# AppScale Handbook Documentation

*Release*

AppScale Team

January 03, 2013

# CONTENTS

Welcome to AppScale!

CHAPTER

ONE

# DOCUMENTATION

Documentation has been written using the Sphinx project. Documentation can be started by installing the `python-sphinx` package and running the command `sphinx-quickstart`. This command will guide you through the setup process.

One of the many advantages of documenting using Sphinx is its ability to generate documentation in multiple formats. The majority of the time, HTML is going to be the primary format, but it's good to be able to generate a PDF if necessary, and Sphinx has that ability built in.

## 1.1 Building documentation

Building HTML is simple, just run the command `make html` in the documentation directory. This will create the directory `_build/html` which is a static site that can be deployed with any webserver.

CHAPTER

TWO

# LANGUAGES

Initial description.

## 2.1 Python

### 2.1.1 Virtual Environments

A useful utility when developing and deploying python projects is Virtualenv. It provides a way of creating containers for Python libraries and executables on a per-project / installation basis. When the installation is no longer necessary the container can be removed without polluting the system-level Python installation.

#### Virtualenv Primer

First install virtualenv:

```
sudo apt-get install python-virtualenv
```

Once installed, a virtualenv can be created by running the command:

```
virtualenv venv
```

This will create the `venv` directory within the directory it was run. So, if you're in `${HOME}` you should see `${HOME}/venv`. At this point, the virtual environment needs to be "activated", or tied to your shell environment. This can be done by running the command:

```
source ${HOME}/venv/bin/activate
```

At this point Python will execute from that environment using any packages that are installed in it. Installing packages is simply a matter of runing `pip install <package name>` or running `python setup.py install` within a downloaded project.

### 2.1.2 Code Style

#### Docstrings

Docstrings are Python's way to add documentation to your code. The docstring format AppScale uses for functions is covered in the Comments section of Google's Python style.

CHAPTER

THREE

# INFRASTRUCTURE

AppScale's internal infrastructure is comprised of a number of tools; some off-the-shelf, some built specifically for AppScale's needs.

## 3.1 APT

An APT repository to serve AppScale debian packages. Currently the appscale-tools Vagrant configuration uses it to install packages necessary to run AppScale Tools.

The APT repository can be accessed at http://apt.appscale.com by adding the following to `/etc/apt/sources.list.d/appscale.list`:

```
deb http://apt.appscale.com lucid main
```

Once that is in the system's configuration, run `apt-get update` followed by `apt-get install <package name>`.

### 3.1.1 Environment Setup

The appscale-tools Vagrant VM is fully configured for building APT packages. This includes the configuration for the `dupload` command to work, hooking the host's GPG configuration directory and setting up the `DEBEMAIL` and `DEBFULLNAME` variables via `~/.appscale-tools/appscale_config.sh`.

Please make sure you have a working GPG installation with a GPG key for yourself. More information on GPG can be found in the The GNU Privacy Handbook.

Make sure your environment contains the following variables:

```
DEBEMAIL="your.email.address@example.org"
DEBFULLNAME="Firstname Lastname"
```

#### Configuring APT packages

Debian packages are controlled with configuration files in the `debian/` directory. The following documentation walks through creating a .deb package for HAProxy. The resulting repository can be viewed at:

https://github.com/AppScale/appscale-haproxy

First grab the pristine HAProxy source from:

http://appscale.cs.ucsb.edu/appscale_packages/pool/haproxy-1.4.4.tar.gz

Debian has a strict naming convention for package names. Because haproxy is a 3rd party source (i.e. a non-native Debian tool) the name of the initial tar ball needs to conform to the convention with a dash in between the package name and version. Native packages, on the other hand, contain an underscore here:

```
<package_name>-<major version>.<minor version>.<revision version>.tar.gz
```

Because we are maintaining this particular package in-house, rename the extracted directory to:

```
appscale-haproxy-1.4.4
```

And then re-tar it:

```
tar czf appscale-haproxy-1.4.4.tar.gz appscale-haproxy-1.4.4
```

Snapshot the pristine contents in a git repo. In the extracted package:

```
git init
git add -A
git commit -m'Added pristine copy of all files from tarball'
```

Now, create the `debian` directory using `dh_make`:

```
dh_make -f ../appscale-haproxy-1.4.4.tar.gz
```

Answer the question with the `s` option since HAProxy is a single binary install.

The command above creates the `debian` directory and the file `../appscale-haproxy_1.4.4.orig.tar.gz`.

Commit the pristine `debian/` contents to git:

```
git add -A
git commit -m'Added pristine debian directory after running command: dh_make -f ../appscale-haproxy-1
```

Now make changes to the auto-generated files in the `debian/` directory:

- customize the Section, Priority, Homepage, Description, and Depends in debian/control
- where applicable, create the build, binary, install targets in debian/rules
- where applicable, copy the debian/postinst.ex template and add the necessary commands to install the package
- where applicable, create the debian/install file to define any additional files to install

The result of these customizations can be seen at:

https://github.com/AppScale/appscale-haproxy/tree/master/debian

## Building APT packages

Before building a package, make sure you have setup GPG. For more information see the *GPG* documentation.

