Project Development using the NCI Cloud, Git and Puppet

Table of Contents

1. Introduction

This is an introduction to manage the development and testing of VMs within an NCI project on the NCI cloud using Puppet and git.

Prerequisites:

1. The Project (tenant), puppet and git repository has been set up for the project.
2. You have been registered as a user of OpenStack with NCI and connected to the project and the associated home directory has been set up on the Cloud login node.

2. Setting up your account for developing your project using the NCI Git repositories

2.1 Registering on the git repository machine

Login to **cloudlogin.nci.org.au** using your NCI account and password.

If you don't already have an ssh keypair in `~/.ssh/id_rsa` and `~/.ssh/id_rsa.pub`, then run the following to create a new passphrase-less key.

```
$ ssh-keygen -t rsa -N ""
```

Add this key to your account on the NCI repository machine:

```
$ ssh abc123@repos.nci.org.au authkeys add `cat ~/.ssh/id_rsa.pub`
```

where `abc123` is your NCI username.

Verify that you now have access to the git repositories by running:

```
$ ssh git@repos.nci.org.au info
```

The output will look like:

```
$ ssh git@repos.nci.org.au info
     hello abc123, this is git@repos running gitolite3 v3.04-22-g0d371ac on git 1.8.0

      R       p/..*
      R W C   p/access.dev/..*
      R       p/z00/..*
      R       p/z15/..*
      R       p/z16/..*
      R       p/z30/..*
          C   u/CREATOR/..*
      R       nci-puppet
      R W     p/access.dev/private/puppet
      R W     p/access.dev/puppet
      R       p/z00/private/puppet
      R       p/z00/puppet
      R       p/z00/testpuppet
```

You are now registered on the repos system, and should no longer need your password when ssh'ing to repos, or when doing git operations (eg git [clone|push|pull]).

2.2 Checking out the repo and setting up git for your project

Clone the repository by running the following in your cloudlogin home directory. Since you may be working on multiple projects, its best to create a suitably named directory for creating a local project repository. In the rest of the cases below we will assume you have created a project directory in your home directory called "your_project"

```
$ mkdir –p ~/your_project
$ cd your_project
```

If you like, you could create a little README file in this directory just as convenience to keep track of the purpose of this.

Next is to get a copy of the repository files for the project.

```
      $ git clone git@repos.nci.org.au:p/your_project/puppet
```

This will create a directory called "puppet", so change into it:

```
      $ cd puppet
```

You will notice that there is a file called .gitmodules and a directory called .git. These are used by git to keep track of your local repository.

Update the submodules by initialising:

```
      $ git submodule init
      [This modifies the file .git/config.]
```

then

```
      $ git submodule update
```

[This will perform updates in the modules subdirectory of puppet.]

Tell git who you are (important for keeping sanity in the commit logs):

```
      $ git config --global user.name "Firstname Lastname"
      $ git config --global user.email "your@email.address.com"

      [ This is recorded in ~/.gitconfig ]
```

Aside: If you are (or will be) working with other repositories on this host, and you would like to record different details, then run the following instead to set your name/email in just this repository:

```
$ git config user.name "Firstname Lastname"
$ git config user.email "your@email.address.com"
```

2.3 Creating your own development area in the repository system

The main git branch ("master") of the repository holds the configuration files for the service released as a production service. So that these production services are protected from the development and testing work, the permission to write to the master branch is restricted to people who manage the production service.

Therefore to develop and test services you will therefore need to create your own branch. To manage within our repository system, the name of the branch is to start with your NCI username. Note that it is normal to have many branches, and in particular to create a new branch for each “atomic task” you are working on.

On cloudlogin, make sure you are in your local copy of the puppet repository:

```
  $ cd ~/your_project/puppet
```

Create the local branch, using your login to replace `abc123`:

```
  $ git branch abc123/test
```

Verify that your new branch has been created by listing the available branches:

```
  $ git branch
       abc123/test
     * master
  $
```

The star `(*)` next to "master" indicates that it is your current branch. As is shown above, the "git branch" command above created the branch, but did not change to it. Note that the "git status" command will, among other things, tell you which branch you are currently on.

Change to using your branch:

```
  $ git checkout abc123/test
  Switched to branch 'abc123/test'
```

Now test again the status of your branch:

```
  $ git branch
  * abc123/test
    master
```

Aside: a quicker way to combine steps 2-3 is to run:

```
      $ git checkout -b abc123/newbranch
      Switched to a new branch 'abc123/newbranch'
```

This has created your branch locally, but it does not yet exist on the server. That needs to before it can be used by OpenStack instances as they need to contact the repos server.

