



puppet

Getting Started with Puppet

Exercise & Lab Guide

Puppet Education

<http://learn.puppet.com/>

Lab 1.1: Puppet product overview

Objective:

After this lab, you will know how Puppet's configuration management products help you to comprehend and manage your IT infrastructure. You'll see a demonstration of Puppet Discovery and Puppet Enterprise and get a high-level overview of their key features.

This lab is an instructor-guided demo of the products.

Discussion Questions

- What are some common problems that are addressed by Puppet Discovery?
- How could Puppet Discovery help to inform a configuration management strategy?
- Why is RBAC important in Puppet Enterprise?

Lab 2.1: Install

Objective:

In this lab you will complete the installation of the open source Bolt utility. Depending on the platform you wish to use (Windows or Linux), the process will look slightly different. Instructions are provided for both.

After completing this exercise you will have done the following:

- Installed the Bolt executable
- Executed a simple command to validate the utility is working

Required resources

These are the resources required to complete this lab:

- A host machine to install the software.
- Install this utility on one of the classroom nodes you have been provided (Windows or Linux)

Task 1: Install Bolt

Lab procedure

Decide where you will install Bolt. You have the choice of installing Bolt onto a classroom provided Windows or Linux machine. If time allows, you can install Bolt onto multiple platforms. Once you've decided where you'll be installing Bolt, connect to the target machine (SSH or RDP depending) and follow the steps for your platform.

- [Windows](#)
- [Linux](#)

continued...

Windows

Connect to the agent



Remember, you can get the details of which student machines you should be using in the welcome page, located at https://class_XXXX_.classroom.puppet.com

1. Start the RDP client to connect to your agent
 - Windows:
 - Remote Desktop Connection
 - Mac OSX
 - Microsoft Remote Desktop 8 or 10
2. Connect with the following information
 - hostname: **XXXXwin_Y_.classroom.puppet.com**
 - Remember, the hostname of your Windows machine can be retrieved from the class welcome page
 - username: Administrator
 - Password: `password from the welcome page`

Step 1

Start a web browser on the server, and enter the URL to download the Puppet **MSI** installation file from: `https://downloads.puppet.com/windows/puppet5/puppet-bolt-x64-latest.msi`

Choose the Downloads folder as the save location.

Step 2

When prompted by the browser, click **Run**. Follow the installation Wizard instructions. No customization is required, just **Accept the license**, click **Next** through the option windows, and click **Install**, and finally **Finish**.

Step 3

Next, you'll validate the installation is working. Open a new PowerShell prompt and run: `bolt --version`

Expected Output

```
PS C:\Users\Administrator> bolt --version
1.13.0
```



Your version will likely vary, and this is not a problem.

continued...

Linux

i The steps below detail connecting to your Linux machine from your student Windows machine (used above). If you would rather connect directly from your own laptop you can, however you will have to translate these instructions to the utilities you are using. The private key for the Linux machine can be downloaded from the Welcome page.

i There are many distributions of Linux. This classroom environment uses CentOS, however Bolt supports many different versions. See the official documentation for more details. The guide below is for CentOS 7.

For this lab you'll be leveraging the built-in package manager to install Bolt.

Connecting to the Linux Host using PuTTY or Openssh method

PuTTY Method

1. On your Windows student machine, open a PowerShell window
2. Launch PuTTY by typing `putty` and pressing enter:
 - In the Host Name field, copy and paste your Linux machine's hostname from the Welcome page
 - On the left hand menu, expand `ssh` and click on **Auth**
 - Under `Private key file for authentication`, click the **Browse** button
 - Select the pre-loaded private key from `C:\keys`.
 - Select **private_key.ppk** and click **Open**
 - Next, under the `Connection` category, click on **Data**
 - In the `Auto-login username` field, type **centos**
 - Return to the session view by clicking **Session** on the left-hand menu in PuTTY
 - In the `Saved Sessions` box, type in **linux_machine** and click **Save** on the right hand side
 - Click **Open**
 - If prompted about the host key not being cached, select **Yes**

i For any future connections to this host, you can now simply double-click **linux_machine** in the `Saved Sessions` window

OpenSSH method

1. On your Windows student machine, open a PowerShell window
2. Install win32-openssh using chocolatey, `choco install openssh` and press enter:
 - When the install is done, restart PowerShell
 - You can then use the credentials already on the host
3. Re-open PowerShell and type in `ssh centos@<your-linux-machine>.classroom.puppet.com`

continued...

Step 1

Execute the following command to install the bolt package:

```
sudo yum install -y puppet-bolt
```

Step 2

Now you will validate if your installation is working properly:

```
$ bolt --version  
1.13.0
```



Your version will likely vary, and this is not a problem.

Lab 2.2: Running commands

Objective:

In this lab you will practice executing simple system commands using the Bolt utility. Additionally, you will learn how to leverage Bolt to run entire scripts, written in any language, against target machines.

After completing this exercise you will have done the following:

- Successfully executed commands via the Bolt utility.
- Used Bolt to run existing scripts.

Required resources

These are the resources required to complete this lab:

- A host machine to run the Bolt utility from.
 - Note, this will be the same machine you installed Bolt on in a previous lab.

continued...

Task 1: Run local system commands

If you are not already, connect to your target machine you will run Bolt from (via SSH or RDP). If Windows, open a PowerShell terminal.

Step 1

Start by stopping the time service on your machine.

i For the remainder of this lab, you will see instructions for both Windows and Linux and should use the same one for your platform.

From your terminal window:

Steps for Windows:

```
PS C:\> bolt command run 'net stop w32time' --nodes winrm://localhost --user Administrator --no-ssl --password
```

i You will be prompted for the password.

i The `--no-ssl` option allows you to use WinRM over 5985, however Bolt supports both SSL and non-SSL WinRM connections.

The output will look similar to the following:

```
STDOUT:
  The Windows Time service is stopping.
  The Windows Time service was stopped successfully.

Successful on 1 node: winrm://localhost
Ran on 1 node in 2.61 seconds
```

continued...

Task 1: Run local system commands

Steps for Linux:

```
$ bolt command run 'sudo systemctl stop ntpd' --nodes ssh://localhost --user centos --no-host-key-check
```

i The required SSH key setup has been pre-built into the student Linux machine so you do not need to supply a password.

i The `--no-host-key-check` option is being used to avoid needing an entry into the SSH `known_hosts` file, however the default behavior of Bolt is to validate SSH key signatures for security purposes.

You will see output similar to the following:

```
Started on localhost...  
Finished on localhost:  
Successful on 1 node: ssh://localhost  
Ran on 1 node in 0.45 seconds
```

continued...

Task 2: Run remote system commands

Now let's start the time service on a remote machine, which is just a bit more interesting.

i For this exercise you'll be using the other student machine. For example, if you're logged into your Windows machine, you'll start the service on your Linux machine.

Step 1

From your terminal window:

Steps for Windows:

```
PS C:\> bolt command run 'sudo systemctl start ntpd' --nodes ssh://<your-linux-machine>.classroom.puppet.com --private-key C:\keys\private_key.pem --user centos --no-host-key-check
```

i You've done this on just one remote host, however you could have supplied a list of hosts to run simultaneously - or even an entire inventory file of hosts.

The output will look similar to the following:

```
Started on <your-linux-machine>.classroom.puppet.com...
Finished on <your-linux-machine>.classroom.puppet.com:
Successful on 1 node: ssh://<your-linux-machine>.classroom.puppet.com
Ran on 1 node in 0.45 seconds
```

Steps for Linux:

```
$ bolt command run 'net start w32time' --nodes winrm://<your-windows-machine>.classroom.puppet.com --user Administrator --no-ssl --password
```

i Like before, you will be prompted for a password.

You should see the following output:

```
STDOUT:
The Windows Time service is starting.
The Windows Time service was started successfully.
```

continued...

Step 2

Now that you've started the service on your remote machine, ensure it by running the commands below.

Steps for Windows:

```
PS C:\> bolt command run 'sudo systemctl status ntpd' --nodes ssh://<your-linux-machine>.classroom.puppet.com --private-key C:\keys\private_key.pem --user centos --no-host-key-check
```


Your command should finish successfully and the output will look similar to the following:

```
STDOUT:
ntpd.service - Network Time Service
Loaded: loaded (/usr/lib/systemd/system/ntpd.service; enabled; vendor preset: disabled)
Active: active (running) since Wed 2018-09-05 11:52:28 UTC; 8s ago
Process: 13756 ExecStart=/usr/sbin/ntpd -u ntp:ntp $OPTIONS (code=exited, status=0/SUCCESS)
Main PID: 13757 (ntpd)
CGroup: /system.slice/ntpd.service
        /usr/sbin/ntpd -u ntp:ntp -g

[...truncated...]
Successful on 1 node: ssh://<your-linux-machine>.classroom.puppet.com
```

Steps for Linux:

```
$ bolt command run 'Get-Service w32time' --nodes winrm://<your-windows-machine>.classroom.puppet.com --user Administrator --no-ssl --password
```

 Like before, you will be prompted for a password.

