# BTS Tools Documentation

***Release 0.4.3***

**Nicolas Wack**

November 12, 2016

Contents

The BTS Tools will help you build, run and monitor any graphene-based client (currently BitShares, Steem, Muse). There is still support for older clients (DACPlay, DPoS-PTS), however this functionality is deprecated and will be removed in a future version.

---

**Note:** these tools were originally developed for the BitShares network, and later expanded to support any graphene-based network. This means that everywhere you will see BitShares mentioned in this documentation, it should be understood as BitShares, Steem or Muse. Similarly, `bts` can be interchanged with `steem` and `muse`.

---

There are 2 tools currently provided:

- command line utility allowing to quickly build and run any graphene-based client
- web application allowing to monitor a running instance of the client and send an email or push notification on failure

If you like these tools, please vote for witness wackou on the Steem, BitShares and Muse networks. Thanks!

To get started, just type the following in a shell:

```
$ pip3 install bts_tools
```

If you're not familiar with installing python packages or if you run into problems during installation, please visit the Installing the tools section for more details.

With the tools installed, you can refer to each section of the documentation for more information about how a certain aspect of the tools work.

Otherwise, if you prefer a more hands-on approach to setting up a delegate from scratch, please head to the following section: How to setup a delegate - the easy tutorial

CHAPTER 1

# Documentation contents

## 1.1 Installing the tools

### 1.1.1 Installing dependencies for the tools

You will need some dependencies installed first before you can install the tools proper.

#### Linux

On Debian-derived OSes (Ubuntu, Mint, etc.), install with:

```
# apt-get install build-essential libyaml-dev python3-dev python3-pip
```

#### Mac OSX

On OSX, you can install the dependencies like that:

```
$ brew install libyaml
```

### 1.1.2 Installing the tools

If the dependencies for the tools are properly installed, you should be able to install the bts_tools package with the following command:

```
$ pip3 install bts_tools
```

**Note:** You might need to run this as root on linux systems

**Note:** In general, when dealing with python packages, it is good practice to learn how to work with virtualenvs, as they make installing python packages more self-contained and avoid potential conflicts with python packages installed by the system. They do require to invest some time learning about them first, so only do it if you feel like you can dedicate that time to it. It is very recommended to do so, though, as it can potentially save you a few headaches in the future.

Please refer to the Appendix: dealing with virtualenvs section for more details.

### 1.1.3 Installing dependencies for building the BitShares command-line client

Even though the tools are properly installed and functional, you also need some dependencies for being able to compile the BitShares client.

The reference documentation for building the BitShares client can be found on the Graphene wiki

#### Linux

On Debian-derived systems, install them with:

```
# apt-get install build-essential git cmake libssl-dev libdb++-dev libncurses5-dev \
                  libreadline-dev libffi-dev libboost-all-dev
```

For Steem, you will also need the qt5 libs:

```
# apt-get install build-essential git cmake libssl-dev libdb++-dev libncurses5-dev \
                  libreadline-dev libffi-dev libboost-all-dev qt5-default qttools5-dev-tools
```

#### Mac OSX

On OSX, you should install dependencies with brew instead of building your own, as the current libs in brew are recent enough to allow to compile the BitShares client. You will also need to force the install of `readline` system-wide and override OSX's native version, as it is antiquated.

```
$ brew install git cmake boost berkeley-db readline openssl
$ brew link --force readline
```

If you already had an "old" version of boost installed (< 1.55.0_2), please upgrade to a newer one:

```
$ brew upgrade boost
```

### 1.1.4 Installing dependencies for building the BitShares GUI client

To build the GUI client, you will need the same dependencies as for the command-line client, plus the following additional ones.

