ubuntu-server-setup

Release 1.3.0

Victor Miti

Jun 05, 2022

CONTENTS

Note: ubuntu-server-setup is an opinionated bash setup script to automate the setup and provisioning of Ubuntu servers, **primarily biased towards python web applications** (Django, Flask, etc.).

Why this documentation?

This documentation was written to provide additional context and rationale behind the engineervix/ubuntu-server-setup project, which was forked from jasonheecs/ubuntu-server-setup.

It is hoped that this will provide a useful reference not only for those who seek to use the setup script to bootstrap their Ubuntu server, but also for those that would like to fork the project and adapt it to their specific needs, just as I did.

Perhaps you like the project and would like to contribute to its improvement – this documentation will hopefully help you get a better understanding of what's going on "under the hood".

Notwithstanding, I also wrote this documentation for my future self, because many times I forget the very code that I wrote, and even wonder why and how I wrote it!

CHAPTER

ONE

INTRODUCTION

You have finished developing your awesome web application – everything works perfectly on your computer, but you now need to put it out there so that others can use it and enjoy it. For Python web applications, moving from development to production can be quite a daunting task, especially given the fact that there are many options available.

Well, the "traditional" approach to deploying web applications is to use a GNU/Linux server in "the cloud". This entails

- setting up a GNU/Linux server (Digital Ocean, Linode, AWS EC2, etc.), preferably using the latest Ubuntu LTS version.
- Securing your server
- Installing and configuring Nginx, PostgreSQL, uWSGI/Gunicorn, Redis, Certbot and any other dependencies

As a developer, you don't want to spend so much time configuring servers instead of focusing on building your application. Because setting up servers can be a tedious task, it is better to automate this process. Again, there are so many approaches towards automation of infrastructure deployments (e.g. Ansible, Salt, Chef, Puppet, Terraform, Docker, Kubernetes, etc.), these have been evolving over the years, and continue to evolve.

A good old approach is simply writing a bash script to bootstrap your server and get things up and running quickly. Rather than starting something completely new, why not check what others have done and see if you could find something to use? Well, this is what I did, and I stumbled upon Jason Hee's awesome setup script.

Jason Hee's setup script automates the setup and provisioning of Ubuntu servers. According to the project's README, it does the following:

- Adds a new user account with sudo access
- Adds a public ssh key for the new user account
- Disables password authentication to the server
- Deny root login to the server
- Setup Uncomplicated Firewall
- Create Swap file based on machine's installed memory
- Setup the timezone for the server (Default to "Asia/Singapore")
- Install Network Time Protocol

This provides an excellent starting point for provisioning Ubuntu Servers. I decided to fork the project and build on top of this strong foundation to develop a heavily opinionated setup for **deploying python web applications**. The idea is to be able to quickly setup a Linux box and deploy a Python web application without much of a hassle.

In the next section, I will highlight the additional features that I introduced and the rationale behind them.

CHAPTER

TWO

PROJECT FEATURES

Note: We focus only on the additional features not available in the upstream version of this fork. The original features are listed in the *Introduction* and in the original project's README.

2.1 Terminal Enhancements

- ZSH + ohmyzsh + Powerlevel10k =
- includes several cool ZSH plugins, for example:
  - zsh-autosuggestions
  - zsh-syntax-highlighting
- colorized `ls` output with colour and icons, courtesy of Color LS
- fancy Tmux configuration using **powerline** and Tmux Plugin Manager
- custom `.zshrc` configuration, with several useful shell functions

2.2 Git Customization

- minimal global `git` configuration with `user.name`, `user.email` and `color.ui`

2.3 Node.js Stack

- You cannot build modern web applications without using the Node.js tech stack. Therefore, the latest LTS version of Node.js is installed, together with Yarn.
- a bunch of useful Node.js global packages:

aggedright

- browser-sync
- caniuse-cmd
- commitizen
- concurrently
- doctoc
- html-minifier
- grunt-cli
- gulp-cli
- lerna
- lite-server
- local-cors-proxy
- maildev
- mdpdf

- mozjpeg
- prettier
- sass
- semantic-release-cli
- serve
- standard-version
- svgo
- uglify-js

2.4 Python Stack

- Python3 and associated build tools
- virtualenvwrapper
- uwsgi
- celery
- TODO: setup pyenv

2.5 Ruby

Ruby has to be installed because

- Certain applications need it in order for them to be setup, for example, the Janus Vim distribution
- there are a couple of Ruby gems that we need (`travis` cli and `colorls`)

2.6 Database Stack

- Postgres + PostGIS

2.7 Webserver

- Nginx, with ready-to-use HTTPS, LetsEncrypt + Cloudflare configuration

2.8 Certbot

- ready to use with Cloudflare, via the dns_cloudflare plugin
- automatic renewal and custom post-renewal script

2.9 Text Editor

- Includes Janus – a Vim Distribution designed to provide minimal working environment using the most popular plugins and the most common mappings.

