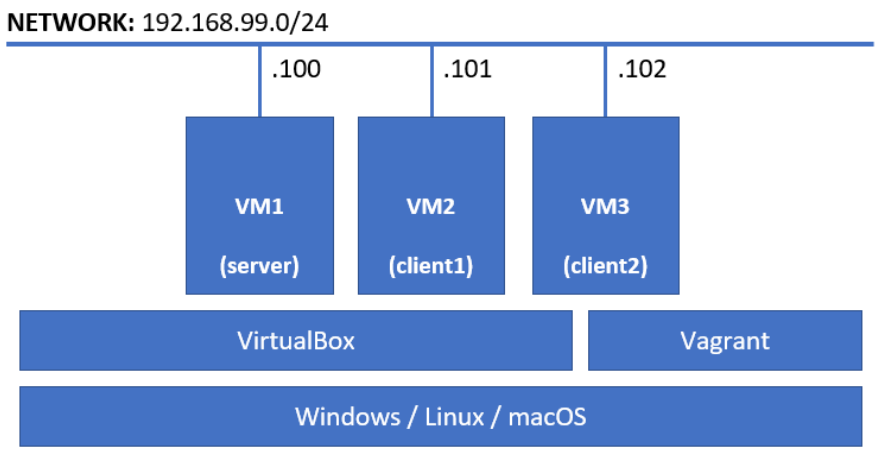
Practice M5: Puppet

For this practice, our lab environment will look like this



We are going to use mostly **CentOS Stream 8** boxes and at least one **Debian**-based box

All configurations and supplementary files are provided as a ZIP archive and can be downloaded from the module section in the official site

Part 1

First, bring up the environment using the provided **Vagrantfile**

Install Puppet Server

The requirements and procedure are outlined here:
https://puppet.com/docs/puppet/7/system_requirements.html#system_requirements

Preparation

Before we start, we should take care for a few preparation steps

First, we should take care of the name resolution. For example, via **/etc/hosts**

Then, we should configure the systems for accurate time

This set of tasks **must be repeated on the client nodes as well**

On CentOS

Install and activate **NTP** solution (for example **Chrony**)

```
sudo dnf install -y chrony
sudo systemctl enable chronyd
sudo systemctl start chronyd
```

Set **SELinux** to permissive mode for the current session

```
sudo setenforce permissive
```

And for the next boot

```
sudo sed -i 's\=enforcing\=permissive\g' /etc/sysconfig/selinux
```

On Debian/Ubuntu

Install **NTP** server (it will be automatically enabled and started)

```
sudo apt-get install -y ntp
```

Actual Installation

In order to install the **Puppet server** we must follow these steps

On CentOS 8.x *

** CentOS 9.x is supported only for agent installation*

Now, we are ready to install the repository and the server itself

Add the repository

```
sudo dnf install -y https://yum.puppet.com/puppet7-release-el-8.noarch.rpm
```

Install the **Puppet** server

```
sudo dnf install -y puppetserver
```

On Debian/Ubuntu

Now, we are ready to install the repository and the server itself

Download the repository package (for **Ubuntu 20.04**)

```
wget https://apt.puppet.com/puppet7-release-focal.deb -O repository.deb
```

Or for Debian 11 (Bullseye)

```
wget https://apt.puppet.com/puppet7-release-bullseye.deb -O repository.deb
```

Install the repository package

```
sudo dpkg -i repository.deb
```

Install the package

```
sudo apt-get update
sudo apt-get install -y puppetserver
```

Post Installation

We must execute few more configuration steps in order to configure security

Adjust the secure path (**secure_path**) variable – add **/opt/puppetlabs/bin**

```
sudo visudo
```

Restart the shell

```
exec $SHELL
```

Do initial configuration of server name

```
sudo puppet config set dns_alt_names puppet-server,puppet-server.do2.lab
sudo puppet config set server puppet-server
sudo puppet config set caserver puppet-server
sudo puppet config set reportserver puppet-server
```

Should we want, we can always check the current configuration with

```
sudo puppet config print
```

Generate certificates

```
sudo puppetserver ca setup
```

We must adjust the configuration before attempt to start the service

By default, the **Puppet** server is configured to use 2GB of RAM and this is the total amount of memory for our VM

