
Contributing to BigchainDB

BigchainDB Contributors

Mar 14, 2019

Contents

1	Contents	3
1.1	Ways to Contribute	3
1.2	Developer Setup, Coding & Contribution Process	5
1.3	Policies	21

There are many ways you can contribute to BigchainDB. It includes several sub-projects.

- [BigchainDB Server](#)
- [BigchainDB Python Driver](#)
- [BigchainDB JavaScript Driver](#)
- [BigchainDB Java Driver](#)
- [cryptoconditions](#) (a Python package by us)
- [py-abc](#) (a Python package we use)
- [BigchainDB Enhancement Proposals \(BEPs\)](#)

1.1 Ways to Contribute

1.1.1 Report a Bug

To report a bug, go to the relevant GitHub repository, click on the **Issues** tab, click on the **New issue** button, and read the instructions.

1.1.2 Write an Issue

To write an issue, go to the relevant GitHub repository, click on the **Issues** tab, click on the **New issue** button, and read the instructions.

1.1.3 Make a Feature Request or Proposal

To make a feature request or proposal, write a BigchainDB Enhancement Proposal (BEP).

1.1.4 Write a BigchainDB Enhancement Proposal (BEP)

If you have an idea for a new feature or enhancement, and you want some feedback before you write a full BigchainDB Enhancement Proposal (BEP), then feel free to:

- ask in the [bigchaindb/bigchaindb Gitter chat room](#) or
- open a new issue in the [bigchaindb/BEPs repo](#) and give it the label **BEP idea**.

If you want to discuss an existing BEP, then open a new issue in the [bigchaindb/BEPs repo](#) and give it the label **discuss existing BEP**.

Steps to Write a New BEP

1. Look at the structure of existing BEPs in the [bigchaindb/BEPs repo](#). Note the section headings. [BEP-2](#) (our variant of the consensus-oriented specification system [COSS]) says more about the expected structure and process.
2. Write a first draft of your BEP. It doesn't have to be long or perfect.
3. Push your BEP draft to the [bigchaindb/BEPs repo](#) and make a pull request. [BEP-1](#) (our variant of C4) outlines the process we use to handle all pull requests. In particular, we try to merge all pull requests quickly.
4. Your BEP can be revised by pushing more pull requests.

1.1.5 Write Docs

If you're writing code, you should also update any related docs. However, you might want to write docs only, such as:

- General explainers
- Tutorials
- Courses
- Code explanations
- How BigchainDB relates to other blockchain things
- News from recent events

You can certainly do that!

- The docs for BigchainDB Server live under `bigchaindb/docs/` in the `bigchaindb/bigchaindb` repo.
- There are docs for the Python driver under `bigchaindb-driver/docs/` in the `bigchaindb/bigchaindb-driver` repo.
- There are docs for the JavaScript driver under `bigchaindb/js-bigchaindb-driver` in the `bigchaindb/js-bigchaindb-driver` repo.
- The source code for the BigchainDB website is in a private repo, but we can give you access if you ask.

The [BigchainDB Transactions Specs](#) (one for each spec version) are in the `bigchaindb/BEPs` repo.

You can write the docs using Markdown (MD) or RestructuredText (RST). Sphinx can understand both. RST is more powerful.

ReadTheDocs will automatically rebuild the docs whenever a commit happens on the `master` branch, or on one of the other branches that it is monitoring.

1.1.6 Answer Questions

People ask questions about BigchainDB in the following places:

- Gitter
 - `bigchaindb/bigchaindb`
 - `bigchaindb/js-bigchaindb-driver`
- Twitter
- Stack Overflow “bigchaindb”

Feel free to hang out and answer some questions. People will be thankful.

1.2 Developer Setup, Coding & Contribution Process

1.2.1 Write Code

Know What You Want to Write Code to Do

Do you want to write code to resolve an open issue (bug)? Which one?

Do you want to implement a BigchainDB Enhancement Proposal (BEP)? Which one?

You should know why you want to write code before you go any farther.

Refresh Yourself about the C4 Process

C4 is the Collective Code Construction Contract. It's quite short: re-reading it will only take a few minutes.

Set Up Your Local Machine. Here's How.

- Make sure you have Git installed.
- Get a text editor. Internally, we like:
 - Vim
 - PyCharm
 - Visual Studio Code
 - Atom
 - GNU Emacs (Trent is crazy)
 - GNU nano (Troy has lost his mind)
- If you plan to do JavaScript coding, get the latest JavaScript stuff (npm etc.).
- If you plan to do Python coding, get the latest Python, and get the latest pip.

Warning: Don't use apt or apt-get to get the latest pip. It won't work properly. Use `get-pip.py` from the pip website.

- Use the latest pip to get the latest virtualenv:

```
$ pip install virtualenv
```

- Create a Python Virtual Environment (virtualenv) for doing BigchainDB Server development. There are many ways to do that. Google around and pick one. An old-fashioned but fine way is:

```
$ virtualenv -p $(which python3.6) NEW_ENV_NAME
$ . NEW_ENV_NAME/bin/activate
```

Be sure to use Python 3.6.x as the Python version for your virtualenv. The virtualenv creation process will actually get the latest pip, wheel and setuptools and put them inside the new virtualenv.

Start Writing Code

Use the [Git Fork and Pull Request Workflow](#). Tip: You could print that page for reference.

Your Python code should follow [our Python Style Guide](#). Similarly for JavaScript.

