# Practice M6: Puppet, Chef, and Salt

During this practice we will assume that we are working in Linux environment. It could be a physical machine or a virtual one. The distribution of choice is not that important, but it will be better to stick to some of the well supported distributions.

Most of the steps can be executed in Windows and/or macOS environment as well either directly or in a VM.

The lab infrastructure will vary during different parts of the module. For each part there is one Vagrantfile file.

## Part 1: Puppet

#### Install Puppet Server (on CentOS)

At first, we should take care for a few preparation steps:

* Install and activate NTP server

**sudo yum install -y ntp**

**sudo systemctl enable ntpd**

**sudo systemctl start ntpd**

* 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**

The above set of tasks must be repeated on the client nodes as well.

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

* Add the repository

**sudo yum install -y https://yum.puppet.com/puppet6/puppet6-release-el-7.noarch.rpm**

* Install the Puppet server

**sudo yum install -y puppetserver**

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.sulab**

**sudo puppet config set server puppet-server**

**sudo puppet config set caserver puppet-server**

**sudo puppet config set reportserver puppet-server**

* Generate certificates:

**sudo puppetserver ca setup**

* Start and enable the server

**sudo systemctl start puppetserver**

**sudo systemctl enable puppetserver**

We can adjust the configuration. For example, by default, the Puppet server is configured to use 2GB of RAM:

* Open the file **/etc/sysconfig/puppetserver**
* Change default values to 1GB:

**JAVA\_ARGS="-Xms1g -Xmx1g"**

* Or to 512MB:

**JAVA\_ARGS="-Xms512m -Xmx512m"**

* And then restart the puppet server

One more post-installation step is necessary:

* We must open a specific port in the firewall if running:

**sudo firewall-cmd --add-port=8140/tcp --permanent**

**sudo firewall-cmd --reload**

#### Install Puppet Server (on Ubuntu) \*

At first, we should take care for just one preparation step:

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

**sudo apt-get install -y ntp**

The above step must be repeated on all the Ubuntu client nodes as well.

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

* Download the repository package

**wget https://apt.puppetlabs.com/puppet6-release-xenial.deb**

* Install the repository package

**sudo dpkg -i puppet6-release-xenial.deb**

* Install the agent

**sudo apt-get update**

**sudo apt-get install -y puppetserver**

The rest is the same as with the Puppet server running on CentOS.

#### Install Puppet Agent on Nodes (on CentOS)

Repeat the pre-installation steps you did on the server.

Puppet agent is installed in two simple steps:

* First, add the repository:

**sudo yum install -y https://yum.puppet.com/puppet6/puppet6-release-el-7.noarch.rpm**

* Then, install the agent:

**sudo yum install -y puppet-agent**

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 puppet-client-1**

* And now start and enable the agent service

**sudo systemctl start puppet**

**sudo systemctl enable puppet**

#### Install Puppet Agent on Nodes (on Ubuntu)

There are few steps that must be executed in order to have Puppet agent installed on Ubuntu:

* Download the repository package

**wget https://apt.puppetlabs.com/puppet6-release-xenial.deb**

* Install the repository package

**sudo dpkg -i puppetlabs-release-xenial.deb**

* Install the agent

**sudo apt-get update**

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

Close and reopen the shell. Then execute the following set of tasks:

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

**sudo visudo**

* Configure the agent:

**sudo puppet config set server puppet-server**

**sudo puppet config set certname puppet-client-2**

* And now start and enable the agent service

**sudo systemctl start puppet**

**sudo systemctl enable puppet**

#### 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**

#### Command Line Experiments

Let’s explore some capabilities of the Puppet:

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

**puppet describe --all**

* 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**

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',}"**

* To create a user demo, execute:

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

* Let’s install package:

**sudo puppet apply -e "package {'htop':}"**

* Now, we can stop and then start a service

**sudo puppet apply -e "service {'ntpd': ensure => stopped }"**

**sudo puppet apply -e "service {'ntpd': ensure => running }"**

#### Syntax Highlighting

Now that we know how to issue simple commands with Puppet, it is time to move on and start creating files. Before diving into this, it would be nice to install a syntax highlighting for vim (if this is our editor of choice). And what would be more beautiful than installing it with the help of Puppet:

* Install the module:

**sudo puppet module install theurbanpenguin-puppet\_vim --version 0.1.3**

* Now apply the module

**sudo puppet apply -e " include puppet\_vim "**

#### 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 World!</H1>',**

**}**

* Save it and exit
* Validate the syntax with:

