---

title: How to develop and test a dApp on a local, multi-client testnet description: This guide will first walk you through how to instantiate and configure a multi-client local Ethereum testnet before using the testnet to deploy & test a dApp. author: "Tedi Mitiku" tags: ["clients", "nodes", "smart contracts", "composability", "consensus layer", "execution layer", "testing",] skill: intermediate lang: en published: 2023-04-11

Introduction {#introduction}

This guide walks you through the process of instantiating a configurable local Ethereum testnet, deploying a smart contract to it, and using the testnet to run tests against your dApp. This guide is designed for dApp developers who want to develop and test their dApps locally against different network configurations before deploying to a live testnet or the mainnet.

In this guide, you will:

- Instantiate a local Ethereum testnet with the `eth-network-package` using Kurtosis,
- Connect your Hardhat dApp development environment to the local testnet to compile, deploy, and test a dApp, and
- Configure the local testnet, including parameters like number of nodes and specific EL/CL client pairings, to enable development and testing workflows against various network configurations.

What is Kurtosis? {#what-is-kurtosis}

Kurtosis is a composable build system designed for configuring multi-container test environments. It specifically enables developers to create reproducible environments that require dynamic setup logic, such as blockchain testnets.

In this guide, the Kurtosis eth-network-package spins up a local Ethereum testnet with support for the `geth` Execution Layer (EL) client, as well as `teku`, `lighthouse`, and `lodestar` Consensus Layer (CL) clients. This package serves as a configurable and composable alternative to networks in frameworks like Hardhat Network, Ganache, and Anvil. Kurtosis offers developers greater control and flexibility over the testnets they use, which is a major reason why the Ethereum Foundation used Kurtosis to test the Merge and continues to use it for testing network upgrades.

Setting up Kurtosis {#setting-up-kurtosis}

Before you proceed, make sure you have:

- Installed and started the Docker engine on your local machine
- Installed the Kurtosis CLI (or upgraded it to the latest release, if you already have the CLI installed)
- Installed Node.js, yarn, and npx (for your dApp environment)

Instantiating a local Ethereum testnet {#instantiate-testnet}

To spin up a local Ethereum testnet, run:

```
python kurtosis --enclave local-eth-testnet run github.com/kurtosis-tech/eth-network-package
```

Note: This command names your network: "local-eth-testnet” using the `--enclave` flag.

Kurtosis will print the steps its taking under the hood as it works to interpret, validate, and then execute the instructions. At the end, you should see an output that resembles the following:

```python INFO[2023-04-04T18:09:44-04:00] ==========
INFO[2023-04-04T18:09:44-04:00] || Created enclave: local-eth-testnet || INFO[2023-04-04T18:09:44-04:00]
========== Name: local-eth-testnet UUID: 39372d756ae8
Status: RUNNING Creation Time: Tue, 04 Apr 2023 18:09:03 EDT

========== Files Artifacts ==========
UUID Name d4085a064230 cl-genesis-data 1c62cb792e4c el-genesis-data bd60489b73a7 genesis-generation-config-cl
b2e593fe5228 genesis-generation-config-el d552a54acf78 geth-prefunded-keys 5f7e661eb838 prysm-password
```

054e7338bb59 validator-keystore-0

========== User Services ========== UUID Name Ports Status e20f129ee0c5 cl-client-0-beacon http: 4000/tcp -> http://127.0.0.1:54261 RUNNING metrics: 5054/tcp -> http://127.0.0.1:54262 tcp-discovery: 9000/tcp -> 127.0.0.1:54263 udp-discovery: 9000/udp -> 127.0.0.1:60470 a8b6c926cdb4 cl-client-0-validator http: 5042/tcp -> 127.0.0.1:54267 RUNNING metrics: 5064/tcp -> http://127.0.0.1:54268 d7b802f623e8 el-client-0 engine-rpc: 8551/tcp -> 127.0.0.1:54253 RUNNING rpc: 8545/tcp -> 127.0.0.1:54251 tcp-discovery: 30303/tcp -> 127.0.0.1:54254 udp-discovery: 30303/udp -> 127.0.0.1:53834 ws: 8546/tcp -> 127.0.0.1:54252 514a829c0a84 prelaunch-data-generator-1680646157905431468 STOPPED 62bd62d0aa7a prelaunch-data-generator-1680646157915424301 STOPPED 05e9619e0e90 prelaunch-data-generator-1680646157922872635 STOPPED

```

Congratulations! You used Kurtosis to instantiate a local Ethereum testnet, with a CL (`lighthouse`) and EL client (`geth`), over Docker.

### Review {#review-instantiate-testnet}

In this section, you executed a command that directed Kurtosis to use the`eth-network-package` hosted remotely on GitHub to spin up a local Ethereum testnet within a Kurtosis Enclave. Inside your enclave, you will find both "file artifacts" and "user services".

The File Artifacts in your enclave include all the data generated and utilized to bootstrap the EL and CL clients. The data was created using the `prelaunch-data-generator` service built from this Docker image

User services display all the containerized services operating in your enclave. You will notice that a single node, featuring both an EL client and a CL client, has been created.

## Connect your dApp development environment to the local Ethereum testnet {#connect-your-dapp}

### Setup the dApp development environment {#set-up-dapp-env}

Now that you have a running local testnet, you can connect your dApp development environment to use your local testnet. The Hardhat framework will be used in this guide to deploy a blackjack dApp to your local testnet.

To set up your dApp development environment, clone the repository that contains our sample dApp and install its dependencies, run:

```
python git clone https://github.com/kurtosis-tech/awesome-kurtosis.git && cd awesome-kurtosis/smart-contract-example && yarn
```

The smart-contract-example folder used here contains the typical setup for a dApp developer using the Hardhat framework:

- `contracts/` contains a few simple smart contracts for a Blackjack dApp
- `scripts/` contains a script to deploy a token contract to your local Ethereum network
- `test/` contains a simple .js test for your token contract to confirm each player in our Blackjack dApp has 1000 minted for them
- `hardhat.config.ts` configures your Hardhat setup

### Configure Hardhat to use the local testnet {#configure-hardhat}

With your dApp development environment set up, you will now connect Hardhat to use the local Ethereum testnet generated using Kurtosis. To accomplish this, replace `&lt;$YOUR_PORT&gt;` in the `localnet` struct in your `hardhat.config.ts` config file with the port of the rpc uri output from any `el-client-&lt;num&gt;` service. In this sample case, the port would be `64248`. Your port will be different.
```

Example in `hardhat.config.ts`:

```js localnet: { url: 'http://127.0.0.1:<$YOUR_PORT>',// TODO: REPLACE $YOUR_PORT WITH THE PORT OF A NODE URI PRODUCED BY THE ETH NETWORK KURTOSIS PACKAGE

// These are private keys associated with prefunded test accounts created by the eth-network-package // https://github.com/kurtosis-tech/eth-network-package/blob/main/src/prelaunch_data_generator/genesis_constants/genesis_constants.star accounts: [ "ef5177cd0b6b21c87db5a0bf35d4084a8a57a9d6a064f86d51ac85f2b873a4e2", "48fcc39ae27a0e8bf0274021ae6ebd8fe4a0e12623d61464c498900b28feb567", "7988b3a148716ff800414935b305436493e1f25237a2a03e5eebc343735e2f31", "b3c409b6b0b3aa5e65ab2dc1930534608239a478106acf6f3d9178e9f9b00b35", "df9bb6de5d3dc59595bcaa676397d837ff49441d211878c024eabda2cd067c9f", "7da08f856b5956d40a72968f93396f6acff17193f013e8053f6fbb6c08c194d6", ], }, ```

Once you save your file, your Hardhat dApp development environment is now connected to your local Ethereum testnet! You can verify that your testnet is working by running:

```
python npx hardhat balances --network localnet
```

The output should look something like this:

```
python 0x878705ba3f8Bc32FCf7F4CAa1A35E72AF65CF766 has balance 1000000000000000000000000
0x4E9A3d9D1cd2A2b2371b8b3F489aE72259886f1A has balance 1000000000000000000000000
0xdF8466f277964Bb7a0FFD819403302C34DCD530A has balance 1000000000000000000000000
0x5c613e39Fc0Ad91AfDA24587e6f52192d75FBA50 has balance 1000000000000000000000000
0x375ae6107f8cC4cF34842B71C6F746a362Ad8EAc has balance 1000000000000000000000000
0x1F6298457C5d76270325B724Da5d1953923a6B88 has balance 1000000000000000000000000
```

This confirms that Hardhat is using your local testnet and detects the pre-funded accounts created by the `eth-network-package`.

Deploy and test your dApp locally {#deploy-and-test-dapp}

With the dApp development environment fully connected to the local Ethereum testnet, you can now run development and testing workflows against your dApp using the local testnet.

To compile and deploy the `ChipToken.sol` smart contract for local prototyping and development, run:

```
python npx hardhat compile npx hardhat run scripts/deploy.ts --network localnet
```

The output should look something like:

```
python ChipToken deployed to: 0xAb2A01BC351770D09611Ac80f1DE076D56E0487d
```

Now try running the `simple.js` test against your local dApp to confirm each player in our Blackjack dApp has 1000 minted for them:

The output should look something like this:

```
python npx hardhat test --network localnet
```

The output should look something like this:

```python ChipToken mint ✔ should mint 1000 chips for PLAYER ONE

1 passing (654ms) ```

## Review {#review-dapp-workflows}

At this point, you've now set up a dApp development environment, connected it to a local Ethereum network created by Kurtosis, and have compiled, deployed, and ran a simple test against your dApp.

