Automating Openstack using PHP7

Mostafa Abd-ElHamid Atwa

bookboon
The eBook company

MOSTAFA ABD-ELHAMID ATWA

AUTOMATING OPENSTACK USING PHP7

Automating Openstack using PHP7
1st edition

ISBN 978-87-403-1581-3

CONTENTS

DEDICATION

For **Manon Niazi** the Deutschlander.

I was going to the agricultural college to see her, and then I had to change to study computer science in the computer academy.

It was 2000, I was still unable to type on the keyboard, I was searching for letters on the keyboard in the agricultural college in the computer laboratory.

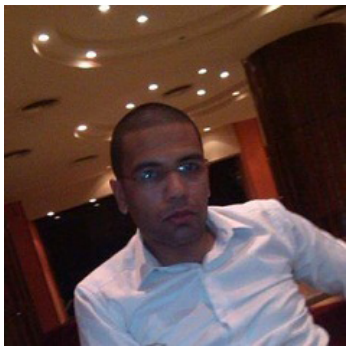
She was sitting beside me and told me about some keys, to make me learn the keyboard.

I took a decision to learn computing to become somehow professional in computer programming and of course learn how to type on the keyboard.

Now, I am shorthand programmer typing fast code to make web applications and programming computer algorithms.

Thank you **Manon**.

ABOUT THE AUTHOR



Mostafa A. Hamid
Bachelor Degree Holder in Management Information Systems
Certified in Java from the American University in Cairo
Certified in Several Programming Languages from SUNY Potsdam USA
Certified in JSF and HTTP 2 (SPDY) from UDEMY USA
Certified IOTx from MIT USA

ABOUT THE TECHNICAL AND OFFICIAL REVIEWER

Sir Engineer Manuel Lemos
The business owner of http://phpclasses.org/
Manuel Lemos is one of the **famous Engineers in PHP and JavaScript** worldwide with more than **10 years of experience**.
Manuel Lemos's website is the largest **PHP repository on the planet** and he has more than **2 billion registered users contributing in PHP and JavaScript** for his other site http://jsclasses.org

PREFACE

Openstack®is the world's #1 Cloud Operating System and Cloud Management Software that is dedicated to manage the cloud infrastructure.

Openstack® is a cloud management tool that is designed to make cloud into 3 or more layers according to the architecture needed by the user. The 4 management layers are mainly focused on:

1. Imaging
2. Storage
3. Networking
4. Orchestration

Throughout our book, we will be using PHP7 as the automation tool beside shell scripting to automate the openstack® environment.

PHP 7 is a programming language which will permit us to create a user interface that will be used to control Openstack®features.

After all, we will be controlling nearly most of the main features of Openstack®from within the power of PHP 7 OOP application.

1 INTRODUCING OPENSTACK®

Openstack®is a cloud operating system which is designed to manage a public or private cloud infrastructure using imaging, block storage, networking, orchestration, and dashboard services as the main components of the Openstack®cloud operating system environment.

Throughout our book, we will introduce to you the Openstack® basic usage through the installation of the development version of Openstack®which is called Devstack.

We will run through the installation of the devstack environment on UBUNTU operating system which is one of the most famous operating systems and is a favorable LINUX distribution.

We will use the aid of screenshots, illustrations from the source company UBUNTU (Canonical LTD) or Openstack (Rackspace).
First of all, I will show you how to download the latest UBUNTU operating system distribution (Server and Desktop) editions.

Please visit the following links:
Desktop: https://www.ubuntu.com/download/desktop
As of the time of writing this book, it is UBUNTU 16.04.01 LTS (Long Term Support)
Server: https://www.ubuntu.com/download/server
This should be the download link for the server edition.
After downloading the required images, you can add them to a bootable USB stick using this tool on Windows or LINUX Operating System:

https://rufus.akeo.ie/

After all, you will be in a need to boot your computer if you will use your bare metal for this purpose, or you will need to create a virtual machine to using the image files that we downloaded the desktop or the server versions.

Next, we will walk through the installation of LAMP Stack.

2 INSTALLING LAMP STACK

Let's go through the installation of the lamp stack:

1. Visit this URL: https://bitnami.com/stack/lamp/installer
2. Download the LAMP stack.
3. Open the terminal using CTRL + ALT + T
   Into the terminal window, type in the following command: "`sudo sh /path/to/downloaded-lamp-stack.run`" OR "`sudo /path/to/downloaded-lamp-stack.run`"
4. Please provide the necessary passwords for mysql root user and complete the setup wizard to finish the installation process.

Now, we have ran through the installation of LAMP stack on UBUNTU to install Apache, MySQL, and PHP7, then we will go ahead and install devstack.

3 INSTALLING DEVSTACK

To install devstack, let us go through the following steps:

1. Open the terminal window by typing CTRL + ALT + T.
2. Into the terminal window, type the following command: "sudo apt-get update"
3. Then type in the following command: "sudo apt-get install git"
4. After that, type in the following command to create a local.conf file as follows:
   - "sudo touch local.conf"
   - "sudo nano local.conf"
   - Then, paste in the following code by pressing CTRL + SHIFT + V

```
[[local|localrc]]
ADMIN_PASSWORD=your_password
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
```

5. Then press CTRL + X and save the file when prompted.
6. After all type in the following command to download devstack package into your system: "git clone https://git.openstack.org/openstack-dev/devstack"
7. Now, let us create a special user on our operating system to manage the installation process and create the necessary privileges for this user as follows: "devstack/tools/create-stack-user.sh; su stack" OR "cd devstack/tools" – "sh create-stack-user.sh && su stack" both ways will do the same functionality.
8. Finally, we start our installation using this command: "cd devstack; ./stack.sh"

Devstack will prompt for passwords if you did not create local.conf file in step number 4. This way, we finished the installation and now, you can browse to http://localhost to see the dashboard and browse the capabilities of openstack using the devstack all-in-one installation.

To exit the terminal window, type in exit, then press the return key, then exit again, then press the return key again to exit the terminal window.

Notice that the terminal window will be only available through the desktop edition of UBUNTU, but the server edition already opens its interface with the full screen command line without any window interface.

4 QUICK REFERENCE TO PHP7

PHP is a programming language, owned now by a company called Zend for the founder Zeev Suraski and it was originally developed by Rasmus Lerdorf, the creator of personal home page before renaming it to the hypertext pre-processor as PHP.

