clearinghoused$_{b}uildDocumentation$

Release 0.1.0

ClearingHouse Team

September 24, 2016

Contents

clearinghoused_build is the automated build system for clearinghoused. This is an alternative method from manual clearinghoused installation and running, which includes a point-and-click Windows installer, as well as a source code build script that takes care of all setup necessary to run `clearinghoused` from source.

Using the build system, you have the following options:

- If you are a **Windows user**, you can either use the installer package, or build from source
- If you are an **Ubuntu Linux user**, you can use the build system to automate your install/setup from source
- If you are **neither**, at this point you will need to follow the manual installation instructions.

CHAPTER 1

When to use?

This build system will probably be especially helpful in any of the following circumstances:

- You are a Windows user, or a Linux user that isn't super experienced with the command line interface.
- You want to deploy `clearinghoused` in a production environment, and have it run automatically on system startup
- You want to build your own `clearinghoused` binaries

CHAPTER 2

Future plans

Future plans for the build system (*pull requests for these features would be greatly appreciated*):

- Add support for Linux distributions beyond Ubuntu Linux
- Add support for Mac OS X automated setup from source
- Add support for creation of installer for Mac OS X
- Add support for creation of `.rpm`, `.deb`., etc. binary packages for Linux

More information on Clearinghouse is available in the specs.

CHAPTER 3

Table of Contents

Note: Documentation on the `clearinghoused` API exists at http://clearinghoused.rtfd.org.

3.1 Setting up viacoind

Warning:
This section sets up `clearinghoused` to run on mainnet, which means that when using it, you will be working with real XC
If you would like to run on testnet instead, please see the section entitled **Running clearinghoused on testnet** in Additional Topics.

`clearinghoused` communicates with the Viacoin reference client (`viacoind`). Normally, you'll run `viacoind` on the same computer as your instance of `clearinghoused` runs on. However, you can also use a `viacoind` instance sitting on a different server entirely.

This step is necessary whether you're building clearinghoused from source or using the installer package.

3.1.1 On Windows

If you haven't already, go to the viacoind download page and grab the installer for Windows. Install it with the default options.

Once installed, type Windows Key-R and enter `cmd.exe` to open a Windows command prompt. Type the following:

```
cd %APPDATA%\Viacoin
notepad viacoin.conf
```

Say Yes to when Notepad asks if you want to create a new file, then paste in the text below:

```
rpcuser=rpc
rpcpassword=rpcpw1234
server=1
daemon=1
txindex=1
```

NOTE:

- If you want `viacoind` to be on testnet, not mainnet, see the section entitled **Running clearinghoused on testnet** in Additional Topics.

- You should change the RPC password above to something more secure.

Once done, press CTRL-S to save, and close Notepad. The config file will be saved here:

```
``%AppData%\Roaming\Counterparty\clearinghoused\clearinghoused.conf``
```

New Blockchain Download

Next, if you haven't ever run Viacoin on this machine (i.e. no blockchain has been downloaded), you can just launch `viacoind` or `viacoin-qt` and wait for the blockchain to finish downloading.

Already have Blockchain

If you have already downloaded the blockchain on your computer (e.g. you're already using the Viacoin client) **and** you did not have the configuration parameter `txindex=1` enabled, you will probably need to open up a command prompt window, change to the Viacoin program directory (e.g. `C:\Program Files (x86)\Viacoin\`) and run:

```
viacoin-qt.exe --reindex
```

or:

```
daemon\viacoind.exe --reindex
```

This will start up viacoin to do a one time reindexing of the blockchain on disk. The reason this is is because we added the `txindex=1` configuration parameter above to the viacoin config file, which means that it will need to run through the blockchain again to generate the necessary indexes, which may take a few hours. After doing this once, you shouldn't have to do it again.

Next steps

Once this is done, you have two options:

- Close Viacoin-QT and run `viacoind.exe` directly. You can run it on startup by adding to your Startup program group in Windows, or using something like NSSM.
- You can simply restart Viacoin-QT (for the configuration changes to take effect) and use that. This is fine for development/test setups, but not normally suitable for production systems. (You can have Viacoin-QT start up automatically by clicking on Settings, then Options and checking the box titled "Start Viacoin on system startup".)

3.1.2 On Ubuntu Linux

If not already installed (or running on a different machine), do the following to install it (on Ubuntu, other distros will have similar instructions):

```
sudo apt-get install software-properties-common python-software-properties
sudo add-apt-repository ppa:viacoin/viacoin
sudo apt-get update
sudo apt-get install viacoind
mkdir -p ~/.viacoin/
echo -e "rpcuser=rpc\nrpcpassword=rpcpw1234\nserver=1\ndaemon=1\ntxindex=1" > ~/.viacoin/viacoin.con
```

Please then edit the `~/.viacoin/viacoin.conf` file and set the file to the same contents specified above in viacoin.conf example for Windows.

New Blockchain Download

Next, if you haven't ever run `viacoin-qt`/`viacoind` on this machine (i.e. no blockchain has been downloaded), you can just start `viacoind`:

```
viacoind
```

In either of the above cases, the viacoin server should now be started. The blockchain will begin to download automatically. You must let it finish downloading entirely before going to the next step. You can check the status of this by running:

```
viacoind getinfo | grep blocks
```

When done, the block count returned by this command will match the value given from this page.

Already have Blockchain

If you *have* already downloaded the blockchain before you modified your config and you did not have `txindex=1` enabled, you'll probably need to launch `viacoind` as follows:

viacoind –reindex

This will start up viacoin to do a one time reindexing of the blockchain on disk. The reason this is is because we added the `txindex=1` configuration parameter above to the viacoin config file, which means that it will need to run through the blockchain again to generate the necessary indexes, which may take a few hours. After doing this once, you shouldn't have to do it again.

