# Practice M7: Databases, Web Servers and Web Applications (Ubuntu)

\* NOTE: Most of the exercises included in this practice guide are **not suitable** for execution in **WSL** or **Docker** environment. It is recommended a virtual infrastructure to be used instead.

For the purpose of the current practice, we will need one virtual machine. It could be with either **CentOS 8.x**, **openSUSE Leap 15.x**, or **Ubuntu 20.04**. Of course, another version of the listed or another distribution can be used, but there can be some differences.

All commands, that we will use until the end of this practice, will be accompanied by the appropriate prompt. This way it will be easier for us to know which user, in which folder, and on which machine is executing the command.

Next steps will be executed on **Ubuntu** machine. If there is a big difference in the way we issue the command, or in its result between distributions, it will be stated clearly.

## Part 1 (Apache Web Server)

If we do not have **unzip**, **pstree**, and **tree** installed, we can do it with:

lsauser@ubuntu:~$ **sudo apt-get update**

...

lsauser@ubuntu:~$ **sudo apt-get install -y unzip psmisc tree**

Now, we can install the web server. The package is named **httpd** in **CentOS** and **apache2** in **openSUSE** and **Ubuntu**.

### Installation

Installation steps are the following:

lsauser@ubuntu:~$ **sudo apt-get install -y apache2**

...

lsauser@ubuntu:~$ **systemctl status apache2**

...

As we can see, the service is enabled to start on boot and is started already

### Processes and Configuration

We can check the process tree for the web server with:

lsauser@ubuntu:~$ **pstree -psT $(pgrep -U 0 apache2)**

systemd(1)───apache2(1496)─┬─apache2(1498)

└─apache2(1499)

lsauser@ubuntu:~$

Then we can examine the configuration files and folders hierarchy with:

lsauser@ubuntu:~$ **tree /etc/apache2**

/etc/apache2

├── apache2.conf

├── conf-available

│   ├── charset.conf

│   ├── localized-error-pages.conf

│   ├── other-vhosts-access-log.conf

│   ├── security.conf

│   └── serve-cgi-bin.conf

├── conf-enabled

│   ├── charset.conf -> ../conf-available/charset.conf

...

├── ports.conf

├── sites-available

│   ├── 000-default.conf

│   └── default-ssl.conf

└── sites-enabled

└── 000-default.conf -> ../sites-available/000-default.conf

6 directories, 187 files

lsauser@ubuntu:~$

Current configuration could be checked with:

lsauser@ubuntu:~$ **apachectl configtest**

AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.1.1. Set the 'ServerName' directive globally to suppress this message

Syntax OK

lsauser@ubuntu:~$

If we receive a message like the one above, it does not mean that the service is not working, but that it has difficulties to resolve the fully qualified domain name (**FQDN**) of our machine. We can tackle the issue by adding it (or editing it) either in the main configuration file or the configuration file for the default virtual host. Let’s go with the first one as it is the preferred way:

lsauser@ubuntu:~$ **sudo vi /etc/apache2/apache2.conf**

...

# Global configuration

#

**ServerName ubuntu.lsa.lab**

...

Then, we must add a corresponding line in our **/etc/hosts** file:

lsauser@ubuntu:~$ **echo '10.0.2.15 ubuntu.lsa.lab ubuntu' | sudo tee -a /etc/hosts**

Where **10.0.2.15** should be changed to our **IP** address.

Now, we can re-execute the check and **restart the web service** eventually.

### Local Access

Next, we can try to open our default web page locally. For this, we can install a command line browser. Some of the most popular ones are **lynx**, **links**, **elinks**, and **w3m**. Let’s install one and navigate to our page:

lsauser@ubuntu:~$ **sudo apt-get install -y lynx**

...

lsauser@ubuntu:~$ **lynx http://ubuntu**

We can quit with the **q** key.

Alternative and very common way of testing web pages is the **curl** command:

lsauser@ubuntu:~$ **curl http://ubuntu**

### Remote Access

We can test our page from the host as well, but first we should take care of a few things.

First, we must open a port in the firewall if running:

lsauser@ubuntu:~$ **sudo ufw allow 80/tcp**

Rule added

Rule added (v6)

lsauser@ubuntu:~$

Then we must check how our virtual network adapter was set up. If it is in **Bridge** mode, then we must check what it our **IP** address. Then open a browser tab and navigate to **http://[ip-of-our-vm]**

If we are in **NAT** mode, then we must set up a port forwarding rule. For example, port **8000** on the host to point to port **80** of the **VM**. Then open a browser tab and navigate to [**http://localhost:8000**](http://localhost:8000)

It is time to change the default web page to something custom. Let’s create a tiny static web page:

lsauser@ubuntu:~$ **echo '<h1>My test site! :)</h1>' | sudo tee /var/www/html/index.htm**l

<h1>My test site! :)</h1>

lsauser@ubuntu:~$ **lynx** [**http://centos**](http://centos)

### Deploy a Web Page

Let’s go to the next level. We will deploy a set of files that belong to a web page.