Initially push this onto the server with:

```
     $ git push -u origin abc123/test

     Total 0 (delta 0), reused 0 (delta 0)
     To git@repos.nci.org.au:p/your_project/puppet
      * [new branch]      abc123/test -> abc123/test
     Branch abc123/test set up to track remote branch abc123/test
from origin.
```

Now when you run "`git pull`" from inside this branch, it will automatically sync with the same branch on the server. When you run "`git push`", it will automatically sync all of your branches on the server with your local branches. If you want to push just the branch you are on then “git push origin branch”.

You can find out what branches exist on the server by running either:

```
     $ git branch -r
       origin/HEAD -> origin/master
       origin/abc123/test
       origin/otheruser123/test
       origin/master
```

Alternatively, you can see all local as well as the remote branches using –a:

```
     $ git branch -a
     * abc123/test
       master
```

```
        remotes/origin/HEAD -> origin/master
        remotes/origin/abc123/test
        remotes/origin/otheruser123/test
        remotes/origin/master
```

3 Setting up your account to use OpenStack

Your OpenStack credentials have been created for you, and are stored in your home directory on cloudlogin.nci.org.au. They are automatically enabled when you login to cloudlogin. The OpenStack "nova" command-line tool is installed and available by default.

Set up the appropriate environment variables for working with OpenStack:

```
   $ . ~/.nci-os-creds-abc123.sh
```
(recall abc123 should be replaced by your username).

Aside: If you are working in multiple projects, then make sure you set your correct project name within the OS_TENANT_NAME environment variable.

Check that you can connect to OpenStack and that it knows who you are:
```
   $ nova credentials
```

You should see output resembling this:
```
+------------------+-------------------------------------------------------------------+
| User Credentials | Value                                                             |
+------------------+-------------------------------------------------------------------+
| id               | 77f4295edcc945e3ba72674f69cf825a                                  |
| name             | bje900                                                            |
| roles            | [{u'id': u'b2b048c7ad954a388577b4eb76699455', u'name': u'admin'}] |
| roles_links      | []                                                                |
| username         | bje900                                                            |
+------------------+-------------------------------------------------------------------+
+---------+----------------------------------------------------------------------------+
| Token   | Value
          |
+---------+----------------------------------------------------------------------------+
| expires | 2013-03-12T06:23:18Z                                                       |
| id      | b481f0ffdfed47b6a0a4b3ad28cc31b4                                           |
| tenant  | {u'id': u'c0cfc82c246a4c4a9d7e0645dbc80e7c', u'enabled': True, u'description': u'NCI Service Tenant',
u'name': u'z00'} |
+---------+----------------------------------------------------------------------------+
```

Register your ssh public key with OpenStack by running:
```
   $ nova keypair-add --pub-key ~/.ssh/id_rsa.pub abc123
```

In this case, "abc123" is the name being given to the keypair. You can choose any name you like, but your NCI username is as good as any.

Verify that your keypair has been registered correctly:
```
   $ nova keypair-list
+--------+-------------------------------------------------+
| Name   | Fingerprint                                     |
+--------+-------------------------------------------------+
| bje900 | 48:4c:f4:62:f5:9d:e2:fa:a5:35:43:62:3e:7c:7b:29 |
+--------+-------------------------------------------------+
```

Preparations for Tenant operations
By default, incoming traffic to the projects VMs are blocked. To be able to ssh in, you

need to open up the firewall.

Note, the following commands need to be run (only once) for any given nova installation. You will get an informational error message if you try to create a duplicate security group.

First, create some new "security groups", called "ssh" and "ping". (Aside: The names could be anything but registering them with service names are instructive).

```
      $ nova secgroup-create ssh "Allow SSH"
      $ nova secgroup-create ping "Allow ping"
```

Now add rules to these security groups to allow ssh and ping packets, respectively

```
      $ nova secgroup-add-rule ssh tcp 22 22 0.0.0.0/0
      $ nova secgroup-add-rule ping icmp -1 -1 0.0.0.0/0
```

4. Starting development instances on OpenStack using the Puppet/Git repo

4.1 Setting up your OpenStack management session on cloudlogin

Set up your session for working with OpenStack. This can be skipped if you already have it set up from the previous session. Ensure you are logged onto cloudlogin.nci.org.au.