#### Linux

On Debian-derived systems, install them with:

```
# apt-get install qt5-default libqt5webkit5-dev qttools5-dev qttools5-dev-tools npm nodejs-legacy
```

#### Mac OSX

TODO

## 1.2 Command-line tools

just run the `bts` script with the command you want to execute:

```
$ bts -h
usage: bts [-h] [-r]
           {version,clean_homedir,clean,build,build_gui,run,run_gui,list,monitor,publish_slate}
           [environment] [args [args ...]]

following commands are available:
  - version          : show version of the tools
  - clean_homedir    : clean home directory. WARNING: this will delete your wallet!
  - clean            : clean build directory
  - build            : update and build bts client
  - build_gui        : update and build bts gui client
  - run              : run latest compiled bts client, or the one with the given hash or tag
  - run_gui          : run latest compiled bts gui client
  - list             : list installed bts client binaries
  - monitor          : run the monitoring web app
  - publish_slate    : publish the slate as described in the given file

Examples:
  $ bts build          # build the latest bts client by default
  $ bts build v0.4.27  # build specific version
  $ bts run
  $ bts run debug  # run the client inside gdb

  $ bts build pts-dev v2.0.1  # build a specific client/version
  $ bts run seed-test         # run environments are defined in the config.yaml file

  $ bts build_gui
  $ bts run_gui

  $ bts publish_slate                      # will show a sample slate
  $ bts publish_slate /path/to/slate.yaml  # publish the given slate


positional arguments:
  {version,clean_homedir,clean,build,build_gui,run,run_gui,list,monitor,publish_slate}
                        the command to run
  environment           the build/run environment (bts, pts, ...)
  args                  additional arguments to be passed to the given command

optional arguments:
  -h, --help            show this help message and exit
  -r, --norpc           run binary with RPC server deactivated

You should also look into ~/.bts_tools/config.yaml to tune it to your liking.
```

### 1.2.1 Building and running the BitShares command-line client

To build and run the command-line client, you can use the following two commands:

```
$ bts build
$ bts run
```

By default, `bts build` will build the latest version of the BitShares client (available on the master branch). If you want to build a specific version, you can do so by specifying either the tag, shortened tag (without `bts/` or `dvs/` prepended), or the git hash. For instance, all those are equivalent and will build the same binary:

```
$ bts build bts/0.5.3
$ bts build 0.5.3
$ bts build 8c908f8
```

After the command-line client is successfully built, it will be installed in the $BIN_DIR directory as defined in the `build_environments` section of the `config.yaml` file. The last built version will also be symlinked as `bitshares_client` in that directory, and this is the binary that a call to `bts run` will execute.

You can see a list of all binaries available by typing:

```
$ bts list
```

#### Passing additional arguments to "bts run"

You can pass additional arguments to "bts run" and the tools will forward them to the actual invocation of the bts client. This can be useful for options that you only use from time to time, eg: re-indexing the blockchain, or clearing the peer database. If they are args that start with a double dash (eg: –my-option), then you need to also prepend those with an isolated double dash, ie:

```
$ bts run -- --resync-blockchain --clear-peer-database
```

otherwise, the "–resync-blockchain" and "–clear-peer-database" would be considered to be an option for the bts script, and not an argument that should be forwarded.

### 1.2.2 Building and running the BitShares GUI client

To build and run the GUI client, the procedure is very similar to the one for the command-line client:

```
$ bts build_gui
$ bts run_gui
```

There is one major difference though: the GUI client will not be installed anywhere and will always be run from the build directory. This is done so in order to be as little intrusive as possible (ie: not mess with a wallet you already have installed) and as install procedures are not as clear-cut as for the command-line client.

### 1.2.3 Publishing a slate

Once you have your delegate up and running, you might want to publish a slate of recommended delegates. To do so, you will need to have your wallet unlocked and have a file with the following format somewhere, say in `/tmp/slate.yaml`:

```
delegate: publishing_delegate_name
paying: paying_account  # optional, defaults to publishing delegate
slate:
 - delegate_1
 - delegate_2
 - ...
 - delegate_N
```

You can then publish your slate like so:

```
$ bts publish_slate /tmp/slate.yaml
```

## 1.3 Monitoring web app

### 1.3.1 Launch the monitoring web app locally

The main entry point to the monitoring app is the `~/.bts_tools/config.yaml` file. You should edit it first and set the values to correspond to your delegate's configuration. See the Format of the config.yaml file page for details.

If this file doesn't exist yet, run the tools once (for instance: `bts -h`) and it will create a default one.

To run the debug/development monitoring web app, just do the following:

```
$ bts monitor
```

and it will launch on `localhost:5000`.