For this we can either download the archive on our host, and then with the help of the **scp** (or **WinSCP** if we are on **Windows**) command to copy it on the **VM** or use a command line tool to download it directly on the **VM**.

In any case the archive is available on the page of the current module and via a direct link: <https://zahariev.pro/files/lsa-m7-practice.zip>

Should we choose the first option, we must execute something like:

[dimitar@lenovo-e450-dz ~]$ **scp -P 30001 lsa-m7-practice.zip lsauser@localhost:.**

Of course, we should adjust both **the port and the path**.

As stated above, we can do all the tasks while on the **VM**. For this purpose, we can download the archive either with:

lsauser@ubuntu:~$ **curl -o lsa-m7-practice.zip** [**https://zahariev.pro/files/lsa-m7-practice.zip**](https://zahariev.pro/files/lsa-m7-practice.zip)

Or with:

lsauser@ubuntu:~$ **wget** [**https://zahariev.pro/files/lsa-m7-practice.zip**](https://zahariev.pro/files/lsa-m7-practice.zip)

Now, that we have the archive, we can first check what it contains with:

lsauser@ubuntu:~$ **unzip -l lsa-m7-practice.zip**

Archive: lsa-m7-practice.zip

Length Date Time Name

--------- ---------- ----- ----

281 01-16-2018 09:39 index.html

78024 10-23-2017 19:25 wizard.png

--------- -------

78305 2 files

lsauser@ubuntu:~$

Then extract it in the target folder:

lsauser@ubuntu:~$ **sudo unzip lsa-m7-practice.zip -d /var/www/html**

Because in the target folder exists file with name **index.html**, we will be asked if we want to overwrite it. If we are sure in our actions since the beginning, we can execute a modified version of the command. It will overwrite files in the target directory without asking:

lsauser@ubuntu:~$ **sudo unzip -o lsa-m7-practice.zip -d /var/www/html**

Now, we can check the result either locally or on our host.

### Virtual Hosts

We can end up in situation when there is a need to serve more than one site on a single host. There are many ways to solve this task. One viable option is to use the virtual hosts. There are slight differences how this is achieved amongst different distributions.

Let’s adjust the existing configuration file **/etc/apache2/sites-available/000-default.conf** for the default and only virtual host:

lsauser@ubuntu:~$ **sudo vi /etc/apache2/sites-available/000-default.conf**

<VirtualHost \*:80>

...

**ServerName ubuntu.lsa.lab**

ServerName "centos.lsa.lab"

DocumentRoot "/var/www/html"

...

</VirtualHost>

You should change the server’s name value to match yours.

Now, we can test the changes with:

lsauser@ubuntu:~$ **apachectl configtest**

If everything is okay, we can make a copy of the configuration file and change it a bit. Instead, let’s create a brand-new file:

lsauser@ubuntu:~$ **sudo vi /etc/apache2/sites-available/001-secret.conf**

<VirtualHost \*:80>

ServerName secret.lsa.lab

DocumentRoot /var/www/secret

</VirtualHost>

Then, enable the new virtual host by executing:

lsauser@ubuntu:~$ **sudo a2ensite 001-secret.conf**

Enabling site 001-secret.

To activate the new configuration, you need to run:

systemctl reload apache2

lsauser@ubuntu:~$

Do not reload the **apache2** service now. Instead, let’s check the configuration again:

lsauser@ubuntu:~$ **apachectl configtest**

AH00112: Warning: DocumentRoot [/var/www/secret] does not exist

Syntax OK

lsauser@ubuntu:~$

It appears that we must create the folder in question. Let’s do it and then check the configuration again:

lsauser@ubuntu:~$ **sudo mkdir /var/www/secret**

lsauser@ubuntu:~$ **apachectl configtest**

Syntax OK

lsauser@ubuntu:~$

We can create a simple html page:

lsauser@ubuntu:~$ **echo "<h1>Secret Web Site</h1>" | sudo tee /var/www/secret/index.html**

