spin-docker Documentation

Release 0.1.0

Andrew T. Baker

April 15, 2015

Contents

1 Example use **3**

2 Contents **5**

2.1 Getting started 5
2.2 API documentation 6
2.3 Activity monitoring 10
2.4 Administering spin-docker 11
2.5 Creating your own spin-docker images 13
2.6 Spin-docker technical architecture 14
2.7 Developing for spin-docker 14

Spin-docker is a lightweight platform-as-a-service that runs on a RESTful API. Designed with simplicity in mind, spin-docker provides enough features to help you stand up your own Docker-powered PaaS in minutes.

Spin-docker was created by Andrew T. Baker as a side project this winter. It might not be the best Docker-based PaaS out there, but building it was a great learning experience. Check out my blog post about it: http://andrewtorkbaker.com/what-makes-a-good-side-project

Spin-docker is open source under the MIT license.

CHAPTER 1

Example use

Spin-docker uses a RESTful API so you can easily start and stop docker containers with an HTTP request. Start a PostgreSQL database with:

```
$ curl http://localhost:8080/v1/containers -X POST -u admin:spindocker -d "image=atbaker/sd-postgres"
```

And you'll receive back:

```
{
  "status": "running",
  "name": "/desperate_poincare",
  "active": "0",
  "ssh_port": "49153",
  "image": "atbaker/sd-postgres",
  "container_id": "0145c3630dccc5529ecd3a61d0e7f04236ef3e7a91d06b2985f50e9aa4dc266d",
  "uri": "/v1/containers/0145c3630dccc5529ecd3a61d0e7f04236ef3e7a91d06b2985f50e9aa4dc266d",
  "app_port": "49154"
}
```

The `atbaker/sd-postgres` docker image runs a PostgreSQL database using Phusion's base docker image. Spin-docker images report their activity back to spin-docker so it can stop idle containers, but spin-docker is fully compatible with all docker images.

You can SSH into the container using Phusion's insecure key (included in the spin-docker root directory):

```
$ ssh -i insecure_key root@127.0.0.1 -p 49153
$ su postgres -c 'psql'
```

And of course you (or an app) can connect to the PostgreSQL database directly if you have the PostgreSQL client installed:

```
$ psql -U postgres -h 127.0.0.1 -p 49154
```

And that's it! Here are a few ideas for how spin-docker could help your organization:

- Use the PostgreSQL, MongoDB, and Django images available now to provide on-demand databases and web-servers for developers and data scientists
- Create a docker image for your own app and use spin-docker to provide on-demand demo servers for your teams
- Jump start your next training or tutorial session by putting a sample environment in a docker image and using spin-docker to quickly expose those environments to your students

Or just play with spin-docker to learn more about how docker works! Using Vagrant and Ansible, you can deploy your own spin-docker server anywhere in minutes. Read *Getting started with spin-docker* for more information.

CHAPTER 2

Contents

2.1 Getting started

Standing up your own spin-docker server locally or in the cloud takes minutes. Setup also includes the PostgreSQL, MongoDB, and Django spin-docker images.

2.1.1 Grabbing the source

First download the spin-docker source code - either by cloning the repository on GitHub or downloading and extracting the current version.

Whether you want to run spin-docker in a local virtual machine or on a server, you need Ansible to provision it. Installing Ansible is easy. You can download the source, get it from apt or yum, or just use pip:

```
$ sudo pip install ansible
```

2.1.2 Running locally with Vagrant

The easiest way to run spin-docker is using a local virtual machine with Vagrant.

Install Vagrant using the instructions on their site. Like those instructions, this guide will also assume you're using Virtualbox as your provider.

Before you start your virtual server, add this environment variable to your session if you want your spin-docker containers to be accessible outside of your virtual machine:

```
$ export FORWARD_DOCKER_PORTS='True'
```

Now `cd` into your spin-docker code and start the vagrant box with `vagrant up`. Vagrant will download Phusion's docker-ready Ubuntu 12.04 server box and run an ansible playbook to install and configure spin-docker.

Note: If you get a vagrant error when starting your box saying that some forwarded ports are unavailable, you will need to quit the application that's using them. I've noticed that Chrome in particular can sit on ports around 49200. Quitting Chrome, starting the vagrant box, and then starting Chrome again usually works for me.

