
Getting Started With Puppet Development Documentation

Release 0.1.0

Tim Hughes

June 02, 2016

1	Setting up your workspace	3
1.1	Taking control of our Ruby environment with rbenv	3
2	Creating our first module	5
2.1	Create a module structure	5
2.2	Customise the module	7
3	Virtualized test environment	11
3.1	Setting up Libvirt virtualization	11
3.2	Configure Libvirt networking	11
3.3	Build puppet and test virtual machines	12
4	Puppet Master	19
4.1	Install Puppet master	19
4.2	Install and configure r10k	20
4.3	Test installation	21
5	Puppet Manifests	23
5.1	Style guide	23
5.2	Create our base manifests	23
6	Git Server	27
7	Static Analysis	29
7.1	Pre commit hooks	29
8	Acceptance Testing	31
8.1	Vagrant setup	31
9	Jenkins	33
10	Workflow	35
10.1	Pre requisites	35
10.2	Workflow	35
11	TODO	37
11.1	Items we need to cover and arrange in order	37
11.2	Resources	38
12	Indices and tables	39

Contents:

Setting up your workspace

Get the same versions of ruby as what you are running in production. Here we use *rbenv* but there are alternatives such as *rvm*

<https://rvm.io/>

1.1 Taking control of our Ruby environment with rbenv

<https://github.com/sstephenson/rbenv>

Initial install of rbenv and the ruby-build plugin that allows installing different versions of ruby.

```
git clone https://github.com/sstephenson/rbenv.git $HOME/.rbenv

git clone https://github.com/sstephenson/ruby-build.git \
  $HOME/.rbenv/plugins/ruby-build
```

Add rbenv bin directory to our path by editing `~/.bashrc`

```
#####
# Ruby configuration
#####
# rbenv
# https://github.com/sstephenson/rbenv
# git clone https://github.com/sstephenson/rbenv.git $HOME/.rbenv
export RBENV_ROOT="$HOME/.rbenv"
export PATH="$RBENV_ROOT/bin:$PATH"
if type "rbenv" &> /dev/null; then
  eval "$(rbenv init -)"
  # Add the version to my prompt
  __rbversion () {
    if type "ruby" &> /dev/null; then
      rbenv_ruby_version=$(rbenv version | sed -e 's/ .*//')
      printf $rbenv_ruby_version
    fi
  }
  export PS1="rb:\${__rbversion}|$PS1"
fi
```

Make sure you source your `.bashrc` file to initialise rbenv

```
source ~/.bashrc
```

Before you install any rubies make sure that the development libraries and tools are installed for your system. - <https://github.com/sstephenson/ruby-build/wiki#suggested-build-environment>

```
sudo yum -y install gcc openssl-devel libyaml-devel readline-devel zlib-devel
```

Install Ruby version.

```
rbench install 2.0.0-p353
```

There is currently a bug in ruby ssl which manifests itself on Fedora and it's derivatives such as CentOS and RHEL. The fix is to follow the instructions at:

- <https://github.com/sstephenson/ruby-build/wiki#make-error-for-200-p247-and-lower-on-fedorared-hat>

You will need to install the *patch* command for this to work:

```
sudo yum -y install patch
```

For Ruby 1.8.7-p352 (the version on CentOS and RHEL) the following patch and compiler options should be used

```
curl -fsSL https://github.com/sstephenson/ruby-build/raw/c636d214f79900d30b8b2e9b6d49891db9ebf068/sha256sums.txt | \
| CFLAGS="-O2 -fno-tree-dce -fno-optimize-sibling-calls" rbench install --patch 1.8.7-p352
```

- <https://bugs.ruby-lang.org/issues/6383#note-1>

If you don't want install the documentation for every package add the following to *~/.gemrc*

```
install: --no-ri --no-rdoc
update: --no-ri --no-rdoc
```

Create a working directory.

```
mkdir ~/ruby_workingdir
cd ~/ruby_workingdir
```

Set the local ruby version for your working directory.

```
rbench local 1.8.7-p352
```

Install the required gems

```
gem install -V puppet bundler
```

Whenever you install a new version of Ruby, or install a gem that provides commands you need to make sure that rbench knows about the commands. To do this run the following command.

```
rbench rehash
```

Creating our first module

Puppet modules are the fundamental building block of puppet and are used for abstracting away the differences between different platforms. A good module for some software should not define how you want the software but provide an API so that the software can be used on multiple platforms without needing to know the intricacies of that platform. An important part of writing reusable modules is that other people can easily understand your code. To this end Puppetlabs has published a [Puppetlab Style Guide](#) which is based on best practices in the Puppet community.