If you had the blockchain index parameter always turned on before, reindexing should not be necessary.

Next steps

At this point you should be good to go from a `viacoind` perspective. For automatic startup of `viacoind` on system boot, this page provides some good tips.

3.2 Using the Installer

Warning: Due to the current pace of `clearinghoused` development, at the current moment, we do not offer a `clearinghoused` binary installer for windows. It is recommended that users instead follow the instructions in Building & Running from Source (which are really not that involved). The reason for this is because the Windows installer always lags current `clearinghoused` progress by a few days normally, and at this point in heavy development, this fact will most likely cause issues for its users. Building from source is the best way to keep up with the frequent updates.
Another alternative is to use BoottleXCP.

3.3 Building & Running from Source

Note: Please make sure you've followed the instructions in Setting up viacoind before moving through this section.

This section provides information about how to install and run `clearinghoused` from source, using this `clearinghoused` build system (as an alternative to setting it up manually). This method is suitable for Linux users, as well as Windows users that want to develop/enhance `clearinghoused` (or just don't want to use the binary installer).

3.3.1 On Windows

Prerequisites

Note: As of clearinghoused v9.34.0 due to issues with some Python modules, a 64-bit version of Python cannot be used to build Clearinghouse out-of-the-box. For time being it is recommended to use a 32-bit version of Python 3.3.5 on both the 32-bit and 64-bit version of Microsoft Windows (confirmed to work with Windows 7 SP1 x64).

Minimally required to build `clearinghoused` from source is the following:

- Python 3.3.5 – grab the 32-bit version or 64-bit version. Install to the default `C:\Python33` location
- Python Win32 extensions – grab the 32-bit version or 64-bit version
- APSW for Windows – grab the 32-bit version or 64-bit version
- Pycrypto for Windows – grab the 32-bit version or 64-bit version
- Git for Windows (should have already been downloaded for the `insight` setup). Download here and install. Use the default installer options (except, select *"Run Git from the Windows Command Prompt"* on the appropriate screen)
- You may need to install Visual C++ 2010 Express

If you want to be able to build the Clearinghoused installer, also download the following:

- Grab NSIS from here – Please choose the default options during installation, and install to the default path
- Download the NSIS SimpleService plugin from here and save the .dll file contained in that zip to your NSIS `plugins` directory (e.g. `C:\Program Files (X86)\NSIS\plugins`)
- cx_freeze – grab the 32-bit version or 64-bit version as appropriate
- Install a binary build of cherrypy-wsgiserver such as this (for 32-bit Python)

Installing

Note: Our install script (setup.py) requires administrator access to run (so that it can create a clearinghoused.bat file in your Windows directory). To allow for this, you must launch a command prompt **as administrator**. To do this under Windows 7, go to Start -> All Programs -> Accessories, then right click on Command Prompt and select "Run as administrator". More information on this is available from this link (method 1 or 2 works fine).

After launching a DOS command window using the instructions in the note above, type the following commands:

```
cd C:\
git clone https://github.com/ClearingHouse/clearinghoused_build
cd clearinghoused_build
C:\Python33\python.exe setup.py
```

The above steps will check out the build scripts to `C:\clearinghoused_build`, and run the `setup.py` script, which will check out the newest version of `clearinghoused` itself from git, create a virtual environment with the required dependencies, and do other necessary tasks to integrate it into the system.

If you chose to start `clearinghoused` at startup automatically, the setup script will also create a shortcut to `clearinghoused` in your Startup group.

Upon the successful completion of this script, you can now run `clearinghoused` using the steps below.

Running clearinghoused built from Source

Clearinghoused does not require elevated ("administrator") privileges to be executed and operated. After installing, open a command window and run `clearinghoused` in the foreground via:

```
clearinghoused server
```

You can then open up another command window and run any of `clearinghoused's` other functions, for example:

```
clearinghoused send --source=12WQTnVbzhJRswra4TvGxq1RyhUkmiVXXm --destination=VQGZ4sCpvCgRizL5v4NniaF
```

For more examples, see this link.

To run the `clearinghoused` testsuite:

```
clearinghoused tests
```

Updating to the newest source

As the code is enhanced and improved on Github, you can refresh your local copy of the repositories like so:

```
cd C:\clearinghoused_build
C:\Python33\python.exe setup.py update
```

If, upon running clearinghoused, you get a missing dependency or some other error, you can always rerun `setup.py`, which will regenerate your dependencies listing to the libraries and versions as listed in pip-requirements.txt:

```
cd clearinghoused_build
C:\Python33\python.exe setup.py
```

In case of a problem, refer to the list of requirements in `pip-requirements.txt` above and update system as necessary. Then rerun the build script again.

Building your own Installer

Complete the instructions under **Prerequisites** above. Then, execute the following commands to build the installer package:

```
cd C:\clearinghoused_build
C:\Python33\python.exe setup.py build
```

If successful, you will be provided the location of the resulting installer package.

3.3.2 On Linux

Prerequisites

Currently, Ubuntu Linux (Server or Desktop) **12.04 LTS**, **13.10**, and **14.04** are supported.

Support for other distributions is a future task.

Installing

As the user you want to run `clearinghoused` **as**, launch a terminal window, and type the following:

```
sudo apt-get -y update
sudo apt-get -y install git-core python3
git clone https://github.com/ClearingHouse/clearinghoused_build ~/clearinghoused_build
cd ~/clearinghoused_build
sudo python3 setup.py
```

The `setup.py` script will install necessary dependencies, check out the newest version of `clearinghoused` itself from git, create the python environment for `clearinghoused`, and install an upstart script that will automatically start `clearinghoused` on startup.