Make sure `pre-commit` actually checks commits. Do:

```
$ pip install pre-commit # might not do anything if it is already installed,  
→ which is okay  
$ pre-commit install
```

That will load the `pre-commit` settings in the file `.pre-commit-config.yaml`. Now every time you do `git commit <stuff>`, `pre-commit` will run all those checks.

To install BigchainDB Server from the local code, and to keep it up to date with the latest code in there, use:

```
$ pip install -e .[dev]
```

The `-e` tells it to use the latest code. The `.` means use the current directory, which should be the one containing `setup.py`. The `[dev]` tells it to install some extra Python packages. Which ones? Open `setup.py` and look for `dev` in the `extras_require` section.

Remember to Write Tests

We like to test everything, if possible. Unit tests and also integration tests. We use the `pytest` framework to write Python tests. Read all about it.

Most tests are in the `tests/` folder. Take a look around.

Running a Local Node/Network for Dev and Test

This is tricky and personal. Different people do it different ways. We documented some of those on separate pages:

- [Dev node setup and running all tests with Docker Compose](#)
- [Dev node setup and running all tests as processes](#)
- [Dev network setup with stack.sh](#)
- [Dev network setup with Ansible](#)
- More to come?

Create the PR on GitHub

Git push your branch to GitHub so as to create a pull request against the branch where the code you want to change *lives*.

Travis and Codecov will run and might complain. Look into the complaints and fix them before merging. Travis gets its configuration and setup from the files:

- Some environment variables, if they are used. See <https://docs.travis-ci.com/user/environment-variables/>
- `.travis.yml`
- `tox.ini` - What is tox? See tox.readthedocs.io
- `.ci/` (as in Travis CI = Continuous Integration)

```
- travis-after-success.sh
- travis-before-install.sh
- travis-before-script.sh
- travis-install.sh
- travis_script.sh
```

Read about the [Travis CI build lifecycle](#) to understand when those scripts run and what they do. You can have even more scripts!

Codecov gets its configuration from the file `codecov.yml` which is also documented at docs.codecov.io. Codecov might also use `setup.cfg`.

First-Time Pull Requests from External Users

First-time pull requests from external users who haven't contributed before will get blocked by the requirement to agree to the BigchainDB Contributor License Agreement (CLA). It doesn't take long to agree to it. Go to <https://www.bigchaindb.com/cla/> and:

- Select the CLA you want to agree to (for individuals or for a whole company)
- Fill in the form and submit it
- Wait for an email from us with the next step. There is only one: copying a special block of text to GitHub.

Merge!

Ideally, we like your PR and merge it right away. We don't want to keep you waiting.

If we want to make changes, we'll do them in a follow-up PR.

You are awesome. Do you want a job? [Apply!](#) Berlin is great. If you got this far, we'd be happy to consider you joining our team. Look at these [Unsplash photos of Berlin](#). So nice.

1.2.2 Notes on Running a Local Dev Node with Docker Compose

Setting up a single node development environment with `docker-compose`

Using the BigchainDB 2.0 developer toolbox

We grouped all useful commands under a simple `Makefile`.

Run a BigchainDB node in the foreground:

```
$ make run
```

There are also other commands you can execute:

- `make start`: Run BigchainDB from source and daemonize it (stop it with `make stop`).
- `make stop`: Stop BigchainDB.
- `make logs`: Attach to the logs.
- `make test`: Run all unit and acceptance tests.

Contributing to BigchainDB

- `make test-unit-watch`: Run all tests and wait. Every time you change code, tests will be run again.
- `make cov`: Check code coverage and open the result in the browser.
- `make doc`: Generate HTML documentation and open it in the browser.
- `make clean`: Remove all build, test, coverage and Python artifacts.
- `make reset`: Stop and REMOVE all containers. WARNING: you will LOSE all data stored in BigchainDB.

Using `docker-compose` directly

The BigchainDB Makefile is a wrapper around some `docker-compose` commands we use frequently. If you need a finer granularity to manage the containers, you can still use `docker-compose` directly. This part of the documentation explains how to do that.

```
$ docker-compose build bigchaindb
$ docker-compose up -d bdb
```

The above command will launch all 3 main required services/processes:

- `mongodb`
- `tendermint`
- `bigchaindb`

To follow the logs of the `tendermint` service:

```
$ docker-compose logs -f tendermint
```

To follow the logs of the `bigchaindb` service:

```
$ docker-compose logs -f bigchaindb
```

To follow the logs of the `mongodb` service:

```
$ docker-compose logs -f mdb
```

Simple health check:

```
$ docker-compose up curl-client
```

Post and retrieve a transaction – copy/paste a driver basic example of a CREATE transaction:

```
$ docker-compose -f docker-compose.yml run --rm bdb-driver ipython
```

TODO: A python script to post and retrieve a transaction(s).

Running Tests

Run all the tests using:

```
$ docker-compose run --rm --no-deps bigchaindb pytest -v
```

Run tests from a file:

```
$ docker-compose run --rm --no-deps bigchaindb pytest /path/to/file -v
```

Run specific tests:

```
$ docker-compose run --rm --no-deps bigchaindb pytest /path/to/file -k "<test_name>" -v
```

Building Docs

You can also develop and build the BigchainDB docs using `docker-compose`:

```
$ docker-compose build bdocs
$ docker-compose up -d bdocs
```

The docs will be hosted on port **33333**, and can be accessed over `localhost`, `127.0.0.1` OR `http://HOST_IP:33333`.

1.2.3 Notes on Running a Local Dev Node as Processes

The following doc describes how to run a local node for developing BigchainDB Tendermint version.

