |
| --to=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | (required) |
| Address of the | |
| | receiver |
| --value=<value> | (required) |
| Amount to | |
| | transfer (in |
| wei) | |

DESCRIPTION

Transfer Celo Brazilian Real (cREAL) to a specified address.

EXAMPLES

```
reals --from 0xa0Af2E71cECc248f4a7fD606F203467B500Dd53B --to
0x5409ed021d9299bf6814279a6a1411a7e866a631 --value 1000000000000000000
```

See code: src/commands/transfer/reals.ts

```
celocli transfer:stable {#celocli-transferstable}
```

Transfer a stable token to a specified address.

USAGE

```
$ celocli transfer:stable --from <value> --to <value> --value <value>
  [--gasCurrency <value>] [--globalHelp] [--comment <value>] [--
stableToken
  cUSD|cusd|cEUR|ceur|cREAL|creal]
```

FLAGS

| | |
|--|--------------|
| --comment=<value> | Transfer |
| comment | |
| --from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | (required) |
| Address of the | |
| | sender |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | |
| | for |
| transaction fees | |
| | (defaults to |
| CELO if no gas | |
| | currency is |
| supplied). It | |
| | must be a |
| whitelisted token. | |
| --globalHelp | View all |
| available global | |
| | flags |
| --stableToken=<option> | Name of the |
| stable to be | |
| | transferred |
| | <options: |
| cUSD cusd cEUR ceur | |
| r cREAL creal> | |
| --to=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | (required) |
| Address of the | |
| | receiver |
| --value=<value> | (required) |
| Amount to | |
| | transfer (in |
| wei) | |

DESCRIPTION

Transfer a stable token to a specified address.

EXAMPLES

```
stable --from 0xa0Af2E71cECc248f4a7fD606F203467B500Dd53B --to
0x5409ed021d9299bf6814279a6a1411a7e866a631 --value 1000000000000000000 --
stableToken cStableTokenSymbol
```

See code: src/commands/transfer/stable.ts

utils.md:

```
celocli utils
=====
```

List the whitelisted fee currencies

```
celocli utils:whitelist
```

```
celocli utils:whitelist {#celocli-utilswhitelist}
```

List the whitelisted fee currencies

USAGE

```
$ celocli utils:whitelist [--gasCurrency <value>] [--globalHelp]
```

FLAGS

| | |
|--|--------------|
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | |
| | for |
| transaction fees | |
| | (defaults to |
| CELO if no gas | |
| | currency is |
| supplied). It | |
| | must be a |
| whitelisted token. | |
| --globalHelp | View all |
| available global | |
| | flags |

DESCRIPTION

List the whitelisted fee currencies

EXAMPLES

```
whitelist
```

validator.md:

```
celocli validator
=====
```

View and manage Validators

```
celocli validator:affiliate ARG1
celocli validator:deaffiliate
celocli validator:deregister
celocli validator:downtime-slash
celocli validator:force-deaffiliate
celocli validator:list
celocli validator:register
celocli validator:requirements
celocli validator:set-bitmaps
celocli validator:show ARG1
celocli validator:signed-blocks
celocli validator:status
celocli validator:update-bls-public-key
```

```
celocli validator:affiliate ARG1 {#celocli-validatoraffiliate-arg1}
```

Affiliate a Validator with a Validator Group. This allows the Validator Group to add that Validator as a member. If the Validator is already a member of a Validator Group, affiliating with a different Group will remove the Validator from the first group's members.

USAGE

```
$ celocli validator:affiliate ARG1 --from <value> [--gasCurrency
<value>]
    [--globalHelp] [--yes]
```

ARGUMENTS

ARG1 ValidatorGroup's address

FLAGS

| | |
|--|---------------|
| --from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | (required) |
| Signer or | Validator's |
| address | |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| --globalHelp | flags |
| available global | Answer yes to |
| --yes | |
| prompt | |

DESCRIPTION

Affiliate a Validator with a Validator Group. This allows the Validator Group to add that Validator as a member. If the Validator is already a member of a Validator Group, affiliating with a different Group will remove the Validator from the first group's members.

EXAMPLES

```
affiliate --from 0x47e172f6cfb6c7d01c1574fa3e2be7cc73269d95
0x97f7333c51897469e8d98e7af8653aab468050a3
```

See code: `src/commands/validator/affiliate.ts`

```
celocli validator:deaffiliate {#celocli-validatordeaffiliate}
```

Deaffiliate a Validator from a Validator Group, and remove it from the Group if it is also a member.

USAGE

```
$ celocli validator:deaffiliate --from <value> [--gasCurrency <value>]
[--globalHelp]
```

FLAGS

| | |
|---|------------------|
| <code>--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Signer or | Validator's |
| address | |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| <code>--globalHelp</code> | available global |
| | flags |

DESCRIPTION

Deaffiliate a Validator from a Validator Group, and remove it from the Group if it is also a member.

EXAMPLES

```
deaffiliate --from 0x47e172f6cfb6c7d01c1574fa3e2be7cc73269d95
```

See code: src/commands/validator/deaffiliate.ts

```
celocli validator:deregister {#celocli-validator:deregister}
```

Deregister a Validator. Approximately 60 days after the validator is no longer part of any group, it will be possible to deregister the validator and start unlocking the CELO. If you wish to deregister your validator, you must first remove it from it's group, such as by deaffiliating it, then wait the required 60 days before running this command.

USAGE

```
$ celocli validator:deregister --from <value> [--gasCurrency <value>]
[--globalHelp]
```

FLAGS

| | |
|---|--------------|
| <code>--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| Signer or | Validator's |
| address | |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| <code>--globalHelp</code> | flags |
| available global | |

DESCRIPTION

Deregister a Validator. Approximately 60 days after the validator is no longer part of any group, it will be possible to deregister the validator and start unlocking the CELO. If you wish to deregister your validator, you must first remove it from it's group, such as by deaffiliating it, then wait the required 60 days before running this command.

EXAMPLES

```
deregister --from 0x47e172f6cfb6c7d01c1574fa3e2be7cc73269d95
```

See code: src/commands/validator/deregister.ts

```
celocli validator:downtime-slash {#celocli-validator:downtime-slash}
```

Downtime slash a validator

USAGE

```
$ celocli validator:downtime-slash --from <value> [--gasCurrency
<value>] [--globalHelp]
  [--validator <value> | --validators <value>] [--intervals <value> | -
-beforeBlock
  <value>]
```

FLAGS

```
--beforeBlock=<value>
  Slash for slashable downtime window before provided block

--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d
  (required) From address to perform the slash (reward recipient)

--gasCurrency=0x1234567890123456789012345678901234567890
  Use a specific gas currency for transaction fees (defaults to CELO
if no gas
  currency is supplied). It must be a whitelisted token.

--globalHelp
  View all available global flags

--intervals='[0:1], [1:2]'
  Array of intervals, ordered by min start to max end

--validator=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d
  Validator (signer or account) address

--validators='["0xb7ef0985bdb4f19460A29d9829aA1514B181C4CD",
"0x47e172f6cfb6c7d01c1574fa3e2be7cc73269d95"]'
  Validator (signer or account) address list
```

DESCRIPTION

Downtime slash a validator

EXAMPLES

```
downtime-slash --from 0x47e172f6cfb6c7d01c1574fa3e2be7cc73269d95
--validator 0xb7ef0985bdb4f19460A29d9829aA1514B181C4CD --intervals
"100:150), [150:200)"
```

```
downtime-slash --from 0x47e172f6cfb6c7d01c1574fa3e2be7cc73269d95
--validator 0xb7ef0985bdb4f19460A29d9829aA1514B181C4CD --
slashableDowntimeBeforeBlock 200
```

See code: [src/commands/validator/downtime-slash.ts

```
celocli validator:force-deaffiliate {#celocli-validatorforce-deaffiliate}
```

Force deaffiliate a Validator from a Validator Group, and remove it from the Group if it is also a member. Used by stake-off admins in order to remove validators from the next epoch's validator set if they are down

and consistently unresponsive, in order to preserve the health of the network. This feature will be removed once slashing for downtime is implemented.

USAGE

```
$ celocli validator:force-deaffiliate --from <value> --validator
<value> [--gasCurrency
  <value>] [--globalHelp]
```

FLAGS

```
--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d      (required)
Initiator
--gasCurrency=0x1234567890123456789012345678901234567890 Use a
specific gas currency                                     for
transaction fees                                         (defaults to
CELO if no gas                                           currency is
supplied). It                                           must be a
whitelisted token.                                       View all
--globalHelp                                             flags
available global                                         (required)
--validator=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d
Validator's                                              address
```

DESCRIPTION

Force deaffiliate a Validator from a Validator Group, and remove it from the Group if it is also a member. Used by stake-off admins in order to remove validators from the next epoch's validator set if they are down and consistently unresponsive, in order to preserve the health of the network. This feature will be removed once slashing for downtime is implemented.

EXAMPLES

```
force-deaffiliate --from 0x47e172f6cfb6c7d01c1574fa3e2be7cc73269d95 --
validator 0xb7ef0985bdb4f19460A29d9829aA1514B181C4CD
```

See code: `src/commands/validator/force-deaffiliate.ts`

```
celocli validator:list {#celocli-validatorlist}
```

List registered Validators, their name (if provided), affiliation, uptime score, and public keys used for validating.

USAGE

```
$ celocli validator:list [--gasCurrency <value>] [--globalHelp] [--  
columns <value>  
| -x] [--filter <value>] [--no-header | [--csv | --no-truncate]] [--  
output  
csv|json|yaml | | ] [--sort <value>]
```

FLAGS

```
-x, --extended  
    show extra columns  
  
--columns=<value>  
    only show provided columns (comma-separated)  
  
--csv  
    output is csv format [alias: --output=csv]  
  
--filter=<value>  
    filter property by partial string matching, ex: name=foo  
  
--gasCurrency=0x1234567890123456789012345678901234567890  
    Use a specific gas currency for transaction fees (defaults to CELO  
if no gas  
    currency is supplied). It must be a whitelisted token.  
  
--globalHelp  
    View all available global flags  
  
--no-header  
    hide table header from output  
  
--no-truncate  
    do not truncate output to fit screen  
  
--output=<option>  
    output in a more machine friendly format  
    <options: csv|json|yaml>  
  
--sort=<value>  
    property to sort by (prepend '-' for descending)
```

DESCRIPTION

List registered Validators, their name (if provided), affiliation, uptime score, and public keys used for validating.

EXAMPLES

```
list
```

See code: src/commands/validator/list.ts

```
celocli validator:register {#celocli-validatorregister}
```

Register a new Validator

USAGE

```
$ celocli validator:register --from <value> --ecdsaKey <value> --blsKey  
<value>  
    --blsSignature <value> [--gasCurrency <value>] [--globalHelp] [--yes]
```

FLAGS

| | |
|--|---------------|
| --blsKey=0x | (required) |
| BLS Public Key | |
| --blsSignature=0x | (required) |
| BLS | Proof-of- |
| Possession | |
| --ecdsaKey=0x | (required) |
| ECDSA Public Key | |
| --from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | (required) |
| Address for the | Validator |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| --globalHelp | flags |
| available global | Answer yes to |
| --yes | |
| prompt | |

DESCRIPTION

Register a new Validator

EXAMPLES

```
register --from 0x47e172F6CfB6c7D01C1574fa3E2Be7CC73269D95 --ecdsaKey  
0x049b7291ab8813a095d6b7913a7930ede5ea17466abd5e1a26c6c44f6df9a400a6f4740  
80098b2c752c6c4871978ca977b90dcd3aed92bc9d564137c8dfa14ee72 --blsKey  
0x4fa3f67fc913878b068d1fa1cdddc54913d3bf988dbe5a36a20fa888f20d4894c408a67  
73f3d7bde11154f2a3076b700d345a42fd25a0e5e83f4db5586ac7979ac2053cd95d8f2ef  
d3e959571ceccaa743e02cf4be3f5d7aaddb0b06fc9aff00 --blsSignature  
0xcdb77255037eb68897cd487fdd85388cbda448f617f874449d4b11588b0b7ad8ddc20d9  
bb450b513bb35664ea3923900
```

See code: `src/commands/validator/register.ts`

```
celocli validator:requirements {#celocli-validatorrequirements}
```

List the Locked Gold requirements for registering a Validator. This consists of a value, which is the amount of CELO that needs to be locked in order to register, and a duration, which is the amount of time that CELO must stay locked following the deregistration of the Validator.

USAGE

```
$ celocli validator:requirements [--gasCurrency <value>] [--globalHelp]
```

FLAGS

| | |
|---|--|
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a specific gas currency |
| | for transaction fees |
| | (defaults to CELO if no gas supplied). It must be a whitelisted token. |
| <code>--globalHelp</code> | View all available global flags |

DESCRIPTION

List the Locked Gold requirements for registering a Validator. This consists of a value, which is the amount of CELO that needs to be locked in order to register, and a duration, which is the amount of time that CELO must stay locked following the deregistration of the Validator.

EXAMPLES

```
requirements
```

See code: `src/commands/validator/requirements.ts`

```
celocli validator:set-bitmaps {#celocli-validatorset-bitmaps}
```

Set validator signature bitmaps for provided intervals

USAGE

```
$ celocli validator:set-bitmaps --from <value> [--gasCurrency <value>]
[--globalHelp]
  [--slashableDowntimeBeforeBlock <value> | --intervals <value> |
  --slashableDowntimeBeforeLatest]
```

FLAGS

| | |
|--|------------|
| <code>--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | (required) |
| From address to | |

| | |
|--|---------------|
| bitmap transactions | sign set |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| --globalHelp | flags |
| available global | Array of |
| --intervals='[0:1], [1:2]' | by min start |
| intervals, ordered | |
| to max end | |
| --slashableDowntimeBeforeBlock=<value> | Set all |
| bitmaps for | slashable |
| downtime window | before |
| provided block | |
| --slashableDowntimeBeforeLatest | Set all |
| bitmaps for | slashable |
| downtime window | before latest |
| block | |

DESCRIPTION

Set validator signature bitmaps for provided intervals

EXAMPLES

```
set-bitmaps --from 0x47e172f6cfb6c7d01c1574fa3e2be7cc73269d95 --
slashableDowntimeBeforeBlock 10000
```

```
set-bitmaps --from 0x47e172f6cfb6c7d01c1574fa3e2be7cc73269d95 --
intervals "[0:100], (100:200]"
```

See code: `src/commands/validator/set-bitmaps.ts`

```
celocli validator:show ARG1 {#celocli-validatorshow-arg1}
```

Show information about a registered Validator.

USAGE

```
$ celocli validator:show ARG1 [--gasCurrency <value>] [--globalHelp]
```

ARGUMENTS

ARG1 Validator's address

FLAGS

`--gasCurrency=0x1234567890123456789012345678901234567890` Use a specific gas currency for transaction fees (defaults to CELO if no gas is supplied). It must be a whitelisted token.
`--globalHelp` View all available global flags

DESCRIPTION

Show information about a registered Validator.

EXAMPLES

`show 0x97f7333c51897469E8D98E7af8653aAb468050a3`

See code: `src/commands/validator/show.ts`

`celocli validator:signed-blocks {#celocli-validatorsigned-blocks}`

Display a graph of blocks and whether the given signer's signature is included in each. A green '.' indicates the signature is present in that block, a red 'X' indicates the signature is not present. A yellow '' indicates the signer is not elected for that block.

USAGE

`$ celocli validator:signed-blocks [--gasCurrency <value>] [--globalHelp] [--signer <value> | --signers <value>] [--wasDownWhileElected] [--at-block <value> |] [--slashableDowntimeLookback | [--lookback <value> |]] [--width <value>]`

FLAGS

`--at-block=<value>`
latest block to examine for signer activity

`--gasCurrency=0x1234567890123456789012345678901234567890`
Use a specific gas currency for transaction fees (defaults to CELO if no gas is supplied). It must be a whitelisted token.

`--globalHelp`
View all available global flags

`--lookback=<value>`

```

    [default: 120] how many blocks to look back for signer activity

--signer=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d
    address of the signer to check for signatures

--signers='["0xb7ef0985bdb4f19460A29d9829aA1514B181C4CD",
"0x47e172f6cfb6c7d01c1574fa3e2be7cc73269d95"]'
    list of signer addresses to check for signatures

--slashableDowntimeLookback
    lookback of slashableDowntime

--wasDownWhileElected
    indicate whether a validator was down while elected for range

--width=<value>
    [default: 40] line width for printing marks

```

DESCRIPTION

Display a graph of blocks and whether the given signer's signature is included in each. A green '.' indicates the signature is present in that block, a red 'X' indicates the signature is not present. A yellow '-' indicates the signer is not elected for that block.

EXAMPLES

```

signed-blocks --signer 0x5409ED021D9299bf6814279A6A1411A7e866A631

signed-blocks --signer 0x5409ED021D9299bf6814279A6A1411A7e866A631 --
follow

signed-blocks --at-block 100000 --signer
0x5409ED021D9299bf6814279A6A1411A7e866A631

signed-blocks --lookback 500 --signer
0x5409ED021D9299bf6814279A6A1411A7e866A631

signed-blocks --lookback 50 --width 10 --signer
0x5409ED021D9299bf6814279A6A1411A7e866A631

```

See code: `src/commands/validator/signed-blocks.ts`

```
celocli validator:status {#celocli-validatorstatus}
```

Shows the consensus status of a validator. This command will show whether a validator is currently elected, would be elected if an election were to be run right now, and the percentage of blocks signed and number of blocks successfully proposed within a given window.

USAGE


```
$ celocli validator:status [--gasCurrency <value>] [--globalHelp] [--
validator
  <value> | --all | --signer <value>] [--start <value>] [--end <value>]
[--columns
  <value> | -x] [--filter <value>] [--no-header | [--csv | --no-
truncate]] [--output
  csv|json|yaml | | ] [--sort <value>]
```

FLAGS

```
-x, --extended
  show extra columns

--all
  get the status of all registered validators

--columns=<value>
  only show provided columns (comma-separated)

--csv
  output is csv format [alias: --output=csv]

--end=<value>
  [default: -1] what block to end at when looking at signer activity.
defaults to the
  latest block

--filter=<value>
  filter property by partial string matching, ex: name=foo

--gasCurrency=0x1234567890123456789012345678901234567890
  Use a specific gas currency for transaction fees (defaults to CEL0
if no gas
  currency is supplied). It must be a whitelisted token.

--globalHelp
  View all available global flags

--no-header
  hide table header from output

--no-truncate
  do not truncate output to fit screen

--output=<option>
  output in a more machine friendly format
  <options: csv|json|yaml>

--signer=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d
  address of the signer to check if elected and validating

--sort=<value>
  property to sort by (prepend '-' for descending)

--start=<value>
```

[default: -1] what block to start at when looking at signer activity. defaults to the last 100 blocks

--validator=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d
address of the validator to check if elected and validating

DESCRIPTION

Shows the consensus status of a validator. This command will show whether a validator is currently elected, would be elected if an election were to be run right now, and the percentage of blocks signed and number of blocks successfully proposed within a given window.

EXAMPLES

```
status --validator 0x5409ED021D9299bf6814279A6A1411A7e866A631

status --validator 0x5409ED021D9299bf6814279A6A1411A7e866A631 --start 1480000

status --all --start 1480000 --end 1490000
```

See code: src/commands/validator/status.ts

celocli validator:update-bls-public-key {#celocli-validatorupdate-bls-public-key}

Update the BLS public key for a Validator to be used in consensus.

USAGE

```
$ celocli validator:update-bls-public-key --from <value> --blsKey <value> --blsPop <value>
[--gasCurrency <value>] [--globalHelp]
```

FLAGS

| | |
|--|--------------|
| --blsKey=0x | (required) |
| BLS Public Key | |
| --blsPop=0x | (required) |
| BLS | Proof-of- |
| Possession | |
| --from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d | (required) |
| Validator's | address |
| --gasCurrency=0x1234567890123456789012345678901234567890 | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | |

supplied). It
whitelisted token.
--globalHelp
available global

currency is
must be a
View all
flags

DESCRIPTION

Update the BLS public key for a Validator to be used in consensus.

Regular (ECDSA and BLS) key rotation is recommended for Validator operational security.

WARNING: By default, the BLS key used by the validator node is derived from the ECDSA private key. As a result, rotating the BLS key without rotating the ECDSA key will result in validator downtime without special configuration. Use this method only if you know what you are doing.

EXAMPLES

```
update-bls-key --from 0x47e172F6CfB6c7D01C1574fa3E2Be7CC73269D95 --
blsKey
0x4fa3f67fc913878b068d1fa1cdddc54913d3bf988dbe5a36a20fa888f20d4894c408a67
73f3d7bde11154f2a3076b700d345a42fd25a0e5e83f4db5586ac7979ac2053cd95d8f2ef
d3e959571ceccaa743e02cf4be3f5d7aaddb0b06fc9aff00 --blsPop
0xcdb77255037eb68897cd487fdd85388cbda448f617f874449d4b11588b0b7ad8ddc20d9
bb450b513bb35664ea3923900
```

See code: `src/commands/validator/update-bls-public-key.ts`

validatorgroup.md:

```
celocli validatorgroup
=====
```

View and manage Validator Groups

```
celocli validatorgroup:commission
celocli validatorgroup:deregister
celocli validatorgroup:list
celocli validatorgroup:member ARG1
celocli validatorgroup:register
celocli validatorgroup:reset-slashing-multiplier ARG1
celocli validatorgroup:show ARG1
```

```
celocli validatorgroup:commission {#celocli-validatorgroupcommission}
```

Manage the commission for a registered Validator Group. This represents the share of the epoch rewards given to elected Validators that goes to the group they are a member of. Updates must be made in a two step process where the group owner first calls uses the queue-update option, then after the required update delay, the apply option. The commission update delay, in blocks, can be viewed with the network:parameters command. A groups next commission update block can be checked with validatorgroup:show

USAGE

```
$ celocli validatorgroup:commission --from <value> [--gasCurrency <value>] [--globalHelp] [--apply | --queue-update <value>]
```

FLAGS

| | |
|---|---------------|
| <code>--apply</code> | Applies a |
| previously queued | update. |
| Should be called | after the |
| update delay. | (required) |
| <code>--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d</code> | Validator |
| Address for the | Group |
| Group or Validator | |
| validator signer | |
| <code>--gasCurrency=0x1234567890123456789012345678901234567890</code> | Use a |
| specific gas currency | for |
| transaction fees | (defaults to |
| CELO if no gas | currency is |
| supplied). It | must be a |
| whitelisted token. | View all |
| <code>--globalHelp</code> | flags |
| available global | Queues an |
| <code>--queue-update=<value></code> | commission, |
| update to the | applied after |
| which can be | delay. |
| the update | |

DESCRIPTION

Manage the commission for a registered Validator Group. This represents the share of the epoch rewards given to elected Validators that goes to the group they are a member

of. Updates must be made in a two step process where the group owner first calls uses the queue-update option, then after the required update delay, the apply option. The

commission update delay, in blocks, can be viewed with the network:parameters command.

A groups next commission update block can be checked with validatorgroup:show

EXAMPLES

```
commission --from 0x47e172F6CfB6c7D01C1574fa3E2Be7CC73269D95 --queue-update 0.1
```

```
commission --from 0x47e172F6CfB6c7D01C1574fa3E2Be7CC73269D95 --apply
```

See code: src/commands/validatorgroup/commission.ts

```
celocli validatorgroup:deregister {#celocli-validatorgroupderegister}
```

Deregister a Validator Group. Approximately 180 days after the validator group is empty, it will be possible to deregister it start unlocking the CELO. If you wish to deregister your validator group, you must first remove all members, then wait the required 180 days before running this command.

USAGE

```
$ celocli validatorgroup:deregister --from <value> [--gasCurrency <value>]
[--globalHelp]
```

FLAGS

```
--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d (required)
Signer or
```

ValidatorGroup's address

```
--gasCurrency=0x1234567890123456789012345678901234567890 Use a
specific gas currency for
transaction fees (defaults to
CELO if no gas currency is
supplied). It must be a
whitelisted token. View all
--globalHelp available global flags
```

DESCRIPTION

Deregister a Validator Group. Approximately 180 days after the validator group is

empty, it will be possible to deregister it start unlocking the CELO.
If you wish to
deregister your validator group, you must first remove all members,
then wait the
required 180 days before running this command.

EXAMPLES

```
deregister --from 0x47e172f6cfb6c7d01c1574fa3e2be7cc73269d95
```

See code: `src/commands/validatorgroup/deregister.ts`

```
celocli validatorgroup:list {#celocli-validatorgroup:list}
```

List registered Validator Groups, their names (if provided), commission,
and members.

USAGE

```
$ celocli validatorgroup:list [--gasCurrency <value>] [--globalHelp] [-  
-columns <value>  
| -x] [--filter <value>] [--no-header | [--csv | --no-truncate]] [--  
output  
csv|json|yaml | | ] [--sort <value>]
```

FLAGS

```
-x, --extended  
    show extra columns  
  
--columns=<value>  
    only show provided columns (comma-separated)  
  
--csv  
    output is csv format [alias: --output=csv]  
  
--filter=<value>  
    filter property by partial string matching, ex: name=foo  
  
--gasCurrency=0x1234567890123456789012345678901234567890  
    Use a specific gas currency for transaction fees (defaults to CELO  
if no gas  
    currency is supplied). It must be a whitelisted token.  
  
--globalHelp  
    View all available global flags  
  
--no-header  
    hide table header from output  
  
--no-truncate  
    do not truncate output to fit screen  
  
--output=<option>  
    output in a more machine friendly format
```

```

    <options: csv|json|yaml>

    --sort=<value>
        property to sort by (prepend '-' for descending)

DESCRIPTION
    List registered Validator Groups, their names (if provided),
    commission, and members.

EXAMPLES
    list

See code: src/commands/validatorgroup/list.ts

celocli validatorgroup:member ARG1 {#celocli-validatorgroupmember-arg1}

Add or remove members from a Validator Group

USAGE
    $ celocli validatorgroup:member ARG1 --from <value> [--gasCurrency
<value>]
    [--globalHelp] [--yes] [--accept | --remove | --reorder <value>]

ARGUMENTS
    ARG1 Validator's address

FLAGS
    --accept
validator whose
is already set
--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d
ValidatorGroup's
--gasCurrency=0x1234567890123456789012345678901234567890
specific gas currency
transaction fees
CELO if no gas
supplied). It
whitelisted token.
--globalHelp
available global
--remove
validator from the
Accept a
affiliation
to the group
(required)
address
Use a
for
(defaults to
currency is
must be a
View all
flags
Remove a
members list

```

```
--reorder=<value>
validator within
```

Reorder a
the members

list. Indices

are 0 based
Answer yes to

```
--yes
prompt
```

DESCRIPTION

Add or remove members from a Validator Group

EXAMPLES

```
member --from 0x47e172f6cfb6c7d01c1574fa3e2be7cc73269d95 --accept 0x97f7333c51897469e8d98e7af8653aab468050a3
```

```
member --from 0x47e172f6cfb6c7d01c1574fa3e2be7cc73269d95 --remove 0x97f7333c51897469e8d98e7af8653aab468050a3
```

```
member --from 0x47e172f6cfb6c7d01c1574fa3e2be7cc73269d95 --reorder 3
0x47e172f6cfb6c7d01c1574fa3e2be7cc73269d95
```

See code: `src/commands/validatorgroup/member.ts`

```
celocli validatorgroup:register {#celocli-validatorgroupregister}
```

Register a new Validator Group

USAGE

```
$ celocli validatorgroup:register --from <value> --commission <value>
[--gasCurrency
  <value>] [--globalHelp] [--yes]
```

FLAGS

```
--commission=<value>
```

The share of the

```
(required)
epoch rewards
elected
```

given to

Validators that goes

to the group.
(required)

```
--from=0xc1912fEE45d61C87Cc5EA59DaE31190FFFFf232d
Address for the
```

Validator

Group

```
--gasCurrency=0x1234567890123456789012345678901234567890
specific gas currency
```

Use a
for

transaction fees

```
(defaults to
```

CELO if no gas

currency is

supplied). It

whitelisted token.

--globalHelp
available global

--yes
prompt

DESCRIPTION

Register a new Validator Group

EXAMPLES

register --from 0x47e172F6CfB6c7D01C1574fa3E2Be7CC73269D95 --commission
0.1

See code: src/commands/validatorgroup/register.ts

celocli validatorgroup:reset-slashing-multiplier ARG1 {#celocli-
validatorgroupreset-slashing-multiplier-arg1}

Reset validator group slashing multiplier.

USAGE

\$ celocli validatorgroup:reset-slashing-multiplier ARG1 [--gasCurrency
<value>]
[--globalHelp]

ARGUMENTS

ARG1 ValidatorGroup's address

FLAGS

--gasCurrency=0x1234567890123456789012345678901234567890 Use a
specific gas currency
transaction fees
CELO if no gas
supplied). It
whitelisted token.
--globalHelp
available global

must be a
View all
flags
for
(defaults to
currency is
must be a
View all
flags

DESCRIPTION

Reset validator group slashing multiplier.

EXAMPLES

reset-slashing-multiplier 0x97f7333c51897469E8D98E7af8653aAb468050a3

See code: src/commands/validatorgroup/reset-slashing-multiplier.ts

celocli validatorgroup:show ARG1 {#celocli-validatorgroupshow-arg1}

Show information about an existing Validator Group

USAGE

\$ celocli validatorgroup:show ARG1 [--gasCurrency <value>] [--globalHelp]

ARGUMENTS

ARG1 ValidatorGroup's address

FLAGS

--gasCurrency=0x1234567890123456789012345678901234567890 Use a specific gas currency for transaction fees (defaults to CELO if no gas currency is supplied). It must be a whitelisted token. View all available global flags

DESCRIPTION

Show information about an existing Validator Group

EXAMPLES

show 0x97f7333c51897469E8D98E7af8653aAb468050a3

See code: src/commands/validatorgroup/show.ts

CIP-contributors.md:

title: Community Improvement Proposals

description: Join a community of developers, designers, dreamers, and doers building prosperity for everyone.

Community Improvement Proposals

Celo's Improvement Proposals \ (CIPs\) describe standards for the Celo platform, including the core protocol specifications, SDK, and contract standards. A CIP is a design document that should provide background information, a rationale for the proposal, detailed solution including technical specifications, and, if any, a list of potential risks. The

proposer is responsible for soliciting community feedback and for driving consensus.

Participation in the Celo project is subject to the Code of Conduct.

Submitting CIPs

Draft all proposals following the template below and submit to the CIPs repository via a PR \(\pull request\).

CIP template:

- Summary: Describe your proposal in 280 characters or less.
- Abstract: Provide a short description of the technical issue being addressed.
- Motivation: Clearly explain why the proposed change should be made. It should layout the current Celo protocol shortcomings it addresses and why doing so is important.
- Specification: Define and explain in detail the technical requirements for new features and/or changes proposed.
- Rationale: Explain the reasoning behind your approach. It should cover alternative approaches considered, related work, and trade-offs made.
- Implementation: For all proposals going through the governance process, this section should reference the code implementing the proposed change. It's recommended to get community feedback before writing any code.
- Risks: Highlight any risks and concerns that may affect consensus, proof-of-stake, governance, protocol economics, the stability protocol, security, and privacy.

:::tip

For questions, comments, and discussions please use the Celo Forum or Discord.

:::

code-contributors.md:

```
---
title: Celo Code Contributors
description: How to contribute to the Celo ecosystem as a member of the
community.
---
```

Code Contributors

How to contribute to the Celo ecosystem as a member of the community.

Who are Code Contributors?

Code Contributors are developers in the Celo community that contribute to the Celo monorepo or the core protocol code. They help improve the protocol and infrastructure by fixing bugs and designing new features that help improve the Celo platform.

How to Get Started

Find an area that is of interest and you would like to help with. Look for issues that are tagged as "good first issue", "help wanted" and "1 hour tasks" to get started. If you'd like to dig deeper, feel free to look at other labels and TODO's in code comments. If there's an issue you're interested in contributing to or taking over, assign yourself to it and add a comment with your plans to address and target timeline. If there's already someone assigned to it, please check with them before adding yourself to the assignee list.

Tasks range from minor to major improvements. Based on your interests, skillset, and level of comfort with the code-base feel free to contribute where you see appropriate. Our only ask is that you follow the guidelines below to ensure a smooth and effective collaboration.

Please make sure your PR:

- Requests the appropriate reviewers. When in doubt, consult the CODEOWNERS file for suggestions.
- Provides a comprehensive description of the problem addressed and changes made.
- Explains dependencies and backwards incompatible changes .
- Contains unit and end-to-end tests and a description of how these were run.
- Includes changes to relevant documentation.

If you are submitting an issue, please double check that there doesn't already exist an issue for the work you have in mind.

Please make sure your issue:

- Is created in the correct repository.
- Has a clear detailed title such that it can't be confused with other Celo issues.
- Provides a comprehensive description of the current and expected behavior including, if relevant, links to external references and specific implementation guidelines.
- Is tagged with the relevant labels.
- Is assigned if you or someone else is already working on it.

Finding Us and Other Contributors

:::tip

For questions, comments, and discussions please use the Celo Forum or Discord.

:::

```
# code-of-conduct.md:
```

```
---
```

```
title: Code of Conduct
```

```
---
```

All communication and contributions to the Celo project are subject to the Celo Code of Conduct.

```
# contributing.md:
```

```
---
```

```
title: Celo Community
```

```
description: Join a community of developers, designers, dreamers, and doers building prosperity for everyone.
```

```
---
```

```
import PageRef from '@components/PageRef'
```

```
import Tabs from '@theme/Tabs';
```

```
import TabItem from '@theme/TabItem';
```

Celo Community

Join a community of developers, designers, dreamers, and doers building prosperity for everyone.

```
---
```

Celo is a decentralized community of creators -- developers, designers, dreamers, doers -- who are motivated by the power of accessible financial tools to make the world a better place. We are guided by Celo's Core tenets.

:::tip

Not ready to become a Celo Contributor? Learn more about Celo.

:::

Get Started with the Celo Community

Find your purpose and join a community focused on building the conditions of prosperity—for everyone

- Developer Events
- Contributors
- Code of Conduct
- Grant Playbook
- Release Process
- Join the Community

:::tip

For questions, comments, and discussions please use the Celo Forum or Discord.

:::

developer-events.md:

title: Developer Events

description: Find Celo events focused on the developer community.

Events

Overview of developer events in the Celo community.

import PageRef from '@components/PageRef'

import Tabs from '@theme/Tabs';

import TabItem from '@theme/TabItem';

Hackathons

- Celo at ETHDenver

documentation-contributors.md:

title: Celo Technical Writers

description: How to contribute to the Celo community documentation.

Technical Writers

How to contribute to the Celo community documentation.

import YouTube from '@components/YouTube';

Who are Celo technical writers?

Technical writers support the Celo community by educating developers about Celo through engaging, informative, and insightful documentation.

How to Contribute

<YouTube videoId="DaAenTNv668"/>

Edit an existing page

To edit an existing page in the documentation, create a fork of the repo, commit your edits and submit a PR.

- Go to the page in the docs
- Click "Edit this page" at the bottom of the page
- Edit the page directly on GitHub
- Describe the edit in the commit
- Select "Create a new branch and start a pull request"
- Describe changes in the Pull Request (PR)
- Select "critesjosh" as a reviewer
- Changes must be approved and pass all of the site build checks before being merged.

Add/remove pages

To add a new page to the documentation, create a fork, add the new pages and update the table of contents file to include your new pages in the appropriate location and submit a PR.

- Add or delete pages directly in Github
- Put new pages where you think makes the most sense, we can move them later
- Create a PR to have your changes added to the live version of the site
- Update the file called "sidebars.js" in the main folder
- This file contains the site layout that you see on the left side of the docs site
- Add or remove the appropriate files from the list

:::tip

For questions, comments, and discussions please use the Celo Forum or Discord.

:::

fundraising.md:

title: Fundraising

description: Hitchhiker's Guide to Fundraising in the Celo Ecosystem.

Fundraising

Hitchhiker's Guide to Fundraising in the Celo Ecosystem.

Overview

The process of raising capital for your company is often opaque and intimidating especially for first time founders. It's a skill that is acquired through practice and repetition. If you're new to fundraising, you'll find that you're crafting a product to sell. You're the seller. The product is your narrative around yourself, your team, company vision and upside potential. The investors are the buyer.

We created this guide to provide Celo Ecosystem builders an end-to-end resource on fundraising best practices as well as a map of Celo's Investor Ecosystem. While our goal isn't to reinvent the wheel and rather it's to distill a few key takeaways from industry experts, we hope you can leverage this guide and included resources to help you navigate your journey. Web3 is an emerging category, but we trust that many of the battle tested methods that have shown success can be applicable in this category.

Expert Resources

To get you started, here are a few comprehensive fundraising resources that we've found to be extremely helpful. This guide attempts to abstract and simplify important takeaways from these resources for entrepreneurs who are newer to fundraising.

- YCombinator's Geoff Ralston - A Guide to Seed Fundraising
- Paul Graham's - How to Raise Money
- NFX's Gigi Levy-Weiss - The Non Obvious Guide to Fundraising
- Carta's Paige Smith - When To Raise Money for A Startup
- Founder Collective's - A Ridiculously Detailed Fundraising Guide

Raising Capital

When should I start thinking about raising capital?

This is a tricky question that has no single right answer. The fact that the answer varies based on the entrepreneur and market shows why this process is opaque. In general, you should keep these factors in mind when considering the optimal time to start your fundraising process:

Warm Relationships

In some cases, founders may have no trouble raising capital with just an idea. Perhaps they may have warm relationships with the investors who can quantify the entrepreneurs' track record. Frankly speaking, entrepreneurs in this category have more flexibility on when they can raise capital given the established relationships.

No Warm Relationships

In most situations where you're a first time founder and may not have pre-existing relationships with investors, they'll generally prefer that you have a vision, some form of a tangible product, attractive market size and early signs of product demand before engaging. Once you've achieved most of the latter, it may be a good time to start thinking

about raising capital to accelerate your growth and capture more of your target market.

Market Timing

It's also important to zoom out and consider the macroeconomic cycle and market timing of your potential fundraise. VC is an asset class that has been flooded with capital as investors continue to search for yield. <https://pitchbook.com/news/articles/2021-us-vc-fundraising-exits-deal-flow-charts>. Fundraising activity in Web3 is no exception and continues to hit all time highs <https://www.cnbc.com/2022/02/02/crypto-start-ups-raised-huge-venture-funding-rounds-in-january.html>. At certain points in the market cycle compared to others, it will be easier to raise capital given investors' need to deploy the large funds they've raised from their limited partners and the chase for "hot sectors". You may have an advantage and more options for capital if you fundraise during these "hot" market cycles. Additionally, it's worth keeping track of investors on social media to understand what they're interested in funding at the moment.

Seasonality

Ensure to time your fundraising process that aligns well when investors are actively investing and have the capacity to run full due diligence processes. As a rule of thumb, try your best not to start a fundraising process around major holiday seasons to reduce the risk of extending your timeline longer than you had originally planned.

What do I need to research before starting my fundraise?

While fundraising can take an unexpected amount of time, it's important to optimize for efficiency to the best of your ability. The last thing you want to do is spend hours with potential investors who aren't investing in your stage/sector of business. We're aware of emerging fundraising models in Web3 (i.e. DAO-based funds), however our aim is to keep this guide confined to more traditional methods for simplicity.

1. Find investors who fit your target investor profile! Using the following databases/resources listed below, you can easily identify the investors who are actively investing in your stage/sector/geography.

- Web3 Focused

- Celo's Ecosystem Investors Map
- Dove Metrics (<https://www.dovemetrics.com/> is an excellent resource for Web3 market fundraising data)

- Investor Databases

- AngelList <https://www.angellist.com/>
- Crunchbase <https://www.crunchbase.com/>
- First Round Capital's Angel Directory <https://angels.firstround.com/directory>
- NFX Signal <https://signal.nfx.com/>

2. Ensure that the investors you identify are actively investing. For example, try finding investors who have made 1-2 relevant investments within the last 2-4 months using the databases below.

3. Once you've identified a subset of investors who may be a fit for your sector/geography/stage, aggregate a list in your personal database to stay organized. Some popular tools include pre-built templates on Airtable or even creating your own tracker on Google Sheets. As your number of conversations grows, it's important to keep track of the status of each conversation and to understand where you should be doubling efforts to increase your chances of success.

4. If one of your target investors is a venture capital fund, take some time to look through their website as well to look for a few things

- If they show their portfolio companies, take some time to review and see if they've invested in any competitors. There may be a conflict of interest even if they are a sector/stage/geographical fit for you and you want to cross out these investors quickly.
- Research the team and see who may be the right person to reach. Venture capital funds often have a broad team and there may be certain individuals who are focused on your sector vs. others at the firm. You want to be sure that you're reaching out to the right person.

5. If one of your target investors is an angel investor, take some time to research their background and personal investment thesis. Do they have attractive experience that you could leverage to grow your company?

6. Frankly speaking and similar to sales, cold emails to investors have a low chance of success statistically speaking. For any investors who you identify to be a prospective fit, search through your social networks and see if you may have any mutual connections. People tend to trust others who they have some history with. If you find any mutual connections, try kindly reaching out to the individual and see if they'd be willing to forward along your company's teaser materials to the prospective investor.

7. If you've found a great venture capital firm and/or angel investor but have no mutual connections to the team or individual, you can attempt a cold outreach. Remember, fundraising at the earliest stages will be a numbers game. However, you also want to be as thorough and thoughtful as possible when reaching out to the investors. In sales, unique approaches can attract a few eyeballs even if they hit the inbox cold. A few things to keep in mind

- Keep the outreach email as concise as possible. Investors receive a ton of inbound in this market so you'll only have a minute if not a few seconds of their attention. You want to get straight to the point.
- Specify why you reached out to them and why they would be a great fit for your company. (I.E. Perhaps they've invested in other companies that are tangential to what you're building or have written an article with a thesis that aligns with your company vision.

- Share a few points on your company taken from your teaser materials and include your investor presentation.
 - What is the problem you're solving for and is it a large market?
 - Why is now the best time to tackle this opportunity?
 - How are you solving this problem?
 - Traction (if applicable and be transparent)
 - How much are you raising?
 - What's special about your team and why are you the ones to solve this problem?

How much capital do I need to raise?

The rule of thumb here is to raise enough capital to give you 1 year of runway at minimum and best case 2 years of runway with the primary goal of reaching your next milestone. In other words, think through how much monthly operating capital you'll need (across hires and other administrative budget items) and multiply that by the number of months you believe it'll take to reach that milestone. This resulting figure can serve as a reference point but it's often helpful to size up and down your potential round size to give yourself and investors a magnitude of what you could realistically achieve with different levels of capital. Finmark (<https://finmark.com/>) is an example of a helpful resource that can help build out your financial plan.

What are the different forms of capital I can raise?

While new forms of funding models emerge in Web3 and a comprehensive list will grow over time, below are a few common investment structures used by founders and investors today.

1. Grants - Many layer 1 protocols have recently set up ecosystem funds in order to incentivize and fund entrepreneurs who build on their technology stack. These grants generally come in the form of non-dilutive capital (meaning that the capital you take does not reduce your ownership of your company) that can help you build a minimum viable product and/or test your market hypothesis.

2. SAFE (Simple Agreement for Future Equity) - YCombinator's SAFE document has become an industry standard used for early stage fundraising and for good reasons - they provide speed and flexibility for both investors and entrepreneurs. They're a form of a convertible note however, unlike convertible debt, SAFEs remove the need for interest payments and repayment of the debt. The capital invested on a SAFE is generally expected to convert into shares of preferred equity in a priced equity round. YCombinator's website <https://www.ycombinator.com/documents/#about> provides excellent best practices on SAFEs and templates as well.

3 SAFT (Simple Agreement for Future Tokens) - For a period of time during the boom of crypto fundraising, SAFTs were frequently used by companies to raise capital. While most investors expect financial upside on their equity in traditional companies, Web3 companies will often build and launch tokens which are eventually available to the public. Ideally, these tokens have sustainable utility and practical use cases and should

therefore accrue value from greater market adoption. Mechanically speaking, SAFTs offer the investors the ability to exchange capital for a promise of discounted tokens from the project at a future point in time. However, SAFTs have largely fallen out of favor due to their legal risks, potential tax implications, and complications that other companies have faced.

4. SAFE + Token Warrants - A combination of a SAFE and a token warrant has become increasingly popular for fundraising in Web3. Unlike SAFTs, these token warrants typically come in the form of an optional side letter and don't guarantee the deployment of tokens - therefore avoiding the legal complications faced by its predecessor. As a result, companies are more commonly raising capital on SAFEs along with a token warrant that offers investors the right to tokens in the amount that can be proportional to their equity ownership of the company. In this manner, investors have exposure to both equity and potential tokens if the companies decide to launch one.

- The team at LiquiFi has written an excellent piece on the topic and provides a practical guidance for token allocation.
<https://medium.com/@liquififinance/crypto-web3-fundraising-with-token-side-letters-or-token-warrants-33dce66f375e>.

- This resource from Coopahtroopa and Lauren Stephanian is also a must read for entrepreneurs who need to understand historical and current benchmarks on tokenomics.
<https://lstephanian.mirror.xyz/kB9Jz5joqbY0ePO8rU1NNDKhiqvzU6OWyYsbSA-Kcc>.

5. Equity - Equity rounds allow companies to set an official valuation for the company and to issue shares to investors at a specific price. Over time, these rounds are generally reserved for Series A and later stage companies. Companies at the seed stage have raised priced rounds historically, although more of them are raising on SAFEs at the earlier stages for the speed of putting operating capital to work.

What should I have prepared for investor conversations?

In general, we recommend that you prepare as much digestible information ahead of time to make your investor meetings productive. Having the baseline items listed below will serve you well in requesting warm introductions to investors, cold email outreaches and investor preparation prior to an initial conversation.

- Brieflink Page (Using <https://brieflink.com/>)
- 1-2 Paragraph Teaser Summary
- Investor Presentation
- Product Demo (You can pre-record these on <https://www.loom.com/>. In Web3, it may even make sense to have an interactive demo.)

The other items listed below are useful to share with investors if they've expressed interest in pursuing additional due diligence.

- Capitalization Table
- Financial Projections

What do I need in my investor presentation?

While this guide isn't meant to be 100% prescriptive, below are the core components of a comprehensive investor presentation. Remember, treat fundraising as if you were launching and selling a product. You want to craft a compelling enough narrative to make your company an easy choice for investors to fund. Your pitch and deck are always a work in progress and it's important to continuously refine your story and materials based on feedback that you receive from others.

- Company Vision (In a concise and powerful statement, what is your company's core value proposition and what key problem are you looking to solve?)
- Problem (Is this a real problem and is it critical to solve?)
- Market Size (What is your Total Addressable Market? It's okay to think big here but be realistic with data)
- Traction (Are there currently any users? TVL? Growth rates for any relevant metrics?)
- Business Model (In traditional companies, investors want to understand how your unit economics and monetization plans? In Web3, the upside is likely more attributable the future value of a utility token or adoption of a platform)
- Solution (What are the features of your product? How does it ultimately solve the problem? What blockchain is it built on? While Web3 is nascent and can be complicated, you want a simple enough explanation so that investors can communicate to their internal stakeholders as well.)
- Product Roadmap (What other segments can your product expand into?)
- Competitive Landscape (Competitive market map and positioning on why your product is different)
- Distribution Strategy (Are there other players in the ecosystem that you can work with to acquire other users in a low cost way?)
- Team (What is your team's background? Are you and your team the right market & team fit?)
- Fundraising Ask & Planned Use of Capital (How much capital are you looking to raise and how are you planning to allocate that capital? What milestones are you going to achieve with this capital?)

The team at NFX has put together a comprehensive sample pitch deck with relevant examples of slides of pitch decks from other companies.

We can't forget to mention beyond having these components to a pitch deck, it's important to think through the visual representation of your pitch deck as well. Most of you will have an investor presentation on screen while meeting investors and it's important to remember that the engagement is both audio and visual. We've found that <https://www.beautiful.ai/> is a great place to start if you're looking to create a visually compelling pitch deck as well.

Pitching Process

Below are a few best practices and things to consider when preparing for your investor pitches. This isn't all inclusive and your mileage will likely vary depending on your circumstances.

- Do your diligence on the investors prior to pitching in order to:
 - Understand focus areas so that you can curate your pitch accordingly
 - Uncover any red flags and or concerns about the investor that you can address during your meeting. Remember, as much as investors are evaluating you/your company, you're also in the position to evaluate them as a potential partner. Capital is a commodity and true value-add is very rare. A good way to do this is to reach out to founders in their portfolio or a broader entrepreneurial community for honest feedback on the investor
- Be concise about what you do
- Be concise about the problem
- Clearly articulate why this is the right market timing
- Confidently explain why you have the right team
- In Web3 specifically, it's more important now than ever to be able to articulate where the upside for investors will come from
- Let investors guide the conversation so that you can tackle the points that they truly care about. Pitches can generally take 15 minutes. You can leave the rest of the meeting time for an interactive conversation. The best investors will come prepared and ask insightful questions. If they don't, just know that they will likely serve only as a source of capital vs. a source of strategic help
- Be a human! Much of the remote nature of what we do today can feel transactional. It's important to build a genuine relationship with investors as they can be your biggest champion.
- Fundraising is a numbers game. You'll receive many more rejections than you will genuine interest. It's just a part of the process. Cast a wide net (as efficiently as possible), collect and then select.
- Know that investors will ghost you even if they've appeared to express interest during your meeting. They're paid to meet founders, so don't take it personally. Just move on to others!
- Follow up promptly with any requested materials. This indicates your level of responsiveness with future investors
- Be transparent about your process. Investors respect that and will align their process accordingly if they're genuinely interested in working with you

Post Investment

After raising capital from investors, it's important to keep a periodic cadence with your investors for a few reasons. Firstly, it helps keep you accountable to the milestones that you've set for your team and mitigates the need for investors to ask you ad-hoc questions - which can generally distract you as you're busy building your company. Secondly, it's a great open forum to request assistance from your investors. You can also gamify your updates by making special mentions of investors who have been the most helpful to you so far. Founder Collective has put together a detailed template for thorough and actionable investor updates. <https://visible.vc/templates/founder-collective-fill-in-the-blank-investor-update/>

Celo Ecosystem Investors

Who should I go to within the Celo Ecosystem?

If you're building on Celo and looking for fundraising, start with our Celo Ecosystem Investors! The Celo Foundation has partnered with a number of dedicated individuals and institutional investors who look to invest in companies building on Celo. Below you'll find a list of different investor groups who are actively investing in Celo's Ecosystem and additional details on their areas of focus.

!fundraising

Celo Camp

Celo Camp is an independent initiative managed by our partners at Upright. For founders who have an idea and would like to surround themselves with a community to build with, Celo Camp is a great program to help you get your idea off the ground. Towards the end of the program, investors in Celo's Ecosystem will attend and you'll have the opportunity to pitch them for a potential investment. To learn more about the program and the application process, please visit this link - <https://www.celocamp.com/>

Celo Web3 Studio (via Bld.ai)

Bld.ai is a Web3 studio that is committed to helping the best entrepreneurs bring to fruition some of their largest ideas on Celo. They work closely with you and your team to assess a market opportunity and provide resources and funding to help you build a product. To learn more about Bld.ai - please visit this link - <https://www.bld.ai/>

Celo Community Fund[Closed]

Celo Community Fund I (CCF I) was the first community approved spend proposal from the on-chain Community Fund (explorer). Community members seeking funding could apply to CCF I, which was managed by three community stewards. As of April 2022, the fund is closed due to achieving its objective and disbursing the amount of Celo recieved.

Prezenti

Prezenti is the next iteration of the CCF1 and was developed with the support of CCF1. A new onchain spend approval was proposed and accepted resulting in 800,000 Celo able to be deployed to people wanting to build their dreams upon the Celo protocol. It is managed by three new stewards and has four focus areas - Education, Research, Community Tooling and Other. Anyone is welcome to apply for a grant from the Prezenti Website and we use Questbook to distribute the funds and manage the grants.

Celo Foundation Grant Program

The Celo Foundation Grants Program is an open program that supports projects that are committed to Celo's mission of building a financial system that creates the conditions for prosperity for all. Applicants to the Celo Foundation Grant program are expected to have a viable product or offering for the community. Admitted applicants will be provided

capital and mentorship. To learn more about the program and the application process, please visit this link - <https://celo.org/experience/grants#purpose>.

Celo Scout Investors

Celo Scouts are individual investors who are hand selected by the Celo Foundation to invest in mission-aligned founders building on Celo. These investors tend to focus on early stage companies who may be looking for angel/pre-seed like funding and write check sizes between 25-50K cUSD on average.

Dedicated Institutional Celo Ecosystem Funds

Celo Ecosystem Funds are firms that the Celo Foundation has partnered with to evaluate larger private equity and/or token investments into companies building on Celo.

!fundraising

Flori Ventures

Flori Ventures is a seed stage venture fund already partnered with many projects built with Celo's technology. Their aim is to work with the financial innovators that help foster an inclusive financial system and are closely working with the Celo Foundation to achieve this goal. Co-Founder and Managing Partner, Maria Alegre, was the former Co-Founder and CEO of Chartboost (acquired by Zynga) and raised \$21M in funding from Sequoia Capital and other stellar investors. Co-Founder and General Partner, Tomer Bariach, has a long history as an expert in token economics and is well versed in economically impactful technologies.

- Stage Focus: Pre-Seed & Seed
- Geography Focus: Global
- Sector Focus: Companies that bring crypto to real world use cases & enable prosperity for all
- Average Check Size: \$100K

Unicorn Growth Fund

Unicorn Growth Capital is a Seed & Series A venture fund focused on partnering exclusively with founders innovating financial services in Africa. They're particularly excited about FinTech & DeFi infrastructure, embedded finance products and broad digital financial services that can foster an inclusive financial ecosystem.

Stage Focus: Seed & Series A

Geography Focus: Africa

Sector Focus: FinTech & DeFi Infrastructure, Embedded Finance, Financial Services

Average Check Size: \$500K for Seed and < 1.5M for Series A

Lightshift Capital

Lightshift Capital is an early stage fund focused on partnering with groundbreaking companies building infrastructure layers, DeFi protocols and emerging tools expanding use cases for Web3. They currently work with a community of over 50 investors, builders and ecosystems including individuals from Andreessen Horowitz, Digital Currency Group, Celo, Kraken and many others. They have a hands-on approach to working with founders at the earliest stages while leveraging support from their in-house technical and business teams to add value.

- Stage Focus: Pre-Seed & Seed
- Geography Focus: Europe, U.S. and Asia
- Sector Focus: Financial Services, DeFi, Infrastructure and Scalability
- Average Check Size: \ \$250K - 500K

TKX Digital Group

TKX is an Asia focused crypto investment bank that utilizes both active and passive strategies for primary and secondary markets of blockchain technology and cryptocurrency. They provide a full service stack including investment to advisory services to blockchain startups.

- Stage Focus: Pre-Seed & Seed
- Geography Focus: Europe, North & South America
- Sector Focus: GameFi - Infrastructure - Defi - NFT - MPC
- Average Check Size: \$50K-\$500K

Polychain Fund

Polychain Capital is a global early stage venture fund focused on investing in protocol layers and cryptocurrency companies. They separately manage the Celo Ecosystem Venture Fund that aims to foster Celo's mission of creating a more accessible financial system by making strategic, seed-stage deployments into tools and services which will leverage Celo's protocol to build new financial infrastructure.

- Stage Focus: Seed & Series A+
- Geography Focus: Global
- Sector Focus: DeFi, Cross-chain Interoperability, Intersection of crypto & gaming
- Average Check Size: \$500K - \$5M

Other Prominent Celo Ecosystem Investors:

The groups below have also invested in companies in the Celo Ecosystem and may be worth considering if you're looking to raise a larger round of institutional capital. If you're an investor that has invested in Celo's Ecosystem and would like to be included in the list, please reach out to celoinvestorecosystem@celo.org.

!fundraising

How do I get in touch with these investors?

Please reach out to celoinvestorecosystem@celo.org with your investor presentation and a short description of the company, and your round details and we can help get you connected with interested investors in Celo's Investor Ecosystem.

grant-playbook.md:

title: Celo Grant Playbook

description: Summary of best practices that grantees are encouraged to follow to further the Celo mission.

Grant Playbook

Summary of best practices that grantees are encouraged to follow to further the Celo mission.

Code of Conduct

The Celo Foundation believes in investing in projects that share Celo's mission of building an open financial system that creates conditions of prosperity for everyone. Everyone who engages with the Celo ecosystem must abide by the Code of Conduct. Please take the time to read through it with your team.

Communication Guidelines

Social Media Engagement

Creating an ecosystem that supports and encourages each other is important to the Celo community. As a team, please make sure to actively engage in social media.

- Setup a Twitter handle for your project (if you don't have one already)
- Publish (at least) one post per week
- Follow @CeloOrg and @CeloDevs
- Follow projects in your space & fellow grant recipients
- Identify and follow influencers who might be interested in your space. To help you get started, here is a list of influencers we follow.
- Post links to your blog posts, repos, etc
- Use the hashtag #CeloGrants so you can join the conversation about the grant program

Blog

Each project should have a dedicated blog. Two suggestions are devpost or Medium.

- Write a blog post at least once per month/milestone outlining your progress to date (i.e. accomplishments and challenges). Include links to repos, demos, or social media.
- If using Medium, please share your Medium profile tag with community@celo.org so we may add you as a writer for the Celo Medium Publication
- Here are some writing tips:
 - Celo Blogging Guide
 - How to write a killer press release
 - How to write a new product announcement

Engagement & Documentation

Source Code

In the spirit of decentralization, The Celo Foundation expects all funded projects to be open-source by default -- this means that the codebase is public and under a permissive license (we recommend Apache 2.0). If there are any parts that need to be kept private (eg. an admin key), please let us know ahead of time ([at grants@celo.org](mailto:grants@celo.org)), and document it in your project's README.

To make sure teams are making progress on projects, send us bi-weekly updates to grants@celo.org. The intent is not to create pressure, but to provide structure and guidance in case things get off track. Some suggestions:

- Create a top-level README.md to provide a description of your project, goals, features, roadmap and installation instructions.
- Ensure that all work is regularly committed to a repo. Track and close issues.

Chat

- Make sure to ask questions in a relevant Discord channel. You can join the Celo Discord [here](#). The cLabs protocol and product teams are all there and available for questions. Try not to take conversations to direct messages -- by having conversations in an open channel, other groups can also benefit from the technical discussions and learnings. You can tag mentors in public channel messages to ensure your messages are seen.
- Please make an effort to remain visible on #general-applications and/or #mobile-development channels in the App Developer section. These channels are dedicated to teams working on projects on the Celo ecosystem. There is a good chance someone has encountered similar challenges. It's also a great way to find ways to integrate with other projects in the space.
- Please make sure your entire team is in this chat room.
- Also please post your tweets and blog posts so that other grantees can share social media.

Milestone Reviews & Payment

Once you have completed a grant milestone, please send an email to your POC at the Celo Foundation. The grant committee will then review to ensure objectives have been met.

After your milestone has been approved by the grant committee, payment will be issued.

Evaluating Grant Success

Before the final payment, grants will be evaluated using the following framework:

- Engagement: How well you follow best practices & engage with the Celo community
- Visibility: Maintaining transparency on your project accomplishments and challenges
- Roadmap: Ability to reference a list of commits that complete your roadmap
- Timeline/Milestones: Ability to meet your timeline and key milestones
- Launch: Successful launch!

guidelines.md:

title: Celo Contributor Guidelines

description: Join a community of developers, designers, dreamers, and doers building prosperity for everyone.

Contributors

Guidelines for submitting contributions to the Celo community.

```
import PageRef from '@components/PageRef'
import Tabs from '@theme/Tabs';
import TabItem from '@theme/TabItem';
```

Celo is open source and we welcome open participation. We strive to fulfill our Community Tenets by being an open and inclusive community where everyone feels welcome and empowered to contribute. This also means following some ground rules and abiding by Celo's Code of Conduct.

- Raise an issue in the repository that you would like to contribute to or find one you feel comfortable solving - fill in the requested information and paste your contribution.
- Clone the repo and follow the guidelines below to submit your contribution.
- Create a pull request

How to Contribute

Our community includes a group of contributors who help develop, write, translate, and improve Celo. Anyone is welcome to join the community and contribute their skills to help empower other community members and grow the Celo ecosystem.

- Code
- CIPs
- Docs
- Tutorials
- Translations

Contributor Guidelines

There are a few basic ground rules for contributing:

- Please fork the repository
- PRs \(\pull requests\) are preferred for solving issues, especially for small changes such as typos. Issues should be used for missing features and for broad-based changes.
- For on-going work, use your own side-branch and not the master branch.
- For non-trivial amounts of work, we encourage you to submit PRs regularly to solicit feedback.
- Please double check your work before submitting it. Submissions with typos, spelling, and grammatical errors may not be merged until fixed.
- Try to remain as objective and fact-based as possible.

Submitting PRs

We encourage you to PR \(\pull request\) your work regularly and often to solicit feedback and to ensure everyone has an idea of what you're working on. If you've just started, we suggest creating a PR with "WIP" \(\Work In Progress\) in the title and let us know when it's ready to review in the comments.

□ Code

Code Contributors are developers in the Celo community that contribute to the Celo monorepo or the core protocol code. They help improve the protocol and infrastructure by fixing bugs and designing new features that help improve the Celo platform.

How to Get Started

Find an area that is of interest and you would like to help with. Look for issues that are tagged as "good first issue", "help wanted" and "1 hour tasks" to get started. If you'd like to dig deeper, feel free to look at other labels and TODO's in code comments. If there's an issue you're interested in contributing to or taking over, assign yourself to it and add a comment with your plans to address and target timeline. If there's already someone assigned to it, please check with them before adding yourself to the assignee list.

Tasks range from minor to major improvements. Based on your interests, skillset, and level of comfort with the code-base feel free to contribute

where you see appropriate. Our only ask is that you follow the guidelines below to ensure a smooth and effective collaboration.

Please make sure your PR

- Requests the appropriate reviewers. When in doubt, consult the CODEOWNERS file for suggestions.
- Provides a comprehensive description of the problem addressed and changes made.
- Explains dependencies and backwards incompatible changes .
- Contains unit and end-to-end tests and a description of how these were run.
- Includes changes to relevant documentation.

If you are submitting an issue, please double check that there doesn't already exist an issue for the work you have in mind.

Please make sure your issue

- Is created in the correct repository.
- Has a clear detailed title such that it can't be confused with other Celo issues.
- Provides a comprehensive description of the current and expected behavior including, if relevant, links to external references and specific implementation guidelines.
- Is tagged with the relevant labels.
- Is assigned if you or someone else is already working on it.

□ CIPs

Celo's Improvement Proposals \ (CIPs\) describe standards for the Celo platform, including the core protocol specifications, SDK, and contract standards. A CIP is a design document that should provide background information, a rationale for the proposal, detailed solution including technical specifications, and, if any, a list of potential risks. The proposer is responsible for soliciting community feedback and for driving consensus.

Submitting CIPs

Draft all proposals following the template below and submit to the CIPs repository via a PR \ (pull request\).

CIP template

- Summary: Describe your proposal in 280 characters or less.
- Abstract: Provide a short description of the technical issue being addressed.
- Motivation: Clearly explain why the proposed change should be made. It should layout the current Celo protocol shortcomings it addresses and why doing so is important.
- Specification: Define and explain in detail the technical requirements for new features and/or changes proposed.

- Rationale: Explain the reasoning behind your approach. It should cover alternative approaches considered, related work, and trade-offs made.
- Implementation: For all proposals going through the governance process, this section should reference the code implementing the proposed change. It's recommended to get community feedback before writing any code.
- Risks: Highlight any risks and concerns that may affect consensus, proof-of-stake, governance, protocol economics, the stability protocol, security, and privacy.

■ Docs

Technical writers support the Celo community by educating developers about Celo through engaging, informative, and insightful documentation.

Edit an existing page

To edit an existing page in the documentation, create a fork of the repo, commit your edits, and submit a PR.

- Go to the page in the docs
- Click Edit this page at the bottom of the page
- Edit the page directly on GitHub
- Describe the edit in the commit
- Select "Create a new branch and start a pull request"
- Describe changes in the Pull Request (PR)
- Select "joenyzio" as a reviewer
- Changes must be approved and pass all of the site build checks before being merged.

Add/remove pages

To add a new page to the documentation, create a fork, add the new pages, and update the table of contents file to include your new pages in the appropriate location and submit a PR.

- Add or delete pages directly in GitHub
- Put new pages where you think makes the most sense, we can move them later
- Create a PR to have your changes added to the live version of the site
- Update the file called "sidebars.js" in the main folder
- This file contains the site layout that you see on the left side of the docs site
- Add or remove the appropriate files from the list

📖 Tutorials

Write about your experience as a member of the Celo community, whether you're a CELO owner, developer, or project founder. Your experience and perspective are valuable and can help others.

File naming

Creating a new post in the blog is straightforward. Create a new file in the blog directory in the documentation repository. Filenames follow the

format of YYYY-MM-DD-post-name.md. For example, this post was written November 8th, 2021 so it has the filename 2021-11-08-contributing.md.

Front Matter

Posts are written in Markdown. Posts include front matter. The front matter is file metadata at the top of the file that provides more information about the post. The front matter for this post looks like this:

```
md
---
title: Contributing to the Blog
description: How to contribute to the blog
slug: blog-contributions
authors:
  - name: Josh Crites
    title: Developer Relations, cLabs
    url: https://github.com/critesjosh
    imageurl: https://github.com/critesjosh.png
tags: [contribute]
image: https://i.imgur.com/mErPwqL.png
hidetableofcontents: false
---
```

Post summary

Pages can also include a `<!--truncate-->` tag that specifies what text will be shown along with the post title on the post list page. Any text above `<!--truncate-->` will appear as the post summary.

Adding static assets

If you would like to include images or other static assets in a post, you can create a folder following the naming convention described above (YYYY-MM-DD-post-name). The contents of the folder can include the images and the post (with filename index.md).

🌐 Translations

Translators support the community expanding Celo to non-English speaking communities by translating and sharing content in different languages.

How to Contribute

- Go to <https://celo.crowdin.com/>
- Create an account
- Start translating!
- Submit questions with Crowdin Messages
- Translation request form

:::tip

For questions, comments, and discussions please use the Celo Forum or Discord.

:::

join-the-community.md:

title: Join the Celo Community

description: Find Celo on social media, contribute to the codebase, or chat with the community.

Join the Community

Find Celo on social media, contribute to the codebase, or chat with the community.

Social Media

Follow on Social Media to learn more about Celo.

- Blog
- GitHub
- Twitter
- Forum
- Chat
- YouTube
- Instagram
- LinkedIn
- Twitch
- Reddit
- Telegram

Discussions

Ask questions, find answers, and get in touch.

- Celo Forum
- Celo Developer Chat on Discord
- Celo Subreddit
- Celo Website
- Host a Meetup

Contributions

Browse the code, raise an issue, or contribute a pull request.

- Monorepo GitHub Page
- Celo Client GitHub Page
- Contributing Guide

- Celo Developers Page

translation-contributors.md:

title: Celo Translators

description: Join a community of developers, designers, dreamers, and doers building prosperity for everyone.

Translators

How to contribute to the Celo community as a translator.

Who are Translation Contributors?

Translators support the community expanding Celo to non-English speaking communities by translating and sharing content in different languages.

How to Contribute

- Go to <https://celo.crowdin.com/>
- Create an account
- Start translating!
- Submit questions with Crowdin Messages
- Translation request form

:::tip

For questions, comments, and discussions please use the Celo Forum or Discord.

:::

attestation-service.md:

title: Celo Attestation Service Release Process

description: Details of the release process for updating the attestation service on the Celo platform.

Attestation Service Release Process

Details of the release process for updating the attestation service on the Celo platform.

:::tip note

This release process is currently in use.

:::

Versioning

Releases of Attestation Service are made as needed. Releases are numbered according to semantic versioning, as described at semver.org.

Development builds should be identified with `-dev`, and only one commit should exist with a released version `x.y.z` for any `(x, y, z)`.

Documentation

Documentation is maintained in the `celo-org/docs` repo and is hosted on docs.celo.org.

Identifying releases

Git branches

Development is done on the master branch, which corresponds to the next major or minor version. Changes to be included in a patch release of an existing minor version are cherry-picked to that existing release branch.

Git tags

Each release should be created on Github and tagged with the version number, e.g. `attestation-service-vX.Y.Z`. Each release should include a summary of the release contents, including links to pull requests and issues with detailed description of any notable changes.

Tags should be signed and can be verified with the following command.

```
bash
git verify-tag attestation-service-vX.Y.Z
```

On Github, each release tag should have attached signatures that can be used to verify the Docker images.

Docker tags

Each Docker image is tagged with `attestation-service-<commithash>`. Just as a Git tag immutably points to a commit hash, the Docker tag should immutably point to an image hash.

In addition, each Docker image corresponding to a released version should be tagged with `attestation-service-vX.Y.Z`.

The latest image qualified for deployment to various networks are also tagged as follows:

- Alfajores: attestation-service-alfajores
- Baklava: attestation-service-baklava
- Mainnet: attestation-service-mainnet

Signatures

Artifacts produced by this build process (e.g. tags, Docker images) will be signed by a core developer key.

Public keys for core developers are hosted on celo.org and can be imported to gpg with the following command:

```
bash
gpg --auto-key-locate wkd --locate-keys $EMAIL
```

Currently hosted core developer keys used for Attestation Service releases include:

- tim@clabs.co

Build process

Docker images

Docker images are built automatically with Google Cloud Build upon pushes to master and all release branches. Automated builds will be tagged in Google Artifact Registry with the corresponding commit hash.

A signature should be produced over the image automatically built at the corresponding commit hash and included with the Github release.

Release image signatures can be verified with the following command:

```
bash
docker save $(docker image inspect us.gcr.io/celo-testnet/celo-
monorepo:attestation-service-vX.Y.Z -f '{{ .Id }}') | gpg --verify
attestation-service-vX.Y.Z.docker.asc -
```

Testing

As well as monorepo CI tests, all releases are expected to go through manual testing as needed to verify security properties, accuracy of documentation, and compatibility with deployed and anticipated versions of celocli and wallets including Valora. Releases currently involve coordinating with Valora to run the verification e2e tests in CI.

Promotion process

Source control

Patch releases should be constructed by cherry-picking all included commits from master to the release/attestation-service/x.y branch, if

necessary created from the attestation-service-vX.Y.Z tag of the most recent major or minor release. The first commit of this process should change the version number encoded in the source from x.y.z to x.y.z+1-dev and the final commit should change the version number to x.y.z+1.

Major and minor releases should be constructed by pushing a commit to the master branch to change the encoded version number from x.y.z-dev to x.y.z. A attestation-service-vX.Y.Z tag should be created at this commit which uniquely references one commit; release notes should be published alongside this. The next commit should change the version number from x.y.z to x.y+1.0-dev, or x+1.0.0-dev if the next planned release is a major release.

Distribution

Distribution of an image follows this schedule:

| Date | Action |
|--------------|--|
| T-1w | <ol style="list-style-type: none">Deploy release candidate build to Alfajores testnetTest manually and via e2e verification tests |
| T | <ol style="list-style-type: none">Confirm Valora production and testing builds against Alfajores experience no issues and that e2e verification tests complete successfullyPublish the release notes and tag the relevant commit on GitHubTag released Docker image with <code>attestation-service-alfajores</code>, <code>attestation-service-baklava</code>, <code>attestation-service-mainnet</code>, and <code>attestation-service-vX.Y.Z</code> tags (removing tags from other releases)Inform the community of the new release via Discord and the Celo Forum |
| T+1w onwards | <ol style="list-style-type: none">Confirm Mainnet services have upgraded without issues |

```

        <li>Continue monitoring dashboards for user issues</li>
    </ol>
</td>
</tr>
</table>

```

Emergency Patches

Bugs which affect the security, stability, or core functionality of the Celo identity protocol or prevent new users onboarding to wallets including Valora may need to be released outside the standard release cycle. In this case, an emergency patch release should be created on top of all supported minor releases which contains the minimal change and corresponding test for the fix.

If the issue is not exploitable, release notes should describe the issue in detail and the image should be distributed publicly.

If the issue is exploitable and mitigations are not readily available, a patch should be prepared privately and signed binaries should be distributed from private commits. Establishing trust is key to pushing out the fix. An audit from a reputable third party may be contracted to verify the release to help earn that trust.

Vulnerability Disclosure

Vulnerabilities in Attestation Service releases should be disclosed according to the security policy.

```
# base-cli-contractkit-dappkit-utils.md:
```

```

---
title: Celo Release Process for CeloCLI and ContractKit
description: Details of the release process for updating CeloCLI and
ContractKit on the Celo platform.
---

```

Release Process for CeloCLI and ContractKit

Details of the release process for updating CeloCLI and ContractKit on the Celo platform.

```
---
```

Versioning

Use the standard MAJOR.MINOR.PATCH semantic versioning scheme described at semver.org.

New releases can be expected as follows:

- Major releases: approximately yearly
- Minor releases: approximately 8 times a year

- Patch releases: as needed

Development builds will be identified as such: x.y.z-dev, and will be published as x.y.z when stable.

Identifying releases

NPM

You can find the npm packages in the following places:

- @celo/celocli
- @celo/contractkit

Github tags

To identify the commits included in a specific release and see which new features were added or bugs fixed, please refer to the release notes in the monorepo. Also to keep track of continual updates to the stable and dev versions of the packages, each package has a CHANGELOG.md file: Celocli and Contractkit.

All releases should be tagged with the version number, e.g. contractkit-vX.Y.Z. Each release should include a summary of the release contents, including links to pull requests and issues with detailed description of any notable changes.

Communication

The community will be notified of package updates through the following channels:

For all releases:

- Each package's CHANGELOG.md file, as mentioned above
- Github releases page, as mentioned above
- Discord: #developer-chat, #mainnet, and #sdk

For major releases:

- Twitter: @CeloDevs
- Mailing list: cLabs' Tech Sync
- Celo Forum

Testing

All builds of these packages are automatically tested for performance and backwards compatibility in CI. Any regressions in these tests should be considered a blocker for a release.

Minor and major releases are expected to go through additional rounds of manual testing as needed to verify behavior under stress conditions.

:::warning

Work in progress

:::

Promotion process

- For a patch release: The first step of this process should be a commit that changes the version number encoded in the source from x.y.z-dev to x.y.z+1-dev and the final step should change the published version number from x.y.z-1 to x.y.z.
- For minor releases, the same process should be followed, except the y value would increment, and the z value would become 0.
- For major releases, the same process should be followed, except the x value would increment, and y and z values would become 0.

Only one commit should ever have a non-dev tag at any given version number. When that commit is created, a tag should be added along with release notes. Once the tag is published it should not be reused for any further release or changes.

Emergency patches

Bugs which affect the security, stability, or core functionality of the network may need to be released outside the standard release cycle. In this case, an emergency patch release should be created on top of all supported minor releases which contains the minimal change and corresponding test for the fix. An emergency patch retro will also be published, and will include information such as why the patch was necessary and what code changes it includes.

Vulnerability Disclosure

Vulnerabilities in any of these releases should be disclosed according to the security policy.

Dependencies

- @celo/mobile - Dappkit relies on this
- Celocli
- All the packages under the "SDK" folder -- These all rely on each other quite a bit, so triple-check that these packages weren't affected by a change in another.

Dependents

- Celocli
- All the packages under the "SDK" folder

blockchain-client.md:

title: Celo Blockchain Client Release Process

description: Details of the release process for updating the blockchain client on the Celo platform.

Blockchain Client Release Process

Details of the release process for updating the blockchain client on the Celo platform.

Versioning

Releases of celo-blockchain are numbered according to semantic versioning, as described at semver.org.

All builds are identified as unstable (a development build) or stable (a commit released as a particular version number). There should only ever exist one commit with a version x.y.z-stable for any (x, y, z).

Signatures

Artifacts produced by this build process (e.g. Docker images) will be signed by cosign.

Documentation

The documentation for client features, such as APIs and commands, is maintained in the docs directory within the celo-blockchain repository. Documentation on protocol features, such as the proof-of-stake protocol, is hosted on docs.celo.org.

Identifying releases:

Git branches

Each minor version of celo-blockchain has its own "release branch", e.g. release/1.0.

Development is done on the master branch, which corresponds to the next major or minor version. Changes to be included in a patch release of an existing minor version are cherry-picked to that existing release branch.

Git tags

All releases should be tagged with the version number, e.g. vX.Y.Z. Each release should include a summary of the release contents, including links to pull requests and issues with detailed description of any notable changes.

Tags should be signed and can be verified with the following command.

bash

```
git verify-tag vX.Y.Z
```

On Github, each release tag should link to the respective Docker image, along with signatures that can be used to verify those images.

Docker tags

Each released Docker image should be tagged with its version number such that for release x.y.z, the image should have tags x, x.y, and x.y.z, with the first two tags potentially being moved from a previous image. Just as a Git tag x.y.z immutably points to a commit hash, the Docker tag, x.y.z should immutably point to an image hash.

Build process

Docker images

Docker images are built automatically with Google Cloud Build upon pushes to master and all release branches. Automated builds will be tagged in Google Artifact Registry with the corresponding commit hash.

A signature should be produced over the image automatically built at the corresponding commit hash and included with the GitHub release.

Release image signatures can be verified with the following command:

```
bash
docker save $(docker image inspect us.gcr.io/celo-org/geth:X.Y.Z -f '{{
.Id }}') | gpg --verify celo-blockchain-vX.Y.Z.docker.asc -
```

Testing

All builds of celo-blockchain are automatically tested for performance and backwards compatibility in CI. Any regressions in these tests should be considered a blocker for a release.

Minor and major releases are expected to go through additional rounds of manual testing as needed to verify behavior under stress conditions, such as a network with faulty nodes, and poor network connectivity.

Promotion process

Source control

Patch releases should be constructed by cherry-picking all included commits from master to the release/x.y branch. The first commit of this process should change the version number encoded in the source from x.y.z-stable to x.y.z+1-unstable and the final commit should change the version number to x.y.z+1-stable.

Major and minor releases should be constructed by pushing a commit to the master branch to change the encoded version number from x.y.z-unstable to x.y.z-stable. A release/x.y branch should be created from this commit. The next commit must change the version number from x.y.z-stable to x.y+1.0-unstable, or x+1.0.0-unstable if the next planned release is a major release.

Only one commit should ever have a "stable" tag at any given version number. When that commit is created, a tag should be added along with release notes. Once the tag is published it should not be reused for any further release or changes.

Emergency Patches

Bugs which affect the security, stability, or core functionality of the network may need to be released outside the standard release cycle. In this case, an emergency patch release should be created on top of all supported minor releases which contains the minimal change and corresponding test for the fix.

If the issue is not exploitable, release notes should describe the issue in detail and the image should be distributed publicly.

If the issue is exploitable and mitigations are not readily available, a patch should be prepared privately, and signed binaries should be distributed from private commits. Establishing trust is key to pushing out the fix. An audit from a reputable third party may be contracted to verify the release to help earn that trust. Once a majority of validators are updated, patch details can be made public.

> Pushing an upgrade with this process will be disruptive to any nodes that do not upgrade quickly. It should only be used when the circumstances require it.

Vulnerability Disclosure

Vulnerabilities in celo-blockchain releases should be disclosed according to the security policy

```
# celo-oracles.md:
```

```
---
```

```
title: Celo Oracles Release Process
```

```
description: Details of the release process for updating the Celo Oracles for the Mento Stability Protocol
```

```
---
```

Celo Oracles Release Process

Details of the release process for updating the Celo Oracles Client for the Mento Stability Protocol.

Versioning

Releases of are made as needed. Releases are numbered according to semantic versioning, as described at semver.org.

Development builds should be identified with a `-dev` suffix, and only one commit should exist with a released version `x.y.z` for any `(x, y, z)`. Release candidates should be identified with a `-rCX` suffix, where `X` is the version of the release candidate.

Documentation

Documentation is maintained in the `celo-org/docs` repo and is hosted on docs.celo.org.

Identifying releases

Git branches

Code for the client is stored in the Celo Oracles GitHub repository. Development is done on the `main` branch, which corresponds to the next major or minor version.

Git tags

Each release should be created on Github and tagged with the version number, e.g. `X.Y.Z`. Each release should include a summary of the release contents, including links to pull requests and issues with detailed description of any notable changes. Tags should be annotated tags.

Tags should be signed and can be verified with the following command.

```
bash
git verify-tag vX.Y.Z
```

On Github, each release tag should have attached signatures that can be used to verify the Docker images.

Docker tags

Docker images are stored in the repository `us-west1-docker.pkg.dev/celo-testnet-production/celo-oracle` repository, stored in Google Cloud Platform Artifact Registry. Commits there are tagged with `celo-oracle-X.Y.Z` and `celo-oracle-<commithash>`. Just as a Git tag immutably points to a commit hash, the Docker tag should immutably point to an image hash.

Testing

As well as automated CI tests, all releases are expected to go through manual testing as needed to verify security properties, accuracy of documentation, and compatibility with current node operators production set up.

Promotion process

Cherry picked branch changes shall be added to a releases protected branch. When merging code to this branch, the version number should be updated accordingly.

Source control

Distribution

| Date | Action |
|--------------|---|
| T-1w | <ol style="list-style-type: none">Deploy release candidate build to Alfajores testnetMonitor metrics and behavior |
| T | <ol style="list-style-type: none">Tag the release and publish Docker image as described in this documentInform the community of the new release via Discord and the Celo Forum |
| T+1w onwards | <ol style="list-style-type: none">Confirm Mainnet services have upgraded without issuesContinue monitoring dashboards for user issues |

Emergency Patches

Security fixes or hotfixes may not have a public commit attached to them in case the vulnerability needs to be patched before disclosing to the general public to prevent an attacker to exploit a vulnerability before operators patch their services. Emergency patches will be distributed using the same method described in Git tags

Vulnerability Disclosure

Vulnerabilities in Celo Oracles should be disclosed according to the security policy.

index.md:

title: Celo Release Process

description: Overview of the release process for updates to the Celo platform.

import PageRef from '@components/PageRef'

Release Process

Overview of the release process for updates to the Celo platform.

:::tip

It is critical that updates to the Celo platform can be released on a regular basis, and in a way that ensures the security and reliability of the Celo network. In order to facilitate this, the following release processes are published here.

:::

- Smart Contracts
- Blockchain Client
- CeloCLI and ContractKit
- Attestation Service

smart-contracts.md:

title: Celo Smart Contracts Release Process

description: Details of the release process for updating smart contracts on the Celo platform.

Smart Contracts Release Process

export const N = "N";

Details of the release process for updating smart contracts on the Celo platform.

:::warning

This release process is a work in progress. Many infrastructure components required to execute it are not in place, and the process itself is subject to change.

:::

Versioning

Each deployed Celo core smart contract is versioned independently, according to semantic versioning, as described at semver.org, with the following modifications:

- STORAGE version when you make incompatible storage layout changes
- MAJOR version when you make incompatible ABI changes
- MINOR version when you add functionality in a backwards compatible manner, and
- PATCH version when you make backwards compatible bug fixes.

Changes to core smart contracts are made via on-chain Governance, approximately four times a year. When a release is made, all smart contracts from the release branch that differ from the deployed smart contracts are released, and included in the same governance proposal. Each release is identified by a unique monotonically increasing version number N, with 1 being the first release.

Core Contracts

Every deployed Celo core contract has its current version number as a constant which is publicly accessible via the `getVersionNumber()` function, which returns the storage, major, minor, and patch versions. The version number is encoded in the Solidity source and updated as part of code changes.

Celo Core Contracts deployed to a live network without the `getVersionNumber()` function, such as the original set of core contracts, are to be considered version 1.1.0.0.

Mixins and libraries

Mixin contracts and libraries are considered part of the contracts that consume them. When a mixin or library has changed, all contracts that consume them should be considered to have changed as well, and thus the contracts should have their version numbers incremented and should be re-deployed as part of the next smart contract release.

Initialize Data

Whenever Celo Core Contracts need to be re-initialized, their initialization arguments should be checked into version control under `packages/protocol/releaseData/initializationData/release${N}.json`.

Release management in Git/Github

Github branches/tags and Github releases are used to coordinate past and ongoing releases. Ongoing smart contract development is done on the master branch (even after release branches are cut). Every smart contract release has a designated release branch, e.g. release/core-contracts/\${N} in the celo-monorepo.

When a new release branch is cut:

1. A new release branch is created release/core-contracts/\${N} with the contracts to be audited.
2. The latest commit on the release branch is tagged with core-contracts.v\${N}.pre-audit.
3. On Github, a pre-release Github release should be created pointing at the latest tag on the release branch.
4. On master branch, .circleci/config.yml should be edited so that the variable RELEASETAG points to the tag celo-core-contracts-v\${N}.pre-audit so that all future changes to master are versioned against the new release.
5. Ongoing audit responses/fixes should continue to go into release/celo-core-contracts/\${N}.

After a completed release process:

1. The release branch should be merged into master with a merge commit (instead of the usual squash merge strategy).
2. On master branch, .circleci/config.yml should be edited so that the variable RELEASETAG points to the tag core-contracts.v\${N}

Release Process

There are several scripts provided (under packages/protocol in celo-org/celo-monorepo and via celocli) for use in the release process and with contract upgrade governance proposals to give participating stakeholders increased confidence.

:::warning

For these to run, you may need to follow the setup instructions. These steps include installing Node and setting nvm to use the correct version of Node. Successful yarn install and yarn build in the protocol package signal a completed setup.

:::

Using these tools, a contract release candidate can be built, deployed, and proposed for upgrade automatically on a specified network. Subsequently, stakeholders can verify the release candidate against a governance upgrade proposal's contents on the network.

Typical script options:

- By default, the scripts expect a celo-blockchain RPC at port 8545 locally. With -f you can specify the scripts to use a hosted forno node

- By default, scripts will output verbose logs under /tmp/celo-\${script-name}.log. You can change the location of the log output with -l file.log

View the tagged releases for each network

```
bash
yarn view-tags
```

Verify the previous Release on the Network

verify-deployed is a script that allows you to assess whether the bytecode on the given network matches the source code of a particular commit. It will run through the Celo Core Contracts and verify that the contracts' bytecodes as specified in the Registry match. Here, we will want to sanity-check that our network is running the previous release's audited commit.

```
bash
Run from packages/protocol in the celo-monorepo
PREVIOUSRELEASE="core-contracts.v${N-1}"
NETWORK=${"baklava"|"alfajores"|"mainnet"}
A -f boolean flag can be provided to use a forno full node to connect to
the provided network
yarn verify-deployed -n $NETWORK -b $PREVIOUSRELEASE -f
```

A libraries.json file is written to disk only necessary for make-release that describes linked library addresses.

Check Backward Compatibility

This script performs some automatic checks to ensure that the smart contract versions in the source code have been set correctly with respect to the latest release. It is run as part of CI and helps ensure that backwards incompatibilities are not accidentally introduced by requiring that devs manually update version numbers whenever smart contract changes are made.

Specifically, it compiles the latest and candidate releases and compares smart contracts:

1. Storage layout, to detect storage version changes
2. ABI, to detect major and minor version changes
3. Bytecode, to detect patch version changes

Finally, it checks release candidate smart contract version numbers and requires that they have been updated appropriately since the latest release by following semantic versioning as defined in the Versioning section above.

The following exceptions apply:

- If the STORAGE version has changed, it does not perform backward compatibility checks
- If the MAJOR version has changed, it checks storage layout compatibility but not ABI compatibility

Critically, this ensures that proxied contracts do not experience storage collisions between implementation versions. See this [article](#) by OpenZeppelin for a good overview of this problem and why it's important to check for it.

The script generates a detailed report on version changes in JSON format.

```
bash
PREVIOUSRELEASE="core-contracts.v${N-1}"
RELEASECANDIDATE="core-contracts.v${N}"
yarn check-versions -a $PREVIOUSRELEASE -b $RELEASECANDIDATE -r
"report.json"
```

This should be used in tandem with `verify-deployed -b $PREVIOUSRELEASE -n $NETWORK` to ensure the compatibility checks compare the release candidate to what is actually active on the network.

Deploy the release candidate

Use the following script to build and deploy a candidate release. This takes as input the corresponding backward compatibility report and canonical library address mapping to deploy changed contracts to the specified network. (Use `-d` to dry-run the deploy). STORAGE updates are adopted by deploying a new proxy/implementation pair. This script outputs a JSON contract upgrade governance proposal.

```
bash
NETWORK=${"baklava"|"alfajores"|"mainnet"}
RELEASECANDIDATE="core-contracts.v${N}"
yarn make-release -b $RELEASECANDIDATE -n $NETWORK -r "report.json" -i
"releaseData/initializationData/release${N}.json" -p "proposal.json" -l
"libraries.json"
```

The proposal encodes STORAGE updates by repointing the Registry to the new proxy. Storage compatible upgrades are encoded by repointing the existing proxy's implementation.

Submit Upgrade Proposal

Submit the autogenerated upgrade proposal to the Governance contract for review by voters, outputting a unique identifier.

```
bash
resultant proposal ID should be communicated publicly
```

```
celocli governance:propose --deposit 100e18 --from $YOURADDRESS --  
jsonTransactions "proposal.json" --descriptionURL  
https://github.com/ceio-org/governance/blob/main/CGPs/cgp-0055.md
```

Fetch Upgrade Proposal

Fetch the upgrade proposal and output the JSON encoded proposal contents.

```
bash  
Make sure you run at least celocli 0.0.60  
celocli governance:show --proposalID <proposalId> --jsonTransactions  
"upgradeproposal.json"
```

Verify Proposed Release Candidate

This script serves the same purpose as verify-deployed but for a not-yet accepted contract upgrade (in the form of the proposal.json you fetched in the step prior). It gives you the confidence that the branch specified in the -b flag in (same as check-versions) will be the resulting network state of the proposal if executed. It does so by going over all Celo Core Contracts and determining updates to the Registry pointers, proxy or implementation contracts and verifying their implied bytecode against the compiled source code.

Additionally, include initializationdata.json from the CGP if any of the contracts have to be initialized.

```
bash  
RELEASECANDIDATE="core-contracts.v${N}"  
NETWORK=${"baklava"|"alfajores"|"mainnet"}  
A -f boolean flag can be provided to use a forno full node to connect to  
the provided network  
yarn verify-release -p "upgradeproposal.json" -b $RELEASECANDIDATE -n  
$NETWORK -f -i initializationdata.json
```

Verify Executed Release

After a release executes via Governance, you can use verify-deployed again to check that the resulting network state does indeed reflect the tagged release candidate:

```
bash  
RELEASE="core-contracts.v${N}"  
NETWORK=${"baklava"|"alfajores"|"mainnet"}  
yarn verify-deployed -n $NETWORK -b $RELEASE -f
```

Testing

All releases should be evaluated according to the following tests.

Unit tests

All changes since the last release should be covered by unit tests. Unit test coverage should be enforced by automated checks run on every commit.

Manual Checklist

After a successful release execution on a testnet, the resulting network state should be spot-checked to ensure that no regressions have been caused by the release. Flows to test include:

- Do a cUSD and CELO transfer

```
bash
celocli transfer:dollars --from <addr> --value <number> --to <addr>
celocli transfer:celo --from <addr> --value <number> --to <addr>
```
- Register a Celo account

```
bash
celocli account:register --from <addr> --name <test-name>
```
- Report an Oracle rate

```
bash
celocli oracle:report --from <addr> --value <num>
```
- Do a CP-DOTO exchange

```
bash
celocli exchange:celo --value <number> --from <addr>
celocli exchange:dollars --value <number> --from <addr>
```
- Complete a round of attestation
- Redeem from Escrow
- Register a Validator

```
bash
celocli validator:register --blsKey <hexString> --blsSignature
<hexString> --ecdsaKey <hexString> --from <addr>
```
- Vote for a Validator
- Run a mock election

```
bash
celocli election:run
```
- Get a validator slashed for downtime and ejected from the validator set
- Propose a governance proposal and get it executed

```
bash
celocli governance:propose --jsonTransactions <jsonFile> --deposit
<number> --from <addr> --descriptionURL
https://gist.github.com/yorhodes/46430eachb8ed2f73f7bf79bef9d58a33
```

Automated environment tests

Stakeholders can use the env-tests package in celo-monorepo to run an automated test suite against the network

Verify smart contracts

Verification of smart contracts should be done both on <https://celoscan.io/> and <https://explorer.celo.org/>.

1. Update your Smart Contract on celoscan
2. Update your Smart Contract on the Celo Exploere

Performance

A ceiling on the gas consumption for all common operations should be defined and enforced by automated checks run on every commit.

For troubleshooting please see Readme.md of protocol package.

Backwards compatibility

Automated checks should ensure that any new commit to master does not introduce a breaking change to storage layout, ABI, or other common backward compatibility issues unless the STORAGE or MAJOR version numbers are incremented.

Backwards compatibility tests will also be run before every release to confirm that no breaking changes exist between the pending release and deployed smart contracts.

Audits

All changes since the last release should be audited by a reputable third party auditor.

Emergency patches

If patches need to be applied before the next scheduled smart contract release, they should be cherry-picked to a new release branch, branched from the latest deployed release branch.

Promotion process

Deploying a new contract release should occur with the following process. On-chain governance proposals should be submitted on Tuesdays for consistency and predictability.

```
<table>
  <tr>
    <td>Date</td>
    <td>Action</td>
  </tr>
  <tr>
    <td>T</td>
    <td>
      <ol>
```

```

        <li>
            Create a Github issue tracking all these checklist items as an
audit
            log
        </li>
        <li>
            Implement the{" "}
            <a href="/community/release-process/smart-contracts#When-a-new-
release-branch-is-cut">git management steps</a>{" "}
            for when a new release branch is cut.
        </li>
        <li>
            Submit the release branch to a reputable third party auditor
for review.
        </li>
        <li>Begin drafting release notes.</li>
    </ol>
</td>
</tr>
<tr>
    <td>T+1w</td>
    <td>
        <ol>
            <li>Receive report from auditors.</li>
            <li>Add audit summary to the final draft of the release
notes.</li>
            <li>
                If all issues in the audit report have straightforward fixes:
                <ol>
                    <li>
                        {" "}
                        Submit a governance proposal draft using this format:
                    </li>
                    <li>
                        {" "}
                        Add any initialization data to the CGP that should be
included as
                        part of the proposal
                    </li>
                    <li>
                        {" "}
                        Announce forthcoming smart contract release on:
                        https://forum.celo.org/c/governance
                    </li>
                </ol>
            </li>
            <li>Commit audit fixes to the release branch</li>
            <li>Submit audit fixes to auditors for review.</li>
            <li>
                Tag the first release candidate commit according to the{" "}
                <a href="/community/release-process/smart-contracts#During-the-
release-proposal-stage">
                    git release management instructions
                </a>

```

```

        .
    </li>
    <li>
        Let the community know about the upcoming release proposal by
posting
        details to the Governance category on https://forum.celo.org
and cross
        post in the{" "}
        <a
href="https://discord.com/channels/600834479145353243/704805825373274134"
>Discord <code>#governance</code> channel.</a>
        See the 'Communication guidelines' section below for
information on what
        your post should contain.
    </li>
</ol>
</td>
</tr>
<tr>
    <td>T+2w</td>
    <td>
        <ol>
        <li>
            On Tuesday: Run the{" "}
            <a href="/community/release-process/smart-contracts#release-
process">smart contract release script</a>{" "}
            in order to to deploy the contracts to Baklava as well as
submit a
            governance proposal.
        <ul>
        <li>
            Transition proposal through Baklava governance process.
        </li>
        <li>
            Update your forum post with the Baklava
<code>PROPOSALID</code>,
            updated timings (if any changes), and notify the community
in the
            Discord <code>#governance</code> channel.
        </li>
        </ul>
        </li>
        </ol>
    </td>
</tr>
<tr>
    <td>T+3w</td>
    <td>
        <ol>
        <li>Confirm all contracts working as intended on Baklava.</li>
        <li>
            Run the{" "}
            <a href="/community/release-process/smart-contracts#release-
process">

```

```

        smart contract release script
    </a>{" "}
    in order to to deploy the contracts to Alfajores as well as
submit a
    governance proposal.
</li>
<li>
    Update your forum post with the Alfajores
<code>PROPOSALID</code>,
    updated timings (if any changes), and notify the community in
the
    Discord <code>#governance</code> channel.
</li>
</ol>
</td>
</tr>
<tr>
    <td>T+4w</td>
    <td>
        <ol>
            <li>Confirm all contracts working as intended on Alfajores.</li>
            <li>
                Confirm audit is complete and make the release notes and forum
post
                contain a link to it.
            </li>
            <li>
                On Tuesday: Run the{" "}
                <a href="https://docs.celo.org/community/release-process/smart-
contracts#build-process">
                    smart contract release script
                </a>{" "}
                in order to to deploy the contracts to Mainnet as well as
submit a
                governance proposal.
            </li>
            <li>
                Update the corresponding governance proposal with the updated
on-chain{" "}
                <code>PROPOSALID</code> and mark CGP status as "PROPOSED".
            </li>
            <li>
                Update your forum post with the Mainnet
<code>PROPOSALID</code>,
                updated timings (if any changes), and notify the community in
the
                Discord <code>#governance</code> channel.
            </li>
            <li>
                At this point all stakeholders are encouraged to{" "}
                <a href="/community/release-process/smart-contracts#verify-
release-process">verify</a> the proposed contracts
                deployed match the contracts from the release branch.
            </li>
        </ol>
    </td>
</tr>

```



```

<li>
  Monitor the progress of the proposal through the{" "}
  <a href="/protocol/governance">governance process.</a>
<ul>
  <li>
    Currently the governance process should take approximately
1 week:
    24 hours for the dequeue process, 24 hours for the approval
    process, and 5 days for the referendum process. After
which, the
    proposal is either declined or is ready to be executed
within 3
    days.
  </li>
  <li>
    For updated timeframes, use the celocli:{" "}
    <code>celocli network:parameters</code>
  </li>
</ul>
</li>
</ol>
</td>
</tr>
<tr>
  <td>T+5w</td>
  <td>
    <ol>
    <li>
      If the proposal passed:
      <ol>
      <li>Confirm all contracts working as intended on
Mainnet.</li>
      <li>
        Update your forum post with the Mainnet governance outcome
        (
          <code>Passed</code> or <code>Rejected</code>) and notify
the
          community in the Discord <code>#governance</code> channel.
        </li>
      <li>Change corresponding CGP status to EXCECUTED.</li>
      <li>
        Merge the release branch into <code>master</code> with a
merge
        commit
      </li>
      </ol>
    <li>
      If the proposal failed:
      <ol>
      <li>Change corresponding CGP status to EXPIRED.</li>
      </ol>
    </li>
  </ol>
</td>
</tr>

```

```
</td>
</tr>
</table>
```

If the contents of the release (i.e. source Git commit) change at any point after the release has been tagged in Git, the process should increment the release identifier, and process should start again from the beginning. If the changes are small or do not introduce new code (e.g. reverting a contract to a previous version) the audit step may be accelerated.

Communication guidelines

Communicating the upcoming governance proposal to the community is critical and may help getting it approved.

Each smart contract release governance proposal should be accompanied by a Governance category forum post that contains the following information:

- Name of proposer (individual contributor or organization).
- Background information.
- Link to the release on Github.
- Link to the audit report(s).
- Anticipated timings for the Baklava and Alfajores testnets and Mainnet.

It's recommended to post as early as possible and at minimum one week before the anticipated Baklava testnet governance proposal date.

Make sure to keep the post up to date. All updates (excluding fixing typos) should be communicated to the community in the Discord #governance channel.

Emergency patches

:::warning

Work in progress

:::

Vulnerability Disclosure

Vulnerabilities in smart contract releases should be disclosed according to the security policy.

Dependencies

None

Dependents

:::warning

Work in progress

:::

celo-ethers-wrapper.md:

title: Celo Ethers.js Wrapper
description: A minimal wrapper to make Ethers.JS compatible with the Celo network.

Celo Ethers.JS Wrapper

A minimal wrapper to make Ethers.JS compatible with the Celo network.

:::tip

This is still experimental. View on GitHub

:::

Install

npm i @celo-tools/celo-ethers-wrapper

or

yarn add @celo-tools/celo-ethers-wrapper

Note this wrapper has Ethers v5 as a peer dependency. Your project must include a dependency on that as well.

Basic Usage

Connect to the network by creating a CeloProvider, which is based on JsonRpc-Provider:

```
js
import { CeloProvider } from "@celo-tools/celo-ethers-wrapper";

// Connecting to Alfajores testnet
const provider = new CeloProvider("https://alfajores-forno.celo-testnet.org");
await provider.ready;
```

Note: for a more efficient provider based on StaticJsonRpcProvider you can use StaticCeloProvider instead.

Next, Create a CeloWallet, which is based on Wallet :

```
js
import { CeloWallet } from "@celo-tools/celo-ethers-wrapper";

const wallet = new CeloWallet(YOURPK, provider);
```

Use the provider or wallet to make calls or send transactions:

```
js
const txResponse = await wallet.sendTransaction({
  to: recipient,
  value: amountInWei,
});
const txReceipt = await txResponse.wait();
console.info(CELO transaction hash received:
${txReceipt.transactionHash});
```

Contract Interaction

CeloWallet can be used to send transactions.

Here's an example of sending cUSD with the Mento StableToken contract. For interacting with contracts you need the ABI and address. Addresses for Celo core contracts can be found with the CLI's `network:contracts` command. The ABIs can be built from the solidity code or extracted in ContractKit's generated folder.

```
js
import { Contract, ethers, utils, providers } from "ethers";

const stableToken = new ethers.Contract(address, abi, wallet);
console.info(Sending ${amountInWei} cUSD);
const txResponse: providers.TransactionResponse =
  await stableToken.transferWithComment(recipient, amountInWei, comment);
const txReceipt = await txResponse.wait();
console.info(cUSD payment hash received: ${txReceipt.transactionHash});
```

Alternative gas fee currencies

The Celo network supports paying for transactions with the native asset (CELO) but also with the Mento stabletoken (cUSD).

This wrapper currently has partial support for specifying feeCurrency in transactions.

```
js
const gasPrice = await wallet.getGasPrice(stableTokenAddress);
const gasLimit = await wallet.estimateGas(tx);

// Gas estimation doesn't currently work properly for non-CELO currencies
// The gas limit must be padded to increase tx success rate
// TODO: Investigate more efficient ways to handle this case
```

```

const adjustedGasLimit = gasLimit.mul(10);

const txResponse = await signer.sendTransaction({
  ...tx,
  gasPrice,
  gasLimit: adjustedGasLimit,
  feeCurrency: stableTokenAddress,
});

# evm-tools.md:

---
title: Celo EVM Compatible Tooling
description: Overview of Celo EVM-compatible tools and the value they
provide to developers.
---

Developer Tools

A guide to available tools, components, patterns, and platforms for
developing applications on EVM-compatible chains.

---

New developers start here

- Solidity - The most popular smart contract language.
- Metamask - Browser extension wallet to interact with Dapps.
- thirdweb - SDKs in every language, smart contracts, tools, and
infrastructure for web3 development.
- Hardhat - Flexible, extensible and fast Ethereum development
environment.
- Cryptotux - A Linux image ready to be imported in VirtualBox that
includes the development tools mentionned above
- OpenZeppelin Starter Kits - An all-in-one starter box for developers to
jumpstart their smart contract backed applications. Includes OpenZeppelin
SDK, the OpenZeppelin/contracts-ethereum-package EVM package of audited
smart contract, a react-app and rimbale for easy styling.
- EthHub.io - Comprehensive crowdsourced overview of Ethereum- its
history, governance, future plans and development resources.
- EthereumDev.io - The definitive guide for getting started with Ethereum
smart contract programming.
- Brownie - Brownie is a Python framework for deploying, testing and
interacting with Ethereum smart contracts.
- Ethereum Stack Exchange - Post and search questions to help your
development life cycle.
- dfuse - Slick blockchain APIs to build world-class applications.
- Biconomy - Do gasless transactions in your dapp by enabling meta-
transactions using simple to use SDK.
- Blocknative - Blockchain events before they happen. Blocknative's
portfolio of developers tools make it easy to build with mempool data.

```

- useWeb3.xyz - A curated overview of the best and latest resources on Ethereum, blockchain and Web3 development.

Developing Smart Contracts

Smart Contract Languages

- Solidity - Ethereum smart contracting language
- Vyper - New experimental pythonic programming language

Frameworks

- thirdweb - Provides the tools needed to build custom smart contracts efficiently by offering a set of prebuilt base contracts and a set of reusable components, or extensions, that can be integrated into your own smart contracts.
- Hardhat - Flexible, extensible and fast Ethereum development environment.
- Embark - Framework for DApp development
- Dapp - Framework for DApp development, successor to DApple
- Etherlime - ethers.js based framework for Dapp deployment
- Parasol - Agile smart contract development environment with testing, INFURA deployment, automatic contract documentation and more. It features a flexible and unopinionated design with unlimited customizability
- Oxcert - JavaScript framework for building decentralized applications
- OpenZeppelin SDK - OpenZeppelin SDK: A suite of tools to help you develop, compile, upgrade, deploy and interact with smart contracts.
- sbt-ethereum - A tab-completey, text-based console for smart-contract interaction and development, including wallet and ABI management, ENS support, and advanced Scala integration.
- Cobra - A fast, flexible and simple development environment framework for Ethereum smart contract, testing and deployment on Ethereum virtual machine(EVM).
- Epirus - Java framework for building smart contracts.

IDEs

- Remix - Web IDE with built in static analysis, test blockchain VM.
- Ethereum Studio - Web IDE. Built in browser blockchain VM, Metamask integration (one click deployments to Testnet/Mainnet), transaction logger and live code your WebApp among many other features.
- Atom - Atom editor with Atom Solidity Linter, Etheratom, autocompletesolidity, and language-solidity packages
- Vim solidity - Vim syntax file for solidity
- Visual Studio Code - Visual Studio Code extension that adds support for Solidity
- Ethcode - Visual Studio Code extension to compile, execute & debug Solidity & Vyper programs
- IntelliJ Solidity Plugin - Open-source plug-in for JetBrains IntelliJ Idea IDE (free/commercial) with syntax highlighting, formatting, code completion etc.
- YAKINDU Solidity Tools - Eclipse based IDE. Features context sensitive code completion and help, code navigation, syntax coloring, build in compiler, quick fixes and templates.

- Eth Fiddle - IDE developed by The Loom Network that allows you to write, compile and debug your smart contract. Easy to share and find code snippets.

Other tools

- Atra Blockchain Services - Atra provides web services to help you build, deploy, and maintain decentralized applications on the Ethereum blockchain.
- Azure Blockchain Dev Kit for Ethereum for VSCode - VSCode extension that allows for creating smart contracts and deploying them inside of Visual Studio Code

Test Blockchain Networks

- ethnode - Run an Ethereum node (Geth or Parity) for development, as easy as `npm i -g ethnode && ethnode`.
- Kaleido - Use Kaleido for spinning up a consortium blockchain network. Great for PoCs and testing
- Besu Private Network - Run a private network of Besu nodes in a Docker container
 - Orion - Component for performing private transactions by PegaSys
 - Artemis - Java implementation of the Ethereum 2.0 Beacon Chain by PegaSys
- Cliquebait - Simplifies integration and accepting testing of smart contract applications with docker instances that closely resembles a real blockchain network
- Local Raiden - Run a local Raiden network in docker containers for demo and testing purposes
- Private networks deployment scripts - Out-of-the-box deployment scripts for private PoA networks
- Local Ethereum Network - Out-of-the-box deployment scripts for private PoW networks
- Ethereum on Azure - Deployment and governance of consortium Ethereum PoA networks
- Ethereum on Google Cloud - Build Ethereum network based on Proof of Work
- Infura - Ethereum API access to Ethereum networks (Mainnet, Ropsten, Rinkeby, Goerli, Kovan)
- CloudFlare Distributed Web Gateway - Provides access to the Ethereum network through the Cloudflare instead of running your own node
- Chainstack - Shared and dedicated Ethereum nodes as a service (Mainnet, Ropsten)
- Alchemy - Blockchain Developer Platform, Ethereum API, and Node Service (Mainnet, Ropsten, Rinkeby, Goerli, Kovan)
- ZMOK - JSON-RPC Ethereum API (Mainnet, Rinkeby, Front-running Mainnet)
- Watchdata - Provide simple and reliable API access to Ethereum blockchain

Alternative Celo Faucet

- LearnWeb3 Faucet

Test Ether Faucets

- Rinkeby faucet
- Kovan faucet
- Ropsten faucet (MetaMask)
- Ropsten faucet (rpanic)
- Goerli faucet
- Universal faucet
- Netheruem.Faucet - A C#/.NET faucet

Communicating with Ethereum

Frontend Ethereum APIs

- thirdweb - Build web3 applications that can interact with your smart contracts using our powerful SDKs.
- Web3.js - Javascript Web3
- Eth.js - Javascript Web3 alternative
- Ethers.js - Javascript Web3 alternative, useful utilities and wallet features
- useDApp - React based framework for rapid DApp development on Ethereum
- light.js A high-level reactive JS library optimized for light clients.
- Web3Wrapper - Typescript Web3 alternative
- Ethereumjs - A collection of utility functions for Ethereum like ethereumjs-util and ethereumjs-tx
- Alchemy-web3.js - Javascript Web3 wrapper with automatic retries, access to Alchemy's enhanced APIs, and robust websocket connections.
- flex-contract and flex-ether - Modern, zero-configuration, high-level libraries for interacting with smart contracts and making transactions.
- ez-ens - Simple, zero-configuration Ethereum Name Service address resolver.
- web3x - A TypeScript port of web3.js. Benefits includes tiny builds and full type safety, including when interacting with contracts.
- Netheruem - Cross-platform Ethereum development framework
- dfuse - A TypeScript library to use dfuse Ethereum API
- Tasit SDK - A JavaScript SDK for making native mobile Ethereum dapps using React Native
- useMetamask - a custom React Hook to manage Metamask in Ethereum DApp projects
- WalletConnect - Open protocol for connecting Wallets to Dapps
- Subproviders - Several useful subproviders to use in conjunction with Web3-provider-engine (including a LedgerSubprovider for adding Ledger hardware wallet support to your dApp)
- Strictly Typed - Javascript alternatives
 - elm-ethereum
 - purescript-web3
- ChainAbstractionLayer - Communicate with different blockchains (including Ethereum) using a single interface.
- Delphereum - a Delphi interface to the Ethereum blockchain that allows for development of native dApps for Windows, macOS, iOS, and Android.
- Torus - Open-sourced SDK to build dapps with a seamless onboarding UX
- Fortmatic - A simple to use SDK to build web3 dApps without extensions or downloads.
- Portis - A non-custodial wallet with an SDK that enables easy interaction with DApps without installing anything.

- create-eth-app - Create Ethereum-powered front-end apps with one command.
- Scaffold-ETH - Beginner friendly forkable github for getting started building smart contracts.
- Notify.js - Deliver real-time notifications to your users. With built-in support for Speed-Ups and Cancels, Blocknative Notify.js helps users transact with confidence. Notify.js is easy to integrate and quick to customize.

Backend Ethereum APIs

- thirdweb - Build web3 applications that can interact with your smart contracts using our powerful SDKs.
- Web3.py - Python Web3
- Web3.php - PHP Web3
- Ethereum-php - PHP Web3
- Web3j - Java Web3
- Nethereum - .Net Web3
- Ethereum.rb - Ruby Web3
- rust-web3 - Rust Web3
- Web3.hs - Haskell Web3
- KEthereum - Kotlin Web3
- Eventeum - A bridge between Ethereum smart contract events and backend microservices, written in Java by Kauri
- Ethereumex - Elixir JSON-RPC client for the Ethereum blockchain
- Ethereum-jsonrpc-gateway - A gateway that allows you to run multiple Ethereum nodes for redundancy and load-balancing purposes. Can be run as an alternative to (or on top of) Infura. Written in Golang.
- EthContract - A set of helper methods to help query ETH smart contracts in Elixir
- Ethereum Contract Service - A MESSG Service to interact with any Ethereum contract based on its address and ABI.
- Ethereum Service - A MESSG Service to interact with events from Ethereum and interact with it.
- Marmo - Python, JS, and Java SDK for simplifying interactions with Ethereum. Uses relayers to offload transaction costs to relayers.
- Ethereum Logging Framework - provides advanced logging capabilities for Ethereum applications and networks including a query language, query processor, and logging code generation
- Watchdata - Provide simple and reliable API access to Ethereum blockchain

Bootstrap/Out-of-Box tools

- Create Eth App - Create Ethereum-powered frontend apps with one command
- Besu Private Network - Run a private network of Besu nodes in a Docker container
- Testchains - Pre-configured .NET devchains for fast response (PoA)
 - \ Blazor/Blockchain Explorer - Wasm blockchain explorer (functional sample)
- Local Raiden - Run a local Raiden network in docker containers for demo and testing purposes
- Private networks deployment scripts - Out-of-the-box deployment scripts for private PoA networks

- Parity Demo-PoA Tutorial - Step-by-Step tutorial for building a PoA test chain with 2 nodes with Parity authority round consensus
- Local Ethereum Network - Out-of-the-box deployment scripts for private PoW networks
- Kaleido - Use Kaleido for spinning up a consortium blockchain network. Great for PoCs and testing
- aragonCLI - aragonCLI is used to create and develop Aragon apps and organizations.
- ColonyJS - JavaScript client that provides an API for interacting with the Colony Network smart contracts.
- ArcJS - Library that facilitates javascript application access to the DAOstack Arc ethereum smart contracts.
- Arkane Connect - JavaScript client that provides an API for interacting with Arkane Network, a wallet provider for building user-friendly dapps.
- Onboard.js - Blocknative Onboard is the quick and easy way to add multi-wallet support to your project. With built-in modules for more than 20 unique hardware and software wallets, Onboard saves you time and headaches.
- web3-react - React framework for building single-page Ethereum dApps

Ethereum ABI (Application Binary Interface) tools

- Online ABI encoder - Free ABI encoder online service that allows you to encode your Solidity contract's functions and constructor arguments.
- ABI decoder - library for decoding data params and events from Ethereum transactions
- ABI-gen - Generate Typescript contract wrappers from contract ABI's.
- Ethereum ABI UI - Auto-generate UI form field definitions and associated validators from an Ethereum contract ABI
- headlong - type-safe Contract ABI and Recursive Length Prefix library in Java
- One Click dApp - Instantly create a dApp at a unique URL using the ABI.
- Ethereum Contract Service - A MESSG Service to interact with any Ethereum contract based on its address and ABI.
- Nethereum-CodeGenerator - A web based generator which creates a Nethereum based CInterface and Service based on Solidity Smart Contracts.
- EVMConnector - Create shareable contract dashboards and interact with arbitrary EVM-based blockchain functions, with or without an ABI.

