End Lab 01:02:05

Caution: When you are in the console, do not deviate from the lab instructions. Doing so may cause your account to be blocked. Learn more.

puppet external IP address
35.196.157.9

username
student-01-3f959667c1b

Download PEM
Download PPK

15/15

Finish a Puppet deployment

1 hour 30 minutes Free ★★★★☆

Introduction

You want to automatically manage the computers in your company's fleet, including a number of different machines with different operating systems. You've decided to use Puppet to automate the configurations on these machines. Part of the setup is already done, but there are more rules that need to be added, and more operating systems to consider.

Prerequisite

Understands the basics of Puppet, including:

- How to create Puppet classes and rules
- How Puppet interacts with different OSs
- How to use the DSL to create complex rules

You'll have 90 minutes to complete this lab.

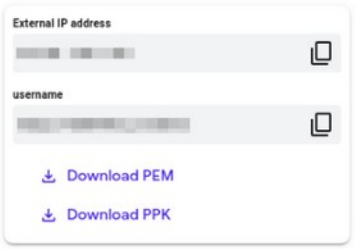
Start the lab

You'll need to start the lab before you can access the materials in the virtual machine OS. To do this, click the green "Start Lab" button at the top of the screen.

Note: For this lab you are going to access the **Linux VM** through your **local SSH Client**, and not use the **Google Console** (**Open GCP Console** button is not available for this lab).

Start Lab

After you click the "Start Lab" button, you will see all the SSH connection details on the left-hand side of your screen. You should have a screen that looks like this:



Accessing the virtual machine

Please find one of the three relevant options below based on your device's operating system.

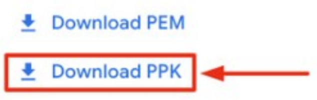
Note: Working with Qwiklabs may be similar to the work you'd perform as an **IT Support Specialist**; you'll be interfacing with a cutting-edge technology that requires multiple steps to access, and perhaps healthy doses of patience and persistence(!). You'll also be using **SSH** to enter the labs -- a critical skill in IT Support that you'll be able to practice through the labs.

Option 1: Windows Users: Connecting to your VM

In this section, you will use the PuTTY Secure Shell (SSH) client and your VM's External IP address to connect.

Download your PPK key file

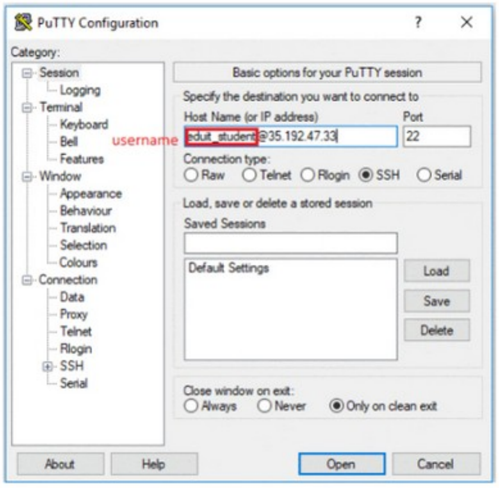
You can download the VM's private key file in the PuTTY-compatible **PPK** format from the Qwiklabs Start Lab page. Click on **Download PPK**.



Connect to your VM using SSH and PuTTY

1. You can download Putty from here
2. In the **Host Name (or IP address)** box, enter username@external_ip_address.

Note: Replace **username** and **external_ip_address** with values provided in the lab.



Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): edut_student@35.192.47.33   Port: 22

Connection type: ( ) Raw ( ) Telnet ( ) Rlogin (●) SSH ( ) Serial

Load, save or delete a stored session

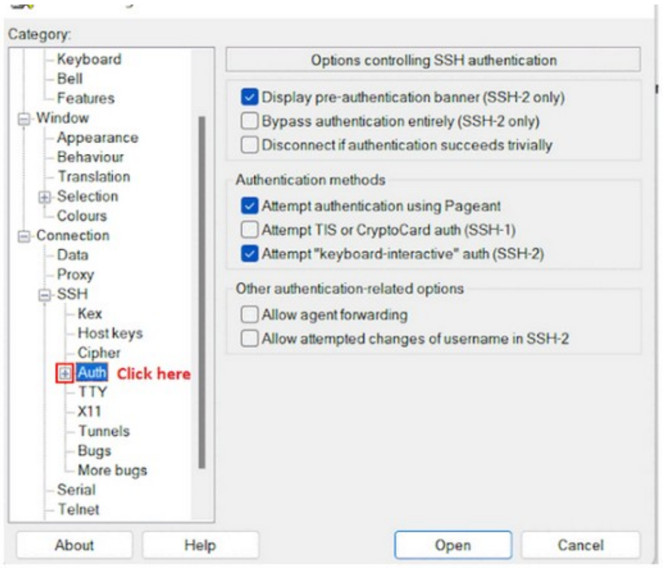
Saved Sessions

Default Settings

Close window on exit: ( ) Always ( ) Never (●) Only on clean exit

3. In the **Connection** list, expand **SSH**.
4. Then expand **Auth** by clicking on **+** icon.
5. Now, select the **Credentials** from the **Auth** list.
6. In the **Private key file for authentication** box, browse to the PPK file that you downloaded and double-click it.
7. Click on the **Open** button.

Note: PPK file is to be imported into PuTTY tool using the Browse option available in it. It should not be opened directly but only to be used in PuTTY.


Category:
- Keyboard
- Bell
- Features
- Window
  - Appearance
  - Behaviour
  - Translation
  - Selection
  - Colours
- Connection
  - Data
  - Proxy
  - SSH
    - Kex
    - Host keys
    - Cipher
    - Auth **Click here**
    - TTY
    - X11
    - Tunnels
    - Bugs
    - More bugs
  - Serial
  - Telnet

Options controlling SSH authentication

- [✓] Display pre-authentication banner (SSH-2 only)
- [ ] Bypass authentication entirely (SSH-2 only)
- [ ] Disconnect if authentication succeeds trivially

Authentication methods

- [✓] Attempt authentication using Pageant
- [ ] Attempt TIS or CryptoCard auth (SSH-1)
- [✓] Attempt "keyboard-interactive" auth (SSH-2)

Other authentication-related options

- [ ] Allow agent forwarding
- [ ] Allow attempted changes of username in SSH-2

8. Click **Yes** when prompted to allow a first connection to this remote SSH server. Because you are using a key pair for authentication, you will not be prompted for a password.

Common issues

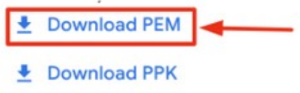
If PuTTY fails to connect to your Linux VM, verify that:

- You entered **<username>@<external ip address>** in PuTTY.
- You downloaded the fresh new PPK file for this lab from Qwiklabs.
- You are using the downloaded PPK file in PuTTY.

Option 2: OSX and Linux users: Connecting to your VM via SSH

Download your VM's private key file.

You can download the private key file in PEM format from the Qwiklabs Start Lab page. Click on **Download PEM**.



Connect to the VM using the local Terminal application

A **terminal** is a program which provides a **text-based interface for typing commands**. Here you will use your terminal as an SSH client to connect with lab provided Linux VM.

1. Open the Terminal application.

   - To open the terminal in Linux use the shortcut key **Ctrl+Alt+t**.

   - To open terminal in **Mac** (OSX) enter **cmd + space** and search for **terminal**.

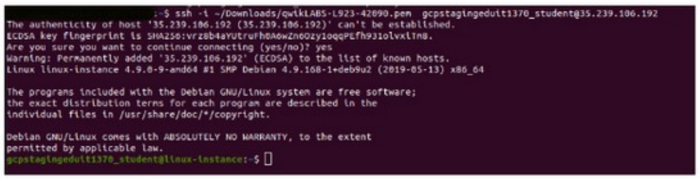
2. Enter the following commands.

