For hands-on exercises, it's important to start with simple tasks and gradually increase the complexity as learners become more comfortable with Puppet Language Constructs. Here’s a series of hands-on exercises designed to reinforce the concepts:

**Exercise 1: Writing a Simple Manifest**

* **Objective:** Create a basic Puppet manifest to manage a file.
* **Instructions:**
  1. Create a file named example.pp.
  2. Write a manifest that ensures a file named /tmp/testfile.txt exists with the content "Hello, Puppet!".
  3. Apply the manifest using puppet apply example.pp.
  4. Verify that the file was created and contains the correct content.

**Manifest: example.pp**

file { '/tmp/testfile.txt':

ensure => 'file',

content => 'Hello, Puppet!',

}

**Exercise 2: Managing a Service**

* **Objective:** Manage a system service using Puppet.
* **Instructions:**
  1. Write a manifest that ensures the ntp service is installed, enabled, and running.
  2. Apply the manifest and check the status of the ntp service.
  3. Stop the ntp service manually and reapply the manifest to see how Puppet enforces the desired state.

**Manifest: ntp\_service.pp**

package { 'ntp':

ensure => installed,

}

service { 'ntp':

ensure => running,

enable => true,

require => Package['ntp'],

}

**Exercise 3: Using Variables**

* **Objective:** Learn how to use variables within a manifest.
* **Instructions:**
  1. Create a new manifest where you define a variable for the file path and another for the content.
  2. Use these variables to create or manage a file.
  3. Experiment with changing the variable values and reapply the manifest to see the changes.

variables.pp

$filepath = '/tmp/variable\_testfile.txt'

$filecontent = 'This is a test file managed by Puppet using variables.'

file { $filepath:

ensure => 'file',

content => $filecontent,

}

**Exercise 4: Defining and Using Classes**

* **Objective:** Create and apply a class in Puppet.
* **Instructions:**
  1. Define a class named myapp that manages a directory /opt/myapp and a configuration file within it.
  2. Create a manifest that declares this class and applies it to the node.
  3. Verify that the directory and file are correctly managed by Puppet.

myapp.pp

class myapp {

file { '/opt/myapp':

ensure => directory,

}

file { '/opt/myapp/config.txt':

ensure => file,

content => 'This is the configuration for myapp.',

require => File['/opt/myapp'],

}

}

include myapp

**Exercise 5: Creating a Puppet Module**

* **Objective:** Organize Puppet code into a module.
* **Instructions:**
  1. Create a new module named mymodule.
     + cd /etc/puppetlabs/code/modules
     + pdk new module user\_name-module\_name
  2. Inside this module, create a class that installs a package (e.g., httpd or nginx), starts the service, and manages its configuration file.
  3. **Include this module** in your **main manifest** and apply it.
  4. Verify that the package is installed, the service is running, and the configuration file is managed.

**mymodule/**

├── manifests/

│ └── init.pp

└── metadata.json

**Manifest: mymodule/manifests/init.pp**

class mymodule {

package { 'httpd':

ensure => installed,

}

service { 'httpd':

ensure => running,

enable => true,

require => Package['httpd'],

}

file { '/etc/httpd/conf/httpd.conf':

ensure => file,

content => 'This is a basic configuration for httpd.',

require => Package['httpd'],

}

}

include mymodule

**Exercise 6: Using Chaining Arrows**

* **Objective:** Understand and use chaining arrows for resource ordering.
* **Instructions:**
  1. Write a manifest that creates a file and ensures a service is restarted only after the file is created or modified.
  2. Use the -> operator to enforce this order.
  3. Test the manifest by changing the file content and reapplying the manifest to see the service restart.

**Manifest: chaining\_arrows.pp**

**file { '/tmp/chain\_testfile.txt':**

**ensure => 'file',**

**content => 'This is a test file.',**

**}**

**service { 'httpd':**

**ensure => running,**

**enable => true,**

**subscribe => File['/tmp/chain\_testfile.txt'],**

**}**

**Exercise 7: Managing Environments**

* **Objective:** Work with multiple Puppet environments.
* **Instructions:**
  1. Create two environments: development and production.
  2. In each environment, create a manifest that manages the same resource but with different configurations (e.g., different content for the same file).
  3. Apply the manifest for each environment and verify that the correct configuration is applied based on the environment.

Manifest for development: development.pp

**file { '/tmp/environment\_testfile.txt':**

**ensure => 'file',**

**content => 'This is the development environment.',**

**}**

Manifest for production: production.pp

**file { '/tmp/environment\_testfile.txt':**

**ensure => 'file',**

**content => 'This is the production environment.',**

**}**

**Exercise 8: Building a Full Catalog**

* **Objective:** Compile a catalog with multiple resources, classes, and modules.
* **Instructions:**
  1. Create a manifest that includes multiple classes and resources.
  2. Ensure that the resources are correctly ordered using chaining arrows where necessary.
  3. Apply the manifest and inspect the generated catalog using puppet catalog commands to understand how Puppet compiles and applies the configurations.