You can see which VMs are currently running in your project by running:

```
   $ nova list
    +--------------------------------------+---------------------+--------+-----------------------------------+
   | ID                                   | Name                | Status | Networks                          |
   +--------------------------------------+---------------------+--------+-----------------------------------+
   | 5adfaab6-72b9-4219-a52f-97f2a50d0426 | puppet1             | ACTIVE | private=10.0.0.53, 192.43.239.145 |
   | a1dc1fcb-57d1-4bd9-ba9a-07aae729ae23 | puppet2.nci.org.au  | ACTIVE | private=10.0.0.52                 |
   | cc8bae2b-2afe-47a1-8950-49e76fa90866 | puppetbuntu         | ACTIVE | private=10.0.0.40                 |
   | 0fa3a00b-c0a4-4b49-99b0-b6d061f95464 | voltest             | ACTIVE | private=10.0.0.51, 192.43.239.148 |
   +--------------------------------------+---------------------+--------+-----------------------------------+
```

4.2 Using Nova commands to Boot and Shutdown a test VM instance

Choose a system image for the VM. Get the list of available images with:

```
   $ nova image-list
```

The "20121101" is the date the image was built, ie. the 1st of November, 2012. (Aside: Ignore some of the incorrectly formatted dates on images. We will need to clear these out at some point. The format for the future will be in the form os-version-YYYYMMDD).

Ignore any images that end in "`-kernel`" or "`-initrd`" - these are used internally by the actual disk images.

The `centos-6-20130416` is a good recent choice and we will use it below.

Choose a "flavor" (note American spelling) for the VM. This indicates the resources that will be provisioned for your VM:

```
$ nova flavor-list
+----+-----------+-----------+------+-----------+------+-------+-------------+-----------+-------------+
| ID | Name      | Memory_MB | Disk | Ephemeral | Swap | VCPUs | RXTX_Factor | Is_Public | extra_specs |
+----+-----------+-----------+------+-----------+------+-------+-------------+-----------+-------------+
| 1  | m1.tiny   | 512       | 0    | 0         |      | 1     | 1.0         | N/A       | {}          |
| 2  | m1.small  | 2048      | 10   | 20        |      | 1     | 1.0         | N/A       | {}          |
| 3  | m1.medium | 4096      | 10   | 40        |      | 2     | 1.0         | N/A       | {}          |
| 4  | m1.large  | 8192      | 10   | 80        |      | 4     | 1.0         | N/A       | {}          |
| 5  | m1.xlarge | 16384     | 10   | 160       |      | 8     | 1.0         | N/A       | {}          |
+----+-----------+-----------+------+-----------+------+-------+-------------+-----------+-------------+
```

For very simple testing and debugging flavor 1 (m1.tiny) is sufficient. For most general-purpose uses, flavor 2 (m1.small) is a reasonable choice.

Booting a VM instance is ordinarily done with the "`nova boot ...`" command. However, a wrapper script has been developed at NCI to simply the booting of VMs that will be using Puppet. The script is called "`nova-boot`" and lives in the "`tools`" subdirectory of the main puppet repository (see Section 2.2). It needs to be run with the current directory being the base of the main Puppet repository for the project.

```
$ cd puppet
```

Get help on using the script with:

```
$ tools/nova-boot --help
```

Now, run the script to boot the VM (note this is all one line):

```
$ tools/nova-boot --name yourVMname --repo git@repos.nci.org.au:p/your
_project/puppet --branch abc123/test -- --image centos-6-20130416 --flavor 2
--key-name abc123 --security-groups ssh,ping
```

Recall that `abc123/test` is the repository branch that we set up earlier.

The script will wait for the VM to be built, ie. when the script exits, the machine starts booting. This can sometimes take a while though system improvements are on the way.

Check using the `nova console-log` command, and wait for the final login prompt to appear to know that the sequence is completed:

```
$ nova console-log abc123test

    ….
    CentOS release 6.3 (Final)
    Kernel 2.6.32-279.11.1.el6.x86_64 on an x86_64

    abc123.local login:
```

Aside: There is a known bug at the moment where sometimes the VM will stall just after starting sshd
Starting sshd: [OK]
If that happens, delete the VM and restart until you get a successfully started VM as per the above.

If you have a floating IP (an IP that is accessible from the outside world) then you can now attach it. Check with your other project members about the assignments of IP to the project. You can see the list by looking at

```
$ nova floating-ip-list
```

Once you have chosen an IP address for the VM, then attach it to the VM using:

```
$ nova add-floating-ip your_vm your.floating.ip.address.com
```

When you are finished with your test VM you will want to shut it down.
To shutdown:
Login to your VM and run "halt" for a gracious shutdown.
From cloudlogin check the console-log to see it shutting down.

```
    $ nova console-log your_vm
```

Then, if you have a floating-ip you must detach it from the VM first.
$ nova remove-floating-ip your_vm your.floating.ip.address.org

Check that the floating IP address has disappeared by looking at `nova list` and ensuring it has been removed.

```
Then:
   $ nova delete your_vm
```

4.3 Logging onto your test VM instance

This VM does not have a public IP address. Instead it only a 10.0.0.xxx internal IP to the OpenStack cloud. Access to it is via the cloudlogin.nci.org.au node.

