Setting Up a Bitcoin/Lightning Network Test Environment

Michael Goldstein Following
Feb 4 · 6 min read

Pierre Rochard has infected me with a zest for lightning development. As a web developer, my main focus is on building web apps that have Lightning payments enabled, rather than protocol-level or wallet-level work. However, in order to work on that, you need the whole stack below functioning properly so you can focus on your level in the stack. I quickly found that jumping into testnet payments is frustrating, because I spent more time getting nodes up and running and loaded up with testnet bitcoins and connecting to other nodes than actually coding on web apps. By using regtest, I can ensure a predictable Bitcoin and Lightning experience under the hood so I can focus on web development.

In this article I am going to walk through my Lightning setup using regtest, so that you can get up and running much faster than I did. This article assumes you have installed `bitcoind` and `lnd` .

What is Regtest?

The Bitcoin developer glossary defines regtest as such:

> *A local testing environment in which developers can almost instantly generate blocks on demand for testing events, and can create private satoshis with no real-world value.*

Basically, you get to create your own Bitcoin network to test individual nodes and also how they operate together. For the sake of what we're doing, we only need one node.

Setting Up Bitcoin

First things first, we need a Bitcoin node. When building a particular app, I like to put all node data directories in a single directory so that it is easier to find and point to particular resources along the way.

```
$ mkdir ~/app-container
$ mkdir ~/app-container/.bitcoin
```

Next, you'll want to create a configuration file `bitcoin.conf` that goes in this directory:

```
regtest=1
daemon=1
txindex=1
rpcauth=<rpc auth username and password>
zmqpubrawblock=tcp://127.0.0.1:28332
zmqpubrawtx=tcp://127.0.0.1:28333
```

This tells Bitcoin Core to use regtest mode, run as a daemon, and keep an index of all transactions. It also sets up RPC authentication and ZMQ ports for LND to use to communicate with the node. To produce the value to enter for `rpcauth` , use the script found at `./share/rpcauth/rpcauth.py` in the Bitcoin Core repo. In this article we'll assume you are using the username `lnd` and the password `lightning` .

Finally, I like to make a bash alias to making calling these programs easier. Open up `~/.bash_profile` and add this to the bottom.

```
export BITSTEIND_DIR="$HOME/app-container/.bitcoin"


alias bitsteind="bitcoind -datadir=$BITSTEIND_DIR"
alias bitstein-cli="bitcoin-cli -datadir=$BITSTEIND_DIR"
```

Be sure to update the path to whatever directory you had set Bitcoin up in. `BITSTEIN_DIR` , `bitsteind` , and `bitstein-cli` can be changed to whatever you prefer.

Save the file and load it in the shell by restarting the terminal or running `source ~/.bash_profile` .

Now, you can run your Bitcoin node from the terminal.

```
$ bitsteind
```

Go ahead and mine the easiest blocks you've ever tried to mine:

```
$ bitstein-cli generate 5
```

Coins are not spendable by miners until there are 100 confirmations, so let's speed things along with more blocks.

```
$ bitstein-cli generate 101
```

Now you should see a positive balance:

```
$ bitstein-cli getbalance
```

Bitcoin is up and running! Learn more about available RPC commands here (hint: `bitstein-cli stop` will help you shut down).

Setting Up LND

Let's make a directory for our LND nodes. Yes, nodes. We're going to make two of them.

```
$ mkdir ~/app-container/.lnd
$ mkdir ~/app-container/.lnd2
```

Now let's make configuration files. The first node can use default values, so it will be a simpler file. Save it as `lnd.conf` in the `.lnd` directory.

```
[Bitcoin]

bitcoin.active=1
bitcoin.regtest=1
bitcoin.node=bitcoind

[Bitcoind]

bitcoind.rpchost=localhost
bitcoind.rpcuser=lnd
bitcoind.rpcpass=lightning
bitcoind.zmqpubrawblock=tcp://127.0.0.1:28332
bitcoind.zmqpubrawtx=tcp://127.0.0.1:28333
```

This tells LND about the Bitcoin node it is using and the RPC and ZMQ details for communicating. Since this is a local test environment, the password is not very secure. If you used a different password in the Bitcoin RPC auth script, be sure to update the value here.

Now we will make `lnd.conf` in the `.lnd2` directory:

```
[Application Options]

listen=0.0.0.0:9734
rpclisten=localhost:11009
restlisten=0.0.0.0:8180

[Bitcoin]

bitcoin.active=1
bitcoin.regtest=1
bitcoin.node=bitcoind

[Bitcoind]

bitcoind.rpchost=localhost
bitcoind.rpcuser=lnd
bitcoind.rpcpass=lightning
bitcoind.zmqpubrawblock=tcp://127.0.0.1:28332
bitcoind.zmqpubrawtx=tcp://127.0.0.1:28333
```

This sets up different ports for network, RPC, and REST connections. The rest is the same. Once again, make sure you use the Bitcoin RPC username/password you used in the script earlier.