On **CentOS**, we must edit the **/etc/sysconfig/puppetserver** file

And on **Debian**/**Ubuntu**, we must edit the **/etc/defaults/puppetserver** file

Change default values to 1GB

```
JAVA_ARGS="-Xms1g -Xmx1g"
```

Or to 512MB

```
JAVA_ARGS="-Xms512m -Xmx512m"
```

And then save and close the file

Start and enable the server

```
sudo systemctl start puppetserver
sudo systemctl enable puppetserver
```

Check the installed version with

```
puppetserver -v
```

One more post-installation step is necessary on the server – to open port **8140/tcp**

If we are running **firewalld**, then we can execute the following

```
sudo firewall-cmd --add-port=8140/tcp --permanent
sudo firewall-cmd --reload
```

More details on the firewall configuration here:
https://puppet.com/docs/puppet/7/system_requirements.html#firewall_configuration

Install and Register Puppet Agents

First, we will install the agent component on all nodes

Install Puppet Agent on Nodes

Do not forget, to execute the **pre-installation steps** that you did on the server here as well

On CentOS 8.x

Puppet agent is installed in two simple steps

First, add the repository

```
sudo dnf install -y https://yum.puppet.com/puppet7-release-el-8.noarch.rpm
```

Then, install the agent:

```
sudo dnf install -y puppet-agent
```

On Debian/Ubuntu

There are a few steps that must be executed in order to have **Puppet** agent installed on **Debian**/**Ubuntu**

Download the repository package (for **Ubuntu 20.04**)

```
wget https://apt.puppet.com/puppet7-release-focal.deb -O repository.deb
```

Or for Debian 11 (Bullseye)

```
wget https://apt.puppet.com/puppet7-release-bullseye.deb -O repository.deb
```

Install the repository package

```
sudo dpkg -i repository.deb
```

Install the agent

```
sudo apt-get update
```

```
sudo apt-get install -y puppet-agent
```

Post Installation

We must execute few more configuration steps in order to configure security

Adjust the secure path (**secure_path**) variable – add **/opt/puppetlabs/bin**

```
sudo visudo
```

Restart the shell

```
exec $SHELL
```

Configure the agent

```
sudo puppet config set server puppet-server
```

```
sudo puppet config set certname <client-name>
```

*The <client-name> variable in our case would be either **puppet-client-1** or **puppet-client-2***

And now start and enable the agent service

```
sudo systemctl start puppet
```

```
sudo systemctl enable puppet
```

Repeat the above steps on all node machines

Register the Nodes with the Server

Now, while on the **server**, let's check do we have any waiting approvals and acknowledge them

The waiting list can be seen by

```
sudo puppetserver ca list
```

Then approve all waiting

```
sudo puppetserver ca sign --all
```

Now we can ask again for the waiting list, but include the approved also

```
sudo puppetserver ca list --all
```

Command Line Experiments

Continue on the **server**

Let's explore some of the capabilities of the **Puppet**

In order to get familiar with the list of supported resource types, execute

```
puppet describe --list
```

Let's ask for file type details

```
puppet describe file
```

The list is too long, so we can ask for a shorter version

```
puppet describe file --short
```

Let's ask for the user resource as well

```
puppet describe user --short
```

We can explore and reverse engineer in a way

In order to check how to create a user like **vagrant**, execute

```
sudo puppet resource user vagrant
```

Or a file like **/etc/os-release**

```
sudo puppet resource file /etc/os-release
```

Now that we know some of the supported types, and their structure, let's use few of them

To create a **readme.txt** file, execute

```
sudo puppet apply -e "file {'/home/vagrant/readme.txt': ensure => 'file', content =>
'This a read me file',}"
```

And check the file's content with

```
cat readme.txt
```

To create a user **demo**, execute

```
sudo puppet apply -e "user {'demo': ensure => 'present', managehome => true,}"
```

And check that it has been created with

```
tail /etc/passwd
```

Let's install a package

```
sudo puppet apply -e "package {'tmux':}"
```

And check with

```
tmux -V
```

Now, we can stop and then start a service. For example, the **cron** service

*Please note that under **CentOS** its name is **crond** and under **Debian/Ubuntu** it is just **cron***

First check that the service is running

```
systemctl status crond
```

Then stop it with

```
sudo puppet apply -e "service {'crond': ensure => stopped }"
```