The output will look similar to the following:

```
STDOUT:

Status   Name      DisplayName
-----
Running  w32time   Windows Time

Successful on 1 node: winrm://<your-windows-machine>.classroom.puppet.com
```

continued...

Task 3: Run scripts

Many times you already have existing scripts for doing the work you'd like to accomplish, however what you don't have is a good way to execute them on a remote machine - or many machines simultaneously. Bolt can do that.

In this exercise you will run a script against your remote host. Regardless of platform, the script does the following:

- Sets up a simple MOTD (message of the day).
- Deploys/Configures NTP.
- Prints a simple inventory report.



For simplicity, the scripts have been pre-loaded onto your student machines. You can check them out at `/tools` on Linux and `C:\tools` on Windows.

Step 1

In your terminal execute the following:

Steps for Windows:

```
PS C:\> bolt script run C:\tools\linux.sh --nodes ssh://<your-linux-machine>.classroom.puppet.com
--private-key C:\keys\private_key.pem --user centos --no-host-key-check --run-as root
```



Notice the use of `--run-as`? This allows us to handle permission escalations, as needed. In this case you are authenticating as `centos` and then changing user to `root` (via `sudo`) when the script executes. Your user has no password access to run `sudo`, however if you have separate `sudo` password you need to supply you can do that as well. See the output of `bolt -h` for details.

Your output will look similar to below:

```
STDOUT:
Setting MOTD...
Completed setting MOTD!
Configuring NTP....
Configured NTP!

Inventory Report:

Hostname      IPAddress    OSName      DiskUsed
<your-linux-machine> 172.31.32.23 CentOS-7.5.1804 13%

Successful on 1 node: ssh://<your-linux-machine>.classroom.puppet.com
Ran on 1 node in 2.00 seconds
```

continued...

Steps for Linux:

```
$ bolt script run /tools/windows.ps1 --nodes winrm://<your-windows-machine>.classroom.puppet.com
--user Administrator --no-ssl --password
```

The (approximate) output you should see:

```
STDOUT:
Updating the MOTD Info...
MOTD Set successfully!
Configure Time server...
The command completed successfully.
Sending resync command to local computer
The command completed successfully.
Time server set correctly!

Inventory report:

Hostname                IPAddress      OSName                DiskUsed
-----
<your-windows-machine> 172.31.44.123 Microsoft Windows Server 2016 Datacenter 49.41

Successful on 1 node: winrm://<your-windows-machine>.classroom.puppet.com
Ran on 1 node in 6.08 seconds
```

continued...

Discussion questions:

- How might you use Bolt for simple ad hoc administration tasks?
- What could you do if you want to run against a large group of hosts instead of a single target?
- How might being able to run existing scripts help you enable other teams?

NOT FOR DISTRIBUTION

Lab 5.1: Agent deployment

Objective:

After this lab, you will be familiar with the installed tools and be able to install the Puppet agent.

- Learn how to connect to your agent node using the OS-appropriate protocol
 - Windows: RDP
 - Linux: SSH
- Identify the pre-installed tools and understand their purpose
- Install the Puppet agent from the command line

You will be using one of the following environments:

- Windows Server 2016 Datacenter
 - Graphical development environment (Visual Studio Code)
 - Best practice workflow
 - In-class workflow support
- CentOS 7
 - Console development environment
 - Assumes comfort using git and vi on the command line

We recommend the Windows environment for most students. You will get practice with Puppet's recommended module development workflow and utilities. These are designed to provide a streamlined developer experience that allows you to quickly start managing infrastructure with Puppet.

Students who are already comfortable working on a Linux command line may use the Linux agent instead. If you choose to use this environment, you should already be comfortable navigating the Linux command line and using common command line tools such as vi. We will provide commands for technologies that are introduced in this course, such as Git and the Puppet agent. In the interest of time, course instructors will not provide general assistance with the Linux command line during class.

Future labs will use the tools and workflow that are introduced in this module.

continued...

Steps for Windows:

Validate and configure the agent environment

1. Launch Microsoft Visual Studio Code. This is the editor that you will use to write Puppet code.
 - Click the Windows button in the lower left
 - Click the **Visual Studio Code** folder
 - Click **Visual Studio Code**
2. Open a Windows PowerShell prompt. You will enter commands at the command line here.
 - Click the Windows button in the lower left
 - Click the **Windows PowerShell** folder
 - Click **Windows PowerShell**
3. Verify that Git is installed. You will use this for version control when you edit Puppet code.
 - Switch to the Windows PowerShell window
 - Enter `git --version` at the prompt
 - You should see output that reflects the current version, such as: `git version 2.18.0.windows.1` (note your version may be different)

continued...

Lab 5.1: Agent deployment

Install the Puppet agent using Bolt:

A basic installation script has been pre-staged in your home directory. You can review the script by opening it in **Visual Studio Code**:

- `C:\Users\Administrator\install_pe_agent.ps1`

Invoke the script in an Administrator PowerShell prompt using Bolt in order to install the agent on your local system.

```
PS C:\Users\Administrator> bolt script run install_pe_agent.ps1 --nodes winrm://localhost --user Administrator --no-ssl --password
```

Verify that the agent installed correctly:

Navigate to the start button, Puppet application folder, and click on `Start Command prompt with Puppet`. In the new window, verify Puppet is properly installed (see below).

```
C:\Windows\System32> puppet agent --version
```

continued...

Steps for Linux:

Validate and configure the agent environment

Verify that Git is installed. You will use this for version control when you edit Puppet code.



You should see output similar to the following (note: your version number may be different):

```
[centos@XXXXlinN ~]$ git --version  
git version 1.8.3.1
```

Install the Puppet agent using Bolt:

A basic installation script has been pre-staged in your home directory, here

```
~/install_pe_agent.sh
```

Invoke the script using Bolt in order to install the agent on your local system:

```
[centos@XXXXlinN ~]$ bolt script run install_pe_agent.sh --nodes localhost
```

Verify that the Puppet agent installed correctly:

```
[centos@XXXXlinN ~]$ puppet --version
```

continued...

Sign Agent Certificates

Now that your agents have been installed, we have to sign the certificates so that the master trusts them and will allow the agents to retrieve configurations.

To start:

- Connect to the Puppet Enterprise console (link is in the Welcome page)
- Log in using `username` : **studentN** and `password` : **puppetlabs**
- Navigate to **Unsigned certs** under **SETUP** on the left-hand menu
- In the `Node name` field, identify your nodes (again, using the hostnames provided in the Welcome page) and click **Accept** next to the hostname

i If you do not see your nodes in the listing, refresh the page. Also, keep in mind you will see all students machines here - please only accept your own and do not click `Accept All`

Back on your machines:

i On Windows, if you do not open a new shell you'll likely get an error about the command `puppet` not being found. This is because your system path has changed after you opened the shell, so open a new one to avoid this.

- On Windows, open a PowerShell window and execute `puppet agent -t`
- On Linux, open a bash session and execute `sudo puppet agent -t`

After a few seconds of runtime you will see a notice like this:

```
Notice: Applied catalog in X.XX seconds
```

Discussion Questions

- What are some of the benefits to new users of using a graphical development environment?
- You installed the agent non-interactively using a single command. How might this be useful in production environments?

Lab 6.1: Puppet resources

Objective:

In this lab, you will familiarize yourself with the `puppet resource` command and how it's used for both inspection and manipulation of Puppet resources.

After completing this exercise, you will have done the following:

- Inspected the state of local users and groups.
- Changed the configuration of users and groups.
- Inspected the state of the previously-configured time service.

Task 1: Use `puppet resource` to display local users

The `puppet resource` command enumerates and displays the current state of resources on a target node. It is invoked with a resource type name and optional resource title, and it emits valid Puppet code describing the state of the requested resources.

continued...

Step 1

Steps for Windows

In your powershell window, type `puppet resource user` and press `Enter`. Next, type `puppet resource user Administrator` and press `Enter`.