PHP version 7 holds a lot of different functionalities than other older versions of PHP including the return types in the function signature, new functions into the engine for handling arrays, new functions into the engine for handling object orientation etc.

The website that contains all the documentation of PHP7 called:
http://www.php.net/

Let us now create our first PHP7 script that will permit us to get to know the place where we can place our PHP7 files.

After installing the lamp stack, you will find a directory called bitnami under /etc directory in our LINUX UBUNTU distribution; let us browse this directory using the terminal as follows:

```
"cd /etc/bitnami/"
```

Then, we will find a directory called apache2, then another directory to go through which will be htdocs. Now let us go through that directory using the following command:

```
"cd /etc/bitnami/apache2/htdocs/"
```

After that, we will create a file called index.php and place some PHP7 code inside it then browse it using the browser on the same machine or from another machine as follows:

```
"sudo touch /etc/bitnami/apache2/htdocs/index.php"
"sudo nano /etc/bitnami/apache2/htdocs/index.php"
```

5 INTRODUCTION TO PROGRAMMING

Programming is a technique in computer science which is creating special kinds of files written in a specific language that the computers and humans can read and understand. The programmer uses these programming languages to create a program with a purpose to make certain tasks or a certain task. One of the programming languages that we will use to create programs will be PHP7 which is the most popular programming language in the world for making websites, web applications and automating operating systems and programs.

Let us add the following code to our index.php file as follows using the copy and paste technique that we illustrated before:

```
<?php
    namespace Application\Bootstrapper;
    use \Autoloader\ApplicationInitializer;
    class App{
      public static function __init() : string{
         require 'libs/Autoloader.php';
```

```
        \Autoloader\ApplicationInitializer\Autoloader::__init();
      }
    }
    \ Application\Bootstrapper\App::__init();
```

Close and save the file as we mentioned using CTRL + C to close the file, then save it when prompted.
Next, we will change the mode of our files to be accessible from outside the server as follows:
Type into the terminal window the following command:

```
"sudo chmod 777 –R /etc/bitnami/apache2/htdocs"
```

Then we will give the www-data the sudo permission to gain access to the shell bashrc commands to be executed from within our web server user account as follows:

```
"sudo adduser www-data sudo"
```
OR
```
"sudo adduser www-data sudoers"
```

Now we gave permission to our web server user account which is www-data to access operating system resources such as the bashrc to execute commands into our operating system.

Let us create a directory and another new file into that directory as follows:

```
"mkdir /etc/bitnami/apache2/htdocs/libs"
```

Then create a file into the libs folder called autoloader as follows:

```
"touch /etc/bitnami/apache2/htdocs/libs/Autoloader.php"
```

Then, let us go ahead and put the following code into it as follows:

```
<?php
    namespace Autoloader\ApplicationInitializer;
    use \Router\Initialize\Route;
    class Autoloader{
      public static function __init() : string{
        require 'libs/Router.php';
        if($_SERVER['REQUEST_URI'] === NULL){
            \Router\Initialize\Route\Router::__init();
        }else{
            $uri = rtrim($_SERVER['REQUEST_URI'], '/');
            $uri = ltrim($uri, '/');
            \Router\Initialize\Route\Router::__init($uri);
```

```
        }
      }
    }
```

Let us create one more PHP file called Router.php into the same libs forlder as follows:

```
"touch /etc/bitnami/apache2/htdocs/libs/Router.php"
"nano /etc/bitnami/apache2/htdocs/libs/Router.php"
```

Then we will paste in the following code into the Router.php file as follows:

```
<?php
    namespace Router\Initialize\Route;
    use \Controller\Main;
    class Router{
      public static function __init($uri) : void{
         if(filter_var($uri,
FILTER_VALIDATE_REGEXP,array("options"=>
array("regexp"=>"/^[a-zA-Z0-9]{4,122}$/")))){
            if(file_exists('/controllers/' . $uri . 'Controller.php')){
               require '/controllers/' . $uri . 'Controller.php';
               \Controller\Main\Controller::__init();
            }else{
               require '/controllers/MainController.php';
               \Controller\Main\MainController::__init();
            }
         }else{
            require '/controllers/MainController.php';
            \Controller\Main\MainController::__init();
         }
      }
    }
```

Now, we created a file called router to route to the controllers upon user request, then route to the initialization function into each controller with filtered route to keep sense of security and avoid any unwanted user input using the URL and force the user to keep the URL according to our filter that we assigned to accept only letters, numbers, and in capital or small letters only.

10 more things to go,

- We will create the main controller that will contain our main functionalities to display the dashboard.
- Create the header file
- Create the footer file
- Create the dashboard file
- Create the login file
- Create the logout file
- Create a session handler
- Create a database library
- Then we will create an .htaccess file to redirect our calls to our index.php file
- Finally, we will modify the apache configuration file to enable the rewrite engine.

We will start with our main controller to display the dashboard that will contain the main user interface to control Openstack environment.
We will create into our main controller three main compartments (header, body, and footer).

The header and footer will be fixed and globalized across our application, but the body is the one that will be changing upon the user request.

Let us create the controllers directory and create a file called MainController.php as follows:

```
"touch /etc/bitnami/apache2/htdocs/controllers/MainController.php"
"nano /etc/bitnami/apache2/htdocs/controllers/MainController.php"
```

Then, we will paste into this file the following code to display the dashboard content after the user performs the login process as follows:

```
<?php
    namespace Controller\Main;
    use \SessionLib\Check;
    use \View\Header;
    use \View\Footer;
    use \View\Login;
    class MainController{
      public static function __init() : string{
         require 'libs/Session.php';
         if(\SessionLib\Check\Session::__checkIfLoggedIn()){
             require '/views/Header.php';
             \View\Header\Header::__init();
```

```
            require '/views/Dashboard.php';
            \View\Dashboard\Dashboard::__init();
            require '/views/Footer.php';
            \View\Footer\Footer::__init();
        }else{
            require '/views/Login.php';
            \View\Login\Login::__init('Sorry, Please Login');
        }
      }
    }
```

Now, let us go ahead and create our Login file called Login.php into a directory called views as follows:

```
"touch /etc/bitnami/apache2/htdocs/Login.php"
"nano /etc/bitnami/apache2/htdocs/Login.php"
```