And check again

```
systemctl status crond
```

Now, start it again with

```
sudo puppet apply -e "service {'crond': ensure => running }"
```

And check its status

```
systemctl status crond
```

First Manifest

Now, it is time to combine some of the manually executed tasks in a simple manifest file

Open an empty **web-centos.pp** file

```
vi web-centos.pp
```

Type the following

```
package { 'httpd': }
service { 'httpd':
  ensure => 'running',
  enable => true,
}
file { '/var/www/html/index.html':
  ensure  => 'file',
  content => '<h1>Hello Puppet World!</h1>',
}
```

Save it and exit

We can omit the comma symbol at the end of the last line in a block

Validate the syntax with

```
sudo puppet parser validate web-centos.pp
```

Copy the file to **client 1**

```
scp web-centos.pp vagrant@puppet-client-1:.
```

Go to **client 1** and apply the file with

```
sudo puppet apply web-centos.pp
```

Check the result with

```
curl http://localhost
```

Now clean up the node

Remove the package

```
sudo dnf remove httpd
```

And return on the **server** node

Part 2

We continue with the same environment from ***Part 1***

Make sure that all machines are up and running and you are on the **server**

Directories and Files

We can install the **tree** command (if not installed) and explore the folder structure with

```
sudo tree /etc/puppetlabs
```

We can see some of the folders described in the slides plus many more

Create an Environment

Environments are stored by default in **/etc/puppetlabs/code/environments/**

There is one created by default – the production environment

Let's create a **development** environment as well

Create a folder

```
sudo mkdir -p /etc/puppetlabs/code/environments/development/manifests
```

The **site.pp** file is the first file read by the nodes

So, let's create one in each environment

Create manifest for the **production** environment

```
sudo vi /etc/puppetlabs/code/environments/production/manifests/site.pp
```

With the following content

```
file {'/tmp/readme.txt':
  ensure  => present,
  mode    => "0644",
  content => "Hello from the Production Environment \n",
}
```

And another one in **development** environment

```
sudo vi /etc/puppetlabs/code/environments/development/manifests/site.pp
```

With the following content

```
file {'/tmp/readme.txt':
  ensure  => present,
  mode    => "0644",
  content => "Hello from the Development Environment \n",
}
```

Now go to one of the client nodes and let's experiment with the environments

Ask for the configuration from the production environment

```
sudo puppet agent --environment=production --test
```

And check the result with

```
cat /tmp/readme.txt
```

Now, ask for a dry run of the configuration from the development environment

```
sudo puppet agent --environment=development --test --noop
```

If executed, it would overwrite the **/tmp/readme.txt** file created earlier

Return on the server node

We can rename one of the files (for example, from ***site.pp*** *to* ***test.pp****) and return on the node and repeat the test*

We will notice that the name is not important as long as the extension is the right one

This would work because we are working with directory of manifests

We can see to which environment a node is attached

```
puppet node find puppet-client-1 | grep environment
```

Let's remove both **site.pp** files from the server

```
sudo rm /etc/puppetlabs/code/environments/development/manifests/site.pp
sudo rm /etc/puppetlabs/code/environments/production/manifests/site.pp
```

Facts Exploration

Like every other configuration management solution, **Puppet** must know details about the nodes that it manages

We can see how and what **Puppet** sees about each node by experimenting with the **Facter** tool

Execute **Facter** to see all collected attributes

```
facter
```

Check for example only the kernel

```
facter kernel
```

There are attributes with nested information:

```
facter os
```

We can address each field as well

```
facter os.family
facter os.release.full
```

We can show the legacy facts as well

```
facter --show-legacy
```

One Web Application on Two Servers

Now that we know about some of the building blocks, let's create a manifest, that will install and activate web server on both client nodes, and create a custom home page

Create the manifest in **production** environment

```
sudo vi /etc/puppetlabs/code/environments/production/manifests/site.pp
```

Enter the following

```
if $facts['os']['family'] == 'RedHat' {
  $vpackage = 'httpd'
}
```

```
else {
  $vpackage = 'apache2'
}
notify { $vpackage: }
```

Now, save, and go to **each node** and execute

```
sudo puppet agent --environment=production --test --noop
```

We can shorten the above command by skipping the environment (as we know that it is the default one) to

```
sudo puppet agent --test --noop
```

