Lab 3. Managing Applications

In this lab, we will cover the following recipes:

- Using public modules
- Managing Apache servers
- Creating Apache virtual hosts
- Creating NGINX virtual hosts
- Managing MariaDB
- Creating databases and users

Introduction

Without applications, a server is just a very expensive space heater. In this lab, I'll present some recipes to manage some specific software with Puppet: MariaDB, Apache, NGINX, and Ruby. I hope the recipes will be useful to you in themselves. However, the patterns and techniques they use are applicable to almost any software, so you can adapt them to your own purposes without much difficulty. One thing that these applications have in common is: they are common. Most Puppet installations will have to deal with a web server, either Apache or NGINX. Most, if not all, will have databases and some of those will have MariaDB. When everyone has to deal with a problem, community solutions are generally better tested and more thorough than homegrown solutions. We'll use modules from the Puppet Forge in this lab to manage these applications.

When you are writing your own `Apache` or `NGINX` modules from scratch, you'll have to pay attention to the nuances of the distributions you support. Some distributions call the Apache package `httpd`, while others use `apache2`; the same can be said for MySQL. In addition, Debian-based distributions use an enabled folder method to enable custom sites in Apache, which are virtual sites, whereas RPM-based distributions do not (for more information on virtual sites, visit http://httpd.apache.org/docs/2.4/vhosts/).

Using public modules

When you write a Puppet module to manage some software or service, you don't have to start from scratch. Community-contributed modules are available at the Puppet Forge site for many popular applications. Sometimes, a community module will be exactly what you need, and you can download and start using it straight away. In most cases, you will need to make some modifications to suit your particular needs and environment.

Like all community efforts, there are some excellent and some less-than-excellent modules on the Forge. You should read the `README` section of the module and decide whether the module is going to work in your installation. At the very least, ensure that your distribution is supported. puppetlabs has introduced a set of modules that are supported, that is, if you are an enterprise customer, they will support your use of the module in your installation. Additionally, most Forge modules deal with multiple operating systems, distributions, and a great number of use cases. In many cases, not using a Forge module is like reinventing the wheel. One caveat, though, is that Forge modules may be more complex than your local modules. You should read the code and get a sense of what the module is doing. Knowing how the module works will help you debug it later.

How to do it...

In this example, we'll use the `puppet module` command to find and install the useful `stdlib` module, which contains many utility functions to help you develop Puppet code. It is one of the aforementioned supported modules by puppetlabs. I'll download the module into my user's home directory and manually install it in the Git repository:

1. Run the following command:

```
t@mylaptop $ puppet module search puppetlabs-stdlib
Notice: Searching https://forgeapi.puppet.com ...
NAME DESCRIPTION AUTHOR KEYWORDS
puppetlabs-stdlib Standard libr... @puppetlabs stdlib
trlinkin-validate_multi Validate that... @trlinkin dsl
puppet-extlib extlib provid... @puppet stdlib
```

2. We verified that we have the right module, so we'll install it with `module install`:

```
t@mylaptop ~ $ puppet module install puppetlabs-stdlib
Notice: Preparing to install into
```

```
/home/thomas/.puppetlabs/etc/code/modules ...
Notice: Downloading from https://forgeapi.puppet.com ...
Notice: Installing -- do not interrupt ...
/home/thomas/.puppetlabs/etc/code/modules
└── puppetlabs-stdlib (v4.25.0)
```

3. The module is now ready to use in your manifests; most good modules come with a `README.md` or `README` file to show you how to do this.

There's more...

Modules on the Forge include a `metadata.json` file, which describes the module and which operating systems the module supports. This file also includes a list of modules that are required by the module. This file was previously named `Modulefile` and was not in JSON format; the old `Modulefile` format was deprecated in version 3.6. As we will see in our next section, when installing a module from the Forge, the required dependencies will automatically be installed as well. Not all publicly-available modules are on the Puppet Forge. Some other great places to look at on GitHub are as follows:

- https://github.com/camptocamp
- https://github.com/example42

Managing Apache servers

Apache is the world's favorite web server, so it's highly likely that part of your Puppetly duties will include installing and managing Apache.

How to do it...

