

Puppet

Lab 1 – Installing Puppet Standalone

In this lab you will get a chance to configure standalone Puppet systems in multiple environments. Step 1 will install and configure Puppet on a minimal Centos VM, nodea. Step 2 will configure Puppet on an Ubuntu VM, nodeb. Step 3 is an optional take home lab and makes use of Puppet with Vagrant as the VM provisioning facility.

One of the key features of Puppet is its ability to map abstract configurations to the underlying operating system, allowing server configurations to be described generically. Using different OS instances will give us experience applying Puppet across system types here and in the labs ahead.

Step 1 – Centos Setup

Virtual Servers are generally built up from an operating system’s minimal installation image. This makes the server easier to secure and configure because it begins with only the required elements of the operating system. This also reduces dependency contention and improves performance by ensuring that the server runs only the services required of it. Currently most OS distributions include specifically created “core” or “atomic” flavors which can be install to provide minimal functionality at the onset.

The challenge with minimal operating systems is that every operating system distribution deems different things necessary. For example, some minimal installations will not have an ssh server (making it difficult to log in for maintenance until configured properly). Because Puppet is idempotent, the desired configuration can be applied repeatedly, updating those that require new services and leaving those that are already properly configured intact.

Launch the nodea Centos image in your target virtualization provider. When the VM is running login as root with the password “user”. Once logged into the VM console you can type “ip a” to display the IP address of the VM.

```
[root@localhost ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eno16777736: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP qlen 1000
    link/ether 00:0c:29:e9:b0:2f brd ff:ff:ff:ff:ff:ff
    inet 192.168.131.144/24 brd 192.168.131.255 scope global dynamic
eno16777736
    valid_lft 1728sec preferred_lft 1728sec
    inet6 fe80::20c:29ff:fee9:b02f/64 scope link
        valid_lft forever preferred_lft forever
```

If your eno interface does not display an IP address run the “dhclient” command to acquire one and then retry the “ip a” command.

You can ssh into the VM with your preferred ssh client using the eno inet addr (putty, moba-xterm and secureCRT are good free/shareware ssh clients for Windows; Linux and Mac consoles have good native ssh clients).

To make our lab system password easy to remember, change the root password to “puppet”:

```
[root@localhost ~]# passwd
Changing password for user root.
New password:
BAD PASSWORD: The password is shorter than 8 characters
Retype new password:
passwd: all authentication tokens updated successfully.
```

To simulate a proper production environment we will place all of our lab systems in the example.com domain. Set the hostname for your Centos VM to “nodea.example.com”:

```
[root@localhost ~]# hostname nodea.example.com
[root@localhost ~]# hostname
nodea.example.com
```

To verify the new hostname and password, logout ...

```
[root@localhost ~]# logout
```

and log back in with the “puppet” password:

```
[root@nodea ~]#
```

While our minimal Centos installation does have sshd running it does not have Puppet installed and Puppet is not presently available through the standard Centos software repositories. Puppet Labs provides Open Source and Enterprise packages for the latest versions of Puppet in RPM format for RHEL/Fedora/Centos and in deb format for Debian/Ubuntu (for more information see: http://docs.puppetlabs.com/guides/puppetlabs_package_repositories.html).

To install Puppet on the nodea Centos VM, make sure that you are logged in as root and then use the following commands (N.B. other operating system and puppet versions will require the use of different links, this yum repo will also evolve over time; be sure to match repo, OS and puppet versions when you install puppet on other systems).

First add the Puppet Labs yum repository to the system's yum indexes.

```
[root@nodea ~]# rpm -ivh https://yum.puppetlabs.com/puppetlabs-release-pcl-el-7.noarch.rpm
Retrieving https://yum.puppetlabs.com/puppetlabs-release-pcl-el-7.noarch.rpm
Preparing... ##### [100%]
Updating / installing...
 1:puppetlabs-release-pcl-0.9.2-1.el7##### [100%]
```