### 1.3.2 Setting up on a production server

For production deployments, it is recommended to put it behind a WSGI server, in which case the entry point is `bts_tools.wsgi:application`.

Do not forget to edit the `~/.bts_tools/config.yaml` file to configure it to suit your needs.

#### Example

We will run the monitoring tools using nginx as frontend. Install using:

```
# apt-get install nginx uwsgi uwsgi-plugin-python3
```

The tools will have to be run from a virtualenv, so let's create it:

```
$ mkvirtualenv -p `which python3` bts_tools
$ pip3 install bts_tools
```

Edit the following configuration files:

`/etc/uwsgi/apps-available/bts_tools.ini` (need symlink to `/etc/uwsgi/apps-enabled/bts_tools.ini`)

```
[uwsgi]
uid = myuser
gid = mygroup
chmod-socket = 666
plugin = python34
virtualenv = /home/myuser/.virtualenvs/bts_tools
enable-threads = true
lazy-apps = true
workers = 1
module = bts_tools.wsgi
callable: application
```

---

**Note:** The important, non-obvious, fields to set in the uwsgi config file are the following:

- set `enable-threads = true`, otherwise you won't get the monitoring thread properly launched
- set `lazy-apps = true`, otherwise the stats object will not get properly shared between the master process and the workers, and you won't get any monitoring data
- set `workers = 1`, otherwise you will get multiple instances of the worker thread active at the same time

The `virtualenv` field also needs to be setup if you installed the tools inside one, otherwise you can leave it out.

`/etc/nginx/sites-available/default` (need symlink to `/etc/nginx/sites-enabled/default`)

```
server {
        listen 80;
        server_name myserver.com;
        charset utf-8;
        location / { try_files $uri @bts_tools; }
        location @bts_tools {
                # optional password protection
                #auth_basic "Restricted";
                #auth_basic_user_file /home/myuser/.htpasswd;
                include uwsgi_params;
                uwsgi_pass unix:/run/uwsgi/app/bts_tools/socket;
        }
}
```

After having changed those files, you should:

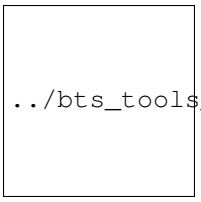
```
# service uwsgi restart
# service nginx restart
```

### 1.3.3 Screenshots

Monitoring the status of your running bts client binary:


You can host multiple delegates accounts in the same wallet, and check feed info:


Monitoring multiple instances (ie: running on different hosts) at the same time, to have an overview while running backup nodes and re-compiling your main node:




## 1.4 Working with other DPOS clients

The BTS Tools have originally been developed for managing BitShares delegates, but due to the similarity with other DPOS clients they are able to handle all the following blockchains:

- 'bts': BitShares
- 'dvs': DevShares
- 'pts': PTS-DPOS
- 'pls': DAC PLAY
- 'bts2': BitShares 2.0 (aka Graphene)
- 'muse': Muse

Support for the other DPOS clients is built-in directly in the `bts` cmdline tool, and you only need to specify the corresponding build or run environment, e.g.:

```
$ bts build_gui dvs  # builds the DevShares GUI client
$ bts build pts      # builds the PTS command-line client
```

As a convenience feature, the following aliases to the `bts` tool are provided:

- `dvs`: builds using the DevShares environment by default
- `pts`: builds using the PTS environment by default
- `pls`: builds using the DAC PLAY environment by default
- `bts2`: builds using the BitShares 2.0 environment by default
- `muse`: builds using the Muse environment by default

This means that:

```
$ pts build
```

and

```
$ bts build pts
```

are exactly equivalent.

## 1.5 Working with other types of nodes

Originally, the focus of the tools has been on maintaining delegate nodes on the BitShares network, but they now support more types of specialized nodes.

Concretely, you can now manage the following types of nodes: - delegate - seed - backbone

Seed nodes are public, do not need an open wallet to run and usually have a high number of network connections.

Backbone nodes are public, do not need an open wallet to run, and do not perform peer exchange (in order to hide IP addresses of the delegates connected to them). They also try to maintain at all time an open connection to all other backbone nodes in order to have the backbone being a fully-connected graph of its nodes.