There are two crucial dependencies required to start a local node:

- MongoDB
- Tendermint

and of course you also need to install BigchainDB Sever from the local code you just developed.

Install and Run MongoDB

MongoDB can be easily installed, just refer to their [installation documentation](#) for your distro. We know MongoDB 3.4 and 3.6 work with BigchainDB. After the installation of MongoDB is complete, run MongoDB using `sudo mongod`

Install and Run Tendermint

Installing a Tendermint Executable

The version of BigchainDB Server described in these docs only works well with Tendermint 0.22.8 (not a higher version number). Install that:

```
$ sudo apt install -y unzip
$ wget https://github.com/tendermint/tendermint/releases/download/v0.22.8/tendermint_0.22.8_linux_amd64.zip
$ unzip tendermint_0.22.8_linux_amd64.zip
$ rm tendermint_0.22.8_linux_amd64.zip
$ sudo mv tendermint /usr/local/bin
```

Installing Tendermint Using Docker

Tendermint can be run directly using the docker image. Refer [here](#) for more details.

Installing Tendermint from Source

Before we can begin installing Tendermint one should ensure that the Golang is installed on system and `$GOPATH` should be set in the `.bashrc` or `.zshrc`. An example setup is shown below

```
$ echo $GOPATH
/home/user/Documents/go
$ go -h
Go is a tool for managing Go source code.

Usage:

    go command [arguments]

The commands are:

    build      compile packages and dependencies
    clean      remove object files
    ...
```

- We can drop `GOPATH` in `PATH` so that installed Golang packages are directly available in the shell. To do that add the following to your `.bashrc`

```
export PATH=${PATH}:${GOPATH}/bin
```

Follow [the Tendermint docs](#) to install Tendermint from source.

If the installation is successful then Tendermint is installed at `$GOPATH/bin`. To ensure Tendermint's installed fine execute the following command,

```
$ tendermint -h
Tendermint Core (BFT Consensus) in Go

Usage:
    tendermint [command]

Available Commands:
    gen_validator      Generate new validator keypair
    help               Help about any command
    init               Initialize Tendermint
    ...
```

Running Tendermint

- We can initialize and run tendermint as follows,

```
$ tendermint init
...
$ tendermint node --consensus.create_empty_blocks=false
```

The argument `--consensus.create_empty_blocks=false` specifies that Tendermint should not commit empty blocks.

- To reset all the data stored in Tendermint execute the following command,

```
$ tendermint unsafe_reset_all
```

Install BigchainDB

To install BigchainDB from source (for dev), clone the repo and execute the following command, (it is better that you create a virtual env for this)

```
$ git clone https://github.com/bigchaindb/bigchaindb.git
$ cd bigchaindb
$ pip install -e .[dev] # or pip install -e '.[dev]' # for zsh
```

Running All Tests

To execute tests when developing a feature or fixing a bug one could use the following command,

```
$ pytest -v
```

NOTE: MongoDB and Tendermint should be running as discussed above.

One could mark a specific test and execute the same by appending `-m my_mark` to the above command.

Although the above should prove sufficient in most cases but in case tests are failing on Travis CI then the following command can be used to possibly replicate the failure locally,

```
$ docker-compose run --rm --no-deps bdb pytest -v --cov=bigchaindb
```

NOTE: before executing the above command the user must ensure that they reset the Tendermint container by executing `tendermint unsafe_reset_all` command in the Tendermint container.

1.2.4 Run a BigchainDB network

NOT for Production Use

You can use the following instructions to deploy a single or multi node BigchainDB network for dev/test using the extensible `stack` script(s).

Currently, this workflow is only supported for the following Operating systems:

- Ubuntu \geq 16.04
- CentOS \geq 7
- Fedora \geq 24
- MacOSX

Minimum Requirements

Minimum resource requirements for a single node BigchainDB dev setup. **The more the better:**

- Memory \geq 512MB
- VCPUs \geq 1

Download the scripts

Note: If you're working on BigchainDB Server code, on a branch based on recent code, then you already have local recent versions of *stack.sh* and *unstack.sh* in your `bigchaindb/pkg/scripts/` directory. Otherwise you can get them using:

```
$ export GIT_BRANCH=master
$ curl -fOL https://raw.githubusercontent.com/bigchaindb/bigchaindb/${GIT_BRANCH}/pkg/
↳scripts/stack.sh

# Optional
$ curl -fOL https://raw.githubusercontent.com/bigchaindb/bigchaindb/${GIT_BRANCH}/pkg/
↳scripts/unstack.sh
```

Quick Start

If you run `stack.sh` out of the box i.e. without any configuration changes, you will be able to deploy a 4 node BigchainDB network with Docker containers, created from master branch of `bigchaindb/bigchaindb` repo and Tendermint version 0.22.8.

