# Practice M4: Network Services (Ubuntu)

This practice assumes that you are working in an on-premise environment

All tasks can be achieved under different configurations (host OS and/or virtualization solution) with the appropriate adjustments

This practice is oriented towards **Ubuntu Server 21.04** and other similar distributions

The infrastructure will vary during the practice but in its most complete stage will include up to three machines:

Chart, waterfall chart

Description automatically generated

## Part 1: Web Servers. Printing Services

For this part we will need an infrastructure with all three machines

Machines can be with or without graphical environment

Network settings shown on the picture reflect the ones, used during the demonstration. You should adjust them according to your setup

### Apache

Let us log on to the **M1** machine with a regular user account

#### Installation and Configuration

The installation of **Apache** is straight forward. Execute

**sudo apt-get update**

**sudo apt-get install apache2**

*Basically, we can also install the* ***tree*** *command to explore the* ***/etc/apache2*** *hierarchy*

Once, it is installed, we can explore the main configuration file **/etc/apache2/apache2.conf**

And then the one for the default site - **/etc/apache2/sites-available/000-default.conf**

More information on the core **Apache** features can be found here: <https://httpd.apache.org/docs/2.4/mod/core.html>

Check the configuration with

**apachectl configtest**

Check the status of the **apache2** service

**systemctl status apache2**

And if not running, enable and start it

**sudo systemctl enable --now apache2**

Don’t forget to open the appropriate port in the firewall if running

**sudo ufw allow "Apache Full"**

Set a custom **index.html** file

**echo '<h1>Hello from M1</h1>' | sudo tee /var/www/html/index.html**

And finally, test it with

**curl http://localhost**

#### Virtual Hosts

Quite often we would want to have multiple sites on one server. One way to achieve this is to use **virtual hosts**

In fact, the default site is already defined using virtual hosts

##### Virtual Hosts (by port)

Create a file **/etc/apache2/sites-available/001-vhost-port.conf** with the following content

**Listen 8080**

**<VirtualHost \*:8080>**

**DocumentRoot /var/www/vhost-port**

**ServerName m1.lsaa.lab**

**</VirtualHost>**

Save and close the file

The **Listen 8080** instruction can be placed in the main configuration file instead in this one

##### Virtual Hosts (by name)

Create a file **/etc/apache2/sites-available/002-vhost-name.conf** with the following content

**<VirtualHost \*:80>**

**DocumentRoot /var/www/vhost-name**

**ServerName www.demo.lab**

**ServerAdmin admin@demo.lab**

**ErrorLog /var/log/apache2/vhost-name-error.log**

**CustomLog /var/log/apache2/vhost-name-access.log combined**

**</VirtualHost>**

Save and close the file

##### Finalization

Create the corresponding **DocumentRoot** folders

**sudo mkdir /var/www/vhost-{name,port}**

Crete two new **index.html** files

**echo '<h1>Hello from vhost by port</h1>' | sudo tee /var/www/vhost-port/index.html**

And

**echo '<h1>Hello from vhost by name</h1>' | sudo tee /var/www/vhost-name/index.html**

Enable the two new sites (virtual hosts)

**sudo a2ensite 001-vhost-port.conf**

**sudo a2ensite 002-vhost-name.conf**

Test the new configuration with

**apachectl configtest**

Restart the **apache2** service

**sudo systemctl restart apache2**

Open port **8080/tcp** in the firewall if running

**sudo ufw allow 8080/tcp**

Add new record in the **/etc/hosts** file

**echo '192.168.81.151 www.demo.lab www' | sudo tee -a /etc/hosts**

Finally, test both virtual hosts with

**curl http://localhost:8080**

**curl http://www.demo.lab**

More information and samples about **Apache** virtual hosts can be found here: <https://httpd.apache.org/docs/2.4/vhosts/examples.html>

#### TLS/SSL

We can use certificates issued from a trusted certificate authority or create self-signed certificate

As this is a demo, let us create a self-signed certificate

##### Preparation

Install the necessary packages with

**sudo apt-get update**

**sudo apt-get install openssl**

Generate the private key

**openssl genrsa -out ca.key 2048**

Create a certificate signing request (CSR)