Command-line arguments and monitoring plugins are automatically defined depending on the type of node (seed or backbone), so config should be straightforward. For delegate nodes, you still have to specify whether you want a 'delegate' or 'watcher_delegate' in the monitoring section of the node.

## 1.6 How to setup a delegate - the easy tutorial

This guide will try to show you how to setup all the infrastructure needed to build, run and monitor a delegate easily and effortlessly. We will start from scratch, and end up with a fully functional delegate client running and being monitored for crashes and missed blocks.

Note also that this guide will not only show you the mininum number of steps required to get it working once, but it will try to guide you into using best practices and useful tools that make maintenance of the delegate over time a seamless experience. (For the curious, that means using virtualenvs, tmux, etc... If you have no idea what these are, don't worry, we'll get to it)

Once everything it setup properly, building the latest version of the client, running it, and launching the monitoring webapp that publishes feeds and sends you notifications is just a matter of:

```
$ bts build
$ bts run
$ bts monitor
```

In details, these are the following steps that this guide will cover:

### 1.6.1 Setup the base OS and build environment

#### Linux

We will do the install on Debian Jessie. It should be very similar on Ubuntu, however some packages might have slightly different names. Adapting it for Ubuntu is left as an exercise to the reader.

Note about the text editor: there are countless wars about which editor is the best, but ultimately it is up to you to pick the one you like best, so no details will be provided when needing to edit text files, it will just be mentioned that you need to do it.

##### Install base up-to-date OS

Installing the base OS will depend on your VPS provider, so check their documentation for that. The tutorial will use Debian Jessie as base distro, so you should install it directly whenever possible. You can download the most recent release of the Debian Jessie installer here, preferably the netinst version.

##### Install required dependencies

The first step is to install the dependencies necessary for installing the tools and for compiling the BitShares client (still as root):

```
# apt-get install build-essential git cmake libssl-dev libdb++-dev libncurses5-dev libreadline-dev \
                  python3-dev python3-pip libyaml-dev libboost-all-dev ntp
```

Note that we also install the `ntp` client here, this is needed to keep your server's time correctly adjusted, which is a requirement for a witness wanting to sign blocks (given that the time slot for a block is 3 seconds, you need to be pretty much spot on when it's your turn to sign a block)

#### Mac OSX

It is possible, and hence recommended, to build the BitShares client using only libraries pulled out of homebrew, as you don't have to compile and maintain dependencies yourself.

```
brew install git cmake boost berkeley-db readline openssl libyaml
brew link --force readline
```

If you already had an "old" version of boost installed, please upgrade to a newer one:

```
$ brew upgrade boost
```

Also make sure that you are running a constantly up-to-date version of cmake, you might encounter weird configuration errors otherwise (ie: cmake not finding properly installed dependencies, etc.)

```
$ brew upgrade cmake
```

##### Choosing the correct version of XCode

**Note:** This happens on the author's computer and may or may not happen to you, so take with a grain of salt. YMMV.

It appears that the most recent version of XCode cannot build the BitShares client, at least on Mavericks (OSX 10.9), because the API generator segfaults. XCode 5.1 does work properly, though, so the recommended way is to:

- download XCode 5.1.1 from the Apple developer center
- install it on your computer, say in `/Applications/Xcode-5.1.1` (you can use it in parallel with the latest version if you like)
- tell your system to use this version on the command-line by running:

```
$ sudo xcode-select -s /Applications/Xcode-5.1.1.app/Contents/Developer
```

- you can now proceed normally!

### 1.6.2 Install the bts_tools package

The first step in your quest for being a delegate is to install the `bts_tools` python package. Make sure you have installed the dependencies as described in the previous section, and run (as root [1]):

```
# pip3 install bts_tools
```

That's it, the tools are installed and you should now be setup for building the BitShares client! To see all that the tools provide, try running:

```
$ bts -h
```

[1] This installs the tools system-wide, and is the simplest way of doing it. However, if you have time to invest in learning about them, it is highly recommended to look into python virtualenvs and how to deal with them. You can find a quick overview about them here: Appendix: dealing with virtualenvs

which should show the online help for the tools. You should definitely get accustomed to the list of commands that are provided.

