stackd.io Documentation

Release 0.8.0.dev20170721015655

Clark Perkins

Jul 21, 2017

Installation Guide

stackd.io is a modern cloud deployment and provisioning framework for everyone. Its purpose is to provide a common platform for deploying and configuring hardware on **any** cloud platform. We currently only support AWS EC2, but the driver framework is expandable to support other cloud providers.

stackd.io is built on top of Django. We currently use salt-cloud to build infrastructure, and salt to orchestrate that infrastructure.

CHAPTER 1

Installation Overview

There are two main options for installation:

Amazon AMI

Reading through long install guides and executing each and every command can be time consuming and error prone. If you would rather just run a script to do a lot of this for you, we have a script to build an AMI for you. Keep in mind that the script is somewhat opinionated and won't let you make many decisions (you're free to modify it to suit your needs though!) Here's a list of things it will do:

- Install all of the necessary stuff (MySQL, python, virtualenv, tons of packages, etc)
- Create a `stackdio` virtualenv at `/usr/share/stackdio`
- Install stackdio and its python dependencies
- Install and configure Nginx
- Install and configure supervisord to run gunicorn, celery, and salt-master
- Create an `admin` stackdio user

Check it out here: *Using the Amazon AMI*

Manual Install

If you'd rather have a more custom-fitted installation that fits your needs, check out *Manual Install* instead.

CHAPTER 2

Manual Install

This guide is intended to quickly march you through the steps of installing and running stackd.io and its dependencies. We're not intending to be complete or provide you with everything needed for a production-ready install, we may make some assumptions you don't agree with, and there may be things we missed. If you feel anything is out of the ordinary, a bit confusing, or just plain missing, please *contact us*.

1. A database

stackd.io needs a relational database to store internal information. Since it's built on Django, it inherently supports many different database servers. It is preferred that you use postgres, as we make use of it's json field type. However, you may use a different database like MySQL if you must, but it is beyond the scope of this guide to install it. For more information on Django's database support, see: https://docs.djangoproject.com/en/1.9/ref/databases/

The OS-specific prep of your choice (below) will walk you through installing postgres.

2. OS-specific preparation

> **Warning:** You must follow the steps in one of the following prep guides for the OS you're installing stackd.io in.

Follow one of the individual guides below to prepare your particular environment for stackd.io. Once you finish, come back here and continue on.

Preparing Ubuntu for stackd.io installation

The steps below were written using Ubuntu 16.04 from a Ubuntu-provided AMI on Amazon Web Services (AWS). The exact AMI we used is `ami-29f96d3e`, and you should be able to easily launch an EC2 instance using this AMI from the AWS EC2 Console.

Prerequisites

All of these steps require `root` or `sudo` access. Before installing anything with `apt-get` you should run `apt-get update` first.

Postgres

Note: Please skip this section if you are using a different database or already have a supported database server running elsewhere.

Install Postgres server:

```
sudo apt-get install postgresql
```

Below we'll create a `stackdio` database and grant permissions to the `stackdio` user for that database.

> **WARNING**: we're not focusing on security here, so the default Postgres setup definitely needs to be tweaked, passwords changed, etc., but for a quick-start guide this is out of scope. Please, don't run this as-is in production.

```
sudo -u postgres psql postgres <<EOF
CREATE USER stackdio WITH UNENCRYPTED PASSWORD 'password';
CREATE DATABASE stackdio;
ALTER DATABASE stackdio OWNER to stackdio;
EOF
```

Core requirements

- libpq-dev (the c header files for compiling the python postgres client)
- python-dev (for compiling native python libraries)
- redis-server (for our cache / message queue)
- nginx (for serving static files)

To quickly get up and running, you can run the following to install the required packages.

```
# Install requirements needed to install stackd.io
sudo apt-get install libpq-dev python-dev redis-server nginx
```

Next Steps

You're now finished with the Ubuntu-specific requirements for stackd.io. You can head back over to the *Manual Install* and continue the installation of stackd.io.

Preparing CentOS for stackd.io installation

The steps below were written using CentOS 7 from a CentOS-provided AMI on Amazon Web Services (AWS). The exact AMI we used is `ami-6d1c2007`, and you should be able to easily launch an EC2 instance using this AMI from the AWS Marketplace.