Now install basic Puppet services:

```
[root@nodea ~]# yum install puppet
Loaded plugins: fastestmirror
base
| 3.6 kB 00:00:00
extras
| 3.4 kB 00:00:00
puppetlabs-pcl
| 2.5 kB 00:00:00
updates
| 3.4 kB 00:00:00
(1/5): base/7/x86_64/group.gz
| 154 kB 00:00:03
(2/5): extras/7/x86_64/primary_db
| 87 kB 00:00:04
(3/5): updates/7/x86_64/primary_db
| 4.0 MB 00:00:18
(4/5): puppetlabs-pcl/x86_64/primary_db
| 24 kB 00:00:27
(5/5): base/7/x86_64/primary_db
| 5.1 MB 00:00:40
Determining fastest mirrors
* base: ftp.cc.uoc.gr
* extras: ftp.cc.uoc.gr
* updates: ftp.cc.uoc.gr
Resolving Dependencies
--> Running transaction check
---> Package puppet-agent.x86_64 0:1.2.4-1.el7 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package                Arch      Version      Repository      Size
=====
Installing:
puppet-agent           x86_64    1.2.4-1.el7  puppetlabs-pcl  18 M
=====

Transaction Summary
=====
Install 1 Package

Total download size: 18 M
Installed size: 82 M
Is this ok [y/d/N]: y
...
```

The first command adds the Puppet Labs package repository for EL7 to the local system. The second command installs the puppet software using the Puppet Labs rpm. Answer “y” to all of the questions to complete the installation.

Puppet versions prior to 4.0 installed puppet binaries on the command path. However, puppet 4.0+ installs puppet executables in /opt/puppetlabs/bin, which is not on the path. Use the puppet --version

command to discover the version of puppet installed, and update your path to make executing puppet commands easier:

```
[root@nodea ~]# /opt/puppetlabs/bin/puppet --version
4.5.1
[root@nodea ~]# export PATH=/opt/puppetlabs/bin:$PATH
[root@nodea ~]# puppet --version
4.5.1
```

To ensure that your path is updated any time you login, update your `.bash_profile`:

```
[root@nodea ~]# vi .bash_profile
[root@nodea ~]# cat .bash_profile
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin:/opt/puppetlabs/bin

export PATH
```

Unlike Puppet versions prior to v4, Puppet v4+ includes its own private version of Ruby making Ruby version conflicts a problem of the past. You can use the `find` command to locate the puppet ruby install and test the ruby interpreter version:

```
[root@nodea ~]# find / -name ruby
/opt/puppetlabs/puppet/bin/ruby
/opt/puppetlabs/puppet/include/facter/ruby
/opt/puppetlabs/puppet/include/ruby-2.1.0/ruby
/opt/puppetlabs/puppet/include/ruby-2.1.0/x86_64-linux/ruby
/opt/puppetlabs/puppet/lib/ruby
[root@nodea ~]# /opt/puppetlabs/puppet/bin/ruby --version
ruby 2.1.9p490 (2016-03-30 revision 54437) [x86_64-linux]
```

The tools used by Puppet are placed in a nested “puppet/bin” directory so as not to override other system installations.

Next we’re going to create a single, simple Puppet manifest to test the Puppet installation. If you attempt to run the vim editor (type “vim” at the command line and pressing enter) you will find that it is not installed as part of the base Centos. We will use Puppet to ensure that vim is always installed on our servers to make system administration easier.

The Puppet installation creates a puppet configuration directory, /etc/puppetlabs/puppet. The installer also creates a default production code environment with a directory for puppet manifests: /etc/puppetlabs/code/environments/production/manifests. Place a new default.pp manifest file there with the following contents (you can use the vi editor).

```
[root@nodea ~]# cd /etc/puppetlabs/code/environments/production/manifests
[root@nodea manifests]# vi default.pp
[root@nodea manifests]# cat default.pp
package { 'vim-enhanced':
    ensure => installed
}

[root@nodea manifests]#
```

This manifest ensures that the package vim-enhanced is installed. After saving the manifest, apply it with the puppet apply command.

```
[root@nodea manifests]# puppet apply default.pp
Notice: Compiled catalog for nodea.localdomain in environment production in
0.87 seconds
Notice: /Stage[main]/Main/Package[vim-enhanced]/ensure: created
Notice: Applied catalog in 91.62 seconds
```

It may take your system a few minutes to download the vim package and install it. Test the vim command to ensure that the manifest was properly applied.

```
[root@nodea manifests]# vim default.pp
[root@nodea manifests]#
```

As a next step let's test Puppet's ability to manage problems. Find the vim binary:

```
[root@nodea manifests]# which vim
/usr/bin/vim
```

Remove the vim binary (e.g. "rm /usr/bin/vim"). Now try to run the vim editor. We have deleted the binary so the vim command should no longer work.