Note: Run `stack.sh` with either root or non-root user with `sudo` enabled.

```
$ bash stack.sh
...Logs..
.....
.....
Finished stacking!
```

Configure the BigchainDB network

The `stack.sh` script has multiple deployment methods and parameters and they can be explored using: `bash stack.sh -h`

```
$ bash stack.sh -h

Usage: $ bash stack.sh [-h]

Deploys the BigchainDB network.

ENV[STACK_SIZE]
  Set STACK_SIZE environment variable to the size of the network you desire.
  Network mimics a production network environment with single or multiple BDB
  nodes. (default: 4).

ENV[STACK_TYPE]
  Set STACK_TYPE environment variable to the type of deployment you desire.
  You can set it one of the following: ["docker", "local", "cloud"].
  (default: docker)

ENV[STACK_TYPE_PROVIDER]
  Set only when STACK_TYPE="cloud". Only "azure" is supported.
  (default: )

ENV[STACK_VM_MEMORY]
  (Optional) Set only when STACK_TYPE="local". This sets the memory
```

(continues on next page)

(continued from previous page)

```

of the instance(s) spawned. (default: 2048)

ENV[STACK_VM_CPUS]
  (Optional) Set only when STACK_TYPE="local". This sets the number of VCPUs
  of the instance(s) spawned. (default: 2)

ENV[STACK_BOX_NAME]
  (Optional) Set only when STACK_TYPE="local". This sets the box Vagrant box_
↳name
  of the instance(s) spawned. (default: ubuntu/xenial64)

ENV[STACK_REPO]
  (Optional) To configure bigchaindb repo to use, set STACK_REPO environment
  variable. (default: bigchaindb/bigchaindb)

ENV[STACK_BRANCH]
  (Optional) To configure bigchaindb repo branch to use set STACK_BRANCH_
↳environment
  variable. (default: master)

ENV[TM_VERSION]
  (Optional) Tendermint version to use for the setup. (default: 0.22.8)

ENV[MONGO_VERSION]
  (Optional) MongoDB version to use with the setup. (default: 3.6)

ENV[AZURE_CLIENT_ID]
  Only required when STACK_TYPE="cloud" and STACK_TYPE_PROVIDER="azure". Steps_
↳to generate:
  https://github.com/Azure/vagrant-azure#create-an-azure-active-directory-aad-
↳application

ENV[AZURE_TENANT_ID]
  Only required when STACK_TYPE="cloud" and STACK_TYPE_PROVIDER="azure". Steps_
↳to generate:
  https://github.com/Azure/vagrant-azure#create-an-azure-active-directory-aad-
↳application

ENV[AZURE_SUBSCRIPTION_ID]
  Only required when STACK_TYPE="cloud" and STACK_TYPE_PROVIDER="azure". Steps_
↳to generate:
  https://github.com/Azure/vagrant-azure#create-an-azure-active-directory-aad-
↳application

ENV[AZURE_CLIENT_SECRET]
  Only required when STACK_TYPE="cloud" and STACK_TYPE_PROVIDER="azure". Steps_
↳to generate:
  https://github.com/Azure/vagrant-azure#create-an-azure-active-directory-aad-
↳application

ENV[AZURE_REGION]
  (Optional) Only applicable, when STACK_TYPE="cloud" and STACK_TYPE_PROVIDER=
↳"azure".
  Azure region for the BigchainDB instance. Get list of regions using Azure CLI.
  e.g. az account list-locations. (default: westeurope)

ENV[AZURE_IMAGE_URN]

```

(continues on next page)

(continued from previous page)

```
(Optional) Only applicable, when STACK_TYPE="cloud" and STACK_TYPE_PROVIDER=
↪"azure".
  Azure image to use. Get list of available images using Azure CLI.
  e.g. az vm image list --output table. (default: Canonical:UbuntuServer:16.04-
↪LTS:latest)

  ENV[AZURE_RESOURCE_GROUP]
  (Optional) Only applicable, when STACK_TYPE="cloud" and STACK_TYPE_PROVIDER=
↪"azure".
  Name of Azure resource group for the instance.
  (default: bdb-vagrant-rg-2018-05-30)

  ENV[AZURE_DNS_PREFIX]
  (Optional) Only applicable, when STACK_TYPE="cloud" and STACK_TYPE_PROVIDER=
↪"azure".
  DNS Prefix of the instance. (default: bdb-instance-2018-05-30)

  ENV[AZURE_ADMIN_USERNAME]
  (Optional) Only applicable, when STACK_TYPE="cloud" and STACK_TYPE_PROVIDER=
↪"azure".
  Admin username of the the instance. (default: vagrant)

  ENV[AZURE_VM_SIZE]
  (Optional) Only applicable, when STACK_TYPE="cloud" and STACK_TYPE_PROVIDER=
↪"azure".
  Azure VM size. (default: Standard_D2_v2)

  ENV[SSH_PRIVATE_KEY_PATH]
  Only required when STACK_TYPE="cloud" and STACK_TYPE_PROVIDER="azure".
↪Absolute path of
  SSH keypair required to log into the Azure instance.

  -h
  Show this help and exit.
```

The parameter that differentiates between the deployment type is `STACK_TYPE` which currently, supports an opinionated deployment of BigchainDB on docker, local and cloud.

STACK_TYPE: docker

This configuration deploys a docker based BigchainDB network on the dev/test machine that you are running `stack.sh` on. This is also the default `STACK_TYPE` config for `stack.sh`.

Example

Deploy a 4 node docker based BigchainDB network on your host.

```
#Optional, since 4 is the default size.
$ export STACK_SIZE=4

#Optional, since docker is the default type.
$ export STACK_TYPE=docker
```

(continues on next page)

(continued from previous page)

```
#Optional, repo to use for the network deployment
# Default: bigchaindb/bigchaindb
$ export STACK_REPO=bigchaindb/bigchaindb

#Optional, codebase to use for the network deployment
# Default: master
$ export STACK_BRANCH=master

#Optional, since 0.22.8 is the default tendermint version.
$ export TM_VERSION=0.22.8

#Optional, since 3.6 is the default MongoDB version.
$ export MONGO_VERSION=3.6

$ bash stack.sh
```

Note: For MacOSX users, the script will not install Docker for Mac, it only detects if docker is installed on the system, if not make sure to install [Docker for Mac](#). Also make sure Docker API Version > 1.25. To check Docker API Version:

```
docker version --format '{{.Server.APIVersion}}'
```

STACK_TYPE: local

This configuration deploys a VM based BigchainDB network on your host/dev. All the services are running as processes on the VMs. For local deployments the following dependencies must be installed i.e.