Now let's explore how you can configure the underlying network for testing our dApps under varying network configurations.

# Configuring the local Ethereum testnet {#configure-testnet}

## Changing the client configurations and number of nodes {#configure-client-config-and-num-nodes}

Your local Ethereum testnet can be configured to use different EL and CL client pairs, as well as a varying number of nodes, depending on the scenario and specific network configuration you want to develop or test. This means that, once set up, you can spin up a customized local testnet and use it to run the same workflows (deployment, tests, etc.) under various network configurations to ensure everything works as expected. To learn more about the other parameters you can modify, visit this link.

Give it a try! You can pass various configuration options to the`eth-network-package` via a JSON file. This network params JSON file provides the specific configurations that Kurtosis will use to set up the local Ethereum network.

Take the default configuration file and edit it to spin up two nodes with different EL/CL pairs:

- Node 1 with `geth`/`lighthouse`
- Node 2 with `geth`/`lodestar`
- Node 3 with `geth`/`teku`

This configuration creates a heterogeneous network of Ethereum node implementations for testing your dApp. Your configuration file should now look like:

```
yaml { "participants": [{ "el_client_type": "geth", "el_client_image": "", "el_client_log_level": "",
"cl_client_type": "lighthouse", "cl_client_image": "", "cl_client_log_level": "", "beacon_extra_params": [],
"el_extra_params": [], "validator_extra_params": [], "builder_network_params": null, }, { "el_client_type":
"geth", "el_client_image": "", "el_client_log_level": "", "cl_client_type": "lodestar", "cl_client_image": "",
"cl_client_log_level": "", "beacon_extra_params": [], "el_extra_params": [], "validator_extra_params": [],
"builder_network_params": null, }, { "el_client_type": "geth", "el_client_image": "", "el_client_log_level": "",
"cl_client_type": "teku", "cl_client_image": "", "cl_client_log_level": "", "beacon_extra_params": [],
"el_extra_params": [], "validator_extra_params": [], "builder_network_params": null, },], "network_params": {
"preregistered_validator_keys_mnemonic": "giant issue aisle success illegal bike spike question tent bar rely
arctic volcano long crawl hungry vocal artwork sniff fantasy very lucky have athlete",
"num_validator_keys_per_node": 64, "network_id": "3151908", "deposit_contract_address":
"0x4242424242424242424242424242424242424242", "seconds_per_slot": 12, "genesis_delay": 120,
"capella_fork_epoch": 5, }, }
```

Each `participants` struct maps to a node in the network, so 3`participants` structs will tell Kurtosis to spin up 3 nodes in your network. Each `participants` struct will allow you to specify the EL and CL pair used for that specific node.

The `network_params` struct configures the network settings that are used to create the genesis files for each node as well as other settings like the seconds per slot of the network.

Save your edited params file in any directory you wish (in the example below, it is saved to the desktop) and then use it to run your Kurtosis package by running:

```
python kurtosis clean -a && kurtosis run --enclave local-eth-testnet github.com/kurtosis-tech/eth-network-
package "$(cat ~/eth-network-params.json)"
```

Note: the `kurtosis clean -a` command is used here to instruct Kurtosis to destroy the old testnet and its contents before starting a new one up.

Again, Kurtosis will work for a bit and print out the individual steps that are taking place. Eventually, the output should look something like:

```python Starlark code successfully run. No output was returned. INFO[2023-04-07T11:43:16-04:00]
========================================================== INFO[2023-04-07T11:43:16-04:00] || Created
enclave: local-eth-testnet || INFO[2023-04-07T11:43:16-04:00]
========================================================== Name: local-eth-testnet UUID: bef8c192008e
Status: RUNNING Creation Time: Fri, 07 Apr 2023 11:41:58 EDT

========================================== Files Artifacts ==========================================
UUID Name cc495a8e364a cl-genesis-data 7033fcdb5471 el-genesis-data a3aef43fc738 genesis-generation-config-cl
```