**openssl req -new -key ca.key -out ca.csr**

Generate the self-signed certificate

**openssl x509 -req -days 365 -in ca.csr -signkey ca.key -out ca.crt**

We can we the result with

**openssl x509 -text -in ca.crt**

Copy the files to the appropriate folders

**sudo cp ca.crt /etc/ssl/certs/ca.crt**

**sudo cp ca.key /etc/ssl/private/ca.key**

**sudo cp ca.csr /etc/ssl/private/ca.csr**

##### Apache settings

Explore the SSL module configuration file

**vi /etc/apache2/mods-available/ssl.conf**

Enable the module (and all its dependencies)

**sudo a2enmod ssl**

Create an SSL enabled default site (virtual host)

**sudo vi /etc/apache2/sites-available/000-default-ssl.conf**

With the following content

**<IfModule mod\_ssl.c>**

**<VirtualHost \_default\_:443>**

**ServerAdmin webmaster@localhost**

**DocumentRoot /var/www/html**

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

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

**SSLEngine on**

**SSLCertificateFile /etc/ssl/certs/ca.crt**

**SSLCertificateKeyFile /etc/ssl/private/ca.key**

**<FilesMatch "\.(cgi|shtml|phtml|php)$">**

**SSLOptions +StdEnvVars**

**</FilesMatch>**

**<Directory /usr/lib/cgi-bin>**

**SSLOptions +StdEnvVars**

**</Directory>**

**</VirtualHost>**

**</IfModule>**

Enable the SSL version of the site

**sudo a2ensite 000-default-ssl.conf**

Test the configuration this time prefixed with sudo (to be able to read the certificates)

**sudo apachectl configtest**

Restart the **apache2** service

**sudo systemctl restart apache2**

Ask for the status

**systemctl status apache2**

You can see that **Apache** is listening on port **443** as well with

**ss -ntpl**

Test the default site with

**curl** [**https://localhost**](https://localhost)

Because the certificate is self-signed you will see an error. Try again with

**curl -k** [**https://localhost**](https://localhost)

Now, you should see our default site

##### Final touches

We must open the appropriate port in the firewall if running and if not used "Apache Full"

**sudo ufw allow "Apache Secure"**

Should we want to be automatically redirected to **https** when visiting **http**, we can modify the virtual host configuration for our default site

**sudo vi /etc/apache2/sites-available/000-default.conf**

Add the following three lines just before the closing **</VirtualHost>**

**RewriteEngine on**

**RewriteCond %{HTTPS} off**

**RewriteRule ^(.\*)$ https://%{HTTP\_HOST}%{REQUEST\_URI} [R=301,L]**

Save and close the file

Enable the appropriate module

**sudo a2enmod rewrite**

Check the configuration

**sudo apachectl configtest**

Restart the **apache2** service

**sudo systemctl restart apache2**

Open a browser tab on the host and navigate to **http://<m1-ip>/**

A warning should appear, accept it. Now, you should see our default site

More information about the **mod\_rewrite** module of **Apache** can be found here: <https://httpd.apache.org/docs/2.4/mod/mod_rewrite.html>