Patterns & Best Practices

Patterns for Smart Contract Development

- Dappsys: Safe, simple, and flexible Ethereum contract building blocks
 - has solutions for common problems in Ethereum/Solidity, eg.
 - Whitelisting
 - Upgradable ERC20-Token
 - ERC20-Token-Vault
 - Authentication (RBAC)
 - ...several more...
 - provides building blocks for the MakerDAO or The TAO
 - should be consulted before creating own, untested, solutions
 - usage is described in Dapp-a-day 1-10 and Dapp-a-day 11-25

- OpenZeppelin Contracts: An open framework of reusable and secure smart contracts in the Solidity language.
 - Likely the most widely-used libraries and smart contracts
 - Blog about Best Practices with Security Audits
- Advanced Workshop with Assembly
- Simpler Ethereum Multisig - especially section Benefits
- CryptoFin Solidity Auditing Checklist - A checklist of common findings, and issues to watch out for when auditing a contract for a mainnet launch.
- aragonOS: A smart contract framework for building DAOs, Dapps and protocols
 - Upgradeability: Smart contracts can be upgraded to a newer version
 - Permission control: By using the auth and authP modifiers, you can protect functionality so only other apps or entities can access it
 - Forwarders: aragonOS apps can send their intent to perform an action to other apps, so that intent is forwarded if a set of requirements are met
- EIP-2535 Diamond Standard
 - Organize contracts so they share the same contract storage and Ethereum address.
 - Solves the 24KB max contract size limit.
 - Upgrade diamonds by adding/replacing/removing any number of functions in a single transaction.
 - Upgrades are transparent by recording them with a standard event.
 - Get information about a diamond with events and/or four standard functions.
- Clean Contracts - A guide to writing clean code

Upgradeability

- Blog von Elena Dimitrova, Dev at colony.io
 - <https://blog.colony.io/writing-more-robust-smart-contracts-99ad0a11e948>
 - <https://blog.colony.io/writing-upgradeable-contracts-in-solidity-6743f0eccc88>
- Aragon research blog
 - Library driven development
 - Advanced Solidity code deployment techniques
- OpenZeppelin on Proxy Libraries

Infrastructure

Ethereum Clients

- Besu - an open-source Ethereum client developed under the Apache 2.0 license and written in Java. The project is hosted by Hyperledger.
- Geth - Go client
- OpenEthereum - Rust client, formerly called Parity
- Aleth - C++ client
- Nethermind - .NET Core client
- Infura - A managed service providing Ethereum client standards-compliant APIs
- Trinity - Python client using py-evm
- Ethereumjs - JS client using ethereumjs-vm

- Seth - Seth is an Ethereum client tool-like a "MetaMask for the command line"
- Mustekala - Ethereum Light Client project of Metamask
- Exthereum - Elixir client
- EWF Parity - Energy Web Foundation client for the Tobalaba test network
- Quorum - A permissioned implementation of Ethereum supporting data privacy by JP Morgan
- Mana - Ethereum full node implementation written in Elixir.
- Chainstack - A managed service providing shared and dedicated Geth nodes
- QuickNode - Blockchain developer cloud with API access and node-as-a-service.
- Watchdata - Provide simple and reliable API access to Ethereum blockchain

Storage

- IPFS - Decentralised storage and file referencing
 - Mahuta - IPFS Storage service with added search capability, formerly IPFS-Store
 - OrbitDB - Decentralised database on top of IPFS
 - JS IPFS API - A client library for the IPFS HTTP API, implemented in JavaScript
 - TEMPORAL - Easy to use API into IPFS and other distributed/decentralised storage protocols
 - PINATA - The Easiest Way to Use IPFS
- Swarm - Distributed storage platform and content distribution service, a native base layer service of the Ethereum web3 stack
- Infura - A managed IPFS API Gateway and pinning service
- 3Box Storage - An api for user controlled, distrubuted storage. Built on top of IPFS and Orbitdb.
- Aleph.im - an offchain incentivized peer-to-peer cloud project (database, file storage, computing and DID) compatible with Ethereum and IPFS.

Messaging

- Whisper - Communication protocol for DApps to communicate with each other, a native base layer service of the Ethereum web3 stack
- DEVp2p Wire Protocol - Peer-to-peer communications between nodes running Ethereum/Whisper
- Pydevp2p - Python implementation of the RLPx network layer
- 3Box Threads - API to allow developers to implement IPFS persisted, or in memory peer to peer messaging.

Testing Tools

- Solidity code coverage - Solidity code coverage tool
- Solidity coverage - Alternative code coverage for Solidity smart-contracts
- Solidity function profiler - Solidity contract function profiler
- Sol-profiler - Alternative and updated Solidity smart contract profiler
- Espresso - Speedy, parallelised, hot-reloading solidity test framework
- Eth tester - Tool suite for testing Ethereum applications

- Cliquebait - Simplifies integration and accepting testing of smart contract applications with docker instances that closely resembles a real blockchain network
- Hevm - The hevm project is an implementation of the Ethereum virtual machine (EVM) made specifically for unit testing and debugging smart contracts
- Ethereum graph debugger - Solidity graphical debugger
- Tenderly CLI - Speed up your development with human readable stack traces
- Solhint - Solidity linter that provides security, style guide and best practice rules for smart contract validation
- Ethlint - Linter to identify and fix style & security issues in Solidity, formerly Solium
- Decode - npm package which parses tx's submitted to a local testrpc node to make them more readable and easier to understand
- Psol - Solidity lexical preprocessor with mustache.js-style syntax, macros, conditional compilation and automatic remote dependency inclusion.
- solpp - Solidity preprocessor and flattener with a comprehensive directive and expression language, high precision math, and many useful helper functions.
- Decode and Publish - Decode and publish raw ethereum tx. Similar to <https://live.blockcypher.com/btc-testnet/decodetx/>
- Doppelgänger - a library for mocking smart contract dependencies during unit testing.
- rocketh - A simple lib to test ethereum smart contract that allow to use whatever web3 lib and test runner you choose.
- pytest-cobra - PyTest plugin for testing smart contracts for Ethereum blockchain.

Security Tools

- MythX - Security verification platform and tools ecosystem for Ethereum developers
- Mythril - Open-source EVM bytecode security analysis tool
- Oyente - Alternative static smart contract security analysis
- Securify - Security scanner for Ethereum smart contracts
- SmartCheck - Static smart contract security analyzer
- Ethersplay - EVM disassembler
- Evmdis - Alternative EVM disassembler
- Hydra - Framework for cryptoeconomic contract security, decentralised security bounties
- Solgraph - Visualise Solidity control flow for smart contract security analysis
- Manticore - Symbolic execution tool on Smart Contracts and Binaries
- Slither - A Solidity static analysis framework
- Adelaide - The SECBIT static analysis extension to Solidity compiler
- solc-verify - A modular verifier for Solidity smart contracts
- Solidity security blog - Comprehensive list of known attack vectors and common anti-patterns
- Awesome Buggy ERC20 Tokens - A Collection of Vulnerabilities in ERC20 Smart Contracts With Tokens Affected
- Free Smart Contract Security Audit - Free smart contract security audits from Callisto Network

- Piet - A visual Solidity architecture analyzer

Monitoring

- Alethio - An advanced Ethereum analytics platform that provides live monitoring, insights and anomaly detection, token metrics, smart contract audits, graph visualization and blockchain search. Real-time market information and trading activities across Ethereum's decentralized exchanges can also be explored.
- amberdata.io - Provides live monitoring, insights and anomaly detection, token metrics, smart contract audits, graph visualization and blockchain search.
- Neufund - Smart Contract Watch - A tool to monitor a number of smart contracts and transactions
- Scout - A live data feed of the activities and event logs of your smart contracts on Ethereum
- Tenderly - A platform that gives users reliable smart contract monitoring and alerting in the form of a web dashboard without requiring users to host or maintain infrastructure
- Chainlyt - Explore smart contracts with decoded transaction data, see how the contract is used and search transactions with specific function calls
- BlockScout - A tool for inspecting and analyzing EVM based blockchains. The only full featured blockchain explorer for Ethereum networks.
- Terminal - A control panel for monitoring dapps. Terminal can be used to monitor your users, dapp, blockchain infrastructure, transactions and more.
- Ethereum-watcher - An extensible framework written in Golang for listening to on-chain events and doing something in response.
- Alchemy Notify - Notifications for mined and dropped transactions, gas price changes, and address activity for desired addresses.
- Blocknative Mempool Explorer - Monitor any contract or wallet address and get streaming mempool events for every lifecycle stage - including drops, confirms, speedups, cancels, and more. Automatically decode confirmed internal transactions. And filter exactly how you want. Recieve events in our visual, no-code, interface or associate them with your API key to get events via a webhook. Mempool Explorer helps exchanges, protocols, wallets, and traders monitor and act on transactions in real-time.
- Eternal - Ethereum block explorer for private chain. Browse transactions, decode function calls, event data or contract variables values on your locally running chain.

Other Miscellaneous Tools

- aragonPM - a decentralized package manager powered by aragonOS and Ethereum. aragonPM enables decentralized governance over package upgrades, removing centralized points of failure.
- Solc - Solidity compiler
- Sol-compiler - Project-level Solidity compiler
- Solidity cli - Compile solidity-code faster, easier and more reliable
- Solidity flattener - Combine solidity project to flat file utility. Useful for visualizing imported contracts or for verifying your contract on Etherscan

- Sol-merger - Alternative, merges all imports into single file for solidity contracts
- RLP - Recursive Length Prefix Encoding in JavaScript
- eth-cli - A collection of CLI tools to help with ethereum learning and development
- Ethereum - Ethereum is a command line tool for managing common tasks in Ethereum
- Eth crypto - Cryptographic javascript-functions for Ethereum and tutorials to use them with web3js and solidity
- Parity Signer - mobile app allows signing transactions
- py-eth - Collection of Python tools for the Ethereum ecosystem
- Decode - npm package which parses tx's submitted to a local testrpc node to make them more readable and easier to understand
- TypeChain - Typescript bindings for Ethereum smartcontracts
- EthSum - A Simple Ethereum Address Checksum Tool
- PHP based Blockchain indexer - allows indexing blocks or listening to Events in PHP
- Purser - JavaScript universal wallet tool for Ethereum-based wallets. Supports software, hardware, and Metamask -- brings all wallets into a consistent and predictable interface for dApp development.
- Node-Metamask - Connect to MetaMask from node.js
- Solidity-docgen - Documentation generator for Solidity projects
- Ethereum ETL - Export Ethereum blockchain data to CSV or JSON files
- prettier-plugin-solidity - Prettier plugin for formatting Solidity code
- Unity3dSimpleSample - Ethereum and Unity integration demo
- Flappy - Ethereum and Unity integration demo/sample
- Wonka - Nethereum business rules engine demo/sample
- Resolver-Engine - A set of tools to standarize Solidity import and artifact resolution in frameworks.
- eth-reveal - A node and browser tool to inspect transactions - decoding where possible the method, event logs and any revert reasons using ABIs found online.
- Ethereum-tx-sender - A useful library written in Golang to reliably send a transaction - abstracting away some of the tricky low level details such as gas optimization, nonce calculations, synchronization, and retries.
- Blocknative Gas Platform - Gas estimation for builders, by builders. Gas Platform harnesses Blocknative's real-time mempool data infrastructure to accurately and consistently estimate Ethereum transaction fees. This provides builders and traders with an up-to-the-moment gas fee API.
- ETH Gas.watch - A gas price watcher with email notifications on price change

Smart Contract Standards & Libraries

ERCs - The Ethereum Request for Comment repository

- Tokens
 - ERC-20 - Original token contract for fungible assets
 - ERC-721 - Token standard for non-fungible assets
 - ERC-777 - An improved token standard for fungible assets
 - ERC-918 - Mineable Token Standard

- ERC-165 - Creates a standard method to publish and detect what interfaces a smart contract implements.
- ERC-725 - Proxy contract for key management and execution, to establish a Blockchain identity.
- ERC-173 - A standard interface for ownership of contracts

Popular Smart Contract Libraries

- Zeppelin - Contains tested reusable smart contracts like SafeMath and OpenZeppelin SDK library for smart contract upgradeability
- cryptofin-solidity - A collection of Solidity libraries for building secure and gas-efficient smart contracts on Ethereum.
- Modular Libraries - A group of packages built for use on blockchains utilising the Ethereum Virtual Machine
- DateTime Library - A gas-efficient Solidity date and time library
- Aragon - DAO protocol. Contains aragonOS smart contract framework with focus on upgradeability and governance
- ARC - an operating system for DAOs and the base layer of the DAO stack.
- 0x - DEX protocol
- Token Libraries with Proofs - Contains correctness proofs of token contracts wrt. given specifications and high-level properties
- Provable API - Provides contracts for using the Provable service, allowing for off-chain actions, data-fetching, and computation
- ABDK Libraries for Solidity - Fixed-point (64.64 bit) and IEEE-754 compliant quad precision (128 bit) floating-point math libraries for Solidity

Developer Guides for 2nd Layer Infrastructure

Scalability

Payment/State Channels

- Ethereum Payment Channel - Ethereum Payment Channel in 50 lines of code
- µRaiden Documentation - Guides and Samples for µRaiden Sender/Receiver Use Cases

Plasma

- Learn Plasma - Website asNode application that was started at the 2018 IC3-Ethereum Crypto Boot Camp at Cornell University, covering all Plasma variants (MVP/Cash/Debit)
- Plasma MVP - OmiseGO's research implementation of Minimal Viable Plasma
- Plasma MVP Golang - Golang implementation and extension of the Minimum Viable Plasma specification
- Plasma Guard - Automatically watch and challenge or exit from OmiseGO Plasma Network when needed.
- Plasma OmiseGo Watcher - Interact with Plasma OmiseGo network and notifies for any byzantine events.

Side-Chains

- POA Network
 - POA Bridge

- POA Bridge UI
- POA Bridge Contracts
- Loom Network
- Matic Network

Privacy / Confidentiality

ZK-SNARKs

- ZoKrates - A toolbox for zkSNARKS on Ethereum
- The AZTEC Protocol - Confidential transactions on the Ethereum network, implementation is live on the Ethereum main-net
- Nightfall - Make any ERC-20 / ERC-721 token private - open source tools & microservices
- Proxy Re-encryption (PRE)
NuCypher Network - A proxy re-encryption network to empower data privacy in decentralized systems
- pyUmbral - Threshold proxy re-encryption cryptographic library
- Fully Homomorphic Encryption (FHE)
\\ NuFHE - GPU accelerated FHE library

Scalability + Privacy

ZK-STARKs

- StarkWare and StarkWare Resources - StarkEx scalability engine storing state transitions on-chain

Prebuilt UI Components

- aragonUI - A React library including Dapp components
- components.bounties.network - A React library including Dapp components
- ui.decentraland.org - A React library including Dapp components
- dapparatus - Reusable React Dapp components
- Metamask ui - Metamask React Components
- DappHybrid - A cross-platform hybrid hosting mechanism for web-based decentralized applications
- Nethereum.UI.Desktop - Cross-platform desktop wallet sample
- eth-button - Minimalist donation button
- Rumble Design System - Adaptable components and design standards for decentralized applications.
- 3Box Plugins - Drop in react components for social functionality. Including comments, profiles, and messaging.

:::info

Inspired by: <https://github.com/ConsenSys/ethereum-developer-tools-list>

:::

fee-currency.md:

title: Implementing Fee Abstraction in Wallets
description: How to allow your wallet users to pay for gas fee using
alternate fee currencies

Celo allows paying gas fees in currency other than the native currency. The tokens that can be used to pay gas fees is controlled via governance and the list of tokens allowed is maintained in FeeCurrencyWhitelist.sol.

Alternate fee currency works with EOAs and no paymaster is required!

This works by specifying a token/adaptor address as a value for the feeCurrency property in the transaction object. The feeCurrency property in the transaction object is exclusive to Celo and allows paying gas fees using assets other than the native currency of the network.

The below steps describe how wallets can implement the alternate fee currency feature in order to allow users to use alternate assets in the user's wallet to pay for gas fees.

Enabling Transactions with ERC20 Token as fee currency

We recommend using the viem library as it has support for the feeCurrency field in the transaction required for sending transaction where the gas fees will be paid in ERC20 tokens.

Estimating gas price

To estimate gas price use the token address (in case of cUSD, cEUR and cREAL) or the adapter address (in case of USDC and USDT) as the value for feeCurrency field in the transaction.

Estimating gas price is important because if the user is trying to transfer the entire balance in an asset and using the same asset to pay for gas fees, the user shouldn't be able to transfer the entire amount as a small portion will be utilized to pay for gas fees.

Example: If the user has 10 USDC and is trying to transfer the entire 10 USDC and chooses to use USDC as the currency to pay for gas, the user shouldn't be allowed to transfer the entire 10 USDC as a small portion has to be used for gas fees.

The following code snippet calculates the transaction fee (in USDC) for a USDC transfer transaction.

Feel free to modify to support the currency of your choice.

```
js
import { createPublicClient, hexToBigInt, http } from "viem";
import { celo } from "viem/chains";

// USDC is 6 decimals and hence requires the adapter address instead of
the token address
const USDCADAPTERMAINNET = "0x2F25deB3848C207fc8E0c34035B3Ba7fc157602B";
```

[illegible]

Sending transaction using Fee Abstraction

Sending transaction with fee currency other than the native currency of the network is pretty straightforward all you need to do is set the `feeCurrency` property in the transaction object with the address of the token/adaptor you want to use to pay for gas fees.

The below code snippets demonstrates transferring 1 USDC using USDC as gas currency.

```
js
import { createWalletClient, http } from "viem";
import { celo } from "viem/chains";
import { privateKeyToAccount } from "viem/accounts";
import { stableTokenAbi } from "@celo/abis";

// Creating account from private key, you can choose to do it any other
// way.
const account = privateKeyToAccount("0x432c...");

// WalletClient can perform transactions.
const client = createWalletClient({
  account,

  // Passing chain is how viem knows to try serializing tx as cip42.
  chain: celo,
  transport: http(),
});

const USDCADAPTERMAINNET = "0x2F25deB3848C207fc8E0c34035B3Ba7fC157602B";
const USDCMAINNET = "0xcebA9300f2b948710d2653dD7B07f33A8B32118C";

/
  The UI of the wallet should calculate the transaction fees, show it and
  consider the amount to not be part of the asset that the user i.e the
  amount corresponding to transaction fees should not be transferrable.
/
async function calculateTransactionFeesInUSDC(transaction) {
  // Implementation of getGasPriceInUSDC is in the above code snippet
  const gasPriceInUSDC = await getGasPriceInUSDC();

  // Implementation of estimateGasInUSDC is in the above code snippet
  const estimatedGasPrice = await estimateGasInUSDC(transaction);

  return gasPriceInUSDC * estimatedGasPrice;
}

async function send(amountInWei) {
  const to = USDCMAINNET;

  // Data to perform an ERC20 transfer
  const data = encodeFunctionData({
    abi: stableTokenAbi,
    functionName: "transfer",
```

```

    args: [
      "0xccc9576F841de93Cd32bEe7B98fE8B9BD3070e3D",
      // Different tokens can have different decimals, cUSD (18), USDC
(6)      amountInWei,
    ],
  });

  const transactionFee = await calculateTransactionFeesInUSDC({ to, data
});

  const tokenReceivedbyReceiver = parseEther("1") - transactionFee;

  /
  Now the data has to be encode again but with different transfer value
  because the receiver receives the amount minus the transaction fee.
  /
  const dataAfterFeeCalculation = encodeFunctionData({
    abi: stableTokenAbi,
    functionName: "transfer",
    args: [
      "0xccc9576F841de93Cd32bEe7B98fE8B9BD3070e3D",
      // Different tokens can have different decimals, cUSD (18), USDC
(6)      tokenReceivedbyReceiver,
    ],
  });

  // Transaction hash
  const hash = await client.sendTransaction({
    ...{ to, data: dataAfterFeeCalculation },

  /
  In case the transaction request does not include the feeCurrency
  property, the wallet can add it or change it to a different currency
  based on the user balance.

  Notice that we use the USDCADAPTERMAINNET address not the token
  address this is because at the protocol level only 18 decimals tokens are
  supported, but USDC is 6 decimals, the adapter acts a unit converter.
  /
  feeCurrency: USDCADAPTERMAINNET,
  });

  return hash;
}

```

If you have any questions, please reach out [here](#).

index.md:

title: Celo Developers Overview
description: There are 6 Billion smartphones on Earth. Build for all of them.

```
import PageRef from '@components/PageRef'  
import Tabs from '@theme/Tabs';  
import TabItem from '@theme/TabItem';
```

Developer Tools and Resources

Explore our comprehensive suite of tools, guides, and resources designed to help you build, test, and deploy on Celo.

:::warning
Celo is currently transitioning from a standalone Layer 1 blockchain to an Ethereum Layer 2. As a result, certain information about developer resources may be outdated.

For the latest information, please refer to our Celo L2 documentation.
:::

Quickstart

Celo Composer allows you to quickly build, deploy, and iterate on decentralized applications using Celo. It provides a number of frameworks, examples, and Celo specific functionality to help you get started with your next dApp.

- Quickstart with Celo Composer
- Celo Composer GitHub
- Community Resources

Developer Tools

- Nodes
- Block Explorer
- Testnet Faucet
- EVM Compatible Tooling

Developer Environments

- Using thirdweb
- Using Remix
- Using Hardhat

Code Examples

- Developer Tutorials
- Dacade

launch-checklist.md:

title: Launch Checklist

description: A comprehensive guide to assist you in launching dapps on Celo.

Launch Checklist

A comprehensive guide to assist you in launching dapps on Celo.

Integration

- [] Wallet Connect: Essential for Valora integration.
- [] Security Audit: Have you finished this? Remember to publish the results and the auditor's details on your website.
- [] Due Diligence: Ensure you share the following forms:
 - [] Entity Form
 - [] Individual intake
 - [] Basic Intake Request

Reporting

Make your dapp stand out by reporting it on these platforms:

- [] Dapp Radar
- [] DeFi Llama (for DeFi & NFT Projects)
- [] Electric Capital Reporting
- [] Celo Dapp showcase
- [] Celo Scan Contract Verification
- [] Celo Explorer Contract Verification
- [] Nansen Contract / Project Verification
- [] Dune

Website & Documentation

- [] Get your app featured on docs.celo.org

If your dapp has smart contracts

- [] Add a tutorial about how to use the contract/token, etc.
- [] Best practices include: using Open Zeppelin contracts and not writing everything from scratch as this could open your contracts to vulnerabilities.
- [] Consider your options for a contract audit.

If your dapp doesn't have smart contracts

- [] Discuss potential extensions or enhancements.

Training

Provide resources for users:

- [] Guide on getting started with your dapp.
- [] Instructions on how to on/off ramp for a seamless dapp experience.

Marketing

- [] Complete the Marketing Intake Form and share your marketing strategies.
- [] Join the Celo Founders Telegram group.
- [] Remember to tag @celoorg in any launch announcements.

Legal

- [] Ensure you have public-facing Terms & Conditions.
- [] Make sure you have a public-facing GDPR privacy policy or meet other privacy requirements.

tools.md:

title: Celo Developer Tools
description: Overview of Celo tools and the value they provide to developers.

Developer Tools

Overview of Celo tools and the value they provide to developers.

```
import PageRef from '@components/PageRef'  
import Tabs from '@theme/Tabs';  
import TabItem from '@theme/TabItem';
```

:::tip

Consider using Dependabot to help keep project dependencies up to date. The following developer tools are being actively developed and keeping your dependencies up-to-date will help your projects stay functional and secure.

:::

SDKs

ContractKit

ContractKit is a library to help developers and validators to interact with the Celo blockchain and is well suited to developers looking for an easy way to integrate Celo Smart Contracts within their applications.

`<PageRef url="/developer-guide/contractkit" pageName="ContractKit" />`

Ethers.JS Wrapper

A minimal wrapper to make Ethers.JS compatible with the Celo network.

`<PageRef url="/developer/celo-ethers-wrapper" pageName="Celo Ethers Wrapper" />`

WalletConnect

WalletConnect is a standard across EVM compatible blockchains to connect wallets to dapps. It allows developers to build connections between wallets and dapps on the same desktop or mobile device, or between desktop dapps and mobile wallets.

`<PageRef url="walletconnect" pageName="WalletConnect" />`

Celo CLI

The Command Line Interface allows users to interact with the Celo Protocol smart contracts.

`<PageRef url="/cli" pageName="Celo CLI" />`

Celo SDK Reference Docs

You can find the reference documentation for all of the Celo SDK packages found in the celo monorepo here. The SDK packages consist of:

- Base
- Connect
- ContractKit
- Explorer
- Governance
- Identity
- Keystores
- Network Utils
- Transactions Uri
- Utils
- Wallet-base
- Wallet-HSM
- Wallet-HSM-AWS
- Wallet-HSM-Azure
- Wallet-ledger
- Wallet-local
- Wallet-remote
- Wallet-rpc
- Wallet-walletconnect

`<PageRef url="https://docs.celo.org/learn/developer-tools#celo-sdk-reference-docs" pageName="Celo SDK Reference Docs" />`

Infrastructure

BlockScout

BlockScout is a cLabs hosted GUI block explorer and API endpoints. It allows you to look up information about the Celo blockchain including average block time, total transactions, and additional transaction details. You may also view details of your own custom smart contracts or existing DeFi contracts to view how value is moving between accounts and on-chain network events.

```
<PageRef url="https://explorer.celo.org/" pageName="BlockScout" />
```

ODIS

ODIS (Oblivious decentralized identity service) is a lightweight identity layer that makes it easy to send cryptocurrency to a phone number

```
<PageRef url="/protocol/identity/odis" pageName="ODIS" />
```

The Graph

The Graph is an indexing protocol for querying networks like Celo, Ethereum and IPFS. Anyone can build and publish open APIs, called subgraphs, making data easily accessible. Many blockchain data querying tools like Dapplooker leverage the Graph.

```
<PageRef url="/https://thegraph.com/" pageName="The Graph" />
```

DappLooker

DappLooker allows you to easily analyze & visualize your Celo smart contracts & subgraph data in various formats and gather it into dashboards from multiple sources. Analyze your data with intuitive Visual SQL which writes queries for you. Share the story your data tells with your team or with your community. Share dashboard insights via URL wherever you need to make your organization truly data driven.

```
<PageRef url="https://dapplooker.com/integration/celo"
pageName="DappLooker" />
```

SQD

SQD is a decentralized hyper-scalable data platform optimized for providing efficient, permissionless access to large volumes of data. It currently serves historical on-chain data, including event logs, transaction receipts, traces, and per-transaction state diffs. SQD offers a powerful toolkit for creating custom data extraction and processing pipelines, achieving an indexing speed of up to 150k blocks per second. Use SQD to retrieve data from both the Celo Mainnet Celo Alfajores Testnet.

```
<PageRef url="https://sqd.dev/" pageName="SQD" />
```

Stats.celo.org

Stats.celo.org allows you to check network activity and health.

```
<PageRef url="http://stats.celo.org" pageName="stats.celo.org" />
```

SubQuery

SubQuery is a leading blockchain data indexer that provides developers with fast, flexible, universal, open source and decentralised APIs for CELO. One of SubQuery's competitive advantages is the ability to aggregate data not only within a chain but across multiple blockchains all within a single project.

```
<PageRef url="https://subquery.network/" pageName="SubQuery" />
```

Hosted Nodes

RPC Endpoint Services

Forno

Forno is a cLabs hosted node service for interacting with the Celo network. This allows you to connect to the Celo Blockchain without having to run your own node.

Forno has HTTP and WebSocket endpoints that you can use to query current Celo data or post transactions that you would like to broadcast to the network. The service runs full nodes in non-archive mode, so you can query the current state of the blockchain, but cannot access the historic state.

Forno can be used as an HTTP Provider with ContractKit.

```
<PageRef url="/network/node/forno" pageName="Forno" />
```

Ankr

Featuring open access to a Public RPC API layer, Ankr Protocol provides reliable, load balanced access to node clusters from anywhere in the world.

```
<PageRef url="https://www.ankr.com/rpc/celo/" pageName="Ankr" />
```

Quicknode

Quicknode is an enterprise grade node service with a dashboard, metrics, security controls, customer support and no rate limits (pay-as-you-go).

```
<PageRef url="https://www.quicknode.com/chains/celo" pageName="Quicknode" />
```

All That Node

All That Node supports public and private RPC nodes for mainnet, alfajores and baklava networks. They offer free private RPC nodes up to 10,000 requests/day and you can upgrade your plan as needed. You can also claim alfajores funds from the faucet in the site without signing up or any time-consuming auth.

```
<PageRef url="https://www.allthatnode.com/celo.dsrv" pageName="All That Node" />
```

Celo Wallets

Celo Wallets are tools that create accounts, manage keys, and help users transact on the Celo network.

```
<PageRef url="/wallet/" pageName="Celo Wallets" />
```

```
# contracts-wrappers-registry.md:
```

```
---
title: Celo Core Contracts (Wrapper/Registry)
description: How to interact with CELO assets using the wrapper and registry Celo Core Contracts.
---
```

Core Contracts (Wrapper/Registry)

How to interact with CELO assets using the wrapper and registry Celo Core Contracts.

```
---
```

Interacting with CELO & cUSD

celo-blockchain has two initial coins: CELO and cUSD (Mento stabletoken). Both implement the ERC20 standard, and to interact with them is as simple as:

```
ts
const goldtoken = await kit.contracts.getGoldToken();

const balance = await goldtoken.balanceOf(someAddress);
```

To send funds:

```
ts
const oneGold = kit.web3.utils.toWei("1", "ether");
const tx = await goldtoken.transfer(someAddress, oneGold).send({
  from: myAddress,
});

const hash = await tx.getHash();
```

```
const receipt = await tx.waitReceipt();
```

To interact with cUSD, is the same but with a different contract:

```
ts
const stabletoken = await kit.contracts.getStableToken();
```

Interacting with Other Celo Contracts

Apart from GoldToken and Mento stabletokens, there are many core contracts.

For the moment, we have contract wrappers for:

- Accounts
- Attestations
- BlockchainParameters
- DoubleSigningSlasher
- DowntimeSlasher
- Election
- Escrow
- Exchange (Uniswap kind exchange between Gold and Stable tokens)
- GasPriceMinimum
- GoldToken
- Governance
- LockedGold
- Reserve
- SortedOracles
- Validators
- StableToken

A Note About Contract Addresses

Celo Core Contracts addresses, can be obtained by looking at the Registry contract.

That's actually how kit obtain them.

We expose the registry api, which can be accessed by:

```
ts
const goldTokenAddress = await
kit.registry.addressFor(CeloContract.GoldToken);
```

Accessing web3 contract wrappers

Some user might want to access web3 native contract wrappers. We encourage to use the Celo contracts instead to avoid mistakes.

To do so, you can:

```
ts
```

```
const web3Exchange = await kit.web3Contracts.getExchange();
```

We expose native wrappers for all Web3 contracts.

The complete list is:

- Accounts
- Attestations
- BlockchainParameters
- DoubleSigningSlasher
- DowntimeSlasher
- Election
- EpochRewards
- Escrow
- Exchange
- FeeCurrencyWhiteList
- GasPriceMinimum
- GoldToken
- Governance
- LockedGold
- Random
- Registry
- Reserve
- SortedOracles
- StableToken
- Validators

Debugging

If you need to debug kit, we use the well known debug node library.

So set the environment variable DEBUG as:

```
bash
DEBUG="kit:,
```

```
# data-encryption-key.md:
```

Data Encryption Key

An account may register a data encryption key (DEK) that can be used for lightweight signing or encryption operations. Some examples of DEK usage are:

1. Supporting private payment comments between two users
2. Signing authentication headers to the Oblivious Decentralized Identifier Service
3. Sharing profile picture and name privately between two users

Most Valora users automatically register a DEK with their wallet when they go through the onboarding flow. The DEK can be set during account creation or registered after as follows:

```
ts
const accountWrapper: AccountsWrapper =
  await contractKit.contracts.getAccounts();
const setKeyTx =
  accountWrapper.setAccountDataEncryptionKey(dekPublicKey);
```

When using the DEK, it's important to check that the DEK is the latest that's registered for a user. This can be done by querying the account contract and comparing the resulting public key with the key that's expected.

```
ts
// Query the on-chain data encryption key for a user
const accountWrapper: AccountsWrapper = await
contractKit.contracts.getAccounts()
const dataEncryptionKey = await
accountWrapper.getDataEncryptionKey(address)
// Check that this matches with the public key
...
```

index.md:

title: Celo ContractKit
description: Overview of ContractKit, its features, purpose, and
resources to help you get started.

import PageRef from '@components/PageRef'

ContractKit

Overview of ContractKit, its features, purpose, and resources to help you
get started.

What is ContractKit?

ContractKit is a library to help developers and validators to interact
with the Celo blockchain and is well suited to developers looking for an
easy way to integrate Celo Smart Contracts within their applications.

Contractkit includes common functionality to make it easier to get
started building.

What you can do?

ContractKit supports the following functionality:

- Connect to a node
- Access Web3 object to interact with node's JSON RPC API
- Send Transaction with Celo's feeCurrency field
- Simple interface to interact with CELO and cUSD
- Simple interface to interact with Celo Core contracts
- Local sign transactions
- Utilities
- Query on-chain identifier for a phone number

Use ContractKit

- Setup
- Using the kit
- SDK Reference Docs

migrating-to-contractkit-v1.md:

title: Migrating to ContractKit v1.0

description: How to migrate to the newest version of ContractKit and make use of its latest features.

Migrating to ContractKit v1.0

How to migrate to from prerelease of ContractKit to v1 and make use of its features.

What is ContractKit version v1.0?

cLabs recently released ContractKit version 1.0.0. In it, the original ContractKit package has been split into several separate packages that all make up the Celo SDK. This document explains the key differences and shows you how you can start using the updated SDK.

:::warning

If you are using a previous version of ContractKit (anything below 1.0.0), you can continue using that version and you will only need to make the following changes when you upgrade.

:::

The main benefit of using the new version include:

- Reduced bundle size
- Better Typescript support
- Improved maintenance by making it easier to use other libraries

ContractKit packages

ContractKit is now a suite of packages.

Main packages

- Connect handles how we communicate to the our chain nodes. It wraps the web3 library and has its own rpcCaller class, to make custom calls to the node. It's the layer in charge of knowing how and which parameters are added by Celo, connect to the node, build the message, send it and handle those responses.
- ContractKit is a reduced subset of the previous versions of ContractKit. This is the layer in charge of loading and using our core contracts. Internally, uses the connect package described above. It has our contracts generated from the ABIs, their wrappers, and also the logic to make claims.

Complementary Packages

- Explorer depends on contractkit and connect. It provides some utility functions that make it easy to listen for new block and log information.
- Governance depends on contractkit and explorer. It provides functions to read and interact with Celo Governance Proposals (CGPs).
- Identity simplifies interacting with ODIS, Celo's lightweight identity layer based on phone numbers.
- Network-utils provides utilities for getting genesis block and static node information.
- Transactions-uri makes it easy to generate Celo transaction URIs and QR codes.

Wallets and Wallet Utility packages

- Wallet-hsm-azure is a Azure Key Vault implementation of a RemoteWallet.
- Wallet-hsm-aws allows you to easily interact with a cloud HSM wallet built on AWS KMS.
- Wallet-ledger provides utilities for interacting with a Ledger hardware wallet.
- Wallet-local provides utilities for locally managing wallet by importing a private key string.
- Wallet-rpc provides utilities for performing wallet functions via RPC.
- Wallet-base provides base utilities for creating Celo wallets.
- Wallet-hsm provides signature utilities for using HSMs.
- Wallet-remote provides utilities for interacting with remote wallets. This is useful for interacting with wallets on secure remote servers.

Importing packages

Importing the packages is slightly different now that many packages are separate from the main ContractKit package. You will have to explicitly import these packages instead of just importing all of them with ContractKit.

For example:

```
javascript
// Previously this would work to import the block-explorer
import { newBlockExplorer } from "@celo/contractkit/lib/explorer/block-explorer";

// With ContractKit v1.x.y, import the block-explorer explicitly
import { newBlockExplorer } from "@celo/explorer/lib/block-explorer";
```

Connecting to the network

Older versions of ContractKit:

```
javascript
// version ^0.4.0
const ContractKit = require("@celo/contractkit");

// Older versions of ContractKit create a new Web3 instance internally
const kit = ContractKit.newKit("https://forno.celo.org");
```

Version 1.x.y

```
javascript
// Since ContractKit no longer instantiates web3, you'll need to
explicitly require it
const Web3 = require("web3");
const web3 = new Web3("https://forno.celo.org");

// Require ContractKit and newKitFromWeb3
const { ContractKit, newKitFromWeb3 } = require("@celo/contractkit");
let contractKit = newKitFromWeb3(web3);
```

Accessing Web3 functions

You can access web3 functions through the connection module.

```
javascript
// version ^0.4.0
let amount = kit.web3.utils.fromWei("1000000", "ether");

// version 1.x.y
let amount = kit.connection.web3.utils.fromWei("1000000", "ether");
```

Backward Compatibility

These ContractKit functions will still work when accessed directly from kit, but it is advised to consume it via connection to avoid future deprecation.

```
// This still works
kit.addAccount

// recommended:
kit.connection.addAccount
```

Connection package

The connection package update includes implementations of some common web3 functions. Here are a few examples:

```
- kit.web3.eth.isSyncing --> kit.connection.isSyncing
- kit.web3.eth.getBlock --> kit.connection.getBlock
- kit.web3.eth.getBlockNumber --> kit.connection.getBlockNumber
- kit.web3.eth.sign --> kit.connection.sign
- kit.isListening --> kit.connection.isListening
- kit.addAccount --> kit.connection.addAccount
```

migrating-to-contractkit-v2.md:

```
---
title: Migrating to ContractKit v2.0
description: How to migrate from v1 to v2 ContractKit suite of packages
and make use of their latest features.
---
```

How to migrate from v1 to v2 of the Celo SDK suite of packages and make use of their latest features.

Why v2?

Bundle Size

The primary motivation in creating v2 was reduced bundlesize and increased modularity. The massive package size for @celo/contractkit has been an elephant in the room and source of dissonance for looking to build mobile first dApps. As of 1.5.2 bundlephobia list the minified size at 3.7MB. 2.0.0 comes in at 1.7MB. still big yet we have a few more tricks. First the packages have been all marked as sideEffects:false, a kit instance is no longer required to any classes in the contractkit package, and the introduction of MiniContractKit.

Modularity

```
sideEffects:false
```

Tells your bundler it can safely only include the code that is explicitly used, reducing bundlesize.

kit no longer needed by everything

In v1 Almost everything required a kit instance to be passed to its constructor. Effectively this meant it was impossible to use any of the classes in @celo/contractkit alone.

In v2 AddressRegistry, Wrappers, WrapperCache, and more can all be constructed using mostly just a Connection (sometimes other arguments too).

MiniContractKit

The prize of no longer needing a full kit is that it became possible to create a slimmed down minimal viable ContractKit.

MiniContractKit provides a subset of ContractKit features with the same interface. For many dapps it will be a drop in opt-in change (eg import {newKit, ContractKit} from "@celo/contractkit/lib/mini/kit"). It reduces size by only including access to Accounts, StableToken, Exchange and GoldToken wrappers and contracts. It can setFeeCurrency, look up info about the current account and, like full Contractkit, it delegates most functionality to connection.

Get Started

Upgrade your project packages to the latest (in this case release beta, but anything over 2 will work when it's out of beta). For example, with ContractKit and Celo utils:

```
yarn add @celo/contractkit@beta @celo/utils@beta
```

If you are directly importing any other @celo/ packages, upgrade them as well.

If you need them, append @celo/phone-utils@beta @celo/cryptographic-utils@beta.

(see section on breaks in @celo/utils to know if you need them)

Breaking changes

Because of how we publish packages, all packages will be upgraded to v2. However not all packages will have breaking changes. Breaking changes are limited to:

- @celo/contractkit
- @celo/utils

@celo/contractkit

Most changes are about eliminating the need to construct an entire kit to use other classes and functions.

AddressRegistry

Now takes an Connection instance instead of a kit instance.

CeloTokens

No longer requires kit, instead it requires a class implementing ContractCacheType to be passed in. Examples are WrapperCache or CeloTokensCache.

Wrappers

Note: If you were constructing wrappers with kit.contracts.getX no change is required.

Rather than take the full Kit Wrappers, now construct it like:

```
javascript
// Most Common
constructor(connection: Connection, contract: Contract)
// The Voting Contracts (Governance, Election, Validator, LockedGold,
Slashers, and Attestations
constructor(connection: Connection, contract: Contract, wrapperCache:
WrapperCache)
// Sorted Oracles
constructor(connection: Connection, contract: Contract, addressRegistry:
AddressRegistry)
```

The WrapperCache takes care of this while constructing them and most likely there will not be many situations where wrappers were constructed directly given they needed a kit before.

AccountsWrapper

authorizeValidatorSigner method now requires a ValidatorsWrapper be passed in as final argument.

v1

```
ts
const accountsInstance = await kit.contracts.getAccountsWrapper();

accountsInstance.authorizeValidatorSigner(signer, sig);
```

v2

```
ts
const accountsInstance = await kit.contracts.getAccountsWrapper();
const validatorsInstance = await kit.contracts.getValidatorsWrapper();

accountsInstance.authorizeValidatorSigner(signer, sig,
validatorsInstance);
```

AttestationsWrapper

```
AttestationsWrapper.getConfig()  
andAttestationsWrapper.getHumanReadableConfig()
```

These functions now require an array of fee payable token addresses. You can get these from the CeloTokens class, the Registry, or Token Contracts

```
typescript  
const celoTokens = kit.celoTokens;  
const eachTokenAddress = await celoTokens.getAddresses();  
const addresses = Object.values(eachTokenAddress);  
  
AttestationsWrapper.getConfig(addresses);  
// OR  
AttestationsWrapper.getHumanReadableConfig(addresses);
```

Web3ContractCache

Instead of a kit instance, it requires only a AddressRegistry (uses AddressRegistry's web3 instances).

@celo/utils

Most of the size savings came from removing functionality from @celo/utils into two new packages @celo/phone-utils and @celo/cryptographic-utils

So depending on what you used you will need to add one or both to your package.json.

Phone Utils

If your packages imports any of the following from @celo/utils you will need to change the import to @celo/phone-utils

```
from countries.ts  
  
- CountryNames  
- LocalizedCountry  
- Countries  
  
from getCountryEmoji.ts  
  
- getCountryEmoji  
  
from getPhoneHash.ts  
  
- default (getPhoneHash)  
  
from inputValidation.ts  
  
- validatePhone
```

- validateInput

from io.ts

- AttestationRequestType
- AttestationResponseType
- AttestationResponse
- AttestationServiceTestRequestType
- AttestationServiceTestRequest
- E164PhoneNumberType
- E164Number
- GetAttestationRequestType

from phoneNumbers.ts

- getCountryCode
- getRegionCode
- getRegionCodeFromCountryCode
- getDisplayPhoneNumber
- getDisplayNumberInternational
- getE164DisplayNumber
- getE164Number
- isE164NumberStrict
- parsePhoneNumber
- getExampleNumber

Cryptographic-utils

If your packages imports any of the following from @celo/Utils you will need to change the import to @celo/cryptographic-utils

from account.ts

- generateKeys
- generateKeysFromSeed
- generateDeterministicInviteCode
- generateSeed
- generateMnemonic
- validateMnemonic
- invalidMnemonicWords
- normalizeMnemonic
- formatNonAccentedCharacters
- getAllLanguages
- mnemonicLengthFromStrength
- detectMnemonicLanguage
- suggestMnemonicCorrections

from bls.ts

- BLSPUBLICKEYSIZE
- BLSPOPSIZE
- blsPrivateKeyToProcessedPrivateKey
- getBlsPublicKey
- getBlsPoP

```
from commentEncryption.ts
```

```
- EncryptionStatus
- encryptData
- decryptData
- encryptComment
- decryptComment
```

```
from dataEncryption.ts
```

```
- compressedPubKey
- decompressPublicKey
- deriveDek
```

```
# notes-web3-with-contractkit.md:
```

```
---
```

```
title: Using Web3 from ContractKit with Celo
description: How to use Web3 from ContractKit to read data from the Celo
blockchain.
```

```
---
```

```
Using Web3 from ContractKit
```

```
How to use Web3 from ContractKit to read data from the Celo blockchain.
```

```
---
```

```
:::tip
```

Although the Web3 library was intended to be used only with Ethereum, due to the nature of Celo, we can still use the majority of its features.

```
:::
```

The ContractKit, for every interaction with the node, uses internally a Web3 instance.

Because of this, the Ethereum JSON-RPC calls done via the web3 (except some specific calls that we will explain in this page) are also supported

For example:

```
ts
const web3 = kit.web3;

web3.eth.getBalance(someAddress);
```

```
or
```

```
ts
```



```
const web3 = kit.web3;

web3.eth.getBlock("latest");
```

will work the same way.

Web3 limitations

As you have read in our guide, Celo uses an extra field: `feeCurrency`, that allows you to pay gas with ERC20 Tokens.

To facilitate the life of every developer, we decided to wrap the Provider set in the Web3 instance, and add our way to handle local signing using these new fields. Similar to what Metamask does, we intercept every transaction and perform a local signing when required. This wrapper is called `CeloProvider`.

This let you use the Web3 instance to interact with node's Json RPC API in a transparent way, just deciding which Provider do you need.

This is also the reason that the Kit requires a valid provider from the beginning.

Local Signing

As part of the Donut hardfork network upgrade that occurred on May 19th, 2021, the Celo network now accepts Ethereum-style transactions as well as Celo transactions. This means that you can use Ethereum transaction signing tools (like Metamask, `web3.js` and `ethers.js`) to sign transactions for the Celo network. Remember that Celo is a separate layer 1 blockchain from Ethereum, so do not send Ethereum assets directly to your Celo account address on Ethereum.

```
# odis.md:
```

```
---
title: Querying on-chain identifiers with ODIS
description: How to use ODIS to query the on-chain identifier given a
phone number.
---
```

Query On-Chain Identifiers with ODIS

How to use ODIS to query the on-chain identifier given a phone number.

```
---
```

What is ODIS?

One of Celo's key features is the ability to associate a phone number to a Celo address. This provides a convenient payment experience for Celo users. To map a phone number to an address, the on-chain identifier for a

given phone number must first be retrieved. With this identifier, the address can be looked up on-chain.

:::info

ODIS requests are rate-limited based on transaction history and balance. Ensure the account that is performing the queries has a balance and has performed transactions on the network. If an out of quota error is hit, this indicates that more transactions need to be sent from the querying account.

:::

There are two methods for ODIS:

1. `getPhoneNumberIdentifier` - Query and compute the identifier for a phone number
2. `getContactMatches` - Find mutual connections between users

:::tip

See this overview document for more details on ODIS.

:::

Authentication

Both methods require authentication to the ODIS server, which can be performed by either the main wallet key or the data-encryption key (DEK) associated with the wallet key. This is managed by `AuthSigner`, which can be either a `WalletKeySigner` for a wallet key or an `EncryptionKeySigner` for the DEK. The DEK method is preferred, since it doesn't require the user to access the same key that manages their funds. You can learn more about DEK [here](#).

You may use the `EncryptionKeySigner` for your `AuthSigner` by passing in the raw private key:

```
ts
const authSigner: AuthSigner = {
  authenticationMethod:
    OdisUtils.Query.AuthenticationMethod.ENCRYPTIONKEY,
  rawKey: privateDataKey,
};
```

Alternatively, you may use the `WalletKeySigner` by passing in a `contractkit` instance with the account unlocked:

```
ts
const authSigner: AuthSigner = {
  authenticationMethod: OdisUtils.Query.AuthenticationMethod.WALLETKEY,
  contractKit,
};
```

Service Context

The ServiceContext object provides the ODIS endpoint URL and the ODIS public key (same as above).

```
ts
const serviceContext: ServiceContext = {
  odisUrl,
  odisPubKey,
};
```

The ODIS endpoint URL for each environment can be found here:

Environment	Key
Alfajores Staging	https://us-centrall-celo-phone-number-privacy-stg.cloudfunctions.net
Alfajores	https://us-centrall-celo-phone-number-privacy.cloudfunctions.net
Mainnet	https://us-centrall-celo-pgpn-mainnet.cloudfunctions.net

The ODIS public key for each environment can be found here:

Environment	Key
Alfajores Staging	7FsWGsFnmVvRfMDpzz95Np76wf/1sPaK0Og9yiB+P8Qbjic8FV67NBans9hzZEKbaQMhiapzgMR6CkzIZPvgwQboAxl65JWRZecGe5V3X04sdKeNemdAZ2TzQuWkuZoA
Alfajores	kPoRxWdEdZ/Nd3uQnp3FJFs54zuiS+ksqvOm9x8vY6KHPG8jrfqysvIRU0wtqYsBKA7SoAsICMBv8C/Fb2ZpDOqhSqvr/sZbZoHmQfqbqrzbtDIPvUIrHgRS0ydJCMsA
Mainnet	FvreHfLmhBjwxHxsxeyrcOLtSonC9j7K3WrS4QapYsQH6LdaDTaNGmnlQMfFY04Bp/K4wAvqQwO9/bqPVCKf8Ze8OZo8Frmog4JY4xAiwrsqOXxugl1+htjEelpj4uMA

Query phone number identifier

This call consumes quota. When the user runs out of quota, it's recommended to prompt the user to "purchase" more quota by sending a transaction to themselves. This method returns the pepper retrieved from the service as well as the the computed on-chain identifier that is generated using this pepper and the phone number.

BLS Blinding Client

It's important for user privacy that the ODIS servers don't have the ability to view the raw phone number. Before making the request, the library first blinds the phone number using a BLS library. This prevents the ODIS from being able to see the phone number but still makes the resulting signature recoverable to the original phone number. The blinding client is written in Rust and compiled to Web Assembly, which is not compatible with React native. If you choose not to pass in a `BLSBlindingClient` it will default to the Web Assembly version. You may create a `ReactBlindingClient` by calling the constructor with the ODIS public key:

```
ts
const blsBlindingClient = new ReactBlsBlindingClient(odisPubKey);
```

Or use the `WasmBlsBlindingClient` if your runtime environment supports Web Assembly:

```
ts
const blsBlindingClient = new WasmBlsBlindingClient(odisPubKey);
```

Now you're ready to get the phone number identifier. `OdisUtils.PhoneNumberIdentifier.getPhoneNumberIdentifier` documentation can be found [here](#).

The response will be an object with the original phone number, the on-chain identifier (`phoneHash`), and the phone number's pepper.

You can view an example of this call in our mobile project [here](#).

Matchmaking

Instead of querying for all the user's contact's peppers and consuming the user's quota, it's recommended to only query the pepper before it's actually used (ex. just before sending funds). However, sometimes it's helpful to let your users know that they have contacts already using the Celo network. To do this, you can make use of the matchmaking interface. Given two phone numbers, it will let you know whether the other party has also registered on the Celo network with this identifier. `OdisUtils.Matchmaking.getContactMatches` documentation can be found [here](#).

The response will be a subset of the input `e164NumberContacts` that are matched by the matchmaking service.

You can view an example of this call in our mobile project [here](#).

```
# setup.md:
```

```
---
title: Celo ContractKit Setup
description: ContractKit requirements, installation, and initialization.
---
```

Setup

ContractKit requirements, installation, and initialization.

Installation and System Requirements

To install, run the following:

```
bash npm2yarn
npm install web3@1.10 @celo/contractkit
```

You will need Node.js v18.x.

Initializing the Kit

To start working with ContractKit you need a kit instance and a valid net to connect with. In this example will use alfajores (you can read more about it [here](#))

```
ts
import Web3 from "web3";
import { newKitFromWeb3 } from "@celo/contractkit";

const web3 = new Web3("https://alfajores-forno.celo-testnet.org");
const kit = newKitFromWeb3(web3);
```

Go to the page about Forno for details about different connection types and network endpoints.

Initialize the Kit with your own node

If you are hosting your own node (you can follow this [guide](#) to run one) you can connect our ContractKit to it.

```
js
import Web3 from "web3";
import { newKitFromWeb3 } from "@celo/contractkit";

// define localUrl and port with the ones for your node

const web3 = new Web3(`${localUrl}:${port}`);
const kit = newKitFromWeb3(web3);
```

Same as Web3 we support WebSockets, RPC and connecting via IPC. For this last one you will have to initialize the kit with an instances of Web3 that has a valid IPC Provider

```
ts
```

```
import Web3 from "web3";
import { newKitFromWeb3 } from "@celo/contractkit";

const web3Instance: Web3 = new Web3(
  new
  Web3.providers.IpcProvider("/Users/myuser/Library/CeloNode/geth.ipc",
  net)
);

const kit = newKitFromWeb3(web3Instance);
```

usage.md:

title: Using the Kit

Setting Default Tx Options

kit allows you to set default transaction options:

ts

```
import { CeloContract } from "@celo/contractkit";
```

```
let accounts = await kit.web3.eth.getAccounts();
```

```
kit.defaultAccount = accounts[0];
```

```
// paid gas in cUSD
```

```
await kit.setFeeCurrency(CeloContract.StableToken);
```

Set feeCurrency for a transaction

You can set the feeCurrency for each transaction individually by setting the feeCurrency field in the .send() method. The feeCurrency field accepts contract addresses of whitelisted fee currencies.

js

```
let cUSDcontract = await kit.contracts.getStableToken();
```

```
let cUSDtx = await cUSDcontract
```

```
  .transfer(someAddress, amount)
```

```
  .send({ feeCurrency: cUSDcontract.address });
```

Getting the Total Balance

This method from the kit will return the CELO, locked CELO, cUSD and total balance of the address

ts

```
let totalBalance = await kit.getTotalBalance(myAddress);
```

Deploy a contract

Deploying a contract with the default account already set. Simply send a transaction with no `to:` field. See more about sending custom transactions below.

You can verify the deployment on the Alfajores block explorer [here](#). Wait for the receipt and log it to get the transaction details.

```
ts
let bytecode = "0x608060405234..."; // compiled Solidity deployment
bytecode

let tx = await kit.sendTransaction({
  data: bytecode,
});

let receipt = tx.waitReceipt();
console.log(receipt);
```

Sending Custom Transactions

The Celo transaction object is not the same as Ethereum's. There is a new optional field present:

- `feeCurrency` (address of the ERC20 contract to use to pay for gas)

`feeCurrency` enables transaction fees to be paid in currencies other than CELO. The currently supported fee currencies are CELO, cUSD and cEUR. You can specify the currency by passing the contract address of the currency you would like the transaction fees to be paid in.

Celo accepts original Ethereum type transactions as well, so you can use Ethereum signing tools (like Metamask) as well as Celo specific wallets and tools. You can read more about these transaction formats in CIP 35.

For a raw transaction:

```
ts
const tx = kit.sendTransaction({
  from: myAddress,
  to: someAddress,
  value: oneGold,
});
const hash = await tx.getHash();
const receipt = await tx.waitReceipt();
```

When interacting with a web3 contract object:

```
ts
const goldtoken = await kit.web3Contracts.getGoldToken();
const oneGold = kit.web3.utils.toWei("1", "ether");
```

```

const txo = await goldtoken.methods.transfer(someAddress, oneGold);
const tx = await kit.sendTransactionObject(txo, { from: myAddress });
const hash = await tx.getHash();
const receipt = await tx.waitReceipt();

```

Interacting with Custom contracts

You can use ContractKit to interact with any deployed smart contract, provided you have the contract address and the ABI. To do so, you will initialize a new web3 Contract instance. Then you can call functions on the contract instance to read state or send transactions to update the contract. You can see some code snippets below.

```

ts
let cUSDContract = await kit.contracts.getStableToken();
let contract = new kit.connection.web3.eth.Contract(ABI, address); //
Init a web3.js contract instance
let name = await instance.methods.getName().call(); // Read contract
state

// Specifying the 'from' account and 'feeCurrency' is optional
// Transactions with an unspecified feeCurrency field will default to
paying fees in CELo
const tx = await instance.methods
    .setName(newName)
    .send({ from: account.address, feeCurrency: cUSDContract.address });

```

Selling CELo only if the rate is favorable

```

ts
// This is at lower price I will accept in cUSD for every CELo
const favorableAmount = 100;
const amountToExchange = kit.web3.utils.toWei("10", "ether");
const oneGold = kit.web3.utils.toWei("1", "ether");
const exchange = await kit.contracts.getExchange();

const amountOfcUsd = await exchange.quoteGoldSell(oneGold);

if (amountOfcUsd > favorableAmount) {
    const goldToken = await kit.contracts.getGoldToken();
    const approveTx = await goldToken
        .approve(exchange.address, amountToExchange)
        .send();
    const approveReceipt = await approveTx.waitReceipt();

    const usdAmount = await exchange.quoteGoldSell(amountToExchange);
    const sellTx = await exchange.sellGold(amountToExchange,
usdAmount).send();
    const sellReceipt = await sellTx.waitReceipt();
}

```


Buying all the CELO I can, with the cUSD in my account

```
ts
const stableToken = await this.contracts.getStableToken();
const exchange = await this.contracts.getExchange();

const cUsdBalance = await stableToken.balanceOf(myAddress);

const approveTx = await stableToken
    .approve(exchange.address, cUsdBalance)
    .send();
const approveReceipt = await approveTx.waitReceipt();

const goldAmount = await exchange.quoteUsdSell(cUsdBalance);
const sellTx = await exchange.sellDollar(cUsdBalance, goldAmount).send();
const sellReceipt = await sellTx.waitReceipt();
```

foundry.md:

title: Deploy with Foundry

description: How to deploy a Smart Contract to Celo using Foundry

Deploy on Celo with Foundry

How to deploy a smart contract to Celo testnet, mainnet, or a local network using Foundry.

Introduction to Foundry

Foundry is a smart contract development toolchain.

Foundry manages your dependencies, compiles your project, runs tests, deploys, and lets you interact with the chain from the command-line and via Solidity scripts.

Prerequisites

You will need the Rust compiler and Cargo, the Rust package manager. The easiest way to install both is with `rustup.rs`.

Using Foundryup

Foundryup is the Foundry toolchain installer. You can find more about it [here](#).

Open your terminal and run the following command:

```
bash
curl -L https://foundry.paradigm.xyz | bash
```

This will install Foundryup, then simply follow the instructions on-screen, which will make the foundryup command available in your CLI.

Running foundryup by itself will install the latest (nightly) precompiled binaries: forge, cast, anvil, and chisel.

Create a new project using Forge

To start a new project with Foundry, use:

```
bash
forge init hellofoundry
```

```
:::info
If you initializing project in an already initialized git repo use:
```

```
bash
forge init projectname --no-git
```

```
:::
```

Adding a dependency

To add a dependency, use:

```
bash
forge install openzeppelin/openzeppelin-contracts
```

Remapping dependencies

Forge can remap dependencies to make them easier to import. Forge will automatically try to deduce some remappings for you

```
bash
$ forge remappings

ds-test/=lib/solmate/lib/ds-test/src/
forge-std/=lib/forge-std/src/
solmate/=lib/solmate/src/
weird-erc20/=lib/weird-erc20/src/
```

These remappings mean:

- To import from forge-std we would write: `import forge-std/Contract.sol;`
- To import from ds-test we would write: `import ds-test/Contract.sol;`
- To import from solmate we would write: `import solmate/Contract.sol;`

- To import from weird-erc20 we would write: `import weird-erc20/Contract.sol;`

You can customize these remappings by creating a `remappings.txt` file in the root of your project.

Removing dependencies

You can remove dependencies using

```
bash
forge remove openzeppelin/openzeppelin-contracts
```

Adding Celo specific config to `foundry.toml`

Add the following configuration to `foundry.toml` file in the root level of your project.

```
toml
[_rpcendpoints]
celo-alfajores = "https://alfajores-forno.celo-testnet.org"
celo = "https://forno.celo.org"
```

Deploying contract

Forge can deploy smart contracts to a given network using:

The below example deploys Counter contract at location `src/Counter.sol` in the project to the Celo Alfajores Testnet.

```
bash
forge create --rpc-url celo-alfajores --private-key <yourprivatekey>
src/Counter.sol:Counter
```

`:::info`

Notice the contract name after `:`, this is because a single solidity file can have multiple contracts.

It is recommended to use `--verify` flag so that the contract gets verified right after deployment, this requires etherscan configuration in the `foundry.toml` file.

`:::`

On successful deployment, you should a following output!

`!github`

`# hardhat.md:`

`---`

title: Deploy with Hardhat
description: How to deploy a Smart Contract to Celo using Hardhat

Deploy on Celo with Hardhat

How to deploy a smart contract to Celo testnet, mainnet, or a local network using Hardhat.

Introduction to Hardhat

Hardhat is a development environment to compile, deploy, test, and debug your Ethereum or Celo software. It helps developers manage and automate the recurring tasks that are inherent to the process of building smart contracts and dApps, as well as easily introducing more functionality around this workflow. This means compiling, running, and testing smart contracts at the very core.

Prerequisites

To deploy on Celo using Hardhat, you should have Celo set up Celo in your local environment. If you prefer to deploy without a local environment, you can deploy using Remix or Replit.

- Using Windows
- Using Mac
- Using Replit

Create Hardhat Project

Choose one of the following items to prepare a dApp to deploy on Celo.

- Follow the installation instructions and quickstart to build and deploy your smart contract.

Update the hardhat.config.js file

Open hardhat.config.js in a text editor and replace its contents with this Celo configuration code. This code is similar to Hardhat settings with a few configuration updates needed to deploy to a Celo network. You will need to create a .env file in the project root directory and install dotenv with npm or yarn in order to read the process.env.MNEMONIC variable in the config file.

Connect to Local Network

Using Celo Ganache CLI creates test accounts at the localhost on port 7545. The private network setup connects to your localhost on this port and gives you access to your accounts on ganache-cli.

```
js
  localhost: {
```

```
    url: "http://127.0.0.1:7545"
  },
```

:::tip

If you choose to Set up a Local Development Chain, your blockchain will also be hosted on a private network on localhost. This same configuration can be used to connect to the local development chain.

:::

Connect to Testnet using Forno

Using Forno allows you to connect to the Celo test blockchain without running a local node. The testnet configuration uses Forno to connect you to the Celo Testnet (Alfajores) using HDWalletProvider and the mnemonic stored in your .env file.

```
js
alfajores: {
  url: "https://alfajores-forno.celo-testnet.org",
  accounts: {
    mnemonic: process.env.MNEMONIC,
    path: "m/44'/52752'/0'/0"
  },
  chainId: 44787
}
```

:::note

Celo uses a different account derivation path than Ethereum, so you have to specify "m/44'/52752'/0'/0" as the path.

:::

Connect to Mainnet using Forno

Using Forno also allows you to connect to the Celo main blockchain without running a local node. The mainnet configuration uses Forno to connect you to the Celo Mainnet using HDWalletProvider and the mnemonic stored in your .env file.

```
js
celo: {
  url: "https://forno.celo.org",
  accounts: {
    mnemonic: process.env.MNEMONIC,
    path: "m/44'/52752'/0'/0"
  },
  chainId: 42220
}
```

:::tip

Forno is a cLabs hosted node service for interacting with the Celo network. This allows you to connect to the Celo Blockchain without having to run your own node.

:::

Deploy to Celo

Run the following command from your root project directory to deploy to Celo Alfajores testnet.

```
shell
npx hardhat run scripts/sample-script.js --network alfajores
```

...or run this command to deploy to Celo mainnet.

```
shell
npx hardhat run scripts/sample-script.js --network celo
```

View Contract Deployment

Copy your contract address from the terminal and navigate to the block explorer to search for your deployed contract. Switch between networks to find your contract using the dropdown by the search bar.

!github

:::tip

Learn more about building and deploying dApps using the HardHat documentation.

:::

Verify Contracts on Celo

- Using Celo Explorer
- Using Remix
- Using CeloScan
- Using Hardhat

index.md:

title: Deploy on Celo

description: How to build and deploy a dApp on Celo.

```
import PageRef from '@components/PageRef'
```

Build with Celo

How to build and deploy a dApp with Celo.

Using Celo Composer

Celo Composer allows you to quickly build, deploy, and iterate on decentralized applications using Celo. It provides a number of frameworks, examples, and Celo specific functionality to help you get started with your next dApp.

```
jsx
npx @celo/celo-composer@latest create
```

:::tip

Learn more about Celo Composer in the README and Documentation

:::

Using EVM Tools

<!-- make the below text code block because crowdin is messing it up -->

mdx-code-block

Developers can build with Celo using many Ethereum compatible tools including Remix, Hardhat, and others. By making a few adjustments to your project's network configuration settings, you can deploy your new or existing dApp on Celo.

- Using thirdweb
- Using Remix
- Using Hardhat

remix.md:

```
title: Deploy with Remix
description: How to deploy a Smart Contract to Celo using
remix.ethereum.org.
```

Deploy on Celo with Remix

How to deploy a smart contract to Celo testnet, mainnet, or a local network using Remix.

Introduction to Remix

The Remix IDE is an open-source web and desktop application for creating and deploying Smart Contracts. Originally created for Ethereum, it fosters a fast development cycle and has a rich set of plugins with intuitive GUIs. Remix is used for the entire journey of contract development and is a playground for learning and teaching Celo.

In this guide, you will learn to deploy a smart contract on Celo using remix.ethereum.org.

:::warning

For Celo L1 Remix does not support Solidity compiler version 0.8.20 and above for EVM versions above Paris. If you try to deploy a smart contract with a higher version, you will receive this error message:

bash

Gas estimation errored with the following message (see below). The transaction execution will likely fail. Do you want to force sending?

invalid opcode: opcode 0x5f not defined

The EVM version used by the selected environment is not compatible with the compiler EVM version.

A workaround is to go into the advanced settings for the compiler in Remix and choose Paris as the EVM version.

For Alfajores L2 everything should be working as on every other EVM compatible chain.

:::

:::tip

To learn more about the features available to you as a smart contract developer with Remix, visit the [Remix documentation](#).

:::

Create a Smart Contract

- Navigate to remix.ethereum.org and select contracts > 1Storage.sol from the File Explorers pane.
- Review the smart contract code and learn more using the Solidity docs or with Solidity by Example.
- Complete any changes to your smart contract and save the final version (Command/Ctrl + S).

!github

Compile the Contract

- Choose the Solidity Compiler Icon on the left side menu.
- Check that your compiler version is within the versions specified in the pragma solidity statement.
- Select the Compile button to compile your smart contract.

!github

Deploy the Contract

- Click the Deploy and Run Transactions Icon on the left side menu.
- Choose Injected Web3 as your environment.
- Connect MetaMask to Celo testnet and verify that the environment reads:
 - Custom (44787) network for Celo testnet
 - Custom (42220) network for Celo mainnet
- Click Deploy and select Confirm in the MetaMask notification window to pay for the transaction

!github

Interacting with the Contract

- Select the dropdown on the newly deployed contract at the bottom of the left panel.
- View the deployed contract's functions using the Deployed Contracts window.
- Select functions to read or write on the Celo testnet using the function inputs as needed.
- Confirm write transactions in the MetaMask Notification Window to pay the transaction's gas fee.

!github

View Contract Details

- Copy the contract address from the Deployed Contracts window on the left panel.
- Navigate to the Celo Block Explorer and use the contract address to search for your contract.
- Explore the details of your deployed smart contract and learn more about the explorer here.

!github

Verify Contracts on Celo

- Using Celo Explorer
- Using Remix
- Using CeloScan
- Using Hardhat

thirdweb.md:

Using thirdweb

Create Contract

To create a new smart contract using thirdweb CLI, follow these steps:

1. In your CLI run the following command:

```
npx thirdweb create contract
```

2. Input your preferences for the command line prompts:

1. Give your project a name
2. Choose your preferred framework: Hardhat or Foundry
3. Name your smart contract
4. Choose the type of base contract: Empty, ERC20, ERC721, or ERC1155
5. Add any desired extensions

3. Once created, navigate to your project's directory and open in your preferred code editor.

4. If you open the contracts folder, you will find your smart contract; this is your smart contract written in Solidity.

The following is code for an ERC721Base contract without specified extensions. It implements all of the logic inside the ERC721Base.sol contract; which implements the ERC721A standard.

```
bash
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@thirdweb-dev/contracts/base/ERC721Base.sol";

contract Contract is ERC721Base {
    constructor(
        string memory name,
        string memory symbol,
        address royaltyRecipient,
        uint128 royaltyBps
    ) ERC721Base(name, symbol, royaltyRecipient, royaltyBps) {}
}
```

This contract inherits the functionality of ERC721Base through the following steps:

- Importing the ERC721Base contract
- Inheriting the contract by declaring that our contract is an ERC721Base contract
- Implementing any required methods, such as the constructor.

5. After modifying your contract with your desired custom logic, you may deploy it to Celo using Deploy.

Alternatively, you can deploy a prebuilt contract for NFTs, tokens, or marketplace directly from the thirdweb Explore page:

1. Go to the thirdweb Explore page: <https://thirdweb.com/explore>

!thirdweb Explore page

2. Choose the type of contract you want to deploy from the available options: NFTs, tokens, marketplace, and more.

3. Follow the on-screen prompts to configure and deploy your contract.

> For more information on different contracts available on Explore, check out thirdweb's documentation.

Deploy Contract

Deploy allows you to deploy a smart contract to any EVM compatible network without configuring RPC URLs, exposing your private keys, writing scripts, and other additional setup such as verifying your contract.

1. To deploy your smart contract using deploy, navigate to the root directory of your project and execute the following command:

```
bash
npx thirdweb deploy
```

Executing this command will trigger the following actions:

- Compiling all the contracts in the current directory.
- Providing the option to select which contract(s) you wish to deploy.
- Uploading your contract source code (ABI) to IPFS.

2. When it is completed, it will open a dashboard interface to finish filling out the parameters.

- name: contract name
- symbol: symbol or "ticker"
- royaltyRecipient: wallet address to receive royalties from secondary sales
- royaltyBps: basis points (bps) that will be given to the royalty recipient for each secondary sale, e.g. 500 = 5%

3. Select Celo as the network

4. Manage additional settings on your contract's dashboard as needed such as uploading NFTs, configuring permissions, and more.

For additional information on Deploy, please reference thirdweb's documentation.

If you have any further questions or encounter any issues during the process, please reach out to thirdweb support at support.thirdweb.com.

index.md:

```
---
title: Dynamic
description: Overview of Dynamic
---
```

Using Dynamic

Dynamic is a powerful web3 auth developer platform with built-in support for Celo. It lets you integrate multiple wallets such as Valora, Celo Wallet, Coinbase Wallet, Metamask, and more into your app or website, handles network switching, multi-wallet linking and more.

Dynamic comes with Celo built-in. You can play around with a live demo of Dynamic [here](#) and see a full video walkthrough [here](#). In this tutorial, we'll go through how to set up Dynamic with Celo.

You can see a CodeSandbox of the example below [here](#) (configured to Celo).

Prerequisites

Dynamic works with React today. You can go through the standard getting started guide [here](#).

Step 1: Create a Dynamic account

1. Sign up to get an environment ID
2. Create an organization and a set up your first project
3. Copy your environmentID from the Dynamic overview page
4. (optional) ` Configure your site's CORS origins`

Step 2: Install the Dynamic npm package

You can install Dynamic's SDK with either yarn or npm. We currently support React and NextJS.

```
shell
npm install @dynamic-labs/sdk-react
or yarn add @dynamic-labs/sdk-react
```

Step 3: Configure the SDK

Copy the following snippet into your project and paste in your environmentId:

```

jsx
import { DynamicContextProvider, DynamicWidget } from "@dynamic-labs/sdk-react";

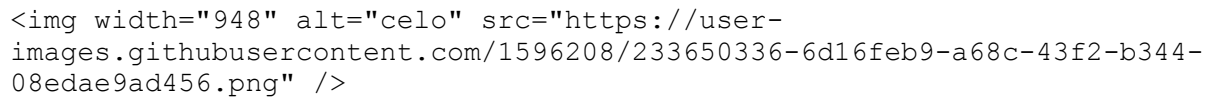
const App = () => (
  <DynamicContextProvider
    settings={{
      environmentId: "<<sandboxEnvironmentId>>",
    }}
  >
    <DynamicWidget />
  </DynamicContextProvider>
);

export default App;

```

Step 4: Turn on Celo in your developer dashboard

Now that we have the basic Dynamic setup, you can go to your developer dashboard, and select configurations from the left menu. Next, click on the EVM card and toggle Celo on. Note that you can also chose to toggle the default network, Ethereum, off.

The image shows a screenshot of a web page with the title "celo". The URL in the browser address bar is "https://user-images.githubusercontent.com/1596208/233650336-6d16feb9-a68c-43f2-b344-08edae9ad456.png". The page content is mostly obscured by a large, semi-transparent watermark that reads "dynamic".

See it in action

Now that you put things together, you can see a CodeSandbox of the finished product here.

Next steps

Now that you set up Dynamic with Celo, there are many additional things you can explore:

- Log in with your wallet, and see data in Dynamic's developer dashboard
- Now that your widget is set up locally, try to log in with a wallet. As soon as you do, head over to the Dynamic developer dashboard and click on user management and analytics. You'll be able to see your user show up!
- Customize your SDK design - There are many ways to customize the Dynamic modal to fit your needs (you can explore them in the SDK configuration section), but to start, we suggest setting a light/dark mode and a primary color for the modal, which you can do in the design section of your developer dashboard.
- Explore how to use the Dynamic SDK - After your users connect their wallets, you'll want to interact with them for various reasons. You can read more about the SDK in Dynamic's docs.

For support, you can also join Dynamic's Slack Community

the-graph.md:

The Graph

Getting historical data on a smart contract can be frustrating when you're building a dapp. The Graph provides an easy way to query smart contract data through APIs known as subgraphs. The Graph's infrastructure relies on a decentralized network of indexers, enabling your dapp to become truly decentralized.

Quick Start

It takes just a few minutes to start indexing subgraphs on Celo. To get started, follow these three steps:

1. Initialize your subgraph project
2. Deploy & Publish
3. Query from your dapp

Pricing: All developers receive 100K free queries per month on the decentralized network. After these free queries, you only pay based on usage at \$4 for every 100K queries.

Here's a step by step walk through:

1. Initialize your subgraph project

Create a subgraph on Subgraph Studio

Go to the Subgraph Studio and connect your wallet. Once your wallet is connected, you can begin by clicking "Create a Subgraph". Please choose a good name for the subgraph because this name can't be edited later. It is recommended to use Title Case: "Subgraph Name Chain Name."

!Create a Subgraph

You will then land on your subgraph's page. All the CLI commands you need will be visible on the right side of the page:

!CLI commands

Install the Graph CLI

On your local machine run the following:

```
npm install -g @graphprotocol/graph-cli
```

Initialize your Subgraph

You can copy this directly from your subgraph page to include your specific subgraph slug:

```
graph init --studio <SUBGRAPHSLUG>
```

You'll be prompted to provide some info on your subgraph like this:

```
!cli sample
```

Simply have your contract verified on the block explorer and the CLI will automatically obtain the ABI and set up your subgraph. The default settings will generate an entity for each event.

2. Deploy & Publish

Deploy to Subgraph Studio

First run these commands:

```
bash
$ graph codegen
$ graph build
```

Then run these to authenticate and deploy your subgraph. You can copy these commands directly from your subgraph's page in Studio to include your specific deploy key and subgraph slug:

```
bash
$ graph auth --studio <DEPLOYKEY>
$ graph deploy --studio <SUBGRAPHSLUG>
```

You will be asked for a version label. You can enter something like v0.0.1, but you're free to choose the format.

Test your subgraph

You can test your subgraph by making a sample query in the playground section. The Details tab will show you an API endpoint. You can use that endpoint to test from your dapp.

```
!Playground
```

Publish Your Subgraph to The Graph's Decentralized Network

Once your subgraph is ready to be put into production, you can publish it to the decentralized network. On your subgraph's page in Subgraph Studio, click on the Publish button:

```
!publish button
```

Before you can query your subgraph, Indexers need to begin serving queries on it. In order to streamline this process, you can curate your own subgraph using GRT.

When publishing, you'll see the option to curate your subgraph. As of May 2024, it is recommended that you curate your own subgraph with at least 3,000 GRT to ensure that it is indexed and available for querying as soon as possible.

!Publish screen

> Note: The Graph's smart contracts are all on Arbitrum One, even though your subgraph is indexing data from Ethereum, BSC or any other supported chain.

3. Query your Subgraph

Congratulations! You can now query your subgraph on the decentralized network!

For any subgraph on the decentralized network, you can start querying it by passing a GraphQL query into the subgraph's query URL which can be found at the top of its Explorer page.

Here's an example from the CryptoPunks Ethereum subgraph by Messari:

!Query URL

The query URL for this subgraph is:

```
https://gateway-arbitrum.network.thegraph.com/api/[api-key]/subgraphs/id/HdVdERFUe8h6lvm2fDyyCHgxjsde5PbB832NHgJfZNqK
```

Now, you simply need to fill in your own API Key to start sending GraphQL queries to this endpoint.

Getting your own API Key

!API keys

In Subgraph Studio, you'll see the "API Keys" menu at the top of the page. Here you can create API Keys.

Appendix

Sample Query

This query shows the most expensive CryptoPunks sold.

```
graphql
{
```



```

    trades(orderBy: priceETH, orderDirection: desc) {
      priceETH
      tokenId
    }
  }
}

```

Passing this into the query URL returns this result:

```

{
  "data": {
    "trades":
      [
        {
          "priceETH": "124457.067524886018255505",
          "tokenId": "9998"
        },
        {
          "priceETH": "8000",
          "tokenId": "5822"
        },
        // ...
      ]
    }
  }
}

```

<aside>

💡 Trivia: Looking at the top sales on [CryptoPunks website it looks like the top sale is Punk #5822, not #9998. Why? Because they censor the flash-loan sale that happened.

</aside>

Sample code

```

jsx
const axios = require('axios');

const graphqlQuery = {
  trades(orderBy: priceETH, orderDirection: desc) {
    priceETH
    tokenId
  }
};

const queryUrl = 'https://gateway-arbitrum.network.thegraph.com/api/[api-key]/subgraphs/id/HdVdERFUE8h61vm2fDyyCHgxjsde5PbB832NHgJfZNqK'

const graphQLRequest = {
  method: 'post',
  url: queryUrl,
  data: {
    query: graphqlQuery,
  },
};

```

```
// Send the GraphQL query
axios(graphQLRequest)
  .then((response) => {
    // Handle the response here
    const data = response.data.data
    console.log(data)

  })
  .catch((error) => {
    // Handle any errors
    console.error(error);
  });
```

Additional resources:

- To explore all the ways you can optimize & customize your subgraph for a better performance, read more about creating a subgraph [here](#).
- For more information about querying data from your subgraph, read more [here](#).

from-ethereum.md:

```
---
title: Celo for Ethereum Developers
description: Overview of the similarities and differences between the
Celo and Ethereum blockchains.
---
```

Celo for Ethereum Developers

Overview of the similarities and differences between the Celo and Ethereum blockchains.

```
---
```

:::tip

For a general overview of the Celo network and architecture, see the [Celo Overview](#) page.

:::

What is Celo's Relationship to Ethereum?

Celo is a layer 1 protocol and blockchain platform, and the Celo Mainnet is entirely separate from the Ethereum network.

While the Celo client originated as a fork of Ethereum Go language client, go-ethereum (or geth), it has several significant differences, including a proof-of-stake based PBFT consensus mechanism. All the cryptoassets on Celo have ERC-20 compliant interfaces, meaning that while they are not ERC-20 tokens on the Ethereum Mainnet, all familiar tooling and code that support ERC-20 tokens can be easily adapted for Celo

assets, including the Celo Native Asset (CELO) and the Celo Dollar (cUSD).

In terms of programmability, Celo is similar to Ethereum. Both networks run the Ethereum Virtual Machine (EVM) to support smart contract functionality.

This means that all programming languages, developer tooling and standards that target the EVM are relevant for both Celo and Ethereum. Developers building on Celo can write smart contracts in Solidity, and take advantage of smart contract standards that have already been developed for Ethereum.

The ERC-20 Token Standard

The ERC20 token standard is a standard API for tokens within smart contracts.

This standard interface allows any tokens to be re-used by different applications.

The ERC20 token standard is blockchain agnostic, so ERC20 tokens can be implemented on any blockchain.

The standard includes the optional functions

```
javascript
function name() public view returns (string)
function symbol() public view returns (string)
function decimals() public view returns (uint8)
```

and the required functions

```
javascript
function totalSupply() public view returns (uint256)
function balanceOf(address owner) public view returns (uint256 balance)
function transfer(address to, uint256 value) public returns (bool success)
function transferFrom(address from, address to, uint256 value) public returns (bool success)
function approve(address spender, uint256 value) public returns (bool success)
function allowance(address owner, address spender) public view returns (uint256 remaining)
```

and includes the following events

```
js
event Transfer(address indexed from, address indexed to, uint256 value)
event Approval(address indexed owner, address indexed spender, uint256 value)
```

An ERC20 compliant contract must include the required functions and events at minimum.

It can include additional functions and events and still be ERC20 compliant.

The Celo Native Asset and the Celo Dollar

This interface is relevant for two important assets on the Celo network, the Celo native asset (CELO) and the Celo Dollar (cUSD).

CELO was called Celo Gold (cGLD) when the contract was deployed, so you will often see references to Celo Gold in the codebase. CELO and cGLD are the same thing. You can view the CELO implementation [here](#).

CELO has an ERC20 interface, so users can interact with CELO via the token standard, but it is important to note that not all CELO transfers are required to go through the token contract.

CELO can also be transferred by specifying the value field of a transaction, in the same way that ETH can be transferred in Ethereum. To properly monitor balance changing operations of CELO, it can be helpful to use Celo Rosetta.

Celo Rosetta provides an easy way to obtain changes that are not easily queryable using the celo-blockchain RPC.

The Celo Dollar (cUSD) is implemented solely as a smart contract, so all cUSD actions are mediated by the smart contract. You can view the implementation [here](#).

Key differences between Celo and Ethereum

Features exclusive to Celo

1. Celo allows users to pay transaction fees in cryptoassets other than the native asset. On Ethereum, users must pay transaction fees in Ether. For example, users can send cUSD, and then pay any transaction fees in cUSD as well.
2. The Celo protocol uses BFT Proof-of-Stake for maintaining consensus. This allows blocks on Celo to be created in 5 seconds, as compared to 12+ seconds on Ethereum. In addition, all blocks are finalized immediately, so there is no need to wait for more than 1 block confirmation to ensure that a transaction won't be reverted.

Things to watch out for

1. As previously mentioned, CELO transfers are not required to happen via the ERC20 interface. A user's CELO balance may change without any interaction with the CELO contract, as they may transfer CELO natively.
2. Celo transaction objects are slightly different from transaction objects on Ethereum.
Ethereum transaction objects include fields to, value, gas, gasPrice, data, nonce, signature (v,r,s).
Celo transaction objects include the same fields as Ethereum transaction objects, plus the feeCurrency field.

This additional field is included to allow users to pay transaction fees in different currencies. As of May 19th, 2021, with the Donut hardfork, the Celo network accepts both Celo transaction objects and Ethereum transaction objects as valid Celo transactions. This means that you can use most Ethereum tools with Celo, right out of the box (just point them at the Celo network). When sending Ethereum formatted transactions on Celo, you will not be able to use Celo features of specifying transaction fee currencies or full node incentives.

1) When using mnemonic seed phrases (or secret phrases), Celo accounts (a private key and corresponding address) are derived differently from Ethereum accounts. The Celo key derivation path is `m/44'/52752'/0'/0` whereas Ethereum's is `m/44'/60'/0'/0`. This means that going from a seed phrase to accounts will be different when using Ethereum vs Celo wallets.

2) The Valora wallet uses two types of accounts: externally owned accounts and meta-transaction wallets. There are important consequences for wallet developers and dapp developers building on Celo as Valora is one of the main interfaces for Celo users. You can find more information about Valora accounts [here](#).

Deploying Ethereum Contracts to Celo

Celo runs the EVM which means that smart contracts written for Ethereum can easily be deployed to Celo, the main difference being that you just need to connect to a Celo node instead of an Ethereum node. You can connect to your own Celo node or to a Celo node service provider like [Quicknode](#).

Protocol Differences

OPCODES & Block headers

Celo does not support the `DIFFICULTY` or `GASLIMIT` opcodes. These fields are also absent from Celo block headers.

Precompiled Contracts

Celo includes all of the precompiled contracts in Ethereum, but also adds additional contracts. Here is the list of Celo precompiled contracts as of Celo version 1.3.2. You can find the latest updates by selecting the most recent release tag.

Core Contract Calls

The blockchain client makes some core contract calls at the end of a block, outside of transactions. Many are done on epoch blocks (epoch rewards, validator elections, etc.), but not all. For example, the gas price minimum update can happen on any block.

Logs created by these contract changes are included in a single additional receipt in that block, which references the block hash as its transaction hash, even though there is no transaction with this hash. If no logs were created by such calls in that block, no receipt is added.

Node management APIs

Celo nodes have a slightly different RPC interface than geth nodes. There are some additional RPC endpoints to help validators manage their nodes, they can be found [here](#) and [here](#).

You can find the full list of RPC API endpoints in [this file](#).

index.md:

title: Particle Network

description: Walkthrough on leveraging Particle Network's Wallet-as-a-Service on Celo.

Particle Network's Wallet-as-a-Service

Particle Network's Wallet Abstraction services enable universal, Web2-adjacent onboarding and interactions through social logins. Its core technology, Smart Wallet-as-a-Service (WaaS) aims to onboard users into MPC-secured accounts supporting any chain. It also allows developers to offer an improved user experience through modular, fully customizable EOA/AA embedded wallets. Particle supports its Smart Wallet-as-a-Service through a Modular L1 powering chain abstraction, acting as a settlement layer across chains for a seamless multi-chain experience.

Particle Network's Wallet-as-a-Service supports the Celo Mainnet and Alfajores Testnet through standard EOA-based social logins. Therefore, developers building on Celo can natively leverage Particle Network to onboard users into application-embedded wallets using social logins through various SDKs with direct Celo compatibility.

On this page, you can find a high-level overview and tutorial on implementing Particle Network's Wallet-as-a-Service within an application built on Celo (specifically Celo Mainnet in this example), highlighting the basics of getting an integration up and running.

Tutorial: Implementing Particle Network's Wallet-as-a-Service on Celo

Particle Network has various avenues for integration, with the best one for you being largely dependent on your platform of choice. If you're building a mobile application, Particle offers various SDKs for iOS, Android, Flutter, React Native, and so on. In this example, we'll focus on using Particle's flagship web SDK, `@particle-network/auth-core-modal`. This SDK facilitates end-to-end implementation of social logins within web apps (using React). To install this library, alongside some supporting ones, run one of the two following commands at the root of your project:

```
shell
```

```
yarn add @particle-network/auth-core-modal @particle-network/chains
```

OR

```
npm install @particle-network/auth-core-modal @particle-network/chains
```

These two libraries are all you'll need to begin using social logins within a new or existing application built on Celo. For this tutorial, we'll put together a basic React application showcasing the utilization of social logins to facilitate wallet generation on Celo.

To get started:

1. Configure Particle Auth Core (@particle-network/auth-core-modal) within index.ts/tsx (or an adjacent file).

To start using Particle Auth Core (Particle's flagship authentication SDK), you'll first need to configure and initialize it within your index.ts/tsx file through AuthCoreContextProvider (imported from @particle-network/auth-core-modal). AuthCoreContextProvider acts as the primary interface for configuration; through options, you'll need to pass the following parameters, all derived from the Particle dashboard:

- projectId, your project ID.
- clientKey, your client Key.
- appId, your app ID.

Each of these values are required as they directly link your instance of Particle Auth Core with the Particle dashboard, therefore enabling no-code customization, user activity tracking, API requests authentication, etc.

Beyond these core parameters (projectId, clientKey, and appId), you have a few other optional configurations you can set (largely visual). An example of an index.ts/tsx file, with AuthCoreContextProvider successfully initialized, is included below.

```
js
import React from 'react'
import ReactDOM from 'react-dom/client'
import { Celo } from '@particle-network/chains';
import { AuthCoreContextProvider, PromptSettingType } from '@particle-network/auth-core-modal';
import App from './App'
```

```
ReactDOM.createRoot(document.getElementById('root') as
HTMLDivElement).render(
  <React.StrictMode>
    <AuthCoreContextProvider
      options={{
        projectId: process.env.REACTAPPPROJECTID,
        clientKey: process.env.REACTAPPCLIENTKEY,
        appId: process.env.REACTAPPAPPID,
        themeType: 'dark', // Optional
        fiatCoin: 'USD', // Optional
```

```

        language: 'en', // Optional
        promptSettingConfig: { // Optional, determines the security
settings that a user has to configure
            promptPaymentPasswordSettingWhenSign: PromptSettingType.first,
            promptMasterPasswordSettingWhenLogin: PromptSettingType.first,
        },
        wallet: { // Optional, streamlines the wallet modal popup
            visible: true, // Displays an embedded wallet popup on the
bottom right of the screen after login
            customStyle: {
                supportChains: [Celo],
            }
        },
    }},
    >
    <App />
    </AuthCoreContextProvider>
</React.StrictMode>
)

```

2. Set up various hooks within your primary App component (or its equivalent).

Now that you've configured Particle Auth Core (through `AuthCoreContextProvider`, as shown above), you're ready to use the full extent of the SDK (to facilitate social logins) within your application, in this case through your main App component or whichever file you intend on using Particle Auth Core within. This file should be specified within your `index.ts/tsx` file, wrapped by `AuthCoreContextProvider`.

For this example, we'll be using three hooks exported by `@particle-network/auth-core-modal` to connect a user through social login, set up a custom EIP-1193 provider (with Ethers), and retrieve information about the user once they've logged in. These hooks are as follows:

- `useConnect` connects and disconnects a user through a specified social login mechanism (or through Particle Network's generalized authentication modal).
- `useEthereum` retrieves the user's address, the associated 1193 provider object, etc.
- `useAuthCore`, pulls data about the user once they've connected (such as their public email used for signing up).

For a complete list of hooks, including those covered above, take a look at the Particle Auth Core documentation.

An example of using each of the three aforementioned hooks to build a sample application that onboards a user through social logins, retrieves and displays relevant data, and executes a sample transaction has been included below:

```

js
import React, { useState, useEffect } from 'react';

```



```

import { useEthereum, useConnect, useAuthCore } from '@particle-network/auth-core-modal';
import { Celo } from '@particle-network/chains';

import { ethers } from 'ethers';
import { notification } from 'antd';

import './App.css';

const App = () => {
  const { provider } = useEthereum(); // For provider retrieval
  const { connect, disconnect } = useConnect(); // For facilitating
  social logins
  const { userInfo } = useAuthCore(); // For retrieving user information

  const [balance, setBalance] = useState(null);

  const customProvider = new ethers.providers.Web3Provider(provider,
"any");

  useEffect(() => {
    if (userInfo) {
      fetchBalance();
    }
  }, [userInfo]);

  const fetchBalance = async () => {
    const balanceResponse = await customProvider.getBalance(await
customProvider.getSigner().getAddress());

    setBalance(ethers.utils.formatEther(balanceResponse));
  }

  // Upon calling, the user will be prompted to login with their social
  account according to authType
  const handleLogin = async (authType) => {
    if (!userInfo) {
      await connect({
        socialType: authType,
        chain: Celo,
      });
    }
  };

  // The user will be required to click on an application-embedded
  confirmation popup, after which this transaction will be sent.
  const executeTx = async () => {
    const signer = customProvider.getSigner();
    console.log(await signer.getAddress())

    const tx= {
      to: "0x0000000000000000000000000000000000000000dEAD0",

```

```

    value: ethers.utils.parseEther("0.001")
  };

  const txResponse = await signer.sendTransaction(tx);
  const txReceipt = await txResponse.wait();

  notification.success({
    message: txReceipt.transactionHash
  })
};

return (
  <div className="App">
    <!userInfo ? (
      <div className="login-section">
        <button className="sign-button" onClick={() =>
handleLogin('google')}>Sign in with Google</button>
        <button className="sign-button" onClick={() =>
handleLogin('twitter')}>Sign in with Twitter</button>
      </div>
    ) : (
      <div className="profile-card">
        <h2>{userInfo.name}</h2>
        <div className="balance-section">
          <small>{balance} CELO</small>
          <button className="sign-message-button"
onClick={executeTx}>Execute Transaction</button>
          <button className="disconnect-button" onClick={() =>
disconnect()}>Logout</button>
        </div>
      </div>
    )}
  </div>
);
};

export default App;

```

As shown above, Particle Network's Wallet-as-a-Service can be plugged in and used in a way similar to any other standard wallet that exposes an EIP-1193 provider on Celo, enabling social logins in just a few lines of code.

Learn More

- Particle Dashboard
- Particle Documentation
- Celo Example Repository

index.md:

```
---
title: Rainbowkit
description: Overview of Rainbowkit
---
```

Rainbowkit (celo example)

Overview of Rainbowkit

```
---
```

Rainbowkit

RainbowKit is a React library that makes it easy to add wallet connection to your dapp. It's intuitive, responsive and customizable. And Supports some of the best wallets on Celo.

Installation

```
sh
npm install @rainbow-me/rainbowkit@2 viem@2 wagmi@2
```

Example Config

```
ts
import '@rainbow-me/rainbowkit/styles.css';
import {
  braveWallet,
  coinbaseWallet,
  injectedWallet,
  safeWallet,
  valoraWallet,
  walletConnectWallet,
} from '@rainbow-me/rainbowkit/wallets';

import {
  RainbowKitProvider,
  darkTheme,
  getDefaultConfig,
  lightTheme,
} from '@rainbow-me/rainbowkit';
import { QueryClient, QueryClientProvider } from '@tanstack/react-query';
import { WagmiProvider, http } from 'wagmi';
import { celo, celoAlfajores } from 'wagmi/chains';

import { WALLETCONNECTPROJECTID } from 'src/config/consts';

const queryClient = new QueryClient();

const config = getDefaultConfig({
  appName: 'Your App Name',
  projectId: WALLETCONNECTPROJECTID,
```

```

chains: [celo, celoAlfajores],
wallets: [
  {
    groupName: 'Recommended',
    wallets: [safeWallet, valoraWallet, braveWallet, coinbaseWallet],
  },
  {
    groupName: 'Fallbacks',
    wallets: [walletConnectWallet, injectedWallet],
  },
],
transports: {
  [celo.id]: http(),
  [celoAlfajores.id]: http(),
},
});
function App() {
  return (
    <WagmiProvider config={config}>
      <QueryClientProvider client={queryClient}>
        <RainbowKitProvider>
          Your APP Here
        </RainbowKitProvider>
      </QueryClientProvider>
    </WagmiProvider>
  )
}

```

celo-sdks.md:

```

---
title: Celo Libraries & SDKs
description: List of Celo libraries & SDKs
---

```

```

import PageRef from '@components/PageRef'
import Tabs from '@theme/Tabs';
import TabItem from '@theme/TabItem';

```

Celo Libraries & SDKs

Because Celo is compitable with Ethereum any ethereum package can be used with Celo. Packages with specific support for Celo include:

```

---
- viem
- ContractKit
- thirdweb SDK
- Rainbowkit
- web3

```

```
# development-chain.md:
```

```
---
```

```
title: Celo Local Development Chain with Protocol Contracts
description: How to set up a Celo development blockchain that includes
all of the core protocol contracts including identity and stability
contracts.
```

```
---
```

Local Development Chain with Protocol Contracts

How to set up a Celo development blockchain that includes all of the core protocol contracts, including identity and stability contracts.

```
---
```

What to expect

At the end of this tutorial, you will have a local Celo development blockchain running exposed at <http://localhost:7545> and will be able to connect to it like any other local node. We will also go over how to inspect the development blockchain using the Celo CLI tool and the ContractKit.

Running the development Celo blockchain is helpful because it greatly speeds up development time. You will start with 10 accounts pre-funded with CELD and all transactions on the network are virtually instant.

You can run the development Celo blockchain in several ways:

Use the celo-devchain NPM package

The easiest is to use a "pre-generated" devchain from the celo-devchain NPM package. For that all you have to do is:

```
sh
```

```
> npm install --save-dev @terminal-fi/celo-devchain
> npx celo-devchain --port 7545
```

or

```
> yarn add --dev @terminal-fi/celo-devchain
> yarn run celo-devchain --port 7545
```

Initialize your own devchain from the monorepo

If you prefer, you can initialize your own devchain and build it from scratch. To start, download the Celo monorepo [here](https://github.com/celo-org/celo-monorepo) or with the following command.

```
text
```

```
git clone https://github.com/celo-org/celo-monorepo.git
```

See this page for instructions on how to build the monorepo.

Once the monorepo is built, move into the contractkit directory.

```
cd packages/sdk/contractkit
```

From the contractkit directory, run

```
yarn test:reset && yarn test:livechain
```

This will start the development Celo blockchain. It will take at least a few minutes to start. The contract migrations will be printed in the terminal as they are deployed.

The process will finish and print Ganache started. Leave this terminal window open to leave the development chain running.

Interacting with the chain

Inspecting the chain

Now that we have a Celo development chain running, we probably want to know what accounts we have access to, how much cGLD and cUSD they have as well as the addresses of the deployed protocol contracts.

We can use the Celo CLI tool for this, or we can use the ContractKit npm package in a node script.

Celo CLI

You can install the CLI using npm by running `npm install -g @celo/celocli`. You can see the package details [here](#). Once it is installed, you should be able to access the tool from the terminal by running `$ celocli`. Try `$ celocli help`.

The CLI will connect to the node at `http://localhost:8545` by default. To connect to port 7545 you can run `$ celocli config:set -n http://localhost:7545`, and then check the connection by running `$ celocli node:get`.

You can see the accounts available on the Celo development chain by running `$ celocli account:list`. You should see something like:

text

```
[ '0x5409ED021D9299bf6814279A6A1411A7e866A631',  
  '0x6Ecbe1DB9EF729CBe972C83Fb886247691Fb6beb',  
  '0xE36Ea790bc9d7AB70C55260C66D52b1eca985f84',  
  '0xE834EC434DABA538cd1b9Fe1582052B880BD7e63',
```

```
'0x78dc5D2D739606d31509C31d654056A45185ECb6',  
'0xA8dDa8d7F5310E4A9E24F8eBA77E091Ac264f872',  
'0x06cEf8E666768cC40Cc78CF93d9611019dDcB628',  
'0x4404ac8bd8F9618D27Ad2f1485AA1B2cFD82482D',  
'0x7457d5E02197480Db681D3fdF256c7acA21bDc12',  
'0x91c987bf62D25945dB517BDAA840A6c661374402' ]
```

If you try to check the balance of the first account with `$ celocli account:balance 0x5409ED021D9299bf6814279A6A1411A7e866A631` you might encounter an error saying that the node is not currently synced. You can silence this by adding this environment variable to the terminal `$ export NOSYNCHECK=true`. Running the command again will print:

```
text  
All balances expressed in units of 10^-18.  
gold: 9.9999999693185872e+25  
lockedGold: 0  
usd: 5e+22  
total: 1.00004973043691287703791575e+26  
pending: 0
```

ContractKit + Node.js

You can also use the ContractKit to access the local node in a node.js script.

As an example, try running this script in an npm project with contractkit installed.

The linked gist is called `getInfo.js`. Run it with `$ node getInfo.js` This will print some of the Celo blockchain information.

You are now prepared to start developing, transacting and deploying contracts on your own Celo development blockchain!

Using Ethereum developer tools with Celo

You can connect the development chain to a tool like Remix to begin interacting with it. Keep in mind that these tools are built primarily for Ethereum development and are compatible with Celo because Celo is similar to Ethereum. The two blockchains have similar block architectures and both run the Ethereum Virtual Machine (EVM) for executing smart contracts.

The main difference between Celo and Ethereum that dapp developers need to keep in mind is that Celo has a slightly different transaction object than Ethereum. Celo requires one additional field in a transaction object, a `feeCurrency`. When Remix is connected to a locally running Celo node, the local node will fill these fields with default values (if the fields are empty). The node will sign the Celo transaction and broadcast it to the network.

mac.md:

title: Celo Local Development Environment Using Mac

description: How to set up a local development environment for Celo using Mac.

Using Mac

How to set up a local development environment for Celo using Mac.

:::tip

While many commands will be the same, you may need to follow the instructions for your specific OS when installing software. The Celo docs have some resources for Windows development.

:::

Before building on Celo, you need to set up a development environment to make sure you have the proper tools to build an application. This setup includes a combination of general development tools, Celo specific tools, and mobile development tools.

Web2 Prerequisites

Xcode

Xcode is Apple's integrated development environment for macOS, used to develop software for macOS, iOS, iPadOS, watchOS, and tvOS.

While Xcode falls under the Celo Prerequisites, Xcode takes a long time to download, which is why it is first on this list. It's best to start this download first, or even start downloading it before going to bed (depending on your internet connection).

Node, NPM and NVM

Node

Node is a JavaScript runtime that allows you to execute JS code outside a web browser. It is an open-source server environment and runs on various platforms.

NPM

npm stands for Node Package Manager and is the world's largest software registry. npm is what you'll use for installing published JS packages. npm comes bundled with Node, so you won't need to install it separately.

NVM

nvm stands for Node Version Manager. Different projects often require different versions of Node, and nvm makes it easy to switch between those versions. This is recommended if you plan to develop a lot of JS-based applications and npm also recommends nvm. For Windows users, there are nodist and nvm-windows.

Install Node and npm via nvm

```
shell
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.0/install.sh
| bash
```

Verify the installation by running:

```
shell
command -v nvm
```

which should output nvm.

You can now install Node with `nvm install <version>`.

:::info

You can switch node versions using

```
shell
nvm use <version>
```

:::

Homebrew

Homebrew is a way of managing packages on macOS.

Install Homebrew

```
shell
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Get the latest packages

```
shell
brew update
```

Yarn

yarn is a package manager similar to npm. For the most part, they do the same job and projects will let you know in the README or package.json which package manager they recommend using. Usually, you can use either or, but you should not use both in the same project. Running the Celo Development Blockchain requires yarn, so you can install it now.

Install yarn

Option 1: Since you already have npm installed, you can use it to install yarn

```
shell
npm install -g yarn
```

This will install the Yarn global binary.

:::info

The -g is short for --global. This flag means you are installing this package on your system and it will be accessible from the cli no matter what project you are in.

:::

Option 2: Install yarn with Homebrew

```
shell
Update brew if you haven't recently
$ brew update
```

```
Install yarn
$ brew install yarn
```

Expo

Expo is a framework and a platform for universal React applications. It lets you build mobile applications for both iOS and Android while using the same JavaScript/TypeScript codebase. This means that you don't need to be a "mobile developer" to build mobile apps!

Install Expo

Since you already setup npm, you can now install the expo-cli package.

```
shell
npm install -g expo-cli
```

Docker

Docker is an application focused on build and deployment tools. It allows developers to build and share containerized apps. Rather than making the developer manually install a bunch of packages, a dev can just download a Docker image and start it and all the application installation and setup will be taken care of for them.

Install Docker

You might need to install Docker Desktop for your OS.

Once you've done that, you can access docker via your cli.

```
shell
docker --version
```

Celo Prerequisites

Local Development Blockchain

See Local Development Chain w/ Protocol Contracts for how to get started with developing on a local test environment.

Celo CLI

The Celo CLI is a command-line tool for interacting with the Celo Protocol smart contracts. Some of the things you can do with it include looking at accounts, checking balances, and signing transactions.

Install Celo CLI

```
shell
npm install -g @celo/celocli
```

... lots of logs

```
+ @celo/celocli@x.x.xx
added x packages from x contributors in x.xs
```

```
:::info
```

Installation can take over a minute, so be patient depending on your internet connection.

```
:::
```

iOS

Xcode

Xcode allows you to build and deploy the Celo Wallet. If you do not have an iOS device, Xcode can emulate one. To install this, you need an Apple

Developer account (free), but you don't need to be part of the Apple Developer Program (costs money) to download Xcode.

Install Xcode

Download Xcode from the Apple Developer website. It is massive (10.6 GB), so we will start this first so it can be downloaded while we do everything else.

Once this is done downloading, install it.

Cocopods, Bundler

Navigate to the iOS directory of the mobile package (/celo-monorepo/packages/mobile/ios) and run the following to install cocoapods and bundler.

```
shell
install cocopods and bundler if you don't already have it
$ gem install cocoapods you might need to run with sudo if this fails
$ gem install bundler
```

```
download the project dependencies
$ bundle install
```

```
run inside mobile/ios
$ bundle exec pod install
```

If your machine does not recognize the gem command, you may need to download Ruby first.

Android

Java

Java allows you to build and deploy the mobile app to Android devices.

Install Java

Install by running the following:

```
shell
$ brew install cask
$ brew tap homebrew/cask-versions
$ brew install homebrew/cask-versions/adoptopenjdk8
```

Android Dev Tools

Install the Android SDK and platform-tools

```
$ brew install android-sdk
```

```
$ brew install android-platform-tools
```

Next, install Android Studio and add the Android NDK.

```
:::info
```

These paths may differ on your machine. You can find the path to the SDK and NDK via the Android Studio menu.

```
:::
```

```
shell
export ANDROIDHOME=/usr/local/share/android-sdk
export ANDROIDNDK=/usr/local/share/android-ndk
export ANDROIDSDKROOT=/usr/local/share/android-sdk
this is an optional gradle configuration that should make builds faster
export GRADLEOPTS='-Dorg.gradle.daemon=true -Dorg.gradle.parallel=true -
Dorg.gradle.jvmargs="-Xmx4096m -XX:+HeapDumpOnOutOfMemoryError"'
```

Then install the Android 29 platform

```
sdkmanager 'platforms;android-29
```

Android Emulator

For the same reason you installed the emulator iOS (you may or may not have an Android device), you can also install the Android emulator.

Install Emulator Manager

One Android emulator option is Genymotion.

```
brew install genymotion
```

Under OSX High Sierra and later, you'll get a message that you need to approve it in System Preferences > Security & Privacy > General.

Do that, and then repeat the line above.

Then make sure the ADB path is set correctly in Genymotion – set Preferences > ADB > Use custom Android SDK tools to /usr/local/share/android-sdk (same as \$ANDROIDHOME)

Extras

Solidity support for VSCode

Solidity support for VSCode (or your preferred IDE/text editor)

Celo smart contracts are written in Solidity. While Solidity syntax is very similar to JavaScript/TypeScript, there are some differences and JS syntax support will not like Solidity files. Installing an extension on your IDE will be a big help as you develop smart contracts.

overview.md:

title: Setup Environment

description: Set up your Celo development environment.

import PageRef from '@components/PageRef'

import Tabs from '@theme/Tabs';

import TabItem from '@theme/TabItem';

Setup Environment

Set up your Celo development environment.

- Using Mac
- Using Windows
- Using Replit
- Testnet Wallet

replit.md:

title: Deploy with Replit

description: How to deploy a smart contract to Celo testnet, mainnet, or a local network using Replit.

Deploy on Celo with Replit

How to deploy a smart contract to Celo testnet, mainnet, or a local network using Replit.

Introduction to Replit

Replit is a coding platform that lets you write code and host apps. It also has many educational features built-in, making it great for teachers and learners too. Every repl you create is a fully functional development and production environment. Hosting from your editor makes it easy to iterate quickly on your work, collaborate with others, and get feedback.

Replit added Solidity to its available programming languages, giving all of the features and functionality Replit provides to Web3 developers creating smart contracts. In this guide, you'll learn how to make use of these features to build and deploy a smart contract on Celo.

:::tip

To learn more about the features available to you as a smart contract developer with Replit, visit the [Replit documentation](#).

:::

Prerequisites

To deploy on Celo using Replit, you don't need any local environment. You should have a wallet available with testnet funds so that you can deploy and test transactions.

- Set up a Development Test Wallet
- Create a Replit account

If you are new to Replit, it will also help to review the Solidity announcement and Replit documentation.

Create a Repl

After creating your Replit account, your home screen will include a dashboard where you can view projects, create projects, manage your account, and do many other things. You can get started by creating a new Repl.

- Choose + Create Repl from the left panel or + in the top right corner of the screen
- Select the Solidity starter (beta) template and give your project a title
- Click + Create Repl to create your project

```
!create repl
```

Explore the Workspace

By creating a project, you now have a fully functional online IDE that allows you to edit, view, and deploy your smart contracts. It also creates a front-end for your smart contract that you can publish to an easily sharable URL. Read [Solidity on Replit](#) for an overview of some of these features.

```
!replit solidity template
```

:::tip

The README provides an overview of the workspace and gives details on how to make the most of your Replit experience. Read this to help get more familiar with your workspace.

:::

Deploy on Celo

Deploying on Celo with Replit is easy. If you don't have a wallet with funds, complete Set up a Development Test Wallet so that you are prepared to deploy and interact with your smart contract.

- Select Run to install all relevant packages, start up the contract deployment UI, and compile your contract.sol file

!compile

- Select Connect wallet, select your account, then choose Connect.

!connect

- Select the contract you would like to deploy from the dropdown (ex. MathTest, SimpleStorage).

- To deploy this contract, select Deploy and confirm the transaction from your wallet.

You can now interact with your contract using the provided user interface or from a sharable URL shown on the interface.

!deploy

Verify Contract Deployment

To view your deployed contract, copy your account address navigate to the Celo explorer, and search for your account (verify that you are searching the correct network).

- View the most recently deployed contract to find additional information on your deployment.

!block explorer

:::tip

Learn more about exploring the Celo network and smart contract details in BlockScout [here](#).

:::

Update your dApp

Your dApp is now live and can be shared with the world. You can use this workspace to edit and redeploy your dApp at any time. Here are a few tips you can use to help improve your dApp.

Front-end updates

Using tools > ui.jsx you can make changes to the front end of your dApp. To start, you can update the on-screen text from Ethereum to Celo and ETH to CELO.

- Use the files panel to navigate to tools > ui.jsx.

```
!ui
```

- Update Replit & Ethereum to read Replit & Celo on (will be around lines 446 and 528).
- Update ETH to CELO next to the wallet address (will be around line 701).

```
!ui updates
```

After making these basic changes, users will be able to determine that they are interacting with a Celo dApp. They can use the sharable URL to visit your dApp and make transactions from their wallet.

- Select Version Control from the left panel and click Connect to GitHub.

```
!connect to github
```

- Name your GitHub Repo, select your preferences, and click Create GitHub repository.
- After your GitHub Repo is created, click the Repo link to go to your new repository.

```
!github
```

You can now manage any additional changes using GitHub from the user interface provided by Replit.

Publish to Replit

Among many other features, Replit allows you to publish your projects to a personal profile. After publishing, projects will show up on your spotlight page for others to explore, interact with, clone, and collaborate.

- Select the project title at the top of the screen.
- Complete your project name and description and click Publish.

```
!publish
```

- Add a cover image, tags (try a Celo tag), and additional details for your project.

```
!cover image
```

- After publishing, navigate to the Spotlight page to view your project.

Your project is now available for other developers to view, share, fork, and comment on.

:::tip

Learn more at [Replit.com](https://replit.com) and in the Replit documentation. Share new projects using #Celo and search other #Celo tags to find the latest Replit dApps deployed on Celo.

:::

wallet.md:

title: Setup Testnet Wallets

description: How to create and fund testnet wallets to use for developing Celo dApps.

Testnet Wallets

How to create and fund testnet wallets to use for developing Celo dApps. Make sure to never use the same wallet for development, where you hold your real-life funds.

Getting Started

While developing and deploying dApps on Celo, it's helpful to have a wallet prepared with funds to pay for any transactions you make on the blockchain. These can be set up using either real or test funds, and this allows application developers and users to interact with Celo applications more easily.

:::tip

This guide will focus on funding an account on MetaMask with Celo (Alfajores testnet tokens). You can also use the Celo Extension Wallet if you prefer. Additionally, you can fund your wallet with real Celo if you would like to deploy to the Celo mainnet.

:::

Download a Wallet

When deploying a dApp, you will need to pay for transactions that write data to the Celo blockchain. This is done using a wallet funded with Celo on your preferred network.

- Download MetaMask for your browser
- If using MetaMask, Connect MetaMask to Celo (video) to access Celo network options from your wallet

Create your Account

You are now able to fund an account that you can access using your browser extension wallet. You can do this using accounts you create with the wallet or from the Celo CLI.

Using a wallet

MetaMask allows you to easily create and manage accounts. This provides you with a quick way to create and fund an account to use when interacting with your dApp.

- From your MetaMask extension, select Celo (Alfajores Tesnet)

```
<!-- !select alfajores network in MM -->
```

- Choose wallet Settings, select Create Account, name your account, and select Create

```
<!-- !select create account MM -->
```

- View your account details and copy your account address from the top center of the user interface

```
<!-- !new account MM -->
```

:::tip

Skip to this step to fund your account.

:::

Using Celo CLI

Install celocli by running

```
bash
npm install -g @celo/celocli
```

Create an Account

```
bash
celocli account:new
```

Creating an account will return its account details as shown below (details will be specific to your account).

```
shell
mnemonic: turtle cash neutral drift brisk young swallow raw payment drill
mail wear penalty vibrant entire adjust near chapter mistake size angry
planet slam demand
accountAddress: 0x5986ac413fA0C4A0379A674Cb986A59a962FC84e
```

```
privateKey:
8cab22c2bb08f0d20bd9e1109a156e87219d63a2c0b40b027483decf194bd787
publicKey:
024baaae61bab2a6e16ccb008c78dddb7132fc48d082e2a6166f8cc52d8d7a5289
address: 0x5986ac413fA0C4A0379A674Cb986A59a962FC84e
```

:::tip

Skip to this step to connect your account to your wallet.

:::

Import Account to Wallet (for Options 2 & 3)

If you created an account using option 2 or 3, you can now import these accounts to your wallet.

- Open your wallet browser extension from your browser
- Select Settings > Import Account
- Copy the private key from your local account, paste it into the window provided, and select Import

<!-- !import account to metamask -->

Fund your Account

No matter where you created your accounts, you can send them testnet funds using your Account Address and the testnet faucet.

- Navigate to the Alfajores Testnet Faucet
- Copy your address from your terminal or wallet
- Paste this address into the Testnet Faucet, complete the Captcha, and click Get Started

!alfajores faucet

Wait for the transaction to process to view the funds in your account.

windows.md:

title: Celo Local Environment using Windows
description: How to set up a local development environment for Celo using Windows.

Using Windows

How to set up a local development environment for Celo using Windows.

:::tip

Many popular tools and resources for blockchain development are written for developers working on UNIX machines. It is common for developers working on Windows to encounter errors that are not covered in the documentation and have no luck with Google. Fortunately, Microsoft makes it easy to run a UNIX machine directly from a Windows desktop with the Windows Subsystem for Linux.

:::

Getting set up with Windows

Open PowerShell as an administrator and run

text

```
Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Windows-Subsystem-Linux
```

Restart your computer when prompted.

Next, install a Linux distribution from the Microsoft Store. When developing this guide, we chose Ubuntu.

Set up your Linux distro by setting a username and password then update and upgrade the packages by running the following command in the terminal:

shell

```
sudo apt update && sudo apt upgrade
```

You can view the source documentation for setting up the Linux distro [here](#) and the Microsoft documentation for setting up the Windows Subsystem for Linux [here](#).

Set up the Linux Environment

Now that you have Linux installed, let's install nvm and yarn. Nvm \ (node version manager\) makes it easy to install and manage different versions of Node.js. The following instructions are from the celo-monorepo setup documentation for Linux.

Run the following commands in the Linux terminal.

bash

Installing Nvm

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.0/install.sh  
| bash  
source ~/.bashrc
```

Setting up the desired Nvm version

```
nvm install x
```

```
nvm alias default x
```

Running `$ node -v` in the terminal should print a node version if it is installed correctly.

Yarn is a package manager similar to npm. The celo-monorepo uses yarn to build and manage packages. Install yarn with the following command.

```
bash
Installing Yarn - https://yarnpkg.com/en/docs/install#debian-stable
curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg | sudo apt-key add -
echo "deb https://dl.yarnpkg.com/debian/ stable main" | sudo tee
/etc/apt/sources.list.d/yarn.list
sudo apt-get update && sudo apt-get install yarn
```

Test that yarn is installed by running `$ yarn --version`.

Developing with WSL

You can now start working on your projects in your Linux environment. Install the WSL VS Code extension for a seamless integration between VS Code and WSL.

Be aware that networking will be different depending on which version of WSL you are using. The details of managing network interfaces goes beyond the scope of this guide, but you can learn more [here](#). You are good to go! If you have any questions, join our [Discord server](#) and just ask.

Additional Resources

- [Windows Subsystem for Linux Installation Guide for Windows 10](#)
- [WSL: Ultimate Guide](#)

```
# index.md:
```

```
---
title: thirdweb SDK
description: Build web3 applications that can interact with your smart
contracts using our powerful SDKs.
---
```

thirdweb SDK

Create Application

thirdweb offers SDKs for a range of programming languages, such as React, React Native, TypeScript, Python, Go, and Unity.