2.10 Mail

- The server is configured to send emails using Postfix and Sendgrid
- includes the Mutt email Client

2.11 System Administration

- Unattended upgrades and automatic reboots when necessary
- Monitoring of server logs and email notifications using Logwatch

2.12 Additional Server Hardening

- restrict access to the server by specifying who is allowed to login
- secure shared memory
- fail2ban
- lynis
- rkhunter

2.13 Additional Packages

aggedright

- Redis
- Memcached
- TeX-Live
- openjdk-8-jdk
- travis cli
- wkhtmltopdf
- pdftk
- ffmpeg
- youtube-dl
- rclone
- volta
- pngquant
- ocrmypdf
- xvfb
- rdiff-backup
- apt-clone
- firefox
- pandoc
- sqlite3
- poppler-utils
- ncdu
- libtool
- dos2unix
- scour
- shellcheck
- jq and yq
- inkscape
- libreoffice-common
- autoconf, automake and autotools-dev
- aspell and hunspell

2.14 Miscellaneous Tasks

- custom scripts and tools (e.g. geckodriver) in `$HOME/bin`
- custom directories for projects, backups and misc/temp

CHAPTER

THREE

PROJECT STRUCTURE

The project structure is as follows:

```
├── .janus
│   ├── jellygrass.vim
│   ├── vim
│   └── vim-kolor
├── configuration_files
│   ├── apticron.conf
│   ├── nginx
│   │   ├── letsencrypt.conf
│   │   └── ssl.conf
│   ├── origin-pull-ca.pem
│   ├── postfix_main.cf
│   ├── .tmux.conf
│   ├── uwsgi.service
│   ├── .vimrc.after
│   └── .zshrc
├── custom_scripts
│   ├── create_db
│   ├── get_release_notes.py
│   ├── janus_setup.sh
│   ├── release.py
│   └── shrinkpdf
├── docs
│   ├── build
│   ├── make.bat
│   ├── Makefile
│   ├── requirements.in
│   ├── requirements.txt
│   └── source
│       ├── 01_introduction.rst
│       ├── 02_features.rst
│       ├── 03_structure.rst
│       ├── 04_functions.rst
│       ├── 05_roadmap.rst
│       ├── 06_contributing.rst
│       ├── conf.py
│       ├── index.rst
│       ├── _static
│       └── _templates
```

(continues on next page)

(continued from previous page)

```
├── tests
│   ├── lib
│   ├── results
│   │   └── .gitkeep
│   ├── tests.sh
│   ├── unit-tests.sh
│   └── Vagrant
│       ├── Vagrantfile.bionic64
│       ├── Vagrantfile.focal64
│       ├── Vagrantfile.trusty32
│       ├── Vagrantfile.trusty64
│       ├── Vagrantfile.xenial32
│       └── Vagrantfile.xenial64
├── .all-contributorsrc
├── .gitattributes
├── .gitignore
├── .gitmodules
├── .readthedocs.yaml
├── .travis.yml
├── .versionrc
├── CHANGELOG.md
├── CONTRIBUTING.md
├── LICENSE
├── package.json
├── package-lock.json
├── README.md
├── setup.sh
└──setupLibrary.sh
```

3.1 Directories

.janus This directory contains custom janus configuration files. It gets copied into the `$HOME` directory

configuration_files Cloudflare, Vim, TMUX, Nginx, uwsgi, Postfix, apticron, `.zshrc`

custom_scripts `shrinkpdf` and `create_db` are copied to `$HOME/bin`. `janus_setup.sh` is used during installing, while `get_release_notes.py` and `release.py` are used during development of this project to automate releases.

docs Sphinx, documentation for the project. This is what you are currently reading!

tests Project test suite. Tests are run against a set of Vagrant VMs.

3.2 Files

Note: We'll not go through each and every file here, the assumption is that certain files are pretty obvious, for example, the git-related files such as `.gitignore` and `.gitattributes`, CI files like `.travis.yml`, etc.

.all-contributorsrc This project uses the all-contributors specification. The data used to generate the contributors list is stored in here.

.readthedocs.yaml The project documentation is hosted on Read the Docs. The configuration for the documentation builds is defined in here.

.versionrc This project uses standard-version. for versioning using semver and CHANGELOG generation powered by Conventional Commits. The `standard-version` configuration is defined in here, based on the conventional-changelog-config-spec.

CONTRIBUTING.md Guidelines on how to contribute to this project.

setup.sh This is the **core** of this project. This is the script that we actually run when setting up a new Ubuntu server. If you want to add additional features, you'll probably wanna edit this file. For details of the additional functions that constitute the basis for this fork, see *Functions*.

setupLibrary.sh Contains the initial setup functions plus a couple of helper functions that are "imported" in `setup.sh` above

CHAPTER

FOUR

FUNCTIONS

Note: We focus only on the additional functions not available in the upstream version of this fork.

You may find this section particularly useful if you would like to customize the existing features or add other features of your own.