Build a package using the command:

```
debuild -i'.git|.swp'
```

Note `-i'.git|.swp'`. This will ignore the `.git` directory and any vim swap files lying around or else a slew of errors will pour onto the screen.

When this command attempts to sign the package, you will be prompted for your GPG key's passphrase twice.

This will create the .deb files along with some tarballs and a .changes file. The .changes file is used with the `dupload` command to publish an APT package.

#### Uploading APT packages

Finally, the packages can be uploaded to the APT repository by running:

```
dupload --to appscale /path/to/package.changes
```

#### Repository maintenance

Packages are automatically maintained on the APT repository with the tool reprepro_watch. Upon uploading with `dupload`, `reprepro_watch` will add it to the repository and sign the package with the AppScale Releases key.

For more information on `reprepro_watch` take a look at the reprepro_watch readme.

## 3.2 Jenkins

Jenkins is a continuous integration system used to monitor and execute repeated jobs. Internally, AppScale uses Jenkins to build images and run automated tests.

### 3.2.1 Setup

Start with a vanilla Ubuntu server and install git and Puppet:

```
sudo apt-get install -y git nginx puppet
```

In order for the system to build and test Python, install the following:

```
sudo apt-get install -y python-dev python-virtualenv build-essential libpq-dev
```

Once the packages are installed, download and apply the Jenkins Puppet module:

```
cd /etc/puppet/modules
git clone git://github.com/baremetal/puppet-jenkins.git jenkins
sudo puppet apply /etc/puppet/modules/jenkins/manifests/init.pp
```

Create an SSH key for the `jenkins` user by logging in as jenkins and running the following command:

```
sudo su - jenkins
ssh-keygen -t rsa -b 2048 -f ~/.ssh/id_rsa -P ''
```

### 3.2.2 Configuration

Jenkins is currently installed on a small Amazon EC2 instance. Maintenance can be done by connecting to `ubuntu@ci.appscale.com` via SSH. Most of the time interaction with Jenkins will be done through its web UI, which is accessible by visiting http://ci.appscale.com/.

#### Adding Security

By default Jenkins allows any user, even anonymous, to do anything on the system. The first thing to do is update Jenkins to only allow logged in users to manage the system. To do this, first click `Jenkins -> Manage Jenkins -> Configure Global Security`. The check the box next to “Enable Security”. Under `Access Control` select “Jenkins’s own user database” Under `Authorization` select “Anyone can do anything” for now. Click on save.

Once that's enabled, go to `Jenkins -> Manage Jenkins -> Manage Users`. Create a user account and proceed to login with that username. Then navigate back to `Jenkins -> Manage Jenkins -> Configure Global Security`.

Under `Access Control` select "Matrix-based Security". Create an entry for the user just created, then click on the little icon next to the red X, which toggles all the checkboxes at once. Click save or apply.

At this point Jenkins is only accessible to users that are logged in.

Repeat the steps above to create more users.

#### Adding Git & Github Support

Once that's enabled, go to `Jenkins -> Manage Jenkins -> Manage Plugins`.

Click on the Available Tab and select the "Git Plugin" and "GitHub Plugin" checkboxes. At the bottom click on `Download now and install after restart`. On the subsequent page, click the checkbox next to the restart option.

Go to `Jenkins -> Manage Jenkins -> Configure System`. Under `Git plugin` set the `Global Config user.name Value` and `Global Config user.email Value` fields.

#### Setup nginx

Configure nginx as the webserver in front of Jenkins. The configuration below proxies requests to ci.example.com over to the Jenkins installation, which listens on port 8080 by default:

```
upstream jenkins_server {
    server 127.0.0.1:8080 fail_timeout=0;
}

server {
    listen 80;
    listen [::]:80 default ipv6only=on;
    server_name ci.example.com;

    location / {
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $http_host;
        proxy_redirect off;

        proxy_pass http://jenkins_server;
    }
}
```

### 3.2.3 Add a build job

Jenkins is now ready to be configured to perform tasks. Click on `Jenkins -> New Job`. Give the job a name. It's best to use names without spaces. Select `Build a free-style software project` and click ok.

Under `Source Code Management` select Git and under the `Repository URL` enter the HTTP url found in the Github project page.

Under `Build Triggers` select `Build when a change is pushed to GitHub`.

Finally, under build select `Add build step -> Execute shell`. Use the box that appears to call an executable that is found within the project repository. For example, create a script in the project at `bin/run_tests`. In the box, simply enter `bin/run_tests`. The reason for this is two-fold. First, you are able to version control your script and secondly, you can run the same test procedure Jenkins runs without having to use Jenkins.

Click Save at the bottom of the page.