1. In your CLI run the following command:

```
bash
npx thirdweb create --app
```

2. Input your preferences for the command line prompts:
 1. Give your project a name
 2. Choose your network: We will choose EVM for Moonbeam
 3. Choose your preferred framework: Next.js, CRA, Vite, React Native, Node.js, or Express
 4. Choose your preferred language: JavaScript or TypeScript
3. Use the React or TypeScript SDK to interact with your application's functions. See section on "interact with your contract"

Interact With a Contract

Initialize SDK On Celo

Wrap your application in the ThirdwebProvider component and change the activeChain to Celo

```
jsx
import { ThirdwebProvider } from "@thirdweb-dev/react";
import { Celo } from "@thirdweb-dev/chains";

const App = () => {
  return (
    <ThirdwebProvider activeChain={Celo}>
      <YourApp />
    </ThirdwebProvider>
  );
};
```

Get Contract

To connect to your contract, use the SDK's getContractmethod.

```
jsx
import { useContract } from "@thirdweb-dev/react";

function App() {
  const { contract, isLoading, error } =
    useContract("{{contractaddress}}");
}
```

Calling Contract Functions

- For extension based functions, use the built-in supported hooks. The following is an example using the NFTs extension to access a list of NFTs owned by an address- useOwnedNFTs

```
jsx
```

```

import { useOwnedNFTs, useContract, useAddress } from "@thirdweb-dev/react";

// Your smart contract address
const contractAddress = "{{contractaddress}}";

function App() {
  const address = useAddress();
  const { contract } = useContract(contractAddress);
  const { data, isLoading, error } = useOwnedNFTs(contract, address);
}

```

Full reference: <https://portal.thirdweb.com/react/react.usenft>

- Use the useContractRead hook to call any read functions on your contract by passing in the name of the function you want to use.

```

jsx
import { useContractRead, useContract } from "@thirdweb-dev/react";

// Your smart contract address
const contractAddress = "{{contractaddress}}";

function App() {
  const { contract } = useContract(contractAddress);
  const { data, isLoading, error } = useContractRead(contract,
"getName");
}

```

Full reference: <https://portal.thirdweb.com/react/react.usecontractread>

- Use the useContractWrite hook to call any write functions on your contract by passing in the name of the function you want to use.

```

jsx
import {
  useContractWrite,
  useContract,
  Web3Button,
} from "@thirdweb-dev/react";

// Your smart contract address
const contractAddress = "{{contractaddress}}";

function App() {
  const { contract } = useContract(contractAddress);
  const { mutateAsync, isLoading, error } = useContractWrite(
    contract,
    "setName"
  );

  return (

```



```

    <Web3Button
      contractAddress={contractAddress}
      // Calls the "setName" function on your smart contract with "My
Name" as the first argument
      action={() => mutateAsync({ args: ["My Name"] })}
    >
      Send Transaction
    </Web3Button>
  );
}

```

Full reference:
<https://portal.thirdweb.com/react/react.usecontractwrite>

Connect Wallet

Create a custom connect wallet experience by declaring supported wallets passed to your provider.

```

jsx
import {
  ThirdwebProvider,
  metamaskWallet,
  coinbaseWallet,
  walletConnectV1,
  walletConnect,
  safeWallet,
  paperWallet,
} from "@thirdweb-dev/react";

function MyApp() {
  return (
    <ThirdwebProvider
      supportedWallets={[
        metamaskWallet(),
        coinbaseWallet(),
        walletConnect({
          projectId: "YOURPROJECTID", // optional
        }),
        walletConnectV1(),
        safeWallet(),
        paperWallet({
          clientId: "YOURCLIENTID", // required
        }),
      ]}
      activeChain={Celo}
    >
      <App />
    </ThirdwebProvider>
  );
}

```

Add in a connect wallet button to prompt end-users to login with any of the above supported wallets.

```
jsx
import { ConnectWallet } from "@thirdweb-dev/react";

function App() {
  return <ConnectWallet />;
}
```

Full reference: <https://portal.thirdweb.com/react/connecting-wallets>

Deploy Application

To host your static web application on decentralized storage, run:

```
jsx
npx thirdweb deploy --app
```

By running this command, your application is built for production and stored using Storage. The built application is uploaded to IPFS, a decentralized storage network, and a unique URL is generated for your application.

This URL serves as a permanent hosting location for your application on the web.

If you have any further questions or encounter any issues during the process, please reach out to thirdweb support at support.thirdweb.com.

celo-explorer.md:

```
---
title: Verify with Celo Explorer
description: How to verify a Smart Contract on Celo using Celo Explorer
---
```

Verify Smart Contract using Celo Explorer

Verifying a smart contract allows developers to review your code from within the Celo Block Explorer.

- Navigate to the Code tab at the Explorer page for your contract's address
- Click Verify & Publish to enter the smart contract verification page

!github

- Upload your smart contract (example: HelloCelo.sol) and it's .json file (example: HelloCelo.json) found in build > contracts folder.

!github

- Click Verify & Publish
- Navigate to the Contract Address Details Page in the block explore to, use the Code, Read Contract, and Write Contract panels to view and interact with your deployed smart contract.

celoscan.md:

title: Verify with CeloScan

description: How to verify a Smart Contract on Celo using CeloScan

Verify Smart Contract using CeloScan

Verifying a smart contract allows developers to review your code from within the CeloScan Block Explorer

- Navigate to the Contract tab at the Explorer page for your contract's address
- Click Verify & Publish to enter the smart contract verification page

!github

- Select the Compile type, version and license.

!github

- Enter the Solidity Code of the contract along with the constructor arguments.
- Complete Captcha and click Verify and Publish.

!github

- If done correctly, you should see the following screen.

!github

foundry.md:

title: Verify with Foundry

description: How to verify a Smart Contract on Celo using Foundry

Verify Smart Contract using Foundry

Verifying a smart contract allows developers to review your code from within the CeloScan Block Explorer

Prerequisites

Project must be setup using Foundry

Add Celo configuration in foundry.toml

Add the below given configuration to the foundry.toml file at the root level of your project.

```
toml
[etherscan]
celo-alfajores = { key = "${CELOSCANAPIKEY}", url = "https://api-alfajores.celoscan.io/api"}
celo = { key = "${CELOSCANAPIKEY}", url = "https://api.celoscan.io/api"}
```

Make sure to also have Celo RPC configuration in foundry.toml file, here it is:

```
toml
[rpcendpoints]
celo-alfajores = "https://alfajores-forno.celo-testnet.org"
celo = "https://forno.celo.org"
```

Verifying Contracts

Use the following command

For Alfajores Testnet:

```
bash
forge verify-contract --chain-id celo-alfajores <contractaddress>
<contractlocation> --watch
```

For Celo Mainnet:

```
bash
forge verify-contract --chain-id celo <contractaddress>
<contractlocation> --watch
```

hardhat.md:

```
---
title: Verify with Hardhat
description: How to verify a Smart Contract on Celo using Hardhat
---
```

Verify Smart Contract using Hardhat

Verifying a smart contract allows developers to review your code from within the CeloScan Block Explorer

:::tip

If you use Celo Composer all the configuration is done for you out of the box, all you need is the CeloScan API keys!

:::

Prerequisites

Before the installation steps you need to have your hardhat project initialized using the command

```
bash
npx hardhat init
```

Make sure to have dependencies installed and the hardhat config file is importing @nomicfoundation/hardhat-toolbox

Hardhat Configuration

Add the following configuration to the config object in hardhat.config.js.

```
js
  networks: {
    alfajores: {
      // can be replaced with the RPC url of your choice.
      url: "https://alfajores-forno.celo-testnet.org",
      accounts: [
        "<YOURPRIVATEKEY>"
      ],
    },
    celo: {
      url: "https://forno.celo.org",
      accounts: [
        "<YOURPRIVATEKEY>"
      ],
    }
  },
  etherscan: {
    apiKey: {
      alfajores: "<CELOSCANAPIKEY>",
      celo: "<CELOSCANAPIKEY>"
    },
    customChains: [
      {
        network: "alfajores",
        chainId: 44787,
        urls: {
          apiURL: "https://api-alfajores.celoscan.io/api",
          browserURL: "https://alfajores.celoscan.io",
        },
      },
      {
```

```

        network: "celo",
        chainId: 42220,
        urls: {
            apiURL: "https://api.celoscan.io/api",
            browserURL: "https://celoscan.io/",
        },
    },
]
},

```

Verifying Contracts

Use the following command (Make sure your contracts are compiled before verification)

Alfajores Testnet

```

bash
npx hardhat verify [CONTRACTADDRESS] [...CONSTRUCTORARGS] --network
alfajores

```

Celo Mainnet

```

bash
npx hardhat verify [CONTRACTADDRESS] [CONSTRUCTORARGS] --network celo

```

index.md:

```

---
title: Verify Contracts Deployed on Celo
description: How to verify contract deployed on Celo.
---

```

```
import PageRef from '@components/PageRef'
```

Verify Contract Deployed on Celo

How to verify contracts deployed on Celo.

```
---
```

The fastest way to verify on Celo is to use hardhat-celo. Alternatively, you can use the Celo Explorer and CeloScan to verify contracts using a user interface.

Verify Contracts on Celo

- Using Celo Explorer
- Using Remix
- Using CeloScan

- Using Hardhat

remix.md:

title: Verify with Remix

description: How to verify a Smart Contract on Celo using Remix

Verify Smart Contract using Remix

- Verifying a smart contract allows anyone to review your code from within the Celo Block Explorer. This can be done using the Remix Sourcify Plugin.

- Navigate back to the Remix IDE, select Plugin Manager from the left side menu.

- Search for Sourcify, click Activate, and open the newly installed Sourcify Plugin.

- Choose Verifier, select the dropdown menu, and choose the location for your deployed contract (example Celo (Alfajores)).

- Paste your contract address into the Contract Address field and select Verify.

:::tip

The source code of the contract that you are verifying will need to be in Remix. Contracts deployed with Hardhat, and other tools can also be verified using the Remix Sourcify plugin, but you will need to copy your contract source code into Remix first.

:::

!github

- Navigate to the Contract Address Details Page in the block explore to, use the Code, Read Contract, and Write Contract panels to view and interact with your deployed smart contract.

index.md:

title: Viem

description: Using Viem with Celo

Viem

Viem is full featured lightweight javascript library for interacting with EVM chains with first class support for Celo.

Viem is used by Wagmi and Rainbowkit.

The Viem docs have excellent examples of how to use it in your project.

With Celo

The TLDR is that passing a celo chain from viem/chains into the config of `createWalletClient` will enable any function that signs a transaction including `sendTransaction` and `writeContract` to accept `feeCurrency` in its parameters object. Don't care about `feeCurrency`? Leave it out to pay with CELO.

```
ts
// see viem docs for more info on setup

// Create a wallet client that will sign the transaction
const client = createWalletClient({
  account,
  // Passing chain is how viem knows to try serializing tx as cip42.
  chain: celoAlfajores,
  transport: http(),
});

client.writeContract({
  abi: ANYCONTRACTABI,
  address: ANYCONTRACTABIADDRESS,
  functionName: "contractMethod",
  args: [to, parseEther(value)],
  // set the fee currency on the contract write call
  feeCurrency: FEECURRENCIESALFAJORES["cusd"],
});
```

Gas Price

When paying for transaction with an alternate `feeCurrency` token it is important to know the price of gas denominated in that token. As such Celo nodes accept an optional param of the address of the token for the `ethgasPrice` call. Therefore rather than use viem's `publicClient.getGasPrice()` function you should fetch it like the example.

```
ts
async function getGasPrice(client, feeCurrencyAddress?: Address) {
  const priceHex = await client.request({
    method: "ethgasPrice",
    params: [feeCurrencyAddress],
  });
  return hexToBigInt(priceHex);
}

tx.maxFeePerGas = await getGasPrice(client, tx.feeCurrency);
```

For an interactive example of using viem with Celo's Fee Abstraction feature see our demo


```
# hello-celo.md:
```

```
---
```

```
title: Sending CELO & Mento Stable Assets
```

```
---
```

Sending CELO & Mento Stable Assets

How to connect to the Celo test network and transfer tokens using ContractKit.

```
---
```

Hello Celo: sending value with Celo

In this guide we are going to write a Node.js script to introduce some of the basic concepts that are important to understand how Celo works. This will get us started with connecting to the Celo network and learning how to develop more advanced applications.

:::info

We assume you already have Node.js and NPM installed on your computer.

:::

Learning Objectives

At the end of this guide, you will be able to:

- Connect to the Celo test network, called Alfajores
- Get test CELO, Mento cUSD (USD) and Mento cEUR (EUR) from the faucet
- Read account and contract information from the test network
- Transferring CELO, Mento cUSD and Mento cEUR on the test network

Getting Started

To start, clone this GitHub repo. This is a Node.js application.

```
git clone https://github.com/critesjosh/helloCelo.git
```

We will be using the Celo ContractKit SDK to interact with the Celo test network (Alfajores). Let's install it. It is already defined in the package.json, so we can get it with

```
cd helloCelo
npm install
```

Importing ContractKit

We will be writing our Node.js app in the helloCelo.js file.

Import the contract kit into our script with

```
javascript title="helloCelo.js"
// 1. Import web3 and contractkit
const Web3 = require("web3");
const ContractKit = require("@celo/contractkit");
```

Now we can use the ContractKit to connect to the test network.

```
javascript title="helloCelo.js"
// 2. Init a new kit, connected to the alfajores testnet
const web3 = new Web3("https://alfajores-forno.celo-testnet.org");
const kit = ContractKit.newKitFromWeb3(web3);
```

:::info

At any point in the file you can console.log() variables to print their output when you run the script.

:::

Reading Alfajores

ContractKit contains a contracts property that we can use to access certain information about deployed Celo contracts.

:::info

The Celo blockchain has two native assets, CELO \ (CELO\) and the Mento cUSD. Both of these assets implement the ERC20 token standard from Ethereum. The CELO asset is managed by the CELO smart contract and Mento cUSD is managed by the cUSD contract. We can access the CELO contract via the SDK with kit.contracts.getGoldToken() and the Mento cUSD contract with kit.contracts.getStableToken(). These functions return promises, so we have to wait for them to resolve before we can interact with the token contracts. If you are unfamiliar with Promises in Javascript, check out this documentation. Promises are a common tool in blockchain development. In this guide, we use the async/await syntax for promises.

:::

Let's read some token balances from the blockchain. Add the following line in the readAccount() function.

```
javascript title="helloCelo.js"
// 3. Get the token contract wrappers
let celotoken = await kit.contracts.getGoldToken();
let cUSDtoken = await kit.contracts.getStableToken();
let cEURtoken = await kit.contracts.getStableToken("cEUR");
```

We can get the CELO balance of an account using the token wrappers like `goldtoken.balanceOf(address)`. Let's check the balance of this address `'0xD86518b29BB52a5DAC5991eACf09481CE4B0710d'`.

```
javascript title="helloCelo.js"
// 4. Address to look up
let anAddress = "0xD86518b29BB52a5DAC5991eACf09481CE4B0710d";

// 5. Get token balances
let celoBalance = await celotoken.balanceOf(anAddress);
let cUSDBalance = await cUSDtoken.balanceOf(anAddress);
let cEURBalance = await cEURtoken.balanceOf(anAddress);

// Print balances
console.log(`${anAddress} CELO balance: ${celoBalance.toString()});
console.log(`${anAddress} Mento cUSD balance: ${cUSDBalance.toString()});
console.log(`${anAddress} Mento cEUR balance: ${cEURBalance.toString()});
```

The `balanceOf(address)` function also returns a Promise, so we wait for the promise to resolve then we print the result.

To view the balances, run the script from the terminal with

```
node helloCelo.js
```

```
:::info
```

Note that the `balanceOf()` function returns objects with type `BigNumber` because balances are represented in Celo as a 256 bit unsigned integer, and JavaScript's number type cannot safely handle numbers of that size. Note also that the balance values are reported in units of CELO Wei, where one CELO = 10^{18} CELO Wei.

```
:::
```

Reading all account balances is a powerful feature of blockchains. Next, let's see how we can send value to each other on the testnet.

In order to do transfers (aka transactions), we need to:

1. Create an account \ (by creating a private key\)
2. Fund it with test CELO and Mento cUSDs
3. Sign and send transactions to the network

Accounts

We are accessing the Celo network via a remote node via HTTP requests at `'https://alfajores-forno.celo-testnet.org'`.

:::info

Don't worry about what this means right now, just understand that it is easier to get started using Celo by accessing remote nodes, rather than running them locally on your machine. You can read more about the details of the Celo network [here](#).

:::

Because we are accessing the network remotely, we need to generate an account to sign transactions and fund that account with test CELO.

There is a short script in `getAccount.js` to either get a Celo account from a mnemonic in the `.secret` file, or create a random account if the file is empty. In the script, we use `web3.js` to create a new private key/account pair. `Web3.js` is a popular javascript library for handling Ethereum related functionality. Celo is a cousin of Ethereum, so this library works well for generating Celo accounts.

We can now use this account to get account information \ (ie the private key and account address\) and to send transactions from `account.address`. Add the following code to read the account balance. Continue adding to `helloCelo.js`.

```
javascript title="helloCelo.js"
//
// Create an Account
//

// 6. Import the getAccount function
const getAccount = require("./getAccount").getAccount;

async function getBalances() {
  // 7. Get your account
  let account = await getAccount();

  // 8. Get the token contract wrappers
  let celoToken = await kit.contracts.getGoldToken();
  let cUSDToken = await kit.contracts.getStableToken();
  let cEURToken = await kit.contracts.getStableToken("cEUR");

  // 9. Get your token balances
  let celoBalance = await celoToken.balanceOf(account.address);
  let cUSDBalance = await cUSDToken.balanceOf(account.address);
  let cEURBalance = await cEURToken.balanceOf(account.address);

  // Print your account info
  console.log(Your account address: ${account.address});
  console.log(Your account CELO balance: ${celoBalance.toString()});
  console.log(Your account Mento cUSD balance:
    ${cUSDBalance.toString()});
  console.log(Your account Mento cEUR balance:
    ${cEURBalance.toString()});
}
```

Run this script again with `node helloCelo.js`. This will print 0, as we have not funded the associated account yet.

Using the faucet

We can get free test CELO and Mento cUSDs on the test network for development via the Celo Alfajores faucet.

Copy your randomly generated account address from the console output mentioned above, and paste it into the faucet.

Once your account has been funded, run `$ node helloCelo.js` again to see your updated balance.

Sending Value

We have an account with CELO and Mento cUSD in it, now how do we send tokens to another account? Remember the token wrappers we used to read account balances earlier? We can use the same wrappers to send tokens, you just need to add the private key associated with your account to `ContractKit` (see line 10).

The token wrappers have a method called `transfer(address, amount)` that allows you to send value to the specified address (line 14).

You need to `send()` the transaction to the network after you construct it. The `send()` methods accepts an option that allows you to specify the `feeCurrency`, which allows the sender to pay transaction fees in CELO or Mento cUSD. The default `feeCurrency` is CELO. In the following example, let's pay transaction fees in CELO when we transfer CELO and pay with Mento cUSD when we transfer Mento cUSD.

The `send()` method returns a transaction object. We will wait for the transaction receipt (which will be returned when the transaction has been included in the blockchain) and print it when we get it. This receipt contains information about the transaction.

After we read the receipt, we check the balance of our account again, using the `balanceOf()` function. The logs print our updated balance!

You may notice that the account balance is a bit smaller than the amount of tokens that we sent. This is because you have to pay for every update to the network.

Add the following code to the `send()` function in `helloCelo.js` to send a transaction.

```
javascript title="helloCelo.js"
async function send() {
  // 10. Get your account
  let account = await getAccount();
```

```

// 11. Add your account to ContractKit to sign transactions
kit.connection.addAccount(account.privateKey);

// 12. Specify recipient Address
let anAddress = "0xD86518b29BB52a5DAC5991eACf09481CE4B0710d";

// 13. Specify an amount to send
let amount = 100000;

// 14. Get the token contract wrappers
let celotoken = await kit.contracts.getGoldToken();
let cUSDtoken = await kit.contracts.getStableToken();
let cEURtoken = await kit.contracts.getStableToken("cEUR");

// 15. Transfer CELO and cUSD from your account to anAddress
// Optional: specify the feeCurrency, default feeCurrency is CELO
let celotx = await celotoken
  .transfer(anAddress, amount)
  .send({ from: account.address });
let cUSDtx = await cUSDtoken
  .transfer(anAddress, amount)
  .send({ from: account.address, feeCurrency: cUSDtoken.address });
let cEURtx = await cEURtoken
  .transfer(anAddress, amount)
  .send({ from: account.address });

// 16. Wait for the transactions to be processed
let celoReceipt = await celotx.waitReceipt();
let cUSDReceipt = await cUSDtx.waitReceipt();
let cEURReceipt = await cEURtx.waitReceipt();

// 17. Print receipts
console.log("CELO Transaction receipt: %o", celoReceipt);
console.log("Mento cUSD Transaction receipt: %o", cUSDReceipt);
console.log("Mento cEUR Transaction receipt: %o", cEURReceipt);

// 18. Get your new balances
let celoBalance = await celotoken.balanceOf(account.address);
let cUSDBalance = await cUSDtoken.balanceOf(account.address);
let cEURBalance = await cEURtoken.balanceOf(account.address);

// 19. Print new balance
console.log(Your new account CELO balance: ${celoBalance.toString()});
console.log(Your new account Mento cUSD balance:
${cUSDBalance.toString()});
console.log(Your new account Mento cEUR balance:
${cEURBalance.toString()});
}

```

Run `$ node helloCelo.js` again to send the transactions and see the printed output in the console.

Connecting to a Ledger Device from a Web Application

The above instructions apply to building NodeJS applications. If you want to build an integration with a web application, you can still use the ContractKit by following slightly modified instructions.

The following code examples are typescript so should be stored in a .tsc file, you will also need to install typescript and then compile your typescript to javascript with `npx tsc` before you can run the code with node.

```
npm install --save-dev typescript
npm install web3 @celo/contractkit @celo/wallet-ledger @ledgerhq/hw-app-eth @ledgerhq/hw-transport-u2f @ledgerhq/hw-transport-webusb
```

Then, you can create a new instance of the ContractKit with the following code:

```
javascript
import { ContractKit, newKitFromWeb3 } from "@celo/contractkit";
import { newLedgerWalletWithSetup } from "@celo/wallet-ledger";
import Eth from "@ledgerhq/hw-app-eth";
import TransportU2F from "@ledgerhq/hw-transport-u2f";
import TransportUSB from "@ledgerhq/hw-transport-webusb";
import Web3 from "web3";

// Handle getting the Celo Ledger transport.
const getCeloLedgerTransport = () => {
  if (window.USB) {
    return TransportUSB.create();
  } else if (window.u2f) {
    return TransportU2F.create();
  }

  throw new Error(
    "Ledger Transport not support, please use Chrome, Firefox, Brave, Opera or Edge."
  );
};

// Handle creating a new Celo ContractKit
const getContractKit = async () => {
  // Create a Web3 provider by passing in the testnet/mainnet URL
  const web3 = new Web3("https://alfajores-forno.celo-testnet.org");

  // Get the appropriate Ledger Transport
  const transport = await getCeloLedgerTransport();

  // Create a new instance of the ETH Ledger Wallet library
  const eth = new Eth(transport);

  // Use the Celo Ledger Wallet setup util
  const wallet = await newLedgerWalletWithSetup(eth.transport);
```

```

// Instantiate the ContractKit
const kit: ContractKit = newKitFromWeb3(web3, wallet);

return kit;
};

```

Once you have successfully created the ContractKit, you can use the various Celo contracts to sign transactions with a connected Ledger device. For example, here's how to transfer gold tokens (just like above in the NodeJS example):

```

javascript
// Use the gold token contract to transfer tokens
const transfer = async (from, to, amount) => {
  const celoTokenContract = await kit.contracts.getGoldToken();
  const tx = await celoTokenContract.transfer(to, amount).send({ from });
  const receipt = await tx.waitReceipt();
  console.log("Transaction Receipt: ", receipt);
};

```

This is the basic setup to integrate the Celo Ledger App with a web application. You can also view the Celo Ledger App example codebase for some other examples of connecting to a Ledger Device from a web application.

Wrapping Up

Congratulations! You have accomplished a lot in this short introduction to developing on Celo.

We covered:

- Installing and setting up ContractKit
- Connecting to the Celo Alfajores network
- Getting the CELO contract wrapper
- Reading account balances using the CELO wrapper
- Generating a new account in Celo
- Funding an account using the Celo Alfajores Faucet
- Sending CELO

no-code-erc20.md:

title: Deploy & Mint a Token

description: How to deploy an ERC20 token contract to Celo.

Deploy an ERC20 token to Celo

How to deploy a token contract that use the ERC20 token standard to Celo without writing code.

Getting Started

In this tutorial, we will go over how to deploy an ERC20 token contract. The process is very similar for deploying other tokens as well.

1. Install Metamask.
2. Add the Celo network to Metamask. We suggest adding the Alfajores testnet to Metamask as well, so you can test contract deployments before deploying to mainnet.
3. Add a small amount of CELO to your Metamask account. In this example, we will deploy to the Alfajores testnet, so we need Alfajores CELO, which you can get from the faucet [here](#).
4. Go to the Open Zeppelin Contracts Wizard.
5. Select ERC20 as the type of contract that you would like to deploy.

!erc20 empty settings.png

6. Name your token. We are calling our token "ProsperityToken" in this example.
7. Select the features for your token. We are making ProsperityToken mintable, burnable and enabling snapshots, so the token may be used for governance. We are also making the contract Ownable, so the deployer of the contract can mint new tokens and distribute them as desired. Ideally, the owner account will be a multi-signature contract, so no single person has control over this token contract.

If you want the block explorer to recognize your token then leave "Upgradeability" unchecked and do not select one of the two radio options below it. Selecting one of these options will prevent the Celo block explorer from recognizing your deployed contract as a token. If you want upgradability and do not care about the block explorer, feel free to make your token contract upgradable.

!erc20 filled settings.png

8. Open your contract in Remix by clicking "Open in Remix". Remix will pop open with the contract code already filled in.
9. Click the big blue button that says "Compile contract-xxxxx.sol". The contract should compile without error.

!remix compile erc20.png

10. Click the Ethereum logo in the left sidebar. This will bring up a new interface for deploying the contract.

!remix deploy erc20.png

11. In the "Environment" section on the top left, select "Injected Web3". This will connect Remix to Metamask. Now clicking the "deploy" button will deploy the contracts to whichever network Metamask is connected to.

You should see a small textbox indicating that Remix is connected to a custom network. The Alfajores network id is 44787.

```
!select injected web3.png
```

12. In the Contract dropdown, select the contract that you want to deploy. In this example, it is called ProsperityToken.

```
!select prosperitytoken erc20 contract.png
```

13. Click Deploy. Metamask should pop open.

```
!deploy prosperity token erc20.png
```

14. Click Confirm. Once the transaction confirms (less than 5 seconds), a contract interface will appear in the bottom left, and transaction details will appear in the console at the bottom.

```
!deployed prosperity token.png
```

That's it! We now have ProsperityToken deployed on Alfajores with the Metamask account as the contract owner.

You can see the contract information on the Alfajores block explorer. Copy and paste the contract address or deployment transaction hash from the console output and paste it into the block explorer search bar or look up the deployment transaction info in the Metamask activity.

Deploying your token on the Mainnet

When you're ready to deploy your token to the Celo Mainnet make sure to change the network of your connected wallet from Alfajores to the Celo Mainnet. Once you have done this you can simply redeploy the contract (you will not need to recompile it).

Note: When deploying to the Mainnet you will need to use real Celo to pay the gas fee (as opposed to using the faucet on the testnet). As of December 2021 this cost is less than \$0.01 US. You can learn how to get Celo [here](#).

Verify

If you are unable to view your token on the block explorer, you may need to Verify it first. If you are able to see your token, you may skip this section.

Verifying your contract with Remix is straight-forward and allows anyone to read and interact with the contract on the block explorer. You can read more about verifying a contract with Remix on this [page](#).

You can find my example contract [here](#).

Let me know what you end up building and reach out if you have any questions, @critesjosh\ on Twitter or joshc#0001 on Discord. Join the Celo discord at <https://chat.celo.org>.

no-code-erc721.md:

title: "Deploy an NFT to Celo"

description: How to deploy ERC721 tokens (NFTs) on the Celo network using autogenerated code.

Deploy an NFT to Celo

How to deploy ERC721 tokens (NFTs) on the Celo network using autogenerated code.

Getting Started

In this example, we will be using IPFS for off-chain storage, but you can use whatever off-chain storage mechanism you want.

Set up your wallet

1. Install Metamask.
2. Add the Celo network to Metamask. We suggest adding the Alfajores testnet to Metamask as well, so you can test contract deployments before deploying to mainnet.
3. Add a small amount of CEL0 to your Metamask account. In this example, we will deploy to the Alfajores testnet, so we need Alfajores CEL0, which you can get from the faucet [here](#).

Prepare the NFT metadata

4. Go to <https://app.pinata.cloud/> and sign up for an account if you don't have one already. Pinata is a service that allows you to easily upload files to IPFS.
5. Upload your NFT images to IPFS. Because storing data on a blockchain can be expensive, NFTs often reference off-chain data. In this example, We are creating a set of NFTs that reference pictures of trees. We uploaded all of the images of trees to IPFS individually. The names of the images correspond to the token ID. This isn't necessary, we just did it for convenience. Notice that each image has a corresponding CID hash, this is the unique identifier for that file or folder. !pinata upload image list.png
6. Once all of your images have been uploaded, you will need to prepare the token metadata in a new folder.
 1. We created a folder called "prosper factory metadata". You can view the contents of the folder [here](#). The folder contains 14 files, numbered 0-13. The names of these files are important. These file names correspond to the token ID of each NFT that will be created by the contract. Make

sure that there are no extensions (.txt, .json, .jpeg, .png) on your files. !ipfs folder contents.png

2. Click on one of the files. The files contain the NFT metadata. In this simple example, the metadata only contains a reference to the unique tree image. You can view the image in a browser that supports IPFS (we are using Brave) here. Each file should have a unique image reference. !ipfs image metadata.png You will need to create a similarly structured folder containing metadata for all of the NFTs that you want to create. 7. Upload the folder containing all of the token metadata to IPFS. This will make your NFT data publicly available. We called ours "prosper factory metadata". Note the CID of this folder. We will need it shortly. !pinata prosper factory metadata folder.png

Design and Deploy the Smart Contracts

8. Go to <https://docs.openzeppelin.com/contracts/5.x/wizard>

9. Select ERC721 as your token choice.

10. Enter your token information.

1. We are calling our token the ProsperityFactory, symbol PF.

2. We entered the IPFS CID of our token metadata folder (prosper factory metadata) in the "Base URI" field. Be sure to add a trailing "/" to the base URI, the token ID will be appended to the end of the base URI to get the IPFS metadata file. So the complete Base URI for our NFT contract is `ipfs://QmdmA3gwGukA8QDPH7YpqlWAoVfX82nx7SaXFvh1T7UmvZ/`. Again, you can view the folder here.

3. We made the token mintable and it will automatically increment the token IDs as the tokens are minted. The counter starts at 0 and adds 1 to each successive token. It is important that the file contents of the IPFS metadata folder are labeled accordingly (ie. 0-13) and correspond to the token IDs.

4. The contract is also marked Ownable, meaning that only the owner of the contract (which is initially set to the account that deploys the contract) can mint new NFTs. !erc721 filled settings.png

11. Click "Open in Remix". Remix will pop open with your contract code already filled in.

12. Click the big blue button on the left side of Remix that says "Compile contract-xxxx.sol". !remix compile erc721.png

13. Once the contract is compiled, click the Ethereum logo on the left side of the window. A new sidebar will appear. !remix deploy page erc721.png

14. In the "Environment" dropdown, select "Injected Web3". This will connect Remix and Metamask. Make sure that Metamask is connected to the correct network. In this example, We are deploying to the Alfajores testnet, so we see a textbox below the dropdown that says Custom (44787) network. 44787 is the network id for Alfajores. !select injected web3.png

15. Select the contract that you want to deploy. We titled the contract the ProsperityFactory. !select erc721 contract.png

16. Click Deploy. Metamask will pop up asking you to confirm the transaction. !remix deploy 721 tx.png

17. Once the contract is deployed, Remix will show a newly deployed contract on the bottom left corner of the window. Expand the ProsperityFactory dropdown to see all of the contract functions. You can see the deployed ProsperityFactory NFT contract here. !remix 721 contract interface.png

18. Let's mint the first NFT. To do that we will call the `safeMint` function. The `safeMint` function needs an account address as an input, so it knows who to mint the token to. I'll just enter the first Metamask address and click the orange button. Metamask will pop up, confirm the transaction. When the transaction is confirmed, this will have minted the first NFT, with token ID 0. `!safeMint.png`

19. Check the token metadata. You can verify that the token was minted by calling the `"tokenURI"` function with the expected token ID. We call the contract with token ID 0 and it returns an IPFS reference. `!721 read token uri.png`

20. Navigating to this IPFS reference will show the token metadata for that NFT. `!ipfs token metadata.png`

21. Going to that IPFS reference will show the image for the token.

We went through the same process and minted all 14 NFTs from this contract.

That's it! You now know how to create your own NFT contract with corresponding metadata!

Let me know what you end up building and reach out if you have any questions, @critesjosh\ on Twitter or joshc#0001 on Discord. Join the Celo discord at <https://chat.celo.org>.

using-js-keystores.md:

title: Using Keystores Library for Local Key Management

description: Introduction to the keystores library and how to use it for local key management.

Keystores Library

Introduction to the keystores library and how to use it for local key management.

Getting Started

This is a JavaScript library that provides functions for creating and interacting with encrypted keystores for private key management. To do this, this library wraps the existing `ethereumjs-wallet` library, which is a standard library for managing keystores according to the Web3 Secret Storage Definition. As specified, secrets are encrypted using the Scrypt KDF (Key Derivation Function); in this case, the private key is encrypted with a passphrase (that should be kept secret) and can be decrypted later by the same phrase. Note that a keystore generated for the same (private key, passphrase) multiple times will not yield the same output due to how the KDF works. Keystore files generated by a geth node can be decrypted and accessed with this library, and vice versa.

Note that keystore files generated by this library do not contain BLS public keys, meaning that these should not be used for validator signer keys used in consensus.

The components of the library are roughly as follows:

- KeystoreBase which wraps the functionality of ethereumjs-wallet and exposes functions to:
 - import PKs (into encrypted keystores)
 - decrypt and get a PK from an encrypted keystore
 - change the passphrase on a keystore
- FileKeystore, InMemoryKeystore which specify the IO in addition to the above base class
- KeystoreWalletWrapper: (not stable; likely to structurally change) a very simple wrapper for a Keystore and LocalWallet, which allows a user to decrypt a keystore and pass the key to the LocalWallet in order to sign transactions.

Usage

:::warning

For accounts containing significant funds or otherwise requiring a high degree of security, we do not recommend this keystore library! This is only for managing keys for low-risk hot wallets and signers.

For more stringent security requirements, check out the guide to [Choosing a Wallet](#).

:::

Depending on your use case, you can either interact directly with the FileKeystore (purely for creating and interacting with keystore files, importing or accessing private keys) or else use the KeystoreWalletWrapper (combines the keystore functionality with convenient access to the LocalWallet for signing transactions).

Using the FileKeystore

Create new keystore and import private key

This snippet will create a keystore directory in the parentDirectory and create an encrypted file in the keystore directory containing the private key. Note that you can only create a new encrypted file for a private key if there is not already an existing file for that private key. If it already exists, you can change the passphrase (see below), but you may not have multiple files for the same private key in the same keystore directory.

```
js
import readline from "readline";
import { FileKeystore } from "@celo/keystores";

// This is the directory that will contain a "keystore" directory
```

```

const parentDirectory = "<INSERTPATHHERE>";
// This creates a "keystore" directory if one does not already exist in
the parentDirectory
const keystore = new FileKeystore(parentDirectory);

// Prompt to enter private key and passphrase on the command line
let rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});
const privateKey: string = await new Promise((resolve) => {
  rl.question("Enter private key:", (answer) => {
    resolve(answer);
  })
});
const passphrase: string = await new Promise((resolve) => {
  rl.question("Enter secret passphrase:", (answer) => {
    rl.close();
    resolve(answer);
  })
});
// Import private key into the keystore, which is then stored as an
encrypted file
// Should create a file with a name like UTC-<DATETIME>-<ACCOUNTADDRESS>
await keystore.importPrivateKey(privateKey, passphrase);
// Retrieve all addresses contained in the keystore
console.log("Addresses in keystore: ", await
keystore.listKeystoreAddresses());

```

Accessing an existing keystore file

```

js
// Keystore already exists
const parentDirectory = '<INSERTPATHHERE>'
const keystore = new FileKeystore(parentDirectory)
const address = '<YOURADDRESSHERE>'
const oldPassphrase = '<OLDPASSPHRASE>'

// Decrypt file and retrieve private key
await keystore.getPrivateKey(address, oldPassphrase)

// Change the passphrase encrypting the file
const newPassphrase = '<NEWPASSPHRASE>'
await keystore.changeKeystorePassphrase(address, oldPassphrase,
newPassphrase)

// Decrypt file and retrieve private key using new passphrase
console.log(await keystore.getPrivateKey(address, newPassphrase))

```

Remove (delete) a keystore file for a particular address

```

js

```

```

const parentDirectory = "<INSERTPATHHERE>";
const keystore = new FileKeystore(parentDirectory);
const address = "<YOURADDRESSHERE>";

// When you know the address
// Get the filename (keystore name)
const keystoreName = await keystore.getKeystoreName(address);
await keystore.removeKeystore(keystoreName);

// Alternatively, you can do this by passing in the filename directly
keystore.removeKeystore("<KEYSTOREFILENAMEDELETE>");

```

Using the KeystoreWalletWrapper

This example will instantiate a KeystoreWalletWrapper, import a private key, and use the inner LocalWallet within the wrapper to sign and send a transaction with ContractKit.

```

js
import { newKit } from "@celo/contractkit";
import { FileKeystore, KeystoreWalletWrapper } from "@celo/keystores";

// This is the directory that will contain a "keystore" directory
const parentDirectory =
  "/celo/celo-monorepo/packages/sdk/wallets/wallet-keystore/test-keystore-dir";
// Instantiate a KeystoreWalletWrapper using a FileKeystore
const keystoreWalletWrapper = new KeystoreWalletWrapper(
  new FileKeystore(parentDirectory)
);
// Make sure to not commit this if using real funds!
// You can also get this as input on the command-line using readline
// as in the example above for FileKeystore
const privateKey = "YOURTESTPRIVATEKEY";
const passphrase = "test-passphrase1! ";

// Import private key or unlock account
await keystoreWalletWrapper.importPrivateKey(privateKey, passphrase);
// If the keystore file already exists for an address, simply unlock:
// const address = 'YOURTESTADDRESS'
// await keystoreWalletWrapper.unlockAccount(address, passphrase)

// Get the wrapper's LocalWallet instance and pass this into ContractKit
const wallet = keystoreWalletWrapper.getLocalWallet();
const kit = newKit("https://alfajores-forno.celo-testnet.org", wallet);
const [from] = wallet.getAccounts();

// Send a test transaction
const gold = await kit.contracts.getGoldToken();
await gold
  .transfer("0x22579ca45ee22e2e16ddf72d955d6cf4c767b0ef", "1")
  .sendAndWaitForReceipt({ from });
console.log("Transaction sent!");

```



```
# web-dapp.md:
```

```
---
```

```
title: React based DApp
```

```
description: The basics of developing a decentralised application (DApp) on Celo.
```

```
---
```

React Based DApp

The basics of developing a decentralised application (DApp) on Celo.

```
---
```

Getting Started

This example will develop using one of the core Celo contracts, Governance.sol, and allowing users of our DApp to vote on active Celo Governance proposals.

Foreword

This guide requires an understanding of a few popular web technologies. Our application will be written in React, utilising hooks for state management and built with Next.js, a popular static site generation framework.

If you find this tutorial lacking in any way or want to dive into the code more thoroughly, checkout the Celo Tools GitHub repository where much of this tutorial has been ported from.

Getting started

Step one of developing our application is scaffolding it out with create-next-app and adding TypeScript compilation so we can develop more confidently.

```
bash
yarn create next-app voting-dapp
cd voting-dapp
touch tsconfig.json
yarn add --dev typescript @types/react @types/node
```

Now running yarn dev should open up our new Next.js website on localhost:3000.

Next we'll need to add a few Celo specific dependencies so we can work with our core contracts.

```
bash
```

```
yarn add @celo/contractkit @celo-tools/use-contractkit bignumber.js
```

Here's what we'll be using each of these packages for:

- @celo/contractkit is a lightweight wrapper around the Web3 object you may already be familiar with. It contains typed interfaces for the core contracts (generated from the Contract ABIs) and helper functions to make common operations on Celo easier
- @celo-tools/use-contractkit is a community provided library to ease establishing the connection with a user's wallet, whether that is a hardware, mobile, or web wallet. When developing with this library, your users can hold Celo via Valora, a Ledger, Metamask and more
- bignumber.js is a library for expressing large numbers in JavaScript. When interacting with a blockchain we often need to handle arbitrary-precision decimal and non-decimal arithmetic.

We'll also need to add some Next.js config to work with these packages. Update next.config.js with the following:

```
javascript title="next.config.js"
module.exports = {
  webpack: (config) => {
    config.resolve.fallback = {
      ...config.resolve.fallback,
      fs: false,
      net: false,
      childprocess: false,
      readline: false,
    };
    return config;
  },
};
```

We'll need to restart the server for the config changes to take effect.

Developing the application

After all our boilerplate has been setup, we're ready to start developing our application.

Connecting to the user's wallet

When a user wants to interact with your DApp we need to somehow allow them to connect their wallet. Interaction with on chain smart contracts is impossible without this step.

Leveraging our previously added @celo-tools/use-contractkit library we can provide a button that prompts the user to connect their wallet.

Update pages/index.js with the following:

```
javascript title="pages/index.js"
```

```

import React from "react";
import { useContractKit } from "@celo-tools/use-contractkit";
import { ContractKitProvider } from "@celo-tools/use-contractkit";
import "@celo-tools/use-contractkit/lib/styles.css";

function App() {
  const { address, connect } = useContractKit();

  return (
    <main>
      <h1>Celo Voting DApp</h1>
      <p>{address}</p>
      <button onClick={connect}>Click here to connect your
wallet</button>
    </main>
  );
}

function WrappedApp() {
  return (
    <ContractKitProvider
      dapp={{
        name: "My awesome dApp",
        description: "My awesome description",
        url: "https://example.com",
      }}
    >
      <App />
    </ContractKitProvider>
  );
}

export default WrappedApp;

```

Clicking this button will show the use-contractkit modal and allow the user to connect with their wallet of choice. Once the modal has been dismissed, the address property exposed by use-contractkit will be filled with the users primary account.

Accessing contracts

After that we've connected to the user's wallet we can show interesting information based on their address. In the context of a governance voting DApp it may make sense to show past proposals they've voted on. If we were creating a simple banking interface, we could imagine wanting to show transfers into and out of the users account.

:::info

On the Celo blockchain, only queued and dequeued proposals are kept in the Governance state. That means to access old proposals we'd need to access an indexed history of the blockchain. This is out of scope for our tutorial however there's many resources online you can find that will help you accessing indexed blockchain state.

For a comprehensive look at how to interpret this on chain state, take a look at the implementation for the celocli governance:list command.

For the purposes of this tutorial, we'll only be looking at dequeued proposals, or proposals we can currently vote on.

:::

Here's how it looks using a combination of the useEffect and useCallback hooks to request and display all dequeued proposals from the blockchain.

```
javascript title="pages/index.js"
import React, { useCallback, useEffect, useState } from "react";
import { useContractKit } from "@celo-tools/use-contractkit";

function GovernanceApp() {
  const { address, connect, kit, getConnectedKit } = useContractKit();
  const [proposals, setProposals] = useState([]);

  const fetchProposals = useCallback(async () => {
    const governance = await kit.contracts.getGovernance();
    const dequeue = await governance.getDequeue();

    const fetchedProposals = await Promise.all(
      dequeue.map(async (id) => ({
        id,
        ...(await governance.getProposalRecord(id)),
      })),
    );
    setProposals(fetchedProposals);
  }, [kit]);

  useEffect(() => {
    fetchProposals();
  }, [fetchProposals]);

  return (
    <div>
      <h1>Celo Voting DApp</h1>
      <p>{address}</p>
      <button onClick={connect}>Click here to connect your
wallet</button>
      <table>
        <thead>
          <tr>
            <th>ID</th>
            <th>Status</th>
            <th>Description URL</th>
          </tr>
        </thead>
        <tbody>
          {proposals.map((proposal) => (
            <tr>
```

```

        <td>{proposal.id.toString()}</td>
        <td>
          {proposal.passed
            ? "Passed"
            : proposal.approved
              ? "Approved"
              : "Not approved"}
        </td>
        <td>
          <a
            href={proposal.metadata.descriptionURL}
            target="blank"
            style={{ color: "blue", textDecoration: "underline" }}
          >
            Link
          </a>
        </td>
      </tr>
    )}}
  </tbody>
</table>
</div>
);
}

```

Be sure to add this new GovernanceApp component to your WrappedApp component.

```

js title="pages/index.js"
function WrappedApp() {
  return (
    <ContractKitProvider
      dapp={{
        name: "My awesome dApp",
        description: "My awesome description",
        url: "https://example.com",
      }}
    >
      <GovernanceApp />
    </ContractKitProvider>
  );
}

```

This works pretty well however it makes sense to additionally show whether the user has voted on any given dequeued governance proposal. To show that information, we can amend our fetchProposals function as follows

```

js title="pages/index.js"
const fetchProposals = useCallback(async () => {
  if (!address) {
    return;
  }

```

```

    }

    const governance = await kit.contracts.getGovernance();
    const dequeue = await governance.getDequeue();

    const fetchedProposals = await Promise.all(
      dequeue.map(async (id) => {
        const [record, voteRecord] = await Promise.all([
          governance.getProposalRecord(id),
          governance.getVoteRecord(address, id),
        ]);

        return {
          id,
          ...record,
          vote: voteRecord ? voteRecord.value : undefined,
        };
      }),
    );
    setProposals(fetchedProposals);
  }, [kit, address]);

```

Now we have access to whether the user voted on this proposal, we can render that information in our table.

```

js title="pages/index.js"
return (
  <table>
    <thead>
      <tr>
        <th>ID</th>
        <th>Status</th>
        <th>Description URL</th>
        <th>Voted</th>
      </tr>
    </thead>
    <tbody>
      {proposals.map((proposal) => (
        <tr>
          <td>{proposal.id.toString()}</td>
          <td>
            {proposal.passed
              ? "Passed"
              : proposal.approved
                ? "Approved"
                : "Not approved"}
          </td>
          <td>
            <a
              style={{ color: "blue", textDecoration: "underline" }}
              href={proposal.metadata.descriptionURL}
              target="blank"
            >

```

```

                Link
            </a>
        </td>
        <td>{proposal.vote ?? "No vote yet"}</td>
    </tr>
    )}
</tbody>
</table>
);

```

Locking Celo (optional)

A prerequisite to voting on Celo governance proposals is having locked Celo to vote with. We won't cover the various flows for locking, unlocking and relocking Celo in this tutorial but you can check the implementation in Celo Tools or take inspiration from the following script:

```

javascript
const lockValue = new BigNumber(res.flags.value);

const lockedGold = await this.kit.contracts.getLockedGold();
const pendingWithdrawalsValue =
    await lockedGold.getPendingWithdrawalsTotalValue(address);
const relockValue = BigNumber.minimum(pendingWithdrawalsValue, value);
const lockValue = value.minus(relockValue);

const txos = await lockedGold.relock(address, relockValue);
for (const txo of txos) {
    await kit.sendAndWaitForReceipt({ from: address });
}

```

All you need to take care of in your React application is handling user input to select the amount to lock and handling errors in case the user tries to lock more CELO than they hold.

It's also possible that users of your DApp already have locked CELO, so you might not need to worry about the complexity of permitting that operation.

Voting on a proposal

To actually vote on a proposal we need to again interact with the Governance.sol smart contract. Our logic for handling a vote looks as follows:

```

typescript title="pages/index.js"
const vote = useCallback(
    async (id: string, value: VoteValue) => {
        const kit = await getConnectedKit();
        const governance = await kit.contracts.getGovernance();
        await (await governance.vote(id, value)).sendAndWaitForReceipt();
    }
);

```

```

    fetchProposals();
  },
  [kit, fetchProposals],
);

```

How you handle calling that function is up to you. With Celo Tools we opted for simple upwards and downwards facing arrows to handle voting on proposals, however the data can be rendered however you'd prefer.

Here's a simple example showing buttons for Yes or No votes when no vote has been cast.

```

javascript title="pages/index.js"
import { VoteValue } from "@celo/contractkit/lib/wrappers/Governance";

return (
  <tr>
    <td>{proposal.id.toString()}</td>
    <td>
      {proposal.passed
        ? "Passed"
        : proposal.approved
          ? "Approved"
          : "Not approved"}
    </td>
    <td>
      <a href={proposal.descriptionURL} target="blank">
        Description link
      </a>
    </td>
    <td>
      {proposal.vote ? (
        <span>{proposal.vote}</span>
      ) : (
        <div>
          <button onClick={() => vote(proposal.id,
            VoteValue.Yes)}>Yes</button>
          <button onClick={() => vote(proposal.id,
            VoteValue.No)}>No</button>
        </div>
      )}
    </td>
  </tr>
);

```

Best practices

We've compiled a short list on best practices to follow when developing DApps. Following these will improve the end user experience and keep them more engaged with the Celo ecosystem. If you have any questions around these, feel free to reach out on Discord, we're always there and happy to chat.

Last used address

@celo-tools/use-contractkit will remember the address a user last logged in with (via browser LocalStorage). Use this to your advantage and allow your DApp to display the same data whether or not the user has connected their wallet. A good test is to refresh your DApp after connecting and see if anything changes. At the very most, buttons for interaction could be disabled, however it's preferable to prompt to connect the wallet on button click.

Keeping the UI consistent by using the last connected address is a quick win we can have with DApps that make the experience using them closer to Web2, an experience more users will be familiar with.

Loading states

Loading times are often the give away that an application is a Web3 DApp. Be liberal with loading screens and prioritise making animations smooth.

Nothing is worse than a perpetually hanging screen that takes multiple seconds to become interactive. By showing a spinner it communicates to the user that things are happening, however slow they may be.

This is often offset by the ability to index a blockchain and provide the data in a more accessible format (maybe a SQL database or behind a GraphQL API). As mentioned earlier we haven't covered that in this tutorial, however there's a lot of content on the web around DApp optimisation through prior state indexing.

Prerender what you can

With modern static site generators we have amazing leverage over what gets computed server side and what the browser has to request and compute before rendering. If you're unable to index the blockchain before a client requests access to a page, consider loading the relevant data server side with a cache invalidated every hour or so.

Next.js `getStaticProps` comes to mind here as a great way to offload heavy computation to the server.

Showing numbers in wei vs. Celo vs. local currency

Take this advice with a grain of salt as it really depends on how familiar with cryptocurrencies and blockchain your users are. At some point in most DApp users are going to need to deal with large numbers. It's up to you whether you display these in wei (1e18) CELO or converted to a currency the user prefers (BTC, USD or EUR for example).

The sweeping generalisation would be to allow entering values in CELO or their preferred currency and never expose the raw wei amounts to end users.

Wrapping up

Hopefully you have a better grasp on developing DApps against the Celo core contracts now. In this tutorial we covered:

- Connecting to user wallets (use-contractkit)
- Fetching on-chain data
- Calling simple functions on the core contracts
- A brief word on best practices with regard to DApp development.

This is not a comprehensive tutorial for Celo's features and capabilities, keep exploring the docs to learn more and please connect with us on Discord if you need any help (or just want to chat)!

```
# index.md:
```

```
---
title: Web3.js
description: Using Web3.js with Celo
---
```

Web3.js

Web3.js was established in 2014, making it the oldest web3 library. With extensive documentation, an active community and modular design, Web3.js is powerful and easy-to-use. It has support for Celo features via plugins since version 4.13.1.

The Web3.js docs have excellent examples of how to use it in your project.

With Celo

For basic use web3 works out of the box.

For Celo' specific features like Fee Abstraction transactions you need to install @celo/web3-plugin-transaction-types as well as web3@4.13.1 or higher. This also adds utils like getCoreContractAddress for fetching core contract address from onchain registry.

```
ts
// see web3 docs for more info on setup

import { Web3 } from "web3";
import { CeloTransactionTypesPlugin } from "@celo/web3-plugin-transaction-types";

const web3 = new Web3("http://127.0.0.1:8545");
web3.registerPlugin(new CeloTransactionTypesPlugin());

// Now web3.celo is available and celo.eth is celo-aware

const cEUR = await web3.celo.getCoreContractAddress("StableTokenEUR");
const txData = {
```

```

    from: "0x123...",
    to: "0x456...",
    value: 123n,
    feeCurrency: cEUR, // optional
  };
  await web3.celo.populateTransaction(txData);
  const tx = await web3.eth.sendTransaction(txData);

```

Gas Price

When paying for transaction with an alternate feeCurrency token it is important to know the price of gas denominated in that token. You can use `web3.celo.populateTransaction` to make sure `maxPriorityFeePerGas`, `maxFeePerGas`, and `gas` are filled properly.

```

ts
await web3.celo.populateTransaction(txData);

```

More examples in the github repository readme.

ethers.md:

```

---
title: Web3Modal SDK
description: Guide of Web3Modal SDK & Wagmi
---

```

Web3Modal SDK & Wagmi

```
---
```

Installation

```

bash npm2yarn
npm install @web3modal/ethers5 ethers@5.7.2

```

Usage

On top of your app set up the following configuration, making sure that all functions are called outside any React component to avoid unwanted rerenders.

```

tsx
import { createWeb3Modal, defaultConfig } from '@web3modal/ethers5/react'

// 1. Get a Project ID from https://cloud.walletconnect.com/
const projectId = 'YOURPROJECTID'

// 2. Set chains
const celo = {

```

```

    chainId: 42220,
    name: 'Celo',
    currency: 'CELO',
    explorerUrl: 'https://explorer.celo.org/mainnet',
    rpcUrl: 'https://forno.celo.org'
  }

  const alfajores = {
    chainId: 44787,
    name: 'Alfajores',
    currency: 'CELO',
    explorerUrl: 'https://explorer.celo.org/alfajores',
    rpcUrl: 'https://alfajores-forno.celo-testnet.org'
  }

  // 3. Create modal
  const metadata = {
    name: 'My Celo App',
    description: 'My Website description',
    url: 'https://mywebsite.com',
    icons: ['https://avatars.mywebsite.com/']
  }

  createWeb3Modal({
    ethersConfig: defaultConfig({ metadata }),
    chains: [celo, alfajores],
    defaultChain: celo,
    projectId
  })

  export default function App() {
    return <YourApp/>
  }

```

Use the `<w3m-button />` web component anywhere in your app to open the wallet modal.

```

tsx
export default function ConnectButton() {
  return <w3m-button />
}

```

```

:::info
Web components are global html elements that don't require importing.
:::

```

Smart Contracts interaction

```

ts
import { useWeb3ModalSigner, useWeb3ModalAccount } from
  '@web3modal/ethers5/react'
import { ethers } from 'ethers'

```

```

const USDTAddress = '0x617f3112bf5397D0467D315cC709EF968D9ba546'

// The ERC-20 Contract ABI, which is a common contract interface
// for tokens (this is the Human-Readable ABI format)
const USDTAbi = [
  // Some details about the token
  "function name() view returns (string)",
  "function symbol() view returns (string)",

  // Get the account balance
  "function balanceOf(address) view returns (uint)",

  // Send some of your tokens to someone else
  "function transfer(address to, uint amount)",

  // An event triggered whenever anyone transfers to someone else
  "event Transfer(address indexed from, address indexed to, uint amount)"
];

function Components() {
  const { address, chainId, isConnected } = useWeb3ModalAccount()
  const { signer } = useWeb3ModalSigner()

  async function getBalance(){
    if(!isConnected) throw Error("User disconnected")

    // The Contract object
    const USDTContract = new ethers.Contract(USDTAddress, USDTAbi,
signer)
    const USDTBalance = await USDTContract.balanceOf(address)

    console.log(ethers.utils.formatUnits(USDTBalance, 18))
  }

  return (
    <button onClick={getBalance}>Get User Balance</button>
  )
}

```

- Learn more about Web3Modal SDK
- Learn more about Ethers

index.md:

```

---
title: Web3Modal SDK
description: Overview of Web3Modal SDK
---

```

Web3Modal SDK

The Web3Modal SDK is a framework agnostic library that allows you to easily connect your Web3 app with wallets. It provides a simple and

intuitive interface for apps to request actions such as signing transactions and interacting with smart contracts on the blockchain.

Guides

Web3Modal SDK supports ethers v5 and Wagmi; two ethereum libraries that offer different DX for you to interact with the Celo blockchain.

Start with:

- Web3Modal & Wagmi
- Web3Modal & Ethers v5

wagmi.md:

title: Wagmi
description: Guide of Web3Modal SDK & Wagmi

Web3Modal SDK & Wagmi

Installation

```
bash npm2yarn  
npm install @web3modal/wagmi wagmi viem
```

Usage

On top of your app set up the following configuration, making sure that all functions are called outside any React component to avoid unwanted rerenders.

```
tsx  
import { createWeb3Modal, defaultWagmiConfig } from  
  '@web3modal/wagmi/react'  
  
import { WagmiConfig } from 'wagmi'  
import { celo, celoAlfajores } from 'viem/chains'  
  
// 1. Get a Project ID from https://cloud.walletconnect.com/  
const projectId = 'YOURPROJECTID'  
  
// 2. Create wagmiConfig  
const metadata = {  
  name: 'My Celo App',  
  description: 'My Website description',
```

```

    url: 'https://mywebsite.com',
    icons: ['https://avatars.mywebsite.com/']
  }

const chains = [celo, celoAlfajores]
const wagmiConfig = defaultWagmiConfig({ chains, projectId, metadata })

// 3. Create modal
createWeb3Modal({
  wagmiConfig,
  projectId,
  chains,
  defaultChain: celo
})

export default function App() {
  return (
    <WagmiConfig config={wagmiConfig}>
      // Rest of your app...
    </WagmiConfig>
  )
}

```

Use the `<w3m-button />` web component anywhere in your app to open the wallet modal.

```

tsx
export default function ConnectButton() {
  return <w3m-button />
}

```

```

:::info
Web components are global html elements that don't require importing.
:::

```

Smart Contracts interaction

Use Wagmi hooks to read or write Smart Contracts.

```

ts
import { useContractRead } from 'wagmi'
import { USDTAbi } from '../abi/USDTAbi'

function App() {
  const { data, isError, isLoading } = useContractRead({
    address: '0x617f3112bf5397D0467D315cC709EF968D9ba546',
    abi: USDTAbi,
    functionName: 'getHunger',
  })
}

```

- Learn more about Web3Modal SDK

- Learn more about Wagmi

architecture.md:

title: Architecture

description: Overview of the Celo Stack including it's blockchain, core contracts, and applications.

Architecture

Overview of the Celo Stack including it's blockchain, core contracts, and applications.

:::warning

Celo is currently transitioning from a standalone Layer 1 blockchain to an Ethereum Layer 2. As a result, certain aspects of our existing architecture and documentation may be outdated.

For the latest information, please refer to our Celo L2 documentation.

:::

Introduction to the Celo Stack

Celo is oriented around providing the simplest possible experience for end-users, who may have no familiarity with cryptocurrencies and may be using low-cost devices with limited connectivity.

A Full-Stack Approach

To achieve this, Celo takes a full-stack approach, where each layer of the stack is designed with the end-user in mind while considering other stakeholders (e.g. operators of nodes in the network) involved in enabling the end-user experience.

Celo Blockchain

An open cryptographic protocol that allows applications to make transactions with and run smart contracts in a secure and decentralized fashion. The Celo blockchain code has shared ancestry with Ethereum and maintains full EVM compatibility for smart contracts. However, it uses a Byzantine Fault Tolerant (BFT) consensus mechanism (Proof-of-Stake) rather than Proof-of-Work and has different block format, transaction format, client synchronization protocols, and gas payment and pricing mechanisms.

Celo Core Contracts

A set of smart contracts running on the Celo blockchain that comprise much of the logic of the platform features including ERC-20 stable currencies, identity attestations, proof-of-stake, and governance. These smart contracts are upgradeable and managed by the decentralized governance process.

Applications

Applications for end users built on the Celo Platform. The Celo Wallet app, the first of an ecosystem of applications, allows end-users to manage accounts and make payments securely and simply by taking advantage of the innovations in the Celo Protocol. Applications take the form of external mobile or backend software: they interact with the Celo blockchain to issue transactions and invoke code that forms the Celo Core Contracts' API. Third parties can also deploy custom smart contracts that their own applications can invoke, which in turn can leverage Celo Core Contracts. Applications may use centralized cloud services to provide some of their functionality: in the case of the Celo Wallet, push notifications, and a transaction activity feed.

The Celo blockchain and Celo Core Contracts together comprise the Celo Protocol.

Celo Network Topology

The topology of a Celo network consists of machines running the Celo blockchain software in several distinct configurations:

Validators

Validators gather transactions received from other nodes and execute any associated smart contracts to form new blocks, then participate in a Byzantine Fault Tolerant (BFT) consensus protocol to advance the state of the network. Since BFT protocols can scale only to a few hundred participants and can tolerate at most a third of the participants acting maliciously, a proof-of-stake mechanism admits only a limited set of nodes to this role.

Full Nodes

Most machines running the Celo blockchain software are either not configured to be, or not elected as, validators. Celo nodes do not do "mining" as in Proof-of-Work networks. Their primary role is to serve requests from light clients and forward their transactions, for which they receive the fees associated with those transactions. These payments create a 'permissionless onramp' for individuals in the community to earn currency. Full nodes maintain at least a partial history of the blockchain by transferring new blocks between themselves and can join or leave the network at any time.

Light Clients

Applications including the Celo Wallet will also run on each user's device an instance of the Celo blockchain software operating as a 'light client'. Light clients connect to full nodes to make requests for account and transaction data and to sign and submit new transactions, but they do not receive or retain the full state of the blockchain.

Celo Wallet

The Celo Wallet application is a fully unmanaged wallet that allows users to self custody their funds using their own keys and accounts. All critical features such as sending transactions and checking balances can be done in a trustless manner using the peer-to-peer light client protocol. However, the wallet does use a few centralized cloud services to improve the user experience where possible, e.g.:

- Google Play Services: to pre-load invitations in the app
- Celo Wallet Notification Service: sends device push notifications when a user receives a payment or requests for payment
- Celo Wallet Blockchain API: provides a GraphQL API to query transactions on the blockchain on a per-account basis, used to implement a user's activity feed.

When end-users download the Celo Wallet from, for example, the Google Play Store, users are trusting both cLabs (or the entity that has made the application available in the Play Store) and Google to deliver a correct binary, and most users would feel that relying on these centralized services to provide this additional functionality is worthwhile.

```
# gallery.md:
```

```
---
```

```
title: Celo DApp Gallery
```

```
description: List of featured DApp examples to help you start building on Celo.
```

```
---
```

```
import PageRef from '@components/PageRef'
```

```
Celo DApp Gallery
```

```
List of featured DApp examples to help you learn more about Celo.
```

```
---
```

```
:::tip
```

```
Head over to Celo Hub and Electric Capital Ecosystem Gallery to track or share new Celo projects.
```

```
:::
```

:::note Add your app

Are you building on Celo? To add your project, please submit a pull request updating this page!

:::

Celo Camp Projects

Celo Camp is a virtual accelerator and competition focused on helping startups and developers build decentralized financial apps (dApps) on Celo. During Celo Camp, teams will build mobile-first dApps and other financial tools and services on Celo. In addition to receiving technical and Web3 support, teams receive guidance from experienced mentors, alongside a custom-built curriculum, that will set them up for the next phase of their entrepreneurial success.

[View Celo Camp](#)

GoodGhosting

GoodGosting is a no-loss DeFi saving game. Users can compete with others to get higher interest rates than when they would save by themselves. We created a shared saving pool smart contract, which plugs into existing Celo DeFi projects (e.g. Moola) to generate interest for all winning players.

- [Website](#)
- [GitHub](#)

Infibridge

Infibridge is a Metamask-like browser extension wallet for Celo. It will support payment of gas fees in Celo or stable currencies like cUSD and cEUR. It has deep integration with Celo to allow staking, voting and token swaps.

- [Website](#)
- [GitHub](#)

Plastiks

Plastiks is an NFT marketplace deployed on CELO that brings People, Organizations and Companies together to fight against plastic pollution.

We are tokenizing and creating NFTs out of the invoices generated by the recovery projects and recyclers worldwide, that states that certain type of plastic and quantity has been recovered, these NFTs becomes Plastic Recovery Guarantees.

- [Website](#)

Biscoin DollarSave

Biscount is working on an out-of-the-box CELO and Celo Dollar hot wallet to be used by projects that require automated processing of withdrawals. They also created a cUSD SPV client written in Typescript.

- Website
- GitHub

Paychant

Paychant is developing an open-source web wallet developed on the Celo blockchain.

- Website
- GitHub

Moola

Moola is working on algorithmic money markets on Celo.

- Website
- GitHub

LeafCelo

LeafCelo is developing lenrefugee, a guarantor-backed DeFi lending platform for refugees.

- Website
- GitHub

El Dorado

El Dorado is building "the crypto dollar wallet of LATAM. We are reinventing the way financial products and services are designed, especially for people living in unstable economies. We are going to provide all the services a bank can offer and more. USD savings accounts and P2P payments/exchange. All powered by Open Blockchains."

- Website
- GitHub

ReSource Network

The ReSource Network is building a resource-based mutual credit blockchain protocol. The initial version is deployed in CELO mainnet on address: 0x39049c02a56c3ecd046f6c2a9be0cffa2bc29c08

- Website
- GitHub

Pesabase

An integration of a Celo Dollar (Mento Stablecoin) and Mpesa, showing flows of payment between a Mpesa User and a Celo Blockchain wallet from

anywhere in the world. Our aim with Pesabase is to provide Africans with a cheap and social option to remit and pay for goods and services.

- Android App
- GitHub

Wallet as a Service

Tangany's custody solution Wallet as a Service enables you to create and manage HSM-secured wallets and interact with the underlying blockchain. Our API makes it incredibly easy for developers to execute transactions, check details of past transactions or query account balances without any worries about private key safekeeping. For the Celo Camp 2020 we have developed a prototype version of our product that integrates the Celo blockchain. The demo project presents the new features and provides some code snippets to get started.

- Website
- Demo Project

impactMarket

impactMarket enables any vulnerable community to create its own unconditional basic income system for their beneficiaries, where each one can claim a fixed amount on a regular basis and make payments for free.

- Website

Celo Toolkit by MugglePay

MugglePay provides a payment SDK for merchants to accept cryptocurrencies. Mugglepay makes crypto payment easy and thousands of merchants are onboarded with MugglePay SDK. For the Celo Camp 2020, we will integrate with cUSD/CELO for payments on the Celo blockchain.

- Website

Celo Toolkit is the open source interface for managing Celo accounts on the Web. Celo developer might start with a funded Celo account (Testnet) now in 1 minute. A Phone number or app downloads are not necessary.

- Website
- GitHub

Dunia Pay Wallet

At Dunia Payment, we are building an electronic wallet that will let people in Sub-Saharan Africa send and receive money directly on their phones. The app will use the Celo light client to process transactions faster, even in low internet connection areas common in Africa. The wallet will be built on top of the Celo platform and also using a set of external open source smart contracts.

- GitHub

Cryptum Woocommerce Checkout

Based on Cryptum APIs, the open source Plugin connects to WordPress e-commerce checkouts, providing fast and easy integration for Celo and cUSD acceptance and management for merchants.

- GitHub

Cryptum Woocommerce NFT

Based on Cryptum APIs, the open source Plugin connects to WordPress e-commerce, providing easy and fast Celo NFT creation and integration through a no-code interface - with marketplace goods for merchants, offering customers unique experiences and extending NFT possibilities for many marketplaces.

- GitHub

Bloinx by BX Smart Labs

A decentralized app that helps users to create reliable savings communities by using smart contracts, in a transparent, verifiable, and trustworthy environment. Bloinx is upgrading the way to manage TANDAS, which are rotating, saving, and credit associations, used in Latin American communities.

- Website
- GitHub

Poof.cash

Poof.cash is a protocol for decentralized, private transactions. It is the first native privacy tool for Celo users.

- Website
- GitHub

Talent Protocol

Talent Protocol is the web3 professional community where high-potential builders can transform loose connections into a support network able to invest in their future.

Their mission is to help the next generation of talent achieve success and fulfilment by enabling collective careers.

Until now careers have mostly been a single-player game where only a privileged few can win. With web3, careers are becoming multiplayer journeys where early support is rewarded and success is shared.

- Website
- GitHub

esolidar

Empowering philanthropy and ESG practice through blockchain. Any nonprofit, sustainable project or cause can choose to receive Celo stablecoins as donations.

- Website
- GitHub

NEFTME

NEFTME is a decentralized social NFT-powered network that allows anyone, anywhere to create, share and sell an NFT, with any media content! Users can also benefit from the network value created through their engagement and connections, powered by their followers/supporters commitment, through NFTs Staking model with \$NEFT token.

- Website
- GitHub

Decentralized Impact Incubator Winners

The Decentralized Impact Incubator is a 6-week program to ideate and prototype blockchain-based solutions to global social and environmental challenges. During the period, participants from around the world gather to form teams, design business models, and draft proposals and code. Teams need to pass through weekly checkpoints and are guided by mentors throughout the process. Winning projects will receive grants to support continued development.

- Website

Bienvenir

Decentralized application to help coordinate and improve the impact metric of non-profit organizations that work with migrants in Latin America.

- GitHub

Symbiotic Protocol

Decentralized and autonomous protocol built for multi-chain social impact, the play-for-good solution for crypto donation, connecting Gamers and Nonprofits.

- GitHub

Additional DApps

Ubeswap

Ubeswap is a mobile-first DeFi exchange, a mobile compatible fork of Uniswap running on the Celo blockchain. Ubeswap is open source on GitHub and available on Safari for iOS and any browser on Android.

- Website
- GitHub

Celo Vote

Celovote automatically distributes your stake to preferred validator groups that have high estimated APY (annual percentage yield) and automatically rebalances votes if any voted groups fail to maintain high uptime. You retain full custody of your CELO and receive 100% of your rewards.

- Website

Celo Tools

Celo Tools is an accessible frontend to the Celo CLI. Anyone and everyone participating in the Celo network should be able to stake and vote on governance proposals, Celo Tools provides this functionality for those without the know-how to access a command line.

- Website

Savings Circle

Savings Circles let you pool funds with your friends to save for large purchases. They are known by a variety of names around the world and are a common way to get liquidity and access to loans without access to formal financial institutions.

- GitHub

Lovecrypto

Lovecrypto enables people to earn cUSD to do tasks in their phones.

- Website
- GitHub

Cent Wallet

Cent is a mobile wallet for buying and holding crypto as well as using Wallet Connect to authorize transactions on exchanges and other services. Cent is open source on GitHub and available on Android and iOS through the app store.

- Website
- GitHub

Symmetric

Symmetric is a fork of Balancer for Celo. Symmetric is open source on GitHub, has a tokenomics model that includes a risk fund protecting traders, is to be DAO controlled, and has a roadmap of new features.

- Website
- GitHub

Tradegen

Tradegen is an asset management and algo trading platform that connects users to traders. Users can invest in pools, manage their own pools, farm pool tokens to earn extra yield, and trade pool tokens as NFTs on the platform's marketplace. Tradegen is open source on GitHub.

- Website
- GitHub

Umoja

UMOJA is an open banking platform that enables NGOs and FSPs to provide flexible micro-financing to anyone with a phone. Umoja is open source on GitHub and consists of a suite of APIs and products to make digital money accounts more accessible (and more easily to develop for other financial applications).

- GitHub

Santym

Santym is a platform that allows Africans to have access to U.S Banking with crypto integration. Santym's available on GitHub. Santym is currently working on an African Stablecoin Exchange that allows continental currencies to be easily swapped on celo's blockchain.

- GitHub

Into The Verse

Into the Verse is a Pixel Parallel Universe, connecting players on Celo. We have a Play2Earn NFT Dungeon Loot Game, game storefront, and Token Swaps on Celo. Open Sourced on GitHub, we aim to bridge Defi and Gaming through token-gated Pixel Metaverses.

- GitHub

RARE GEMS

Non-custodial multichain NFT marketplace, connected to Celo

- Website
- Celo NFT collections

Bru Finance

Globally the largest issuer of tokenized real world assets (\$600 Million & Counting), Brú Finance is a new paradigm of DeFi 2.5 that brings Tokenized Emerging Market Asset Backed Bonds to decentralised finance.

- Website
- Github

EcoBytes - Earn Your Mobile Data Dividend

EcoBytes is a Regenerative Crypto platform that rewards consumers with crypto for efficient use of mobile data networks thus saving Mobile Operators billions of networking and retention costs as user data consumption skyrockets. EcoBytes is a native mobile-first DApp that directly connects to Celo and a Valora wallet which allows users to redeem their network efficiency points to cUSD and a sponsored "relationship" reward token via a success-sharing arrangement with their Operator. These freely-earned tokens can be used to direct donations to their favorite causes or used in a marketplace for acquiring goods or services. With wide adoption, many Millions of dollars/month will flow passively from Operators to their EcoByte enabled customers as they maximize their WiFi usage. DApps interested in being in the EcoByte marketplace can contact us.

- GitHub
- Website

Celo Tracker

The Front Page of Celo. Track and manage all your assets from one place. We support all major projects on Celo, allow swapping using the best exchange rate on Celo and also track and show NFTs.

- Website
- GitHub

Cyberbox

Cyberbox is ReFi NFT Marketplace that helps anyone become carbon neutral by offsetting carbon (CO2) through NFT trading.

- Website
- GitHub

inTheory NFTs

inTheory is a DeSci funding platform that allows crypto users to fund, support, and track scientific research through the trading of AI-generative artworks. This application allows users to mint NFTs based on their research interests prior to the launch of the full inTheory platform.

- Website
- GitHub

WalliD

WalliD aims to provide the ultimate onboarding and digital trust solution by aggregating authenticators and digital ID providers. Our aggregator architecture brings together centralized Certificate authorities and their Public Key Infrastructures (PKIs), DID and Zero-Knowledge protocols, as well as authentication providers. This allows us to offer a comprehensive toolkit with different suites of products that cater to the needs of WebApp or dApp developers and users across the globe.

- Website
- GitHub

glossary.md:

title: Glossary

description: List of key terms related to the Celo platform, networks, tools, and blockchain technology.

Glossary

List of key terms related to the Celo platform, networks, tools, and blockchain technology.

Account

Identifies an account on Celo. There are two types of account. Externally owned accounts have an associated CELO balance and are controlled by a user holding the associated public-private keypair. Contract accounts contain the code and data of a single smart contract which can be called and manipulate its own stored data.

Address

A unique identifier for an account on the Celo blockchain.

Alfajores

The first public Celo test network, available for developers to use freely subject to the Alfajores Testnet Disclaimer.

Attestation

Generally, support for an entity having an associated identity. In Celo, each attestation confirms that an account has access to a message sent to a specific identifier typically a mobile phone number via an Attestation Provider via Celo's ODIS.

Attestation Service

Former name for what has become Social Connect.

Baklava

The second public Celo test network, intended for use as a testing ground for protocol changes and validator configurations. It is subject to the Baklava Testnet Disclaimer.

Block

The unit of update to the blockchain. A block consists of a header identifying its position in the chain and other metadata, and a body that contains a list of transactions, and data structures that describe the new state after executing those transactions.

Blockchain

A database maintained by a distributed set of computers that do not share a trust relationship or common ownership. This arrangement is referred to as decentralized. The content of a blockchain's database, or ledger, is authenticated using cryptographic techniques, preventing its contents being added to, edited or removed except according to a protocol operated by the network as a whole.

Byzantine Fault Tolerant (BFT) Consensus

A form of consensus algorithm in which up to a third of participants can be faulty or malicious.

cLabs, Celo Labs

The team that, before mainnet release, has worked closely to develop and shape the Celo protocol and Celo Wallet application with the input of the larger Celo Community.

Carbon Offsetting Fund

An account that receives a transfer as part of epoch rewards to cover the cost of offsetting the carbon generated by the hardware footprint required to operate the Celo network. The account that receives these funds and the amount is determined from time to time by on-chain governance proposal.

Celo

An open platform that makes financial tools accessible to anyone with a mobile phone.

CELO

The ticker symbol for the Celo native asset, often written in capital letters to avoid confusion with references to the Celo protocol.

Celo Foundation

A mission-driven foundation responsible for education, community engagement, and ecosystem support of the Celo platform.

Celo Gold

The deprecated name for the Celo native asset, which now is referred to simply as "Celo" or preferably "CELO".

Celo Native Asset

The Celo native asset which may be used to take actions on-chain, such as participating in Governance.

Claim

Part of the metadata that Celo can associate with an address, a claim is used by an account to assert it has control over a particular off-chain entity (for example, a DNS domain name, an account on a third party service, etc). Claims can only be verified off-chain.

Client

See Node.

Consensus

An algorithm that enables multiple computers to reach a decision on a single value proposed by one of them, despite network or computer failures.

ContractKit

A library to help developers and operators of Validator nodes interact with the Celo Blockchain and Celo Core Contracts.

DApp

Short for Decentralized Application. An application, usually a mobile application, which to deliver its functionality connects to a decentralized network like Celo, rather than to centralized services in a single organization's data centers.

DeFi

Decentralized Finance; open source software and networks without intermediaries in the financial space.

Derivation Path

A derivation path defines how private keys and addresses are derived from a mnemonic. The Bitcoin community defined the standards for derivation

paths in BIP39 and BIP42 and BIP44. A registry of coins/chains and their paths is maintained and includes Celo and Ethereum paths.

Double Signing

When a validator signs two different blocks at the same height and with the same parent hash in the blockchain.

Epoch

A fixed number of blocks, configured in the network's genesis block, during which the same validator set is used for consensus. A validator election is carried out after the last block of an epoch, and any resulting changes to the validator set are written into that block's header.

Epoch Rewards

Funds disbursed by the protocol at the end of every epoch as incentives for validators, validator groups, holders of Locked Gold that voted for validator groups that elected one or more validators, the Reserve, the Community Fund, and the Carbon Offsetting Fund.

ERC-20

A standard interface for implementing tokens as smart contracts. Balances associated with addresses are typically maintained inside the contract's storage. Both CELO and Celo Dollars implement the ERC-20 interface.

Ethereum

A project with which the code of the Celo Blockchain has shared ancestry. Ethereum facilitates building general-purpose decentralized applications.

EVM

The Ethereum Virtual Machine. A runtime environment used by smart contracts on Ethereum and Celo.

Externally Owned Account (EOA)

An account owned by a private key which has full control to send transactions from the account by signing and submitting the transaction to the blockchain. All transaction on Celo must originate from an EOA, which pays for the transaction fees.

Full Node

A computer running the Celo Blockchain software that maintains a full copy of the blockchain locally and, in Celo, receives transaction fees in exchange for servicing light clients.

Gas

A step of execution of a smart contract. Different operations consume different amounts of gas. To prevent denial-of-service attacks, transactions specify a maximum gas which bounds the steps of execution before a transaction is reverted.

Gas Price

Determines the unit price for gas, i.e. cost for a transaction to perform one step of execution. This is used to prioritize which transactions the network applies.

Gas Price Minimum

The minimum unit price for gas that the Celo protocol will accept. This value changes from block to block in response to congestion, increasing as more transactions are competing for the limited capacity of the network. A transaction specifying a gas price below the Gas Price Minimum will not be processed until the Gas Price Minimum falls.

Genesis block

The very first block in the blockchain, provided as configuration to Celo Blockchain nodes.

Geth

go-ethereum, a Golang implementation of the Ethereum protocol from which the Celo Blockchain software is forked.

Governance

A part of Celo that allow the protocol to be upgraded, and other actions to be taken on behalf of the network, by holding a referendum process in which CELO holders vote for proposals submitted by the community.

Governable

A smart contract that is owned by the Celo Governance mechanism and so can be changed or updated by an on-chain governance proposal.

Group Share

The proportion of epoch rewards for an elected validator that is passed to the validator group that caused it to be elected. This is a property that can be configured by each group.

Header

See Block.

HSM

Hardware Security Module. A hardware device that hosts one or more private keys and signs data without passing the key off the device.

Key Rotation

The creation of a new cryptographic key to replace an existing key in active use.

Community Fund

An account that supports the development and operational costs of the Celo protocol. The Community Fund is maintained by a transfer made as part of Epoch Rewards, and is intended to cover costs beyond the other specific incentives provided to validators and validator groups. Awards can be made through an on-chain governance proposal.

Istanbul

Istanbul, or IBFT, is the original name of the implementation of the Byzantine Fault Tolerant consensus algorithm used by Celo. Istanbul is also the name of a hard fork of the Ethereum network.

Light Client

A device or computer running the Celo Blockchain software that keeps typically only the most recent blockchain state, such that it can send transactions and identify what other data to request as necessary. Every Celo Wallet installation includes a Celo Blockchain light client.

Locked Gold

CELO balances held in escrow at the Locked Gold contract for the account that deposited it there. This permits that balance to be used for voting in validator elections, governance proposals, and to meet staking requirements for registering a validator or validator group.

Locked Gold is in the process of being renamed along with other references to Celo Gold (cGLD), which is now referred as CELO.

Mainnet

The Celo production network.

Node

A running instance of the Celo Blockchain software. This could be configured to run as a Validator, Full Node, or Light Client. Used interchangeably with 'Client'.

On-chain

An interaction that takes place solely through a transaction being executed on the blockchain and updating the state of the ledger.

Proof-of-Stake

The system that determines the participants in a Byzantine Fault Tolerant consensus mechanism. Celo's Proof-of-Stake mechanism permits accounts to convert units of CELO into Locked Gold then vote for Validator Groups, such that an election held at the end of every epoch selects a new set of validators for the following epoch.

RC1

RC1, which stands for Release Candidate 1, was the first network that had the potential to become the Celo mainnet. It was promoted to Mainnet after the Celo community voted to enable CELO transfers on the network on May 18, 2020.

Savings Circle

A common practice in societies without easy access to banking (source); a peer-to-peer savings and loan group.

SBAT

Should Be Able To (Acronym used in GitHub issue title)

SNBAT

Should Not Be Able To (Acronym used in GitHub issue title)

Sync

The process, when a node joins the network, of requesting and receiving block headers so that the node catches up to the network's latest state.

SDK

Software Development Kit. Generally, a suite of developer tools that enable applications to be built on a platform.

Slashing

The reduction in the stake of a validator, a validator group, or both, for a particular action not conducive to the health of the network.

Slashing Penalty

A variable that is tracked for each validator group by the proof-of-stake mechanism that causes rewards to the group, its validators and its voters to be temporarily reduced because of a recent slashing.

Solidity

The preferred language for writing Smart Contracts on the Celo platform.

Smart Contracts

Programs that are deployed to a blockchain and execute on its nodes. They operate on data on the blockchain, and on external inputs received in transactions or messages to the blockchain, and may update the state of the blockchain, including account balances. On Celo and Ethereum, smart contracts are written in languages like Solidity.

Stablecoin

A stablecoin is a type of cryptocurrency whose price tracks an external currency or commodity.

Stake

Locked Gold that a validator or validator group puts at risk at the point of registration. A portion of a stake can be slashed for particular actions not conducive to the health of the network.

Testnet

A test network. Its tokens hold no real world economic value.

Transaction

Requests to make a change to the state of the blockchain. They can: transfer value between accounts; execute a function in a smart contract and pass in arguments \ (perhaps causing other smart contracts to be called, update their storage, or transfer value\); or create a new smart contract.

Unlocking Period

The elapsed time between an account requesting an amount of Locked Gold be unlocked and the first point it can be withdrawn.

Uptime Score

A variable that is tracked for each validator by the proof-of-stake mechanism that approximates how regularly that validator participates in consensus.

Validator

Both: the entity in the proof-of-stake mechanism that can be associated with a validator group and subsequently elected; and a running instance of the Celo Blockchain software that is configured and ready (if elected) to participate in the Byzantine Fault Tolerant consensus algorithm to agree new blocks to append to the blockchain ledger.

Validator Group

The entity in the proof-of-stake mechanism that can associate validators, receive votes from holders of Locked Gold and cause those validators to be elected.

Validator Set

The set of elected validators (with respect to a specific epoch) that participate in consensus

Wallet

A DApp that allows a user to manage an account, and usually stores the associated private key.

DApp that allows a user to manage an account, and usually stores the associated private key.

history.md:

title: Our History

import ColoredText from '/src/components/ColoredText';

Celo launched on Earth Day 2020 as an energy-efficient and low-cost Layer 1 blockchain. Since its inception, Celo has continually evolved to adapt to the changing landscape of blockchain technology and needs of its users in the broader crypto ecosystem.

The Evolution of Celo: From Layer 1 to Layer 2

This timeline highlights the key milestones in Celo's development, from its launch as a Layer 1 blockchain to its exploration of becoming an Ethereum Layer 2 solution.

Timeline of Celo's Journey:

July 2015: Ethereum Launch

Ethereum debuts as a Proof-of-Work blockchain, introducing smart contracts but facing high fees and energy consumption.

April 2020: Celo Mainnet Launch

Celo officially launches its mainnet on Earth Day, marking its debut as a Layer 1 blockchain network. Designed with a focus on energy efficiency and low transaction costs, Celo sets out to provide a more sustainable and accessible blockchain solution.

May 2021: Celo Donut Hard Fork

The Donut hard fork is implemented on Celo, enhancing its EVM (Ethereum Virtual Machine) compatibility and introducing cross-chain interoperability with other blockchain networks.

September 2022: Ethereum Proof of Stake Transition

Ethereum completes its transition from Proof of Work to Proof of Stake, significantly reducing its energy consumption and addressing some of the sustainability concerns that Celo originally set out to solve.

January 2023: Celo 2.0 Announcement

cLabs announces Celo 2.0 with improved Ethereum compatibility, better performance, and enhanced tokenomics.

July 2023: Layer 2 Transition Proposal

In a strategic move, cLabs proposes transitioning Celo from an independent Layer 1 blockchain to an Ethereum Layer 2 to leverage Ethereum's security and expand its reach within the Ethereum ecosystem.

September 2023: Gingerbread Hard Fork Proposal

The Gingerbread hardfork is implemented to prepare for Celo L2 with a focus on improving Celo's compatibility with Ethereum by streamlining code and introducing [Ultragreen Money](#), on-chain carbon offsetting through transaction fees, enhancing both performance and sustainability.

April 2024: Celo joins Optimism OP Stack

cLabs [proposed the OP Stack](#) for its Layer 2 migration to align more closely with Ethereum, reduce production time, enhance security, and maintain its unique features with minimal migration risk.

White Papers

For a deeper understanding of Celo's evolution and the technical foundations behind it, check out our [white papers](#). These documents cover the innovations and decisions that have shaped Celo.

[Read our White Papers](#)

index.md:

title: Discover Celo

```
import ReactYouTube from "react-youtube";
import ColoredText from '/src/components/ColoredText';
```

Celo is a blockchain network designed for the real world, with a mission to build a regenerative digital economy that fosters prosperity for all.

What is Celo?

Celo is an emerging Ethereum Layer-2 designed to make blockchain technology accessible to all. With its focus on scalability, low fees, and ease of use, Celo is ideal for building blockchain products that reach millions of users around the globe.

```
<ReactYouTube videoId={'4a70pVEcRw4'} />
```

Why Build on Celo?

Celo is fully EVM-compatible, offering the same development experience as Ethereum with improved scalability and lower costs.

Built for Everyday Users:

Celo is designed with features that lower the entry barrier for those new to cryptocurrency.

- Fee Abstraction: Users can pay transaction fees with several different tokens, making payments simple and flexible.
- Sub-Cent Fees: Celo maintains low gas fees, often below a cent, keeping transactions affordable.
- Native Stablecoins: Celo provides native stablecoins like cUSD, cEUR, cREAL, and cKES, offering a stable way to send and receive money. Check out Mento to learn more.

Mobile-First Approach:

Celo is optimized for mobile devices, making blockchain accessible to billions of smartphone users worldwide.

- Ultralight Client: Celo's lightweight client allows fast transactions on mobile phones without needing to download large amounts of data, ideal for areas with limited internet.
- Phone number mapping: Celo's native protocol Social Connect, links wallets to phone numbers, making it easy to send and receive payments using just a phone number.

Optimized for Global Reach:

- Scalability: Celo's focus on real-world use cases has already helped scale applications to over 100,000 daily active users (DAUs), proving its capability to support large-scale deployment.
- Community Engagement: Celo boasts an active global community of users and builders. With over 4 years of experience in bringing blockchain to real-world users, Celo offers a supportive environment for developers to test, launch, and scale their applications. Community members are engaged and eager to provide feedback, helping builders refine their products for

broad, practical use. If you are interested in contributing, make sure to reach out.

Carbon Negative:

Celo is committed to environmental sustainability, offsetting more carbon than it produces.

- Proof-of-Stake: Celo uses an energy-efficient consensus mechanism that avoids energy-intensive mining, reducing its carbon footprint.
- Ultragreen-Money: A portion of every transaction fee goes towards carbon offsetting and sustainability projects, supporting environmental efforts with every transaction.

Understanding the Celo Ecosystem

What is CELO?

CELO is the platform-native asset that supports the growth and development of the Celo blockchain and ecosystem. CELO holders can earn rewards, stake with validators, and vote on proposals that shape the future of Celo.

<ReactYouTube videoId={'mkpTmbkRv4A'} />

What Makes Celo's Native Stablecoins Unique?

Named for the currencies they follow, Celo Dollars (cUSD), Celo Euros (cEUR), Celo Reals (cREAL), and Celo Kenyan Shilling (cKES) are **Mento** stablecoins that allow you to share value faster, cheaper, and more easily on your mobile phone. Mento stablecoins instantly unlock access for everyday uses like low-cost remittances and cross-border payments, global distribution of charitable aid, effortlessly paying online, or transferring value within exchanges, particularly in markets subject to currency volatility.

<ReactYouTube videoId={'nlklJcjTnp8'} />

Get Started with Celo

Whether you're an experienced developer or just starting out, Celo provides the essential tools and resources to turn your ideas into reality.

Start building on Celo today and contribute to a growing ecosystem that supports a regenerative digital economy, fostering prosperity for all.

Connect with us on **Discord** or on **X**, and be sure to explore our ecosystem resources.

<!-- ---

title: What is Celo?

description: Celo's mission is to build a financial system that creates the conditions for prosperity—for everyone.

```
import YouTube from '@site/src/components/YouTube';
import PageRef from '@site/src/components/PageRef';
```

Celo's mission is to build a financial system that creates the conditions for prosperity—for everyone.

Cryptocurrency for a beautiful planet

Celo is an emerging Ethereum Layer-2 and mobile-first blockchain network built for the real world and designed for fast, low-cost payments worldwide. Here are few of the key features of Celo:

- Layer-1 to an Ethereum Layer-2 protocol
- Proof-of-stake
- Carbon negative
- Mobile-first identity
- Ultra-light clients
- Localized stablecoins (cUSD, cEUR, cREAL, cKES)
- Gas payable in multiple currencies

What is the Celo Platform?

Celo makes sending payments as easy as sending a text, to anyone with an internet connection, anywhere in the world. Celo maps phone numbers to wallet addresses using a novel decentralized address-based identity layer. Mobile participants can earn rewards for securing and maintaining the system.

<YouTube videoId="4a70pVEcRw4"/>

What is CELO?

CELO is the platform-native asset that supports the growth and development of the Celo blockchain and ecosystem. CELO holders can earn rewards, stake with validators, and vote on proposals that shape the future of Celo.


<YouTube videoId="mkpTmbkRv4A"/>

What can Celo Dollars do?

Named for the currencies they follow, Celo Dollars (cUSD), Celo Euros (cEUR) and Celo Reals (cREAL) are Mento stablecoins that allow you to share value faster, cheaper, and more easily on your mobile phone. Mento stablecoins instantly unlock access for everyday uses like low-cost remittances and cross-border payments, global distribution of charitable

aid, effortlessly paying online, or transferring value within exchanges, particularly in markets subject to currency volatility.

<YouTube videoId="nlklJcjTnp8"/>

:::tip Learn more 

Read Celo: Building a Regenerative Economy, Celo Spotlight, and the Celo 2021 Annual Report for an in-depth look at Celo and how it's creating the conditions of prosperity for everyone.

::: -->

web2-to-web3.md:

title: Web2 to Web3 - The Anatomy of a dApp
description: Overview of the anatomy of a dApp and its architecture difference from a web2 app.

Overview of the anatomy of a dApp and its architecture difference from a web2 app.

Introduction

This article assumes you have experience writing front-ends for web apps (called web2 apps here) and want to gain an understanding of the differences and commonalities when creating front-ends for web3 apps.

Web3 dApps expand on the front-end architectural systems of Web2 Applications. Concepts like SinglePageApp, State and UI Management, Data Fetching and Caching serve as the foundation of any Web3 dApp. Compared to Web2 Apps, even more computation is pushed to the client/browser as computing on the blockchain costs literal money while running your own traditional backend between the blockchain and frontend increases centralization and is not used by most web3 dApps.

Typical DApp Stack looks something like this:

Web2 Foundation

App Framework – Something like Next.js, create-react-app.

UI Management – As in web2, React is the most popular, Vue also is used.

State Management -- This is intricately tied to your UI Management Library. For the biggest dApps redux is popular.

Web3 Specific

Just as in web2 where there is overlap and tie in between App Framework, UI and State Management, the web3 tools may blur boundaries especially between the Web3 Helper Libraries and Wallet Connection Manager categories.

The three categories below are not industry terms. Nevertheless It can be helpful to understand them as groupings of functionality. Lets look at these from bottom to top, where higher means a tool will require a specific lower tool.

EVM Blockchain Toolkit

Examples – Web3.js or Ethers.js.

Web3.js has significant disadvantages on browsers due to its giant size and lack of modularity. New projects use ethers.js) These make the calls from your dapp to contracts you read and write to on the blockchain via JSON-RPC calls to a blockchain node. These tools can run both in node.js and browsers.

Web3 Helper Libraries (optional but helpful)

Examples – Wagmi, useDapp, etc.

These sit between EVM Blockchain Toolkit and Wallet Connection Manager, or are in some cases are used by or have the functionality of a Wallet Connection Manager. They help simplify the complexities interacting with smart contracts which has additional UX considerations compared to a http api call. For instance rather than just wait for call to be returned we must 1) wait for wallet to sign 2) wait for tx to be sent to node 3) wait for it to be included in a block.

Wallet Connection Manager

Examples – Rainbowkit, web3modal

These are the interface between the user and the dApp via the user's wallet. They will manage which chain the dapp is on and establish connection to the wallet where the user will sign the transaction data with their private key thus authenticating that operation. Most wallets do this thru either the WalletConnect standard or, for browser extensions, by injecting code into window (such as metamask)

Differences

Authentication

Whereas web2 apps will use a user name and password or auth service like SignInWith(FANG) for authentication. In Web3 each write operation is authenticated by signing with the users wallet. For the purposes of User experiences wallet-connection managers will display whatever wallet

address they receive from the wallet as the "logged in" user. However unless address is verified to be owned by the user with a standard like `SignInWithEthereum` it should NOT be trusted for showing anything that isn't public info already. A good rule of thumb data from the blockchain is ok. If you have additional data for a user, authenticate before display.

Off Browser

GraphQL (optional but common) — Originally a way of allow the client to specify which the data server would give it. GraphQL has been adopted by web3 for getting analytical type data via a tool called TheGraph. DApps make their own subgraph and index their smart contracts to show users info like change in Total Value Locked over time etc. Note the connection from browser to GraphQL server will be read only.

Full Node The "server" that the frontend calls to interact with smart contracts on the chain. Most dApps use a node as a service although some also run their own. Whereas Web2 dApps communicate with Restful JSON APIs over http and web-sockets, Dapp communication with the full node uses JSON RPC calls over http or web-sockets.

```
# whitepapers.md:
```

```
---
```

```
title: Celo Whitepapers
```

```
description: Overview of Celo Whitepapers describing Celo's protocol, economics, and social impact.
```

```
---
```

Whitepapers

Overview of Celo Whitepapers describing Celo's protocol, economics, and social impact.

```
---
```

Protocol

Celo Whitepaper: A Multi-Asset Cryptographic Protocol for Decentralized Social Payments

- Read paper
- 阅读

Plumo: Towards Scalable Interoperable Blockchains Using Ultra Light Validation Systems

- Read paper

Economics

An Analysis of the Stability Characteristics of Celo

- Read paper

Influencing the Velocity of Central Bank Digital Currencies

- Read paper
- Lee el informe

Shaping the Future of Digital Currencies

- Read paper

Social Impact

Future-Proof Aid Policy

- Read paper
- Exec Summary

Delivering Humanitarian COVID Aid using the Celo Platform

- Read paper

builders.md:

title: Celo Builders

Whether you're just starting or are an experienced developer, Celo offers a variety of ways for you to engage and grow your projects.

Builders in the Celo Ecosystem

The Celo Builder Community is a global network of developers, entrepreneurs, and enthusiasts dedicated to building a regenerative digital economy that promotes prosperity for all. By joining, you'll have the opportunity to create onchain apps and tools that drive financial inclusion and support sustainable development on a global scale.

How to Get Involved:

There are several ways to get started as a builder on Celo:

1. Join the Builder Community: Connect with like-minded individuals on platforms like Discord and the Celo Forum. You'll find channels dedicated to everything from smart contract development to mobile-first dApp creation.

2. Participate in Hackathons: Hackathons are an excellent way to challenge your skills, collaborate with other developers, and potentially win rewards for innovative solutions. Find ecosystem events on our community event platform.
3. Contribute to Open Source Projects: You can contribute directly to Celo's open-source codebase or build complementary projects.
4. Learn through the Celo Academy: For those new to blockchain development, Celo Academy provides tutorials and courses that guide you from deploying your first contract to advanced topics.
5. Apply for Grants: Celo has grants programs, like Prezenti and Celo Public Goods which supports projects that align with its mission to provide financial services to the next billion people.
6. Apply for an Accelerator: Check out Celo Camp, the accelerator focused on the Celo ecosystem. If you are not yet at that stage, they also offer Startup Pathway a program, that leads you through your first steps to becoming a founder.
7. Access Exclusive Resources: Register as a Celo Builder to gain access to resources and support available to active builders on Celo.

contributors.md:

title: Open Source Contribution on Celo

description: How to contribute to the Celo ecosystem as a member of the community.

```
import ColoredText from '/src/components/ColoredText';
```

Code Contributors

How to contribute to open source projects and further the growth of the Celo ecosystem.

How to Contribute

Contributing to the Celo ecosystem through open-source projects is a valuable way to support public goods and connect with the community. Whether you're new to open-source or an experienced contributor, the guidelines below will help you get started.

:::tip

For quick questions, check the docs or create a ticket in the Celo Discord. Please avoid filing an issue on GitHub just to ask a question; using the resources above will provide faster responses.

:::

Prerequisites

To contribute to Celo, the following accounts are necessary:

- `<ColoredText>GitHub:</ColoredText>` Required for raising issues, contributing code, or editing documentation.
- `<ColoredText>Discord:</ColoredText>` Required for engaging with the Celo community.

Repositories to Get Started

Browse the code, raise an issue, or contribute a pull request.

- Monorepo GitHub Page
- Celo Client GitHub Page
- Celo Developers Page

Look for issues that are tagged as "good first issue", "help wanted", or "1 hour tasks". These labels will help you find appropriate starting points. If you want to dive deeper, explore other labels and TODOs in the code.

To work on an issue:

1. Reach out to the repository maintainer to assign you to the issue.
2. Add a comment outlining your plan and timeline.
3. If someone is already assigned, check with the repo maintainer if they are still working on it.
4. Ensure no duplicate issues exist for the work you're planning.

Submitting Issues

If you're interested in creating a new issue, first explore existing projects and ensure that the issue doesn't already exist. When submitting a new issue, follow these guidelines:

1. Ensure the issue is placed in the correct repository.
2. Provide a clear and specific title.
3. Include a comprehensive description outlining the current and expected behavior.
4. Add relevant labels to categorize the issue.

Tasks range from minor to major improvements. Based on your interests, skillset, and level of comfort with the code-base feel free to contribute where you see appropriate. Our only ask is that you follow the guidelines below to ensure a smooth and effective collaboration.

Contribution Workflow

Celo uses a standard "contributor workflow" where changes are made through pull requests (PRs). This workflow enables peer review, easy testing, and social collaboration.

Following these guidelines will help ensure that your pull request (PR) gets approved. Each protocol may have its own specific guidelines, so review them before contributing. Celo-specific contribution guidelines can be found [here](#).

To contribute:

1. [Fork the repository](#). Make sure you also add an upstream to be able to update your fork.
2. [Clone your fork](#) to your computer.
3. [Create a topic branch](#) and name it appropriately. Starting the branch name with the issue number is a good practice and a reminder to fix only one issue in a Pull-Request (PR).
4. [Make your changes](#) adhering to the coding conventions described below. In general a commit serves a single purpose and diffs should be easily comprehensible. For this reason do not mix any formatting fixes or code moves with actual code changes.
5. [Commit your changes](#) see [How to Write a Git Commit Message](#) article by Chris Beams.
6. [Test your changes locally](#) before pushing to ensure that what you are proposing is not breaking another part of the software. Check the repository for the needed tests. Your PR should contain unit and end-to-end tests and a description of how these were run.
7. [Include changes to relevant documentation](#). You should update the documentation based on your changes.
8. [Push your changes](#) to your remote fork (usually labeled as `origin`).
9. [Create a pull-request \(PR\)](#) on the repository. If it's not ready to review, make it a [Draft PR](#). If the PR addresses an existing issue, include the issue number in the PR title in square brackets (for example, `[#2374]`).
10. Provide a [comprehensive description](#) of the problem addressed and changes made. Explains dependencies and backwards incompatible changes.
11. [Add labels](#) to identify the type of your PR. For example, if your PR fixes a bug, add the "bug" label.
12. If the PR address an existing issue, comment in the issue with the PR number.
13. [Ensure your changes are reviewed](#). Request the appropriate reviewers. When in doubt, consult the CODEOWNERS file for suggestions. Let the project you are contributing to know in the issue comments on GitHub or using the Discord sever chat channels that your PR is ready for review. If you are a maintainer, you can choose reviewers, otherwise this will be done by one of the maintainers.
14. [Make any required changes](#) on your contribution from the reviewers feedback. Make the changes, commit to your branch, and push to your remote fork.
15. [When your PR is approved, validated](#), all tests pass and your branch has no conflicts, it can be merged. Again, this action needs to be done by a maintainer - usually the same person who approves will also merge it.

You contributed to Celo! Congratulations and Thanks!

:::tip

If you've commented on an existing issue and have been waiting for a reply, or want to message us for any other reason, please use the Celo Forum or Discord.

:::

daos.md:

title: Celo Regional DAOs

import ColoredText from '/src/components/ColoredText';

Regional DAOs are community-driven organizations that operate autonomously, focusing on local development and empowering communities to use and build on Celo. Explore the many ways to get involved.

How Regional DAOs Are Furthering Celo's Mission

Regional DAOs have been an essential part of Celo's growth. They onboard builders, developers and users into the ecosystem by hosting IRL events, providing mentorship and governance.

Each regional DAO has a different focus which might change over time, but the general idea is to further Celo's mission for Prosperity for All on the ground.

Celo Africa DAO

Celo Africa DAO, <ColoredText>launched in April 2023 </ColoredText> has been building a huge network of builders and founders on the ground in Africa and was able to maintain and grow it through showing up consistently. If you live in one of the listed countries below, make sure to connect and to try joining one of the IRL events. Read up on the reports and proposals in the <ColoredText>Celo Forum </ColoredText>.

Reach out to the main DAO or the chapters on <ColoredText>Twitter </ColoredText> or <ColoredText>Telegram </ColoredText>.

Local Chapters

- Celo Kenya
- Celo Nigeria

- Celo Uganda
- Celo South Africa
- Celo Ghana

Celo Europe DAO

Celo Europe DAO launched in [June 2023](#), has been hosting events and fostering the community around ReFi and RWA protocols as well as Celo Gather and supporting other Ecosystem DAOs with similar events. Read up on the reports and proposals in the [Celo Forum](#).

- [Twitter](#)

Koh Celo (Celo Thailand DAO) DAO

Koh Celo has been launched in the [beginning of 2024](#), starting off by proposing a program for the Road to DevCon 2024. Read up on the reports and proposals in the [Celo Forum](#).

- [Twitter](#)

CeLatam

CeLatam has been launched in the [April 2023](#), starting off by proposing a program for the Road to DevCon 2024. Read up on the reports and proposals in the [Celo Forum](#).

- [Twitter](#)

Get Involved with Regional DAOs

By participating in a Regional DAO, you can contribute to the growth of the Celo ecosystem in your local area. Each DAO offers opportunities for developers, entrepreneurs, and community members to collaborate, share ideas, and drive impactful projects. Explore your region's DAO to see how you can get involved and help further Celo's mission of financial inclusion and sustainability.

For more information on how to join or collaborate with a Regional DAO, visit the [Celo Forum](#) or [Discord](#) to connect with the community.

governance.md:

```
---
title: Governance
---
```


This is an overview of Celo governance and how the network is managed using the stakeholder proposal process. Celo ecosystem is evolving and this guide includes Celo network management as well as community funding proposals.

What is Celo Governance?

Celo uses a formal onchain governance mechanism to manage and upgrade the protocol such as for upgrading smart contracts, adding new stable currencies, or modifying the reserve target asset allocation. All changes must be agreed upon by CELO holders. A quorum threshold model is used to determine the number of votes needed for a proposal to pass.

Stakeholder Proposal Process

Changes are managed via the Celo Governance smart contract. This contract acts as an "owner" for making modifications to other protocol smart contracts. Such smart contracts are termed governable. The Governance contract itself is governable, and owned by itself.

The change procedure happens in the following phases:

1. Proposal
2. Approval
3. Referendum
4. Execution

:::info

Please follow this guide to create a proposal, but make sure, to go through this page, to fully understand the process before.

:::

Overview of Phases

Each proposal starts on the Proposal Queue where it may receive upvotes to move forward in the queue relative to other queued proposals. Proposal authors should work to find community members to upvote their proposal (proposers may also upvote their proposals). Up to three proposals from the top of the queue are dequeued and promoted to the approval stage automatically per day. Any proposal that remains in the queue for 4 weeks will expire.

- Approval lasts 1 days (24 hours), during which the proposal must be approved by the Approver(s). Approved proposals are promoted to the Referendum stage.
- Referendum lasts five days, during which owners of Locked Celo vote yes or no on the proposal. Proposals that satisfy the necessary quorum are promoted to the execution phase.
- Execution lasts up to three days, during which anybody may trigger the execution of the proposal.

Proposal

Any user may submit a Proposal to the Governance smart contract, along with a small deposit of CELO. This deposit is required to avoid spam

proposals, and is refunded to the proposer if the proposal reaches the Approval stage. A Proposal consists of a list of transactions, and a description URL where voters can get more information about the proposal. It is encouraged that this description URL points to a CGP document in the celo-org/celo-proposals repository. Transaction data in the proposal includes the destination address, data, and value. If the proposal passes, the included transactions will be executed by the Governance contract.

Submitted proposals are added to the queue of proposals. While a proposal is on this queue, voters may use their Locked Celo to upvote the proposal. Once per day the top three proposals, by weight of the Locked Celo upvoting them, are dequeued and moved into the Approval phase. Note that if there are fewer than three proposals on the queue, all may be dequeued even if they have no upvotes. If a proposal has been on the queue for more than 4 weeks, it expires and the deposit is forfeited.

Types of Proposals

Governance Proposals must fall within one of the following categories to be considered acceptable.

Proposal Type	Governance Platform	Description	Submission Requirements	Quorum	Approval Threshold
---	---	---	---	---	---
Celo Protocol Governance	Celo Governance Contracts	Celo Network decisions and Celo Protocol Improvements	Deposit of 10,000 Locked Celo.	Dynamic based on the current Celo Algorithm.	Dynamic based on the current Celo Algorithm.
Smart Contract Governance	Celo Governance Contracts	onchain smart contract changes	Deposit of 10,000 Locked Celo.	Dynamic based on the current Celo Algorithm.	Dynamic based on the current Celo Algorithm.
Celo Community Treasury Governance	Celo Governance Contracts	Funding proposals that do not fall within a current Celo Public Good Budget or aim to request over \$500,000 in value in a single proposal.	Deposit of 10,000 Locked Celo.	Dynamic based on the current Celo Algorithm.	Dynamic based on the current Celo Algorithm.
Mento Governance	Celo Governance Contracts	Mento reserve and protocol decisions. To separate once, Mento will establish their own Governance system in 2024.	Deposit of 10,000 Locked Celo.	Dynamic based on the current Celo Algorithm.	Dynamic based on the current Celo Algorithm.
Celo Public Goods Governance	Celo Public Goods Snapshot	Program selection within approved Celo Public Goods budgets.	Minimum of 10,000 Locked Celo Balance	2.5M Celo	50%

Feedback and Review

Proposals must be posted in the Celo Forum for review by the Celo community. It is required to post the proposal as a new discussion thread in the Governance category and to mark it with [DRAFT] in the title. Proposal authors are expected to be responsive to feedback.

A proposal needs to be up for discussion for at least 7 full days, during which responsiveness from the author is mandatory.

After a proposal has received feedback and has been presented on the governance call the proposal author shall update the proposal thread title from [Draft] to [Final]. Authors shall also include a summary of incorporated feedback as a comment on their proposal thread so future reviewers can understand the proposal's progress. If feedback was gathered outside of the Forum (e.g., on Discord), proposal authors should include relevant links.

Approval

Every day the top three proposals at the head of the queue are popped off and move to the Approval phase. At this time, the original proposers are eligible to reclaim their Locked Celo deposit. In this phase, the proposal needs to be approved by the Approver. The Approver is initially a 3 of 9 multi-signature address held by individuals selected by the Celo Foundation, and will move to a DAO in the future. The Approval phase lasts 1 day and if the proposal is not approved in this window, it is considered expired and does not move on to the "Referendum" phase.

Referendum

Once the Approval phase is over, approved proposals graduate to the referendum phase. Any user may vote yes, no, or abstain on these proposals. Their vote's weight is determined by the weight of their Locked Celo. After the Referendum phase is over, which lasts five days, each proposal is marked as passed or failed as a function of the votes and the corresponding passing function parameters.

In order for a proposal to pass, it must meet a minimum threshold for participation, and agreement:

- Participation is the minimum portion of Locked Celo which must cast a vote for a proposal to pass. It exists to prevent proposals passing with very low participation. The participation requirement is calculated as a governable portion of the participation baseline, which is an exponential moving average of final participation in past governance proposals.
- Agreement is the portion of votes cast that must be "yes" votes for a proposal to pass. Each contract and function can define a required level of agreement, and the required agreement for a proposal is the maximum requirement among its constituent transactions.

Execution

Proposals that graduate from the Referendum phase to the Execution phase may be executed by anyone, triggering a call operation code with the arguments defined in the proposal, originating from the Governance smart contract. Proposals expire from this phase after three days.

Cool-off period for failed proposals

If a proposal is not accepted, a cool-off period is required for additional conversation and potential changes before the proposal can be resubmitted. There are two situations in which a cool-off period is required:

1. If a proposal is rejected due to not reaching a quorum but having a majority of yes votes, the proposal is moved back to the discussion stage and may be submitted for a vote after waiting for 14 days.

2. If a proposal is rejected and has a majority of no votes, the proposal is moved back to the discussion stage and may be submitted for a vote after receiving approval from the Governance Guardians and waiting for 28 days.

Note: In the event that a proposal meets or exceeds quorum, but if it is not approved in time, then they should be able to re-submit as soon as they are able. This would happen in a rare situations when approvers are unable to approve in the 72 hour window following a referendum vote.

Smart Contract Upgradeability

Smart contracts deployed to an EVM blockchain like Celo are immutable. To allow for improvements, new features, and bug fixes, the Celo codebase uses the Proxy Upgrade Pattern. All of the core contracts owned by Governance are proxied. Thus, a smart contract implementation can be upgraded using the standard onchain governance process.

Upgrade risks

The core contracts define critical behavior of the Celo network such as CELO and Celo Dollar asset management or validator elections and rewards. Malicious or inadvertent contract bugs could compromise user balances or potentially cause harm, irreversible without a blockchain hard fork.

Great care must be taken to ensure that any Governance proposal that modifies smart contract code will not break the existing system. To this end, the contracts have a well defined release process, which includes soliciting security audits from reputable third-party auditors.

As Celo is a decentralized network, all Celo network participants are invited to participate in the governance proposals discussions on the forum.

Validator Hotfix Process

The cadence and transparency of the standard onchain governance protocol make it poorly suited for proposals that patch issues that may compromise the security of the network, especially when the patch would reveal an exploitable bug in one of the core contracts. Instead, these sorts of changes are better suited for the more responsive, and less transparent, hotfix protocol.

Anyone can make a proposal in the hotfix protocol by submitting the hash of their proposal to the Governance smart contract. If that hash is approved by the approver and a quorum of validators, the proposer can execute the contents of that proposal immediately.

Note that this means the validators may not always know the contents of the proposal that they are voting on. Revealing the contents of the proposal to all validators may compromise the integrity of the hotfix

protocol, as only one validator would need to be malicious in order to exploit the vulnerability or share it publicly. Instead, to convince the validators that the hash represents a proposal that should be executed via the hotfix protocol, the proposer should consider contacting reputable, third party, security firms to publicly vouch for the contents of the proposal.

Celo Blockchain Software Upgrades

Some changes cannot be made through the onchain governance process (via proposal or hotfix) alone. Examples include changes to the underlying consensus protocol and changes which would result in a hard-fork.

Governance Tooltik

Mechanisms for Main Onchain Celo Governance Proposals

Celo Governance Contract: The onchain voting contract for Celo Governance. This is also the address of the Celo Community Treasury.

Celo Mondo: The new UI friendly interface to Lock, Stake, Delegate and Vote.

Celo.Stake.id: A front-end interface for Celo Governance maintained by Staking Fund.

Celo Terminal: A desktop application allowing Celo chain interactions and governance participation.

StakedCelo dApp: An application that allows for liquid staking of Celo and voting on Governance proposals.

Mechanisms for Celo Public Goods Proposals

Celo Public Goods Snapshot: A locked Celo snapshot to allow votes to occur on Snapshot to decide about Celo Public Goods Proposals.

Mechanisms for Discussions

The Celo Forum: The platform for governance and community discussion.

Discord: For informal governance discussion and feedback.

Github: Governance guidelines and CGP proposals are tracked via Github.

Implementation and Administration

Celo Governance represented by Celo Governance Guardians and can help answer any questions about the governance process.

Celo Governance Guardians Overview

The current Celo Governance Guardians (Formerly known as CGP Editors), actively participating in the Governance Process.

- Guardian: 0xj4an (@0xj4an-work, Twitter)
- Guardian: Wade (@Wade, Twitter)
- Guardian: 0xGoldo (@0xGoldo, Twitter.)
- Advisors Guardians:
 - Eric (@ericnakagawa, Twitter)
 - Anna (@annaalexa, Twitter)

overview.md:

title: The Celo Ecosystem

Explore the many ways to engage with the Celo ecosystem and contribute to its growth.

As a User

- Explore Projects on Celo: Start using and engaging with Celo apps.

As a Developer

- Join our builder community: Connect with other builders in the ecosystem.
- Contribute to open source projects: Help grow Celo by contributing to key projects.
- Get involved in a local chapter: Attend in-person workshops for mentoring and support.
- Participate in Celo Public Goods: Explore ongoing funding rounds.

As a Founder

- Register as an active Celo builder: Access exclusive resources and support.
- Get your page added to our ecosystem page managed by Celo Public Goods.
- Introduce your project in the Celo Forum in the founders' category.
- Apply to Celo Camp: Join the accelerator focused on scaling apps on Opera MiniPay.
- Engage with Celo DAOs: Gain user feedback, access networking opportunities, and validate your project.
- Apply for Prezenti Grants: Explore grant opportunities.
- Check out the Grant Playbook and Fundraising Guide for helpful tips to raise capital.

As a Contributor

- Participate in governance: Engage in current discussions and connect with the ecosystem.
- Contribute to local chapters: Connect with the community, offer support, and mentor others.

Social Media

Follow to stay updated with the latest news about Celo.