Creating a default config

Follow the instructions listed under the **Config and Logging** section in ViacoinTest Additional Topics.

Running clearinghoused built from Source

After installing and creating the necessary basic config, run `clearinghoused` in the foreground to make sure everything works fine:

```
clearinghoused server
```

(The above assumes `/usr/local/bin` is in your PATH, which is where the `clearinghoused` symlink (which just points to the `run.py` script) is placed. If not, run `/usr/local/bin/clearinghoused` instead.

Once you're sure it launches and runs fine, press CTRL-C to exit it, and then run `clearinghoused` as a background process via:

```
sudo service clearinghoused start
```

You can then open up another command window and run any of `clearinghoused's` other functions, for example:

```
clearinghoused send --source=V2WQTnVbzhJRswra4TvGxq1RyhUkmiVXXm --destination=VQGZ4sCpvCgRizL5v4NniaH
```

For more examples, see this link.

To run the `clearinghoused` testsuite:

```
clearinghoused tests
```

Updating to the newest source

As the code is enhanced and improved on Github, you can refresh your local copy of the repositories like so:

```
cd ~/clearinghoused_build
sudo python3 setup.py update
```

Clearinghouse for Windows must also be updated from a console window started with elevated privileges.

If, upon running clearinghoused, you get a missing dependency or some other error, you can always rerun `setup.py`, which will regenerate your dependencies listing to the libraries and versions as listed in pip-requirements.txt:

```
cd ~/clearinghoused_build
sudo python3 setup.py
```

The same approach applies to Windows - this operation requires elevation.

3.3.3 Mac OS X

Mac OS support is forthcoming. (Pull requests to add such support are more than welcome!)

3.4 ViacoinTest Additional Topics

This section contains some tidbits of info that you may find useful when working with `clearinghoused`.

For a good overview of what you can do with `clearinghoused`, see this link.

3.4.1 Finding the Data Directory

`clearinghoused` stores its configuration, logging, and state data in a place known as the `clearinghoused` data directory.

Under Linux, the data directory is normally located in `~/.config/clearinghoused` (when `clearinghoused` is installed normally, via the `setup.py` installer).

Under Windows, the data directory is normally located at `%APPDATA%\ClearingHouse\clearinghoused`. Examples of this are:

- `C:\Users\<your username>\AppData\Roaming\ClearingHouse\clearinghoused` (Windows 7/8/Server)
- `C:\Documents and Settings\<your username>\Application Data\ClearingHouse\clearinghoused` (Windows XP)

3.4.2 Editing the Config

`clearinghoused` can read its configuration data from a file. The build system uses this method to allow for automated startup of `clearinghoused`.

If using the Windows installer, a configuration file will be automatically created for you from data gathered via the installation wizard.

If not using the Windows installer, the `setup.py` script will create a basic `clearinghoused.conf` file for you that contains options that tell `clearinghoused` where and how to connect to your `viacoind` process. Here's an example of the default file created:

```
[Default]
viacoind-rpc-connect=localhost
viacoind-rpc-port=5222
viacoind-rpc-user=rpc
viacoind-rpc-password=rpcpw1234
rpc-user=my_api_user
rpc-password=my_api_password
```

After running the `setup.py` script to create this file, you'll probably need to edit it and tweak the settings to match your exact `viacoind` configuration (e.g. especially `rpc-password`). Note that the above config connects to `viacoind` on mainnet (port 5222).

Note that also, with the config above, it will set up `clearinghoused` to listen on localhost (127.0.0.1) on port 7300 (if on mainnet) or port 17300 (if on testnet) for API connections (these are the default ports, and can be changed by specifying the `rpc-host` and/or `rpc-port` parameters).

3.4.3 Viewing the Logs

By default, `clearinghoused` logs data to a file named `clearinghoused.log`, located within the `clearinghoused` data directory.

Under Linux, you can monitor these logs via a command like `tail -f ~/.config/clearinghoused/clearinghouse.log`.

Under Windows, you can use a tool like Notepad++ to view the log file, which will detect changes to the file and update if necessary.

3.4.4 Running clearinghoused on testnet

Here's the steps you'll need to take to set up an additional viacoind on testnet for `clearinghoused` testing. This assumes that you're already running `viacoind` (or `viacoin-qt`) on mainnet, and would like to set up a second instance for testnet:

Windows

First, find your current `viacoind` data directory, which is normally located at `%APPDATA%\Viacoin`. Examples of this are:

- `C:\Users\<your username>\AppData\Roaming\Viacoin` (Windows 7/8/Server)
- `C:\Documents and Settings\<your username>\Application Data\Viacoin` (Windows XP)

Alongside that directory (e.g. at the root of your AppDataRoaming dir), create another directory, name it something like `ViacoinTest`.

- `C:\Users\<your username>\AppData\Roaming\ViacoinTest` (Windows 7/8/Server)
- `C:\Documents and Settings\<your username>\Application Data\ViacoinTest` (Windows XP)

In this `ViacoinTest` directory, create a `viacoin.conf` file with the following contents:

```
rpcuser=rpc
rpcpassword=rpcpw1234
server=1
daemon=1
```

```
txindex=1
testnet=1
```

Now, make a shortcut to something like the following (assuming you installed to the default install directory from the .exe installer):

To run viacoin-qt: "C:\Program Files (x86)\Viacoin\viacoin-qt.exe" --datadir="C:\Users\<your username\AppData\Roaming\ViacoinTest" To run viacoind: "C:\Program Files (x86)\Viacoin\viacoind.exe" --datadir="C:\Users\<your username>\AppData\Roaming\ViacoinTest"

Note that you can run either. If you want the GUI, run viacoin-qt (which will also listen on the RPC interface). If you are comfortable using viacoind commands (or are using a server), just run viacoind.