2.1 Create a module structure

PuppetForge module naming convention is the name or organisation that is developing them and the name module name separated by a hyphen such as this:

```
fullstackpuppet-ntp
```

Lets create a directory to store all our modules we are going to create and then use the *puppet module* command to create a basic module structure:

```
mkdir ~/ruby_workingdir/modules
cd ~/ruby_workingdir/modules
```

```
[thughes@titanium: ~/ruby_workingdir/modules]$ puppet module generate fullstackpuppet-ntp
We need to create a metadata.json file for this module. Please answer the
following questions; if the question is not applicable to this module, feel free
to leave it blank.
```

```
Puppet uses Semantic Versioning (semver.org) to version modules.
```

```
What version is this module? [0.1.0]
```

```
-->
```

```
Who wrote this module? [fullstackpuppet]
```

```
-->
```

```
What license does this module code fall under? [Apache 2.0]
```

```
-->
```

```
How would you describe this module in a single sentence?
```

```
--> Manages NTP on linux machines.
```

```
Where is this module's source code repository?
```

```
--> https://github.com/fullstack-puppet/fullstackpuppet-ntp.git
```

```
Where can others go to learn more about this module? [https://github.com/fullstack-puppet/fullstackpuppet]
-->
```

```
Where can others go to file issues about this module? [https://github.com/fullstack-puppet/fullstackpuppet]
-->
```

```
-----
{
  "name": "fullstackpuppet-ntp",
  "version": "0.1.0",
  "author": "fullstackpuppet",
  "summary": "Manages NTP on linux machines.",
  "license": "Apache 2.0",
  "source": "https://github.com/fullstack-puppet/fullstackpuppet-ntp.git",
  "project_page": "https://github.com/fullstack-puppet/fullstackpuppet-ntp",
  "issues_url": "https://github.com/fullstack-puppet/fullstackpuppet-ntp/issues",
  "dependencies": [
    { "version_requirement": ">= 1.0.0", "name": "puppetlabs-stdlib" }
  ]
}
-----
```

```
About to generate this metadata; continue? [n/Y]
```

```
--> y
```

```
Notice: Generating module at /home/thughes/ruby_workingdir/fullstackpuppet-ntp...
```

```
Notice: Populating templates...
```

```
Finished; module generated in fullstackpuppet-ntp.
```

```
fullstackpuppet-ntp/tests
```

```
fullstackpuppet-ntp/tests/init.pp
```

```
fullstackpuppet-ntp/Rakefile
```

```
fullstackpuppet-ntp/spec
```

```
fullstackpuppet-ntp/spec/classes
```

```
fullstackpuppet-ntp/spec/classes/init_spec.rb
```

```
fullstackpuppet-ntp/spec/spec_helper.rb
```

```
fullstackpuppet-ntp/Gemfile
```

```
fullstackpuppet-ntp/manifests
```

```
fullstackpuppet-ntp/manifests/init.pp
```

```
fullstackpuppet-ntp/metadata.json
```

```
fullstackpuppet-ntp/README.md
```

```
cd fullstackpuppet-ntp
```

Create `.gitignore` file

```
cat <<EOF > .gitignore
pkg/
Gemfile.lock
.bundle/
.rspec_system/
.*.SW*
/spec/fixtures/.librarian
/spec/fixtures/.tmp
/spec/fixtures/Puppetfile.lock
/spec/fixtures/modules
/spec/fixtures/manifests
/spec/fixtures/vendor
EOF
```

2.2 Customise the module

Now we have a basic module structure which we can use to create our working NTP module. The main file and entry point to the module is *manifests/init.pp* and if we open that up we will see a load of comments and an empty *ntp* class construct. This class will later provide the API to the ntp module but for now we will leave it as it is.

```
# == Class: ntp
#
# Full description of class ntp here.
#
# === Parameters
#
# Document parameters here.
#
# [*sample_parameter*]
#   Explanation of what this parameter affects and what it defaults to.
#   e.g. "Specify one or more upstream ntp servers as an array."
#
# === Variables
#
# Here you should define a list of variables that this module would require.
#
# [*sample_variable*]
#   Explanation of how this variable affects the function of this class and if
#   it has a default. e.g. "The parameter enc_ntp_servers must be set by the
#   External Node Classifier as a comma separated list of hostnames." (Note,
#   global variables should be avoided in favor of class parameters as
#   of Puppet 2.6.)
#
# === Examples
#
# class { 'ntp':
#   servers => [ 'pool.ntp.org', 'ntp.local.company.com' ],
# }
#
# === Authors
#
# Author Name <author@domain.com>
#
# === Copyright
#
# Copyright 2014 Your name here, unless otherwise noted.
#
class ntp {
}
```

The basic pattern of most modules on Linux servers involves 3 components:

Software installation This can be anything from RPM, DEB package management through Ruby Gems through to a Zip or TAR file.

Configure the software This is usually through a configuration file but may involve creating databases or directories with the correct permissions.

Start the software This is usually through whatever service management system is being used on the system. This may be things like systemd, chkconfig, supervisord. This step may not be needed if it not software that runs as a service.

To make our code easy to understand we split these three jobs into separate classes and keep them in their own files.

```
install.pp class modulename::install
```

```
config.pp class modulename::config
```

```
service.pp class modulename::service
```

We create one more class which contains the default parameters for our module.

```
params.pp class modulename::params
```

Lets start with the software installation step of our module in *manifests/install.pp*.