Observe the output in each case, it should look similar to (abridged here):

```
PS C:\Users\Administrator> puppet resource user
user { 'Administrator':
  ensure => 'present',
  comment => 'Built-in account for administering the computer/domain',
  groups => ['BUILTIN\Administrators'],
  uid => 'S-1-5-21-545430610-555444065-152929764-500',
}
user { 'DefaultAccount':
  ensure => 'present',
  comment => 'A user account managed by the system.',
  groups => ['BUILTIN\System Managed Group'],
  uid => 'S-1-5-21-545430610-555444065-152929764-503',
}
[...]
```

Next, ...

```
PS C:\Users\Administrator> puppet resource user Administrator
user { 'Administrator':
  ensure => 'present',
  comment => 'Built-in account for administering the computer/domain',
  groups => ['BUILTIN\Administrators'],
  uid => 'S-1-5-21-545430610-555444065-152929764-500',
}
```

continued...

Steps for Linux

At the command line, type `sudo puppet resource user` and press `Enter`. Next, type `sudo puppet resource user root` and press `Enter`.

Observe the output in each case, and it should look similar to (abridged here):

```
$ sudo puppet resource user
user { 'adm':
  ensure      => 'present',
  comment     => 'adm',
  gid         => 4,
  home        => '/var/adm',
  password    => '*',
  password_max_age => 99999,
  password_min_age  => 0,
  password_warn_days => 7,
  shell       => '/sbin/nologin',
  uid         => 3,
}
user { 'bin':
  ensure      => 'present',
[...]
```

Next, ...

```
$ sudo puppet resource user root
user { 'root':
  ensure      => 'present',
  comment     => 'root',
  gid         => 0,
  home        => '/root',
  password    => '!!',
  password_max_age => 99999,
  password_min_age  => 0,
  password_warn_days => 7,
  shell       => '/bin/bash',
  uid         => 0,
}
```

continued...

Step 2

The `puppet resource` command can display the current state of any known resource on the target node. Similar to displaying user resources in the previous step, the command can also display groups.

Steps for Windows:

At the command line, type `puppet resource group` and press `Enter`. Next, type `puppet resource group 'power users'` and press `Enter`.

Observe the output in each case, and it should look similar to (abridged here):

```
PS C:\Users\Administrator> puppet resource group
[...]
group { 'Performance Log Users':
  ensure => 'present',
  gid    => 'S-1-5-32-559',
}
group { 'Performance Monitor Users':
  ensure => 'present',
  gid    => 'S-1-5-32-558',
}
group { 'Power Users':
  ensure => 'present',
  gid    => 'S-1-5-32-547',
}
group { 'Print Operators':
  ensure => 'present',
  gid    => 'S-1-5-32-550',
}
[...]
```

Next, ...

```
PS C:\Users\Administrator> puppet resource group 'power users'
group { 'power users':
  ensure => 'present',
  gid    => 'S-1-5-32-547',
}
```

continued...

Steps for Linux:

At the command line, type `sudo puppet resource group` and press `Enter`. Next, type `sudo puppet resource group wheel` and press `Enter`.

Observe the output in each case, and it should look similar to (abridged here):

```
$ sudo puppet resource group
group { 'adm':
  ensure => 'present',
  gid    => 4,
}
group { 'audio':
  ensure => 'present',
  gid    => 63,
}
group { 'bin':
  ensure => 'present',
  ...
}
```

Next, ...

```
$ sudo puppet resource group wheel
group { 'wheel':
  ensure => 'present',
  gid    => 10,
}
```

continued...

Step 3

As mentioned previously, the `puppet resource` command not only inspects the current state of resources on a target node, it is also useful for modifying their state on the fly without writing any Puppet code.

Steps for Windows:

On the command line, type `puppet resource user mimi ensure=present`. Notice the extra `ensure=present` command line argument. Any other combination of resource attributes and values can be added on the command line as well. Once the command executes, it will emit Puppet code indicating the current state of the resource that was updated. In this case, a brand-new user was created on the target node.

Observe the output after creating the new user with the `puppet resource` command:

```
PS C:\Users\Administrator> puppet resource user mimi ensure=present
Notice: /User[mimi]/ensure: created
user { 'mimi':
  ensure => 'present',
}
```

Steps for Linux:

On the command line, type `sudo puppet resource user mimi ensure=present`. Notice the extra `ensure=present` command line argument. Any other combination of resource attributes and values can be added on the command line as well. Once the command executes, it will emit Puppet code indicating the current state of the resource that was updated. In this case, a brand-new user was created on the target node.

Observe the output after creating the new user with the `puppet resource` command:

```
# sudo puppet resource user mimi ensure=present
Notice: /User[mimi]/ensure: created
user { 'mimi':
  ensure => 'present',
}
```

Interestingly, if you run the same command again, the output will simply show the state of the resource on the target node but not attempt to create the user again. That is because Puppet operates **idempotently** and only takes action on the node when it is required. Once the user has been created, there is no need to create it again.

continued...

Step 4

The `puppet resource` command may be used to observe the configuration of the NTP service previously applied to your agent node.

Steps for Windows:

On the command line, enter `puppet resource service W32Time` and press `Enter`.

```
PS C:\> puppet resource service W32Time
service { 'W32Time':
  ensure => 'running',
  enable => 'true',
}
```

Steps for Linux:

On the command line, enter `sudo puppet resource service ntpd` and press `Enter`.

```
# sudo puppet resource service ntpd
service { 'ntpd':
  ensure => 'running',
  enable => 'true',
}
```

Lab 6.2: Using and extending facter

Objective:

In this lab, you will familiarize yourself with the core facts gathered by Facter. You will then extend Facter with an external fact for the next lab.

After completing this exercise, you will have done the following:

- Invoked Facter to display all of the facts on a single node.
- Invoked Facter to display the value of a structured fact.
- Invoked Facter to display the value of a single fact.
- Installed an external fact script so Facter can invoke it.
- Invoked Facter to display the value of the external fact.
- Displayed fact values in the Puppet Enterprise console.

Task 1: Invoke Facter

Lab Procedure

Facter is invoked on managed nodes during the Puppet agent run. It gathers information about the node and sends it to the Puppet master for use during the catalog compilation phase. Typical information gathered by Facter might include IP address, amount of installed memory, kernel version, and other machine-specific information.

You will invoke the following commands on your Windows or Linux classroom agent node, not on your local workstation.

Step 1



Steps 1-4 can be executed from either Windows or Linux. The output will vary, but the commands are the same. Try on both operating systems if you have time to compare the differences.

At the command line, type `facter` and press `Enter`.

continued...

Lab 6.2: Using and extending facter

Output should look similar to (abridged here):

```
$ facter
aio_agent_version => 5.5.1
augeas => {
  version => "1.10.1"
}
disks => {
  nvme0n1 => {
    model => "Amazon Elastic Block Store",
    size => "30.00 GiB",
    size_bytes => 32212254720
  }
}
dmi => {
  bios => {
    release_date => "10/16/2017",
    vendor => "Amazon EC2",
    version => "1.0"
  },
  is_virtual => true
[...]
```

continued...

Step 2

Notice in the previous sample output, the `is_virtual` fact is a simple string value while others are a nested structure of values. You can also invoke `Factor` so it will display a single fact instead of the complete set of facts that it knows about.

At the command line, type `factor os` and press `Enter`. This form of the command will retrieve the single `os` fact. This fact is structured and will display similar to the following (abridged here):

```
$ factor os
{
  architecture => "x86_64",
  distro => {
    codename => "Core",
    description => "CentOS Linux release 7.4.1708 (Core)",
    id => "CentOS",
    release => {
      full => "7.4.1708",
      major => "7",
      minor => "4"
    },
    specification => ":core-4.1-amd64:core-4.1-noarch:cxx-4.1-amd64:cxx-4.1-noarch:desktop-4.1-amd64:desktop-4.1-noarch:languages-4.1-amd64:languages-4.1-noarch:printing-4.1-amd64:printing-4.1-noarch",
  },
  family => "RedHat",
  hardware => "x86_64",
  name => "CentOS",
  release => {
    full => "7.4.1708",
  },
  [...]
}
```

Step 3

`Factor` also gathers and displays facts that have simple string values.

To show this capability, type `factor kernel` at the command line and press `Enter`. This fact has a simple string value, and the output will look like:

```
$ factor kernel
Linux
```

Step 4

So far, you have explored using `Factor` at the command line. As noted previously, facts gathered by `Factor` are sent to the Puppet master during catalog compilation. They are then available for display in the Puppet Enterprise console.

Open the console and click on the **INSPECT->Nodes** submenu. Click on a node in the list, and you will be taken to a page that displays the list of facts gathered from that node. Notice that both single values and structured values are displayed in the list.

continued...