Now, we will put the following code into our Login.php file as follows:

```
<?php
    namespace View\Login;
    class Login{
      public static function __init($message) : void{
        if($message){
            print_r($message);
        }else{
            echo '<script type="text/javascript" src="http://crypto-
js.googlecode.com/svn/tags/3.1.2/build/rollups/md5.js"></script>
            <script type="text/javascript" src="http://
introducingmyresume.netau.net/digestAuthRequest.js"></script>
            <script>
            function __validateLogin(){
                var email = document.getElemnentById("email").value;
                var password = document.getElementById("password").value;
                if((email === null) || (email === "")){
                   document.getElementById("result").
innerHTML= "Please enter a valid e-mail.";
                   return false;
                }
                if((password === null) || (password === "")){
                   document.getElementById("result").
innerHTML = "Please enter a valid password.";
                   return false;
                }
                var url = "/LoginProcess";
                var req = new digestAuthRequest('GET',
```

```
                url, email, password);
                req.request(function(data) {
                    if(data === "Done")
                        window.location = "/Dashboard";
                },function(errorCode){
                    document.getElementById('result').
innerHTML = JSON.sgringify(errorCode);
                });
        }
            </script>
                <div style = "text-align: center;">
                <br /><br />
                <input type = "email" class = "form-
control" placeholder = "E-Mail" id = "email" />
                <br /><br />
                <input type = "password" class = "form-
control" placeholder = "Password" id = "password" />
                <br /><br /><br />
                <button type = "button" class = "btn btn-primary"
onclick = "javascript:return __validateLogin();">Login</button>
                </div>';
        }
      }
    }
```

Now, we notice that we created a login form here in our file that will post data to / LoginProgress controller. Let us create our LoginProgress controller to handle our login request and validate the data through our database connection that we will create into a database library.

“touch /etc/bitnami/apache2/controllers/LoginProgress.php”
“nano /etc/bitnami/apache2/controllers/LoginProgress.php”

Let us put in the following code to check if the user is available to login or not as follows:

```
<?php

    namespace Controller\Login;
    use \Db\Connection;
    use \Session\Handler;

    class LoginProgress{
      public static function __init() : boolean{
         $email = $_REQUEST['email'];
```

```
        $password = $_REQUEST['password'];
        if(!filter_var(base64_decode($email), FILTER_VALIDATE_EMAIL)){
            echo 'Problem with E-Mail, please provide a valid e-mail.';
                header("HTTP/1.1 401 Unauthorized");
            exit;
        }elseif(!filter_var(base64_decode($password), FILTER_VALIDATE_RE
GEXP,array("options"=>array("regexp"=>"/^[a-zA-Z0-9]{4,122}$/")))){
            echo 'Problem with Password, please provide a valid password';
            header("HTTP/1.1 401 Unauthorized");
            exit;
        }else{
            require 'libs/Crud.php';
            if(\Db\Connection\Crud::__getNum("SELECT id FROM users
WHERE email = '$email' AND password = md5('$password')")){
                $realm = "RestrictedAccessArea";
                if (empty($_SERVER['PHP_AUTH_DIGEST'])) {
                   header('HTTP/1.1 401 Unauthorized');
                   header('WWW-Authenticate: Digest realm="'.$realm
.'",qop="auth",nonce="'.uniqid().'",opaque="'.md5($realm).'"');
                }
                header("HTTP/1.1 200 OK");
$A1 = md5($data['email'] . ':' . $realm . ':' . $users[$data['email']]);
                   $A2 = md5($_SERVER['REQUEST_METHOD'].':'.$data['uri']);

$valid_response = md5($A1
                   .':'.$data['nonce'].
                   ':'.$data['nc'].':'
                   .$data['cnonce'].':'
                   .$data['qop'].':'.$A2);
                   if ($data['response'] != $valid_response){

die('Wrong Credentials!');
                   }
                   require 'libs/SessionHandler.php';
                   if(\Session\Handler\Session::__start($email)){
                        echo 'Done';
                        return true;
                   }else{
                       header('HTTP/1.1 401 Unauthorized');
                       return false;
                   }
                }elseif(
                   !($data = self::http_digest_parse(
                       $_SERVER['PHP_AUTH_DIGEST'])
                   )
                       || !isset($users[$data['email']])
                   ){
                   header('HTTP/1.1 401 Unauthorized');
```

```
                    return false;
                }else{
                    header('HTTP/1.1 401 Unauthorized');
                    return false;
            }
        }
        protected static function http_digest_parse($txt) : array{
            $needed_parts =
                array('nonce'=>1, 'nc'=>1,
                'cnonce'=>1,
                 'qop'=>1,
                   'email'=>1,
                   'uri'=>1, 'response'=>1
                );
            $data = array();
            $keys = implode('|', array_keys($needed_parts));
            preg_match_all('@(' . $keys . ')=(?:([\'"])
([^\2]+?)\2|([^\s,]+))@', $txt, $matches, PREG_SET_ORDER);
        foreach ($matches as $m) {
            $data[$m[1]] = $m[3] ? $m[3] : $m[4];
            unset($needed_parts[$m[1]]);
            return $needed_parts ? false : $data;
        }
    }
}
```

Now, we created a server side digest authentication over a protocol called HTTP, and created a caller javascript using XmlHttpRequest caller to authenticated to the server side using that digest method.

We will create a database on MySQL using a tool called PHPmyadmin as follows:

```
"sudo apt-get install phpmyadmin"
```

Then we will browse our localhost and point to phpmyadmin as follows:
http://localhost/phpmyadmin

Let us now create a new database, then create a table and fill into the table with 2 entries as users as follows:

Open a tab called SQL, and type in the following command:

```
"CREATE DATABASE openstack_mangement;"
```

Then click GO to execute the database statement.

After that we will create a new table into the database after selecting the database to create the new table inside of it as follows:

```
"USE openstack_management"
```

This statement instructed the database server to use the database that we created in the previous step.

Then we will create the table as follows:

```
"CREATE TABLE openstack_users
(
id int NOT NULL AUTO_INCREMENT,
email varchar(255) NOT NULL,
password varchar(32) NOT NULL,
PRIMARY KEY (id)
);"
```

This way, we created a table into our openstack_management database called openstack_users over mysql server.