Content for *manifests/install.pp*

```
# == Class: ntp::install
class ntp::install inherits ntp {

  package { ['ntp']:
    ensure => installed,
  }

}
```

In this class we inherit from *ntp*. This inheritance will be used later to inherit properties that the main *ntp* class configures. The second thing it does is define a Puppet type *package* named *ntp* and ensure it is installed. Puppet abstracts away the actual installation so that this will install *ntp* on systems with different package managers. For instance, on Debian it will use *apt-get* and on Redhat it will use *yum install*

Next we need to configure our ntp software. The config file for ntp is */etc/ntp.conf*

Content for *manifests/config.pp*

```
# == Class: ntp::config
class ntp::config inherits ntp {

  file { ['/etc/ntp.conf']:
    ensure => file,
    owner  => 'root',
    group  => 'root',
    mode   => 0644,
    content => template($module_name/ntp.conf.erb),
  }

}
```

As with *ntp::install* we inherit from the main *ntp* class. Next we use the Puppet type *file* to manage the ntp config file setting its permissions and content which is created by a *template*

Templates go in the template directory of our module. They can contain variables and basic logic to construct the end file but for now we will just create one with no template logic and we can come back to it later.

Content for *templates/ntp.conf.erb*

```
driftfile /var/lib/ntp/drift

restrict default kod nomodify notrap nopeer noquery
restrict -6 default kod nomodify notrap nopeer noquery

restrict 127.0.0.1
restrict -6 ::1
```

```
server 0.centos.pool.ntp.org iburst
server 1.centos.pool.ntp.org iburst
server 2.centos.pool.ntp.org iburst
server 3.centos.pool.ntp.org iburst

includefile /etc/ntp/crypto/pw

keys /etc/ntp/keys
```

The final part of a basic module is making sure that the software is running. This is managed in the *manifests/service.pp* file.

Content for *manifests/service.pp*

```
# == Class: ntp::service
class ntp::service inherits ntp {

  service { 'ntp':
    ensure    => running,
    enable    => true,
    hasstatus => true,
    hasrestart => true,
    require => Package['ntp'],
  }

}
```

Once again we inherit from the main *ntp* class. This time we use the Puppet type *service* which is an abstraction of most major service control systems on Linux. We ensure that the service *ntp* is running and that it is enabled on boot. It also requires the *package ntp* because there is no point trying to start software that isn't installed.

We haven't created the *manifests/params.pp* file yet because everything is hard coded in our module.

Now we have our three subclasses defined we need to let the main *ntp* class know about them. To do this we need to include our subclasses in our main *ntp* class.

```
class ntp {

  include ntp::install
  include ntp::config
  include ntp::service

}
```

If you haven't used git before then you will need to do a little setup. Set you name and email and editor of choice.

```
git config --global user.email "you@example.com"
git config --global user.name "Your Name"

git config --global core.editor vim
git config --global push.default simple
```

Initialize git repo:

```
git init .
git add --all .
```

Do the initial commit:

```
git commit -am "Initial commit of fullstackpuppet-ntp"
```

Note: Further Reading

- [Puppetlab Style Guide](#)
 - [Puppet Module Fundamentals](#)
-

Virtualized test environment

To test our puppet module on a server that is not a production server we can create a virtual puppet environment on our local machines. This way we can make sure that everything works as we expect before we deploy to production and our lives will hopefully be a little less stressful.

In our example here we are going to use Libvirt on a Fedora Workstation.

3.1 Setting up Libvirt virtualization

Install the required packages for libvirt on fedora:

```
sudo yum -y install libvirt-devel libxslt-devel libxml2-devel \
    lzma-devel qemu-img qemu-kvm virt-install libvirt \
    libvirt-daemon-kvm qemu-kvm libvirt-daemon-config-network \
    virt-manager
```

To allow access to virt-manager to members of the *wheel* group without entering password, create a file named `/etc/polkit-1/rules.d/80-libvirt-manage.rules` with following content.

```
polkit.addRule(function(action, subject) {
    if (action.id == "org.libvirt.unix.manage" && subject.local && subject.active && subject.isInGroup(
        "wheel")) {
        return polkit.Result.YES;
    }
});
```

Remember to add your user to the wheel group:

```
sudo usermod -a -G wheel tim
```

Make sure that libvirt is configured to start on boot and that it is running:

```
sudo systemctl enable libvirtd.service
sudo systemctl start libvirtd.service
sudo systemctl status libvirtd.service
```

3.2 Configure Libvirt networking

Edit the default NAT network used by libvirt on your machine. You need to add the *domain* and two statically assigned entries, one for the puppet master and one for a test machine. I use test03 here so that I can remember what the last octet in the ip address is but you can call it whatever you want.

```
sudo virsh net-edit default
```

```
<network>
  <name>default</name>
  <uuid>91aaf6cb-af4a-4e30-8c3e-b44757ff4cbc</uuid>
  <forward mode='nat' />
  <bridge name='virbr0' stp='on' delay='0' />
  <domain name='corp.guest' />
  <ip address='192.168.122.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.122.2' end='192.168.122.254' />
      <host mac='52:54:00:a8:7a:02' name='puppet.corp.guest' ip='192.168.122.2' />
      <host mac='52:54:00:a8:7a:03' name='test03.corp.guest' ip='192.168.122.3' />
    </dhcp>
  </ip>
</network>
```

The MAC addresses above were generated with the following bash function.

```
function gen-virt-mac-from-ip () {
  IP=$1
  printf '52:54:00:%02X:%02X:%02X\n' $(echo $IP | cut -d'.' --output-delimiter=' ' -f2,3,4) | tr [A-Z]
```