Now let's try to restore vim with Puppet. Rerun the puppet apply command on default.pp. It should complete quickly and with no errors. Now try to run vim again. As you can see vim is not operational even though Puppet just applied a manifest that requires the vim package. The problem here is that Puppet is monitoring the package, which rpm thinks is still installed.

```
[root@nodea manifests]# yum list installed | grep vim
vim-common.x86_64                2:7.4.160-1.el7                @base
vim-enhanced.x86_64             2:7.4.160-1.el7                @base
vim-filesystem.x86_64           2:7.4.160-1.el7                @base
```

Next try removing the vim-enhanced package.

```
[root@nodea manifests]# yum remove vim-enhanced
```

```
Loaded plugins: fastestmirror
Resolving Dependencies
--> Running transaction check
---> Package vim-enhanced.x86_64 2:7.4.160-1.el7 will be erased
--> Finished Dependency Resolution
```

```
Dependencies Resolved
```

Package	Arch	Version	Repository	Size
Removing: vim-enhanced	x86_64	2:7.4.160-1.el7	@base	2.2 M

```
Transaction Summary
```

```
Remove 1 Package
```

```
Installed size: 2.2 M
Is this ok [y/N]: y
Downloading packages:
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
Erasing      : 2:vim-enhanced-7.4.160-1.el7.x86_64
1/1
warning: file /usr/bin/vim: remove failed: No such file or directory
Verifying    : 2:vim-enhanced-7.4.160-1.el7.x86_64
1/1
```

```
Removed:
vim-enhanced.x86_64 2:7.4.160-1.el7
```

```
Complete!
```

Now rerun the puppet apply on default.pp. Notice that this run takes a bit longer as Puppet reinstalls vim. Test vim to ensure that your vim installation is restored.

As we will see in future labs, Puppet can monitor packages but it can also monitor files, user accounts and almost any other type of configured feature, allowing us to have Puppet repair missing resources of almost any type.

Step 2 Ubuntu Setup

In this step we will configure the Ubuntu 14.04 nodeb VM to run Puppet. The nodeb VM is a minimal install Ubuntu 14.04 machine. Since 12.04 the kernels used by Ubuntu Desktop and Ubuntu Server are identical, making this a broadly typical Ubuntu machine. Launch the nodeb VM. This base system does not have sshd running. To login at the console use the user account “user” with the password “user”. Next, for simplicity, set the “root” password and the “user” password to “puppet”:

```
user@ubuntu:~$ sudo passwd user
[sudo] password for user:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
user@ubuntu:~$ sudo passwd root
[sudo] password for user:
Enter new UNIX password:
Retype new UNIX password:
```

```
passwd: password updated successfully
```

As a first priority we should get sshd running so that we do not have to work within the restrictive Ubuntu console window. This is exactly the type of thing Puppet is useful for, however Puppet is not installed by default on a minimal Ubuntu system. Unlike REHL, Ubuntu provides direct package repository support for Puppet so we can use the Ubuntu package installer to add Puppet to the system without adding additional repositories.

This VM was installed from a virtual cdrom, to avoid having the system attempt to load packages from the cdrom remove (or comment) all lines referencing the cdrom in the apt sources.list file.

```
user@ubuntu:~$ sudo vi /etc/apt/sources.list
user@ubuntu:~$ head /etc/apt/sources.list
#
# deb cdrom:[Ubuntu-Server 14.04.1 LTS _Trusty Tahr_ - Release amd64
(20140722.3)]/ trusty main restricted

# See http://help.ubuntu.com/community/UpgradeNotes for how to upgrade to
# newer versions of the distribution.
deb http://us.archive.ubuntu.com/ubuntu/ trusty main restricted
deb-src http://us.archive.ubuntu.com/ubuntu/ trusty main restricted
```

When the repository sources list is updated rebuild the indexes with the apt-get update command.

```
user@ubuntu:~$ sudo apt-get update
...
```

To see which version of puppet the default Ubuntu packages will install use the apt-cache policy command:

```
user@ubuntu:~$ apt-cache policy puppet
puppet:
  Installed: (none)
  Candidate: 3.4.3-1ubuntu1.1
  Version table:
     3.4.3-1ubuntu1.1 0
                   500 http://us.archive.ubuntu.com/ubuntu/ trusty-updates/main amd64
Packages
     3.4.3-1 0
                   500 http://us.archive.ubuntu.com/ubuntu/ trusty/main amd64 Packages
```