While logged onto cloudlogin, ssh to your VM by simply doing:

```
$ ssh root@10.0.0.xxx
```

or, if you need X11 forwarding enabled, do:

```
$ ssh -Y root@10.0.0.xxx
```

If you need external access to certain ports on your VM, eg. Port 80 for web development testing, then you will need to setup ssh port forwarding when you connect to cloudlogin, eg:

```
$ ssh -L 12080:10.0.0.xxx:80 -g cloudlogin.nci.org.au
```

In the first instance, this probably won't be needed.

When on your VM, you can have a look around.

```
   $ hostname –f
```

The Puppet repo lives in `/puppet`:

```
    $ ls -la /puppet
```

The output from the NCI/Puppet-specific boot script is in `/var/log/userdata.log`. This is good to check for errors or problems during bootup. Puppet output is coloured, so use the –R flag of the 'less' program:

```
    $ less –R /var/log/userdata.log
```

5. Developing/Testing VM puppet configurations

There are two ways to make changes and test. The first is described in 5.1. It involves booting a test VM and make modifications to it. Once happy with the changes you can commit the change to the server version of the repository and restart the test server for a final check.

The second method is described in 5.2. It involves making changes from the cloudlogin and then push the changes to the test VM.

The advantage of the first is that you can develop changes more freely without recording the inevitable failed attempts and polluting the repository change history. The second method could be used for making a larger "merge" of others changes.

In any case, both are instructive to see.

5.1 Testing on a currently running VM

Boot up a test VM and login as root as per steps in Section 4.

Change into the puppet repository:

```
  bash# cd /puppet
```

The repository is already in the branch that you specified when running tools/nova-boot. You can check that you are in your test branch by doing a "git branch –a", and if you are in the wrong branch then do a "git checkout <branch>" to be in the correct one.

You can now edit and work on the test VM version of the puppet config as necessary. See section 6 on the Puppet configs and some sample changes that you can apply.

To apply your changes on a running system, issue the following command:

```
     bash# puppet apply /puppet/manifests/site.pp
```

Before the next step (committing), you need to set your $VISUAL environment variable to your preferred editor (the default value of "vi" is not installed, which will lead to errors). If you are comfortable with vi/vim, then do:

```
     export VISUAL=vim
```

If you would prefer to use emacs, then you will need to add emacs to your VMs puppet configuration before setting $VISUAL to emacs:

```
     class { "emacs": }
```

When you are happy with your changes, you can commit them into the git repository. Any newly created files must be first added to the index by

```
 "git add file"
```

followed by:

```
   git commit –m “added file”

Or “git commit –a ”
```

This has committed the changes to the local git repository, but not updated the server. Push to the server with:

```
   git push
```

Aside: If you get errors with the above, you might need to incorporate changes that are already on the server (or you might want to do this in any case). To do this, run:

```
git pull
```

5.2 Adjusting your test VM configuration when it is not running

If you are working in the /puppet area, first commit your changes by running:

```
$ git commit -a
```

and enter a suitable commit message. eg. "Added support for syslog." By convention, git commit messages are a single line with a brief description, followed by a blank line and then any further detail necessary.

Push your changes to the central repository by running:

```
$ git push
```

Aside: If you get errors with the above, you might need to incorporate changes that are already on the server (or you might want to do this in any case). To do this, run:

```
git pull
```

Then boot up your test VM as usual.

Aside: If at this point you realise that you were actually running a test VM, then it won’t have picked up these changes. But you can update your running VM by logging into your VM and run the "puppet-update" script:

```
$ ssh root@10.0.0.xxx
$ puppet-update
```

or, to do this as a single command:

```
$ ssh root@10.0.0.xxx puppet-update
```

The script will then pull down the latest version of the repository and then have Puppet apply it to the running system. You should see output indicating that the necessary packages are being installed, etc, and once it has finished (it could be many minutes as packages are fetched, in some cases compiled, and installed) you should see system logs being written in /var/log.