> **Note:** Substitute the **path/filename for the PEM** file you downloaded, **username** and **External IP Address**.

You will most likely find the PEM file in **Downloads**. If you have not changed the download settings of your system, then the path of the PEM key will be **~/Downloads/qwikLABS-XXXXX.pem**

```
chmod 600 ~/Downloads/qwikLABS-XXXXX.pem
```

```
ssh -i ~/Downloads/qwikLABS-XXXXX.pem username@External Ip
Address
```



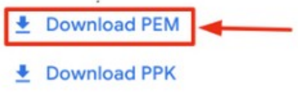
```
[redacted]:~$ ssh -i ~/Downloads/qwikLABS-L923-42090.pem  gcpstagingeduit1370_student@35.239.106.192
The authenticity of host '35.239.106.192 (35.239.106.192)' can't be established.
ECDSA key fingerprint is SHA256:vrz8b4aYUtruFh0AowZn6Ozy1oqqPEfh931olvxiTm8.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '35.239.106.192' (ECDSA) to the list of known hosts.
Linux linux-instance 4.9.0-9-amd64 #1 SMP Debian 4.9.168-1+deb9u2 (2019-05-13) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
gcpstagingeduit1370_student@linux-instance:~$ 
```

Option 3: Chrome OS users: Connecting to your VM via SSH

Note: Make sure you are not in **Incognito/Private mode** while launching the application.

Download your VM's private key file.

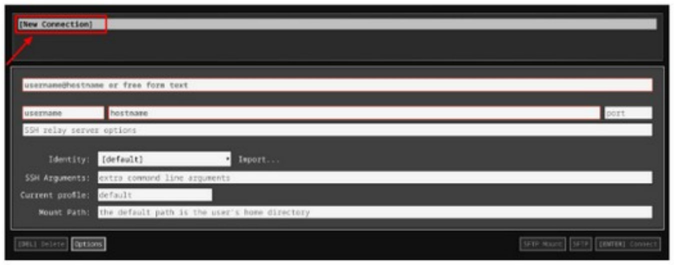
You can download the private key file in PEM format from the Qwiklabs Start Lab page. Click on **Download PEM**.



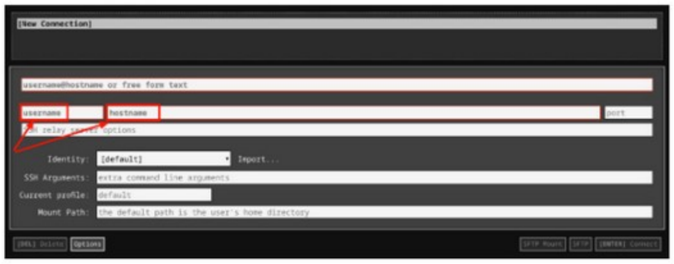
Connect to your VM

1. Add Secure Shell from here to your Chrome browser.

2. Open the Secure Shell app and click on **[New Connection]**.



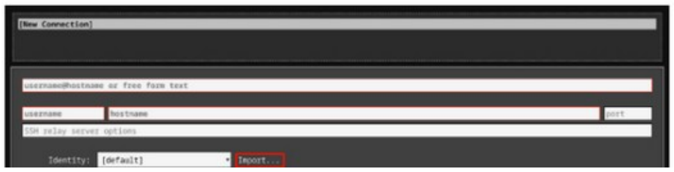
3. In the **username** section, enter the username given in the Connection Details Panel of the lab. And for the **hostname** section, enter the external IP of your VM instance that is mentioned in the Connection Details Panel of the lab.