Prerequisites

All of the CentOS-provided AMIs have SELinux and iptables enabled. We disabled both of these to be as straight forward as possible during this guide. SELinux causes issues that are beyond the scope of the guide, and we disabled iptables because we leverage EC2's security groups for firewall access.

iptables

Let's just turn it off for now. Please note, if you're using EC2 or some other cloud provider that has firewall rules enabled by default, you will need to configure the particular firewall rules to gain access to the web server we'll start in the guide.

```
sudo service iptables stop
```

If you'd like to lock down security more, here are the ports that need to be opened up:

22 - SSH 80 - HTTP 443 - HTTPS (optional) 4505 - salt 4506 - salt

SELinux

Getting things working using SELinux could be an entirely separate guide. For our purposes, it's completely out of scope, so we're going to disable it.

Note: You will be required to restart the machine during this step.

```
# Edit /etc/sysconfig/selinux and make sure the line beginning
# with SELINUX looks like:
SELINUX=disabled

# If it was already disabled you can skip the following, however
# if you switched the policy from anything other than 'disabled'
# you need to relabel the filesystem to remove the garbage that
# SELinux has added. This *requires* a restart to take effect.
touch /.autorelabel
reboot

# When the machine is back up, you can confirm SELinux is not
# running
>>> selinuxenabled
>>> echo $?
>>> 1

# If the output is 1 you're good to go.
```

Postgres

Note: Please skip this section if you are using a different database or already have a supported database server running elsewhere.

Install Postgres server:

```
sudo yum install https://download.postgresql.org/pub/repos/yum/9.5/redhat/rhel-7-x86_
↪64/pgdg-centos95-9.5-3.noarch.rpm
sudo yum install postgresql
```

Start Postgres server:

```
sudo service postgresql start
```

Below we'll create a `stackdio` database and grant permissions to the `stackdio` user for that database.

Warning: We're not focusing on security here, so the default postgres setup definitely needs to be tweaked, passwords changed, etc., but for a quick-start guide this is out of scope. Please, don't run this as-is in production :)

```
sudo -u postgres psql postgres <<EOF
CREATE USER stackdio WITH UNENCRYPTED PASSWORD 'password';
CREATE DATABASE stackdio;
ALTER DATABASE stackdio OWNER to stackdio;
EOF
```

Core requirements

- libpq-devel (the c header files for compiling the python postgres client)
- python-devel (for compiling native python libraries)
- redis-server (for our cache / message queue)
- nginx (for serving static files)

To quickly get up and running, you can run the following to install the required packages.

```
# Install requirements needed to install stackd.io
sudo yum install libpq-devel python-devel redis-server nginx
```

Next Steps

You're now finished with the CentOS-specific requirements for stackd.io. You can head back over to the *Manual Install* and continue the installation of stackd.io.

3. Create a virtualenv (Optional)

We recommend that you create a virtualenv to separate your stackd.io dependencies out into a separate environment, but it is completely optional. The following commands utilize virtualenvwrapper. If you'd like to use it, it's documentation is found here: https://virtualenvwrapper.readthedocs.io/en/latest/

Let's create a virtualenv to install stackd.io into:

```
mkvirtualenv stackdio
```

The virtualenv should automatically activate when you create it. If you exit your current shell and come back later, you need to activate the virtualenv again. To do this, virtualenvwrapper gives you the `workon` command:

```
workon stackdio
```

4. Install stackd.io

Note: If you created a virtualenv, double-check that it is activated or else this will probably complain that you don't have permissions to install (because it's trying to install into the global python site-packages directory).

We recommend pulling the latest version from pypi with pip, like this:

```
workon stackdio  # Activate the virtualenv if you created one (optional)
pip install stackdio-server[production,postgres]
```

5. Configuration

After the install, you'll have a `stackdio` command available to interact with much of the platform. First off, we need to configure stackd.io a bit. The `stackdio init` command will prompt you for several pieces of information. If you followed all steps above verbatim, then all defaults may be accepted, but if you deviated from the path you will need to provide the following information:

- an existing user on the system that will run everything (it will default to the `stackdio` user)
- an existing location where stackd.io can store its data (the default is `$HOME/.stackdio/storage` and will be created for you if permissions allow)
- a database URL that points to a running database you have access to (if you're using the postgres install from above, the default `postgresql://stackdio:password@localhost:5432/stackdio` is appropriate)

```
stackdio init
```

Now, let's populate the database with a schema:

```
stackdio manage.py migrate
```

6. stackd.io users

LDAP

stackd.io can easily integrate with an LDAP server. See our *LDAP Guide* for more information on configuring stackd.io to work with LDAP. If you choose to go the LDAP route, you can skip this entire section because users who successfully authenticate and are members of the right groups via LDAP will automatically be created in stackd.io.