Return on the **server** and continue with the **site.pp** file

Remove the last line (notify) and enter

```
package { $vpackage: }
service { $vpackage:
  ensure => running,
  enable => true,
}
file {'/var/www/html/index.html':
  ensure  => 'file',
  content => "<h1>Hello Puppet World!</h1><br /><hr /><h5>Running on
${facts['os']['family']}</h5>",
}
```

Save and exit

Now go to **each node** and check what is the setting for how often the nodes will connect to the master

```
sudo puppet config print runinterval
```

The default value is 1800 seconds or 30 minutes

Let's change it to 30 seconds

```
sudo puppet config set runinterval 30
```

Wait a while, and connect to each node, and test if the configuration is applied

Check if there is **apache2** / **httpd** installed and running

```
systemctl status httpd
```

Check if the default index page is set as expected

```
curl http://localhost
```

Manifest to Clean

Now, that we managed to make it work, let's clean a bit

Return on the **server** and open the **site.pp** file

```
sudo vi /etc/puppetlabs/code/environments/production/manifests/site.pp
```

Change its content to match this

```
if $facts['os']['family'] == 'RedHat' {
  $vpackage = 'httpd'
}
else {
  $vpackage = 'apache2'
}
package { $vpackage:
  ensure   => 'purged',
}
```

Save and close the file

After a while (at least 30 seconds), check on the **stations**. The web server should be removed

Return on the **server** and delete the manifest

```
sudo rm /etc/puppetlabs/code/environments/production/manifests/site.pp
```

Install Additional Modules

First, let's see where **Puppet** will look for modules for the production environment

```
sudo puppet config print modulepath --section server --environment production
```

Now, let's refresh our memory and ask for the list of built-in resource types

```
puppet describe --list
```

Here, we do not have anything for managing the firewall for example

How can we deal with this? We can install module(s) from **Puppet Forge**

First, let's see if there are any modules installed already

```
puppet module list
```

No, there aren't any, at least on system level

We can check if there are any in the production environment

```
puppet module list --environment production
```

Again, nothing

Okay, let's install one (for the firewall)

Navigate here: https://forge.puppet.com/modules/puppetlabs/firewall

And explore the documentation

Once done, to install it, execute

```
puppet module install puppetlabs-firewall --version 4.0.1
```

If we rerun the commands that we used to check for installed modules

```
puppet module list
```

We will notice that even if the module is installed in the home folder of our user, it is still visible to **Puppet**

Let's prepare a simple manifest in our home folder (as we will test it locally) that will open port 80/tcp

First, check that it is not open

```
sudo firewall-cmd --list-all
```

And with **iptables**

```
sudo iptables -L
```

Nothing related to HTTP (80/tcp)

Create the following file

```
vi ~/firewall.pp
```

And enter the following code

```
class { 'firewall': }
firewall { '000 accept 80/tcp':
  action   => 'accept',
  dport    => 80,
  proto    => 'tcp',
}
```

Save and close the file

Validate it

```
sudo puppet parser validate firewall.pp
```

And apply it with

```
sudo puppet apply firewall.pp
```

Ha, an error message

So, **Puppet** cannot find the module

Let's ask for information about the module with

```
puppet describe firewall
```

It works. We can see the information

Now, let's try with **sudo** in front

```
sudo puppet describe firewall
```

Nothing. So, perhaps this is a good hint

Let's copy the installed module(s) to a system-level folder

```
sudo cp -vR ~/.puppetlabs/etc/code/modules/ /etc/puppetlabs/code
```

And try again the previous command

```
sudo puppet describe firewall
```

This time it worked

Now, retry the manifest

```
sudo puppet apply firewall.pp
```

This time it worked fine but it stopped the **firewalld** service

Never mind, let's see if the port is open

```
sudo iptables -L
```

Yes, it is

Of course, we could have used the **firewalld** module instead

Now, let's stop the two **iptables** services

```
sudo systemctl disable --now iptables
sudo systemctl disable --now ip6tables
```

And start the **firewalld** service

```
sudo systemctl enable --now firewalld
```

Manifest With Additional Modules

Now, let's install another module but this time not only on the server but also on the nodes

```
puppet module install puppetlabs/mysql
sudo cp -vR ~/.puppetlabs/etc/code/modules/ /etc/puppetlabs/code/
```