Step 5

Facter gathers built-in (core) facts that are packaged with Facter, and it also can be configured with custom facts that are supplied by scripts you write or are provided by a third party. In this exercise, you will install a custom fact script that creates a `datacenter` fact and verify its function.

First, check that the `datacenter` fact does not already exist. At the command line, type `facter datacenter` and observe that there is no value displayed.

Next, move the provided custom fact script into place.

Steps for Windows:

- Open Windows Explorer.
- Download `datacenter.ps1` from the presentation download section and copy it to `C:\ProgramData\PuppetLabs\facter\facts.d`.
- Note: This moves the custom fact script into place where Facter can execute it. This is a simple PowerShell script that you can view in Microsoft Visual Studio Code. The script echoes a key/value pair and exits.

Steps for Linux:

- Download `datacenter.sh` to the Linux machine from the download section using curl.
 - On the downloads page, right click the script and select **Copy Link address**
 - Run `curl -o datacenter.sh '<pasted location>'`
- Create the facter location by running `sudo mkdir -p /etc/puppetlabs/facter/facts.d`
- Type `sudo mv datacenter.sh /etc/puppetlabs/facter/facts.d` and press `Enter`.
- Make the script executable by running `sudo chmod +x /etc/puppetlabs/facter/facts.d/datacenter.sh`
- Note: This command moves the custom fact script into place where Facter can execute it. This is a simple shell script that you can view with the command `vi /etc/puppetlabs/facter/facts.d/datacenter.sh`. The script echoes a key/value pair and exits.

Now verify that the fact has been installed and is accessible by Facter by typing `sudo facter datacenter` in Linux or `facter datacenter` in Windows. Notice the new output:

```
$ sudo facter datacenter
pdx

@@@ PowerShellConsole code_wrap nochrome
PS C:\> facter datacenter
pdx
```

NOT FOR DISTRIBUTION

Lab 7.1: Puppet forge

Objective:

In this lab, you will be introduced to the Puppet Forge, a large repository of pre-built Puppet modules. In many situations, using modules from the Puppet Forge will save you a great amount of time versus writing your own modules from scratch.

The Puppet Forge is an excellent resource for discovering Puppet code that configures popular technologies, such as time servers, database servers, web servers, and many other applications and technology stacks.

Lab Objective

In this lab, you will navigate the Puppet Forge web site and become familiar with its major features and some of the Puppet modules housed in it.

After completing this exercise, you will have done the following:

- Navigated the Puppet Forge web site and understood its major features.
- Displayed the modulepath where the Puppet master expects modules to be installed.
- Installed a Puppet module to manage the current system time.
- Classified a node with the system time configuration module and run the Puppet agent to configure it.
- Verified that the node's system time is correct after configuring system time with Puppet.

continued...

Task 1: Learn the Puppet Forge web site interface

Lab Procedure

Follow along with the instructor as he or she demonstrates the capabilities of the Puppet Forge. As the demonstration proceeds, you may perform the same steps in your own web browser.

Step 1

Navigate to the Puppet Forge web site <http://forge.puppet.com/> and observe the current number of modules available for download. New modules are continually added by Puppet, partner companies and 3rd party developers. If you eventually develop a useful module that you think the community could benefit from, the Puppet Forge is the place to house it.

The Forge web site has been constructed to help you quickly find what you need. Notice that you are prompted for something that you want to automate, what type of module (supported or otherwise), what platform support you need and whether you want a module with tasks included or not.

Step 2

Enter the word `ntp` in the text box on the Puppet Forge site, and press the `Search` button. If you are using a Windows agent node in this class, enter the word `winntp` instead. Notice that a large number of module listings are returned. The Forge web site includes a handy key along the right side of the results page to show you which modules are supported, have tasks or other features you may want.

You can customize your searches and search results by using the drop-down menus provided on the web site. For instance, the site allows you to constrain your searches to a specific operating system. And once you receive the search results, you can sort the results in various ways to find the modules that interest you the most.

Step 3

Locate the `puppetlabs/ntp` (Linux) or `tse/winntp` (Windows) module in the results from the search you performed above. Click into the result to view the detail page for the module.

You'll notice that the page includes a great deal of useful information, including supported operating systems, Puppet Enterprise version compatibility, how to add this module to your Puppet master (via a Puppetfile or the `puppet module install` command), and a complete set of documentation among other things. Spend some time navigating around the page and documentation tabs.

Also notice along the right side of the module's detail page that there is a quality score and community rating. Along with how frequently the module is updated and when the last release occurred, these numbers will help when you are looking for a module that meets your needs for stability and reliability.

After using the module yourself, you can also contribute by filling in the survey questions presented on the page. The Forge web site is designed with the community in mind, helping

Task 1: Learn the Puppet Forge web site interface

Puppet developers find what they need and rate the modules that are useful to them so others can find them as well.

continued...

NOT FOR DISTRIBUTION

Task 1: Learn the Puppet Forge web site interface

Step 4

Puppet modules are installed on the Puppet master so they are available during catalog compilation. They can also be installed on a workstation for use with the `puppet apply` command during Puppet code development. In either case, they must be installed in a **modulepath** directory so the Puppet toolset can find them.

To determine what the `modulepath` setting is, type `puppet config print modulepath` at a command prompt. You will see one or more directories displayed, separated by the platform separator character.

Observe the output after running the command (may look different for you):

Steps for Windows:

```
PS C:\Users\Administrator> puppet config print modulepath
C:/ProgramData/PuppetLabs/code/environments/production/modules;C:/ProgramData/PuppetLabs/code/mod
ules;C:/opt/puppetlabs/puppet/modules
```

Steps for Linux:

```
$ sudo puppet config print modulepath
/etc/puppetlabs/code/environments/production/modules:/root/puppetcode/modules:/etc/puppetlabs/cod
e/modules
```

Step 5

Puppet modules are very simply packaged as compressed `.tar` files. One installation method is shown on the module detail page using the `puppet module install` command.

Some modules have dependencies. That is, they borrow code from another module. In this lab, we'll install the `stdlib` module on our nodes, before installing the `ntp` or `winntp` modules that require it.

Copy and paste the module installation commands -- shown on each module's Forge page -- into the command line on the node:

Steps in Windows:

1. `puppet module install puppetlabs/stdlib --version 4.25.1`
2. `puppet module install tse/winntp --version 1.0.1`

Steps in Linux:

1. `sudo puppet module install puppetlabs/stdlib --version 4.25.1`
2. `sudo puppet module install puppetlabs/ntp`

continued...

Task 1: Learn the Puppet Forge web site interface

Step 6

Verify the installation of the module by running `puppet module list --tree` at the command line. Observe the output (which may look slightly different):

In Windows:

```
PS C:\Users\Administrator> puppet module list --tree
C:/ProgramData/PuppetLabs/code/environments/production/modules
├─ tse-winntp (v1.0.1)
│   └─ puppetlabs-stdlib (v4.25.1)
│       └─ puppetlabs-registry (v2.0.2)
C:/ProgramData/PuppetLabs/code/modules (no modules installed)
```

In Linux:

```
$ sudo puppet module list --tree
/etc/puppetlabs/code/environments/production/modules
├─ puppetlabs-ntp (v7.2.0)
│   └─ puppetlabs-stdlib (v4.25.1)
/etc/puppetlabs/code/modules (no modules installed)
```

Puppet modules may **depend** on other modules, meaning that they use some code from the dependency module. In this situation, all required modules must be installed to satisfy the dependency requirements.

The `puppet module install` command automatically installs all required module dependencies. In the example output above, notice that the `puppetlabs/stdlib` module is required for proper operation of the `puppetlabs/ntp` module.

continued...

Task 1: Learn the Puppet Forge web site interface

Step 7

The instructor has already installed the modules to the Puppet master, and they are available for classification and can be used to configure agent nodes with the time service on Linux or the Windows platform.

1. Log in to your Puppet Enterprise console account and browse to the **CONFIGURE->Classification** menu.
2. We are going to create a new classification group for your hosts and then configure the time service via Puppet.
3. At the top of screen, click the **Add Group** link
 - Leave **Parent name** set to **All Nodes**
 - In the **Group Name** field, type in **studentN-hosts** (substituting your proper student number in the name)
 - Leave **Environment** set to **production**
 - Leave **Environment group** unchecked
 - You may leave **Description** blank
4. Click **Add**
5. Click on your node's classification group
 - On the first tab that opens, the **Rules** tab, you need to include your hosts into your node group. In the bottom of the window, you will see an option to **Pin specific nodes to the group**. Click on the field labeled **Node name**. In the dropdown, select either your Linux or your Windows machine and click on the **Pin node** button.
 - Once completed, click on the **Commit 1 change** box in the lower right-hand corner of your screen.
6. Select the **Configuration** tab.
7. Start typing **ntp** in the **Add new class** text box, and you will see a list of classes that match that string.
 - If you are configuring a Windows machine, select **winntp** from the list.
 - If you are configuring a Linux machine, select **ntp** from the list.
 - Click the **Add Class** button to add the class to your group.
8. Configure a class parameter by clicking **Parameter name** in the **Parameter** box
 - Select **servers**.
 - Enter the value **["time.google.com", "time1.google.com"]** in the **Value** box
 - Click **Add parameter**.
9. Click the **Commit 1 change** button at the bottom of the page.