We'll install and use the `puppetlabs-apache` module to install and start Apache. This time, when we run the `puppet module install` command, we'll use the `-i` option to tell Puppet to install the module in our Git repository's module's directory:

1. Install the module using `puppet module install`:

```
t@mylaptop $ puppet module install -i modules puppetlabs-apache
Notice: Preparing to install into /home/thomas/puppet/modules ...
Notice: Downloading from https://forgeapi.puppet.com ...
Notice: Installing -- do not interrupt ...
/home/thomas/puppet/modules
└─┬ puppetlabs-apache (v3.1.0)
```

```
  ├── puppetlabs-concat (v4.2.1)
  └── puppetlabs-stdlib (v4.25.0)
```

2. Add the modules to your Git repository and push them out:

```
t@mylaptop $ git add modules
t@mylaptop $ git commit -m 'adding modules to repo'
[production 66f2ed5] adding modules to repo
 1052 files changed, 80823 insertions(+)
 create mode 100644 modules/apache/CHANGELOG.md
...
t@mylaptop $ git push origin production
...
remote: create mode 100644 modules/stdlib/types/windowspath.pp
To git.example.com:repos/puppet.git
 04178ab..66f2ed5 production -> production
```

3. Create a web server node definition in `site.pp`:

```
node webserver {
  class {'apache': }
}
```

4. Run Puppet to apply the default Apache module configuration:

```
[root@webserver ~]# puppet agent -t
Info: Caching certificate for webserver.example.com Info: Caching certificate for
webserver.example.com
Info: Using configured environment 'production' Info: Retrieving pluginfacts
Info: Retrieving plugin Notice:
/File[/opt/puppetlabs/puppet/cache/lib/facter/apache_version.rb]/ensure: defin
ed content as '{md5}2562993ee2b268e532b48cffc56bdaa0' Notice:
/File[/opt/puppetlabs/puppet/cache/lib/puppet/functions/apache]/ensure: created
...
Info: Concat[15-default.conf]: Scheduling refresh of Class[Apache::Service]
Info: Class[Apache::Service]: Scheduling refresh of Service[httpd]
Notice: /Stage[main]/Apache::Service/Service[httpd]/ensure: ensure changed 'stopped'
to 'running'
Info: /Stage[main]/Apache::Service/Service[httpd]: Unscheduling refresh on
Service[httpd]
Notice: Applied catalog in 3.06 seconds
```

5. Verify that you can reach `webserver.example.com`:

```
[root@webserver ~]# curl http://webserver.example.com
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<html>
 <head>
  <title>Index of /</title>
 </head>
 <body>
<h1>Index of /</h1>
```

```
  <table>
   <tr><th valign="top"><img src="/icons/blank.gif" alt="[ICO]"></th><th><a href="?
C=N;O=D">Name</a></th><th><a href="?C=M;O=A">Last modified</a></th><th><a href="?
C=S;O=A">Size</a></th><th><a href="?C=D;O=A">Description</a></th></tr>
   <tr><th colspan="5"><hr></th></tr>
   <tr><th colspan="5"><hr></th></tr>
</table>
</body></html>
```

How it works...

Installing the `puppetlabs-apache` module from the Forge causes both `puppetlabs-concat` and `puppetlabs-stdlib` to be installed into our modules directory. The `concat` module is used to stitch snippets of files together in a specific order. It is used by the Apache module to create the main Apache configuration files.

We then defined a web server node and applied the `apache` class to that node. We used all the default values and let the Apache module configure our server to be an Apache web server.

The Apache module then rewrote all our Apache configurations. By default, the module purges all the configuration files from the Apache directory (`/etc/apache2` or `/etc/httpd` , depending on the distribution). The module can configure many different distributions and handle the nuances of each distribution. As a user of the module, you don't need to know how your distribution deals with the Apache module configuration.

After purging and rewriting the configuration files, the module ensures that the `apache2` service is running (`httpd` on the test box based on **Enterprise Linux** (**EL**)).

We then tested the webserver using cURL. There was nothing returned but an empty index page. This is the expected behavior. Normally, when we install Apache on a server, there are some files that display a default page (`welcome.conf` on EL-based systems); since the module purged those configurations, we only see an empty page.

