StackStrap Documentation

Release 0.2.2

Evan Borgstrom

April 16, 2015

A tool that uses vagrant + salt to make development more awesome.

CHAPTER 1

Overview

Let's face it, writing code is awesome, but writing code in a team without a great strategy and a solid devops team supporting you can get really frustrating when trying to manage development environments within the team.

Vagrant does an amazing job of automating the task of bringing up virtual machines for development, but it doesn't do too much in the way of configuring the operating system. Enter it's provisioners, these allow you the ability to configure the system after Vagrant creates it but knowing how you should be configuring the system is a whole other question.

StackStrap aims to ease this situation by utilizing the Salt provisioner in Vagrant along with a community repository of Salt states that allow you to quickly and reliably create development environments using our simple macros. These macros are coupled with a Jinja parsed Template that lays out your ideal project file structure. This way you can easily strap your favourite framework in just the way your team likes to use it. You can have a bootable development ready environment to play with in minutes.

CHAPTER 2

Here be dragons

January 1st, 2014: New year, new approach. We’ve done a complete 180 degree turn on the approach for how stackstrap works. If you used it prior to Jan 1 2014 please make sure to read the docs again to familiarize yourself with the changes. They are drastic.

CHAPTER 3

Getting Help

You can find us in #stackstrap on freenode if you want to chat or need help.

There is also a Google Group (stackstrap@googlegroups.com).

CHAPTER 4

Contents

4.1 Installing StackStrap

StackStrap is a command line tool written in Python. It will install on any Mac OS X or Linux computer. Windows usage is definitely possible but has not been documented yet.

4.1.1 Installing Vagrant

Since StackStrap uses Vagrant for the management of the virtual machine images that will be used for development you will need to have Vagrant installed on your workstation. See the Vagrant Installation Documentation for instructions.

4.1.2 Installing StackStrap

It's very easy to install StackStrap as it's a standard Python package that's available on the cheeseshop.

The documentation will install it globally using `easy_install` but you can certainly install it using `pip` to a virtualenv if you desire.

Easy Installation

To install StackStrap we're going to be using the `easy_install` command as root so that StackStrap will be globally available.

Open up a terminal and run the following command:

```
sudo easy_install stackstrap
```

This will install StackStrap and all of it's dependencies. It will place a command named `stackstrap` in your `$PATH` so that you can use it from any where on your system.

The better way - pip

The *Easy Installation* method is a very fast way to get up and running with StackStrap, but using `easy_install` is not the best way to install and manage packages on your system. You should be using `pip` to do this. For background on why check out this Stack Overflow question: Why use pip over easy_install?

However, unlike `easy_install` you must first install `pip` before it can be used.

Mac OS X

On OS X you first need to have the developer command line tools installed. Then you need to decide between between two different package managers.

Homebrew is a newer package manager that integrates into your base OS X system libraries and allows for quick & easy installation of packages into a specific directory, named `Cellar`, so that it can easily be removed if necessary.

To get running with Homebrew:

1. Install Homebrew
2. `brew install python`
3. `pip install stackstrap`

Mac Ports is package manager that has been around for a longer time and does not integrate with the base OS X system libraries (the FAQ explains why). It allows for easy installation and use of multiple Python versions (ie. 2.6, 2.7, 3.2 & 3.3).

To get running with Mac Ports and Python 2.7:

1. Install Mac Ports
2. `sudo port selfupdate`
3. `sudo port install py27-pip`
4. `sudo port select --set pip pip27`
5. `sudo pip install stackstrap`

Note: IMHO This section contains the opinions of the developers of StackStrap. These are not the only two methods of installing pip on OS X. You can also install pip directly to the base OS X system or use Fink. (Plus probably some others)

Linux

If you're on a Linux system your package manage will most likely have a package for pip ready for you to use.

On Debian based systems:

```
sudo apt-get install python-pip
```

On RedHat based systems with yum:

```
sudo yum install python-pip
```

Windows

If you're on a Windows sytem then you'll need to install the setuptools and pip packages after install python.