#### How jobs and build steps work

A job is a collection of build steps that can be performed on a project. A build step, as the name implies, is one or more tasks that should be done to build a project. Jenkins expects a build step to return 0 when it's successful and non-zero when an error occurs. This means that if subsequent steps will not process if a preceeding step exits with a non-zero status.

### 3.2.4 Add a Github WebHook

The Github repository is configured with a Webhook to notify Jenkins whenever changes are pushed to the repository.

On Github's appscale project page, click on `Settings -> Service Hooks -> WebHook URLs` and add the following URL:

```
http://ci.appscale.com/github-webhook/
```

Now, whenever changes are pushed to Github's git repository, the project will build.

CHAPTER

FOUR

# TOOLS

## 4.1 APT Cacher

APT Cacher is a way to cache packages locally so that multiple builds don't have to go out to the internet to pull down packages.

To start using APT cacher, install the apt-cacher package:

```
sudo apt-get install apt-cacher
```

Next, make sure the proxy is set to start automatically by setting `AUTOSTART=1` in `/etc/default/apt-cacher`.

Next, let any host use the cache by changing the following line in `/etc/apt-cacher/apt-cacher.conf`:

```
allowed_hosts = *
```

Finally, on any machine that should use the cache, create the file `/etc/apt/apt.conf.d/01proxy` with the contents:

```
Acquire::http::Proxy "http://<IP address or hostname of the apt-cacher server>:3142";
```

More information can be found at:

https://help.ubuntu.com/community/Apt-Cacher-Server

## 4.2 GPG

GPG is used to sign Debian packages. All the standard debian tools such as `dupload` and `reprepro` as well as package management tools like `apt-get` expect packages to be signed. In order to sign packages, you will need to create a GPG key. If you don't have one, please see below for instructions.

### 4.2.1 Creating a GPG key

A GPG key can be created using the `gpg --gen-key` command. When asked to select what kind of key you want, select the default, `RSA and RSA`'. The default key size of `2048` is good as well. Your first key should never expire, so select `0`, confirm that this is really what you want. Proceed to enter in your name, your email address, and a comment or description for this key.

Depending on how much the system is used, coming up with enough randomness to create the key may take some time. If this happens, try to use the system: navigate the filesystem, compile an application, download stuff, etc.

### 4.2.2 Building software packages

At this point you are ready to use your GPG key to build software packages. Note, the AppScale Tools Vagrant VM links your Mac's GPG configuration directory with the vagrant user on the VM, which means you do not have to do any additional work to use the key in the appscale-tools VM.

## 4.3 Vagrant

Vagrant provides an easy way to create reproducible development environments. It's useful to distribute environment changes to a team of users. Currently the AppScale Tools repository makes full use of Vagrant to provision and deploy a working environment for developing the project as well as providing a command console for AppScale clouds.

More information about Vagrant can be found on the Vagrant Website.

### 4.3.1 How Vagrant works

Vagrant's magic is two-fold. First, it allows provisioning based on pre-configured "boxes". AppScale currently uses the `lucid` base box that is provided by the Vagrant project. With this as the base box a project specifies additional configuration in a `Vagrantfile` within the projects's root.

The `Vagrantfile` defines network configuration, shared folders and additional provisioning once the VM is up and running. When changes are made the `Vagrantfile`, they can be applied to an existing VM by running the `vagrant reload` command.

#### Provisioning

Provisioning tools reconfigure a machine so that it matches a defined state. This state includes the contents of files, installed packages, running services, etc. Vagrant has built-in support for Puppet or Chef; AppScale Tools uses Puppet.

When the command `vagrant up` is run, Vagrant will create the VM based on the settings in the `Vagrantfile` and once the VM is running Puppet completes VM's configuration. At any point in time, the machine can be re-provisioned by running the `vagrant provision` command.

### 4.3.2 Using Vagrant

Please refer to the Getting Started section of the AppScale Tools documentation for more information on how to use Vagrant specifically for AppScale Tools (but the instructions are generally applicable to any Vagrant-based project).

# TESTING

When the `appscale` repository is updated on Github, a notification is sent to Jenkins via Webhook. Jenkins then processes the updates by creating an EC2 image for testing.

## 5.1 Cleaning up

At the moment, there is no automated process for removing EC2 images created by the CI server. This should be done manually to prevent unnecessary clutter and charges. The command below will shows all images created by Jenkins:

```
(appscale-tools)vagrant@lucid64:~$ ec2-describe-images | grep appscale_cluster | awk '{print $2,$3}'
ami-df79fab6 839953741869/appscale_cluster-1354931524-jenkins-c827882
```

The images can then be removed using the ec2-deregister command:

```
(appscale-tools)vagrant@lucid64:~$ ec2-deregister ami-df79fab6
IMAGE       ami-df79fab6
```

CHAPTER

SIX

# INDICES AND TABLES

- *genindex*
- *modindex*
- *search*