Creating a Local Development Environment

Kurtosis has created an easy way to spin up a local NEAR testing environment using a Docker container . This Kurtosis NEAR Package contains the following components:

- Indexer for Explorer
- NEAR Explorer
- NEAR Wallet
- Local RPC Endpoint

Visit here to see a short demo of the package in action.

Prerequisites

- Docker
- Yarn
- NEAR-CLI
- Kurtosis CLI
-
    - Start Kurtosis engine after installation using:kurtosis engine start

Setup

Launch Kurtosis NEAR Package

Launch your Kurtosis NEAR Package in four easy steps!

1. LaunchDocker
2. Copy theKurtosis NEAR Package launch script
3. by running the following:

curl -o ~/launch-local-near-cluster.sh https://raw.githubusercontent.com/kurtosis-tech/near-package/master/launch-local-near-cluster.sh -L 1. Grant write permission to the script file you just downloaded:

chmod u+x ~/launch-local-near-cluster.sh 1. Launch the Kurtosis NEAR Package:

If you're running the NEAR-in-Kurtosis cluster on your local machine:

~/launch-local-near-cluster.sh If you're running your NEAR-in-Kurtosis cluster on a machine you intend to access remotely, replace1.2.3.4 with the IP address of the machine you're running the cluster on:

~/launch-local-near-cluster.sh '{"backend_ip_address":"1.2.3.4"}' Example response: Created directory '/Users/zerix/.neartosis' for storing all NEAR-in-Kurtosis output INFO[2022-12-06T12:59:04+05:30] Creating a new enclave for Starlark to run inside... INFO[2022-12-06T12:59:14+05:30] Enclave 'near' created successfully INFO[2022-12-06T12:59:14+05:30] Kurtosis CLI is running in a non interactive terminal. Everything will work but progress information and the progress bar will not be displayed.

> print "Starting the near-package with input struct(backend_ip_address = \"127.0.0.1\")" Starting the near-package with input struct(backend_ip_address = "127.0.0.1")
>
> print "Launching contract helper postgresql" Launching contract helper postgresql
>
> print "Adding contract helper Posgresql DB running on port '5432'" Adding contract helper Posgresql DB running on port '5432'
>
> add_service service_id="contract-helper-db" Service 'contract-helper-db' added with service GUID 'contract-helper-db-1670311755'
>
> exec service_id="contract-helper-db" command=["sleep", "10"] Command returned with exit code '0' with no output
>
> exec service_id="contract-helper-db" command=["psql", "-U", "near", "-c", "\l"] Command returned with exit code '0' and the following output: List of databases Name | Owner | Encoding | Collate | Ctype | Access privileges -----------+-------+----------+-------------+-------------+------------------- near | near | UTF8 | en_US.utf8 | en_US.utf8 | postgres | near | UTF8 | en_US.utf8 | en_US.utf8 | template0 | near | UTF8 | en_US.utf8 | en_US.utf8 | =c/near + | | | | | near=CTc/near template1 | near | UTF8 | en_US.utf8 | en_US.utf8 | =c/near + | | | | | near=CTc/near (4 rows)
>
> exec service_id="contract-helper-db" command=["psql", "-U", "near", "-c", "create database indexer with owner=near"] Command returned with exit code '0' and the following output: 'CREATE DATABASE'
>
> exec service_id="contract-helper-db" command=["psql", "-U", "near", "-c", "create database analytics with owner=near"] Command returned with exit code '0' and the following output: 'CREATE DATABASE'
>
> exec service_id="contract-helper-db" command=["psql", "-U", "near", "-c", "create database telemetry with owner=near"] Command returned with exit code '0' and the following output: 'CREATE DATABASE'

print "Contract helper postgresql db info struct(analytics_db = \"analytics\", db_user_password = \"near\", db_username = \"near\", indexer_db = \"indexer\", private_url = struct(ip_address = \"contract-helper-db\", path = \"\", port_number = 5432, protocol = \"postgres\"), telemetry_db = \"telemetry\")" Contract helper postgresql db info struct(analytics_db = "analytics", db_user_password = "near", db_username = "near", indexer_db = "indexer", private_url = struct(ip_address = "contract-helper-db", path = "", port_number = 5432, protocol = "postgres"), telemetry_db = "telemetry")