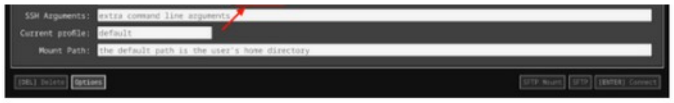


4. In the **Identity** section, import the downloaded PEM key by clicking on the **Import...** button beside the field. Choose your PEM key and click on the **OPEN** button.

Note: If the key is still not available after importing it, refresh the application, and select it from the **Identity** drop-down menu.

5. Once your key is uploaded, click on the **[ENTER] Connect** button below.





```
 SSH Arguments: extra command line arguments
Current profile: default
     Mount Path: the default path is the user's home directory
```

6. For any prompts, type **yes** to continue.

7. You have now successfully connected to your Linux VM.

You're now ready to continue with the lab!

Puppet rules

The goal of this exercise is for you to see what Puppet looks like in action. During this lab, you'll be connecting to two different VMs. The VM named **puppet** is the Puppet Master that has the Puppet rules that you'll need to edit. The VM named **linux-instance** is a client VM that you'll use to test that your catalog was applied successfully.

The manifests used for the production environment are located in the directory `/etc/puppet/code/environments/production/manifests`, which contains a site.pp file with the node definitions that will be used for this deployment. On top of that, the modules directory contains a bunch of modules that are already in use. You'll be extending the code of this deployment to add more functionality to it.

Install packages

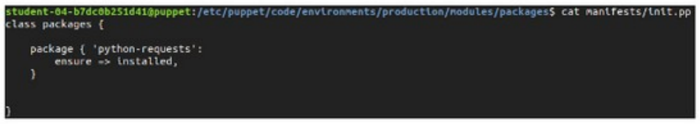
There's a module named **packages** on the **Puppet VM instance** that takes care of installing the packages that are needed on the machines in the fleet. Use the command to visit the module:

```
cd /etc/puppet/code/environments/production/modules/packages
```

This module already has a resource entry specifying that **python-requests** is installed on all machines. You can see the `init.pp` file using the **cat** command on the **Puppet VM instance.**

```
cat manifests/init.pp
```

Output:



```
student-04-b7dc0b251d41@puppet:/etc/puppet/code/environments/production/modules/packages$ cat manifests/init.pp
class packages {

    package { 'python-requests':
        ensure => installed,
    }


}
```

Now, add an additional resource in the same `init.pp` file within the path `/etc/puppet/code/environments/production/modules/packages`, ensuring the **golang** package gets installed on all machines that belong to the `Debian` family of operating systems (which includes Debian, Ubuntu, LinuxMint, and a bunch of others).

This resource will be very similar to the previous **python-requests** one. Add edit permission to the file before moving forward using:

```
sudo chmod 646 manifests/init.pp
```

To install the package on Debian systems only, you'll need to use the **os family** fact, like this:

```
if $facts[os][family] == "Debian" {
# Resource entry to install golang package
}
```

Now, open the file using nano editor and add the resource entry specifying golang package to be installed on all machines of Debian family after the previous resource entry.

The snippet would now look like this:

```
if $facts[os][family] == "Debian" {
     package { 'golang':
       ensure => installed,
     }
  }
```

The complete **init.pp** file would now look similar to the below file:

```
class packages {
   package { 'python-requests':
       ensure => installed,
   }
   if $facts[os][family] == "Debian" {
     package { 'golang':
       ensure => installed,
     }
  }
}
```

After this, we will also need to ensure that the **nodejs** package is installed on machines that belong to the `RedHat` family. Refer to the below snippet for this.

```
if $facts[os][family] == "RedHat" {
  #Resource entry
}
```

Complete the above snippet just like the previous one.

The complete **init.pp** file should now look like this:

```
class packages {
   package { 'python-requests':
       ensure => installed,
   }
   if $facts[os][family] == "Debian" {
     package { 'golang':
       ensure => installed,
     }
  }
   if $facts[os][family] == "RedHat" {
     package { 'nodejs':
       ensure => installed,
     }
  }
}
```

Once you've edited the file and added the necessary resources, you'll want to check that the rules work successfully. We can do this by connecting to another machine in the network and verifying that the right packages are installed.