#### PHP

Let us install the necessary packages

**sudo apt-get update**

**sudo apt-get install php**

Once the installation is complete, we can check with

**php -v**

Then we can restart the **apache2** service

**sudo systemctl restart apache2**

Remove the existing **index.html** file

**sudo rm /var/www/html/index.html**

Create a new **index.php** file with

**echo '<?php phpinfo(); ?>' | sudo tee /var/www/html/index.php**

Do test with either **curl** locally or a with browser tab on the host

#### Reverse Proxy

##### Preparation

Log on to the **M2** machine and install **apache2**

**sudo apt-get update**

**sudo apt-get install apache2**

Then create a custom **index.html** page

**echo '<h1>Hello from M2</h1>' | sudo tee /var/www/html/index.html**

And last, but not least, don’t forget to adjust the firewall if running

**sudo ufw allow "Apache Full"**

Don’t forget to test if locally everything is okay

**curl http://localhost**

##### Configuration

Return to the **M1** machine

Enable the proxy module with

**sudo a2enmod proxy**

**sudo a2enmod proxy\_http**

Now, let us create a configuration for the reverse proxy module

**sudo vi /etc/apache2/conf-available/reverse-proxy.conf**

Enter the following

**<IfModule mod\_proxy.c>**

**ProxyRequests Off**

**<Proxy \*>**

**Require all granted**

**</Proxy>**

**ProxyPass / http://m2.lsaa.lab/**

**ProxyPassReverse / http://m2.lsaa.lab/**

**</IfModule>**

Save and close the file

Enable the configuration

**sudo a2enconf reverse-proxy**

Test the configuration

**sudo apachectl configtest**

Restart the **apache2** service

**sudo systemctl restart apache2**

Test by opening a browser tab on the host and navigating to **https://<m1-ip>/**

You should see the index page of **M2**

More information about the **mod\_proxy** module of **Apache** can be found here: <https://httpd.apache.org/docs/2.4/mod/mod_proxy.html>

#### Load Balancing

##### Preparation

Log on to the **M3** machine and install **apache2**

**sudo apt-get update**

**sudo apt-get install apache2**

Then create a custom **index.html** page

**echo '<h1>Hello from M3</h1>' | sudo tee /var/www/html/index.html**

And last, but not least, don’t forget to adjust the firewall if running

**sudo ufw allow "Apache Full"**

Don’t forget to test locally the site

**curl http://localhost**

##### Configuration

We are one step away from turning our setup into a fully functional load balancing solution

Return on **M1**

Open the proxy module configuration file

**sudo vi /etc/apache2/conf-available/reverse-proxy.conf**

And modify the file by deleting the **ProxyPass** and **ProxyPassReverse** lines

Then enter the following

**ProxyPass / balancer://demo/**

**ProxyPassReverse / balancer://demo/**

**<Proxy balancer://demo>**

**BalancerMember http://m2.lsaa.lab**

**BalancerMember http://m3.lsaa.lab**

**ProxySet lbmethod=bytraffic**

**</Proxy>**

Save and close the file

Enable the appropriate modules

**sudo a2enmod proxy\_balancer**

**sudo a2enmod lbmethod\_bytraffic**

Check the configuration

**sudo apachectl configtest**

Restart the **apache2** service

**sudo systemctl restart apache2**

Test by opening a browser tab on the host and navigating to **https://<m1-ip>/**

Refresh a few times. Everything should work as expected

More information about the **mod\_proxy** module of **Apache** can be found here: <https://httpd.apache.org/docs/2.4/mod/mod_proxy.html>

And about the load balancing extension, here:

<https://httpd.apache.org/docs/2.4/mod/mod_proxy_balancer.html>

### NGINX

We can continue with the same infrastructure, just reset the machines. Alternatively, spin up a new set

Let us log on to the **M1** machine with a regular user account

#### Installation and Configuration

The installation of **NGINX** is straight forward, execute

**sudo apt-get update**

**sudo apt-get install nginx**

Once, it is installed, we can check the main configuration file

**sudo vi /etc/nginx/nginx.conf**

Now, we can modify a little bit the configuration file for the default site

**sudo vi /etc/nginx/sites-available/default**

We can adjust the **server\_name** (on line 46) and set it to, for example to **m1.lsaa.lab** or leave as it is

Save and close the file

Test the **NGINX** configuration with

**sudo nginx -t**

Then, we can change the default (welcome) web page

Execute the following to change it

**echo '<h1>Hello from M1</h1>' | sudo tee /var/www/html/index.html**

Restart the **nginx** service

**sudo systemctl restart nginx**

Check the status of the **nginx** service

**systemctl status nginx**

Don’t forget to open the appropriate port in the firewall if running

**sudo ufw allow "Nginx Full"**

And finally, test with

**curl http://localhost**

#### Virtual Hosts

Quite often we would want to have multiple sites on one server. One way to achieve this is to use **virtual hosts**

In fact, we are using them already (for the default site)

##### Virtual Hosts (by port)

Create a file **/etc/nginx/sites-available/vhost-port.conf** with the following content

**server {**

**listen 8080;**

**location / {**

**root /var/www/vhost-port;**

**index index.html;**

**}**

**}**

Save and close the file

##### Virtual Hosts (by name)

Create a file **/etc/nginx/sites-available/vhost-name.conf** with the following content

