

Debugging Puppet Installation

1 hour 30 minutesFree

Rate Lab

Introduction

You're a member of the IT team at your company. One of your coworkers has recently set up a Puppet installation, but it doesn't seem to be doing its job. So, you're asked to debug the profile class, which is supposed to append a path to the environment variable `$PATH`. But it's somehow misbehaving and causing the `$PATH` variable to be broken. You'll need to locate the issue and fix it with the knowledge that you learned in this module.

What you'll do

- Check out what Puppet rules look like
- Run the Puppet agent locally
- Understand how file permissions are represented by numbers
- Learn how scripts under `/etc/profile.d/` perform startup tasks, including setting up your own environment variables
- Append a string to an environment variable

You'll have 90 minutes to complete this lab.

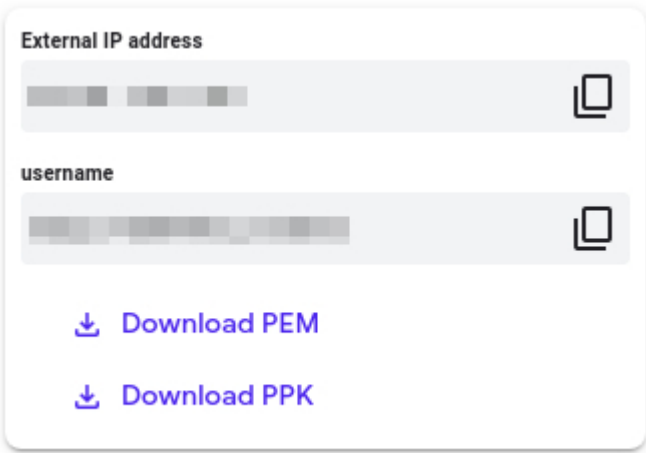
Start the lab

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).

A green rectangular button with the text "Start Lab" in white.

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:

A white rounded rectangle containing SSH connection details. It has two input fields: "External IP address" and "username", each with a copy icon to its right. Below these are two download links: "Download PEM" and "Download PPK", each preceded by a download icon.

External IP address

username

Download PEM

Download PPK

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**.

 [Download PEM](#)

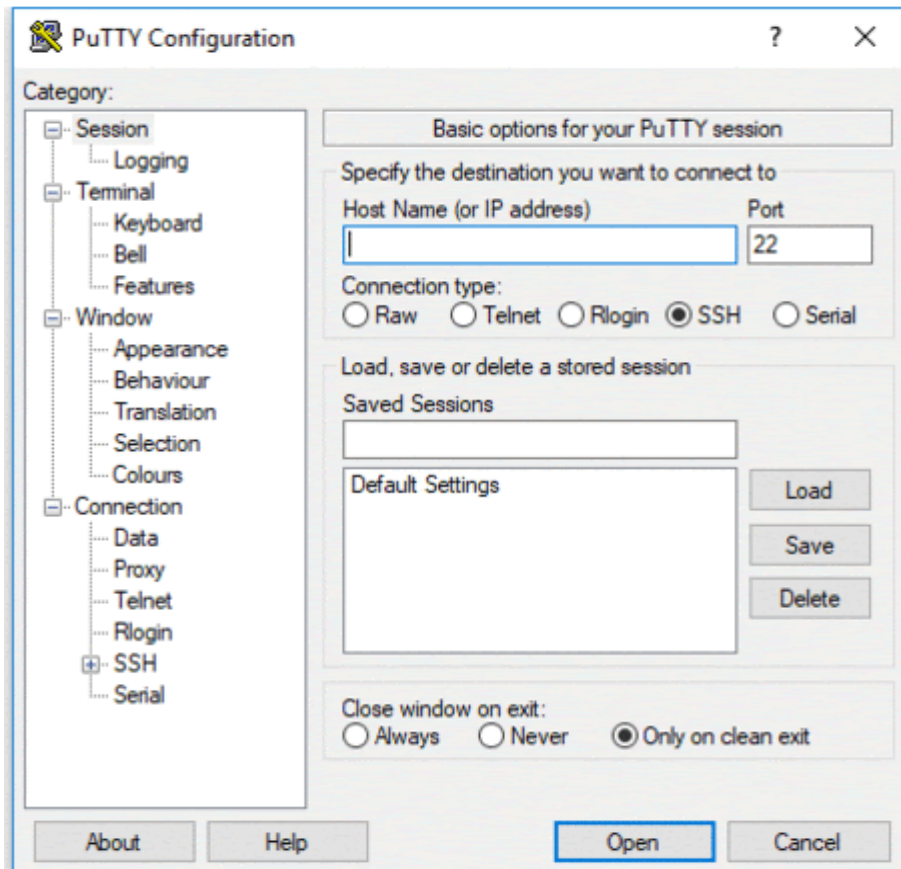
 [Download PPK](#)