The puppet-common package provides the core Puppet engine in older versions of puppet, used by standalone systems and both the agent (client) and master (server) roles in client server puppet

systems. The version (3.4.3) is getting a bit old however. We will use the Puppet Labs apt repository to install the latest version of Puppet. Execute the following commands.

```
user@ubuntu:~$ wget https://apt.puppetlabs.com/puppetlabs-release-pcl-trusty.deb
--2015-09-27 12:52:29-- https://apt.puppetlabs.com/puppetlabs-release-pcl-trusty.deb
Resolving apt.puppetlabs.com (apt.puppetlabs.com)... 192.155.89.90,
2600:3c03::f03c:91ff:fedb:6b1d
Connecting to apt.puppetlabs.com (apt.puppetlabs.com)|192.155.89.90|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2612 (2.6K) [application/x-debian-package]
Saving to: 'puppetlabs-release-pcl-trusty.deb'

100%[=====>] 2,612      --.-K/s   in 0s

2015-09-27 12:52:29 (48.6 MB/s) - 'puppetlabs-release-pcl-trusty.deb' saved
[2612/2612]

user@ubuntu:~$ sudo dpkg -i puppetlabs-release-pcl-trusty.deb
Selecting previously unselected package puppetlabs-release-pcl.
(Reading database ... 51416 files and directories currently installed.)
Preparing to unpack puppetlabs-release-pcl-trusty.deb ...
Unpacking puppetlabs-release-pcl (0.9.2-1trusty) ...
Setting up puppetlabs-release-pcl (0.9.2-1trusty) ...
user@ubuntu:~$ sudo apt-get update
...
```

Now that we have the Puppet Labs repository configured we can install the latest version of Puppet. Unlike the Centos install from the previous step, we cannot install the “puppet” package as this would install the older 3.4.3 package. To install the latest puppet code use the puppet-agent package:

```
user@ubuntu:~$ sudo apt-get install puppet-agent
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  puppet-agent
0 upgraded, 1 newly installed, 0 to remove and 151 not upgraded.
Need to get 9,901 kB of archives.
After this operation, 52.2 MB of additional disk space will be used.
Get:1 http://apt.puppetlabs.com/ trusty/PC1 puppet-agent amd64 1.2.4-1trusty
[9,901 kB]
Fetched 9,901 kB in 42s (235 kB/s)
Selecting previously unselected package puppet-agent.
(Reading database ... 51462 files and directories currently installed.)
Preparing to unpack .../puppet-agent_1.2.4-1trusty_amd64.deb ...
Unpacking puppet-agent (1.2.4-1trusty) ...
Processing triggers for ureadahead (0.100.0-16) ...
Setting up puppet-agent (1.2.4-1trusty) ...
Processing triggers for ureadahead (0.100.0-16) ...
```

Now check the installed Puppet version.


```
user@ubuntu:~$ export PATH=$PATH:/opt/puppetlabs/bin
user@ubuntu:~$ puppet --version
4.5.1
```

Update the “user” and “root” .profile source to set the PATH for puppet use as in step one.

```
user@ubuntu:~$ vi .profile
user@ubuntu:~$ cat .profile
# ~/.profile: executed by the command interpreter for login shells.
# This file is not read by bash(1), if ~/.bash_profile or ~/.bash_login
# exists.
# see /usr/share/doc/bash/examples/startup-files for examples.
# the files are located in the bash-doc package.

# the default umask is set in /etc/profile; for setting the umask
# for ssh logins, install and configure the libpam-umask package.
#umask 022

# if running bash
if [ -n "$BASH_VERSION" ]; then
    # include .bashrc if it exists
    if [ -f "$HOME/.bashrc" ]; then
        . "$HOME/.bashrc"
    fi
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi

PATH="$PATH:/opt/puppetlabs/bin"
```

Our next step is to create a manifest that configures sshd. As in step one, create a new manifest called default.pp and save it within the /etc/puppetlabs/puppet/manifests directory:

```
user@ubuntu:~$ su -
Password:
root@ubuntu:~# cd /etc/puppetlabs/code/environments/production/manifests
# vi default.pp
# cat default.pp
package { 'openssh-server':
    ensure => installed,
}

service { 'ssh':
    ensure => running,
    enable => true,
    require => Package['openssh-server'],
}
```

This manifest declares two resources, a package type resource and a service type resource. The package resource ensures that the openssh-server package is installed. The service resource ensures that the ssh service is enabled and running and that the package that it depends on is present.