In a production environment, you would modify the defaults applied by the Apache module; the suggested configuration from the README is as follows:

```
class { 'apache':
  default_mods => false,
  default_confd_files => false,
}
```

Creating Apache virtual hosts

Apache virtual hosts are created with the `apache` module with the `apache::vhost` defined type. We will create a new `vhost` on our Apache webserver called `navajo` , after one of the Apache tribes.

How to do it...

Follow these steps to create Apache virtual hosts:

1. Create a `navajo apache::vhost` definition as follows:

```
apache::vhost { 'navajo.example.com':
  port => '80',
  docroot => '/var/www/navajo',
}
```

2. Create an index file for the new `vhost`:

```
$navajo = @(NAVAJO)
  <html>
    <head>
      <title>navajo.example.com</title>
    </head>
    <body>http://en.wikipedia. org/wiki/Navajo_people
    </body>
  </html>
  | NAVAJO
file {'/var/www/navajo/index.html':
  content => $navajo,
  mode => '0644',
  require => Apache::Vhost['navajo.example.com']
}
```

3. Run Puppet to create the new `vhost`:

```
[root@webserver ~]# puppet agent -t
Info: Caching catalog for webserver.example.com
...
Notice:
/Stage[main]/Main/Node[webserver]/Apache::Vhost[navajo.example.com]/File[/var/www/navajo
 created
Notice: /Stage[main]/Main/Node[webserver]/Apache::Vhost[navajo.example.com]/Concat[25-
navajo.example.com.conf]/File[/etc/httpd/conf.d/25-navajo.example.com.conf]/ensure:
defined content as '{md5}c2c851756828cc876e0cadb159d573fa'
Info: Concat[25-navajo.example.com.conf]: Scheduling refresh of Class[Apache::Service]
Info: Class[Apache::Service]: Scheduling refresh of Service[httpd]
```

4. Verify that you can reach the new virtual host:

```
[root@webserver ~]# curl http://navajo.example.com
<html>
 <head>
 <title>navajo.example.com</title>
 </head>
 <body>http://en.wikipedia. org/wiki/Navajo_people
 </body>
</html>
```

Note

You will need `navajo.example.com` to resolve to your host; you can create a host entry with the following snippet: `host { 'navajo.example.com': ip => '127.0.0.1' }` Alternatively, you can specify how to reach `navajo.example.com` when issuing the `curl` command, as follows: `curl -H 'Host: navajo.example.com' http://127.0.0.1`

How it works...

The `apache::vhost` defined type creates a virtual host configuration file for Apache, `25-navajo.example.com.conf`. The file is created with a template; `25` at the beginning of the filename is the priority level of this virtual host. When Apache first starts, it reads through its configuration directory and starts executing files in alphabetical order. Files that begin with numbers are read before files that start with letters. In this way, the `Apache` module ensures that the virtual hosts are read in a specific order, which can be specified when you define the virtual host. The contents of this file are as follows:

```
# ************************************
# Vhost template in module puppetlabs-apache
# Managed by Puppet
# ************************************
<VirtualHost *:80>
 ServerName navajo.example.com
## Vhost docroot
 DocumentRoot "/var/www/navajo"
## Directories, there should at least be a declaration for /var/www/navajo
<Directory "/var/www/navajo">
 Options Indexes FollowSymLinks MultiViews
 AllowOverride None
 Require all granted
```

```
 </Directory>
## Logging
 ErrorLog "/var/log/httpd/navajo.example.com_error.log"
 ServerSignature Off
 CustomLog "/var/log/httpd/navajo.example.com_access.log" combined
</VirtualHost>
```

As you can see, the default file has created log files and set up directory access permissions and options, in addition to specifying the listen port and `DocumentRoot`.

The `vhost` definition creates the `DocumentRoot` directory, specified as `root` to the `apache::virtual` definition. The directory is created before the virtual host configuration file; after that file has been created, a `notify` trigger is sent to the Apache process to restart.

Our manifest included a file that required the `Apache::Vhost['navajo.example.com']` resource; our file was then created after the directory and the virtual host configuration file.

When we run cURL on the new website, we see the contents of the index file that we created.

There's more...

Both the defined type and the template take into account a multitude of possible configuration scenarios for virtual hosts. It is highly unlikely that you will find a setting that is not covered by this module. You should look at the definition for `apache::virtual` and the sheer number of possible arguments.