Manifest: full\_catalog.pp

class webserver {

package { 'httpd':

ensure => installed,

}

service { 'httpd':

ensure => running,

enable => true,

require => Package['httpd'],

}

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

ensure => file,

content => 'Welcome to the web server!',

require => Package['httpd'],

}

}

class firewall {

exec { 'open\_http\_port':

command => '/usr/sbin/iptables -A INPUT -p tcp --dport 80 -j ACCEPT',

path => '/bin:/usr/bin:/sbin:/usr/sbin',

}

}

include webserver

include firewall

**Exercise 9: Troubleshooting and Debugging**

* **Objective:** Practice troubleshooting Puppet manifests.
* **Instructions:**
  1. Introduce intentional errors in a manifest, such as syntax errors or resource conflicts.
  2. Use Puppet’s debugging tools (puppet apply --debug, puppet parser validate) to identify and resolve these issues.
  3. Discuss common errors and their solutions.

Manifest with Errors: error\_manifest.pp

# Intentional syntax error: missing comma after 'installed'

package { 'httpd'

ensure => installed

}

# Resource conflict: file resource is declared twice with different attributes

file { '/tmp/conflict\_testfile.txt':

ensure => 'file',

content => 'First declaration',

}

file { '/tmp/conflict\_testfile.txt':

ensure => 'file',

content => 'Second declaration',

}

**Exercise 10: Puppet Best Practices**

* **Objective:** Apply best practices in writing Puppet manifests.
* **Instructions:**
  1. Review and refactor one of your previous manifests to follow best practices (e.g., using proper naming conventions, organizing code into modules).
  2. Share your refactored code with peers for review and feedback.

Refactored Manifest Example: best\_practices.pp

class webserver {

package { 'httpd':

ensure => installed,

}

service { 'httpd':

ensure => running,

enable => true,

require => Package['httpd'],

}

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

ensure => file,

content => 'Welcome to the web server!',

require => Package['httpd'],

}

}

class ntp\_setup {

package { 'ntp':

ensure => installed,

}

service { 'ntp':

ensure => running,

enable => true,

require => Package['ntp'],

}

}

include webserver

include ntp\_setup

Day 3

### 1. ****Using Classes and Inheritance in Puppet****

Puppet classes are reusable blocks of code that can be included or inherited by other classes or nodes. Inheritance allows you to create a base class and extend or override its behavior in child classes.

#### Example: Defining and Using Classes

# base\_class.pp

class base\_class {

$package\_name = 'nginx'

$service\_name = 'nginx'

package { $package\_name:

ensure => installed,

}

service { $service\_name:

ensure => running,

enable => true,

require => Package[$package\_name],

}

}

# webserver.pp

class webserver inherits base\_class {

$document\_root = '/var/www/html'

file { $document\_root:

ensure => directory,

owner => 'www-data',

group => 'www-data',

mode => '0755',

}

}

# Applying the class

include webserver

**Base.pp**

# modules/profile/manifests/base.pp

class profile::base {

$package\_name = 'ntp'

$service\_name = 'ntpd'

$motd\_content = "Managed by Puppet\n"

package { $package\_name:

ensure => installed,

}

service { $service\_name:

ensure => running,

enable => true,

}

file { '/etc/motd':

ensure => file,

content => $motd\_content,

}

}

**# modules/profile/manifests/webserver.pp**

class profile::webserver **inherits** profile::base {

# Override variables

$package\_name = 'httpd'

$service\_name = 'httpd'

$motd\_content = "Welcome to the Web Server\n"

# Use the overridden variables

package { $package\_name:

ensure => installed,

}

service { $service\_name:

ensure => running,

enable => true,

}

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

ensure => file,

content => "<h1>Welcome to the Web Server</h1>\n",

}

}

Site.pp

# Example in site.pp or another manifest

**node 'webserver.example.com' {**

**include profile::webserver**

**}**

cd /etc/puppetlabs/code/environments/production/

sudo nano hiera.yaml

================

version: 5

defaults:

datadir: data

data\_hash: yaml\_data

hierarchy:

- name: "Per-Node Data"

path: "nodes/%{trusted.certname}.yaml"

- name: "Common Data"

path: "common.yaml"

mkdir -p /etc/puppetlabs/code/environments/production/data/nodes

common.yaml

# common.yaml

ntp::servers:

- time1.example.com

- time2.example.com

nginx::server\_name: 'www.example.com'

# modules/profile/manifests/ntp.pp

class profile::ntp {

$ntp\_servers = lookup('ntp::servers', Array[String], 'first', ['time1.default.com'])

package { 'ntp':

ensure => installed,

}

file { '/etc/ntp.conf':

ensure => file,

content => template('profile/ntp.conf.erb'),

require => Package['ntp'],

}

service { 'ntpd':

ensure => running,

enable => true,

require => File['/etc/ntp.conf'],

}

}

# templates/ntp.conf.erb

server <%= scope.lookupvar('ntp::servers').join("\nserver ") %>