Then, just launch that shortcut. (Or, if you are having problems, you can just open up a command window and try running that directly.)

Once launched, viacoind/viacoin-qt will be listening on testnet RPC API port 18332. You can just run clearinghoused with its --datadir parameter to point to a directory with its own clearinghoused.conf file that has the connection parameters to your testnet viacoin daemon that's now running.

This means, that like with viacoind, you may have two separate clearinghoused data directories, each with their own configuration file and database. The difference between the configuration files in each datadir will be that the one for your "testnet" clearinghoused will simply specify rpc-port=18332, while the one for your "mainnet" clearinghoused will specify rpc-port=8332.

Linux

Similar to the above, create a second viacoin data directory (maybe name it .viacoin-test, instead of .viacoin). Place it alongside your main .viacoin directory (e.g. under ~). In this directory, create a viacoin.conf file with the same contents as in the above Windows section.

Now, run viacoind or viacoin-qt, as such:

To run viacoin-qt: "viacoin-qt --datadir=~/.viacoin-test To run viacoind: viacoind --data-dir=~/.viacoin-test

For more information, see the Windows section above.

3.4.5 Next Steps

Once clearinghoused is installed and running, you can start running clearinghoused commands directly, or explore the (soon to exist) built-in API via the documentation at the main clearinghoused repository.

3.5 Setting up insight

As part of operating, clearinghoused may and clearblockd does require data that viacoind cannot currently provide. This includes things such as a BTC balance from an address not in the local viacoind wallet, and a list of unspent transaction outputs (UXTO) for an arbitrary address. In order to facilitate getting access to this information in a way that is robust and doesn't depend on a 3rd party site such as blockchain.info, we make use of insight, which is a free and open source server product made by BitPay which offers and API that supplements the information viacoind's API provides.

Both clearinghoused and clearblockd can communicate with insight. If you need to install insight, normally, you'll run it on the same computer as your instance of viacoind and clearinghoused runs on. However, you can also run an instance of it on a different server entirely.

3.5.1 On Windows

Prerequisites

You need to be running Windows Server, Windows 7, or Windows 8. You'll also need the following to install insight:

- Git for Windows. Download here and install. Use the default installer options (except, select *"Run Git from the Windows Command Prompt"* on the appropriate screen)
- Node.js. Go to the the node.js download page and grab the .msi installer for Windows (64-bit or 32-bit). Install it with the default options.
- Python 2.7.x – grab the 32-bit version or 64-bit version.
- For 64-bit systems you will also need the Windows 7 64-bit SDK. **NOTE** that if the install fails, try uninstalling any C++ 2010 x64&x86 Redistributables that you have installed first.
- Microsoft Visual Studio Express 2012.

Installing

Once these are installed, type Windows Key-R and enter `%comspec% /k ""C:\Program Files (x86)\Microsoft Visual Studio 11.0\Common7\Tools\VsDevCmd.bat""` to open a Windows Visual Studio 2012 development command prompt. Then, type the following:

```
cd C:\
git clone https://github.com/viacoin/insight-api.git
cd C:\insight-api
npm install
```

Next, locate your current viacoind data directory, which is normally located at %APPDATA%\Viacoin. Examples of this are:

- C:\Users\<your username>\AppData\Roaming\Viacoin (Windows 7/8/Server)
- C:\Documents and Settings\<your username>\Application Data\Viacoin (Windows XP)

Copy this down to a text file.

After this, type Windows Key-R and enter `rundll32.exe shell32.dll,Control_RunDLL sysdm.cpl,,3`, then press OK. This will launch the System Properties panel. Here, click on Environment Variables, and under User Variables, add the following:

- INSIGHT_NETWORK: Set this to livenet if your viacoind is running on mainnet. If on testnet, set this to testnet
- VIACOIND_DATADIR: Set this to your viacoind data dir you found in the step above
- VIACOIND_USER: Whatever your viacoind RPC user is set to (rpcuser= in the viacoin.conf in your viacoind data dir)
- VIACOIND_PASS: Whatever your viacoind RPC password is set to (rpcpassword= in the viacoin.conf in your viacoind data dir)

If you are running on a different host, or set of ports, you will also need to set `VIACOIND_HOST`, `VIACOIND_PORT`, and `VIACOIND_P2P_PORT` as appropriate.

Once done, click OK on both the Environment Variables and System Properties windows to save your changes and close them out.

Running

Open up a command window and run:

```
node C:\insight-api\insight.js
```

You can run it on startup by adding to your Startup program group in Windows, or using something like NSSM.

Next Steps

After running `insight`, it should start parsing the blockchain data from the `viacoind` data directory you specified (at `VIACOIND_DATADIR`).

You can do other things during this time, including normal use of `clearinghoused`. Please do note that `clearblockd` (or `clearinghoused` where you are querying the API on behalf of addresses not in the local `viacoind`'s wallet) will not provide reliable results until this indexing is fully completed.