The module also takes care of several settings for you. For instance, if we change the listen port on our navajo virtual host from `80` to `8080`, the module will make the following changes in `/etc/httpd/conf.d/ports.conf`:

```
-Listen 80
+Listen 8080
-NameVirtualHost *:80
+NameVirtualHost *:8080
```

And it will make these changes in our virtual host file:

```
-<VirtualHost *:80>
+<VirtualHost *:8080>
```

After a `puppet run` on our web server, we would now be able to `curl` on port `8080` and see the same results:

```
[root@webserver ~]# curl http://navajo.example.com:8080
<html>
 <head>
 <title>navajo.example.com</title>
 </head>
 <body>http://en.wikipedia. org/wiki/Navajo_people
 </body>
</html>
```

And when we use `curl` to connect to `najavo.example.com` on `port 80,` we see the following:

```
[root@webserver ~]# curl http://navajo.example.com
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<html>
 <head>
  <title>Index of /</title>
 </head>
 <body>
<h1>Index of /</h1>
  <table>
   <tr><th valign="top"><img src="/icons/blank.gif" alt="[ICO]"></th><th><a href="?
C=N;O=D">Name</a></th><th><a href="?C=M;O=A">Last modified</a></th><th><a href="?
C=S;O=A">Size</a></th><th><a href="?C=D;O=A">Description</a></th></tr>
   <tr><th colspan="5"><hr></th></tr>
   <tr><th colspan="5"><hr></th></tr>
</table>
</body></html>
```

As we can see, the virtual host is no longer listening on port `80` and we receive the default empty directory listing we saw in our earlier example.

Creating NGINX virtual hosts

NGINX is a fast, lightweight web server that is preferred over Apache in many contexts, especially where high performance is important. NGINX is configured slightly differently than Apache; like Apache though, there is a Forge module that can be used to configure NGINX for us. Unlike Apache, however, the module that is suggested for use is not supplied by puppetlabs, but by James Fryman. This module uses some interesting tricks to configure itself. Previous versions of this module used R.I. Pienaar's `module_data package` . This package is used to configure `hieradata` within a module. It's used to supply default values to the `nginx` module. I wouldn't recommend starting out with this module at this point, but it is a good example of where module configuration may be headed in the future. Giving modules the ability to modify hieradata may prove useful.

How to do it...

In this example, we'll use a Forge module to configure NGINX. We'll download the module and use it to configure `virtualhosts` :

1. Download the `puppet-nginx` module from the Forge:

```
t@mylaptop $ puppet module install -i modules puppet-nginx
Notice: Preparing to install into /home/thomas/puppet/modules ...
Notice: Created target directory /home/thomas/puppet/modules
Notice: Downloading from https://forgeapi.puppet.com ...
Notice: Installing -- do not interrupt ...
/home/thomas/puppet/modules
└┬┐ puppet-nginx (v0.11.0)
 ├── puppetlabs-concat (v4.2.1) [/home/thomas/.puppetlabs/etc/code/modules]
 └── puppetlabs-stdlib (v4.25.0) [/home/thomas/.puppetlabs/etc/code/modules]
```

2. Replace the definition for webserver with an `nginx` configuration:

```
node webserver {
  service {'httpd': ensure => false }
  class {'nginx': }
  nginx::resource::server{ 'mescalero.example.com':
    www_root => '/var/www/mescalero',
  }
  file {'/var/www/mescalero':
    ensure => 'directory',
    mode => '0755',
    require => Nginx::Resource::Server['mescalero.example.com'],
  }
  $mescalero = @(MESCALERO)
    <html>
      <head>
        <title>mescalero.example.com</title>
      </head>
      <body>
        http:// en.wikipedia.org/wiki/Mescalero
      </body>
    </html>
    | MESCALERO
  file {'/var/www/mescalero/index.html':
    content => $mescalero,
    mode => '0644',
    require => File['/var/www/mescalero'],
  }
}
```