6. Managing the Puppet configuration for a project.

6.1 Navigating the Puppet repository for a VM

So far, your VM will just be using the "default" definition of a node. We can adjust this definition for your VM and commit the changes as per Sections 5.1 or 5.2.

For a VM configuration, change into the relevant Puppet repo and switch to your branch:

```
$ cd puppet
```

Aside: You can check that you are in your test branch by doing a “git branch –a”, and if you are in the wrong branch then do a “git checkout <branch>” to be in the correct one.

Open the file `manifests/nodes.pp` in an editor. This file sets some default variables and has the basic default node definition:

```
$default_allow_access = "(access.dev) (access.admin) ycx548"
$default_production_fqdn = "accessdev.nci.org.au"

node default {
     class { “accessdev-node”:
     }       }
```

The comments at the start of the file explain the variables, which indicate the list of users and groups that are allowed to login, and the hostname that is considered to host the “production” service.

In the example above, by default all nodes use the definitions in the “accessdev-node” class, which is defined in the modules/accessdev-node/manifests/init.pp file. This file contains some code to detect whether or not the current hostname is the production hostname, and then goes on to define the make-up of an “accessdev” node – this includes both the main production accessdev.nci.org.au machine, as well as any other development nodes that may be created by users (which could be named anything).

As an example, add support for system logfiles by adding the following to the accessdev-node class definition:

```
     class { "syslog-ng": }
```

6.2 Adding standard packages to the CentOS image provided within the NCI OpenStack

As an example of how to make changes, we are going to look at adding standard software packages that are provided by the base Linux distribution, eg. CentOS. We will use cron as an example. We also essentially assume that this is performed on a test VM so that any changes are easy to track and show are working.

First, determine the name of the package. On CentOS, use the command “`yum search foo`” to find a list of packages that match the keyword “foo” (you can use cloudlogin as a convenient CentOS 6 machine). In the case of cron, this gives:

```
[abc123@cloudlogin ~]$ yum search cron
Loaded plugins: fastestmirror
Determining fastest mirrors
 * base: centos.mirror.uber.com.au
 * epel: fedora.mirror.uber.com.au
 * extras: centos.mirror.uber.com.au
 * updates: centos.mirror.uber.com.au
base                                                     | 3.7 kB     00:00
epel                                                     | 4.3 kB     00:00
epel/primary_db                                          | 5.0 MB     00:00
extras                                                   | 3.5 kB     00:00
updates                                                  | 3.5 kB     00:00
updates/primary_db                                       | 5.1 MB     00:00
epel/pkgtags                                             |  327 B     00:00
================================ N/S Matched: cron =================================
PackageKit-cron.x86_64 : Cron job and related utilities for PackageKit
cronie.x86_64 : Cron daemon for executing programs at set times
cronie-noanacron.x86_64 : Utility for running simple regular jobs in old cron style
crontabs.noarch : Root crontab files used to schedule the execution of programs
incron.x86_64 : Inotify cron system
perl-Schedule-Cron-Events.noarch : Take a line from a crontab and find out when
                                 : events will occur
perl-Set-Crontab.noarch : Expand crontab(5)-style integer lists
yum-cron.noarch : Files needed to run yum updates as a cron job
cronie-anacron.x86_64 : Utility for running regular jobs
cronolog.x86_64 : Web log rotation program for Apache
rpm-cron.noarch : Create daily logs of installed packages.

  Name and summary matches only, use "search all" for everything.
[abc123@cloudlogin ~]$
```

So this shows us that the name of the package we want is probably "cronie". We can check with the "`yum info cronie`" command:

```
[abc123@cloudlogin ~]$ yum info cronie
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: centos.mirror.uber.com.au
 * epel: fedora.mirror.uber.com.au
 * extras: centos.mirror.uber.com.au
 * updates: centos.mirror.uber.com.au
Available Packages
Name        : cronie
Arch        : x86_64
Version     : 1.4.4
Release     : 7.el6
Size        : 70 k
Repo        : base
Summary     : Cron daemon for executing programs at set times
URL         : https://fedorahosted.org/cronie
License     : MIT and BSD and ISC and GPLv2
Description : Cronie contains the standard UNIX daemon crond that runs specified
            : programs at scheduled times and related tools. It is a fork of the
            : original vixie-cron and has security and configuration enhancements
            : like the ability to use pam and SELinux.
```

Alternatively, to find which package provides a given file (eg. a command), use the "`yum resolvedep`" command, eg:

```
[abc123@cloudlogin ~]$ yum resolvedep /usr/bin/crontab
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
```

```
 * base: centos.mirror.uber.com.au
 * epel: fedora.mirror.uber.com.au
 * extras: centos.mirror.uber.com.au
 * updates: centos.mirror.uber.com.au
0:cronie-1.4.4-7.el6.x86_64
```

So, within the base puppet repo we create a module for the package by creating the `modules/cron` directory, and then a location within that called "`manifests`" to store the Puppet manifest files:

```
cd puppet
mkdir –p modules/cron
mkdir modules/cron/manifests
```

Now create a Puppet manifest file called `modules/cron/manifests/init.pp`, containing:

```
class cron {
    package { "cronie":
        ensure => present,
    }
}
```

This defines what it means for nodes to be classified as "cron"-nodes.