Now let's set up some aliases.

```
export LND1_DIR="$HOME/app-container/.lnd"
export LND2_DIR="$HOME/app-container/.lnd2

alias lnd1="lnd --lnddir=$LND1_DIR";
alias lncli1="lncli -n regtest --lnddir=$LND1_DIR"

alias lnd2="lnd --lnddir=$LND2_DIR";
alias lncli2="lncli -n regtest --lnddir=$LND2_DIR --
rpcserver=localhost:11009"
```

The CLI command has to know that the node is using regtest so it looks in the right path for the macaroons and TLS certs. The second node has a non-default RPC server, so we go ahead and set that in the CLI too. Thanks to aliases, we don't have to type those out every time we make a command.

Running LND

Is your Bitcoin node still running? Good. Let's get the Lightning nodes up. Open up a terminal window.

```
$ lnd1
```

We need to create or unlock the wallet. Open up another terminal window.

```
$ lncli1 create
```

Follow the prompt and save the mnemonic. Once again, this is a local test environment, so don't worry much about security. There's no real money at stake.

```
$ lncli1 getinfo
```

This should bring up some basic info about the node, a good barometer of if everything worked properly.

Next, do the same but with `lnd2` / `lncli2` .

Connecting the Nodes

If you run `lncli1 listpeers` , you'll find out your node is still all alone. Let's get them connected. Let's get the second node's information.

```
$ lncli2 getinfo
```

The output should have the following:

```
{
    "identity_pubkey":
```

```
"02fea0a032d365dfabe00347bad493281172fcf30564f40b8a45da6bc17
6349a67",
    "alias": "02fea0a032d365dfabe0",
    [...]
    "chains": [
        "bitcoin"
    ],
    "uris": [
    ],
    "best_header_timestamp": "1549304792",
    "version": "0.5.1-beta commit=",
    "num_inactive_channels": 0
}
```

Let's connect the first node to the second with the following command:

```
$ lncli1 connect
02fea0a032d365dfabe00347bad493281172fcf30564f40b8a45da6bc176
349a67@localhost:9734
```

If you connect from the second node to the first, you'll use port `9735` , which is the default.

Now run `lncli1 listpeers` again. Your output will be similar to this:

```
{
    "peers": [
        {
            "pub_key":
"02fea0a032d365dfabe00347bad493281172fcf30564f40b8a45da6bc17
6349a67",
            "address": "127.0.0.1:50821",
            "bytes_sent": "8764",
            "bytes_recv": "14968",
            "sat_sent": "0",
            "sat_recv": "600",
            "inbound": true,
            "ping_time": "499"
        }
    ]
}
```

Now you're ready to create a channel and send money.

Creating a Channel

To create a channel, first we need some bitcoins in our second node. Let's get an address.

```
$ lncli2 newaddress np2wkh
{
    "address": "2N3CvQotjCttq1xE9seJJ1RdgrmXoepuyNP"
}
```

Now, send one of those hard-earned bitcoins you mined earlier to the node. We'll generate some blocks to give it confirmations.

```
$ bitstein-cli sendtoaddress
2N3CvQotjCttq1xE9seJJ1RdgrmXoepuyNP 1
$ bitstein-cli generate 6
```

Now, we can open a channel. Let's get the pubkey of the first node:

```
$ lncli1 getinfo
```

Start a channel using the value you get there. We'll make the channel 100,000 satoshis:

```
$ lncli2 openchannel
026721a3077374a439d21d71987e63d8c111ea80272f3765658099075788
6765da 100000
```

Now mine some more blocks so the channel has adequate confirmations.

```
$ bitstein-cli generate 10
```

You should actually only need 3, but mining is fun.

Let's see if it worked:

```
$ lncli1 listchannels
{
    "channels": [
        {
            "active": true,
            "remote_pubkey":
"026721a3077374a439d21d71987e63d8c111ea80272f376565809907578
86765da",
            "channel_point":
"a28e4fda3d21fb681ffeb2cc72ff84e5cfc60e30674d3b8df0ce1eec620
87773:0",
            "chan_id": "131941395398656",
            "capacity": "500000",
            "local_balance": "600",
            "remote_balance": "490350",
            "commit_fee": "9050",
            "commit_weight": "724",
            "fee_per_kw": "12500",
            "unsettled_balance": "0",
            "total_satoshis_sent": "0",
            "total_satoshis_received": "600",
            "num_updates": "12",
            "pending_htlcs": [
            ],
            "csv_delay": 144,
            "private": false
        }
    ]
}
```

Lightning enabled!