**server {**

**listen 80;**

**server\_name www.demo.lab;**

**location / {**

**root /var/www/vhost-name;**

**index index.html;**

**}**

**}**

Save and close the file

##### Finalization

Create the corresponding **root** folders

**sudo mkdir /var/www/vhost-{name,port}**

Crete two new **index.html** files

**echo '<h1>Hello from M1 (vhost by port)</h1>' | sudo tee /var/www/vhost-port/index.html**

And

**echo '<h1>Hello from M1 (vhost by name)</h1>' | sudo tee /var/www/vhost-name/index.html**

Enable the two sites

**sudo ln -s /etc/nginx/sites-available/vhost-port.conf /etc/nginx/sites-enabled/**

**sudo ln -s /etc/nginx/sites-available/vhost-name.conf /etc/nginx/sites-enabled/**

Test the new configuration with

**sudo nginx -t**

Restart the **nginx** service

**sudo systemctl restart nginx**

Open port **8080/tcp** in the firewall if running

**sudo ufw allow 8080/tcp**

Add new record in the **/etc/hosts** file

**echo '<m1-ip> www.demo.lab www' | sudo tee -a /etc/hosts**

Finally, test both virtual hosts with

**curl http://localhost:8080**

**curl http://www.demo.lab**

#### TLS/SSL

We can use certificates issued from a trusted certificate authority or create self-signed certificate

As this is a demo, let us create a self-signed certificate

##### Preparation

Repeat the same steps as with **Apache**

Install the necessary packages with

**sudo apt-get update**

**sudo apt-get install openssl**

Generate the private key

**openssl genrsa -out ca.key 2048**

Create a certificate signing request (CSR)

**openssl req -new -key ca.key -out ca.csr**

Generate the self-signed certificate

**openssl x509 -req -days 365 -in ca.csr -signkey ca.key -out ca.crt**

We can we the result with

**openssl x509 -text -in ca.crt**

Copy the files to the appropriate folders

**sudo cp ca.crt /etc/nginx/ca.crt**

**sudo cp ca.key /etc/nginx/ca.key**

**sudo cp ca.csr /etc/nginx/ca.csr**

##### NGINX settings

Open the configuration file for the default site

**sudo vi /etc/nginx/sites-available/default**

Uncomment lines between **27** and **28**

Add the following two lines bellow

**ssl\_certificate /etc/nginx/ca.crt;**

**ssl\_certificate\_key /etc/nginx/ca.key;**

Save and close the file

Test the new configuration with

**sudo nginx -t**

Restart the **nginx** service

**sudo systemctl restart nginx**

Test the default site with

**curl -k** [**https://localhost**](https://localhost)

Now, you should see our default site

##### Final touches

We must open the appropriate port in the firewall if running and if not open with "Nginx Full"

**sudo ufw allow "Nginx HTTPS"**

Should we want to be automatically redirected to **https** when visiting **http**, we can modify the virtual host configuration for our default site

**sudo vi /etc/nginx/sites-available/default**

Modify the **srever** block to look like:

**server {**

**listen 80 default\_server;**

**listen [::]:80 default\_server;**

**return 301 https://$host$request\_uri;**

**}**

Just after it, open a new **server** block with

**server {**

Save and close the file

Test the new configuration with

**sudo nginx -t**

Restart the **nginx** service

**sudo systemctl restart nginx**

Open a browser tab on the host and navigate to **http://<m1-ip>/**

A warning should appear, accept it. Now, you should see our default site

#### PHP

Let us install the necessary packages

**sudo apt-get update**

**sudo apt-get install php php-fpm**

Once the installation is complete, we can check with

**php -v**

Open the default site configuration file

sudo vi /etc/nginx/sites-enabled/default

And modify the **index** clause to include **index.php** as well

Then, uncomment lines 65, 66, 69, and 72 to enable the **Fast CGI** communication