You now need to make your VM (ie. accessdev) belong to this class of nodes. Update the `modules/accessdev-node/manifests/init.pp` file, and add the following line inside the accessdev-node class definition:

```
class accessdev-node … {
    …
    class { "cron": }
}
```

Run the "`puppet-update`" script as usual to apply these changes to test this on a running VM.

However, simply installing the package is not sufficient – the cron *service* must be enabled. To do this, first determine the name of the service by examining the relevant files in `/etc/init.d`:

```
-bash-4.1# ls -la /etc/init.d/*cron*
-rwxr-xr-x 1 root root 2793 Jul 19  2011
/etc/init.d/crond
-bash-4.1#
```

So the service name is "crond". Now, edit the `modules/cron/manifests/init.pp` file to be:

```
class cron {
    package { "cronie":
        ensure => present,
    }
    service { "crond":
        ensure => running,
        require => Package["cronie"],
    }
}
```

These additions indicate that the crond service should be running, and that if Puppet needs to start it, that should be done after installing the package (which is obvious to humans, but Puppet needs this spelled out).

To invoke this on a running VM, then re-run "puppet-update" and check that the service has been started by running the command "`service crond status`".

Finally, you will need to commit your changes to git. However, this should always be done in a way that separates the actual module *definition*, from any *usage* of it. Doing this makes applying the module to other git branches or repositories (such as the main upstream `nci/puppet` repo) much easier. So, review the changes that have been made by running git status:

```
$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   modules/accessdev-node/manifests/init.pp
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       modules/cron/
no changes added to commit (use "git add" and/or "git commit -a")
$ git diff
diff --git a/modules/accessdev-node/manifests/init.pp b/modules/accessdev-
node/manifests/init.pp
index 82a351d..da41566 100644
--- a/modules/accessdev-node/manifests/init.pp
+++ b/modules/accessdev-node/manifests/init.pp
@@ -89,5 +89,7 @@ class accessdev-node ($production = undef, $production_fqdn =
$default_productio
                message => "Testing system status. Please try again later.",
        }

+       class { "cron": }
+
 }
$
```

Notice that the untracked files aren't diffed, since of course all their contents are new.

First, commit just the `modules/cron` directory (and all its sub-contents) by doing:

```
$ git add modules/cron/
$ git commit
```

and entering a suitable log message (eg. “Added cron module for enabling access to cron jobs.”). Then simply add the rest of the changes (which is just the change to the accessdev-node module) by running “`git commit –a`” (and a suitable message, eg. “Updated the accessdev-node definition to use cron.”).

You can now check the history (including diffs) by running “`git log -p`”.

```
$ git log -p
commit ceb6108958e5df0ccf60c3d065c286c7a2aad005
Author: Kevin Pulo <kevin.pulo@anu.edu.au>
Date:   Mon Feb 18 12:07:05 2013 +1100

    Updated accessdev-node definition to use cron.

diff --git a/modules/accessdev-node/manifests/init.pp b/modules/accessdev-
node/manifests/init.pp
index 82a351d..da41566 100644
--- a/modules/accessdev-node/manifests/init.pp
+++ b/modules/accessdev-node/manifests/init.pp
@@ -89,5 +89,7 @@ class accessdev-node ($production = undef, $production_fqdn =
$default_productio
                message => "Testing system status. Please try again later.",
        }

+       class { "cron": }
+
 }

commit 9b1c75517d8045138683cffe6a090eab542a3d47
Author: Kevin Pulo <kevin.pulo@anu.edu.au>
Date:   Mon Feb 18 12:06:26 2013 +1100

    Added cron module for enabling access to cron jobs.

diff --git a/modules/cron/manifests/init.pp b/modules/cron/manifests/init.pp
new file mode 100644
index 0000000..753d0fa
--- /dev/null
+++ b/modules/cron/manifests/init.pp
@@ -0,0 +1,11 @@
+
+class cron {
+       package { "cronie":
+               ensure => present,
+       }
+       service { "crond":
+               ensure => running,
+               require => Package["cronie"],
+       }
+}
+

…
$
```

You can now upload your changes to the central repo by doing “`git push`”.