- Vagrant
 - Vagrant plugins.
 - * vagrant-cachier
 - * vagrant-vbguest
 - * vagrant-hosts
 - * vagrant-azure
 - vagrant plugin install vagrant-cachier vagrant-vbguest
vagrant-hosts vagrant-azure
- Virtualbox

Example

Deploy a 4 node VM based BigchainDB network.

```
$ export STACK_TYPE=local

# Optional, since 4 is the default size.
$ export STACK_SIZE=4

# Optional, default is 2048
$ export STACK_VM_MEMORY=2048

#Optional, default is 1
```

(continues on next page)

(continued from previous page)

```
$ export STACK_VM_CPUS=1

#Optional, default is ubuntu/xenial64. Supported/tested images: bento/centos-7, ↵
↵fedora/25-cloud-base
$ export STACK_BOX_NAME=ubuntu/xenial64

#Optional, repo to use for the network deployment
# Default: bigchaindb/bigchaindb
$ export STACK_REPO=bigchaindb/bigchaindb

#Optional, codebase to use for the network deployment
# Default: master
$ export STACK_BRANCH=master

#Optional, since 0.22.8 is the default tendermint version
$ export TM_VERSION=0.22.8

#Optional, since 3.6 is the default MongoDB version.
$ export MONGO_VERSION=3.6

$ bash stack.sh
```

STACK_TYPE: cloud

This configuration deploys a docker based BigchainDB network on a cloud instance. Currently, only Azure is supported. For cloud deployments the following dependencies must be installed i.e.

- Vagrant
 - Vagrant plugins.
 - * vagrant-cachier
 - * vagrant-vbguest
 - * vagrant-hosts
 - * vagrant-azure
 - vagrant plugin install vagrant-cachier vagrant-vbguest vagrant-hosts vagrant-azure

Example: stack

Deploy a 4 node docker based BigchainDB network on an Azure instance.

- [Create an Azure Active Directory\(AAD\) Application](#)
- Generate or specify an SSH keypair to login to the Azure instance.

– **Example:**

```
$ ssh-keygen -t rsa -C "<name>" -f /path/to/key/<name>
```

- Configure parameters for `stack.sh`

```

# After creating the AAD application with access to Azure Resource
# Group Manager for your subscription, it will return a JSON object

$ export AZURE_CLIENT_ID=<value from azure.appId>

$ export AZURE_TENANT_ID=<value from azure.tenant>

# Can be retrieved via
# az account list --query "[?isDefault].id" -o tsv
$ export AZURE_SUBSCRIPTION_ID=<your Azure subscription ID>

$ export AZURE_CLIENT_SECRET=<value from azure.password>

$ export STACK_TYPE=cloud

# Currently on azure is supported
$ export STACK_TYPE_PROVIDER=azure

$ export SSH_PRIVATE_KEY_PATH=</path/to/private/key>

# Optional, Azure region of the instance. Default: westeurope
$ export AZURE_REGION=westeurope

# Optional, Azure image urn of the instance. Default: Canonical:UbuntuServer:16.04-
↳LTS:latest
$ export AZURE_IMAGE_URN=Canonical:UbuntuServer:16.04-LTS:latest

# Optional, Azure resource group. Default: bdb-vagrant-rg-yyyy-mm-dd(current date)
$ export AZURE_RESOURCE_GROUP=bdb-vagrant-rg-2018-01-01

# Optional, DNS prefix of the Azure instance. Default: bdb-instance-yyyy-mm-
↳dd(current date)
$ export AZURE_DNS_PREFIX=bdb-instance-2018-01-01

# Optional, Admin username of the Azure instance. Default: vagrant
$ export AZURE_ADMIN_USERNAME=vagrant

# Optional, Azure instance size. Default: Standard_D2_v2
$ export AZURE_VM_SIZE=Standard_D2_v2

$ bash stack.sh

```

Delete/Unstack a BigchainDB network

Export all the variables exported for the corresponding `stack.sh` script and run `unstack.sh` to delete/remove/unstack the BigchainDB network/stack.

```

$ bash unstack.sh

OR

# -s implies soft unstack. i.e. Only applicable for local and cloud based
# networks. Only deletes/stops the docker(s)/process(es) and does not
# delete the instances created via Vagrant or on Cloud. Default: hard
$ bash unstack.sh -s

```

1.2.5 Run a BigchainDB network with Ansible

NOT for Production Use

You can use the following instructions to deploy a single or multi node BigchainDB network for dev/test using Ansible. Ansible will configure the BigchainDB node(s).

Currently, this workflow is only supported for the following distributions:

- Ubuntu \geq 16.04
- CentOS \geq 7
- Fedora \geq 24
- MacOSX

Minimum Requirements

Minimum resource requirements for a single node BigchainDB dev setup. **The more the better:**

- Memory \geq 512MB
- VCPUs \geq 1

Clone the BigchainDB repository

```
$ git clone https://github.com/bigchaindb/bigchaindb.git
```

Install dependencies

- Ansible

You can also install `ansible` and other dependencies, if any, using the `bootstrap.sh` script inside the BigchainDB repository. Navigate to `bigchaindb/pkg/scripts` and run the `bootstrap.sh` script to install the dependencies for your OS. The script also checks if the OS you are running is compatible with the supported versions.