print "Launching contract helper dynamo db" Launching contract helper dynamo db

print "Adding contract helper DynamoDB running on default port '8000'" Adding contract helper DynamoDB running on default port '8000'

add_service service_id="contract-helper-dynamo-db" Service 'contract-helper-dynamo-db' added with service GUID 'contract-helper-dynamo-db-1670311770'

print "Contract helper dynamodb info struct(private_url = struct(ip_address = \"contract-helper-dynamo-db\", path = \"\", port_number = 8000, protocol = \"TCP\"))" Contract helper dynamodb info struct(private_url = struct(ip_address = "contract-helper-dynamo-db", path = "", port_number = 8000, protocol = "TCP"))

print "Launching indexer" Launching indexer

print "Adding indexer service..." Adding indexer service...

upload_files src="github.com/kurtosis-tech/near-package/static_files/near-configs/localnet" artifact_id="8f5279c5-d012-4543-88e7-e9829b6d6040" Files uploaded with artifact ID '8f5279c5-d012-4543-88e7-e9829b6d6040'

add_service service_id="indexer-node" Service 'indexer-node' added with service GUID 'indexer-node-1670311774'

exec service_id="indexer-node" command=["sleep", "10"] Command returned with exit code '0' with no output

exec service_id="indexer-node" command=["cat", "/root/.near/validator_key.json"] Command returned with exit code '0' and the following output: { "account_id": "test.near", "public_key": "ed25519:3Kuyi2DUXdoHgoaNEvCxa1m6G8xqc6Xs7WGajaqLhNmW", "secret_key": "ed25519:2ykcMLiM7vCmsSECcgfmUzihBtNdBv7v2CxNi94sNt4R8ar4xsrMMYvtsSNGQDfSRhNWXEnZvgx2wzS9ViBiS9jW" }

print "Indexer launched with struct(private_rpc_url = struct(ip_address = \"indexer-node\", path = \"\", port_number = 3030, protocol = \"http\"), public_rpc_url = struct(ip_address = \"127.0.0.1\", path = \"\", port_number = 8332, protocol = \"http\"), validator_key = {\"account_id\": \"test.near\", \"public_key\": \"ed25519:3Kuyi2DUXdoHgoaNEvCxa1m6G8xqc6Xs7WGajaqLhNmW\", \"secret_key\": \"ed25519:2ykcMLiM7vCmsSECcgfmUzihBtNdBv7v2CxNi94sNt4R8ar4xsrMMYvtsSNGQDfSRhNWXEnZvgx2wzS9ViBiS9jW\"})" Indexer launched with struct(private_rpc_url = struct(ip_address = "indexer-node", path = "", port_number = 3030, protocol = "http"), public_rpc_url = struct(ip_address = "127.0.0.1", path = "", port_number = 8332, protocol = "http"), validator_key = {"account_id": "test.near", "public_key": "ed25519:3Kuyi2DUXdoHgoaNEvCxa1m6G8xqc6Xs7WGajaqLhNmW", "secret_key": "ed25519:2ykcMLiM7vCmsSECcgfmUzihBtNdBv7v2CxNi94sNt4R8ar4xsrMMYvtsSNGQDfSRhNWXEnZvgx2wzS9ViBiS9jW"})

print "Launching contract helper" Launching contract helper

print "Adding contract helper service running on port '3000'" Adding contract helper service running on port '3000'

add_service service_id="contract-helper-service" Service 'contract-helper-service' added with service GUID 'contract-helper-service-1670311796'

print "Contract helper launchded with struct(private_url = struct(ip_address = \"contract-helper-service\", path = \"\", port_number = 3000, protocol = \"http\"), public_url = struct(ip_address = \"127.0.0.1\", path = \"\", port_number = 8330, protocol = \"http\"))" Contract helper launchded with struct(private_url = struct(ip_address = "contract-helper-service", path = "", port_number = 3000, protocol = "http"), public_url = struct(ip_address = "127.0.0.1", path = "", port_number = 8330, protocol = "http"))

print "Launching explorer backend" Launching explorer backend

print "Adding explorer backend service" Adding explorer backend service

add_service service_id="explorer-backend" Service 'explorer-backend' added with service GUID 'explorer-backend-1670311799'