Now, we will create our database library as follows:

```
"touch /etc/bitnami/apache2/htdocs/libs/Crud.php"
"nano /etc/bitnami/apache2/htdocs/libs/Crud.php"
```

Then, we will copy and paste the following code inside of it as follows:

```
<?php

namespace Db\Connection;
use \PDO;
class Crud{
    protected static function __connect() : \PDO{
      try {
         return new \PDO('mysql:dbname=openstack_management;host=localhost',
             'mysql_user', 'mysql_passowrd');
      }catch(\PDOException $e){
         return \PDOException;
      }
    }
    public static function __getNum ($sql) : int{
      $dbh = self::__connect();
      $sel = $dbh->prepare($sql);
      $sel->execute();
      return (int)$sel->rowCount();
```

```
    }
    public static function __exec($sql) : boolean{
      $dbh = self::__connect();
      $exec = $dbh->prepare($sql);
      $exec->execute();
      $exec ? return TRUE: return FALSE;
    }
}
```

For now, this will be our database library file until we add more functions to it later.
We will now create our Header.php file which will contain the main menu for our programmed application to control the main functionalities of Openstack. This main menu will be including 5 main sections:

1. Openstack Networking
2. Openstack Storage
3. Openstack Imaging
4. Openstack Compute
5. Openstack Orchestration

I encourage you to take a look at the following diagram that shows the architecture of the Openstack cloud operating system:
http://docs.openstack.org/admin-guide/_images/openstack-arch-kilo-logical-v1.png
Notice that the main components are Networking, Block Storage, Imaging, Compute, and Orchestration Services, and of course there are a lot of other services but not directly involved in the interaction with the direct user of Openstack and hidden with automation and directions from the main 5 services that we mentioned in the previous section.

Let us now create our Header.php file into our views director as follows:

```
"touch /etc/bitnami/apache2/htdocs/views/Header.php"
"nano /etc/bitnami/apache2/htdocs/views/Header.php"
```

Now, we put in the following code into the file as follows:

```
<?php
      namespace View\Header;
      class Header{
         public static function __init() : string{
             return '<style>
                 ul {
                    list-style-type: none;
                    margin: 0;
```

```
    padding: 0;
    overflow: hidden;
    background-color: #333;
}
li {
     float: left;
}
li a {
    display: block;
    color: white;
    text-align: center;
    padding: 14px 16px;
    text-decoration: none;
}
li a:hover {
    background-color: #111;
}
</style>
<ul>
    <li><a href="/Dashboard ">Dashboard</a></li>
    <li><a href="/Networking">Networking</a></li>
    <li><a href="/Storage">Storage</a></li>
    <li><a href="/Imaging">Imaging</a></li>
    <li><a href="/Compute">Compute</a></li>
```

```
                <li><a href="/Orchestration">Orchestration</a></li>
                <li><a href="/Logout">Logout</a></li>
            </ul>
            ';
    }
}
```

Then, let us get to know more about Openstack and its architecture…

6 GETTING TO KNOW ABOUT AUTOMATION IN OPENSTACK

Automation in Openstack is a kind of mechanism to make everything running automatically and created within the Openstack environment upon user request from the user interface that we will create programmatically to control most of the common functionalities of Openstack.

Throughout our book we will create a programmable user interface and a server side structure to handle our requests and instruct Openstack according to our user request to create and manipulate a private cloud according to the user's needs.

In the next section, we will be creating a session handler, to complete the user login process as follows:

```
<?php
    namespace \Session\Handler;
    class Session{
      public static function __init($email) : boolean{
         if(!isset($_SESSION)){
             session_start();
             session_regenerate_id();
             $_SESSION['email'] = md5($email);
             session_regenerate_id();
             return TRUE:
         }else{
             return FALSE;
         }
      }
      public static function __checkIfLoggedIn() : boolean{
         if(!isset($_SESSION)){
             return FALSE;
         }else{
             session_start();
             if(!empty($_SESSION['email'])){
                 return TRUE:
             }else{
                 return FALSE;
             }
         }
      }
    }
```

Now, we created a session handler to put our user into the state where the user can be logged in or not using the session in PHP7.

7 AUTOMATING OPENSTACK® USER AND ROLE MANAGEMENT

Openstack user management throughout this book will be done using PHP7 and the shell extension executed from PHP to manage openstack users and roles.

User and Role Management in Openstack is a kind of an easy task to be done through a service in openstack called LDAP or lightweight directory access protocol.

The LDAP is used from commands to be executed from the shell environment using our PHP7 engine which will be shown in the next section by creating a file called UsersController.php as the first file that will do our first automation tasks which will be creating a new user in Openstack environment:

```
"touch /etc/bitnami/apache2/htdocs/controllers/UsersController.php"
"nano /etc/bitnami/apache2/htdocs/controllers/UsersController.php"
```

Before we program our file, we need to make some commands to make our environment configured properly and ready for commands execution as follows:

```
'mysql -u root -p'
```

Then we put in the password when prompted for mysql server that we already put when installing devstack in the beginning and also before that when installing our Bitnami LAMP stack.

```
'CREATE DATABASE IF NOT EXISTS keystone;'
'GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'localhost' \
 IDENTIFIED BY 'KEYSTONE_DBPASS';'
'GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'%' \
 IDENTIFIED BY 'KEYSTONE_DBPASS';'
'nano /etc/keystone/keystone.conf'
```

Then we add the following line to our configuration file under the section called [default] As follows:

```
admin_token = ADMIN_TOKEN
verbose = True
```

Then, under the section called [database], we add the following line of code as follows:

```
connection = mysql://keystone:KEYSTONE_DBPASS@controller/keystone
```

After that, we add in the following code under the section called [memcache] and if any of these section does not exist into our file, create it maually:

```
servers = localhost:11211
```

Under the [token] section, we add in the following:

```
provider = keystone.token.providers.uuid.Provider
driver = keystone.token.persistence.backends.memcache.Token
```

Under the [revoke] section, we add in the following line of code to configure the mysql invocation driver as follows:

```
driver = keystone.contrib.revoke.backends.sql.Revoke
```

Close the configuration file and save it, then into the terminal window, run in the following command as follows:

```
su -s /bin/sh -c "keystone-manage db_sync" keystone
```