### 1.6.3 Build and run the BitShares client

To build the BitShares client, just type the following:

```
$ bts build
```

This will take some time, but you should end up with a BitShares binary ready to be executed. To make sure this worked, and see all the versions available on your system, type:

```
$ bts list
```

This should also show you the default version of the client that will be run.

To run it, you just need to:

```
$ bts --norpc run
```

The first time you run it, you need to pass it the `--norpc` param (or `-r`) in order to not launch the RPC server, as it is not configured yet. After the first run, this will have created the `~/.BitShares` directory (`~/Library/Application Support/BitShares` on OSX) and you should go there, edit the `config.json` file, and fill in the user and password for the RPC connection. Next time you will only need to:

```
$ bts run
```

to launch the client.

At this point, you want to create a wallet, an account and register it as delegate. Please refer to the BitShares wiki for instructions.

#### Pro Tip: running the client in tmux

Running the client inside your shell after having logged in to your VPS is what you want to do in order to be able to run it 24/7. However, you want the client to still keep running even after logging out. The solution to this problem is to use what is called a terminal multiplexer, such as screen or tmux. Don't worry about the complicated name, what a terminal multiplexer allows you to do is to run a shell to which you can "attach" and "detach" at will, and which will keep running in the background. When you re-attach to it, you will see your screen as if you had never disconnected.

Here we will use `tmux`, but the process with `screen` is extremely similar (although a few keyboard shortcuts change).

The first thing to do is to launch `tmux` itself, simply by running the following in your shell:

```
$ tmux
```

You should now see the same shell prompt, but a status bar should have appeared at the bottom of your screen, meaning you are now running "inside" tmux.

---

**Note:** The keyboard shortcuts are somewhat arcane, but this is the bare minimum you have to remember:

when outside of tmux:

- `tmux` : create a new tmux session
- `tmux attach` : re-attach to a running session

when inside of tmux:

- `ctrl+b d` : detach the session - do this before disconnecting from your server
- `ctrl+b [` : enter "scrolling mode" - you can scroll back the screen (normal arrows and sliders from your terminal application don't work with tmux...) Use `q` to quit this mode

---

So let's try attaching/detaching our tmux session now: as you just ran 'tmux', you are now inside it type `ctrl-b d`, and you should now be back to your shell before launching it

```
$ tmux attach  # this re-attaches to our session
$ bts run      # we run the bitshares client inside tmux
```

type `ctrl-b d`, you are now outside of tmux, and doesn't see anything from the bts client

```
$ tmux attach  # this re-attaches your session, and you should see the bts client still in action
```

To get more accustomed to tmux, it is recommended to find tutorials on the web, this one for instance seems to do a good job of showing the power of tmux while not being too scary...

### 1.6.4 Run the monitoring webapp

This is the good part :)

Now that you know how to build and run the delegate client, let's look into setting up the monitoring of the client. Say you want to monitor the delegate called `mydelegate`. The possible events that we can monitor and the actions that we can take also are the following:

- monitor when the client comes online / goes offline (crash), and send notifications when that happens (email or iOS)
- monitor when the client loses network connection
- monitor when the client misses a block
- publish feeds
- ensure that version number is the same as the one published on the blockchain, and if not, publish a new version

These can be set independently for each delegate that you monitor, and need to be specified in the `nodes` attribute of the `config.yaml` file.

A node specifies the following properties:

- the type of client that it runs (BitShares, PTS, ...) In our case here, this will be `"bts"`.
- the type of the node will be `"delegate"` (could be `"seed"` too, but we're setting up a delegate here).
- the name here will be set to `"mydelegate"` (replace with your delegate's name)
- the `"monitoring"` variable will contain: `[version, feeds, email]`. As online status, network connections and missed blocks are always monitored for a delegate node, you only need to specify whether you want to receive the notifications by email or boxcar, in this case here we want `email`. You will also need to configure the `email` section in the `config.yaml` in order to be able to send them out.

This gives the following:

```
nodes:
    -
        client: bts
        type: delegate
        name: mydelegate
        monitoring: [version, feeds, email]
```

Once you have properly edited the `~/.bts_tools/config.yaml` file, it is just a matter of running:

```
$ bts monitor
```

and you can now go to http://localhost:5000/ in order to see it.