3.5.2 On Ubuntu Linux

Open up a command window and run the following to install:

```
sudo apt-get update
sudo apt-get install git-core npm

#fix for https://github.com/TooTallNate/node-gyp/issues/363
GYP_DIR=`python -c 'import gyp, os; print os.path.dirname(gyp.__file__)'`
sudo mv ${GYP_DIR} ${GYP_DIR}_bkup

git clone https://github.com/viacoin/insight-api.git ~/insight-api && cd ~/insight-api
npm install
```

Running

To run insight, you'd do something like the following at a command prompt:

```
export INSIGHT_NETWORK=livenet
export VIACOIND_DATADIR=$USER_HOME/.viacoin
export VIACOIND_USER=`cat $USER_HOME/.viacoin/viacoin.conf | sed -n 's/.*rpcuser=\([^ \n]*\).*/\1/p'`
export VIACOIND_PASS=`cat $USER_HOME/.viacoin/viacoin.conf | sed -n 's/.*rpcpassword=\([^ \n]*\).*/\1
#VIACOIND_HOST -- specify to not use the default (localhost)
#VIACOIND_PORT -- specify to not use the default (5222)
#VIACOIND_P2P_PORT -- specify to not use the default (5223)
node ~/insight-api/insight.js
```

(Note that there is also an `insight.conf.template` and `insight-testnet.conf.template` upstart scripts that you can use in the `clearinghoused_build/dist/linux/init` directory. Simply take them, copy over to `/etc/init` (without the `.template` suffix to the file name) and modify `!RUN_AS_USER!` to be the username that you have installed insight as, then you can simply do something like:

```
sudo service insight start
```

Next steps

After running `insight`, it should start parsing the blockchain data from the `viacoind` data directory you specified (at `VIACOIND_DATADIR`).

You can do other things during this time, including normal use of `clearinghoused`. Please do note that `clearblockd` (or `clearinghoused` where you are querying the API on behalf of addresses not in the local `viacoind`'s wallet) will not provide reliable results until this indexing is fully completed.

3.6 Setting up a Clearblock Federated Node

3.6.1 Introduction

A Clearblock Federated Node is a self-contained server that runs the software necessary to support one or more "roles". Such roles may be:

- Clearwallet server
- Vending machine (future)
- Block explorer server (future)
- A plain old `counterpartyd` server

Each backend server runs multiple services (some required, and some optional, or based on the role chosen). As each server is self-contained, they can be combined by the client-side software to allow for high-availability/load balancing.

For instance, software such as Clearwallet may then utilize these backend servers in making API calls either sequentially (i.e. failover) or in parallel (i.e. consensus-based). For instance, with Clearwallet, when a user logs in, this list is shuffled so that in aggregate, user requests are effectively load-balanced across available servers. Indeed, by setting up multiple such (Clearblock) Federated Nodes, one can utilize a similar redundancy/reliability model in one's own 3rd party application that Clearwallet utilizes. Or, one can utilize a simplier configuration based on a single, stand-alone server.

This document describes how one can set up their own Clearblock Federated Node server(s). It is primarily intended for system administrators and developers.

3.6.2 Node Services/Components

clearinghoused (Required)

`clearinghoused` is the Clearinghouse reference client itself. It's responsibilities include parsing out Clearinghouse transactions from the Viacoin blockchain. It has a basic command line interface, and a reletively low-level API for getting information on specific transactions, or general state info.

clearblockd (Required, unless clearinghoused only)

The `clearblockd` daemon provides a more high-level API that layers on top of `clearinghoused`'s API, and includes extended information, such as market and price data, trade operations, asset history, and more. It is used extensively by Clearwallet itself, and is appropriate for use by applications that require additional API-based functionality beyond the scope of what `clearinghoused` provides.

`clearblockd` also provides a proxy-based interface to all `clearinghoused` API methods, via the `proxy_to_clearinghoused` API call.

insight (Optional)

`insight` allows for local querying of balance information and UTXOs for arbitrary addresses. This is a feature not available to `viacoind` itself. Alternatives to running `insight` on the server are using a service like `blockr.io`, which both `clearinghoused` and `clearblockd` support. For the most reliable service, we recommend that production servers (at least) run `insight` locally.

armory_utxsvr (Optional)

This service is used by `clearblockd` with Clearwallet, to allow for the creation of unsigned transaction ASCII text blocks, which may then be used with an Offline Armory configuration. This service requires Armory itself, which is automatically installed as part of the Federated Node setup procedure.

Clearwallet, etc.

The specific end-functionality, that builds off of the base services provided. For instance.

3.6.3 Federated Node Provisioning

Production

Here are the recommendations and/or requirements when setting up a production-grade Clearblock Federated Node:

Server Hardware/Network Recommendations:

- Xeon E3+ or similar-class processor
- 16GB+ RAM (ECC)
- Disk drives in RAID-1 (mirrored) configuration (SSD prefered)
- Hosted in a secure data center with physical security and access controls
- DDOS protection recommended if you will be offering your service to others

** Disk Space Requirements ** The exact disk space required will be dependent on what services are run on the node:

```
* Base System: **20GB** (to be safe)
* ``clearinghoused``, ``clearblockd`` databases: **~200MB**
* ``insight``: **~30GB** (mainnet), **~3GB** (testnet)
* ``armory_utxsvr``: **~25GB** (mainnet), **~3GB** (testnet)
```

Generally, we recommend building on a server with at least 120GB of available disk space.

Server Software:

- Ubuntu 14.04 64-bit required

Server Security:

The build script includes basic automated security hardening.