- Official Celo Twitter
- Celo Devs Twitter
- cLabs Twitter

- Warpcast
- Reddit
- GitHub
- Medium Blogs
- LinkedIn
- Instagram
- YouTube
- Twitch

Discussions

Ask questions, find answers, and connect with the community.

- Celo Developer Chat on Discord
- Celo Official Telegram
- Celo Forum
- Celo Subreddit
- Celo Website
- Host a Meetup

create-proposal.md:

```
---
title: Create Proposal for Governance
description: How to Create a Proposal in Celo Governance- A Detailed Guide
---
```

Prerequisites

Before diving into the Celo governance process and creating a proposal, there are several prerequisites that need to be met:

1. Celo CLI Knowledge: Familiarity with the Celo Command Line Interface (CLI) is crucial. The CLI is the primary tool for interacting with the Celo network, including proposal submission.
2. Minimum Amount of CELO Needed: Proposers must have at least 100 CELO tokens. A token requirement ensures that proposers have a stake in the network and helps to prevent spam proposals. The staked CELO will be refunded to the proposer if the proposal reaches the Approval stage. If a proposal has been on the queue for more than 4 weeks, it expires and the deposit is forfeited.
3. Multi-Signature Wallet: If the proposal requests funds from the treasury, receipt of the funds into a multisig wallet is advisable. Multisig signers are also advised to self-identify on the corresponding Forum post. Multisig wallets add a layer of security and trust, as multiple parties must agree to execute transactions. Multiple parties self-identifying their involvement in a wallet also demonstrates more oversight over the usage of any requested funds.

Life Cycle of a Proposal on Celo

Step 1: Drafting the Proposal

The initial phase in the lifecycle of a governance proposal is the drafting stage. Here, you must comprehensively outline the proposal's purpose, scope, and impact. This should include:

- Objective: Clearly state what the proposal aims to achieve.
- Rationale: Explain why this proposal is necessary and the problems it addresses.
- Technical Specifications: If applicable, provide technical details or code changes.
- Budget and Funding: Outline any financial requirements, including a detailed breakdown of costs.

Setting up a secure multisig wallet is recommended for proposals requesting funds, as it ensures enhanced security and trust within the community.

Step 2: Posting the Proposal on Celo Forum

Once your proposal is drafted, post it on forum.celo.org to initiate community discussion. This post should:

- Detail the Proposal: Share every aspect of the proposal, leaving no ambiguity.
- Include Multisig Information: If requesting funds, provide the multisig wallet details.
- Solicit Feedback: Encourage community input to refine and improve the proposal.
- Respond and Iterate: Actively engage with the community, addressing queries and incorporating feedback to strengthen the proposal.

Step 3: Applying for the Governance Community Call

To further socialize your proposal, apply to present it during the Celo Governance Community Call by:

- Booking a Slot: Comment on the GitHub governance repo issue for the next upcoming Governance call to reserve your presentation slot.
- Join a Governance Call Discussion: Discuss your proposal in depth and answer questions from the community and approvers.

Step 4: Publishing the Proposal on GitHub

After refining your proposal through community feedback, the next step is to formalize it by publishing on GitHub. This involves:

- Create a Pull Request (PR): Submit your proposal as a PR to the [celo-org/governance](https://github.com/celo-org/governance) repository. This should include the proposal in markdown format and any associated code in a JSON file. Review previous proposal that are similar to your proposal for hints on how the code should be structured.

- Celo Governance Proposal (CGP) Editors' Review: CGP editors will review your submission to ensure it meets the required standards and provide feedback or request changes if necessary. The CGP Editors will assign an id.
- Acceptance of PR: Once the CGP editors approve the PR, it signals that your proposal is ready to be submitted on-chain.
- On-chain Submission: Accepted proposals can then be submitted on-chain through the Celo CLI for formal voting by the community.

:::danger

Wait for PRs to be merged on proposals prior to submitting on chain

:::

Step 5: Formal Proposal Submission On-Chain

The formal submission of your proposal to the blockchain will be covered in a separate section but involves using celocli to submit your proposal on-chain for approval and voting.

:::info

Additionally, when submitting on-chain: the CGP markdown title will not update on celo.stake.id. While the body of the document can be edited if necessary, it is best to ensure all details are correct before submission.

:::

Step 6: Voting

After submission, your proposal enters the voting phase, which consists of:

- Upvoting: If multiple proposals are submitted at the same time in a 24 hour period, then they must receive upvotes. Anyone, including yourself, can upvote proposals. The highest upvoted proposal moves automatically to the Referendum stage.
- Approval Voting: Approvers review the proposal for any security risks or potential harm to the network. They may follow up with questions so ensure you are easy to contact via your profile on forum or via a connection with a CGP Editor.
- Referendum Voting: Once a proposal is in Referendum stage, all CELO holders may vote on the proposal. They can vote "Yes" to support the proposal, "No" to reject the proposal, and "Abstain" to acknowledge but defer the vote to the remaining community.
- Quorum: For a proposal to pass successfully, the total number of CELO voted must meet or exceed Quorum. "Yes", "No", and "Abstain" votes all count towards quorum. A successfully proposal must receive a 60% majority of votes above quorum. Quorum needed for proposals is dynamic and depends on previous number of votes on recent proposals.

- Conclusion: Approvers have until the proposal's deadline to approve any passing proposals. Once approved, "Yes" votes exceed 60% of necessary quorum, then they may be Executed.

Step 7: Execution

If the proposal passes the voting phase, it moves to execution. This involves enacting the changes or transferring the funds as outlined in the proposal. Detailed steps for execution will be provided in a subsequent section.

By following these steps, your proposal will go through the necessary stages from conception to execution in the Celo governance process. Proposals must be executed within 3 days from the referendum stage or they will be rejected automatically by the system.

:::info

Warning: Anyone on the network can execute a successful proposal at any time.

:::

Creating On-Chain Proposal

Step 1: Create the Proposal File (mainnet.json)

In your proposal file in Governance repository that is either merged or waiting to be merged, create a folder with your proposal number inside the CGPs folder and create a file called mainnet.json. Check this for example -

Inside this file, paste the following content:

```
json
[
  {
    "contract": "GoldToken",
    "function": "approve",
    "args": [
      "0xE1061b397cC3C381E95a411967e3F053A7c50E70",
      "59803140000000000000000000000000"
    ],
    "value": "0"
  }
]
```

:::info

💡 If requesting funds from Treasury

:::

```

json
{
  "contract": "StableToken",
  "address": "0x765DE816845861e75A25fCA122bb6898B8B1282a",
  "function": "increaseAllowance",
  "args": [
    "0x71f433514957d00287A9d33Da759f1e0C1732381",
    "1700000000000000000000000000000000"
  ],
  "value": "0"
}

```

:::info

💡 If making contract call from Governance Contract

:::

Make sure to replace your address and the amount of CELO you want to approve. Save this file and add it to CGPs folder in the Governance repository.

Step 2: Submit the Proposal On-Chain

After your PR is merged, we can submit the proposal on-chain.

Step 2.1: Install celocli

In your terminal, run the following command to install the Celo CLI:

```

bash
npm install -g @celo/celocli

```

Step 2.2: Target the json file

We will submit the proposal using the mainnet.json file you created earlier. To do this, in your terminal:

```

bash
cd governance // repository folder
cd CGPs
cd cgp-(your proposal number)
cat mainnet.json

```

Once you see the content of your mainnet.json file in the terminal output, you can submit the proposal using:

```

bash
celocli governance:propose --jsonTransactions=mainnet.json --
deposit=10000e18 --descriptionURL=https://github.com/celo-

```

```
org/governance/blob/main/CGPs/cgp-<YOURPROPOSALIDINGITHUB>.md --  
from=<YOURADDRESS> --privateKey=<PRIVATEKEY>
```

Replace the --descriptionURL, --from fields with your proposal Github file URL.

:::info

! Note that 10,000 CELO tokens are required in the account to submit a proposal. This amount will be refunded to the proposer if the proposal reaches the Approval stage. If a proposal has been on the queue for more than 4 weeks, it expires and the deposit is forfeited.

:::

:::info

! You can see your proposal ID in the terminal output. Save this ID for future use. To see your proposal in detail, run the following command in your terminal: `celocli governance:show --proposalID <number>`

:::

Step 3: Execute the Proposal

Once the proposal passes the series of votes, you need to execute it. Connect your ledger to your computer and run the following command in your terminal:

```
bash  
celocli governance:execute --proposalID <number> --from=<address> --  
privateKey=<PRIVATEKEY>
```

Replace number with the proposal ID.

Best Practices for Creating a Proposal

- **Clarity and Justification:** Ensure your proposal is clearly written, with straightforward language and a strong justification for why it's needed. Provide as much detail as possible about what you are proposing and why it is beneficial for the Celo ecosystem.
- **Community Engagement:** Engage with the community early on. Seek feedback and address concerns before formal submission. This not only improves your proposal but also builds community support.
- **Security Assessment:** If your proposal involves smart contract code, conduct a thorough security audit. Include the audit report in your proposal to increase credibility and trust.
- **Transparency:** Be transparent about your affiliations and intentions. If your proposal involves funding, detail how the funds will be used.
- **Follow Governance Structure:** Adhere strictly to the governance process as laid out by Celo, including any templates or formats required for proposals.

What to Expect in Governance Calls

- **Presentation Slot:** Be prepared to present your proposal succinctly. Typically, you may be allocated a specific time slot, such as 5 minutes for presentation and 5 minutes for Q&A.
- **Technical Questions:** Expect technical questions from the community, especially if your proposal involves code changes. Be ready to explain complex concepts in accessible language.
- **Community Feedback:** Governance calls are an opportunity for the community to provide direct feedback. Take notes and be open to incorporating this feedback into your proposal.
- **Approval Indicators:** Use these calls as a temperature check on your proposal's likelihood of passing. Positive engagement and constructive feedback are good indicators.
- **Networking:** These calls are an excellent opportunity to network with other members of the Celo ecosystem, which can be valuable for building future support.

Each of these elements is important for navigating the governance calls effectively and maximizing the chance of your proposal's success.

FAQ Section for Celo Governance

1. What is a multisig wallet, and why is it recommended for proposals?
 - A multisig wallet requires multiple signatures to authorize transactions, providing increased security and trust for proposals involving treasury funds.
2. Who are the Governance approvers?
 - Governance approvers are individuals or entities with the authority to review and approve proposals before they go to a community vote, ensuring they meet certain criteria and standards.
3. Who are the CGP editors?
 - CGP (Celo Governance Proposal) editors are responsible for reviewing and managing the content of governance proposals submitted on GitHub to ensure clarity, completeness, and adherence to the format.
4. How to reach CGP editors?
 - To reach CGP editors, you can use the Discord channel. They are available to assist with questions and provide guidance on the proposal process.
5. Can I submit the proposal again?
 - If a proposal is rejected or needs significant revisions, it may be resubmitted after addressing the community's feedback and making necessary adjustments.
6. What if I made a mistake in the proposal?
 - If you discover a mistake in your proposal, it's important to communicate this to the community and CGP editors as soon as possible. Depending on the stage of the proposal, you may need to withdraw and resubmit it with corrections.

How to create Multisig with Safe and withdraw fund from Treasury?

Step: 1 Log in to your Safe

Log in and click New Transaction on the left hand side of the page. This will bring up a modal, click Contract Interaction.

> Note: Don't use <https://safe.celo.org/> as it is not officially supported. Use official Safe at - <https://safe.global/>

!Governance Safe Creation process step 1

Step 2: Enter ABI

Enter the Celo Proxy Address.

!Governance Safe Creation process step 2

Step 3. Change the ABI to the CELO Contract

Step 2 will auto populate the ABI with the proxy ABI. We will want to change that to the actual CELO ERC20 ABI to access the transferFrom function

> For the image below

>

> 1. This is the Celo Token Proxy

Address(0x471ece3750da237f93b8e339c536989b8978a438)

> 2. The ABI of the CELO Token Contract

> 3. This is where our contract interaction is being sent to and this will be the CELO Token proxy.

> 4. The method we want to interact with(We specify the transferFrom)

!Governance Safe Creation process step 3

Step 4 Fill in the the appropriate addresses

You will now need to add the addresses you are wishing to interact with

> For the image below:

>

>

> 5. from(address): This is the Celo Governance

Contract(0xD533Ca259b330c7A88f74E000a3FaEa2d63B7972)

> 6. to(address): This is the address you are wishing to transfer funds from the Governance contract to. It could be the multisig you are using to do this interaction from or any other address.

> value(uint256): The wei amount of funds you are wishing to transfer. eg
1 CELO = 10 18

> you can use this webpage to easily convert CELO to the wei value

!Governance Safe Creation process step 4

Step 5: Create Transaction

Click Add Transaction, and after that other multisig signers will need to confirm the transaction. Once the required number of signers have confirmed the transaction, it will be executed. You can also add a

description to the transaction to help other signers understand the purpose of the transaction.

fundraising.md:

title: Fundraising

description: Hitchhiker's Guide to Fundraising in the Celo Ecosystem.

Fundraising

Hitchhiker's Guide to Fundraising in the Celo Ecosystem.

Overview

The process of raising capital for your company is often opaque and intimidating especially for first time founders. It's a skill that is acquired through practice and repetition. If you're new to fundraising, you'll find that you're crafting a product to sell. You're the seller. The product is your narrative around yourself, your team, company vision and upside potential. The investors are the buyer.

We created this guide to provide Celo Ecosystem builders an end-to-end resource on fundraising best practices as well as a map of Celo's Investor Ecosystem. While our goal isn't to reinvent the wheel and rather it's to distill a few key takeaways from industry experts, we hope you can leverage this guide and included resources to help you navigate your journey. Web3 is an emerging category, but we trust that many of the battle tested methods that have shown success can be applicable in this category.

Expert Resources

To get you started, here are a few comprehensive fundraising resources that we've found to be extremely helpful. This guide attempts to abstract and simplify important takeaways from these resources for entrepreneurs who are newer to fundraising.

- YCombinator's Geoff Ralston - A Guide to Seed Fundraising
- Paul Graham's - How to Raise Money
- NFX's Gigi Levy-Weiss - The Non Obvious Guide to Fundraising
- Carta's Paige Smith - When To Raise Money for A Startup
- Founder Collective's - A Ridiculously Detailed Fundraising Guide

Raising Capital

When should I start thinking about raising capital?

This is a tricky question that has no single right answer. The fact that the answer varies based on the entrepreneur and market shows why this

process is opaque. In general, you should keep these factors in mind when considering the optimal time to start your fundraising process:

Warm Relationships

In some cases, founders may have no trouble raising capital with just an idea. Perhaps they may have warm relationships with the investors who can quantify the entrepreneurs' track record. Frankly speaking, entrepreneurs in this category have more flexibility on when they can raise capital given the established relationships.

No Warm Relationships

In most situations where you're a first time founder and may not have pre-existing relationships with investors, they'll generally prefer that you have a vision, some form of a tangible product, attractive market size and early signs of product demand before engaging. Once you've achieved most of the latter, it may be a good time to start thinking about raising capital to accelerate your growth and capture more of your target market.

Market Timing

It's also important to zoom out and consider the macroeconomic cycle and market timing of your potential fundraise. VC is an asset class that has been flooded with capital as investors continue to search for yield. <https://pitchbook.com/news/articles/2021-us-vc-fundraising-exits-deal-flow-charts>. Fundraising activity in Web3 is no exception and continues to hit all time highs <https://www.cnbc.com/2022/02/02/crypto-start-ups-raised-huge-venture-funding-rounds-in-january.html>. At certain points in the market cycle compared to others, it will be easier to raise capital given investors' need to deploy the large funds they've raised from their limited partners and the chase for "hot sectors". You may have an advantage and more options for capital if you fundraise during these "hot" market cycles. Additionally, it's worth keeping track of investors on social media to understand what they're interested in funding at the moment.

Seasonality

Ensure to time your fundraising process that aligns well when investors are actively investing and have the capacity to run full due diligence processes. As a rule of thumb, try your best not to start a fundraising process around major holiday seasons to reduce the risk of extending your timeline longer than you had originally planned.

What do I need to research before starting my fundraise?

While fundraising can take an unexpected amount of time, it's important to optimize for efficiency to the best of your ability. The last thing you want to do is spend hours with potential investors who aren't investing in your stage/sector of business. We're aware of emerging fundraising models in Web3 (i.e. DAO-based funds), however our aim is to keep this guide confined to more traditional methods for simplicity.

1. Find investors who fit your target investor profile! Using the following databases/resources listed below, you can easily identify the investors who are actively investing in your stage/sector/geography.

- Web3 Focused

- Dove Metrics (<https://www.dovemetrics.com/> is an excellent resource for Web3 market fundraising data)

- Investor Databases

- AngelList <https://www.angellist.com/>
 - Crunchbase <https://www.crunchbase.com/>
 - First Round Capital's Angel Directory <https://angels.firstround.com/directory>
 - NFX Signal <https://signal.nfx.com/>

2. Ensure that the investors you identify are actively investing. For example, try finding investors who have made 1-2 relevant investments within the last 2-4 months using the databases below.

3. Once you've identified a subset of investors who may be a fit for your sector/geography/stage, aggregate a list in your personal database to stay organized. Some popular tools include pre-built templates on Airtable or even creating your own tracker on Google Sheets. As your number of conversations grows, it's important to keep track of the status of each conversation and to understand where you should be doubling efforts to increase your chances of success.

4. If one of your target investors is a venture capital fund, take some time to look through their website as well to look for a few things

- If they show their portfolio companies, take some time to review and see if they've invested in any competitors. There may be a conflict of interest even if they are a sector/stage/geographical fit for you and you want to cross out these investors quickly.
- Research the team and see who may be the right person to reach. Venture capital funds often have a broad team and there may be certain individuals who are focused on your sector vs. others at the firm. You want to be sure that you're reaching out to the right person.

5. If one of your target investors is an angel investor, take some time to research their background and personal investment thesis. Do they have attractive experience that you could leverage to grow your company?

6. Frankly speaking and similar to sales, cold emails to investors have a low chance of success statistically speaking. For any investors who you identify to be a prospective fit, search through your social networks and see if you may have any mutual connections. People tend to trust others who they have some history with. If you find any mutual connections, try kindly reaching out to the individual and see if they'd be willing to forward along your company's teaser materials to the prospective investor.

7. If you've found a great venture capital firm and/or angel investor but have no mutual connections to the team or individual, you can attempt a cold outreach. Remember, fundraising at the earliest stages will be a numbers game. However, you also want to be as thorough and thoughtful as possible when reaching out to the investors. In sales, unique approaches can attract a few eyeballs even if they hit the inbox cold. A few things to keep in mind

- Keep the outreach email as concise as possible. Investors receive a ton of inbound in this market so you'll only have a minute if not a few seconds of their attention. You want to get straight to the point.
- Specify why you reached out to them and why they would be a great fit for your company. (I.E. Perhaps they've invested in other companies that are tangential to what you're building or have written an article with a thesis that aligns with your company vision.
- Share a few points on your company taken from your teaser materials and include your investor presentation.
 - What is the problem you're solving for and is it a large market?
 - Why is now the best time to tackle this opportunity?
 - How are you solving this problem?
 - Traction (if applicable and be transparent)
 - How much are you raising?
 - What's special about your team and why are you the ones to solve this problem?

How much capital do I need to raise?

The rule of thumb here is to raise enough capital to give you 1 year of runway at minimum and best case 2 years of runway with the primary goal of reaching your next milestone. In other words, think through how much monthly operating capital you'll need (across hires and other administrative budget items) and multiply that by the number of months you believe it'll take to reach that milestone. This resulting figure can serve as a reference point but it's often helpful to size up and down your potential round size to give yourself and investors a magnitude of what you could realistically achieve with different levels of capital. Finmark (<https://finmark.com/>) is an example of a helpful resource that can help build out your financial plan.

What are the different forms of capital I can raise?

While new forms of funding models emerge in Web3 and a comprehensive list will grow over time, below are a few common investment structures used by founders and investors today.

1. Grants - Many layer 1 protocols have recently set up ecosystem funds in order to incentivize and fund entrepreneurs who build on their technology stack. These grants generally come in the form of non-dilutive capital (meaning that the capital you take does not reduce your ownership of your company) that can help you build a minimum viable product and/or test your market hypothesis.

2. SAFE (Simple Agreement for Future Equity) - YCombinator's SAFE document has become an industry standard used for early stage fundraising and for good reasons - they provide speed and flexibility for both investors and entrepreneurs. They're a form of a convertible note however, unlike convertible debt, SAFEs remove the need for interest payments and repayment of the debt. The capital invested on a SAFE is generally expected to convert into shares of preferred equity in a priced equity round. YCombinator's website <https://www.ycombinator.com/documents/#about> provides excellent best practices on SAFEs and templates as well.

3 SAFT (Simple Agreement for Future Tokens) - For a period of time during the boom of crypto fundraising, SAFTs were frequently used by companies to raise capital. While most investors expect financial upside on their equity in traditional companies, Web3 companies will often build and launch tokens which are eventually available to the public. Ideally, these tokens have sustainable utility and practical use cases and should therefore accrue value from greater market adoption. Mechanically speaking, SAFTs offer the investors the ability to exchange capital for a promise of discounted tokens from the project at a future point in time. However, SAFTs have largely fallen out of favor due to their legal risks, potential tax implications, and complications that other companies have faced.

4. SAFE + Token Warrants - A combination of a SAFE and a token warrant has become increasingly popular for fundraising in Web3. Unlike SAFTs, these token warrants typically come in the form of an optional side letter and don't guarantee the deployment of tokens - therefore avoiding the legal complications faced by its predecessor. As a result, companies are more commonly raising capital on SAFEs along with a token warrant that offers investors the right to tokens in the amount that can be proportional to their equity ownership of the company. In this manner, investors have exposure to both equity and potential tokens if the companies decide to launch one.

- The team at LiquiFi has written an excellent piece on the topic and provides a practical guidance for token allocation.
<https://medium.com/@liquififinance/crypto-web3-fundraising-with-token-side-letters-or-token-warrants-33dce66f375e>.

- This resource from Cooopahtroopa and Lauren Stephanian is also a must read for entrepreneurs who need to understand historical and current benchmarks on tokenomics.
<https://lstephanian.mirror.xyz/kB9Jz5joqbY0ePO8rU1NNDKhiqvzU6OWyYsbSA-Kcc>.

5. Equity - Equity rounds allow companies to set an official valuation for the company and to issue shares to investors at a specific price. Over time, these rounds are generally reserved for Series A and later stage companies. Companies at the seed stage have raised priced rounds historically, although more of them are raising on SAFEs at the earlier stages for the speed of putting operating capital to work.

What should I have prepared for investor conversations?

In general, we recommend that you prepare as much digestible information ahead of time to make your investor meetings productive. Having the baseline items listed below will serve you well in requesting warm introductions to investors, cold email outreaches and investor preparation prior to an initial conversation.

- Brieflink Page (Using <https://brieflink.com/>)
- 1-2 Paragraph Teaser Summary
- Investor Presentation
- Product Demo (You can pre-record these on <https://www.loom.com/>. In Web3, it may even make sense to have an interactive demo.)

The other items listed below are useful to share with investors if they've expressed interest in pursuing additional due diligence.

- Capitalization Table
- Financial Projections

What do I need in my investor presentation?

While this guide isn't meant to be 100% prescriptive, below are the core components of a comprehensive investor presentation. Remember, treat fundraising as if you were launching and selling a product. You want to craft a compelling enough narrative to make your company an easy choice for investors to fund. Your pitch and deck are always a work in progress and it's important to continuously refine your story and materials based on feedback that you receive from others.

- Company Vision (In a concise and powerful statement, what is your company's core value proposition and what key problem are you looking to solve?)
- Problem (Is this a real problem and is it critical to solve?)
- Market Size (What is your Total Addressable Market? It's okay to think big here but be realistic with data)
- Traction (Are there currently any users? TVL? Growth rates for any relevant metrics?)
- Business Model (In traditional companies, investors want to understand how your unit economics and monetization plans? In Web3, the upside is likely more attributable the future value of a utility token or adoption of a platform)
- Solution (What are the features of your product? How does it ultimately solve the problem? What blockchain is it built on? While Web3 is nascent and can be complicated, you want a simple enough explanation so that investors can communicate to their internal stakeholders as well.)
- Product Roadmap (What other segments can your product expand into?)
- Competitive Landscape (Competitive market map and positioning on why your product is different)
- Distribution Strategy (Are there other players in the ecosystem that you can work with to acquire other users in a low cost way?)
- Team (What is your team's background? Are you and your team the right market & team fit?)
- Fundraising Ask & Planned Use of Capital (How much capital are you looking to raise and how are you planning to allocate that capital? What milestones are you going to achieve with this capital?)

The team at NFX has put together a comprehensive sample pitch deck with relevant examples of slides of pitch decks from other companies.

We can't forget to mention beyond having these components to a pitch deck, it's important to think through the visual representation of your pitch deck as well. Most of you will have an investor presentation on screen while meeting investors and it's important to remember that the engagement is both audio and visual. We've found that <https://www.beautiful.ai/> is a great place to start if you're looking to create a visually compelling pitch deck as well.

Pitching Process

Below are a few best practices and things to consider when preparing for your investor pitches. This isn't all inclusive and your mileage will likely vary depending on your circumstances.

- Do your diligence on the investors prior to pitching in order to:
 - Understand focus areas so that you can curate your pitch accordingly
 - Uncover any red flags and or concerns about the investor that you can address during your meeting. Remember, as much as investors are evaluating you/your company, you're also in the position to evaluate them as a potential partner. Capital is a commodity and true value-add is very rare. A good way to do this is to reach out to founders in their portfolio or a broader entrepreneurial community for honest feedback on the investor
- Be concise about what you do
- Be concise about the problem
- Clearly articulate why this is the right market timing
- Confidently explain why you have the right team
- In Web3 specifically, it's more important now than ever to be able to articulate where the upside for investors will come from
- Let investors guide the conversation so that you can tackle the points that they truly care about. Pitches can generally take 15 minutes. You can leave the rest of the meeting time for an interactive conversation. The best investors will come prepared and ask insightful questions. If they don't, just know that they will likely serve only as a source of capital vs. a source of strategic help
- Be a human! Much of the remote nature of what we do today can feel transactional. It's important to build a genuine relationship with investors as they can be your biggest champion.
- Fundraising is a numbers game. You'll receive many more rejections than you will genuine interest. It's just a part of the process. Cast a wide net (as efficiently as possible), collect and then select.
- Know that investors will ghost you even if they've appeared to express interest during your meeting. They're paid to meet founders, so don't take it personally. Just move on to others!
- Follow up promptly with any requested materials. This indicates your level of responsiveness with future investors
- Be transparent about your process. Investors respect that and will align their process accordingly if they're genuinely interested in working with you

Post Investment

After raising capital from investors, it's important to keep a periodic cadence with your investors for a few reasons. Firstly, it helps keep you accountable to the milestones that you've set for your team and mitigates the need for investors to ask you ad-hoc questions - which can generally distract you as you're busy building your company. Secondly, it's a great open forum to request assistance from your investors. You can also gamify your updates by making special mentions of investors who have been the most helpful to you so far. Founder Collective has put together a detailed template for thorough and actionable investor updates. <https://visible.vc/templates/founder-collective-fill-in-the-blank-investor-update/>

Celo Ecosystem Investors

Who should I go to within the Celo Ecosystem?

If you're building on Celo and looking for fundraising, start with our Celo Ecosystem Investors! The Celo Foundation has partnered with a number of dedicated individuals and institutional investors who look to invest in companies building on Celo. Below you'll find a list of different investor groups who are actively investing in Celo's Ecosystem and additional details on their areas of focus.

!fundraising

Celo Camp

Celo Camp, started in 2022 is an independent initiative managed by our partners at Upright. For founders who have an idea and would like to surround themselves with a community to build with, Celo Camp is a great program to help you get your idea off the ground. Towards the end of the program, investors in Celo's Ecosystem will attend and you'll have the opportunity to pitch them for a potential investment. Make sure to sign up for the next cohort.

Startup Pathway

Startup Pathway is a free online learning program offered by the Celo Camp accelerator. It helps you to get your Web3 venture ready to lead in the Celo ecosystem. Rewards are waiting for you along the way.

Prezenti

Prezenti is the next iteration of the CCF1 and was developed with the support of CCF1. A new onchain spend approval was proposed and accepted resulting in 800,000 Celo able to be deployed to people wanting to build their dreams upon the Celo protocol. It is managed by three new stewards and has four focus areas - Education, Research, Community Tooling and Other. Anyone is welcome to apply for a grant from the Prezenti Website and we use Questbook to distribute the funds and manage the grants.

Celo Community Fund [Closed]

Celo Community Fund I (CCF I) was the first community approved spend proposal from the on-chain Community Fund (explorer). Community members seeking funding could apply to CCF I, which was managed by three community stewards. As of April 2022, the fund is closed due to achieving its objective and disbursing the amount of Celo received.

Celo Foundation Grant Program [Closed]

The Celo Foundation Grants Program is an open program that supports projects that are committed to Celo's mission of building a financial system that creates the conditions for prosperity for all. Applicants to the Celo Foundation Grant program are expected to have a viable product or offering for the community. Admitted applicants will be provided capital and mentorship. To learn more about the program and the application process, please visit this link - <https://celo.org/experience/grants#purpose>.

Celo Scout Investors

Celo Scouts are individual investors who are hand selected by the Celo Foundation to invest in mission-aligned founders building on Celo. These investors tend to focus on early stage companies who may be looking for angel/pre-seed like funding and write check sizes between 25-50K cUSD on average.

Dedicated Institutional Celo Ecosystem Funds

Celo Ecosystem Funds are firms that the Celo Foundation has partnered with to evaluate larger private equity and/or token investments into companies building on Celo.

!fundraising

Verda Ventures

Verda Ventures invests growth capital in category-leading Real World Asset (RWA) companies with profitable moats. As of September 2024 they are working on a new MiniPay fund to accelerate distribution-focused mobile applications. Check their Twitter to get the newest updates.

Flori Ventures

Flori Ventures is a seed stage venture fund already partnered with many projects built with Celo's technology. Their aim is to work with the financial innovators that help foster an inclusive financial system and are closely working with the Celo Foundation to achieve this goal. Co-Founder and Managing Partner, Maria Alegre, was the former Co-Founder and CEO of Chartboost (acquired by Zynga) and raised \$21M in funding from Sequoia Capital and other stellar investors. Co-Founder and General Partner, Tomer Bariach, has a long history as an expert in token economics and is well versed in economically impactful technologies.

- Stage Focus: Pre-Seed & Seed
- Geography Focus: Global
- Sector Focus: Companies that bring crypto to real world use cases & enable prosperity for all
- Average Check Size: \ \$100K

Unicorn Growth Fund

Unicorn Growth Capital is a Seed & Series A venture fund focused on partnering exclusively with founders innovating financial services in Africa. They're particularly excited about FinTech & DeFi infrastructure, embedded finance products and broad digital financial services that can foster an inclusive financial ecosystem.

Stage Focus: Seed & Series A

Geography Focus: Africa

Sector Focus: FinTech & DeFi Infrastructure, Embedded Finance, Financial Services

Average Check Size: \$500K for Seed and < 1.5M for Series A

Lightshift Capital

Lightshift Capital is an early stage fund focused on partnering with groundbreaking companies building infrastructure layers, DeFi protocols and emerging tools expanding use cases for Web3. They currently work with a community of over 50 investors, builders and ecosystems including individuals from Andreessen Horowitz, Digital Currency Group, Celo, Kraken and many others. They have a hands-on approach to working with founders at the earliest stages while leveraging support from their in-house technical and business teams to add value.

- Stage Focus: Pre-Seed & Seed
- Geography Focus: Europe, U.S. and Asia
- Sector Focus: Financial Services, DeFi, Infrastructure and Scalability
- Average Check Size: \ \$250K - 500K

TKX Digital Group

TKX is an Asia focused crypto investment bank that utilizes both active and passive strategies for primary and secondary markets of blockchain technology and cryptocurrency. They provide a full service stack including investment to advisory services to blockchain startups.

- Stage Focus: Pre-Seed & Seed
- Geography Focus: Europe, North & South America
- Sector Focus: GameFi - Infrastructure - Defi - NFT - MPC
- Average Check Size: \$50K-\$500K

Polychain Fund

Polychain Capital is a global early stage venture fund focused on investing in protocol layers and cryptocurrency companies. They separately manage the Celo Ecosystem Venture Fund that aims to foster

Celo's mission of creating a more accessible financial system by making strategic, seed-stage deployments into tools and services which will leverage Celo's protocol to build new financial infrastructure.

- Stage Focus: Seed & Series A+
- Geography Focus: Global
- Sector Focus: DeFi, Cross-chain Interoperability, Intersection of crypto & gaming
- Average Check Size: \$500K - \$5M

Other Prominent Celo Ecosystem Investors

The groups below have also invested in companies in the Celo Ecosystem and may be worth considering if you're looking to raise a larger round of institutional capital. If you're an investor that has invested in Celo's Ecosystem and would like to be included in the list, please reach out to celoinvestorecosystem@celo.org.

!fundraising

How do I get in touch with these investors?

Please reach out to celoinvestorecosystem@celo.org with your investor presentation and a short description of the company, and your round details and we can help get you connected with interested investors in Celo's Investor Ecosystem.

grant-playbook.md:

```
---
title: Celo Grant Playbook
description: Summary of best practices that grantees are encouraged to
follow to further the Celo mission.
---
```

Grant Playbook

Summary of best practices that grantees are encouraged to follow to further the Celo mission.

Code of Conduct

The Celo Foundation believes in investing in projects that share Celo's mission of building an open financial system that creates conditions of prosperity for everyone. Everyone who engages with the Celo ecosystem must abide by the Code of Conduct. Please take the time to read through it with your team.

Communication Guidelines

Social Media Engagement

Creating an ecosystem that supports and encourages each other is important to the Celo community. As a team, please make sure to actively engage in social media.

- Setup a Twitter handle for your project (if you don't have one already)
- Publish (at least) one post per week
- Follow @CeloOrg and @CeloDevs
- Follow projects in your space & fellow grant recipients
- Identify and follow influencers who might be interested in your space. To help you get started, here is a list of influencers we follow.
- Post links to your blog posts, repos, etc
- Use the hashtag #CeloGrants so you can join the conversation about the grant program

Blog

Each project should have a dedicated blog. Two suggestions are devpost or Medium.

- Write a blog post at least once per month/milestone outlining your progress to date (i.e. accomplishments and challenges). Include links to repos, demos, or social media.
- Here are some writing tips:
 - Celo Blogging Guide
 - How to write a killer press release
 - How to write a new product announcement

Engagement & Documentation

Source Code

In the spirit of decentralization, The Celo Foundation expects all funded projects to be open-source by default -- this means that the codebase is public and under a permissive license (we recommend Apache 2.0). If there are any parts that need to be kept private (eg. an admin key), please let us know ahead of time (at grants@celo.org), and document it in your project's README.

To make sure teams are making progress on projects, send us bi-weekly updates to grants@celo.org. The intent is not to create pressure, but to provide structure and guidance in case things get off track. Some suggestions:

- Create a top-level README.md to provide a description of your project, goals, features, roadmap and installation instructions.
- Ensure that all work is regularly committed to a repo. Track and close issues.

Chat

- Make sure to ask questions in a relevant Discord channel. You can join the Celo Discord here. The cLabs protocol and product teams are all there and available for questions. Try not to take conversations to direct

messages -- by having conversations in an open channel, other groups can also benefit from the technical discussions and learnings. You can tag mentors in public channel messages to ensure your messages are seen.

- Please make an effort to remain visible on #general-applications and/or #mobile-development channels in the App Developer section. These channels are dedicated to teams working on projects on the Celo ecosystem. There is a good chance someone has encountered similar challenges. It's also a great way to find ways to integrate with other projects in the space.
- Please make sure your entire team is in this chat room.
- Also please post your tweets and blog posts so that other grantees can share social media.

Milestone Reviews & Payment

Once you have completed a grant milestone, please send an email to your POC at the Celo Foundation. The grant committee will then review to ensure objectives have been met.

After your milestone has been approved by the grant committee, payment will be issued.

Evaluating Grant Success

Before the final payment, grants will be evaluated using the following framework:

- Engagement: How well you follow best practices & engage with the Celo community
- Visibility: Maintaining transparency on your project accomplishments and challenges
- Roadmap: Ability to reference a list of commits that complete your roadmap
- Timeline/Milestones: Ability to meet your timeline and key milestones
- Launch: Successful launch!

guidelines.md:

```
---
title: Celo Contributor Guidelines
description: Join a community of developers, designers, dreamers, and doers building prosperity for everyone.
---
```

Celo Contributor Guidelines

Guidelines for submitting contributions to the Celo community.

Celo is open source and we welcome open participation. We strive to fulfill our Community Tenets by being an open and inclusive community

where everyone feels welcome and empowered to contribute. This also means following some ground rules and abiding by Celo's Code of Conduct.

- Raise an issue in the repository that you would like to contribute to or find one you feel comfortable solving - fill in the requested information and paste your contribution.
- Clone the repo and follow the guidelines below to submit your contribution.
- Create a pull request

How to Contribute

Our community includes a group of contributors who help develop, write, translate, and improve Celo. Anyone is welcome to join the community and contribute their skills to help empower other community members and grow the Celo ecosystem.

- Code
- CIPs

Contributor Guidelines

There are a few basic ground rules for contributing:

- Please fork the repository
- PRs \(\pull requests\) are preferred for solving issues, especially for small changes such as typos. Issues should be used for missing features and for broad-based changes.
- For on-going work, use your own side-branch and not the master branch.
- For non-trivial amounts of work, we encourage you to submit PRs regularly to solicit feedback.
- Please double check your work before submitting it. Submissions with typos, spelling, and grammatical errors may not be merged until fixed.
- Try to remain as objective and fact-based as possible.

Submitting PRs

We encourage you to PR \(\pull request\) your work regularly and often to solicit feedback and to ensure everyone has an idea of what you're working on. If you've just started, we suggest creating a PR with "WIP" \(\Work In Progress\) in the title and let us know when it's ready to review in the comments.

☐ Code

Code Contributors are developers in the Celo community that contribute to the Celo monorepo or the core protocol code. They help improve the protocol and infrastructure by fixing bugs and designing new features that help improve the Celo platform.

How to Get Started

Find an area that is of interest and you would like to help with. Look for issues that are tagged as "good first issue", "help wanted" and "1

hour tasks" to get started. If you'd like to dig deeper, feel free to look at other labels and TODO's in code comments. If there's an issue you're interested in contributing to or taking over, assign yourself to it and add a comment with your plans to address and target timeline. If there's already someone assigned to it, please check with them before adding yourself to the assignee list.

Tasks range from minor to major improvements. Based on your interests, skillset, and level of comfort with the code-base feel free to contribute where you see appropriate. Our only ask is that you follow the guidelines below to ensure a smooth and effective collaboration.

Please make sure your PR

- Requests the appropriate reviewers. When in doubt, consult the CODEOWNERS file for suggestions.
- Provides a comprehensive description of the problem addressed and changes made.
- Explains dependencies and backwards incompatible changes .
- Contains unit and end-to-end tests and a description of how these were run.
- Includes changes to relevant documentation.

If you are submitting an issue, please double check that there doesn't already exist an issue for the work you have in mind.

Please make sure your issue

- Is created in the correct repository.
- Has a clear detailed title such that it can't be confused with other Celo issues.
- Provides a comprehensive description of the current and expected behavior including, if relevant, links to external references and specific implementation guidelines.
- Is tagged with the relevant labels.
- Is assigned if you or someone else is already working on it.

□ CIPs

Celo's Improvement Proposals \ (CIPs\) describe standards for the Celo platform, including the core protocol specifications, SDK, and contract standards. A CIP is a design document that should provide background information, a rationale for the proposal, detailed solution including technical specifications, and, if any, a list of potential risks. The proposer is responsible for soliciting community feedback and for driving consensus.

Submitting CIPs

Draft all proposals following the template below and submit to the CIPs repository via a PR \ (pull request\).

CIP template

- **Summary:** Describe your proposal in 280 characters or less.
- **Abstract:** Provide a short description of the technical issue being addressed.
- **Motivation:** Clearly explain why the proposed change should be made. It should layout the current Celo protocol shortcomings it addresses and why doing so is important.
- **Specification:** Define and explain in detail the technical requirements for new features and/or changes proposed.
- **Rationale:** Explain the reasoning behind your approach. It should cover alternative approaches considered, related work, and trade-offs made.
- **Implementation:** For all proposals going through the governance process, this section should reference the code implementing the proposed change. It's recommended to get community feedback before writing any code.
- **Risks:** Highlight any risks and concerns that may affect consensus, proof-of-stake, governance, protocol economics, the stability protocol, security, and privacy.

■ Docs

Technical writers support the Celo community by educating developers about Celo through engaging, informative, and insightful documentation.

Edit an existing page

To edit an existing page in the documentation, create a fork of the repo, commit your edits, and submit a PR.

- Go to the page in the docs
- Click Edit this page at the bottom of the page
- Edit the page directly on GitHub
- Describe the edit in the commit
- Select "Create a new branch and start a pull request"
- Describe changes in the Pull Request (PR)
- Select "therealharpaljadeja", "viral-sangani" or "GigaHierz" as a reviewer
- Changes must be approved and pass all of the site build checks before being merged.

Add/remove pages

To add a new page to the documentation, create a fork, add the new pages, and update the table of contents file to include your new pages in the appropriate location and submit a PR.

- Add or delete pages directly in GitHub
- Put new pages where you think makes the most sense, we can move them later
- Create a PR to have your changes added to the live version of the site
- Update the file called "sidebars.js" in the main folder
- This file contains the site layout that you see on the left side of the docs site
- Add or remove the appropriate files from the list

📖 Tutorials

Write about your experience as a member of the Celo community, whether you're a CELO owner, developer, or project founder. Your experience and perspective are valuable and can help others.

File naming

Creating a new post in the blog is straightforward. Create a new file in the blog directory in the documentation repository. Filenames follow the format of YYYY-MM-DD-post-name.md. For example, this post was written November 8th, 2021 so it has the filename 2021-11-08-contributing.md.

Front Matter

Posts are written in Markdown. Posts include front matter. The front matter is file metadata at the top of the file that provides more information about the post. The front matter for this post looks like this:

```
md
---
title: Contributing to the Blog
description: How to contribute to the blog
slug: blog-contributions
authors:
  - name: Josh Crites
    title: Developer Relations, cLabs
    url: https://github.com/critesjosh
    imageurl: https://github.com/critesjosh.png
tags: [contribute]
image: https://i.imgur.com/mErPwqL.png
hidetableofcontents: false
---
```

Post summary

Pages can also include a `<!--truncate-->` tag that specifies what text will be shown along with the post title on the post list page. Any text above `<!--truncate-->` will appear as the post summary.

Adding static assets

If you would like to include images or other static assets in a post, you can create a folder following the naming convention described above (YYYY-MM-DD-post-name). The contents of the folder can include the images and the post (with filename index.md).

🌐 Translations

Translators support the community expanding Celo to non-English speaking communities by translating and sharing content in different languages.

How to Contribute

- Go to <https://celo.crowdin.com/>
- Create an account
- Start translating!
- Submit questions with Crowdin Messages
- Translation request form

:::tip

For questions, comments, and discussions please use the Celo Forum or Discord.

:::

bridging.md:

title: Bridging

Bridging allows users to transfer assets between the Celo network and other blockchain networks. This section provides an overview of available bridging and swapping options.

:::warning

Be sure you understand and review the risks when bridging assets between chains.

:::

Bridging To and From Celo

Popular Bridges

- Squid Router V2
- Portal Bridge
- Jumper Exchange

Gasless Bridge

- SmolRefuel

Native Bridge

:::info

Native Bridging is only possible once Celo has become an L2. For now you can test it on the Alfajores L2 testnet.

:::

- Superbridge Celo Mainnet - not released

- Superbridge Testnet
- Superbridge Celo Mainnet - not released
- Superbridge Testnet

dexes.md:

title: DEXs on Celo

Swapping assets on Celo allows users to easily exchange tokens within the Celo ecosystem.

:::warning

Be sure you understand and review the risks when swapping assets.

:::

Decentralized Exchanges

- Uniswap
- Ubeswap
- Other DEXes

Stablecoin Swaps

- Mento

gas-fees.md:

title: Getting CELO for Gas Fees

When using a Celo-optimized wallet, gas fees can be paid with various ERC-20 tokens including ETH, USDC, USDT, cUSD and CELO. For non Celo-optimized wallets, you'll need CELO tokens for gas fees.

Using an Exchange

CELO is listed on 20+ exchanges worldwide.

- Get CELO
- Get cUSD
- Get cEUR

:::warning

Be sure to research which exchanges operate within your legal jurisdiction and support the tokens you need.

:::

Bridging from Other Chains

Bridge assets from other blockchains to CELO using:

- SmolRefeul (Gas-free)
- Other Bridges

voting.md:

title: Voting on Governance Proposals

Celo uses a formal onchain governance mechanism to manage and upgrade the protocol. More information about our governance system can be found in the Governance Section of the docs.

Celo Mondo

Celo Mondo is a decentralized application for staking and governance within the Celo ecosystem. It enables users to lock and stake their CELO tokens to earn rewards and participate in the network's governance by voting on onchain proposals.

- Launch App
- Celo Mondo GitHub
- Become a Delegate

:::note

If you would like to keep up-to-date with all the news happening in the Celo community, including validation, node operation and governance, please sign up to our Celo Signal mailing list [here](#).

:::

:::tip

You can add the Celo Signal public calendar as well which has relevant dates.

:::

wallets.md:

title: Wallets

Celo is designed to work seamlessly with a range of wallets, each offering features to meet different user needs. This section provides an overview on how you can use Celo with various types of wallets.

Celo Optimized Wallets

Celo-optimized wallets provide built-in support for Celo and make use of its unique features. These wallets allow users to fully benefit from Celo's native functionalities, such as phone number mapping, ultra-light mobile clients, and fee abstraction.

Opera MiniPay

MiniPay is a stablecoin-based non-custodial wallet natively integrated into the Opera Mini Android browser. It enables instant transactions using a phone number and provides easy access to dApps directly within the browser, all while consuming less than 2MB of data.

How to Get Started with MiniPay:

1. Download Opera Mini: Get the Opera Mini browser for free from the Google Play Store.
2. Create an Account: Open the browser, go to MiniPay, and follow the prompts to create an account, with your google mail address and your phone number. (It will only be shown in the countries that it's live in, meaning where on and off-ramp providers are integrated.)
3. Explore dApps: Use the built-in dApp discovery page to find and interact with various Celo-based applications.

MiniPay (standalone)

MiniPay is a stablecoin-based non-custodial wallet now also available as a standalone dapp. It enables instant transactions using a phone number and provides easy access to dApps directly within the browser, all while consuming less than 2MB of data.

How to Get Started with MiniPay:

1. Download Opera Mini: Get the Opera Mini browser for free from the Google Play Store.
2. Create an Account: Open the browser, go to MiniPay, and follow the prompts to create an account, with your google mail address and your phone number. (It will only be shown in the countries that it's live in, meaning where on and off-ramp providers are integrated.)
3. Explore dApps: Use the built-in dApp discovery page to find and interact with various Celo-based applications.

Valora

Valora is a mobile wallet optimized for the Celo blockchain. It enables users to exchange and securely store digital assets on a mobile phone. Valora supports both Celo and other Ethereum networks.

How to use Celo on Valora:

1. Download Valora: Install from the App Store or Google Play.
2. Set Up Account: Create a new account or recover an existing one.
3. Verify Phone Number: Complete phone verification for easy transactions.
4. Start Using Celo: Begin sending, receiving and using dApps.

Other Wallets

You can add Celo as a custom network to any EVM-compatible wallet, such as MetaMask.

MetaMask

To add Celo as a custom network in MetaMask:

1. Open MetaMask: Launch the MetaMask extension or app.
2. Add Network: Go to the network dropdown and select "Add Network."
3. Enter Network Details:
 - Network Name: Celo Mainnet
 - RPC URL: <https://forno.celo.org>
 - Chain ID: 42220
 - Currency Symbol: CELO
 - Block Explorer: <https://explorer.celo.org>
4. Save: Click "Save" to add the network.

Once saved, select Celo from the network dropdown menu to connect.

index.md:

title: Get CELO

description: Start sending, spending, and earning crypto from your mobile phone

```
import PageRef from '@components/PageRef'
import Tabs from '@theme/Tabs';
import TabItem from '@theme/TabItem';
```

Celo Holders

Start sending, spending, and earning crypto from your mobile phone.

Celo Owners can start transacting with 6 billion other smartphone users around the world from your phone and on the go. Transact in seconds – at a fraction of the cost of other crypto platforms. Anyone who holds any amount of CELO is empowered to vote on governance proposals that direct how the core technology operates today and in the future.

Discover CELO

CELO is listed on 20+ exchanges worldwide. You should always do your own research on which exchange is available in your legal jurisdiction and which has the token available that you are looking for. As always the rule is DYOR (do your own research). Below you can find links to a page where you can see the current token price as well as options to buy those.

- Get CELO
- Get cUSD
- Get cEUR

Manage Celo Assets

- Self-Custody CELO
- Understand ReleaseGold
- Exchange Celo Assets
- Asset Management

Voting as a Celo Owner

- Voting for Validators
- Voting on Governance
- Governance Cheatsheet

Asset Recovery

- Recover ETH from a Celo Address
- Recover CELO from an Ethereum Address

:::tip

For questions, comments, and discussions please use the Celo Forum or Discord.

:::

asset.md:

title: Celo Asset Management Guide

description: Access and account management for holding, exchanging, or sending Celo Dollars (cUSD) and Mento stablecoins.

Asset Management

Access and account management for holding, exchanging, or sending Celo Dollars (cUSD) and Mento stablecoins.

Prerequisites

This guide assumes:

- You have read Key Management on Celo
- You have installed the Celo Command Line Interface (Celo CLI)

Choose a Node

In order to execute the tasks listed below, you will need to point the Celo CLI to a node that is synchronized with the Mainnet.

Create an Account

There are two ways to create an account:

- (Recommended) use accounts generated by Ledger, if you possess a Ledger hardware wallet
- Use CLI to generate an account -- this approach is less secure and hence not recommended

After creating an account, record its address in environment variables:

```
export CELOACCOUNTADDRESS=<YOUR-CELO-ACCOUNT-ADDRESS>
```

Exchange CELO for Mento Stablecoins

Once you have deposited CELO to your account, you can check your balance:

```
celocli account:balance $CELOACCOUNTADDRESS
```

As an example of a common stablecoin swap, you can exchange CELO for cUSD using the following command. This exchanges CELO for stable tokens (cUSD by default) via the stability mechanism. Note that the unit of value is Celo Wei (1 CELO = 10¹⁸ Celo Wei).

```
celocli exchange:celo --value <VALUE-TO-EXCHANGE> --from  
$CELOACCOUNTADDRESS
```

Transfer Mento Stablecoins

When you have sufficient balance, you can send Mento stablecoins such as cUSD to other accounts. Note that the unit of value is cUSD Wei (1 cUSD = 10^{18} cUSD Wei).

```
celocli transfer:dollars --from $CELOACCOUNTADDRESS --to <RECIPIENT-ADDRESS> --value <VALUE-TO-TRANSFER>
```

```
# exchange.md:
```

```
---
title: Exchange Celo Assets
description: How to use the Celo exchange bot to exchange CELO and Celo
stable tokens.
---
```

Exchange Celo Assets

How to use the Celo exchange bot to exchange CELO and Celo stable tokens.

```
---
```

```
:::warning
The celo-exchange-bot is currently not being maintained. If you want to
use please check and update dependencies to solve potential security
vulnerabilities of old dependencies
:::
```

Celo Exchange Bot

CELO (previously Celo Gold) can be exchanged for Celo stable tokens (e.g. Celo Dollar or Celo Euro) using Mento, an automated market maker that powers the stability protocol. Mento is a Constant Product Market Maker (CPMM) that allows you to exchange CELO for Celo stable tokens and vice versa. Sales of Celo stable tokens to Mento in exchange for CELO burn the Celo stable tokens from supply, and sales of CELO to Mento in exchange for Celo stable tokens mint Celo stable tokens into supply. Trades with Mento incur slippage, meaning that Mento exchanges move the price out of favor of the trader. Generally, larger trade amounts incur more significant amounts of slippage. Mento also resets the price of CELO quoted in the Celo stable token every few minutes according to a price oracle.

Because of slippage and the Mento price occasionally changing according to a price oracle, those who wish to mint Celo stable tokens into supply may wish to slowly sell CELO for Celo stable tokens over time, rather than in a single exchange. Executing a smaller volume exchange every few seconds over a period of time is likely to result in less slippage when

minting Celo stable tokens. celo-exchange-bot was created to easily allow community members to exchange CELO for Celo stable tokens over a period of time to avoid incurring significant amounts of slippage.

Running the bot

celo-exchange-bot is intended to be operated by the exchanger as it requires access to the source key, which must own CELO funds to exchange and is the account that performs the exchanges. Operating the bot requires some technical knowledge of dealing with keys and operating infrastructure. Currently, the bot requires the source key to be an HSM in Azure's Key Vault service. Information on how to use an Azure Cloud HSM can be found [here](#).

See the repository's README for information on building a Docker image and configuring the bot. Example infrastructure using Azure's Container Instances is also provided in the repository [here](#). While the bot does require Azure Key Vault to be used for the source key and the provided example infrastructure is ran on Azure, the bot itself can be ran from anywhere as long as it's able to access its Azure Key Vault Cloud HSM.

release-gold.md:

```
---
title: Celo ReleaseGold
description: Introduction to ReleaseGold including examples, use cases,
and FAQ.
---
```

Understanding ReleaseGold

Introduction to ReleaseGold including examples, use cases, and FAQ.

What is ReleaseGold?

ReleaseGold is a smart contract that enables CELO to be released programmatically to a beneficiary over a period of time. In a deployed ReleaseGold smart contract, only the CELO balance that has been released according to the release schedule can be withdrawn by the contract's beneficiary. The unreleased CELO cannot be withdrawn, but can be used for specific functions in Celo's Proof of Stake protocol, namely voting and validating.

The intent of the ReleaseGold contract is to allow beneficiaries to participate in Celo's Proof of Stake protocol with CELO that has not yet been fully released to them. Beneficiaries are able to lock CELO for voting and validating with the full ReleaseGold balance, including both released and unreleased CELO.

Increasing the volume of CELO that can be used in Celo's Proof of Stake consensus promotes network security and even greater decentralization.

See below for details on specific features of the ReleaseGold contract, as well as how they are implemented. The source code includes documentation, and technical readers are encouraged to find further details there.

Example

To illustrate with an example, let's consider a ReleaseGold contract deployed with a total balance of 100 CELO. For example purposes, we'll assume this contract enables both voting and validating.

Let's also assume the beneficiary is an individual who is receiving CELO based on a vesting schedule (or a 'release schedule'). According to this release schedule, the beneficiary will receive 10% of the total CELO balance each month.

In three months time after deployment, there will be 30 released CELO in the contract, because 10 CELO (10% of 100 CELO) was released each month, for 3 months. Now, the beneficiary can transfer this 30 CELO freely.

The beneficiary does not yet have full rights to the remaining 70 unreleased CELO. However, this 70 CELO while unavailable for withdrawal, can still be used by the beneficiary for voting and validating. This unreleased balance will also continue to release at the rate of 10 CELO per month, until the total balance is empty.

Addresses Involved

Beneficiary

The beneficiary address is the recipient of the CELO in the ReleaseGold contract. As the CELO is released over time, it is incrementally made withdrawable solely to the beneficiary. The beneficiary is also able to use both unreleased and released CELO to participate in Celo's Proof of Stake consensus protocol, via locking gold and voting or validating.

Release Owner

The releaseOwner is the address involved in administering the ReleaseGold contract. The release owner may be able to perform actions including setting the liquidity provision for the contract, setting the maximum withdrawal amount, or revoking the contract, depending on the ReleaseGold configuration.

Refund Address

The refundAddress is the address where funds that have not been released will be sent if a ReleaseGold contract is revoked. Contracts that are not revocable do not have a refundAddress.

Use Cases for ReleaseGold

Two anticipated use cases for ReleaseGold contracts are for "holders" and "earners". Note that these are not specified in ReleaseGold explicitly,

rather they represent sample configurations that the ReleaseGold contract supports.

In the "holder" case, a recipient may have purchased or been awarded an amount of CELO, but is subject to a distribution schedule limiting the amount of CELO that can be liquidated at any given time. These recipients may be able to validate and vote with the full ReleaseGold balance, and also are not subject to the contract's revocation by another party (eg. an employer).

In the "earner" case, a grant recipient may have entered a legal contract wherein an exchange of services earns them an amount of CELO over a releasing, or vesting, schedule. These grants are characterized by extra restrictions because the total grant amount is still being earned. The ReleaseGold balance cannot be used for running a validator, but it can be used to vote for validators and governance proposals on the Celo network. Additionally, these contracts may be revocable and may be subject to the liquidityProvision flag, which prevents CELO distribution when markets are incapable of absorbing additional CELO without significant slippage.

Release Schedule

In ReleaseGold smart contracts, a fixed amount of CELO becomes accessible to the beneficiary over time.

The following arguments specify a ReleaseGold smart contract schedule:

<!-- make the below text code block because crowdin is messing it up -->

mdx-code-block

- releasePeriod - the frequency, in seconds, at which CELO is released
 - Some common values: monthly (2628000), every 3 months (7884000)
- amountReleasedPerPeriod - the amount of CELO to be released each releasePeriod
- numReleasePeriods - the number of releasePeriods in which CELO will be released
- releaseCliff - the time at which the release cliff expires.

The total balance for the ReleaseGold account can be determined by multiplying the numReleasePeriods by amountReleasedPerPeriod.

Similar to vesting-type schedules with cliffs used for other assets, ReleaseGold allows for a releaseCliff (expressed in seconds) before which the released CELO cannot be withdrawn by its beneficiary. A common value for this is 31536000, which is 1 year.

Released and Unreleased CELO

In deployed ReleaseGold accounts, you can conceptually think of CELO in two states -- released, and unreleased. There are other states including locked, but for the purposes of the contract, these are the two primary states to consider.

Released CELO can be withdrawn to the beneficiary where it can be used freely. Unreleased CELO comes with some restrictions. Foremost, it cannot be withdrawn by the beneficiary. If `canVote` and `canValidate` are set to false, the beneficiary cannot vote or validate, respectively.

If the contract permits voting and validating using the unreleased balance, the specific keys to perform these actions must first be authorized. For example, if the beneficiary desires to vote using their ReleaseGold contract, they must authorize a voting key to vote on the contract's behalf.

FAQ

Can I vote for validators, or run a validator using my ReleaseGold CELO balance?

- Keep in mind that in a ReleaseGold contract, there is both released CELO, and unreleased CELO. You can always vote or validate with the released balance if you are the beneficiary. However, for unreleased CELO, you can only vote or validate if `canVote` or `canValidate` properties are respectively set to true on the contract .

Can the releaseOwner access my CELO?

- No, the releaseOwner cannot make transactions with the CELO balance in a ReleaseGold contract. However, they can perform some administrative functions if the permissions are given at time of deployment. For example, a releaseOwner cannot revoke a contract unless the property `revocable` is set to true when the contract is deployed.
- It is highly recommended to review the contract at its deployed address, to learn specific details of a ReleaseGold contract.

Can I move the CELO released by the ReleaseGold contract to another address?

- Of course! Once CELO is released and the cliff has passed, the beneficiary is free to do what they want with it.

Why do I need to authorize separate keys for voting and validating? Can't I do it using the private key for my beneficiary address?

- You may use any keys for your voting and validating signers, so long as those keys are not for a registered account or for another signing purpose. This means you could use your beneficiary address as one of your signing roles, but you would need another account for an additional role.

Can I change the beneficiary?

- Yes, but changing the beneficiary requires signatures from both the releaseOwner and the current beneficiary of the ReleaseGold contract. This is implemented as a two out of two multisig contract.

What if I lose the private key associated with the beneficiary address?

- Unfortunately, if you lose the private key for the beneficiary address, then you won't be able to access your funds. Please be careful in ownership of this key, as it's loss is irreversible.

What happens if there is a bug in the ReleaseGold contract?

- The ReleaseGold contract has been reviewed by security firms, and has passed smart contract audits. That said, if any unforeseen bugs are found, it is possible to modify the contract and redeploy it. This process requires a 2/2 multisig agreement from both releaseOwner and beneficiary.

What is the distribution ratio?

- Some grants are subject to "distribution schedules," which control the release of funds outside of a traditional vesting schedule for legal reasons. This schedule is controlled by the distributionRatio and is adjustable by the releaseOwner.

self-custody.md:

title: Self-Custody CELO

description: Account access and reward details for self-custodying holder of CELO on the Celo Mainnet.

Self-Custody CELO

Account access and reward details for self-custodying holder of CELO on the Celo Mainnet

Prerequisites

This guide assumes:

- You are self-custodying (you hold the private key to your address), and that you have provided that address directly to cLabs. If you are using a custody provider (Anchorage, Coinbase, CoinList, or others), please contact them for directions.

- Your address is the beneficiary of a ReleaseGold contract, which releases CELO programmatically to a beneficiary over a period of time.

- You have been informed by cLabs that the ReleaseGold instance corresponding to your address has been deployed.

- You have your private key held on a Ledger Nano S or Ledger Nano X device, and you have a second such device available for managing a voting key. If you only have a single Ledger available, see below.

:::warning

Warning: Self-custodying keys have associated security and financial risks. Loss or theft of keys can result in irrecoverable loss of funds. This guide also requires technical knowledge. You should be comfortable with using a Command Line Interface (CLI) and understand the basics of how cryptographic network accounts work.

:::

Support

If you have any questions or need assistance with these instructions, please contact cLabs or ask in the #celo-holders channel on Celo's Discord server. Remember that Discord is a public channel: never disclose recovery phrases (also known as backup keys, or mnemonics), private keys, unsanitized log output, or personal information.

Please refer to the Ledger Troubleshooting for issues using Ledgers with the Celo CLI.

Outline

In this guide, you will:

- Install the Celo CLI (and optionally, a local node to connect to the network)
- Access the ReleaseGold account associated with your address using your existing Ledger
- Authorize a voting key, which you will hold on a new, second Ledger
- Lock some of the Gold in your ReleaseGold account
- Use that Locked CELO to vote for Validator Groups to operate Celo's Proof of Stake network (and in doing so be ready to receive epoch rewards of 6% when the community enables them in a forthcoming governance proposal)

Preparing Ledgers

You will need:

- Your Beneficiary Ledger: One Ledger Nano S or X configured with your beneficiary key (used to produce the address you supplied cLabs). Once you have completed this guide, this will become a "cold wallet" that you can keep offline most of the time.
- Your Vote Signer Ledger: One Ledger Nano S or X configured with a new, unused key. This will become a "warm wallet" you can use whenever you want to participate in validator elections or governance proposals.

As a first step, follow these instructions for both Ledgers to install the Ledger Celo app, obtain and verify the associated addresses, and (recommended) run a test transaction on the Alfajores test network.

:::info

The latest version of the Celo Ledger app is 1.1.8. If you are already using a Ledger with an earlier version installed, please upgrade.

:::

The remainder of this guide assumes you are using the first address available on each Ledger. You can add the flags described in these instructions to commands below to use different addresses.

Using a single Ledger

If you only have a single Ledger, and are comfortable losing the security advantage of keeping the beneficiary key offline when voting, you can configure a second address on the same Ledger as your voting key.

First, read these instructions carefully. Then, wherever you see instructions to connect your Vote Signer Ledger, for each command line containing `--useLedger` also add `--ledgerCustomAddresses "[1]"`. If in doubt, ask for help.

Deployment

If you haven't already, open a terminal window and install the Celo CLI:

```
bash
npm install -g @celo/celocli
```

If you have previously installed the CLI, ensure that you are using version 0.0.47 or later:

```
bash
celocli --version
```

And if not, upgrade by running the same command as above.

You will now need to point the Celo CLI to a node that is synchronized with the Mainnet network. There are two options:

- Local Celo Blockchain node: You can run a full node on your local machine which will communicate with other nodes and cryptographically verify all data it receives. Since this approach does not require you to trust the network, it is most secure.

To do this, follow the tutorial for running a full node (and make sure to pass `--usb`).

Then run:

```
bash
celocli config:set --node http://localhost:8545
```

- cLabs-operated node: As an alternative to using your own node, you can use an existing transaction

node service. Forno, operated by cLabs, is one example. While this approach does not require you to deploy a node locally, it requires you to trust cLabs and the remote Forno nodes (in the same way you would trust a centralized web service). An attacker may be able to manipulate data returned to you from the service, which the CLI may rely on to complete operations.

To use Forno, run this command:

```
bash
celocli config:set --node https://forno.celo.org
```

Locate and verify your ReleaseGold contract address

First, copy the beneficiary address into the clipboard, and set it in an environment variable:

```
bash
export CELOBENEFICIARYADDRESS=<Beneficiary>
```

Next, you will find the address of the ReleaseGold contract deployed for your beneficiary address. The ReleaseGold contract has its own address and is separate from the beneficiary address, but there are certain aspects of it that can be controlled only by the beneficiary. For more details, please refer to the Understanding ReleaseGold page.

Open the list of all ReleaseGold deployments and locate your address (use Edit>Find in your browser, then paste the beneficiary address). Copy the matching value next to ContractAddress into your clipboard.

If you cannot locate your address in these mappings, please contact cLabs.

If you have more than one beneficiary address, you'll want to step through this guide and complete the steps for each one separately.

Record the value of the ContractAddress in an environment variable:

```
bash
export CELORGADDRESS=<ContractAddress>
```

You should find your beneficiary account already has a very small CELO balance to pay for transaction fees (values are shown in wei, so For example, 1 CELO = 1000000000000000000):

```
bash
celocli account:balance $CELOBENEFICIARYADDRESS
```

Next, check the details of your ReleaseGold contract:

```
bash
celocli releasecelo:show --contract $CELORGADDRESS
```

Verify the configuration, balance, and beneficiary details. You can find an explanation of these parameters on the ReleaseGold page.

If any of these details appear to be incorrect, please contact cLabs, and do not proceed with the remainder of this guide.

If the configuration shows `canVote: true`, your contract allows you to participate in electing Validator Groups for Celo's Proof of Stake protocol, and potentially earn epoch rewards for doing so. Please continue to follow the remainder of this guide (or you can come back and continue at any time).

Otherwise, you're all set. You don't need to take any further action right now.

Authorize Vote Signer Keys

To allow you to keep your Beneficiary Ledger offline on a day-to-day basis, it's recommended to use a separate Authorized Vote Signer Account that will vote on behalf of the beneficiary.

:::info

A vote signer can either be another Ledger device or a cloud Hardware Security Module (HSM). Explore this guide to learn more about cloud HSM setup and celocli integration.

:::

This is a two step process. First, you create a "proof of possession" that shows that the holder of the beneficiary key also holds the vote signer key. Then, you will use that when the beneficiary signs a transaction authorizing the vote signer key. This proves to the Celo network that a single entity holds both keys.

:::info

Connect your Vote Signer Ledger now, unlock it, and open the Celo application.

:::

First, obtain your vote signer address:

```
bash
Using the Vote Signer Ledger
celocli account:list --useLedger
```


Your address is listed under Ledger Addresses. Create an environment variable for your vote signer address.

```
bash
export CELOVOTESIGNERADDRESS=<YOUR-VOTE-SIGNER-ADDRESS>
```

Then create the proof of possession:

```
bash
Using the Vote Signer Ledger
celocli account:proof-of-possession --signer $CELOVOTESIGNERADDRESS --
account $CELORGADDRESS --useLedger
```

The Ledger Celo app will ask you to confirm the transaction. Toggle right on the device until you see Sign Message on screen. Press both buttons at the same time to confirm.

Take note of the signature produced by the proof-of-possession command and create an environment variable for it.

```
bash
export CELOVOTESIGNERSIGNATURE=<YOUR-VOTE-SIGNER-SIGNATURE>
```

Now switch ledgers.

```
:::info
```

Connect your Beneficiary Ledger now, unlock it, and open the Celo application.

```
:::
```

Next, register the ReleaseGold contract as a "Locked CELO" account:

```
bash
Using the Beneficiary Ledger
celocli releasecelo:create-account --contract $CELORGADDRESS --useLedger
```

You'll need to press right on the Ledger several times to review details of the transactions, then when the device says "Accept and send" press both buttons together.

Check that the ReleaseGold contract address is associated with a registered Locked CELO Account:

```
bash
celocli account:show $CELORGADDRESS
```

Now, using the proof-of-possession you generated above, as the Locked CELO Account account, you will authorize the vote signing key to vote on the Locked CELO Account's behalf:

```
bash
Using the Beneficiary Ledger
celocli releasecelo:authorize --contract $CELOGADDRESS --role=vote --
signer $CELOVOTESIGNERADDRESS --signature $CELOVOTESIGNERSIGNATURE --
useLedger
```

Finally, verify that your signer was correctly authorized:

```
bash
celocli account:show $CELOGADDRESS
```

The vote address under authorizedSigners should match \$CELOVOTESIGNERADDRESS.

The ReleaseGold contract was funded with an additional 1 CELO that it sends to the first vote signer account to be authorized. This allows the vote signer account to cover transaction fees. You can confirm this:

```
bash
celocli account:balance $CELOVOTESIGNERADDRESS
```

:::warning

Warning: If you authorize a second vote signer, it will not be automatically funded by the ReleaseGold contract. You will need to transfer a fraction of 1 CELO from your beneficiary address to it in order to cover transaction fees when using it.

:::

Lock CELO

To vote for Validator Groups and on governance proposals you will need to lock CELO. This is to keep the network secure by making sure each unit of CELO can only be used to vote once.

Specify the amount of CELO you wish to lock (don't include the < > braces). All amounts are given as wei, i.e., units of 10^{-18} CELO. For example, 1 CELO = 1000000000000000000.

:::warning

Make sure to leave at least 1 CELO unlocked to pay for transaction fees.

:::

```
bash
Using the Beneficiary Ledger
celocli releasecelo:locked-gold --contract $CELOGADDRESS --action lock
--useLedger --value <CELO-GOLD-AMOUNT>
```

Check that your CELO was successfully locked.

```
bash
celocli lockedgold:show $CELOGADDRESS
```

Vote for a Validator Group

Similar to staking or delegating in other Proof of Stake cryptocurrency protocols, CELO holders can lock CELO and vote for Validator Groups on the Celo network. By doing this, not only do you contribute to the health and security of the network, but you can also earn epoch rewards.

For more details, check out the [Voting for Validators](#) page, which contains useful background on how voting Validator Elections work, as well as more guidance on how to select a Validator Group to vote for. For now, all you need to know is that:

- in Celo, CELO holders vote for Validator Groups, not Validators directly
- you only earn epoch rewards if the Validator Group you voted for gets at least 1 Validator elected

Keeping this in mind, you will need to find a Validator Group to vote for and copy its address. You can find this information on community validator explorers such as the [cLabs Validator explorer](#) and [Bi23 Labs' thecelo dashboard](#).

You can also see registered Validator Groups through the Celo CLI. This will display a list of Validator Groups, the number of votes they have received, the number of additional votes they are able to receive, and whether or not they are eligible to elect Validators:

```
bash
celocli election:list
```

Once you have found one or more Validator Groups you'd like to vote for, create an environment variable for its Group address (don't include the < > braces):

```
bash
export CELOVALIDATORGROUPADDRESS=<VALIDATOR-GROUP-ADDRESS-TO-VOTE-FOR>
```

For each vote you will need to select the amount of locked CELO you wish to vote with. You can look up your balance again if you need to:

```
bash
celocli account:balance $CELOORGADDRESS
```

All CELO amounts should be expressed in wei: that means 1 CELO = 1000000000000000000. Don't include the < > braces in the line below.

To vote, you will use your vote signer key, which is voting on behalf of your Locked CELO account.

```
:::info
```

Connect your Vote Signer Ledger now, unlock it, and open the Celo application.

```
:::
```

```
bash
Using the Vote Signer Ledger
celocli election:vote --from $CELOVOTESIGNERADDRESS --for
$CELOVALIDATORGROUPADDRESS --useLedger --value <CELO-GOLD-AMOUNT>
```

Verify that your votes were cast successfully. Since your Vote Signer account votes on behalf of the Celo Locked CELO account, you want to check the election status for that account:

```
bash
celocli election:show $CELOORGADDRESS --voter
```

Your locked CELO votes should be displayed next to pending under votes.

The next day: Activate your Vote

Your vote will apply starting at the next Validator Election, held once per day, and will continue to apply at each subsequent election until you change it.

After that election has occurred, you will need to activate your vote. This will allow you to receive epoch rewards if in that election (or at any subsequent one, until you change your vote) the Validator Group for which you voted elected at least one Validator. Rewards will get added to your votes for that Group and will compound automatically.

```
:::info
```

Epoch lengths in Mainnet are set to be the number of blocks produced in a day. As a result, votes may need to be activated up to 24 hours after they are cast.

```
:::
```

Check that your votes were cast in a previous epoch:

```
bash
celocli election:show $CELOGADDRESS --voter
```

Your vote should be displayed next to pending under votes.

```
:::info
```

Connect your Vote Signer Ledger now, unlock it, and open the Celo application.

```
:::
```

Now activate your votes:

```
bash
Using the Vote Signer Ledger
You must do this in an epoch after the one you voted in: this may take up
to 24h
celocli election:activate --from $CELOVOTESIGNERADDRESS --useLedger
```

If you run election:show again, your vote should be displayed next to active under votes.

Congratulations! You're all set.

At the end of the epoch following your vote activation, you may receive voter rewards (if at least one Validator from the Validator Group for which you voted was elected).

You can see rewards using:

```
bash
celocli rewards:show --voter $CELOGADDRESS
```

Or by searching for your ReleaseGold address on the Block Explorer and clicking the "Celo Info" tab.

Next Steps

You are now set up to participate in the Celo network!

You might want to read more about choosing a Validator Group to vote for, and how voter rewards are calculated. You can vote for up to ten different Groups from a single account.

Now you've locked CELO, you can use it to participate in voting for or against Governance proposals. You can do this without affecting any vote you have made for Validator Groups.

You can also read more about how Celo's Proof of Stake and on-chain Governance mechanisms work.

Revoking Votes

At any point you can revoke votes cast for a Validator Group. For example, a Group may be performing poorly and affecting your rewards, and you may prefer to vote for another Group.

:::info

When you revoke your votes you will stop receiving voter rewards.

:::

Specify the amount of CELO you wish to revoke (don't include the < > braces). All CELO amounts should be expressed in 18 decimal places. For example, 1 CELO = 1000000000000000000.

:::info

Connect your Vote Signer Ledger now, unlock it, and open the Celo application.

:::

Revoke votes for the Group:

bash

Using the Vote Signer Ledger

```
celocli election:revoke --from $CELOVOTESIGNERADDRESS --for  
$CELOVALIDATORGROUPADDRESS --value <CELO-GOLD-AMOUNT> --useLedger
```

You can immediately re-use this locked CELO to vote for another Group.

Unlocking and Withdrawing

At some point, the terms of your ReleaseGold contract will allow you to withdraw funds and transfer them to your beneficiary address.

There are actually several steps to this process:

1. First, revoke all outstanding votes as above (including for governance proposals)
2. Unlock the non-voting Locked CELO, starting a 72 hour unlocking period
3. After the three day unlocking period is complete, withdraw the CELO back to the ReleaseGold contract
4. Assuming vesting and distribution requirements are met, withdraw the CELO to the beneficiary address

Check the current status of outstanding votes:

bash

```
celocli election:show $CELORGADDRESS --voter
```

You can view the balance of locked CELO:

```
bash
celocli account:balance $CELORGADDRESS
```

```
:::info
```

Connect your Beneficiary Ledger now, unlock it and open the Celo application.

```
:::
```

Assuming you have non-voting Locked Celo, you can initiate the process to unlock:

```
bash
Using the Beneficiary Ledger
celocli releasecelo:locked-gold --contract $CELORGADDRESS --action unlock
--useLedger --value <CELO-GOLD-AMOUNT>
```

After the 72 hour unlocking period has passed, withdraw the CELO back to the ReleaseGold contract:

```
bash
Using the Beneficiary Ledger
celocli releasecelo:locked-gold --contract $CELORGADDRESS --action
withdraw --useLedger --value <CELO-GOLD-AMOUNT>
```

Finally, request that the ReleaseGold contract transfer an amount to your beneficiary address:

```
bash
Using the Beneficiary Ledger
celocli releasecelo:withdraw --contract $CELORGADDRESS --useLedger --
value <CELO-GOLD-AMOUNT>
```

To vote with any CELO in your beneficiary account, you'll want to register it as a Locked CELO Account, authorize a new vote signing key for it, then lock CELO.

```
# from-celo-address.md:
```

```
---
```

```
title: Recover ETH or ERC-20 tokens from a Celo address
description: How to recover ETH or ERC-20 tokens if you accidentally
transferred them to a CELO (previously Celo Gold) address.
```

Recover ETH or ERC-20 tokens from a Celo address

How to recover ETH or ERC-20 tokens if you accidentally transferred them to a CELO (previously Celo Gold) address.

Prerequisites

This guide assumes that you have access to the following:

- The 24-word recovery phrase of your Celo address
- An Ethereum wallet manager such as MyEtherWallet or MyCrypto. It is recommended to use the Desktop or offline version over a webapp version.

:::danger

There are risks associated with using a recovery phrase or a private key. Please do not share them with anyone else.

:::

Steps

Please follow the instructions below closely because missteps can lead to errors or permanent loss of your tokens. To understand these steps, please read [What is Ethereum and Celo Overview](#).

MyEtherWallet

1. Download and run MEW Offline (scroll down until you see the download instructions)
2. Open MEW and click on Access My Wallet > Software > Mnemonic Phrase > Continue
3. Change to a 24 Value mnemonic and enter in the words from your Recovery Phrase. Then click Continue
4. Change the HD Derivation Path to Add Custom path
5. Call your Alias "Celo" for easy reference. Enter in your Path as m/44'/52752'/0'/0. Click Add Custom Path.
6. Go back to your Derivation Path dropdown and select Celo (m/44'/52752'/0'/0). You should see the addresses change. Select the address that matches your Valora Account Number (it's usually the first one).
7. Accept the Terms
8. Click Access My Wallet
9. Now you can send your token back to your original address:
 1. Click on Send Transaction
 2. Fill in the Type, Amount, and To Address.
 3. Confirm that the information is correct and you have enough funds to cover any fees.
 4. Click Send Transaction.

MyCrypto

1. Download then open the MyCrypto desktop app and click on View & Send
2. Access using Mnemonic Phrase
3. Enter in your Valora Recovery Phrase into the Mnemonic Phrase box (tip: click the eye icon to easily see your phrase) then click Choose address
4. Change the Address from "Default (ETH)" to "Custom". Enter m/44'/52752'/0'/0 into the box that pops up next to it.
5. Select the address that matches your Valora Account number (it's usually the first one).
6. Click Unlock
7. Scroll down and look for a button that says Scan For Tokens. Click it.
8. You should be able to see your token.
9. Now you can send your token back to your original address:
 1. Enter your address in the To Address box
 2. Change the currency.
 3. Fill in the Amount or click the "double up-arrow" icon to send your entire balance.
 4. Confirm that the information is correct and you have enough funds to cover any fees.
 5. Click Send Transaction

from-eth-address.md:

title: Recover CELO from an Ethereum Address

description: How to recover CELO (previously Celo Gold) if you accidentally transferred them to an account generated using an Ethereum wallet.

Recover CELO from an Ethereum Address

How to recover CELO (previously Celo Gold) if you accidentally transferred them to an account generated using an Ethereum wallet.

:::note

You only need to go through this process when going from mnemonic (secret phrase) to account because of the different account derivation paths between Celo and Ethereum. This is only relevant if you're using a wallet that can't connect to both the Ethereum and Celo networks or you can't export the private key. Private key--account pairs are the same for both Celo and Ethereum, it's just mnemonic (secret phrase) to private keys that are different.

With Metamask, recovery is easy because you just switch Metamask to the network where the funds are--the accounts and private keys for both networks are the same. The problem occurs when the wallet only accepts the secret phrase and derives private keys and accounts differently based on the network it is designed for.

:::

Prerequisites

This guide assumes you have access to the recipient's mnemonic recovery phrase (note, in Valora and Celo Wallet it's called your 'Recovery Phrase').

:::danger

There are risks associated with using a recovery phrase or a private key. Please keep these secret.

:::

Recovering with CeloWallet.app

The Celo Wallet for Web and Desktop can be used to import mnemonic phrases with custom derivation paths (like Ethereum's) in order to recover your funds.

Steps to access funds

1. Visit celowallet.app in a modern browser (Chrome is recommended).
2. For small accounts, you can import in the web version directly. For larger accounts, downloading the desktop version is strongly recommended.
3. Click 'Use Existing Account', then 'Use Recovery Phrase', then choose the 'Advanced' tab.
4. Specify your derivation path and click import.
5. Set a strong password and click Continue. You should reach the home screen and see your account funds.

Steps to migrate funds

Once you've been able to access your funds, it's recommended that you move them to an account derived from the Celo derivation path.

1. Create a new account: you can use the celowallet.app again, Valora, the Celo CLI, or any other wallets compatible with Celo.
2. Copy the address of your new account
3. In the first Celo Wallet window, click the Send button in the top-left.
4. Send your funds to your new account.
5. If you intend to keep this new account permanently, be sure to save its password and recovery phrase in a safe place!

Recovering with the Celo CLI

Prerequisites

This guide assumes that you have access to the following:

- A computer with a Command Line Interface. You can access it following these instructions:
 - Windows
 - Mac
 - Linux
- Celo Command Line Interface installed on your computer
- The 24-word recovery phrase of your Ethereum address

Steps

Please follow the instructions below closely, because missteps can lead to errors or permanent loss of your tokens. To understand these steps, please read [What is Ethereum](#) and [Celo Overview](#).

Prepare your recovery phases

Write your recovery phrase to a file using the following commands:

1. `nano recovery.txt`
2. Paste `<word1> <word2> ... <word24>`
3. Replace the `<word>`s in brackets with the words from your recovery phrase (usually 24 words, but can be 12, 15, 18, 21 or 24 words, as specified in the BIP 39 standard)
4. Press `ctrl-o` to save
5. Press `ctrl-x` to exit

Recover your Ethereum address on Celo

Recover your Ethereum address on the Celo network:

```
celocli account:new --mnemonicPath recovery.txt --derivationPath "eth" --  
node https://forno.celo.org
```

This command will return you with:

- `accountAddress`: the same address as your Ethereum address
- `privateKey`: the private key associated with your address -- please record this private key on paper and not share with anyone else
- `publicKey`: the public key associated with your address

Note

Using the `eth` derivation path as above will work for the default Ethereum path used by nearly all Ethereum wallets (`"m/44'/60'/0'/0/0"`). If your address was generated using a different derivation path you can specify that using a combination of the flags `addressIndex`, `changeIndex` and `derivationPath`.

For example, for the address corresponding to the path `m/44'/78'/1'/4/23` use:

```
celocli account:new --mnemonicPath recovery.txt --derivationPath
"m/44'/78'/1'" --changeIndex 4 --addressIndex 23 --node
https://forno.celo.org
```

Check your CELO balance

Check your Celo account balance using this command:

```
celocli account:balance <accountAddress> --node https://forno.celo.org
```

Replace `<accountAddress>` with the `accountAddress` you got from the previous step.

Transfer CELO

Now, you can transfer your CELO to an address of choice:

```
celocli transfer:celo --from <accountAddress> --to <addressOfChoice> --
value <valueInCeloWei> --privateKey <privateKey> --node
https://forno.celo.org
```

- Replace `<accountAddress>` with the `accountAddress` you got from the previous step.
- Replace `<addressOfChoice>` with the address that you want to send CELO to.
- Replace `<valueInCeloWei>` with the amount you want to send, but this number needs to be slightly lower than your balance, as there's a transaction fee.

:::info

Note that the value has a unit of CELO Wei ($1 \text{ CELO} = 10^{18} \text{ CELO Wei}$), so if you want to send 1 CELO, the `<valueInCeloWei>` should be 1000000000000000000.

:::

governance-parameters.md:

title: Celo Governance Cheat Sheet

description: List of governable parameters and governance restrictions on Celo.

Governance Cheat Sheet

List of governable parameters and governance restrictions on Celo.

Governable Parameters

- The stability protocol, including the exchange
- What the protocol does with data feeds from Oracles
- Adding or removing Mento stablecoins
- Whitelisting Mento stablecoins (or other ERC20s) for use in paying gas fees
- The identity protocol, including how phone number attestations works
- Linking of signers and off-chain metadata (e.g claims) to accounts
- Most of Proof of Stake protocol, including elections, locked gold, slashing parameters
- On-chain governance itself
- MinimumClientVersion
- BlockGasLimit
- IntrinsicGasForAlternativeFeeCurrency

Things That Can't Be Modified By Governance

- The protocol by which nodes communicate
- The format of block headers, block bodies, the fields in transactions, etc
- How nodes sync
- How nodes store their data locally
- Most parameters that affect the blockchain

governance.md:

title: Celo Voting on Governance Proposals

description: How to use the Celo CLI to participate in Governance and create a Governance proposal.

Voting on Governance Proposals

How to use the Celo CLI to participate in Governance and create a Governance proposal.

Governance

Celo uses a formal on-chain governance mechanism to manage and upgrade the protocol. More information about the Governance system can be found in the Governance section of the protocol documentation.

:::info

If you would like to keep up-to-date with all the news happening in the Celo community, including validation, node operation and governance, please sign up to our Celo Signal mailing list [here](#).

You can add the Celo Signal public calendar as well which has relevant dates.

:::

:::info

In the following commands <VARIABLE> is used as a placeholder for something you should specify on the command line.

:::

Viewing Proposals

A list of active proposals can be viewed with the following command:

```
bash
celocli governance:list
```

Included will be three lists of proposals by status:

- Queued proposals have been submitted, but are not yet being considered. Voters can upvote proposals in this list, and proposals with the most upvotes from this list will be moved from the queue to be considered.
- Dequeued proposals are actively being considered and will pass through the Approval, Referendum, and Execution stages, as discussed in the protocol documentation.
- Expired proposals are no longer being considered.

Understanding Proposal Details

You can view information about a specific proposal with:

```
bash
celocli governance:show --proposalID=<PROPOSALID>
```

For example, the proposal 14 on Mainnet was as follows:

Running Checks:

```
✓ 14 is an existing proposal
proposal:
  0:
    contract: Governance
    function: setBaselineQuorumFactor
    args:
```

```

    0: 50000000000000000000000000000000
    params:
      baselineQuorumFactor: 50000000000000000000000000000000 (5.000e+23)
      value: 0
  metadata:
    proposer: 0xF3EB910DA09B8AF348E0E5B6636da442cFa79239
    deposit: 100000000000000000000000000000000 (1.000e+20)
    timestamp: 1609961608 (1.610e+9)
    transactionCount: 1
    descriptionURL: https://github.com/celestia-org/celestia-
proposals/blob/master/CGPs/0016.md
    stage: Referendum
    upvotes: 0
    votes:
      Yes: 30992399904903465125627698 (3.099e+25)
      No: 0
      Abstain: 0
    passing: true
    requirements:
      constitutionThreshold: 0.7
      support: 0.99883105743491071638
      required: 29107673282861669327494319.531832308424 (2.910e+25)
      total: 30992399904903465125627698 (3.099e+25)
    isApproved: true
    isProposalPassing: true
    timeUntilStages:
      referendum: past
      execution: 57 minutes, 59 seconds
      expiration: 3 days, 57 minutes, 59 seconds

```

To see how many votes a proposal needs to pass (depending on what type of commands are being executed), you can refer to the requirements section of the response.

In the proposal above, there is a constitutionThreshold target of "0.7" or 70% of votes must be in support, "0.998" or 99.8% of votes have currently voted "yes", the number of votes required to pass are 29.1M CELO, with 30.9M CELO currently voted in total.

Voting on Proposals

When a proposal is Queued, you can upvote the proposal to indicate you'd like it to be considered.

```
:::info
```

If you are using a Ledger wallet, make sure to include --useLedger and --ledgerAddresses in the following commands.

```
:::
```

```
bash
```

```
celocli governance:upvote --proposalID=<PROPOSALID> --  
from=<YOURVOTERADDRESS>
```

At a defined frequency, which can be checked with the `celocli network:parameters` command, proposals can be dequeued, with the highest upvoted proposals being dequeued first.

After a proposal is dequeued, it will first enter the Approval phase. In this phase, the Governance Approver may choose to approve the proposal, which will allow it to proceed to the Referendum phase after the configured length of time.

Once a proposal has reached the Referendum phase, it is open to community for voting.

```
bash  
celocli governance:vote --proposalID=<PROPOSALID> --  
value=<Abstain|Yes|No> --from=<YOURVOTERADDRESS>
```

Executing a Proposal

If a Governance Proposal receives enough votes and passes in the Referendum phase, it can be executed by anyone.

```
bash  
celocli governance:execute --proposalID=<PROPOSALID> --  
from=<YOURVOTERADDRESS>
```

Vote Delegation

Contract Release 10 introduced vote delegation, which allows the governance participant to delegate their voting power.

```
:::note  
Validators and Validator groups cannot delegate.  
:::
```

Delegating Votes

You can delegate votes using the following command:

```
bash  
celocli lockedgold:delegate --from <DELEGATORADDRESS> --to  
<DELEGATEEADDRESS> --percent <PERCENTAGEBETWEEN1AND100>
```

```
:::note  
Currently, participants can only delegate to 10 delegates.  
:::
```


You can view the max number of delegates one can have using the following command:

```
bash
celocli lockedgold:max-delegates-count
```

Revoking Delegated Votes

You can use the following command to revoke delegated votes:

```
bash
celocli lockedgold:revoke-delegate --from <DELEGATORADDRESS> --to
<DELEGATEEADDRESS> --percent <PERCENTAGETOBEREVOKED>
```

For example, If you have delegated 15% to the delegatee and pass 5% as percent then 5% will be subtracted from the 15% resulting in 10% delegation.

Total percent of Locked Celo delegated by an account

You can use the following command to get the total percent of locked celo delegated by an account:

```
bash
celocli lockedgold:delegate-info --account <ACCOUNTADDRESS>
```

List of Delegates of a Delegator

You can use the following command to get the list of delegates of an account:

```
bash
celocli lockedgold:delegate-info --account <ACCOUNTADDRESS>
```

Total Delegated Votes to an address

You can use the following command to get the total delegated votes to an address:

```
bash
celocli lockedgold:delegate-info --account <ACCOUNTADDRESS>
```

validator.md:

title: Voting for Validator Groups

description: Overview of Validator elections for validator groups including technical details, policies, and explorers.

Voting for Validator Groups

Resources for Validator Groups elections including technical details, policies, and Validator explorers.

What are Validators?

Validators play a critical role in the Celo protocol, determining which transactions get applied and producing new blocks. Selecting organizations that operate well-run infrastructure to perform this role effectively is essential for Celo's long-term success.

The Celo community makes these decisions by locking CELO and voting for Validator Groups, intermediaries that sit between voters and Validators. Every Validator Group has an ordered list of up to 5 candidate Validators. Some organizations may operate a group with their own Validators in it; some may operate a group to which they have added Validators run by others.

:::info

If you would like to keep up-to-date with all the news happening in the Celo community, including validation, node operation and governance, please sign up to our Celo Signal mailing list [here](#).

You can add the Celo Signal public calendar as well which has relevant dates.

:::

Validator Elections

Validator elections are held every epoch (approximately once per day). The protocol elects a maximum of 110 Validators. At each epoch, every elected Validator must be re-elected to continue. Validators are selected in proportion to votes received for each Validator Group.

If you hold CELO, or are a beneficiary of a ReleaseGold contract that allows voting, you can vote for Validator Groups. A single account can split their LockedGold balance to have outstanding votes for up to 10 groups.

CELO that you lock and use to vote for a group that elects one or more Validators receives epoch rewards every epoch (approximately every day) once the community passes a governance proposal enabling rewards. The initial level of rewards is anticipated to be around 6% per annum equivalent (but is subject to change).

Unlike a number of Proof of Stake protocols, CELO used for voting is never at risk. The actions of the Validator Groups or Validators you vote

for can cause you to receive lower or higher rewards, but the CELO you locked will always be available to be unlocked in the future. Slashing in the Celo protocol applies only to Validators and Validator Groups.

Choosing a Validator Group

As a CELO holder, you have the opportunity to impact the Celo network by voting for Validator Groups. As Validators play an integral role in securing Celo, it is crucial that voters choose groups that contribute to both the technical health of the network, as well as the community. Some factors to consider when deciding which Validator Group to vote for include:

Technical

- Proven identity: Validators and groups can supply verifiable DNS claims. You can use these to securely identify that the same entity has access both to the account of a Validator or group and the supplied DNS records.
- Can receive votes: Validator Groups can receive votes up to a certain voting cap. You cannot vote for groups with a balance that would put it beyond its cap.
- Will get elected: CELO holders only receive voter rewards during an epoch if their CELO is used to vote for a Validator Group that elects at least one Validator during that epoch. Put another way, your vote does not contribute to securing the network or earning you rewards if your group does not receive enough other votes to elect at least one Validator.
- Secure: The operational security of Validators is essential for everyone's use of the Celo network. All Validators that participated in the Stake Off were eligible for a security audit. You can see scores under the "Master Validator Challenge" column in the Stake Off leaderboard. Scores of 80% or greater were awarded the "Master Validator" badge, indicating a serious proven commitment to operational security.
- Reliable: Celo's consensus protocol relies on two-thirds of elected Validators being available in order to produce blocks and process transactions. Voter rewards are directly tied to the uptime score of all elected Validators in the group for which the vote was made. Any period of consecutive downtime greater than a minute reduces a Validator's uptime score.
- No recent slashing: When Validators and groups register, their Locked Gold becomes "staked", in that it is subject to penalties for conduct that could seriously adversely affect the health of the network. Voters' Locked Gold is never slashed, but voter rewards are affected by a group's slashing penalty, which is halved when a group or one of its Validators is slashed. Look for groups with a last slashing time long in the past, ideally 0 (never), and a slashing penalty value of 1.0.

- **Runs an Attestation Service:** The Attestation Service is an important service that Validators can run that allows users to verify that they have access to a phone number and map it to an address. Supporting Validators that run this service makes it easier for new users to begin using Celo.

- **Runs a Validator on Baklava:** A group that runs a Validator on the Baklava helps maintain the testnet and verify that upgrades to the Celo Blockchain software can be deployed smoothly.

Community

- **Promotes the Celo mission:** Celo's mission is to build a monetary system that creates the conditions of prosperity for all. Consider Validator Groups that further this mission through their own activities or initiatives around financial inclusion, education and sustainability.

- **Broadens Diversity:** The Celo community aims to be inclusive to the largest number of contributors, with the most varied and diverse backgrounds possible. Support that diversity by considering what new perspectives and strengths the teams you support offer. As well as the backgrounds and experiences of the team, consider that the network security and availability is improved by Validators operating at different network locations, on different platforms, and with different toolchains.

- **Contributes to Celo:** Support Validator Groups that strengthen the Celo developer community, for example through building or operating services for the Celo ecosystem, participating actively in on-chain governance, and answering questions and supporting others, on Discord or the Forum.

The Celo Foundation Voting Policy

As described above, there are many criteria to consider when deciding which group to vote for. While it is highly recommended that all CELO holders do their independent research when deciding which group to vote for, another option is to vote for Validator Groups that have received votes from the Celo Foundation.

The Celo Foundation has a Validator Group voting policy that it follows when voting with the CELO that it holds. This policy has been developed by the Foundation board and technical advisors with the express goal of promoting the long-term security and decentralization of the network. Validator Groups have an opportunity to apply for Foundation votes every 3 months, and a new cohort is selected based on past performance and contributions.

You can find the full set of Validator Groups currently receiving votes, and their addresses linked [here](#).

Validator Explorers

The Celo ecosystem includes a number of great services for browsing registered Validator Groups and Validators.

:::warning

Warning: Exercise caution in relying on Validator-supplied names to determine their real-world identity. Malicious participants may attempt to impersonate other Validators in order to attract votes.

Validators and groups can also supply verifiable DNS claims, and the Celo Validator Explorer displays these. You can use these to securely identify that the same entity has access both to the account of a Validator or group and the supplied DNS records.

:::

Celo Validator Explorer (cLabs)

The Celo Validator Explorer has tabs to show either Mainnet or the Baklava Testnet.

The list shows Validator Groups and, when you expand each group, the Validators that are affiliated to that group.

A white check mark next to the name of a Validator Group shows that there is one or more DNS metadata claims verified for that group (see below).

The Votes Available column shows:

- On the left: Votes made for the group, as a percentage of the total Locked Gold
- On the right: The voting cap of that group, as a percentage of the total Locked Gold
- In the middle: votes made for the group as a proportion of the voting cap

TheCelo (Bi23 Labs)

TheCelo contains a range of valuable information on the Celo project and active Celo networks. The "Groups" tab shows a detailed view of Validator Groups. Click on a group to drilldown to see group metadata and affiliated Validators.

Celovote Scores (WOTrust | celovote.com)

Celovote shows ranking of validator groups based on their estimated annual rate of return (ARR). Estimate is calculated based on past performance.

Vido (Atalma)

Vido is a block visualization and monitoring suite for Mainnet and Baklava testnet, showing missed blocks and downtime for the validator

group set and subscribable metrics to get alerted if your validator is no longer signing.

checklist.md:

title: Celo Integration Checklist

description: Checklist for applications building and integrating on Celo.

Checklist

Checklist for applications building and integrating on Celo.

General

Addresses

Addresses are identical to Ethereum addresses. When displaying and asking for user-inputted addresses, consider using and validating address checksums following the EIP55 standard to detect typos.

For core smart contracts, developers are highly encouraged to use the Registry to reference the contracts in case they will have to be repointed (via Governance)

QR Codes

Celo has WIP QR code standard CIP16 that aims to standardize how applications can ask wallets for transactions to avoid the user having to manually copy/paste addresses and other transaction metadata.

Custodian/Exchange

Please read more under Custody, but here is a shortened version:

Detect Transfers

Stable-value currencies, currently cUSD and cEUR, are contracts, StableToken and StableTokenEUR respectively, that can be accessed via the ERC20 interface. The native asset CELO can be accessed via the GoldToken ERC20 interface, or natively, similar to ETH on Ethereum.

Addresses for those contracts can be found by querying the registry or in the Listing Guide.

Proof of Stake

Users may want to participate in Celo's Proof of Stake system to help secure the network and earn rewards.

Authorized Signers

Celo's core smart contracts use Celo's Accounts abstraction to allow balance-moving keys to be held in cold storage, while other keys can be authorized to vote and be held in warm storage or online.

Release Gold

There is an audited ReleaseGold smart contract which allows for the release of CELO over a set schedule through which CELO might be distributed to a user.

Wallets

These suggestions apply to any application that custodies a key and allows users to interact and transfer value on the Celo platform.

Key Derivation

Celo wallets should follow the BIP44 for deriving private keys from BIP39 mnemonics. Celo's key derivation path is at m/44'/52752'/0'/0. The first key typically is the account key that wallets should register themselves with and accept balance transfers on. The second key can be derived to be an account's dataEncryptionKey to allow other users on Celo to encrypt information to.

Identity Protocol

Celo has a lightweight identity protocol that allows users to address each other via their phone number instead of addresses that Celo wallets should implement. Since user privacy is important, Celo wallets should leverage the built-in Phone Number Privacy protocol to protect against large-scale harvesting of user phone numbers.

Wallet Address

When transferring assets to an account, wallets should check the receiving account's walletAddress at which they want to receive funds at. Use cases might be smart contract accounts that want different recovery characteristics, but receive funds at a different address. Also, walletAddresses of 0x0 should indicate that the account requires a different mechanism to acquire the walletAddress.

Transaction metadata

cUSD (aka StableToken) adds an additional method to the ERC20 interface called transferWithComment which allows senders to specify an additional comment that Celo wallets should support. Additionally, comments should be encrypted to the dataEncryptionKey when applicable.

Validator Group Explorers

Validator Group Explorers are critical to Celo's Proof of Stake system. Explorers will consider using the following standards to provide a minimum experience across all explorers.

Names

All Celo accounts on Accounts.sol can claim any name they want. While explorers should display it, they should also be cognizant of the fraud potential.

Identities

Celo accounts can make claims to existing identities, some of which are verifiable (Domain Names or Keybase profiles). Explorers should consider displaying those identities to reduce the potential for impersonation.

Performance indicators

Validator Groups and their validators can perform their duties differently and explorers should reflect that to allow voters to ensure an optimal validator set. While uptime in the form of block signatures by the validators ultimately affect rewards, explorers should also consider displaying other metrics that impact the success of the Celo ecosystem, such as validators' performance in the identity protocol.

cloud-hsm.md:

title: Using a Cloud HSM with Celo

description: How to create a cloud HSM in Azure and connect it to celocli.

Using a Cloud HSM

How to create a cloud HSM in Azure and connect it to celocli.

Introduction to HSM

A cloud Hardware Security Module (HSM) provides a good balance between security and accessibility. A cloud HSM can manage a Celo private key and can be used seamlessly with celocli and contractkit. Similar to a ledger device, a key in an HSM avoids the key from ever being sent over the network or stored on disk since the key can never leave the hardware boundary and all signing is performed within the HSM. To authenticate to the HSM, it's recommended to create a service principal account that has been granted access to sign with the managed keys. A cloud HSM can be a great option for managing vote signer keys, since you may want these keys to be portable but also maintain good security practices.

Create an Azure subscription

If you don't have an Azure subscription already, you can create a free trial here that starts with \$200 credit. You can view the pricing for Elliptic Curve Cryptography (ECC) HSM keys here.

Deploy your Azure Key Vault

The Key Vault can store keys, secrets, and certificates. Permission can be specified to perform certain actions across the entire Key Vault (ex. key signing).

- Search the marketplace for "Key Vault"
- Click Create and fill out the deployment information
- Ensure you select the Premium pricing tier for HSM support
- Enable soft-delete and purge protection to ensure your keys aren't accidentally deleted

Create your key

Next, we'll create the ECDSA key.

- Navigate to your newly created Key Vault and click on the Keys section.
- Click on "Generate/Import"
- Select "EC-HSM"
- Select "SECP256K1"

You'll see your newly generated key listed in the Keys section.

bash

On your local machine

```
export AZUREVAULTNAME=<VAULT-NAME>
```

```
export AZUREKEYNAME=<KEY-NAME>
```

Create a Service Principal

A Service Principal (SP) is preferred over your personal account so that permission can be heavily restricted. In general, Service Principal accounts should be used for any automation or services that need to access Azure resources.

Use the Cloud Shell to create the client credentials.

Create a service principal and configure its access to Azure resources:

bash

In the Cloud Shell

```
az ad sp create-for-rbac -n <your-application-name> --skip-assignment
```

The account will be created and will output the account's credentials.

bash

```
{
```

```

    "appId": "generated-app-ID",
    "displayName": "dummy-app-name",
    "name": "http://dummy-app-name",
    "password": "random-password",
    "tenant": "tenant-ID"
  }

```

Set these as environment variables so that they can be used by celocli or contractkit.

```

bash
On your local machine
export AZURECLIENTID=<GENERATED-APP-ID>
export AZURECLIENTSECRET=<PASSWORD>
export AZURETENANTID=<TENANT-ID>

```

Grant your Service Principal access to the key

In the Cloud Shell or Access Policies pane of the Key Vault, set the [GET, LIST, SIGN] permission for the new account.

```

bash
In the Cloud Shell
az keyvault set-policy --name <your-key-vault-name> --spn $AZURECLIENTID
--key-permissions get list sign

```

Connecting CeloCLI to KeyVault

Now that your environment variables are set, we just need to let celocli know that we want to use this Key Vault signer. We do this by passing in the flag `--useAKV` and `--azureVaultName`. Similar to `--useLedger`, all CLI commands will use the HSM signer when `--useAKV` is specified.

```

bash
On your local machine
celocli account:list --useAKV --azureVaultName $AZUREVAULTNAME

```

Your Key Vault address will show up under "Local Addresses". If you'd like to use this key as your vote signer key, you can follow this guide and replace `--useLedger` with `--useAKV --azureVaultName $AZUREVAULTNAME`.

Connecting ContractKit to KeyVault

To leverage your HSM keys in contractkit, first create an `AzureHSMWallet` object and use it to create a `ContractKit` object with `newKitFromWeb3`. Note that `AzureHSMWallet` expects `AZURECLIENTID`, `AZURECLIENTSECRET`, and `AZURETENANTID` environment variables to be specified.

```

js
import { ContractKit, newKitFromWeb3 } from "@celo/contractkit";

```

```
import { AzureHSMWallet } from "@celo/wallet-hsm-azure";

const azureVaultName = "AZURE-VAULT-NAME";
const akvWallet = await new AzureHSMWallet(azureVaultName);
await akvWallet.init();
console.log(Found addresses: ${await akvWallet.getAccounts()});
const contractKit = newKitFromWeb3(this.web3, akvWallet);
```

Summary

You can now leverage a cloud HSM key to perform signing as a user or application. This improves both security and availability of your Celo keys. We also recommend enabling two-factor authentication across your Azure subscription and to leverage Managed Service Identities where possible.

custody.md:

```
---
title: Celo Custody Integrations
description: Details for custodians, exchanges, and other services that
intend to custody Celo assets such as Celo Dollar and CELO on behalf of a
user.
---
```

Custody

Details for custodians, exchanges, and other services that intend to custody Celo assets such as Celo Dollar and CELO on behalf of a user.

Custody Overview

Generally speaking, custodizing CELO, the native token on the Celo network, requires understanding the various states that CELO can exist in at any time. This is to provide useful services beyond custody such as allowing users to lock up their CELO and vote with it. Many of these "states" are implemented as smart contracts, and involve sending CELO from a user owned account to a contract address. Thus, in order to be able to show a user's true balance, services need to be able to observe every balance changing operation and reconcile CELO balances from all the various contracts and states CELO can be in.

Balance Model

As a fork of Ethereum, Celo retains the account model to keep track of users' balances. Celo Dollar and CELO implement the ERC20 interface. As mentioned previously, it is common for smart contracts to hold balances on behalf of other addresses. One example is the LockedGold smart contract that holds the "locked portion of a user's CELO balance". Another one is the ReleaseGold smart contract that holds CELO that is

being released to a beneficiary address over time according to some schedule.

:::warning

Celo assets exist on an independent blockchain, and although they implement the ERC20 interface, they cannot be accessed through wallets that connect to the Ethereum network. Wallets and other integrations must connect to the Celo network to transfer tokens on Celo.

:::

Applications that display balances may need to be written to be aware of this possibility.

Transfers

CELO and Celo Dollars implement the ERC20 interface, as will any future core stable Celo currencies. CELO, as the native currency of the network, can also be transferred by specifying the value field of a transaction, in the same way that ETH can be transferred in Ethereum. Therefore, for CELO, application developers should be aware that transactions can be specified in both ways.

CELO State Machine

CELO as described previously can also exist in various states that represent a specific user behavior. For example, if a user wants to lock CELO to either participate in consensus directly or vote, that CELO will be sent to the LockedGold smart contract. To understand the high level flow, please read this description of the various states CELO can exist in.

Smart Contracts

The following smart contracts are helpful to understand in order to map the conceptual states to actual accounts and function calls.

Accounts

Accounts.sol allows the mapping of an address to an account in storage, after which all further functionality (locking, voting, etc.) can be accessed.

The createAccount function indexes the address as an account in storage, and is required to differentiate an arbitrary key-pair from a user-owned account in the Celo network.

The Accounts contract also allows for the authorization of various signer keys, such as a vote signer key. This allows for the user who owns the primary account key to authorize a separate key that can only vote on behalf of the account. This allows for the ability to custody keys in a manner corresponding to their exposure or "warmth". Eg. the primary account private key can be kept in cold storage after authorizing the

signer keys, which can be in warmer environments, and potentially more exposed to the network. See the key management guide for more details.

LockedGold

LockedGold.sol, which references Celo Gold, the deprecated name for the native token, is used as part of Celo's proof-of-stake mechanism. Users can lock CELO by sending it to the LockedGold contract after creating an account via the Accounts contract as described above. This allows users to vote in validator elections, receive epoch rewards, and participate in on-chain governance.

There are two ways in which users can vote:

- Directly, by sending voting transactions with the same key used to lock up CELO
- Via an authorized vote signer, which can submit voting transactions on behalf of the account with locked CELO

LockedGold has a mapping of addresses to balances which is a type that contains both the nonvoting amount of CELO as well as pendingWithdrawals, which contain values corresponding to timestamps at which they can be withdrawn. The reason for the latter is because all locked CELO has an unlocking period that is set at time of contract initialization, which is 3 days in the Celo network's deployed LockedGold contract. Hence, if users unlock CELO in tranches, multiple pending withdrawals could exist at once. Once the timestamp has eclipsed, CELO can be withdrawn back to the user's address.

Election

Once CELO has been locked via LockedGold, it can then be used to vote for validator groups. Election.sol is the contract that manages this functionality.

The votes in this contract are tracked by a Votes type which has pending, active, and total votes. Pending votes are those that have been cast for a validator group, and active votes are those that have been activated after an epoch, meaning that these votes generate voter rewards.

Votes are cast for a validator group using the vote function. This increments the pending and total votes in the Election contract, and decrements the equivalent amount of CELO from the nonvoting balance in the LockedGold contract, for the associated account.

The activate function can then be called to shift pending votes into active votes in a following epoch. Votes in either state can then be revoked, which decrements votes from the Election contract and returns them to the LockedGold balance for the associated account. Users can revoke votes at any time and this takes effect instantly.

ReleaseGold

A common problem in other proof-of-stake protocols is the tension between wanting early token holders' balances to release over time to ensure long-term alignment, while also wanting them to be able to participate in consensus to increase the security of the network. To bridge both goals, many early token balances in the Celo network are released via the ReleaseGold contract. Beneficiaries of these contracts can then participate in the proof-of-stake system by staking and voting with CELO that has not yet been "released" for transfers. Please find more high level information about the ReleaseGold contract [here](#).

From a technical perspective, ReleaseGold can be thought of as a "puppet" account controlled by the "puppeteer", or the beneficiary private key corresponding to the beneficiary address in the contract. This beneficiary key can then authorize validator signer and vote signer keys that can then call respective functions associated with validating or voting. Most of the required function calls described above can be made by the signer keys directly to the LockedGold or Election contracts associated with the ReleaseGold account. However, some functions in the ReleaseGold contract are proxied to the underlying LockedGold or Election contracts, and have a separate function signature that can be called by the beneficiary address. Notably:

- createAccount
- authorizeVoteSigner and similar functions for other signer keys
- lockGold and unlockGold

Notice that all these functions have corresponding functions that are called on the underlying contract. The ReleaseGold contract can then just be thought of as brokering the transaction to the correct place, when necessary.

Other Balance Changing Operations

In addition to transfers (both native and ERC-20) and locking / voting flows affecting user balances, there are also several additional Celo network features that may cause user balances to change:

- Gas fee payments: the fee paid by transaction senders to use the network
- Epoch rewards distribution: reward payments to voters, validators, and validator groups

Some of these may occur as events rather than transactions on the network, and therefore when updating balances, special attention should be paid to them.

Useful Tools

Since monitoring balance changing operations is important to be able to display user balances properly, it can be helpful to use a tracing or reconciling system. Celo Rosetta is an RPC server that exposes an API to query the Celo blockchain, obtain balance changing operations, and construct airgapped transactions. With a special focus on getting balance

change operations, Celo Rosetta provides an easy way to obtain changes that are not easily queryable using the celo-blockchain RPC.

```
# general.md:
```

```
---
```

```
title: Celo General Integration Information
description: General information about integrations regardless of your
service or use case.
```

```
---
```

General

General information about integrations regardless of your service or use case.

```
---
```

Accessing the chain

There are a myriad of ways through which you can access chain data:

Running your own node

To be completely independent and have a reliable view into the latest chain data, you will likely want to run your own node(s).

You can just clone celo-blockchain and then run make geth to receive the binary.

By default, geth will use /root/.celo as the data dir, if you would like to change that specify the --datadir argument.

This is all you should need to connect to a network:

For Mainnet:

```
bash
geth
```

For Alfajores:

```
bash
geth --alfajores
```

For Baklava:

```
bash
geth --baklava
```

For more command line options, please see <https://github.com/ethereum/go-ethereum/wiki/Command-Line-Options>

Forno

Forno is a hosted node service for interacting with the Celo network. This allow the user to get connected to the Celo Blockchain without having to run its own node.

Can be used as an Http Provider with ContractKit

As Forno is a public node you will have to sign transactions locally because with your own private key, because Forno doesn't store them. But don't worry, the ContractKit will handle this for you.

Forno networks:

Alfajores = 'https://alfajores-forno.celo-testnet.org'

Baklava = 'https://baklava-forno.celo-testnet.org'

Mainnet = 'https://forno.celo.org'

Blockscout

We also expose data on the cLabs run blockscout instance. Blockscout itself exposes an API.

Alfajores = 'https://alfajores-blockscout.celo-testnet.org'

Baklava = 'https://baklava-blockscout.celo-testnet.org'

Mainnet = 'https://explorer.celo.org/'

Signing Transactions

Compared to Ethereum transactions, Celo transactions have an additional optional field:

- feeCurrency - Specifies the address of the currency in which fees should be paid. If null, the native token CELO is assumed.

```
<!-- TODO: Fix this link when this part of the docs is done  
Read more about Celo Transactions  
-->
```

To sign transactions, you have the following options:

- Use the JSON-RPC sendTransaction method to your node which would have the account in question unlocked. (Either manually or via a library such as web3)
- Use ContractKit's local signing feature.

index.md:

title: Integrate with Celo

description: Collection of resources to help integrate Celo with your service.

import PageRef from '@components/PageRef'

import Tabs from '@theme/Tabs';

import TabItem from '@theme/TabItem';

Celo Integrations

Collection of resources to help integrate Celo with your service.

Celo provides you with the tools to easily integrate DeFi into your existing mobile application or blockchain service. Integrating with Celo allows you to accept payments, send payouts, and manage all of your DeFi needs using our global financial infrastructure.

- General information
- Integration Checklist
- Custody
- Listings
- Using a Cloud HSM

listings.md:

title: Celo Listing Integrations

description: Support for digital asset exchanges or ranking sites that would like to run a Celo node and audit your setup.

Listings

Support for digital asset exchanges or ranking sites that would like to run a Celo node and audit your setup.

Support

If you have any questions or need assistance with these instructions, please contact cLabs or ask in the #exchanges channel on Celo's Discord server. Remember that Discord is a public channel: never disclose recovery phrases (also known as backup keys, or mnemonics), private keys, unsanitized log output, or personal information.

This guide will also help you find all the necessary information about brand assets, how to integrate with Celo and what useful listing information are made available to you as well as any information about looking for support.

Celo Brand Assets for Listing

If you are listing Celo on your exchange, you will probably need access to the Celo Platform brand assets. They can be found here.

Please ensure your use of the Celo Platform assets provided follows the brand policy found here.

How To's

Integrating Celo With Your Infrastructure

There are several ways to integrate the Celo Platform with your infrastructure.

A general overview of integrations that would be relevant to you listing Celo Platform are shown here.

For more specific use-cases for exchanges, please checkout the Custody and Exchange Integration Guide as well.

Important Information

Celo Native Asset and Stable Value Currencies

There are key assets on the Celo network, the Celo native asset (CELO) and Celo-powered Stable Value Currencies, such as Celo Dollar (cUSD) and Celo Euro (cEUR). CELO was formerly called Celo Gold (cGLD) when the contract was deployed, so you will often see references to Celo Gold and CGLD in the codebase. To learn more about the two, please read this section of the docs.

You can also view the forum post about the name change here.

Resources

Address for CELO and Stable Value Currencies

- CELO (\\$CELO) - 0x471ece3750da237f93b8e339c536989b8978a438
- Celo Dollar (\\$cUSD) - 0x765de816845861e75a25fca122bb6898b8b1282a
- Celo Euro (\\$cEUR) - 0xd8763cba276a3738e6de85b4b3bf5fded6d6ca73
- Celo Brazilian Real (\\$cREAL) - 0xe8537a3d056DA446677B9E9d6c5dB704EaAb4787

Useful API endpoints

The following are useful API endpoints available to you that would help you in your listings of the CELO and cUSD digital assets.