Now, edit the file located in /etc/bitnami/apache2/conf/apache2.conf and replace the original file with the following content as follows:

```
ServerName controller
```

Then create a file called /etc/apache2/sites-available/wsgi-keystone.conf as follows:

```
'touch /etc/apache2/sites-available/wsgi-keystone.conf '
'nano /etc/apache2/sites-available/wsgi-keystone.conf'
```

Then, we put into the file the following content as follows:

```
Listen 5000
Listen 35357
<VirtualHost *:5000>
      WSGIDaemonProcess keystone-public processes=5
threads=1 user=keystone display-name=%{GROUP}
      WSGIProcessGroup keystone-public
      WSGIScriptAlias / /var/www/cgi-bin/keystone/main
      WSGIApplicationGroup %{GLOBAL}
      WSGIPassAuthorization On
      <IfVersion >= 2.4>
         ErrorLogFormat "%{cu}t %M"
      </IfVersion>
      LogLevel info
```

```
      ErrorLog /var/log/apache2/keystone-error.log
      CustomLog /var/log/apache2/keystone-access.log combined
</VirtualHost>
<VirtualHost *:35357>
      WSGIDaemonProcess keystone-admin processes=5
threads=1 user=keystone display-name=%{GROUP}
      WSGIProcessGroup keystone-admin
      WSGIScriptAlias / /var/www/cgi-bin/keystone/admin
      WSGIApplicationGroup %{GLOBAL}
      WSGIPassAuthorization On
      <IfVersion >= 2.4>
         ErrorLogFormat "%{cu}t %M"
      </IfVersion>
      LogLevel info
      ErrorLog /var/log/apache2/keystone-error.log
      CustomLog /var/log/apache2/keystone-access.log combined
</VirtualHost>
```

Now, we create something called symbolic link to enable our new host as follows:

```
ln -s /etc/bitnami/apache2/sites-available/wsgi-
keystone.conf /etc/bitnami/apache2/sites-enabled
```

Then, we create a directory called keystone as follows:

```
mkdir -p /var/www/cgi-bin/keystone
```

After that, we pull in the content from the original openstack website as follows:

```
curl http://git.openstack.org/cgit/openstack/keystone/
plain/httpd/keystone.py?h=stable/kilo \
    | tee /var/www/cgi-bin/keystone/main /var/www/cgi-bin/keystone/admin
```

After all, we give the proper permission to our directory for other services to access as follows:

```
chown -R keystone:keystone /var/www/cgi-bin/keystone
chmod 755 /var/www/cgi-bin/keystone/*
```

After all, we need to restart our apache web server as follows:

service apache2 restart

Then, we remove all the un-nessessary sqlite files because, we will work using mysql server as follows:

```
rm -f /var/lib/keystone/keystone.db
```

Let us program our user's automation system as follows:

```
<?php

namespace Controllers\Users;
class UsersController{
      public static function __init($command, $role =
array(), $user, $email = NULL, $service_token, $password,
$tenant, $role, $server_name, $port_number) : boolean{
        if($command === 'createUser'){
            self::__createUser($role = array(), $user, $email
= NULL, $service_token, $password, $tenant, $role, $server_
name, $port_number) ? return TRUE : return FALSE;
        }elseif($command === 'updateUser'){
            self::__updateUser($subCommand, $user) ? return TRUE : return FALSE;
        }elseif($command === 'deleteUser'){
            self::__deleteUser($user) ? return TRUE : return FALSE;
        }elseif($command === 'getUsersList'){
            self::__getUsersList() ? return TRUE : return FALSE;
        }elseif($command === 'addUserToRole'){
            self::__addUserToRole() ? return TRUE : return FALSE;
        }
      }
      protected static function __createUser($role = array(),
$user, $email = NULL, $service_token, $password, $tenant, $role,
$server_name, $port_number, $description = NULL) : boolean{
      shell_exec('export OS_SERVICE_TOKEN=' . $token);
      shell_exec('export OS_SERVICE_ENDPOINT=http://' . $server_
name . ':' . $port_number . ' /v2.0');   shell_exec('keystone tenant-
create --name ' . $tenant . ' --description "' . $tenant . '"');
      shell_exec('keystone user-create –name ' . $user
. ' --pass ' .$password . ' --email' . $email);
      shell_exec('keystone role-create –name ' . $role);
      shell_exec('keystone user-role-add --user ' . $user
. ' --tenant ' . $tenant . ' –role ' . $role);
      return TRUE;
    }
    protected static function __updateUser($subCommand,
$user, $newName, $newEmail){
      if($subCommand === 'disableUser') shell_
exec('openstack user set ' . $user . ' --disable');
      elseif($subCommand === 'enableUser') shell_
exec('openstack user set ' . $user . ' --enable');
      elseif($subCommand === 'changeDescription') shell_exec('openstack
user set ' . $user . ' --name ' . $newName . ' –email ' . $newEmail);
      else RETURN FALSE;
    }
    protected static function __getUsersList() : boolean{
```

```
      print(shell_exec('openstack user list'));
      return TRUE;
    }
    protected static function __deleteUser($user) : boolean{
      shell_exec('openstack user delete ' .
$user) ? return TRUE : return FALSE;
    }
    protected static function __addUserToRole($userName,
$tenant, $roleName) : boolean{
      shell_exec('openstack role add --user ' . $userName . ' --project
' . $tenant . ' ' . $roleName) ? return TRUE : return FALSE;
    }
}
```

Now, we finished managing our users. Let us continue to manage roles and projects as follows:

First, we will create a file called RolesController.php as follows

```
"touch /etc/bitnami/apache2/htdocs/controllers/RolesController.php"
"nano /etc/bitnami/apache2/htdocs/controllers/RolesController.php"
```

Now, we put into the file the following code as follows:

```
<?php

namespace Controllers\Roles;

class RolesControllers{
      public static function __init($command,
$role, $userName, $tenant) : boolean{
        if($command === 'listRoles') return shell_
exec('openstack role list');
        elseif($command === 'createRole') return shell_
exec('openstack role create ' . $role);
        elseif($command === 'deleteRole') return shell_exec('openstack
role remove --user ' . $userName . ' --project ' . $tenant . ' ' . $role);
    }
}
```

Now, we created a class that can control the roles to list roles, create and delete roles.