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

* Copy the file to client node 1:

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

* Go to client node 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 yum remove httpd**

#### 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**

* Put a file **site.pp** (the first file read by nodes) in the production environment:

**file {'/tmp/readme.txt':**

**ensure => present,**

**mode => "0644",**

**content => "Hello from the Production Environment \n",**

**}**

* And a similar file in the development environment:

**file {'/tmp/readme.txt':**

**ensure => present,**

**mode => "0644",**

**content => "Hello from the Development Environment \n",**

**}**

Now go to one of the nodes and let’s experiment with the environments:

* Ask for the configuration from the production environment:

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

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

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

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

#### 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**

#### 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 **/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**

* Return on the server and continue with the site.pp file. Remove the last line and enter:

**package { $vpackage: }**

**service { $vpackage:**

**ensure => running,**

**enable => true,**

**}**

**file {'/var/www/html/index.html':**

**ensure => 'file',**

**content => "<h1>Hello 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**

* Set it to 30 seconds:

**sudo puppet config set runinterval 30**

Wait a while, and connect to each node, and execute:

* 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**

## Part 2: Chef

#### Install Chef Server (on CentOS)

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

* Install and activate NTP server

**sudo yum install -y ntp**

**sudo systemctl enable ntpd**

**sudo systemctl start ntpd**

* 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**

The above set of tasks must be repeated on the client nodes as well.

In order to install a stand-alone Chef server on CentOS, we must follow these steps:

* Download the package:

**wget -P /tmp https://packages.chef.io/files/stable/chef-server/12.18.14/el/7/chef-server-core-12.18.14-1.el7.x86\_64.rpm**

* Install the package:

**sudo rpm -Uvh /tmp/chef-server-core-12.18.14-1.el7.x86\_64.rpm**

* Configure the services:

**sudo chef-server-ctl reconfigure**

* Create administrator user:

**sudo chef-server-ctl user-create chefadmin Chef Admin chefadmin@sulab.local 'Password1' --filename /home/vagrant/chefadmin.pem**

* Create an organization:

**sudo chef-server-ctl org-create demo-org 'Demo Org.' --association\_user chefadmin --filename /home/vagrant/demoorg-validator.pem**

One more post-installation step is necessary:

* We must open a specific port in the firewall if running:

**sudo firewall-cmd --add-port=80/tcp --permanent**

**sudo firewall-cmd --add-port=443/tcp --permanent**

**sudo firewall-cmd --reload**

We can install few additional components. Amongst them there are the web management console and the reporting functionality. First, let’s install the web ui:

* To install the web management console, execute:

**sudo chef-server-ctl install chef-manage**

* Then reconfigure the server with:

**sudo chef-server-ctl reconfigure**

* And then the configure the management tools with:

**sudo chef-manage-ctl reconfigure --accept-license**

* Now, we can open a browser tab and navigate to **https://192.168.99.100**

We could install the reporting functions as well:

* Install the component with:

**sudo chef-server-ctl install opscode-reporting**

* Then reconfigure the server with:

**sudo chef-server-ctl reconfigure**

* And then the configure the reporting component with:

**sudo opscode-reporting-ctl reconfigure --accept-license**

* Test the installation

**sudo opscode-reporting-ctl test**

* Now, we can open a browser tab and navigate to [**https://192.168.99.100**](https://192.168.99.100) and check the **Reports** tab

#### Install Chef Server (on Ubuntu) \*

As first step, we should take care for just one task:

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

**sudo apt-get install -y ntp**

The above step must be repeated on all the Ubuntu client nodes as well.

In order to install a stand-alone Chef server on Ubuntu 16.04, we must follow these steps:

* Download the package:

**wget -P /tmp https://packages.chef.io/files/stable/chef-server/12.18.14/ubuntu/16.04/chef-server-core\_12.18.14-1\_amd64.deb**

* Install the package:

**sudo dpkg -i /tmp/chef-server-core\_12.18.14-1\_amd64.deb**

* Configure the services:

**sudo chef-server-ctl reconfigure**

* Create administrator user:

**sudo chef-server-ctl user-create chefadmin Chef Admin chefadmin@sulab.local 'Password1' --filename /home/vagrant/chefadmin.pem**

* Create an organization:

**sudo chef-server-ctl org-create demo-org 'Demo Org.' --association\_user chefadmin --filename /home/vagrant/demoorg-validator.pem**

One more post-installation step is necessary – ports 80/tcp and 443/tcp must be open.