Then we can restart the **nginx** service

**sudo systemctl restart nginx**

Create a new **index.php** file with

**echo '<?php phpinfo(); ?>' | sudo tee /var/www/html/index.php**

Do test with either **curl** locally or a with browser tab on the host

#### Reverse Proxy

##### Preparation

Log on to the **M2** machine and install **nginx**

**sudo apt-get update**

**sudo apt-get install nginx**

Change the **/etc/nginx/sites-enabled/default** file by adding the following after the **root** directive (around line 42)

**set\_real\_ip\_from <machines-network>/24; 🡨 for example, 192.168.81.0/24**

**real\_ip\_header X-Forwarded-For;**

And adjusting the **server\_name** line if needed

Save and close the file

Test the new configuration with

**sudo nginx -t**

Restart the **nginx** service

**sudo systemctl restart nginx**

Then create a custom **index.html** page

**echo '<h1>Hello from M2</h1>' | sudo tee /var/www/html/index.html**

Don’t forget to adjust the firewall if running

**sudo ufw allow "Nginx Full"**

Test locally

**curl http://localhost**

##### Configuration

Return to the **M1** machine

Open the default site configuration file for editing

**sudo vi /etc/nginx/sites-enabled/default**

We can do the following changes only in the TLS section because of our set up

Enter the following lines just above the **location** (line 57)

**proxy\_redirect off;**

**proxy\_set\_header X-Real-IP $remote\_addr;**

**proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;**

**proxy\_set\_header Host $http\_host;**

Add the following to the **location** block

**proxy\_pass** [**http://m2.lsaa.lab/**](http://m2.lsaa.lab/)**;**

Save and close the file

Test the configuration

**sudo nginx -t**

Restart the **nginx** service

**sudo systemctl restart nginx**

Open a browser tab on the host and navigate to **https://<m1-ip>/**

You should see the index page of **M2**

#### Load Balancing

##### Preparation

Log on to the **M3** machine and install **nginx**

**sudo apt-get update**

**sudo apt-get install nginx**

Change the **/etc/nginx/sites-enabled/default** file by adding the following after the **root** directive (around line 42)

**set\_real\_ip\_from <machines-network>/24; 🡨 for example, 192.168.81.0/24**

**real\_ip\_header X-Forwarded-For;**

And adjusting the **server\_name** line if needed

Save and close the file

Test the new configuration with

**sudo nginx -t**

Restart the **nginx** service

**sudo systemctl restart nginx**

Then create a custom **index.html** page

**echo '<h1>Hello from M3</h1>' | sudo tee /var/www/html/index.html**

Don’t forget to adjust the firewall if needed

**sudo ufw allow "Nginx Full"**

Test the site locally

**curl http://localhost**

##### Configuration

We are one step away from turning our setup into a fully functional load balancing solution

Return on the **M1** machine

Open the main configuration file for editing

**sudo vi /etc/nginx/sites-enabled/default**

Add the following lines in before the first **server** section

**upstream backend {**

**server m2.lsaa.lab;**

**server m3.lsaa.lab;**

**}**

Then change the **location** (in our case just for the TLS section) to

**proxy\_pass http://backend;**

Save and close the file

Check the configuration

**sudo nginx -t**

Restart the **nginx** service

**sudo systemctl restart nginx**

Test by opening a browser tab on the host and navigating to **https://<m1-ip>/**

Refresh a few times. Everything should work as expected

### CUPS

We must either reset the existing infrastructure or prepare a new set of machines. In fact, we will need just one (M1)

#### Installation

Install the necessary packages with

**sudo apt-get update**

**sudo apt-get install cups**

Check its status (and start it if not running)

**systemctl status cups**

#### Configuration

Now, let us adjust a few settings in the **/etc/cups/cupsd.conf** configuration file

**sudo vi /etc/cups/cupsd.conf**

Change line 18 from **Listen localhost:631** to **Listen 0.0.0.0:631** to set **CUPS** listen on all interfaces

Add also the **ServerAlias \*** instruction

Add **Allow @LOCAL** to both **<Location />** and **<Location /admin>** to allow access from the local network

Save and close the file

Restart the **cups** service

**sudo systemctl restart cups**

Open the appropriate firewall port if needed

**sudo ufw allow 631/tcp**

Now, you can open a browser tab on the host and navigate to **http://<m1-ip>:631**