Before running this script, we strongly advise the following:

- SSH should run on a different port, with root access disabled

- Use ufw (software firewall) in addition to any hardware firewalls:
  - sudo ufw allow ssh #(or whatever your ssh port is, as '12345/tcp', in place of 'ssh')
  - sudo ufw allow http
  - sudo ufw allow https
  - sudo ufw enable
- Only one or two trusted individuals should have access to the box. All root access through `sudo`.
- Utilize 2FA (two-factor authentication) on SSH and any other services that require login. Duo is a good choice for this (and has great SSH integration).
- The system should have a proper hostname (e.g. clearblock.myorganization.org), and your DNS provider should be DDOS resistant
- If running multiple servers, consider other tweaks on a per-server basis to reduce homogeneity.
- Enable Ubuntu's automated security updates (our script will do this if you didn't)

Testing / Development

If you'd like to set up a Clearblock Federated Node system for testing and development, the requirements are minimal. Basically you need to set up a Virtual Machine (VM) instance (or hardware) at the Ubuntu version listed above, at least **2 GB** of memory, and enough disk space to cover the installation and use of the desired components.

3.6.4 Node Setup

Once the server is provisioned and set up as above, you will need to install all of the necessary software and dependencies. We have an installation script for this, that is fully automated **and installs ALL dependencies, including ''viacoind'' and ''insight''**:

```
cd && wget -qO setup_federated_node.py https://raw.github.com/Clearinghousexch/clearinghoused_build/r
sudo python3 setup_federated_node.py
```

Then just follow the on-screen prompts (choosing to build from *master* if you are building a production node, or from *develop* **only** if you are a developer or want access to bleeding edge code that is not fully tested).

Once done, start up `viacoind` daemon(s):

```
sudo service viacoind start
sudo service viacoind-testnet start

sudo tail -f ~xch/.viacoin/debug.log
```

That last command will give you information on the Viacoin blockchain download status. After the blockchain starts downloading, **if you've elected to install and use** `insight`, you can launch the `insight` daemon(s):

```
sudo service insight start
sudo service insight-testnet start

sudo tail -f ~xch/insight-api/insight.log
```

As well as `armory_utxsvr`, if you're using that (Clearwallet role only):

```
sudo service armory_utxsvr start
sudo service armory_utxsvr-testnet start
```

```
sudo tail -f ~xch/.config/armory/armory_utxsvr.log
```

And `clearinghoused` itself:

```
sudo service clearinghoused start
sudo service clearinghoused-testnet start

sudo tail -f ~xch/.config/clearinghoused/clearinghoused.log
```

Then, watching these log, wait for the insight sync (as well as the viacoind sync and clearinghoused syncs) to finish, which should take between 7 and 12 hours. After this is all done, reboot the box for the new services to start (which includes both `clearinghoused` and `clearblockd`).

`clearblockd`, after starting up must then sync to `clearinghoused`. It will do this automatically, and the process will take between 20 minutes to 1 hour most likely. You can check on the status of `clearblockd`'s sync using:

```
sudo tail -f ~xch/.config/clearblockd/clearblockd.log
```

Once it is fully synced up, you should be good to proceed. The next step is to simply open up a web browser, and go to the IP address/hostname of the server. You will then be presented to accept your self-signed SSL certificate, and after doing that, should see the web interface for the role you selected (e.g. Clearwallet login screen, if Clearwallet was chosen at node setup time). From this point, you can proceed testing the necessary functionality on your own system(s).

3.6.5 Getting a SSL Certificate

By default, the system is set up to use a self-signed SSL certificate. If you are hosting your services for others, you should get your own SSL certificate from your DNS registrar so that your users don't see a certificate warning when they visit your site. Once you have that certificate, create a nginx-compatible `.pem` file, and place that at `/etc/ssl/certs/clearblockd.pem`. Then, place your SSL private key at `/etc/ssl/private/clearblockd.key`.

After doing this, edit the `/etc/nginx/sites-enabled/clearblock.conf` file. Comment out the two development SSL certificate lines, and uncomment the production SSL cert lines, like so:

```
#SSL - For production use
ssl_certificate      /etc/ssl/certs/clearblockd.pem;
ssl_certificate_key  /etc/ssl/private/clearblockd.key;

#SSL - For development use
#ssl_certificate      /etc/ssl/certs/ssl-cert-snakeoil.pem;
#ssl_certificate_key  /etc/ssl/private/ssl-cert-snakeoil.key;
```

Then restart nginx:

```
sudo service nginx restart
```

3.6.6 Troubleshooting

If you experience issues with your Clearblock Federated Node, a good start is to check out the logs. Something like the following should work:

```
#mainnet
sudo tail -f ~xch/.config/clearinghoused/clearinghoused.log
sudo tail -f ~xch/.config/clearblockd/countewalletd.log
sudo tail -f ~xch/.config/clearinghoused/api.error.log
```

```
sudo tail -f ~xch/.config/clearblockd/api.error.log

#testnet
sudo tail -f ~xch/.config/clearinghoused-testnet/clearinghoused.log
sudo tail -f ~xch/.config/clearblockd-testnet/clearblockd.log
sudo tail -f ~xch/.config/clearinghoused-testnet/api.error.log
sudo tail -f ~xch/.config/clearblockd-testnet/api.error.log

#relevant nginx logs
sudo tail -f /var/log/nginx/clearblock.access.log
sudo tail -f /var/log/nginx/clearblock.error.log
```

These logs should hopefully provide some useful information that will help you further diagnose your issue. You can also keep tailing them (or use them with a log analysis tool like Splunk) to gain insight on the current status of `clearinghoused`/`clearblockd`.