Once the vagrant box is running, you can use `vagrant ssh` to get in and poke around. If you're ready to learn more about spin-docker, check out the *API documentation* or *Administering spin-docker*.

2.1.3 Deploying to a server using Ansible

Spin-docker comes with a complete Ansible playbook that can provision any Ubuntu 12.04 server to run spin-docker. DigitalOcean and Amazon EC2 are always good choices.

Once you have an Ubuntu 12.04 server ready, edit the hosts file in the ansible_playbook subdirectory and add the correct hostname, SSH user, and SSH key for your server.

```
ec2-54-137-143-5.compute-1.amazonaws.com vagrant_box=False ansible_ssh_user=ubuntu ansible_ssh_privat
```

Take care not to accidentally delete the `vagrant_box=False` variable.

Now you're ready to run the Ansible play to install and configure spin-docker:

```
$ ansible-playbook -i hosts spin_docker.yml
```

Ansible will upgrade the linux kernel, install docker, and install spin-docker with its dependencies. It will also pull down the spin-docker PostgreSQL, MongoDB, and Django images from the docker registry so you can start using the server immediately.

Note: Keep an eye on your server resources. If you're running spin-docker on an AWS micro instance, for example, you may experience issues running more than one container. Using the ephemeral instance storage as additional swap space can help: http://serverfault.com/questions/218750/why-dont-ec2-ubuntu-images-have-swap

---

When you're ready to learn more about spin-docker, check out the *API documentation* or *Administering spin-docker*.

2.1.4 Disabling container monitoring

By default spin-docker is configured to stop idle containers after two and a half hours. You may want to disable this feature if you don't want to adapt your containers to report their activity to spin-docker.

To do this, edit the spin_docker.yaml file in the ansible_playbook directory. Set the `sd_disable_timeouts` variable to False:

```
sd_disable_timeouts: False
```

Then reprovision your spin-docker server. If you're using vagrant, run `vagrant provision`. If you're running your own server run `ansible-playbook -i hosts spin_docker.yml`.

Learn more about how spin-docker monitors container activity by reading *Activity monitoring*.

2.2 API documentation

Spin-docker is built using the excellent Flask-RESTful Flask extension. Though the API is basic right now, it can be easily extended to keep pace as docker evolves.

The spin-docker API currently has just two endpoints - *Images (/v1/images)* and *Containers (/v1/containers)*:

2.2.1 Authentication

Spin-docker currently uses basic HTTP authentication for all requests. The username is `admin` and the default password is `spindocker`. All requests to spin-docker need to use these credentials.

If you would like to change the password, edit the `sd_password` setting in asible_playbook/spin_docker.yaml.

```
sd_password: 'spindocker'
```

2.2.2 Images (/v1/images)

The images endpoint exposes the tagged docker images available on a server. It currently only supports GET requests.

A request to `/v1/images` yields:

```
[
  "atbaker/sd-mongo:latest",
  "atbaker/sd-postgres:latest",
  "atbaker/sd-django:latest",
  "phusion/baseimage:0.9.8"
]
```

This is useful for checking if an image exists on your server before attempting to create a container with it.

2.2.3 Containers (/v1/containers)

The containers endpoint manages docker containers running on the server. It supports GET, POST, and PATCH requests.

Querying containers

A GET request to `/v1/containers` yields a list of all containers spin-docker has ever started on the server:

```
[
  {
    "status": "running",
    "name": "/hungry_bell",
    "active": "0",
    "ssh_port": "49155",
    "image": "atbaker/sd-postgres",
    "container_id": "2cd8a1b037d8752a481b373f225ac72ced9ec89ba784b7868dbb784072358f0e",
    "uri": "/v1/containers/2cd8a1b037d8752a481b373f225ac72ced9ec89ba784b7868dbb784072358f0e",
    "app_port": "49156"
  },
  {
    "status": "running",
    "name": "/hopeful_ritchie",
    "active": "1",
    "ssh_port": "49157",
    "image": "atbaker/sd-postgres",
    "container_id": "ba87a9911beb99b9fab64c387267235bd157c7ab69f9eb15695ebbfffe393ccc",
    "uri": "/v1/containers/ba87a9911beb99b9fab64c387267235bd157c7ab69f9eb15695ebbfffe393ccc",
    "app_port": "49158"
  },
  {
    "status": "stopped",
    "name": "/desperate_poincare",
    "active": "0",
    "ssh_port": "",
    "image": "atbaker/sd-postgres",
    "container_id": "0145c3630dccc5529ecd3a61d0e7f04236ef3e7a91d06b2985f50e9aa4dc266d",
    "uri": "/v1/containers/0145c3630dccc5529ecd3a61d0e7f04236ef3e7a91d06b2985f50e9aa4dc266d",
```

```
    "app_port": ""
  }
]
```

Note that in this example, the last container is stopped and thus is not mapped to any ports.