Step 8

Trigger a Puppet run on your machine with the `sudo puppet agent -t` command for Linux, or use `puppet agent -t` on Windows. Observe the output of the command showing the Puppet run and configuration of the time service on the node.

Once the run completes, verify the time on the system and ensure that it is accurate. For further verification that the service is configured correctly, open the `/etc/ntp.conf` file on the Linux platform to check the list of NTP servers or issue the `w32tm /query /status` command on the

Task 1: Learn the Puppet Forge web site interface

Windows platform.

continued...

NOT FOR DISTRIBUTION

Task 1: Learn the Puppet Forge web site interface

Step 9

You will need additional modules for future labs, and now that you are familiar with the `puppet module install` command, you can make sure extra modules are installed.

To complete the module installation process, run the following command:

Step on Windows:

```
puppet module install puppet/windows_firewall --version 2.0.1
```

Step on Linux:

```
sudo puppet module install ghoneycutt/ssh
```

NOT FOR DISTRIBUTION

Lab 8.1: Create a wrapper module

Objective:

In this lab you will create a new cross-platform module that will configure the time server on both the Windows and Linux platforms. You will use Puppet Development Kit (PDK) command line application to seed the initial module structure with content.

Task 1: Create Module Structure

In this task, you will create the `time` module with a single `time` class. The file that contains this class is named `init.pp` because the module that it contains is named `time`. This allows Puppet to automatically load this class by searching the `modulepath` for its name when you declare it.

Step 1

Creating a new PDK module for time. This step is the same for Windows and Linux.

In your command prompt type `pdk new module` and press `Enter`. You will see several questions requiring an answer. Enter the answers as you see below:



Replace the N in studentN with your student number, e.g. `student8`

Question	Answer
Module Name	<code>time</code>
Forge Name	<code>studentN</code>
Credit author	<code>Student N</code>
License	<code>Apache-2.0</code>
Operating systems	RedHat and Windows

- Type `y` to generate the module in the current directory.

continued...

Step 2

Change the current working directory to the module you created using the `cd` command. Then run the `pdk` command to create a new class called `time`. The commands are the same for Windows and Linux:

1. `cd time`
2. `pdk new class time`

You should see output that it created the new class files as such:

For Windows:

```
pdk (INFO): Creating 'C:/Users/Administrator/time/manifests/init.pp' from template.  
pdk (INFO): Creating 'C:/Users/Administrator/time/spec/classes/time_spec.rb' from template.
```

For Linux:

```
pdk (INFO): Creating '/home/centos/time/manifests/init.pp' from template.  
pdk (INFO): Creating '/home/centos/time/spec/classes/time_spec.rb' from template.
```

continued...

Task 2: Add classes to your wrapper class

Open up the Visual Studio Code application to add resources to the `time` class.

Steps for Windows:

1. Choose File -> Open File.
2. Change directory into the `time` module directory.
3. Change directory into the `manifests` directory.
4. Open the `init.pp` file.
5. Declare the classes `winntp` and `ntp`.
6. Both of these two Forge modules' main classes have a parameter `servers`. Use a variable named `$servers` to pass the array `['time.google.com']` to both classes.
7. Create a case statement that uses the `$facts['kernel']` fact to apply the `winntp` class to Windows and apply the `ntp` class as the default selection.

Code:

```
class time {  
  
    $servers = ['time.google.com']  
  
    case $facts['kernel'] {  
        'windows': {  
            class { 'winntp':  
                servers => $servers,  
            }  
        }  
        default: {  
            class { 'ntp':  
                servers => $servers,  
            }  
        }  
    }  
}
```

continued...

Steps for Linux:

You will be adding these changes through vi.

1. Open the init.pp file using vi, `vi manifests/init.pp`
2. Replace content in the file with this code. Press `i` to enter insert mode and copy the following content into the file:

Code:

```
class time {  
  
    $servers = ['time.google.com']  
  
    case $facts['kernel'] {  
        'windows': {  
            class { 'winntp':  
                servers => $servers,  
            }  
        }  
        default: {  
            class { 'ntp':  
                servers => $servers,  
            }  
        }  
    }  
}
```

1. Save the file. Hit the `esc` key. Type `:wq`, then hit `Enter`.

continued...

Task 3: Version control your module

Now that you have created your module, you should add it to version control to track changes. To do this you must first create a local git repository and `add` the current state of all files to be tracked by git.

Step 1

Before we can use git we need to configure it. These steps are the same for Windows and Linux. In your console window type:

1. `git config --global user.email "noreply@puppet.com"`
2. `git config --global user.name "studentN"`

Step 2

Now commit the code of your newly created module and class.

Steps for Windows:

1. Open the Visual Studio Code editor.
2. Choose File -> Open Folder.
3. Change directories back up into the main `time` directory and click `Select Folder` to open it.
4. Click on the Source Control icon on the left of the window (third down, below the magnifying glass icon).
5. Click on the small diamond icon in the upper left of the pane (its tool tip says "Initialize Repository").
6. Click the button `Initialize Repository` in the resulting dialog.
7. Click the ellipsis `...` where the small diamond icon was in the upper left of the pane.
8. Choose `Stage All Changes`.
9. Using the ellipsis choose `Commit Staged`.
10. Type "Initial Commit" into the commit message dialog and hit Enter.

Changes should now be displayed as (0).

continued...

Steps for Linux:

If you are not in the base directory of your module, change directories until you are at the top of your module time, i.e.:

```
time ## YOU WANT TO BE HERE
├─ appveyor.yml
├─ CHANGELOG.md
├─ examples
├─ files
├─ Gemfile
├─ manifests
│   └─ init.pp
├─ metadata.json
├─ Rakefile
├─ README.md
├─ spec
│   ├── classes
│   │   └─ time_spec.rb
│   ├── default_facts.yml
│   └─ spec_helper.rb
├─ tasks
└─ templates
```

Type the following:

1. `git init`
2. `git add --all`
3. `git commit -m"Initial Commit"`

continued...

Task 4: Publish your module

The commands in Step 1 and 2 are the same for Windows or Linux.

Step 1

To allow your module to be shared among your colleagues and Puppet masters, you need to push it to a `remote` git server. The following command will add the URL to push to using the name `origin`.

From your console window, in the base of your time module, run the following command:

```
git remote add origin git@gitlab.classroom.puppet.com:puppet/time.git
```

Step 2

To keep your module changes from affecting other users in the class, create a branch named after your student number.

```
git checkout -b studentN
```

continued...

Step 3

Steps for Windows:

1. Open Visual Studio Code.
2. Click on the Source Control icon on the left of the window (third down, below the magnifying glass icon).
3. Click the ellipsis `...` where the small diamond icon was in the upper left of the pane.
4. Choose `Publish branch`.
5. Choose `No` in the dialog in the bottom right corner relating to periodically running `git fetch`.

Steps for Linux:

1. Type `git push origin studentN`

Step 4

Review your published module

1. Open your web browser.
2. Navigate to `https://classXXX-gitlab.classroom.puppet.com/puppet/time/tree/studentN`
3. You should see the files you previously committed displayed.

If prompted to sign-in, use the username `studentN` and the password `puppetlabs`

Step 5 (Optional)

To get the now existing time module on the opposite workstation, follow these steps.

Steps if you created the module on Windows and want to clone it to Linux:

1. As the centos user run, `git clone git@gitlab.classroom.puppet.com:puppet/time.git`
2. Type (substituting your student number) `git fetch origin studentN`
3. Type (substituting your student number) `git checkout studentN`

Steps if you created the module on Linux and want to clone it to Windows:

1. Click on View --> Command Palette
2. Type in `git clone` and choose Git: Clone
3. Insert the URI `git@gitlab.classroom.puppet.com:puppet/time.git` and press enter

Task 4: Publish your module

4. In the pop-up window, top left click `New folder` and name it time
5. Click the Select Repository Location
6. Next click on the Open Repository button at the bottom left

NOT FOR DISTRIBUTION

Lab 9.1: Test module syntax and style

Objective:

In this lab, you will perform syntax, style, and compilation testing on the module you created in the previous lab. Unit and system testing will not be covered.