8e968005fc9d genesis-generation-config-el 3182cca9d3cd geth-prefunded-keys 8421166e234f prysm-password d9e6e8d44d99 validator-keystore-0 23f5ba517394 validator-keystore-1 4d28dea40b5c validator-keystore-2

========== User Services ========== UUID Name Ports Status 485e6fde55ae cl-client-0-beacon http: 4000/tcp -> http://127.0.0.1:65010 RUNNING metrics: 5054/tcp -> http://127.0.0.1:65011 tcp-discovery: 9000/tcp -> 127.0.0.1:65012 udp-discovery: 9000/udp -> 127.0.0.1:54455 73739bd158b2 cl-client-0-validator http: 5042/tcp -> 127.0.0.1:65016 RUNNING metrics: 5064/tcp -> http://127.0.0.1:65017 1b0a233cd011 cl-client-1-beacon http: 4000/tcp -> 127.0.0.1:65021 RUNNING metrics: 8008/tcp -> 127.0.0.1:65023 tcp-discovery: 9000/tcp -> 127.0.0.1:65024 udp-discovery: 9000/udp -> 127.0.0.1:56031 validator-metrics: 5064/tcp -> 127.0.0.1:65022 949b8220cd53 cl-client-1-validator http: 4000/tcp -> 127.0.0.1:65028 RUNNING metrics: 8008/tcp -> 127.0.0.1:65030 tcp-discovery: 9000/tcp -> 127.0.0.1:65031 udp-discovery: 9000/udp -> 127.0.0.1:60784 validator-metrics: 5064/tcp -> 127.0.0.1:65029 c34417bea5fa cl-client-2 http: 4000/tcp -> 127.0.0.1:65037 RUNNING metrics: 8008/tcp -> 127.0.0.1:65035 tcp-discovery: 9000/tcp -> 127.0.0.1:65036 udp-discovery: 9000/udp -> 127.0.0.1:63581 e19738e6329d el-client-0 engine-rpc: 8551/tcp -> 127.0.0.1:64986 RUNNING rpc: 8545/tcp -> 127.0.0.1:64988 tcp-discovery: 30303/tcp -> 127.0.0.1:64987 udp-discovery: 30303/udp -> 127.0.0.1:55706 ws: 8546/tcp -> 127.0.0.1:64989 e904687449d9 el-client-1 engine-rpc: 8551/tcp -> 127.0.0.1:64993 RUNNING rpc: 8545/tcp -> 127.0.0.1:64995 tcp-discovery: 30303/tcp -> 127.0.0.1:64994 udp-discovery: 30303/udp -> 127.0.0.1:58096 ws: 8546/tcp -> 127.0.0.1:64996 ad6f401126fa el-client-2 engine-rpc: 8551/tcp -> 127.0.0.1:65003 RUNNING rpc: 8545/tcp -> 127.0.0.1:65001 tcp-discovery: 30303/tcp -> 127.0.0.1:65000 udp-discovery: 30303/udp -> 127.0.0.1:57269 ws: 8546/tcp -> 127.0.0.1:65002 12d04a9dbb69 prelaunch-data-generator-1680882122181135513 STOPPED 5b45f9c0504b prelaunch-data-generator-1680882122192182847 STOPPED 3d4aaa75e218 prelaunch-data-generator-1680882122201668972 STOPPED ```

Congratulations! You've successfully configured your local testnet to have 3 nodes instead of 1. To run the same workflows you did before against your dApp (deploy & test), perform the same operations we did before by replacing the `<$YOUR_PORT>` in the `localnet` struct in your `hardhat.config.ts` config file with the port of the rpc uri output from any `el-client-<num>` service in your new, 3-node local testnet.

Conclusion {#conclusion}

And that's it! To recap this short guide, you:

- Created a local Ethereum testnet over Docker using Kurtosis
- Connected your local dApp development environment to the local Ethereum network
- Deployed a dApp and ran a simple test against it on the local Ethereum network
- Configured the underlying Ethereum network to have 3 nodes

We'd love to hear from you on what went well for you, what could be improved, or to answer any of your questions. Don't hesitate to reach out via GitHub or email us!

Other examples and guides {#other-examples-guides}

We encourage you to check out our quickstart (where you'll build a Postgres database and API on top) and our other examples in our awesome-kurtosis repository where you'll find some great examples, including packages for:

- Spinning up the same local Ethereum testnet, but with additional services connected such as a transaction spammer (to simulate transactions), a fork monitor, and a connected Grafana and Prometheus instance
- Performing a sub-networking test against the same local Ethereum network