To get information about just one container, make a GET request to that container's URI. Following our example, hitting this URL:

```
/v1/containers/2cd8a1b037d8752a481b373f225ac72ced9ec89ba784b7868dbb784072358f0e
```

yields

```
{
  "status": "running",
  "name": "/hungry_bell",
  "active": "0",
  "ssh_port": "49155",
  "image": "atbaker/sd-postgres",
  "container_id": "2cd8a1b037d8752a481b373f225ac72ced9ec89ba784b7868dbb784072358f0e",
  "uri": "/v1/containers/2cd8a1b037d8752a481b373f225ac72ced9ec89ba784b7868dbb784072358f0e",
  "app_port": "49156"
}
```

Creating containers

A POST request to `/v1/containers` creates a new container. Only one data field is required: the name of the docker image to start.

Spin-docker determines which ports to forward by referencing the ports exposed on the docker image. If port 22 is exposed, spin-docker will map it to the ssh_port field. If any other port is exposed, spin-docker will map it to the app_port field.

So a POST request to `/v1/containers` with this data:

```
{
  "image": "atbaker/sd-postgres"
}
```

Or using curl:

```
$ curl http://localhost:8080/v1/containers -X POST -u admin:spindocker -d "image=atbaker/sd-postgres"
```

Will create a new container and return its details:

```
{
  "status": "running",
  "name": "/hungry_bell",
  "active": "0",
  "ssh_port": "49155",
  "image": "atbaker/sd-postgres",
  "container_id": "2cd8a1b037d8752a481b373f225ac72ced9ec89ba784b7868dbb784072358f0e",
  "uri": "/v1/containers/2cd8a1b037d8752a481b373f225ac72ced9ec89ba784b7868dbb784072358f0e",
  "app_port": "49156"
}
```

By default spin-docker will stop containers that haven't reported any activity to the `/check_in` URL in two and a half hours. See *Activity monitoring* for more details.

Starting and stopping containers

Once created, containers can be started or stopped manually through PATCH requests to their specific URIs.

Following our example, a PATCH request to `/v1/containers/2cd8a1b037d8752a481b373f225ac72ced9ec89ba784b7` with this data:

```
{
  "status": "stopped"
}
```

Will begin stopping that container and return:

```
{
  "status": "stopping",
  "name": "/hungry_bell",
  "active": "0",
  "ssh_port": "49155",
  "image": "atbaker/sd-postgres",
  "container_id": "2cd8a1b037d8752a481b373f225ac72ced9ec89ba784b7868dbb784072358f0e",
  "uri": "/v1/containers/2cd8a1b037d8752a481b373f225ac72ced9ec89ba784b7868dbb784072358f0e",
  "app_port": "49156"
}
```

Stopping a container may take up to 10 seconds, so spin-docker does this asynchronously and returns the `stopping` status instead to acknowledge your stop order. Doing a GET request to this container's URI in a few seconds will show the container at rest:

```
{
  "status": "stopped",
  "name": "/hungry_bell",
  "active": "0",
  "ssh_port": "",
  "image": "atbaker/sd-postgres",
  "container_id": "2cd8a1b037d8752a481b373f225ac72ced9ec89ba784b7868dbb784072358f0e",
  "uri": "/v1/containers/2cd8a1b037d8752a481b373f225ac72ced9ec89ba784b7868dbb784072358f0e",
  "app_port": ""
}
```

Starting the container again is easy. Make another PATCH request to the same URI, this time with:

```
{
  "status": "running"
}
```

And spin-docker will start the container immediately, giving you new ports you can use to connect to it:

```
{
  "status": "running",
  "name": "/hungry_bell",
  "active": "0",
  "ssh_port": "49157",
  "image": "atbaker/sd-postgres",
  "container_id": "2cd8a1b037d8752a481b373f225ac72ced9ec89ba784b7868dbb784072358f0e",
  "uri": "/v1/containers/2cd8a1b037d8752a481b373f225ac72ced9ec89ba784b7868dbb784072358f0e",
  "app_port": "49158"
}
```

Deleting containers

When you're ready to completely remove a container from your server, send a DELETE request to that container's URI. Using curl to continue our example:

```
$ curl http://localhost:8080/v1/containers/2cd8a1b037d8752a481b373f225ac72ced9ec89ba784b7868dbb78407
```

Spin-docker will stop the container if it is running, delete it, and return an empty 204 response.