Non-LDAP admin user

Admin users in stackd.io have less restriction to various pieces of the platform. For example, only admin users are allowed to create and modify cloud providers and profiles that other users can use to spin up their stacks.

Note: You will need at least one admin user to configure some key areas of the system.

```
stackdio manage.py createsuperuser

# and follow prompts...
```

Non-LDAP regular users

When not using LDAP, the easiest way to create new non-admin users is to use the built-in Django admin interface. First we need the server to be up and running so keep following the steps below and we'll come back to adding users later.

7. Web server configuration

For this guide, we'll use the `stackdio` command to generate the necessary configuration for Nginx to serve our static content as well as proxying the Python app through gunicorn.

To configure Nginx for CentOS:

```
# CENTOS ONLY

# add execute permissions to the user's home directory for static content to serve␣
↪correctly
chmod +x ~/

stackdio config nginx | sudo tee /etc/nginx/conf.d/stackdio.conf > /dev/null

# rename the default server configuration (only files matching *.conf get picked up)
sudo mv /etc/nginx/conf.d/default.conf /etc/nginx/conf.d/default.conf.bak
```

To configure Nginx for Ubuntu:

```
# UBUNTU ONLY
stackdio config nginx | sudo tee /etc/nginx/sites-available/stackdio > /dev/null
sudo ln -s /etc/nginx/sites-available/stackdio /etc/nginx/sites-enabled

# remove the default configuration symlink
sudo rm /etc/nginx/sites-enabled/default
```

After this, generate the static content we'll need to serve:

```
stackdio manage.py collectstatic --noinput
```

and finally, start Nginx:

```
sudo service nginx restart
```

8. Redis, celery, and salt

Start the redis server:

```
sudo service redis-server start
```

For celery and salt-master, we'll be using supervisord. The required packages should already be installed, so we'll just need to configure supervisor and start the services.

```
# generate supervisord configuration that controls gunicorn, celery, and salt-master␣
↪and store it in the .stackdio directory.
stackdio config supervisord > ~/.stackdio/supervisord.conf

# launch supervisord and start the services
supervisord -c ~/.stackdio/supervisord.conf
supervisorctl -c ~/.stackdio/supervisord.conf start all
```

9. Try it out!

At this point, you should have everything configured and running, so fire up a web browser and point it to your hostname and you should see the stackd.io login page. If you're using LDAP, try logging in with a user that is a member of the `stackdio-admin` and `stackdio-user` groups, or login with the admin user you created earlier.

10. Creating additional users

Note: If you're using LDAP, you can skip this step.

The superuser we created earlier will give us admin access to stackd.io, however, you probably want at least one non-superuser. Point your browser to http://<hostname>/__private/admin and use the username and password for the superuser you created earlier. You should be presented with the Django admin interface. To create additional users, follow the steps below.

- click Users
- click Add user in the top right of the page
- set the username and password of the user and click save
- optionally provide first name, last name, and email address of the user and click save

The newly created users will now have access to stackd.io. Test this by logging out and signing in with one of the non-admin users.

CHAPTER 3

Using the Amazon AMI

To make installation easier, we provide an packer build that creates an Amazon Machine Image (AMI). This AMI is built from an Ubuntu 16.04 LTS image, and is HVM based. While our ultimate plan is to provide this AMI on the AWS Marketplace, we don't have this set up yet.

Building an AMI

Note: The build script that runs packer requires you to have python installed locally.

1. Install Packer

If you haven't already, install packer using their documentation here. We recommend using homebrew for the installation if you're using OSX.

2. Accept License

Before building with packer, you must accept the license agreement for the base Ubuntu AMI: https://aws.amazon.com/marketplace/pp/B01JBL2M0O

3. Clone Repository

```
git clone https://github.com/stackdio/stackdio.git
cd stackdio
```

4. Export AWS Credentials

Ensure packer knows about your aws credentials:

```
export AWS_ACCESS_KEY='<YOUR_ACCESS_KEY>'
export AWS_SECRET_KEY='<YOUR_SECRET_KEY>'
```

5. Run the packer build

Finally, run the packer build, where `<version>` is the version you want to build:

```
./packer/build.py <version>
```

After a few minutes, you should have a usable AMI.

Using the AMI

After you've built the AMI, you can launch an instance from it. Once the instance is running, you can navigate to `http://<instance-ip>/` and login using the following credentials:

```
username: admin
password: stackdio
```

We recommend changing this password immediately after logging in the first time.

CHAPTER 4

LDAP Guide

django-auth-ldap

Under the hood, we use django-auth-ldap for all our interaction with an LDAP server. It's a very useful library that allows us to integrate with LDAP in just a few lines of configuration.