Note: `bootstrap.sh` only supports Ubuntu \geq 16.04, CentOS \geq 7 and Fedora \geq 24 and MacOSX.

```
$ cd bigchaindb/pkg/scripts/  
$ bash bootstrap.sh --operation install
```

BigchainDB Setup Configuration(s)

Local Setup

You can run the Ansible playbook `bigchaindb-start.yml` on your local dev machine and set up the BigchainDB node where BigchainDB can be run as a process or inside a Docker container(s) depending on your configuration.

Before, running the playbook locally, you need to update the `hosts` and `stack-config.yml` configuration, which will notify Ansible that we need to run the play locally.

Update Hosts

Navigate to `bigchaindb/pkg/configuration/hosts` inside the BigchainDB repository.

```
$ cd bigchaindb/pkg/configuration/hosts
```

Edit all configuration file:

```
# Delete any existing configuration in this file and insert
# Hostname of dev machine
<HOSTNAME> ansible_connection=local
```

Update Configuration

Navigate to `bigchaindb/pkg/configuration/vars` inside the BigchainDB repository.

```
$ cd bigchaindb/pkg/configuration/vars/stack-config.yml
```

Edit `bdb-config.yml` configuration file as per your requirements, sample configuration file(s):

```
---
stack_type: "docker"
stack_size: "4"

OR

---
stack_type: "local"
stack_size: "1"
```

BigchainDB Setup

Now, You can safely run the `bigchaindb-start.yml` playbook and everything will be taken care of by Ansible. To run the playbook please navigate to the `bigchaindb/pkg/configuration` directory inside the BigchainDB repository and run the `bigchaindb-start.yml` playbook.

```
$ cd bigchaindb/pkg/configuration/
$ ansible-playbook bigchaindb-start.yml -i hosts/all --extra-vars "operation=start_
↪home_path=$(pwd) "
```

After successful execution of the playbook, you can verify that BigchainDB docker(s)/process(es) is(are) running.

Verify BigchainDB process(es):

```
$ ps -ef | grep bigchaindb
```

OR

Verify BigchainDB Docker(s):

```
$ docker ps | grep bigchaindb
```

Contributing to BigchainDB

You can now send transactions and verify the functionality of your BigchainDB node. See the [BigchainDB Python Driver documentation](#) for details on how to use it.

Note: The `bdb_root_url` can be one of the following:

```
# BigchainDB is running as a process
bdb_root_url = http://<HOST-IP>:9984

OR

# BigchainDB is running inside a docker container
bdb_root_url = http://<HOST-IP>:<DOCKER-PUBLISHED-PORT>
```

Note: BigchainDB has [other drivers](#) as well.

Experimental: Running Ansible a Remote Dev/Host

Remote Setup

You can also run the Ansible playbook `bigchaindb-start.yml` on remote machine(s) and set up the BigchainDB node where BigchainDB can run as a process or inside a Docker container(s) depending on your configuration.

Before, running the playbook on a remote host, you need to update the `hosts` and `stack-config.yml` configuration, which will notify Ansible that we need to run the play on a remote host.

Update Hosts

Navigate to `bigchaindb/pkg/configuration/hosts` inside the BigchainDB repository.

```
$ cd bigchaindb/pkg/configuration/hosts
```

Edit all configuration file:

```
# Delete any existing configuration in this file and insert
<Remote_Host_IP/Hostname> ansible_ssh_user=<USERNAME> ansible_sudo_pass=<PASSWORD>
```

Note: You can add multiple hosts to the `all` configuration file. Non-root user with `sudo` enabled password is needed because ansible will run some tasks that require those permissions.

Note: You can also use other methods to get inside the remote machines instead of password based SSH. For other methods please consult [Ansible Documentation](#).

Update Configuration

Navigate to `bigchaindb/pkg/configuration/vars` inside the BigchainDB repository.

```
$ cd bigchaindb/pkg/configuration/vars/stack-config.yml
```

Edit `stack-config.yml` configuration file as per your requirements, sample configuration file(s):

```
---
stack_type: "docker"
stack_size: "4"
```

(continues on next page)

(continued from previous page)

```
OR
---
stack_type: "local"
stack_type: "1"
```

After, the configuration of remote hosts, *run the Ansible playbook and verify your deployment.*

1.3 Policies

1.3.1 Contributor Code of Conduct

As contributors and maintainers of this project, and in the interest of fostering an open and welcoming community, we pledge to respect all people who contribute to the project.

We are committed to making participation in this project a harassment-free experience for everyone, regardless of level of experience, gender, gender identity and expression, sexual orientation, disability, personal appearance, body size, race, ethnicity, age, religion, nationality, or species—no picking on Wrigley for being a buffalo!

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery
- Personal attacks
- Trolling or insulting/derogatory comments
- Public or private harassment
- Publishing other’s private information, such as physical or electronic addresses, without explicit permission
- Deliberate intimidation
- Other unethical or unprofessional conduct

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

By adopting this Code of Conduct, project maintainers commit themselves to fairly and consistently applying these principles to every aspect of managing this project. Project maintainers who do not follow or enforce the Code of Conduct may be permanently removed from the project team.

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community.

Instances of abusive, harassing, or otherwise unacceptable behavior directed at yourself or another community member may be reported by contacting a project maintainer at contact@bigchaindb.com. All complaints will be reviewed and investigated and will result in a response that is appropriate to the circumstances. Maintainers are obligated to maintain confidentiality with regard to the reporter of an incident.

