CodeChain Documentation

CodeChain Team <codechain@kodebox.io>

Jun 22, 2018

Contents

CodeChain is a programmable open source blockchain technology optimal for developing and customizing multi-asset management systems.

CHAPTER 1

Setup

1.1 Build Dependencies

CodeChain requires Rust version 1.26 to build. Using rustup is recommended.

- For Linux Systems:
  - Ubuntu

> **Note:** `gcc` and `g++` are required for installing packages.

```
$ curl https://sh.rustup.rs -sSf | sh
```

- For Mac Systems:
  - MacOS 10.13.2 (17C88) tested

> **Note:** `clang` is required for installing packages.

```
$ curl https://sh.rustup.rs -sSf | sh
```

- For Windows Systems:
  - Currently not supported for Windows. If on a Windows system, please install WSL to continue as Ubuntu.

Please make sure that all of the binaries above are included in your `PATH`. These conditions must be fulfilled before building CodeChain from source.

1.2 Building From Source

Download CodeChain's source code and go into its directory.

```
git clone git@github.com:CodeChain-io/codechain.git
cd codechain
```

1.2.1 Build as Release Version

```
cargo build --release
```

This will produce an executable in the ./target/release directory.

CHAPTER 2

Configuration

CodeChain can be configured with either CLI options or a config file. When it comes to which options take precedence, it goes from CLI, user's own config.toml file, and config.dev.toml in that order.

CLI options can be listed by running the command `$codechain --help`. By using the CLI options, or custom config files, the user can overwrite config.dev.toml's configurations.

2.1 Config File

The default preset `config.dev.toml` file can be located in `codechain/config/presets/config.dev.toml`.

Config files can be customized by the user and its location can be designated by using the CLI command `--config-path`. Custom config files created by the user must have the proper custom path.

2.2 Default config.dev.toml

The following represents the default configuration values of `config.dev.toml`.

```
[codechain]
quiet = false
db_path = "db"
chain = "solo"
enable_block_sync = true
enable_parcel_relay = true
secret_key = "0x0000000000000000000000000000000000000000000000000000000000000001"
```

CodeChain is set to use the Tendermint consensus algorithm by default. Tendermint is not suitable for solo testing purposes, since it requires a minimum of 4 users to function properly.

In order to test CodeChain alone, one may want to change `change_type` into Solo. To do this, use `--chain solo`.

2.3 CLI Options for CodeChain client

--config-path=[PATH] Specify the certain config file path that you want to use to configure CodeChain to your needs.

--port=[PORT] Listen for connections on PORT. (default: 3485)

--bootstrap-addresses=[BOOTSTRAP_ADDRESSES] Bootstrap addresses to connect.

--no-network Do not open network socket.

--min-peers=[NUM] Sets the minimum number of connections the user would like. (default: 10)

--max-peers=[NUM] Sets the maximum number of connections the user would like. (default: 30)

--instance-id=[ID] Specify instance id for logging. Used when running multiple instances of CodeChain.

--quiet Do not show any synchronization information in the console.

--chain=[CHAIN] Sets the blockchain type out of solo, solo_authority, tendermint or a path to chain spec file. (default: tendermint)

--db-path=[PATH] Specify the database directory path.

--no-sync Do not run block sync extension.

--no-parcel-relay Do not relay parcels.

--jsonrpc-port=[PORT] Listen for rpc connections on PORT. (default: 8080)

--no-jsonrpc Do not run jsonrpc.

--secret-key=[KEY] Secret key used by node.

--author=[ADDRESS] Specify the block's author (aka "coinbase") address for sending block rewards from sealed blocks.

--engine-signer=[ADDRESS] Specify the address which should be used to sign consensus messages and issue blocks.

--no-discovery Do not use discovery. No automated peer finding.

--discovery="kademlia" | "unstructured" Decides which p2p discovery extension to use. Options are kademlia and unstructured. In a testing environment, an unstructured p2p network is desirable because it is more than sufficient when there are a few users. (default: unstructured)

--discovery-bucket-size=[NUM] Bucket size for discovery. Choose how many addresses to exchange at a time during discovery.

--discovery-refresh=[ms] Refresh timeout of discovery (ms). It may conflict with:‘‘ –no-discovery‘‘.

2.3.1 Subcommands

CodeChain has a subcommand called `account`. It has subcommands of its own, which are the following:

create create account

--passphrase <PASSWORD> account passphrase

import import private key

--passphrase <PASSWORD> set account passphrase

--raw-key <RAW-KEY> specify key to import

list list managed accounts

For example, if you want to create an account with a password of '1234', run the following:

```
./target/release/codechain account create --passphrase 1234
```

2.4 Logging

For logging, run the following to configure: `$ RUST_LOG=<level> codechain`

2.4.1 Log Levels

CodeChain currently offers five different `<level>`. They are error, warn, info, debug, and trace.