print "Explorer backend launchded with struct(private_url = struct(ip_address = \"explorer-backend\", path = \"\", port_number = 8080, protocol = \"http\"), public_url = struct(ip_address = \"127.0.0.1\", path = \"\", port_number = 18080, protocol = \"http\"))" Explorer backend launchded with struct(private_url = struct(ip_address = "explorer-backend", path = "", port_number = 8080, protocol = "http"), public_url = struct(ip_address = "127.0.0.1", path = "", port_number = 18080, protocol = "http"))

print "Launching explorer frontend" Launching explorer frontend

print "Adding explorer frontend service running on port '3000'" Adding explorer frontend service running on port '3000'

add_service service_id="explorer-frontend" Service 'explorer-frontend' added with service GUID 'explorer-frontend-

1670311803'

print "Explorer frontend launchded with struct(public_url = struct(ip_address = \"127.0.0.1\", path = \"\", port_number = 8331, protocol = \"http\"))" Explorer frontend launchded with struct(public_url = struct(ip_address = "127.0.0.1", path = "", port_number = 8331, protocol = "http"))

print "Launching wallet" Launching wallet

print "Adding wallet service running on port '3004" Adding wallet service running on port '3004

print "Replacing variable 'NODE_URL' to 'http://127.0.0.1:8332' using regexp: '([,{])NODE_URL:[^,}*([,}])*'" *Replacing variable 'NODE_URL' to 'http://127.0.0.1:8332' using regexp: '([,{])NODE_URL:[^,}*([,}])'*

print "Replacing variable 'ACCOUNT_HELPER_URL' to 'http://127.0.0.1:8330' using regexp: '([,{])ACCOUNT_HELPER_URL:[^,}*([,}])*'" *Replacing variable 'ACCOUNT_HELPER_URL' to 'http://127.0.0.1:8330' using regexp: '([,{])ACCOUNT_HELPER_URL:[^,}*([,}])'*

print "Replacing variable 'EXPLORER_URL' to 'http://127.0.0.1:8331' using regexp: '([,{])EXPLORER_URL:[^,}*([,}])*'" *Replacing variable 'EXPLORER_URL' to 'http://127.0.0.1:8331' using regexp: '([,{])EXPLORER_URL:[^,}*([,}])'*

print "Replacing variable 'NETWORK_ID' to 'localnet' using regexp: '([,{])NETWORK_ID:[^,}*([,}])*'" *Replacing variable 'NETWORK_ID' to 'localnet' using regexp: '([,{])NETWORK_ID:[^,}*([,}])'*

print "Replacing variable 'ACCOUNT_ID_SUFFIX' to 'test.near' using regexp: '([,{])ACCOUNT_ID_SUFFIX:[^,}*([,}])*'" *Replacing variable 'ACCOUNT_ID_SUFFIX' to 'test.near' using regexp: '([,{])ACCOUNT_ID_SUFFIX:[^,}*([,}])'*

print "Replacing variable 'ACCESS_KEY_FUNDING_AMOUNT' to '300000000000000000000000' using regexp: '([,{])ACCESS_KEY_FUNDING_AMOUNT:[^,}*([,}])*'" *Replacing variable 'ACCESS_KEY_FUNDING_AMOUNT' to '300000000000000000000000' using regexp: '([,{])ACCESS_KEY_FUNDING_AMOUNT:[^,}*([,}])'*

add_service service_id="wallet" Service 'wallet' added with service GUID 'wallet-1670311807'

print "Explorer wallet struct(public_url = struct(ip_address = \"127.0.0.1\", path = \"\", port_number = 8334, protocol = \"http\"))" Explorer wallet struct(public_url = struct(ip_address = "127.0.0.1", path = "", port_number = 8334, protocol = "http"))