Apply this manifest with puppet:

```
# puppet apply default.pp
Notice: Compiled catalog for ubuntu.localdomain in environment production in 0.69 seconds
Notice: /Stage[main]/Main/Package[openssh-server]/ensure: created
Notice: Applied catalog in 11.89 seconds
```

Run the “`service ssh status`” command, you should find the ssh service running. Try to login to the VM using an external ssh client (use “`$ ip a`” to discover the host IP address). You may need to login as user “user” with the password “puppet” over ssh. If so just issue a “`su -`” command once logged in to change into a root shell.

Now that we have ssh working with nodeb, try to run “vim”. As you can see it is not installed by default. Most modern Linux systems install the base vim package, which implements vi, but not the full featured vim. The package housing the full featured vim on Ubuntu is “vim”. Thus the package declaration we used for Node A, “vim-enhanced” will not work for Node B.

Update the Node B default.pp manifest to include the vim package.

```
package { 'openssh-server':
    ensure => installed,
}

service { 'ssh':
    ensure => running,
    enable => true,
    require => Package['openssh-server'],
}

package { 'vim':
    ensure => installed,
}
```

Apply the updated default.pp manifest. Note that Puppet is idempotent, and will make no changes to resources that are already in place and properly configured. When the Puppet run completes use the vim editor to view your default.pp file to test your updated installation.

As we will see in future labs, OS specific providers implement the declarative configuration statements from our Puppet manifests. Many things can be declared generically across systems, but many things

also require additional specification if they are to operate across operating systems. The vim editor package is an example of a package resource with a different name in different OS environments. Puppet knows to use yum on Centos and apt-get on Ubuntu for the Package type, but it has no way to know that vim and vim-enhanced are the same thing. We will see how to improve the portability of our manifests in the next set of labs.

Step 3 [Optional] Vagrant 1.3.3 Setup on Ubuntu 10.04

In this step you will get a chance to use Vagrant with Puppet to streamline desktop based development and testing of Puppet nodes. Though this is an optional step, Vagrant can be a very productive tool for admins working with puppet.

Vagrant: *noun*, a person without a settled home

Vagrant is a tool which allows you to save one or more virtual machine instances, known as “base boxes”, to a library and then launch these base boxes with various customizations to create an assortment of running VMs. These customized VMs can be used to develop and test new server configurations, new software, critical platform changes, application interactions and any number of other activities. Vagrant VMs can easily be moved directly into acceptance test and production environments, even directly to AWS or other cloud infrastructures.

The key benefit of Vagrant is that it makes it easy to launch a Phoenix server (a server that is created and destroyed over and over, but which always comes back with the same configuration). When Vagrant launches a VM it uses a base box as a seed, enhancing the base with configuration data from a Vagrantfile. For example, using the same base box, Vagrant could be used to launch a Web server, a Database server, an Application server, a Messaging server, a client host, etc. Each server is defined by the base box and a small Vagrantfile which lists the adjustments to make to the base box.

Rather than reinventing the wheel, Vagrant relies on a configuration management tool, usually Puppet or Chef, to handle the base box configuration management. Vagrant also depends on a virtualization provider, like VMWare Fusion, to run the VM. Vagrant VMWare Fusion/Workstation integration is excellent however a \$79/seat license is required to use Vagrant with VMWare in addition to the VMWare Fusion/Workstation license (VMWare does not allow Vagrant integration with Player). The Virtual Box provider is another option, it is free and built into Vagrant.

To complete this lab you will either need to install Vagrant and the free Oracle Box virtualization software or VMWare Fusion 5-6/Workstation 9-10 with the Vagrant VMWare integration add on.