3. Run `puppet agent` on your `webserver` node:

```
[root@webserver ~]# puppet agent -t
...
Info: Caching catalog for webserver.example.com [87/1835]
Info: Applying configuration version '1522256325'
Notice: /Stage[main]/Main/Node[webserver]/Service[httpd]/ensure: ensure changed
'running' to 'stopped'
Notice: /Stage[main]/Nginx::Package::Redhat/Yumrepo[nginx-release]/ensure: created
Info: Yumrepo[nginx-release](provider=inifile): changing mode of
```

```
/etc/yum.repos.d/nginx-release.repo from 600 to 6$
4
Notice: /Stage[main]/Nginx::Package::Redhat/Package[nginx]/ensure: created
Info: Class[Nginx::Package]: Scheduling refresh of Class[Nginx::Service]
Notice: /Stage[main]/Nginx::Config/File[/etc/nginx/conf.stream.d]/ensure: created
Notice: /Stage[main]/Nginx::Config/File[/etc/nginx/conf.mail.d]/ensure: created
...
Notice: /Stage[main]/Nginx::Config/File[/etc/nginx/nginx.conf]/content:
--- /etc/nginx/nginx.conf 2017-10-17 13:21:13.000000000 +0000
+++ /tmp/puppet-file20180328-19387-1bx1j5k 2018-03-28 17:00:01.290886282 +0000
@@ -1,32 +1,64 @@
+# MANAGED BY PUPPET
+user nginx;
...
Notice:
/Stage[main]/Main/Node[webserver]/Nginx::Resource::Server[mescalero.example.com]/Concat
av
ailable/mescalero.example.com.conf]/File[/etc/nginx/sites-
available/mescalero.example.com.conf]/ensure: defined con
tent as '{md5}69f8354a0f61a5175e3c2e6ee195262a'
Info: Concat[/etc/nginx/sites-available/mescalero.example.com.conf]: Scheduling
refresh of Class[Nginx::Service]
Notice:
/Stage[main]/Main/Node[webserver]/Nginx::Resource::Server[mescalero.example.com]/File[me

.conf symlink]/ensure: created
Info:
/Stage[main]/Main/Node[webserver]/Nginx::Resource::Server[mescalero.example.com]/File[me

onf symlink]: Scheduling refresh of Class[Nginx::Service]
Info: Class[Nginx::Service]: Scheduling refresh of Service[nginx]
Notice: /Stage[main]/Nginx::Service/Service[nginx]/ensure: ensure changed 'stopped' to
'running'
Info: /Stage[main]/Nginx::Service/Service[nginx]: Unscheduling refresh on
Service[nginx]
Notice: /Stage[main]/Main/Node[webserver]/File[/var/www/mescalero]/ensure: created
Notice: /Stage[main]/Main/Node[webserver]/File[/var/www/mescalero/index.html]/ensu
```

```
re: defined content as '{md5}cb3f
6427eb6923b58fa159916ef296f6'
Notice: Applied catalog in 18.53 seconds
```

4. Verify that you can reach the new `virtualhost`:

```
[root@webserver ~]# curl -H 'Host: mescalero.example.com' http://127.0.0.1
<html>
 <head>
 <title>mescalero.example.com</title>
 </head>
 <body>
 http:// en.wikipedia.org/wiki/Mescalero
```

```
 </body>
</html>
```

How it works...

Installing the `puppet-nginx` module causes the `concat` and `stdlib` modules to be installed (we already had a copy of `stdlib`, so all this did was verify we had it). We run Puppet on our master to have the plugins created by these modules added to our running master. The `stdlib` and `concat` have facter and Puppet plugins that need to be installed for the NGINX module to work properly.

With the plugins synchronized, we can then run the Puppet agent on our web server. As a precaution, we add a resource to stop Apache if it was previously started (we can't have NGINX and Apache both listening on port `80`). After the puppet agent runs, we verified that NGINX was running and the virtual host was configured.

There's more...

This module had a lot of interesting things like the `apache` modules, the `NGINX` module can be used to create very complex configurations. One interesting option is the ability to configured `NGINX` as a proxy server in only a few lines of code. I suggest looking at the `README.md` of the module or the module page on the forge at https://forge.puppet.com/puppet/nginx

In the next section, we'll use a supported module to configure and manage MySQL installations.