<h1>Secret Web Site</h1>

lsauser@ubuntu:~$

Let’s restart the web server:

lsauser@ubuntu:~$ **sudo systemctl restart apache2**

We must add a row in our **/etc/hosts** file:

lsauser@ubuntu:~$ **echo '127.0.0.1 secret.lsa.lab secret' | sudo tee -a /etc/hosts**

127.0.0.1 secret.lsa.lab secret

lsauser@ubuntu:~$

And finally, let’s test both virtual hosts locally:

lsauser@ubuntu:~$ **curl http://ubuntu.lsa.lab**

<html>

<head>

<title>Congratulations!</title>

</head>

<body>

<div align="center">

<h1>Congratulations You Made It!</h1>

<img src="wizard.png" />

<h1>May the force be with you on your journey as Linux Administrator!</h1>

</div>

</body>

</html>

lsauser@ubuntu:~$ **curl http://secret.lsa.lab**

<h1>Secret Web Site</h1>

lsauser@ubuntu:~$

### **Logs**

Let’s change the default log configuration for one of the virtual hosts:

lsauser@ubuntu:~$ **sudo vi /etc/apache2/sites-available/001-secret.conf**

<VirtualHost \*:80>

ServerName secret.lsa.lab

DocumentRoot /var/www/secret

**ErrorLog ${APACHE\_LOG\_DIR}/secret.log**

**CustomLog ${APACHE\_LOG\_DIR}/secret.log combined**

</VirtualHost>

Should we want to learn more about logs in general, we can explore the main configuration file:

lsauser@ubuntu:~$ **cat /etc/apache2/apache2.conf | grep -B 1 -A 1 -i logformat**

#

LogFormat "%v:%p %h %l %u %t \"%r\" %>s %O \"%{Referer}i\" \"%{User-Agent}i\"" vhost\_combined

LogFormat "%h %l %u %t \"%r\" %>s %O \"%{Referer}i\" \"%{User-Agent}i\"" combined

LogFormat "%h %l %u %t \"%r\" %>s %O" common

LogFormat "%{Referer}i -> %U" referer

LogFormat "%{User-agent}i" agent

lsauser@ubuntu:~$

We can test the configuration and if everything is okay, we must restart the service:

lsauser@ubuntu:~$ **apachectl configtest**

Syntax OK

lsauser@ubuntu:~$ **sudo systemctl restart apache2**

Now, we can ask for the access log, then access the site, and finally ask for the log again:

lsauser@ubuntu:~$ **sudo cat /var/log/apache2/secret.log**

cat: /var/log/apache2/secret.log: No such file or directory

lsauser@ubuntu:~$ **curl http://secret.lsa.lab**

...

lsauser@ubuntu:~$ **sudo cat /var/log/apache2/secret.log**

127.0.0.1 - - [18/Jun/2021:12:06:51 +0000] "GET / HTTP/1.1" 200 252 "-" "curl/7.68.0"

lsauser@ubuntu:~$

### Access Control by IP

We can open the configuration file of one of the virtual hosts and add a restriction by IP address:

lsauser@ubuntu:~$ **sudo vi /etc/apache2/sites-available/000-default.conf**

...

**<Location />**

**Require ip 127.0.0.1**

**</Location>**

...

Save and close the file. Now test the configuration and then restart the service:

lsauser@ubuntu:~$ **apachectl configtest**

Syntax OK

lsauser@ubuntu:~$ **sudo systemctl restart apache2**

Now, we can try to access the site locally both on the localhost address and the external address. The first try must succeed, while the second should fail.

### Access Control by Users

Now, let’s change the configuration of the second virtual host:

lsauser@ubuntu:~$ **sudo vi /etc/apache2/sites-available/001-secret.conf**

<VirtualHost \*:80>

ServerName secret.lsa.lab

DocumentRoot /var/www/secret

ErrorLog ${APACHE\_LOG\_DIR}/secret.log

CustomLog ${APACHE\_LOG\_DIR}/secret.log combined

**<Directory /var/www/secret>**

**Require valid-user**

**AuthType Basic**

**AuthName "Private Access"**

**AuthBasicProvider file**

**AuthUserFile /etc/apache2/secret**

**</Directory>**

</VirtualHost>

Now, we must save and close the file. Then check the configuration and restart the service:

lsauser@ubuntu:~$ **apachectl configtest**

Syntax OK

lsauser@ubuntu:~$ **sudo systemctl restart apache2**

Before we can test the secured access to our resources, we must create two users:

lsauser@ubuntu:~$ **sudo htpasswd -c /etc/apache2/secret john**

New password: **12345**

Re-type new password: **12345**

Adding password for user john

lsauser@ubuntu:~$ **sudo htpasswd /etc/apache2/secret jane**

New password: **qwerty**

Re-type new password: **qwerty**

Adding password for user jane

Please note, that the passwords won’t be displayed on the screen.