In `setup.sh`, where these functions live, there's also a `main()` function which, as the name implies, is the main function. All these functions are called inside this `main()` function. If you write your own function, you'll wanna find somewhere within the `main()` function to call it.

extraHardening Here we restrict access to the server and secure shared memory

setupHostname At the time of developing this script, I was dealing primarily with AWS EC2 deployments, where you have to update the hostname and the `/etc/hosts` file, plus made some changes to the `/etc/cloud/cloud.cfg`. I realized that this was not required for other cloud service providers (e.g. Digital Ocean droplets and Hetzner Cloud servers). Therefore, by default, the call to this function is commented out in the `main()` function. You might wanna uncomment if deploying on AWS.

setupNodeYarn Install Node.js, yarn and some important global node packages

setupGit Global git configuration involving setting up of `user.name`, `user.email` and `color.ui true`

setupZSH Here we install and configure zsh, Oh My Zsh and the powerlevel10k theme.

setupRuby Simple ruby setup. Ruby is needed for the Janus Vim distribution, colorls and the Travis CI Client, among others.

setupTmux Tmux comes already installed with Ubuntu, and so there's no need to install it. Here we just install the Tmux Plugin Manager and add some configurations and styling (using Powerline)

setupPythonDev First and foremost, we install python, pip and related dev / build tools. Then, we install and configure virtualenvwrapper and uWSGI. Lastly, but not the least, we prepare the server for Celery, based on this blog post.

setupVim Install and configure Vim & related plugins, courtesy of the Janus Vim Distribution.

setupDatabases Install and configure PostgreSQL, PostGIS, Redis and Memcached.

setupWebServer Install and configure Nginx and Certbot (plus the `certbot-dns-cloudflare` plugin). This includes adding a cron job whith a **Let's Encrypt** renewal hook. Also generate a strong set of 4096 bit DH (Diffie-Hellman) parameters using openSSL.

setupMail Install and configure Postfix and related mail utilities. The setup assumes you're using Sendgrid's SMTP server/relay, but this can easily be customized to use other providers such as Mailjet, for instance.

configureSystemUpdatesAndLogs Updates notification, unattended upgrades, logs and other necessary System Administration stuff.

furtherHardening Install and configure `fail2ban`, lynis and rkhunter.

miscellaneousTasks Setup some folders for common operations, copy the custom scripts to the `~/bin` directory and install geckodriver (for use with selenium).

installExtraPackages Here we install a bunch of other packages that I find to be very useful. See *Additional Packages* for the details of these extra packages. Note that the list includes `texlive-full`, which may take a while to download and install. So if you're in a hurry and don't really need a full TeX distribution, then perhaps you might wanna comment it out.

CHAPTER

FIVE

ROADMAP

The upstream version of this setup script has been tested against Ubuntu 14.04, Ubuntu 16.04, Ubuntu 18.04 and Ubuntu 20.04. However, this fork primarily targets **Ubuntu 20.04**, and has only been tested on:

- official **AWS** Ubuntu 20.04 AMIs (Amazon EC2 Instances)
- Ubuntu 20.04 droplets on **DigitalOcean**
- Ubuntu 20.04 cloud servers on **Hetzner**

As the author, naturally, I will focus only on select cloud providers of interest, as I do not have the time to test on all the various platforms. If you have used this script on Azure, or GCP or vultr, Linode, etc, please share some feedback so that we know how things went!

For now, the focus is on

- fixing bugs whenever they are spotted
- adding minor features where necessary, with a focus on having a robust and secure server
- seeking to find ways to speed up the setup by minimizing manual input

In the TODO section of the README on GitHub, you'll probably find a concrete list of things that need to be done. If you find this project useful, please help improve it by contributing your code!

The bottomline

The idea is to ensure that the script **always works** on the *latest Ubuntu LTS release*.

CHAPTER

SIX

CONTRIBUTING

Contributions of any kind welcome!

When contributing to ubuntu-server-setup, please first create an issue to discuss the change you wish to make before making a change.

6.1 Before making a pull request

Note: This repo uses some Node.js-based tools (commitizen and standard-version) as part of the dev workflow. So, before you proceed, you'll need to ensure that Node.js is installed on your machine, and you are able to use `npm` to install Node.js packages

1. First, install the Commitizen cli tool if you don't already have it on your system: `sudo npm install commitizen -g`
2. Fork the repository.
3. Clone the repository from your GitHub.
4. Run `npm install` to install local Node.js packages
5. Update `.git/hooks/prepare-commit-msg` with the following code:

```
#!/bin/bash
exec < /dev/tty && node_modules/.bin/cz --hook || true
```

6. Check out a new branch and add your modification.
7. Run shellcheck: `npm run shellcheck` and ensure that it completes with an `exit 0` status. If your changes have introduced some warnings, please try and address them. If you have good reason to ignore some of them, then mention this in the "longer description" portion of your commit.
8. Update `README.md` for your changes.
9. Commit your changes via `git commit`, following the prompts to appropriately categorize your commit.
10. Send a pull request