Managing MariaDB

MySQL is a very widely used database server, and you'll probably need to install and configure a MySQL server at some point. On January 27, 2010, Oracle bought Sun Microsystems. Sun Microsystems originally developed MySQL. When Oracle took over, the project was forked into MariaDB. MariaDB was made to keep the MySQL server open source and maintained. When we use the `puppetlabs-mysql` module, we will actually be installing MariaDB. The `puppetlabs-mysql` module can simplify your MariaDB deployments.

How to do it...

Follow these steps to create the example:

1. Install the `puppetlabs-mysql` module:

```
t@burnaby $ puppet module install -i modules puppetlabs-mysql
Notice: Preparing to install into /home/thomas/puppet/modules ...
Notice: Downloading from https://forgeapi.puppet.com ...
Notice: Installing -- do not interrupt ...
/home/thomas/puppet/modules
└┬┬ puppetlabs-mysql (v5.3.0)
 ├── puppet-staging (v3.1.0)
 ├── puppetlabs-stdlib (v4.25.0)
 └── puppetlabs-translate (v1.1.0)
```

2. Create a new node definition for your MySQL server:

```
node dbserver {
  class { 'mysql::server':
       root_password => 'fenago',
    override_options => {
```

```
            'mysqld' => {
   'max_connections' => '1024'
      }
    }
  }
}
```

3. Run Puppet to install the database server and apply the new root password:

```
Info: Applying configuration version '1522263661'
Notice: /Stage[main]/Mysql::Server::Config/File[mysql-config-file]/ensure: defined
content as '{md5}15d890f0648fc49e43fcffc6ed7bd2d8'
Notice: /Stage[main]/Mysql::Server::Install/Package[mysql-server]/ensure: created
Notice: /Stage[main]/Mysql::Server::Installdb/Mysql_datadir[/var/lib/mysql]/ensure:
created
Notice: /Stage[main]/Mysql::Server::Service/Service[mysqld]/ensure: ensure changed
'stopped' to 'running'
Info: /Stage[main]/Mysql::Server::Service/Service[mysqld]: Unscheduling refresh on
Service[mysqld]
Notice:
/Stage[main]/Mysql::Server::Root_password/Mysql_user[root@localhost]/password_hash:
defined 'password_hash' as '*6ABB0D4A7D1381BAEE4D078354557D495ACFC059'
Notice: /Stage[main]/Mysql::Server::Root_password/File[/root/.my.cnf]/ensure: defined
content as '{md5}a65d0eb64218eebf5eeed997de0e72a8'
Info: Creating state file /opt/puppetlabs/puppet/cache/state/state.yaml
Notice: Applied catalog in 16.42 seconds
```

4. Verify that you can connect to the database:

```
[root@dbserver ~]# mysql
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 10
Server version: 5.5.56-MariaDB MariaDB Server

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> \q
Bye
```

How it works...

The MySQL module installs the MariaDB server and ensures that the server is running. It then configures the root password for MariaDB. The module does a lot of other things for you as well. It creates a `.my.cnf` file with the root user password. When we run the `mysql client`, the `.my.cnf` file sets all the defaults, so we do not need to supply any arguments.

There's more...

In the next section, we'll show you how to create databases and users.

Creating databases and users

Managing a database means more than ensuring the service is running; a database server is nothing without databases. Databases need users and privileges. Privileges are handled with `GRANT` statements. We will use the `puppetlabs-mysql` package to create a database and a user with access to that database. We'll create a MariaDB user, `drupal`, and a database, called `drupal`. We'll create a `nodes` table within the `drupal` database and place data into that table.

How to do it...

Follow these steps to create the databases and users:

1. Create a database definition within your `dbserver` class:

```
mysql::db { 'drupal':
  host     => 'localhost',
  user     => 'drupal',
  password => 'fenago',
  sql      => '/root/drupal.sql',
  require  => File['/root/drupal.sql']
}
$drupal = @(DRUPAL)
  CREATE TABLE users (
    id INT PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(255),
    body TEXT);
  INSERT INTO users (id, title, body) VALUES (1,'First Node','Contents of first
Node');
  INSERT INTO users (id, title, body) VALUES (2,'Second Node','Contents of second
Node');
  | DRUPAL
file { '/root/drupal.sql':
  ensure  => present,
  content => $drupal,
}
```