Task 1: Validate the Syntax of your module

1. Open the Visual Studio Code editor.
2. Choose File -> Open File.
3. Select the `time` module directory that you created previously.
4. Change directories into `manifests`.
5. Edit the `init.pp` file you created previously.
6. Select the ellipsis in the upper right hand corner of the editor pane.
7. Choose 'PDK Validate' from the drop down menu.

Expected Output

```
PS C:\Users\Administrator\time> pdk validate
pdk (INFO): Running all available validators...
pdk (INFO): Using Ruby 2.4.4
pdk (INFO): Using Puppet 5.5.2
[✓] Checking metadata syntax (metadata.json tasks/*.json).
[✓] Checking module metadata style (metadata.json).
[✓] Checking Puppet manifest syntax (**/*.pp).
[✓] Checking Puppet manifest style (**/*.pp).
[✓] Checking Ruby code style (**/*.rb).
PS C:\Users\Administrator>
```



If the output of your PDK Validate does not appear close to the above, review the error and attempt to fix it using the steps in the next task.

continued...

Task 2: Simulate a Syntax Error

If you had no errors in your syntax, then proceed to simulate one. If you had an error skip to Step 2.

Step 1

1. Edit the `init.pp` file to introduce a syntax error.
2. Remove the `$` from the `$servers` variable.
3. Choose `File -> Save`.

Step 2

1. Select the ellipsis in the upper right hand corner of the editor pane.
2. Choose 'PDK Validate' from the drop down menu.

Expected Output

```
PS C:\Users\Administrator\time> pdk validate
pdk (INFO): Running all available validators...
pdk (INFO): Using Ruby 2.4.4
pdk (INFO): Using Puppet 5.5.2
[✓] Checking metadata syntax (metadata.json tasks/*.json).
[✗] Checking module metadata style (metadata.json).
[✗] Checking Puppet manifest syntax (**/*.pp).
[✓] Checking Ruby code style (**/*.rb).
Error: puppet-syntax: manifests/init.pp:9:7: Could not parse for environment production: Illegal
attempt to assign to 'a Name'. Not an assignable reference
```

In the error message above, the message indicates the error in syntax is in `manifests/init.pp` on line `9` and column `7`.

Task 3: Fix your syntax Error(s)

Using the `PDK Validate` output and the syntax highlighting, find all errors in your code and resolve them.

1. Identify each line and column and match that to the error message from PDK.
2. Attempt to fix the error by referring to the solutions guide or working with your instructor.

Lab 10.1: Create roles and profiles

Objective:

After completing this exercise you will have done the following:

- Cloned a Puppet Enterprise control repository.
- Declared the time module's class from the previous lab in the base profile.
- Created a bastion server role that will include the base profile.

i Any time you see classXXXX, it is referring to your class id, such as class1809.

i Any time you see studentN, it is referring to your assigned student number, such as student4.

Required Resources

These are the resources required to complete this lab:

- A Windows host machine using Visual Studio Code (or Linux shell if you prefer)

continued...

Task 1: Clone the control repository

In this task, you will clone the control repository down to your Windows workstation.

Step 1

Clone the control repository.

Steps for Windows:

To begin, open `Visual Studio Code`.

- Once open, click on **View**
- Click on **Command Palette**
 - In the box that opens, start typing `git clone` and select `Git: Clone` when it appears
 - In the `Repository URL` field, enter `git@gitlab.classroom.puppet.com:puppet/control-repo.git` and press enter
 - In the dialog box that opens, click on **Select Repository Location**
 - You will be prompted in the lower right-hand corner to `Open Repository`; click the button
- Now that you've successfully cloned the repository, switch to your student branch.
 - In the lower left-hand corner you will see a branch icon followed by the word `production`, click on it
 - In the `Select a ref to checkout` dialog list, select `origin/studentN` where N is your student id

Steps for Linux:

```
git clone git@gitlab.classroom.puppet.com:puppet/control-repo.git
cd control-repo
git checkout studentN
```

Task 2: Create Base Profile

With your control repository cloned locally, you are ready to start updating Puppet code. In this task, you will take the `time` class you created in the previous lab and declare it in a profile.

Step 1

Time is a perfect module to add to the base profile: that is, a profile that should be configured on every workstation. Declare its class now.

In the `control-repo/site/profile/manifests` directory, create a new file called `base.pp`. It will open in the Visual Studio Code window on the right. In this simple profile, you are going to include your time profile. In this way, you will ensure that all systems apply the `base` profile and included in the `base` profile is the time class.

Task 1: Clone the control repository

Update your file to look like this:

```
class profile::base {  
  include time  
}
```

The below commands use relative paths, and it's assumed you're running them from `C:\Users\Administrator`

Test the syntax. From your shell window run:

```
puppet parser validate control-repo\site\profile\manifests\base.pp
```

Now when you apply the base profile to all workstations, they will have NTP configured by default.

continued...

Task 2: Add time module to control repo branch.

In Visual Studio Code or `vi`, open the file named **Puppetfile** in the control repository.

In the Puppetfile, add a module (`mod`) entry for the git remote you pushed your module to previously.

```
mod 'time',
  :git => 'git@gitlab.classroom.puppet.com:puppet/time.git',
  :branch => 'studentN'
```

Set the branch key to your student number.

Task 3: Create Bastion Host Role

Now that you have a base profile, go one level further of abstraction and place it in a role. Roles are used to easily classify node types in Puppet Enterprise Console.

For this example, you'll ensure all bastion hosts have the base profile applied. It applies the class `ntp` or `winntp` to Linux or Windows respectively.

Step 1

To add the new role in Visual Studio Code, navigate from **site** to **role** to **manifests**.

```
site
|_role
|_manifests
```

Add a new file under **manifests** and name it **bastion.pp**.

Step 2

To configure the bastion role, enter the following code into the empty file:

```
class role::bastion {
  include profile::base
}
```

Test the syntax. From your shell window run:

```
puppet parser validate control-repo\site\role\manifests\bastion.pp
```

As before, if any syntax errors are detected, fix them and re-run the validation command.

continued...

Step 3

Now that you have tested your profile and role, it is time to apply it to your workstation. In this lab, you will test it with `puppet apply`. In a future lab, you will see how to classify your node in the Puppet Enterprise master.

You are going to create an `examples` directory where you can place files to test your profile.

- In Visual Studio Code, right click **site** and then **role** and select **New Folder** and name it `examples`
- Inside the `examples` folder, add a file called `bastion.pp`

Example:

```
site
|_role
|_examples
|_bastion.pp
```

Add these contents into that file and save the file:

```
include role::bastion
```

Save the `bastion.pp` file.

Now you can smoke test this class and then apply it locally. Click the Windows Start button in the bottom left hand corner of the workstation and scroll down to Puppet. In that folder, open the program **Start Command Prompt with Puppet**.

Once that is open, change directory into your `control-repo\site\role` directory. First run a smoke test (noop) and then `puppet apply` the role.

Steps for Windows:

```
cd c:\Users\Administrator\control-repo\site\role
# This is the smoke test. --noop is a dry-run.
puppet apply examples\bastion.pp --noop --
modulepath='C:/ProgramData/PuppetLabs/code/environments/production/modules;C:/Users/Administrator/control-repo/site;C:/Users/Administrator'

# Apply it for real
puppet apply examples\bastion.pp --
modulepath='C:/ProgramData/PuppetLabs/code/environments/production/modules;C:/Users/Administrator/control-repo/site;C:/Users/Administrator'
```

Verify Puppet actually changed your time server `time.google.com`.

Type `w32tm /query /peers`.

Your output should look like this:

```
#Peers: 1

Peer: time.google.com,0x9
State: Active
```

Task 2: Add time module to control repo branch.

Steps for Linux:

```
cd control-repo/site/role
```

This is the smoke test. --noop is a dry-run. (This is a single line command.)

```
sudo puppet apply examples/bastion.pp --noop --  
modulepath=/etc/puppetlabs/code/environments/production/modules:/home/centos/control-  
repo/site:/home/centos
```

Apply it for real. (This is a single line command)

```
sudo puppet apply examples/bastion.pp --  
modulepath=/etc/puppetlabs/code/environments/production/modules:/home/centos/control-  
repo/site:/home/centos
```



What is `modulepath`? We need to supply this option so Puppet knows where to look for Puppet code.

Verify Puppet actually changed your time server to `time.google.com`

Type `grep server /etc/ntp.conf`.

Your output should look like this:

```
server time.google.com
```

continued...