Next, we will create a class that will control the tenants (projects) as follows:

```
"touch /etc/bitnami/apache2/htdocs/controllers/TenantsController.php"
"nano /etc/bitnami/apache2/htdocs/controllers/TenantsController.php"
```

Then, we will put in the following code into the file as follows:

```
<?php

namespace Controllers\Tenants;

class TenantsController{
      public static function __init($command, $project,
$description, $domain, $subCommand, $newName) : boolean{
      if($command === 'listProjects') shell_exec('openstack project list');
      elseif($command === 'createProject') shell_exec('openstack
project create --description "' . $description . '" ' . $project
. ' --domain' . $domain) ? return TRUE : return FALSE;
      elseif($command === 'updateProject' && $subCommand
=== 'disableProject') shell_exec('openstack project set ' .
$project . ' --disable) ? return TRUE : return FALSE;
      elseif($command === 'updateProject' && $subCommand
=== 'enableProject') shell_exec('openstack project set ' .
$project . ' --enable') ? return TRUE : return FALSE;
      elseif($command === 'updateProject' && $subCommand ===
'updateName') shell_exec('openstack project set ' . $project
. ' --name ' . $newName) ? return TRUE : return FALSE;
      elseif($command === 'deleteProject') shell_exec('openstack
project delete ' . $project) ? return TRUE : return FALSE;
      }
}
```

Now, we finished Openstack services for something called LDAP which is the lightweight directory access protocol to control users, roles, projects (tenants) and in the next section, we will work on managing images for something called Glance which is the image service as follows:

8 AUTOMATING OPENSTACK IMAGES

Let us create a file called ImagesController.php as follows:

```
"touch /etc/bitnami/apache2/htdocs/controllers/ImagesController.php"
"nano /etc/bitnami/apache2/htdocs/controllers/ImagesController.php"
```

Now, we put in the following code into our file as follows:

```
<?php

namespace Controller\Images;

class ImagesController{
      public static function __init($command) : boolean{
         if($command === 'createImage') return self::__
createImage($imageLocation, $imageName);
         elseif($command === 'setImageShortId') return
self::__setImageShortId($shortId, $imageName) ;
         elseif($command === 'createImageIso') return
self::__createImageIso($imageName, $filePath);
         elseif($command === 'getImages') return self::__getImagesList();
         elseif($command === 'deleteImage') return
self::__deleteImage($imageName);
    }
    protected static function __createImage($imageLocation,
$imageName) : boolean{
         shell_exec('openstack image create --disk-format qcow2
--container-format bare --public --file ' . $imageLocation
. ' ' . $imageName) ? return TRUE : return FALSE;
    }
    protected static function __setImageShortId($shortId,
$imageName) : boolean{
      shell_exec('openstack image set --property short-id=' .
$shortId . ' ' . $imageName) ? return FALSE : return TRUE;
    }
    protected static function __
createImageIso($imageName, $filePath) : boolean{
      shell_exec('openstack image create ' . $imageName
. ' –file ' . $filePath . '--disk-format iso --container-
format bare') ? return TRUE : return FALSE;
      }
      protected static function __getImagesList() : boolean{
```

```
        print shell_exec('openstack image list') ;
        return TRUE;
    }
    protected static function __deleteImage($imageName) : boolean{
    shell_exec('openstack image delete ' . $imageName);
  }
}
```

This way, we finished the images management, and let us go ahead and create a file to manage Openstack storage for something called cinder.

9 AUTOMATING OPENSTACK® STORAGE

Let us create a file to manage openstack storage called StorageController.php as follows:

```
"touch /etc/bitnami/apache2/htdocs/controllers/StorageController.php"
"nano /etc/bitnami/apache2/htdocs/controllers/StorageController.php"
```

Now, we put in the following code to our file that we created as follows:

```
<?php

namespace Controller\Stroage;
class StorageController{
    public static function __init($command, $keyPath, $ip,
$user, $fileType, $targetPath, $mountPoint, $volumeId,
$volumeSize, $image, $availabilityZone) : boolean{
      if($command === 'createVolume') return self::__createVolume();
      elseif($command === 'getStorageList')
return self::__getStorageList();
      elseif($command === 'attachVolumeToImage')
return self::__attachVolumeToImage();
      elseif($command === 'resizeVolume') return self::__resizeVolume();
      elseif($command === 'mountVolume) return self::__
mountVolume($keyPath, $ip, $user, $fileType, $targetPath);
      elseif($command === 'transferVolume') return
self::__transferVolume($volumeId);
      elseif($command === 'deleteVolume') return self::__deleteVolume();
    }
    protected static function __createVolume($volumeSize,
$image, $availabilityZone) : boolean{
      shell_exec('openstack volume create --image ' . $image . ' -' .
$volumeSize . ' -' . $availabilityZone) ? return TRUE : return FALSE;
    }
    protected static function __getStorageList() : boolean{
      shell_exec('openstack volume list') ? return TRUE : return FALSE;
    }
    protected static function __attachVolumeToImage($
targetPath, $volumeId, $serverId) : boolean{
      shell_exec('openstack server add volume ' . $serverId . ' ' .
$volumeId . ' -device ' . $targetPath) ? return TRUE : return FALSE;
      }
      protected static function __resizeVolume($volumeId, $volumeSize) : boolean{
```

```
        shell_exec('openstack server remove volume ' . $volumeId);
        shell_exec('openstack volume set ' .
$volumeId . ' –size ' . $size);
        return TRUE;
      }
      protected static function __mountVolume($keyPath,
$ip, $user, $fileType, $targetPath) : boolean{
        shell_exec('ssh -i ' . $keyPath . ' ' . $user . '@' . $ip);
        shell_exec('fdisk -l');
        shell_exec($fileType . ' ' . $targetPath);
        shell_exec('mkdir ' . $mountPoint);
        shell_exec('mount ' . $targetPath . ' ' . $mySpace);
      }
      protected static function __transferVolume($volumeId) : boolean{
        shell_exec('openstack volume transfer request
create ' . $volumeId) ? return TRUE : return FALSE;
      }
      protected static function __deleteVolume($volumeId) : boolean{
        shell_exec('openstack volume delete ' .
$volumeId) ? return FALSE : return TRUE;
      }
}
```

Now, we managed our volumes and explored some of the capabilities that we can do using Openstack and PHP7 to manage volumes and storage environment.