We will be connecting to **linux-instance** using its external IP address. To fetch the external IP address of **linux-instance**, use the following command:

```
gcloud compute instances describe linux-instance --zone=us-
east1-c --
format='get(networkInterfaces[0].accessConfigs[0].natIP)'
```

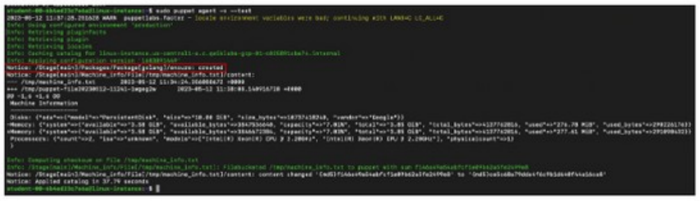
This command outputs the external IP address of **linux-instance**. Copy the **linux-instance** external IP address, open another terminal and connect to it. Follow the instructions given in the section `Accessing the virtual machine` by clicking on `Accessing the virtual machine` from the navigation pane at the right side.

Now manually run the Puppet client on your **linux-instance** VM instance terminal:

```
sudo puppet agent -v --test
```

This command should run successfully and the catalog should be applied.

Output:

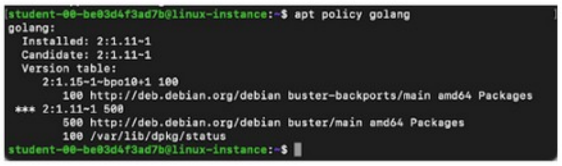


```
student-00-eb4ed33c7eda@linux-instance:~$ sudo puppet agent -v --test
2023-05-12 11:37:28.281628 WARN  puppetlabs.facter - locale environment variables were bad; continuing with LANG=C LC_ALL=C
Info: Using configured environment 'production'
Info: Retrieving pluginfacts
Info: Retrieving plugin
Info: Retrieving locales
Info: Caching catalog for linux-instance.us-central1-a.c.qwiklabs-gcp-01-[illegible].internal
Info: Applying configuration version '[illegible]'
Notice: /Stage[main]/Packages/Package[golang]/ensure: created
Notice: /Stage[main]/Machine_info/File[/tmp/machine_info.txt]/content:
--- /tmp/machine_info.txt       2023-05-12 11:34:24.[illegible] +0000
+++ /tmp/puppet-file20230512-11241-[illegible]      2023-05-12 11:38:[illegible] +0000
@@ -1,6 +1,6 @@
 Machine Information
 -------------------
 Disks: {"sda"=>{"model"=>"PersistentDisk", "size"=>"10.00 GiB", "size_bytes"=>10737418240, "vendor"=>"Google"}}
-Memory: {"system"=>{"available"=>"3.58 GiB", "available_bytes"=>[illegible], "capacity"=>"7.01%", "total"=>"3.85 GiB", "total_bytes"=>4137762816, "used"=>"276.78 MiB", "used_bytes"=>[illegible]}}
+Memory: {"system"=>{"available"=>"3.58 GiB", "available_bytes"=>[illegible], "capacity"=>"7.03%", "total"=>"3.85 GiB", "total_bytes"=>4137762816, "used"=>"277.41 MiB", "used_bytes"=>[illegible]}}
 Processors: {"count"=>2, "isa"=>"unknown", "models"=>["Intel(R) Xeon(R) CPU @ 2.20GHz", "Intel(R) Xeon(R) CPU @ 2.20GHz"], "physicalcount"=>1}
 }

Info: Computing checksum on file /tmp/machine_info.txt
Info: /Stage[main]/Machine_info/File[/tmp/machine_info.txt]: Filebucketed /tmp/machine_info.txt to puppet with sum [illegible]
Notice: /Stage[main]/Machine_info/File[/tmp/machine_info.txt]/content: content changed '{md5}[illegible]' to '{md5}[illegible]'
Notice: Applied catalog in 37.79 seconds
student-00-eb4ed33c7eda@linux-instance:~$
```