LDAP and stackd.io

If you'd like to integrate with LDAP, you must install the `ldap` extra and add an `ldap` section to the `stackdio.yaml` file (usually located at `/etc/stackdio/stackdio.yaml` or `$HOME/.stackdio/stackdio.yaml`).

To install the `ldap` extra:

```
pip install stackdio-server[ldap]
```

An example configuration is shown below.

The example contains most of the relevant bits you may want to use, but we defer anything deeper to the django-auth-ldap documentation.

Any key / value pair that appears in the `ldap` section of the config file will be uppercased and appended to `AUTH_LDAP_` before being placed in the config file. For example, having `server_uri: ldaps://example.com` in your stackdio config file will translate to the django-auth-ldap config of `AUTH_LDAP_SERVER_URI = 'ldaps://example.com'`.

Note: If you're using the packer-built AMI described in *Using the Amazon AMI*, your stackdio.yaml is located at `/etc/stackdio/stackdio.yaml`.

```
##
# Optional LDAP configurations
# To be appended to your stackdio.yaml file.
```

```
ldap:
  # This must be set to true for anything below to take effect
  enabled: true

  # The url of your server (can be a comma-separated list of servers)
  server_uri: ldaps://ldap.example.com

  # Should we bind to LDAP as the user trying to login?
  bind_as_authenticating_user: false

  # if bind_as_authenticating_user is false, provide the bind user credentials
  bind_dn: 'uid=binduser,ou=People,dc=example,dc=com'
  bind_password: my_password

  # Should groups in ldap be mirrored to django groups in the database?
  mirror_groups: true

  # Deny login if a valid LDAP user isn't in this list of groups
  #require_group:
  #  - 'cn=mygroup,ou=People,dc=example,dc=com'

  # The search parameters for users.
  #   The result of a search using these parameters should return EXACTLY ONE
  #   user for this to work properly.
  user_search_base: 'ou=People,dc=example,dc=com'
  user_search_scope: SCOPE_SUBTREE
  user_search_filter: '(&(objectClass=Person)(uid=%(user)s))'

  # The search parameters for groups.
  #   The result of a search using these parameters should return an exhaustive list
  #   of groups you would like to make available.
  group_search_base: 'ou=Group,dc=eample,dc=com'
  group_search_scope: SCOPE_SUBTREE
  #group_search_filter: '(objectClass=*)'

  # The type of the ldap groups
  group_type: GroupOfNamesType

  # A map from django user object attributes to the associated attributes in LDAP
  user_attr_map:
    first_name: givenName
    last_name: sn
    email: mail

  # A map that associates boolean user flags to LDAP groups
  # i.e. if an LDAP user is in the specified LDAP group, the specified user flag is␣
↪set to 'True'
  user_flags_by_group:
    is_superuser: 'cn=admin,ou=Group,dc=example,dc=com'
    is_staff: 'cn=admin,ou=Group,dc=example,dc=com'

  # Any connection options you need
  connection_options:
    OPT_X_TLS_REQUIRE_CERT: OPT_X_TLS_NEVER
    OPT_X_TLS_NEWCTX: 0
```

CHAPTER 5

Webserver Guide

This guide will help you quickly get the web portion of stackd.io running behind Nginx. **Seeing as stackd.io is just a wsgi app, you can also run it behind apache using mod_wsgi if you'd like, but that is beyond the scope of this guide.** You should've already worked through this manual install guide before running through the steps below. As with this guide, our focus is not entirely on building out a production-ready system, but merely helping you quickly get a system stood up to become familiar with stackd.io. Once you understand how it works, then we can start hardening the system for production use.

Common Steps

To do some of the steps below you will need to have already installed stackdio and be in the virtual environment. To make sure you're in the virtualenv if you created on when installing (optional):

```
workon stackdio
```

Nginx needs a place to store logs and some static files to serve up. This step should be run before proceeding with configuring Nginx.

```
# And tell Django to collect its static files into a common directory for the␣
↪webserver to serve up
stackdio manage.py collectstatic --noinput
```

In our configuration, Nginx will be used to serve static files and as a proxy to send requests down to the Django application running via gunicorn on port 8000. The configuration we'll generate is useful to use a quick start mechanism to get you up and running behind Nginx/gunicorn very quickly.