2.2.4 Check-in (/v1/check-in)

The check-in URL isn't an endpoint - it's just a URL that containers can use to report their current activity back to spin-docker. This is covered in more detail *Creating your own spin-docker images*, but the request itself is simple.

To report on your container's activity, have it regularly POST to the `/v1/check-in` URL with one data field: `active-connections`. Authentication is not required for this URL.

You will also need to determine the appropriate IP address to POST to. To reach spin-docker from within a container, always send your check-ins to the container's default gateway IP address. You can determine the default gateway with `ip route show`:

```
default via 172.17.42.1 dev eth0
172.17.0.0/16 dev eth0  proto kernel  scope link  src 172.17.0.2
```

In this case, the container's default gateway is `172.17.42.1`. If you wanted to use curl to report this container's activity, you would use this command:

```
$ curl http://172.17.42.1/v1/check-in -X POST -d "active-connections=1"
```

But writing a script and using a cron job to report container activity is more practical. Read more about that in *Creating your own spin-docker images*.

Or to learn more about administering spin-docker servers, continue to *Administering spin-docker*.

2.3 Activity monitoring

Spin-docker's activity monitoring feature automatically stops containers that aren't reporting any active connections. This helps you reduce the computing resources you need to run your PaaS by only running containers that are actually in use.

If you would prefer to disable this feature and instead stop your containers manually with HTTP calls to the *containers endpoint*, see the *Disabling container monitoring* section of the quickstart.

2.3.1 Container self-reporting

To keep spin-docker compatible with as many applications as possible it relies on the containers themselves to report their activity back to spin-docker, rather than spin-docker actively monitoring the containers' activity itself.

This self-reporting is done by making a POST request to a container's default gateway at the `/v1/check-in` URL. Read `/check_in` to learn more about the check-in URL.

A container's activity is tracked in its `active` field. When this field has any value other than 0, the container is considered active.

To learn more about how to write automated scripts to monitor your containers, read *Creating your own spin-docker images*.

2.3.2 Stopping idle containers (example workflow)

As each container is started, an asynchronous celery task is added to check on the container's activity. This task is scheduled to occur after the container has been running for the number of seconds listed in the `sd_initial_timeout_interval` setting.

When the initial timeout interval has elapsed and the first activity check occurs, spin-docker references the `active` field for the container.

If the `active` field is 0, then the container is idle. Spin-docker will schedule one last check on the container using the `sd_timeout_interval` setting. If the container is still inactive after this second check, spin-docker will stop (but not delete) the container.

If during any check the `active` field is not 0, then spin-docker will schedule another activity check to occur after the number of seconds in the `sd_timeout_interval` setting. This will continue indefinitely until the container is found to be idle.

Once stopped, a container can always be started again with a PATCH request to the *containers endpoint*. Keep in mind that the container will likely have different ports exposed when it starts up again.

Read *Administering spin-docker* to continue learning about how to run your own spin-docker server.

2.4 Administering spin-docker

2.4.1 Configuration

Spin-docker uses only a few configurable settings. They are all in the Ansible playbook in the spin_docker.yml file:

```
# Spin-docker configuration settings
sd_environment: 'development'
sd_password: 'spindocker'
sd_disable_timeouts: False
sd_initial_timeout_interval: 7200
sd_timeout_interval: 1800
```

sd_environment - Set to `production` to disable spin-docker's debug mode.

sd_password - The password used for basic HTTP authentication for the `admin` user.

sd_disable_timeouts - When set to `True` spin-docker will never stop a container with no active connections. See *Activity monitoring* for more details.

sd_initial_timeout_interval - How many seconds spin-docker should wait before the first time it checks a container's activity. Use this setting to control the minumum amount of time a container will stay running.

sd_timeout_interval - How many seconds spin-docker should wait between each activity check. Increasing this value will extend the amount of time a container stays running without activity.