Now verify whether the **golang** package was installed on this instance. This being an machine of the Debian family should have golang installed. Use the following command to verify this:

```
apt policy golang
```

Output:



```
student-00-be03d4f3ad7b@linux-instance:~$ apt policy golang
golang:
  Installed: 2:1.11~1
  Candidate: 2:1.11~1
  Version table:
     2:1.15~1~bpo10+1 100
        100 http://deb.debian.org/debian buster-backports/main amd64 Packages
 *** 2:1.11~1 500
        500 http://deb.debian.org/debian buster/main amd64 Packages
        100 /var/lib/dpkg/status
student-00-be03d4f3ad7b@linux-instance:~$
```

With this, you've seen how you can use Puppet's facts and package resources to install specific packages on machines within your fleet.

Click *Check my progress* to verify the objective.

✓

Install packages

Check my progress

Successfully installed packages.

Fetch machine information

It's now time to navigate to the **machine_info** module in our Puppet environment. In the **Puppet VM terminal,** navigate to the module using the following command:

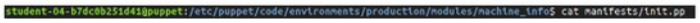
```
cd /etc/puppet/code/environments/production/modules/machine_info
```

The **machine_info** module gathers some information from the machine using **Puppet** facts and then stores it in a file. Currently, the module is always storing this information in `/tmp/machine_info.`

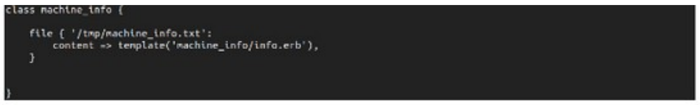
Let's check this out:

```
cat manifests/init.pp
```

Output:



```
student-04-b7dc0b251d41@puppet:/etc/puppet/code/environments/production/modules/machine_info$ cat manifests/init.pp
```



```
class machine_info {

    file { '/tmp/machine_info.txt':
        content => template('machine_info/info.erb'),
    }


}
```

You can view the path in the above file. This path doesn't work for Windows machines. So, you need to adapt this rule for Windows.

Add edit permission to the file using the following command before we adapt the rule.

```
sudo chmod 646 manifests/init.pp
```

Now we will be using `$facts[kernel]` fact to check if the kernel is "windows". If so, set a **$info_path** variable to "C:\Windows\Temp\Machine_Info.txt", otherwise set it to "/tmp/machine_info.txt". To do this, open the file using nano editor and add the below rule after the default path within the class `machine_info`.

```
  if $facts[kernel] == "windows" {
       $info_path = "C:\Windows\Temp\Machine_Info.txt"
  } else {
       $info_path = "/tmp/machine_info.txt"
  }
```

The file should now look similar to:

```
class machine_info {
   file { '/tmp/machine_info.txt':
       content => template('machine_info/info.erb'),
   }
   if $facts[kernel] == "windows" {
       $info_path = "C:\Windows\Temp\Machine_Info.txt"
   } else {
       $info_path = "/tmp/machine_info.txt"
   }
}
```

By default the file resources are stored in the path defined in the name of the resource (the string in the first line) within the class. We can also set different paths, by setting the path attribute.

We will now be renaming the resource to "machine_info" and then use the variable in the path attribute. The variable we are using to store the path in the above rule is **$info_path**.