#### Install Chef Workstation (Chef DK on CentOS)

First step is the installation itself:

* Download the package:

**wget -P /tmp https://packages.chef.io/files/stable/chefdk/3.6.57/el/7/chefdk-3.6.57-1.el7.x86\_64.rpm**

* Install the package:

**sudo rpm -Uvh /tmp/chefdk-3.6.57-1.el7.x86\_64.rpm**

Most likely we will need a git client installed, so let’s tackle it:

* Install the git client with:

**sudo yum install -y git**

Now, we must further configure the workstation:

* Check the ruby environment:

**which ruby**

* Switch to the Ruby version provided by Chef:

**echo 'eval "$(chef shell-init bash)"' >> ~/.bash\_profile**

* Modify the PATH variable:

**echo 'export PATH="/opt/chefdk/embedded/bin:$PATH"' >> ~/.bash\_profile && source ~/.bash\_profile**

* Check that ruby is setup correctly

**which ruby**

Next step is to create the local working environment. There are multiple ways to achieve this, but if we have installed the web console, then we can do the following:

* Login to the console and go to **Administration**, then select the organization (**demo-org**), click on **Starter Kit**, and then on **Download Starter Kit**
* Now move the archive to the workstation machine:

**scp ~/Downloads/chef-starter.zip vagrant@192.168.99.105:.**

* Then unzip the file, but first install the unzip tool:

**sudo yum install -y unzip**

* And then extract the file contents

**unzip chef-starter.zip**

* Then go to the repository directory

**cd chef-repo**

* Now get the certificates from the server:

**knife ssl fetch**

* List all known nodes

**knife client list**

* Optionally add the workstation to the server:

**knife bootstrap 192.168.99.105 -N workstation -x vagrant -P vagrant --sudo**

#### Install Chef Workstation (Chef DK on Ubuntu) \*

First step is the installation itself:

* Download the package:

**wget -P /tmp https://packages.chef.io/files/stable/chefdk/3.6.57/ubuntu/16.04/chefdk\_3.6.57-1\_amd64.deb**

* Install the package:

**sudo dpkg -i /tmp/chefdk\_3.6.57-1\_amd64.deb**

The rest of the steps are the same as with the CentOS installation.

#### Install Chef Nodes

Installation process on the nodes can be done in multiple ways. Two viable options is the bootstrap method and via URL. We will install each node with different method.

To install and setup the node based on CentOS (client 1), we must do:

* Open ssh session to the node
* Execute:

**curl -L https://omnitruck.chef.io/install.sh | sudo bash**

* In order to finish the setup, we must download the certificate and the configuration, and then use the knife tool. It would be better and easier to switch to the workstation and execute:

**knife bootstrap 192.168.99.101 -N client-1 -x vagrant -P vagrant --sudo**

To use the bootstrap method, go to the workstation and:

* Execute the following:

**knife bootstrap 192.168.99.102 -N client-2 -x vagrant -P vagrant --sudo**

* Check if the node is recognized by the server:

**knife client show client-2**

* Get the list of all nodes:

**knife client list**

#### Check the Starter Recipe/Cookbook

Let’s make our first steps in the kitchen:

* Log on to the workstation and go to the **cookbooks** folder in the repository folder:

**cd ~/chef-repo/cookbooks**

* Execute the tree command if available
* It appears that there is a starter cookbook with one recipe in it. Let’s explore it
* Now, let’s run it locally:

**chef-client --local-mode --override-runlist starter**

* In order other nodes to be able to run this cookbook, we must upload it to the server:

**knife cookbook upload starter**

* Now, we can ask for the list of available cookbooks:

**knife cookbook list**

* Let’s assign the cookbook to client #1

**knife node run\_list add client-1 "recipe[starter]"**

* Now, go to client #1 and execute:

**sudo chef-client**

* You should see the message

#### First Own Cookbook and Recipe(s)

Let’s create one simple cookbook, that will put two text files on the target station. Return on the workstation and:

* Go to the **cookbooks** folder of the repository and execute:

**chef generate cookbook demo\_cookbook**

* Explore the newly created set of folders and files
* Open the **demo\_cookbook/recipes/default.rd** for editing:

**vi demo\_cookbook/recipes/default.rb**

* And type in the following:

**file "#{ENV['HOME']}/readme.txt" do**

**content 'Hello from Chef!'**

**end**

**file '/tmp/readme.txt' do**

**content 'Chef was here as well :)'**

**end**

* Save the file and exit
* Execute it locally:

**chef-client --local-mode --override-runlist demo\_cookbook**

* Check the contents of both files:

**cat ~/readme.txt**

**cat /tmp/readme.txt**

* Remove one of them and re-execute the cookbook

**rm ~/readme.txt**

**chef-client --local-mode --override-runlist demo\_cookbook**

* Just the missing file gets re-created

Now, let’s add one more recipe in the cookbook. This time we will create two users:

* Create new file **users.rb**

**vi demo\_cookbook/recipes/users.rb**

* We need one preparation step – generate encrypted password:

**openssl passwd -1 "Password1"**

* And enter the following:

**user 'demo-user-1' do**

**comment 'Demo user #1'**

**manage\_home true**

**shell '/bin/bash'**

**password '$1$1neZnO0Q$MT3yOQE2wmkJ8CtPNxA1K.'**

**end**

**user 'another user' do**

**username 'demo-user-2'**

**comment 'Demo user #2'**

**manage\_home true**

**shell '/bin/bash'**

**password '$1$1neZnO0Q$MT3yOQE2wmkJ8CtPNxA1K.'**

**end**

* Save and exit
* Now execute locally together with the previous recipe:

**sudo chef-client --local-mode --override-runlist demo\_cookbook,demo\_cookbook::users**

* Check that users are there
* If we want, we can upload the cookbook

**knife cookbook upload demo\_cookbook**

* And eventually assign it (in fact the two recipes) as a run list to the client #2

**knife node run\_list add client-2 "recipe[demo\_cookbook], recipe[demo\_cookbook::users]"**

* Now, go to client #2 and execute:

**sudo chef-client**

* We should see both the files and the users

#### One Web Application on Two Servers

It is time to create another cookbook. It will install and run an Apache web server, and create a custom index file on both client nodes:

* Being on the workstation, and in the cookbooks repository folder, create an empty cookbook:

**chef generate cookbook apache**

* Open the default recipe for editing:

**vi apache/recipes/default.rb**

* And enter the following set of instructions:

**if node['platform\_family'] == 'debian'**

**vpackage = 'apache2'**

**else**

**vpackage = 'httpd'**

**end**

**package 'Install Apache web server' do**

**package\_name "#{vpackage}"**

**end**

**service 'Start and Enable Apache web server' do**

**service\_name "#{vpackage}"**

**action [ :enable, :start ]**

**end**

**file 'Create custom index.html file' do**

**path '/var/www/html/index.html'**

**content "<h1>Hello World!</h1><br /><hr /><h5>Running on %{p}</h5>" % {p: node['platform\_family']}**

**end**

* We can do a dry run in order to test it locally

**sudo chef-client --local-mode --override-runlist apache --why-run**

* Then we can upload the cookbook

**knife cookbook upload apache**

Now we can go to the console and attach our cookbook to the two client nodes and then go to each node and execute the chef client. Check the results.

Of course, there is an option to start the client software as daemon. The easiest way, just for testing purposes, for the current lab and session is to execute:

**sudo chef-client -i 60 -d**

Where **60** is the interval in seconds for the communication with the server, and **-d** is instruction to demonize the service.

## Part 3: Salt

#### Install Salt Server

There are multiple ways to install Salt server (master). Two viable options are to use the target system package manager, or to use the bootstrap script, provided by Salt. For this practice, we will choose the second option:

* Download the bootstrap script

**wget -O bootstrap-salt.sh https://bootstrap.saltstack.com**

* Install the latest stable version – just the master (-M), without minion (-N) part, do not start the daemons (-X):

**sudo sh bootstrap-salt.sh -M -N -X**

* Now, we must open the firewall ports:

**sudo firewall-cmd --permanent --add-port=4505-4506/tcp**

**sudo firewall-cmd --reload**

* It is time to enable, start the Salt master service, and check if everything is okay

**sudo systemctl enable salt-master**

**sudo systemctl start salt-master**

**systemctl status salt-master**

#### Install Salt Nodes (Minions)

As with the server part, here also exist multiple options, but again, we will stick to the bootstrap way. On both client stations execute:

* Download the bootstrap script

**wget -O bootstrap-salt.sh https://bootstrap.saltstack.com**

* Install the latest stable version of just the minion part and run it as a daemon:

**sudo sh bootstrap-salt.sh**

* Now, we must point the minions to the master. Open the **/etc/salt/minion** file, uncomment the master section, and enter the salt server name:

**sudo vi /etc/salt/minion**

* And restart the service:

**sudo systemctl restart salt-minion**

#### Accept Salt Minions

Now return to the master, check if there are waiting minions, and accept them eventually:

* In order to examine the situation with the waiting minions, you must execute:

**sudo salt-key -L**

* Then you can examine a key:

**sudo salt-key -f salt-client-1**

* And accept all unauthorized keys:

**sudo salt-key -A**

#### First Commands

We should be now ready to start our journey with Salt. First, let’s experiment with few command line constructions:

* Test the communication with all minions:

**sudo salt '\*' test.ping**

* We can ask for different information this time, let’s ask for the usage of disks and then for their blkids:

**sudo salt '\*' disk.usage**

**sudo salt '\*' disk.blkid**

* We can narrow the target systems, for example ask only the client #1:

**sudo salt salt-client-1 disk.blkid**

* In order to check what functions are supported by particular minion, we can execute:

**sudo salt salt-client-1 sys.doc**

* We can do few more executions:

**sudo salt '\*' cmd.run 'cat /etc/os-release'**

**sudo salt '\*' network.ip\_addrs**

**sudo salt '\*' pkg.install htop**

#### Targeting Nodes with Grains

We can filter the target set of minions by utilizing the detailed information available for each one. This set of data in Salt is called Grains and it is equivalent to the Facts used in other Configuration Management solutions. Let’s explore a little bit in this direction:

* Check what information is available for both minions:

**sudo salt '\*' grains.items**

* Now narrow the ping command for example, and apply it only on the CentOS hosts:

**sudo salt -G 'os:CentOS' test.ping**

#### Simple State File

Before we continue further, we must configure and initialize the so called Salt State Tree. For this purpose we must:

* Open the master configuration file:

**sudo vi /etc/salt/master**

* Uncomment the lines:

**file\_roots:**

**base:**

**- /srv/salt**

* Save and exit
* Restart the Salt master:

**sudo systemctl restart salt-master**

Now, we can go to the **/srv** folder and prepare the structure and files:

* Create the **/srv/salt** folder
* Create file **top.sls** in **/srv/salt** with the following content:

**base:**

**'\*':**

**- demo**

* Now, create the **/srv/salt/demo.sls** file with the following content:

**create.user:**

**user:**

**- name: demo**

**- fullname: Demo User**

**- createhome: True**

**- present**

**install.screen:**

**pkg:**

**- name: screen**

**- installed**

* Let’s save the file and apply the state to all minions:

**sudo salt '\*' state.apply**

* If we ask again for state application nothing, must change

Let’s create a new state file:

* Create **/srv/salt/apache.sls** with:

**install.apache:**

**pkg:**

**- name: httpd**

**- installed**

**run.apache:**

**service.running:**

**- name: httpd**

**- require:**

**- pkg: httpd**

* Change the **/srv/salt/top.sls** file to refer to the new state

**base:**

**'\*':**

**- apache**

* Apply the new state only to client #1:

**sudo salt salt-client-1 state.apply**

#### One Web Application on Two Servers

It is time to extend our **apache.sls** file to cover Ubuntu as well:

* Open the file for edit and enter:

**{% if grains['os\_family'] == 'RedHat' %}**

**{% set vpackage = 'httpd' %}**

**{% else %}**

**{% set vpackage = 'apache2' %}**

**{% endif %}**

**install.webserver:**

**pkg:**

**- name: {{ vpackage }}**

**- installed**

**run.webserver:**

**service.running:**

**- name: {{ vpackage }}**

**- require:**

**- pkg: {{ vpackage }}**

**set.index:**

**file.managed:**

**- name: /var/www/html/index.html**

**- contents: '<h1>Hello World!</h1><br /><hr /><h5>Running on {{ grains['os\_family'] }}</h5>'**

**- require:**

**- pkg: {{ vpackage }}**

* Save and execute a dry run:

**sudo salt '\*' state.apply test=True**

* If all seems to be okay, then apply the state to both clients:

**sudo salt '\*' state.apply**