Execute the above block on all machines (we need it on the server for testing / validation purposes)

Imagine, that we need it to setup a small web application on the two nodes

It has web component and a database component which together are installed on one machine

They are available here:

- Web - https://zahariev.pro/files/app/index.php.txt
- DB - https://zahariev.pro/files/app/db.sql

Create an empty manifest

```
vi app.pp
```

And enter the following code

```
$packages = [ 'httpd', 'php', 'php-mysqlnd' ]

package { $packages: }

service { httpd:
  ensure => running,
  enable => true,
}

file { '/var/www/html/index.php':
  ensure => present,
  source => "https://zahariev.pro/files/app/index.php.txt",
}

file { '/tmp/db.sql':
  ensure => present,
  source => "https://zahariev.pro/files/app/db.sql",
}

class { '::mysql::server':
  root_password           => '12345',
  remove_default_accounts => true,
  restart                 => true,
  override_options => {
    mysqld => { bind-address => '0.0.0.0'}
  },
}

mysql::db { 'sentences':
  user        => 'root',
  password    => '12345',
  dbname      => 'sentences',
  host        => '%',
  sql         => ['/tmp/db.sql'],
  enforce_sql => true,
}
```

Save and close the file

As you can see, it is targeted towards CentOS based systems

Validate the syntax with

```
sudo puppet parser validate app.pp
```

Copy the file to **client 1**

```
scp app.pp vagrant@puppet-client-1:.
```

Go to **client 1** and apply the file with

```
sudo puppet apply app.pp
```

Check the result with

```
curl http://localhost
```

It works 😊

** If the old index.html file appear instead of the expected output from index.php, remove it with*

sudo rm /var/www/html/index.html

Part 3

Be sure to reset the environment but change the server and the two clients to be CentOS based

Bolt

Now, it is time to try **Bolt**

The content here is borrowed from https://puppet.com/docs/bolt/latest/getting_started_with_bolt.html

It has some differences, and this is on purpose (to demonstrate some additional things)

Install Bolt

Log on to the **server** machine

On CentOS 8.x

Add the repository

```
sudo dnf install https://yum.puppet.com/puppet-tools-release-el-8.noarch.rpm
```

Install the package

```
sudo dnf install puppet-bolt
```

On Debian 11

Download the repository package

```
wget https://apt.puppet.com/puppet-tools-release-bullseye.deb
```

Install the repository

```
sudo dpkg -i puppet-tools-release-bullseye.deb
```

Install the package

```
sudo apt-get update
sudo apt-get install puppet-bolt
```

On Ubuntu 20.04

Download the repository package

```
wget https://apt.puppet.com/puppet-tools-release-focal.deb
```

Install the repository

```
sudo dpkg -i puppet-tools-release-focal.deb
```

Install the package

```
sudo apt-get update
sudo apt-get install puppet-bolt
```

Run Commands

Let's start with something simple like

```
bolt command run whoami -t puppet-client-1 -u vagrant -p vagrant --no-host-key-check
```

Okay, it works

Now, what if we want to execute commands that have arguments? We can change the command to this

```
bolt command run 'uname -a' -t puppet-client-1 -u vagrant -p vagrant --no-host-key-check
```

Now, prepare a short script

```
vi test.sh
```

With the following content

```
#!/bin/bash
hostname
uname -a
```

Save and close the file

Now, execute it against the first node

```
bolt command run @test.sh -t puppet-client-1 -u vagrant -p vagrant --no-host-key-check
```

It works

To execute it against multiple nodes (first and second), we can use this command

```
bolt command run @test.sh -t puppet-client-1,puppet-client-2 -u vagrant -p vagrant --no-host-key-check
```

This one worked as well

Run Scripts

Even though the last attempt looked like we are executing a script against the nodes, this is not the case

Instead, **Bolt** is sending the commands that are inside it and not the script as a unit

Should we want to execute it as a script, then we must use this command

```
bolt script run ./test.sh -t puppet-client-1,puppet-client-2 -u vagrant -p vagrant --no-host-key-check
```

Now, let's change the script to match this

```
#!/bin/bash
hostname
uname -a
if [ $# -ne 0 ]; then
  echo "Argument: $1"
fi
```