#### Install the tools behind Nginx + uWSGI

Although outside of the scope of this tutorial, if you want to set up your delegate properly and have the web interface accessible from the outside, it is recommended to put it behind an Nginx server. Please refer here for an example: *Setting up on a production server* (skip the part about the virtualenv if it doesn't apply to you)

Note that there are some choices of software that are quite opinionated in this guide, however this should not be considered as the ony way to do things, but rather just a way that the author thinks makes sense and found convenient for himself.

## 1.7 Format of the config.yaml file

The `config.yaml` file contains the configuration about the delegates. It contains the following main sections:

### 1.7.1 Build environments

These are the types of clients that you can build. `bts`, `dvs`, `pts`, etc. Mostly the default should work and you shouldn't change those values.

### 1.7.2 Run environments

These represent the clients that you can run. The default should mostly work, here but you may want to edit them if you want to have multiple instances of the client running, for instance a delegate node and a seed node.

**The clients define the following variables:**

- **type**: the type of build you want to run. Needs to be a valid build env (ie: bts, pts, ...)
- **debug**: *[optional]* set to true to run client in gdb (only available in linux for now)
- **data_dir**: *[optional]* the data dir (blockchain, wallet, etc.) of the bts client. If not specified, uses the standard location of the client
- **run_args**: *[optional]* any additional flags you want to pass to the cmdline invocation of the client

To run a specific client, type:

```
$ bts run client_name
```

If you don't specify a client on the command-line (ie: `bts run`), the tools will run the client using the `bts` run environment by default.

#### Example

```
run_environments:
    seed-bts:
        type: bts
        debug: false
        run_args: ['--p2p-port', '1778', '--clear-peer-database']
```

This allows you to run a seed node on port 1778, and clear its peer database each time you run it.

### 1.7.3 Backbone nodes

This section contains the list of `host:ip` for the clients that constitute the backbone nodes. Defaults should work ok, but you can configure them yourselves if you want to try alternate backbone nodes.

#### Example

```
backbone:
    - backbone01.digitalgaia.io:1777
    - backbone02.digitalgaia.io:1777
    - backbone03.digitalgaia.io:1777
```

### 1.7.4 Nodes list

This is the part that you should edit to configure it to your needs.

In the `nodes` variable you should specify the list of delegate accounts that you want the tools to monitor.

For each node, you need to specify the following attributes:

- `client`: the client being monitored. This needs to be a valid run environment, and will allow to fetch the RPC parameters automatically. If you don't specify it, you need to fill in the `rpc_host`, `rpc_port`, `rpc_user`, `rpc_password` and `venv_path` instead. You will need ssh access if it is a remote host. **TODO:** expand doc about ssh
- `type`: the type of the node being monitored. Either `seed`, `delegate` or `backbone`
- `name`: the name of the node (delegate account name)
- `monitoring`: the list of monitoring plugins that should be run on this node
- `notification`: the type of notification to be sent for events of this node

Nodes can be of 3 types:

- delegate nodes
- seed nodes
- backbone nodes

The choice of node type will tune a bit the interface and enable/disable functionality, such as feed publishing for delegates but not for seed nodes or backbone nodes

#### Monitoring plugins

For each node you can specify which type of monitoring you want:

- `seed`: will set the number of desired/max connections as specified in the `monitoring` config section

- `backbone`: will check that the backbone node runs as intended
- `feeds`: check price feeds, and optionally publish them if the node is a delegate
- `version`: check if version number matches published one, publishes it otherwise
- `missed`: check for missed blocks for a delegate
- `network_connections`: check that number of active connections to the network is higher than a threshold
- `payroll`: periodically distribute delegate pay amongst the configured accounts in the `monitoring` section.
- `wallet_state`: check when wallet is opened/closed and locked/unlocked
- `fork`: tries to detect whether client is being moved to a minority fork
- `voted_in`: check when a delegate is voted in/out

You can also use the following special monitoring plugins as wildcards:

- `delegate`: use for monitoring a full-fledged delegate. It will activate the following plugins: `missed`, `network_connections`, `voted_in`, `wallet_state`, `fork`, `version`, `feeds`
- `watcher_delegate`: use for monitoring a delegate without publishing any information (feeds or version). It will activate the following plugins: `missed`, `network_connections`, `voted_in`, `wallet_state`, `fork`

### Notification plugins

You should also configure which type of notification you want to receive:

- `email`: send an email notification when client crashes or loses network connections
- `boxcar`: send an iOS push notification to the Boxcar app when client crashes or loses network connections

### Example

```
nodes:
    -
        client: bts
        type: delegate          # delegate node type: run a single delegate account
        name: delegate1         # the name of the delegate. This needs to be an existing account
        monitoring: [delegate]  # activate default monitoring plugins for delegate
        notification: [email]
    -
        type: seed        # seed node type: no need for open wallet, high number of connections
        client: seed-bts  # need to be a valid run environment
        name: seed01      # the name for this seed node. This is just for you, it serves no other pur
        # you can specify the rpc connection params. This will override the values
        # from the data directory
        rpc_port: 5678
        rpc_user: username
        rpc_password: secret-password
    -
        type: delegate   # remote delegate node type: access to a remote node's delegate info. You ne
        name: delegate3  # the name for this remote node. This is just for you, it serves no other pu
        venv_path: ~/.virtualenvs/bts_tools # virtualenv dir in which the bts tools are installed on
        rpc_host: user@myhost  # hostname. Anything you can pass to "ssh" you can put here (eg: your
        rpc_port: 5678
        rpc_user: username
        rpc_password: secret-password
```

### 1.7.5 Monitoring plugins configuration

In the `monitoring` section comes the configuration of the various monitoring plugins. Configure to your taste!

### 1.7.6 Notifications

In the `notification` section, you will be able to configure how notifications will be sent to you. There are 2 ways of being notified: `email` and `boxcar` (iOS push notifications).

See default provided `config.yaml` file for the fields you need to configure.

## 1.8 Appendix: dealing with virtualenvs

When dealing with python packages, it is possible to install them as root and make them available for the entire system. This is not always recommended as it can sometimes cause conflicts with the packages installed by your OS.

In the python world a solution to deal with that problem has emerged and allows to create sandboxes in which to install python packages, so that they do not interfere with those of the system. These sandboxes are called virtualenvs, short for "virtual environments".

Although very powerful, the usage of the bare virtualenv functionality can sometimes be cumbersome, so it is very recommended to use another project instead that gives you an easier API to work with: virtualenvwrapper

The main commands that virtualenvwrapper provides are the following:

- `mkvirtualenv` creates a new virtualenv (`rmvirtualenv` deletes it)
- `workon` allows to "activate" a virtualenv, meaning all packages that you install after that will be installed inside this virtualenv, and they will take precedence over those of the system (basically, they will be active). (use `deactivate` to stop using it)

### 1.8.1 Example

If you want to create a new virtualenv with python3 being used as interpreter of choice, you would run the following:

```
$ mkvirtualenv -p `which python3` bts_tools
```

Note that after creating it, the virtualenv is already active, so you don't need to call `workon bts_tools` right after creating it. You will have to do it next time you reboot or open a shell, though.

If you then run the following:

```
$ pip install bts_tools
```

it will install the tools inside the virtualenv, and won't interfere with your system.

## 1.9 Appendix: tips and tricks

This is a collection of various tips and tricks that didn't fit in any particular section, but that you probably want to know, or at least want to know that they exist :)

### 1.9.1 Use clang as a compiler on linux instead of gcc

When running on debian/ubuntu, the best solution (the "native" one) is to change your default compiler, like that:

```
sudo apt-get install clang
sudo update-alternatives --config c++
```

Alternatively, you can set the following in your `config.yaml` file:

```
CONFIGURE_OPTS = ['CC=/usr/bin/clang', 'CXX=/usr/bin/clang++']
```

### 1.9.2 Compiling Steem without dependency on Qt5

Add the following to your `config.yaml` file:

```
build_environments:
    steem:
        cmake_args: ['-DENABLE_CONTENT_PATCHING=OFF', '-DLOW_MEMORY_NODE=ON']
```