If we are curios, we can check what is inside the secret file:

lsauser@ubuntu:~$ **cat /etc/apache2/secret**

john:$apr1$TyQmtE43$ozVKcS0ZumoAlzqHLqU7n1

jane:$apr1$rbaWsVZN$PtF41XwxSJwHxlnq85WmS1

lsauser@ubuntu:~$

Finally, we can test if our security mechanism is working as expected:

lsauser@ubuntu:~$ **lynx http://secret.lsa.lab**

## Part 2 (MariaDB)

### Installation

Let’s install one of the most common relational databases under Linux – **MariaDB**:

lsauser@ubuntu:~$ **sudo apt-get update**

...

lsauser@ubuntu:~$ **sudo apt-get install -y mariadb-server mariadb-client**

The corresponding packages in **openSUSE** are named **mariadb** and **mariadb-client**, while in **Ubuntu** they are **mariadb-server** and **mariadb-client**.

After the installation is finished, the service is automatically started and enabled to start on boot. We can check its status:

lsauser@ubuntu:~$ **systemctl status mariadb**

...

It is a good practice to strengthen the security a bit of a new installation:

lsauser@ubuntu:~$ **sudo mysql\_secure\_installation**

### Work with the Database

As with other client-server solutions, here there is also a client tool, which is named **mysql** (still).

Let’s try to connect to the database:

lsauser@ubuntu:~$ **sudo** **mysql -u root -p**

...

MariaDB [(none)]>

### User and Database Creation

It is time to create an empty database. Let’s create one, named employees:

**create database employees;**

We must restrict the access to the database. For this purpose, we will create a user:

**grant all on employees.\* to user@localhost identified by 'Password1';**

...

**flush privileges;**

### Our First Table

What is a database without its tables? Let’s create one. It will store our employees:

**USE employees;**

**CREATE TABLE persons (**

**id INT,**

**department\_id INT,**

**person\_name VARCHAR(50),**

**person\_sex CHAR(1),**

**person\_salary DECIMAL(15, 3)**

**);**

Having an empty table is not a big deal. Let’s add few records:

**INSERT INTO persons (id, department\_id, person\_name, person\_sex, person\_salary)**

**VALUES (1, 1, 'John Smith', 'M', 5000),**

**(2, 2, 'Tony James', 'M', 3000),**

**(3, 3, 'Kate Hudson', 'F', 1500),**

**(4, 1, 'Ann Evans', 'F', 3500);**

### Work with Data

Now, that we have a table with some data, it is time to ask questions against it. For example, we can ask for the identification number and person name of all records:

**SELECT id, person\_name**

**FROM persons;**

Then we can change our question to – give me identification number and person name of all women:

**SELECT id, person\_name**

**FROM persons**

**WHERE person\_sex='F';**

Even though there are only few rows, we can start experimenting. We can add more rows if needed.

### Second Table

It would be good to have a second table with some data. This way, could test our knowledge about multi table queries. Let’s create one named departments:

**CREATE TABLE departments (**

**id INT,**

**department\_name VARCHAR(25)**

**);**

Now, let’s add few records:

**INSERT INTO departments (id, department\_name) VALUES**

**(1, 'IT'),**

**(2, 'Accounting'),**

**(3, 'Sales');**

### Combine and Update Data

Even though it is not considered a good practice when querying for data, we will break the rules, and will construct a query, that will answer a question like – give me everything about all rows in the table:

**SELECT \***

**FROM departments;**

Now, let’s combine information from both tables. We will create a list of all employees and the department to which they belong:

**SELECT p.person\_name, d.department\_name**

**FROM persons p**

**INNER JOIN departments d**

**ON p.department\_id = d.id;**

Sooner or later, we will find ourselves in a need to update records. Let’s do it now. We will increase the salary of all women with 2000 units:

**UPDATE persons**

**SET salary=salary+2000**

**WHERE person\_sex = 'F';**

We can check what we did:

**SELECT \***

**FROM persons;**

### Backup and Restore

Let’s try a simple backup-restore routing, but first we must close our session to the database:

**quit**

Now, in order to create a backup of our database, we must execute:

lsauser@ubuntu:~$ **sudo** **mysqldump -u root -p employees > employees.sql**

Enter password:

lsauser@ubuntu:~$ **ls -al employees.sql**

-rw-rw-r--. 1 lsauser lsauser 2775 19 мар 22,31 employees.sql

[lsauser@centos ~]$

In order to test the backup, we must connect again to the database and delete something:

lsauser@ubuntu:~$ **sudo** **mysql -u root -p**

**USE employees;**

**DELETE FROM persons;**

Close again the session with:

**quit**

It is time to check if we can restore the database as it was before the deletion:

lsauser@ubuntu:~$ **sudo** **mysql -u root -p employees < employees.sql**

...