CentOS Installation

Install required packaged, generate and write configuration file, and restart server:

```
sudo yum install nginx

stackdio config nginx | sudo tee /etc/nginx/conf.d/stackdio.conf > /dev/null

# rename the default server configuration
sudo mv /etc/nginx/conf.d/default.conf /etc/nginx/conf.d/default.conf.bak

sudo service nginx restart
```

Ubuntu Installation

```
sudo apt-get install nginx

stackdio config nginx | sudo tee /etc/nginx/sites-enabled/stackdio.conf > /dev/null

# remove the default configuration symlink
sudo rm /etc/nginx/sites-enabled/default

sudo service nginx restart
```

CHAPTER 6

Contact Information

- Static site - http://stackd.io
- IRC - #stackdio on Freenode
- Twitter - @stackdio or #stackdio
- Github - https://github.com/stackdio/stackdio

Filing issues and feature requests

We want stackd.io to be solid, but there's always going to be issues to fix and new features to enhance the product. If you find any problems or would like to request a particular feature, feel free to head over to stackd.io's issue tracker. Below are some general guidelines.

Feature Requests

We definitely want this project to be better in every way, but sometimes we just can't see the forest because of those trees. If you think of something that would make stackd.io be more intuitive, easier, faster, or efficient, we encourage you to file a feature request, but please keep the following in mind:

1. Double-check that the feature hasn't already made its way into the tracker and/or been completed
2. After filing it, please be patient – sometimes it might take a while for things to get prioritized, so don't be bummed that it's not available tomorrow ;)
3. As always, the quickest way to get your shiny new feature is to get your hands dirty and implement it yourself!

Bug Reports

If you think you've found a bug, here's what we ask of you:

1. Check the issue tracker and/or search for your issue to make sure we aren't already tracking the problem
2. See if you can reproduce the issue on the develop branch of the project to confirm we haven't already fixed the problem
3. Try to be as descriptive as possible when filing your issues. We'd like to know the operating system you're on, any stack traces you've seen in the logs, a good account of the steps required to reproduce the problem along with what you feel is the expected outcome. Providing more information is almost always better so we can quickly identify and fix the issue.
4. Be patient :)
5. Optionally, it would be fantastic if you could help us out and fix the problem yourself... if you're brave enough, see the *Contributor Guidelines* on contributing to the project.

Usage Questions

Usage questions are best handled in one of the following ways:

- Asking a question on the IRC channel
- Emailing the contributors, especially those who have been recently active on the project.

Long term development comments

Long term development tasks will usually have a branch opened on the main repository, along with a pull request back into master. This is the place to make comments on the development cycle for that feature.

CHAPTER 7

Contributor Guidelines

We're hopeful that you'll find value in using stackd.io along with your existing tools to manage your infrastructure. We're also hopeful that you'll find ways to help contribute. Either through finding and reporting bugs, providing new features, or even getting your hands dirty and contributing some code or docs. However you feel comfortable contributing, we offer a few helpful guidelines to make it that much easier.

> Note that stackd.io is built on SaltStack, and we believe that its community will find stackd.io useful. As such, we're trying to stay close to the conventions and guidelines they've adopted to make it easier for folks in that community to help out – and we've borrowed from the SaltStack Development guide :)

Filing issues and feature requests

The process for filing issues and feature requests is described in the *Contact Information* page.

Contributing Code

Since we're using Github, the recommended workflow for fixing bugs, adding features, or modifying documentation is to fork and submit pull requests. The process is pretty straightforward, but if you're unfamiliar with Github, take some time to browse through Github's Help.

In a nutshell, we'll need you to:

- fork the stackd.io project into your personal account [Tutorial]
- make the necessary changes to the code/docs and issue a pull request. [Tutorial]
- keep your local fork in sync with the parent stackd.io repository to minimize the chance of merge conflicts. [Tutorial]
- and, if you're working on multiple things or your changes are going to be somewhat large, it's generally recommended to create a branch for each piece of work you're doing. [Tutorial]

> NOTE: SaltStack has a great guide on how to work within their project and it mostly applies to stackd.io as well

Pull request guidelines

TODO

CLA

Contribution to stackd.io requires a CLA before pull requests will be merged. This is currently handled manually by the repo admins, but may be handled by a bot in the future.

Branch naming

The branch name that the pull request originates from should start with either `feature/` or `bugfix/`, depending on its contents. The rest of the branch name should describe the contents of the patch, preferably by being an Issue#. Issue#'s are required for `bugfix/` branches.

Code style and quality

TODO

PEP8 compatibility

All pull requests must meet PEP8 / pylint compatibility or a good reason not to follow it.

Tests

Pull requests will be easier to review and understand if they contain automated tests for the functionality changed. As such, pull requests with tests are more likely to be accepted more quickly.