Restart the virtual network:

```
sudo virsh net-destroy default
sudo virsh net-start default
```

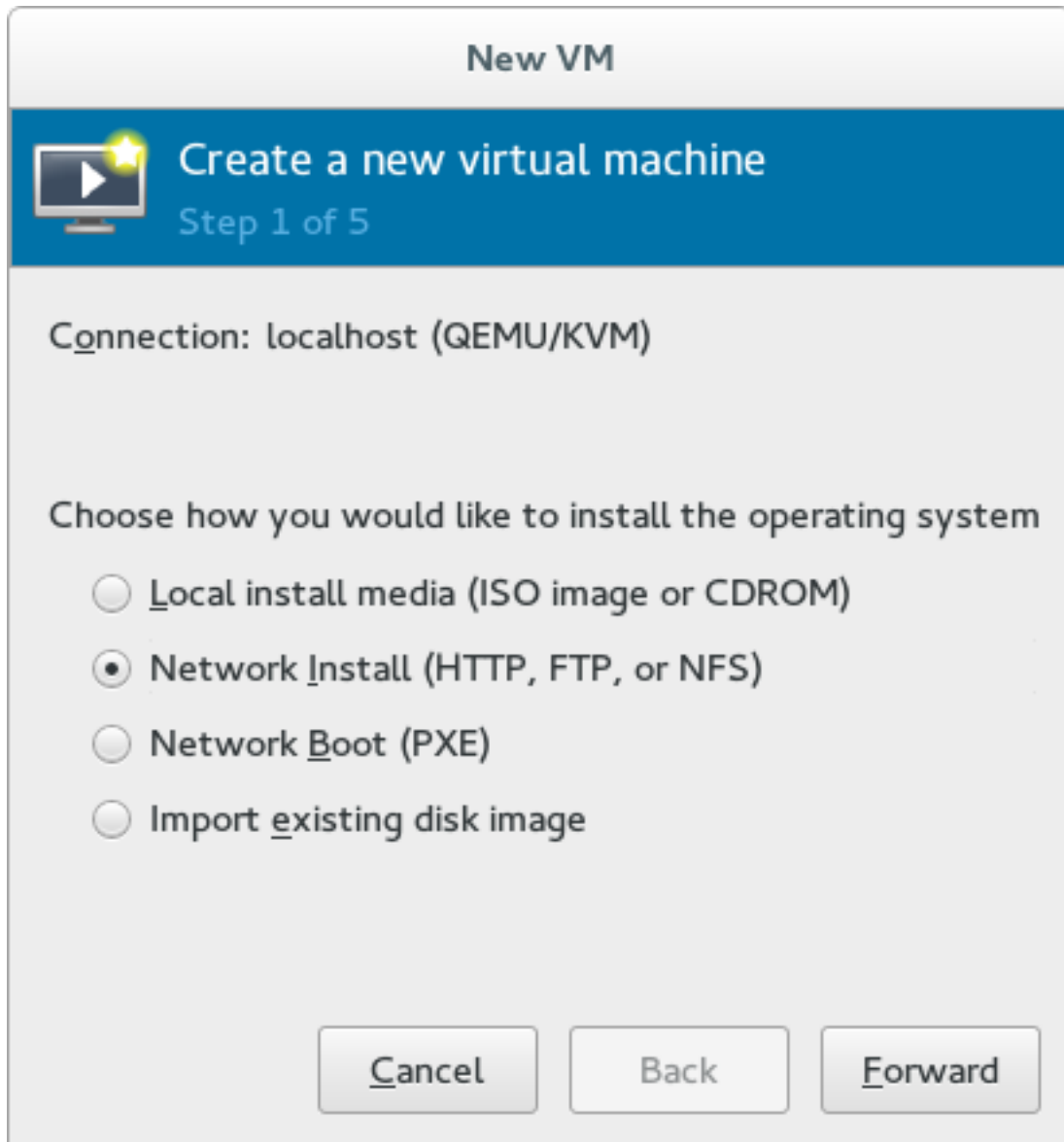
Optionally we can put the hostnames in our hosts file to make it easier to ssh to them.

```
192.168.122.2 puppet.corp.guest
192.168.122.3 test03.corp.guest
```

3.3 Build puppet and test virtual machines

Now create the two guest machines. Make sure you set the mac addresses to match above. The following screen shots are of *virt-manager* aka “*Virtual Machine Manager*” which we installed earlier.

Installation Network install




You can use the kickstart available at <https://raw.githubusercontent.com/timhughes/getting-started-with-puppet-development/master/resources/centos-minimal-ks.cfg>

Hint: The kickstart sets the root password to *password* so download the file and change it to something more secure. The length of the URL doesn't work with anaconda so this will force you to be hosted elsewhere. If you cannot find a web server then download the kickstart to a empty folder on your local machine and once you have edited the password start a simple web server using this python command `python -m SimpleHTTPServer` and then you should be able to make your kickstart location `http://192.168.122.1:8000/centos-minimal-ks.cfg`

Install URL http://mirror.bytemark.co.uk/centos/6/os/x86_64/

Kickstart URL <http://192.168.122.1:8000/centos-minimal-ks.cfg>

New VM



Create a new virtual machine

Step 2 of 5

Provide the operating system install URL

URL:

▼ URL Options

Kickstart URL:

Kernel options:


☒ Automatically detect operating system based on install media

OS type: Linux

Version: Red Hat Enterprise Linux 6.5

The defaults are fine.