And the final proof:

lsauser@ubuntu:~$ **sudo** **mysql -u root -p**

**USE employees;**

**SELECT \***

**FROM persons;**

**quit**

## Part 3 (PHP + Apache HTTP + MariaDB)

The only missing **LAMP** component so far is the **PHP**. We will add it during this part.

### PHP Installation

Let’s first install **PHP** itself and one library used to connect from **PHP** to **MySQL/MariaDB** and then restart the web server:

lsauser@ubuntu:~$ **sudo apt-get install -y php php-mysql**

...

lsauser@ubuntu:~$ **sudo systemctl restart apache2**

Before we continue further, we can check if everything with the PHP installation is okay:

lsauser@ubuntu:~$ **php -i**

Then, we can test the integration with the web server:

lsauser@ubuntu:~$ **sudo vi /var/www/html/index.php**

**<h1>Current PHP configuration:</h1>**

**<?php**

**phpinfo();**

**?>**

lsauser@ubuntu:~$ **lynx http://localhost/index.php**

### Combine PHP with MariaDB

It won’t be a good practice without a working example combining **PHP** with **MariaDB**. We will prepare a script, which will list all employees and the departments they belong to.

The task can be achieved in many ways. For our approach, we will use this script:

lsauser@ubuntu:~$ **sudo vi /var/www/secret/index.php**

<!DOCTYPE HTML>

<html>

<head>

<title>Employees List</title>

</head>

<body>

<h1>Employees List</h1>

<?php

// variables initialization

$host="localhost";

$user="user";

$password="Password1";

$db="employees";

// establish a connection

if ($dbh=mysqli\_connect($host, $user, $password, $db)) {

// initialize the query statement

$query="SELECT p.person\_name, d.department\_name FROM persons p LEFT JOIN departments d ON p.department\_id=d.id";

if ($result=mysqli\_query($dbh, $query)) {

// we have data, so let's list it

echo "<ol>";

while ($row=mysqli\_fetch\_array($result, MYSQLI\_ASSOC)) {

echo "<li>".$row['person\_name'].", ".$row['department\_name']."</li>";

}

echo "</ol>";

}

else

echo "<h2>ERROR: No data!</h2>";

}

else

echo "<h2>ERROR: No connection!</h2>";

?>

</body>

</html>

Save and close the file.

Now, it is time to test what we produced:

lsauser@ubuntu:~$ **lynx http://secret.lsa.lab/index.php**

## Part 4 (Web Interfaces)

### Cockpit

This is one of the available solutions for remote web management of Linux servers.

There are different installation options. We will go with the easiest – we will install it like an ordinary package:

lsauser@ubuntu:~$ **sudo apt-get update**

...

lsauser@ubuntu:~$ **sudo apt-get install -y cockpit**

When the installation is finished, we can test it with:

<https://ip-address-of-machine:9090>

### Webmin

This is one of the oldest solutions with rich set of functionalities. As usual, there are multiple ways for installation.

**Option 1:**

Download the package:

lsauser@ubuntu:~$ **wget http://prdownloads.sourceforge.net/webadmin/webmin\_1.979\_all.deb**

Then install required dependencies:

lsauser@ubuntu:~$ **sudo** **apt-get install perl libnet-ssleay-perl openssl libauthen-pam-perl libpam-runtime libio-pty-perl apt-show-versions python unzip**

And then install it:

lsauser@ubuntu:~$ **sudo dpkg --install webmin\_1.979\_all.deb**

**Option 2:**

We can add a new repository and then install it like a standard package. For this to happen, we must edit the /etc/apt/sources.list file and add the following line:

**deb https://download.webmin.com/download/repository sarge contrib**

Then download and import the key:

lsauser@ubuntu:~$ **wget https://download.webmin.com/jcameron-key.asc**

lsauser@ubuntu:~$ **sudo** **apt-key add jcameron-key.asc**

And finally install it:

lsauser@ubuntu:~$ **sudo apt-get update**

...

lsauser@ubuntu:~$ **sudo apt-get install -y webmin**

Now, we can test it, by visiting the following address:

<https://ip-address-of-machine:10000/>

You must use the **root** user or another user with **sudo** privileges to login