7. Working with Git Branches

In Git, a *branch* is the equivalent of a patch, and *merge* of a branch is the equivalent of applying that patch.

7.1 Finding and tracking someone else's branch

When you run "git branch -a" you will see other people's branches listed as remote branches. To see the contents of one of these branches (and/or work with/on it), you must create a local branch that "tracks" the remote branch:

```
      $ git checkout -b xyz789/test origin/xyz789/test
```

Branch xyz789/test set up to track remote branch xyz789/test from origin.
Switched to a new branch 'xyz789/test'
$

7.2 Merging a branch

Suppose you have been working in a local branch, and are happy with the contents of it. To now merge it back into the master branch (or any other branch), do:

```
      $ git checkout master
      $ git merge xyz789/test
      Updating 5ad7234..1236871
      Fast-forward
       manifests/nodes.pp |   16 ++++++++++++++++
       1 files changed, 16 insertions(+), 0 deletions(-)
      $
```

Or to merge a specific list of commits, do:

```
      $ git merge
387b5c218c7c7f46f8d63f8e0e732e99d9bb1a2895c84786a033ccf85d3329
bb33fcbaf4bfcb3988 ...
```

If git tells you that there is a conflict then to abort the merge, simply run

```
      $ git merge --abort".
```

To resolve the conflict, you can just edit the files in an editor (the syntax is a little strange, but usually easy enough to follow). Alternatively you can use

```
      $ git mergetool".
```

At this stage, "git diff" will show you three-way diffs. Once the conflicts have been resolved, just commit the files as per normal with "git commit -a" (or "git add file" followed by "git commit").

Another less-commonly used alternative to merging is "rebasing". Rebasing is only really used when playing in a local branch. Use merge unless you are confident that you understand what rebasing is, and that it is appropriate for the task at hand.

```
      $ git rebase xyz789/test
```

7.3. Deleting a local branch

If the branch has been fully merged into another, then you can use:

```
      $ git branch -d abc123/test
```

If the branch has NOT been fully merged into another, and you want to throw away the contents of the branch, then you can use:

```
      $ git branch -D abc123/test
```

7.4. Deleting a branch from the server

Deleting a branch from a server can be useful. This is achieved by pushing "nothing" into the branch on the server, effectively wiping it out. The command to do this is:

```
      $ git push origin :abc123/test
```

(The "nothing" is the part before the ":", ie. ordinarily, you could do "git push origin abc123/other:abc123/test" to push your local "abc123/other" branch into the "abc123/test" branch on the server. When the source branch ("abc123/other") is empty (""), then git interprets that as meaning to delete the remote branch.)

7.5. Merging in 'upstream' changes from the NCI/Puppet repository.

First, add a 'remote' which is the upstream repository:

```
      $ git remote add upstream git@repos.nci.org.au:nci/puppet
      $ git remote update upstream
```

Now, create a local branch to track the master branch of the NCI/puppet repository:

```
      $ git checkout -b upstream remotes/upstream/master
```

Run "git pull" on this branch to ensure it is up-to-date. Have a look around (eg. "git log") and make sure that you are happy with the contents. Before merging it directly into your master branch (or whatever branch you might be working on), it's a good idea to test the merge for conflicts. Create a branch for that purpose and change to use it (the following assumes you want to merge into the master branch):

```
      $ git checkout -b testmerge master
```

Now you can try the merge:

```
      $ git merge upstream
```

If the merge is successful, then you can do:

```
      $ git checkout master
      $ git merge upstream
      $ git branch -D testmerge
```

To redo the merge in your master branch and then delete the testmerge branch.

If the merge failed, you can investigate it and try to resolve the conflict. If you successfully resolve the merge, then after the merge-concluding "git commit" you can do

```
      $ git checkout master
      $ git merge testmerge
      $ git branch -d testmerge
```

to subsequently merge the test branch with master. (This second merge will always work, since it will be a fast-forward.)

If you want to abandon your merge attempt and start again, then you can simply do

```
        $ git checkout master
        $ git branch -D testmerge
        $ git checkout -b testmerge master
        $ git merge upstream
```

to try again.

7.6 Selectively merging from other branches. Cherrypicking.

8. Common administration tasks

8.1 Upgrading a production VM while in service

After development work from developers personal branches has been locally tested, and then merged into the master branch, it must be propagated to the running "prodserver" service.