New VM



Create a new virtual machine

Step 3 of 5

Choose Memory and CPU settings

Memory (RAM): MiB


Up to 7845 MiB available on the host

CPU(s):

Up to 4 available

The defaults are fine.

New VM



Create a new virtual machine

Step 4 of 5

☒ **Enable storage for this virtual machine**

☒ **Create a disk image on the computer's hard drive**


9.0

–

+

GiB

206.9 GiB available in the default location

☐ Allocate entire disk now 

☐ **Select managed or other existing storage**

Browse...

Cancel

Back

Forward

Set static mac addresses match the ones above.


puppet 52:54:00:a8:7a:02

test03 52:54:00:a8:7a:03

16

Chapter 3. Virtualized test environment

New VM



Create a new virtual machine

Step 5 of 5

Ready to begin the installation

Name:

OS: Red Hat Enterprise Linux 6.5

Install: URL Install Tree

Memory: 1024 MiB

CPU: 1

Storage: 9.0 GiB /var/lib/libvirt/images/puppet.qcow2

☐ Customise configuration before install

▼ Advanced options

Virtual network 'default' : NAT ▼

☒ Set a fixed MAC address

Puppet Master

To manage the puppet master we are using a piece of software called **r10k**. In the interest of understanding how things work together we are going to do this part by hand rather than with puppet and leave it as an exercise to the reader to automate this.

<https://github.com/puppetlabs/r10k>

4.1 Install Puppet master

First ssh into you new puppet master server:

```
ssh root@puppet.corp.guest
```

Install puppet, git and rubygems on your puppet master then install r10k

```
yum -y localinstall http://yum.puppetlabs.com/puppetlabs-release-el-6.noarch.rpm
yum -y install puppet git rubygems vim
```

Create the puppet master certificates:

```
puppet cert generate $(hostname -f)
```

Configure apache as the webserver for puppet master:

```
yum localinstall http://www.mirrorservice.org/sites/dl.fedoraproject.org/pub/epel/6/i386/epel-release
yum -y install httpd mod_ssl mod_passenger rubygem-rack
```

There is a example apache vhost rack config file `/usr/share/puppet/ext/rack/example-passenger-vhost.conf` installed by the rpm for puppet. Copy it to `/etc/httpd/conf.d/` and edit it to make the certificate files match the hostname of your puppet master:

```
cp /usr/share/puppet/ext/rack/example-passenger-vhost.conf /etc/httpd/conf.d/puppet.conf
```

Link the ssl certificates to where the vhost config expects them to be and change the config certificate names to match the ones that were created with *puppet cert generate* earlier:

```
ln -s /var/lib/puppet/ssl /etc/puppet/ssl
sed -i 's/squigley.namespace.at/puppet.corp.guest/g' /etc/httpd/conf.d/puppet.conf
```

Make the directories for *Rack* and copy the example *rack config.ru* file into place:

```
mkdir -p /etc/puppet/rack/{public,tmp}
cp /usr/share/puppet/ext/rack/config.ru /etc/puppet/rack/
```

If you are using iptables you need to enable port 8140/tcp. This inserts a rule into the default centos fire wall at position 5, you may need to change it:

```
iptables -I INPUT 5 -p tcp -m state --state NEW -m tcp --dport 8140 -j ACCEPT
service iptables save
```

Fix selinux rules. the following *.te* file was created with *grep httpd /var/log/audit/audit.log| audit2allow -m httpd_passenger_socket*:

```
cat << EOF > httpd_passenger_socket.te

module httpd_passenger_socket 1.0;

require {
    type passenger_tmp_t;
    type httpd_t;
    class sock_file write;
}

#===== httpd_t =====
allow httpd_t passenger_tmp_t:sock_file write;
EOF
```

```
checkmodule -M -m -o httpd_passenger_socket.mod httpd_passenger_socket.te
semodule_package -o httpd_passenger_socket.pp -m httpd_passenger_socket.mod
semodule -i httpd_passenger_socket.pp
```

Create some required puppet directories:

```
mkdir /etc/puppet/manifests /etc/puppet/environments
```

We need to fix the permissions so that we don't have any permission errors:

```
chown -R puppet.puppet /etc/puppet/manifests /etc/puppet/environments /etc/puppet/rack
restorecon -R /etc/puppet/manifests /etc/puppet/environments /etc/puppet/rack
```

Make sure that apache will start on reboot:

```
chkconfig httpd on
service httpd restart
```

Puppet should now be running at <https://puppet.corp.guest:8140/> If you open that in a browser it will present the error string:

```
The environment must be purely alphanumeric, not ''
```

4.2 Install and configure r10k

You can install this using *gem install r10k* but I prefer to use RPM so I created a yum repo and built the rpms. You can use the following to install it via RPM:

```
urlgrabber -o /etc/yum.repos.d/timhughes-r10k-epel-6.repo https://copr.fedoraproject.org/coprs/timhu
yum -y install rubygem-r10k
```