Step 4

Commit your branch.

Steps for Windows:

In Visual Studio Code, click the source control icon on the left menu. It will ask for a commit message. Type:

```
Adding Bastion Role and Base Profile
```

Beside all of your additional and modified files, click the + to stage the changes. Click the checkmark at the top to commit the changes. Click the ... in the Source Control:Git menu and select **Push**.

Steps for Linux:

```
cd control-repo
git add .
git commit -m 'Adding Bastion Role and Base Profile'
git push origin studentN
```

In the next lab, you will expand on the base profile.

Lab 12.1: Expand initial roles and profiles

Objective:

In this lab you will create a security profile. This role will include a Windows firewall profile and a Linux ssh profile. The base role will be expanded to include the new security profile.

In this lab, you will apply Puppet code locally to make changes, commit the code to GitLab and then push the changes to your workstations via the Puppet master.

After completing this exercise you will have done the following:

- Expanded the base profile
- Implicitly expanded the bastion role
- Extended learning of applying roles by classifying in the PE console

Required Resources

These are the resources required to complete this activity:

- A Windows or Linux host machine using Visual Studio Code or Linux shell

Additional modules were preloaded for you, including the `window_firewall` and `ssh` modules and all of their dependencies. If you would like to see them on the Puppet Forge, they are located here:

[Windows_Firewall](#)

[SSH](#)

continued...

Task 1:

Step 1

Now you are going to create folders to contain OS-specific profiles.

i Starter code is already in the control-repo

Steps for Windows

The content is to enable the Windows Firewall and open three ports, for ICMP (ping), WinRM (remote scripting) and RDP (remote console) access.

In Visual Studio Code, validate the content of the `site/profile/manifests/baseline/windows/firewall.pp` is as follows.

```
class profile::baseline::windows::firewall {

  class { '::windows_firewall':
    ensure => 'running',
  }

  windows_firewall::exception { 'Permit ICMPv4':
    ensure      => present,
    direction   => 'in',
    action      => 'allow',
    enabled     => true,
    protocol    => 'ICMPv4',
    display_name => 'Permit ICMPv4',
    description => 'Permit ICMPv4',
  }

  windows_firewall::exception { 'WINRM':
    ensure      => present,
    direction   => 'in',
    action      => 'allow',
    enabled     => true,
    protocol    => 'TCP',
    local_port  => 5985,
    remote_port => 'any',
    display_name => 'Windows Remote Management HTTP-In',
    description => 'Inbound rule for Windows Remote Management via WS-Management. [TCP 5985]',
  }

  windows_firewall::exception { 'RDP':
    ensure      => present,
    direction   => 'in',
    action      => 'allow',
    enabled     => true,
    protocol    => 'TCP',
    local_port  => 3389,
    remote_port => 'any',
    display_name => 'Windows RDP',
    description => 'Inbound rule for Windows RDP. [TCP 3389]',
  }
}
```

continued...

Task 1:

Steps for Linux

Adding the Linux security profile.

Like the firewall profile for Windows, this will enable root login and create a session timeout of two hours on Linux workstations.

Validate the content of `site/profile/manifests/baseline/linux/ssh_config.pp` as follows

```
class profile::baseline::linux::ssh_config {
  class { ['::ssh']:
    permit_root_login      => 'yes',
    sshd_client_alive_interval => '7200',
    sshd_client_alive_count_max => '0',
  }
}
```

Step 2

Adding a security_baseline profile.

Now it is time to wrap together the security profiles into a single profile. As you add additional security modules under each operating system, they can be nested into this `security_baseline` profile for simplified classification.

This profile contains a case statement that takes the `kernel` fact and determines what profiles to apply based on the operating system.

Validate the content of `site/profile/manifests/security_baseline.pp` as follows

```
class profile::security_baseline {
  # OS-specific
  case $facts['kernel'] {
    'windows': {
      include profile::baseline::windows::firewall
    }
    'Linux': {
      include profile::baseline::linux::ssh_config
    }
    default: {
      fail('Unsupported operating system!')
    }
  }
}
```

continued...

Step 3

Adding the security_baseline to the base profile.

Security is something you want on all workstations, so it would be a good profile to add to `base.pp` to ensure every workstation gets the proper security configuration.

Open the file `site/profile/manifests/base.pp`, and declare the `profile::security_baseline` class.

```
class profile::base {
  include time
  include profile::security_baseline
}
```

Save the `base.pp` file.

Step 4 (Optional)

Adding the smoke test.

For good measure, add the example file to allow local `puppet apply` testing.

- In Visual Studio Code, right click **site** and then **profile** and select **New Folder** and name it `examples`
- Inside the `examples` folder, add a file called `security_baseline.pp`

Add the following line to the `security_baseline.pp` file:

```
include profile::security_baseline
```

Save the `security_baseline.pp` file.

Steps for Windows

You can test the new profile by switching to your Command Prompt with Puppet terminal window.

Change directory to `C:\Users\Administrator\control-repo\site` and execute `puppet apply`:

```
# This is the smoke test.
cd C:\Users\Administrator\control-repo\site
puppet apply profile\examples\security_baseline.pp `
  --modulepath='C:/ProgramData/PuppetLabs/code/environments/production/modules;'`
  C:/ProgramData/PuppetLabs/code/modules;.` --noop
```

continued...

Task 1:

Steps for Linux

```
cd control-repo/site
# This is the smoke test.
sudo puppet apply profile/examples/security_baseline.pp \
  --modulepath=/etc/puppetlabs/code/environments/production/modules:\
  /home/centos/control-repo/site:/home/centos --noop
```



You will see yellow *This method is deprecated* warnings. These are safe to ignore for this lab.

Step 5

Commit changes and merge them.

In Visual Studio Code, click the source control (git icon) on the left menu. It will ask for a commit message. Type:

```
Adding security_baseline profile
```

Beside all of your additional and modified files, click the + to stage the changes. Click the checkmark at the top to commit the changes, and click the ... in the Source Control:Git menu and select **Push**.

continued...

Task 2: PE Roles and Profiles

Create Roles and Profiles in the PE console.

This task involves creating similar roles and profiles that you made in the control repository, but this time it will be used in Puppet Enterprise to be used in the next task of classifying nodes.

Log into your Puppet Enterprise console from your browser: `https://classXXX-master.classroom.puppet.com/`

Enter the credentials:

username: `studentN`

password: `puppetlabs`

In the left menu, under **CONFIGURE**, click **Classification**.

Create an environment group

The first step is to create a new node group that will have the environment group checkbox selected. This group will assign your nodes to run in your environment (studentN), which corresponds to your branch in the control-repo.

To start:

- Expand **All Nodes**.
- Click **Add group...** at the top.
- Enter the following:
 - Parent name: `Production Environment`
 - Group name: `studentN-env`
 - Environment: `studentN` # Where N is your student number
 - Environment Group: `checked`
 - Description: `This group sets the environment for studentN`
 - Click **Add**
- Now that the group has been created, expand the `Production Environment` group (click the + sign to the left of the name) and click on your new **studentN-env** group
- On the `Rules` tab, navigate to the section that says `Pin specific nodes to the group`
- Click on **Node name** and select only your Linux machine from the list
- Next, add your Windows machine from the list
- Finally, click on **Commit 2 changes** in the lower right hand corner

Next, we are going to update the classes assigned in your `studentN-hosts` group.

To begin:

- Navigate back to the main classification page

Task 2: PE Roles and Profiles

- Click on **studentN-hosts**
- First, we need to update the metadata of this node group to filter classes that belong to the `studentN` environment (your branch)
 - Click on the **Edit node group metadata** link
 - In the Parent dropdown, select `studentN-env` from the list
 - In the Environment dropdown, select `studentN` from the list
 - Click **Commit 1 change** in the lower right-hand portion of the screen
- Navigate to the `Configuration` tab and click on **x Remove all classes** and click on **Commit ... changes** in the lower right-hand corner
- Now, in the `Add new class` field, start typing `role::` and select `role::bastion` from the list.
- Note, if you do not see that role in the list please click on **Refresh** on the right hand side of the screen to force the console to re-read the code on the PE master.
- Click **Add class** and finally **Commit 1 change**
- Back on the rules tab, ensure both of your hosts are pinned to this group.

continued...

Task 3: Deploying the Role

Force Puppet to run and apply the bastion role.

In the left menu, under **RUN**, click **Puppet**:

- In the **Job Description** field, enter `Deploy the bastion role`
- Under **Inventory** select **Node Group**
- Under **Choose a node group** select `studentN-hosts`
- Click the **select** button.

You should see both workstations returned.