Remove the following part from the file **manifests/init.pp**.

```
   file { '/tmp/machine_info.txt':
       content => template('machine_info/info.erb'),
   }
```

And add the following resource after the rule within the class definition:

```
   file { 'machine_info':
        path => $info_path,
        content => template('machine_info/info.erb'),
    }
```

The complete manifests/init.pp file should now look like this:

```
class machine_info {
  if $facts[kernel] == "windows" {
       $info_path = "C:\Windows\Temp\Machine_Info.txt"
   } else {
       $info_path = "/tmp/machine_info.txt"
   }
 file { 'machine_info':
       path => $info_path,
       content => template('machine_info/info.erb'),
   }
}
```

Puppet Templates

Templates are documents that combine code, data, and literal text to produce a final rendered output. The goal of a template is to manage a complicated piece of text with simple inputs.

In Puppet, you'll usually use templates to manage the content of configuration files (via the content attribute of the file resource type).

Templates are written in a templating language, which is specialized for generating text from data. Puppet supports two templating languages:

- **Embedded Puppet (EPP)** uses Puppet expressions in special tags. It's easy for any Puppet user to read, but only works with newer Puppet versions. (≥ 4.0, or late 3.x versions with future parser enabled.)
- **Embedded Ruby (ERB)** uses Ruby code in tags. You need to know a small bit of Ruby to read it, but it works with all Puppet versions.

Now, take a look at the template file using the following command.

```
cat templates/info.erb
```

Puppet templates generally use data taken from Puppet variables. Templates are files that are pre-processed, some values gets replaced with variables. In this case, the file currently includes the values of three facts. We will be adding a new fact in this file now.

Add edit permissions to the file using `templates/info.erb` using the following command:

```
sudo chmod 646 templates/info.erb
```

Now open the file using nano editor and add the following fact just after the last fact within the file:

```
Network Interfaces: <%= @interfaces %>
```

The template should now look like this:

```
Machine Information
-------------------
Disks: <%= @disks %>
Memory: <%= @memory %>
Processors: <%= @processors %>
Network Interfaces: <%= @interfaces %>
}
```

To check if this worked correctly, return to **linux-instance** VM terminal and manually run the client on that machine using the following command:

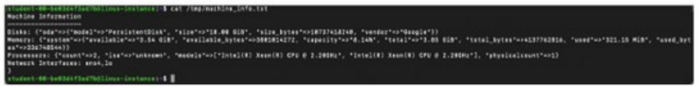
```
sudo puppet agent -v --test
```

This command should run successfully and the catalog should be applied.

Now verify that the **machine_info** file has the required information using:

```
cat /tmp/machine_info.txt
```

Output:



```
student-00-be03d4f3ad7b@linux-instance:~$ cat /tmp/machine_info.txt
Machine Information
-------------------
Disks: {"sda"=>{"model"=>"PersistentDisk", "size"=>"10.00 GiB", "size_bytes"=>10737418240, "vendor"=>"Google"}}
Memory: {"system"=>{"available"=>"3.54 GiB", "available_bytes"=>3801014272, "capacity"=>"8.14%", "total"=>"3.85 GiB", "total_bytes"=>4137762816, "used"=>"321.15 MiB", "used_byt
es"=>336748544}}
Processors: {"count"=>2, "isa"=>"unknown", "models"=>["Intel(R) Xeon(R) CPU @ 2.20GHz", "Intel(R) Xeon(R) CPU @ 2.20GHz"], "physicalcount"=>1}
Network Interfaces: ens4,lo
}
student-00-be03d4f3ad7b@linux-instance:~$
```

And with that, you've seen how you can fetch machine information and store it according to the operating system.

Click *Check my progress* to verify the objective.

Fetch machine information

Check my progress

Success: Retrieved network interfaces information from machine.

Reboot machine

For the last exercise, we will be creating a new module named **reboot**, that checks if a node has been online for more than **30 days**. If so, then reboot the computer.

To do that, you'll start by creating the module directory.