Configure Directory Environments in */etc/puppet/puppet.conf*:

```
[main]
environmentpath = $confdir/environments
```

Note: Change this to use a git server at `git@puppet.corp.guest:fullstackpuppet-environment.git`

Create a config file for your r10k installation:

```
cat <<EOF >/etc/r10k.yaml
# The location to use for storing cached Git repos
:cachedir: '/var/cache/r10k'

# A list of git repositories to create
:sources:
  # This will clone the git repository and instantiate an environment per
  # branch in /etc/puppet/environments
  :opstree:
    #remote: 'https://github.com/fullstack-puppet/fullstackpuppet-environment.git'
    remote: '/var/lib/git/fullstackpuppet-environment.git'
    basedir: '/etc/puppet/environments'
EOF
```

Depending on how you access your git repo for the environment you may need to setup a ssh key or some other method of accessing the repo without a password.

Install your puppet manifests and modules

```
r10k deploy environment -pv
```

Create a cron job to update the puppet environments every minute:

```
cat << EOF > /etc/cron.d/r10k.conf
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
* * * * * root r10k deploy environment -p
EOF
```

4.3 Test installation

Check that it has all worked and you can compile the puppet manifest for your puppet master. You should be able to run the following command and get the YAML manifest for your puppet master.

```
curl --cert /etc/puppet/ssl/certs/puppet.corp.guest.pem \
--key /etc/puppet/ssl/private_keys/puppet.corp.guest.pem \
--cacert /etc/puppet/ssl/ca/ca.crt.pem \
-H 'Accept: yaml' \
https://puppet.corp.guest:8140/production/catalog/puppet.corp.guest
```

Puppet Manifests

5.1 Style guide

https://docs.puppetlabs.com/guides/style_guide.html

5.2 Create our base manifests

Create a new git repository and change directory into it. The default environment for puppet is named *production* and we are going to use a tool to automatically create environments based on git branch name so we create the default branch now.

```
git init fullstackpuppet-environment && cd $_
git checkout -b production
```

Create a *.gitignore* file, add it to the git index and then commit it.

```
cat << EOF > .gitignore
.ruby-version
EOF

git add .
git commit -m 'Initial commit'
```

Create the directory structure. The *site* directory is for your modules that used to be in your old monolithic code repository. Don't put anything into the *modules* directory as we are going to let *r10k* manage that later. This setup uses Directory Environments. To enable these we need to set *environmentpath* in the *main* section of our *puppet.conf* on the puppet-master which we will do later.

```
mkdir {site,manifests,modules}
```

```
cat << EOF > site/.gitignore
!.gitignore
EOF
```

This *.gitignore* file ensures that the modules folder will exist but the only thing managed by git in it will be the *.gitignore* file itself.

```
cat << EOF > modules/.gitignore
# Ignore everything in this directory
*
# Except this file
```

```
!.gitignore
EOF
```

Setup our environment configuration.

```
cat << EOF > ./environment.conf
# See https://docs.puppetlabs.com/puppet/latest/reference/config_file_environment.html
modulepath = modules:site
EOF
```

Create our initial site manifest. The *oldmodule* is from your old monolithic code repository. The *mymodule* we create in the next step.

```
cat << EOF > ./manifests/site.pp
# site.pp

# The filebucket option allows for file backups to the server
#
# $ puppet filebucket backup /etc/passwd
# /etc/passwd: 429b225650b912a2ee067b0a4cf1e949
# $ puppet filebucket get 429b225650b912a2ee067b0a4cf1e949
# $ puppet filebucket restore /tmp/passwd 429b225650b912a2ee067b0a4cf1e949
#
# See 'puppet help filebucket'
# Defaults to 'server => 'puppet''
filebucket { main: server => 'puppet' }

# Set global defaults - including backing up all files to the main
# filebucket and adds a global path
File { backup => main }
Exec { path => "/usr/bin:/usr/sbin:/bin:/sbin" }

node default {
  include ntp
}
EOF
```

Note: Change this to use a git server at [git@puppet.corp.guest:fullstackpuppet-ntp.git](https://github.com/rodjek/librarian-puppet)

The puppet file is what *r10k* uses to find and install our modules.

```
cat << EOF > ./Puppetfile
# https://github.com/rodjek/librarian-puppet

# Specify a puppet forge as the default source
forge "https://forgeapi.puppetlabs.com"

# Install puppetlabs from the default source
mod 'puppetlabs/stdlib'
mod 'puppetlabs/concat'

# Install from git repos
mod "ntp",
  #:git => "git://github.com/fullstack-puppet/fullstackpuppet-ntp.git"
  :git => "/var/lib/git/fullstackpuppet-ntp.git"
EOF
```

Add it all to git and push to your repository. Keep the commit message short but informative.

```
git add .  
git commit -a -m 'Initial files for puppet master'  
git push
```

Git Server

`yum install git`

<http://git-scm.com/book/en/v2/Git-on-the-Server-Setting-Up-the-Server>

```
which git-shell >> /etc/shells useradd -system -home /var/lib/git -create-home -skel /dev/null -  
shell $(which git-shell) git mkdir /var/lib/git/.ssh touch /var/lib/git/.ssh/authorized_keys chmod 700  
/var/lib/git/.ssh
```

```
su git -s /bin/sh -c 'git init --bare /var/lib/git/fullstackpuppet-environment.git' su git -s /bin/sh -c 'git init  
--bare /var/lib/git/fullstackpuppet-ntp.git'
```

```
cat /tmp/id_rsa.john.pub >> /var/lib/git/.ssh/authorized_keys
```

Static Analysis

7.1 Pre commit hooks

- http://projects.puppetlabs.com/projects/1/wiki/puppet_version_control
- <https://github.com/mattiasgeniar/puppet-pre-commit-hook/>
- <https://github.com/gini/puppet-git-hooks/>

The one by *gini* looks to be the most complete. It performs the following checks.