2. Allow the Drupal user to modify the nodes table:

```
mysql_grant { 'drupal@localhost/drupal.nodes':
  ensure     => 'present',
  options    => ['GRANT'],
  privileges => ['ALL'],
  table      => 'drupal.nodes',
  user       => 'drupal@localhost',
}
```

3. Run Puppet to have the user, database, and `GRANT` created:

```
[root@dbserver ~]# puppet agent -t
Info: Using configured environment 'production'
Info: Retrieving pluginfacts
Info: Retrieving plugin
```

```
Info: Loading facts
Info: Caching catalog for dbserver.example.com
Info: Applying configuration version '1522266120'
Notice: /Stage[main]/Main/Node[dbserver]/File[/root/drupal.sql]/ensure: defined
content as '{md5}f0f695b4b093cd225da626ae207f425a'
Notice:
/Stage[main]/Main/Node[dbserver]/Mysql::Db[drupal]/Mysql_database[drupal]/ensure:
created
Info: /Stage[main]/Main/Node[dbserver]/Mysql::Db[drupal]/Mysql_database[drupal]:
Scheduling refresh of Exec[drupal-import]
Notice:
/Stage[main]/Main/Node[dbserver]/Mysql::Db[drupal]/Mysql_user[drupal@localhost]/ensure:
 created
Notice:
/Stage[main]/Main/Node[dbserver]/Mysql_grant[drupal@localhost/drupal.nodes]/ensure:
created
Notice:
/Stage[main]/Main/Node[dbserver]/Mysql::Db[drupal]/Mysql_grant[drupal@localhost/drupal.*
 created
```

```
Notice: /Stage[main]/Main/Node[dbserver]/Mysql::Db[drupal]/Exec[drupal-import]:
Triggered 'refresh' from 1 event
Notice: Applied catalog in 0.32 seconds
```

4. Verify that the database and table have been created:

```
[root@dbserver ~]# mysql -u drupal -pfenago drupal
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 40
Server version: 5.5.56-MariaDB MariaDB Server

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [drupal]> show tables;
+------------------+
| Tables_in_drupal |
+------------------+
| users |
+------------------+
1 row in set (0.00 sec)
```

5. Verify that our default data has been loaded into the table:

```
MariaDB [drupal]> select * from users;
+----+-------------+-------------------------+
| id | title | body |
+----+-------------+-------------------------+
```

```
| 1 | First Node | Contents of first Node |
| 2 | Second Node | Contents of second Node |
+----+-------------+------------------------+
2 rows in set (0.00 sec)

MariaDB [drupal]> \q
Bye
```

How it works...

We will start with the definition of the new `drupal` database:

```
mysql::db { 'drupal':
  host     => 'localhost',
  user     => 'drupal',
  password => 'fenago',
  sql      => '/root/drupal.sql',
  require  => File['/root/drupal.sql']
 }
```

We specify that we'll connect from `localhost` using the `drupal` user. We give the password for the user and specify a SQL file that will be applied to the database after the database has been created. We require that the SQL file exists before we create the user; that way, the `CREATE TABLE` in the SQL file can be executed:

```
$drupal = @(DRUPAL)
  CREATE TABLE users (
    id INT PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(255),
    body TEXT);
  INSERT INTO users (id, title, body) VALUES (1,'First Node','Contents of first
Node');
  INSERT INTO users (id, title, body) VALUES (2,'Second Node','Contents of second
Node');
  | DRUPAL
file { '/root/drupal.sql':
  ensure  => present,
  content => $drupal,
}
```

We then ensure that the user has the appropriate privileges with a `mysql_grant` resource:

```
mysql_grant { 'drupal@localhost/drupal.nodes':
  ensure     => 'present',
  options    => ['GRANT'],
  privileges => ['ALL'],
  table      => 'drupal.nodes',
  user       => 'drupal@localhost',
}
```

There's more...

Using the `puppetlabs-mysql` and `puppetlabs-apache` modules, we can create an entire functioning web server. The `puppetlabs-apache` module will install Apache, and we can include the PHP module as well. We can then use the `puppetlabs-mysql` module to install the MariaDB server, and then create the required Drupal databases and seed the database with the data.

Deploying a new Drupal installation would be as simple as including a class on a node. There are numerous examples on the web of this workflow. Systems such as Wordpress and Django can be installed from scratch in this way.