Next, we will be managing Openstack networking.

10 AUTOMATING OPENSTACK® NETWORKING AND VIRTUAL SWITCHING ENVIRONMENT

First, we will install the networking extension on UBUNTU environment on the server version as follows:

Type in the following command:

```
touch /etc/sysctl.conf
```

Then type in the following command as follows:

```
nano /etc/sysctl.conf
```

Then make sure the following configuration lines are included in the file as follows:

```
net.ipv4.ip_forward=1
net.ipv4.conf.all.rp_filter=0
net.ipv4.conf.default.rp_filter=0
```

Save the changes to the file and update the system with the new configuration using the following command:

```
sysctl -p
```

After that, let us install the neutron environment as follows:

```
apt-get install neutron-plugin-ml2 neutron-plugin-
openvswitch-agent neutron-l3-agent neutron-dhcp-agent
```

This command includes packages for UBUNTU containing open virtual switch, the DHCP which is the dynamic host configuration protocol to assign automatic IP addresses for hosts and build dynamic schema for IP addresses assigned and build automatic routing tables, etc. Also the neutron l3-agent is installed with the previous packages which will perform the configuration throughout the created public or private project networks.

Now, let us configure neutron as follows:

```
nano /etc/neutron/neutron.conf
```

Then, under the section called [DEFAULT], we put in the following configuration as follows:

```
rpc_backend = rabbit
rabbit_host = localhost
rabbit_password = YOUR_RABBIT_PASS
auth_strategy = keystone
core_plugin = ml2
service_plugins = router
allow_overlapping_ips = True
verbose = True
```

Then, under the section called [keystone_authtoken], we put in the following configuration as follows:

```
auth_uri = http://localhost:5000/v2.0
identity_uri = http://localhost:35357
admin_tenant_name = admin
admin_user = neutron
admin_password = YOUR_NEUTRON_PASS
```

Then, we will configure the open virtual switch as follows:

nano /etc/plugins/ml2/ml2_configuration/ml2_configuration.ini

After that, under the section called [ml2], we put in the following configuration as follows:

```
type_drivers = flat,gre
tenant_network_types = gre
mechanism_drivers = openvswitch
```

Next, under the section called [ml2_type_flat], we put in the following configuration as follows:

```
flat_networks = external
```

Next, under the section called [ml2_type_gre], we put in the following configuration as follows:

```
tunnel_id_ranges = 1:1000
```

Then, under the section called [securitygroup], we put in the following configuration as follows:

```
enable_security_group = True
enable_ipset = True
firewall_driver = neutron.agent.linux.iptables_
firewall.OVSHybridIptablesFirewallDriver
```

After that, under the section called [ovs], or Open Virtual Switch, we put in the following configuration as follows:

```
tunnel_types = gre
After that, we will make configuration changes to a file
called /etc/neutron/l3_agent.ini as follows:
```

Under the section called [DEFAULT], we will add the following lines:

```
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
use_namespaces = True
external_network_bridge = br-ex
router_delete_namespaces = True
verbose = True
Then, we will make changes in the configuration to the
following file /etc/neutron/dhcp_agent.ini as follows:
```

Under the section called [DEFAULT], we will put in the following configuration:

```
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
dhcp_driver = neutron.agent.linux.dhcp.Dnsmasq
use_namespaces = True
dhcp_delete_namespaces = True
verbose = True
dnsmasq_config_file = /etc/neutron/dnsmasq-neutron.conf
```

Now, we will create a file called dnsmasq-neutron.conf which will allow a process called DNS or domain name system masquerading and this file is located at:

```
/etc/neutron/dnsmasq-neutron.conf as follows:

touch /etc/neutron/dnsmasq-neutron.conf
nano /etc/neutron/dnsmasq-neutron.conf
```

After that, we will put in the following configuration into the file as follows:

```
dhcp-option-force=26,1454
```

After all, we will save the file and close, then we will execute the following command from the terminal window as follows:

```
pkill dnsmasq
```

This command is to kill any current DNS masquerading in your current LINUX installation. After that, we will edit the configuration of the metadata components in Openstack as follows: We will edit a configuration file called /etc/neutron/metadata_agent.ini using this command:

```
nano /etc/neutron/metadata_agent.ini
```

Then, under the [DEFAULT] section, we will put in the following content as follows:

```
auth_url = http://controller:5000/v2.0
auth_region = regionOne
admin_tenant_name = service
admin_user = neutron
admin_password = NEUTRON_PASS
nova_metadata_ip = controller
metadata_proxy_shared_secret = METADATA_SECRET
verbose = True
```

Then, we edit the file /etc/nova/nova.conf as follows:

```
nano /etc/nova/nova.conf
```

Then, we put in the following content under the section called [neutron] as follows:

```
service_metadata_proxy = True
metadata_proxy_shared_secret = METADATA_SECRET
```

After all, we will restart the services called nova-api and openvswitch-switch as follows:

```
service nova-api restart
service openvswitch-switch restart
```

After that, we will execute the following command to add a bridge as follows:

```
ovs-vsctl add-br br-ex
```

After all, we need to add a port to the bridge that we created by executing the following command as follows:

```
ovs-vsctl add-port br-ex eth0
```

OR

```
ovs-vsctl add-port br-ex wlan0
```

You choose whether the first command or the second command according to your Ethernet or Wireless interface that you use and replace eth0 with your Ethernet interface name or wlan0 with your wireless LAN interface name and you can recognize the interface name before running the command by running another command that lists all the interfaces as follows:

```
ifconfig
```

Then press the enter key on your keyboard to see a list of all the interfaces that is attached to the physical layer of your hardware.