CELO and Stable Value Currencies

Total CELO supply

For querying the API on total coins in circulation in CELO, which are the total amount of coins in existence right now, the following endpoint will provide you with that:

```
sh
$ curl https://thecelo.com/api/v0.1.js?method=extotalcoins
{"code":"200","msg":"success","data":{"CELO":608485841.9959723,"cUSD":10250632.56099673}}
```

Stable Value Currencies

cUSD Circulating Supply

Circulating Supply refers to the of coins that are circulating in the market and in the general public's hands.

```
sh
$ curl https://thecelo.com/api/v0.1.js?method=excusdcirculating
11353464.550486518
```

cEUR Circulating Supply

This endpoint is not yet available.

CP-DOTO (Stability Algorithm)

CP-DOTO information can be found [here](#).

For API endpoints useful for listing that follow CMC requirements

Mento Addresses

- cUSD/CELO contract - 0x67316300f17f063085Ca8bCa4bd3f7a5a3C66275
- cEUR/CELO contract - 0xE383394B913d7302c49F794C7d3243c429d53D1d
- cREAL/CELO contract - 0x8f2cf9855C919AFAC8Bd2E7acEc0205ed568a4EA

Summary

Summary overview of market data for all tickers and all markets. These endpoints don't yet support cEUR.

```
sh
```

```
$ curl https://thecelo.com/api/v0.1.js?method=exsummary
```

```
{"tradingpairs":"CELOCUSD","lastprice":2.6143,"lowestask":2.5933609958506225,"highestbid":2.5676,"basevolume":37524.320000000003,"quotevolume":14714.520000000002,"pricechangepercent24h":3.7027120070382127,"highestprice24h":2.649,"lowestprice24h":2.4787}}
```

Assets

In depth details of the assets available on the exchange.

```
sh
```

```
$ curl https://thecelo.com/api/v0.1.js?method=exassets
```

```
{"code":"200","msg":"success","data":{"CELO":{"name":"CELO","unifiedcryptoassetid":"5567","canwithdraw":"true","candeposit":"true","minwithdraw":"0.000000000000000001","maxwithdraw":"0.000000000000000001","makerfee":"0.00","takerfee":"0.005"},"CUSD":{"name":"Celo Dollars","unifiedcryptoassetid":"825","canwithdraw":"true","candeposit":"true","minwithdraw":"0.000000000000000001","maxwithdraw":"0.0000000000000000000001","makerfee":"0.00","takerfee":"0.005"}}
```

Ticker

24-hour rolling window price change statistics.

```
sh
```

```
$ curl https://thecelo.com/api/v0.1.js?method=exticker
```

```
{"code":"200","msg":"success","data":{"CELOCUSD":{"baseid":"5567","quoteid":"825","lastprice":2.6124,"quotevolume":14789.520000000002,"basevolume":37720.300000000003,"isFrozen":"0"}}
```

Orderbook

Market depth of a trading pair. One array containing a list of ask prices and another array containing bid prices.

```
sh
```

```
$ curl https://thecelo.com/api/v0.1.js?method=exorderbook
```

```
{"code":"200","msg":"success","data":{"timestamp":1601061465962,"bids":[["2.5964","100"]],"asks":[["2.622606871230003","100"]]}}
```

CELO cUSD

Recently completed (past 24h) trades.

```
sh
```

```
$ curl https://thecelo.com/api/v0.1.js?method=excelocUSD
```

```
{"code": "200", "msg": "success", "data": {"CELOCUSD": [{"tradeid": 2697341, "timestamp": 1601061491, "price": 0.38238291620515147, "quotevolume": 25, "basevolume": 65.37948987916423, "type": "Sell"}, {"tradeid": 2697336, "timestamp": 1601061466, "price": 0.382293821845672, "quotevolume": 25, "basevolume": 65.39472670341044, "type": "Sell"}]}}
```

Whitepapers

To learn about the Celo Protocol, please refer to the whitepaper.

If you need more information to explore other aspects of the Celo Protocol, there's a useful links page.

To learn more about the Stability Mechanism, you can find it over here. The Stability Analysis Whitepaper and blog post will provide a lot more information on the stability algorithm.

If you want to find more information about the Celo Reserve, a diversified portfolio of cryptocurrencies supporting the ability of the Celo protocol to expand and contract the supply of Celo stable assets, please visit <https://celoreserve.org>.

Github

The Celo Protocol GitHub is located here.

Audits

All the security audits on the smart contracts, security and economics of the Celo Platform can be found here.

```
# add-gas-currency.md:
```

```
---
```

```
title: Adding Gas Currencies to Celo
```

```
description: Documentation relevant to how to add currencies on Celo
```

```
---
```

Adding Gas Currencies to Celo

The Celo protocol supports paying gas in tokens other than the native one, Celo. This document outlines requirements and processes to add a new gas token.

Pre-requisites

Token implementation

A gas token is a ERC-20 token that has to implement two functions with the following signatures:

```
function debitGasFees(address from, uint256 value) external;

function creditGasFees(
    address refundRecipient,
    address tipRecipient,
    address gatewayFeeRecipient,
    address baseFeeRecipient,
    uint256 refundAmount,
    uint256 tipAmount,
    uint256 gatewayFeeAmount,
    uint256 baseFeeAmount
) external;
```

Both these functions should have a modifier to make sure they can only be called by the zero address, which is the address the Celo VM impersonates when calling smart contracts. An example of the implementation of this modifier can be found in Celo's monorepo.

debitGasFees is called by the VM before the body of the tx is executed while assembling a block. It removes from the caller's balance the maximum amount of tokens the transaction will take. This is done to make sure the user has enough to pay for gas.

creditGasFees is called by the VM after the body of the tx is executed while assembling a block. It distributes the fees to the validator signers and the contract that the protocol designated to accumulate base fees, as well as refunding the user for unused gas. The addresses of the fee recipients as well as the amounts are passed by the VM as parameters.

As a consequence, debitGasFees and creditGasFees are executed atomically in the same tx.

If any of these two functions reverts, the transaction will not be included in a block.

Example implementations of debitGasFees and creditGasFees can be found in Mento's StableCoins contracts codebase.

Oracle

Celo blockchains uses the core contract SortedOracle as a the source of the rate the tokens should be priced at the moment a validator is attempting to include the transaction in a block. The address of the token and the address of the oracle has to be added to SortedOracle using the function addOracle(address token, address oracleAddress).

Then, at least one rate should have been submitted using `report(address token, uint256 value, address lesserKey, address greaterKey)`. The price reported is the price of Celo in units of the token to be added.

An reference implementation of oracle provider client can be found [here](#).

Sorted Oracles was originally designed to support the Mento protocol. For this reason, the reference implementation of the oracle client supports reporting with a high frequency (in the context of a blockchain). If the goal of the oracle is solely to support a gas currency, reports could be significantly more spaced out, as gas pricing is not as sensible to manipulation as an arbitrage protocol.

Governance Process

To enable a token as gas currency, two governance proposals need to be passed.

Enabling the Oracle

The first proposal is meant to enable the oracle, by calling `SortedOracle.addOracle(address token, address oracleAddress)`. Potentially, this proposal could also transfer some Celo from the Community Fund to the oracle addresses to pay for gas.

An example of such proposal can be found [here](#).

Reporting

Before submitting the second proposal, at least one of the oracle addresses need to call `report(address token, uint256 value, address lesserKey, address greaterKey)`. This will make a price available to the protocol for gas pricing.

Enabling as Gas Token

The second proposal enables the gas token by calling `addToken(address tokenAddress)`. After this proposal passes, EOAs should be able to pay for gas in the enabled token.

An example of such proposal can be found [here](#).

It would be a good consideration to update popular tooling (like `contractkit`) before this proposal passes so that most developers are ready to use the new gas as soon as it enabled.

Enabling with just one proposal

It is possible to simplify this process to one governance proposal. That'd proposal would have to enable the Governance contract as an oracle for the token to enable as gas token.

Then, the governance proposal itself should report a reference value within the same proposal. This is particularly safe to do with stablecoins.

After this, and as cleanup, Governance should be removed from the list of oracles and the calls from the two previous proposals should be included to finalize the process.

Ongoing effort

Oracles are expected to keep reporting prices periodically, especially in moments of high volatility.

CELO-coin-summary.md:

title: Celo

description: Summary of CELO and the value it provides to the community.

CELO

Summary of CELO and the value it provides to the community.

Platform-Native Digital Asset

CELO is the platform-native asset that supports the growth and development of the Celo blockchain and ecosystem. CELO holders can earn rewards, stake with validators, and vote on proposals that shape the future of Celo.

CELO aligns incentives of Celo stakeholders

- Celo's native asset is the core utility, reserve, staking, and governance asset
- CELO has a fixed supply and variable value, related to the use of the platform, governance and the amount of stablecoins in use.

Utility

CELO's primary function is stabilization, ensuring the healthy velocity of stablecoins that globally circulate and transact on Celo. Serving as the primary reserve asset in Celo's groundbreaking stability mechanism (Mento), CELO allows platform-native Mento stablecoins like Celo Dollars (cUSD) and Celo Euros (cEUR) to algorithmically follow the value of their namesake currencies.

Governance

CELO plays a critical role in the governance of the Celo platform and technology. Anyone who holds any amount of CELO is empowered to vote on

governance proposals that direct how the core technology operates today and in the future.

Staking

Leading the industry in energy-efficient, carbon neutral blockchain technology, Celo's energy-efficiency is born out of its Proof-of-Stake (PoS) consensus protocol, requiring it's daily elected validators to stake CELO as a signal of participation in confirming transactions on the blockchain, earning Celo Dollars (cUSD) as compensation to cover costs in return.

Learn more about CELO

An introductory Guide to Celo

```
# celo-economic-model.md:
```

```
---
```

```
title: Celo Economic Model
```

```
description: Celo's economic model is derived from research in Sacred Economics and Theory of Change.
```

```
---
```

```
import YouTube from '@components/YouTube';
```

Economic Model

Celo's economic model is derived from research in Sacred Economics and Theory of Change.

```
---
```

Features of Money

Sep Kamvar speaks about the Future of Value.

```
<YouTube videoId="tX5ep1JzY6k"/>
```

Sacred Economics

Celo's philosophy is grounded in the work of Charles Eisenstein's Sacred Economics.

Theory Of Change

A theory of change is a framework that describes how short-term actions will lead to long-term social impact. Celo's aim is to create the conditions for prosperity, defined as people fulfilling basic needs, growing along their own unique paths, and supporting each other. All of Celo's projects have a specific hypothesis that ties our activities to one or more of these indicators of prosperity.

celo-ecosystem.md:

title: Celo Ecosystem

description: Celo tools, assets, and community are focused on creating a future where everyone can prosper.

Ecosystem

Celo tools, assets, and community are focused on creating a future where everyone can prosper.

Mainnet

The Celo Mainnet is Celo's production network and was launched in April 2020. Mainnet is used to describe when a blockchain protocol is fully developed and deployed, meaning that cryptocurrency transactions are being broadcasted, verified, and recorded on a distributed ledger technology (blockchain).

CELO

CELO is a platform-native reserve and governance asset, serving as the primary asset in the stability mechanism (Mento) that supports stable digital currencies like cUSD and cEUR. CELO's primary function is stabilization, ensuring the healthy velocity of assets that globally circulate and transact on Celo. CELO also serves as the key governance mechanism for the Celo Platform. Owning and using CELO supports the growth and development of the Celo Platform by enabling each CELO owner to vote on proposals that direct how the core technology operates today and in the future.

Stable Assets

The Celo Reserve supports the Celo Platform and helps to ensure the value of stabilized assets that circulate on Celo. The Celo Reserve supports the stability of stabilized value assets by hosting a diversified portfolio of cryptocurrencies supporting the ability of the Celo protocol to expand and contract the supply of Celo stable assets, in line with user demand. The Celo Reserve is composed of cryptocurrencies including CELO, BTC, ETH, and DAI—ensuring that all user transactions with the reserve can happen on-chain, in a decentralized manner, in fully auditable ways.

Celo Dollars (cUSD)

Celo Dollars (cUSD) are Celo's first Mento stable asset that tracks the value of the U.S. dollar. Celo Dollars were designed to be as usable and stable as everyday digital money with the added benefits of fully mobile global circulation. Because Celo Dollars are digital, stable, and mobile-

first, users can easily save, pay, lend, and send cUSD to any mobile number in the world in faster, cheaper, and more secure ways.

Celo Euro (cEUR)

Celo Euro (cEUR) is Celo's Mento stable asset that tracks the value of the Euro. Celo Euro was designed to be as usable and stable as everyday digital money with the added benefits of fully mobile global circulation. Because Celo Euro are digital, stable, and mobile-first, users can easily save, pay, lend, and send cEUR to any mobile number in the world in faster, cheaper, and more secure ways.

The Celo Foundation

The Celo Foundation is a U.S.-based organization that helps to bring Celo's mission to life, believing in a future where everyone can prosper through financial inclusion, environmental health, and connectivity through the growth and development of innovative technologies, including the open-source, mobile-first Celo Platform and ecosystem of technologies. The Celo Foundation contributes grants to education, technical research, environmental health, community engagement, and ecosystem development and outreach efforts - activities that support and encourage an inclusive and sustainable financial system that creates the conditions for prosperity-for everyone.

The Alliance for Prosperity

The Alliance for Prosperity is a group of mission-aligned Celo Foundation members focused on developing use cases for the Celo Platform and ecosystem of technologies, furthering the mission of financial inclusion and prosperity, and driving adoption of the Celo blockchain and ecology of digital assets.

By joining the Alliance for Prosperity, Alliance members are furthering the Celo mission in one (or more) of five ways: (1) Backing project development efforts, (2) Building infrastructure, (3) Implementing desired use cases on the Celo Platform, (4) Integrating Celo assets in their projects, (5) Collaborating on education campaigns in their communities to advance the use of blockchain technology.

- Alliance members are welcomed along all of the use cases outlined on celo.org/alliance. Feel free to reach out via the application if your organization is interested in membership.

Celo Camp

Celo Camp is a virtual accelerator and competition focused on helping startups and developers build decentralized financial apps (dApps) on Celo. During Celo Camp, teams will build mobile-first dApps and other financial tools and services on Celo. In addition to receiving technical and Web3 support, teams receive guidance from experienced mentors, alongside a custom-built curriculum, that will set them up for the next phase of their entrepreneurial success.

The Celo Foundation Grants Program

The Celo Foundation Grants Program is an open initiative supporting projects committed to the mission of building a financial system that creates the conditions for prosperity for all, focusing grants based on four key areas: (1) fostering innovation and development of the Celo Platform and Celo reference wallets, (2) lowering barriers to entry to encourage the accessibility of Celo to developers and end-users, (3) educating and expanding Celo's global community, (4) expanding Celo's access, opportunity, and impact.

Celo Ambassadors

Celo Ambassadors are active Celo community members who are passionate about furthering Celo's mission by driving the adoption of the Celo ecosystem. To become an ambassador, [sign up here](#)

Kuneco

Kuneco is a monthly community call designed to bring the Celo community together in celebration of achievements, growth, milestones, and progress towards prosperity.

cLabs

cLabs is a member of the Alliance for Prosperity and part of the community working on Celo. Based in Berlin, Buenos Aires, San Francisco, with Partners based all around the world, cLabs' first product release is Valora, a mobile payments app built on Celo.

Valora

Valora is a global payments app native to the Celo Platform available on both iOS and Android devices. Valora debuts a utility that finally makes blockchain useful and accessible on even the most basic smartphones, enabling the 6 billion smartphone users worldwide to effortlessly and reliably send, receive, and store value on Celo.

celo-highlights.md:

title: Key Concepts

description: Summary of benefits, features, and data behind the Celo platform.

Key Concepts

Summary of benefits, features, and data behind the Celo platform.

The Platform for Mobile DeFi

The Celo protocol provides a platform upon which the Celo community can create stabilized value digital assets. Named for the currencies they follow, Celo Dollars (cUSD) and Celo Euros (cEUR) are stablecoins that allow anyone to share value faster, cheaper and more easily from a mobile phone.

The Celo protocol also includes mechanisms for lightweight identity and ultralight mobile clients.

Innovative tools to build native mobile dApps:

- Stable Value Currencies
- Phone Number Public Key Infrastructure
- On-chain Governance
- Self Custody
- Proof-of-Stake
- Open-source
- Permissionless
- High Speed Sync for Ultra-light Clients
- Gas Payable in Multiple Stablecoins
- Programmable (full EVM Compatibility)

Optimized for Financial Applications

Powered by Celo's industry-leading decentralized phone number verification, payment applications built on Celo allow users to easily send or request digital currencies from any mobile number, anywhere in the world, capable of offering their users features like:

- Non-custodial wallets
- Ultra low network transaction fees
- Digital currency exchange capabilities
- QR Code Support
- Mobile first SDK

Core Contracts

Designed to support an ecology of stable value currencies. The first Mento stablecoin, cUSD, tracks the value of the US Dollar.

- Algorithmic reserve-backed stabilization mechanism
- Crypto-asset collateralized
- Native support for multiple stablecoins

Blockchain

Open source permissionless smart contract platform built on decentralized infrastructure.

- Proof-of-Stake based consensus with high throughput, low latency, and zero carbon
- Incentives for serving mobile devices
- On-chain governance

Build on Celo

Visit celo.org to learn more.

Details

Celo is a mobile-first, carbon-neutral blockchain that makes decentralized financial (DeFi) tools and services accessible to anyone with a mobile phone-bringing the powerful benefits of DeFi to the users of the 6 billion smartphones in circulation today.

Stable Value Currencies

Celo includes native support for multiple ERC20-like stable currencies pegged to 'fiat' currencies like the US dollar, to facilitate the use of Celo as a means of payment.

Accounts Linked to Phone Numbers

Celo maintains a secure decentralized mapping of phone numbers that allow wallet users to send and receive payments with their existing contacts simply and with confidence that the payment will reach the intended recipient.

Transaction Fees in Any Currency

Users can pay transaction fees in stable currencies so that they do not need to manage the balances of different currencies.

Immediate Syncing Even on Slow Connections

Extremely fast, secure synchronization between mobile devices and the Celo network means that even wallet users with high latency, low bandwidth, or high-cost data tariffs can use Celo. Celo removes the need to check every header before a received header can be trusted. The performance will be further improved with BLS signature aggregation and succinct zero-knowledge proofs, via zk-SNARKs.

Proof-of-Stake

Celo uses a Proof-of-Stake (PoS) consensus algorithm. In comparison to Proof-of-Work (PoW) systems like Bitcoin and Ethereum, this eliminates the negative environmental impact and means that users can make transactions that are cheaper, faster, and where the outcome cannot be changed once complete.

On-chain Governance

Since great user experiences and operating a modern networking protocol need iteration and improvement, Celo supports rapid upgrades and protocol

changes via on-chain governance in which all CELO holders can participate.

Programmable (Full EVM Compatibility)

Celo includes a programmable smart contract platform that is compatible with the Ethereum Virtual Machine (EVM), which is already widely adopted, familiar to developers and has strong tool support. This enables Celo to deliver rich user features and rapidly support a wide ecosystem of third-party applications and extensions.

Self Custody

Users have access to and fully control their funds and account keys, and don't need to depend on third parties to make payments.

Metrics

Validators

jsx
110 Globally

Celo has a validator set with a current size of 110 validators. Validators are elected to this set on a daily basis; typically more validators stand for election than seats available. The validator set size may be raised in the future through the on-chain governance process.

You can view more detailed information about the current validator set at stats.celo.org.

Total Blocks

For real-time updates and information, please visit explorer.celo.org

Average Block Time

Celo's average blocktime is 5 seconds. For real-time updates and information, please visit <https://explorer.celo.org/>

Average User-Perceived Transaction Time Average Network Transaction (Gas) Fee

Transactions are interactions that occur between user addresses and Celo (e.g., transferring funds).

< \$0.01 Average Network Transaction Fee

:::tip

Real-time gas fees are captured in gwei on stats.celo.org.

:::

Circulating Supply

- CELO: Metric updated in real-time, please visit coinmarketcap.com/currencies/celo
- cUSD + cEUR + cREAL: Metrics updated in real-time, please visit celoreserve.org

:::tip

For current circulating supply of cUSD and cEUR please see the section titled 'outstanding supply' and reference the top number in black.

:::

3007.7 Tons of Carbon Offset

For real-time updates, please visit wren.co/profile/celo

:::tip

Tons of carbon offset reflect Celo community purchases of carbon credits by way of funding the Community tree planting project with Wren.

:::

celo-milestones.md:

title: Celo Milestones
description: Important milestones for the Celo protocol and ecosystem.

Milestones

Important milestones for the Celo protocol and ecosystem.

Deutsche Telekom

Deutsche Telekom made a strategic purchase of the Celo native digital asset (CELO) and joined the Celo Alliance for Prosperity as the first mobile carrier member. They also became a validator on the Celo network. This partnership is very aligned with Celo's mission and the services Deutsche Telekom provides its customers worldwide, the majority of which primarily use their phones to access financial services. By building on Celo's mobile-first platform, mobile carriers like Deutsche Telekom are able to seamlessly integrate mobile-friendly blockchain-based solutions and serve customers in new, meaningful ways. Deutsche Telekom is leading in this effort, and we're excited to see how mobile carriers and other

organizations take advantage of all that blockchain and cryptocurrency has to offer.

Celo Euro (cEUR)

We launched Celo Euro (cEUR), our second native Mento stablecoin on the platform for mobile payments that we've launched in the last year. cEUR creates an entirely new remittance market for the European Union where users can quickly and easily transfer digital money between countries with a phone number. It's as easy as sending a text message and can be done in as little as 5 seconds for less than \$0.01. Combined with the open nature of Celo's platform, we can enable a mobile-friendly form of digital money that can be used by the 6 billion smartphone users around the world.

Optics

We introduced Optics, a new gas-efficient bridging standard that connects Celo with Ethereum. Optics enables interoperability between layer-one blockchains. It's trustless, non-custodial, and designed to minimize gas costs to users. This means that developers can permissionlessly build custom cross-chain applications, cheaply move tokens and data between chains, and help users reach existing applications on chain.

PayU

PayU, one of the largest payment providers for emerging markets, is giving its nearly half a million merchants the ability to accept cUSD as a payment option. PayU's merchant and customer base is mostly concentrated in high-growth markets, such as Latin America, Africa, and Southeast Asia, so having a decentralized, over-collateralized algorithmic stablecoin provides access to USD and EUR-backed digital assets that are insulated from the volatility of some fiat currencies and cryptocurrencies more broadly. This integration is a big step toward making digital assets easier for customers to use and for merchants to accept, which we hope will accelerate the adoption of stablecoins in more communities around the world.

Valora

Valora is now a standalone company! When the cLabs team officially launched Valora in February, the immediate uptick in user adoption far exceeded what we envisioned at that stage of the project. It's since continued on this upward trajectory: today, Valora has 200K users with a balance and 53K monthly active users in more than 100 countries. This reception made clear that Valora has grown to become more than a project on the Celo platform and has the potential to scale even further. That fueled the decision to make Valora an independent, standalone company, with Jackie Bona—who cLabs' Head of Consumer Growth for Valora—as the Chief Executive Officer. The Valora team also raised a \$20 million Series A to fuel the company's growth.

Carbon Negative

In May we shared that Celo officially became one of the first carbon negative blockchains and highlighted some community initiatives underway to offset carbon emissions on Celo. Building on that, we recently submitted an on-chain government proposal to allocate .5% of the Celo Reserve to natural backed assets, the first being Mc02 tokens. This makes Celo the first blockchain to incorporate natural backed assets into its reserve, enabling users to support and protect the environment—just by using Celo Dollar (cUSD)—and we hope it will set a new standard for the industry.

Donut Hardfork

On May 19th 2021, the donut hardfork went live. The Donut Hardfork is a non-contentious hardfork that includes many exciting network upgrades making Celo more gas efficient, improving interoperability, and allowing Celo users to connect to popular tools like MetaMask.

Important updates from the Donut Hardfork:

- CIP-20 | Toolkit for adding cryptographic functions that are useful to smart-contract devs
- CIP-25: Will help enable future bridging to Solana, Cosmos, and NEAR thanks to Chorus One
- CIP-35: Makes Ethereum transaction types accessible on Celo, which will enable access to all Ethereum's tools

celo-onboarding.md:

title: Celo Onboarding

description: Learn the basics of the Celo platform.

Celo Onboarding

Learn the basics of the Celo platform.

What is Celo?

Celo is a mobile-first and carbon-neutral blockchain that makes decentralized financial (DeFi) tools and services accessible to anyone with a mobile phone—bringing the powerful benefits of DeFi to the users of the 6 billion smartphones in circulation today.

With its interoperability, cross-chain compatibility and vision for inclusivity, Celo enables native and non-native digital assets—both cryptographic and Central Bank Digital Currencies (CBDCs)—to circulate freely, at extremely low costs and high speeds, across devices, carriers, and countries—making money mobile, global and accessible like never before.

Getting Started with Celo

- Hello from Celo
- What if money were beautiful?
- Cryptocurrency for a Beautiful Planet
- Meet the team working on Celo
- Celo Tech Talks Building a mobile first blockchain platform

Celo Basics

- What is the Celo Platform?
- What is CELO?
- What can Celo Dollars do?

Celo Tour

- About Celo
- Join
- Validators
- Developers
- Alliance
- Community
- Blog
- GitHub

Celo First Steps

- Valora
- Get CELO
- Get Test Funds
- Block Explorer

Connect with the Community

- Reddit
- Blog
- GitHub
- Twitter
- YouTube
- Instagram
- Defi Pulse
- LinkedIn
- Twitch

Get Support

- Discord
- Forum
- Telegram

celo-overview.md:

title: Welcome to Celo

description: Celo's mission, vision, and goals for financial prosperity for everyone.

Welcome to Celo

Celo's mission, vision, and goals for financial prosperity for everyone.

Crypto made for Mobile

Celo is a mobile-first blockchain that makes decentralized financial (DeFi) tools and services accessible to anyone with a mobile phone. It aims to break down barriers by bringing the powerful benefits of DeFi to the users of the 6 billion smartphones in circulation today.

Celo's Mission

The company's mission is to build a financial system that creates the conditions of prosperity for everyone. Celo enables native and non-native digital assets-both cryptographic and Central Bank Digital Currencies (CBDCs)-to circulate freely across devices, carriers, and countries. This makes money mobile, global and accessible like never before. Celo is supported by a community of organizations and individuals, including Jack Dorsey, a16z, and Deutsche Telekom.

Watch: What if Money were Beautiful

Celo Ecosystem

An Ethereum-compatible technology capable of reaching global users at scale, Celo turns crypto into usable money with a multi-asset system: a governance and staking asset (CELO) and a family of Mento stablecoins. Since the launch of Mainnet in 2020, Celo's network now supports 1000+ projects from builders, developers, and artists, who everyday create new applications and issue digital currencies from over 100 countries around the world.

CELO is a platform-native reserve and governance asset, serving as the primary asset in the stability mechanism (Mento) that supports stable digital currencies like cUSD and cEUR. CELO's primary function is stabilization, ensuring the healthy velocity of assets that globally circulate and transact on Celo. CELO also serves as the key governance mechanism for the Celo Platform. Owning and using CELO supports the growth and development of the Celo Platform by enabling each CELO owner to vote on proposals that direct how the core technology operates today and in the future.

Watch: Meet the team working at Celo

Get started with Celo

To build technology and products that are used and loved by people and solve real-world problems, Celo is building a community with many different perspectives and experiences. Let's build a monetary system that creates the conditions for prosperity for all.

Celo-payments.md:

title: Celo Payments SDK

description: Celo's Pay is a standard protocol along with a few reference implementations on how developers can integrate decentralized payments into their dApps, wallets and services.

Celo Protocol

Summary of the Celo Payments Protocol and the value it provides to the community.

Read the specification.

The Platform for Mobile Payments

The Celo blockchain has a transaction finality of less than five seconds with less than \$0.001 fees, it is fully decentralized and it's a carbon negative blockchain to preserve our most precious resources.

Stable Value Currencies

Powered by a platform algorithmic native stable coins, Celo Dollars (cUSD), Celo Euros (cEUR) and Celo Reals (cREAL) are ideal for making payments on your local currency through your mobile phone number. These stablecoins can be used for our merchant payments API and developers can choose which ones they want to expose for their users.

Supporting Wallets

- Valora
 - iOS
 - Android)

Requirements

This is the exhaustive list of requirements to run this repository

- node 12
- yarn
- A CELO private key for development purposes.
- A DEK for development purposes.

Project Structure

The following is a short description of each of the package directories:

packages/cli - CLI code and scripts that demonstrate the SDK capabilities

packages/sdk - the actual SDK implementation code

packages/server - an example/reference code that demonstrate how a typical server might use the SDK

packages/types - protocol compliant schemas and types

Regarding the CELO private key

For one-off uses, simply use `celocli account:new`. To get the CELO CLI, follow the instructions here: <https://docs.celo.org/cli>.

To make things a bit simpler, I recommend running that command to create a `.env` file for testing:

Make sure you're working on alfajores network

```
celocli config:set --node https://alfajores-forno.celo-testnet.org/
```

If this is the first time you're running this command, the script will create a blockchain account/private-key for you and ask you to fund it using Celo faucet. Then this account will be used automatically to pay for the requested payment.

Create an account and store it well formatted in an `.env` file

```
celocli account:new | sed -E 's/: (.+)/="\\1"/g' | grep '=' > .env  
source .env
```

Copy the account address to your clipboard

```
echo $accountAddress | pbcopy
```

Head to the faucet to get some money and paste your account address there
open <https://faucet.celo.org>

Verify you got money successfully

```
celocli account:balance $accountAddress
```

Register your account

```
celocli account:register --from $accountAddress -k $privateKey
```

Regarding the DEK

For the server to know about your DEK, your account created above needs to be registered and have the public key of your DEK registered too.

Why do you need a DEK?

First, some context: the payment SDK is meant to be used within Valora initially, and they already handle creating keys and accounts under the

hood for the user. However, it's important to understand why you need to sign with a different key than the root key of your account. Generally we want a separation of concerns when dealing with private keys, they should do one thing and one thing only. Your account key, as it can move funds, is like the keys to the castle and should be kept as cold as possible. Your DEK, vote signing key, etc are less sensitive a

In order to do that, this command may be useful:

Register a public data encryption key
celocli account:register-data-encryption-key --from \$accountAddress -k privateKey --publicKey <DEKPUBLICKEY>

How to run each package

The payments monorepo is composed of a couple packages. But only 2 are meant to be used from the outside: the CLI and the development server. The other packages (sdk, types, and utils) are used by the CLI and server respectively.

First, run yarn && yarn lerna bootstrap to install the dependencies and bootstrap the monorepo.

The payments API reference server

You can run yarn start to run the server with hot-reloading enabled. It will run by default on port 3000, useful for the next section.

The payments CLI

Run the payment CLI in interactive mode yarn cli init -p <PRIVATEKEY> -d <DEK> -u http://localhost:3000 -r SIMPLE from the root of the repository. You can also see the full list of available flags via yarn cli init --help and all available commands via yarn cli help. You may also omit all the flags to run the command interactively.

The reference servers implements two types of payments: KYC and SIMPLE, you try both via the -r flag. If you decide to implement a new purchase example, you can find them in the folder packages/server/src/storage/items.

Accepting Payments

This is the main export that wallets will want to integrate with. It allows you to get the payment info and subsequently make the payment transaction.

Example run through of Charge usage

Here is the api url that the Charge instance will be communicating with.

typescript

```
const apiBase = "merchantpayments.com/api";
```

The id of the payment request used by the api. The api will need to create a payment object for the SDK to respond to. This will need to have the info in the PaymentInfo type and the referenceId will refer to this object.

```
typescript
const referenceId = "123abc";
```

The 'ChainHandler' instance imported from the payments-sdk and initialized with a contract kit instance. This kit will represent the Payer in the process.

```
typescript
const chainHandler = new ContractKitTransactionHandler(kit);
```

Whether or not a DEK should be used for authorizing on chain transactions.

```
typescript
const useAuthentication = true;
```

How many times requests should be retried.

```
typescript
const retries = 4;
```

```
const charge = new Charge(
  apiBase,
  referenceId,
  chainHandler,
  useAuthentication,
  retries
);
```

The info regarding the payment matching the reference id coming from the api. See @celo/payment-types PaymentInfo. Includes the requiredPayerData field that must be used for the submit method. Also, includes payment meta data to show to the user.

```
typescript
const paymentInfo: PaymentInfo = await charge.getInfo();
```

Examples

How much you want to set as the payment:


```
typescript
console.log(paymentInfo.action.amount);
```

Specifying the token you want to use

```
typescript
console.log(paymentInfo.action.currency);
```

Merchants might require some KYC data on who is paying for the transaction. This will be passed into the submit method.

```
typescript
const payerDataExample = {
  phoneNumber: '12345678',
};

try {
```

This is the method to submit the transaction on chain

```
typescript
  await charge.submit(payerDataExample);
} catch(e) {
```

If for some reason the transaction fails to submit the promise returned by submit will be rejected.

The charge can be aborted to let the api know not to continue watching for the transaction. See @celo/payment-types AbortCodes for abort code options.

```
typescript
charge.abort(AbortCodes.INSUFFICIENTFUNDS);
```

Reaching here would mean the payment was successfully submitted on chain.

```
typescript
console.log("Payment submitted");
```

ChainHandlers

Wrappers to help the PaymentsSDK interact with the blockchain.

ContractKitTransactionHandler

Used to wrap ContractKit to make a ChainHandler for the Charge class

Helpers

A variety of helper methods to facilitate payments-sdk interactions

celo-protocol-summary.md:

title: Celo Protocol

description: Summary of the Celo Protocol and the value it provides to the community.

Celo Protocol

Summary of the Celo Protocol and the value it provides to the community.

The Platform for Mobile DeFi

The Celo protocol provides a platform upon which the Celo community can create stabilized value digital assets. Named for the currencies they follow, Celo Dollars (cUSD), Celo Euros (cEUR) and Celo Reals are Mento stablecoins that allow anyone to share value faster, cheaper and more easily from a mobile phone.

The Celo protocol also includes mechanisms for lightweight identity and ultralight mobile clients.

Innovative tools to build native mobile dApps:

- Stable Value Currencies
- Phone Number Public Key Infrastructure
- On-chain Governance
- Self Custody
- Proof-of-Stake
- Open-source
- Permissionless
- High Speed Sync for Ultra-light Clients
- Gas Payable in Multiple Stablecoins
- Programmable (full EVM Compatibility)

Optimized for Financial Applications

Powered by Celo's industry-leading decentralized phone number verification, payment applications built on Celo allow users to easily send or request digital currencies from any mobile number, anywhere in the world, capable of offering their users features like:

- Non-custodial wallets
- Ultra low network transaction fees
- Digital currency exchange capabilities
- QR Code Support
- Mobile first SDK

Core Contracts

Designed to support an ecology of stable value currencies. The first Mento stablecoin, cUSD, tracks the value of the US Dollar.

- Algorithmic reserve-backed stabilization mechanism
- Crypto-asset collateralized
- Native support for multiple stablecoins

Blockchain

Open source permissionless smart contract platform built on decentralized infrastructure.

- Proof-of-Stake based consensus with high throughput, low latency, and zero carbon
- Incentives for serving mobile devices
- On-chain governance

Build on Celo

Visit docs.celo.org to learn more.

celo-protocol.md:

title: The Celo Protocol

description: Introduction to the Celo Protocol's consensus, governance, incentives, and key features.

The Celo Protocol

Introduction to the Celo Protocol's consensus, governance, incentives, and key features.

What is the Celo Protocol?

The Celo blockchain and Celo Core Contracts together comprise the Celo Protocol. This term describes both what services the decentralized Celo network provides to applications and the way in which nodes in the network cooperate to achieve this. This section introduces some of these services.

Consensus and Proof-of-Stake

Celo is a Proof-of-Stake blockchain. In comparison to Proof-of-Work systems like Bitcoin and Ethereum, this eliminates the negative environmental impact and means that users can make transactions that are cheaper, faster, and whose outcome cannot be changed once complete.

The Celo blockchain implements a Byzantine Fault Tolerant (BFT) consensus algorithm in which a well-defined set of validator nodes broadcast signed messages between themselves in a sequence of steps to reach agreement even when up to a third of the total nodes are offline, faulty, or malicious. When a quorum of validators has reached an agreement, that decision is final.

Celo uses a Proof-of-Stake mechanism for selecting the validator set for a fixed period termed an epoch. Anyone can earn rewards by locking CELO and by participating in validator elections and governance proposals. Initially, the number of validators will be capped to one hundred nodes elected by CELO holders. Validators earn additional fixed rewards in Celo Dollars (cUSD) to cover their costs plus margin.

On-Chain Governance

Celo uses an on-chain governance mechanism to manage and upgrade aspects of the protocol that reside in the Celo Core Contracts and for a number of parameters used by the Celo blockchain. This includes operations like upgrading smart contracts, adding new stable currencies, modifying the reserve target asset allocation, and changing how validator elections are decided.

The Governance contract is set as "owner" for all of the Celo Core Contracts. This allows the protocol to carry out agreed governance proposals by executing code in the context of the Governance contract. Proposals are selected for consideration and voted on by CELO holders using a weighted vote based on the same Locked CELO commitment used to vote to elect validators.

Ultralight Synchronization

Celo provides extremely fast, secure synchronization to enable light clients to begin to track the current state of the Celo blockchain ledger almost immediately. This means that even wallet users with high latency, low bandwidth, or high-cost data tariffs can use Celo.

In Ethereum, verifying whether data received from an untrusted full node really does represent the current state of a blockchain requires fetching every block header ever produced to confirm they form a cryptographically secure chain. A consequence of Celo using a BFT consensus algorithm is that it can do that verification by building a chain only of changes in the validator set, not each individual block.

Roadmap: Synchronization performance will be further improved with BLS signature aggregation and succinct zero-knowledge proofs, via zk-SNARKs.

Incentives for Operating Full Nodes

In Ethereum, there are few incentives to run a full node that is not mining. Few nodes serve light clients, and this results in a poor experience for mobile wallets.

Celo introduces a scheme that incentivizes users to operate regular nodes. Light clients pay transaction fees to full nodes. Clients include in every transaction the address of a node that, when the transaction is processed, receives the fee. While a full node provides other services for which they receive no specific fee, it is expected that failing to service these requests will cause clients to seek other full nodes that do, who will then receive fees when they next make a transaction.

Since light clients need not trust full nodes, as they can verify their work, this also provides the 'permissionless on-ramp' for users to receive CELO or Celo Dollars (cUSD) without already holding it that is missing in other Proof-of-Stake networks.

Stable Cryptocurrencies

Celo enables a family of Mento stablecoins that track the value of any asset, including fiat currencies, commodities, and even natural resources. Mento stablecoins supported include the Celo Dollar (cUSD) and the Celo Euro (cEUR), which track the value of the U.S. Dollar and Euro respectively. CELO and a basket of other assets including BTC and ETH serve as the collateral for these stablecoins. These stablecoins are redeemable for CELO, ensuring that transactions can occur quickly, cheaply, and reliably on-chain.

Celo's stability mechanism allows users to create a new cUSD and cEUR by sending CELO to the reserve or burn cUSD and cEUR by redeeming it for their equivalent value in CELO.

This mechanism relies on a series of Oracles, or information feeds from exchanges external to the network, to report the CELO to US Dollar or Euro market rates. To minimize the risk of a run on CELO collateral when these reported values are inaccurate or out-of-date, Celo uses an on-chain constant-product-market-maker model, inspired by the Uniswap system. This mechanism adjusts the redemption price of CELO until either arbitrage occurs (so that the on-chain price dynamically adjusts until the offered rate meets the external rate) or Oracles reset the on-chain price.

The Celo Protocol ensures that there is sufficient CELO collateral to redeem the amount of CELO in circulation through several sources. These include a stability fee levied on Celo Dollar (cUSD) balances, a transfer from epoch rewards, plus the proceeds from the spread when interacting with the on-chain market-maker mechanism.

In addition, a backup reserve of cryptocurrencies is held off-chain. This off-chain reserve is managed to preserve value and minimize volatility by maintaining a diversified portfolio of cryptocurrencies through algorithmic rebalancing trading and periodically "topping-up" the CELO collateral available to ensure it exceeds the amount required to redeem Celo Dollars (cUSD) in circulation. The approved cryptocurrencies, distribution ratios, and rebalancing period are all subject to on-chain governance.

Roadmap: Celo envisages a number of stable currencies tracking different fiat currencies as well as natural resources such as forests. In addition, once bridges between other chains and the Celo blockchain are fully developed, and liquid trading on decentralized exchanges occurs, the rebalancing can be handled transparently on-chain.

Lightweight Identity

Celo offers a lightweight identity layer that allows users of applications including Celo Wallet to identify and securely transact with other users via their contacts' phone numbers. Celo Wallet enables payments directly to users listed in their device's contacts list.

The Attestations contract allows a user to request attestations to their phone number for a small fee. A secure decentralized source of randomness is used to pick a number of validators that will produce and send via SMS signed secret messages that act as attestations of ownership of the phone number. The user then submits these back to the Attestations contract which verifies them and installs a mapping for the phone number to the user's account.

Richer Transactions

Celo provides a number of enhancements to regular transactions as familiar to Ethereum developers.

The Celo native asset has duality as both the native currency and is also an ERC-20 token, simplifying the work of application developers.

Celo assets exist on an independent blockchain, and cannot be accessed through wallets that connect to the Ethereum network. Only use wallets designed to work with the Celo network.

In Celo, transaction fees can be paid in stable cryptocurrencies. A user sending Celo Dollars will be able to pay their transaction fee out of their Celo Dollar (cUSD) balance, so they do not need to hold a separate balance of CELO in order to make transactions. The protocol maintains a list of currencies that can be used to pay for transaction fees. These smart contracts implement an extension of the ERC-20 interface, with additional functions that allow the protocol to debit and credit transaction fees.

The Escrow contract allows users to send payments to other users who can be identified by a phone number but don't yet have an account. These payments are stored in this contract itself and can be either withdrawn by the intended recipient after creating an account and attesting their identity or reclaimed by the sender.

Transfers between two accounts with associated identities support end-to-end encrypted comments. A comment encrypted to the identity's public key is passed when making the transfer and included in an event that can be located on the blockchain ledger.

celo-resources.md:

title: Celo Resources

description: Curated collection of Celo resources for developers, designers, dreamers, and doers.

Resources

Curated collection of Celo resources for developers, designers, dreamers, and doers.

Developer Guide

- Local Development Chain with Protocol Contracts
- Celo for Ethereum Developers

Tech Talks

- Building a mobile-first blockchain platform
- Simplifying blockchain development with the Celo SDK
- Proof of Stake
- Plumo Towards Scalable Interoperable Blockchains Using Ultra Light Validation Sys
- Celo Stability Protocol Walkthrough
- Price Oracles
- Valora
- Engineering in the field
- Keeping Phone Numbers Private
- Celo Blockchain Internals Deepdive
- Developing & deploying your first DApp on Celo
- Building your first DApps on Celo, Part 2
- Feeless Onboarding
- Plumo Ceremony
- Cross Chain Interoperability
- Building your first DApps on Celo
- Porting existing Ethereum DApps onto Celo
- Contract Release
- Leading Valora & Forging Celo User Adoption
- Empowering Celo Developer Experience
- CIP-8
- Hello Celo Q&A
- Building Celo Ecosystem and Community

Pirate Radio Interview Series

- Interview with The Graph
- Interview with Polkadot
- Interview with Skale
- Interview with Aurora
- Interview with Optics Team

Cross-Chain Salon Workshops

- Regen Network Workshop
- Celo Wallet Connect Workshop
- How to build xApps Workshop
- Skynet Workshop
- Observing Optics Workshop
- GoodDollar Workshop
- Building a Subgraph on Celo Workshop
- Optics and the Journey of the Cross-Chain Message Workshop
- Moonbeam: Building the Multi-Chain Universe Workshop
- Skale Workshop

Discussion Panels

- Stablecoins Discussion Panel

The Great Celo Stake Off

- What is The Great Celo Stake Off?
- Celo Mastering the Art of Validating Webinar
- The Great Celo Stake Off Info Session

Celo in the World

- Crypto + Banking: The Next Billion Users
- Future of Finance Spotlight - Celo
- Celo - NOAH19 Berlin
- SFBW19 - Stability of Celo's Stablecoin Protocol - Asa Oines
- [Celo Alfajores Testnet Launch Livestream [High Quality]] (<https://www.youtube.com/watch?v=m4RJGLofox8&list=PLsQbsop73cfHFhdXrfCfg7Z7k6EJSDQG&index=5>)
- 17 Celo: Financial Inclusion and Mobile First Light Clients with Marek Olszewski
- BFTree - Scaling HotStuff to Millions of Validators
- Celo - Banking the unbanked: lessons from the field
- Validator Elections on Celo

How Celo Works

- Taking Crypto Beyond Volatility: Stablecoins
- Celo - Advancing Financial Inclusion with Permissionless Cryptocurrencies
- 17 Celo: Financial Inclusion and Mobile First Light Clients with Marek Olszewski
- zk-SNARKs and Celo's Light Client Protocol
- BFTree - Scaling HotStuff to Millions of Validators
- Celo - Banking the unbanked: lessons from the field
- Validator Elections on Celo

Celo's 4 Tenents

- What if money were beautiful?

- Meet the team working on Celo
- The Circle of Celo
- Sep Kamvar - Curitiba Bus Token, Blockchain for Social Impact Coalition
- Earth Day Live Stream

Celo Prosper Series

- Prosper Series Discover Your Unique Purpose
- Prosper Series Blockchain for Prosperity
- Ignite Your Unique Purpose Zine Making Workshop
- Creating Prosperity Through New Digital Livelihoods
- Prosper Series Technology for a Sustainable Future
- Prosper Series Gifts of the Journey
- The Celo Prosper Series - Meet the Protectors of Prosperity
- Prosper Series How Blockchain Brings Prosperity to Migrant Workers
- Prosper Series Financial Inclusion for International Health
- Prosper Series Emerging Technology, Emerging Markets, & Activism
- Collaborative Economy for Prosperity

Central Bank Digital Currency Series

- Types of Stablecoins
- Stability Protocol
- Stability Research
- Managing Financial Crime Risk on Distributed Ledgers
- Influencing the Velocity of Central Bank Digital Currencies

Kuneco

- The first Kuneco
- Kuneco Community All-Hands

Celo Camp 2020

- Celo Camp 2020: Celo's Values with Sep Kamvar
- Celo Camp 2020: AMA With Marek Olszewski, Co-founder of Celo
- Celo Camp 2020 Investors AMA

Coinbase Earn

- What is the Celo Platform?
- What is CELO?
- What can Celo Dollars do?

Eth Denver 2020

- Evolution of Community
- Celo at #EthDenver
- You Had Me At Celo - Marek Olszewski / Toss a CELO To Your Prestwicher:
- James Prestwich
- Make it Mobile or Else
- Plumo Ceremony
- Celo Bounties at #ETHDenver

#MakeitMobile Hackathon

- Everything You Need to #MakeItMobile on Celo
- How to Build a DApp on Celo
- Accelerating the Movement to Mobile-First DeFi
- How to Port Your Ethereum DApp to Celo

Tech Blog

- Fast and Light
- Cryptocurrency for a Beautiful Planet
- Winning the Celo Stake Off
- The Celo Validator Community: Security Audits and Lessons Learned
- How much could you earn on Celo?
- Celo's Proof of Stake mechanism
- Consensus and Proof of Stake in the Celo protocol
- Build Mobile-First DeFi Apps with the Celo SDK
- BFTree – Scaling HotStuff to Millions of Validators
- Zooming in on the Celo Expansion & Contraction Mechanism
- A Look at the Celo Stability Analysis White Paper
- Diving into the Celo Price Stability Protocol
- A Look at the Celo White Paper
- Arbitrage and Winning a Cryptocurrency Trading Competition at MIT

Design Blog

- A Celebration of Heart
- Celo Camp and The Third Place
- The How of the Celo Coin (Part 2 of 3)
- The What of the Celo Coin (Part 3 of 3)
- What is my celo?
- Three reasons I'm into "crypto"
- Field Notes on Kenya & Mobile Money
- The Celo Alliance for Prosperity grows to 100 members
- Alliance for Prosperity: 75 Members Strong & First Integrations Live
- Meet the Celo Wave 1 Grantees
- We are part of the new Rebel Alliance
- 50 Mission-aligned organizations join the Celo Alliance for Prosperity
- Ang Programa sa Pagboto ng Celo Foundation
- Proposing the Brazilian Real on Celo
- The Celo Foundation Voting Program
- Why we should rename Celo Gold
- CELO Holders: Make Your Voice Heard Through On-chain Governance

General Updates

- Celo Sets Sights On Becoming Fastest EVM Chain Through Collaboration With Mysten Labs
- Meet the cLabs Team at Messari Mainnet!
- Optics is Here: The Future is Multichain
- cLabs Named as a Finalist in Global CBDC Competition
- Aave, Curve, PoolTogether, and Sushi Among Leading Ethereum DeFi Projects Joining Celo on DeFi for the People Collaboration
- The many and varied uses of stablecoins

- Celo and DuniaPay: The Next Chapter of Banking in West Africa
- Field Notes: Undercollateralized DeFi Pilot in Colombia
- Celo at EthCC 4
- Celo and D'CENT: Bringing Greater Access to DeFi dApps in Korea
- Congratulating Valora on an Exciting New Chapter
- NFTs on Celo
- DeFi with Celo and Ethereum
- A Carbon Negative Blockchain? It's Here and it's Celo

celo-stack.md:

title: Architecture

description: Overview of the Celo Stack including it's blockchain, core contracts, and applications.

Architecture

Overview of the Celo Stack, including its blockchain, core contracts, and applications.

Introduction to the Celo Stack

Celo is oriented around providing the simplest possible experience for end-users, who may have no familiarity with cryptocurrencies and may be using low-cost devices with limited connectivity.

A Full-Stack Approach

To achieve this, Celo takes a full-stack approach, where each layer of the stack is designed with the end-user in mind while considering other stakeholders (e.g. operators of nodes in the network) involved in enabling the end-user experience.

Celo Blockchain

An open cryptographic protocol that allows applications to make transactions with and run smart contracts in a secure and decentralized fashion. The Celo blockchain code has shared ancestry with Ethereum and maintains full EVM compatibility for smart contracts. However, it uses a Byzantine Fault Tolerant (BFT) consensus mechanism (Proof-of-Stake) rather than Proof-of-Work and has different block formats, transaction formats, client synchronization protocols, and gas payment and pricing mechanisms.

Celo Core Contracts

A set of smart contracts running on the Celo blockchain that comprise much of the logic of the platform features including ERC-20 stable currencies, identity attestations, proof-of-stake, and governance. These smart contracts are upgradeable and managed by the decentralized governance process.

Applications

Applications for end users built on the Celo Platform. The Celo Wallet app, the first of an ecosystem of applications, allows end-users to manage accounts and make payments securely and simply by taking advantage of the innovations in the Celo Protocol. Applications take the form of external mobile or backend software: they interact with the Celo blockchain to issue transactions and invoke code that forms the Celo Core Contracts' API. Third parties can also deploy custom smart contracts that their own applications can invoke, which in turn can leverage Celo Core Contracts. Applications may use centralized cloud services to provide some of their functionality: in the case of the Celo Wallet, push notifications, and a transaction activity feed.

The Celo blockchain and Celo Core Contracts together comprise the Celo Protocol.

Celo Network Topology

The topology of a Celo network consists of machines running the Celo blockchain software in several distinct configurations:

Validators

Validators gather transactions received from other nodes and execute any associated smart contracts to form new blocks, then participate in a Byzantine Fault Tolerant (BFT) consensus protocol to advance the state of the network. Since BFT protocols can scale only to a few hundred participants and can tolerate at most a third of the participants acting maliciously, a proof-of-stake mechanism admits only a limited set of nodes to this role.

Full Nodes

Most machines running the Celo blockchain software are either not configured to be, or not elected as, validators. Celo nodes do not do "mining" as in Proof-of-Work networks. Their primary role is to serve requests from light clients and forward their transactions, for which they receive the fees associated with those transactions. These payments create a 'permissionless onramp' for individuals in the community to earn currency. Full nodes maintain at least a partial history of the blockchain by transferring new blocks between themselves and can join or leave the network at any time.

Light Clients

Applications including the Celo Wallet will also run on each user's device an instance of the Celo blockchain software operating as a 'light client'. Light clients connect to full nodes to make requests for account and transaction data and to sign and submit new transactions, but they do not receive or retain the full state of the blockchain.

Celo Wallet

The Celo Wallet application is a fully unmanaged wallet that allows users self custody of their funds, using their own keys and accounts. All critical features such as sending transactions and checking balances can be done in a trustless manner using the peer-to-peer light client protocol. However, the wallet does use a few centralized cloud services to improve the user experience where possible, e.g.:

- Google Play Services: to pre-load invitations in the app
- Celo Wallet Notification Service: sends device push notifications when a user receives a payment or requests for payment
- Celo Wallet Blockchain API: provides a GraphQL API to query transactions on the blockchain on a per-account basis, used to implement a user's activity feed.

When end-users download the Celo Wallet from, for example, the Google Play Store, users are trusting both cLabs (or the entity that has made the application available in the Play Store) and Google to deliver a correct binary, and most users would feel that relying on these centralized services to provide this additional functionality is worthwhile.

```
# celo-summary.md:
```

```
---
```

```
title: Celo
```

```
description: Summary of Celo and the value it provides to the community.
```

```
---
```

Celo

Summary of Celo and the value it provides to the community.

```
---
```

Crypto made for mobile

Celo is a mobile-first, carbon-neutral blockchain that makes decentralized financial (DeFi) tools and services accessible to anyone with a mobile phone-bringing the powerful benefits of DeFi to the users of the 6 billion smartphones in circulation today.

With its interoperability, cross-chain compatibility and vision for inclusivity, Celo enables native and non-native digital assets-both private and public)-to circulate freely, at extremely low costs and high

speeds, across devices, carriers, and countries-making money mobile, global and accessible like never before.

Celo's mission is to build a financial system that creates the conditions of prosperity—for everyone.

- 5s Average block time
- Carbon Offset 65.7 Avg Tons Per Month
- Average Network Transaction Fee <\$0.01

Mobile-First

Celo makes sending payments as easy as sending a text, to anyone with an internet connection, anywhere in the world. Celo maps phone numbers to wallet addresses using a novel decentralized address-based identity layer. Mobile participants can earn rewards for securing and maintaining the system.

Multi-Asset

Celo is turning crypto into usable money with a multi-asset system: a utility, governance, and staking asset (CELO) and a growing family of Mento stablecoins named for the currencies they algorithmically follow (e.g., cUSD, cEUR, cREAL).

EVM-Compatible

An EVM-compatible solution capable of reaching billions of global users at scale, Celo's technology supports 1000+ projects from builders, developers, and even artists who everyday launch new applications and issue digital currencies from everywhere in the world usable by anyone in the world with a desktop computer or mobile phone.

Get Started with Celo

Visit celo.org to learn more.

celo-whitepapers.md:

```
---
title: Celo Whitepapers
description: Overview of Celo Whitepapers describing Celo's protocol,
economics, and social impact.
---
```

Whitepapers

Overview of Celo Whitepapers describing Celo's protocol, economics, and social impact.

Protocol

Celo Whitepaper: A Multi-Asset Cryptographic Protocol for Decentralized Social Payments

- Read paper
- 阅读

Plumo: Towards Scalable Interoperable Blockchains Using Ultra Light Validation Systems

- Read paper

Economics

An Analysis of the Stability Characteristics of Celo

- Read paper

Influencing the Velocity of Central Bank Digital Currencies

- Read paper
- Lee el informe

Shaping the Future of Digital Currencies

- Read paper

Social Impact

Future-Proof Aid Policy

- Read paper
- Exec Summary

Delivering Humanitarian COVID Aid using the Celo Platform

- Read paper

developer-onboarding.md:

```
---
title: Celo Develoepr Onboarding
description: Start your journey toward developing decentralized
applications on the Celo platform.
---
```

Developer Onboarding

Start your journey toward developing decentralized applications on the Celo platform.

What to expect

These resources help you learn about the Celo mission, platform architecture, and how you can stay connected with our community. You'll also be introduced to our developer ecosystem and make use of our tools and resources to build and deploy your first decentralized application on the Celo platform.

Prerequisite knowledge

In order to be successful, we recommend having experience developing with Web 2.0 technologies along with a basic understanding of Bitcoin, Ethereum, decentralized finance (DeFi), and blockchain technologies.

If you're unfamiliar with any of these topics, here's a few places to get started:

- Celo.org
- Bitcoin.org
- Ethereum.org
- What is DeFi

Getting Started as a Developer

- Why Build on Celo
- Celo Decentralized Application Examples
- Celo Architecture
- Celo Tech Talks Simplifying blockchain development with the Celo SDK
- Celo Developer Tools
- A Look at the Celo Stability Analysis White Paper
- Celo Whitepapers

Set up Local Environment

- Using Windows
- Using Mac
- Using Replit

Deploy a dApp with Celo

- Deploy with Remix
- Deploy with Hardhat

Build Celo Applications

- Build Mobile-First DeFi Apps with the Celo SDK
- Developing & deploying your first DApp on Celo
- Building your first DApps on Celo, Part 2

Create a Web dApp with Celo

- Interact with Celo core contracts

Stay Connected

- Discord
- Forum
- Telegram

developer-tools.md:

title: Celo Developer Tools
description: Overview of Celo tools and the value they provide to developers.

Developer Tools Overview

Overview of Celo tools and the value they provide to developers.

```
import PageRef from '@components/PageRef'  
import Tabs from '@theme/Tabs';  
import TabItem from '@theme/TabItem';
```

:::tip

Consider using Dependabot to help keep project dependencies up to date. The following developer tools are being actively developed and keeping your dependencies up-to-date will help your projects stay functional and secure.

:::

SDKs

viem

viem is a lightweight javascript library for interacting with EVM chains. It supports celo specific features. If you're building with react, consider wagmi a viem wrapper library that speeds up your development time.

Ethers.JS Wrapper

A minimal wrapper for ethers to create Celo Easy Fee transactions. If paying for gas with tokens is not important for your use case, then the standard ethers.js works perfectly well.

```
<PageRef url="https://github.com/jmrossy/celo-ethers-wrapper"  
pageName="Ethers.JS Wrapper" />
```

WalletConnect

WalletConnect is a standard across EVM compatible blockchains to connect wallets to dapps. It allows developers to build connections between wallets and dapps on the same desktop or mobile device, or between desktop dapps and mobile wallets.

<PageRef url="wallet-connect" pageName="WalletConnect" />

ContractKit

ContractKit is a library to help developers and validators to interact with the Celo blockchain and is well suited to developers looking for an easy way to integrate Celo Smart Contracts within their applications.

<PageRef url="/developer-guide/contractkit" pageName="ContractKit" />

Celo CLI

The Command Line Interface allows users to interact with the Celo Protocol smart contracts.

<PageRef url="/cli" pageName="Celo CLI" />

Celo SDK Reference Docs

You can find the reference documentation for all of the Celo SDK packages found in the celo monorepo here. The SDK packages consist of:

- Base
- Connect
- ContractKit
- Explorer
- Governance
- Identity
- Keystores
- Network Utils
- Transactions Uri
- Utils
- Wallet-base
- Wallet-HSM
- Wallet-HSM-AWS
- Wallet-HSM-Azure
- Wallet-ledger
- Wallet-local
- Wallet-remote
- Wallet-rpc
- Wallet-walletconnect

<PageRef url="https://docs.celo.org/learn/developer-tools#celo-sdk-reference-docs" pageName="Celo SDK Reference Docs" />

Infrastructure

BlockScout

BlockScout is a cLabs hosted GUI block explorer and API endpoints. It allows you to look up information about the Celo blockchain including average block time, total transactions, and additional transaction details. You may also view details of your own custom smart contracts or existing DeFi contracts to view how value is moving between accounts and on-chain network events.

<PageRef url="https://explorer.celo.org/" pageName="BlockScout" />

ODIS

ODIS (Oblivious decentralized identity service) is a lightweight identity layer that makes it easy to send cryptocurrency to a phone number

<PageRef url="/protocol/identity/odis" pageName="ODIS" />

The Graph

The Graph is an indexing protocol for querying networks like Celo, Ethereum and IPFS. Anyone can build and publish open APIs, called subgraphs, making data easily accessible. Many blockchain data querying tools like Dapplooker leverage the Graph.

DappLooker

DappLooker allows you to easily analyze & visualize your Celo smart contracts & subgraph data in various formats and gather it into dashboards from multiple sources. Analyze your data with intuitive Visual SQL which writes queries for you. Share the story your data tells with your team or with your community. Share dashboard insights via URL wherever you need to make your organization truly data driven.

<PageRef url="https://dapplooker.com/integration/celo" pageName="DappLooker" />

Stats.celo.org

Stats.celo.org allows you to check network activity and health.

<PageRef url="http://stats.celo.org" pageName="stats.celo.org" />

SubQuery

SubQuery is a leading blockchain data indexer that provides developers with fast, flexible, universal, open source and decentralised APIs for CELO. One of SubQuery's competitive advantages is the ability to aggregate data not only within a chain but across multiple blockchains all within a single project.

<PageRef url="https://subquery.network/" pageName="SubQuery" />

Hosted Nodes

RPC Endpoint Services

Forno

Forno is a cLabs hosted node service for interacting with the Celo network. This allows you to connect to the Celo Blockchain without having to run your own node.

Forno has HTTP and WebSocket endpoints that you can use to query current Celo data or post transactions that you would like to broadcast to the network. The service runs full nodes in non-archive mode, so you can query the current state of the blockchain, but cannot access the historic state.

Forno can be used as an HTTP Provider with ContractKit.

<PageRef url="/network/node/forno" pageName="Forno" />

Infura

RPC end point provider that supports Celo and several other EVM L1s. Infura's node infrastructure powers some of the biggest projects today.

<PageRef url="https://docs.infura.io/networks/celo" pageName="Infura" />

Infura is L2 ready.

Ankr

Featuring open access to a Public RPC API layer, Ankr Protocol provides reliable, load balanced access to node clusters from anywhere in the world.

<PageRef url="https://www.ankr.com/rpc/celo/" pageName="Ankr" />

Quicknode

Quicknode is an enterprise grade node service with a dashboard, metrics, security controls, customer support and no rate limits (pay-as-you-go).

Quicknode is L2 ready.

<PageRef url="https://www.quicknode.com/chains/celo" pageName="Quicknode" />

All That Node

All That Node supports public and private RPC nodes for mainnet, alfajores and baklava networks. They offer free private RPC nodes up to 10,000 requests/day and you can upgrade your plan as needed. You can also claim alfajores funds from the faucet in the site without signing up or any time-consuming auth.

<PageRef url="https://www.allthatnode.com/celo.dsrv" pageName="All That Node" />

Lava

Lava is a multi-chain RPC provider. They also provide managed and decentralized options for your applications.

<PageRef url="https://lavanet.xyz" pageName="Lava" />

Celo Wallets

Celo Wallets are tools that create accounts, manage keys, and help users transact on the Celo network.

<PageRef url="../wallet/" pageName="Celo Wallets" />

key-concepts.md:

title: Key Concepts

id: key-concepts

slug: /key-concepts

description: Celo believes in a future where everyone can prosper.

Key Concepts

Crypto made for mobile.

import YouTube from '@components/YouTube';

What is the Celo Platform?

With its interoperability, cross-chain compatibility and vision for inclusivity, Celo enables native and non-native digital assets-both private and public)-to circulate freely, at extremely low costs and high speeds, across devices, carriers, and countries-making money mobile, global and accessible like never before.

<YouTube videoId="4a70pVEcRw4"/>

What is CELO?

CELO is a platform-native reserve and governance asset, serving as the primary asset in the stability mechanism (Mento) that supports stable digital currencies like cUSD and cEUR. CELO's primary function is stabilization, ensuring the healthy velocity of assets that globally circulate and transact on Celo. CELO also serves as the key governance mechanism for the Celo Platform. Owning and using CELO supports the growth and development of the Celo Platform by enabling each CELO owner to vote on proposals that direct how the core technology operates today and in the future.

<YouTube videoId="PLodjpBer4M"/>

What can Celo Dollars Do?

Named for the currencies they follow, Celo Dollars (cUSD) and Celo Euros (cEUR) are Mento stablecoins that allow you to share value faster, cheaper, and more easily on your mobile phone. Mento stablecoins instantly unlock access for everyday uses like low-cost remittances and cross-border payments, global distribution of charitable aid, effortlessly paying online, or transferring value within exchanges, particularly in markets subject to currency volatility.

<YouTube videoId="bu4P6jZKXgA"/>

platform-native-stablecoins-summary.md:

title: Platform-Native Stablecoins (cUSD, cEUR)
description: Summary of Celo Platform-Native Stablecoins and the value they provide to the community.

Platform-Native Stablecoins

Summary of Celo Platform-Native Stablecoins and the value they provide to the community.

Stable with Celo

Named for the currencies they follow, Celo Dollars (cUSD), Celo Euros (cEUR) and Celo Reals (cREAL) are Mento stablecoins that allow you to share value faster, cheaper, and more easily on your mobile phone. Mento stablecoins instantly unlock access for everyday uses like low-cost remittances and cross-border payments, global distribution of charitable aid, effortlessly paying online, or transferring value within exchanges, particularly in markets subject to currency volatility.

- Average Network Transaction Fee <\$.01
- Algorithmic stabilization mechanism
- Publicly verifiable stability
- 5 second block times
- Can be used to pay gas fees

Pay instantly

With payment applications and mobile solutions built on Celo users can easily transfer stable value globally using only a mobile number while retailers can accept Celo Dollars (cUSD) and Celo Euros (cEUR) as stable forms of payment online or and in-person,

Share directly

Designed to enable a future without high transaction fees or expensive third party intermediaries for peer to peer payments, payments applications built on Celo allows its users to securely and instantly send and receive money locally or internationally without the burden of high transaction fees or third parties. With Celo, users can conveniently share directly on their mobile phones for as little as \$0.01 to virtually anyone around the world.

Borrow easily

Many individuals don't have a way of establishing assets or credit history on which to receive loans. With Celo, companies can reimagine the possibilities of financing by enabling anyone with even a basic feature phone to access to save, send peer-to-peer payments, and even obtain loans.

Get Started with Celo

- Read Stability White Paper
- Celo Dollars (cUSD)
- Celo Euros (cEUR)
- Celo Real (cREAL)

topology-of-a-celo-network.md:

title: Celo Network Topology

description: Introduction to the Celo Network's topology including validators, full nodes, light clients, and wallets.

Topology of a Celo Network

Introduction to the Celo Network's topology including validators, full nodes, light clients, and wallets.

What is the Celo Network Topology?

The topology of a Celo network consists of machines running the Celo blockchain software in several distinct configurations:

Validators

Validators gather transactions received from other nodes and execute any associated smart contracts to form new blocks, then participate in a Byzantine Fault Tolerant (BFT) consensus protocol to advance the state of the network. Since BFT protocols can scale only to a few hundred

participants and can tolerate at most a third of the participants acting maliciously, a proof-of-stake mechanism admits only a limited set of nodes to this role.

Full Nodes

Most machines running the Celo blockchain software are either not configured to be, or not elected as, validators. Celo nodes do not do "mining" as in Proof-of-Work networks. Their primary role is to serve requests from light clients and forward their transactions, for which they receive the fees associated with those transactions. These payments create a 'permissionless onramp' for individuals in the community to earn currency. Full nodes maintain at least a partial history of the blockchain by transferring new blocks between themselves and can join or leave the network at any time.

Light Clients

Applications including the Celo Wallet will also run on each user's device an instance of the Celo blockchain software operating as a 'light client'. Light clients connect to full nodes to make requests for account and transaction data and to sign and submit new transactions, but they do not receive or retain the full state of the blockchain.

Celo Wallet

The Celo Wallet application is a fully unmanaged wallet that allows users to self custody their funds using their own keys and accounts. All critical features such as sending transactions and checking balances can be done in a trustless manner using the peer-to-peer light client protocol. However, the wallet does use a few centralized cloud services to improve the user experience where possible, e.g.:

- Google Play Services: to pre-load invitations in the app
- Celo Wallet Notification Service: sends device push notifications when a user receives a payment or requests for payment
- Celo Wallet Blockchain API: provides a GraphQL API to query transactions on the blockchain on a per-account basis, used to implement a users' activity feed.

When end-users download the Celo Wallet from, for example, the Google Play Store, users are trusting both cLabs (or the entity that has made the application available in the Play Store) and Google to deliver a correct binary, and most users would feel that relying on these centralized services to provide this additional functionality is worthwhile.

```
# valora-summary.md:
```

```
---
```

```
title: Valora
```

```
description: Summary of Valora and the value it provides to the community.
```

Valora

Summary of Valora and the value it provides to the community.

Use crypto like everyday money

Valora is a mobile wallet focused on making global peer-to-peer payments simple and accessible to anyone. It supports the Celo Identity Protocol which allows users to verify their phone number and send payments to their contacts.

Valora syncs to your contacts list, which means you can make payments, send international remittances or simply split the bill with someone in your contact list - no matter where they are in the world.

Valora's mission is to be the most reliable way to send and receive value worldwide

- Available on mobile for all iOS and Android users
- Confirms transactions in less than 5 seconds on average
- Transaction fees as low as 1/100th of the current cost

Send money like a text message

Valora makes global transfers easy. Send money to your friends and family back home, as simply as sending a text message.

Near-zero fee for transactions on the Celo network

Valora gives you control of your value by making your phone your wallet. Send transactions directly from your mobile phone to your friends at a fraction of the current cost.

Spend on the things you love

Use Valora to buy gift cards for major retailers, services, and subscriptions around the world.

Use crypto like everyday money

Send, pay, and spend cryptocurrency like everyday money - all from the palm of your hand.

- Send funds across borders without any intermediaries.
- Receive global transactions in seconds with Valora.
- Pay online or in-store where Celo is accepted.

Get Started with Valora

Visit valoraapp.com to learn more.

```
# wallet-connect.md:
```

```
---
```

```
title: WalletConnect
```

```
---
```

```
WalletConnect
```

WalletConnect is a standard across EVM compatible blockchains to connect wallets to dapps. It allows developers to build connections between wallets and dapps on the same desktop or mobile device, or between desktop dapps and mobile wallets.

```
# why-celo.md:
```

```
---
```

```
title: What is Celo?
```

```
description: Celo's mission is to build a financial system that creates the conditions for prosperity—for everyone.
```

```
---
```

```
import YouTube from '@site/src/components/YouTube';
```

```
import PageRef from '@site/src/components/PageRef';
```

Celo's mission is to build a financial system that creates the conditions for prosperity—for everyone.

```
---
```

```
Cryptocurrency for a beautiful planet
```

Celo was designed to enable a new universe of financial solutions accessible for mobile users, creating a global financial ecosystem where an end-user can onboard into the Celo ecosystem with just a mobile number. Here are few of the key features of Celo:

- Layer-1 protocol
- EVM compatible
- Proof-of-stake
- Carbon negative
- Mobile-first identity
- Ultra-light clients
- Localized stablecoins (cUSD, cEUR, cREAL)
- Gas payable in multiple currencies

```
What is the Celo Platform?
```

Celo makes sending payments as easy as sending a text, to anyone with an internet connection, anywhere in the world. Celo maps phone numbers to wallet addresses using a novel decentralized address-based identity

layer. Mobile participants can earn rewards for securing and maintaining the system.

<YouTube videoId="4a70pVEcRw4"/>

What is CELO?


CELO is the platform-native asset that supports the growth and development of the Celo blockchain and ecosystem. CELO holders can earn rewards, stake with validators, and vote on proposals that shape the future of Celo.

<YouTube videoId="mkpTmbkRv4A"/>

What can Celo Dollars do?

Named for the currencies they follow, Celo Dollars (cUSD), Celo Euros (cEUR) and Celo Reals (cREAL) are Mento stablecoins that allow you to share value faster, cheaper, and more easily on your mobile phone. Mento stablecoins instantly unlock access for everyday uses like low-cost remittances and cross-border payments, global distribution of charitable aid, effortlessly paying online, or transferring value within exchanges, particularly in markets subject to currency volatility.

<YouTube videoId="n1klJcjTnp8"/>

:::tip Learn more 

Read Celo: Building a Regenerative Economy, Celo Spotlight, and the Celo 2021 Annual Report for an in-depth look at Celo and how it's creating the conditions of prosperity for everyone.

:::

index.md:

title: Network Details

description: How to choose a Celo network based on your needs and objectives.

Overview of Celo Mainnet, Alfajores L2 Testnet and Baklava Testnet.

Celo Mainnet

Name	Value
-----	-----
-----	-----

```

-----
--- |
| Network Name          | Celo Mainnet
|
| Description           | The production Celo network
|
| Chain ID              | 42220
|
| Currency Symbol       | CELO
|
| RPC Nodes             | https://docs.celo.org/learn/developer-
tools#hosted-nodes
|
| RPC Endpoint (best effort) | https://forno.celo.org <br/> Note to
developers: Forno is rate limited, as your usage increases consider
options that can provide the desired level of support (SLA). |
| Block Explorers       |
<ul><li>https://explorer.celo.org</li><li>https://celoscan.io</li></ul>
|
| Network Status        | https://stats.celo.org
|
| Validator Explorer    | https://validators.celo.org
|

```

Celo Alfajores L2

```

| Name                  | Value
|
| -----
| -----
| -----
| -----
| -----
| -----
| -----
| -----
| -----
| -----
|
| Network Name          | Celo Alfajores
|
| Description           | The Developer Testnet network
|
| Currency Symbol       | CELO
|
| Chain ID              | 44787
|
| RPC Endpoint (best effort) | https://alfajores-forno.celo-testnet.org
|
| RPC Nodes             | <ul><li>Ethereum JSON-RPC endpoint:
https://alfajores-forno.celo-testnet.org (op-geth kind)</li><li>OP RPC
endpoint: https://op.alfajores.celo-testnet.org/, (op-node kind)OP RPC
endpoint: https://op.alfajores.celo-testnet.org/ (op-node
kind)</li><li>Infura: https://www.infura.io/networks/celo</li></ul> |
| Block Explorer        | <ul><li>https://celo-
alfajores.blockscout.com/
</li><li>https://alfajores.celoscan.io</li></ul>
|

```

Bridge Link	<p> https://superbridge.app/celo-testnet </p><p>Note: Ensure you enable Testnet in settings </p>
Network Status	https://alfajores-celostats.celo-testnet.org
Faucet Link	<p> https://faucet.celo.org </p><p>For large Faucet requests you can apply here.</p>

:::info

Your use of the Alfajores Testnet is subject to the Alfajores Testnet Disclaimer.

:::

Baklava

Name	Value
-----	-----
Network Name	Baklava
Description	The Node Operator Testnet network
Currency Symbol	CELO
RPC Endpoint (best effort)	https://baklava-forno.celo-testnet.org
RPC Nodes	https://docs.celo.org/network/node/forno#baklava-testnet
Block Explorer	https://baklava-blockscout.celo-testnet.org
Network Status	https://baklava-celostats.celo-testnet.org
Faucet Request Form	https://forms.gle/JTYkMAJWTAUQplsv9

The Baklava Testnet is a non-production Testnet for the Validator community.

It serves several purposes:

- Operational excellence: Build familiarity with the processes used on Mainnet, and to verify the security and stability of your infrastructure with the new software.
- Detecting vulnerabilities: Discover bugs in new software releases before they reach Mainnet.

- Testing ground: Experiment with new infrastructure configurations in a low-risk environment.

:::warning

The Baklava Testnet is designed for testing and experimentation by developers. Its tokens hold no real world economic value. The testnet software will be upgraded and the entirety of its data reset on a regular basis. This will erase your accounts, their balance and your transaction history. The testnet software will be upgraded on a regular basis. You may encounter bugs and limitations with the software and documentation.

:::

:::info

Your use of the Baklava Testnet is subject to the Baklava Testnet Disclaimer.

:::

disclaimer.md:

title: Celo Alfajores Testnet Disclaimer

description: Important considerations, warnings, and legal regulations for users of the Alfajores Testnet.

Alfajores Testnet Disclaimer

Important considerations, warnings, and legal regulations for users of the Alfajores Testnet.

Terms and Conditions

By using, and contributing to, the Celo Alfajores Testnet, you \ (the User\) agree to these terms and acknowledge and agree that the Celo protocol and platform is in development and that use of the Alfajores Testnet is entirely at the User's sole risk. You also agree to adhere to the Celo Community Code of Conduct.

The User acknowledges and agrees that they have an adequate understanding of the risks associated with use of the Celo Alfajores Testnet and that all information and materials published, distributed or otherwise made available on the Celo Alfajores Testnet are provided for non-commercial, personal use only. This includes test CELO, test Celo Dollars, and any other stable value asset test units, which have no economic value and are provided only for the purpose of testing on the Celo Alfajores Testnet.

All content provided on the Celo Alfajores Testnet is subject to the License included and is provided on an 'AS IS' and 'AS AVAILABLE' basis, without any representations or warranties of any kind. All implied terms are excluded to the fullest extent permitted by law. No party involved in, or having contributed to the development of, the Celo Alfajores Testnet including any of their affiliates, directors, employees, contractors, service providers or agents \the Parties Involved\ accepts any responsibility or liability to the User or any third parties in relation to any materials or information accessed or downloaded via the Celo Alfajores Testnet. The User acknowledges and agrees that the Parties Involved are not responsible for any damage to the User's computer systems, loss of data, or any other loss or damage resulting \directly or indirectly\ from use of the Celo Alfajores Testnet.

To the fullest extent permitted by law, in no event shall the Parties Involved have any liability whatsoever to any person for any direct or indirect loss, liability, cost, claim, expense or damage of any kind, whether in contract or in tort, including negligence or otherwise, arising out of or related to the use of all or part of the Celo Alfajores Testnet.

The Celo Labs software is not subject to the EAR based on Section 734.7 of the U.S. Export Administration Regulations \("EAR", 15 CFR Parts 730-774\) and Section 742.15\ (b\) of the EAR, which applies to software containing or designed for use with encryption software that is publicly available as open-source. However, products developed using the Celo Labs software may be subject to the EAR or local laws/regulations. The User is responsible for compliance with U.S. and local country export/import laws and regulations.

Celo Labs software may not be exported/reexported, either directly or indirectly, to any destination subject to U.S. embargoes or trade sanctions unless formally authorized by the U.S. Government. The embargoed destinations are subject to change and the scope of what is included in the embargo is specific to each embargoed country. For the most current information on U.S. embargoed and sanctioned countries, see the Treasury Department regulations.

faucet.md:

title: Celo Testnet Funds

description: How to fund your Celo wallet account with testnet funds.

Celo Wallet Testnet Funds

How to fund your Celo wallet account with testnet funds.

Getting Started

To start experimenting with the Alfajores Testnet, you will first need to get a funded account.

:::warning

Alfajores Testnet accounts hold no real world economic value. The testnet's data may be reset on a regular basis. This will erase your accounts, their balance and your transaction history.

:::

Getting an account is really being given or generating a public-private keypair. This gives you control of balances accessible with the address corresponding to that key. For CELLO, this is a native balance stored at the account whose address matches your key. For Celo Dollars, an ERC-20 token, the Mento Stablecoin smart contract maintains in its storage a mapping of the balance of each address.

Using an existing EVM Address

You can reuse the same address and private key you use on other EVM networks as long as you are using a wallet that has support for Celo Networks like Alfajores. Open Config for Alfajores

Creating an empty account with the Celo Client

You can also use the Celo Blockchain client to create and manage account keypairs.

Prerequisites

- You have Docker installed. If you don't have it already, follow the instructions here: [Get Started with Docker](#). It will involve creating or signing in with a Docker account, downloading a desktop app, and then launching the app to be able to use the Docker CLI. If you are running on a Linux server, follow the instructions for your distro here. You may be required to run Docker with sudo depending on your installation environment.

Create and cd into the directory where you want to store the keypair. You can name this whatever you'd like, but here's a default you can use:

```
bash
mkdir celo-data-dir $ cd celo-data-dir
```

Create an account by running this command:

```
bash
docker run -v pwd:/root/.celo --rm -it us.gcr.io/celo-org/geth:alfajores
account new
```


It will prompt you for a passphrase, ask you to confirm it, and then will output your account address:

Address: <YOUR-ACCOUNT-ADDRESS>

This creates a keypair and stores it. Save this address to an environment variable, so that you can reference it later:

```
bash
export CELOACCOUNTADDRESS=<YOUR-ACCOUNT-ADDRESS>
```

Add funds to an existing account with the Faucet

The Alfajores Testnet Faucet is an easy way to get more funds deposited to an account, however it was created.

Visit faucet.celo.org, and enter your account address. If you are using the Celo Wallet, you can find your account address in the Settings page. Complete the Captcha, and click 'Add Funds'.

Each time you complete a faucet request, your account is funded with an additional CELO Tokens

```
# index.md:
```

```
---
title: Celo's Alfajores Testnet
description: Collection of resources to get started with Celo Alfajores
Testnet (Celo's Developer Testnet).
---
```

Alfajores Testnet

Collection of resources to get started with Celo Alfajores Testnet (Celo's Developer Testnet).

```
---
```

What is the Alfajores Testnet?

The Alfajores Testnet is a Celo test network for developers building on the Celo platform. You can use it to try out the Celo Wallet or the Celo CLI \ (by sending transfers to yourself or other users of the testnet\). You can also assist in running the network by operating a full node on your machine \ (or on a cloud or hosting provider\).

```
:::info
```

The Baklava Testnet is focused on building operational experience and best practices for node operators.

```
:::
```

:::tip

Please refer to Key Concepts for background on blockchains and an explanation of terms used in the section.

:::

:::warning

The Alfajores Testnet is designed for testing and experimentation by developers. Its tokens hold no real world economic value. The testnet software will be upgraded and the entirety of its data reset on a regular basis. This will erase your accounts, their balance and your transaction history. The testnet software will be upgraded on a regular basis. You may encounter bugs and limitations with the software and documentation.

:::

Please help the community to improve Celo by asking questions on the Forum!

run-full-node.md:

title: Run an Alfajores Full Node on Celo
description: How to run a full node on the Alfajores Network using a prebuilt Docker image.

Run an Alfajores Full Node

How to run a full node on the Alfajores Network using a prebuilt Docker image.

What is a Full Node?

Full nodes play a special purpose in the Celo ecosystem, acting as a bridge between the mobile wallets \(\running as light clients\) and the validator nodes.

:::tip

If you'd prefer a simple, one click hosted setup for running a node on one of the major cloud providers (AWS and GCP), checkout our hosted nodes documentation.

:::

:::info

If you would like to keep up-to-date with all the news happening in the Celo community, including validation, node operation and governance, please sign up to our Celo Signal mailing list.

You can add the Celo Signal public calendar as well which has relevant dates.

:::

Prerequisites

- You have Docker installed. If you don't have it already, follow the instructions here: [Get Started with Docker](#). It will involve creating or signing in with a Docker account, downloading a desktop app, and then launching the app to be able to use the Docker CLI. If you are running on a Linux server, follow the instructions for your distro here. You may be required to run Docker with sudo depending on your installation environment.

:::info

Code you'll see on this page is bash commands and their output.

When you see text in angle brackets `<>`, replace them and the text inside with your own value of what it refers to. Don't include the `<>` in the command.

:::

Celo Networks

First we are going to setup the environment variables required for Alfajores network. Run:

```
bash
export CELOIMAGE=us.gcr.io/celo-org/geth:alfajores
```

Pull the Celo Docker image

We're going to use a Docker image containing the Celo node software in this tutorial.

If you are re-running these instructions, the Celo Docker image may have been updated, and it's important to get the latest version.

```
bash
docker pull $CELOIMAGE
```

Set up a data directory

First, create the directory that will store your node's configuration and its copy of the blockchain. This directory can be named anything you'd

like, but here's a default you can use. The commands below create a directory and then navigate into it. The rest of the steps assume you are running the commands from inside this directory.

```
bash
mkdir celo-data-dir
cd celo-data-dir
```

Create an account and get its address

In this step, you'll create an account on the network. If you've already done this and have an account address, you can skip this and move on to configuring your node.

Run the command to create a new account:

```
bash
docker run -v $PWD:/root/.celo --rm -it $CELOIMAGE account new
```

It will prompt you for a passphrase, ask you to confirm it, and then will output your account address: Public address of the key: <YOUR-ACCOUNT-ADDRESS>

Save this address to an environment variables, so that you can reference it below (don't include the braces):

```
bash
export CELOACCOUNTADDRESS=<YOUR-ACCOUNT-ADDRESS>
```

:::info

This environment variable will only persist while you have this terminal window open. If you want this environment variable to be available in the future, you can add it to your `/.bashprofile`

:::

Start the node

This command specifies the settings needed to run the node, and gets it started.

```
bash
docker run --name celo-fullnode -d --restart unless-stopped --stop-
timeout 300 -p 127.0.0.1:8545:8545 -p 127.0.0.1:8546:8546 -p 30303:30303
-p 30303:30303/udp -v $PWD:/root/.celo $CELOIMAGE --verbosity 3 --
syncmode full --http --http.addr 0.0.0.0 --http.api
eth,net,web3,debug,admin,personal --light.serve 90 --light.maxpeers 1000
--maxpeers 1100 --etherbase $CELOACCOUNTADDRESS --alfajores --datadir
/root/.celo
```

You'll start seeing some output. After a few minutes, you should see lines that look like this. This means your node has started syncing with the network and is receiving blocks.

```
text
INFO [07-16|14:04:24.924] Imported new chain segment
blocks=139 txs=319 mgas=61.987 elapsed=8.085s mgasps=7.666 number=406
hash=9acf16...4fddc8 age=6h58m44s cache=1.51mB
INFO [07-16|14:04:32.928] Imported new chain segment
blocks=303 txs=179 mgas=21.837 elapsed=8.004s mgasps=2.728 number=709
hash=8de06a...77bb92 age=6h33m37s cache=1.77mB
INFO [07-16|14:04:40.918] Imported new chain segment
blocks=411 txs=0 mgas=0.000 elapsed=8.023s mgasps=0.000 number=1120
hash=3db22a...9fa95a age=5h59m30s cache=1.92mB
INFO [07-16|14:04:48.941] Imported new chain segment
blocks=335 txs=0 mgas=0.000 elapsed=8.023s mgasps=0.000 number=1455
hash=7eb3f8...32ebf0 age=5h31m43s cache=2.09mB
INFO [07-16|14:04:56.944] Imported new chain segment
blocks=472 txs=0 mgas=0.000 elapsed=8.003s mgasps=0.000 number=1927
hash=4f1010...1414c1 age=4h52m31s cache=2.34mB
```

You will have fully synced with the network once you have pulled the latest block number, which you can lookup by visiting at the Alfajores Network Stats or [Alfajores Block Explorer]] (<https://alfajores-blockscout.celo-testnet.org/>) pages.

:::danger

Security: The command line above includes the parameter `--http.addr 0.0.0.0` which makes the Celo Blockchain software listen for incoming RPC requests on all network adaptors. Exercise extreme caution in doing this when running outside Docker, as it means that any unlocked accounts and their funds may be accessed from other machines on the Internet. In the context of running a Docker container on your local machine, this together with the `docker -p` flags allows you to make RPC calls from outside the container, i.e from your local host, but not from outside your machine. Read more about Docker Networking [here](#).

:::

Command Line Interface

Once the full node is running, it can serve the Command Line Interface tool `celocli`. For example:

```
bash
$ npm install -g @celo/celocli
...
$ celocli node:synced
true
$ celocli account:new
...
```

disclaimer.md:

title: Celo Baklava Testnet Disclaimer

description: Important considerations, warnings, and legal regulations for users of the Baklava Testnet.

Baklava Testnet Disclaimer

Important considerations, warnings, and legal regulations for users of the Baklava Testnet.

Terms and Conditions

By using, and contributing to, the Celo Baklava Testnet, you \the User\ agree to these terms and acknowledge and agree that the Celo protocol and platform is in development and that use of the Baklava Testnet is entirely at the User's sole risk. You also agree to adhere to the Celo Code of Conduct.

The User acknowledges and agrees that they have an adequate understanding of the risks associated with use of the Celo Baklava Testnet and that all information and materials published, distributed or otherwise made available on the Celo Baklava Testnet are provided for non-commercial, personal use only. This includes test CELO, test Celo Dollars, and any other stable value asset test units, which have no economic value and are provided only for the purpose of testing on the Celo Baklava Testnet.

All content provided on the Celo Baklava Testnet is subject to the License included and is provided on an 'AS IS' and 'AS AVAILABLE' basis, without any representations or warranties of any kind. All implied terms are excluded to the fullest extent permitted by law. No party involved in, or having contributed to the development of, the Celo Baklava Testnet including any of their affiliates, directors, employees, contractors, service providers or agents \the Parties Involved\ accepts any responsibility or liability to the User or any third parties in relation to any materials or information accessed or downloaded via the Celo Baklava Testnet. The User acknowledges and agrees that the Parties Involved are not responsible for any damage to the User's computer systems, loss of data, or any other loss or damage resulting \directly or indirectly\ from use of the Celo Baklava Testnet.

To the fullest extent permitted by law, in no event shall the Parties Involved have any liability whatsoever to any person for any direct or indirect loss, liability, cost, claim, expense or damage of any kind, whether in contract or in tort, including negligence or otherwise, arising out of or related to the use of all or part of the Celo Baklava Testnet.

The Celo Labs software is not subject to the EAR based on Section 734.7 of the U.S. Export Administration Regulations \("EAR", 15 CFR Parts 730-774\) and Section 742.15\(\b\) of the EAR, which applies to software containing or designed for use with encryption software that is publicly available as open-source. However, products developed using the Celo Labs software may be subject to the EAR or local laws/regulations. The User is responsible for compliance with U.S. and local country export/import laws and regulations.

Celo Labs software may not be exported/reexported, either directly or indirectly, to any destination subject to U.S. embargoes or trade sanctions unless formally authorized by the U.S. Government. The embargoed destinations are subject to change and the scope of what is included in the embargo is specific to each embargoed country. For the most current information on U.S. embargoed and sanctioned countries, see the Treasury Department regulations.

index.md:

title: Celo Baklava Testnet

description: Collection of resources to get started with Celo Baklava Testnet (Celo's Node Operator Testnet).

Baklava Testnet

Collection of resources to get started with Celo Baklava Testnet (Celo's Node Operator Testnet).

What is the Baklava Testnet?

The Baklava Testnet is a non-production Testnet for the Validator community.

It serves several purposes:

- Operational excellence: Build familiarity with the processes used on Mainnet, and to verify the security and stability of your infrastructure with the new software.
- Detecting vulnerabilities: Discover bugs in new software releases before they reach Mainnet.
- Testing ground: Experiment with new infrastructure configurations in a low-risk environment.

:::warning

The Baklava Testnet is designed for testing and experimentation by developers. Its tokens hold no real world economic value. The testnet software will be upgraded and the entirety of its data reset on a regular

basis. This will erase your accounts, their balance and your transaction history. The testnet software will be upgraded on a regular basis. You may encounter bugs and limitations with the software and documentation.

:::

Please help the community to improve Celo by asking questions on Discord the Forum!

:::info

Useful links

- Baklava Faucet Request Form - to request faucettted funds to become a Validator on the Baklava network.
- Baklava Network Status - to check the current availability of the testnet
- Baklava Network Block Explorer - explore the history of the blockchain and view transaction details

run-full-node.md:

title: Run a Baklava Full Node

description: How to get a full node running on the Baklava Network using a prebuilt Docker image.

Run a Baklava Full Node

How to get a full node running on the Baklava Network using a prebuilt Docker image.

What is a Baklava Full Node?

Full nodes play a special purpose in the Celo ecosystem, acting as a bridge between the mobile wallets \(\running as light clients\) and the validator nodes.

:::info

If you would like to keep up-to-date with all the news happening in the Celo community, including validation, node operation and governance, please sign up to our Celo Signal mailing list here.

You can add the Celo Signal public calendar as well which has relevant dates.

:::

:::info

If you are transitioning from the Baklava network prior to the June 24 reset, you will need to start with a fresh chain database. You can either shut down your existing node, delete the celo folder, and continue by following the guide below or create a new node following these directions.

Key differences are:

- New network ID is 62320
- A new image has been pushed to us.gcr.io/celo-org/gets:baklava
- A new genesis block and bootnode enode are included in the Docker image
- ReleaseGold contracts are available for all previously faucettted addresses here

:::

Prerequisites

- You have Docker installed. If you don't have it already, follow the instructions here: [Get Started with Docker](#). It will involve creating or signing in with a Docker account, downloading a desktop app, and then launching the app to be able to use the Docker CLI. If you are running on a Linux server, follow the instructions for your distro here. You may be required to run Docker with sudo depending on your installation environment.

:::info

Code you'll see on this page is bash commands and their output.

When you see text in angle brackets `<>`, replace them and the text inside with your own value of what it refers to. Don't include the `<>` in the command.

:::

Celo Networks

First we are going to setup the environment variables required for Baklava network. Run:

```
bash
export CELOIMAGE=us.gcr.io/celo-org/gets:baklava
```

Pull the Celo Docker image

We're going to use a Docker image containing the Celo node software in this tutorial.

If you are re-running these instructions, the Celo Docker image may have been updated, and it's important to get the latest version.

```
bash
docker pull $CELOIMAGE
```

Set up a data directory

First, create the directory that will store your node's configuration and its copy of the blockchain. This directory can be named anything you'd like, but here's a default you can use. The commands below create a directory and then navigate into it. The rest of the steps assume you are running the commands from inside this directory.

```
bash
mkdir celo-data-dir
cd celo-data-dir
```

Create an account and get its address

In this step, you'll create an account on the network. If you've already done this and have an account address, you can skip this and move on to configuring your node.

Run the command to create a new account:

```
bash
docker run -v $PWD:/root/.celo --rm -it $CELOIMAGE account new
```

It will prompt you for a passphrase, ask you to confirm it, and then will output your account address: Public address of the key: <YOUR-ACCOUNT-ADDRESS>

Save this address to an environment variables, so that you can reference it below (don't include the braces):

```
bash
export CELOACCOUNTADDRESS=<YOUR-ACCOUNT-ADDRESS>
```

```
:::info
```

This environment variable will only persist while you have this terminal window open. If you want this environment variable to be available in the future, you can add it to your `/.bashprofile`

```
:::
```

Start the node

This command specifies the settings needed to run the node, and gets it started.

```
bash
```

```
docker run --name celo-fullnode -d --restart unless-stopped --stop-  
timeout 300 -p 127.0.0.1:8545:8545 -p 127.0.0.1:8546:8546 -p 30303:30303  
-p 30303:30303/udp -v $PWD:/root/.celo $CELOIMAGE --verbosity 3 --  
syncmode full --http --http.addr 0.0.0.0 --http.api  
eth,net,web3,debug,admin,personal --light.serve 90 --light.maxpeers 1000  
--maxpeers 1100 --etherbase $CELOACCOUNTADDRESS --baklava --datadir  
/root/.celo
```

You'll start seeing some output. After a few minutes, you should see lines that look like this. This means your node has started syncing with the network and is receiving blocks.

```
text  
INFO [07-16|14:04:24.924] Imported new chain segment  
blocks=139 txs=319 mgas=61.987 elapsed=8.085s mgasps=7.666 number=406  
hash=9acf16...4fddc8 age=6h58m44s cache=1.51mB  
INFO [07-16|14:04:32.928] Imported new chain segment  
blocks=303 txs=179 mgas=21.837 elapsed=8.004s mgasps=2.728 number=709  
hash=8de06a...77bb92 age=6h33m37s cache=1.77mB  
INFO [07-16|14:04:40.918] Imported new chain segment  
blocks=411 txs=0 mgas=0.000 elapsed=8.023s mgasps=0.000 number=1120  
hash=3db22a...9fa95a age=5h59m30s cache=1.92mB  
INFO [07-16|14:04:48.941] Imported new chain segment  
blocks=335 txs=0 mgas=0.000 elapsed=8.023s mgasps=0.000 number=1455  
hash=7eb3f8...32ebf0 age=5h31m43s cache=2.09mB  
INFO [07-16|14:04:56.944] Imported new chain segment  
blocks=472 txs=0 mgas=0.000 elapsed=8.003s mgasps=0.000 number=1927  
hash=4f1010...1414c1 age=4h52m31s cache=2.34mB
```

You will have fully synced with the network once you have pulled the latest block number, which you can lookup by visiting at the Baklava Network Stats or [Baklava Block Explorer]](<https://baklava-blockscout.celo-testnet.org/>) pages.

:::danger

Security: The command line above includes the parameter `--http.addr 0.0.0.0` which makes the Celo Blockchain software listen for incoming RPC requests on all network adaptors. Exercise extreme caution in doing this when running outside Docker, as it means that any unlocked accounts and their funds may be accessed from other machines on the Internet. In the context of running a Docker container on your local machine, this together with the `docker -p` flags allows you to make RPC calls from outside the container, i.e from your local host, but not from outside your machine. Read more about Docker Networking [here](#).

:::

Command Line Interface

Once the full node is running, it can serve the Command Line Interface tool `celocli`. For example:

```
bash
$ npm install -g @celo/celocli
...
$ celocli node:syncd
true
$ celocli account:new
...
```

run-validator.md:

```
---
title: Run Baklava Testnet Validator on Celo
description: How to get a Validator node running on the Celo Mainnet
(Celo's Node Operator Testnet).
---
```

Run Baklava Testnet Validator

How to get a Validator node running on the Celo Baklava Testnet (Celo's Node Operator Testnet).

Why run a Baklava Testnet Validator?

The Baklava testnet is the best place to get started running a validator, or test out new validator configurations before deploying to Mainnet.

:::info

If you would like to keep up-to-date with all the news happening in the Celo community, including validation, node operation and governance, please sign up to our Celo Signal mailing list [here](#).

You can add the Celo Signal public calendar as well which has relevant dates.

:::

:::info

If you are transitioning from the Baklava network prior to the June 24, 2021 reset, you will need to start with a fresh chain database. You can create new nodes from fresh machines, as described in this guide, or you may delete your chaindata folder, which is named celo in the node data directory, and start over by running the provided init commands for each node described below. All on-chain registration steps, the commands completed with celocli, will need to be run on the new network.

Key differences are:

- New network ID is 62320
- A new image has been pushed to [`us.gcr.io/celo-org/`](https://us.gcr.io/celo-org/)`geth:baklava`
- A new genesis block, bootnode enode, and the new network ID are included in the Docker image

:::

Prerequisites

Staking Requirements

Celo uses a proof-of-stake consensus mechanism, which requires Validators to have locked CELO to participate in block production. The current requirement is 10,000 CELO to register a Validator, and 10,000 CELO per member validator to register a Validator Group.

Participating in the Baklava testnet requires testnet units of CELO, which can only be used in the Baklava testnet. You can request a distribution of testnet CELO by filling out the faucet request form. If you need any help getting started, please join the discussion on Discord or email [`community@celo.org`](mailto:community@celo.org).

Faucetted funds will come as 2 transactions, one for your validator address and another for your validator group address.

Hardware requirements

The recommended Celo Validator setup involves continually running three instances:

- 1 Validator node: should be deployed to single-tenant hardware in a secure, high availability data center
- 1 Validator Proxy node: can be a VM or container in a multi-tenant environment (e.g. a public cloud), but requires high availability

Celo is a proof-of-stake network, which has different hardware requirements than a Proof of Work network. proof-of-stake consensus is less CPU intensive, but is more sensitive to network connectivity and latency. Below is a list of standard requirements for running Validator and Proxy nodes on the Celo Network:

Validator node

- CPU: At least 4 cores / 8 threads x8664 with 3ghz on modern CPU architecture newer than 2018 Intel Cascade Lake or Ryzen 3000 series or newer with a Geekbench 5 Single Threaded score of >1000 and Multi Threaded score of > 4000
- Memory: 32GB
- Disk: 512GB SSD or NVMe (resizable). Current chain size at August 16th is 190GB, so 512GB is a safe bet for the next 1 year. We recommend using a cloud provider or storage solution that allows you to resize your disk without downtime.
- Network: At least 1 GB input/output Ethernet with a fiber (low latency) Internet connection, ideally redundant connections and HA switches.

Some cloud instances that meet the above requirements are:

- GCP: n2-highmem-4, n2d-highmem-4 or c3-highmem-4
- AWS: r6i.xlarge, r6in.xlarge, or r6a.xlarge
- Azure: StandardE4v5, or StandardE4dv5 or StandardE4asv5

Proxy or Full node

- CPU: At least 4 cores / 8 threads x8664 with 3ghz on modern CPU architecture newer than 2018 Intel Cascade Lake or Ryzen 3000 series or newer with a Geekbench 5 Single Threaded score of >1000 and Multi Threaded score of > 4000
- Memory: 16GB
- Disk: 512GB SSD or NVMe (resizable). Current chain size at August 16th is 190GB, so 512GB is a safe bet for the next 1 year. We recommend using a cloud provider or storage solution that allows you to resize your disk without downtime.
- Network: At least 1 GB input/output Ethernet with a fiber (low latency) Internet connection, ideally redundant connections and HA switches.

Some cloud instances that meet the above requirements are:

- GCP: n2-standard-4, n2d-standard-4 or c3-standard-4
- AWS: M6i.xlarge, M6in.xlarge, or M6a.xlarge
- Azure: StandardD4v5, or StandardD4v4 or StandardD4asv5

In addition, to get things started, it will be useful to run a node on your local machine that you can issue CLI commands against.

Networking requirements

In order for your Validator to participate in consensus and complete attestations, it is critically important to configure your network correctly.

Your Proxy nodes must have static, external IP addresses, and your Validator node must be able to communicate with the Proxy, either via an internal network or via the Proxy's external IP address.

On the Validator machine, port 30503 should accept TCP connections from the IP address of your Proxy machine. This port is used by the Validator to communicate with the Proxy.

On the Proxy machine, port 30503 should accept TCP connections from the IP address of your Validator machine. This port is used by the Proxy to communicate with the Validator.

On the Proxy machines, port 30303 should accept TCP and UDP connections from all IP addresses. This port is used to communicate with other nodes in the network.

To illustrate this, you may refer to the following table:

Machine	IPs open to	0\0\0\0\0\0 (all)	your-validator-ip	your-proxy-ip
Validator				
Proxy		tcp:30303, udp:30303	tcp:30503	

Software requirements

On each machine

- You have Docker installed.

If you don't have it already, follow the instructions here: [Get Started with Docker](#). It will involve creating or signing in with a Docker account, downloading a desktop app, and then launching the app to be able to use the Docker CLI. If you are running on a Linux server, follow the instructions for your distro here. You may be required to run Docker with sudo depending on your installation environment.

You can check you have Docker installed and running if the command `docker info` works properly.

On your local machine

- You have celocli installed.

See Command Line Interface (CLI) for instructions on how to get set up.

- You are using the latest Node.js 12.x

Some users have reported issues using the most recent version of node. Use the LTS for greater reliability.

:::info

A note about conventions:

The code snippets you'll see on this page are bash commands and their output.

When you see text in angle brackets `< >`, replace them and the text inside with your own value of what it refers to. Don't include the `< >` in the command.

:::

Key Management

Private keys are the central primitive of any cryptographic system and need to be handled with extreme care. Loss of your private key can lead to irreversible loss of value.

This guide contains a large number of keys, so it is important to understand the purpose of each key. Read more about key management.

Unlocking

Celo nodes store private keys encrypted on disk with a password, and need to be "unlocked" before use. Private keys can be unlocked in two ways:

1. By running the `celocli account:unlock` command. Note that the node must have the "personal" RPC API enabled in order for this command to work.
2. By setting the `--unlock` flag when starting the node.

It is important to note that when a key is unlocked you need to be particularly careful about enabling access to the node's RPC APIs.

Environment variables

There are a number of environment variables in this guide, and you may use this table as a reference.

Variable	Explanation
-----	-----
CELOIMAGE	The Docker image used for the Validator and proxy containers
CELOVALIDATORGROUPADDRESS	The account address for the Validator Group
CELOVALIDATORADDRESS	The account address for the Validator
CELOVALIDATORGROUPSIGNERADDRESS	The validator (group) signer address authorized by the Validator Group account.
CELOVALIDATORGROUPSIGNERSIGNATURE	The proof-of-possession of the Validator Group signer key
CELOVALIDATORSIGNERADDRESS	The validator signer address authorized by the Validator Account
CELOVALIDATORSIGNERPUBKEY	The ECDSA public key associated with the Validator signer address
CELOVALIDATORSIGNERSIGNATURE	The proof-of-possession of the Validator signer key
CELOVALIDATORSIGNERBLSPUBKEY	The BLS public key for the Validator instance

CELOVALIDATORSIGNERBLSSIGNATURE	A proof-of-possession of the BLS public key
CELOVALIDATORGROUPVOTESIGNERADDRESS	The address of the Validator Group vote signer
CELOVALIDATORGROUPVOTESIGNERPUBKEY	The ECDSA public key associated with the Validator Group vote signer address
CELOVALIDATORGROUPVOTESIGNERSIGNATURE	The proof-of-possession of the Validator Group vote signer key
CELOVALIDATORVOTESIGNERADDRESS	The address of the Validator vote signer
CELOVALIDATORVOTESIGNERPUBKEY	The ECDSA public key associated with the Validator vote signer address
CELOVALIDATORVOTESIGNERSIGNATURE	The proof-of-possession of the Validator vote signer key
PROXYENODE	The enode address for the Validator proxy
PROXYINTERNALIP	(Optional) The internal IP address over which your Validator can communicate with your proxy
PROXYEXTERNALIP	The external IP address of the proxy. May be used by the Validator to communicate with the proxy if PROXYINTERNALIP is unspecified
DATABASEURL	The URL under which your database is accessible, currently supported are postgres://, mysql:// and sqlite://
APPSIGNATURE	The hash with which clients can auto-read SMS messages on android
SMSPROVIDERS	A comma-separated list of providers you want to configure, Celo currently supports nexmo & twilio

Validator Node Setup

This section outlines the steps needed to configure your Proxy and Validator nodes so that they are ready to sign blocks once elected.

Environment Variables

First we are going to set up the main environment variables related to the Baklava network. Run these on both your Validator and Proxy machines:

```
bash
export CELOIMAGE=us.gcr.io/celo-org/geth:baklava
```

Pull the Celo Docker image

In all the commands we are going to see the CELOIMAGE variable to refer to the Docker image to use. Now we can get the Docker image on your Validator and Proxy machines:

```
bash
docker pull $CELOIMAGE
```

The us.gcr.io/celo-org/geth:baklava image contains the genesis block in addition to the Celo Blockchain binary.

Account Creation

```
:::info
```

Please complete this section if you are new to validating on Celo.

```
:::
```

Account and Signer keys

Running a Celo Validator node requires the management of several different keys, each with different privileges. Keys that need to be accessed frequently (e.g. for signing blocks) are at greater risk of being compromised, and thus have more limited permissions, while keys that need to be accessed infrequently (e.g. for locking CELO) are less onerous to store securely, and thus have more expansive permissions. Below is a summary of the various keys that are used in the Celo network, and a description of their permissions.

Name of the key	Purpose
-----	-----
-----	-----
-----	-----
-----	-----
Account key	This is the key with the highest level of permissions, and is thus the most sensitive. It can be used to lock and unlock CELO, and authorize vote, validator keys. Note that the account key also has all of the permissions of the other keys.
Validator signer key	This is the key that has permission to register and manage a Validator or Validator Group, and participate in BFT consensus.
Vote signer key	This key can be used to vote in Validator elections and on-chain governance.

Note that Account and all the signer keys must be unique and may not be reused.

Generating Validator and Validator Group Keys

First, you'll need to generate account keys for your Validator and Validator Group.

:::danger

These keys will control your locked CELO, and thus should be handled with care.

Store and back these keys up in a secure manner, as there will be no way to recover them if lost or stolen.

:::

bash

On your local machine

mkdir celo-accounts-node

cd celo-accounts-node

docker run -v \$PWD:/root/.celo --rm -it \$CELOIMAGE account new

docker run -v \$PWD:/root/.celo --rm -it \$CELOIMAGE account new

This will create a new keystore in the current directory with two new accounts.

Copy the addresses from the terminal and set the following environment variables:

bash

On your local machine

export CELOVALIDATORGROUPADDRESS=<YOUR-VALIDATOR-GROUP-ADDRESS>

export CELOVALIDATORADDRESS=<YOUR-VALIDATOR-ADDRESS>

Start your Accounts node

Next, we'll run a node on your local machine so that we can use these accounts to lock CELO and authorize the keys needed to run your validator.

To run the node:

bash

On your local machine

mkdir celo-accounts-node

cd celo-accounts-node

docker run --name celo-accounts -it --restart always --stop-timeout 300 -

p 127.0.0.1:8545:8545 -v \$PWD:/root/.celo \$CELOIMAGE --verbosity 3 --

syncmode full --http --http.addr 0.0.0.0 --http.api

eth,net,web3,debug,admin,personal --baklava --light.serve 0 --datadir

/root/.celo

:::danger

Security: The command line above includes the parameter `--http.addr 0.0.0.0` which makes the Celo Blockchain software listen for incoming RPC requests on all network adaptors. Exercise extreme caution in doing this when running outside Docker, as it means that any unlocked accounts and their funds may be accessed from other machines on the Internet. In the context of running a Docker container on your local machine, this together with the `docker -p 127.0.0.1:localport:containerport` flags allows you to make RPC calls from outside the container, i.e from your local host, but not from outside your machine. Read more about Docker Networking [here](#).

:::

Deploy a Validator

To actually register as a validator, we'll need to generate a validating signer key. On your Validator machine (which should not be accessible from the public internet), follow very similar steps:

```
bash
On the validator machine
Note that you have to export $CELOIMAGE on this machine
export CELOIMAGE=us.gcr.io/celo-org/geth:baklava
mkdir celo-validator-node
cd celo-validator-node
docker run -v $PWD:/root/.celo --rm -it $CELOIMAGE account new
export CELOVALIDATORSIGNERADDRESS=<YOUR-VALIDATOR-SIGNER-ADDRESS>
```

Proof-of-Possession

:::info

Please complete this step if you are running a validator on Celo for the first time.

:::

In order to authorize our Validator signer, we need to create a proof that we have possession of the Validator signer private key. We do so by signing a message that consists of the Validator account address. To generate the proof-of-possession, run the following command:

```
bash
On the validator machine
Note that you have to export CELOVALIDATORADDRESS on this machine
export CELOVALIDATORADDRESS=<CELO-VALIDATOR-ADDRESS>
docker run -v $PWD:/root/.celo --rm -it $CELOIMAGE account proof-of-
possession $CELOVALIDATORSIGNERADDRESS $CELOVALIDATORADDRESS
```

Save the signer address, public key, and proof-of-possession signature to your local machine:

```
bash
```

On your local machine

```
export CELOVALIDATORSIGNERADDRESS=<YOUR-VALIDATOR-SIGNER-ADDRESS>
export CELOVALIDATORSIGNERSIGNATURE=<YOUR-VALIDATOR-SIGNER-SIGNATURE>
export CELOVALIDATORSIGNERPUBKEY=<YOUR-VALIDATOR-SIGNER-PUBLIC-KEY>
```

Validators on the Celo network use BLS aggregated signatures to create blocks in addition to the Validator signer (ECDSA) key. While an independent BLS key can be specified, the simplest thing to do is to derive the BLS key from the Validator signer key. When we register our Validator, we'll need to prove possession of the BLS key as well, which can be done by running the following command:

```
bash
```

On the validator machine

```
docker run -v $PWD:/root/.celo --rm -it $CELOIMAGE account proof-of-
possession $CELOVALIDATORSIGNERADDRESS $CELOVALIDATORADDRESS --bls
```

Save the resulting signature and public key to your local machine:

```
bash
```

On your local machine

```
export CELOVALIDATORSIGNERBLSSIGNATURE=<YOUR-VALIDATOR-SIGNER-SIGNATURE>
export CELOVALIDATORSIGNERBLSPUBKEY=<YOUR-VALIDATOR-SIGNER-BLS-PUBLIC-
KEY>
```

We'll get back to this machine later, but for now, let's give it a proxy.

Deploy a proxy

```
bash
```

On the proxy machine

```
mkdir celo-proxy-node
cd celo-proxy-node
export CELOVALIDATORSIGNERADDRESS=<YOUR-VALIDATOR-SIGNER-ADDRESS>
```

You can then run the proxy with the following command. Be sure to replace <YOUR-VALIDATOR-NAME> with the name you'd like to appear on Celostats. The validator name shown in Celostats will be the name configured in the proxy.

Additionally, you need to unlock the account configured in the etherbase option. It is recommended to create a new account and independent account only for this purpose. Be sure to write a new password to `./password` for this account (different to the Validator Signer password)

```
bash
```

On the proxy machine

First, we create a new account for the proxy

```
docker run --name celo-proxy-password -it --rm -v $PWD:/root/.celo
$CELOIMAGE account new --password /root/.celo/.password
```

Notice the public address returned by this command, that can be exported and used for running the proxy node:

```
bash
```

On the proxy machine

```
export PROXYADDRESS=<PROXY-PUBLIC-ADDRESS>
docker run --name celo-proxy -it --restart unless-stopped --stop-timeout
300 -p 30303:30303 -p 30303:30303/udp -p 30503:30503 -p 30503:30503/udp -
v $PWD:/root/.celo $CELOIMAGE --verbosity 3 --syncmode full --proxy.proxy
--proxy.proxiedvalidatoraddress $CELOVALIDATORSIGNERADDRESS --
proxy.internalendpoint :30503 --etherbase $PROXYADDRESS --unlock
$PROXYADDRESS --password /root/.celo/.password --allow-insecure-unlock --
baklava --datadir /root/.celo --celostats=<YOUR-VALIDATOR-NAME>@baklava-
celostats-server.celo-testnet.org
```

```
:::info
```

You can detach from the running container by pressing ctrl+p ctrl+q, or start it with -d instead of -it to start detached. Access the logs for a container in the background with the docker logs command.

```
:::
```

NOTES

- For the proxy to be able to send stats to Celostats, both the proxy and the validator should set the celostats flag
- If you are deploying multiple proxies for the same validator, the celostats flag should be added in only one of them

Get your Proxy's connection info

Once the Proxy is running, we will need to retrieve its enode and IP address so that the Validator will be able to connect to it.

```
bash
```

On the proxy machine, retrieve the proxy enode

```
docker exec celo-proxy geth --exec
"admin.nodeInfo['enode'].split('///')[1].split('@')[0]" attach | tr -d '"'
```

Now we need to set the Proxy enode and Proxy IP address in environment variables on the Validator machine.

If you don't have an internal IP address over which the Validator and Proxy can communicate, you can set the internal IP address to the external IP address.

If you don't know your proxy's external IP address, you can get it by running the following command:

```
bash
On the proxy machine
dig +short myip.opendns.com @resolver1.opendns.com
```

Then, export the variables on your Validator machine.

```
bash
On the Validator machine
export PROXYENODE=<YOUR-PROXY-ENODE>
export PROXYEXTERNALIP=<PROXY-MACHINE-EXTERNAL-IP-ADDRESS>
export PROXYINTERNALIP=<PROXY-MACHINE-INTERNAL-IP-ADDRESS>
```

You will also need to export PROXYEXTERNALIP on your local machine.

```
bash
On your local machine
export PROXYEXTERNALIP=<PROXY-MACHINE-EXTERNAL-IP-ADDRESS>
```

Connect the Validator to the Proxy

When your Validator starts up it will attempt to create a network connection with the proxy machine. You will need to make sure that your proxy machine has the appropriate firewall settings to allow the Validator to connect to it.

Specifically, on the proxy machine, port 30303 should allow TCP and UDP connections from all IP addresses. And port 30503 should allow TCP connections from the IP address of your Validator machine.