#### Installing a PDF printer

Return on **M1**

We will need the **cups-pdf** package which we will install like this

**sudo apt-get install printer-driver-cups-pdf**

Next, we can explore its configuration

**sudo vi /etc/cups/cups-pdf.conf**

Once done exploring, restart the **cups** service

**sudo systemctl restart cups**

Let us ask for the list of printers

**lpstat -p -d**

And now, that we have a printer, let us make it the default printer with

**lpoptions -d PDF**

#### Printing

We can either create a text file, or use on existing

In order to find one, we can execute

**find /usr/share/doc -name README**

We can pick up for example

**cat /usr/share/doc/sudo/README**

And print it with

**lp /usr/share/doc/sudo/README**

Then, if we list the content of our folder, we will see a **~/PDF/README.pdf** file

We can copy it on the host and open it

## Part 2: Directory Services

For this part we will need an infrastructure with just two machines

Machines can be with or without graphical environment

Network settings shown on the picture reflect the ones, used during the demonstration. You should adjust them according to your setup

### BIND

#### Caching Server

Log on to the **M1** machine

Start with installing the required packages

**sudo apt-get update**

**sudo apt-get install bind9 bind9-utils**

Open the main configuration file

**sudo vi /etc/bind/named.conf**

It appears that it contains only includes. We can continue here or open the options file. Let’s go with the second option

**sudo vi /etc/bind/named.conf.options**

Add the following block above the **options** instruction

**acl trusted-clients {**

**localhost;**

**<machines-network>/24; 🡨 for example 192.168.81.0/24**

**}**

This will allow us to grant all stations in our internal network the right to query the DNS server

Next, change the interfaces the **bind** service listens on

For example, change the IPv4 **localhost** (**127.0.0.1**) to **any** (to listen on all interfaces)or to **none** (to stop listen)

Same applies for IPv6but instead of **127.0.0.1** the address is **::1**

Let us set both **listen-on** (add this one) and **listen-on-v6** (change this one) to **any**

Now, change (add it if missing) the **allow-query** to **{ trusted-clients; }**

Save and close the file

Check that everything with the configuration is okay

**sudo named-checkconf**

It appears that have an error due to a missing semicolon

Open again the file and correct the error

Check the configuration again. Now all should be fine

It is a good idea to check the ownership of the file

**ls -l /etc/bind/named.conf\***

It must be owned by the **root** user and the **bind** group

If the service is not enabled and started, do it

**sudo systemctl enable --now named**

If it is already started, restart it

**sudo systemctl restart named**

And check its status

**systemctl status named**

Additionally, the status can be checked with

**sudo rndc status**

Don’t forget to allow the **DNS** service in the firewall if running

**sudo ufw allow "Bind9"**

Now, log on to the **M2** machine

Install the required packages

**sudo apt-get update**

**sudo apt-get install bind9-utils**

Set **M1** as a **DNS** server

Change local settings for the **DNS** server

Open the **Netplan** configuration file

**sudo vi /etc/netplan/00-installer-config.yaml**

And set the **addresses** option in the **nameservers** position to **<m1-ip>**

*A sample file may look like:*

***network:***

***ethernets:***

***eth0:***

***dhcp4: false***

***addresses:***

***- "192.168.81.171/24"***

***nameservers:***

***addresses: ["192.168.81.170"]***

***routes:***

***- to: 0.0.0.0/0***

***via: 192.168.81.171***

***version: 2***

*Please note that you may need to adjust the network interface name as well*

Save and close the file and apply the changes

**sudo netplan apply**

Let us look up information for a domain, for example **opensuse.org**

**dig opensuse.org**

We can see plenty of information

Notice the time it took to answer the query

Repeat the command once more

Now, the answer is returned fast, because of the cache

Let us try a reverse lookup query with one of the returned IP addresses

**dig -x 195.135.221.140**

*You may notice that the answer came from* ***127.0.0.53*** *and not the* ***<m1-ip>***

*This is the normal behavior in* ***Ubuntu***

*Check the actual uplink* ***DNS*** *server with*

***resolvectl status***

#### Forwarding Server

Now, we can modify the settings of our server and turn it into a forwarding server