See the following StackOverflow question for info: How to install pip on windows?

4.2 Using StackStrap

Once StackStrap has been installed you'll have the `stackstrap` command available to you in the terminal. You'll use this command to create new projects from a *template repository*.

4.2.1 Adding a template

To add a template use the `stackstrap template add` command and provide it a name along with the GIT URL of a StackStrap template:

```
stackstrap template add django https://github.com/stackstrap/stackstrap-django.git
```

4.2.2 Creating a new project

To create a project use the `stackstrap create` command and provide it a project name and the name of an available template:

```
stackstrap create myproject django
```

4.2.3 Command line help

The `stackstrap` command has built in documentation, run it with the `-h` or `--help` flag for more information on the options you can set from the command line.

You can also get help about a specific command by passing it help:

```
stackstrap create --help
```

4.3 Project Templates

Project Templates are GIT repositories that contain a base layout for a project and are coupled with pillar & state data that Salt uses to provision the project.

The files inside the template can be parsed as Jinja templates, by marking them to be parsed in the meta-data, if you need any of your files to be updated (which you pretty much always do).

You can also apply the same template logic to path names so that you can apply transformations to the filesystem to ensure that best practicies for naming and consistency are applied.

The fact that we have Project Templates and Jinja has a template engine that we use for files can become confusing. To help with the confusion we will always refer to our Project Templates with uppercase letters and to Jinja templates with a lowercase 't'.

4.3.1 Project Template meta-data

The meta-data for Project Templates is stored within a file at the root of the project named `stackstrap.yml`. This directory only exists in the Template itsef and is not present in the projects that are generated.

stackstrap.yml

When loaing a Template into a Project, Stackstrap will take time to parse files and paths with the Jinja template engine. This allows you to seed your Template with project specific name spacing and other goodies like secrets.

Defining your Template for Stackstrap

Give StackStrap some information about your Template so that it can help you set it up.

- **'template_name'** - a short name to describe the Project Template
- **'template_author'** - your name (formatted as "Full Name <email@address>")
- **'template_description'** - a longer multi-line description of the Project Template

Example:

```
template_name: "Flask with Capistrano (nginx + uwsgi)"
template_author: "Brent Smyth <brent@fatbox.ca>"
template_description:
    This template contains a Flask app with Flask Script for managing it.
    It is deployed using nginx as the HTTP endpoint and uwsgi as the
    application server.
```

Cleaning up files

You may have files in your Template which you do not want in your Projects. Perhaps you have a README that is only for the Template and another that you want to use for the Project. In this case just remove the README here and move the other one into place with a path transform (below):

```
stackstrap:
  cleanup:
    - README
```

Parsing files as Jinja templates

To tell StackStrap that a file should be passed through the Jinja template engine when creating a project instance you need to define a list named `file_templates` inside the `stackstrap` name space containing the file paths relative to the root of the project:

```
stackstrap:
  file_templates:
    - .ruby-gemset
    - deployment/deploy.rb
    - deployment/js_compress.json
    - foundation/config.rb
```

Transforming filesystem paths

To tell StackStrap that filesystem paths should be transformed based on the context data when creating a project instance you need to define a dictionary named `path_templates` inside the `stackstrap` name space containing the original path name as the key and the transformed path name, using the available context variables, as the value:

```
stackstrap:
  path_templates:
    - PROJECT-README: README
    - 'project_app/something/else': 'project_app/something/{{ project.id }}'
    - 'project_app': '{{ project.slug }}_app'
```

Filesystem paths are transformed in the order they are listed, so list your more specific matches first as in the example above. Also, the filesystem transforms are applied after the `file_templates` (above) so if you're specifying a file to both be treated as a template and have its filesystem path transformed specify the original path name in the `file_templates` list and not the transformed one.

Available context variables

When the pillar data and your templates are parsed the following variables are made available in the context:

- **'name'** - The computer friendly name which you chose for the project. You can use this to seed your project's files and paths.

4.3.2 Salt Files

The salt macros and pillar data are stored in a folder called *salt* at the root of the project.

TODO: Document this better.