Test that your network is configured correctly by running the following commands:

```
bash
On your local machine, test that your proxy is accepting TCP connections
over port 30303.
Note that it will also need to be accepting UDP connections over this
port.
nc -vz $PROXYEXTERNALIP 30303
```

```
bash
On your Validator machine, test that your proxy is accepting TCP
connections over port 30503.
nc -vz $PROXYINTERNALIP 30503
```

Once that is completed, go ahead and run the Validator. Be sure to write your Validator signer password to `./password` for the following command to work, or provide your password another way.

```
bash
```

On the Validator machine

```
cd celo-validator-node
```

```
docker run --name celo-validator -it --restart unless-stopped --stop-  
timeout 300 -p 30303:30303 -p 30303:30303/udp -v $PWD:/root/.celo  
$CELOIMAGE --verbosity 3 --syncmode full --mine --etherbase  
$CELOVALIDATORSIGNERADDRESS --nodiscover --proxy.proxied --  
proxy.proxyenodeurlpairs=enode://$PROXYENODE@$PROXYINTERNALIP:30503\;enod  
e://$PROXYENODE@$PROXYEXTERNALIP:30303 --  
unlock=$CELOVALIDATORSIGNERADDRESS --password /root/.celo/.password --  
celostats=<YOUR-VALIDATOR-NAME>@baklava-celostats-server.celo-testnet.org  
--baklava --datadir /root/.celo
```

At this point your Validator and Proxy machines should be configured, and both should be syncing to the network. You should see Imported new chain segment in your node logs, about once every 5 seconds once the node is synced to the latest block which you can find on the Baklava Network Stats page.

```
:::info
```

You can run multiple proxies by deploying additional proxies per the instructions in the Deploy a proxy section. Then add all of the proxies' enodes as a comma separated list using the `--proxy.proxyenodeurlpairs` option. E.g. if there are two proxies, that option's usage would look like --

```
proxy.proxyenodeurlpairs=enode://$PROXYENODE1@$PROXYINTERNALIP1:30503\;en  
ode://$PROXYENODE1@$PROXYEXTERNALIP1:30303,enode://$PROXYENODE2@$PROXYINT  
ERNALIP2:30503\;enode://$PROXYENODE2@$PROXYEXTERNALIP2:30303
```

```
:::
```

Registering as a Validator

Register the Accounts

You've now done all the infrastructure setup to get a validator and proxy running. The cLabs team will review your submission to receive funds and send you 12,000 Baklava testnet CELO to each of your Validator and Validator Group account addresses. These funds have no real world value but will allow you to submit transactions to the network via `celocli` and put up a stake to register as a validator and validator group.

You can view your CELO balances by running the following commands:

```
bash
```

On your local machine

```
celocli account:balance $CELOVALIDATORGROUPADDRESS
```

```
celocli account:balance $CELOVALIDATORADDRESS
```


At some point the output of these commands will change from 0 to 12e12, indicating you have received the testnet CELD. This process involves a human, so please be patient. If you haven't received a balance within 24 hours, please get in touch again.

You can also look at an account's current balance and transaction history on Blockscout. Enter the address into the search bar.

Once these accounts have a balance, unlock them so that we can sign transactions. Then, we will register the accounts with the Celo core smart contracts:

```
bash
```

On your local machine

```
celocli account:unlock $CELOVALIDATORGROUPADDRESS
```

```
celocli account:unlock $CELOVALIDATORADDRESS
```

```
celocli account:register --from $CELOVALIDATORGROUPADDRESS --name <NAME  
YOUR VALIDATOR GROUP>
```

```
celocli account:register --from $CELOVALIDATORADDRESS --name <NAME YOUR VALIDATOR>
```

Check that your accounts were registered successfully with the following commands:

```
bash
```

On your local machine

```
celocli account:show $CELOVALIDATORGROUPADDRESS
```

```
celocli account:show $CELOVALIDATORADDRESS
```

Lock up CELO

Lock up testnet CELO for both accounts in order to secure the right to register a Validator and Validator Group. The current requirement is 10,000 CELO to register a validator, and 10,000 CELO per member validator to register a Validator Group. For Validators, this gold remains locked for approximately 60 days following deregistration. For groups, this gold remains locked for approximately 60 days following the removal of the Nth validator from the group.

```
bash
```

On your local machine

```
celocli lockedgold:lock --from $CELOVALIDATORGROUPADDRESS --value
10000000000000000000000000
```

```
celocli lockedgold:lock --from $CELOVALIDATORADDRESS --value  
1000000000000000000000
```

This amount (10,000 CELO) represents the minimum amount needed to be locked in order to register a Validator and Validator group. Since your balance is in fact higher than this, you may wish to lock more with these

accounts. Note that you will want to be sure to leave enough CELO unlocked to be able to continue to pay transaction fees for future transactions (such as those issued by running some CLI commands).

Check that your CELO was successfully locked with the following commands:

```
bash
On your local machine
celocli lockedgold:show $CELOVALIDATORGROUPADDRESS
celocli lockedgold:show $CELOVALIDATORADDRESS
```

Run for election

In order to be elected as a Validator, you will first need to register your group and Validator. Note that when registering a Validator Group, you need to specify a commission, which is the fraction of epoch rewards paid to the group by its members.

We don't want to use our account key for validating, so first let's authorize the validator signing key:

```
bash
On your local machine
celocli account:authorize --from $CELOVALIDATORADDRESS --role validator -
-signature 0x$CELOVALIDATORSIGNERSIGNATURE --signer
0x$CELOVALIDATORSIGNERADDRESS
```

Confirm by checking the authorized Validator signer for your Validator:

```
bash
On your local machine
celocli account:show $CELOVALIDATORADDRESS
```

Then, register your Validator Group by running the following command. Note that because we did not authorize a Validator signer for our Validator Group account, we register the Validator Group with the account key.

```
bash
On your local machine
celocli validatorgroup:register --from $CELOVALIDATORGROUPADDRESS --
commission 0.1
```

You can view information about your Validator Group by running the following command:

```
bash
On your local machine
celocli validatorgroup:show $CELOVALIDATORGROUPADDRESS
```

Next, register your Validator by running the following command. Note that because we have authorized a Validator signer, this step could also be performed on the Validator machine. Running it on the local machine allows us to avoid needing to install the celocli on the Validator machine.

```
bash
```

On your local machine

```
celocli validator:register --from $CELOVALIDATORADDRESS --ecdsaKey
$CELOVALIDATORSIGNERPUBLICKEY --blsKey $CELOVALIDATORSIGNERBLSPUBLICKEY -
--blsSignature $CELOVALIDATORSIGNERBLSSIGNATURE
```

Affiliate your Validator with your Validator Group. Note that you will not be a member of this group until the Validator Group accepts you. This command could also be run from the Validator signer, if running on the validator machine.

```
bash
```

On your local machine

```
celocli validator:affiliate $CELOVALIDATORGROUPADDRESS --from
$CELOVALIDATORADDRESS
```

Accept the affiliation:

```
bash
```

On your local machine

```
celocli validatorgroup:member --accept $CELOVALIDATORADDRESS --from
$CELOVALIDATORGROUPADDRESS
```

Next, double check that your Validator is now a member of your Validator Group:

```
bash
```

On your local machine

```
celocli validator:show $CELOVALIDATORADDRESS
```

```
celocli validatorgroup:show $CELOVALIDATORGROUPADDRESS
```

Use both accounts to vote for your Validator Group. Note that because we have not authorized a vote signer for either account, these transactions must be sent from the account keys. Since you're likely to need to place additional votes throughout the course of the stake-off, consider creating and authorizing vote signers for additional operational security.

bash

On your local machine

```
celocli election:vote --from $CELOVALIDATORADDRESS --for
$CELOVALIDATORGROUPADDRESS --value 1000000000000000000000000
```

```
celocli election:vote --from $CELOVALIDATORGROUPADDRESS --for
$CELOVALIDATORGROUPADDRESS --value 10000000000000000000000
```

Double check that your votes were cast successfully:

```
bash
```

On your local machine

```
celocli election:show $CELOVALIDATORGROUPADDRESS --group
celocli election:show $CELOVALIDATORGROUPADDRESS --voter
celocli election:show $CELOVALIDATORADDRESS --voter
```

Users in the Celo protocol receive epoch rewards for voting in Validator Elections only after submitting a special transaction to enable them. This must be done every time new votes are cast, and can only be made after the most recent epoch has ended. For convenience, we can use the following command, which will wait until the epoch has ended before sending a transaction:

```
bash
```

On your local machine

Note that this may take some time, as the epoch needs to end before votes can be activated

```
celocli election:activate --from $CELOVALIDATORADDRESS --wait && celocli
election:activate --from $CELOVALIDATORGROUPADDRESS --wait
```

Check that your votes were activated by re-running the following commands:

```
bash
```

On your local machine

```
celocli election:show $CELOVALIDATORGROUPADDRESS --voter
celocli election:show $CELOVALIDATORADDRESS --voter
```

If your Validator Group elects validators, you will receive epoch rewards in the form of additional Locked CELO voting for your Validator Group from your account addresses. You can see these rewards accumulate with the commands in the previous set, as well as:

```
bash
```

On your local machine

```
celocli lockedgold:show $CELOVALIDATORGROUPADDRESS
celocli lockedgold:show $CELOVALIDATORADDRESS
```

You're all set! Elections are finalized at the end of each epoch, roughly once an hour in the Alfajores or Baklava Testnets. After that hour, if you get elected, your node will start participating BFT consensus and validating blocks. After the first epoch in which your Validator participates in BFT, you should receive your first set of epoch rewards.

:::info

Roadmap: Different parameters will govern elections in a Celo production network. Epochs are likely to be daily, rather than hourly. Running a Validator will also include setting up proxy nodes to protect against DDoS attacks, and using hardware wallets to secure the key used to sign blocks. We plan to update these instructions with more details soon.

:::

You can inspect the current state of the validator elections by running:

```
bash
On your local machine
celocli election:list
```

If you find your Validator still not getting elected you may need to faucet yourself more funds and lock more gold in order to be able to cast more votes for your Validator Group!

You can check the status of your validator, including whether it is elected and signing blocks, at baklava-ethstats.celo-testnet.org or by running:

```
bash
On your local machine with celocli >= 0.0.30-beta9
celocli validator:status --validator $CELOVALIDATORADDRESS
```

You can see additional information about your validator, including uptime score, by running:

```
bash
On your local machine
celocli validator:show $CELOVALIDATORADDRESS
```

Deployment Tips

Running the Docker containers in the background

There are different options for executing Docker containers in the background. The most typical one is to use in your docker run commands the `-d` option. Also for long running processes, especially when you run in on a remote machine, you can use a tool like `screen`. It allows you to connect and disconnect from running processes providing an easy way to manage long running processes.

It's out of the scope of this documentation to go through the `screen` options, but you can use the following command format with your docker commands:

```
bash
```

```
screen -S <SESSION NAME> -d -m <YOUR COMMAND>
```

For example:

```
bash
screen -S celo-validator -d -m docker run --name celo-validator -it --
restart unless-stopped --stop-timeout 300 -p 127.0.0.1:8545:8545 .....
```

You can list your existing screen sessions:

```
bash
screen -ls
```

And re-attach to any of the existing sessions:

```
bash
screen -r -S celo-validator
```

Stopping containers

You can stop the Docker containers at any time without problem. If you stop your containers that means those containers stop providing service. The data directory of the Validator and the proxy are Docker volumes mounted in the containers from the celo--dir you created at the very beginning. So if you don't remove that folder, you can stop or restart the containers without losing any data.

It is recommended to use the Docker stop timeout parameter -t when stopping the containers. This allows time, in this case 60 seconds, for the Celo nodes to flush recent chain data it keeps in memory into the data directories. Omitting this may cause your blockchain data to corrupt, requiring the node to start syncing from scratch.

You can stop the celo-validator and celo-proxy containers running:

```
bash
docker stop celo-validator celo-proxy -t 60
```

And you can remove the containers (not the data directory) by running:

```
bash
docker rm -f celo-validator celo-proxy
```

disclaimer.md:

title: Celo Mainnet Disclaimer

description: Important considerations, warnings, and legal regulations for users of the Celo Mainnet.

Mainnet Disclaimer

Important considerations, warnings, and legal regulations for users of Celo mainnet.

Terms and Conditions

By using, and contributing to, Celo blockchain and cryptographic network (Celo), you (the User) agree to these terms and acknowledge that the use of Celo, including any and all release candidates, is entirely at the User's sole risk. You also agree to adhere to the Celo Code of Conduct.

All content provided on Celo is subject to the License included and is provided on an 'AS IS' and 'AS AVAILABLE' basis, without any representations or warranties of any kind. All implied terms are excluded to the fullest extent permitted by law. No party involved in, or having contributed to the development of, Celo including any of their affiliates, directors, employees, contractors, service providers or agents (the Parties Involved) accepts any responsibility or liability to the User or any third parties in relation to any materials or information accessed or downloaded via Celo. The User acknowledges and agrees that the Parties Involved are not responsible for any damage to the User's computer systems, loss of data, or any other loss or damage resulting (directly or indirectly) from use of Celo.

Users also understand that release candidates may be replaced, in which event the network may cease to operate, all on-chain data may be destroyed and any rewards or other benefits may be lost and the cliff for ReleaseGold contracts reset.

To the fullest extent permitted by law, in no event shall the Parties Involved have any liability whatsoever to any person for any direct or indirect loss, liability, cost, claim, expense or damage of any kind, whether in contract or in tort, including negligence or otherwise, arising out of or related to the use of all or part of Celo.

The Celo software is not subject to the EAR based on Section 734.7 of the U.S. Export Administration Regulations ("EAR", 15 CFR Parts 730-774) and Section 742.15(b) of the EAR, which applies to software containing or designed for use with encryption software that is publicly available as open-source. However, products developed using Celo may be subject to the EAR or local laws/regulations. The User is responsible for compliance with U.S. and local country export/import laws and regulations.

Celo software may not be exported/reexported, either directly or indirectly, to any destination subject to U.S. embargoes or trade sanctions unless formally authorized by the U.S. Government. The embargoed destinations are subject to change and the scope of what is

included in the embargo is specific to each embargoed country. For the most current information on U.S. embargoed and sanctioned countries, see the Treasury Department regulations.

index.md:

title: Celo Mainnet

description: Collection of resources to get started with Celo Mainnet (Celo's production network).

Mainnet

Collection of resources to get started with Celo Mainnet (Celo's production network).

:::note

Celo Mainnet was previously known as the Release Candidate 1 network.

:::

:::tip

Please refer to Key Concepts for background on blockchains and an explanation of terms used in the section.

:::

Useful links

- Mainnet Validator Explorer - to view the current status of Validator elections

- Mainnet Network Status - to check the current availability of the network

- Mainnet Network Block Explorer - explore the history of the blockchain and view transaction details

run-full-node.md:

title: Run a Celo Full Node

description: How to run a full node on the Celo Mainnet Network using a prebuilt Docker image.

Run a Full Node

How to run on the Mainnet Network using a prebuilt Docker image.

:::tip Hosted Nodes

If you'd prefer a simple, one click hosted setup for running a node on one of the major cloud providers (AWS and GCP), checkout our hosted nodes documentation.

:::

:::info

If you would like to keep up-to-date with all the news happening in the Celo community, including validation, node operation and governance, please sign up to our Celo Signal mailing list [here](#).

You can add the Celo Signal public calendar as well which has relevant dates.

:::

Full nodes play a special purpose in the Celo ecosystem, acting as a bridge between the mobile wallets \(\running as light clients\) and the validator nodes.

Prerequisites

- You have Docker installed. If you don't have it already, follow the instructions [here](#): Get Started with Docker. It will involve creating or signing in with a Docker account, downloading a desktop app, and then launching the app to be able to use the Docker CLI. If you are running on a Linux server, follow the instructions for your distro [here](#). You may be required to run Docker with sudo depending on your installation environment.

:::info

Code you'll see on this page is bash commands and their output.

When you see text in angle brackets `<>`, replace them and the text inside with your own value of what it refers to. Don't include the `<>` in the command.

:::

Celo Networks

First we are going to setup the environment variables required for the mainnet network. Run:

```
bash
export CELOIMAGE=us.gcr.io/celo-org/geth:mainnet
```

Pull the Celo Docker image

We're going to use a Docker image containing the Celo node software in this tutorial.

If you are re-running these instructions, the Celo Docker image may have been updated, and it's important to get the latest version.

```
bash
docker pull $CELOIMAGE
```

Set up a data directory

First, create the directory that will store your node's configuration and its copy of the blockchain. This directory can be named anything you'd like, but here's a default you can use. The commands below create a directory and then navigate into it. The rest of the steps assume you are running the commands from inside this directory.

```
bash
mkdir celo-data-dir
cd celo-data-dir
```

Create an account and get its address

In this step, you'll create an account on the network. If you've already done this and have an account address, you can skip this and move on to configuring your node.

Run the command to create a new account:

```
bash
docker run -v $PWD:/root/.celo --rm -it $CELOIMAGE account new
```

It will prompt you for a passphrase, ask you to confirm it, and then will output your account address: Public address of the key: <YOUR-ACCOUNT-ADDRESS>

Save this address to an environment variables, so that you can reference it below (don't include the braces):

```
bash
export CELOACCOUNTADDRESS=<YOUR-ACCOUNT-ADDRESS>
```

:::info

This environment variable will only persist while you have this terminal window open. If you want this environment variable to be available in the future, you can add it to your `/.bashprofile`

:::

Start the node

This command specifies the settings needed to run the node, and gets it started.

```
bash
docker run --name celo-fullnode -d --restart unless-stopped --stop-
timeout 300 -p 127.0.0.1:8545:8545 -p 127.0.0.1:8546:8546 -p 30303:30303
-p 30303:30303/udp -v $PWD:/root/.celo $CELOIMAGE --verbosity 3 --
syncmode full --http --http.addr 0.0.0.0 --http.api
eth,net,web3,debug,admin,personal --light.serve 90 --light.maxpeers 1000
--maxpeers 1100 --etherbase $CELOACCOUNTADDRESS --datadir /root/.celo
```

You'll start seeing some output. After a few minutes, you should see lines that look like this. This means your node has started syncing with the network and is receiving blocks.

```
text
INFO [07-16|14:04:24.924] Imported new chain segment
blocks=139 txs=319 mgas=61.987 elapsed=8.085s mgasps=7.666 number=406
hash=9acf16...4fddc8 age=6h58m44s cache=1.51mB
INFO [07-16|14:04:32.928] Imported new chain segment
blocks=303 txs=179 mgas=21.837 elapsed=8.004s mgasps=2.728 number=709
hash=8de06a...77bb92 age=6h33m37s cache=1.77mB
INFO [07-16|14:04:40.918] Imported new chain segment
blocks=411 txs=0 mgas=0.000 elapsed=8.023s mgasps=0.000 number=1120
hash=3db22a...9fa95a age=5h59m30s cache=1.92mB
INFO [07-16|14:04:48.941] Imported new chain segment
blocks=335 txs=0 mgas=0.000 elapsed=8.023s mgasps=0.000 number=1455
hash=7eb3f8...32ebf0 age=5h31m43s cache=2.09mB
INFO [07-16|14:04:56.944] Imported new chain segment
blocks=472 txs=0 mgas=0.000 elapsed=8.003s mgasps=0.000 number=1927
hash=4f1010...1414c1 age=4h52m31s cache=2.34mB
```

You will have fully synced with the network once you have pulled the latest block number, which you can lookup by visiting the Network Stats or Block Explorer pages.

:::danger

Security: The command line above includes the parameter `--http.addr 0.0.0.0` which makes the Celo Blockchain software listen for incoming RPC requests on all network adaptors. Exercise extreme caution in doing this when running outside Docker, as it means that any unlocked accounts and their funds may be accessed from other machines on the Internet. In the context of running a Docker container on your local machine, this together with the `docker -p` flags allows you to make RPC calls from outside the container, i.e from your local host, but not from outside your machine. Read more about Docker Networking [here](#).

:::

Running an Archive Node

If you would like to run an archive node for celo-blockchain, you can run the following command:

```
bash
docker run --name celo-fullnode -d --restart unless-stopped --stop-
timeout 300 -p 127.0.0.1:8545:8545 -p 127.0.0.1:8546:8546 -p 30303:30303
-p 30303:30303/udp -v $PWD:/root/.celo $CELOIMAGE --verbosity 3 --
syncmode full --gcmode archive --txlookuplimit=0 --cache.preimages --http
--http.addr 0.0.0.0 --http.api eth,net,web3,debug,admin,personal --
light.serve 90 --light.maxpeers 1000 --maxpeers 1100 --etherbase
$CELOACCOUNTADDRESS --datadir /root/.celo
```

We add the following flags: `--gcmode archive --txlookuplimit=0 --cache.preimages`

In celo-blockchain, this is called `gcmode` which refers to the concept of garbage collection. Setting it to `archive` basically turns it off.

Command Line Interface

Once the full node is running, it can serve the Command Line Interface tool `celocli`. For example:

```
bash
$ npm install -g @celo/celocli
...
$ celocli node:syncd
true
$ celocli account:new
...
```

```
# run-validator.md:
```

```
---
title: Running a Celo Validator
description: How to get a Validator node running on the Celo Mainnet.
---
```

Running a Validator

How to get a Validator node running on the Celo Mainnet.

```
---
```

:::info

If you would like to keep up-to-date with all the news happening in the Celo community, including validation, node operation and governance, please sign up to our Celo Signal mailing list [here](#).

You can add the Celo Signal public calendar as well which has relevant dates.

:::

What is a Validator?

Validators help secure the Celo network by participating in Celo's proof-of-stake protocol. Validators are organized into Validator Groups, analogous to parties in representative democracies. A Validator Group is essentially an ordered list of Validators.

Just as anyone in a democracy can create their own political party, or seek to get selected to represent a party in an election, any Celo user can create a Validator group and add themselves to it, or set up a potential Validator and work to get an existing Validator group to include them.

While other Validator Groups will exist on the Celo Network, the fastest way to get up and running with a Validator will be to register a Validator Group, register a Validator, and affiliate that Validator with your Validator Group. The addresses used to register Validator Groups and Validators must be unique, which will require that you create two accounts in the step-by-step guide below.

Because of the importance of Validator security and availability, Validators are expected to run a "proxy" node in front of each Validator node. In this setup, the Proxy node connects with the rest of the network, and the Validator node communicates only with the Proxy, ideally via a private network.

Read more about Celo's mission and why you may want to become a Validator.

Prerequisites

Staking Requirements

Celo uses a proof-of-stake consensus mechanism, which requires Validators to have locked CELO to participate in block production. The current requirement is 10,000 CELO to register a Validator, and 10,000 CELO per member validator to register a Validator Group.

If you do not have the required CELO to lock up, you can try out of the process of creating a validator on the Baklava network by following the [Running a Validator in Baklava](#) guide

We will not discuss obtaining CELO here, but it is a prerequisite that you obtain the required CELO.

Hardware requirements

The recommended Celo Validator setup involves continually running three instances:

- 1 Validator node: should be deployed to single-tenant hardware in a secure, high availability data center
- 1 Validator Proxy node: can be a VM or container in a multi-tenant environment (e.g. a public cloud), but requires high availability
- <!-- - 1 Attestation node: can be a VM or container in a multi-tenant environment (e.g. a public cloud), and has moderate availability requirements -->

Celo is a proof-of-stake network, which has different hardware requirements than a Proof of Work network. proof-of-stake consensus is less CPU intensive, but is more sensitive to network connectivity and latency. Below is a list of standard requirements for running Validator and Proxy nodes on the Celo Network:

Validator node

- CPU: At least 4 cores / 8 threads x8664 with 3ghz on modern CPU architecture newer than 2018 Intel Cascade Lake or Ryzen 3000 series or newer with a Geekbench 5 Single Threaded score of >1000 and Multi Threaded score of > 4000
- Memory: 32GB
- Disk: 512GB SSD or NVMe (resizable). Current chain size at August 16th is 190GB, so 512GB is a safe bet for the next 1 year. We recommend using a cloud provider or storage solution that allows you to resize your disk without downtime.
- Network: At least 1 GB input/output Ethernet with a fiber (low latency) Internet connection, ideally redundant connections and HA switches.

Some cloud instances that meet the above requirements are:

- GCP: n2-highmem-4, n2d-highmem-4 or c3-highmem-4
- AWS: r6i.xlarge, r6in.xlarge, or r6a.xlarge
- Azure: StandardE4v5, or StandardE4dv5 or StandardE4asv5

Proxy or Full node

- CPU: At least 4 cores / 8 threads x8664 with 3ghz on modern CPU architecture newer than 2018 Intel Cascade Lake or Ryzen 3000 series or newer with a Geekbench 5 Single Threaded score of >1000 and Multi Threaded score of > 4000
- Memory: 16GB
- Disk: 512GB SSD or NVMe (resizable). Current chain size at August 16th is 190GB, so 512GB is a safe bet for the next 1 year. We recommend using a cloud provider or storage solution that allows you to resize your disk without downtime.
- Network: At least 1 GB input/output Ethernet with a fiber (low latency) Internet connection, ideally redundant connections and HA switches.

Some cloud instances that meet the above requirements are:

- GCP: n2-standard-4, n2d-standard-4 or c3-standard-4
- AWS: M6i.xlarge, M6in.xlarge, or M6a.xlarge
- Azure: StandardD4v5, or StandardD4v4 or StandardD4asv5

Archive Nodes

- CPU: At least 4 cores / 8 threads x8664 with 3ghz on modern CPU architecture newer than 2018 Intel Cascade Lake or Ryzen 3000 series or newer with a Geekbench 5 Single Threaded score of >1000 and Multi Threaded score of > 4000
- Memory: 16GB
- Disk: 3TB SSD or NVMe (resizable). Current chain size at August 16th is 2.1TB, so 3TB is a safe bet for the next 6 months. We recommend using a cloud provider or storage solution that allows you to resize your disk without downtime.
- Network: At least 1 GB input/output Ethernet with a fiber (low latency) Internet connection, ideally redundant connections and HA switches.

Some cloud instances that meet the above requirements are:

- GCP: n2-standard-4, n2d-standard-4 or c3-standard-4
- AWS: M6i.xlarge, M6in.xlarge, or M6a.xlarge
- Azure: StandardD4v5, or StandardD4v4 or StandardD4asv5

Networking requirements

In order for your Validator to participate in consensus, it is critically important to configure your network correctly.

Your Proxy nodes must have static, external IP addresses, and your Validator node must be able to communicate with the Proxy, either via an internal network or via the Proxy's external IP address.

On the Validator machine, port 30503 should accept TCP connections from the IP address of your Proxy machine. This port is used by the Validator to communicate with the Proxy.

On the Proxy machine, port 30503 should accept TCP connections from the IP address of your Validator machine. This port is used by the Proxy to communicate with the Validator.

On the Proxy machines, port 30303 should accept TCP and UDP connections from all IP addresses. This port is used to communicate with other nodes in the network.

To illustrate this, you may refer to the following table:

Machine	IPs open to	0\0\0\0\0\0 (all)	your\-validator\-ip	your\-proxy\-ip
Validator				
tcp:30503				

| Proxy | tcp:30303, udp:30303 | tcp:30503 |
|

Software requirements

On each machine

- You have Docker installed.

If you don't have it already, follow the instructions here: [Get Started with Docker](#). It will involve creating or signing in with a Docker account, downloading a desktop app, and then launching the app to be able to use the Docker CLI. If you are running on a Linux server, follow the instructions for your distro here. You may be required to run Docker with `sudo` depending on your installation environment.

You can check you have Docker installed and running if the command `docker info` works properly.

On your local machine

- You have `celocli` installed.

See [Command Line Interface \(CLI\)](#) for instructions on how to get set up.

- You are using the latest Node 10.x LTS

Some users have reported issues using the most recent version of node. Use the LTS for greater reliability.

:::info

A note about conventions:

The code snippets you'll see on this page are bash commands and their output.

When you see text in angle brackets `<>`, replace them and the text inside with your own value of what it refers to. Don't include the `<>` in the command.

:::

Key Management

Private keys are the central primitive of any cryptographic system and need to be handled with extreme care. Loss of your private key can lead to irreversible loss of value.

This guide contains a large number of keys, so it is important to understand the purpose of each key. Read more about [key management](#).

Unlocking

Celo nodes store private keys encrypted on disk with a password, and need to be "unlocked" before use. Private keys can be unlocked in two ways:

1. By running the `celocli account:unlock` command. Note that the node must have the "personal" RPC API enabled in order for this command to work.
2. By setting the `--unlock` flag when starting the node.

It is important to note that when a key is unlocked you need to be particularly careful about enabling access to the node's RPC APIs.

Environment variables

There are a number of environment variables in this guide, and you may use this table as a reference.

Variable	Explanation
-----	-----
CELOIMAGE	The Docker image used for the Validator and proxy containers
CELOVALIDATORGROUPADDRESS	The account address for the Validator Group
CELOVALIDATORADDRESS	The account address for the Validator
CELOVALIDATORGROUPSIGNERADDRESS	The validator (group) signer address authorized by the Validator Group account.
CELOVALIDATORGROUPSIGNERSIGNATURE	The proof-of-possession of the Validator Group signer key
CELOVALIDATORSIGNERADDRESS	The validator signer address authorized by the Validator Account
CELOVALIDATORSIGNERPUBKEY	The ECDSA public key associated with the Validator signer address
CELOVALIDATORSIGNERSIGNATURE	The proof-of-possession of the Validator signer key
CELOVALIDATORSIGNERBLSPUBKEY	The BLS public key for the Validator instance
CELOVALIDATORSIGNERBLSSIGNATURE	A proof-of-possession of the BLS public key
CELOVALIDATORGROUPVOTESIGNERADDRESS	The address of the Validator Group vote signer
CELOVALIDATORGROUPVOTESIGNERPUBKEY	The ECDSA public key associated with the Validator Group vote signer address

CELOVALIDATORGROUPVOTESIGNERSIGNATURE	The proof-of-possession of the Validator Group vote signer key
CELOVALIDATORVOTESIGNERADDRESS	The address of the Validator vote signer
CELOVALIDATORVOTESIGNERPUBKEY	The ECDSA public key associated with the Validator vote signer address
CELOVALIDATORVOTESIGNERSIGNATURE	The proof-of-possession of the Validator vote signer key
PROXYENODE	The enode address for the Validator proxy
PROXYINTERNALIP	(Optional) The internal IP address over which your Validator can communicate with your proxy
PROXYEXTERNALIP	The external IP address of the proxy. May be used by the Validator to communicate with the proxy if PROXYINTERNALIP is unspecified
CELOATTESTATIONSIGNERADDRESS	The address of the attestation signer authorized by the Validator Account
CELOATTESTATIONSIGNERSIGNATURE	The proof-of-possession of the attestation signer key
CELOATTESTATIONSERVICEURL	The URL to access the deployed Attestation Service
METADATAURL	The URL to access the metadata file for your Attestation Service
DATABASEURL	The URL under which your database is accessible, currently supported are postgres://, mysql:// and sqlite://
SMSPROVIDERS	A comma-separated list of providers you want to configure, Celo currently supports nexmo & twilio

Network Deployment Timeline

The setup of mainnet is similar to the new Baklava network and the deployment timeline is as follows (all dates are subject to change):

Done:

- 4/19 00:00 UTC: Docker image with genesis block distributed
- 4/19 - 4/22: Infrastructure setup
- 4/22 16:00 UTC: Block production begins
- 4/22: Celo Core Contracts and ReleaseGold contracts are deployed
- 4/27: Governance proposals submitted to unfreeze validator elections and validator epoch rewards

- 5/1: Elections and validator rewards enabled if governance proposals pass; validators have been registered and affiliated with a Validator Group for first election
- 5/14: Governance proposals submitted to enable voter rewards and enable CELO transfers
- 5/18: RC1 is declared Mainnet and CELO transfers are enabled, if governance proposals pass

Upcoming:

- 5/25: Deployment of CELO/USD Oracles
- 5/31: Governance proposal submitted to unfreeze stability protocol
- 6/3: Stability protocol goes live if governance proposal passes

:::info

A timeline of the Release Candidate 1 and Mainnet networks is available to provide further context.

:::

Validator Node Setup

This section outlines the steps needed to configure your Proxy and Validator nodes so that they are ready to sign blocks once elected.

Environment Variables

First we are going to set up the main environment variables related to the mainnet network. Run these on both your Validator and Proxy machines:

```
bash
export CELOIMAGE=us.gcr.io/celo-org/geth:mainnet
```

Pull the Celo Docker image

In all the commands we are going to see the CELOIMAGE variable to refer to the Docker image to use. Now we can get the Docker image on your Validator and Proxy machines:

```
bash
docker pull $CELOIMAGE
```

Account Creation

:::info

Please complete this section if you are new to validating on Celo.

:::

Account and Signer keys

Running a Celo Validator node requires the management of several different keys, each with different privileges. Keys that need to be accessed frequently (e.g. for signing blocks) are at greater risk of being compromised, and thus have more limited permissions, while keys that need to be accessed infrequently (e.g. for locking CELO) are less onerous to store securely, and thus have more expansive permissions. Below is a summary of the various keys that are used in the Celo network, and a description of their permissions.

Name of the key	Purpose
Account key	This is the key with the highest level of permissions, and is thus the most sensitive. It can be used to lock and unlock CELO, and authorize vote, validator, and attestation keys. Note that the account key also has all of the permissions of the other keys.
Validator signer key	This is the key that has permission to register and manage a Validator or Validator Group, and participate in BFT consensus.
Vote signer key	This key can be used to vote in Validator elections and on-chain governance.
Attestation signer key	This key is used to sign attestations in Celo's lightweight identity protocol.

Note that Account and all the signer keys must be unique and may not be reused.

Generating Validator and Validator Group Keys

First, you'll need to generate account keys for your Validator and Validator Group.

```
:::warning
```

These keys will control your locked CELO, and thus should be handled with care.

Store and back these keys up in a secure manner, as there will be no way to recover them if lost or stolen.

```
:::
```

```
bash
```

```
On your local machine
```

```
mkdir celo-accounts-node
```

```
cd celo-accounts-node
```

```
docker run -v $PWD:/root/.celo --rm -it $CELOIMAGE account new
```

```
docker run -v $PWD:/root/.celo --rm -it $CELOIMAGE account new
```

This will create a new keystore in the current directory with two new accounts.

Copy the addresses from the terminal and set the following environment variables:

```
bash
```

On your local machine

```
export CELOVALIDATORGROUPADDRESS=<YOUR-VALIDATOR-GROUP-ADDRESS>
```

```
export CELOVALIDATORADDRESS=<YOUR-VALIDATOR-ADDRESS>
```

Start your Accounts node

Next, we'll run a node on your local machine so that we can use these accounts to lock CELO and authorize the keys needed to run your validator. To do this, we need to run the following command to run the node.

```
bash
```

On your local machine

```
mkdir celo-accounts-node
```

```
cd celo-accounts-node
```

```
docker run --name celo-accounts -it --restart always --stop-timeout 300 -  
p 127.0.0.1:8545:8545 -v $PWD:/root/.celo $CELOIMAGE --verbosity 3 --  
syncmode full --http --http.addr 0.0.0.0 --http.api  
eth,net,web3,debug,admin,personal --datadir /root/.celo
```

:::danger

Security: The command line above includes the parameter `--http.addr 0.0.0.0` which makes the Celo Blockchain software listen for incoming RPC requests on all network adaptors. Exercise extreme caution in doing this when running outside Docker, as it means that any unlocked accounts and their funds may be accessed from other machines on the Internet. In the context of running a Docker container on your local machine, this together with the `docker -p 127.0.0.1:localport:containerport` flags allows you to make RPC calls from outside the container, i.e from your local host, but not from outside your machine. Read more about Docker Networking [here](#).

:::

Deploy a Validator Machine

To actually register as a validator, we'll need to generate a validating signer key. On your Validator machine (which should not be accessible from the public internet), follow very similar steps:

```
bash
```

On the validator machine

Note that you have to export `$CELOIMAGE` on this machine

```
export CELOIMAGE=us.gcr.io/celo-org/geth:mainnet
mkdir celo-validator-node
cd celo-validator-node
docker run -v $PWD:/root/.celo --rm -it $CELOIMAGE account new
export CELOVALIDATORSIGNERADDRESS=<YOUR-VALIDATOR-SIGNER-ADDRESS>
```

Proof-of-Possession

:::info

Please complete this step if you are running a validator on Celo for the first time.

:::

In order to authorize our Validator signer, we need to create a proof that we have possession of the Validator signer private key. We do so by signing a message that consists of the Validator account address. To generate the proof-of-possession, run the following command:

```
bash
On the validator machine
Note that you have to export CELOVALIDATORADDRESS on this machine
export CELOVALIDATORADDRESS=<CELO-VALIDATOR-ADDRESS>
docker run -v $PWD:/root/.celo --rm -it $CELOIMAGE account proof-of-
possession $CELOVALIDATORSIGNERADDRESS $CELOVALIDATORADDRESS
```

Save the signer address, public key, and proof-of-possession signature to your local machine:

```
bash
On your local machine
export CELOVALIDATORSIGNERADDRESS=<YOUR-VALIDATOR-SIGNER-ADDRESS>
export CELOVALIDATORSIGNERSIGNATURE=<YOUR-VALIDATOR-SIGNER-SIGNATURE>
export CELOVALIDATORSIGNERPUBKEY=<YOUR-VALIDATOR-SIGNER-PUBLIC-KEY>
```

Validators on the Celo network use BLS aggregated signatures to create blocks in addition to the Validator signer (ECDSA) key. While an independent BLS key can be specified, the simplest thing to do is to derive the BLS key from the Validator signer key. When we register our Validator, we'll need to prove possession of the BLS key as well, which can be done by running the following command:

```
bash
On the validator machine
docker run -v $PWD:/root/.celo --rm -it $CELOIMAGE account proof-of-
possession $CELOVALIDATORSIGNERADDRESS $CELOVALIDATORADDRESS --bls
```

Save the resulting signature and public key to your local machine:

```
bash
On your local machine
export CELOVALIDATORSIGNERBLSSIGNATURE=<YOUR-VALIDATOR-SIGNER-SIGNATURE>
export CELOVALIDATORSIGNERBLSPUBLICKEY=<YOUR-VALIDATOR-SIGNER-BLS-PUBLIC-KEY>
```

We'll get back to this machine later, but for now, let's give it a proxy.

Deploy a proxy

```
bash
On the proxy machine
mkdir celo-proxy-node
cd celo-proxy-node
export CELOVALIDATORSIGNERADDRESS=<YOUR-VALIDATOR-SIGNER-ADDRESS>
```

You can then run the proxy with the following command. Be sure to replace `<YOUR-VALIDATOR-NAME>` with the name you'd like to appear on Celostats. The validator name shown in Celostats will be the name configured in the proxy.

Additionally, you need to unlock the account configured in the etherbase option. It is recommended to create a new account and independent account only for this purpose. Be sure to write a new password to `./password` for this account (different to the Validator Signer password)

```
bash
On the proxy machine
First, we create a new account for the proxy
docker run --name celo-proxy-password -it --rm -v $PWD:/root/.celo
$CELOIMAGE account new --password /root/.celo/.password
```

Notice the public address returned by this command, that can be exported and used for running the proxy node:

```
bash
On the proxy machine
export PROXYADDRESS=<PROXY-PUBLIC-ADDRESS>
docker run --name celo-proxy -it --restart unless-stopped --stop-timeout
300 -p 30303:30303 -p 30303:30303/udp -p 30503:30503 -p 30503:30503/udp -
v $PWD:/root/.celo $CELOIMAGE --verbosity 3 --syncmode full --proxy.proxy
--proxy.proxiedvalidatoraddress $CELOVALIDATORSIGNERADDRESS --
proxy.internalendpoint :30503 --etherbase $PROXYADDRESS --unlock
$PROXYADDRESS --password /root/.celo/.password --allow-insecure-unlock --
light.serve 0 --datadir /root/.celo --celostats=<YOUR-VALIDATOR-
NAME>@stats-server.celo.org
```

Hint: If you are running into trouble peering with the full nodes, one of the first things to check is whether your container's ports are properly

configured (i.e. specifically, `-p 30303:30303 -p 30303:30303/udp` - which is set in the proxy node's command, but not the account node's command).

:::info

You can detach from the running container by pressing `ctrl+p ctrl+q`, or start it with `-d` instead of `-it` to start detached. Access the logs for a container in the background with the `docker logs` command.

:::

NOTES

- For the proxy to be able to send stats to Celostats, both the proxy and the validator should set the `celostats` flag
- If you are deploying multiple proxies for the same validator, the `celostats` flag should be added in only one of them

Get your Proxy's connection info

Once the Proxy is running, we will need to retrieve its `enode` and IP address so that the Validator will be able to connect to it.

bash

On the proxy machine, retrieve the proxy `enode`

`docker exec celo-proxy geth --exec`

`"admin.nodeInfo['enode'].split('///')[1].split('@')[0]" attach | tr -d '"'`

Now we need to set the Proxy `enode` and Proxy IP address in environment variables on the Validator machine.

If you don't have an internal IP address over which the Validator and Proxy can communicate, you can set the internal IP address to the external IP address.

If you don't know your proxy's external IP address, you can get it by running the following command:

bash

On the proxy machine

`dig +short myip.opendns.com @resolver1.opendns.com`

Then, export the variables on your Validator machine.

bash

On the Validator machine

`export PROXYENODE=<YOUR-PROXY-ENODE>`

`export PROXYEXTERNALIP=<PROXY-MACHINE-EXTERNAL-IP-ADDRESS>`

`export PROXYINTERNALIP=<PROXY-MACHINE-INTERNAL-IP-ADDRESS>`

You will also need to export `PROXYEXTERNALIP` on your local machine.


```
bash
On your local machine
export PROXYEXTERNALIP=<PROXY-MACHINE-EXTERNAL-IP-ADDRESS>
```

Connect the Validator to the Proxy

When your Validator starts up it will attempt to create a network connection with the proxy machine. You will need to make sure that your proxy machine has the appropriate firewall settings to allow the Validator to connect to it.

Specifically, on the proxy machine, port 30303 should allow TCP and UDP connections from all IP addresses. And port 30503 should allow TCP connections from the IP address of your Validator machine.

Test that your network is configured correctly by running the following commands:

```
bash
On your local machine, test that your proxy is accepting TCP connections
over port 30303.
Note that it will also need to be accepting UDP connections over this
port.
nc -vz $PROXYEXTERNALIP 30303
```

```
bash
On your Validator machine, test that your proxy is accepting TCP
connections over port 30503.
nc -vz $PROXYINTERNALIP 30503
```

Once that is completed, go ahead and run the Validator. Be sure to write your Validator signer password to `./password` for the following command to work, or provide your password another way.

```
bash
On the Validator machine
cd celo-validator-node
docker run --name celo-validator -it --restart unless-stopped --stop-
timeout 300 -p 30303:30303 -p 30303:30303/udp -v $PWD:/root/.celo
$CELOIMAGE --verbosity 3 --syncmode full --mine --etherbase
$CELOVALIDATORSIGNERADDRESS --nodiscover --proxy.proxied --
proxy.proxyenodeurlpairs=enode://$PROXYENODE@$PROXYINTERNALIP:30503\;enod
e://$PROXYENODE@$PROXYEXTERNALIP:30303 --
unlock=$CELOVALIDATORSIGNERADDRESS --password /root/.celo/.password --
light.serve 0 --celostats=<YOUR-VALIDATOR-NAME>@stats-server.celo.org
```

At this point your Validator and Proxy machines should be configured, and both should be syncing to the network. You should see Imported new chain

segment in your node logs, about once every 5 seconds once the node is synced to the latest block which you can find on the Network Stats page.

```
:::info
```

You can run multiple proxies by deploying additional proxies per the instructions in the Deploy a proxy section. Then add all of the proxies' enodes as a comma separated list using the `--proxy.proxyenodeurlpairs` option. E.g. if there are two proxies, that option's usage would look like --

```
proxy.proxyenodeurlpairs=enode://$PROXYENODE1@$PROXYINTERNALIP1:30503\;enode://$PROXYENODE1@$PROXYEXTERNALIP1:30303,enode://$PROXYENODE2@$PROXYINTERNALIP2:30503\;enode://$PROXYENODE2@$PROXYEXTERNALIP2:30303
```

```
:::
```

Registering as a Validator

Register the Accounts

You've now done all the infrastructure setup to get a validator and proxy running. To run a validator on Mainnet, you must lock CELO to participate in block production. Once you have CELO in your validator and validation group accounts, you can view their balances:

```
bash
```

On your local machine

```
celocli account:balance $CELOVALIDATORGROUPADDRESS
```

```
celocli account:balance $CELOVALIDATORADDRESS
```

You can also look at an account's current balance and transaction history on Celo Explorer. Enter the address into the search bar.

Once these accounts have a balance, unlock them so that we can sign transactions. Then, we will register the accounts with the Celo core smart contracts:

```
bash
```

On your local machine

```
celocli account:unlock $CELOVALIDATORGROUPADDRESS
```

```
celocli account:unlock $CELOVALIDATORADDRESS
```

```
celocli account:register --from $CELOVALIDATORGROUPADDRESS --name <NAME YOUR VALIDATOR GROUP>
```

```
celocli account:register --from $CELOVALIDATORADDRESS --name <NAME YOUR VALIDATOR>
```

Check that your accounts were registered successfully with the following commands:

```
bash
```

On your local machine

```
celocli account:show $CELOVALIDATORGROUPADDRESS
```



```
celocli account:show $CELOVALIDATORADDRESS
```

Then, register your Validator Group by running the following command. Note that because we did not authorize a Validator signer for our Validator Group account, we register the Validator Group with the account key.

```
bash
On your local machine
celocli validatorgroup:register --from $CELOVALIDATORGROUPADDRESS --
commission 0.1
```

You can view information about your Validator Group by running the following command:

```
bash
On your local machine
celocli validatorgroup:show $CELOVALIDATORGROUPADDRESS
```

Next, register your Validator by running the following command. Note that because we have authorized a Validator signer, this step could also be performed on the Validator machine. Running it on the local machine allows us to avoid needing to install the celocli on the Validator machine.

```
bash
On your local machine
celocli validator:register --from $CELOVALIDATORADDRESS --ecdsaKey
$CELOVALIDATORSIGNERPUBKEY --blsKey $CELOVALIDATORSIGNERBLSPUBKEY -
-blsSignature $CELOVALIDATORSIGNERBLSSIGNATURE
```

Affiliate your Validator with your Validator Group. Note that you will not be a member of this group until the Validator Group accepts you. This command could also be run from the Validator signer, if running on the validator machine.

```
bash
On your local machine
celocli validator:affiliate $CELOVALIDATORGROUPADDRESS --from
$CELOVALIDATORADDRESS
```

Accept the affiliation:

```
bash
On your local machine
celocli validatorgroup:member --accept $CELOVALIDATORADDRESS --from
$CELOVALIDATORGROUPADDRESS
```

Next, double check that your Validator is now a member of your Validator Group:

```
bash
On your local machine
celocli validator:show $CELOVALIDATORADDRESS
celocli validatorgroup:show $CELOVALIDATORGROUPADDRESS
```

Use both accounts to vote for your Validator Group. Note that because we have not authorized a vote signer for either account, these transactions must be sent from the account keys.

```
bash
On your local machine
celocli election:vote --from $CELOVALIDATORADDRESS --for
$CELOVALIDATORGROUPADDRESS --value 1000000000000000000000000
celocli election:vote --from $CELOVALIDATORGROUPADDRESS --for
$CELOVALIDATORGROUPADDRESS --value 1000000000000000000000000
```

Double check that your votes were cast successfully:

```
bash
On your local machine
celocli election:show $CELOVALIDATORGROUPADDRESS --group
celocli election:show $CELOVALIDATORGROUPADDRESS --voter
celocli election:show $CELOVALIDATORADDRESS --voter
```

Users in the Celo protocol receive epoch rewards for voting in Validator Elections only after submitting a special transaction to enable them. This must be done every time new votes are cast, and can only be made after the most recent epoch has ended. For convenience, we can use the following command, which will wait until the epoch has ended before sending a transaction:

```
bash
On your local machine
Note that this may take some time, as the epoch needs to end before votes
can be activated
celocli election:activate --from $CELOVALIDATORADDRESS --wait && celocli
election:activate --from $CELOVALIDATORGROUPADDRESS --wait
```

Check that your votes were activated by re-running the following commands:

```
bash
On your local machine
celocli election:show $CELOVALIDATORGROUPADDRESS --voter
celocli election:show $CELOVALIDATORADDRESS --voter
```

If your Validator Group elects validators, you will receive epoch rewards in the form of additional Locked CELO voting for your Validator Group from your account addresses. You can see these rewards accumulate with the commands in the previous set, as well as:

```
bash
On your local machine
celocli lockedgold:show $CELOVALIDATORGROUPADDRESS
celocli lockedgold:show $CELOVALIDATORADDRESS
```

You're all set! Elections are finalized at the end of each epoch, roughly once a day in the Mainnet network. After that hour, if you get elected, your node will start participating BFT consensus and validating blocks. After the first epoch in which your Validator participates in BFT, you should receive your first set of epoch rewards. You can inspect the current state of the validator elections by running:

```
bash
On your local machine
celocli election:list
```

You can check the status of your validator, including whether it is elected and signing blocks, at stats.celo.org or by running:

```
bash
celocli validator:status --validator $CELOVALIDATORADDRESS
```

You can see additional information about your validator, including uptime score, by running:

```
bash
On your local machine
celocli validator:show $CELOVALIDATORADDRESS
```

Deployment Tips

Running the Docker containers in the background

There are different options for executing Docker containers in the background. The most typical one is to use in your docker run commands the -d option. Also for long running processes, especially when you run in on a remote machine, you can use a tool like screen. It allows you to connect and disconnect from running processes providing an easy way to manage long running processes.

It's out of the scope of this documentation to go through the screen options, but you can use the following command format with your docker commands:

```
bash
```

```
screen -S <SESSION NAME> -d -m <YOUR COMMAND>
```

For example:

```
bash
screen -S celo-validator -d -m docker run --name celo-validator -it --
restart unless-stopped --stop-timeout 300 -p 127.0.0.1:8545:8545 .....
```

You can list your existing screen sessions:

```
bash
screen -ls
```

And re-attach to any of the existing sessions:

```
bash
screen -r -S celo-validator
```

Stopping containers

You can stop the Docker containers at any time without problem. If you stop your containers that means those containers stop providing service. The data directory of the Validator and the proxy are Docker volumes mounted in the containers from the celo--dir you created at the very beginning. So if you don't remove that folder, you can stop or restart the containers without losing any data.

It is recommended to use the Docker stop timeout parameter -t when stopping the containers. This allows time, in this case 60 seconds, for the Celo nodes to flush recent chain data it keeps in memory into the data directories. Omitting this may cause your blockchain data to corrupt, requiring the node to start syncing from scratch.

You can stop the celo-validator and celo-proxy containers running:

```
bash
docker stop celo-validator celo-proxy -t 60
```

And you can remove the containers (not the data directory) by running:

```
bash
docker rm -f celo-validator celo-proxy
```

forno.md:

title: Celo Forno

description: How to connect to Celo without running a full node using Forno.

Forno

How to connect to Celo without running a full node using Forno.

What is Forno?

Forno is a cLabs hosted node service for interacting with the Celo network. This allows you to connect to the Celo Blockchain without having to run your own node.

:::tip

Forno does not offer a terms of service and there are no guarantees about service uptime. For production applications, consider using or Quicknode.

:::

Forno has HTTP and websocket endpoints that you can use to query current Celo data or post transactions that you would like to broadcast to the network. The service runs full nodes in non-archive mode, so you can query the current state of the blockchain, but cannot access historic state.

Forno can be used as an Http Provider with ContractKit.

```
javascript
const Web3 = require("web3");
const ContractKit = require("@celo/contractkit");

const web3 = new Web3("https://forno.celo.org");
const kit = ContractKit.newKitFromWeb3(web3);
```

Forno is a public node, so to send transactions from a Forno connection you will have to sign transactions with a private key before sending them to Forno.

Forno networks

Consult this page to determine which network is right for you.

Celo Mainnet

```
bash
https://forno.celo.org
```

Websocket support:


```
bash
wss://forno.celo.org/ws
```

Alfajores Testnet

```
bash
https://alfajores-forno.celo-testnet.org
```

Websocket support:

```
bash
wss://alfajores-forno.celo-testnet.org/ws
```

Websocket connections & Event listeners

Websocket connections are useful for listening to logs (aka events) emitted by a smart contract, but Forno only allows a websocket connection for 20 minutes before disconnecting. On disconnect, you can reconnect to the websocket endpoint to keep listening. Here is an example script of how to set up an event listener that reconnects when the connection is broken.

overview.md:

```
---
title: Nodes and Services
description: Nodes as a Service
---
```

Nodes and Services

Connect to nodes and services in the Celo Ecosystem.

:::warning
Celo is currently transitioning from a standalone Layer 1 blockchain to an Ethereum Layer 2. As a result, certain information about developer resources may be outdated.

For the latest information, please refer to our Celo L2 documentation.
:::

Forno

Forno is a cLabs hosted node service for interacting with the Celo network. This allows you to connect to the Celo Blockchain without having to run your own node.

Forno has HTTP and WebSocket endpoints that you can use to query current Celo data or post transactions that you would like to broadcast to the network. The service runs full nodes in non-archive mode, so you can query the current state of the blockchain, but cannot access the historic state.

Forno can be used as an HTTP Provider with ContractKit.

cLabs Hosted Node

- Forno

Run a Node

Running your own RPC endpoint.

- Mainnet Full Node
- Alfajores Full Node
- Baklava Full Node
- Hosted Nodes

As a Service

Paid RPC endpoint hosting.

- Ankr
- Infura
- Quicknode
- All that node
- dRPC

run-alfajores.md:

title: Run an Alfajores Full Node on Celo
description: How to run a full node on the Alfajores Network using a prebuilt Docker image.

Run an Alfajores Full Node

How to run a full node on the Alfajores Network using a prebuilt Docker image.

What is a Full Node?

Full nodes play a special purpose in the Celo ecosystem, acting as a bridge between the mobile wallets \ (running as light clients\) and the validator nodes.

:::tip

If you'd prefer a simple, one-click hosted setup for running a node on one of the major cloud providers (AWS and GCP), check out our hosted nodes documentation.

:::

:::info

If you would like to keep up-to-date with all the news happening in the Celo community, including validation, node operation, and governance, please sign up for our Celo Signal mailing list.

You can add the Celo Signal public calendar as well which has relevant dates.

:::

Prerequisites

- You have Docker installed. If you don't have it already, follow the instructions here: [Get Started with Docker](#). It will involve creating or signing in with a Docker account, downloading a desktop app, and then launching the app to be able to use the Docker CLI. If you are running on a Linux server, follow the instructions for your distro here. You may be required to run Docker with sudo depending on your installation environment.

:::info

The code you'll see on this page is bash commands and their output.

When you see text in angle brackets `<>`, replace them and the text inside with your own value of what it refers to. Don't include the `<>` in the command.

:::

Celo Networks

First, we are going to setup the environment variables required for Alfajores network. Run:

```
bash
export CELOIMAGE=us.gcr.io/celo-org/geth:alfajores
```

Pull the Celo Docker image

We're going to use a Docker image containing the Celo node software in this tutorial.

If you are re-running these instructions, the Celo Docker image may have been updated, and it's important to get the latest version.

```
bash
docker pull $CELOIMAGE
```

Set up a data directory

First, create the directory that will store your node's configuration and its copy of the blockchain. This directory can be named anything you'd like, but here's a default you can use. The commands below create a directory and then navigate into it. The rest of the steps assume you are running the commands from inside this directory.

```
bash
mkdir celo-data-dir
cd celo-data-dir
```

Create an account and get its address

In this step, you'll create an account on the network. If you've already done this and have an account address, you can skip this and move on to configuring your node.

Run the command to create a new account:

```
bash
docker run -v $PWD:/root/.celo --rm -it $CELOIMAGE account new
```

It will prompt you for a passphrase, ask you to confirm it, and then output your account address: Public address of the key: <YOUR-ACCOUNT-ADDRESS>

Save this address to an environment variable, so that you can reference it below (don't include the braces):

```
bash
export CELOACCOUNTADDRESS=<YOUR-ACCOUNT-ADDRESS>
```

:::info

This environment variable will only persist while you have this terminal window open. If you want this environment variable to be available in the future, you can add it to your `/.bashprofile`

:::

Start the node

This command specifies the settings needed to run the node, and gets it started.

```
bash
docker run --name celo-fullnode -d --restart unless-stopped --stop-
timeout 300 -p 127.0.0.1:8545:8545 -p 127.0.0.1:8546:8546 -p 30303:30303
-p 30303:30303/udp -v $PWD:/root/.celo $CELOIMAGE --verbosity 3 --
syncmode full --http --http.addr 0.0.0.0 --http.api
eth,net,web3,debug,admin,personal --light.serve 90 --light.maxpeers 1000
--maxpeers 1100 --etherbase $CELOACCOUNTADDRESS --alfajores --datadir
/root/.celo
```

You'll start seeing some output. After a few minutes, you should see lines that look like this. This means your node has started syncing with the network and is receiving blocks.

```
text
INFO [07-16|14:04:24.924] Imported new chain segment
blocks=139 txs=319 mgas=61.987 elapsed=8.085s mgasps=7.666 number=406
hash=9acf16...4fddc8 age=6h58m44s cache=1.51mB
INFO [07-16|14:04:32.928] Imported new chain segment
blocks=303 txs=179 mgas=21.837 elapsed=8.004s mgasps=2.728 number=709
hash=8de06a...77bb92 age=6h33m37s cache=1.77mB
INFO [07-16|14:04:40.918] Imported new chain segment
blocks=411 txs=0 mgas=0.000 elapsed=8.023s mgasps=0.000 number=1120
hash=3db22a...9fa95a age=5h59m30s cache=1.92mB
INFO [07-16|14:04:48.941] Imported new chain segment
blocks=335 txs=0 mgas=0.000 elapsed=8.023s mgasps=0.000 number=1455
hash=7eb3f8...32ebf0 age=5h31m43s cache=2.09mB
INFO [07-16|14:04:56.944] Imported new chain segment
blocks=472 txs=0 mgas=0.000 elapsed=8.003s mgasps=0.000 number=1927
hash=4f1010...1414c1 age=4h52m31s cache=2.34mB
```

You will have fully synced with the network once you have pulled the latest block number, which you can lookup by visiting at the Alfajores Network Stats or Alfajores Block Explorer pages.

:::danger

Security: The command line above includes the parameter `--http.addr 0.0.0.0` which makes the Celo Blockchain software listen for incoming RPC requests on all network adaptors. Exercise extreme caution in doing this when running outside Docker, as it means that any unlocked accounts and their funds may be accessed from other machines on the Internet. In the context of running a Docker container on your local machine, this together with the `docker -p` flags allows you to make RPC calls from outside the container, i.e from your local host, but not from outside your machine. Read more about Docker Networking [here](#).

:::

Command Line Interface

Once the full node is running, it can serve the Command Line Interface tool `celocli`. For example:

```
bash
$ npm install -g @celo/celocli
...
$ celocli node:syncd
true
$ celocli account:new
...
```

```
# run-baklava.md:
```

```
---
title: Run a Baklava Full Node
description: How to get a full node running on the Baklava Network using
a prebuilt Docker image.
---
```

Run a Baklava Full Node

How to get a full node running on the Baklava Network using a prebuilt Docker image.

What is a Baklava Full Node?

Full nodes play a special purpose in the Celo ecosystem, acting as a bridge between the mobile wallets \ (running as light clients\) and the validator nodes.

:::info

If you would like to keep up-to-date with all the news happening in the Celo community, including validation, node operation and governance, please sign up to our Celo Signal mailing list [here](#).

You can add the Celo Signal public calendar as well which has relevant dates.

:::

:::info

If you are transitioning from the Baklava network prior to the June 24 reset, you will need to start with a fresh chain database. You can either shut down your existing node, delete the celo folder, and continue by following the guide below or create a new node following these directions.

Key differences are:

- New network ID is 62320

- A new image has been pushed to `us.gcr.io/celo-org/geth:baklava`
- A new genesis block and bootnode enode are included in the Docker image
- ReleaseGold contracts are available for all previously faucettted addresses here

:::

Prerequisites

- You have Docker installed. If you don't have it already, follow the instructions here: [Get Started with Docker](#). It will involve creating or signing in with a Docker account, downloading a desktop app, and then launching the app to be able to use the Docker CLI. If you are running on a Linux server, follow the instructions for your distro here. You may be required to run Docker with `sudo` depending on your installation environment.

:::info

Code you'll see on this page is bash commands and their output.

When you see text in angle brackets `<>`, replace them and the text inside with your own value of what it refers to. Don't include the `<>` in the command.

:::

Celo Networks

First, we are going to setup the environment variables required for Baklava network. Run:

```
bash
export CELOIMAGE=us.gcr.io/celo-org/geth:baklava
```

Pull the Celo Docker image

We're going to use a Docker image containing the Celo node software in this tutorial.

If you are re-running these instructions, the Celo Docker image may have been updated, and it's important to get the latest version.

```
bash
docker pull $CELOIMAGE
```

Set up a data directory

First, create the directory that will store your node's configuration and its copy of the blockchain. This directory can be named anything you'd like, but here's a default you can use. The commands below create a

directory and then navigate into it. The rest of the steps assume you are running the commands from inside this directory.

```
bash
mkdir celo-data-dir
cd celo-data-dir
```

Create an account and get its address

In this step, you'll create an account on the network. If you've already done this and have an account address, you can skip this and move on to configuring your node.

Run the command to create a new account:

```
bash
docker run -v $PWD:/root/.celo --rm -it $CELOIMAGE account new
```

It will prompt you for a passphrase, ask you to confirm it, and then output your account address: Public address of the key: <YOUR-ACCOUNT-ADDRESS>

Save this address to an environment variable, so that you can reference it below (don't include the braces):

```
bash
export CELOACCOUNTADDRESS=<YOUR-ACCOUNT-ADDRESS>
```

:::info

This environment variable will only persist while you have this terminal window open. If you want this environment variable to be available in the future, you can add it to your `/.bashprofile`

:::

Start the node

This command specifies the settings needed to run the node and gets it started.

```
bash
docker run --name celo-fullnode -d --restart unless-stopped --stop-
timeout 300 -p 127.0.0.1:8545:8545 -p 127.0.0.1:8546:8546 -p 30303:30303
-p 30303:30303/udp -v $PWD:/root/.celo $CELOIMAGE --verbosity 3 --
syncmode full --http --http.addr 0.0.0.0 --http.api
eth,net,web3,debug,admin,personal --light.serve 90 --light.maxpeers 1000
--maxpeers 1100 --etherbase $CELOACCOUNTADDRESS --baklava --datadir
/root/.celo
```


You'll start seeing some output. After a few minutes, you should see lines that look like this. This means your node has started syncing with the network and is receiving blocks.

```
text
INFO [07-16|14:04:24.924] Imported new chain segment
blocks=139 txs=319 mgas=61.987 elapsed=8.085s mgasps=7.666 number=406
hash=9acf16...4fddc8 age=6h58m44s cache=1.51mB
INFO [07-16|14:04:32.928] Imported new chain segment
blocks=303 txs=179 mgas=21.837 elapsed=8.004s mgasps=2.728 number=709
hash=8de06a...77bb92 age=6h33m37s cache=1.77mB
INFO [07-16|14:04:40.918] Imported new chain segment
blocks=411 txs=0 mgas=0.000 elapsed=8.023s mgasps=0.000 number=1120
hash=3db22a...9fa95a age=5h59m30s cache=1.92mB
INFO [07-16|14:04:48.941] Imported new chain segment
blocks=335 txs=0 mgas=0.000 elapsed=8.023s mgasps=0.000 number=1455
hash=7eb3f8...32ebf0 age=5h31m43s cache=2.09mB
INFO [07-16|14:04:56.944] Imported new chain segment
blocks=472 txs=0 mgas=0.000 elapsed=8.003s mgasps=0.000 number=1927
hash=4f1010...1414c1 age=4h52m31s cache=2.34mB
```

You will have fully synced with the network once you have pulled the latest block number, which you can look up by visiting at the Baklava Network Stats or Baklava Block Explorer pages.

:::danger

Security: The command line above includes the parameter `--http.addr 0.0.0.0` which makes the Celo Blockchain software listen for incoming RPC requests on all network adaptors. Exercise extreme caution in doing this when running outside Docker, as it means that any unlocked accounts and their funds may be accessed from other machines on the Internet. In the context of running a Docker container on your local machine, this together with the `docker -p` flags allows you to make RPC calls from outside the container, i.e., from your local host, but not from outside your machine. Read more about Docker Networking [here](#).

:::

Command Line Interface

Once the full node is running, it can serve the Command Line Interface tool `celocli`. For example:

```
bash
$ npm install -g @celo/celocli
...
$ celocli node:synced
true
$ celocli account:new
...
```

```
# run-hosted.md:
```

```
---
```

```
title: Hosted Nodes
```

```
description: How to get a preconfigured Celo blockchain node running on  
one of the major cloud providers.
```

```
---
```

Hosted Nodes

How to get a preconfigured Celo blockchain node running on one of the major cloud providers.

```
---
```

Before getting started

cLabs currently provides machine images for launching full and lightest nodes on Alfajores and Mainnet. These prebuilt images are updated with every release of the Celo blockchain client and are available on Amazon Web Services and Google Cloud Platform.

Before proceeding with a hosted Celo blockchain node, you'll need to have an account with your cloud provider of choice and basic knowledge of networking.

:::info

If you would like to keep up-to-date with all the news happening in the Celo community, including validation, node operation and governance, please sign up to our Celo Signal mailing list.

You can add the Celo Signal public calendar as well which has relevant dates.

:::

Currently, cLabs provides the following machine images:

- celo-alfajores-full-node-latest
- celo-alfajores-lightest-node-latest
- celo-mainnet-full-node-latest
- celo-mainnet-lightest-node-latest

Please note that the time taken to sync a full node could be significant.

Google Cloud Platform

GCP by default won't display public machine images when you search for them in your console. This means you'll need to go via the API or gcloud command line to launch a node.

Depending on the type of node you'd like to launch (see the above list), the gcloud command to use may look a bit like this:

```
bash
gcloud compute instances create <INSTANCENAME> --image <IMAGENAME> --
image-project devopsre --project <YOURGCPPROJECT>
```

If you are running a image with full syncmode, please increase the disk size and instance type, and optionally use a SSD disk:

```
bash
--boot-disk-size 250 --boot-disk-type pd-ssd --machine-type=n2-standard-4
```

For full sync mode, it will take several days to sync the whole chain. You can check the status by running the next command:

```
bash
containername=celo-full-node
docker exec -it $containername geth attach --exec 'eth.syncing'
```

For more information please check the excellent GCP documentation on how to launch a compute instance from a public image.

run-mainnet.md:

```
---
title: Run a Celo Full Node
description: How to run a full node on the Celo Mainnet Network using a
prebuilt Docker image.
---
```

Run a Full Node

How to run on the Mainnet Network using a prebuilt Docker image.

:::tip Hosted Nodes

If you'd prefer a simple, one click hosted setup for running a node on one of the major cloud providers (AWS and GCP), checkout our hosted nodes documentation.

:::

:::info

If you would like to keep up-to-date with all the news happening in the Celo community, including validation, node operation and governance, please sign up to our Celo Signal mailing list [here](#).

You can add the Celo Signal public calendar as well which has relevant dates.

:::

Full nodes play a special purpose in the Celo ecosystem, acting as a bridge between the mobile wallets \(\running as light clients\) and the validator nodes.

Prerequisites

- You have Docker installed. If you don't have it already, follow the instructions here: [Get Started with Docker](#). It will involve creating or signing in with a Docker account, downloading a desktop app, and then launching the app to be able to use the Docker CLI. If you are running on a Linux server, follow the instructions for your distro here. You may be required to run Docker with sudo depending on your installation environment.

:::info

Code you'll see on this page is bash commands and their output.

When you see text in angle brackets `<>`, replace them and the text inside with your own value of what it refers to. Don't include the `<>` in the command.

:::

Celo Networks

First we are going to setup the environment variables required for the mainnet network. Run:

```
bash
export CELOIMAGE=us.gcr.io/celo-org/geth:mainnet
```

Pull the Celo Docker image

We're going to use a Docker image containing the Celo node software in this tutorial.

If you are re-running these instructions, the Celo Docker image may have been updated, and it's important to get the latest version.

```
bash
docker pull $CELOIMAGE
```

Set up a data directory

First, create the directory that will store your node's configuration and its copy of the blockchain. This directory can be named anything you'd like, but here's a default you can use. The commands below create a directory and then navigate into it. The rest of the steps assume you are running the commands from inside this directory.

```
bash
mkdir celo-data-dir
cd celo-data-dir
```

Create an account and get its address

In this step, you'll create an account on the network. If you've already done this and have an account address, you can skip this and move on to configuring your node.

Run the command to create a new account:

```
bash
docker run -v $PWD:/root/.celo --rm -it $CELOIMAGE account new
```

It will prompt you for a passphrase, ask you to confirm it, and then will output your account address: Public address of the key: <YOUR-ACCOUNT-ADDRESS>

Save this address to an environment variables, so that you can reference it below (don't include the braces):

```
bash
export CELOACCOUNTADDRESS=<YOUR-ACCOUNT-ADDRESS>
```

:::info

This environment variable will only persist while you have this terminal window open. If you want this environment variable to be available in the future, you can add it to your `/.bashprofile`

:::

Start the node

This command specifies the settings needed to run the node, and gets it started.

```
bash
docker run --name celo-fullnode -d --restart unless-stopped --stop-
timeout 300 -p 127.0.0.1:8545:8545 -p 127.0.0.1:8546:8546 -p 30303:30303
-p 30303:30303/udp -v $PWD:/root/.celo $CELOIMAGE --verbosity 3 --
syncmode full --http --http.addr 0.0.0.0 --http.api
eth,net,web3,debug,admin,personal --light.serve 90 --light.maxpeers 1000
--maxpeers 1100 --etherbase $CELOACCOUNTADDRESS --datadir /root/.celo
```

You'll start seeing some output. After a few minutes, you should see lines that look like this. This means your node has started syncing with the network and is receiving blocks.

text

```
INFO [07-16|14:04:24.924] Imported new chain segment
blocks=139 txs=319 mgas=61.987 elapsed=8.085s mgasps=7.666 number=406
hash=9acfl6...4fddc8 age=6h58m44s cache=1.51mB
INFO [07-16|14:04:32.928] Imported new chain segment
blocks=303 txs=179 mgas=21.837 elapsed=8.004s mgasps=2.728 number=709
hash=8de06a...77bb92 age=6h33m37s cache=1.77mB
INFO [07-16|14:04:40.918] Imported new chain segment
blocks=411 txs=0 mgas=0.000 elapsed=8.023s mgasps=0.000 number=1120
hash=3db22a...9fa95a age=5h59m30s cache=1.92mB
INFO [07-16|14:04:48.941] Imported new chain segment
blocks=335 txs=0 mgas=0.000 elapsed=8.023s mgasps=0.000 number=1455
hash=7eb3f8...32ebf0 age=5h31m43s cache=2.09mB
INFO [07-16|14:04:56.944] Imported new chain segment
blocks=472 txs=0 mgas=0.000 elapsed=8.003s mgasps=0.000 number=1927
hash=4f1010...1414c1 age=4h52m31s cache=2.34mB
```

You will have fully synced with the network once you have pulled the latest block number, which you can lookup by visiting the Network Stats or Block Explorer pages.

:::danger

Security: The command line above includes the parameter `--http.addr 0.0.0.0` which makes the Celo Blockchain software listen for incoming RPC requests on all network adaptors. Exercise extreme caution in doing this when running outside Docker, as it means that any unlocked accounts and their funds may be accessed from other machines on the Internet. In the context of running a Docker container on your local machine, this together with the `docker -p` flags allows you to make RPC calls from outside the container, i.e from your local host, but not from outside your machine. Read more about Docker Networking [here](#).

:::

Running an Archive Node

If you would like to run an archive node for celo-blockchain, you can run the following command:

```
bash
docker run --name celo-fullnode -d --restart unless-stopped --stop-
timeout 300 -p 127.0.0.1:8545:8545 -p 127.0.0.1:8546:8546 -p 30303:30303
-p 30303:30303/udp -v $PWD:/root/.celo $CELOIMAGE --verbosity 3 --
syncmode full --gcmode archive --txlookuplimit=0 --cache.preimages --http
--http.addr 0.0.0.0 --http.api eth,net,web3,debug,admin,personal --
```

```
light.serve 90 --light.maxpeers 1000 --maxpeers 1100 --etherbase
$CELOACCOUNTADDRESS --datadir /root/.celo
```

We add the following flags: `--gcmode archive --txlookuplimit=0 --cache.preimages`

In celo-blockchain, this is called `gcmode` which refers to the concept of garbage collection. Setting it to `archive` basically turns it off.

Command Line Interface

Once the full node is running, it can serve the Command Line Interface tool `celocli`. For example:

```
bash
$ npm install -g @celo/celocli
...
$ celocli node:syncd
true
$ celocli account:new
...
```

```
# index.md:
```

```
---
title: Celo Protocol
description: Introduction to the Celo protocol, its implementation, and
its relationship to Ethereum.
---
```

```
import PageRef from '@components/PageRef'
```

Celo Protocol

Introduction to the Celo protocol, its implementation, and its relationship to Ethereum.

```
---
```

What is the Celo Protocol?

Celo's blockchain reference implementation is based on go-ethereum, the Go implementation of the Ethereum protocol. The project team is indebted to the Geth community for providing these shoulders to stand on and, while recognizing that Ethereum is an independent project with its own trajectory, hopes to contribute changes where it makes sense to do so.

In addition to the blockchain client, some core components of the Celo protocol are implemented at the smart contract level and even off-chain (e.g. phone number verification via SMS).

Protocol Upgrades

There are a number of substantial changes and additions have been made in service of Celo's product goals, including the following:

- Consensus
- Governance
- Stability Mechanism
- Transactions
- Identity
- Plumo Ultralight Sync

plumo.md:

title: Plumo Ultralight Sync

description: Introduction to Plumo (Ultralight Sync), it's core concepts, architecture, process, and implementation.

Plumo Ultralight Sync

Introduction to Plumo Ultralight Sync, it's core concepts, architecture, process, and implementation.

What is Plumo Ultralight Sync?

Plumo is a zk-SNARK based system that allows mobile and resource constrained nodes on the Celo network to sync to the Celo blockchain faster and with less data. It accomplishes this by using zero-knowledge proofs, which allow the quick verification of the chain syncing computation without having to run it locally.

Background

Celo is a mobile-first blockchain platform with a mission to enable the conditions for prosperity across the world. This means that many of Celo and the Valora wallet's first users will be in emerging markets. In these countries, while mobile phone penetration has significantly increased, data plans and device hardware still lags. This means that for average users, data is a scarce commodity and devices are underpowered in compute relative to the latest and greatest iPhone or Android devices.

If these users are to trustlessly sync and interact with a blockchain network, they'll need to run a light client on devices. However, traditional light clients in proof of work (PoW) blockchains need to download every block header, which grows linearly with new blocks. Over time, this becomes untenable, and even light clients take minutes to sync.

Proof of Stake Light Clients

Celo uses a proof of stake (PoS) consensus system, enabling it to offer a better light client sync experience, right off the bat:

1. First, Celo light clients only need to download epoch block headers, which contain information about the current validator set. The validator set in Celo changes once every epoch, roughly about once per day. This means instead of downloading a block header for every block on the Celo network, they can just download one a day.

2. Second, Celo uses BLS signatures to aggregate validator signatures. Similar to Tendermint consensus, Celo's IBFT consensus mechanism requires a quorum of 2/3 or more validators to sign each block to commit it to the blockchain. Instead of having each of these signatures be separately stored on-chain, Celo uses BLS signatures which can combine them into one signature per block.

These two innovations reduce the time and data required to sync significantly, but over time, sync time continues to increase. One option is to checkpoint the chain ever so frequently so that clients do not need to sync from the genesis block. This certainly alleviates the challenge, but checkpointing requires getting consensus and agreement from the community, which can take time. What if there were a way to "checkpoint" cryptographically?

Enter Plumo

This is where Plumo comes in. Plumo zk-SNARKs allow light clients to quickly verify transitions from epoch X to epoch $X + Y$, where Y is hundreds of epochs. This means that instead of verifying each epoch, light clients can "hop" from SNARK to SNARK until they reach the latest proof. From there, they can download the remaining epoch block headers to sync to the head of the chain.

The Plumo proofs are generated by a prover that can be run by multiple nodes across the Celo network. Proofs are then requested by light clients on the modified light client protocol. Light clients are able to verify proofs using the verification key that is included with the binary.

Using Plumo, light clients can drastically reduce the amount of data and time required to trustlessly sync with the Celo blockchain. But one more step is required for Plumo to be live on the Celo network -- the Plumo MPC setup.

Plumo MPC Setup

In order to generate the prover and verifier keys for the Plumo SNARK circuit, it is necessary to run a multi-party computation (MPC).

An MPC is a cryptographic mechanism for different parties to jointly perform a computation. SNARK circuits require a "trusted setup" where a shared secret is used to generate public parameters that can be used to prove and verify SNARKs. If one person ran this setup, then they could potentially prove incorrect things by exploiting a backdoor in the

circuit. However, with an MPC, this setup process is split amongst tens or hundreds of contributors, and if even one of the participants is honest (keeps their inputs private), then the system will be secure.

In the case of the Plumo Ceremony, this collective computation will be a series of joint actions done by a group of participants from within the Celo community and beyond.

How to Contribute

The Plumo MPC setup is broken up into two phases:

- Phase 1 - Powers of Tau
 - The output of this phase can be used for either Groth16 circuits up to the size of the phase, or universal proving systems such as PLONK and Marlin.
- Phase 2 - Plumo circuit
 - In this phase, the participants contribute to the Plumo circuit keys, which would be used by provers to create proofs and verifiers to verify them.

Phase 1 will take place starting early December 2020, and continue until January 2021. It will consist of multiple rounds of 6-10 contributors each running the Plumo setup for approximately 36 hours. While much of the activity is passive and involves simply running the computation continuously, contributors should not expect to use their machines for other intensive activities through the duration of the setup.

Phase 2 will commence roughly a month after Phase 1.

Contributors will also receive a gift for the time and efforts, and be recognized for their contributions to the Celo network.

Pre-requisites

You can run the contributor software locally or on cloud VMs, but desktop machines are preferable. The minimum machine requirements are detailed below:

Machine requirements:

- CPU model newer than 2016
- Minimum processor requirements:
 - 2.6GHz, 6 cores, 12 threads, OR
 - 2.3GHz, 8 cores, 16 threads, OR
 - 3.6GHz, 4 cores, 4 threads
- Operating system: Linux, macOS, Windows
- Recommended internet connection speed: 10Mbit upload

With these kind of machines, participation in the setup should take around 30 hours, potentially a bit more or much less, depending on your specific hardware and internet connection.

Running the Setup

Phase 1

The instructions below are for the first phase of the Plumo setup, Powers of Tau.

Generate your address

The first step to participating is generating your Celo address in a secure location.

- Obtain the generate binary:
 - Option one - compile the generate binary:
 - Install Rust 1.49 using `rustup install 1.49.0`. If you don't have rustup installed, follow the instructions in <https://rustup.rs/>. If you're using an Apple M1 machine, install the beta version of Rust using `rustup install beta`.
 - `git clone https://github.com/celo-org/snark-setup-operator`
 - `cd snark-setup-operator`
 - `git checkout ac3d5603256fc250500e00bae21ba646dd316d6f`
 - `cargo build --release --bin generate`
 - Now you can use `./target/release/generate`
 - Option two - if you prefer using a precompiled binary, download the generate binary corresponding to your OS from [here](#).
- Use `b2sum` to check the hash of the binary against the hash in the download page by running `b2sum FILE`, where `FILE` is the contribute binary name for your OS. If you don't have `b2sum` installed, you can download it from <https://github.com/dchest/b2sum>, for example.
- Run it in a command line - navigate in the command line to the relevant folder - and follow the instructions.
 - When asked to Enter some entropy for your Plumo seed:, you can use any source of entropy.
 - Make sure to save your passphrase - you will need it later.
 - If you are using a USB thumbdrive, you can use the flag `--keys-file KEYSFILE`, passing in the respective file path to save the key on the drive itself.
- Send the address generated to `plumo@celo.org`.
- Keep the resulting `plumo.keys` file. Ideally, it will be stored in a location that's easy to destroy afterwards, such as a USB thumbdrive.

Run the Contributor software

Next you'll obtain the contribute binary and begin contributing to the Plumo setup.

cLabs is running the coordinator server, which has a list of approved participant addresses. Additionally, cLabs is running a few verifiers that verify contributions on-demand, allowing the setup to make progress.

- After receiving confirmation from cLabs, obtain the contribute binary:
 - Option one - compile the contribute binary:
 - Install Rust 1.49 using `rustup install 1.49.0`. If you don't have rustup installed, follow the instructions in <https://rustup.rs/>. If

you're using an Apple M1 machine, install the beta version of Rust using `rustup install beta`.

- `git clone https://github.com/celo-org/snark-setup-operator`
- `cd snark-setup-operator`
- `git checkout ac3d5603256fc250500e00bae21ba646dd316d6f`
- `cargo build --release --bin contribute`
- Now you can use `./target/release/contribute`
- Option two - if you prefer using a precompiled binary, download the contribute binary corresponding to your OS from [here](#).
- Use `b2sum` to check the hash of the binary against the hash in the download page by running `b2sum FILE`, where `FILE` is the contribute binary name for your OS. If you don't have `b2sum` installed, you can download it from <https://github.com/dchest/b2sum>, for example.
- Run it as follows, assuming your keys reside in `KEYSFILE`:

- Windows users: `./contribute-windows.exe --keys-file KEYSFILE`
- macOS users: `./contribute-macos --keys-file KEYSFILE`
- Linux users: `./contribute-linux --keys-file KEYSFILE`

For example, assuming you're using Linux and your keys reside in `/mnt/plumo/plumo.keys`, you'd run: `./contribute-linux --keys-file /mnt/plumo/plumo.keys`.

- You will be asked for your passphrase - enter the same one from earlier.
- Follow the same process from earlier when prompted for additional entropy.
- Wait until you see 0/256 on the progress bar. This means that your contribution has started, and you are successfully running the contributor binary.

Once this is running, you can leave the machine running -- no direct action is needed. This will run for about 36 hours, after which the software will terminate running and you will have finished contributing to the Plumo setup!

Phase 2

The instructions below are for the second phase of the Plumo setup, the Plumo circuit.

Generate your address

The first step to participating is generating your Celo address in a secure location.

- Obtain the generate binary:
 - Option one - compile the generate binary:
 - Install Rust 1.52 using `rustup install 1.52.1`. If you don't have `rustup` installed, follow the instructions in <https://rustup.rs/>. If you're using an Apple M1 machine, install the beta version of Rust using `rustup install beta`.
 - `git clone https://github.com/celo-org/snark-setup-operator`
 - `cd snark-setup-operator`

- git checkout 33717c3b0869c605e6c3627446e916f268712e00
- cargo build --release --bin generate
- Now you can use ./target/release/generate
- Option two - if you prefer using a precompiled binary, download the generate binary corresponding to your OS from here.
- Use b2sum to check the hash of the binary against the hash in the download page by running b2sum FILE, where FILE is the contribute binary name for your OS. If you don't have b2sum installed, you can download it from <https://github.com/dchest/b2sum>, for example.
- Run it in a command line - navigate in the command line to the relevant folder - and follow the instructions.
 - When asked to Enter some entropy for your Plumo seed:, you can use any source of entropy.
 - Make sure to save your passphrase - you will need it later.
 - If you are using a USB thumbdrive, you can use the flag --keys-file KEYSFILE, passing in the respective file path to save the key on the drive itself.
- Send the address generated to plumo@celo.org.
- Keep the resulting plumo.keys file. Ideally, it will be stored in a location that's easy to destroy afterwards, such as a USB thumbdrive.

Run the Contributor software

Next you'll obtain the contribute binary and begin contributing to the Plumo setup.

cLabs is running the coordinator server, which has a list of approved participant addresses. Additionally, cLabs is running a few verifiers that verify contributions on-demand, allowing the setup to make progress.

- After receiving confirmation from cLabs, obtain the contribute binary:
 - Option one - compile the contribute binary:
 - Install Rust 1.52 using rustup install 1.52.1. If you don't have rustup installed, follow the instructions in <https://rustup.rs/>. If you're using an Apple M1 machine, install the beta version of Rust using rustup install beta.
 - git clone <https://github.com/celo-org/snark-setup-operator>
 - cd snark-setup-operator
 - git checkout 33717c3b0869c605e6c3627446e916f268712e00
 - cargo build --release --bin contribute
 - Now you can use ./target/release/contribute
 - Option two - if you prefer using a precompiled binary, download the contribute binary corresponding to your OS from here.
 - Use b2sum to check the hash of the binary against the hash in the download page by running b2sum FILE, where FILE is the contribute binary name for your OS. If you don't have b2sum installed, you can download it from <https://github.com/dchest/b2sum>, for example.
 - Run it as follows, assuming your keys reside in KEYSFILE:
 - Windows users: ./contribute-windows.exe --keys-file KEYSFILE
 - macOS users: ./contribute-macos --keys-file KEYSFILE
 - Linux users: ./contribute-linux --keys-file KEYSFILE

For example, assuming you're using Linux and your keys reside in `/mnt/plumo/plumo.keys`, you'd run: `./contribute-linux --keys-file /mnt/plumo/plumo.keys`.

- You will be asked for your passphrase - enter the same one from earlier.
- Follow the same process from earlier when prompted for additional entropy.
- Wait until you see 0/256 on the progress bar. This means that your contribution has started, and you are successfully running the contributor binary.

Once this is running, you can leave the machine running -- no direct action is needed. This will run for about 36 hours, after which the software will terminate running and you will have finished contributing to the Plumo setup!

Publish your attestation

The contribute binary produces a `plumo.attestation` file that contains a signature with your address. After your participation finishes (and not before!), please post it (here) [<https://github.com/celo-org/plumo-ceremony-attestations>] by creating a new issue! If you use precompiled binaries, be sure to mention you've verified the binary hashes posted on the releases page match the downloaded files.

If you don't have access to the `plumo.attestation` anymore, that's still OK, you can fill in the other details in the issue.

Troubleshooting

This section contains some common issues contributors may run into while running the setup software. If you have any questions, please reach out on the `#plumo` channel in the Celo Discord server. It is recommended that you have another device to be able to reach out in case of persistent issues.

Permissions:

- You may need to change the permissions on both the generate and contribute files to be able to run them. For example, on macOS, you can run `chmod u+x generate-macos contribute-macos`.
- On Windows, you might be presented with a warning that this program is from an unsigned developer. Click "run anyway" to continue.
- On macOS, you might be shown a "permission denied" error. This is because it's a downloaded file and is by default in quarantine. You can remove it from quarantine by running `xattr -d com.apple.quarantine contribute-macos`. See here for more details: <https://superuser.com/questions/526920/how-to-remove-quarantine-from-file-permissions-in-os-x>.

Hardware:

- Disable sleep - if your machine is automatically set to sleep or turn hard disks off, it's best to switch these options off.
- Battery - it's best to connect the machine to an outlet, as the setup is computationally intensive and would drain a battery fast.
- Frozen machine - if your computer freezes, don't worry -- the Plumo setup software is using all the threads on the CPU, and it should resolve itself.

If either of these happen, don't worry - stop and restart the contribution program and re-enter the passphrase to continue.

Network:

- If you have issues uploading your contributions, it could be that you have some other processes that are consuming your bandwidth. Make sure to quit any processes such as:
 - A Nest or home camera uploading feeds
 - Torrenting / seeding any files
 - Backup services

Additional Resources

- Plumo Paper
- Plumo Celo Improvement Proposal
- Zero Knowledge Summit Presentation on Plumo
- Transcript of Plumo presentation at Stanford Blockchain Conference
- Zero Knowledge Summit Presentation on the Plumo setup system, Espero

randomness.md:

```
---
title: Celo Randomness
description: How unpredictable pseudo-randomness is achieved on the Celo
blockchain.
---
```

Randomness

How unpredictable pseudo-randomness is achieved on the Celo blockchain and offered as a service for dapp developers.

Producing Pseudo-randomness

Producing unpredictable pseudo-randomness without a trusted third party is not trivial. Several solutions for this problem exist or are being currently researched. They include Verifiable Random Functions \ (for example, based on BLS threshold signatures\), Verifiable Delay Functions, and commit-reveal schemes.

Currently, Celo implements a simple RANDAO commit-reveal scheme which is secure enough for many uses, offering validators only 1 bit of influence:

a validator can affect randomness only by choosing not to propose a block, which results in the next validator revealing their pre-committed randomness. A more sophisticated solution might be implemented as the network evolves, especially if randomness becomes necessary for other purposes that require stronger assumptions about the randomness's security \((for example if it was decided that a randomized leader election algorithm should replace the current round robin)\).

In a proposed block, the proposer attaches two values related to the randomness scheme - randomness corresponding to their previous commitment, and a new commitment to freshly generated random bytes that will be revealed in the future. The revealed randomness is added to an entropy pool accessible on-chain from the Random smart contract.

Randomness Equation

More formally, the s_n block proposed by a given validator contains values (r_n, s_n) such that $\text{keccak256}(r_n) = s_{n-1}$. The one exception to this is the validator's first block, the case where $s_n = 1$, since they have not previously committed to randomness yet. Here, the protocol instead requires that $r_1 = 1$.

Using Onchain Randomness

This randomness can be used by any smart contracts deployed to a Celo network using the Random core contract, e.g.:

```
solidity
import "celo-monorepo/packages/protocol/identity/interfaces/IRandom.sol";
import "celo-monorepo/packages/protocol/common/interfaces/IRegistry.sol";

contract Example {
    function test() external view returns (bytes32 randomness) {
        randomness = IRandom(
            IRegistry(0x0000000000000000000000000000000000000000000000000000000000000000ce10)
                .getAddressFor(keccak256(abi.encodePacked("Random")))
        ).random();
    }
}
```

Alternatively, through inheritance of UsingRegistry.

```
solidity
import "celo-monorepo/packages/protocol/common/UsingRegistryV2.sol";

contract Example is UsingRegistryV2 {
    function test() external view returns (bytes32 randomness) {
        randomness = getRandom().random();
    }
}
```


index.md:

title: Celo Bridges

description: How to bridge from Ethereum, Polygon, and Solana to Celo.

Celo Bridges

How to bridge assets from Ethereum, Polygon, Solana, and others to Celo.

:::warning

Be sure you understand and review the risks pages when bridging assets between chains.

:::

Token bridges

- SmolRefuel (Gassless Bridging)
- Squid Router V2
- Portal (Wormhole)
- AllBridge
- Satellite (Axelar)
- O3
- Transporter (Chainlink CCIP)

Cross-Chain Messaging

- Chainlink CCIP
- Hyperlane
- Wormhole
- Layer Zero
- Axelar Network

index.md:

title: Consensus Introduction

description: Overview of Celo's consensus protocol and network validators.

Consensus

Overview of Celo's consensus protocol and network validators.

Protocol

Celo's consensus protocol is based on an implementation called Istanbul, or IBFT. IBFT was developed by AMIS and proposed as an extension to go-ethereum but never merged. Variants of IBFT exist in both the Quorum and Pantheon clients. We've modified Istanbul to bring it up to date with the latest go-ethereum releases and we're fixing correctness and liveness issues and improving its scalability and security.

Finality

Blocks in IBFT protocol are final, which means that there are no forks and any valid block must be somewhere in the main chain. The only way to revert a block would be to utilise social coordination to get all participants to manually revert the block.

Validators

Celo's consensus protocol is performed by nodes that are selected as validators. There is a maximum cap on the number of active validators that can be changed by governance proposal, which is currently set at 110 validators. The active validator set is determined via the proof-of-stake process and is updated at the end of each epoch, a fixed period of approximately one day.

locating-nodes.md:

```
---
title: Locating Nodes
description: How Celo nodes join the network, establish a connection, and
communiante their IP address.
---
```

Locating Nodes

How Celo nodes join the network, establish a connection, and communiante their IP address.

V4 Discovery Protocol

All Celo nodes \ (including our validators\) are using a variant of Ethereum's V4 discovery protocol to find other nodes within the network. Details of Ethereum's protocol can be found [here](#).

Joining the Network

When a node attempts to join the network, it will execute Celo's discovery protocol.

It will first send a request to the bootnodes to retrieve a list of other nodes of the network. The bootnodes will then reply with that list, and then the joining node will then send additional requests to nodes in that

list to find additional nodes in the network. The main difference in Celo's discovery protocol compared to Ethereum's is that it will require that the joining node's networkID be the same as the bootnodes' \ (and the same as all other network's nodes\).

Also, all of the messages in Celo's discovery protocol must be hashed with a special salt to be accepted by other nodes. The reason why these changes were made is so that each node within a network will only store information of other nodes that have the same networkID (to distinguish nodes from other networks) and the same special salt \ (to distinguish nodes from other blockchains, such as Ethereum\).

Establishing a Connection

Once a joining node finds other nodes, it will establish direct TCP connections to a subset of them. This will allow that node to sync it's blockchain and transactions. Validators will additionally attempt to establish TCP connections to the rest of the validators, so that it can send consensus messages directly to them, instead of via gossip. The reason that the validators do this is to minimize the latency of messages that are sent and received among the validators, and to ultimately help minimize block time.

Communicating IP Address

The way that validators communicate their IP address to other validators is by periodically gossiping a subprotocol message that we call an IstanbulAnnounce message.

That message will contain n copies (where n is the total number of validators for the current epoch) of the sending validator's IP address where each copy is encrypted with the other validators' public key. Once a validator receives a gossiped IstanbulAnnounce message, it will decrypt the encrypted IP address that was encrypted with its public key, and then establish a TCP connection to it. All consensus related messages will then sent via those direct TCP connections.

When an epoch ends, a validator will establish new connections with any newly elected validator and disconnect from any removed validators. If the validator itself is removed from the new epoch's validator set, then it will disconnect with all the validators.

ultralight-sync.md:

title: Celo Ultralight Sync

description: Introduction to Celo's ultralight sync mode and how it improves the speed of the Celo network.

Ultralight Sync

Introduction to Celo's ultralight sync mode and how it improves the speed of the Celo network.

Introduction to Ultralight Sync

In addition to the full, fast, and light sync modes supported by Ethereum, Celo supports an ultralight sync mode. Ultralight nodes compute the validator set for the current epoch by downloading the last header of each previous epoch and applying the validator set diff. They then download the latest block header, which can be verified by checking that at least two-thirds of the validator set for the current epoch signed the block header.

Ultralight Sync Speed

Ultralight nodes download approximately 17,000 times fewer headers than light nodes in order to sync the latest block on Celo mainnet with 5-second block periods and 1-day epochs.

:::tip note

In the future, Celo will support zk-SNARK-based proofs of the ultralight sync mode called Plumo, which will lower the sync time even more.

:::

validator-set-differences.md:

title: Validator Set Differences

description: How validator sets are elected and managed with the Celo protocol.

Validator Set Differences

How validator sets are elected and managed with the Celo protocol.

Computing Set Differences

The validator set for a given epoch is elected at the end of the last block of the previous epoch. The new validator set is written to the extradata field of the header for this block. As an optimization, the validator set is encoded as the difference between the new and previous validator sets. Nodes that join the network are able to compute the validator set for the current epoch by starting with the initial validator set \(\text{encoded in the genesis block}\) and iteratively applying these diffs.

add-contract.md:

title: Add a contract in celo-monorepo

description: How to set up Unit/Migration tests on Celo

Adding a contract in celo-monorepo

Set up a unit/migration test suit for the contract you just created in celo-monorepo and a short guide to running it successfully on celo test net. We'll be using Accounts.sol as an example.

After initial contract creation

After the contract is created and it's ready to be tested, run yarn build to trigger typechain which is essentially a TS wrapper for the contract. Keep in mind that everytime you change your contract you have to run yarn build once again.

Unit tests

The test directory is organized the same way as the contracts directory so feel free to navigate to the parent folder of your currently created contract and create a corresponding(.ts) file for it. For example: celo-monorepo/packages/protocol/contracts/common/Accounts.sol → celo-monorepo/packages/protocol/test/common/accounts.ts.

:::tip

Some build issues can be resolved by simply deleting the build and the typechain folder. Don't forget to run yarn build once again.

:::

create-proposal.md:

title: Create Proposal for Governance

description: How to Create a Proposal in Celo Governance- A Detailed Guide

Prerequisites

Before diving into the Celo governance process and creating a proposal, there are several prerequisites that need to be met:

1. Celo CLI Knowledge: Familiarity with the Celo Command Line Interface (CLI) is crucial. The CLI is the primary tool for interacting with the Celo network, including proposal submission.

2. Minimum Amount of CELO Needed: Proposers must have at least 100 CELO tokens. A token requirement ensures that proposers have a stake in the network and helps to prevent spam proposals. The staked CELO will be refunded to the proposer if the proposal reaches the Approval stage. If a proposal has been on the queue for more than 4 weeks, it expires and the deposit is forfeited.

3. Multi-Signature Wallet: If the proposal requests funds from the treasury, receipt of the funds into a multisig wallet is advisable. Multisig signers are also advised to self-identify on the corresponding Forum post. Multisig wallets add a layer of security and trust, as multiple parties must agree to execute transactions. Multiple parties self-identifying their involvement in a wallet also demonstrates more oversight over the usage of any requested funds.

Life Cycle of a Proposal on Celo

Step 1: Drafting the Proposal

The initial phase in the lifecycle of a governance proposal is the drafting stage. Here, you must comprehensively outline the proposal's purpose, scope, and impact. This should include:

- Objective: Clearly state what the proposal aims to achieve.
- Rationale: Explain why this proposal is necessary and the problems it addresses.
- Technical Specifications: If applicable, provide technical details or code changes.
- Budget and Funding: Outline any financial requirements, including a detailed breakdown of costs.

Setting up a secure multisig wallet is recommended for proposals requesting funds, as it ensures enhanced security and trust within the community.

Step 2: Posting the Proposal on Celo Forum

Once your proposal is drafted, post it on forum.celo.org to initiate community discussion. This post should:

- Detail the Proposal: Share every aspect of the proposal, leaving no ambiguity.
- Include Multisig Information: If requesting funds, provide the multisig wallet details.
- Solicit Feedback: Encourage community input to refine and improve the proposal.
- Respond and Iterate: Actively engage with the community, addressing queries and incorporating feedback to strengthen the proposal.

Step 3: Applying for the Governance Community Call

To further socialize your proposal, apply to present it during the Celo Governance Community Call by:

- Booking a Slot: Comment on the GitHub governance repo issue for the next upcoming Governance call to reserve your presentation slot.
- Join a Governance Call Discussion: Discuss your proposal in depth and answer questions from the community and approvers.

Step 4: Publishing the Proposal on GitHub

After refining your proposal through community feedback, the next step is to formalize it by publishing on GitHub. This involves:

- Create a Pull Request (PR): Submit your proposal as a PR to the celo-org/governance repository. This should include the proposal in markdown format and any associated code in a JSON file. Review previous proposal that are similar to your proposal for hints on how the code should be structured.
- Celo Governance Proposal (CGP) Editors' Review: CGP editors will review your submission to ensure it meets the required standards and provide feedback or request changes if necessary. The CGP Editors will assign an id.
- Acceptance of PR: Once the CGP editors approve the PR, it signals that your proposal is ready to be submitted on-chain.
- On-chain Submission: Accepted proposals can then be submitted on-chain through the Celo CLI for formal voting by the community.

:::danger

Wait for PRs to be merged on proposals prior to submitting on chain

:::

Step 5: Formal Proposal Submission On-Chain

The formal submission of your proposal to the blockchain will be covered in a separate section but involves using celocli to submit your proposal on-chain for approval and voting.

:::info

Additionally, when submitting on-chain: the CGP markdown title will not update on celo.stake.id. While the body of the document can be edited if necessary, it is best to ensure all details are correct before submission.

:::

Step 6: Voting

After submission, your proposal enters the voting phase, which consists of:

- Upvoting: If multiple proposals are submitted at the same time in a 24 hour period, then they must receive upvotes. Anyone, including yourself, can upvote proposals. The highest upvoted proposal moves automatically to the Referendum stage.

- Approval Voting: Approvers review the proposal for any security risks or potential harm to the network. They may follow up with questions so ensure you are easy to contact via your profile on forum or via a connection with a CGP Editor.
- Referendum Voting: Once a proposal is in Referendum stage, all CELO holders may vote on the proposal. They can vote "Yes" to support the proposal, "No" to reject the proposal, and "Abstain" to acknowledge but defer the vote to the remaining community.
- Quorum: For a proposal to pass successfully, the total number of CELO voted must meet or exceed Quorum. "Yes", "No", and "Abstain" votes all count towards quorum. A successfully proposal must receive a 60% majority of votes above quorum. Quorum needed for proposals is dynamic and depends on previous number of votes on recent proposals.
- Conclusion: Approvers have until the proposal's deadline to approve any passing proposals. Once approved, "Yes" votes exceed 60% of necessary quorum, then they may be Executed.

Step 7: Execution

If the proposal passes the voting phase, it moves to execution. This involves enacting the changes or transferring the funds as outlined in the proposal. Detailed steps for execution will be provided in a subsequent section.

By following these steps, your proposal will go through the necessary stages from conception to execution in the Celo governance process. Proposals must be executed within 3 days from the referendum stage or they will be rejected automatically by the system.

:::info

Warning: Anyone on the network can execute a successful proposal at any time.

:::

Creating On-Chain Proposal

Step 1: Create the Proposal File (mainnet.json)

In your proposal file in Governance repository that is either merged or waiting to be merged, create a folder with your proposal number inside the CGPs folder and create a file called mainnet.json. Check this for example -

Inside this file, paste the following content:

```
json
[
  {
    "contract": "GoldToken",
    "function": "approve",
    "args": [
      "0xE1061b397cC3C381E95a411967e3F053A7c50E70",
```



```

    "5980314000000000000000000000"
  ],
  "value": "0"
}
]

```

:::info

💡 If requesting funds from Treasury

:::

```

json
{
  "contract": "StableToken",
  "address": "0x765DE816845861e75A25fCA122bb6898B8B1282a",
  "function": "increaseAllowance",
  "args": [
    "0x71f433514957d00287A9d33Da759f1e0C1732381",
    "17000000000000000000000000000000"
  ],
  "value": "0"
}

```

:::info

💡 If making contract call from Governance Contract

:::

Make sure to replace your address and the amount of CELO you want to approve. Save this file and add it to CGPs folder in the Governance repository.

Step 2: Submit the Proposal On-Chain

After your PR is merged, we can submit the proposal on-chain.

Step 2.1: Install celocli

In your terminal, run the following command to install the Celo CLI:

```

bash
npm install -g @celo/celocli

```

Step 2.2: Target the json file

We will submit the proposal using the mainnet.json file you created earlier. To do this, in your terminal:

```

bash

```

```
cd governance // repository folder
cd CGPs
cd cgp-(your proposal number)
cat mainnet.json
```

Once you see the content of your mainnet.json file in the terminal output, you can submit the proposal using:

```
bash
celocli governance:propose --jsonTransactions=mainnet.json --
deposit=10000e18 --descriptionURL=https://github.com/celo-
org/governance/blob/main/CGPs/cgp-<YOURPROPOSALIDINGITHUB>.md --
from=<YOURADDRESS> --privateKey=<PRIVATEKEY>
```

Replace the --descriptionURL, --from fields with your proposal Github file URL.

```
:::info
```

! Note that 10,000 CELO tokens are required in the account to submit a proposal. This amount will be refunded to the proposer if the proposal reaches the Approval stage. If a proposal has been on the queue for more than 4 weeks, it expires and the deposit is forfeited.

```
:::
```

```
:::info
```

! You can see your proposal ID in the terminal output. Save this ID for future use. To see your proposal in detail, run the following command in your terminal: celocli governance:show --proposalID <number>

```
:::
```

Step 3: Execute the Proposal

Once the proposal passes the series of votes, you need to execute it. Connect your ledger to your computer and run the following command in your terminal:

```
bash
celocli governance:execute --proposalID <number> --from=<address> --
privateKey=<PRIVATEKEY>
```

Replace number with the proposal ID.

Best Practices for Creating a Proposal

- **Clarity and Justification:** Ensure your proposal is clearly written, with straightforward language and a strong justification for why it's

needed. Provide as much detail as possible about what you are proposing and why it is beneficial for the Celo ecosystem.

- Community Engagement: Engage with the community early on. Seek feedback and address concerns before formal submission. This not only improves your proposal but also builds community support.
- Security Assessment: If your proposal involves smart contract code, conduct a thorough security audit. Include the audit report in your proposal to increase credibility and trust.
- Transparency: Be transparent about your affiliations and intentions. If your proposal involves funding, detail how the funds will be used.
- Follow Governance Structure: Adhere strictly to the governance process as laid out by Celo, including any templates or formats required for proposals.

What to Expect in Governance Calls

- Presentation Slot: Be prepared to present your proposal succinctly. Typically, you may be allocated a specific time slot, such as 5 minutes for presentation and 5 minutes for Q&A.
- Technical Questions: Expect technical questions from the community, especially if your proposal involves code changes. Be ready to explain complex concepts in accessible language.
- Community Feedback: Governance calls are an opportunity for the community to provide direct feedback. Take notes and be open to incorporating this feedback into your proposal.
- Approval Indicators: Use these calls as a temperature check on your proposal's likelihood of passing. Positive engagement and constructive feedback are good indicators.
- Networking: These calls are an excellent opportunity to network with other members of the Celo ecosystem, which can be valuable for building future support.

Each of these elements is important for navigating the governance calls effectively and maximizing the chance of your proposal's success.

FAQ Section for Celo Governance

1. What is a multisig wallet, and why is it recommended for proposals?
 - A multisig wallet requires multiple signatures to authorize transactions, providing increased security and trust for proposals involving treasury funds.
2. Who are the Governance approvers?
 - Governance approvers are individuals or entities with the authority to review and approve proposals before they go to a community vote, ensuring they meet certain criteria and standards.
3. Who are the CGP editors?
 - CGP (Celo Governance Proposal) editors are responsible for reviewing and managing the content of governance proposals submitted on GitHub to ensure clarity, completeness, and adherence to the format.
4. How to reach CGP editors?
 - To reach CGP editors, you can use the Celo Discord channel. They are available to assist with questions and provide guidance on the proposal process.
5. Can I submit the proposal again?

- If a proposal is rejected or needs significant revisions, it may be resubmitted after addressing the community's feedback and making necessary adjustments.

6. What if I made a mistake in the proposal?

- If you discover a mistake in your proposal, it's important to communicate this to the community and CGP editors as soon as possible. Depending on the stage of the proposal, you may need to withdraw and resubmit it with corrections.

How to create Multisig with Safe and withdraw fund from Treasury?

Step: 1 Log in to your Safe

Log in and click New Transaction on the left hand side of the page. This will bring up a modal, click Contract Interaction.

> Note: Don't use <https://safe.celo.org/> as it is not officially supported. Use official Safe at - <https://safe.global/>

!Governance Safe Creation process step 1

Step 2: Enter ABI

Enter the Celo Proxy Address.

!Governance Safe Creation process step 2

Step 3. Change the ABI to the CELO Contract

Step 2 will auto populate the ABI with the proxy ABI. We will want to change that to the actual CELO ERC20 ABI to access the transferFrom function

> For the image below

>

> 1. This is the Celo Token Proxy

Address(0x471ece3750da237f93b8e339c536989b8978a438)

> 2. The ABI of the CELO Token Contract

> 3. This is where our contract interaction is being sent to and this will be the CELO Token proxy.

> 4. The method we want to interact with(We specify the transferFrom)

!Governance Safe Creation process step 3

Step 4 Fill in the the appropriate addresses

You will now need to add the addresses you are wishing to interact with

> For the image below:

>

>

> 5. from(address): This is the Celo Governance

Contract(0xD533Ca259b330c7A88f74E000a3FaEa2d63B7972)

> 6. to(address): This is the address you are wishing to transfer funds from the Governance contract to. It could be the multisig you are using to do this interaction from or any other address.
> value(uint256): The wei amount of funds you are wishing to transfer. eg
1 CELO = 10 18
> you can use this webpage to easily convert CELO to the wei value

!Governance Safe Creation process step 4

Step 5: Create Transaction

Click Add Transaction, and after that other multisig signers will need to confirm the transaction. Once the required number of signers have confirmed the transaction, it will be executed. You can also add a description to the transaction to help other signers understand the purpose of the transaction.

index.md:

title: Celo Governance

description: Overview of Celo governance and how the network is managed using the stakeholder proposal process.

Governance

Overview of Celo governance and how the network is managed using the stakeholder proposal process.

What is Celo Governance?

Celo uses a formal on-chain governance mechanism to manage and upgrade the protocol such as for upgrading smart contracts, adding new stable currencies, or modifying the reserve target asset allocation. All changes must be agreed upon by CELO holders. A quorum threshold model is used to determine the number of votes needed for a proposal to pass.

Stakeholder Proposal Process

Changes are managed via the Celo Governance smart contract. This contract acts as an "owner" for making modifications to other protocol smart contracts. Such smart contracts are termed governable. The Governance contract itself is governable, and owned by itself.

The change procedure happens in the following phases:

1. Proposal
2. Approval
3. Referendum
4. Execution

Overview of Phases

Each proposal starts on the Proposal Queue where it may receive upvotes to move forward in the queue relative to other queued proposals. Proposal authors should work to find community members to upvote their proposal (proposers may also upvote their proposals). Up to three proposals from the top of the queue are dequeued and promoted to the approval stage automatically per day. Any proposal that remains in the queue for 4 weeks will expire.

- Approval lasts 1 days (24 hours), during which the proposal must be approved by the Approver(s). Approved proposals are promoted to the Referendum stage.
- Referendum lasts five days, during which owners of Locked Celo vote yes or no on the proposal. Proposals that satisfy the necessary quorum are promoted to the execution phase.
- Execution lasts up to three days, during which anybody may trigger the execution of the proposal.

Proposal

Any user may submit a Proposal to the Governance smart contract, along with a small deposit of CEL0. This deposit is required to avoid spam proposals, and is refunded to the proposer if the proposal reaches the Approval stage. A Proposal consists of a list of transactions, and a description URL where voters can get more information about the proposal. It is encouraged that this description URL points to a CGP document in the celo-org/celo-proposals repository. Transaction data in the proposal includes the destination address, data, and value. If the proposal passes, the included transactions will be executed by the Governance contract.

Submitted proposals are added to the queue of proposals. While a proposal is on this queue, voters may use their Locked Celo to upvote the proposal. Once per day the top three proposals, by weight of the Locked Celo upvoting them, are dequeued and moved into the Approval phase. Note that if there are fewer than three proposals on the queue, all may be dequeued even if they have no upvotes. If a proposal has been on the queue for more than 4 weeks, it expires and the deposit is forfeited.

Approval

Every day the top three proposals at the head of the queue are popped off and move to the Approval phase. At this time, the original proposers are eligible to reclaim their Locked Celo deposit. In this phase, the proposal needs to be approved by the Approver. The Approver is initially a 3 of 9 multi-signature address held by individuals selected by the Celo Foundation, and will move to a DAO in the future. The Approval phase lasts 1 day and if the proposal is not approved in this window, it is considered expired and does not move on to the "Referendum" phase.

Referendum

Once the Approval phase is over, approved proposals graduate to the referendum phase. Any user may vote yes, no, or abstain on these proposals. Their vote's weight is determined by the weight of their Locked Celo. After the Referendum phase is over, which lasts five days, each proposal is marked as passed or failed as a function of the votes and the corresponding passing function parameters.

In order for a proposal to pass, it must meet a minimum threshold for participation, and agreement:

- Participation is the minimum portion of Locked Celo which must cast a vote for a proposal to pass. It exists to prevent proposals passing with very low participation. The participation requirement is calculated as a governable portion of the participation baseline, which is an exponential moving average of final participation in past governance proposals.
- Agreement is the portion of votes cast that must be "yes" votes for a proposal to pass. Each contract and function can define a required level of agreement, and the required agreement for a proposal is the maximum requirement among its constituent transactions.

Execution

Proposals that graduate from the Referendum phase to the Execution phase may be executed by anyone, triggering a call operation code with the arguments defined in the proposal, originating from the Governance smart contract. Proposals expire from this phase after three days.

Smart Contract Upgradeability

Smart contracts deployed to an EVM blockchain like Celo are immutable. To allow for improvements, new features, and bug fixes, the Celo codebase uses the Proxy Upgrade Pattern. All of the core contracts owned by Governance are proxied. Thus, a smart contract implementation can be upgraded using the standard on-chain governance process.

Upgrade risks

The core contracts define critical behavior of the Celo network such as CELO and Celo Dollar asset management or validator elections and rewards. Malicious or inadvertent contract bugs could compromise user balances or potentially cause harm, irreversible without a blockchain hard fork.

Great care must be taken to ensure that any Governance proposal that modifies smart contract code will not break the existing system. To this end, the contracts have a well defined release process, which includes soliciting security audits from reputable third-party auditors.

As Celo is a decentralized network, all Celo network participants are invited to participate in the governance proposals discussions on the forum.

Validator Hotfix Process

The cadence and transparency of the standard on-chain governance protocol make it poorly suited for proposals that patch issues that may compromise the security of the network, especially when the patch would reveal an exploitable bug in one of the core contracts. Instead, these sorts of changes are better suited for the more responsive, and less transparent, hotfix protocol.

Anyone can make a proposal in the hotfix protocol by submitting the hash of their proposal to the Governance smart contract. If that hash is approved by the approver and a quorum of validators, the proposer can execute the contents of that proposal immediately.

Note that this means the validators may not always know the contents of the proposal that they are voting on. Revealing the contents of the proposal to all validators may compromise the integrity of the hotfix protocol, as only one validator would need to be malicious in order to exploit the vulnerability or share it publicly. Instead, to convince the validators that the hash represents a proposal that should be executed via the hotfix protocol, the proposer should consider contacting reputable, third party, security firms to publicly vouch for the contents of the proposal.

Celo Blockchain Software Upgrades

Some changes cannot be made through the on-chain governance process (via proposal or hotfix) alone. Examples include changes to the underlying consensus protocol and changes which would result in a hard-fork.

encrypted-cloud-backup.md:

title: PEAR 🍓

Pin/Password Encrypted Account Recovery.

Secure and reliable account key backups are critical to the experience of non-custodial wallets, and Celo more generally.

Day-to-day, users store their account keys on their mobile device, but if they lose their phone, they need a way to recover access to their account.

Described in this document is a protocol for encrypted backups of a user's account keys in their cloud storage account.

Summary

Using built-in support for iOS and Android, mobile apps can save data backups to Apple iCloud and Google Drive respectively.

When a user installs the wallet onto a new device, possibly after losing their old device, or reinstalls the app on the same device, it can check the user's Drive or iCloud account for account backup data. If available, this data can be downloaded and used to initialize the application with the recovered account information.

Access to the user's cloud storage requires logging in to their Google or Apple account.

This provides a measure of security as only the owner of the cloud storage account can see the data, but is not enough to confidently store the wallet's account key.

In order to provide additional security, the account key backup should be encrypted with a secret, namely a PIN or password, that the user has memorized or stored securely.

This way, the users account key backup is only accessible to someone who can access their cloud storage account and knows their secret.

Because user-chosen secrets, especially PINs, are susceptible to guessing, this secret must be hardened before it can be used as an encryption key.

Using ODIS for key hardening, this scheme derives an encryption key for the account key backup that is resistant to guessing attacks.

With these core components, we can construct an account recovery system that allows users who remember their password or PIN, and maintain access to a cloud storage account, to quickly and reliably recover their account while providing solid security guarantees.

Valora is currently working to implement encrypted account recovery, using the user's access PIN for encryption.

Similar protocols

- iCloud Keychain uses 6-digit PIN, hardened by an HSM app, and encrypts iCloud Keychain backups.
- Signal SVR uses a 4-digit PIN or alphanumeric password, hardened by an Intel SGX app, to encrypt contacts and metadata.
- Coinbase Wallet uses a password encrypted cloud backup to store user account keys. It is unclear if any hardening is used.
- WhatsApp E2E Encrypted Backups uses OPAQUE to harden a password encrypted backup
- MixIn Network TIP uses 6-digit PINs, hardened by a set of signers, to derive account keys

User experience

Here we describe the user experience of the protocol as designed. Wallets may alter this flow to suite the needs of their users.