Starlark code successfully run. Output was: { "contract_helper_service_url": "http://127.0.0.1:8330", "explorer_url": "http://127.0.0.1:8331", "near_node_rpc_url": "http://127.0.0.1:8332", "network_name": "localnet", "root_validator_key": { "account_id": "test.near", "public_key": "ed25519:3Kuyi2DUXdoHgoaNEvCxa1m6G8xqc6Xs7WGajaqLhNmW", "secret_key": "ed25519:2ykcMLiM7vCmsSECcgfmUzihBtNdBv7v2CxNi94sNt4R8ar4xsrMMYvtsSNGQDfSRhNWXEnZvgx2wzS9ViBiS9jW" }, "wallet_url": "http://127.0.0.1:8334" } INFO[2022-12-06T13:00:10+05:30] == INFO[2022-12-06T13:00:10+05:30] || Created enclave: near || INFO[2022-12-06T13:00:10+05:30] ==
== SUCCESS == ACTION Paste the following in your terminal to declare the following variables so you can use them:

export NEAR_ENV="local" export NEAR_CLI_LOCALNET_NETWORK_ID="localnet" export NEAR_NODE_URL="http://127.0.0.1:8332" export NEAR_CLI_LOCALNET_KEY_PATH="/Users/zerix/.neartosis/2022-12-06T12.59.04/validator-key.json" export NEAR_WALLET_URL="http://127.0.0.1:8334" export NEAR_HELPER_URL="http://127.0.0.1:8330" export NEAR_HELPER_ACCOUNT="test.near \"ed25519:2ykcMLiM7vCmsSECcgfmUzihBtNdBv7v2CxNi94sNt4R8ar4xsrMMYvtsSNGQDfSRhNWXEnZvgx2wzS9ViBiS9jW\"}) ed25519:2ykcMLiM7vCmsSECcgfmUzihBtNdBv7v2CxNi94sNt4R8ar4xsrMMYvtsSNGQDfSRhNWXEnZvgx2wzS9ViBiS9jW"}) test.near" export NEAR_EXPLORER_URL="http://127.0.0.1:8331"

ACTION Paste the following into your terminal now to use the 'local_near' command as a replacement for the NEAR CLI for connecting to your local cluster (e.g. 'local_near login'):

alias local_near='NEAR_ENV="local" NEAR_CLI_LOCALNET_NETWORK_ID="localnet" NEAR_NODE_URL="http://127.0.0.1:8332" NEAR_CLI_LOCALNET_KEY_PATH="/Users/zerix/.neartosis/2022-12-06T12.59.04/validator-key.json" NEAR_WALLET_URL="http://127.0.0.1:8334" NEAR_HELPER_URL="http://127.0.0.1:8330" NEAR_HELPER_ACCOUNT="test.near \"ed25519:2ykcMLiM7vCmsSECcgfmUzihBtNdBv7v2CxNi94sNt4R8ar4xsrMMYvtsSNGQDfSRhNWXEnZvgx2wzS9ViBiS9jW\"}) ed25519:2ykcMLiM7vCmsSECcgfmUzihBtNdBv7v2CxNi94sNt4R8ar4xsrMMYvtsSNGQDfSRhNWXEnZvgx2wzS9ViBiS9jW"}) test.near" NEAR_EXPLORER_URL="http://127.0.0.1:8331" near'

ACTION If you want the 'local_near' command available in all your new terminal windows, add the above alias into your .bash_profile/.bashrc/.zshrc file and open a new terminal window.

ACTION To stop your cluster, run the following:

kurtosis enclave stop near

ACTION To remove your cluster, run:

kurtosis clean -a

========== SUCCESS ========== tip The URLs and validator key value above will be the same for each run of Kurtosis, so you can safely use these values in your config files. tip If you're running Kurtosis on a remote machine, you'll also need to:

- Replace the127.0.0.1
- IP addresses in the environment variables with the IP address of your remote machine
- Copy the validator key from where it lives on the machine running Kurtosis (inNEAR_CLI_LOCALNET_KEY_PATH
-) to somewhere on your local machine
- Adjust the value of theNEAR_CLI_LOCALNET_KEY_PATH
- on your local machine to match the location you stored the key at If you ever forget the above URLs, you can inspect the cluster:

kurtosis enclave inspect near

Setup Environment Variables

After deploying your Kurtosis NEAR Package, you will need to setup some environment variables to make life a lot easier. Notice theACTION sections in your terminal log from the package deployment. You will be using these exact values to setup these variables.

1. Follow the first ACTION item from the deployment log by copying all of the export commands and running them in your terminal.