This Code of Conduct is adapted from the [Contributor Covenant](http://contributor-covenant.org/version/1/3/0/), version 1.3.0, available at <http://contributor-covenant.org/version/1/3/0/>

```
shortname: BEP-6
name: Shared Workspace Protocol
type: Meta
```

(continues on next page)

```
status: Draft
editor: Alberto Granzotto <alberto@bigchaindb.com>
```

1.3.2 Abstract

The “Shared Workspace Protocol” provides a simple set of rules for people working in the same workspace.

1.3.3 Motivation

Focusing in a shared workspace (or in the office in general) can be a really challenging task. Working from home solves the problem, but it is not an ideal solution if forced by the difficulty to focus in the office.

1.3.4 Language

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “NOT RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [BCP 14 \[RFC2119\] \[RFC8174\]](#) when, and only when, they appear in all capitals, as shown here.

1.3.5 Rationale

SWP provides a reusable collaboration model for people working in the same workspace. It has these specific goals:

1. To maximize focus and productivity, by reducing the sources of distraction.
2. To reduce the burden to ask people to “behave” or “leave the room”, by having an explicit protocol people agree before entering the room.

1.3.6 Implementation

1. When implemented, this protocol **MUST** be visible at the entrance of the shared workspace.
2. A person **SHOULD** write a message to another person before speaking with them.
3. A person **SHOULD NOT** be loud.
4. A person **SHOULD NOT** talk to more than one person at a time.
5. One-on-one conversations **MUST** be short and occasional.
6. A person **MUST NOT** have calls unless everyone in the room is involved.

1.3.7 Copyright Waiver

1.3.8 Python Style Guide

This guide starts out with our general Python coding style guidelines and ends with a section on how we write & run (Python) tests.

General Python Coding Style Guidelines

Our starting point is [PEP8](#), the standard “Style Guide for Python Code.” Many Python IDEs will check your code against PEP8. (Note that PEP8 isn’t frozen; it actually changes over time, but slowly.)

BigchainDB uses Python 3.5+, so you can ignore all PEP8 guidelines specific to Python 2.

We use [pre-commit](#) to check some of the rules below before every commit but not everything is realized yet. The hooks we use can be found in the [.pre-commit-config.yaml](#) file.

Python Docstrings

PEP8 says some things about docstrings, but not what to put in them or how to structure them. [PEP257](#) was one proposal for docstring conventions, but we prefer [Google-style docstrings](#) instead: they’re easier to read and the [napoleon extension](#) for Sphinx lets us turn them into nice-looking documentation. Here are some references on Google-style docstrings:

- [Google’s docs on Google-style docstrings](#)
- [napoleon’s docs include an overview of Google-style docstrings](#)
- [Example Google-style docstrings](#) (from [napoleon’s docs](#))

Maximum Line Length

PEP8 has some [maximum line length guidelines](#), starting with “Limit all lines to a maximum of 79 characters” but “for flowing long blocks of text with fewer structural restrictions (docstrings or comments), the line length should be limited to 72 characters.”

We discussed this at length, and it seems that the consensus is: *try* to keep line lengths less than 79/72 characters, unless you have a special situation where longer lines would improve readability. (The basic reason is that 79/72 works for everyone, and BigchainDB is an open source project.) As a hard limit, keep all lines less than 119 characters (which is the width of GitHub code review).

Single or Double Quotes?

Python lets you use single or double quotes. PEP8 says you can use either, as long as you’re consistent. We try to stick to using single quotes, except in cases where using double quotes is more readable. For example:

```
print('This doesn\'t look so nice.')
print("Doesn't this look nicer?")
```

Breaking Strings Across Multiple Lines

Should we use parentheses or slashes (\) to break strings across multiple lines, i.e.

```
my_string = ('This is a very long string, so long that it will not fit into just one_
↳line '
              'so it must be split across multiple lines.')
# or
my_string = 'This is a very long string, so long that it will not fit into just one_
↳line ' \
            'so it must be split across multiple lines.'
```

It seems the preference is for slashes, but using parentheses is okay too. (There are good arguments either way. Arguing about it seems like a waste of time.)

How to Format Long import Statements

If you need to `import` lots of names from a module or package, and they won't all fit in one line (without making the line too long), then use parentheses to spread the names across multiple lines, like so:

```
from Tkinter import (
    Tk, Frame, Button, Entry, Canvas, Text,
    LEFT, DISABLED, NORMAL, RIDGE, END,
)

# Or

from Tkinter import (Tk, Frame, Button, Entry, Canvas, Text,
                     LEFT, DISABLED, NORMAL, RIDGE, END)
```

For the rationale, see [PEP 328](#).

Using the % operator or `format()` to Format Strings

Given the choice:

```
x = 'name: %s; score: %d' % (name, n)
# or
x = 'name: {}; score: {}'.format(name, n)
```

we use the `format()` version. The [official Python documentation](#) says, “This method of string formatting is the new standard in Python 3, and should be preferred to the `%` formatting described in String Formatting Operations in new code.”

Running the Flake8 Style Checker

We use [Flake8](#) to check our Python code style. Once you have it installed, you can run it using:

```
flake8 --max-line-length 119 bigchaindb/
```

Writing and Running (Python) Tests

The content of this section was moved to [bigchaindb/tests/README.md](#).

Note: We automatically run all tests on all pull requests (using Travis CI), so you should definitely run all tests locally before you submit a pull request. See the above-linked README file for instructions.