Onboarding

During onboarding on a supported device, after the PIN or password is set and the account key is created, the user should be informed about the

account backup and given a chance opt-out of backup system for their account.

If they opt out, the rest of the setup should be skipped as they will not be using this account recovery system.

On Android, when the user opts-in, they should be prompted to select a Google account that they would like to use to store the backup.

On iOS, the user need not be prompted as there is a single Apple account on the device and the permissions architecture allows access to application-specific iCloud data without prompting the user.

In the background, the chosen PIN or password and a locally generated salt value should be used to query ODIS.

The resulting hardened key should be used to encrypt the BIP-39 account key mnemonic.

The encrypted mnemonic and metadata, including the salt, should be stored in the user's cloud storage.

Recovery

During recovery, the application should determine if a backup is available in their cloud account.

On iOS, this can be done automatically.

On Android, the user may choose to restore from a cloud backup, at which point they should be prompted to choose their Google account.

If a backup is available the user may select to restore from a cloud backup, at which point they should be asked for their PIN or password.

Given the PIN or password, the application should combine it with the salt value and query ODIS to retrieve the hardened key for decrypting the account key backup.

If successful, the user will be sent to the home screen.

If unsuccessful, the user will be given the option to try again or enter their mnemonic phrase instead.

Users should, by requirement of security, be given a limited number of attempts to enter their PIN or password.

Attempts should be rate limited with a certain number of attempts available immediately (e.g. 3-5 attempts within the first 24 hours), and a limited number of additional attempts available after one or more waiting periods (e.g. up to 10-15 attempts over 3 days).

Once all attempts are exhausted, the backup will become unrecoverable and the user will only be able to recover their account if they have their mnemonic phrase written down.

Implementation

Client support for the encrypted backup protocol described here is implemented in the @celo/encrypted-backup package.

Creating a backup file consists of a number of steps to derive the encryption key, and assemble the backup file.

1. Generate a random nonce and hash it with the password or PIN input to get the initial key.
2. Generate a random fuse key and hash it with the initial key to get an updated key.

Encrypt this fuse key to the public key of the circuit breaker service and discard the plaintext fuse key.

3. Send the key as a blinded message to the ODIS to be hashed under a password hardening domain.

Use an authentication key derived from the backup nonce such that only a user with access to the backup can make queries to ODIS.

Hash the response from ODIS together with the key to generate the hardened key.

4. Encrypt the account mnemonic phrase with the hardened encryption key, and assemble it together with the nonce, ODIS domain information, encrypted fuse key, and environment metadata for ODIS and the circuit breaker.

If the implementing service does not wish to include a circuit breaker, which is described in more detail below, step two can be skipped.

The backup file created in this protocol can then be stored by the wallet that implements this protocol in some authenticated storage, such as iCloud or Google Drive.

In order to open the backup and recover the users account mnemonic the encrypted backup file is first retrieved from authenticated storage, then the decryption key is derived in the following steps similar to the steps above.

1. Hash the password or PIN input with the nonce in the backup to get the initial key.
2. Query the circuit breaker to unwrap the encrypted fuse key and hash it with the initial key to get an updated key.
3. Send the key as a blinded message to the ODIS to be hashed under the included password hardening domain.

Use an authentication key derived from the backup nonce.

Hash the response from ODIS together with the key to generate the hardened key.

4. Decrypt the backup data with the hardened decryption key and return it as the account mnemonic.

Circuit breaker

In order to handle the event of an ODIS service compromise, this is protocol includes a recommended circuit breaker service.

A circuit breaker service is essentially an online decryption service with a well-known public key that can be taken offline if needed to prevent access to the decryption key.

By using a fuse key which is decrypted to the circuit breaker service, and therefore can only be accessed if the service is online, as a step to derive the encryption key for the backup, the circuit breaker service operator is able to disable decryption of backup files in case of an emergency to protect user funds.

In particular, if the ODIS key hardening service is discovered to be compromised, the circuit breaker operator will take their service offline, preventing backups using the circuit breaker from being opened. This ensures that an attacker who has compromised ODIS cannot leverage their attack to forcibly open backups created with this function.

PIN Blocklist

When using a 4 or 6 digit PIN code to encrypt a backup, there are a number of PINs that are far more common than others. Sequences (123456), patterns (124578) and important dates (110989) are chosen most frequently.

Within 30 guesses, an attacker has a 5-9% chance of guessing a users first-choice PIN code, as suggested by research into PIN security. In order to address this, it is highly recommended to block the most easily guessed PINs.

One way to do this is to block PINs that are most popular.

A suggested implementation, which is implemented by the Valora wallet, is to create a blocklist from the top 25k most frequently seen PINs in the HIBP Passwords dataset.

index.md:

title: Celo Identity Overview

description: How Celo maps wallet addresses to phone numbers to make financial tools more accessible to mobile phone users.

Identity Overview

How Celo maps wallet addresses to phone numbers to make financial tools more accessible to mobile phone users.

Introduction to Identity on Celo

Celo's unique purpose is to make financial tools accessible to anyone with a mobile phone. One barrier for the usage of many other platforms is their required usage of 30+ hexadecimal-character-long strings as addresses. It's like bank account numbers, but worse. Hard to remember, easy to mess up. They are so hard to use that the predominant way of exchanging addresses is usually via copy-paste over an existing messaging channel or via QR-codes in person. Both approaches are practically interactive protocols and thus do not cover many use cases in which people would like to transact. Celo offers an optional lightweight identity layer that starts with a decentralized mapping of phone numbers to wallet addresses, allowing users to transact with one another via the most common identity scheme everyone is familiar with: their address book.

Adding their phone number to the mapping

To allow Bob to find an address mapped to her phone number, Alice can use the decentralized attestations protocol to link an account address to her phone number. Alice starts by making a request to the Attestations contract; transferring a fee along with her request. After a brief waiting time of 4 blocks (20 seconds), the Attestations contract will use the Random contract to produce a random selection of validators, from the current elected set in the Validators contract, to issue the attestation challenges.

As part of the expectation of validators, they run the attestation service whose endpoint they register in their Metadata. After attestation issuers have been selected for their requests, Alice determine the validators' attestation service URLs from her Metadata and requests an attestation message to her phone number by sending a direct HTTPS request. In turn, the attestation service produces a signed secret message attesting to the ownership of the given phone number by the requesting account. The validator sends the message to Alice's phone number via SMS. Read more under attestation service.

When Alice receives the text message, she can take that signed message to the Attestations contract, which can verify that the attestation came from the validator indeed. Upon a successful attestation, the validator can redeem for the attestation request fee to pay them for the cost of sending the SMS. In the end, we have recorded an attestation by the validator to a mapping of Alice's phone number to her account address.

Using the mapping for payment

Once Alice has completed attestations for their phone number/address, Bob, who has her phone number in his contact book, can see that Alice has an attested account address with her phone number. He can use that address to send funds to Alice, without her having to specifically communicate her address to Bob.

The Attestations contract records all attestations of a phone number to any number of addresses. That for example could happen when a user loses their private key and wants to map a new wallet address. However, it could also happen through the collusion of a validator with Alice. Therefore, it is important that clients of the identity protocol highlight possible conflicting attestations.

Some risk exists for attestations to be added without the permission of the "legitimate" owner of the phone number. One such risk is that the phone service provider or SIM swap attacker could take control of the phone number and complete a number of attestations. Another risk is that a sufficient number of Attestation Service providers may collude to complete fake attestations. Notably, completing malicious attestations does not lead to a loss of funds, as the private key is still the necessary and sufficient condition for transactions of an account.

However, without proper care, future senders may be tricked into sending funds to the newly associated address. In general the number and age of attestations for an address should be taken into account to identify the valid owner of a phone number.

There are additional measures we can take to further secure the integrity of the mapping's usage. In the future we plan to provide reference implementations in the wallet for some of these. For example, we plan to detect remapping of wallet addresses. Many users are already accustomed to sending small amounts first and verifying the receipt of those funds before attempting to transfer larger amounts.

Preventing harvesting of phone numbers

To protect user privacy by preventing mass harvesting of phone numbers, the Celo platform includes a service that obfuscates the information saved on the blockchain. The service is enabled by default for all Celo Wallet users. Details of its functionality and architecture are explained in [Phone Number Privacy](#)

Attestation service

The attestation service is a simple Node.js service that validators run to send signed messages for attestations. It can be configured with SMS providers, as different providers have different characteristics like reliability, trustworthiness and performance in different regions. The attestation service currently supports Twilio and Nexmo. Celo should widen the number of supported providers over time.

<!--

We have been experimenting with a SMS provider that we would like community feedback on. Instead of sending the SMS via conventional providers like Twilio, users of a Rewards Mobile App could register themselves with a Verification Pool and be made responsible for sending those text messages. It would allow users with cheap or leftover SMS capacity from their cell phone plan to effectively acquire a share of the attestation request fees. It would represent a unique on-ramp for users who do not have access to classic on-ramps like exchanges. Validators could configure their attestation service to use such a SMS provider which could in theory provide better inclusion and performance.

-->

Future improvements to privacy

Celo is committed to meet the privacy needs of its users. More details about areas for future research can be found in [Privacy Research](#)

metadata.md:

title: Celo Metadata and Claims

description: How the Celo protocol's metadata and claims feature makes it possible to connect on-chain with off-chain identities.

Metadata and Claims

How the Celo protocol's metadata and claims feature makes it possible to connect on-chain with off-chain identities.

Use Cases

- Tools want to present public metadata supplied by a validator or validator group as part of a list of candidate groups, or a list of current elected validators.
- Governance Explorer UIs may want to present public metadata about the creators of governance proposals
- The Celo Foundation receives notice of a security vulnerability and wants to contact elected validators to facilitate them to make a decision on applying a patch.
- A DApp makes a request to the Celo Wallet for account information or to sign a transaction. The Celo Wallet should provide information about the DApp to allow the user to make a decision whether to sign the transaction or not.

Furthermore, these tools may want to include user chosen information such as names or profile pictures that would be expensive to store on-chain. For this purpose, the Celo protocol supports metadata that allows accounts to make both verifiable as well as non-verifiable claims. The design is described in CIP3.

On the Accounts smart contract, any account can register a URL under which their metadata file is available. The metadata file contains an unordered list of claims, signed by the account.

Types of Claim

ContractKit currently supports the following types of claim:

- Name Claim - An account can claim a human-readable name. This claim is not verifiable.
- Attestation Service URL Claim - For the lightweight identity layer, validators can make a claim under which their Attestation Service is reachable to provide attestations. This claim is not verifiable.
- Keybase User Claim - Accounts can make claims on Keybase usernames. This claim is verifiable by signing a message with the account and hosting it on the publicly accessible path of the Keybase file system.
- Domain Claim - Accounts can make claims on domain names. This claim is verifiable by signing a message with the account and embedding it in a TXT record.

In the future ContractKit may support other types of claim, including:

- Twitter User Claim - Accounts can make claims on Twitter usernames. This claim is verifiable by signing a message with the account and posting it as a tweet. Any client can verify the claim with a reference to the tweet in the claim.

Handling Metadata

You can interact with metadata files easily through the CLI, or in your own scripts, tools or DApps via ContractKit. Most commands require a node being available under <http://localhost:8545> to make view calls, and to modify metadata files, you'll need the relevant account to be unlocked to sign the files.

You can create an empty metadata file with:

```
bash
celocli account:create-metadata ./metadata.json --from $ACCOUNTADDRESS
```

You can add claims with various commands:

```
bash
celocli account:claim-attestation-service-url ./metadata.json --from
$ACCOUNTADDRESS --url $ATTESTATIONSERVICEURL
```

You can display the claims in your file and their status with:

```
bash
celocli account:show-metadata ./metadata.json
```

Once you are satisfied with your claims, you can upload your file to your own web site or a site that will host the file (for example, <https://gist.github.com>) and then register it with the Accounts smart contract by running:

```
bash
celocli account:register-metadata --url $METADATAURL --from
$ACCOUNTADDRESS
```

Then, anyone can lookup your claims and verify them by running:

```
bash
celocli account:get-metadata $ACCOUNTADDRESS
```

```
# odis-domain-sequential-delay-domain.md:
```

```
---
```

```
title: Sequential Delay Domain
```

```
import PageRef from '@components/PageRef'
```

The Sequential Delay Domains is an ODIS Domain supporting signature-authenticated rate limits defined as a series of time-delayed stages. The motivating use case is allowing wallets to define how often users can attempt to recover their account via the scheme outlined in Pin/Password Encrypted Account Recovery, but can be used in any other application that need an authenticated rate limit represented as a series of time delayed stages.

Specification

A full specification of the Sequential Delay Domain is available in an extension to CIP-40.

- Sequential Delay Domain Specification

```
# odis-domain.md:
```

```
title: ODIS Domains
```

```
import PageRef from '@components/PageRef'
```

```
:::tip
```

Domain API features described here are not deployed to Mainnet ODIS as of April 1, 2022.

```
:::
```

In order to support use cases such as password hardening, and future applications, ODIS implements Domains.

A Domain instance is structured message sent to ODIS along with the secret blinded message.

Unlike the blinded message, the Domain instance is visible to the ODIS service and allows the client to specify context information about their request.

This context information is used to decide what rate limit and/or authentication should be applied to the request, and is combined into the result to ensure output is unique to the context.

The Domain instance and blinded message are both passed to the ODIS partially oblivious pseudorandom function (POPRF), which is a new construction extending upon the OPRF function used in the phone number privacy service.

As an example, a Domain for hashing an account password might specify an application username of "vitalik.eth" (context) and a cap of 10 password attempts (rate-limiting parameter).

These would be combined with the user's password (blinded input) in the POPRF, which acts as a one-way function, to form the final output. As a result the rate limiting parameters, in this case allowing a total of 10 queries, can be set to arbitrary values but are effectively binding once chosen.

This allows the parameters to be tuned to the needs of the individual user or application and prevents potential overlap of different use cases.

Queries with distinct domain specifiers will receive uncorrelated output. For example, output from ODIS with the phone number domain and message 18002738255 will be distinct from and unrelated to the output when requesting with a password domain and message 18002738255.

In order to make this scheme flexible, allowing for user-defined tuning of rate-limits and the introduction of new rate limiting and authorization rules in the future, domains are defined as serializable structs.

New domain types, with associated rate-limiting rules, may be added in the future to meet the needs of new applications.

Specification

A full specification of Domains and the related ODIS APIs is available in CIP-40.

- CIP-40

Implemented Domains

- Sequential Delay Domain

Creating a Domain Type

The Domains interface is designed to be flexible to facilitate new applications for the ODIS POPRF function.

If you have an application that would benefit from a new Domain type and rate limiting ruleset, the first step is to open an extension to the CIP-40 standard.

New Domain types are standardized through a lighter version of the general CIP process.

Open a PR against the celo-org/celo-proposals repository to add a specification for your new domain to the CIP-40 extensions folder.

As an example for what you should include, take a look at the specification for the SequentialDelayDomain.

When it is ready for review, contact a CIP editor to help get reviews from the ODIS core development team.

Implementing a new Domain type, which includes new rate limiting to be enforced by the ODIS operators, requires an upgrade to the ODIS server implementation.

Once the new domain type is standardized, this implementation can be written and deployed to the staging and production ODIS service operators.

odis-use-case-key-hardening.md:

title: Key Hardening

Passwords are useful primitive in a number of applications, allowing a user to authenticate themselves by knowing the secret information. Unfortunately, effective offline password cracking techniques limit the use of passwords to derive encryption or authentication keys. An attacker with access to a signature or encrypted file that used a password, or the hash of a password, as the key can make repeated guesses until they find the password. Given advanced tools such as hashcat and extensive experience, hackers are very good at guessing passwords.

Rate-limited or expensive hashing can be used to make it much more difficult to crack a password. Computationally expensive password hashing functions, such as PBKDF and scrypt, are commonly used for this purpose, but provide limited protection and are expensive to run on end-user devices. ODIS implements hashing (i.e. PRF evaluation) with a rate limit controlled by the committee of ODIS operators, and can be used to harden a password into a stronger cryptographic key. As long as this committee remains collectively honest and secure, an attacker cannot make more guesses at a users password than ODIS allows, making it extremely unlikely a good password will be broken.

Using ODIS for key hardening allows passwords to be used in a number of applications, including to create encrypted account backups and as a factor in smart contract account recovery.

Rate limiting

Choosing an appropriately restrictive rate limit is crucial. Using a rate limit that is too restrictive may cause users to become frustrated as their access is denied if they take too many tries to recall their password, and a rate limit that is too loose can allow an attacker a much better chance at guessing the users password. The appropriate rate limit is related to how much entropy the user secret has.

- A strong user password can tolerate a loose rate limit, allowing millions of attempts without significant chance of attacker success.
- An average user password can tolerate a moderate rate limit, allowing hundreds of attempts.
- A 4 or 6 digit PIN can tolerate tens of attempts before the attacker has a significant chance of success.

Because the right rate limit is context specific, Domains can be configured to the needs of the user. The Sequential Delay Domain is designed for the use case of PIN and password hashing, and can be used to allow for a fixed number of attempts over a configurable time period (e.g. 15 attempts over 3 days). The Sequential Delay Domain additionally supports signature-based authentication to prevent quota from being consumed by any except the intended user.

Salting

Even with the use of ODIS to prevent brute-force guessing of a password, it remains important to include a user-specific value in the hashing request as a salt to prevent rainbow table attacks. A salt can be included in the Domain parameter of the request to ODIS to ensure a rate limit is enforced specific to the user's context. Using a random salt value is recommended, however a client identifier such as a username or phone number hash can also be used.

Password filtering

In addition to using ODIS to harden passwords chosen by users, it is recommended that the application help the user choose a good password during onboarding. Password filtering, blocking the user from setting a password which may be weak, can greatly improve the quality of a user's password and prevent it being broken by guessing the most common passwords (e.g. "password"). NIST 800-63 recommends that passwords should be checked against a list of known compromised passwords, such as HIBP Passwords. Additional research has found other practical techniques for increasing the strength of passwords chosen by users.

```
# odis-use-case-phone-number-privacy.md:
```

```
---
```

```
title: Phone Number Privacy
```

```
---
```

```
import PageRef from '@components/PageRef'
```

Celo's identity protocol allows users to associate their phone number with one or more addresses on the Celo blockchain. This allows users to find each other on the Celo network using phone number instead of cumbersome hexadecimal addresses. The Oblivious Decentralized Identifier Service (ODIS) was created to help preserve the privacy of phone numbers and addresses.

- ODIS

Understanding the problem

When a user sends a payment to someone in their phone's address book, the mobile client must look up the identifier for that phone number on-chain to find the corresponding Celo blockchain address.

This address is needed in order to create a payment transaction, and the user may only know the phone number of the person they want to pay.

If cleartext phone numbers were used as identifiers directly on the Celo network, then anyone would be able to associate all phone numbers with blockchain accounts and balances (e.g. After searching for addresses with a high balance, they could look up the associated phone number to phish the account owner).

If instead, the identifier was the hash of the recipient's phone number, attackers would still be able to associate phone numbers with accounts and balances via a rainbow table attack.

The solution

The basis of the solution is to derive a user's identifier from both their phone number and a secret pepper that is provided by the Oblivious Decentralized Identifier Service (ODIS).

In order to associate a phone number with a Celo blockchain address, the mobile wallet first queries ODIS for the pepper.

It then uses the pepper to compute the unique identifier that's used on-chain.

Peppers produced by ODIS are cryptographically strong, and so cannot be guessed in a brute force or rainbow table attack.

ODIS imposes a rate limit controlling how many peppers any individual can request, and so prevents an attacker from scanning a large number of phone numbers in an attempt to compromise user privacy.

Pepper request rate limiting

ODIS imposes a rate limit on requests for peppers in order to limit the feasibility of rainbow table attacks.

When ODIS receives a request for a pepper, it authenticates the request and ensures the requester has not exceeded their quota.

Since blockchain accounts and phone numbers are not naturally Sybil-resistant (i.e. individuals can have many accounts or phone numbers), ODIS bases request quota on the following factors:

- Requester transaction history
- Requester phone number attestation count and success rate
- Requester account balance

The requirements for these factors are configured to make it prohibitively expensive to scrape large quantities of phone numbers while still allowing typical user flows to remain unaffected.

In particular, it should be possible for a user to look up their contacts in order to send them payments.

odis.md:

title: Oblivious Decentralized Identifier Service (ODIS)

import PageRef from '@components/PageRef'

The Oblivious Decentralized Identifier Service (ODIS) allows for privacy preserving phone number mappings, password hardening, and other use cases by implementing a rate limited oblivious pseudorandom function (OPRF). Essentially, it is a service that allows users to compute a limited number of hashes (i.e. PRF evaluations), without letting the service see the data being hashed.

Many useful applications are built on top of this primitive, such as privacy protected phone number mappings, password hardening, and captchas for bot detection.

Distributed key generation

For the sake of user privacy and security, no single party should have the ability to unilaterally compute the OPRF function.

To ensure this, ODIS was designed to be decentralized across a set of reputable participants.

Before ODIS was deployed, a set of operators participated in a Distributed Key Generation (DKG) ceremony to generate shared secret, with its pieces split between the operators.

Details of the DKG setup can be found in the Celo Threshold BLS repository.

Each ODIS node holds a share of the key which can be used to calculate a piece of the OPRF evaluation that will be sent to the user.

When enough of these pieces are combined, their combination can be used to derive the unique OPRF evaluation (i.e. hash).

The number of key holders (m) and threshold of signatures required (k) to construct a full evaluation are both configurable at the time of the DKG ceremony.

Production setup

As of October 2021, ODIS operates with 7 signers and a threshold of 5 (i.e. $m=7$, $k=5$).

As a result, 5 of the 7 parties must cooperate in order to produce an output from the (P)OPRF function, and as long as at least 3 are honest and secure, no unauthorized requests will be served.

<!-- TODO(victor): Once the new set is in production, information about the 7 operators should be included here -->

Security properties

The goal the distributed key generation is to make it harder for a hacker, or a corrupt ODIS operator, to compromise the security of ODIS. In particular, if an attacker has control over any less than the threshold k of keys, they cannot make an unauthorized computation (e.g. querying the pepper for a phone number without quota) of the OPRF function.

Additionally, as long as k operators remain honest and have access to their keys, honest users will continue to be able to use the service even if $m-k$ corrupt operators are refusing their requests.

For example, consider the phone number privacy protocol when there are 7 ODIS operators and the required threshold is 5. An attacker may compute the pepper for all phone numbers if 5 operators are compromised or corrupt. If 3 are corrupt or taken offline (e.g. by DDoS attack) then an attacker may prevent the rest of the operators from generating the pepper for users.

In the case that a single key is compromised, user data will remain private and the service operational; however, it's important that we can detect and perform a key rotation before the number of keys compromised exceeds k or $m - k + 1$ (whichever is lower).

Rotating keys

If a key held by one of the operators is leaked, or if the operator becomes corrupt, a key rotation can allow the group to generate a new set of keys. Once the new keys are in place, operators can destroy their old keys, preventing any use from the compromised key. Key rotation can also allow new ODIS operators to be added, by creating new keys for all the existing operators as well as the newly added operator.

To rotate keys, a new DKG ceremony must be performed with at least k of the m original keys.

These newly generated keys will not be compatible with the old keys; however if k of the old keys are used, an attacker may still reach the necessary threshold.

Therefore, it's extremely important that all of the old keys are destroyed after a successful key rotation.

This DKG ceremony also provides the opportunity to change the values for k and m , adding or removing operators, or changing the threshold required to compute the OPRF.

Note that this process for key rotation does not change the public key the client uses to verify the results.

Blinding

When a client queries ODIS to get an OPRF evaluation, the client first blinds the phone number locally using a secret one-time key.

This blinding process preserves the privacy of underlying message (e.g. a mobile number or password) such that ODIS nodes won't learn any of the user's sensitive information.

In addition to protecting the user's privacy, it reduces the risk of targeted censorship.

ODIS operators compute the OPRF against this hidden input value, and return a result which is also hidden from the operators.

After the application receives the response, it unblinds it to receive the final evaluation result.

Note that this blinding process provides privacy to the user even if all of the ODIS operators were corrupted.

This blinding process is what makes the oblivious pseudo random function (OPRF) "oblivious".

Verification

Query results from ODIS can be verified against the services public key, which is shared with users along with the client library.

By verifying the results, the client can be sure that the service computed the OPRF correctly and that no one could have intercepted and changed the result.

Combiner

To facilitate the communication needed for the \mathbb{G} of \mathbb{G} OPRF evaluation, ODIS includes a combiner service which performs this orchestration for the convenience of wallets and other clients building on Celo.

Like the ODIS operators, the combiner only receives the blinded message and therefore it cannot learn anything about the user's sensitive information.

The combiner also verifies the response from each operator to ensure a corrupt operator cannot affect the resulting pepper.

Clients can additionally verify the response they get from the combiner to ensure the combiner could not have tampered with it.

Anyone can run a combiner, for their own use or for the public. Currently, cLabs operates one such combiner that may be used by any project building on Celo.

Rate limiting

As part of its core function, ODIS enforces rate limits on user queries. Rate limits depend on the application context in which ODIS is being used (e.g. the rate limit is much higher for deriving peppers for phone numbers than for hardening a 6-digit PIN)

Phone number privacy

The original API, targeted for phone number privacy, enforces a rate limit based on the actions, balance, and verification status of the user on the Celo blockchain.

In order to measure the quota for a given requester, ODIS must check their on-chain account information.

To prove ownership over their account, the POST request contains an Authorization header with the signed message body.

When ODIS nodes receive the request, it authenticates the user by recovering the message signer from the header and comparing it to the value in the message body.

Domains

In the newer domain separated API, the rate limit can depend on a variety of factors configured to each domain type.

More information about the domains API and the implemented domain types can be found in the respective pages.

- Domains
- Sequential Delay Domain

A full specification of the Domains API can be found in CIP-40.

- CIP-40

Request flow diagram

!request flow diagram

Architecture

!architecture diagram

The hosted architecture is divided into two components, the combiner and the signers.

Currently the combiner is a cloud function and the signers are independent NodeJS servers run by the operators.

Both services leverage the Celo Threshold BLS library which has been compiled to a Web Assembly module.

The combiner and signers maintain some minimal state in a SQL database, mainly related to quota tracking.

For storage of the BLS signing key, the signers currently support three cloud-based keystores: Azure Key Vault, AWS Secret Manager, and Google Secret Manager.

privacy-research.md:

Future Privacy Research

Celo is committed to meet the privacy needs of its users. This section describes future plans for delivering on this commitment, while also sharing the current limitations of the Celo networks.

Privacy mode

One downside to this identity protocol is that knowledge of a phone number can let anyone quickly determine the balance of the associated wallet, which of course may be unacceptable for many use cases. For these circumstances, the contract allows users to use the Attestations contract in privacy mode. In this mode, the user does not map their phone number to their wallet address, but to an account that is not meant to be the recipient of transfers. Through a registered encryption key on the user's account on the contract, schemes can be derived to allow users to selectively reveal their true wallet addresses to authorized participants.

<!-- Transaction and Balance Privacy

As with most public blockchains \ (e.g. Bitcoin, Ethereum\), transactions and smart contracts calls on Celo are public for everyone to see. This means that if a user wants to map the hash of their phone number to their wallet address, people with knowledge of that user's phone number will be able to see their transactions and balances.

To address this issue, the cLabs team, Matterlabs and other esteemed zk-SNARK cryptographers and Celo community members are working to create a framework that makes it easy to create gas-efficient tokens that offer Zcash-like privacy, using a shared anonymity pool. Such an implementation could allow wallets to use the default identity mode easily without the risk that someone with your phone number could see your balance and transaction history. -->

smart-contract-accounts.md:

title: Smart Contract Accounts

Smart contract accounts are used to enable features beyond what can be accomplished with an externally owned account (EOA) alone. In this document, we'll describe some of the features and considerations associated with smart contract accounts in general, and the architecture used by the Valora wallet in particular as an example of how smart contract accounts can be used.

EOAs are what most people think of when they imagine a blockchain wallet. EOAs are comprised of an ECDSA public/private key pair from which the on-chain address is derived.

The account address is derived from the public key, and transactions are authorized by the private key.

In most wallets, the EOA is generated and stored on the user's mobile device and backed up via a BIP-39 mnemonic phrase.

A smart contract account on the other hand is a smart contract that can be used to interact with other smart contracts on behalf of the owner. Celo provides an open-source implementation of a smart contract account; the meta-transaction wallet (MTW).

In general, ownership can be determined in arbitrary ways, but most commonly an EOA is designated as the owner and can authorize transactions by signing a meta-transaction containing the details of the authorized transaction.

This is how the meta-transaction wallet works.

In this case you can think of the smart contract account as the primary account, and the EOA as the controller of this account.

Benefits of a smart contract account

Separation of signer and payer

When new users create a wallet, they start with an empty balance. This makes it difficult for the new users to verify their phone number as they need to pay for both the Celo transactions and the Attestation Service fees (see here for more details).

To make this experience more intuitive and frictionless for new users, cLabs operates an onboarding service called Komenci that pays for the transactions on behalf of the user.

It does this by first deploying a meta-transaction wallet contract and setting the wallet EOA address as the signer.

At this point, the EOA can sign transactions and submit them to Komenci. Komenci will wrap the signed transaction into a meta-transaction, which it pays for and submits to the network.

In general, smart contract accounts allow the someone other than the account owner to pay for the transaction fees required to submit a transaction to the blockchain, enabling a number of useful operations not otherwise possible.

Account recovery

Smart contract accounts can also be useful if a user ever loses their phone and recovery phrase.

Unlike EOAs, smart contract accounts can support account recovery methods that do not rely solely on recovering the underlying keys.

The meta-transaction wallet implements a function to assign another Celo address as the Guardian of the account.

This Guardian can be a simple backup key or a smart contract implementing social recovery, KELP, or another account recovery protocol.

With the authorization of the Guardian, the meta-transaction wallet will update the owner of the account to replace the lost key.

Any funds or privileges held by the meta-transaction wallet are then recovered to the user who can control the account using their new key.

Transaction batching

With smart contract accounts, including the meta-transaction wallet, transactions can be batched together to execute atomically.

This makes for a better user experience, as transactions can be guaranteed to execute all together or entirely revert.

It can also prevent some cases where front-running would be possible by splitting the user's transactions.

Valora accounts

Behind every Valora wallet are two types of accounts: an externally owned account (EOA) and a meta-transaction wallet.

Valora generates the EOA during onboarding, and has a meta-transaction wallet deployed for it by Komenci with the generated EOA as the signer.

Using this configuration, Valora users gain the benefits listed above, including having Valora pay for the transaction fees associated with onboarding.

Sending to a Valora wallet

When performing a payment to a Valora wallet, it's important that the address that is receiving funds is the EOA, and not the MTW since funds in the MTW are not displayed or directly accessible to Valora users. To look up a wallet using a phone number:

1. Use ODIS to query the phone number pepper
2. Use the phone number pepper to get the on-chain identifier
3. Use the on-chain identifier to get the account address
4. Use the account address to get the wallet address (EOA)

The first two steps are covered extensively in this guide.

To get the account address (step 3) you can use the Attestation contract method `lookupAccountsForIdentifier`.

To get the wallet address from the account (step 4) you can use the Account contract method `getWalletAddress`.

It may also be necessary to lookup the data encryption key (ex. for comment encryption). This key can similarly be queried with the account by using the Account contract method `getDataEncryptionKey`.

You can view a working example of this all tied together in the `celocli` command `identity:get-attestations`.

Enabling Valora to interact with your dApp

Signatures

Since all Valora users will have the use a meta-transaction wallet, it's important to keep in mind that transactions may originate from an EOA as well as a smart contract.

If your contract relies upon EIP-712 signed typed data, be sure to also support typed data originating from contracts.

This data can't be signed by the `msg.sender` since it's originating from a contract, but is implicitly authorized by originating from the contract.

Implementation

The implementation of the meta-transaction wallet can be found [here](#).

```
# band-protocol.md:
```

```
---
```

```
title: Using Band Protocol
```

```
description: Tutorial on how to use the Band Protocol on Celo
```

```
---
```

```
import ImageWrapper from '@components/ImageWrapper'
```

By the end of this tutorial, you will understand how to query the Band Protocol reference data smart contract from another Solidity smart contract on Celo.

This tutorial will go over:

- What is Band?
- Deploying an example Oracle contract that calls the Band reference data contract
- Calling the reference data contract for current rates of different assets

What is the Band Protocol?

Band Protocol is a cross-chain data oracle platform that aggregates and connects real-world data and APIs to smart contracts. You can read more about the specific details of the protocol [here](#).

Deploy Oracle

1. Follow this [link](#) to Remix. The link contains an encoded example DemoOracle.sol contract.
2. Compile the contract with compiler version 0.6.11.
3. Switch to the Deploy tab of Remix.
 1. Select "Injected Web3" in the Environment dropdown in the top left to connect Metamask.
 2. Make sure that Metamask is connected to the Alfajores test network. You can read about adding Alfajores to Metamask [here](#).

<ImageWrapper path="/img/doc-images/band-protocol-how-to/remix-environment.png" alt="environment" width="400" />

4. Enter the Alfajores testnet Band reference data aggregator contract address 0x660cBc25F0cFD31F0Bdcaa43525f0bACC6DB2ABc to the DemoOracle constructor and deploy the contract. You can access the reference data aggregator contract on mainnet at 0xDA7a001b254CD22e46d3eAB04d937489c93174C3. Make sure you check the current address on their page, as it might happen that they updated their reference contract and then your oracle data will not be correct anymore.

<ImageWrapper path="/img/doc-images/band-protocol-how-to/deploy.png" alt="environment" />

An interface to interact with the contract will appear in the bottom left corner of Remix.

Get Rates

Clicking the getPrice button will return the current price of CELO in USD. This function calls getReferenceData(string memory base, string memory quote) on the Band reference data contract, passing "CELO" and "USD", indicating CELO as the base and USD as the quote. The rate returned is base/quote multiplied by 1e18.

<ImageWrapper path="/img/doc-images/band-protocol-how-to/get-price.png" alt="get price" width="300" />

Note that the DemoOracle contract only returns the latest rate, but the reference contract also returns values of the last time the base and quote references were updated.

The price is offset by $1e18$. The returned value at the time of testing is 3747326500000000000. Multiplying by $1e-18$ gives the current USD price given by the reference contract, 3.7473265 CELO/USD.

Clicking the getMultiPrices button returns multiple quotes in the same call, BTC/USD and BTC/ETH in this case. This function calls getReferenceDataBulk(string[] memory bases, string[] memory quotes) on the Band reference data contract, passing "CELO" as the base and "USD" and "ETH" for the quotes. This will return the current CELO prices in USD and ETH, as an array of integers. The call also returns just the exchange rates (multiplied by $1e18$), but can be modified to return the last updated times for the bases and quotes.

```
<ImageWrapper path="/img/doc-images/band-protocol-how-to/getmultiprices.png" alt="get prices" width="300" />
```

The "savePrice" function will save any base/quote rate that is passed to it in the storage variable named price. This storage data will only be updated when the "savePrice" function is called, so the saved price value will go stale unless this function is called repeatedly.

```
<ImageWrapper path="/img/doc-images/band-protocol-how-to/saveprice.png" alt="get prices" width="300" />
```

Mainnet Reference Data Contract

You can access the reference data aggregator contract on mainnet at 0xDA7a001b254CD22e46d3eAB04d937489c93174C3.

Available Reference Data

You can view the available reference data on the Band Data site [here](#).

Bandchain.js

Band also has a javascript library that makes it easy to interact with BandChain directly from Javascript or Typescript applications. The library provides classes and methods for convenient to send transactions, query data, OBI encoding, and wallet management. You can read more about it [here](#).

```
# index.md:
```

```
---
```

```
title: Oracles on Celo
```

```
description: A list of oracles available on the Celo network.
```

```
---
```

Oracles on Celo

This page lists Oracles running on the Celo network.

Oracles are essential components in blockchain ecosystems, acting as bridges that connect on-chain smart contracts with real-world data. They allow smart contracts to access information that exists outside the blockchain, such as financial market data, weather conditions, or other off-chain events. This capability is crucial because blockchains, by design, cannot access external data directly due to their isolated nature.

Here are lists of all on-chain Oracles:

- Band Protocol
- RedStone Oracles
- Celo Reserve Oracles
- Supra
- Chainlink, Price Feed Oracles
- Pyth Network
- Witnet

redstone.md:

```
---
title: Using RedStone oracles
description: Tutorial on how to use the RedStone oracles on Celo
---
```

```
import ImageWrapper from '@components/ImageWrapper'
```

By the end of this tutorial you will understand how to intergrate your dApp built on Celo with RedStone oracles.

This document will cover:

- What is RedStone?
- How to use RedStone?
- Examples

What is RedStone?

RedStone is a data ecosystem that delivers fast and accurate financial information in a decentralised fashion using an innovative approach of on-demand data fetching.

RedStone offers a radically different design of Oracles catering for the needs of modern Defi protocols.

- Leverage Arweave blockchain as a cheap and permanent storage
- Use token incentives to motivate data providers to maintain data integrity and the uninterrupted service
- Use signed meta-transactions to deliver prices on-chain

- Although the data at RedStone is persisted on the Arweave chain, it could be used with any other blockchain

You can read much more about the RedStone protocol in the RedStone compiled documentation.

What data is available

Thanks to our innovative architecture, we offer more than one thousand of pricing data feeds, including tokens, stocks, ETFs, commodities, and much more for a fraction of regular Oracles integration costs.

You can check available assets and data providers using `app.redstone.finance`.

How to use RedStone?

IMPORTANT: RedStone contracts are still undergoing security audit and we are working on the infrastructure security improvements. So, before using RedStone oracles in production dApps, please reach out to us on Discord. We will be happy to help you with the integration and will set up a new pool of data provider nodes if there is a need.

Please read this short documentation to learn how to integrate your dApp with RedStone oracles.

💡 Note: currently RedStone is integrated only with ethers.js library, so in order to use it on Celo blockchain dApps you should use ethers.js along with @celo-tools/celo-ethers-wrapper.

Code examples

- Repo with examples
- Generating pseudo-random values
- Example with multiple contracts
- Synthetic commodities dApp on Celo

Need help?

Please feel free to contact RedStone team on Discord if you have any questions.

run.md:

```
---
title: Running Oracles
description: How to run an oracle for Mento, the stability protocol.
---
```

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 calling `SortedOracles.report(address token, uint256 value, address lesserKey, address greaterKey)`. `SortedOracles` is a Celo Core Contract.

A reference implementation of such a client is written in TypeScript and would be used for this guide. Releases for this client can be found [here](#).

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 found [here](#).
- The private key of an address on Celo, which can be stored on a private key file, on a Hardware Security Module (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 the Celo 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:syncd --node YOURNODEHOSTNAME:YOURNODEPORT
```

Also make sure your node is accessible via WS, usually full nodes listen in port 8546.

:::warning

Using Forno or other public full node providers to run the oracles in production is strongly discouraged. Oracles doing so wouldn't be eligible 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 the Github repository. A list of all the available, as well as required variables, can be found here.

Running with HSM

Using High Security Modules (HSM) is the recommended way to store the keys for the oracles. Currently supported HSM are Azure and AWS. If you have already configured HSM, the relevant variables to add to your `.env.prod` are:

AWS:

- `WALLETTTYPE=AWSHSM`
- `AWSKEYREGION`

Azure:

- `WALLETTTYPE=AZUREHSM`
- `AZUREKEYVAULTNAME`

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 of `privateKey` should be stored in a file and its path should be set in the env variable `PRIVATEKEYPATH`. Additionally `WALLETTTYPE` should also be set to `PRIVATEKEY`. 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 found here.

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 found here 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 is PRICESOURCES. This sets what exchanges and prices shall be used to report. It is recommended to store this in a file called pricesources and export the content to a new env variable with cat.

```
export PRICESOURCES=$(cat pricesources)
```

An example of such a 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 connectors

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 PRICESOURCES=$PRICESOURCES 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 CURRENCYPAIR

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 the Celo 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 a proposal can be found [here](#).

Using kubernetes

You can reference Helm Charts configuration used by cLabs, which can be found in the celo-monorepo repository.

Metrics

The oracle client supports metrics suitable for Prometheus. Available metrics and their configuration can be found in the technical 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-now-2d?orgId=2>)
2. One with low sampling but longer timeframe-now-1M?orgId=2>)

Building from source

Instructions can be found in the development documentation.

supra.md:

```
---
title: Supra Oracle
description: Supra is a novel, high-throughput Oracle & IntraLayer. A vertically integrated toolkit of cross-chain solutions (data oracles, asset bridges, automation network, and more) that interlink all blockchains, public (L1s and L2s) or private (enterprises).
---
```

What is Supra?

Supra provides decentralized oracle price feeds that can be used for on-chain and off-chain use-cases such as spot and perpetual DEXes, lending protocols, and payments protocols. Supra's oracle chain and consensus algorithm makes it the fastest-to-finality oracle provider, with layer-1 security guarantees. The pull oracle has a sub-second response time. Aside from speed and security, Supra's rotating node architecture gathers

data from 40+ data sources and applies a robust calculation methodology to get the most accurate value. The node provenance on the data dashboard also provides a fully transparent historical audit trail. Supra's Distributed Oracle Agreement (DORA) paper was accepted into ICDCS 2023, the oldest distributed systems conference.

Check out our developer docs here.

becoming-a-validator.md:

Becoming a Validator

To participate in the network, an operator must put up a slashable commitment of locked CELO, register as a validator, and join a validator group. A minimum stake of one CELO and a notice period of 60 days is required to be a validator in the Alfajores Testnet.

Any account that meets the minimum stake and notice period requirements can register as a validator. By doing so, the locked funds on that account become 'at risk': a fraction of the stake can be slashed automatically for an evolving set of misbehaviors. In addition, the community can use governance proposals to slash funds, which avoids having to anticipate and encode in the protocol every possible misbehavior. As long as the CELO staked for a validator account is not slashed, it's eligible to earn rewards like any other Locked Gold account.

A validator joins a validator group by affiliating itself with it. However, to avoid untrusted or malicious validators joining a group, the validator group must accept the affiliation. Once done, the validator is added to the list of validators in the group. A validator can remove itself from a validator group at any time. Changes only take effect at the next subsequent election, so if the validator is currently participating in consensus, it's expected to do so until the end of the epoch in which it deregisters itself.

epoch-rewards-carbon-offsetting-fund.md:

title: Carbon Offsetting Fund

description: Introduction to the Carbon Offsetting fund, its purpose, and governance process.

Carbon Offsetting Fund

Introduction to the Carbon Offsetting fund, its purpose, and governance process.

What is the Carbon Offsetting Fund?

The Carbon Offsetting Fund provides for making the infrastructure of the Celo platform carbon-negative, by making a transfer every epoch to an organization that commits to using those assets off-chain for carbon offsetting projects.

Governance

Through the on-chain governance process, CELO holders can set the fraction of the total desired epoch rewards, initially planned to be 0.1%, that is received by the carbon offsetting fund, and the address of a carbon offsetting partner to which to direct these transfers. The on-target amount is adjusted by the epoch rewards multiplier, as with all epoch rewards.

epoch-rewards-community-fund.md:

title: Celo Community Fund

description: Introduction to the community fund, its assets, and its relationship to the on-chain reserve.

Community Fund

Introduction to the community fund, its assets, and its relationship to the on-chain reserve.

What is the Community Fund?

The Community Fund provides for the general upkeep of the Celo platform. CELO holders decide how to allocate these funds through governance proposals in the governance forum. Funds might be used to pay bounties for bugs or vulnerabilities, security audits, or grants for protocol development.

Community Fund Assets

The Community Fund receives assets from two sources:

- The Community Fund obtains a desired epoch reward defined as a fraction of the total desired epoch rewards \(\text{governable, initially planned to be } \\$\\$25\backslash\%\\$\\$ \backslash\). This amount is subject to adjustment up or down in the event of under- or over-spending against the epoch rewards target schedule.
- The Community Fund is the default destination for slashed assets.

epoch-rewards-locked-gold.md:

title: Celo Locked CELO Rewards
description: How to earn locked CELO rewards and adjust the rate for voting participation, target schedule, and deductions.

Locked CELO Rewards

How to earn locked CELO rewards and adjust the rate for voting participation, target schedule, and deductions.

Introduction to Locked CELO Rewards

Holders of Locked CELO that voted in the previous epoch for a group that elected one or more validators and have activated their votes are eligible for rewards. Rewards are added directly to the Locked CELO voting for that group, and re-applied as votes for that same group, so future rewards are compounded without the account holder needing to take any action. The voting process is described further here.

:::tip

Rewards to Locked CELO are totally independent from validator and validator group rewards, and are not subject to the group share.

:::

Adjusting the Reward Rate for Voting Participation

The protocol has a target for the proportion of circulating CELO that is locked and used for voting. An on-target reward rate is determined and then adjusted at every epoch to increase or reduce the attractiveness of locking up additional supply. This aims to balance having sufficient liquidity for CELO, while making it more challenging to buy enough CELO to meaningfully influence the outcome of a validator election.

The reward rate is adjusted as follows:

where r is the reward rate or voting yield, v is the voting fraction calculated as locked CELO for voting divided by circulating CELO supply, and a is the adjustment factor. If the voting participation is below the target at the end of an epoch, the on-target reward rate is increased; if the voting participation is above the target at the end of an epoch, the reward is decreased.

Adjusting the Reward Rate for Target Schedule and Deductions

Adjusting the on-target reward rate to account for under- or over-spending against the target schedule gives a baseline reward, essentially the percentage increase for a unit of Locked CELO voting for a group eligible for rewards.

The reward for activated Locked CELO voting for a given group is determined as follows. First, if the group elected no validators in the current epoch, rewards are zero. Otherwise, the baseline reward rate factors in two deductions. It is multiplied by the slashing penalty for the group, and by the average epoch uptime score for validators in the group elected in the current epoch. Finally, the group's activated pool of Locked CELO is increased by this rate.

```
# epoch-rewards-validator.md:
```

```
---
title: Celo Rewards for Validators and Validator Groups
description: Overview of epoch rewards for Validators and Validator Groups.
---
```

Validator Rewards

Overview of epoch rewards for Validators and Validator Groups.

```
---
```

The protocol aims to incentivize validator uptime performance and penalize past poor behavior in future rewards, while ensuring that payments are economically reasonable in size independent of fluctuations of the price of CELO.

Five factors affect validator and group rewards:

- The on-target reward amount for this epoch
- The protocol's overall spending vs target of epoch rewards
- The validator's 'uptime score'
- The current value of the slashing penalty for the group of which it was a member at the last election
- The group share for the group of which it was a member at the last election

Epoch rewards to validators and validator groups are denominated in Celo Dollars, since it is anticipated that most of their expenses will be incurred in fiat currencies, allowing organizations to understand their likely return regardless of volatility in the price of CELO. To enable this, the protocol mints new Celo Dollars that correspond to the epoch reward equivalent of CELO which are maintained on chain to preserve the collateralization ratio. Of course, the effect on the target schedule depends on the prevailing exchange rate.

On-target Rewards

The on-target validator reward is a constant value (as block rewards typically would be) and is intended to cover costs plus an attractive margin for amortized capital and operating expenses associated with a recommended set up that includes redundant hosts with hardware wallets in a secure co-lo facility, proxy nodes at cloud or edge hosting providers, as well as security audits. As with most parameters of the Celo protocol, it can be changed by governance proposal.

In the usual case where no validator in the group has been slashed recently, and the validator has signed almost every block in the epoch, then the validator receives the full amount of the on-target reward, less the fraction sent to the validator group based on the group share. Unlike in some other proof-of-stake schemes, epoch rewards to validators do not depend on the number of votes the validator's group has received.

Calculating Uptime Score

The Celo protocol tracks an 'uptime score' for each validator. When a validator proposes a block, it also includes in the block body every signature that it has received from validators committing the previous block.

For a validator to be 'up' at a given block, it must have its signature included in at least one in the previous twelve blocks. This cannot be done during the first 11 blocks of the epoch. At each epoch, this counter is reset to 0. Because the proposer order is shuffled at each election, it is very hard for a malicious actor withholding an honest validator's signatures to affect this measure.

Then, a validator's uptime for the epoch is the proportion of blocks in the epoch for which it is 'up': $u = (\text{counter} + \text{downtimegraceperiod}) / (\text{epochsize} - 11)$. Its epoch uptime score $S_v = u^k$, where $\text{downtimegraceperiod}$ and k are a governable constants. This means that even repeated downtimes of less than around a minute are ignored and longer downtimes also won't count against the validator as long as their total duration stays below $\text{downtimegraceperiod}$. After that the score will reduce rapidly due to the exponent k .

The validator's overall uptime score is an exponential moving average of the uptime score from this and previous epochs. $S\{v\} = \min(S_v, S_v \cdot x + S\{v-1\} \cdot (1-x))$ where $0 < x < 1$ and is governable. Since S_v starts out at zero, validators have a disincentive to change identities and an incentive to prioritize activities that improve long-term availability.

Calculating Slashing Penalty

The protocol also tracks for each group a 'slashing penalty', initially equal to one but successively reduced on each occasion a validator in

that group is slashed. The penalty returns to one 30 days after it was last reduced.

This factor is applied to all rewards to validators in that group, to the group itself, and to voters for the group.

The slashing penalty gives groups a further incentive to vet validators they accept as members, not only to avoid reducing their own future rewards from existing validators but to attract and retain the best validators.

Validators have an incentive to be elected through groups with a high value, so a recent slashing makes a group less attractive. Validators also have an incentive to select groups where they believe careful vetting processes are in place, because poor vetting of other validators in the group reduces their own expectation of future rewards.

When a validator is slashed, reduced rewards may lead other validators in the same group to consider equivalently 'safe' slots in other groups, if they are available. A validator disassociating from the group would cause the group's rewards to further decline. While that may cause churn in the set of groups through which validators are elected, it is unlikely that a validator would move to a group where they could not be elected (since in this case they would receive no rewards, as opposed to fewer rewards), hence making the votes by which they were previously elected unproductive.

Group Share

Validator groups are compensated by taking a share of the rewards allocated to validators. Validator groups set a group share rate when they register, and can change that at any time. The protocol automatically deducts this share, sending that portion of the epoch rewards to the validator group of which they were a member at the time of the last election.

Since the sum of a validator's reward and its validator group's reward are the same regardless of the 'group share' that the group chooses, no side-channel collusion is possible to avoid deductions for downtime or previous slashing.

epoch-rewards.md:

title: Celo Epoch Rewards Overview

description: Introduction to Celo epoch rewards and the target reward release schedule.

Epoch Rewards

Introduction to Celo epoch rewards and the target reward release schedule.

What are Epoch Rewards?

Epoch Rewards are similar to the familiar notion of block rewards in other blockchains, minting and distributing new units of CELO as blocks are produced, to create several kinds of incentives.

Epoch rewards are paid in the final block of the epoch and are used to:

- Distributed rewards for validators and validator groups
- Distribute rewards to holders of Locked CELO voting for groups that elected validators
- Make payments into a Community Fund for protocol infrastructure grants
- Make payments into a Carbon Offsetting Fund.

A total of 400 million CELO will be released for epoch rewards over time. CELO is a utility and governance asset on Celo, and also the reserve collateral for Celo Dollar (and possibly in the future other whitelisted tokens). It has a fixed total supply and in the long term will exhibit deflationary characteristics similarly to Ethereum.

Reward Disbursement

The total amount of disbursements is determined at the end of every epoch via a two step process.

Step 1

In step one, economically desired on-target rewards are derived. These are explained in the following pages. Several factors can increase or decrease the value of the payments that would ideally be made in a given epoch (including the CELO to Dollar exchange rate, the collateralization of the reserve, and whether payments to validators or groups are held back due to poor uptime or prior slashing).

Step 2

In step two, these on-target rewards are adjusted to generate a drift towards a predefined target epoch rewards schedule. This process aims to solve the trade-off between paying reasonable rewards in terms of purchasing power and avoiding excessive over- or underspending with respect to a predefined epoch rewards schedule. More detail about the two steps is provided below.

Adjusting Rewards for Target Schedule

There is a target schedule for the release of CELO epoch rewards. The proposed target curve \(\text{subject to change}\) of remaining epoch rewards declines linearly over 15 years to 50% of the initial 400 million CELO, then decays exponentially with half life of $t_{1/2} = \ln(2) \times 15 = 10.3$ afterwards. The choice of $t_{1/2}$ guarantees a smooth transition from the linear to the exponential regime.

The total actual rewards paid out at the end of a given epoch result from multiplying the total on-target rewards with a Rewards Multiplier. This adjustment factor is a function of the percentage deviation of the remaining epoch rewards from the target epoch rewards remaining. It evaluates to 1 if the remaining epoch rewards are at the target and to smaller \(\text{or larger}\) than 1 if the remaining rewards are below \(\text{or above, respectively}\) the target. This creates a drag towards the target schedule.

The sensitivity of the adjustment factor to the percentage deviation from the target are governable parameters: one for an underspend, one for an overspend.

index.md:

title: Celo Proof of Stake Overview

description: Overview of Celo's proof-of-stake algorithm, mechanisms, and implementation.

import YouTube from '@components/YouTube'

Proof of Stake

Overview of Celo's proof-of-stake algorithm, mechanisms, and implementation.

Mastering the Art of Validating

<YouTube videoId="3UIudzzCb8o" />

Validator Types

Celo uses a Byzantine Fault Tolerant consensus protocol to agree on new blocks to append to the blockchain. The instances of the Celo software that participate in this consensus protocol are known as validators. More accurately, they are active validators or elected validators, to distinguish them from registered validators which are configured to participate but are not actively selected.

Proof-of-Stake

Celo's proof-of-stake mechanism is the set of processes that determine which nodes become active validators and how incentives are arranged to secure the network.

Active Validators

The first set of active validators are determined in the genesis block. Thereafter at the end of every epoch, a fixed number of blocks fixed at network creation time, an election is run that may lead to validators being added or removed.

Validator Elections

In Celo's Validator Elections, holders of the native asset, CELO, may participate and earn rewards for doing so. Accounts do not make votes for validators directly, but instead vote for validator groups.

Before they can vote, holders of CELO move balances into the Locked Gold smart contract. Locked Gold can be used concurrently for: placing votes in Validator Elections, maintaining a stake to satisfy the requirements of registering as a validator or validator group, and also voting in on-chain Governance proposals. This means that validators and groups can vote and earn rewards with their stake.

:::tip note

Unlike in other proof-of-stake systems, holding Locked Gold or voting for a group does not put that amount 'at risk' from slashing due to the behavior of validators or validator groups. Only the stake put up by a validator or group may be slashed.

:::

Implementation

Most of Celo's proof-of-stake mechanism is implemented as smart contracts, and as such can be changed through Celo's on-chain Governance process.

- Accounts.sol manages key delegation and metadata for all accounts including Validators, Groups and Locked Gold holders.
- LockedGold.sol manages the lifecycle of Locked Gold.
- Validators.sol handles registration, deregistration, staking, key management and epoch rewards for validators and validator groups, as well as routines to manage the members of groups.
- Election.sol manages Locked Gold voting and epoch rewards and runs Validator Elections.

In Celo blockchain:

- consensus/istanbul/backend/backend.go performs validator elections in the last block of the epoch and calculates the new validator set diff.

- consensus/istanbul/backend/pos.go is called in the last block of the epoch to process validator uptime scores and make epoch rewards.

locked-gold.md:

title: Locked CELO and Voting

description: Introduction to locked CELO and how to use validator elections to participate in voting.

Locked CELO and Voting

Introduction to Celo locked gold (CELO) and how to use validator elections to participate in voting.

:::note

This page references "Locked Gold". The native asset of Celo was called Celo Gold (cGLD), but is now called CELO. Many references have been updated, but code and smart contract references may still mention Gold as it is more difficult to reliably and securely update the protocol code.

:::

Validator Election Participation

To participate in validator elections, users must first make a transfer of CELO to the LockedGold smart contract.

Concurrent Use of Locked CELO

Locking up CELO guarantees that the same asset is not used more than once in the same vote. However every unit of Locked CELO can be deployed in several ways at once. Using an amount for voting for a validator does not preclude that same amount also being used to vote for a governance proposal, or as a stake at the same time. Users do not need to choose whether to have to move funds from validator elections in order to vote on a governance proposal.

Unlocking Period

Celo implements an unlocking period, a delay of 3 days after making a request to unlock Locked CELO before it can be recovered from the escrow.

This value balances two concerns. First, it is long enough that an election will have taken place since the request to unlock, so that those units of CELO will no longer have any impact on which validators are managing the network. This deters an attacker from manipulations in the form of borrowing funds to purchase CELO, then using it to elect malicious validators, since they will not be able to return the borrowed

funds until after the attack, when presumably it would have been detected and the borrowed funds' value have fallen.

Second, the unlocking period is short enough that it does not represent a significant liquidity risk for most users. This limits the attractiveness to users of exchanges creating secondary markets in Locked CELO and thereby pooling voting power.

Locking and Voting Flow

The flow is as follows:

- An account calls lock, transferring an amount of CELO from their balance to the LockedGold smart contract. This increments the account's 'non-voting' balance by the same amount.
- Then the account calls vote, passing in an amount and the address of the group to vote for. This decrements the account's 'non-voting' balance and increments the 'pending' balance associated with that group by the same amount. This counts immediately towards electing validators. Note that the vote may be rejected if it would mean that the account would be voting for more than 3 distinct groups, or that the voting cap for the group would be exceeded.
- At the end of the current epoch, the protocol will first deliver epoch rewards to validators, groups and voters based on the current epoch (pending votes do not count for these purposes), and then run an election to select the active validator set for the following epoch.
- The pending vote continues to contribute towards electing validators until it is changed, but the account must call activate (in a subsequent epoch to the one in which the vote was made) to convert the pending vote to one that earns rewards.
- At the end of that epoch, if the group for which the vote was made had elected one or more validators in the prior election, then the activated vote is eligible for Locked CELO rewards. These are applied to the pool of activated votes for the group. This means that activated voting Locked CELO automatically compounds, with the rewards increasing the account's votes for the same group, thereby increasing future rewards, benefitting participants who have elected to continuously participate in governance.
- The account may subsequently choose to unvote a specific amount of voting Locked CELO from a group, up to the total balance that the account has accrued there. Due to rewards, this Locked CELO amount may be higher than the original value passed to vote.
- This Locked CELO immediately becomes non-voting, receives no further Epoch Rewards, and can be re-used to vote for a different group.

- The account may choose to unlock an amount of Locked CELO at any time, provided that it is inactive: this means it is non-voting in Validator Elections, the deregistrationPeriod has elapsed if the amount has been used as a validator or validator group stake, and not active in any Governance proposals. Once an unlocking period of 3 days has passed, the account can call withdraw to have the LockedGold contract transfer them that amount.

Votes persist between epochs, and the same vote is applied to each election unless and until it is changed. Vote withdrawal, vote changes, and additional CELO being used to vote have no effect on the validator set until the election finalizes at the end of the epoch.

Vote Delegation

Contract Release 10 introduced vote delegation, which allows the governance participant to delegate their voting power.

```
:::note
Validators and Validator groups cannot delegate.
:::
```

The governance participants who cannot actively participate to vote on governance proposals in the Celo ecosystem can now delegate their votes to utilize the dormant votes.

Currently, participants can only delegate to 10 other delegates.

Participants can follow the steps here to perform delegation using CeloCLI.

```
# penalties.md:
```

```
---
title: Celo Validator Penalties
description: Introduction to validator penalties, enforcement mechanisms,
and conditions.
---
```

Validator Penalties

Introduction to validator penalties, enforcement mechanisms, and conditions.

```
---
```

What is Slashing?

Slashing accomplishes punishment of misbehaving validators by seizing a portion of their stake. Without these punishments, for example, the Celo Protocol would be subject to the nothing at stake problem. Validator misbehavior is classified as a set of slashing conditions below.

Enforcement Mechanisms

The protocol has three means of recourse for validator misbehavior. Each slashing condition applies a combination of these, as described below.

- Slashing of validator and group stake - Some slashing conditions take a fixed amount of the Locked Gold stake put up by a validator. In these cases, the group through which that validator was elected for the epoch in which the slashing condition was proven is also slashed the same fixed amount. A validator or group's stake may be forfeit while it is registered or, after being deregistered, during the notice period (60 days for validators, 180 days for groups) and before the amount is withdrawn from the LockedGold contract.

- Suppression of future rewards - Every validator group has a slashing penalty, initially 1.0. All rewards to the group and to voters for the group are weighted by this factor. If a validator is slashed, the group through which that validator was elected for the epoch in which it misbehaved has the value of its slashing penalty halved. So long as no further slashing occurs, the slashing penalty is reset to 1.0 after slashingpenaltyresetepochs epochs.

- Ejection - When a validator is slashed, it is immediately removed from the group of which it is currently a member (even if this group is not the group that elected the validator at the point the misbehavior was recorded). Since no changes in the active validator set are made during an epoch, this means an elected validator continues participate in consensus until the end of the epoch. The group can choose to re-add the validator at any point, provided the usual conditions are met (including that the validator has sufficient Locked Gold as stake).

Slashing Conditions

There are three categories of slashing conditions:

- Provable \ (initiated off-chain, verifiable on-chain\)
- Governed \ (verified only by off-chain knowledge\)

Provable

Provable slashing conditions cannot be initiated automatically on chain but information provided from an external source can be definitively verified on-chain.

In exchange for sending a transaction which initiates a successful provable slashing condition on-chain, the reporter receives a "reward", a portion of the slashed amount (which will always be greater than the gas costs of the proof). The reward is added to the reporter's balance of non-voting LockedGold. The remainder of the slashed amount is sent to the Community Fund.

- Persistent downtime - A validator which can be shown to be absent from 8640 consecutive BLS signatures will be slashed 100 CELO, have future

rewards suppressed, and (most importantly in this case) will be ejected from its current group.

- Double Signing - A validator which can be shown to have produced BLS signatures for 2 distinct blocks at the same height and in the same consensus round but with different hashes will be slashed 9000 CELO, have future rewards suppressed, and will be ejected from its current group. Note that unlike some proof-of-stake networks, Celo does not penalize validators for double signing regular consensus messages. In particular, one side-effect of how Celo provides liveness can result in cases where honest validators may legitimately double sign blocks across different rounds at the same height (Cosmos terms this amnesia and also specifically excludes it from slashing).

Governed

For misbehavior which is harder to formally classify and requires some off-chain knowledge, slashing can be performed via governance proposals. These conditions are important for preventing nuanced validator attacks.

validator-elections.md:

```
---
title: Celo Validator Elections
description: Introduction to Celo validator elections and management of
groups and votes throughout the process.
---
```

Validator Elections

Introduction to Celo validator elections and management of groups and votes throughout the process.

Updating the Active Validator Set

The active validator set is updated by running an election in the final block of each epoch, after processing transactions and Epoch Rewards.

Group Voting Caps

One way to consider the security of a proof-of-stake system is the marginal cost of getting a malicious validator elected. In a steady state, assuming the Celo community set the incentives appropriately, a full complement of validators is likely to be elected, which means the attack cost is the cost of acquiring sufficient CELO to receive more votes than the currently elected validator with fewest votes, and thereby supplant it.

Goal of Validator Elections

The objective of Celo's validator elections differs from real-world elections: they aim to translate voter preferences into representation while promoting decentralization and creating a moat around existing, well-performing elected validators. Two design choices influence this: a limit on the maximum number of member validator that a group can list, and a voting cap on the number of votes that any one group can receive.

Handling Excess Votes

Since voting for a group can cause only the group's member validators to get elected, and no more, votes in excess of the number needed to achieve that are unproductive in the sense that they do not raise the number of votes needed to get the least-voted-for validator elected. This would translate into a lower cost for a malicious actor to acquire enough CELO to supplant that validator. This is particularly true because the protocol limits the maximum number of members in a group, to promote decentralization.

Per-Group Vote Cap

The Celo protocol addresses this by enforcing a per-group vote cap. This cap is set to be the number of votes that would be needed to elect all of its validators, plus one more validator. The cap is enforced at the point of voting: a user can only cast a vote for a group if it currently has fewer votes than this cap. An account holder may not set or increase the amount of gold they have voting for a particular validator group j , if it already has at least $[(\text{groupmembers}_j + 1) / \min(\text{totalgroupmembers}, \text{maxvalidators})]$ of the total Locked Gold.

Adding New Validators

If a group adds a new validator, or the total amount of voting Locked Gold increases, the group's cap rises and new votes are permitted. If a group removes a validator or a validator chooses to leave, or the total amount of voting Locked Gold falls, then the group's cap falls: if it has more votes than this new cap, then new votes are no longer permitted, but all existing votes continue to be counted.

The Celo protocol allows an account to divide its vote between up to ten groups, since there may be cases where the vote cap prevents an account allocating its entire vote to its first choice group.

Running the Election

The Election contract is called from the IBFT block finalization code to select the validators for the following epoch. The contract maintains a sorted list of the Locked Gold voting (either pending or activated) for each Validator Group. The D'Hondt method, a closed party list form of proportional representation, is applied to iteratively select validators from the Validator Groups with the greatest associated vote balances.

Filtering Groups

The list of groups is first filtered to remove those that have not achieved a certain fraction of the votes of the total voting Locked Gold.

Assigning Seats

Then, in the first iteration, the algorithm assigns the first seat to the group that has at least one member and with the most votes. Thereafter, it assigns the seat to the group that would 'pay', if its next validator were elected, the highest vote averaged over its candidates that have been selected so far plus the one under consideration.

Number of Active Validators

There is a minimum target and a maximum cap on the number of active validators that may be selected. If the minimum target is not reached, the election aborts and no change is made to the validator set this epoch.

```
# validator-groups.md:
```

```
---
```

```
title: Celo Validator Groups
```

```
description: Celo's proof-of-stake mechanism introduces the concept of Validator Groups as intermediaries between voters and validators.
```

```
---
```

Validator Groups

Celo's proof-of-stake mechanism introduces the concept of Validator Groups as intermediaries between voters and validators.

```
---
```

What is a Validator Group?

A validator group has members, an ordered list of candidate validators. There is a fixed limit to the number of members that a group may have.

Why use a Validator Group?

Validator groups can help mitigate the information disparity between voters and validators. It is anticipated that groups might emerge that do not necessarily operate validators themselves but attract votes for their reputation for ensuring their associated validators have known real-world identities, have high uptime, are well maintained and regularly audited. Since every validator needs to be accepted by a single group to stand for election, that group will be more able to build up long-term judgements on their validators' operational practices and security setups than each of the numerous CELO holders that might vote for it would.

Fielding Multiple Validators

Equally, a number of organizations may want to attempt to field multiple validators under their own control, or be able to interchange the specific machines or keys under which they validate in the case of hardware or connectivity failure. By switching out validators in the list, groups can accomplish this without users having to change their votes.

Validator Group Limits

Validator groups can have no more than a small, fixed maximum number of validators -- currently 5 in Mainnet. This means an organization wanting to get more validators elected than this maximum has the added challenge of managing multiple group identities and reputations simultaneously. This further promotes decentralization and strengthens operational security, making it more likely that the validator set will be composed of nodes operated in different fashions by independent individuals and organizations.

Registration

Any account that has at least the minimum stake requirement in Locked Gold, whether voting or non-voting, can register an empty validator group. If a validating key is specified it may be used for this registration.

Deregistration

The account that creates a validator group is able to deregister that group if it has no members.

While an account has a registered validator group, or for up to a `deregistrationPeriod` after it is deregistered, attempts to unlock the account's amount of Locked Gold will fail if they would cause the remaining amount to fall below the minimum stake requirement.

Group Share

Validator groups are compensated by taking a share (the 'Group Share') of the validator rewards from any of its member validators that are elected during an epoch. This value is set at registration time and can be changed later.

Changing Group Members

The account owner controls the list of validators in their group and can at any time add, remove, or re-order validators.

For a validator to be added to a group, several conditions must hold: the number of members in the group must be less than the maximum; the Locked Gold balance of the group's account must be sufficient (the stake is per-member validator); and the validator must first have set its affiliation to the group.

This means that while a group can unilaterally remove a validator, and a validator can unilaterally leave by changing its affiliation, both parties have to agree before a validator can become a member of a group.

Votes and Voting Cap

Validator Groups can receive votes from Locked Gold up to a voting cap. This value is set to be the number of votes that would be needed to elect all of its validators, plus one more validator. The cap is enforced at the point of voting: a user can only cast a vote for a group if it currently has fewer votes than this cap.

Slashing Penalty

A slashing penalty, initially 1.0, is also tracked for each validator group. This value may be reduced as a penalty for misbehavior of the validator in the group. It affects the future rewards of the group, its validators, and Locked Gold holders receiving rewards for voting for the group.

Metadata

Both validators and validator groups can use Accounts Metadata to provide unverified metadata (such as name and organizational affiliation) as well as claims that can be verified off-chain for control of third-party accounts. All validators are encouraged to make a verifiable claim for domain names.

Dissolving of a Validator Group

There is a 180 day unlocking period for Celo locked when creating a validator group.

```
# adding-stable-assets.md:
```

```
---
title: Add Stable Assets to Celo
description: Overview of the requirements and steps to add a new stable
asset to the Celo platform.
---
```

Add Stable Assets

Overview of the requirements and steps to add a new stable asset to the Celo platform.

```
---
```

```
:::tip Note
```

This example assumes we want to add to the platform a new stable asset cX tracking the value of X (where X can be a fiat currency like ARS or MXN), using the Mento exchange.

:::

Requirements

Liquidity

The asset X has to be liquidly traded against CELO, in a CELO/X ticker. In absence of that, X has to be liquidly traded, including weekends, against well known assets that trade 24/7, like BTC or ETH, such that the price of X with respect to Celo can be inferred. In this second case, an implicit pair can be calculated for the oracle reports.

Determine pre-mint addresses and amounts

It is possible to pre-mint a fixed amount at the time of launching a new stable asset, good candidates to receive the pre-mint are the community fund and other entities committed to distribute this initial allocation to grant recipients and liquidity providers.

:::tip

A good criteria to a successfully decide a pre-mint amount is to check by how much it would affect the reserve collateralization ratio, this is, the ratio of all stable assets, divided by all the reserve holdings. Reserve information, as well as the collateralization ration can be found on the Reserve website.

:::

Procedure

Including contracts on the registry

Currently, the addition of new assets is tied to the Contract Release Cycle, as the contracts ExchangeX and StableTokenX need to be checked in [^1]. These new contracts inherit from Exchange and StableToken, that are the ones originally used for cUSD. As StableToken cX will be initialized by the contract release, key parameters like spread and reserveFraction should be included, although they can be later modified by setters in the following governance proposals. The only value that can't be changed is the pre-mint amount.

Freezing

These contracts should be set as frozen to prevent cX from being transferable before Mento supports it in a governance proposal. At this point, as there are no oracles, the contract ExchangeX can't update buckets and it is thus impossible to mint and burn cX. There is an issue open to include this step as part of the Contract Release.

For the deployment of cEUR, this was included as part of the Oracle activation proposal.

Constitutional parameters

<!-- TODO: SDK urls will need to be changed when the SDK type docs are separated from the rest of docs -->

As new contracts are added to the registry, new constitution parameters need to be set. There's an issue open to include this in the tooling to support it as part of the Contract Release.

Oracle activation

A following governance proposal needs to be submitted to enable oracles to report. This oracle proposal needs to enable addresses to report to the StableTokenX address and, optionally, fund them to pay for gas fees. An example of this proposal is the cEUR oracle activation proposal^[2].

Full activation

The last governance proposal is expected to unfreeze the contract and attach the last strings in the process to get a fully transferable asset stabilized by the Reserve. This propose involves:

1. Unfreezing both StableTokenX & ExchangeX.
2. Making ExchangeX able to pull CELO out of the Reserve for the buckets
Reserve.addExchangeSpender
3. Declaring the token to the Reserve as an asset to be stabilized
calling Reserve.addToken
4. Enable StableTokenX as a fee currency, so that it can be used to pay for gas
FeeCurrencyWhitelist.addToken.
5. In case necessary, parameters such as reserveFraction and spread can also be updated in this governance proposal.
6. Granda Mento activation

After passing this last proposal, cX should be fully activated.

Tooling

Adding a new stable asset involves updating many parts of the tooling, such as:

- Update the Ledger app integration such that it displays the names of the newly added token.
- Update oracles and generating their keys and addresses.
- Adding support on contractkit.
- Adding support on kliento.
- Adding support on eksportisto.
- Update on the cli, an example list of things to add are included on this issue.
- Supporting alfajores faucet.
- Supporting on Dapp kit.

[^1] There are opened issues trying to de-couple the addition of new assets to the reserve to the release cycle.

:::note

[^2] Please note this example proposal also includes freezing, this is because, at the time of writing (22-march-2021), the tooling for proposing a contract release doesn't support freezing those contracts on the same proposal. Proposals shall not be modified manually given that the tool is meant to run verifications.

:::

doto.md:

title: Celo Stability Algorithm (Mento)
description: How the supply of the Celo Dollar is achieved in the Celo protocol using the constant-product decentralized one-to-one mechanism (CP-DOTO).

Stability Algorithm (Mento)

How the supply of the Celo Dollar is achieved in the Celo protocol using the constant-product decentralized one-to-one mechanism.

What is Mento?

On a high level, Mento (previously known as CP-DOTO) allows user demand to determine the supply of celo stable assets by enabling users to create, for example, a new Celo Dollar by sending 1 US Dollar worth of CELO to the reserve, or to burn a Celo Dollar by redeeming it for 1 US Dollar worth of CELO. The mechanism requires an accurate Oracle value of the CELO to US Dollar market rate to work.

Incentives

This creates incentives such that when demand for the Celo Dollar rises and the market price is above the peg, users can profit using their own efforts by buying 1 US Dollar worth of CELO on the market, exchanging it with the protocol for one Celo Dollar, and selling that Celo Dollar for the market price.

Similarly, when demand for the Celo Dollar falls and the market price is below the peg, users can profit using their own efforts by purchasing Celo Dollar at the market price, exchanging it with the protocol for 1 US Dollar worth of CELO, and selling the CELO to the market.

Mitigating Risk

In cases in which the CELO to US Dollar oracle value is not an accurate reflection of the market price, exploiting such discrepancies can lead to a depletion of the reserve. Mento, inspired by the Uniswap system,

mitigates this risk of depletion as follows: The Celo protocol maintains two virtual buckets of CELO and Celo Dollar. The amounts in these virtual buckets are recalibrated every time the reported oracle value is updated, provided the difference between the current time and the oracle timestamp is less than $\text{\$oracle\staleness\threshold\$\$}$.

Model Equations

The equation for the constant-product-market-maker model fixes the product of the wallet quantities.

$$\begin{array}{l} \text{\$} \\ Gt \times Dt = k \\ \text{\$} \end{array}$$

where $\text{\$}Gt\text{\$}$ and $\text{\$}Dt\text{\$}$ denote the quantities in the CELO and Celo Dollar buckets respectively and $\text{\$}k\text{\$}$ is some constant. Given the above rule, it can be shown that the price of CELO, to be paid in Celo Dollar units, is

$$\begin{array}{l} \text{\$} \\ P_t = \frac{Dt}{Gt} \\ \text{\$} \end{array}$$

for traded amounts that are small relative to the bucket quantities.

Oracle Rates

Whenever the CELO to US Dollar oracle rate is updated, the protocol adjusts the bucket quantities such that they equalize the on-chain CELO to Celo Dollar exchange rate $\text{\$}P_t\text{\$}$ to the current oracle rate. During such a reset, the CELO bucket must remain smaller than the total reserve gold balance. To achieve this, the CELO bucket size is defined as the total reserve balance times $\text{\$}gold\backslash bucket\backslash size\text{\$}$, with $\text{\$}0 < gold\backslash bucket\backslash size < 1\text{\$}$ and the Celo Dollar bucket size is then chosen such that $\text{\$}P_t\text{\$}$ mirrors the oracle price. To discourage excessive on-chain trading, a transaction fee is imposed by adding small spread around the above exchange rate.

If the oracle precisely mirrors the market rate, the on-chain CELO to Celo Dollar rate will equal the CELO to US Dollar market rate and no profit opportunity will exist as long as Celo Dollar precisely tracks the US Dollar. If the oracle price is imprecise, the two rates will differ, and a profit opportunity will be present even if Celo Dollar accurately tracks the US Dollar. However, as traders exploit this opportunity, the on-chain price $\text{\$}P_t\text{\$}$ will dynamically adjust in response to changes in the tank quantities until the opportunity ceases to exist. This limits the depletion potential in Mento in the case of imprecise or manipulated oracle rates.

:::tip

For a more detailed explanation, read the article [Zooming in on the Celo Expansion & Contraction Mechanism](#).

:::

Multi-mento Deployment

Many instances of mento can be deployed in parallel for different stable assets. Currently, cEUR and cUSD live side-by-side, with independent buckets and oracle reports (although both of them are using the same SortedOracles instance). They all fill the CELO bucket with funds from the Reserve, but not necessarily at the same time.

granda-mento.md:

title: Granda Mento

description: Introduction to Granda Mento (CIP 38), its design, and how to manage exchange proposals.

Granda Mento

Introduction to Granda Mento (CIP 38), its design, and how to manage exchange proposals.

What is Granda Mento?

Granda Mento, described in CIP 38, is a mechanism for exchanging large amounts of CELO for Celo stable tokens that aren't suitable for Mento or over-the-counter (OTC).

Mento has proven effective at maintaining the stability of Celo's stable tokens, but the intentionally limited liquidity of its constant-product market maker results in meaningful slippage when exchanging tens of thousands of tokens at a time. Slippage is the price movement experienced by a trade. Generally speaking, larger volume trades will incur more slippage and execute at a less favorable price for the trader.

Similar to Mento, exchanges through Granda Mento are effectively made against the reserve. Purchased stable tokens are created into existence ("minted"), and sold stable tokens are destroyed ("burned"). Purchased CELO is taken from the reserve, and sold CELO is given to the reserve. For example, a sale of 50,000 CELO in exchange for 100,000 cUSD would involve the 50,000 CELO being transferred to the reserve and the 100,000 cUSD being created and given to the exchanger.

At the time of writing, exchanging about 50,000 cUSD via Mento results in a slippage of about 2%. Without Granda Mento, all launched Celo stable tokens can only be minted and burned using Mento, with the exception of cUSD that is minted as validator rewards each epoch. Granda Mento was created to enable institutional-grade liquidity to mint or burn millions of stable tokens at a time.

The mainnet Granda Mento contract address is 0x03f6842B82DD2C9276931A17dd23D73C16454a49 (link), was introduced in Contract Release 5, and activated in CGP 31.

How it works

A Granda Mento exchange requires rough consensus from the Celo community and, unlike the instant and atomic Mento exchanges, involves the exchanger locking their funds to be sold for multiple days before they are exchanged.

Design

At a high level, the life of an exchange is:

1. Exchanger creates an "exchange proposal" on-chain that locks their funds to be sold and calculates the amount of the asset being purchased according the current oracle price and a configurable spread.
2. If rough consensus from the community is achieved, a multi-sig (the "approver") that has been set by Governance approves the exchange proposal on-chain.
3. To reduce trust in the approver multi-sig, a veto period takes place where any community member can create a governance proposal to "veto" an approved exchange proposal.
4. After the veto period has elapsed, the exchange is executable by any account. The exchange occurs with the price locked in at stage (1).

Processes

Processes surrounding Granda Mento exchanges, like how to achieve rough consensus from the community, are outlined in CIP 46. At the minimum, it takes about 7 days to achieve rough consensus.

The approver multi-sig that is ultimately responsible for approving an exchange proposal that has achieved rough consensus from the community is 0xf10011424A0F35B8411e9abcF120eCF067E4CF27 (link) and has the following signers:

Name	Affiliation	Discord Handle
Andrew Shen	Bi23 Labs	Shen \ Bi23 Labs #6675
Pinotio	Pinotio	Pinotio.com #5357
Serge Kiema	DuniaPay	sergeduniapay #5152
Will Kraft	Celo Governance Working Group	Will Kraft #2508
Zviad Metreveli	WOTrust	zm #1073
human	OpenCelo	human #6811

```
| Deepak Nuli      | Kresko      | Deepak \ | Kresko#3647    |  
0x099f3F5527671594351E30B48ca822cc90778a11 |
```

```
# index.md:
```

```
---
```

```
title: Celo Stability Mechanism Overview
```

```
description: Overview of the Celo protocol's Stability Mechanisms.
```

```
---
```

Watch this video to get an intro to the Celo Stability Mechanism. Please note that "Celo Gold" is the deprecated name for "CELO".

```
import YouTube from '@site/src/components/YouTube';
```

```
import PageRef from '@site/src/components/PageRef';
```

Stability Mechanism

Overview of the Celo protocol's Stability Mechanisms.

```
---
```

Stability of Mento Stablecoin Protocol

```
<YouTube videoId="kYhDUMKuGCY"/>
```

The Celo protocol's stability mechanism comprises the following:

- Stability Algorithm (Mento)
- Granda Mento
- Oracles
- Stability Fees
- Adding Stable Tokens

```
# oracles.md:
```

```
---
```

```
title: Oracles
```

```
description: How the SortedOracles smart contract uses governance to  
collect reports and maintain the oraclized rate or the Celo dollar.
```

```
---
```

Oracles

How the SortedOracles smart contract uses governance to collect reports and maintain the oraclized rate or the Celo dollar.

```
---
```

SortedOracles Smart Contract

As mentioned in the previous section, the stability mechanism needs to know the market price of CELO with respect to the US dollar. This value is made available on-chain in the SortedOracles smart contract.

Collecting Reports

Through governance, a whitelist of reporters is selected. These addresses are allowed to make reports to the SortedOracles smart contract. The smart contract keeps a list of most recent reports from each reporter. To make it difficult for a dishonest reporter to manipulate the oraclized rate, the official value of the oracle is taken to be the median of this list.

Maintaining Oracle Values

To ensure the oracle's value doesn't go stale due to inactive reporters, any reports that are too old can be removed from the list. "Too old" here is defined based on a protocol parameter that can be modified via governance.

Celo-Oracle Repository

You can find more information about the technical specification of the Celo Oracles feeding data to the reserve in the GitHub repository [here](#).

stability-fees.md:

title: Celo Stability Fees

description: Overview of stability fee parameters, timing, frequency, amounts, management, and updates.

Stability Fees

Overview of stability fee parameters, timing, frequency, amounts, management, and updates.

Parameters Governing the Stability Fee

`inflationPeriod` how long to wait between rounds of applying inflation

`inflationRate` the multiplier by which the inflation factor is adjusted per `inflationPeriod`

Timing, Frequency, and Amount of Fee

The `inflationRate` is the multiplier by which the `inflationFactor` is increased per `inflationPeriod`. It is initially set to 1 which leaves it to governance to enable the stability fee later on.

Both, the inflationRate as well as the inflationPeriod, are specified for a given stable token and subject to changes based on governance decisions.

Stability Fee Levied on Balance

Each account's stable token balance is stored as 'units', and inflationFactor describes the units/value ratio. The Celo Dollar value of an account can therefore be computed as follows.

Account cUSD Value = Account cUSD Units / inflationFactor

When a transaction occurs, a modifier checks if the stability fee needs updating and, if so, the inflationFactor is updated.

Updates to the Inflation Factor

To apply periodic inflation, the inflation factor must be updated at regular intervals. Every time an event triggering an inflationFactor update (eg a transfer) occurs, the updateInflationFactor modifier is called (pseudocode below), which does the following:

1. Decide if one or more inflationPeriod have passed since the last time inflationFactor was updated
2. If so, find out how many have passed
3. Compute the new inflationFactor and update the last updated time:

$\text{inflationFactor} = \text{inflationFactor} \times \text{inflationRate}^{\text{inflationPeriods since last update}}$

Changes to Inflation Factor

Desired inflation rates may vary over time. When a new rate needs to be set, a governance proposal is required to update the inflation rate. If successful, the above function is called, which ensures inflationFactor is up to date, then updates the inflationRate and inflationPeriod parameters.

Inflation Factor Update Schedule

The updateInflationFactor modifier is called by the following functions:

- setInflationParameters
- approve
- mint
- transferWithComment
- burn
- transferFrom
- transfer
- debitFrom

erc20-transaction-fees.md:

title: Paying for Gas with Tokens

description: How to pay gas fees using allowlisted ERC20 tokens on Celo.

Introduction

In most L1 and L2 networks, transaction fees can only be paid with one asset, typically, the native asset for the ecosystem which is often volatile in nature. In order to simplify the process of sending funds on Celo, these fees can be paid with allowlisted ERC20 tokens such as USDT, USDC, cUSD, and others, in addition to CELO. This means that a user sending a stablecoin to friends or family will be able to pay the transaction fee out of their stablecoin balance, and will not need to hold a separate CELO balance in order to transact. Critically, Celo supports this functionality natively without Account Abstraction, Pay Masters, or Relay Services. Instead, wallets simply need to add an extra `feeCurrency` field on transaction objects to take advantage of this feature.

Fee Currency Field

The protocol maintains a governable allowlist of smart contract addresses which can be used to pay for transaction fees. These smart contracts implement an extension of the ERC20 interface, with additional functions that allow the protocol to debit and credit transaction fees. When creating a transaction, users can specify the address of the currency they would like to use to pay for gas via the `feeCurrency` field. Leaving this field empty will result in the native currency, CELO, being used. Note that transactions that specify non-CELO gas currencies will cost approximately 50k additional gas.

Allowlisted Gas Fee Addresses

To obtain a list of the gas fee addresses that have been allowlisted using Celo's Governance Process, you can run the `getWhitelist` method on the `FeeCurrencyWhitelist` contract. All other notable mainnet core smart contracts are listed [here](#).

Tokens with Adapters

After Contract Release 11, addresses in the allowlist are no longer guaranteed to be full ERC20 tokens and can now also be adapters. Adapters are allowlisted in-lieu of tokens in the scenario that a ERC20 token has decimals other than 18 (e.g. USDT and USDC).

The Celo Blockchain natively works with 18 decimals when calculating gas pricing, so adapters are needed to normalize the decimals for tokens that use a different one. Some stablecoins use 6 decimals as a standard.

Transactions with those ERC20 tokens are performed as usual (using the token address), but when paying gas currency with those ERC20 tokens, the adapter address should be used. This adapter address is also the one that should be used when querying Gas Price Minimum.

Adapters can also be used to query `balanceOf(address)` of an account, but it will return the balance as if the token had 18 decimals and not the native ones. This is useful to calculate if an account has enough balance to cover gas after multiplying `gasPrice` `estimatedGas` without having to convert back to the token's native decimals.

Adapters by network

Mainnet

Name	Token
Adapter	
-----	-----
-----	-----
-----	-----
-----	-----
USDC	0xcebA9300f2b948710d2653dD7B07f33A8B32118C 0x2F25deB3848C207fc8E0c34035B3Ba7fC157602B
USDT	0x48065fbbe25f71c9282ddf5e1cd6d6a887483d5e 0x0e2a3e05bc9a16f5292a6170456a710cb89c6f72

Alfajores (testnet)

Name	Token
Adapter	
-----	-----
-----	-----
-----	-----
-----	-----
USDC	0x2F25deB3848C207fc8E0c34035B3Ba7fC157602B 0x4822e58de6f5e485eF90df51C41CE01721331dC0

Baklava (testnet)

N/A

Enabling Transactions with ERC20 Token as fee currency in a wallet

We recommend using the `viem` library as it has support for the `feeCurrency` field in the transaction required for sending transactions where the gas fees will be paid in ERC20 tokens. `Ethers.js` and `web.js` currently don't support `feeCurrency`.

Estimating gas price

To estimate gas price use the token address (in case of `cUSD`, `cEUR` and `cREAL`) or the adapter address (in case of `USDC` and `USDT`) as the value for `feeCurrency` field in the transaction.

:::info

The Gas Price Minimum value returned from the RPC has to be interpreted in 18 decimals.
:::

Preparing a transaction

When preparing a transaction that uses ERC20 token for gas fees, use the token address (in case of cUSD, cEUR and cREAL) or the adapter address (in case of USDC and USDT) as the value for feeCurrency field in the transaction.

The recommended transaction type is 123, which is a CIP-64 compliant transaction read more about it [here](#).

Here is how a transaction would look like when using USDC as a medium to pay for gas fees.

```
js
let tx = {
  // ... other transaction fields
  feeCurrency: "0x2f25deb3848c207fc8e0c34035b3ba7fc157602b", // USDC
  Adapter address
  type: "0x7b",
};
```

:::info
To get details about the underlying token of the adapter you can call adaptedToken function on the adapter address, which will return the underlying token address.
:::

escrow.md:

```
---
title: Celo's Escrow Contract
description: Introduction to the Celo Escrow contract and how to use it
to withdraw, revoke, and reclaim funds.
---
```

Escrow

Introduction to the Celo Escrow contract and how to use it to withdraw, revoke, and reclaim funds.

What is the Escrow Contract?

The Escrow contract utilizes Celo's Lightweight identity feature to allow users to send payments to other users who don't yet have a public/private key pair or an address. These payments are stored in this contract itself and can be either withdrawn by the intended recipient or reclaimed by the

sender. This functionality supports both versions of Celo's lightweight identity: identifier-based (such as a phone number to address mapping) and privacy-based. This gives applications that intend to use this contract some flexibility in deciding which version of identity they prefer to use.

How it works

If Alice wants to send a payment to Bob, who doesn't yet have an associated address, she will send that payment to this Escrow contract and will also create a temporary public/private key pair. The associated temporary address will be referred to as the `paymentId`. Alice will then externally share the newly created temporary private key, also known as an invitation, to Bob, who will later use it to claim the payment. This `paymentId` will now be stored in this contract and will be mapped to relevant details related to this specific payment such as: the value of the payment, an optional identifier of the intended recipient, an optional amount of attestations the recipient must have before being able to withdraw the payment, an amount of time after which the sender can revoke the payment (via the `expirySeconds` field - more on that in the "withdrawing" section below), which asset is being transferred in this payment, etc.

Withdrawing

The recipient of an escrowed payment can choose to withdraw their payment assuming they have successfully created their own public/private key pair and now have an address. To prove their identity, the recipient must be able to prove ownership of the `paymentId`'s private key, which should have been given to them by the original sender. If the sender set a minimum number of attestations required to withdraw the payment, that will also be checked in order to successfully withdraw. Following the same example as above, if Bob wants to withdraw the payment Alice sent him, he must sign a message with the private key given to him by Alice. The message will be the address of Bob's newly created account. Bob will then be able to withdraw his payment by providing the `paymentId` and the `v`, `r`, and `s` outputs of the generated ECDSA signature. An escrowed payment may have `expirySeconds` set, which references the amount of time that must pass before the sender can revoke the payment. Note that after `expirySeconds` have passed, the payment recipient may still withdraw the payment as long as it has not already been revoked.

Revoking & Reclaiming

Alice sends Bob an escrowed payment. Let's say Bob never withdraws it, or worse, the temporary private key he needs to withdraw the payment gets lost or sent to the wrong person. For this purpose, Celo's protocol also allows for senders to reclaim any unclaimed escrowed payment that they sent. After an escrowed payment's `expirySeconds` (set by the sender on creation of the payment) has passed, the sender of the payment can revoke the payment and reclaim their funds with just the `paymentId`.

gas-pricing.md:

title: Celo Gas Pricing

description: Introduction to gas prices, calculations, transactions, and fees on the Celo network.

Gas Pricing

Introduction to gas prices, calculations, transactions, and fees on the Celo network.

Gas Price Minimum

Celo uses a gas market based on EIP-1559. The protocol establishes a gas price minimum that applies to all transactions regardless of which validator processes them.

The gas price minimum will respond to demand, increasing during periods of sustained demand, but allowing temporary spikes in gas demand without price shocks. The Celo protocol aims to have blocks filled at the target density, a certain proportion of the total block gas limit. When blocks are being filled more than the target, the gas price minimum will be raised until demand subsides. If blocks are being filled at less than the target rate, the gas price minimum will decrease until demand rises. The rate of change is determined by a governable parameter, adjustmentspeed.

Calculating Gas Price

In the Celo protocol, the gas price minimum for the next block is calculated based on the current block:

$$\text{gasprice minimum}' = \text{gasprice minimum} \cdot (1 + ((\text{total gas used} / \text{block gas limit}) - \text{target density}) \cdot \text{adjustment speed}) + 1$$

Every transaction is required to pay for gas at or above the gas price minimum in order to be processed. Full nodes will reject transactions whose gas price is below the current gas price minimum, and will discard outstanding transactions if the gas price minimum subsequently falls below the gas price that the transactions specify.

Selecting a Transaction Gas Price

This approach provides a simple mechanism for clients to determine what gas price they should pay. A GasPriceMinimum smart contract provides access to the current gas price minimum. For example, with the parameters specified for the Celo testnets, a gas price of 3x the current gas price minimum will be valid in all scenarios for the following 30 seconds.

When the client wants to ensure that their transaction is processed quickly, they may wish to further increase the gas price to encourage validators proposing new blocks to include it in preference to other transactions.

Transaction Fee Recipients

The required portion of gas fee, known as the base, is set as $\text{base} = \text{gasprice}_{\text{minimum}} \times \text{gasused}$ and is sent to the Gas Fee Handler smart contract, which is controlled by governance and handles how the fees are used (e.g., for carbon removal and burning). The rest of the gas fee, known as the tip, is rewarded to the validator that proposes the block. Block producers only receive the tip and not the base of the gas fee, which means that they do not have an incentive to artificially inflate the gas price minimum by flooding the network with transactions.

```
# index.md:
```

```
---
```

```
title: Celo Transactions Overview
```

```
description: Introduction to Celo transactions and gas prices.
```

```
---
```

Transactions

Introduction to Celo transactions and gas prices.

```
---
```

Celo vs Ethereum Transactions

Transactions in the Celo protocol include payments, contract calls, and other operation which modifies state. They are similar to Ethereum transaction with the following key differences.

- Gas prices must meet or exceed the gas price minimum.
- Gas fees may be paid in currencies other than the native CELO.

```
# native-currency.md:
```

```
---
```

```
title: Celo Native Currency
```

```
description: Introduction to CELO and its compliance to the ERC20 standard.
```

```
---
```

Native Currency

Introduction to CELO and its compliance to the ERC20 standard.

```
---
```

What is CELO?

The native currency in the Celo protocol, CELO, conforms to the ERC20 interface. This is made possible by way of a permissioned "Transfer" precompile, which only the CELO ERC20 smart contract can call. The address of the contract exposing this interface can be looked up via the Registry smart contract, and has the "GoldToken" identifier.

:::tip note

As the native currency of the protocol, CELO, much like Ether, can still be sent directly via transactions by specifying a non-zero "value", bypassing the ERC20 interface.

:::

tx-comment-encryption.md:

title: Celo Encrypted Payment Comments
description: Overview of encrypted payment comments and its technical details related to symmetric and asymmetric encryption.

Encrypted Payment Comments

Overview of encrypted payment comments and its technical details related to symmetric and asymmetric encryption.

Introduction to Comment Encryption

As part of Celo's identity protocol, a public encryption key is stored along with a user's address in the Accounts contract.

Both the address key pair and the encryption key pair are derived from the backup phrase. When sending a transaction the encryption key of the recipient is retrieved when getting his or her address. The comment is then encrypted using a 128 bit hybrid encryption scheme (ECDH on secp256k1 with AES-128-CTR). This system ensures that comments can only be read by the sending and receiving parties and that messages will be recovered when restoring a wallet from its backup phrase.

Comment Encryption Technical Details

A 128 bit randomly generated session key, *sk*, is generated and used to symmetrically encrypt the comment. *sk* is asymmetrically encrypted to the sender and to the recipient.

Encrypted = ECIES(*sk*, to=pubSelf) | ECIES(*sk*, to=pubOther) | AES(ke=*sk*, km=*sk*, comment)

Symmetric Encryption \ (AES-128-CTR\)

- Takes encryption key, ke, and MAC key, km, and the data to encrypt, plaintext
- Cipher: AES-128-CTR using a randomly generated iv
- Authenticate iv \| ciphertext using HMAC with SHA-256 and km
- Return iv \| ciphertext \| mac

Asymmetric Encryption \ (ECIES\)

1. Takes data to encrypt, plaintext, and the public key of the recipient, pubKeyTo
2. Generate an ephemeral keypair, ephemPubKey and ephemPrivKey
3. Derive 32 bytes of key material, k, from ECDH between ephemPrivKey and pubKeyTo using ConcatKDF \ (specified as NIST 800-56C Rev 1 One Step KDF\) with SHA-256 for H \ (x\)
4. The encryption key, ke, is the first 128 bits of k
5. The MAC key, km, is SHA-256 of the second 128 bits of k
6. Encrypt the plaintext symmetrically with AES-128-CTR using ke, km, and a random iv
7. Return ephemPubKey \| AES-128-CTR-HMAC \ (ke, km, plaintext\) where the public key needs to be uncompressed \ (current limitation with decrypt\) .

celo-foundation-voting-policy.md:

title: Celo Foundation Voting Policy

description: How the Celo Foundation anticipates allocating its votes to validator groups, with special attention to the first allocated groups at the Celo Mainnet release and the months thereafter.

How the Celo Foundation anticipates allocating its votes to validator groups, with special attention to the first allocated groups at the Celo Mainnet release and the months thereafter.

:::info

The policy described here can change at any time as determined by the Foundation Board.

:::

Policy Objectives

The Foundation voting policy aims to:

- Be fair by avoiding preferential treatment to certain groups;
- Vote in-line with the Foundation's purpose, which is to encourage financial inclusion and prosperity for all;
- Encourage professional, secure, and reliable validators;

- Be equal opportunity by enabling new groups to have validators elected; and
- Promote network stability by encouraging a gradual turnover in elected validators instead of abrupt election changes

Process

Every 4 months, the Foundation, through its Board, will distribute a portion of its total available votes to a cohort of validator groups. These validators must meet certain basic standards (details below) and alignment with the Foundation's purpose. The total number of validator groups in a cohort can vary.

Validator groups who will be selected for a cohort (and will thus receive a portion of the Foundation's votes) will be informed by the following (non-exhaustive) considerations:

1. The number of elected validators in earlier cohorts;
2. Network stability;
3. CELO governance participation (e.g., how many CELO holders are actively participating in voting); and
4. The quality of validator group applicants

Each validator group selected in the cohort will receive a portion of the Foundation votes for a period of 12 months. During this period, so long as a validator in the group is not slashed or otherwise engages in misbehavior, the validator group will continue to receive these votes. If the validator group is slashed or engages in misbehavior, however, the votes for that validator group will be withdrawn for the remainder of the period. If the validator group is slashed, it may reapply to the Foundation after a 6 month period. In addition, the Foundation may also withdraw its votes if the validator group or the validators in the group fail to meet other standards, including running an attestation service.

Eligibility Criteria

Network Criteria

To support effective and responsible validators, the Foundation considers the following, main criteria for network performance, which must be met by all applicants who receive Foundation votes.

- Zero Slashing Incidents. The validator members of any applying group must not have been slashed within the last 6 months of application. (Note, there are a variety of reasons for slashing, including downtime, security issues, etc. At the outset, and because groups can re-apply at 6 months and 1 day of the slashing, all slashing will be considered equal at this stage)

- Attestation Performance. Ability and commitment to running attestation services with high completion rates.

- Uptime Performance. High performance uptime score over the past 30 days on Mainnet (or Baklava if not elected on Mainnet)

Note: If you are NOT ELECTED on Mainnet, you must be validating on Baklava testnet for at least 30 days. If you are ELECTED you must run validators and attestation service for at least one month (30 days) on Mainnet. If you are ELECTED on mainnet but for less than 30 days, you must be validating on Baklava for 30 days at least.

Supporting Criteria

On top of the main criteria outlined in the previous example, the Foundation considers the following, supporting criteria, which must be met by all applicants who receive Foundation votes:

- Audit Checklist and Self Reporting. As part of the application process, the Foundation will publish a list of recommended validator settings. The members of every group applying will self attest to complying with the recommended checklist.

- Education. An effective validator must be secure. Applicants' members will take an education course. The course must be completed annually.

- Basic Diligence. Because the Foundation holds a substantial number of votes, and its voting may determine whether a validator is elected, the Foundation will conduct a basic diligence process for voted groups. The diligence would include name, location, entity information. This diligence would occur on an annual basis for any group receiving votes.

Additional Criteria

In addition to meeting the main and supporting criteria, outlined above, the Foundation anticipates prioritizing validator groups who are mission aligned and/or will provide greater network resilience. These criteria may include:

- The geographical location of the validator group

- Non-profit organizations

- Organizations who commit to donating a percentage of rewards to non-profit organizations

- The likelihood of the validator group having substantial network support from other voters

This criteria assumes the validators perform well in the main and supporting criteria. It is used as an additional way to evaluate validator applicants assuming there's a limited number of seats in a cohort and that the validators being evaluated all performed well in network performance as outlined in the main criteria.

Application

The following new deadlines will be established for the next 3 cohorts as fixed dates.

Each cohort will last 12 months, there's a 4 months gap between each cohort.

- Cohort 11: November 1 new date for voting in
- Cohort 12: March 1 new date for voting in
- Cohort 13: July 1 new date for voting in

For each cohort, the deadline to apply/be evaluated (if you are reapplying) is exactly 1 month prior to the date of being voted in. So for Cohort 8, it'll be October 1 for the deadline, etc.

New Applicants

Application Prerequisites

Before applying all validator group members should have:

- Important: Run at least one Validator and Validator Group on Baklava
- Important: Run an Attestation Service on Baklava
- Important: Register a validator group on Mainnet and get 150k CELO voted for your validator group
- Completed the Mastering the Art of Validating and Validator Group Marketing courses
- Completed the Security Self Assessment Audit, which includes completing this checklist

Application Details

Before applying be ready to share the following:

- A personal statement telling the Foundation why your group should get votes (max 1,500 characters)
- Validator Group details: email, name, website, address on Mainnet and Baklava, and geographic location
- Information about your team: full names, link to professional profiles such as LinkedIn or GitHub, and an explanation of the team's relevant experience
- Whether your Group:
 - Is validating or has validated in the past 1 month on the Baklava Testnet (Need to provide validator group address and validator address on Baklava)
 - Has been slashed in the past 6 months and if so why (for reapplicants)
 - Members have all completed the online training (see prerequisites)
 - Members have all completed the self-audit (see prerequisites)
- Optional:
 - The list of contributions made to the Celo ecosystem

- Date, audit firm name, and report of your last security audit if your Group has been audited by an external firm in the past 12 months

Reapplicants

If you're part of an existing cohort with expiring votes and interested in reapplying, the re-application process is much more simpler as an existing cohort.

You will receive an email from Celo Foundation asking you if you are interested in reapplying for the new Cohort.

At the application deadline date for new applicants, your validator group will be evaluated on Performance Score and Attestation Score. If you score above the Foundation's threshold, you will be considered for the new cohort along with the new applicants reapplying, limited by seat availability in that cohort. If you don't make the new cohort, you are invited to reapply for the next cohort application.

Cohort Information

Past Foundation votes recipients:

- Cohort 1: The Great Celo Stake Off leaderboard participants at ranking 26-50 -- votes expired on Aug 1, 2020
- Cohort 2: The Great Celo Stake Off leaderboard participants at ranking 1-25 -- votes expired on Nov 1, 2020
- Cohort 3: 6 validator groups -- votes expired on Feb 1, 2021
- Cohort 4: 22 validator groups -- votes expired on May 1, 2021
- Cohort 5: 24 validator groups -- votes expired on November 1, 2021
- Cohort 6: 7 validator groups -- votes will expire on March 1, 2022
- Cohort 7: 23 validator groups -- votes will expire on July 1, 2022
- Cohort 8: 24 validator groups -- votes will expire on November 1, 2022
- Cohort 9: 7 validator groups -- votes will expire on March 1, 2023

Currently receiving Foundation votes:

- Cohort 10: 24 validator groups -- votes will expire on July 1, 2023
- Cohort 11: 24 validator groups -- votes will expire on November 1, 2023
- Cohort 12: 5 validator groups -- votes will expire on March 1, 2024

Coming soon:

- Cohort 13: 24 validator groups -- votes will expire on July 1, 2024

:::info

If you would like to keep up-to-date with all the news happening in the Celo community, including validation, node operation and governance, please sign up to our Celo Signal mailing list [here](#).

You can add the Celo Signal public calendar as well which has relevant dates.

:::

celo-signal.md:

title: Celo Signal

description: How to join the mailing list for everything involving the Celo core community and owners who participate in governance.

Celo Signal

How to join the mailing list for everything involving the Celo core community and owners who participate in governance.

Why join the Mailing List?

If you are a:

- Validator
- Node Operator
- Dapp Running Its Own Node
- Exchange or Custodian
- Owner who is Staking + Participating in Governance
- Core Developer or Contributor

Then Celo Signal Mailing List is a great way to keep up with everything happening in Celo's core community.

Updates include information on following events:

- Celo All-Core Dev Call (where Celo Platform enhancements and proposals are discussed in the community)
- Celo Governance Call (where governance proposals are discussed prior to submission on-chain)
- Celo Foundation Voting Program updates
- celo-blockchain node and attestation service release updates
- Hardfork Updates
- Celo Core Contract Release schedule
- Core Community Happy Hour
- Ongoing or Upcoming Hackathons & Updates

:::note

If you would like to keep up-to-date with all the news happening in the Celo community, including validation, node operation and governance, please sign up to our Celo Signal mailing list [here](#).

:::

:::tip

You can add the Celo Signal public calendar as well which has relevant dates.

:::

devops-best-practices.md:

title: Celo DevOps Best Practices
description: Best practices for running cloud infrastructure for Celo nodes and services.

DevOps Best Practices

Best practices for running cloud infrastructure for Celo nodes and services.

Cloud Infrastructure Best Practices

Node Redundancy

If you are running your celo-blockchain nodes for mainnet in the cloud as a validator, then we recommend having more than one node running.

You can use the redundant validator node as a backup node. It's important that it should only be used as a backup node so you must not enable block-signing with it (to avoid double signing).

In case your primary validator node fails for some reason, then having the redundant node is extremely valuable as you can add the validator keys to it and point it to your proxy to continue signing blocks.

Snapshotting

Another useful thing you can do is enabling snapshotting on your redundant node.

There's no best answer on cadence for snapshotting your redundant node, but one snapshot a week is a good estimate, depending on budget and how the cloud provider charges for snapshotting.

That way, in the event of a node or instance failure on your validator box, which can potentially lead to database failure and requiring you to resync your validator node, then you can use your snapshot as a starting point for syncing and don't have to wait too long to sync.

Kubernetes

We are working on getting a Kubernetes recommended specification and will update this section once we have a recommended spec. If you are using Kubernetes with your validator node, feel free to submit a PR to update this section with your setup.

index.md:

title: Celo Validators

description: Collection of resources to support Validators on the Celo network.

import PageRef from '@components/PageRef'

import Tabs from '@theme/Tabs';

import TabItem from '@theme/TabItem';

Celo Validators

Secure the Celo network by participating in the consensus of the Celo protocol.

Celo Validators participate in the consensus of the Celo protocol. They help secure the Celo network by verifying transactions and proposing blocks to add to the Celo blockchain.

:::tip

Not ready to become a Celo Validator? Learn more about Celo.

:::

Run a Validator

- Run a Baklava Testnet Validator
- Run a Mainnet Validator

Important Information

- Key Management

Nodes and Services

- Securing Celo Nodes and Services
- Upgrading a Node
- Monitoring
- Running Proxies

Validator Tools

- Validator Explorer

Voting Policy

- Celo Foundation Voting Policy

:::tip

For questions, comments, and discussions please use the Celo Forum or Discord.

:::

monitoring.md:

title: Celo Monitoring
description: Commands, metrics, APIs, and services for monitoring Validators and Proxies.

Monitoring

Commands, metrics, APIs, and services for monitoring Validators and Proxies.

Monitoring Validators and Proxies

Logging

Several command line options control logging:

- `--verbosity`: Sets logging verbosity. 3 outputs logs up to INFO level and is recommended. 4 outputs up to DEBUG level; 5 is TRACE.
- `--vmodule`: Overrides this verbosity in specific modules. For example, to configure TRACE level logging of consensus activity, use `consensus/istanbul/=5`.
- `--consoleoutput`: Sends output to the given path, or to stdout.
- (Deprecated in v1.5) `--consoleformat`: Formats logs for easy viewing in a terminal (`term`), or as structured JSON (`json`).
- (Introduced in v1.5) `--log.json`: Formats logs as structured JSON (`true`), or for easy viewing in a terminal (`false`, default option).

Useful messages to record or set up log-based metrics on:

- `msg="Validator Election Results"`: When the last block of any epoch (number) has been agreed, elected shows whether the validator was selected in the validator election.

- msg="Elected but didn't sign block": This validator was elected but did not have its signature included in the block given by number (in fact, in the child's parent seal). This block could count towards downtime if 12 successive blocks are missed.

Metrics

Celo Blockchain inherits go-ethereum's metrics system, but additional Celo-specific metrics have been added.

Metrics reporting is enabled with the `--metrics` flag.

Pull-based metrics are available using the `--pprof` flag. This enables the pprof debugging HTTP server, by default on `http://localhost:6060`. The `--pprof.addr` and `--pprof.port` options can be used to configure the interface and port respectively. If the node is running inside a Docker container, you will need to set `--pprof.addr 0.0.0.0`, then on your Docker command line add `-p 127.0.0.1:6060:6060`.

:::warning

Be sure never to expose the pprof service to the public internet.

:::

Prometheus format metrics are available at `http://localhost:6060/debug/metrics/prometheus`.

ExpVar format metrics are available at `http://localhost:6060/debug/metrics`.

Support for pushing metrics to InfluxDB is available via `--metrics.influxdb` and related flags. This works without the pprof server.

Note that metric name separators differ between these endpoints.

All metrics are soft-state and are cleared when the process is restarted.

Memory metrics

Memory metrics derived from mstats:

- `systemmemoryheld`: Gauge of virtual address space allocated by the Celo Blockchain process, measured in bytes.
- `systemmemoryused`: Gauge of Memory in use by the Celo Blockchain process, measured as bytes of allocated heap objects.
- `systemmemoryallocs`: Counter for memory allocations made, measured in bytes. Consider monitoring the rate.
- `systemmemorypauses`: Counter for stop-the-world Garbage Collection pauses, measured in nanoseconds. Consider monitoring the rate.

CPU metrics

- `systemcpusysload`: Gauge of load average for the system.

- systemcpusyswait: Gauge of IO wait time for the system.
- systemcpuprocload: Gauge of load average for the Celo Blockchain process.

Network metrics

- p2ppeers: The number of connected peers. This should remain at exactly 1 for a proxied validator (just its proxy). It should remain at a relatively steady level for proxy nodes.
- p2pingress: Counter for total inbound traffic, measured in bytes. Consider monitoring the rate.
- p2pegress: Counter for total outbound traffic, measured in bytes. Consider monitoring the rate.
- p2pdials: Counter for outbound connection attempts. Consider monitoring the rate.
- p2pserves: Counter for accepted inbound connection attempts. Consider monitoring the rate.

Blockchain metrics

- chaininsertscount: The count of insertions of new blocks into this node's chain. The rate of this metric should be close to constant at 0.2 /second.

Validator health metrics

A number of metrics are tracked for the parent of the last sealed block received (i.e. this is always two fewer than the current consensus sequence):

- consensusistanbulblockselected: Counts the number of blocks for which this validator has been elected
- consensusistanbulblockssignedbyus: Counts the blocks for which this validator was elected and its signature was included in the seal. This means the validator completed consensus correctly, sent a COMMIT, its commit was received in time to make the seal of the parent received by the next proposer, or was received directly by the next proposer itself, and so the block will not count as downtime. Consider monitoring the rate.
- consensusistanbulblocksmissedbyus: Counts the blocks for which this validator was elected but not included in the child's parent seal (this block could count towards downtime if 12 successive blocks are missed). Consider monitoring the rate.
- consensusistanbulblocksmissedbyusinarow: (since 1.0.2) Counts the blocks for which this validator was elected but not included in the child's parent seal in a row. Consider monitoring the gauge.

- `consensusistanbulblocksproposedbyus`: (since 1.0.2) Counts the blocks for which this validator was elected and for which a block it proposed was successfully included in the chain. Consider monitoring the rate.
- `consensusistanbulblocksdowntimeevent`: (since 1.0.2) Counts the blocks for which this validator was elected and for blocks where it is considered down (occurs when `missedbyusinarow` is ≥ 12). Consider monitoring the rate.

Consensus metrics

- `consensusistanbulcoredesiredround`: Current desired round for this validator, i.e the round we are waiting to see a quorum of validators send `RoundChange` messages for. Usually this value should be 0. Desired rounds increment with each timeout, which backoff exponentially. A value of 5 indicates consensus has stalled for more than 30 seconds. Values above that means the validator is unable to participate in quorum (either because it is disconnected, out of sync, etc, or because of network partition or failure of other validators).
- `consensusistanbulcoreround`: : Current consensus round for this validator, i.e the round for which this validator has received a quorum of `RoundChange` messages. Usually this value should be 0. If this value is less than `consensusistanbulcoredesiredround` the validator is not connected to a quorum of other validators that are also unable to participate (for instance, they did see a proposed block, but this validator did not). If it is equal, it means the validator remains connected to a quorum of other validators but cannot agree on a block.
- `consensusistanbulcoresequence`: Current consensus sequence number, i.e the block number currently being proposed.

Network consensus health metrics

- `consensusistanbulblockstotalsigs`: The number of validators whose signatures were included in the child's parent seal. This can be used to determine how many validators are up and contributing to consensus. If this number falls towards two thirds of validator set size, network block production is at risk.
- `consensusistanbulblocksmisssedrounds`: Sum of the round included in the `parentAggregatedSeal` for the blocks seen. That is, the cumulative number of consensus round changes these blocks needed to make to get to this agreed block. This metric is only incremented when a block is successfully produced after consensus rounds fails, indicating down validators or network issues.
- `consensusistanbulblocksmisssedroundsasproposer`: (since 1.0.2) A meter noting when this validator was elected and could have proposed a block with their signature but did not. In some cases this could be required by the Istanbul BFT protocol.
- `consensusistanbulblocksvalidators`: (since 1.0.2) Total number of validators eligible to sign blocks.

- consensusistanbulcoreconsensuscount: Count and timer for successful completions of consensus (Use quantile tag to find percentiles: 0.5, 0.75, 0.95, 0.99, 0.999)

Management APIs

Celo blockchain inherits and extends go-ethereum's Javascript console, exposing management APIs and web3 DApp APIs.

Connect a client using a variant of the attach command line option:

```
bash
geth attach --datadir DATADIR
geth attach ipc:PATH/TO/geth.ipc
geth attach http://localhost:8545
geth attach ws://localhost:8546
```

<!--

Celo adds specific functions around consensus:

```
bash
```

-->

Community Monitoring Tools

Atalma Signature & Attestation Viewer (Celo Vido)

- Visualizer of current and historic data on validator signatures collected in each block on Mainnet and Baklava.
- Visualizer of current and historic attestation requests and completions, and attestation endpoint versions and status on Mainnet and Baklava.

Virtual Hive Celo Network Validator Exporter

Prometheus exporter that scrapes downtime and meta information for a specified validator signer address from the Celo blockchain. All data is collected from a blockchain node via RPC.

<!--

Monitoring Network Health, Elections, and Accounts

-->

node-upgrade.md:

title: Upgrade a Celo Node

description: How to upgrade to the newest available version of a Celo node.

Upgrade a Node

How to upgrade to the newest available version of a Celo node.

Recent Releases

- You can view the latest releases [here](#).

When an upgrade is required

Upgrades to the Celo node software will often be optional improvements, such as improvements to performance, new useful features, and non-critical bug fixes. Occasionally, they may be required when the upgrade is necessary to continue operating on the network, such as hard forks, or critical bug fixes.

Upgrading a non-validating node

Use these instructions to update non-validating nodes, such as your account node or your attestation node on the Baklava testnet. Also use these instructions to upgrade your proxy node, but remember not to stop the proxy of a running validator.

Pull the latest Docker image

```
bash
export CELOIMAGE=us.gcr.io/celo-org/geth:mainnet
docker pull $CELOIMAGE
```

Stop and remove the existing node

Stop and remove the existing node. Make sure to stop the node gracefully (i.e. giving it time to shut down and complete any writes to disk) or your chain data may become corrupted.

Note: The docker run commands in the documentation have been updated to now include `--stop-timeout 300`, which should make the `-t 300` in docker stop below redundant. However, it is still recommended to include it just in case.