After that we will restart the whole networking stack as follows:

```
service neutron-plugin-openvswitch-agent restart
service neutron-l3-agent restart
service neutron-dhcp-agent restart
service neutron-metadata-agent restart
```

Now, we have completely installed our networking stack services on our LINUX operating system that are integrated to our Openstack environment, and coming up next, working on automating the Openstack networking environment as follows:

```
<?php
namespace Controller\Networking;
class NetworkingController{
    public static function __init($command, $networkName, $projectName,
$subnetName, $dnsIp, $port, $imageName, $flavor, $vmName) : boolean{
    if($command === 'createNetwork') return self::__
createNetwork($networkName, $projectName);
      elseif($command === 'createSubnet') return
self::__createSubnet($subnetName, $address);
      elseif($command === 'listNetworks') return self::__listNetworks();
      elseif($command === 'listPorts') return self::__listPorts();
      elseif($command === 'createGlobalNetwork') return
self::__createGlobalNetwork($networkName);
      elseif($command === 'createSubnetWithIp') return self::__
createSubnetWithIp($subnetName, $networkName, $ip);
      elseif($command === 'createSubnetWithoutIp')
return self::__createSubnetWithoutIp();
      elseif($command === 'createSubnetWithoutDhcp')
return self::__createSubnetWithoutDhcp();
      elseif($command === 'createSubnetWithDns') return self::__
createSubnetWithDns($subnetName, $networkName, $dnsIp);
      elseif($command === 'disablePort') return self::__disablePort($port);
      elseif($command === 'bootImageWithNic') return self::__
bootImageWithNic($imageName, $flavor, $subnet, $vmName);
      }
      protected static function __createNetwork($networkName,
$projectName) : boolean{
         shell_exec('openstack network create --project ' .
$projectName . ' ' . $networkName) ? return TRUE : return FALSE;
      }
      protected static function __createSubnet($subnetName, $address) : boolean{
      shell_exec('openstack subnet create '
. $subnetName . ' ' . $address);
      }
      protected static function __listNetworks() : string{
         return shell_exec('openstack network list');
      }
      protected static function __listPorts() : string{
```

```
        return shell_exec('openstack port list');
      }
      protected static function __createGloba
lNetwork($networkName) : boolean{
        shell_exec('openstack network create –share '
. $networkName) ? return TRUE : return FALSE;
      }
      protected static function __createSubnetWithIp($subnetName,
$networkName, $ip) : Boolean{
        shell_exec('openstack subnet create ' . $subnetName . ' --gateway
' . $ip . ' –network ' . $networkName) ? return TRUE : return FALSE;
      }
      protected static function __createSubnetWitho
utIp($networkName, $subnetName) : boolean{
        shell_exec('openstack subnet create ' . $subnetName . ' --no-
gateway –network ' . $networkName) ? return TRUE : return FALSE;
      }
      protected static function __createSubnetWithou
tDhcp($subnetName, $networkName) : boolean{
        shell_exec('openstack subnet create ' . $subnetName . ' --network
' . $networkName . ' --no-dhcp') ? return TRUE : return FALSE;
      }
      protected static function __createSubnetWithDns(
$subnetName, $networkName, $dnsIp) : boolean{
        shell_exec('openstack subnet create ' .
$subnetName . ' --network ' . $networkName . ' --dns-
nameserver ' . $dnsIp) ? return TRUE : return FALSE;
      }
      protected static function __disablePort($port) : boolean{
        shell_exec('openstack port set ' . $port .
' --disable') ? return TRUE : return FALSE;
      }
      protected static function __bootImageWithNic($imageName,
$flavor, $subnetName, $vmName) : boolean{
        shell_exec('openstack server create --image ' .
$imageName . ' --flavor ' . $flavor . ' --nic net-id=' .
$subnetName . ' ' . $vmName) ? return TRUE : return FALSE;
      }
}
```

Now, we finished most of the topics associated with networking, but one more thing is left which is: Orchestration and in our next section, we will be working on heat orchestration templates, how to create images, configuration, storage, networking and all of our previous components that we made using one and one only file to automate the installation of our content and services that we need for our devstack or openstack installation.

11 AUTOMATING OPENSTACK ORCHESTRATION USING HEAT ORCHESTRATION TEMPLATES

Orchestration is the operation of creating stacks and managing the stacks using a kind of files called heat orchestration templates.
The heat orchestration templates will be created in our LINUX environment, then we will create a file to handle the operations of creating a stack, updating stacks, and get information about stacks as follows:

Lets first create our .yaml heat orchestration template file as follows:

```
"touch heat_orchestration_template.yaml"
"nano heat_orchestration_template.yaml"
```

Now, let us put in the following content into our file as follows:
heat_template_version: 2015-04-30

description: Simple template to deploy a single compute instance

```
resources:
    my_instance:
      type: OS::Nova::Server
      properties:
         key_name: my_key
         image: F18-x86_64-cfntools
         flavor: m1.small
```

Now let us create a file called OrchestrationController.php in our /etc/bitnami/apache2/ htdocs/controllers/ directory as follows:

```
"touch /etc/bitnami/apache2/htdocs/controllers/OrchestrationController.php"
"nano /etc/bitnami/apache2/htdocs/controllers/OrchestrationController.php"
```

Then we will put in the following content into our file as follows;

```
<?php

namespace Controller\Orchestration;
class OrchestrationController{
    public static function __init($command, $hotFilePath, $imageName, $stackName){
        if($command === 'createStack') return self::__
createStack($hotFilePath, $imageName, $stackName);
        elseif($command === 'listStacks') return self::__getStackList();
        elseif($command === 'updateStack') return self::__
updateStack($hotFilePath, $imageName, $stackName);
      }
      protected static function __createStack($hotFilePath,
$imageName, $stackName) : boolean{
      shell_exec('openstack stack create --template ' .
$hotFilePath . ' --parameter "image=' . $imageName . '"
' . $stackName) ? return TRUE : return FALSE;
      }
      protected static function __getStackList() : string{
         return shell_exec('openstack stack list');
      }
      protected static function __updateStack($hotFilePath,
```

```
$imageName, $stackName) : boolean{
        shell_exec('openstack stack update --template '
. $hotFilePath . ' --parameter "image=' . $imageName . '
" ' . $stackName) ? return TRUE : return FALSE;
      }
}
```

This way, we finished most of the important and basic operations of handling Openstack environment and we created files to handle all these operations in php.

Now, we learn how to run our files as follows:

```
php file_path.php
```

Or we run everything throughout the browser by navigating to the file directly as follows:

```
http://localhost/controllers/ControllerName.php
```

And both steps can be run after adding the following line of code at the end of the file as follows:

```
\namespace\subnamespace\classname::functionname(
parameter1, parameter2, parameter3);
```

Then saving the file before running our php filename command.

Thank you so much for reading
Thank you Manon Niazi
Thank you Sir Eng. Manuel Lemos