- **Vagrant:** <http://www.vagrantup.com/>
- **Virtual Box:** <https://www.virtualbox.org/>
- **VMWare Workstation 10 Free Trial** (Windows/Linux):
<http://www.vmware.com/products/workstation/workstation-evaluation>
- **VMWare Fusion 6 Free Trial** (OS X):
https://my.vmware.com/web/vmware/info/slug/desktop_end_user_computing/vmware_fusion/6_0
- **Vagrant VMWare Integration (\$79):** <http://www.vagrantup.com/vmware>

If you have VMWare installed but would like to try Vagrant, you can install Virtual Box for free. Virtual Box and VMWare can coexist and you can run VMs in both environments concurrently, however there are issues to manage. It is important to note that only one virtualization platform can use VT hardware acceleration at a time and the network bridging drivers should also not be mixed. You will need to disable VT and network bridging in one or both virtualization platforms if you plan to run VMs in each concurrently. Because many of our labs assume that the NodeA & B VMs are network bridged it would be better not to mix multiple providers on the same host if you are not extremely comfortable adjusting the VM networking settings on your workstation.

A. Install Vagrant and a Virtualization Provider

Using the links above, install Vagrant and a virtualization provider on your workstation.

B. Add a base box to the Vagrant library

Base Boxes are added to the Vagrant library using the command:

```
$ vagrant box add {title} {url}
```

This command will copy the designated base box into the local base box library (`~/.vagrant.d/boxes`). The `{title}` is the name you will use to reference the base box in Vagrantfiles and with vagrant commands at the command line. The `{url}` is the path to the source base box (can be local, `file:///` or remote, `http://` or `ftp://`, etc.). There is nothing particularly special about a base box. It is simply a VM configured such that it will be easy to use as a base for various Vagrantfile/Puppet specific configurations you will use (must have `sshd`, `puppet`, `nat` and a few other things configured for Vagrant to use). Many organizations provide base boxes for use by the public. A registry of over 100 Vagrant base boxes can be found here:

<http://www.vagrantbox.es/>

The Puppet Labs web site hosts several Vagrant base boxes here:

<http://puppet-vagrant-boxes.puppetlabs.com/>

Base boxes are all virtualization provider specific. The same title can be used for base boxes configured for multiple providers.

Download the appropriate Ubuntu Server 10.01.1 base box for your virtualization provider from the Puppet Labs site and name it “`puppet_class`”. Do not choose the “no CM” (no configuration management) version, these do not have puppet installed.

- VMWare:
<http://puppet-vagrant-boxes.puppetlabs.com/ubuntu-server-10044-x64-fusion503.box>
- VirtualBox:
<http://puppet-vagrant-boxes.puppetlabs.com/ubuntu-server-10044-x64-vbox4210.box>

For example, to install for VMWare:

```
D:\VMs\NodeC> vagrant box add puppet_class http://puppet-vagrant-  
boxes.puppetlabs.com/ubuntu-server-10044-x64-fusion503.box
```

```
Downloading or copying the box...  
Extracting box...ate: 692k/s, Estimated time remaining: --:--:--)  
Successfully added box 'puppet_class' with provider 'vmware_fusion'!
```

You can list the boxes in your local library using the “vagrant box list” command and you can remove boxes with the “vagrant box remove” command.

```
D:\VMs\NodeC> vagrant box list  
precise32      (virtualbox)  
precise64      (virtualbox)  
puppet_class   (virtualbox)  
puppet_class   (vmware_fusion)  
  
D:\VMs\NodeC> vagrant box remove precise64 virtualbox  
Removing box 'precise64' with provider 'virtualbox'...
```

C. Create a new Vagrant VM instance

New Vagrant VM instances are created from the Base Boxes using the init command:

```
$ vagrant init {title}
```

The {title} is the name of the local base box to use with this Vagrant VM instance. If the title is associated with more than one provider the provider must follow the base box title. Base boxes can be added to the local library implicitly with the init command by following the title with a URL. For example:

```
$ vagrant init precise64 http://files.vagrantup.com/precise64.box
```

The above command creates a new base box called precise64 and downloads the image from the URL, then inits a new Vagrant instance based on this box.

Create a working directory for a new Vagrant based VM called NodeC. For example, “~/NodeC”. Make this directory the current working directory and initialize a new Vagrant instance based on the puppet_class base box.

```
D:\VMs\NodeC> vagrant init puppet_class  
A `Vagrantfile` has been placed in this directory. You are now  
ready to `vagrant up` your first virtual environment! Please read  
the comments in the Vagrantfile as well as documentation on  
`vagrantup.com` for more information on using Vagrant.
```