- puppet parser validate
- puppet-lint
- syntax check templates
- loads YAML and JSON files (Heira) to check their syntax

This script would be useful both on your local client version of git and also on a remote git if you arent using github or the like.

Install it in your local git repo:

```
urlgrabber -o .git/hooks/pre-commit \
https://github.com/gini/puppet-git-hooks/raw/master/hooks/pre-commit

chmod +x .git/hooks/pre-commit
```

If you wish to share the git hook with your team then a little trick is to add them to a hidden directory which is checked into git and symlink them into place like this:

```
mkdir .githooks/
urlgrabber -o .githooks/pre-commit \
https://github.com/gini/puppet-git-hooks/raw/master/hooks/pre-commit

ln -s ../../.githooks/pre-commit .git/hooks/pre-commit
chmod +x .githooks/pre-commit
git add .githooks
```

This way each member of your team just needs to create the symlink and you can all get updates to your hooks.

You may wish to look at a pre-push hook for some longer running things such as running puppet-rspec checks.

TODO: puppet-syntax module for syntax checking

Create Gemfile file

```
cat <<EOF> Gemfile
source "https://rubygems.org"
gem "puppet"
gem "puppetlabs_spec_helper"
gem "puppet-lint"
gem "rcov"
gem "rspec-puppet", :git => 'https://github.com/rodjek/rspec-puppet.git'
EOF
```

Create fixtures file

```
cat <<EOF > .fixtures.yml
fixtures:
  repositories:
    stdlib: "git://github.com/puppetlabs/puppetlabs-stdlib.git"
  symlinks:
    MODULE_NAME: "#{source_dir}"
EOF
```

Acceptance Testing

Acceptance testing for Puppet is done using [Beaker](#). This product was developed in house at Puppet labs and grew out of the old rspec-system tool. Documentation for Beaker is a little patchy but getting rapidly better.

Beaker can use loads of different cloud and virtualization infrastructures to create SUTs (System Under Test) but for our purposes we will use [Vagrant](#) with a couple of plugins [vagrant-libvirt](#) and [vagrant-mutate](#).

8.1 Vagrant setup

Install a virtualization platform.

```
sudo yum -y install qemu-kvm virt-install libvirt libvirt-daemon-kvm qemu-kvm libvirt-daemon-config-
```

To access virt-manager without entering password, create a file named /etc/polkit-1/rules.d/80-libvirt-manage.rules (or similar) with following content.

```
polkit.addRule(function(action, subject) {
  if (action.id == "org.libvirt.unix.manage" && subject.local && subject.active && subject.isInGroup
    return polkit.Result.YES;
  }
});
```

Remember to add your user to the wheel group:

```
sudo usermod -a -G wheel $USER
```

Download and install Vagrant from <https://www.vagrantup.com/downloads.html>. The Centos RPM works fine on fedora 21 and up.

```
sudo yum -y localinstall vagrant-X.X.X.rpm
```

Install dependencies for vagrant-libvirt and vagrant-mutate:

```
sudo yum -y install libvirt-devel libxslt-devel libxml2-devel lzma-devel qemu-img
```

Install the vagrant-libvirt plugin that allows you to use Libvirt as a virtualization provider and vagrant-mutate to convert boxes built for other platforms to ones that will run on Libvirt:

```
vagrant plugin install vagrant-libvirt
vagrant plugin install vagrant-mutate
```

Now we should be able to import a prebuilt box and convert it to Libvirt format. There are a selection of boxes provided by Puppetlabs and a much larger selection indexed on the Vagrant website. - <http://puppet-vagrant-boxes.puppetlabs.com/> - <http://www.vagrantbox.es/>

```
vagrant box add centos-65-x64-vbox436-nocm http://puppet-vagrant-boxes.puppetlabs.com/centos-65-x64-vbox436-nocm
vagrant mutate centos-65-x64-vbox436-nocm libvirt
```

Or you can use my prebuilt CentOS-6-x86_64-Minimal.box or CentOS-7-x86_64-Minimal.box available at <http://store01.timhughes.org/~tim/>:

```
vagrant box add CentOS-6-x86_64-Minimal http://store01.timhughes.org/~tim/CentOS-6-x86_64-Minimal.box
```

Create a *Vagrantfile* and change the box name inside it to match your box that you want to use:

```
vagrant init
```

Test that it all works by starting your Vagrant box and checking in *virt-manager*:

```
vagrant up --provider=libvirt
```

Jenkins

<http://jenkins-ci.org/>

- TODO: look at <https://github.com/matthewbarr/puppet-ci> for ideas
- TODO: should we use RVM on Jenkins instead of rbenv because of compile issue