Creating and Sending an Invoice

Let's round this out by sending a payment through our newly created channel.

```
$ lncli1 addinvoice —amt 100
{
    "r_hash":
"9a8d4f2400ec802e01466e16f3e7e5b69e9d137101026f7bc0a7bcb47f9
09ad7",
    "pay_req":
"lnbcrt1u1pw93syppp5n2x57fqqajqzuq2xdct08el9k60f6ym3qypx777q
577tglusnttsdqqcqzys8dwa6ycaxvk70l0zkwek7f3canyv99uqxzlsxzyk
0pual02wjc838utz5vv8e6zlzdlsdrnxrtnjw4u5688jrgswln0vfh0u8vjs
pmqq2ey0sj",
    "add_index": 34
}
```

Your `add_index` will be lower, since I've been creating many invoices already. Copy the `pay_req` value and take note of `r_hash` for later.

Let's inspect the invoice on the other node:

```
$ lncli2 decodepayreq
lnbcrt1u1pw93syppp5n2x57fqqajqzuq2xdct08el9k60f6ym3qypx777q5
77tglusnttsdqqcqzys8dwa6ycaxvk70l0zkwek7f3canyv99uqxzlsxzyk0
pual02wjc838utz5vv8e6zlzdlsdrnxrtnjw4u5688jrgswln0vfh0u8vjsp
mqq2ey0sj
{
    "destination":
"02fea0a032d365dfabe00347bad493281172fcf30564f40b8a45da6bc17
6349a67",
    "payment_hash":
"9a8d4f2400ec802e01466e16f3e7e5b69e9d137101026f7bc0a7bcb47f9
09ad7",
    "num_satoshis": "100",
    "timestamp": "1549320321",
    "expiry": "3600",
    "description": "",
    "description_hash": "",
    "fallback_addr": "",
    "cltv_expiry": "144",
    "route_hints": [
    ]
}
```

Now let's pay!

```
$ lncli2 payinvoice
lnbcrt1u1pw93syppp5n2x57fqqajqzuq2xdct08el9k60f6ym3qypx777q5
77tglusnttsdqqcqzys8dwa6ycaxvk70l0zkwek7f3canyv99uqxzlsxzyk0
pual02wjc838utz5vv8e6zlzdlsdrnxrtnjw4u5688jrgswln0vfh0u8vjsp
mqq2ey0sj
Description:
Amount (in satoshis): 100
Destination:
02fea0a032d365dfabe00347bad493281172fcf30564f40b8a45da6bc176
349a67
Confirm payment (yes/no): yes
{
    "payment_error": "",
    "payment_preimage":
"73b6dcd8d2f2df5ea0d3ded7facd2a277fdf0db0781a5dc8b00f7ff0e59
72290",
    "payment_route": {
        "total_time_lock": 270,
        "total_amt": 100,
        "hops": [
            {
                "chan_id": 131941395398656,
                "chan_capacity": 500000,
                "amt_to_forward": 100,
                "expiry": 270,
                "amt_to_forward_msat": 100000,
                "pub_key":
"02fea0a032d365dfabe00347bad493281172fcf30564f40b8a45da6bc17
6349a67"
            }
        ],
        "total_amt_msat": 100000
    }
}
```

Let's look up the invoice on the receiving node using the payment hash:

```
$ lncli1 lookupinvoice
9a8d4f2400ec802e01466e16f3e7e5b69e9d137101026f7bc0a7bcb47f90
9ad7
{
    "memo": "",
    "receipt": null,
    "r_preimage":
"c7bc2NLy316g097X+s0qJ3/fDbB4Gl3IsA9/8OWXIpA=",
    "r_hash":
"mo1PJADsgC4BRm4W8+fltp6dE3EBAm97wKe8tH+Qmtc=",
    "value": "100",
    "settled": true,
    "creation_date": "1549320321",
    "settle_date": "1549320499",
    "payment_request":
"lnbcrt1u1pw93syppp5n2x57fqqajqzuq2xdct08el9k60f6ym3qypx777q
577tglusnttsdqqcqzys8dwa6ycaxvk70l0zkwek7f3canyv99uqxzlsxzyk
0pual02wjc838utz5vv8e6zlzdlsdrnxrtnjw4u5688jrgswln0vfh0u8vjs
pmqq2ey0sj",
    "description_hash": null,
    "expiry": "3600",
    "fallback_addr": "",
    "cltv_expiry": "144",
    "route_hints": [
    ],
    "private": false,
    "add_index": "34",
    "settle_index": "7",
    "amt_paid": "100000",
    "amt_paid_sat": "100",
    "amt_paid_msat": "100000"
}
```

Settled: true.

If you're done, you can shut down the nodes:

```
$ bitstein-cli stop
$ lncli1 stop
$ lncli2 stop
```

Conclusion

This has been a quick walk through of my basic setup for a Bitcoin/Lightning test environment using regtest. In a future article, we'll walk through how to connect this node to Python scripts and Flask web apps.

If you have any questions, suggestions, or errata to submit, my DMs are open on Twitter. I hope this helps you get your creative dev juices flowing like it did for me!