Example exports: (DO NOT COPY ~ yours will be slightly different)

export NEAR_ENV="local" export NEAR_CLI_LOCALNET_NETWORK_ID="localnet" export NEAR_NODE_URL="http://127.0.0.1:8332" export NEAR_CLI_LOCALNET_KEY_PATH="/Users/zerix/.neartosis/2022-06-03T18.04.32/validator-key.json" export NEAR_WALLET_URL="http://127.0.0.1:8334" export NEAR_HELPER_URL="http://127.0.0.1:8330" export NEAR_HELPER_ACCOUNT="test.near" export NEAR_EXPLORER_URL="http://127.0.0.1:8331" 1. Proceed to the second ACTION item which asks you to create an alias forlocal_near 2. . This is what we will use when runningnear-cli 3. commands with our test environment.

Example alias: (DO NOT COPY ~ yours will be slightly different)

alias local_near='NEAR_ENV="local" NEAR_CLI_LOCALNET_NETWORK_ID="localnet" NEAR_NODE_URL="http://127.0.0.1:8332" NEAR_CLI_LOCALNET_KEY_PATH="/Users/zerix/.neartosis/2022-06-03T18.04.32/validator-key.json" NEAR_WALLET_URL="http://127.0.0.1:8334" NEAR_HELPER_URL="http://127.0.0.1:8330" NEAR_HELPER_ACCOUNT="test.near" NEAR_EXPLORER_URL="http://127.0.0.1:8331" near' Now replacingnear withlocal_near when runningnear-cli commands will perform these actions in your local test environment.

Testing

Ensure that your alias is working correctly by checking the state of the root accounttest.near .

Run the following in your terminal:

local_near state test.near This should return something similar to the following output:

Loaded master account test.near key from /Users/zerix/.neartosis/2022-06-03T18.04.32/validator-key.json with public key = ed25519:3Kuyi2DUXdoHgoaNEvCxa1m6G8xqc6Xs7WGajaqLhNmW Account test.near { amount: '1000000000000000000000000000000000', block_hash: 'G8jx4pYgqFSFSCDyM9MvVYj3HAdgRuxhkAHGweNhUNrY', block_height: 224, code_hash: '11111111111111111111111111111111', locked: '500000000000000000000000000000', storage_paid_at: 0, storage_usage: 182, formattedAmount: '1,000,000,000' } Congratulations! Setup is complete and you are ready to start exploring your local NEAR blockchain!

tip The Kurtosis Team has created a greatvideo presentation that covers the above steps as well as demoing the functionality of this local network setup.

Using Wallet and Explorer

Local NEAR Wallet

Now that you haveeverything setup , create an account using your local NEAR Wallet at 127.0.0.1:8334.

The account creation is exactly the same as on mainnet or testnet butonly the passphrase recovery mode will work here. Also note that the root account istest.near instead oftestnet ormainnet . This means that all the accounts you create will besubaccounts oftest.near . (ex.benji.test.near)

tip Because the Wallet will start on the same host and port, and the Wallet stores your account information in your browser's local storage, if you've created an account with a previous Kurtosis network then the Wallet will be storing the old account's information (which won't work with your new network). If this is the case for you, you'll need to clear the Wallet's storage by visitingyour Chrome's local storage , removing the127.0.0.1 entry, and refreshing the Wallet. Now that you've created an account, try interacting with it using the local CLI. In order to use this account you will need to "login" with it via CLI which will save a full access key locally for that account.near login is the command to perform this action but as you are onlocalnet you will need to replacenear withlocal_near .

local_near login This launches the local wallet site and will ask for confirmation for this action. Once you authorize you should see confirmation in your terminal similar to this:

Logged in as [kevin.test.near] with public key [ed25519:8EaALn...] successfully * Export your account ID to an environment variable by running the following: (replacing YOUR_ACCOUNT_ID)

export ACCOUNT_ID=YOUR_ACCOUNT_ID * Now create a test transaction by sending 1 NEAR to the root accounttest.near * :

local_near send ACCOUNT_ID test.near 1 Example response: Sending 1 NEAR to test.near from kevin.test.near Loaded master account test.near key from /Users/zerix/.neartosis/2022-06-03T18.04.32/validator-key.json with public key = ed25519:3Kuyi2DUXdoHgoaNEvCxa1m6G8xqc6Xs7WGajaqLhNmW Transaction Id 3e3H5zqj9noKGYTCMqeZ5pb4NWK7tQsjYKak6ybtpgvD To see the transaction in the transaction explorer, please open this url in your browser http://127.0.0.1:8331/transactions/3e3H5zqj9noKGYTCMqeZ5pb4NWK7tQsjYKak6ybtpgvD

Local NEAR Explorer

Again, now that you haveeverything setup you can view the transaction details of the command you just performed at the end of the last section . Notice that the last line of the terminal log displays a link to the transaction details in your local NEAR Explorer.