Also, you can start up the daemons in the foreground, for easier debugging, using the following sets of commands:

```
#viacoind
sudo su -s /bin/bash -c 'viacoind -datadir=/home/xch/.viacoin' xchd
sudo su -s /bin/bash -c 'viacoind -datadir=/home/xch/.viacoin-testnet' xchd

#clearinghoused & clearblockd mainnet
sudo su -s /bin/bash -c 'clearinghoused --data-dir=/home/xch/.config/clearinghoused' xchd
sudo su -s /bin/bash -c 'clearblockd --data-dir=/home/xch/.config/clearblockd -v' xchd

#clearinghoused & clearblockd testnet
sudo su -s /bin/bash -c 'clearinghoused --data-dir=/home/xch/.config/clearinghoused-testnet --testnet
sudo su -s /bin/bash -c 'clearblockd --data-dir=/home/xch/.config/clearblockd-testnet --testnet -v' x
```

You can also run `viacoind` commands directly, e.g.:

```
#mainnet
sudo su - xchd -s /bin/bash -c "viacoind -datadir=/home/xch/.viacoin getinfo"

#testnet
sudo su - xchd -s /bin/bash -c "viacoind -datadir=/home/xch/.viacoin-testnet getinfo"
```

3.6.7 Monitoring the Server

To monitor the server, you can use a 3rd-party service such as [Pingdom](http://www.pingdom.com) or [StatusCake](http://statuscake.com). The federated node allows these (and any other monitoring service) to query the basic status of the server (e.g. the `nginx`, `clearblockd` and `clearinghoused` services) via making a HTTP GET call to one of the following URLs:

- `/_api/` (for mainnet)
- `/_t_api/` (for testnet)

If all services are up, a HTTP 200 response with the following data will be returned:

```
{"clearinghoused": "OK", "clearblockd_ver": "1.3.0", "clearinghoused_ver": "9.31.0", "clearblockd": "
"clearblockd_check_elapsed": 0.0039348602294921875, "clearinghoused_last_block": {
"block_hash": "000000000000000313c4708da5b676f453b41d566832f80809bc4cb141ab2cd", "block_index": 3112
"block_time": 1405638212}, "local_online_users": 7, "clearinghoused_check_elapsed": 0.003687143325805
"clearblockd_error": null, "clearinghoused_last_message_index": 91865}
```

Note the `"clearinghoused": "OK"` and `"clearblockd": "OK"` items.

If all services but `clearinghoused` are up, a HTTP 500 response with `"clearinghoused": "NOT OK"`, for instance.

If `clearblockd` is not working properly, `nginx` will return a HTTP 503 (Gateway unavailable) or 500 response.

If `nginx` is not working properly, either a HTTP 5xx response, or no response at all (i.e. timeout) will be returned.

3.6.8 Other Topics

User Configuration

Note that when you set up a federated node, the script creates two new users on the system: `xch` and `xchd`. (The `xch` user also has an `xch` group created for it as well.)

The script installs `clearinghoused`, `clearwallet`, etc into the home directory of the `xch` user. This user also owns all installed files. However, the daemons (i.e. `viacoind`, `insight`, `clearinghoused`, `clearblockd`, and `nginx`) are actually run as the `xchd` user, which has no write access to the files such as the `clearwallet` and `clearinghoused` source code files. The reason things are set up like this is so that even if there is a horrible bug in one of the products that allows for a RCE (or Remote Control Exploit), where the attacker would essentially be able to gain the ability to execute commands on the system as that user, two things should prevent this:

- The `xchd` user doesn't actually have write access to any sensitive files on the server (beyond the log and database files for `viacoind`, `clearinghoused`, etc.)
- The `xchd` user uses `/bin/false` as its shell, which prevents the attacker from gaining shell-level access

This setup is such to minimize (and hopefully eliminate) the impact from any kind of potential system-level exploit.

Easy Updating

To update the system with new code releases, you simply need to rerun the `setup_federated_node` script, like so:

```
cd ~xch/clearinghoused_build
sudo ./setup_federated_node.py
```

As prompted, you should be able to choose just to update from git ("G"), instead of to rebuild. However, you would choose the rebuild option if there were updates to the `clearinghoused_build` system files for the federated node itself (such as the `nginx` configuration, or the init scripts) that you wanted/needed to apply. Otherwise, update should be fine.

3.6.9 Clearwallet-Specific

Clearwallet Configuration File

Clearwallet can be configured via creating a small file called `clearwallet.conf.json` in the `clearwallet/` directory. This file will contain a valid JSON-formatted object, containing an a number of possible configuration properties. For example:

```
{
  "servers": [ "clearblock1.mydomain.com", "clearblock2.mydomain.com", "clearblock3.mydomain.com" ],
  "forceTestnet": true,
  "googleAnalyticsUA": "UA-48454783-2",
  "googleAnalyticsUA-testnet": "UA-48454783-4",
  "rollbarAccessToken": "39d23b5a512f4169c98fc922f0d1b121",
  "disabledFeatures": ["rps", "betting"],
```

```
  "restrictedAreas": {
    "pages/betting.html": ["US"],
    "pages/openbets.html": ["US"],
    "pages/matchedbets.html": ["US"],
    "pages/rps.html": ["US"],
    "dividend": ["US"]
  },
  "autoBTCEscrowServer": "btcescrow.clearwallet.co"
}
```

Here's a description of the possible fields:

Required fields:

- **servers**: Clearwallet should work out-of-the-box in a scenario where you have a single Counterblock Federated Node that both hosts the

static site content, as well as the backend Clearblock API services. However, Counterwallet can also be set up to work in MultiAPI mode, where it can query more than one server (to allow for both redundancy and load balancing). To do this, set this `servers` parameter as a list of multiple server URIs. Each URI can have a `http://` or `https://` prefix (we strongly recommend using HTTPS), and the strings must *not* end in a slash (just leave it off). If the server hostname does not start with `http://` or `https://`, then `https://` is assumed.

If you just want to use the current server (and don't have a multi-server setup), just specify this as ''[]'' (empty list).