Connect to your VM using SSH and PuTTY

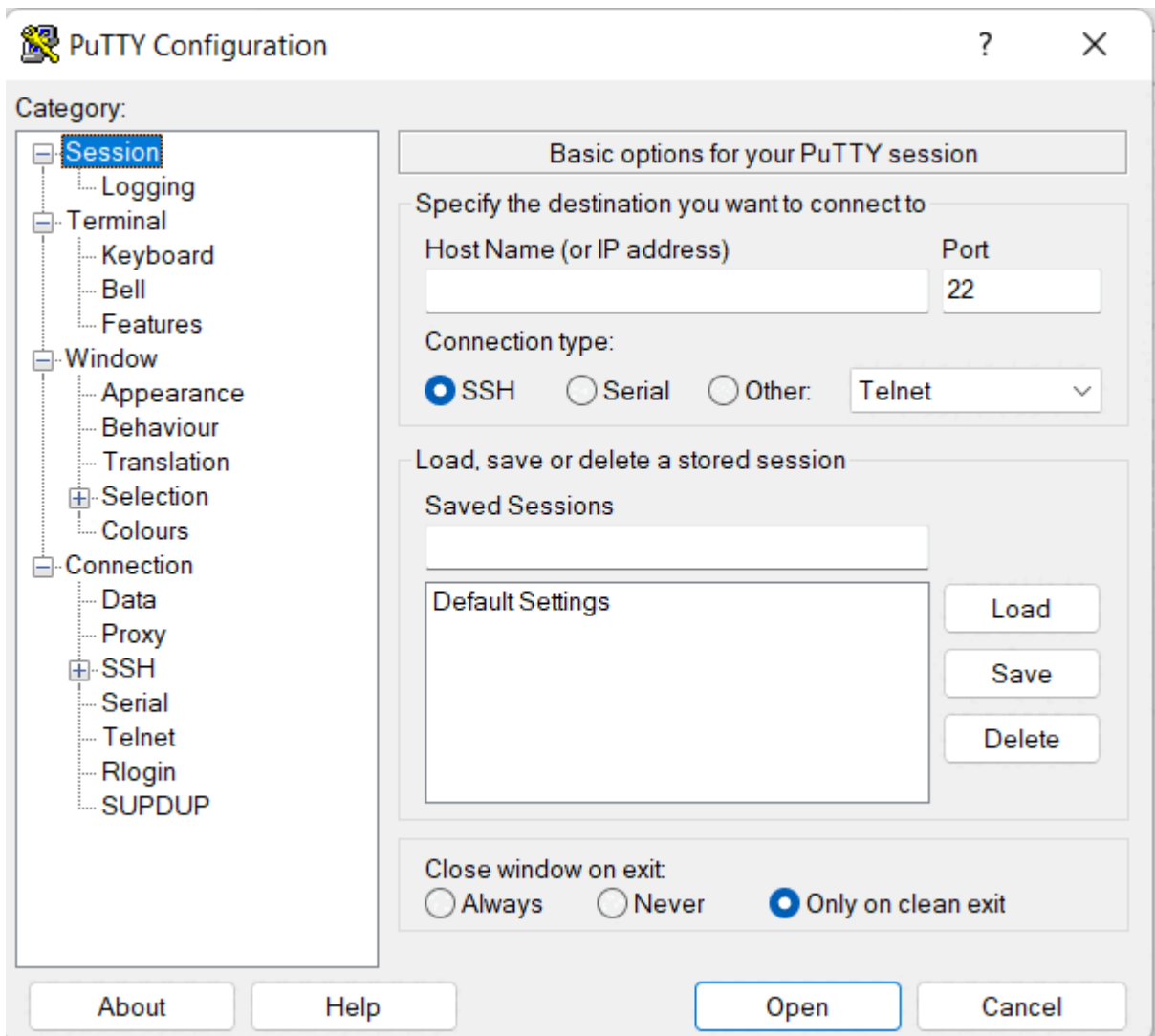
1. You can download Putty from [here](#)
2. In the **Host Name (or IP address)** box, enter `username@external_ip_address`.