Sending 1 NEAR to test.near from kevin.test.near Loaded master account test.near key from /Users/zerix/.neartosis/2022-06-03T18.04.32/validator-key.json with public key = ed25519:3Kuyi2DUXdoHgoaNEvCxa1m6G8xqc6Xs7WGajaqLhNmW Transaction Id 3e3H5zqj9noKGYTCMqeZ5pb4NWK7tQsjYKak6ybtpgvD To see the transaction in the transaction explorer, please open this url in your browser http://127.0.0.1:8331/transactions/3e3H5zqj9noKGYTCMqeZ5pb4NWK7tQsjYKak6ybtpgvD * Click on this link or copy/paste it into your browser:

Here everything behaves exactly like thetestnet ormainnet NEAR Explorer except it is retrieving data from your local NEAR blockchain!

- If you ever need to open your local NEAR Explorer, you can always visit 127.0.0.1:8331 or run:

echo NEAR_EXPLORER_URL Example Response:

http://127.0.0.1:8331

Deploy a Smart Contract

With everything setup and yourtest.near account created, it's time to deploy a smart contract onlocalnet . For this example you will deploy an NFT use a pre-compiled WASM smart contract fromthis NFT example .

- Download the smart contract:

curl -o ~/main.wasm https://github.com/near-examples/nft-tutorial/raw/main/out/main.wasm -L * Deploy the smart contract:

local_near deploy --wasmFile ~/main.wasm --accountId ACCOUNT_ID Example response: Loaded master account test.near key from /Users/benjaminkurrek/.neartosis/2021-12-02T13.37.41/validator-key.json with public key = ed25519:AnLHi4ZAxfxFAQSXniycyZS6dpBqxhmVZH3zBCZbqAS6 Starting deployment. Account id: goteam.test.near, node: http://127.0.0.1:62285, helper: http://127.0.0.1:62286, file: /Users/benjaminkurrek/main.wasm Transaction Id 7atHm2piVehEitYeMF2FxWuRJVd6ZdRQEo3K83P98GuR To see the transaction in the transaction explorer, please open this url in your browser http://127.0.0.1:62290/transactions/7atHm2piVehEitYeMF2FxWuRJVd6ZdRQEo3K83P98GuR Done deploying to goteam.test.near * Click on the clink to the Explorer and verify that the contract was deployed:

Now, let's interact with the deployed contract.

- First, initialize the contract by running the following command:

local_near call ACCOUNT_ID new_default_meta '{"owner_id": "'ACCOUNT_ID'"}' --accountId ACCOUNT_ID This will initialize the contract with some default metadata and set our account ID as the owner of the contract.

- Now mint your first NFT!

local_near call ACCOUNT_ID nft_mint '{"token_id": "team_token", "metadata": { "title": "Go Team!", "description": "Go Team!", "media": "https://bafybeiftczwrtyr3k7a2k4vutd3amkwsmaqyhrdzlhvpt33dyjivufqusq.ipfs.dweb.link/goteam-gif.gif", "copies": 1}, "receiver_id": "'ACCOUNT_ID'"}' --accountId ACCOUNT_ID --amount 0.1 Once the NFT has been minted, you can view the token on the local wallet's collectibles tab. If you already had the wallet site open, simply refresh. Otherwise open your local NEAR Wallet instance and view your collectible.

We won't spoil what the NFT is, but once you switch over to the collectibles tab your beautiful token should be there!

Connecting a dApp tolocalnet

The ability to develop decentralized applications locally is a dream come true for dApp developers and the Kurtosis NEAR Package really simplifies this process. Here you'll integratelocalnet into one of the examples atnear.dev .