Switch back to **puppet** VM terminal and run the following command:

```
sudo mkdir -p
/etc/puppet/code/environments/production/modules/reboot/manifests
```

Go to the `manifests/` directory.

```
cd
/etc/puppet/code/environments/production/modules/reboot/manifests
```

Create an **init.pp** file for the reboot module in the `manifests/` directory.

```
sudo touch init.pp
```

Open **init.pp** with nano editor using sudo.

```
sudo nano init.pp
```

In this file, you'll start by creating a class called `reboot`.

The way to reboot a computer depends on the OS that it's running. So, you'll set a variable that has one of the following reboot commands, based on the kernel fact:

- **shutdown /r** on windows
- **shutdown -r now** on Darwin (macOS)
- **reboot** on Linux.

Hence, add the following snippet in the file **init.pp**:

```
class reboot {
  if $facts[kernel] == "windows" {
    $cmd = "shutdown /r"
  } elsif $facts[kernel] == "Darwin" {
    $cmd = "shutdown -r now"
  } else {
    $cmd = "reboot"
  }
}
```

With this variable defined, we will now define an exec resource that calls the command, but only when the **uptime_days** fact is larger than 30 days.

Add the following snippet after the previous one within the class definition in the file

reboot/manifests/init.pp:

```
if $facts[uptime_days] > 30 {
        exec { 'reboot':
           command => $cmd,
        }
    }
```

The complete **reboot/manifests/init.pp** should now look like this:

```
class reboot {
  if $facts[kernel] == "windows" {
    $cmd = "shutdown /r"
  } elsif $facts[kernel] == "Darwin" {
    $cmd = "shutdown -r now"
  } else {
    $cmd = "reboot"
  }
  if $facts[uptime_days] > 30 {
    exec { 'reboot':
      command => $cmd,
     }
   }
}
```

Finally, to get this module executed, make sure to include it in the `site.pp` file.

So, edit `/etc/puppet/code/environments/production/manifests/site.pp` using the following command:

```
sudo nano
/etc/puppet/code/environments/production/manifests/site.pp
```

Add an extra line that includes the reboot module.

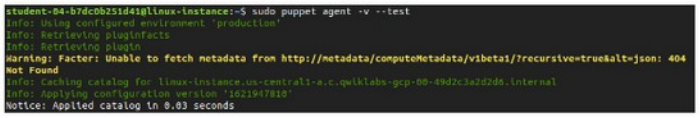
The file `/etc/puppet/code/environments/production/manifests/site.pp` should now look like this:

```
node default {
   class { 'packages': }
   class { 'machine_info': }
   class { 'reboot': }
}
```

Run the client on **linux-instance** VM terminal:

```
sudo puppet agent -v --test
```

Output:



```
student-04-b7dc0b251d41@linux-instance:~$ sudo puppet agent -v --test
Info: Using configured environment 'production'
Info: Retrieving pluginfacts
Info: Retrieving plugin
Warning: Facter: Unable to fetch metadata from http://metadata/computeMetadata/v1beta1/?recursive=true&alt=json: 404
Not Found
Info: Caching catalog for linux-instance.us-central1-a.c.qwiklabs-gcp-00-49d2c3a2d2d6.internal
Info: Applying configuration version '1621947810'
Notice: Applied catalog in 0.03 seconds
```

Click *Check my progress* to verify the objective.

✓ Reboot machine

Check my progress

Reboot module added successfully.

And with that, you've added a whole new module to your deployment!

Congratulations!

Woohoo! You've successfully improved the Puppet deployment, by modifying existing modules and adding new ones!

End your lab

When you have completed your lab, click **End Lab**. Qwiklabs removes the resources you've used and cleans the account for you.

You will be given an opportunity to rate the lab experience. Select the applicable number of stars, type a comment, and then click **Submit**.

The number of stars indicates the following:

- 1 star = Very dissatisfied
- 2 stars = Dissatisfied
- 3 stars = Neutral
- 4 stars = Satisfied
- 5 stars = Very satisfied

You can close the dialog box if you don't want to provide feedback.

For feedback, suggestions, or corrections, please use the **Support** tab.