D. Launch the vagrant VM instance

The “vagrant up” command launches the virtual machine configured in the current directory.

```
D:\VMs\NodeC> vagrant up
Bringing machine 'default' up with 'vmware_fusion' provider...
[default] Importing base box 'puppet_class'...
[default] Matching MAC address for NAT networking...
[default] Setting the name of the VM...
[default] Clearing any previously set forwarded ports...
[default] Fixed port collision for 22 => 2222. Now on port 2200.
[default] Creating shared folders metadata...
[default] Clearing any previously set network interfaces...
[default] Preparing network interfaces based on configuration...
[default] Forwarding ports...
[default] -- 22 => 2200 (adapter 1)
[default] Booting VM...
[default] Waiting for machine to boot. This may take a few minutes...
[default] Machine booted and ready!
[default] Mounting shared folders...
[default] -- /vagrant
```

When running vagrant up for the first time on a newly initialized VM, vagrant imports the base box, making a copy of the base box for the virtualization provider to use going forward. These new VM images are stored wherever the provider stores its VMs by default. For instance VirtualBox saves new VMs here: “~/VirtualBox VMS”.

Once the new VM is running you can display the status of the machine, halt it or suspend it.

```
D:\VMs\NodeC>vagrant status
Current machine states:

default                running (vmware_fusion)

The VM is running. To stop this VM, you can run `vagrant halt` to
shut it down forcefully, or you can run `vagrant suspend` to simply
suspend the virtual machine. In either case, to restart it again,
simply run `vagrant up`.
```

E. Login to the new VM

When a Vagrant VM loads it runs sshd on port 22 but maps that port to port 2222 on the host. To login to the Vagrant VM you can use the command “vagrant ssh” which will automatically ssh and login as vagrant. You can login with other ssh tools (like putty) using 127.0.0.1:2222 on the host.

Here is an example vagrant ssh login from a mingw shell on Windows.

```
Randy@VAVAU /d/VMs/NodeC
$ vagrant ssh
Welcome to Ubuntu 12.04.3 LTS (GNU/Linux 3.2.0-53-generic x86_64)

* Documentation:  https://help.ubuntu.com/

System information as of Wed Sep 25 06:43:19 UTC 2013
```

```
System load: 0.0          Processes: 79
Usage of /: 8.9% of 39.36GB Users logged in: 0
Memory usage: 5%         IP address for eth0: 10.0.2.15
Swap usage: 0%
```

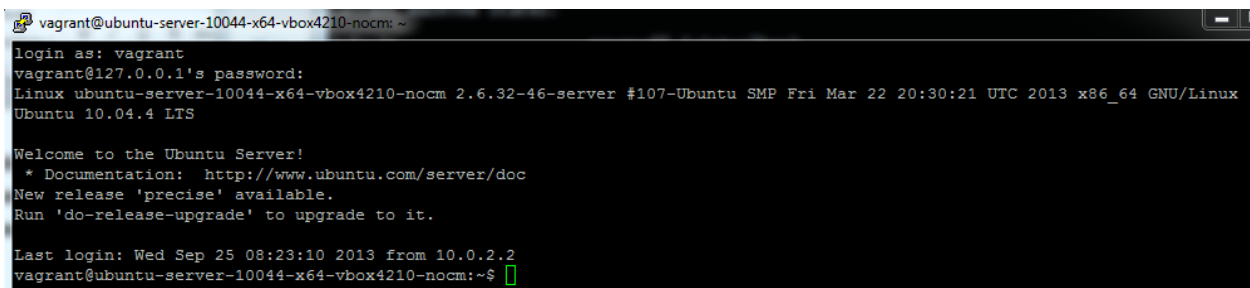
Graph this data and manage this system at <https://landscape.canonical.com/>

Get cloud support with Ubuntu Advantage Cloud Guest:
<http://www.ubuntu.com/business/services/cloud>

Use Juju to deploy your cloud instances and workloads:
<https://juju.ubuntu.com/#cloud-precise>

Last login: Mon Sep 23 23:57:15 2013 from 10.0.2.2
vagrant@vagrant-ubuntu-precise-64:~\$

If you login outside of the vagrant shell it is convention to create a user “vagrant” with the password “vagrant” and full root sudo permissions. Here is an example logging in with putty.



```
vagrant@ubuntu-server-10044-x64-vbox4210-nocm: ~
login as: vagrant
vagrant@127.0.0.1's password:
Linux ubuntu-server-10044-x64-vbox4210-nocm 2.6.32-46-server #107-Ubuntu SMP Fri Mar 22 20:30:21 UTC 2013 x86_64 GNU/Linux
Ubuntu 10.04.4 LTS

Welcome to the Ubuntu Server!
 * Documentation:  http://www.ubuntu.com/server/doc
New release 'precise' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Wed Sep 25 08:23:10 2013 from 10.0.2.2
vagrant@ubuntu-server-10044-x64-vbox4210-nocm:~$
```