Save and close the file

So now, our script accepts arguments, then we can pass them like this

```
bolt script run ./test.sh -t puppet-client-1,puppet-client-2 -u vagrant -p vagrant --no-host-key-check myargument
```

Bolt Project

Create a folder to host our project and navigate to it

```
mkdir boltdemo; cd boltdemo
```

Initialize the project

```
bolt project init boltdemo
```

Let's imagine that we want to install **Apache**

For this, we will need a modules folder and two subfolders

```
mkdir -p modules/apache/{plans,files}
```

Check the project hierarchy so far

```
tree .
```

Now, open the inventory file for editing

```
vi inventory.yaml
```

Enter the following

```
groups:
- name: nodes
  targets:
    - uri: puppet-client-1
      name: client1
    - uri: puppet-client-2
      name: client2
  config:
    transport: ssh
    ssh:
      user: vagrant
      password: vagrant
      host-key-check: false
```

Save and close the file

Now, we can use the inventory and execute a command against one of the nodes

```
bolt command run hostname -t client1
```

It works. Now, execute the same but against the **nodes** group

```
bolt command run hostname -t nodes
```

If we had more groups, we could target a command against **all** of them with

```
bolt command run hostname -t all
```

Run Tasks

We can run tasks as well

The list of available tasks can be seen with

```
bolt task show
```

And the documentation for a task with

```
bolt task show package
```

So, to install the **tmux** package, we should execute

```
bolt task run package --targets nodes action=install name=tmux
```

Hm, it won't work

We have to add one additional flag

```
bolt task run package --targets nodes action=install name=tmux --run-as root
```

This time it worked

Bolt Plans

Prepare the folders

```
mkdir -p modules/apache/plans
```

Create a new file

```
vi modules/apache/plans/install.yaml
```

With the following content

```
parameters:
  targets:
    type: TargetSpec
  pkg:
    type: String

steps:
  - name: install_apache
    task: package
    targets: $targets
    parameters:
      action: install
      name: $pkg
```

Save and close the file

So far, our folders should look like

```
tree .
```

Now, let's execute the plan

```
bolt plan run apache::install -t nodes pkg=httpd --run-as root
```

Okay, let's extend the plan with a second task that will execute a script

First, prepare the folders

```
mkdir -p modules/apache/files
```

Create new script file

```
vi modules/apache/files/apachestart.sh
```

Enter the following content

```
#!/bin/bash


systemctl is-active httpd &> /dev/null


if [ $? -eq 0 ]
then
  echo "Apache is running"
else
  echo "Starting Apache"
  systemctl start httpd
fi
```

Save and close the file

Open the plan file

```
vi modules/apache/plans/install.yaml
```

And add this block at the end

```
  - name: start_apache
    script: apache/apachestart.sh
    targets: $targets
```

Save and close the file

Check the folder structure so far

```
tree .
```

Let's execute it

```
bolt plan run apache::install -t nodes pkg=httpd --run-as root
```

Now, let's prepare an **index.html** file to be uploaded to the machines

Create new file

```
echo '<h1>Bolt is Fun!</h1>' > modules/apache/files/index.html
```

Now, change the plan file

```
vi modules/apache/plans/install.yaml
```

To match this

```
parameters:
  targets:
    type: TargetSpec
  pkg:
    type: String
  src:
    type: String

steps:
  - name: install_apache
    task: package
    targets: $targets
    parameters:
      action: install
      name: $pkg
    description: "Install the $pkg package"
  - name: start_apache
    script: apache/apachestart.sh
    targets: $targets
    description: "Start the service with script"
  - name: upload_homepage
    upload: $src
    destination: /var/www/html/index.html
    targets: $targets
    description: "Upload custom index.html file"
```

Save and close the file

Execute it with the following command

```
bolt plan run apache::install -t nodes pkg=httpd src=apache/index.html --run-as root
```

Depending on the boxes in use, you may need to adjust the firewall of the client machines or go to each one of them in order to see the custom index page

Puppet and Vagrant

Now, let's improvise a bit and create a simple two machine environment with the help of **Puppet** and **Vagrant**

We will use the files and approach from **Part 2 > Manifest with additional modules**

But will change the **index.php** file to look for the database on another machine

And will add a second machine to host the database

All files for this part are in the accompanying archive (in folder **puppet-vagrant**)