```
bash
docker stop -t 300 celo-fullnode
docker rm celo-fullnode
```

Start the new node

Start the new node using docker run as detailed in the appropriate section of the getting started guide. Remember to recover any environment

variables, if using a new terminal, before running the documented commands.

- Full node
- Accounts node
- Proxy node

Upgrading a Validating Node

Upgrading a validating node is much the same, but requires extra care to be taken to prevent validator downtime.

One option to complete a validating node upgrade is to perform a key rotation onto a new node. Pull the latest Docker image, as mentioned above, then execute a Validator signing key rotation, using the latest image as the new Validator signing node. A recommended procedure for key rotation is documented in the Key Management guide.

A second option is to perform a hot-swap to switch over to a new validator node. The new validator node must be configured with the same set of proxies as the existing validator node.

Hotswapping Validator Nodes

:::info

Hotswap is being introduced in version 1.2.0. When upgrading nodes that are not yet on 1.2.0 refer to the guide to perform a key rotation.

:::

Validators can be configured as primaries or replicas. By default validators start as primaries and will persist all changes around starting or stopping. Through the istanbul management RPC API the validator can be configured to start or stop at a specified block. The validator will participate in consensus for block numbers in the range [start, stop).

:::warning

Note that the replica node must use the same set of proxies as the primary node. If it does not it will not be able to switchover without downtime due to needing to complete the announce protocol from scratch. Replicas behind the same set of proxies as the primary node will be able to switchover without downtime.

:::

RPC Methods

- `istanbul.start()` and `istanbul.startAtBlock()` start validating immediately or at a block
- `istanbul.stop()` and `istanbul.stopAtBlock()` stop validating immediately or at a block

- `istanbul.replicaState` will give you the state of the node and the start/stop blocks
- `istanbul.validating` will give you true/false if the node is validating

:::info

`startAtBlock` and `stopAtBlock` must be given a block in the future.

:::

Geth Flags

- `--istanbul.replica` flag which starts a validator in replica mode.

On startup, nodes will look to see if there is a `replicastate` folder inside it's data directory. If that folder exists the node will configure itself as a validator or replica depending on the previous stored state. The stored state will take precedence over the command line flags. If the folder does not exist the node will store its state as configured by the command line. When RPC calls are made to start or stop validating, those changes will be persisted to the `replicastate` folder.

:::warning

If reconfiguring a node to be a replica or reusing a data directory, make sure that the node was previously configured as replica or that the `replicastate` folder is removed. If there is an existing `replicastate` folder from a node that was not configured as a replica the node will attempt to start validating.

:::

Steps to upgrade

1. Pull the latest docker image.
2. Start a new validator node on a second host in replica mode (`--istanbul.replica` flag). It should be otherwise configured exactly the same as the existing validator.
 - It needs to connect to the existing proxies and the validator signing key to connect to other validators in listen mode.
 - If reconfiguring a node to be a replica or reusing a data directory, make sure that the node was previously configured as replica or that the `replicastate` folder is removed.
3. Once the replica is synced and has validator enode urls for all validators, it is ready to be swapped in.
 - Check validator enode urls with `istanbul.valEnodeTableInfo` in the geth console. The field `enode` should be filled in for each validator peer.
4. In the geth console on the primary run `istanbul.stopAtBlock(XXXX)`
 - Make sure to select a block number comfortably in the future.
 - You can check what the stop block is with `istanbul.replicaState` in the geth console.
 - You can run `istanbul.start()` to clear the stop block
5. In the geth console of the replica run `istanbul.startAtBlock(XXXX)`

- You can check what the start block is with `istanbul.replicaState` in the geth console.
 - You can run `istanbul.stop()` to clear the start block
6. Confirm that the transition occurred with `istanbul.replicaState`
 - The last block that the old primary will sign is block number xxxx - 1
 - The first block that the new primary will sign is block number xxxx
 7. Tear down the old primary once the transition has occurred.

Example geth console on the old primary.

```
bash
> istanbul.replicaState
{
  isPrimary: true,
  startValidatingBlock: null,
  state: "Primary",
  stopValidatingBlock: null
}
> istanbul.stopAtBlock(21000)
null
> istanbul.replicaState
{
  isPrimary: true,
  startValidatingBlock: null,
  state: "Primary in given range",
  stopValidatingBlock: 21000
}
> istanbul.replicaState
{
  isPrimary: false,
  startValidatingBlock: null,
  state: "Replica",
  stopValidatingBlock: null
}
```

Example geth console on the replica being promoted to primary. Not shown is confirming the node is synced and connected to validator peers.

```
bash
> istanbul.replicaState
{
  isPrimary: false,
  startValidatingBlock: null,
  state: "Replica",
  stopValidatingBlock: null
}
> istanbul.startAtBlock(21000)
null
> istanbul.replicaState
{
  isPrimary: false,
  startValidatingBlock: 21000,

```

```

    state: "Replica waiting to start",
    stopValidatingBlock: null
  }
> istanbul.replicaState
{
  isPrimary: true,
  startValidatingBlock: null,
  state: "Primary",
  stopValidatingBlock: null
}

```

Upgrading Proxy Nodes

:::danger

Release 1.2.0 is backwards incompatible in the Validator and Proxy connection. Validators and proxies must be upgraded to 1.2.0 at the same time.

:::

With multi-proxy, you can upgrade proxies one by one or can add newly synced proxies with the latest Docker image and can remove the old proxies. If upgrading the proxies in place, a rolling upgrade is recommended as the validator will re-assign direct connections as proxies are added and removed. These re-assignments will allow the validator to continue to participate in consensus.

proxy.md:

```

---
title: Running Proxies on Celo
description: How to ensure Validator uptime by running proxy nodes.
---

```

Running Proxies

How to ensure Validator uptime by running proxy nodes.

Why run a Proxy?

Validator uptime is essential for the health of the Celo blockchain. To help with validator uptime, operators can use the proxy node, which will provide added security for the validator. It allows the validator to run within a private network, and to communicate to the rest of the Celo network via the proxy.

Also, starting from the Celo client 1.2 release, we will support assigning multiple proxies per validator. This provides better uptime for the validator for the case of a proxy going down. Also, it will help with

making each proxy enode URL less public by only sharing it with a subset of the other validators.

:::danger

The communication protocol between the validator and its proxies implemented in release 1.2 is NOT backwards compatible to the pre-1.2 protocol. So if the proxy or validator is being upgraded to 1.2, then both need to be upgraded to that version. Note that validators and proxies using release 1.2 are still compatible with remote nodes.

:::

There are two ways to specify the proxy information to a validator. It can be done on validator startup via the command line argument, or by the rpc api when the validator is running.

Command Line

Instructions on how to add proxies via the command line is described in the Getting Started guide for mainnet.

RPC API

- `istanbul.addProxy(<proxy's internal enode URL>, <proxy's external enode URL>)` can be used on the validator to add a proxy to the validator's proxy set
- `istanbul.removeProxy(<proxy's internal enode URL>)` can be used on the validator to remove a proxy from the validator's proxy set
- `istanbul.proxies` can be used on the validator to list the validator's proxy set
- `istanbul.proxiedValidators` can be used on the proxies to list the proxied validators

security.md:

title: Run Secure Celo Nodes and Services
description: Recommendations for running secure Celo nodes and services.

Run Secure Nodes and Services

Recommendations for running secure Celo nodes and services.

:::warning

Running Celo nodes and services securely, especially as part of running a validator, is of utmost importance. Failure to do so can lead to severe

consequences including, but not limited to loss of funds, slashing due to double signing, etc.

:::

RPC Endpoints

Celo nodes can be interacted with through an RPC interface for common interactions such as querying the blockchain, inspecting network connectivity and much more. The RPC interface is exposed via HTTP, WebSockets or a local IPC socket. There are two considerations:

1. There is no authentication in the RPC interface. Anyone with access to the interface will be able to execute any actions that are enabled with the command-line options. This includes sensitive RPC modules like `personal` which interacts with the private keys stored on the node (admin is another one). It is not recommended to enable RPC modules unless you explicitly need them. Other RPC modules might be less sensitive but could create unnecessary load on your machine (like the `debug` module) to execute a DoS attack.

2. If you do need access to the RPC modules (for example to use `celocli` or the attestation service), use a firewall and similar mechanisms to restrict access to the RPC interface. You almost never want the interface to be accessible from outside the machine itself.

Public Endpoints

Beyond the RPC interface, Celo nodes and services have other interfaces that actually need to be exposed to the public internet. While varying degrees of protection exist within the software, such as validating attestation requests against the blockchain or monitoring connections in the discovery protocol, additional measures are recommended to reduce the impact of malicious traffic. Examples include, but are not limited to:

- DDoS protection: Protected public endpoints from a DDoS attack is highly recommended to allow valid requests to be served
- Whitelist endpoints: The attestation service exposes a limited number of paths to function correctly. You could use a reverse proxy to reject paths that don't match them.

troubleshooting-faq.md:

title: Celo Validator Troubleshooting FAQ

description: Answers to frequently asked questions while troubleshooting issues as a Validator.

Validator FAQ

Answers to frequently asked questions while troubleshooting issues as a Validator.

How do I reset my local Celo state?

You may desire to reset your local chain state when updating parameters or wishing to perform a clean reset. Note that this will cause the node to resync from the genesis block which will take a couple hours.

```
bash
Remove the celo state directory
sudo rm -rf celo
```

How do I backup a local Celo private key?

It's important that local accounts are properly backed up for disaster recovery. The local keystore files are encrypted with the specified account password and stored in the keystore directory. To copy this file to your local machine you may use ssh:

```
bash
ssh USERNAME@IPADDRESS "sudo cat /root/.celo/keystore/<KEYSTOREFILE>" >
./nodeIdentity
```

You can then back this file up to a cloud storage for redundancy.

:::warning

It's important that you use a strong password to encrypt this file since it will be held in potentially insecure environments.

:::

How do I install and use celocli on my node?

To install celocli on a Linux machine, run the following:

```
bash
sudo apt-get update
sudo apt-get install libusb-1.0-0 -y
sudo npm install -g @celo/celocli --unsafe-perm
```

To install celocli on a Mac/Windows machine, run the following:

```
bash
npm install @celo/celocli
```

You can then run celocli and point it to your local geth.ipc file:

```
bash
```

```
Check if node is synced using celocli
sudo celocli node:syncd --node geth.ipc
```

```
# validator-explorer.md:
```

```
---
title: Validator Explorer
description: How to use the Validator Explorer to view Validator
performance.
---
```

Validator Explorer

How to use the Validator Explorer to view Validator performance.

```
---
```

Introduction to the Explorer

You can interact with the Validator Explorer that allows you to have a complete view of how the different validators are performing. This is one resource voters may use to find validator groups to vote for. The Validator Explorer tool is available in the following address:
<https://celo.org/validators/explore/>

All of the existing validators and groups in the Celo network are included in this view. The default view shows all registered validator groups - if you click on any of the group names it will expand to show the validators affiliated with that group. You can also sort results by each column's value by clicking on the header field.

If you are looking to see how your validator is performing, you should first find the group your validator is affiliated with. Then you can click on the group name to see your validator and the rest of the validators affiliated with this group.

If you are running a validator group, one way to demonstrate your credibility to voters is claiming your validator badges by following the instructions here.

A critical element of this explorer is the Validator Group name, which can help voters recognize organizations or active community members. This name is fetched from the account information registered on-chain for your validator and validator group. In order to combat name impersonation, a group can register a domain claim within their metadata. This verification is done by adding a TXT record to their domain which includes a signature of their domain claim signed by their associated account. This claim is then verified by the validator explorer. Individual users may also verify a claim using `celocli account:get-metadata`.

For example, if a group was run by the owners of example.com, they may want to register their Validator Group with the name Example. The name does not need to be the same as the name of your domain, but for simplicity we do so here. To give credence to this name, they may want to add a DNS claim. They can do this by adding a DNS claim to their metadata, claiming the URL example.com, while simultaneously adding a TXT Record to example.com that includes this claim signed by their group address. Let's go through this example in detail, using a ReleaseGold contract as our validator group.

Assuming you have already deployed your Validator Group via a ReleaseGold contract, you will need these environment variables set to claim your domain.

Environment variables

Variable	Explanation
-----	-----
CELOVALIDATORGROUPRGADDRESS	The ReleaseGold contract address for the Validator Group
CELOVALIDATORRGADDRESS	The ReleaseGold contract address for the Validator
CELOVALIDATORSIGNERADDRESS	The address of the validator signer authorized by the validator account
CELOVALIDATORGROUPSIGNERADDRESS	The address of the validator (group) signer authorized by the validator account

First let's create the metadata file:

```
bash
On your local machine
celocli account:create-metadata ./groupmetadata.json --from
$CELOVALIDATORGROUPRGADDRESS
```

Now we can set the group's name:

```
bash
On your local machine
celocli releasecelo:set-account --contract $CELOVALIDATORGROUPRGADDRESS -
-property name --value Example.com
```

Now we can generate a claim for the domain associated with this name example.com:

```
bash
On your local machine
celocli account:claim-domain ./groupmetadata.json --domain example.com --
from $CELOVALIDATORGROUPSIGNERADDRESS
```

This will output your claim signed under the provided signer address. This output should then be recorded via a TXT Record on your desired domain, so in this case we should add a TXT Record to example.com with this signed output.

You can now view and simultaneously verify the claims on your metadata:

```
bash
On your local machine
celocli account:show-metadata ./groupmetadata.json
```

Take a look at the output and verify these claims look right to you. This tool also automatically verifies the signatures on claims you've added.

Once that record is added, we can then register this metadata under on our Validator Group account for external validation.

Before we do this, you may also want to associate some validators with this domain. The benefit of doing this is to extend your DNS claim to your validators as well, meaning your validators can also verifiably be associated with your domain. You could also do this by adding individual DNS claims for each validator, but this would require separate TXT Records for each, which is inconvenient. Instead, you can simply associate the group and validators together under a single claim.

In order to do so, you will need to claim each validator address on your group's metadata. You will also need to claim your group account on each of your validator's metadata to complete the association. We will run through an example of a single validator now:

First lets claim the validator address from the group account:

```
bash
On your local machine
celocli account:claim-account ./groupmetadata.json --address
$CELOVALIDATORRGADDRESS --from $CELOVALIDATORGROUPSIGNERADDRESS
```

Now let's submit the corresponding claim from the validator account on the group account (note: if you followed the directions to set up the attestation service, you may have already registered metadata for your validator. If that is the case, skip the steps to create the validator's metadata and just add the account claim.)

```
bash
On your local machine
celocli account:create-metadata ./validatormetadata.json --from
$CELOVALIDATORRGADDRESS
celocli account:claim-account ./validatormetadata.json --address
$CELOVALIDATORGROUPRGADDRESS --from $CELOVALIDATORSIGNERADDRESS
```

And then host both metadata files somewhere reachable via HTTP. You can use a service like gist.github.com. Create two gists, each with the contents of the respective files and then click on the Raw button to receive the permalinks to the machine-readable file. If you had already registered a metadata URL for your validator you just need to update that registered gist, so you can skip the validator metadata registration below.

Now we can register these URLs on each account:

```
bash
On your local machine
celocli releasecelo:set-account --contract $CELOVALIDATORGROUPRGADDRESS -
-property metaURL --value <VALIDATORGROUPMETADATAURL>
celocli releasecelo:set-account --contract $CELOVALIDATORRGADDRESS --
-property metaURL --value <VALIDATORMETADATAURL>
```

If everything goes well users should be able to see your claims by running:

```
bash
On your local machine
celocli account:get-metadata $CELOVALIDATORGROUPRGADDRESS
```

If everything went well, you should now have your group and validator associated with each other and with your associated domain!

detailed.md:

```
---
title: Celo Detailed Role Descriptions
description: Detailed description of the various account roles found in
the Celo protocol with examples of how to designate an account as playing
a particular role.
---
```

Detailed Role Descriptions

Detailed descriptions of the various account roles as found in the Celo protocol with examples of how to designate an account as playing a particular role.

Celo Accounts

Any private key generated for use in the Celo protocol has a corresponding address. The account address is the last 20 bytes of the hash of the corresponding public key, just as in Ethereum. Celo account keys can be used to sign and send transactions on the Celo network.

Celo Accounts can be designated as Locked Gold Accounts or authorized as signer keys on behalf of a Locked Gold Account by sending special transactions using celocli. Note that Celo accounts that have not been designated as Locked Gold Accounts or authorized signers may not be able to send certain transactions related to proof-of-stake.

Locked CELO Accounts

Locked CELO Account keys have the highest level of privilege in the Celo protocol. These keys can be used to lock and unlock CELO in order to be used in proof-of-stake. Furthermore, Locked CELO Account keys can be used to authorize other keys to sign transactions and messages on behalf of the Locked CELO Account.

In most cases, the Locked CELO Account key has all the privileges as any authorized signers. For example, if a voter signer is authorized, a user can place votes on behalf of the Locked CELO Account with both the authorized vote signer and the Locked CELO Account.

Because of the significant privileges afforded to the Locked CELO Account, it is best to store this key securely and access it as infrequently as is possible. Authorizing other signers is one way to minimize how frequently you need to access your Locked CELO Account key. The Locked CELO Account key will only be used to send transactions and can be stored on a Ledger hardware wallet.

Creating a Locked CELO Account

A Celo account may be designated as a Locked CELO Account by running the following command:

```
shell
Designate the Celo account as a Locked CELO Account
celocli account:register --from $ADDRESSTODESIGNATE --useLedger
```

```
Confirm the address was designated as a Locked CELO Account
celocli account:show $ADDRESSTODESIGNATE
```

Note that ReleaseGold beneficiary keys are considered vanilla Celo accounts with respect to proof-of-stake, and that the ReleaseGold contract address is what ultimately gets designated as a Locked CELO Account.

Authorized Vote Signers

Any Locked CELO Account may optionally authorize a Celo account as a vote signer. Authorized vote signers can vote for validator groups and for on-chain governance proposals on behalf of the Locked CELO Account.

Note that the vote signer must first generate a "proof-of-possession" indicating that signer's willingness to be authorized on behalf of the Locked CELO Account.

Authorized vote signers can only be used to send voting transactions and can be stored on a Ledger hardware wallet.

Authorizing a Vote Signer

A Celo account may be authorized as a vote signer on behalf of a Locked CELO Account by running the following commands:

```
shell
Create a proof-of-possession. Note that the signer private key must be
available.
celocli account:proof-of-possession --account $LOCKEDGOLDACCOUNT --signer
$SIGNERTOAUTHORIZE --useLedger
```

Authorize the vote signer. Note that the Locked Gold Account private key must be available.

```
celocli account:authorize --from $LOCKEDGOLDACCOUNT --role vote --signer
$SIGNERTOAUTHORIZE --signature $SIGNERPROOFOFPOSSESSION --useLedger
```

Confirm that the vote signer was authorized

```
celocli account:show $LOCKEDGOLDACCOUNT
```

You can also look up account info via the authorized signer

```
celocli account:show $SIGNERTOAUTHORIZE
```

Authorized Validator Signers

Any Locked CELO Account may optionally authorize a Celo account as a validator signer. Authorized validator signers can be used to register and manage a validator or validator group on behalf of the Locked CELO Account. If the authorized validator signer is used to register and run a validator, the signer key is also used to sign consensus messages.

Authorized Validator Signers for Validator Groups

An authorized validator signer key that will be used to register a validator group can be used to send group management transactions (e.g. register, add member A, queue commission update to 0.25, etc.) Because this key does not participate directly in consensus it can be stored on a Ledger hardware wallet.

Authorized Validator Signers for Validators

An authorized validator signer key that will be used to register a validator can be used to send validator management transactions (e.g. register, affiliate with group A, etc.) This key will also be used to sign consensus messages and thus cannot be stored on a Ledger hardware wallet as signing consensus messages is not currently supported by the Celo Ledger App.

Note that the validator signer must first generate a "proof-of-possession" indicating the signer's willingness to be authorized on behalf of the Locked CELO Account.

Authorizing a Validator Signer

A Celo account may be authorized as a validator signer on behalf of a Locked CELO Account by running the following commands:

```
shell
```

Create a proof-of-possession. Note that the signer private key must be available.

Note that the signing key can be kept on a Ledger if it will be used to run a Validator Group.

```
celocli account:proof-of-possession --account $LOCKEDGOLDACCOUNT --signer $SIGNERTOAUTHORIZE
```

Authorize the validator signer. Note that the Locked CELO Account private key must be available.

Note that if a Validator has previously been registered on behalf of the Locked CELO Account it

may be desirable to include the BLS key here as well. Please see the documentation on

validator key rotation for more information.

```
celocli account:authorize --from $LOCKEDGOLDACCOUNT --role validator --signer $SIGNERTOAUTHORIZE --signature $SIGNERPROOFOFPOSSESSION --useLedger
```

Confirm that the vote signer was authorized

```
celocli account:show $LOCKEDGOLDACCOUNT
```

You can also look up account info via the authorized signer

```
celocli account:show $SIGNERTOAUTHORIZE
```

Authorized Validator BLS Signers

The Celo protocol uses BLS signatures in consensus to ultimately determine whether or not a particular block is valid. Many BLS signatures over the same content can be combined into a single "aggregated signature", allowing several kilobytes of signatures to be compressed into fewer than 100 bytes, ensuring that the block headers remain compact and light client friendly.

When registering a Validator on behalf of a Locked CELO Account, users must provide a BLS public key, as well as a proof-of-possession to protect against rogue key attacks.

By default users can derive the BLS key directly from their authorized validator signer key. From a key management and security perspective, this means that the authorized BLS signer key is exactly the same as the authorized validator signer key.

Most users will only need to think about BLS signer keys when registering a validator, or when authorizing a new validator signer after registering a validator. It follows that when a validator authorizes a new validator

signer, the BLS public key and proof-of-possession for the new authorized validator signer should be provided as well.

Advanced users may optionally derive their BLS key separately, but that is out of the scope of this documentation.

Deriving a BLS public key

To derive a BLS public key and proof-of-possession from the authorized validator signer key, and use that information to register a validator, run the following commands:

```
shell
```

Derive the BLS public key and create a proof-of-possession. Note that the signer private key must be available.

Also note that BLS proof-of-possession are not currently supported by celocli

```
docker run -v $PWD:/root/.celo --rm -it $CELOIMAGE account proof-of-possession $AUTHORIZEDVALIDATORSIGNER $LOCKEDGOLDACCOUNT --bls
```

Register the Validator with the authorized validator signer on behalf of the Locked CELO Account

```
celocli validator:register --from $AUTHORIZEDVALIDATORSIGNER --blsKey $BLSSIGNERPUBLICKEY --blsSignature $BLSSIGNERPROOFOFPOSSESSION
```

Confirm that the validator was registered

```
celocli validator:show $LOCKEDGOLDACCOUNT
```

You can also look up the validator via the authorized signer

```
celocli validator:show $AUTHORIZEDVALIDATORSIGNER
```

Authorized Attestation Signers

Any Locked CELO Account may optionally authorize a Celo account as an attestation signer. Authorized attestation signers can sign attestation messages on behalf of the Locked Gold Account in Celo's lightweight identity protocol.

Note that the Celo Ledger App does yet not support signing attestation messages and as such attestation signer keys cannot be stored on a Ledger hardware wallet.

Note that the attestation signer must first be used to generate a "proof-of-possession" indicating the signer's willingness to be authorized on behalf of the Locked Gold Account.

Authorizing an Attestation Signer

A Celo account may be authorized as a vote signer on behalf of a Locked CELO Account by running the following commands:

```
shell
```

Create a proof-of-possession. Note that the signer private key must be available.

```
celocli account:proof-of-possession --account $LOCKEDGOLDACCOUNT --signer $SIGNERTOAUTHORIZE
```

If celocli is unavailable on the attestations node, the proof-of-possession can be generated with celo-blockchain

```
docker run -v $PWD:/root/.celo --rm -it $CELOIMAGE account proof-of-possession $SIGNERTOAUTHORIZE $LOCKEDGOLDACCOUNT
```

Authorize the attestation signer. Note that the Locked CELO Account private key must be available.

```
celocli account:authorize --from $LOCKEDGOLDACCOUNT --role attestations -  
-signer $SIGNERTOAUTHORIZE --signature $SIGNERPROOFOFPOSSESSION --  
useLedger
```

Confirm that the vote signer was authorized

```
celocli account:show $LOCKEDGOLDACCOUNT
```

You can also look up account info via the authorized signer

```
celocli account:show $SIGNERTOAUTHORIZE
```

key-rotation.md:

title: Celo Validator Signer Key Rotation

description: How to manage signer key rotations as a Celo Validator.

Validator Signer Key Rotation

How to manage signer key rotations as a Celo Validator.

Why Rotate Keys?

As detailed in the Celo account roles description page, Celo Locked CELO accounts can authorize separate signer keys for various roles such as voting or validating. This way, if an authorized signer key is lost or compromised, the Locked CELO account can authorize a new signer to replace the old one, without risking the key that custodies funds. This prevents losing an authorized signer key from becoming a catastrophic event. In fact, it is recommended as an operational best practice to regularly rotate keys to limit the impact of keys being silently compromised.

Validator Signer Rotation

Because the Validator signer key is constantly in use to sign consensus messages, special care must be taken when authorizing a new Validator signer key. The following steps detail the recommended procedure for rotating the validator signer key of an active and elected validator:

1. Create a new Validator instance as detailed in the Deploy a Validator section of the getting started documentation. When using a proxy, additionally create a new proxy and peer it with the new validator instance, as described in the same document. Wait for the new instances to sync before proceeding. Please note that when running the proxy, the `-proxy.proxiedvalidatoraddress` flag should reflect the new validator signer address. Otherwise, the proxy will not be able to peer with the validator.

:::warning

Before proceeding to step 2 ensure there is sufficient time until the end of the epoch to complete key rotation.

:::

2. Authorize the new Validator signer key with the Locked CELO Account to overwrite the old Validator signer key.

bash

With `$SIGNERTOAUTHORIZE` as the new validator signer:

On the new validator node which contains the new `$SIGNERTOAUTHORIZE` key
docker run -v `$PWD:/root/.celo` --rm -it `$CELOIMAGE` account proof-of-possession `$SIGNERTOAUTHORIZE` `$VALIDATORACCOUNTADDRESS`
docker run -v `$PWD:/root/.celo` --rm -it `$CELOIMAGE` account proof-of-possession `$SIGNERTOAUTHORIZE` `$VALIDATORACCOUNTADDRESS` --bls

1. If `VALIDATORACCOUNTADDRESS` corresponds to a key you possess:

bash

From a node with access to the key for `VALIDATORACCOUNTADDRESS`
celocli account:authorize --from `$VALIDATORACCOUNTADDRESS` --role validator --signer `$SIGNERTOAUTHORIZE` --signature `0x$SIGNERPROOFOFPOSSESSION` --blsKey `$BLSPUBLICKEY` --blsPop `$BLSPROOFOFPOSSESSION`

2. If `VALIDATORACCOUNTADDRESS` is a ReleaseGold contract:

bash

From a node with access to the beneficiary key of `VALIDATORACCOUNTADDRESS`
celocli releasecelo:authorize --contract `$VALIDATORACCOUNTADDRESS` --role validator --signer `$SIGNERTOAUTHORIZE` --signature `0x$SIGNERPROOFOFPOSSESSION` --blsKey `$BLSPUBLICKEY` --blsPop `$BLSPROOFOFPOSSESSION`

:::warning

Please note that the BLS key will change along with the validator signer ECDSA key on the node. If the new BLS key is not authorized, then the

validator will be unable to process aggregated signatures during consensus, resulting in downtime. For more details, please read the BLS key section of the Celo account role descriptions.

:::

1. Leave all validator and proxy nodes running until the next epoch change. At the start the next epoch, the new Validator signer should take over participation in consensus.

2. Verify that key rotation was successful. Here are some ways to check:

<!-- TODO: The following URL assumes that the user is running against the baklava network. This will need to be updated -->

- Open `baklava-blockscout.celo-testnet.org/address/<SIGNERTOAUTHORIZE>/validations` to confirm that blocks are being proposed.
- Open `baklava-celostats.celo-testnet.org` to confirm that your node is signing blocks.
- Run `celocli validator:signed-blocks --signer $SIGNERTOAUTHORIZE` with the new validator signer address to further confirm that your node is signing blocks.

:::warning

The newly authorized keys will only take effect in the next epoch, so the instance operating with the old key must remain running until the end of the current epoch to avoid downtime.

:::

5. Shut down the validator instance with the now obsolete signer key.

summary.md:

title: Celo Key Management Summary

description: Introduction to the philosophy and account roles related to key management on Celo.

Summary

Introduction to the philosophy and account roles related to key management on Celo.

Quote

> Crypto is a tool for turning a whole swathe of problems into key management problems. Key management problems are way harder than (virtually all) cryptographers think.

>
> @LeaKissner on Twitter

Philosophy

The Celo protocol was designed with the understanding that there is often an inherent tradeoff between the convenience of accessing a private key and the security with which that private key can be custodied. In general Celo is unopinionated about how keys are custodied, but also allows users to authorize private keys with specific, limited privileges. This allows users to custody each private key according to its sensitivity (i.e. what is the impact of this key being lost or stolen?) and usage patterns (i.e. how often and under which circumstances will this key need to be accessed).

Summary

The table below outlines a summary of the various account roles in the Celo protocol. Note that these roles are often mutually exclusive. An account that has been designated as one role can often not be used for a different purpose. Also note that under the hood, all of these accounts are based on secp256k1 ECDSA private keys with the exception of the BLS signer. The different account roles are simply a concept encoded into the Celo proof-of-stake smart contracts, specifically Accounts.sol.

For more details on a specific key type, please see the more detailed sections below.

Role	Description
Ledger compatible	
-----	-----
-----	-----
Celo Account	An account used to send transactions in the Celo protocol
Yes	
Locked CELO Account	Used to lock and unlock CELO and authorize signers
	Yes
Authorized vote signer	Can vote on behalf of a Locked CELO Account
	Yes
Authorized validator (group) signer	Can register and manage a validator group on behalf of a Locked CELO Account
Yes	
Authorized validator signer	Can register, manage a validator, and sign consensus messages on behalf of a Locked CELO Account
	No
Authorized validator BLS signer	Used to sign blocks as a validator
No	
Authorized attestation signer	Can sign attestation messages on behalf of a Locked CELO account
	No

:::warning

A Locked CELO Account may have at most one authorized signer of each type at any time. Once a signer is authorized, the only way to deauthorize that signer is to authorize a new signer that has never previously been used as an authorized signer or Locked CELO Account. It follows then that a newly deauthorized signer cannot be reauthorized.

:::

baklava.md:

title: Run Baklava Testnet Validator on Celo
description: How to get a Validator node running on the Celo Mainnet (Celo's Node Operator Testnet).

Run Baklava Testnet Validator

How to get a Validator node running on the Celo Baklava Testnet (Celo's Node Operator Testnet).

Why run a Baklava Testnet Validator?

The Baklava testnet is the best place to get started running a validator, or test out new validator configurations before deploying to Mainnet.

:::info

If you would like to keep up-to-date with all the news happening in the Celo community, including validation, node operation and governance, please sign up to our Celo Signal mailing list [here](#).

You can add the Celo Signal public calendar as well which has relevant dates.

:::

:::info

If you are transitioning from the Baklava network prior to the June 24 reset, you will need to start with a fresh chain database. You can create new nodes from fresh machines, as described in this guide, or you may delete your chaindata folder, which is named celo in the node data directory, and start over by running the provided init commands for each node described below. All on-chain registration steps, the commands completed with celocli, will need to be run on the new network.

Key differences are:

- New network ID is 62320
- A new image has been pushed to [`us.gcr.io/celo-org/`](https://us.gcr.io/celo-org/)`geth:baklava`
- A new genesis block, bootnode enode, and the new network ID are included in the Docker image

:::

Prerequisites

Staking Requirements

Celo uses a proof-of-stake consensus mechanism, which requires Validators to have locked CELO to participate in block production. The current requirement is 10,000 CELO to register a Validator, and 10,000 CELO per member validator to register a Validator Group.

Participating in the Baklava testnet requires testnet units of CELO, which can only be used in the Baklava testnet. You can request a distribution of testnet CELO by filling out the faucet request form. If you need any help getting started, please join the discussion on Discord or email community@celo.org.

Faucetted funds will come as 2 transactions, one for your validator address and another for your validator group address.

Hardware requirements

The recommended Celo Validator setup involves continually running two instances:

- 1 Validator node: should be deployed to single-tenant hardware in a secure, high availability data center
- 1 Validator Proxy node: can be a VM or container in a multi-tenant environment (e.g. a public cloud), but requires high availability

Celo is a proof-of-stake network, which has different hardware requirements than a Proof of Work network. proof-of-stake consensus is less CPU intensive, but is more sensitive to network connectivity and latency. Below is a list of standard requirements for running Validator and Proxy nodes on the Celo Network:

Validator node

- CPU: At least 4 cores / 8 threads x8664 with 3ghz on modern CPU architecture newer than 2018 Intel Cascade Lake or Ryzen 3000 series or newer with a Geekbench 5 Single Threaded score of >1000 and Multi Threaded score of > 4000
- Memory: 32GB
- Disk: 512GB SSD or NVMe (resizable). Current chain size at August 16th is 190GB, so 512GB is a safe bet for the next 1 year. We recommend using a cloud provider or storage solution that allows you to resize your disk without downtime.

- Network: At least 1 GB input/output Ethernet with a fiber (low latency) Internet connection, ideally redundant connections and HA switches.

Some cloud instances that meet the above requirements are:

- GCP: n2-highmem-4, n2d-highmem-4 or c3-highmem-4
- AWS: r6i.xlarge, r6in.xlarge, or r6a.xlarge
- Azure: StandardE4v5, or StandardE4dv5 or StandardE4asv5

Proxy or Full node

- CPU: At least 4 cores / 8 threads x8664 with 3ghz on modern CPU architecture newer than 2018 Intel Cascade Lake or Ryzen 3000 series or newer with a Geekbench 5 Single Threaded score of >1000 and Multi Threaded score of > 4000
- Memory: 16GB
- Disk: 512GB SSD or NVMe (resizable). Current chain size at August 16th is 190GB, so 512GB is a safe bet for the next 1 year. We recommend using a cloud provider or storage solution that allows you to resize your disk without downtime.
- Network: At least 1 GB input/output Ethernet with a fiber (low latency) Internet connection, ideally redundant connections and HA switches.

Some cloud instances that meet the above requirements are:

- GCP: n2-standard-4, n2d-standard-4 or c3-standard-4
 - AWS: M6i.xlarge, M6in.xlarge, or M6a.xlarge
 - Azure: StandardD4v5, or StandardD4v4 or StandardD4asv5
- In addition, to get things started, it will be useful to run a node on your local machine that you can issue CLI commands against.

Networking requirements

In order for your Validator to participate in consensus, it is critically important to configure your network correctly.

Your Proxy node must have static, external IP addresses, and your Validator node must be able to communicate with the Proxy, either via an internal network or via the Proxy's external IP address.

On the Validator machine, port 30503 should accept TCP connections from the IP address of your Proxy machine. This port is used by the Validator to communicate with the Proxy.

On the Proxy machine, port 30503 should accept TCP connections from the IP address of your Validator machine. This port is used by the Proxy to communicate with the Validator.

On the Proxy machine, port 30303 should accept TCP and UDP connections from all IP addresses. This port is used to communicate with other nodes in the network.

To illustrate this, you may refer to the following table:

Machine	IPs open to	0\0\0\0\0\0 \ (all\)	your\-validator\ -ip	your\-proxy\ -ip
Validator				
tcp:30503				
Proxy		tcp:30303, udp:30303	tcp:30503	

Software requirements

On each machine

- You have Docker installed.

If you don't have it already, follow the instructions here: [Get Started with Docker](#). It will involve creating or signing in with a Docker account, downloading a desktop app, and then launching the app to be able to use the Docker CLI. If you are running on a Linux server, follow the instructions for your distro here. You may be required to run Docker with sudo depending on your installation environment.

You can check you have Docker installed and running if the command `docker info` works properly.

On your local machine

- You have celocli installed.

See Command Line Interface (CLI) for instructions on how to get set up.

- You are using the latest Node.js 12.x

Some users have reported issues using the most recent version of node. Use the LTS for greater reliability.

:::info

A note about conventions:

The code snippets you'll see on this page are bash commands and their output.

When you see text in angle brackets `< >`, replace them and the text inside with your own value of what it refers to. Don't include the `< >` in the command.

:::

Key Management

Private keys are the central primitive of any cryptographic system and need to be handled with extreme care. Loss of your private key can lead to irreversible loss of value.

This guide contains a large number of keys, so it is important to understand the purpose of each key. Read more about key management.

Unlocking

Celo nodes store private keys encrypted on disk with a password, and need to be "unlocked" before use. Private keys can be unlocked in two ways:

1. By running the `celocli account:unlock` command. Note that the node must have the "personal" RPC API enabled in order for this command to work.
2. By setting the `--unlock` flag when starting the node.

It is important to note that when a key is unlocked you need to be particularly careful about enabling access to the node's RPC APIs.

Environment variables

There are a number of environment variables in this guide, and you may use this table as a reference.

Variable	Explanation
-----	-----
CELOIMAGE	The Docker image used for the Validator and proxy containers
CELOVALIDATORGROUPADDRESS	The account address for the Validator Group
CELOVALIDATORADDRESS	The account address for the Validator
CELOVALIDATORGROUPSIGNERADDRESS	The validator (group) signer address authorized by the Validator Group account.
CELOVALIDATORGROUPSIGNERSIGNATURE	The proof-of-possession of the Validator Group signer key
CELOVALIDATORSIGNERADDRESS	The validator signer address authorized by the Validator Account
CELOVALIDATORSIGNERPUBKEY	The ECDSA public key associated with the Validator signer address
CELOVALIDATORSIGNERSIGNATURE	The proof-of-possession of the Validator signer key
CELOVALIDATORSIGNERBLSPUBKEY	The BLS public key for the Validator instance

CELOVALIDATORSIGNERBLSSIGNATURE	A proof-of-possession of the BLS public key
CELOVALIDATORGROUPVOTESIGNERADDRESS	The address of the Validator Group vote signer
CELOVALIDATORGROUPVOTESIGNERPUBKEY	The ECDSA public key associated with the Validator Group vote signer address
CELOVALIDATORGROUPVOTESIGNERSIGNATURE	The proof-of-possession of the Validator Group vote signer key
CELOVALIDATORVOTESIGNERADDRESS	The address of the Validator vote signer
CELOVALIDATORVOTESIGNERPUBKEY	The ECDSA public key associated with the Validator vote signer address
CELOVALIDATORVOTESIGNERSIGNATURE	The proof-of-possession of the Validator vote signer key
PROXYENODE	The enode address for the Validator proxy
PROXYINTERNALIP	(Optional) The internal IP address over which your Validator can communicate with your proxy
PROXYEXTERNALIP	The external IP address of the proxy. May be used by the Validator to communicate with the proxy if PROXYINTERNALIP is unspecified
DATABASEURL	The URL under which your database is accessible, currently supported are postgres://, mysql:// and sqlite://

Validator Node Setup

This section outlines the steps needed to configure your Proxy and Validator nodes so that they are ready to sign blocks once elected.

Environment Variables

First we are going to set up the main environment variables related to the Baklava network. Run these on both your Validator and Proxy machines:

```
bash
export CELOIMAGE=us.gcr.io/celo-org/geth:baklava
```

Pull the Celo Docker image

In all the commands we are going to see the CELOIMAGE variable to refer to the Docker image to use. Now we can get the Docker image on your Validator and Proxy machines:

```
bash
docker pull $CELOIMAGE
```

The `us.gcr.io/celo-org/geth:baklava` image contains the genesis block in addition to the Celo Blockchain binary.

Account Creation

```
:::info
```

Please complete this section if you are new to validating on Celo.

```
:::
```

Account and Signer keys

Running a Celo Validator node requires the management of several different keys, each with different privileges. Keys that need to be accessed frequently (e.g. for signing blocks) are at greater risk of being compromised, and thus have more limited permissions, while keys that need to be accessed infrequently (e.g. for locking CELO) are less onerous to store securely, and thus have more expansive permissions. Below is a summary of the various keys that are used in the Celo network, and a description of their permissions.

Name of the key	Purpose
-----	-----
-----	-----
-----	-----
-----	-----
Account key	This is the key with the highest level of permissions, and is thus the most sensitive. It can be used to lock and unlock CELO, and authorize vote and validator keys. Note that the account key also has all of the permissions of the other keys.
Validator signer key	This is the key that has permission to register and manage a Validator or Validator Group, and participate in BFT consensus.
Vote signer key	This key can be used to vote in Validator elections and on-chain governance.

Note that Account and all the signer keys must be unique and may not be reused.

Generating Validator and Validator Group Keys

First, you'll need to generate account keys for your Validator and Validator Group.

```
:::danger
```

These keys will control your locked CELO, and thus should be handled with care.
Store and back these keys up in a secure manner, as there will be no way to recover them if lost or stolen.

:::

```
bash
On your local machine
mkdir celo-accounts-node
cd celo-accounts-node
docker run -v $PWD:/root/.celo --rm -it $CELOIMAGE account new
docker run -v $PWD:/root/.celo --rm -it $CELOIMAGE account new
```

This will create a new keystore in the current directory with two new accounts.
Copy the addresses from the terminal and set the following environment variables:

```
bash
On your local machine
export CELOVALIDATORGROUPADDRESS=<YOUR-VALIDATOR-GROUP-ADDRESS>
export CELOVALIDATORADDRESS=<YOUR-VALIDATOR-ADDRESS>
```

Start your Accounts node

Next, we'll run a node on your local machine so that we can use these accounts to lock CELO and authorize the keys needed to run your validator.

To run the node:

```
bash
On your local machine
mkdir celo-accounts-node
cd celo-accounts-node
docker run --name celo-accounts -it --restart always --stop-timeout 300 -
p 127.0.0.1:8545:8545 -v $PWD:/root/.celo $CELOIMAGE --verbosity 3 --
syncmode full --http --http.addr 0.0.0.0 --http.api
eth,net,web3,debug,admin,personal --baklava --light.serve 0 --datadir
/root/.celo
```

:::danger

Security: The command line above includes the parameter `--http.addr 0.0.0.0` which makes the Celo Blockchain software listen for incoming RPC requests on all network adaptors. Exercise extreme caution in doing this when running outside Docker, as it means that any unlocked accounts and their funds may be accessed from other machines on the Internet. In the context of running a Docker container on your local machine, this together with the `docker -p 127.0.0.1:localport:containerport` flags

allows you to make RPC calls from outside the container, i.e from your local host, but not from outside your machine. Read more about Docker Networking [here](#).

:::

Deploy a Validator

To actually register as a validator, we'll need to generate a validating signer key. On your Validator machine (which should not be accessible from the public internet), follow very similar steps:

bash

On the validator machine

Note that you have to export \$CELOIMAGE on this machine

```
export CELOIMAGE=us.gcr.io/celo-org/geth:baklava
```

```
mkdir celo-validator-node
```

```
cd celo-validator-node
```

```
docker run -v $PWD:/root/.celo --rm -it $CELOIMAGE account new
```

```
export CELOVALIDATORSIGNERADDRESS=<YOUR-VALIDATOR-SIGNER-ADDRESS>
```

Proof-of-Possession

:::info

Please complete this step if you are running a validator on Celo for the first time.

:::

In order to authorize our Validator signer, we need to create a proof that we have possession of the Validator signer private key. We do so by signing a message that consists of the Validator account address. To generate the proof-of-possession, run the following command:

bash

On the validator machine

Note that you have to export CELOVALIDATORADDRESS on this machine

```
export CELOVALIDATORADDRESS=<CELO-VALIDATOR-ADDRESS>
```

```
docker run -v $PWD:/root/.celo --rm -it $CELOIMAGE account proof-of-possession $CELOVALIDATORSIGNERADDRESS $CELOVALIDATORADDRESS
```

Save the signer address, public key, and proof-of-possession signature to your local machine:

bash

On your local machine

```
export CELOVALIDATORSIGNERADDRESS=<YOUR-VALIDATOR-SIGNER-ADDRESS>
```

```
export CELOVALIDATORSIGNERSIGNATURE=<YOUR-VALIDATOR-SIGNER-SIGNATURE>
```

```
export CELOVALIDATORSIGNERPUBKEY=<YOUR-VALIDATOR-SIGNER-PUBLIC-KEY>
```


Validators on the Celo network use BLS aggregated signatures to create blocks in addition to the Validator signer (ECDSA) key. While an independent BLS key can be specified, the simplest thing to do is to derive the BLS key from the Validator signer key. When we register our Validator, we'll need to prove possession of the BLS key as well, which can be done by running the following command:

```
bash
On the validator machine
docker run -v $PWD:/root/.celo --rm -it $CELOIMAGE account proof-of-
possession $CELOVALIDATORSIGNERADDRESS $CELOVALIDATORADDRESS --bls
```

Save the resulting signature and public key to your local machine:

```
bash
On your local machine
export CELOVALIDATORSIGNERBLSSIGNATURE=<YOUR-VALIDATOR-SIGNER-SIGNATURE>
export CELOVALIDATORSIGNERBLSPUBLICKEY=<YOUR-VALIDATOR-SIGNER-BLS-PUBLIC-
KEY>
```

We'll get back to this machine later, but for now, let's give it a proxy.

Deploy a proxy

```
bash
On the proxy machine
mkdir celo-proxy-node
cd celo-proxy-node
export CELOVALIDATORSIGNERADDRESS=<YOUR-VALIDATOR-SIGNER-ADDRESS>
```

You can then run the proxy with the following command. Be sure to replace <YOUR-VALIDATOR-NAME> with the name you'd like to appear on Celostats. The validator name shown in Celostats will be the name configured in the proxy.

Additionally, you need to unlock the account configured in the etherbase option. It is recommended to create a new account and independent account only for this purpose. Be sure to write a new password to `./password` for this account (different to the Validator Signer password)

```
bash
On the proxy machine
First, we create a new account for the proxy
docker run --name celo-proxy-password -it --rm -v $PWD:/root/.celo
$CELOIMAGE account new --password /root/.celo/.password
```

Notice the public address returned by this command, that can be exported and used for running the proxy node:

```
bash
```

On the proxy machine

```
export PROXYADDRESS=<PROXY-PUBLIC-ADDRESS>
docker run --name celo-proxy -it --restart unless-stopped --stop-timeout
300 -p 30303:30303 -p 30303:30303/udp -p 30503:30503 -p 30503:30503/udp -
v $PWD:/root/.celo $CELOIMAGE --verbosity 3 --syncmode full --proxy.proxy
--proxy.proxiedvalidatoraddress $CELOVALIDATORSIGNERADDRESS --
proxy.internalendpoint :30503 --etherbase $PROXYADDRESS --unlock
$PROXYADDRESS --password /root/.celo/.password --allow-insecure-unlock --
baklava --datadir /root/.celo --celostats=<YOUR-VALIDATOR-NAME>@baklava-
celostats-server.celo-testnet.org
```

:::info

You can detach from the running container by pressing ctrl+p ctrl+q, or start it with -d instead of -it to start detached. Access the logs for a container in the background with the docker logs command.

:::

NOTES

- For the proxy to be able to send stats to Celostats, both the proxy and the validator should set the celostats flag
- If you are deploying multiple proxies for the same validator, the celostats flag should be added in only one of them

Get your Proxy's connection info

Once the Proxy is running, we will need to retrieve its enode and IP address so that the Validator will be able to connect to it.

bash

On the proxy machine, retrieve the proxy enode

```
docker exec celo-proxy geth --exec
"admin.nodeInfo['enode'].split('///')[1].split('@')[0]" attach | tr -d '"'
```

Now we need to set the Proxy enode and Proxy IP address in environment variables on the Validator machine.

If you don't have an internal IP address over which the Validator and Proxy can communicate, you can set the internal IP address to the external IP address.

If you don't know your proxy's external IP address, you can get it by running the following command:

bash

On the proxy machine

```
dig +short myip.opendns.com @resolver1.opendns.com
```

Then, export the variables on your Validator machine.

```
bash
On the Validator machine
export PROXYENODE=<YOUR-PROXY-ENODE>
export PROXYEXTERNALIP=<PROXY-MACHINE-EXTERNAL-IP-ADDRESS>
export PROXYINTERNALIP=<PROXY-MACHINE-INTERNAL-IP-ADDRESS>
```

You will also need to export PROXYEXTERNALIP on your local machine.

```
bash
On your local machine
export PROXYEXTERNALIP=<PROXY-MACHINE-EXTERNAL-IP-ADDRESS>
```

Connect the Validator to the Proxy

When your Validator starts up it will attempt to create a network connection with the proxy machine. You will need to make sure that your proxy machine has the appropriate firewall settings to allow the Validator to connect to it.

Specifically, on the proxy machine, port 30303 should allow TCP and UDP connections from all IP addresses. And port 30503 should allow TCP connections from the IP address of your Validator machine.

Test that your network is configured correctly by running the following commands:

```
bash
On your local machine, test that your proxy is accepting TCP connections
over port 30303.
Note that it will also need to be accepting UDP connections over this
port.
nc -vz $PROXYEXTERNALIP 30303
```

```
bash
On your Validator machine, test that your proxy is accepting TCP
connections over port 30503.
nc -vz $PROXYINTERNALIP 30503
```

Once that is completed, go ahead and run the Validator. Be sure to write your Validator signer password to `./password` for the following command to work, or provide your password another way.

```
bash
On the Validator machine
cd celo-validator-node
docker run --name celo-validator -it --restart unless-stopped --stop-
timeout 300 -p 30303:30303 -p 30303:30303/udp -v $PWD:/root/.celo
$CELOIMAGE --verbosity 3 --syncmode full --mine --etherbase
$CELOVALIDATORSIGNERADDRESS --nodiscover --proxy.proxied --
```

```
proxy.proxyenodeurlpairs=enode://$PROXYENODE@$PROXYINTERNALIP:30503\;enode://$PROXYENODE@$PROXYEXTERNALIP:30303 --
unlock=$CELOVALIDATORSIGNERADDRESS --password /root/.celo/.password --
celostats=<YOUR-VALIDATOR-NAME>@baklava-celostats-server.celo-testnet.org
--baklava --datadir /root/.celo
```

At this point your Validator and Proxy machines should be configured, and both should be syncing to the network. You should see Imported new chain segment in your node logs, about once every 5 seconds once the node is synced to the latest block which you can find on the Baklava Network Stats page.

:::info

You can run multiple proxies by deploying additional proxies per the instructions in the Deploy a proxy section. Then add all of the proxies' enodes as a comma separated list using the --proxy.proxyenodeurlpairs option. E.g. if there are two proxies, that option's usage would look like --

```
proxy.proxyenodeurlpairs=enode://$PROXYENODE1@$PROXYINTERNALIP1:30503\;enode://$PROXYENODE1@$PROXYEXTERNALIP1:30303,enode://$PROXYENODE2@$PROXYINTERNALIP2:30503\;enode://$PROXYENODE2@$PROXYEXTERNALIP2:30303
```

:::

Registering as a Validator

Register the Accounts

You've now done all the infrastructure setup to get a validator and proxy running. The cLabs team will review your submission to receive funds and send you 12,000 Baklava testnet CELO to each of your Validator and Validator Group account addresses. These funds have no real world value but will allow you to submit transactions to the network via celocli and put up a stake to register as a validator and validator group.

You can view your CELO balances by running the following commands:

```
bash
On your local machine
celocli account:balance $CELOVALIDATORGROUPADDRESS
celocli account:balance $CELOVALIDATORADDRESS
```

At some point the output of these commands will change from 0 to 12e12, indicating you have received the testnet CELO. This process involves a human, so please be patient. If you haven't received a balance within 24 hours, please get in touch again.

You can also look at an account's current balance and transaction history on Blockscout. Enter the address into the search bar.

Once these accounts have a balance, unlock them so that we can sign transactions. Then, we will register the accounts with the Celo core smart contracts:

bash

On your local machine

```
celocli account:unlock $CELOVALIDATORGROUPADDRESS
```

```
celocli account:unlock $CELOVALIDATORADDRESS
```

```
celocli account:register --from $CELOVALIDATORGROUPADDRESS --name <NAME  
YOUR VALIDATOR GROUP>
```

```
celocli account:register --from $CELOVALIDATORADDRESS --name <NAME YOUR VALIDATOR>
```

Check that your accounts were registered successfully with the following commands:

```
bash
```

On your local machine

```
celocli account:show $CELOVALIDATORGROUPADDRESS
```

```
celocli account:show $CELOVALIDATORADDRESS
```

Lock up CELO

Lock up testnet CELO for both accounts in order to secure the right to register a Validator and Validator Group. The current requirement is 10,000 CELO to register a validator, and 10,000 CELO per member validator to register a Validator Group. For Validators, this gold remains locked for approximately 60 days following deregistration. For groups, this gold remains locked for approximately 60 days following the removal of the Nth validator from the group.

```
bash
```

On your local machine

```
celcli lockedgold:lock --from $CELOVALIDATORGROUPADDRESS --value
10000000000000000000000000
```

```
celocli lockedgold:lock --from $CELOVALIDATORADDRESS --value  
1000000000000000000000
```

This amount (10,000 CELO) represents the minimum amount needed to be locked in order to register a Validator and Validator group. Since your balance is in fact higher than this, you may wish to lock more with these accounts. Note that you will want to be sure to leave enough CELO unlocked to be able to continue to pay transaction fees for future transactions (such as those issued by running some CLI commands).

Check that your CELO was successfully locked with the following commands:

bash

On your local machine

```
celocli lockedgold:show $CELOVALIDATORGROUPADDRESS
```

```
celocli lockedgold:show $CELOVALIDATORADDRESS
```

Run for election

In order to be elected as a Validator, you will first need to register your group and Validator. Note that when registering a Validator Group, you need to specify a commission, which is the fraction of epoch rewards paid to the group by its members.

We don't want to use our account key for validating, so first let's authorize the validator signing key:

```
bash
On your local machine
celocli account:authorize --from $CELOVALIDATORADDRESS --role validator -
--signature 0x$CELOVALIDATORSIGNERSIGNATURE --signer
0x$CELOVALIDATORSIGNERADDRESS
```

Confirm by checking the authorized Validator signer for your Validator:

```
bash
On your local machine
celocli account:show $CELOVALIDATORADDRESS
```

Then, register your Validator Group by running the following command. Note that because we did not authorize a Validator signer for our Validator Group account, we register the Validator Group with the account key.

```
bash
On your local machine
celocli validatorgroup:register --from $CELOVALIDATORGROUPADDRESS --
commission 0.1
```

You can view information about your Validator Group by running the following command:

```
bash
On your local machine
celocli validatorgroup:show $CELOVALIDATORGROUPADDRESS
```

Next, register your Validator by running the following command. Note that because we have authorized a Validator signer, this step could also be performed on the Validator machine. Running it on the local machine allows us to avoid needing to install the celocli on the Validator machine.

```
bash
On your local machine
```

```
celocli validator:register --from $CELOVALIDATORADDRESS --ecdsaKey
$CELOVALIDATORSIGNERPUBLICKEY --blsKey $CELOVALIDATORSIGNERBLSPUBLICKEY -
-blsSignature $CELOVALIDATORSIGNERBLSSIGNATURE
```

Affiliate your Validator with your Validator Group. Note that you will not be a member of this group until the Validator Group accepts you. This command could also be run from the Validator signer, if running on the validator machine.

```
bash
On your local machine
celocli validator:affiliate $CELOVALIDATORGROUPADDRESS --from
$CELOVALIDATORADDRESS
```

Accept the affiliation:

```
bash
On your local machine
celocli validatorgroup:member --accept $CELOVALIDATORADDRESS --from
$CELOVALIDATORGROUPADDRESS
```

Next, double check that your Validator is now a member of your Validator Group:

```
bash
On your local machine
celocli validator:show $CELOVALIDATORADDRESS
celocli validatorgroup:show $CELOVALIDATORGROUPADDRESS
```

Use both accounts to vote for your Validator Group. Note that because we have not authorized a vote signer for either account, these transactions must be sent from the account keys. Since you're likely to need to place additional votes throughout the course of the stake-off, consider creating and authorizing vote signers for additional operational security.

```
bash
On your local machine
celocli election:vote --from $CELOVALIDATORADDRESS --for
$CELOVALIDATORGROUPADDRESS --value 1000000000000000000000000
celocli election:vote --from $CELOVALIDATORGROUPADDRESS --for
$CELOVALIDATORGROUPADDRESS --value 1000000000000000000000000
```

Double check that your votes were cast successfully:

```
bash
On your local machine
celocli election:show $CELOVALIDATORGROUPADDRESS --group
celocli election:show $CELOVALIDATORGROUPADDRESS --voter
```

```
celocli election:show $CELOVALIDATORADDRESS --voter
```

Users in the Celo protocol receive epoch rewards for voting in Validator Elections only after submitting a special transaction to enable them. This must be done every time new votes are cast, and can only be made after the most recent epoch has ended. For convenience, we can use the following command, which will wait until the epoch has ended before sending a transaction:

```
bash
```

On your local machine

Note that this may take some time, as the epoch needs to end before votes can be activated

```
celocli election:activate --from $CELOVALIDATORADDRESS --wait && celocli
```

```
election:activate --from $CELOVALIDATORGROUPADDRESS --wait
```

Check that your votes were activated by re-running the following commands:

```
bash
```

On your local machine

```
celocli election:show $CELOVALIDATORGROUPADDRESS --voter
```

```
celocli election:show $CELOVALIDATORADDRESS --voter
```

If your Validator Group elects validators, you will receive epoch rewards in the form of additional Locked CELO voting for your Validator Group from your account addresses. You can see these rewards accumulate with the commands in the previous set, as well as:

```
bash
```

On your local machine

```
celocli lockedgold:show $CELOVALIDATORGROUPADDRESS
```

```
celocli lockedgold:show $CELOVALIDATORADDRESS
```

You're all set! Elections are finalized at the end of each epoch, roughly once an hour in the Alfajores or Baklava Testnets. After that hour, if you get elected, your node will start participating BFT consensus and validating blocks. After the first epoch in which your Validator participates in BFT, you should receive your first set of epoch rewards.

```
:::info
```

Roadmap: Different parameters will govern elections in a Celo production network. Epochs are likely to be daily, rather than hourly. Running a Validator will also include setting up proxy nodes to protect against DDoS attacks, and using hardware wallets to secure the key used to sign blocks. We plan to update these instructions with more details soon.

```
:::
```


You can inspect the current state of the validator elections by running:

```
bash
On your local machine
celocli election:list
```

If you find your Validator still not getting elected you may need to faucet yourself more funds and lock more gold in order to be able to cast more votes for your Validator Group!

You can check the status of your validator, including whether it is elected and signing blocks, at baklava-ethstats.celo-testnet.org or by running:

```
bash
On your local machine with celocli >= 0.0.30-beta9
celocli validator:status --validator $CELOVALIDATORADDRESS
```

You can see additional information about your validator, including uptime score, by running:

```
bash
On your local machine
celocli validator:show $CELOVALIDATORADDRESS
```

Deployment Tips

Running the Docker containers in the background

There are different options for executing Docker containers in the background. The most typical one is to use in your docker run commands the `-d` option. Also for long running processes, especially when you run in on a remote machine, you can use a tool like `screen`. It allows you to connect and disconnect from running processes providing an easy way to manage long running processes.

It's out of the scope of this documentation to go through the `screen` options, but you can use the following command format with your docker commands:

```
bash
screen -S <SESSION NAME> -d -m <YOUR COMMAND>
```

For example:

```
bash
screen -S celo-validator -d -m docker run --name celo-validator -it --
restart unless-stopped --stop-timeout 300 -p 127.0.0.1:8545:8545 .....
```

You can list your existing screen sessions:

```
bash
screen -ls
```

And re-attach to any of the existing sessions:

```
bash
screen -r -S celo-validator
```

Stopping containers

You can stop the Docker containers at any time without problem. If you stop your containers that means those containers stop providing service. The data directory of the Validator and the proxy are Docker volumes mounted in the containers from the celo--dir you created at the very beginning. So if you don't remove that folder, you can stop or restart the containers without losing any data.

It is recommended to use the Docker stop timeout parameter -t when stopping the containers. This allows time, in this case 60 seconds, for the Celo nodes to flush recent chain data it keeps in memory into the data directories. Omitting this may cause your blockchain data to corrupt, requiring the node to start syncing from scratch.

You can stop the celo-validator and celo-proxy containers running:

```
bash
docker stop celo-validator celo-proxy -t 60
```

And you can remove the containers (not the data directory) by running:

```
bash
docker rm -f celo-validator celo-proxy
```

mainnet.md:

title: Running a Celo Validator

description: How to get a Validator node running on the Celo Mainnet.

Running a Validator

How to get a Validator node running on the Celo Mainnet.

:::info

If you would like to keep up-to-date with all the news happening in the Celo community, including validation, node operation and governance, please sign up to our Celo Signal mailing list [here](#).

You can add the Celo Signal public calendar as well which has relevant dates.

:::

What is a Validator?

Validators help secure the Celo network by participating in Celo's proof-of-stake protocol. Validators are organized into Validator Groups, analogous to parties in representative democracies. A Validator Group is essentially an ordered list of Validators.

Just as anyone in a democracy can create their own political party, or seek to get selected to represent a party in an election, any Celo user can create a Validator group and add themselves to it, or set up a potential Validator and work to get an existing Validator group to include them.

While other Validator Groups will exist on the Celo Network, the fastest way to get up and running with a Validator will be to register a Validator Group, register a Validator, and affiliate that Validator with your Validator Group. The addresses used to register Validator Groups and Validators must be unique, which will require that you create two accounts in the step-by-step guide below.

Because of the importance of Validator security and availability, Validators are expected to run a "proxy" node in front of each Validator node. In this setup, the Proxy node connects with the rest of the network, and the Validator node communicates only with the Proxy, ideally via a private network.

Read more about Celo's mission and why you may want to become a Validator. - This article still uses the term Celo Gold which is the deprecated name for the Celo native asset, which now is referred to simply as "Celo" or preferably "CELO".

Prerequisites

Staking Requirements

Celo uses a proof-of-stake consensus mechanism, which requires Validators to have locked CELO to participate in block production. The current requirement is 10,000 CELO to register a Validator, and 10,000 CELO per member validator to register a Validator Group.

If you do not have the required CELO to lock up, you can try out of the process of creating a validator on the Baklava network by following the [Running a Validator in Baklava guide](#)

We will not discuss obtaining CELO here, but it is a prerequisite that you obtain the required CELO.

Hardware requirements

The recommended Celo Validator setup involves continually running two instances:

- 1 Validator node: should be deployed to single-tenant hardware in a secure, high availability data center
- 1 Validator Proxy node: can be a VM or container in a multi-tenant environment (e.g. a public cloud), but requires high availability

Celo is a proof-of-stake network, which has different hardware requirements than a Proof of Work network. proof-of-stake consensus is less CPU intensive, but is more sensitive to network connectivity and latency. Below is a list of standard requirements for running Validator and Proxy nodes on the Celo Network:

Validator node

- CPU: At least 4 cores / 8 threads x8664 with 3ghz on modern CPU architecture newer than 2018 Intel Cascade Lake or Ryzen 3000 series or newer with a Geekbench 5 Single Threaded score of >1000 and Multi Threaded score of > 4000
- Memory: 32GB
- Disk: 512GB SSD or NVMe (resizable). Current chain size at August 16th is 190GB, so 512GB is a safe bet for the next 1 year. We recommend using a cloud provider or storage solution that allows you to resize your disk without downtime.
- Network: At least 1 GB input/output Ethernet with a fiber (low latency) Internet connection, ideally redundant connections and HA switches.

Some cloud instances that meet the above requirements are:

- GCP: n2-highmem-4, n2d-highmem-4 or c3-highmem-4
- AWS: r6i.xlarge, r6in.xlarge, or r6a.xlarge
- Azure: StandardE4v5, or StandardE4dv5 or StandardE4asv5

Proxy node

- CPU: At least 4 cores / 8 threads x8664 with 3ghz on modern CPU architecture newer than 2018 Intel Cascade Lake or Ryzen 3000 series or newer with a Geekbench 5 Single Threaded score of >1000 and Multi Threaded score of > 4000
- Memory: 16GB
- Disk: 512GB SSD or NVMe (resizable). Current chain size at August 16th is 190GB, so 512GB is a safe bet for the next 1 year. We recommend using a cloud provider or storage solution that allows you to resize your disk without downtime.
- Network: At least 1 GB input/output Ethernet with a fiber (low latency) Internet connection, ideally redundant connections and HA switches.

Some cloud instances that meet the above requirements are:

- GCP: n2-standard-4, n2d-standard-4 or c3-standard-4
- AWS: M6i.xlarge, M6in.xlarge, or M6a.xlarge
- Azure: StandardD4v5, or StandardD4v4 or StandardD4asv5

In addition, to get things started, it will be useful to run a node on your local machine that you can issue CLI commands against.

Networking requirements

In order for your Validator to participate in consensus, it is critically important to configure your network correctly.

Your Proxy node must have static, external IP addresses, and your Validator node must be able to communicate with the Proxy, either via an internal network or via the Proxy's external IP address.

On the Validator machine, port 30503 should accept TCP connections from the IP address of your Proxy machine. This port is used by the Validator to communicate with the Proxy.

On the Proxy machine, port 30503 should accept TCP connections from the IP address of your Validator machine. This port is used by the Proxy to communicate with the Validator.

On the Proxy machine, port 30303 should accept TCP and UDP connections from all IP addresses. This port is used to communicate with other nodes in the network.

To illustrate this, you may refer to the following table:

Machine \ IPs open to	0\0\0\0/0 \ (all\)	your\-validator\-ip	your\-proxy\-ip
Validator			
tcp:30503			
Proxy	tcp:30303, udp:30303	tcp:30503	

Software requirements

On each machine

- You have Docker installed.

If you don't have it already, follow the instructions here: [Get Started with Docker](#). It will involve creating or signing in with a Docker account, downloading a desktop app, and then launching the app to be able to use the Docker CLI. If you are running on a Linux server, follow the instructions for your distro here. You may be required to run Docker with sudo depending on your installation environment.

You can check you have Docker installed and running if the command `docker info` works properly.

On your local machine

- You have celocli installed.

See Command Line Interface (CLI) for instructions on how to get set up.

- You are using the latest Node 10.x LTS

Some users have reported issues using the most recent version of node. Use the LTS for greater reliability.

:::info

A note about conventions:

The code snippets you'll see on this page are bash commands and their output.

When you see text in angle brackets `<>`, replace them and the text inside with your own value of what it refers to. Don't include the `<>` in the command.

:::

Key Management

Private keys are the central primitive of any cryptographic system and need to be handled with extreme care. Loss of your private key can lead to irreversible loss of value.

This guide contains a large number of keys, so it is important to understand the purpose of each key. Read more about key management.

Unlocking

Celo nodes store private keys encrypted on disk with a password, and need to be "unlocked" before use. Private keys can be unlocked in two ways:

1. By running the `celocli account:unlock` command. Note that the node must have the "personal" RPC API enabled in order for this command to work.
2. By setting the `--unlock` flag when starting the node.

It is important to note that when a key is unlocked you need to be particularly careful about enabling access to the node's RPC APIs.

Environment variables

There are a number of environment variables in this guide, and you may use this table as a reference.

Variable	Explanation

	-----		-----

	CELOIMAGE		The Docker image used for the Validator and proxy containers
	CELOVALIDATORGROUPADDRESS		The account address for the Validator Group
	CELOVALIDATORADDRESS		The account address for the Validator
	CELOVALIDATORGROUPSIGNERADDRESS		The validator (group) signer address authorized by the Validator Group account.
	CELOVALIDATORGROUPSIGNERSIGNATURE		The proof-of-possession of the Validator Group signer key
	CELOVALIDATORSIGNERADDRESS		The validator signer address authorized by the Validator Account
	CELOVALIDATORSIGNERPUBLICKEY		The ECDSA public key associated with the Validator signer address
	CELOVALIDATORSIGNERSIGNATURE		The proof-of-possession of the Validator signer key
	CELOVALIDATORSIGNERBLSPUBLICKEY		The BLS public key for the Validator instance
	CELOVALIDATORSIGNERBLSSIGNATURE		A proof-of-possession of the BLS public key
	CELOVALIDATORGROUPVOTESIGNERADDRESS		The address of the Validator Group vote signer
	CELOVALIDATORGROUPVOTESIGNERPUBLICKEY		The ECDSA public key associated with the Validator Group vote signer address
	CELOVALIDATORGROUPVOTESIGNERSIGNATURE		The proof-of-possession of the Validator Group vote signer key
	CELOVALIDATORVOTESIGNERADDRESS		The address of the Validator vote signer
	CELOVALIDATORVOTESIGNERPUBLICKEY		The ECDSA public key associated with the Validator vote signer address
	CELOVALIDATORVOTESIGNERSIGNATURE		The proof-of-possession of the Validator vote signer key
	PROXYENODE		The enode address for the Validator proxy

PROXYINTERNALIP	(Optional) The internal IP address over which your Validator can communicate with your proxy
PROXYEXTERNALIP	The external IP address of the proxy. May be used by the Validator to communicate with the proxy if PROXYINTERNALIP is unspecified
DATABASEURL	The URL under which your database is accessible, currently supported are postgres://, mysql:// and sqlite://

Validator Node Setup

This section outlines the steps needed to configure your Proxy and Validator nodes so that they are ready to sign blocks once elected.

Environment Variables

First we are going to set up the main environment variables related to the mainnet network. Run these on both your Validator and Proxy machines:

```
bash
export CELOIMAGE=us.gcr.io/celo-org/geth:mainnet
```

Pull the Celo Docker image

In all the commands we are going to see the CELOIMAGE variable to refer to the Docker image to use. Now we can get the Docker image on your Validator and Proxy machines:

```
bash
docker pull $CELOIMAGE
```

Account Creation

```
:::info
```

Please complete this section if you are new to validating on Celo.

```
:::
```

Account and Signer keys

Running a Celo Validator node requires the management of several different keys, each with different privileges. Keys that need to be accessed frequently (e.g. for signing blocks) are at greater risk of being compromised, and thus have more limited permissions, while keys that need to be accessed infrequently (e.g. for locking CELO) are less onerous to store securely, and thus have more expansive permissions. Below is a summary of the various keys that are used in the Celo network, and a description of their permissions.

Name of the key	Purpose
-----	-----
Account key	This is the key with the highest level of permissions, and is thus the most sensitive. It can be used to lock and unlock CELO, and authorize vote and validator keys. Note that the account key also has all of the permissions of the other keys.
Validator signer key	This is the key that has permission to register and manage a Validator or Validator Group, and participate in BFT consensus.
Vote signer key	This key can be used to vote in Validator elections and on-chain governance.

Note that Account and all the signer keys must be unique and may not be reused.

Generating Validator and Validator Group Keys

First, you'll need to generate account keys for your Validator and Validator Group.

```
:::warning
```

These keys will control your locked CELO, and thus should be handled with care.

Store and back these keys up in a secure manner, as there will be no way to recover them if lost or stolen.

```
:::
```

```
bash
```

```
On your local machine
```

```
mkdir celo-accounts-node
```

```
cd celo-accounts-node
```

```
docker run -v $PWD:/root/.celo --rm -it $CELOIMAGE account new
```

```
docker run -v $PWD:/root/.celo --rm -it $CELOIMAGE account new
```

This will create a new keystore in the current directory with two new accounts.

Copy the addresses from the terminal and set the following environment variables:

```
bash
```

```
On your local machine
```

```
export CELOVALIDATORGROUPADDRESS=<YOUR-VALIDATOR-GROUP-ADDRESS>
```

```
export CELOVALIDATORADDRESS=<YOUR-VALIDATOR-ADDRESS>
```

Start your Accounts node

Next, we'll run a node on your local machine so that we can use these accounts to lock CELO and authorize the keys needed to run your validator. To do this, we need to run the following command to run the node.

```
bash
```

```
On your local machine
```

```
mkdir celo-accounts-node
```

```
cd celo-accounts-node
```

```
docker run --name celo-accounts -it --restart always --stop-timeout 300 -  
p 127.0.0.1:8545:8545 -v $PWD:/root/.celo $CELOIMAGE --verbosity 3 --  
syncmode full --http --http.addr 0.0.0.0 --http.api  
eth,net,web3,debug,admin,personal --datadir /root/.celo
```

:::danger

Security: The command line above includes the parameter `--http.addr 0.0.0.0` which makes the Celo Blockchain software listen for incoming RPC requests on all network adaptors. Exercise extreme caution in doing this when running outside Docker, as it means that any unlocked accounts and their funds may be accessed from other machines on the Internet. In the context of running a Docker container on your local machine, this together with the `docker -p 127.0.0.1:localport:containerport` flags allows you to make RPC calls from outside the container, i.e from your local host, but not from outside your machine. Read more about Docker Networking [here](#).

:::

Deploy a Validator Machine

To actually register as a validator, we'll need to generate a validating signer key. On your Validator machine (which should not be accessible from the public internet), follow very similar steps:

```
bash
```

```
On the validator machine
```

```
Note that you have to export $CELOIMAGE on this machine
```

```
export CELOIMAGE=us.gcr.io/celo-org/geth:mainnet
```

```
mkdir celo-validator-node
```

```
cd celo-validator-node
```

```
docker run -v $PWD:/root/.celo --rm -it $CELOIMAGE account new  
export CELOVALIDATORSIGNERADDRESS=<YOUR-VALIDATOR-SIGNER-ADDRESS>
```

Proof-of-Possession

:::info

Please complete this step if you are running a validator on Celo for the first time.

:::

In order to authorize our Validator signer, we need to create a proof that we have possession of the Validator signer private key. We do so by signing a message that consists of the Validator account address. To generate the proof-of-possession, run the following command:

```
bash
On the validator machine
Note that you have to export CELOVALIDATORADDRESS on this machine
export CELOVALIDATORADDRESS=<CELO-VALIDATOR-ADDRESS>
docker run -v $PWD:/root/.celo --rm -it $CELOIMAGE account proof-of-
possession $CELOVALIDATORSIGNERADDRESS $CELOVALIDATORADDRESS
```

Save the signer address, public key, and proof-of-possession signature to your local machine:

```
bash
On your local machine
export CELOVALIDATORSIGNERADDRESS=<YOUR-VALIDATOR-SIGNER-ADDRESS>
export CELOVALIDATORSIGNERSIGNATURE=<YOUR-VALIDATOR-SIGNER-SIGNATURE>
export CELOVALIDATORSIGNERPUBKEY=<YOUR-VALIDATOR-SIGNER-PUBLIC-KEY>
```

Validators on the Celo network use BLS aggregated signatures to create blocks in addition to the Validator signer (ECDSA) key. While an independent BLS key can be specified, the simplest thing to do is to derive the BLS key from the Validator signer key. When we register our Validator, we'll need to prove possession of the BLS key as well, which can be done by running the following command:

```
bash
On the validator machine
docker run -v $PWD:/root/.celo --rm -it $CELOIMAGE account proof-of-
possession $CELOVALIDATORSIGNERADDRESS $CELOVALIDATORADDRESS --bls
```

Save the resulting signature and public key to your local machine:

```
bash
On your local machine
export CELOVALIDATORSIGNERBLSSIGNATURE=<YOUR-VALIDATOR-SIGNER-SIGNATURE>
export CELOVALIDATORSIGNERBLSPUBKEY=<YOUR-VALIDATOR-SIGNER-BLS-PUBLIC-
KEY>
```

We'll get back to this machine later, but for now, let's give it a proxy.

Deploy a proxy

```
bash
On the proxy machine
```

```
mkdir celo-proxy-node
cd celo-proxy-node
export CELOVALIDATORSIGNERADDRESS=<YOUR-VALIDATOR-SIGNER-ADDRESS>
```

You can then run the proxy with the following command. Be sure to replace <YOUR-VALIDATOR-NAME> with the name you'd like to appear on Celostats. The validator name shown in Celostats will be the name configured in the proxy.

Additionally, you need to unlock the account configured in the etherbase option. It is recommended to create a new account and independent account only for this purpose. Be sure to write a new password to `./.password` for this account (different to the Validator Signer password)

```
bash
On the proxy machine
First, we create a new account for the proxy
docker run --name celo-proxy-password -it --rm -v $PWD:/root/.celo
$CELOIMAGE account new --password /root/.celo/.password
```

Notice the public address returned by this command, that can be exported and used for running the proxy node:

```
bash
On the proxy machine
export PROXYADDRESS=<PROXY-PUBLIC-ADDRESS>
docker run --name celo-proxy -it --restart unless-stopped --stop-timeout
300 -p 30303:30303 -p 30303:30303/udp -p 30503:30503 -p 30503:30503/udp -
v $PWD:/root/.celo $CELOIMAGE --verbosity 3 --syncmode full --proxy.proxy
--proxy.proxiedvalidatoraddress $CELOVALIDATORSIGNERADDRESS --
proxy.internalendpoint :30503 --etherbase $PROXYADDRESS --unlock
$PROXYADDRESS --password /root/.celo/.password --allow-insecure-unlock --
light.serve 0 --datadir /root/.celo --celostats=<YOUR-VALIDATOR-
NAME>@stats-server.celo.org
```

Hint: If you are running into trouble peering with the full nodes, one of the first things to check is whether your container's ports are properly configured (i.e. specifically, `-p 30303:30303 -p 30303:30303/udp` - which is set in the proxy node's command, but not the account node's command).

```
:::info
```

You can detach from the running container by pressing `ctrl+p ctrl+q`, or start it with `-d` instead of `-it` to start detached. Access the logs for a container in the background with the `docker logs` command.

```
:::
```

NOTES

- For the proxy to be able to send stats to Celostats, both the proxy and the validator should set the celostats flag
- If you are deploying multiple proxies for the same validator, the celostats flag should be added in only one of them

Get your Proxy's connection info

Once the Proxy is running, we will need to retrieve its enode and IP address so that the Validator will be able to connect to it.

bash

On the proxy machine, retrieve the proxy enode

```
docker exec celo-proxy geth --exec
```

```
"admin.nodeInfo['enode'].split('///')[1].split('@')[0]" attach | tr -d '"'
```

Now we need to set the Proxy enode and Proxy IP address in environment variables on the Validator machine.

If you don't have an internal IP address over which the Validator and Proxy can communicate, you can set the internal IP address to the external IP address.

If you don't know your proxy's external IP address, you can get it by running the following command:

bash

On the proxy machine

```
dig +short myip.opendns.com @resolver1.opendns.com
```

Then, export the variables on your Validator machine.

bash

On the Validator machine

```
export PROXYENODE=<YOUR-PROXY-ENODE>
```

```
export PROXYEXTERNALIP=<PROXY-MACHINE-EXTERNAL-IP-ADDRESS>
```

```
export PROXYINTERNALIP=<PROXY-MACHINE-INTERNAL-IP-ADDRESS>
```

You will also need to export PROXYEXTERNALIP on your local machine.

bash

On your local machine

```
export PROXYEXTERNALIP=<PROXY-MACHINE-EXTERNAL-IP-ADDRESS>
```

Connect the Validator to the Proxy

When your Validator starts up it will attempt to create a network connection with the proxy machine. You will need to make sure that your proxy machine has the appropriate firewall settings to allow the Validator to connect to it.

Specifically, on the proxy machine, port 30303 should allow TCP and UDP connections from all IP addresses. And port 30503 should allow TCP connections from the IP address of your Validator machine.

Test that your network is configured correctly by running the following commands:

```
bash
```

On your local machine, test that your proxy is accepting TCP connections over port 30303.

Note that it will also need to be accepting UDP connections over this port.

```
nc -vz $PROXYEXTERNALIP 30303
```

```
bash
```

On your Validator machine, test that your proxy is accepting TCP connections over port 30503.

```
nc -vz $PROXYINTERNALIP 30503
```

Once that is completed, go ahead and run the Validator. Be sure to write your Validator signer password to `./password` for the following command to work, or provide your password another way.

```
bash
```

On the Validator machine

```
cd celo-validator-node
```

```
docker run --name celo-validator -it --restart unless-stopped --stop-  
timeout 300 -p 30303:30303 -p 30303:30303/udp -v $PWD:/root/.celo  
$CELOIMAGE --verbosity 3 --syncmode full --mine --etherbase  
$CELOVALIDATORSIGNERADDRESS --nodiscover --proxy.proxied --  
proxy.proxyenodeurlpairs=enode://$PROXYENODE@$PROXYINTERNALIP:30503\;enod  
e://$PROXYENODE@$PROXYEXTERNALIP:30303 --  
unlock=$CELOVALIDATORSIGNERADDRESS --password /root/.celo/.password --  
light.serve 0 --celostats=<YOUR-VALIDATOR-NAME>@stats-server.celo.org
```

At this point your Validator and Proxy machines should be configured, and both should be syncing to the network. You should see Imported new chain segment in your node logs, about once every 5 seconds once the node is synced to the latest block which you can find on the Network Stats page.

```
:::info
```

You can run multiple proxies by deploying additional proxies per the instructions in the Deploy a proxy section. Then add all of the proxies' enodes as a comma separated list using the `--proxy.proxyenodeurlpairs` option. E.g. if there are two proxies, that option's usage would look like --

```
proxy.proxyenodeurlpairs=enode://$PROXYENODE1@$PROXYINTERNALIP1:30503\;en  
ode://$PROXYENODE1@$PROXYEXTERNALIP1:30303,enode://$PROXYENODE2@$PROXYINT  
ERNALIP2:30503\;enode://$PROXYENODE2@$PROXYEXTERNALIP2:30303
```

:::

Limiting block space for transactions paid in alternative ERC-20 gas currencies

As described in the protocol documentation, Celo allows users to pay for gas using tokens. There is a governable list of accepted tokens. However, the Celo blockchain client starting with version 1.8.1 implements a protective mechanism that allows validators to control the percentage of available block space used by transactions paid with an alternative fee currency (other than CELO) more precisely.

There are two new flags that control this behavior:

1. celo.feecurrency.limits with a comma-separated currencyaddresshash=limit mappings for currencies listed in the FeeCurrencyWhitelist contract, where limit represents the maximal fraction of the block gas limit as a float point number available for the given fee currency. The addresses are not expected to be checksummed.

For

example, 0x765DE816845861e75A25fCA122bb6898B8B1282a=0.1,0xD8763CBa276a3738E6DE85b4b3bF5FDed6D6cA73=0.05,0xE6961928066D3238134933ee9cDD510Ff157a6e=0.

2. celo.feecurrency.default - an overridable default value (initially set to 0.5) for currencies not listed in the limits map, meaning that if not specified otherwise, a transaction with a given fee currency can take up to 50% of the block space. CELO token doesn't have a limit.

Based on historical data, the following default configuration is proposed:

bash

```
--celo.feecurrency.limits.default=0.5
```

```
--
```

```
celo.feecurrency.limits="0x765DE816845861e75A25fCA122bb6898B8B1282a=0.9,0xD8763CBa276a3738E6DE85b4b3bF5FDed6D6cA73=0.5,0xe8537a3d056DA446677B9E9d6c5dB704EaAb4787=0.5"
```

It imposes the following limits:

- cUSD up to 90%
- cEUR up to 50%
- cREAL up to 50%
- any other token except CELO - 50%
- CELO doesn't have a limit

Registering as a Validator

Register the Accounts

You've now done all the infrastructure setup to get a validator and proxy running. To run a validator on Mainnet, you must lock CELO to participate in block production. Once you have CELO in you validator and validation group accounts, you can view their balances:

```
bash
On your local machine
celocli account:balance $CELOVALIDATORGROUPADDRESS
celocli account:balance $CELOVALIDATORADDRESS
```

You can also look at an account's current balance and transaction history on Celo Explorer. Enter the address into the search bar.

Once these accounts have a balance, unlock them so that we can sign transactions. Then, we will register the accounts with the Celo core smart contracts:

```
bash
On your local machine
celocli account:unlock $CELOVALIDATORGROUPADDRESS
celocli account:unlock $CELOVALIDATORADDRESS
celocli account:register --from $CELOVALIDATORGROUPADDRESS --name <NAME
YOUR VALIDATOR GROUP>
celocli account:register --from $CELOVALIDATORADDRESS --name <NAME YOUR
VALIDATOR>
```

Check that your accounts were registered successfully with the following commands:

```
bash
On your local machine
celocli account:show $CELOVALIDATORGROUPADDRESS
celocli account:show $CELOVALIDATORADDRESS
```

Lock up CELO

Lock up CELO for both accounts in order to secure the right to register a Validator and Validator Group. The current requirement is 10,000 CELO to register a validator, and 10,000 CELO per member validator to register a Validator Group. For Validators, these locked-up CELO remain locked for approximately 60 days following deregistration. For groups, these locked-up CELO remains locked for approximately 60 days following the removal of the Nth validator from the group.

```
bash
On your local machine
celocli lockedgold:lock --from $CELOVALIDATORGROUPADDRESS --value
10000000000000000000000000
celocli lockedgold:lock --from $CELOVALIDATORADDRESS --value
10000000000000000000000000
```


This amount (10,000 CELO) represents the minimum amount needed to be locked in order to register a Validator and Validator group. Note that you will want to be sure to leave enough CELO unlocked to be able to continue to pay transaction fees for future transactions (such as those issued by running some CLI commands).
Check that your CELO was successfully locked with the following commands:

```
bash
On your local machine
celocli lockedgold:show $CELOVALIDATORGROUPADDRESS
celocli lockedgold:show $CELOVALIDATORADDRESS
```

Run for election

In order to be elected as a Validator, you will first need to register your group and Validator. Note that when registering a Validator Group, you need to specify a commission, which is the fraction of epoch rewards paid to the group by its members.
We don't want to use our account key for validating, so first let's authorize the validator signing key:

```
bash
On your local machine
celocli account:authorize --from $CELOVALIDATORADDRESS --role validator -
--signature 0x$CELOVALIDATORSIGNERSIGNATURE --signer
0x$CELOVALIDATORSIGNERADDRESS
```

Confirm by checking the authorized Validator signer for your Validator:

```
bash
On your local machine
celocli account:show $CELOVALIDATORADDRESS
```

Then, register your Validator Group by running the following command. Note that because we did not authorize a Validator signer for our Validator Group account, we register the Validator Group with the account key.

```
bash
On your local machine
celocli validatorgroup:register --from $CELOVALIDATORGROUPADDRESS --
commission 0.1
```

You can view information about your Validator Group by running the following command:

```
bash
On your local machine
celocli validatorgroup:show $CELOVALIDATORGROUPADDRESS
```

Next, register your Validator by running the following command. Note that because we have authorized a Validator signer, this step could also be performed on the Validator machine. Running it on the local machine allows us to avoid needing to install the celocli on the Validator machine.

```
bash
```

On your local machine

```
celocli validator:register --from $CELOVALIDATORADDRESS --ecdsaKey
$CELOVALIDATORSIGNERPUBLICKEY --blsKey $CELOVALIDATORSIGNERBLSPUBLICKEY --
--blsSignature $CELOVALIDATORSIGNERBLSSIGNATURE
```

Affiliate your Validator with your Validator Group. Note that you will not be a member of this group until the Validator Group accepts you. This command could also be run from the Validator signer, if running on the validator machine.

```
bash
```

On your local machine

```
celocli validator:affiliate $CELOVALIDATORGROUPADDRESS --from
$CELOVALIDATORADDRESS
```

Accept the affiliation:

bash

On your local machine

```
celocli validatorgroup:member --accept $CELOVALIDATORADDRESS --from
$CELOVALIDATORGROUPADDRESS
```

Next, double check that your Validator is now a member of your Validator Group:

```
bash
```

On your local machine

```
celocli validator:show $CELOVALIDATORADDRESS
```

```
celocli validatorgroup:show $CELOVALIDATORGROUPADDRESS
```

Use both accounts to vote for your Validator Group. Note that because we have not authorized a vote signer for either account, these transactions must be sent from the account keys.

```
bash
```

On your local machine

```
celocli election:vote --from $CELOVALIDATORADDRESS --for
$CELOVALIDATORGROUPADDRESS --value 1000000000000000000000000
```

```
celocli election:vote --from $CELOVALIDATORGROUPADDRESS --for
$CELOVALIDATORGROUPADDRESS --value 1000000000000000000000000
```

Double check that your votes were cast successfully:

```
bash
```

On your local machine

```
celocli election:show $CELOVALIDATORGROUPADDRESS --group
```

```
celocli election:show $CELOVALIDATORGROUPADDRESS --voter
```

```
celocli election:show $CELOVALIDATORADDRESS --voter
```

Users in the Celo protocol receive epoch rewards for voting in Validator Elections only after submitting a special transaction to enable them. This must be done every time new votes are cast, and can only be made after the most recent epoch has ended. For convenience, we can use the following command, which will wait until the epoch has ended before sending a transaction:

```
bash
```

On your local machine

Note that this may take some time, as the epoch needs to end before votes can be activated

```
celocli election:activate --from $CELOVALIDATORADDRESS --wait && celocli
```

```
election:activate --from $CELOVALIDATORGROUPADDRESS --wait
```

Check that your votes were activated by re-running the following commands:

```
bash
```

On your local machine

```
celocli election:show $CELOVALIDATORGROUPADDRESS --voter
```

```
celocli election:show $CELOVALIDATORADDRESS --voter
```

If your Validator Group elects validators, you will receive epoch rewards in the form of additional Locked CELO voting for your Validator Group from your account addresses. You can see these rewards accumulate with the commands in the previous set, as well as:

```
bash
```

On your local machine

```
celocli lockedgold:show $CELOVALIDATORGROUPADDRESS
```

```
celocli lockedgold:show $CELOVALIDATORADDRESS
```

You're all set! Elections are finalized at the end of each epoch, roughly once a day in the Mainnet network. After that hour, if you get elected, your node will start participating BFT consensus and validating blocks. After the first epoch in which your Validator participates in BFT, you should receive your first set of epoch rewards. You can inspect the current state of the validator elections by running:

```
bash
```

On your local machine

```
celocli election:list
```

You can check the status of your validator, including whether it is elected and signing blocks, at stats.celo.org or by running:

```
bash
celocli validator:status --validator $CELOVALIDATORADDRESS
```

You can see additional information about your validator, including uptime score, by running:

```
bash
On your local machine
celocli validator:show $CELOVALIDATORADDRESS
```

Deployment Tips

Running the Docker containers in the background

There are different options for executing Docker containers in the background. The most typical one is to use in your docker run commands the `-d` option. Also for long running processes, especially when you run in on a remote machine, you can use a tool like `screen`. It allows you to connect and disconnect from running processes providing an easy way to manage long running processes.

It's out of the scope of this documentation to go through the `screen` options, but you can use the following command format with your docker commands:

```
bash
screen -S <SESSION NAME> -d -m <YOUR COMMAND>
```

For example:

```
bash
screen -S celo-validator -d -m docker run --name celo-validator -it --
restart unless-stopped --stop-timeout 300 -p 127.0.0.1:8545:8545 .....
```

You can list your existing screen sessions:

```
bash
screen -ls
```

And re-attach to any of the existing sessions:

```
bash
screen -r -S celo-validator
```

Stopping containers

You can stop the Docker containers at any time without problem. If you stop your containers that means those containers stop providing service. The data directory of the Validator and the proxy are Docker volumes mounted in the containers from the celo--dir you created at the very beginning. So if you don't remove that folder, you can stop or restart the containers without losing any data.

It is recommended to use the Docker stop timeout parameter -t when stopping the containers. This allows time, in this case 60 seconds, for the Celo nodes to flush recent chain data it keeps in memory into the data directories. Omitting this may cause your blockchain data to corrupt, requiring the node to start syncing from scratch.

You can stop the celo-validator and celo-proxy containers running:

```
bash
docker stop celo-validator celo-proxy -t 60
```

And you can remove the containers (not the data directory) by running:

```
bash
docker rm -f celo-validator celo-proxy
```

coinbase-wallet.md:

title: Coinbase Wallet Programmatic Setup on Celo
description: How dApp developers can use Coinbase Wallet to interact with the Celo network.

Adding a Celo Network to Coinbase Wallet

To add a Celo Network to your dApp, you can use Coinbase's RPC API's walletaddEthereumChain method. \ (See documentation\).

Here is a JavaScript snippet you can use:

```
jsx
await window.ethereum.request({
  method: 'walletaddEthereumChain',
  params: [<INSERTNETWORKPARAMSHERE>],
});
```

Where it says INSERTNETWORKPARAMSHERE, please replace with any of the following constants, depending on which network you'd like to connect to.

Mainnet

```
jsx
const CELOPARAMS = {
  chainId: "0xa4ec",
  chainName: "Celo",
  nativeCurrency: { name: "Celo", symbol: "CELO", decimals: 18 },
  rpcUrls: ["https://forno.celo.org"],
  blockExplorerUrls: ["https://explorer.celo.org/"],
  iconUrls: ["future"],
};
```

Alfajores

```
jsx
const ALFAJORESPARAMS = {
  chainId: "0xaef3",
  chainName: "Alfajores Testnet",
  nativeCurrency: { name: "Alfajores Celo", symbol: "A-CELO", decimals:
18 },
  rpcUrls: ["https://alfajores-forno.celo-testnet.org"],
  blockExplorerUrls: ["https://alfajores-blockscout.celo-testnet.org/"],
  iconUrls: ["future"],
};
```

Adding Tokens \ (e.g. cUSD, cEUR\)

To watch an asset on a Celo network \ (e.g. cUSD, cEUR\) in your dApp, you can use MetaMask's RPC API's `walletwatchAsset` method. \ (See [documentation](#)).

Here is a JavaScript snippet you can use:

```
jsx
await window.ethereum.request({
  method: "walletwatchAsset",
  params: {
    type: "ERC20",
    options: {
      address: "<INSERTADDRESSHERE>",
      symbol: "<INSERTSYMBOLHERE>",
      decimals: 18,
    },
    iconUrls: ["future"],
  },
});
```

- Where it says `INSERTADDRESSHERE`, please replace with any of the following constants, depending on which network and which asset you'd like to connect to.

- Where it says INSERTSYMBOLHERE, please replace with the correct symbol for the asset you'd like to watch. For Celo Dollars, it's cUSD and for Celo Euros, it's cEUR.

:::tip

View available token addresses for Celo assets to add to Coinbase Wallet [here](#).

index.md:

title: Celo Wallets

description: Overview of digital wallets available to send, spend, and earn Celo assets.

Celo Wallets

On this page you will find an overview of digital wallets available to send, spend, and earn Celo assets.

Choosing a Wallet

Wallets are tools that create accounts, manage keys, and help users transact on the Celo network.

:::warning

It's important to be careful when choosing a wallet because they manage your secret account keys. You should only use reputable wallets that are well-maintained by organizations/people that you trust. We added links to the source code so you can see when it's been last updated.

:::

The Celo Native Wallets section shows some popular wallets that were built specifically for the Celo network. They often include features that more general wallets do not, like paying for fees with ERC20 tokens, like cUSD. The Celo Compatible Wallets section has wallets that can work with Celo but were built for other networks (like Ethereum) or through company partnerships (like Pesabase).

Celo Native Wallets

Valora

Valora is a mobile wallet focused on making global peer-to-peer payments simple and accessible to anyone. It supports the Celo Identity Protocol which allows users to verify their phone number and send payments to their contacts.

- valoraapp.com

- Platforms: iOS, Android
- Maintainers: Valora
- Ledger support: No
- Source Code

MiniPay

MiniPay is a non-custodial 2MB wallet that allows users to send and receive stablecoins with meager transaction fees—less than 1 cent. It was first launched in Africa to assist people in sending and receiving stablecoins using mobile numbers.

- Countries: Ghana, Nigeria, Kenya, South Africa, Uganda
- Platforms: Android
- Maintainers: Opera
- Ledger support: No

Othello Wallet

Othello Wallet (formally know as CeloWallet) is a lightweight, mobile-friendly wallet for both web and desktop. It supports core Celo functionality like payments, exchanges, staking, and governance.

- Platforms: Web, MacOS, Linux, Windows
- Maintainers: J M Rossy
- Ledger support: Yes
- Source Code

Celo Terminal

Celo Terminal

Celo Terminal is a wallet and DApp platform. It aims to be a hub for installing and running Celo DApps locally on your desktop.

- Platforms: MacOS, Linux, Windows
- Maintainers: WOTrust
- Ledger support: Yes
- Source Code

CeloExtensionWallet

Celo Extension Wallet is a fork of Metamask for the Celo Network. It's a browser extension for Chrome.

- Platforms: Chrome Browser
- Maintainers: DSRV Labs
- Ledger support: Yes
- Source Code

Enkrypt

Enkrypt

Enkrypt is a self-custodial, open-sourced, client-side and multichain browser wallet with CELO natively integrated.

- Platforms: Chrome, Brave, Firefox, Safari, Opera
- Maintainers: MyEtherWallet
- Source Code

Omni

Omni (formally known as Steakwallet) is a non-custodial, multi chain staking wallet. It aims to be the hub for mobile DeFi, supporting Celo from the very start.

- Platforms: Android, iOS
- Maintainers: Omni
- Source Code

Celo Compatible Wallets

Below you can find more Celo-compatible wallets. They don't have the functionality to use different tokens for gas fees.

Metamask

You can learn more about connecting Metamask to the Celo network [here](#).

- Platforms: Browser, iOS, Android

Wallet Connect

Strictly speaking, Wallet Connect is not a wallet; it is an open protocol for connecting wallets to Dapps. Celo wallets are implementing Wallet Connect version 1, so dapp developers should use V1 as well.

- Platforms: Browser, iOS, Android
- Source Code

Wigwam Wallet

Dove Wallet

- dovewallet.com
- Platforms: Web

Wigwam Wallet

- Platforms: Browser, Web
- Source Code

Pesabase

- Platforms: iOS, Android

D'CENT

- Hardware wallet
- Platforms: Browser, iOS, Android
- Source Code

Bitfy

- Countries: Brazil
- Platforms: iOS, Android
- Maintainer: BWS

Cobru

- Countries: Colombia

Kotani Pay

- Countries: Kenya
- Source Code

Coinprofile

- Countries: Nigeria

Bitmama

- <https://bitmama.io/>
- Countries: Nigeria, Ghana
- Source Code

BloomX

- Exchange
- Countries: Philippines

Frontier

- Source Code

El Dorado

- Countries: Argentina, Brazilm, Colombia, Panama, Peru, Venezuela
- Platforms: Android, iOS

setup.md:

title: Set up a Ledger Wallet with Celo

description: How to set up a Ledger Nano S or X hardware wallet with Celo.

Set up a Ledger Wallet with Celo

How to set up a Ledger Nano S or X hardware wallet.

Hardware Security Module

A hardware wallet or Hardware Security Module (HSM) holds a single random seed (expressed as a mnemonic) which can be used to generate any number of public-private keypairs, that is, any number of accounts ("wallets"), each with an associated address.

:::info

The steps below require technical knowledge. You should be comfortable with the Command Line Interface (CLI) and understand the basics of how cryptographic network accounts work.

:::

Requirements

Make sure to have the following before you begin:

- Initialized your Ledger Nano X or S
- The latest firmware is installed
- Ledger Live is ready to be used.

Installation Instructions

Install the Celo Application

Start by installing the Celo application and setting a PIN on your Ledger device by following steps 1 and 2 on this page.

:::danger

Make sure to securely back up both the PIN and the recovery phrase (also known as a backup key or mnemonic). If you lose them, or they are stolen, you lose access to your Celo assets with no recovery possible. The recovery phrase will be shown only once.

:::

Open the Ledger Live App on your computer and follow the instructions on the screen.

Search for "Celo" in the app store.

Click Install for the Celo app, this will install the Celo App Version 1.0.3 on your device.

:::info

If you've previously installed the Celo app on your device, you'll see an Upgrade option instead of Install.

:::

The installation is completed once you see the green tick and Installed label.

You should now see on your device's screen Celo app. You may need to toggle left or right using the buttons on the device to find the app.

Quit the Ledger Live app on your compute but keep the Ledger wallet connected to your computer.

Setting up the Celo app

On your Ledger Nano device enter the PIN if prompted and press both buttons at the same time to open into the Celo app.

Press both buttons on the device at the same time to continue.

The Celo app is now ready for use and you should see Application is ready on the screen.

Connect your Leger to Celo applications

- Celo Terminal App
- Celo Web Wallet
- Celo CLI

to-celo-cli.md:

title: Connect Ledger to Celo CLI

description: How to connect a ledger wallet to the Celo CLI.

Connect Ledger to Celo CLI

How to connect a ledger wallet to the Celo CLI.

Install the Celo CLI

Open the terminal application on your computer and install the Celo CLI:

```
bash
npm install -g @celo/celocli
```

If you have previously installed the CLI, ensure that you are using version 0.0.47 or later:

```
bash
celocli --version
```

And if not, upgrade by running the same command as above.

You will now need to point the Celo CLI to a node that is synchronized with one of Celo's networks.

Configure for Celo Mainnet

Configure the Celo CLI so that it uses a cLabs node on the Alfajores network.

```
bash
celocli config:set --node https://forno.celo.org/
```

:::danger

Connecting celocli to an untrusted node may allow that node to influence the transactions sent by the Celo CLI to the Ledger for signing. When in doubt, always use a node that you trust or are running yourself.

:::

Configure for Celo Alfajores Testnet

Configure the Celo CLI so that it uses a cLabs node on the Alfajores network.

```
bash
celocli config:set --node https://alfajores-forno.celo-testnet.org/
```

:::danger

Connecting celocli to an untrusted node may allow that node to influence the transactions sent by the Celo CLI to the Ledger for signing. When in doubt, always use a node that you trust or are running yourself.

:::

Check that the node is synchronized to Celo CLI:

```
bash
celocli node:synced
```

The output should display true. If it displays false you may need to wait a bit and try again.

Confirm Addresses on Celo CLI

The Ledger's current seed phrase determines the device's accounts. In the terminal on your computer, you can view the first account's address with the following command:

```
bash
celocli account:list --useLedger --ledgerAddresses 1
```

:::info

If you wish to generate more than one address from your seed phrase, you can display the first N (e.g. 10) addresses use the `--ledgerAddresses` flag.

```
bash
celocli account:list --useLedger --ledgerAddresses N
```

To display addresses at specific indexes M and N (e. 2 and 654) use the `--ledgerCustomAddresses "[M, N]"` flag

```
bash
celocli account:list --useLedger --ledgerCustomAddresses "[M, N]"
```

:::

:::info

Advanced: Celo uses a BIP-32 derivation path of `m/44'/52752'/0'/0'/index`, where `index >= 0`.

:::

Performing a Testnet transaction on Celo CLI

Before using your address on the Celo Mainnet, you may want to test it on the Celo Alfajores Testnet with the following instructions.

Visit the Alfajores Faucet and send yourself some testnet CELO at the following URL:

<https://faucet.celo.org>

Check that you received the funds with the following command:

```
bash
celocli account:balance <your-address> --node https://alfajores-
forno.celo-testnet.org/
```

Next, you'll need to enable "Contract Data" in the ledger app. Open the Celo App on your ledger device and go to Settings, then enable "Contract Data" to "Allowed". This setting is required because the celocli uses the ERC20 "pre-wrapped" version of CELO and so sending transactions requires sending data to a smart contract.

Perform a test transaction by running the following command:

```
bash
celocli transfer:gold --from=<your-address> --
to=0x000000000000000000000000000000000000000000000000000000001 --value=10000 --useLedger -
-node https://alfajores-forno.celo-testnet.org/
```

You'll need to then approve the transaction on the Ledger device. Toggle right on the device until you see Approve on screen. Press both buttons at the same time to confirm.

Finally, you can see if your transaction was mined on the network by copying the transaction hash (txHash) outputted by the command, and searching for it on the Alfajores Block Explorer.

Using celocli

You can use celocli to securely sign transactions or proof-of-posessions with your Ledger.

To use celocli with your Ledger, ensure the device is connected to your computer, unlocked, and the Celo app is open and displaying Application is ready.

Then, simply append the --useLedger flag to any celocli commands with which you'd like to use a Ledger. You may also append the --ledgerConfirmAddress flag, which will require that you manually verify on the Ledger the address from which the transaction is being sent.

View Account Balance

In order to view your account Balance on your Ledger with celocli, you need to run the following command:

```
sh
If you haven't set the node config to mainnet, do it first
celocli config:set --node=https://forno.celo.org
celocli account:balance <your-address>
```

This will display the specific account balance for your address on Celo Mainnet.

Receive Crypto Assets

In order to receive Celo on your address, whether it's CELO or cUSD or any Mento stablecoin in the future, you must share your specific address with the sender.

Once a sender has confirmed they sent you the assets to your Ledger address, ask for the transaction ID which you can lookup on the Explorer.

Send Crypto Assets

In order to send CELO or cUSD from your Ledger, you just need a recipient address to send to. Once you have that and the amount you would like to send (in our example, 10 CELO), we will go over how to send CELO using celocli.

```
sh
celocli transfer:gold --from=<your-address> --to=<recipient-address> --
value=10 --useLedger
```

You'll need to then approve the transaction on the Ledger device. Toggle right on the device until you see Approve on screen. Press both buttons at the same time to confirm.

You'll then get a transaction hash when it's confirmed that the transaction was mined by the network, and you can check the status of the transaction on the explorer [here](#).

Troubleshooting

If you have issues connecting to the Ledger, try the following:

- Check that the Ledger device is connected, powered on, and that you've unlocked it using the PIN.
- Check that no other applications are using the device. Close Ledger Live. Stop any local Celo Blockchain node, or ensure it is run with the `-usb` option.
- Try unplugging and replugging the device. Some devices appear to trigger a warning on Macs saying: "USB Devices Disabled. Unplug the device using too much power to re-enable USB devices" which is usually resolved by reconnecting.

- Ensure that you are using the original cable supplied with your Ledger.
- Ensure that your Ledger has the latest firmware. For Ledger Nano S, a firmware version of 1.6 or later is required.
- Ensure that you are running the latest version of the Celo CLI.

There have been reports of a possible issue that appears to affect developer store apps on the Ledger Nano X including the Celo Ledger App. This is believed to be fixed in version 1.0.3. In earlier versions, a user clicking through the Pending Ledger review notice too rapidly can cause the device to freeze. If this occurs, wait until the device's battery is depleted, then charge and power up again. Then use Ledger Live Manager to update the installed version of the Celo Ledger App.

to-celo-terminal.md:

title: Connect Ledger to Celo Terminal

description: How to connect a ledger wallet to Celo Terminal.

Connect Ledger to Celo Terminal

How to connect a ledger wallet to Celo Terminal.

Download Celo Terminal

Navigate to <https://celoterminal.com/> to download application for your OS type.

!https://celoterminal.com

Add Account

Select "Add account"

!Add account

Add Ledger Account

Connect your ledger device and choose "Add Ledger Account"

!Connect ledger device

Choose your Account

A new window will appear listing available accounts, the default is selected (index 0). Choose your account, default selection typically works for standard accounts.

!Account selection

After selecting "Add" you will need to confirm the address with your ledger device.

Congrats, you have successfully attached your Ledger account to Celo Terminal.

```
# to-celo-web.md:
```

```
---
```

```
title: Connect Ledger to Celo Web Wallet
```

```
description: How to connect a ledger wallet to the Celo Web Wallet.
```

```
---
```

Connect Ledger to Celo Web Wallet

How to connect a ledger wallet to the Celo Web Wallet.

```
---
```

Download Celo Web Wallet

Navigate to <https://celowallet.app/setup>. Choose "Use Existing Account".

```
!https://Celowallet.app/setup
```

Use Ledger

Select "Use Ledger"

```
!https://celowallet.app/setup/existing
```

Import Account

The first account starts at index 0 (default account address). After Address index has been selected, choose "Import Account"

```
!https://celowallet.app/setup/ledger
```

Ledger Plugin

Plugin your ledger device and select connect.

```
!https://celowallet.app/setup
```

Congrats, you have successfully attached your Ledger account to Celo Wallet.

```
# import.md:
```

```
---
```

```
title: Import Valora Wallet to MetaMask
```

description: How to import your Celo account to MetaMask from a Valora wallet.

Import Valora to MetaMask

How to import your Celo account to MetaMask from a Valora wallet.

Getting Started

Importing a Celo Account to MetaMask allows you to extend Celo to any application that integrates with MetaMask. This guide helps you import your Celo account using a Valora Wallet and so you to access your Celo account using MetaMask.

- Prerequisites required to start using MetaMask and Celo
- Create a simple project directory to temporarily store project information
- Access your Valora account and private key with the celocli
- Import your private key to MetaMask to access your Celo account

Prerequisites

Before getting started, it's important to have downloaded MetaMask, Valora, and have completed some basic configuration on each account. Follow the links below for additional guides on each topic.

- Download MetaMask and create an account.
- Download Valora and create an account.
- Configure MetaMask to work with Celo
- Install the celocli

Set up your project

Create a new project directory.

```
mkdir valora-metamask
```

Change into your project directory.

```
cd valora-metamask
```

Create a text file to temporarily store your Valora Recovery Phrase.

```
touch valora-recovery-phrase.txt
```

Open the text file to store your Recovery Phrase.

```
open valora-recovery-phrase.txt
```

Valora Private Keys

- Open the Valora App, navigate to Menu > Recovery Phrase, and Enter PIN to reveal your Recovery Phrase.

:::warning

The Recovery Phrase is a series of 24 unique words specific to your Valora wallet. Do not lose these words or share them with anyone at any time.

:::

Populate Text File

Populate your text file with the Recovery Phrase shown in your Valora wallet.

one two three four five six seven...

Access Private Key

Open your terminal and type the following command to read your account information.

```
celocli account:new --mnemonicPath valora-recovery-phrase.txt
```

This command will display your Valora wallet mnemonic, accountAddress, privateKey, publicKey, and address.

```
mnemonic: one two three four five six seven...
accountAddress: 0x...
privateKey: [COPY THIS PRIVATE KEY]
publicKey: ...
address: 0x...
```

- Copy the privateKey from your terminal window.

:::note

This celocli command also shows your **accountAddress**. You won't be using this in this guide, but it is important to verify that

this is the correct address for your Valora wallet. Before moving on, confirm that the digits of the **Account Address** from your Valora Wallet match the **accountAddress** displayed in your terminal.

:::

:::warning

Anyone that has access to your private key will be able to access and control the funds in your wallet. Only store your private key in a safe location and do not share it with anyone.

:::

Import Private Key

- Open MetaMask and select Celo (Mainnet) as your network.
- Select Settings > Import Account select type Private Key and paste the private key from your terminal window.
- Select Import to import your Celo Account from your Valora Wallet.

Delete Project Directory

MetaMask is now connected to your Valora wallet. The value of your Valora wallet should show in your MetaMask account and you can now use MetaMask to access your funds.

:::note

You may now delete your project directory along with the text file used to store your wallet address.

:::

setup.md:

title: MetaMask Programmatic Setup on Celo
description: How dApp developers can use MetaMask to interact with the Celo network.

```
import Tabs from '@theme/Tabs'
import TabItem from '@theme/TabItem'
```

Programmatic Setup

How dApp developers can use MetaMask to interact with the Celo network.

Adding a Celo Network to MetaMask

To add a Celo Network to your dApp, you can use MetaMask's RPC API's `walletaddEthereumChain` method. \ (See documentation\).

Here is a JavaScript snippet you can use:

```
jsx
await window.ethereum.request({
  method: 'walletaddEthereumChain',
  params: [<INSERTNETWORKPARAMSHERE>],
});
```

Where it says `INSERTNETWORKPARAMSHERE`, please replace with any of the following constants, depending on which network you'd like to connect to.

Mainnet

```
jsx
const CELOPARAMS = {
  chainId: "0xa4ec",
  chainName: "Celo",
  nativeCurrency: { name: "Celo", symbol: "CELO", decimals: 18 },
  rpcUrls: ["https://forno.celo.org"],
  blockExplorerUrls: ["https://explorer.celo.org/"],
  iconUrls: ["future"],
};
```

Alfajores

```
jsx
const ALFAJORESPARAMS = {
  chainId: "0xaef3",
  chainName: "Alfajores Testnet",
  nativeCurrency: { name: "Alfajores Celo", symbol: "A-CELO", decimals:
18 },
  rpcUrls: ["https://alfajores-forno.celo-testnet.org"],
  blockExplorerUrls: ["https://alfajores-blockscout.celo-testnet.org/"],
  iconUrls: ["future"],
};
```

Adding Tokens \ (e.g. cUSD, cEUR\)

To watch an asset on a Celo network \ (e.g. cUSD, cEUR\) in your dApp, you can use MetaMask's RPC API's `walletwatchAsset` method. \ (See documentation\).

Here is a JavaScript snippet you can use:

```
jsx
await window.ethereum.request({
  method: "walletwatchAsset",
```

```
params: {
  type: "ERC20",
  options: {
    address: "<INSERTADDRESSHERE>",
    symbol: "<INSERTSYMBOLHERE>",
    decimals: 18,
  },
  iconUrls: ["future"],
},
});
```

- Where it says INSERTADDRESSHERE, please replace with any of the following constants, depending on which network and which asset you'd like to connect to.
- Where it says INSERTSYMBOLHERE, please replace with the correct symbol for the asset you'd like to watch. For Celo Dollars, it's cUSD and for Celo Euros, it's cEUR.

:::tip

View available token addresses for Celo assets to add to MetaMask [here](#).

:::

:::warning

We strongly suggest that you disable your dApp's functionality when MetaMask is connected to a non-Celo network. MetaMask has an API for determining what network/chain you're connected to. See [here](#) for more documentation around that.

:::

use.md:

title: MetaMask and Celo
description: Overview of MetaMask and how you can get started with MetaMask on Celo.

MetaMask and Celo

Overview of MetaMask and how you can get started with MetaMask on Celo.

:::danger

Do not send ETH to your Celo address. Do not send CELO assets to your Ethereum address. Always make sure that you are connected to the correct network.

:::

MetaMask is a crypto wallet that can be used in a web browser and on mobile devices to interact with the Ethereum blockchain. Many dApps in the space integrate with MetaMask, and we're excited to bring its functionality to the Celo ecosystem.

Since Celo network's Donut Hardfork, activated on Mainnet on May 19th, 2021, the protocol now supports Ethereum-compatible transactions. This means that users may use MetaMask to interact with the Celo blockchain and dApp developers can more easily port Ethereum dApps to the Celo blockchain.

How to use MetaMask with Celo

For end users:

- Configure a MetaMask Desktop or Web Wallet to Work with Celo
- Setup a Ledger to Work with MetaMask to Work With Celo

For developers:

- Setup MetaMask to Work with Your dApp

Things to Keep in Mind

MetaMask does not natively support Celo compatibility and some features won't work perfectly. Here are some things to be aware of when using MetaMask with Celo.

Private Key Import

Celo and Ethereum use different derivation paths for generating seed phrases. Because MetaMask does not let you specify a derivation path to use:

- You can't import an existing Celo account into the MetaMask wallet using its seed phrase, as you'd get the Ethereum version of it. Instead, you have to import it using the associated private key.
- If you want to import the Celo account you made on MetaMask to a different Celo wallet (e.g. Valora) you'd have to import it using the private key itself, NOT the seed phrase that MetaMask gives you.
- See this guide if you would like to Import a Valora Account to MetaMask with a Private Key
- See these guides if you accidentally sent ETH to CELO addresses or CELO to ETH addresses.

Gas Fees Require CELO

While gas on Celo can usually be paid in many different currencies, when using MetaMask, gas fees will automatically be paid in CELO. This is because MetaMask will be using the Ethereum-compatible Celo transaction format, which doesn't include the feeCurrency field.

Incorrect Logo

In some cases, the MetaMask UI may display the Ethereum logo in places where it should display a CELO logo or no logo at all.

:::info

MetaMask is primarily used for interacting with the Ethereum blockchain and does not natively support Celo compatibility. Alternatively, you may choose a Celo native wallet [here](#).

:::

markdown-page.md:

title: Markdown page example

Markdown page example

You don't need React to write simple standalone pages.