Click **Run job** in the blue bar at the bottom of the screen.

This will take a few minutes as Puppet runs and collects data on both nodes. You should see them return Succeeded in the Job node status.



If you click the date/time stamp under Report, it will give you details of the Puppet run.

Lab 13.1: Class params

Objective:

As you've learned, roles and profiles are an important construct that allow you to minimize rework and ensure that you have common implementations of various technologies. In this lesson, you'll expand on this topic and show how to allow customization of your shared profiles so that they are truly reusable for all conditions.



Any time you see classXXXX, it is referring to your class id, such as class1809.



Any time you see studentN, it is referring to your assigned student number, such as student4.

Activity Objective

In this activity, you will be updating the time profile you had written previously. We'll reclassify our nodes several times to show all the ways you can manipulate data from within the Puppet Enterprise console.

After completing this exercise you will have a solid understanding of:

- How to add class parameters to a piece of Puppet code.
- How to adjust data for a class from within the Puppet Enterprise console.
- How classification impacts class parameters.

Required Resources

These are the resources required to complete this activity:

- A Windows host machine using Visual Studio Code (or Linux shell if you prefer)

Task 1: Update time module

In this task we'll make a slight change to the existing code in your `time` module.

continued...

Step 1

Connect to your Windows student machine, and open Visual Studio Code with your cloned control-repo. You are going to change your code from this:

```
class time {
    $servers = ['time.google.com']
    ...
}
```

To this:

```
class time (
    Optional[Array[String]] $servers = undef,
) {
    ...
}
```

So what did we just do? Well, for one, we deleted the static values supplied on line 3 of the previous version, `$servers = ['time.google.com']`. We've updated this class to make use of a class parameter (line 2 of the new version). Line 2 defines the `$servers` parameter, which is undefined by default. This means that if no value is passed for this parameter, the default time servers in class `winntp` and `ntp` will be used.

Go ahead and commit your code, and push to GitLab.

Steps for Windows:

- In Visual Studio Code, click the Source Control (fork) icon on the left.
- Click the checkbox next to `SOURCE CONTROL: GIT` (and click `yes` if prompted).
- Type in your commit message and press Enter.
- Click the ellipsis (...) to the right of `SOURCE CONTROL: GIT` and click `Push`.
- Ask the instructor to deploy code for your environment.

Steps for Linux:

```
cd time
git add .
git commit -m 'Added servers param to time class'
git push origin studentN
```

continued...

Step 2

Now that we've updated the time profile, we will proceed to make some changes to the time servers we are supplying.

- Open the Puppet Enterprise console.
- In classification, navigate to your `studentN-hosts` node group.
- In the top right corner of the `studentN-hosts` node group interface, click the **Run** dropdown. Click **Puppet**.
- In the `Run Puppet` UI, click **Run job** in the lower right hand corner.

You'll no doubt notice that the Puppet run has made changes, but you didn't update anything. However, you did remove the static servers list, so the modules `ntp` and `winntp` are now using their default values.

Step 3

For step 3, we are going to set the values of the time servers in the Puppet Enterprise console. Before we can do this, we have to make some changes to our classification. Right now we are using a role to classify our hosts. Before we see how to adjust data values of a profile included in a role, we will first do so for a profile that is directly classified.

Start by changing the classification rules:

- Open the Puppet Enterprise console.
- In classification, navigate to `studentN-hosts`.

Now, add your parameter:

- Scroll down to the **Data** section at the bottom of the **Configuration** tab.
- Populate the `Class` field with the following data: `time`
- Populate the `Parameter` field with the following data: `servers`
- Populate the `Value` field with the following data: `["time.google.com", "time1.google.com"]`.
- Click **Add data** on the right hand side.
- Finally, click **Commit ... changes** in the lower right corner.



Remember that you set up your time profile to accept an array? The Puppet Enterprise console also allows you to supply structured data (like you just did) in JSON format. You must enter any structured data in JSON format or it will be converted to a string. Make sure to use double quotes around the values inside the JSON array.

You're ready to run Puppet:

Again, in the Puppet Enterprise console on the `studentN-hosts` group:

- In the top right corner, click the **Run** drop down and select `Puppet`

Lab 13.1: Class params

- Click **Run job**
- You will now see that both hosts have made changes. Click on their respective **Report** links to view the details.

NOT FOR DISTRIBUTION

Lab 14.1: Application

Objective:

The labs completed thus far were primarily focused on operating system configurations, but you can do much more than that with Puppet. To demonstrate this, the next lab will have you go through building a cross platform role that can be used to deploy a simple Java application.

After completing this exercise you will have done the following:

- Created the required role and profile structure for deploying a Java application.
- The role/profile will:
 - Set up required directory structure.
 - Install Java.
 - Copy the application JAR into place.
 - Configure the JAR to be run as a system service.
 - Work for both Windows and Linux.

Task 1: Create the role/profile

The first thing we need to do is actually create our new role and supporting profiles.

Step 1

- Open Visual Studio Code on your Windows student machine:
 - Within your control-repo, navigate to `site/role/manifests` and create a new file called `sample_app.pp`.

Enter the following contents:

```
class role::sample_app {  
  include profile::base  
  include profile::sample_app  
}
```

- Now you'll create the supporting `profile::sample_app` profile:
 - Within your control-repo, navigate to `site/profile/manifests` and create a new file called `sample_app.pp`.

continued...

Lab 14.1: Application

Enter the following contents:

```
class profile::sample_app {
  case $facts['kernel'] {
    'windows': {
      include profile::sample_app::windows
    }
    'Linux': {
      include profile::sample_app::linux
    }
    default: {
      fail('Unsupported operating system!')
    }
  }
}
```

i You are adding one more level of abstraction here, a different profile depending on if this is a Windows or Linux host. This gives us the ability to use one top-level role that can be applied regardless of platform (similar to the `role::bastion` that you used before).

You now need to create the operating system specific profiles:

- Within the `site/profile/manifests` folder, create a new folder called `sample_app`.
- The contents of the Linux and Windows manifests can be found in the `Downloads` page on the presentation for this module. Download a copy of `lab14_linux.pp` and `lab14_windows.pp` to your student machine.
- Copy `lab14_linux.pp` to the `site/profile/manifests/sample_app` folder in your control-repo, and rename the file to `linux.pp`.
- Copy `lab14_windows.pp` to the `site/profile/manifests/sample_app` folder in your control-repo, and rename the file to `windows.pp`.
- Next, you need to copy your application JAR file into place.

i For this lab you use the built-in Puppet file server to serve up the JAR file (~5MB). Normally, application artifacts would be served from an artifact repository, HTTP server, or similar file share and not from a module. To keep it simple, this lab has you store the binary with the module.

continued...

Lab 14.1: Application

- To place the JAR file, create a new folder called `files` inside of the `site/profile` directory.
 - Download the JAR file from the `Downloads` page on the presentation for this module (the file is called `puppet_webapp.jar`).
 - Copy the JAR file into the `files` directory you just created
- Inspect the manifests. For both platforms you will see the profile:
 - Creates a directory structure for installing the application.
 - Copies the required JAR file from the Puppet master file server.
 - Sets up logging.
 - Creates a service definition and starting it.
- Finally, you need to commit and push your code updates.

Cheat sheet:

- In Visual Studio Code, click the Source Control (fork) icon on the left.
- Click the checkbox next to `SOURCE CONTROL: GIT` (and click `yes` if prompted).
- Type in your commit message and press Enter.
- Click the ellipsis (...) to the right of `SOURCE CONTROL: GIT` and click `Push`.

continued...

Task 2: Classification

Now that you've got your new role and profiles ready, you need to update your classification.

- To start, open the PE console:
 - Navigate to `Classification` and click on the **studentN-host** group.
 - On the `Configuration` tab, remove all classes
 - Click on the **Remove all classes** link
 - Click on **Commit ... Changes** in the lower right-hand corner
 - Now, you'll add your new role:
 - Click the **Add new class** input field and type/select `role::sample_app`.
 - Click **Add class**.
 - Click **Commit 1 change** in the lower right-hand corner.

continued...

Task 3: Run Puppet

Now run your code and see the application get deployed.

- In the `studentN-hosts` node group, click on **Run** in the top right corner and select `Puppet` from the drop down.
- In the `Run Puppet` UI, click on **Run job** in the lower right hand corner.
- Inspect the agent reports to view the changes.

Validate the application is running properly

- Windows:
 - Open a web browser and navigate to: `http://localhost:9999`
- Linux:
 - Use curl to validate the embedded web server is responding: `curl -L localhost:9999`

NOT FOR DISTRIBUTION