Clone Example dApp

- Clone theNEAR Guestbook
- repository:

git clone https://github.com/near-examples/guest-book-examples.git

Configure Network

- Open thesrc/config.js
- file inside the guestbook repo and scroll down to thelocal
- config:

case

'local' : return

{ networkId :

'local' , nodeUrl :

'http://localhost:3030' , keyPath :

{ process . env . HOME } /.near/validator_key.json , walletUrl :

'http://localhost:4000/wallet' , contractName :

CONTRACT_NAME } ; Here you will need to update all of the valuesexcept thecontractName . Copy the following into yourlocal case. This will automatically assign the config to your environment variables if you've specified them. This way, you don't need to change anything when running new enclaves.

case

'local' : return

{ networkId : process . env . NEAR_CLI_LOCALNET_NETWORK_ID

||

'local' , nodeUrl : process . env . NEAR_NODE_URL

||

'http://localhost:3030' , keyPath : process . env . NEAR_CLI_LOCALNET_KEY_PATH

||

{ process . env . HOME } /.near/validator_key.json , walletUrl : process . env . NEAR_WALLET_URL

||

'http://localhost:4000/wallet' , contractName :

CONTRACT_NAME , } ; The last thing you will need to do is set yourNODE_ENV in your terminal tolocal so your dApp will use the values we configured above.

- Run:

export NODE_ENV=local Your dApp is now fully configured to uselocalnet !

Create Contract Account

As mentioned earlier, you do not need to change thecontractName in theconfig.js file you updated earlier. This is an environment variable you will configure now. Let's quickly create a subaccount fromthe account you created earlierto deploy the guest-book contract to.

- Using thelocal_near CLI
- , run the following command:

local_near create-account guest-book.ACCOUNT_ID --masterAccount ACCOUNT_ID --initialBalance 5 Example Response:

Loaded master account test.near key from /Users/benjaminkurrek/.neartosis/2021-12-02T13.37.41/validator-key.json with public key = ed25519:AnLHi4ZAxfxFAQSXniycyZS6dpBqxhmVZH3zBCZbqAS6 Saving key to 'undefined/localnet/guest-book.goteam.test.near.json' Account guest-book.goteam.test.near for network "localnet" was created. * Export theCONTRACT_NAME * environment variable as the account you just created:

export CONTRACT_NAME=guest-book.ACCOUNT_ID

Deploy Contract tolocalnet

With the network setup and contract account created you are now ready to launch your dApp!

- Run the following command in the root directory of the guest book repo:

yarn && yarn start Example Response:

- Open the dApp by clicking on the server address in the terminal:

Server running at http://localhost:1234 ✨ Built in 1.20s. You should see the Guest Book landing page:

tip If you run into any problems signing into try clearing your browser's local storage. If you've used the guest-book before your browser might think you're still logged in with yourtestnet account and it will throw an error saying it can't find that account onlocalnet . Once you've logged in, you can sign a message with an optional donation.

- Sign the Guest Book which will create a transaction onlocalnet
- .
- Once complete, open your local NEAR explorer and you can view the transaction you just created!

Congratulations! You've successfully deployed and interacted with a dApp on a local NEAR blockchain!

Managing NEAR Pacakages

The Kurtosis NEAR Pacakages you create will continue to run on your local machine for as long as your Docker engine is running. This package runs inside of a Kurtosis "enclave" which is an environment isolated from both your computer and other enclaves. In practice, this means that you can have multiple independent local NEAR clusters running on your machine simply by rerunning the script we executed from thesetup instructions .

View Package Status

- To see the status of your existing enclaves, run:

kurtosis enclave ls

View Package Details

- To see detailed information about an enclave, copy an enclave ID and run:

kurtosis enclave inspect near

Terminate Package

- To shut down your NEAR Package and free up resources on your machine, run the following:

kurtosis enclave stop near note You will not be able to restart the cluster! If this is something you need, pleasefile an issue here so we can prioritize it.

Delete Package

- Stopping an enclave leaves its resources intact so that you can examine them if need be. To destroy a stopped enclave and free its resources, run:

kurtosis clean

Delete All Package

If you would like to destroyall enclaves, regardless of if they're running, pass the-a flag toclean like so:

kurtosis clean -a This can be a handy way to clear all your Kurtosis data.Edit this page Last updatedonApr 12, 2024 byDamian Parrino Was this page helpful? Yes No Need some help?Chat with us or check ourDev Resources ! Twitter Telegram Discord Zulip

Previous Integration Tests Next Deploying and Using