For small, simple updates with low risk, the changes (once pushed to the master branch on the repos server) can be updated on prodserver simply by running the following from the test instance:

$ ssh root@<prodserver>.nci.org.au puppet-update

However, any substantial changes should first be thoroughly tested in a simulated production environment before being applied to the production service. To do this, define a production test node as follows:

```
node "prodserver-test.nci.org.au" {
        class { "prodserver-node":
                production => true,
        }
}
```

And then boot it with a nova-boot command that is the same as prodserver except using the prodserver-test.nci.org.au IP address, eg:

```
$ tools/nova-boot --name prodserver-test.nci.org.au --ip
192.43.239.151 --repo git@repos.nci.org.au:p/myproject/puppet --
--image centos-6-20130416 --flavor 2 --key-name abc123 --security-
groups ssh,ping
```

This machine can then be used to test the production environment with the new changes in the repository. When satisfied, the changes can be applied to the actual running prodserver VM as described above.

If preferred, the production server can be shutdown, killed with "nova delete", and then booted afresh.

8.2 Allowing user direct access

The list of usernames and groups that are allowed to login via ssh is kept in the `manifests/nodes.pp` file. It is defined in the `$default_allow_access` variable, and takes the form:

```
$default_allow_access = "(usergroup1) (unixgroup2) abc123
bcd321"
```

where the variable is a space separated list of usernames and groups. Groups must be enclosed in parentheses. This variable is automatically used by the prodserver-node class.

To define a particular node with its own set of allowed users, it can be overridden by passing the allow_access parameter to the prodserver-node class, eg:

```
node "abc900.local" {
        class { "prodserver-node":
                allow_access => "abc900 (z00)",
        }
}
```

8.3 Adjusting the firewall

To be documented.

8.4 Setting up logging

Not yet implemented, but when it is, it will take the substantive form:

```
class { [ "idmapd", "nfsclient", ]: }
nfs-mount { "/var/log":
    source => "os-home.nci.org.au:/data/exports/$fqdn/log",
    require => Class["ldap::client"],
}
class { [ "syslog-ng", "logrotate", "cron", ]:
    require => Nfs-Mount["/var/log"],
}
```

8.5 Adding, installing or updating software hosted in a version control

repository

TBA. For the time being, refer to the cycl and rose modules as more-or-less exemplars for this use case.

Appendix A

Q1. How can I tell what Puppet classes exist and can be loaded?

A. In your local repository look in the modules directory. Each module has a file called `manifests/init.pp.` This describes the class and what it does.

`package [ 'xxx' ]` relates to a standard package available to an operating system's standard package manager. For example, for the centos6 VM's created with this guide,

```
package { ['xxx',]:  ensure => present, }
```

will simply run

```
      'yum -y install xxx'.
```

If that package doesn't exist, `'puppet apply'` will fail. For an exhaustive list of packages available (14,170 at last count), type:

```
      yum list available
```

For simple packages, this is usually all that is required to install them on your VM. If you do not see a package you were looking for, you can create your own puppet class to add it. Looking at existing puppet modules is a good way to get some ideas.

Q2. My VM doesn't appear to be working correctly.

A. There could be a few reasons:

1) If some puppet scripts do not complete fully without error, then any number of classes could have failed. If one class fails, there is no guarantee that any other unrelated classes successfully completed. Look in `/var/log/userdata.log` if it failed at boot, or look at the output from puppet apply `/puppet/manifests/site.pp`.

2) If you've logged in and nothing at all appears to work, check
`nova console-log <instance>`
or
`/var/log/userdata.log.`

In particular, if the last line is "Starting sshd: [OK]" then the full boot process did not complete and so `tools/nova-boot` has not run correctly. This may just be an intermittent problem with the underlying system rather than a problem in the configuration. In the first instance, just delete your VM as described in 4.2 and run tools/nova-boot again.

Appendix B: Useful Resources

B1. Learning Puppet

An excellent tutorial, from the very basics of Puppet up to advanced usage.
http://docs.puppetlabs.com/learning/

B2. Puppet Forge

Community-contributed puppet modules.
http://forge.puppetlabs.com/

B3. Git References

Git for SVN users crash course

Excellent intro to git for people familiar with Subversion.
https://git.wiki.kernel.org/index.php/GitSvnCrashCourse

Git pro book

http://git-scm.com/book

Git immersion

http://gitimmersion.com/

Git from the bottom up

http://ftp.newartisans.com/pub/git.from.bottom.up.pdf

Git for the confused

http://www.gelato.unsw.edu.au/archives/git/0512/13748.html