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



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.



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
```

```
Copied!
```

```
content_copy
```

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

```
Copied!
```

content_copy

```
~$ ssh -i ~/Downloads/qwikLABS-L923-42090.pem gcpstagingedit1370_stu
The authenticity of host '35.239.106.192 (35.239.106.192)' can't be established.
ECDSA key fingerprint is SHA256:vrz8b4aYUtruFh0A6wZn6Ozy1oqqPEfh931olvxITm8.
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.
gcpstagingedit1370_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]**.

[New Connection]

username@hostname or free form

username

hostname

SSH relay server options

Identity: [default]

SSH Arguments: extra command line

Current profile: default

Mount Path: the default path

[DEL] Delete

Options

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.

[New Connection]

username@hostname or free form text

username

hostname

SSH relay server options

Identity: [default]

SSH Arguments: extra command line arg

Current profile: default

Mount Path: the default path is th

[DEL] Delete

Options

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.

[New Connection]

username@hostname or free form text

username

hostname

SSH relay server options

Identity: [default]

SSH Arguments: extra command line arguments

Current profile: default

Mount Path: the default path is the

[DEL] Delete

Options

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 Puppet rules are present in the init file. The purpose of this script is to append /java/bin to the environment variable \$PATH, so that scripts in that directory can be executed without writing the full path.

The purpose of this rule is to append `/java/bin` to the environment variable `$PATH` so that scripts in that directory can be executed without writing the full path.

Issue detection

Now, take a look at the environment variable `$PATH` by running the following command:

```
echo $PATH  
Copied!  
content_copy
```

Output:

```
student-03-6390fb6e1031@puppet:~$ echo $PATH  
/java/bin:/snap/bin  
student-03-6390fb6e1031@puppet:~$
```

Oops, something is wrong; the main directories used for executing binaries in Linux (`/bin` and `/usr/bin`) are missing.

You can check this by executing the following command:

ls /

Copied!

content_copy

Output:

```
student-03-6390fb6e1031@puppet:~$ ls /  
Command 'ls' is available in '/bin/ls'  
The command could not be located because '/bin/  
ls: command not found  
student-03-6390fb6e1031@puppet:~$
```

Commands like `ls`, `cd`, `mkdir`, `rm` and others are just small programs that usually live inside a directory on our systems called `/usr/bin`.

To get back to a working environment, run the following command:

```
export PATH=/bin:/usr/bin
```

Copied!

content_copy

By running this command, you manually add the directories that contain executing binaries (`/bin` and `/usr/bin`) to the environment variable `$PATH`. That way, the system will be able to find the commands when you try to run them. Now, run the list directory command, again:

ls /

Copied!

content_copy

Output:

```
student-03-6390fb6e1031@puppet:~$ ls /  
bin      dev      home      initrd.img.old  lib64  
boot     etc      initrd.img  lib             lost+found  
student-03-6390fb6e1031@puppet:~$
```

Alright, now that we have a working PATH, let's look at the rule responsible for this breakage. It's located in the **profile** module of Puppet's **production** environment. To look at it, go to the **manifests** directory that contains the Puppet rules by using the following command:

```
cd /etc/puppet/code/environments/production/modules/profile/manifests
Copied!
content_copy
```

Use **cat** to check out the contents of the **init.pp** file:

```
cat init.pp
Copied!
content_copy
```

```
student-03-6390fb6e1031@puppet: /etc/puppet/code/environments/production/modules/profile/manifests
class profile {
    file { [ '/etc/profile.d/append-path.sh' :
        owner      => 'root',
        group      => 'root',
        mode       => '0646',
        content    => "PATH=/java/bin\n",
    ]
}
student-03-6390fb6e1031@puppet: /etc/puppet/code/environments/production/modules/profile/manifests
```

This rule is creating a script under **/etc/profile.d/**. Scripts in this path will perform startup tasks, including setting up a user's own environment variables. The files under **/etc/profile.d/** should only be editable by root.