Our jenkins server is easy if you have user the previous instructions as the puppet master will have a node definition for setting up a node called *jenkins.guest*. All you should need to do is SSH into your Jenkins server and then run puppet by hand the first time:

```
ssh root@192.168.122.3
puppet agent --test --waitforcert=60
```

It will attempt to connect to a server which resolves as *puppet*. If you have a problem check that you can ping the puppet server and then make sure you can connect to the puppet server on port 8140

```
ping puppet
```

```
telnet puppet 8140
```

Once the puppet job runs you may need to sign the certificate on the puppet master using the following command:

```
puppet cert sign jenkins.guest
```

You should now be able to connect to your Jenkins server on <http://192.168.122.3:8080>

For the first Jenkins task you will need to create it by hand so that you can get a config file to generate your other tasks.

- TODO: Add instructions on configuring Jenkins

Once you have created the first task and got it working you can get a copy of the config.xml and use it to create new jobs for other modules. The location for the config.xml file is http://192.168.122.3:8080/job/first_task/config.xml

Edit the xml to change the url to the git repository of your new module and then create a new job by posting the xml to the Jenkins server with a command similar to the following:

```
curl -X POST -d @config.xml -H "Content-Type: text/xml" -s http://192.168.122.3:8080/createItem?name=
```

More docs on the Jenkins API are available at <http://192.168.122.3:8080/api/>

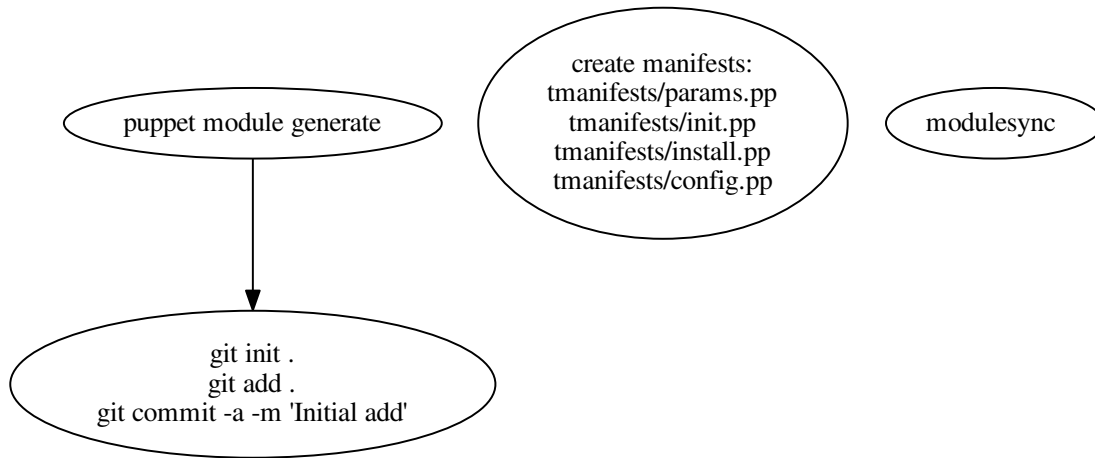
- **if all tests pass**
 - update Modulefile with new version and push as tag into git.
 - Have job that pushes it into pulp with new module version
 - trigger r10k to pull new module into puppet master from pulp puppet forge

10.1 Pre requisites

- docker
- vagrant
- rbenv

10.2 Workflow

- puppet module generate
- create manifests
- create .fixtures.yml
- git init . && git add . && git commit -a -m 'Initial add'
- git push to remote
- use modulesync to add extra files
- git pull
- add some tests and manifest code
- rake githooks
- rake lint
- rake validate
- rake spec
- rake spec/acceptance



TODO

Workspace

- rbnb

Puppet manifests Puppet modules Vagrant Puppet master Puppet agent Static Testing

- puppet validate
- puppet-syntax
- puppet-lint
- puppet-rspec

11.1 Items we need to cover and arrange in order

- install_ruby;
- create_module_template;
- install_gems;
- Module Testing
- Catalog Testing
- Dynamic Analysis
- Integration Testing
- Nightly Rebuilds
- Deployment

Install_Puppet_Master -> librarian_puppet/r10k -> sync_puppet_modules;

- sync_puppet_manifests;
- sync_hiera_data;
- hiera_gpg;
- jenkins_install_configure;
- git_repos_configure -> github_hook_jenkins;
- jenkins_push_to_puppet_master;

11.2 Resources

- http://docs.puppetlabs.com/guides/style_guide.html
- <https://www.youtube.com/watch?v=yUj0hsxGn6U>
- <http://puppetlabs.com/automation-2/video-continuous-integration-for-your-puppet-code>
- <http://forge.puppetlabs.com/mbarr/puppetci>
- <http://puppetlabs.com/blog/git-workflow-and-puppet-environments>
- <http://sysadminsjourney.com/content/using-git-submodules-dynamic-puppet-environments/>
- <http://puppetlabs.com/blog/testing-modules-in-the-puppet-forge>
This_is_good
- <http://somethingsinistral.net/blog/rethinking-puppet-deployment/>
- <https://github.com/adrienthebo/r10k>

Indices and tables

- `genindex`
- `modindex`
- `search`