1.3.9 Our Release Process

Notes

BigchainDB follows the Python form of [Semantic Versioning](#) (i.e. MAJOR.MINOR.PATCH), which is almost identical to regular semantic versioning, but there's no hyphen, e.g.

- 0.9.0 for a typical final release
- 4.5.2a1 not 4.5.2-a1 for the first Alpha release
- 3.4.5rc2 not 3.4.5-rc2 for Release Candidate 2

Note 1: For Git tags (which are used to identify releases on GitHub), we append a `v` in front. For example, the Git tag for version 2.0.0a1 was `v2.0.0a1`.

Note 2: For Docker image tags (e.g. on Docker Hub), we use longer version names for Alpha, Beta and Release Candidate releases. For example, the Docker image tag for version 2.0.0a2 was `2.0.0-alpha2`.

We use 0.9 and 0.9.0 as example version and short-version values below. You should replace those with the correct values for your new version.

We follow [BEP-1](#), which is our variant of C4, the Collective Code Construction Contract, so a release is just a [tagged commit](#) on the `master` branch, i.e. a label for a particular Git commit.

The following steps are what we do to release a new version of *BigchainDB Server*. The steps to release the Python Driver are similar but not the same.

Steps

1. Create a pull request where you make the following changes:

- Update `CHANGELOG.md`
- Update all Docker image tags in all Kubernetes YAML files (in the `k8s/` directory). For example, in the files:
 - `k8s/bigchaindb/bigchaindb-ss.yaml` and
 - `k8s/dev-setup/bigchaindb.yaml`
 find the line of the form `image: bigchaindb/bigchaindb:0.8.1` and change the version number to the new version number, e.g. 0.9.0. (This is the Docker image that Kubernetes should pull from Docker Hub.) Keep in mind that this is a *Docker image tag* so our naming convention is a bit different; see Note 2 in the **Notes** section above.
- In `bigchaindb/version.py`:
 - update `__version__` to e.g. 0.9.0 (with no `.dev` on the end)
 - update `__short_version__` to e.g. 0.9 (with no `.dev` on the end)
- In the docs about installing BigchainDB (and Tendermint), and in the associated scripts, recommend/install a version of Tendermint that *actually works* with the soon-to-be-released version of BigchainDB. You can find all such references by doing a search for the previously-recommended version number, such as 0.22.8.
- In `setup.py`, *maybe* update the development status item in the `classifiers` list. For example, one allowed value is `"Development Status :: 5 - Production/Stable"`. The allowed values are listed at pypi.python.org.

2. **Wait for all the tests to pass!**

3. Merge the pull request into the `master` branch.

4. Go to the [bigchaindb/bigchaindb Releases page on GitHub](#) and click the “Draft a new release” button.

5. Fill in the details:

- **Tag version:** version number preceded by `v`, e.g. `v0.9.1`

- **Target:** the last commit that was just merged. In other words, that commit will get a Git tag with the value given for tag version above.
- **Title:** Same as tag version above, e.g `v0.9.1`
- **Description:** The body of the changelog entry (Added, Changed, etc.)

6. Click “Publish release” to publish the release on GitHub.
7. On your local computer, make sure you’re on the `master` branch and that it’s up-to-date with the `master` branch in the `bigchaindb/bigchaindb` repository (e.g. `git pull upstream master`). We’re going to use that to push a new `bigchaindb` package to PyPI.
8. Make sure you have a `~/.pypirc` file containing credentials for PyPI.
9. Do `make release` to build and publish the new `bigchaindb` package on PyPI. For this step you need to have `twine` installed. If you get an error like `Makefile:135: recipe for target 'clean-pyc' failed then try doing`

```
sudo chown -R $(whoami):$(whoami) .
```

10. Log in to readthedocs.org and go to the **BigchainDB Server** project, then:
 - Click on “Builds”, select “latest” from the drop-down menu, then click the “Build Version:” button.
 - Wait for the build of “latest” to finish. This can take a few minutes.
 - Go to Admin → Advanced Settings and make sure that “Default branch:” (i.e. what “latest” points to) is set to the new release’s tag, e.g. `v0.9.1`. (It won’t be an option if you didn’t wait for the build of “latest” to finish.) Then scroll to the bottom and click “Save”.
 - Go to Admin → Versions and under **Choose Active Versions**, do these things:
 - (a) Make sure that the new version’s tag is “Active” and “Public”
 - (b) Make sure the **stable** branch is *not* active.
 - (c) Scroll to the bottom of the page and click “Save”.
11. Go to [Docker Hub](https://dockerhub.com) and sign in, then:
 - Click on “Organizations”
 - Click on “bigchaindb”
 - Click on “bigchaindb/bigchaindb”
 - Click on “Build Settings”
 - Find the row where “Docker Tag Name” equals `latest` and change the value of “Name” to the name (Git tag) of the new release, e.g. `v0.9.0`.
 - If the release is an Alpha, Beta or Release Candidate release, then a new row must be added. You can do that by clicking the green “+” (plus) icon. The contents of the new row should be similar to the existing rows of previous releases like that.
 - Click on “Tags”
 - Delete the “latest” tag (so we can rebuild it)
 - Click on “Build Settings” again
 - Click on the “Trigger” button for the “latest” tag and make sure it worked by clicking on “Tags” again
 - If the release is an Alpha, Beta or Release Candidate release, then click on the “Trigger” button for that tag as well.

Congratulations, you have released a new version of BigchainDB Server!