You can run multiple Vagrant VMs simultaneously. Each new Vagrant VM will detect port collisions on the host and choose a new port to map its sshd service to. Here is an example launching a second, separate Vagrant VM, notice the sshd port mapping messages.

```
Randy@VAVAU /d/VMs/vagrantthrift
$ vagrant up
Bringing machine 'default' up with 'vmware_fusion' provider...
[default] Clearing any previously set forwarded ports...
[default] Fixed port collision for 22 => 2222. Now on port 2200.
[default] Creating shared folders metadata...
[default] Clearing any previously set network interfaces...
[default] Preparing network interfaces based on configuration...
[default] Forwarding ports...
[default] -- 22 => 2200 (adapter 1)
[default] Running 'pre-boot' VM customizations...
[default] Booting VM...
[default] Waiting for machine to boot. This may take a few minutes...
[default] Machine booted and ready!
[default] The guest additions on this VM do not match the installed version
of
VirtualBox! In most cases this is fine, but in rare cases it can
cause things such as shared folders to not work properly. If you see
```

```
shared folder errors, please update the guest additions within the
virtual machine and reload your VM.
```

```
Guest Additions Version: 4.1.12
VirtualBox Version: 4.2
[default] Mounting shared folders...
[default] -- /thrift
[default] -- /vagrant
```

```
Randy@VAVAU /d/VMs/vagrantthrift
$
```

Shutting down the guest through the ssh connection will shutdown the VM. If you make configuration changes to the VM they will remain intact until you destroy the VM using the “vagrant destroy” command. You can use “vagrant destroy” and then issue the “vagrant up” command to create a fresh copy of the base box.

F. Configure the VM to install vim

As a final step we will configure our new vagrant VM to run puppet apply on a manifest to configure our VM each time it is started.

Once you have logged in to the new VM, run the “puppet --version” command. As you can see puppet is installed. Now run the “service --status-all” command. You will note that there are no puppet services running. Vagrant is typically configured to run puppet apply in the VM just after it boots. You can also ask vagrant to run puppet apply from the host shell at any other time by using the command “vagrant provision”.

Assume we require the vim editor in our VM. If you attempt to run the vim editor from the command line you will see that it is not installed. The vagrant init command creates a Vagrantfile in the VM init directory. This file describes the type of machine to be configured from the base box when “vagrant up” is invoked. Vagrantfiles are text files which describe how to customize features of the VM over and above those defined by the base box.

When you run “vagrant up”, vagrant looks in the cwd for a Vagrantfile. If one is not found it searches the parent directory, and so on, until a Vagrantfile is found. The default project Vagrantfile generated by vagrant init provides commented examples which can be used to perform custom port forwarding (e.g. mapping guest port 80 to host port 8888), static private network addressing (giving the guest a fixed 192.168.0.0 address) and/or sharing folders between the guest and the host.

Modify your Vagrantfile to match the following listing.

```
Vagrant.configure("2") do |config|
  config.vm.box = "puppet_class"
  config.vm.hostname = "NodeC.example.com"
  config.vm.provision "puppet"
end
```


The Vagrant.configure line sets the configuration to version 2 and the config.vm.box line sets the base box to use. The config.vm.provision line sets the Vagrant provisioner to puppet. When the VM is started in the future with “vagrant up” it will automatically run puppet apply on a host file named “./manifests/default.pp”. You can customize the puppet configuration if you like using the instructions found here: http://docs.vagrantup.com/v2/provisioning/puppet_apply.html.

Now we can create a puppet manifest to automatically install vim on our new VM. Create a manifest called default.pp in the manifests directory under the NodeC home directory. Add the following to the default.pp file:

```
package { 'vim':  
  ensure => "installed"  
}
```

Now shutdown the guest VM and restart it with “vagrant up”. Notice that it reports running Puppet on startup. Ssh into the new VM and test vim. Now you can start/stop/destroy and recreate this VM and it will always have vim installed thanks to our Puppet configuration.