After changing these settings, you will need to provision your spin-docker server again (using `vagrant provision` or `ansible-playbook -i hosts spin_docker.yml`) to apply them.

2.4.2 Adding new images

Spin-docker does not currently support adding new images to the server. Until this feature exists (pull requests welcome!) you will need to add new images manually using the docker command line interface.

Use docker pull to get new images from the docker index. Use docker build to compile your own Dockerfiles.

Spin-docker is compatible with all docker images, but it can only monitor the activity of images that report their activity back to spin-docker at the `/v1/check-in` URL.

To see the list of publicly available spin-docker images, visit the spin-docker namespace on the docker registry: https://index.docker.io/u/atbaker/

> **Warning:** Unless you plan to only run spin-docker images on your spin-docker server, you should *disable container monitoring* in your configuration settings. Otherwise, all containers you create will be stopped after two and a half hours whether they are in use or not.

Learn how to create your own spin-docker compatible images in *Creating your own spin-docker images*.

2.4.3 Auditing containers

On rare occasions spin-docker's information about containers can get out of sync with docker's. This usually only happens when you have been messing with docker on the machine manually.

Spin-docker can perform an audit of all containers it is aware of if you add an `audit=true` parameter on a GET request to `/v1/containers`. You can do this using curl:

```
curl http://localhost:8080/v1/containers -X GET -u admin:spindocker -d "audit=true"
```

Before spin-docker returns the full list of containers it will check each one and update its fields. It will also reset its `active` field to 0 to ensure the container properly reports any activity.

Spin-docker automatically audits containers each time it starts, so it will automatically pick up any container changes that occur because of a server reboot.