For example, the log level will be set to debug, if you run the following:

```
$ RUST_LOG="debug" codechain
```

- The **error** level represents an event where something can be dangerous, but can still run. In the case in which it cannot run anymore, it must crash ASAP instead of logging.
- The **warn** level represents an event which can be potentially dangerous.
- The **info** level represents an event which is not dangerous, but can be useful information for users.
- The **debug** level represents an event that is useful for developers, but not for users.
- The **trace** level is used for tracing.

2.4.2 Log Targets

Log levels can be set differently for each log targets. For example, you can run the following to set `tx`'s log level as `trace` and `parcel`'s log level as `info` with the following code:

```
$ RUST_LOG="tx=trace, parcel=info" codechain
```

The possible log targets are as follows:

```
"blockchain"
"client"
"discovery"
"engine"
"external_parcel"
"io"
"miner"
"net"
"netapi"
"own_parcel"
"parcel_queue"
"poa"
"shutdown"
"snapshot"
"solo_authoirty"
"spec"
"state"
```

(continues on next page)

(continued from previous page)

```
"state_db"
"stratum"
"sync"
"test_script"
"trie"
"tx"
```

CHAPTER 3

Basic Usage

3.1 Run Built Executable

To get started, you must first run the built executable of CodeChain.

In order to run CodeChain, run

```
./target/release/codechain
```

You can create a block by sending a parcel through JSON-RPC. In order to utilize JSON-RPC, you can use Curl or JavaScript SDK.

3.2 Blockchain Configuration

When configuring CodeChain's blockchain type, you can set it to either `Solo` or `Tendermint`.

3.2.1 Solo Configuration

CodeChain uses this configuration as default. In order to change it into solo from another configuration, run:

```
--chain solo
```

3.2.2 Tendermint Configuration

In order to properly get Tendermint to get going, you need to have 4 nodes up and running. To do this, first run a single node by running the following:

```
codechain --db-path db/db0 --port 3485 --jsonrpc-port 8080 --secret-key␣
↪0000000000000000000000000000000000000000000000000000000000000001 -c tendermint
```

This creates a node in db0 (database 0) at port 3485(used for nodes to communicate with each other) and jsonRPC port 8080(port used for external access) with a secret key of 1. By default, secret key values of 1,2,3,and 4 correspond to the public keys of the validator, which are located in the tendermint spec file. All the public keys must be satisfied by having a corresponding secret key amongst the nodes of the blockchain network. Only then will Tendermint function properly.

Then create more nodes, and allocate each node with a secret key that corresponds to one of the four public keys listed in Tendermint's validator property. When creating new nodes, the db, port and jsonRPC port all must be configured as a different value. So for example, the next node should be set up like this:

```
codechain --db-path db/db1 --port 3486 --jsonrpc-port 8081 --secret-key␣
↪0000000000000000000000000000000000000000000000000000000000000002 -c tendermint
```

Once each public key has a corresponding node with a corresponding secret key, use the boostrap address command to interlink all the nodes together. The way each node is connected does not matter, as long as each node is connected to another node. For example, in order to make a certain node connect to the node with a secret key of 1, use this command:

```
codechain --db-path db/db0 --port 3485 --jsonrpc-port 8080 --secret-key␣
↪0000000000000000000000000000000000000000000000000000000000000001 -c tendermint --
↪bootstrap-addresses 127.0.0.1:8080
```

3.3 Checking if CodeChain is Configured Properly

JSON-RPC is a stateless, light-weight remote procedure call (RPC) protocol. Primarily this specification defines several data structures and the rules around their processing. It is transport agnostic in that the concepts can be used within the same process, over sockets, over HTTP, or in many various message passing environments. It uses JSON (RFC 4627) as data format.

3.3.1 Using Curl

First, check whether CodeChain's RPC port is listening for RPC connections. By default it should be PORT 8080.

In order to check whether CodeChain is configured properly or not, send a ping to check whether CodeChain's RPC server is actually responding. To do this, do the following:

```
curl \
    -H 'Content-Type: application/json' \
    -d '{"jsonrpc": "2.0", "method": "ping", "params": [], "id": null}' \
    localhost:8080
```

You should get the following response, or something similar:

```
{"jsonrpc":"2.0","result":"pong","id":null}
```

3.3.2 Using JavaScript SDK

In order to use this method, first install the sdk by running the following:

```
npm install codechain-sdk
```

or

```
yarn add codechain-sdk
```

Then, make sure that your CodeChain RPC server is listening. In the examples, we assume it is localhost:8080

If you run the following code, your should receive a ping response:

```
// ping.js (javascript)
var SDK = require("codechain-sdk");

var sdk = new SDK("http://localhost:8080");

sdk.ping().then(function (response) {
    console.log("Ping response:", response);
}).catch(console.error);
```

If you want to run the above example in the command line, first install `nvm` by running the following:

```
wget -qO- https://raw.githubusercontent.com/creationix/nvm/v0.33.11/install.sh | bash
```

Then run the following:

```
node -e 'var SDK = require("codechain-sdk"); var sdk = new SDK("http://localhost:8080
↪");sdk.ping().then(function (response) {console.log("Ping response:", response); }).
↪catch(console.error);'
```

You should receive the following response:

```
Ping response: pong
```