Optional fields:

- **forceTestnet**: Set to true to always use testnet (not requiring 'testnet' in the FQDN, or the '?testnet=1' parameter in the URL.
- **googleAnalyticsUA / googleAnalyticsUA-testnet**: Set to enable google analytics for mainnet/testnet. You must have a google analytics account.
- **rollbarAccessToken**: Set to enable client-side error tracking via rollbar.com. Must have a rollbar account.
- **disabledFeatures**: Set to a list of zero or more features to disable in the UI. Possible features are: `betting`, `rps`, `dividend`, `exchange`, `leaderboard`, `portfolio`, `stats` and `history`. Normally this can just be `[]` (an empty list) to not disable anything.
- **restrictedAreas**: Set to an object containing a specific page path as the key (or "dividend" for dividend functionality), and a list of one or more ISO 2-letter country codes as the key value, to allow for country-level blacklisting of pages/features.
- **autoBTCEscrowServer**: The hostname to use for automatic BTC escrow services (where an external server will hold the BTC related to open orders selling BTC and make BTCpays from it automatically). If not specified, or left blank, then automatic BTC escrows will be disabled.

Once done, save this file and make sure it exists on all servers you are hosting Clearwallet static content on. Now, when you go to your Clearwallet site, the server will read in this file immediately after loading the page, and set the list of backend API hosts from it automatically.

Giving Op Chat Access

Clearwallet has its own built-in chatbox. Users in the chat box are able to have operator (op) status, which allows them to do things like ban or rename other users. Any op can give any other user op status via the `/op` command, typed into the chat window. However, manual database-level intervention is required to give op status to the first op in the system.

Doing this, however, is simple. Here's an example that gives `testuser1` op access. It needs to be issued at the command line for every node in the cluster:

```
#mainnet
mongo clearblockd
db.chat_handles.update({handle: "testuser1"}, {$set: {op: true}})

#testnet
mongo clearblockd_testnet
db.chat_handles.update({handle: "testuser1"}, {$set: {op: true}})
```

Clearwallet MultiAPI specifics

Note: By default, Clearblock Federated Nodes can also host Clearwallet content (this will change in the future). Regarding this, the Clearinghouse team itself operates the primary Clearwallet platform. However, as Clearwallet is open source software, it is possible to host your own site with Clearwallet site (for your personal use, or as an offering to others), or to even host your own Clearwallet servers to use with your own Clearinghouse wallet implementation. The Clearinghouse team supports this kind of activity (as long as the servers are secure), as it aids with increasing decentralization.

Also note that due to the nature of Clearwallet being a deterministic wallet, users using one Clearwallet platform (i.e. the official one, for instance) have the flexibility to start using a different Clearwallet platform instead at any time, and as funds (i.e. private keys) are not stored on the server in any fashion, they will be able to see their funds on either. (Note that the only thing that will not migrate are saved preferences, such as address aliases, the theme setting, etc.)

Clearwallet utilizes a sort of a "poor man's load balancing/failover" implementation called multiAPI (and implemented [here](https://github.com/Clearinghouse/clearwallet/blob/master/src/js/util.api.js)). multiAPI can operate in a number of fashions.

multiAPIFailover for Read API (''get_'') Operations

multiAPIFailover functionality is currently used for all read API operations. In this model, the first Federated Node on the shuffled list is called for the data, and if it returns an error or the request times out, the second one on the list is called, and so on. The result of the first server to successfully return are used.

Here, a "hacked" server could be modified to return bogus data. As (until being discovered) the server would be in the shuffled list, some clients may end up consulting it. However, as this functionality is essentially for data queries only, the worse case result is that a Clearwallet client is shown incorrect/modified data which leads to misinformed actions on the user's behalf. Moreover, the option always exists to move all read-queries to use multiAPIConsensus in the future should the need arise.

multiAPIConsensus for Action/Write (''create_'') Operations

Based on this multiAPI capability, the wallet itself consults more than one of these Federated Nodes via consensus especially for all `create_`-type operations. For example, if you send xch, clearinghoused on each server is still composing and sending back the unsigned raw transaction, but for data security, it compares the results returned from all servers, and will only sign and broadcast (both client-side) if all the results match). This is known as *multiAPIConsensus*.

The ultimate goal here is to have a federated net of semi-trusted backend servers not tied to any one country, provider, network or operator/admin. Through requiring consensus on the unsigned transactions returned for all `create_` operations, 'semi-trust' on a single server basis leads to an overall trustworthy network. Worst case, if backend server is hacked and owned (and the clearinghoused code modified), then you may get some invalid read results, but it won't be rewriting your xch send destination address, for example. The attackers would have to hack the code on every single server in the same exact way, undetected, to do that.

Moreover, the Clearwallet web client contains basic transaction validation code that will check that any unsigned Viacoin transaction returned from a Clearblock Federated Node contains expected inputs and outputs. This provides further protection against potential attacks.

multiAPIConsensus actually helps discover any potential "hacked" servers as well, since a returned consensus set with a divergent result will be rejected by the client, and thus trigger an examination of the root cause by the team.

multiAPINewest for Redundant storage

In the same way, these multiple servers are used to provide redundant storage of client-side preferences, to ensure we have no single point of failure. In the case of the stored preferences for instance, when retrieved on login, the data from all servers is taken in, and the newest result is used. This *multiAPINewest* functionality effectively makes a query across all available Federated Nodes, and chooses the newest result (based on a "last updated"-type timestamp).

Note that with this, a "hacked" server could be modified to always return the latest timestamp, so that its results were used. However, wallet preferences (and other data stored via this functionality) is non-sensitive, and thus user's funds would not be at risk before the hacked server could be discovered and removed.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search