We see that the file resource is part of the **profile** class. The general rules for creating a Puppet class definition used here are:

- The class definition starts with the "class" keyword, followed by the name of the class. In this case, "profile".
- The contents of the class are defined between curly braces and generally contain at least one resource declaration. In this case, a "file" resource.

There could be other resources in this class, but for now it has only one file resource. Let's look at what it's doing. The file defined by this resource is **/etc/profile.d/append-path.sh**, and the Puppet rule is using some of the available "file" attributes:

- It sets both the owner and group of the file to "root".

- It then sets the "mode" of the file to "0646". This number represents the permissions the file will have.
- You might remember that every file and directory on a Linux system is assigned permissions for three groups of people: the owner, the group and the others.. And for each group, the permissions refer to the possibility of reading, writing and executing the file.
- It's common to use numbers to represent the permissions: 4 for read, 2 for write and 1 for execute. The sum of the permissions given to each of the groups is then a part of the final number. For example, a permission of 6 means read and write, a permission of 5 means read and execute, and a permission of 7 means read, write and execute.
- In this example, we are using 4 numbers. The first one represents any special permissions that the file has (no special permissions). The second one is the permissions for the owner, (read and write), and then come the permissions for the group (read), and finally the permissions for the others (read and write)
- Finally, it sets the actual contents of the file. Here, the content is being set to `PATH=/java/bin\n"`

Fixing the problem

Before fixing the issue, let's learn a bit more about the PATH variable. This is an environment variable that contains an ordered list of paths that Linux will search for executables when running a command. Using these paths implies that we don't have to specify the absolute path for each command we want to run.

The PATH variable typically contains a few different paths, which are separated by colons. The goal of the Puppet rule that we saw was to add one specific directory to the list of paths, but

unfortunately it's currently completely overwriting the contents. We need to change the Puppet rule to **append** the directory without overwriting the other paths in the variable.

To do this, we'll first include the current contents of the PATH variable followed by a colon, and then append the new contents we want to add.

Open the file init.pp using the nano editor:

```
sudo nano init.pp
Copied!
content_copy
```

Now, the content attribute should be changed. The line `content => "PATH=/java/bin\n"` should be changed to `content => "PATH=\$PATH:/java/bin\n"` since we want to append `/java/bin` to the environment variable `$PATH`, and not completely replace the current content in `$PATH`.

The extra backslash before the `$` is necessary because Puppet also uses `$` to indicate variables. But in this case, we want the dollar sign in the contents of the file.

On top of this, files in the `/etc/profile.d` directory should only be editable by the root user. In order to do this we'll need to change the mode of the file to avoid giving others permission to write the file. In other words, the mode should be 0644 not 0646.

Once you're done fixing the mode and content attributes, remember to save the file and close it.

After that, you can trigger a manual run of the Puppet agent by running the following command:

```
sudo puppet agent -v --test
Copied!
content_copy
```

Go ahead and initiate a **second SSH connection** following the same instructions from the `Accessing the linux virtual machine` section. Then, verify that the profile is fixed by checking the contents of the PATH variable.

To check the PATH variable has the right contents, use the command below in the second SSH connection:

```
echo $PATH
Copied!
content_copy
```

Output:

```
student-03-4b621fa0d965@puppet:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local
student-03-4b621fa0d965@puppet:~$
```

You've successfully fixed the misbehaving `$PATH` variable.

Click [Check my progress](#) to verify the objective.

[Fix issue in init.pp](#)

[Check my progress](#)

Congratulations!

Woohoo! You have used Puppet to automate the configurations on the machines and fixed the broken `$PATH` variable. A great first step towards getting comfortable using Puppet.

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.