2.4.4 SSH known hosts issues

If you work on spin-docker long enough, docker will eventually reuse a port for a new container which it had previously used for another container. Because each container generates its own SSH host keys, your SSH client may refuse to connect you to an instance because it suspects foul play:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
2e:3e:ce:d9:10:29:d2:34:48:ea:0d:ba:6b:21:f2:be.
Please contact your system administrator.
Add correct host key in /Users/atbaker/.ssh/known_hosts to get rid of this message.
Offending RSA key in /Users/atbaker/.ssh/known_hosts:16
RSA host key for [localhost]:49153 has changed and you have requested strict checking.
Host key verification failed.
```

The easiest way to fix this is to edit your `known_hosts` file and remove all entries for spin-docker containers you have connected to in the past.

Now that you know how to administer a spin-docker server, read *Creating your own spin-docker images* to learn how to make your own apps fully compatible with spin-docker.

2.5 Creating your own spin-docker images

Spin-docker is compatible with all docker images, but it can only use activity monitoring if it runs containers that are configured to report their activity back to spin-docker using the `/v1/check-in` URL.

To see the list of publicly available spin-docker images, visit the spin-docker namespace on the docker registry: https://index.docker.io/u/atbaker/. Someone may have created an image that meets your needs already.

2.5.1 Image requirements

A docker image must satisfy only three requirements to be fully compatible with spin-docker:

- Expose port 22 for SSH access
- Expose any other port for use by an application
- Regularly send HTTP POST requests to the server's `/v1/check-in` URL reporting its activity

How you meet these requirements is up to you. Sample Dockerfiles are available on GitHub to help guide you:

- **Template:** https://github.com/atbaker/sd-base
- **PostgreSQL:** https://github.com/atbaker/sd-postgres
- **MongoDB:** https://github.com/atbaker/sd-mongo
- **Django + Gunicorn:** https://github.com/atbaker/sd-django

2.5.2 Using the Phusion base image

The easiest way to build a custom spin-docker image is to start with Phusion's baseimage-docker.

This image is based on Ubuntu 12.04 LTS and includes SSH, cron, and other helpful tools while staying lean. It uses runit instead of Upstart to provide a correct init process for your containers without adding much overhead. All the spin-docker images use it.

Follow the instructions starting at Using baseimage-docker as a base image to build your own spin-docker compatible Dockerfile.

Don't overlook the section on adding additional daemons to configure your own app to work with runit.

> **Warning:** Note that all the spin-docker examples use Phusion's insecure key. The insecure key is enabled by the `RUN /usr/sbin/enable_insecure_key` line in the Dockerfiles.
> **This key should not be used in production** because the public and private keys are available to all. You should instead create your own SSH keys to load into containers and distribute to users who will need SSH access.

2.5.3 Using a provisioning tool

Dockerfiles are great for building simple images, but complex apps might be better served by a provisioning tool like Ansible or Puppet - especially if your app already uses one of these tools.

If you choose this route you should still use the `sd-base` Dockerfile and the Phusion base image to start a container with SSH, cron, and runit support. You can then start a container with that base image, run your provisioning tool, and then use `docker commit` to create a docker image out of that container's current state.

One drawback of this approach is that you must use the `--run="{ ... }"` option with `docker commit` to specify the correct init command (`"Cmd": ["/sbin/my_init"]`) and which ports to expose (`"PortSpecs": ["22", "80"]`).

Read up on docker's commit command for more information.

2.5.4 Reporting container activity

Your image can use any means you like to send HTTP POST requests back to the spin-docker server. See `/check_in` for details on how to create that request.

The easiest way to report on your container's activity is to write a small script which gathers the relevant information from your container and POSTs it back to spin-docker. The definition of activity is up to you - spin-docker considers any POST with non-zero `active` field to be from an active container.

The Phusion baseimage comes with cron installed, so once you have your script just add it to the crontab to keep it running regularly.

See the sd-postgres script or the sd-django script for a complete example.

2.5.5 Keeping it slim: container vs. virtual machine

Running multiple processes inside a docker container is a controversial subject. Michael Crosby of Docker has said that running an init process inside a container is almost always a mistake. Docker CEO Solomon Hykes has said that docker images like Phusions are legitimate, though the more overhead you put in one image the less efficient it becomes. One of the Dockerfile examples in the Docker documentation uses supervisord to launch multiple processes in a container.

For now, the moral of the story is to keep your docker images as slim as possible. If you want to use spin-docker's activity monitoring, your containers will by necessity be heavier than if they ran a single process. Spin-docker is fully compatible with all docker images, so you are not limited if you choose run only a single process in your images.

2.6 Spin-docker technical architecture

Spin-docker is a Python application built with the following components:

- Docker for lightweight virtualization
- Flask-RESTful (and Flask) for easy API development
- Celery for asynchronous tasks
- Redis for a fast and persistent data store

Spin-docker is deployed with the gunicorn WSGI server and Nginx reverse proxy.

It is provisioned with Ansible and virtualized locally with Vagrant.

2.7 Developing for spin-docker

Contributing to spin-docker development is easy! Fork the GitHub repository and you're halfway there!

2.7.1 Development environment setup

The easiest way to set up a spin-docker development environment is to follow the instructions in *Running locally with Vagrant*.

Once you have provisioned your vagrant box, you can run the Flask development server with these steps:

1. `vagrant ssh` into the box
2. Switch to the root user (necessary to start the development server on port 80, which is forwarded by Vagrant)
3. Stop the nginx, gunicorn, and celery services:

```
$ service nginx stop
$ service gunicorn stop
$ service celery stop
```

4. `cd` to the `/var/www`
5. Activate the spin-docker virutal environment: `source venv/bin/activate`
6. `cd` into the `spin-docker` directory
7. Apply the spin-docker environment variables with `source .env`
8. Start the Flask development server: `python runserver.py`

Activity monitoring, stopping, and deleting containers are managed by Celery. You can start the Celery worker in a separate terminal session (but still as root, because docker is currently configured only for root):

1. `cd` to the `/var/www`
2. Activate the spin-docker virutal environment: `source venv/bin/activate`
3. `cd` into the `spin-docker` directory
4. Apply the spin-docker environment variables with `source .env`
5. Run the command `celery -A spindocker.tasks.celery worker -l info`

2.7.2 Running tests

To run the spin-docker tests and view current code coverage, follow these steps (again as root):

1. `cd` to the `/var/www`
2. Activate the spin-docker virutal environment: `source venv/bin/activate`
3. `cd` into the `spin-docker` directory
4. Apply the spin-docker environment variables with `source .env`
5. Install the test requirements in you haven't already: `pip install -r requirements/test.txt`
6. Run the command `coverage run --source=spindocker tests.py`

After the tests run, you can view current coverage with `coverage report` and generate an HTML report with `coverage html`.

2.7.3 Building the docs

If you make changes to the spin-docker documentation, you can check your work by building the docs with `make html` in the `docs` directory.