Return on **M1** machine

Open the auxiliary configuration file

**sudo vi /etc/bind/named.conf.options**

Insert the following block after the **allow-query** instruction

**forwarders {**

**8.8.8.8;**

**8.8.4.4;**

**};**

**forward only;**

Save and close the file

Then, check it for errors

**sudo named-checkconf**

Execute the following to make **bind** reload its configuration

**sudo rndc reload**

And this one to flush the cache

**sudo rndc flush**

Switch to the **M2** machine and repeat the lookup queries

Even the first attempt resolves much faster now

#### Internal DNS server

Return on the **M1** machine

Open the auxiliary configuration file for editing

**sudo vi /etc/bind/named.conf.local**

Position the cursor at the end and type the following for the forward lookup zone

**zone "lsaa.lab" IN {**

**type master;**

**file "/etc/bind/lsaa.lab.zone";**

**allow-update { none; };**

**};**

And then add the following for the reverse lookup zone

**zone "81.168.192.in-addr.arpa" IN {**

**type master;**

**file "/etc/bind/81.168.192.zone";**

**allow-update { none; };**

**};**

Save and close the file

Check the configuration with

**sudo named-checkconf**

Create a new forward lookup zone file

**sudo vi /etc/bind/lsaa.lab.zone**

And enter the following

**$ORIGIN lsaa.lab.**

**$TTL 86400**

**@ IN SOA m1.lsaa.lab. root.lsaa.lab. (**

**2021092401 ; serial**

**3600 ; refresh in 1 hour**

**1800 ; retry in 30 minutes**

**604800 ; expires after 7 days**

**86400 ; minimum TTL of 1 day**

**)**

**IN NS m1.lsaa.lab.**

**IN A <m1-ip>**

**IN MX 10 m1.lsaa.lab.**

**m1 IN A <m1-ip>**

**m2 IN A <m2-ip>**

**client IN CNAME m2.lsaa.lab.**

Save and close the file

Ensure that the file permissions and ownership are as expected

**sudo chmod 644 /etc/bind/lsaa.lab.zone**

**sudo chown root:bind /etc/bind/lsaa.lab.zone**

Check the zone with

**sudo named-checkzone lsaa.lab /etc/bind/lsaa.lab.zone**

Create a new reverse lookup zone file

**sudo vi /etc/bind/81.168.192.zone**

And enter the following

**$TTL 86400**

**81.168.192.in-addr.arpa. IN SOA m1.lsaa.lab. root.lsaa.lab. (**

**2021092401 ; serial**

**3600 ; refresh in 1 hour**

**1800 ; retry in 30 minutes**

**604800 ; expires after 7 days**

**86400 ; minimum TTL of 1 day**

**)**

**IN NS m1.lsaa.lab.**

**<last-part-ip-m1> IN PTR m1.lsaa.lab.**

**<last-part-ip-m2> IN PTR m2.lsaa.lab.**

Save and close the file

Check the zone

**sudo named-checkzone 81.168.192.in-addr.arpa /etc/bind/81.168.192.zone**

Adjust the permissions and ownership

**sudo chmod 644 /etc/bind/81.168.192.zone**

**sudo chown root:bind /etc/bind/81.168.192.zone**

Restart the **named** service

**sudo systemctl restart named**

And check its status

**systemctl status named**

Change local settings for the **DNS** server

Open the **Netplan** configuration file

**sudo vi /etc/netplan/00-installer-config.yaml**

And set the **addresses** option in the **nameservers** position to **<m1-ip>**

Save and close the file and apply the changes

**sudo netplan apply**

Do a forward query for the **m2.lsaa.lab** machine

**dig m2.lsaa.lab**  
Then do a reverse lookup

**dig -x <m2-ip>**

You can switch to **M2** machine and test the same

### FreeIPA

*Please note that at the moment* ***FreeIPA*** *cannot be installed successfully on* ***Ubuntu Server 18.04****,* ***Ubuntu Server 20.04*** *or* ***Ubuntu Server 21.04***

*The same applies for the recent versions of* ***Debian*** *including* ***Debian Sid (Testing)***

*Alternatively, you can use a container technology like* ***Docker*** *or* ***Podman****. For instructions, check here: https://hub.docker.com/r/freeipa/freeipa-server/*

## Part 3: Mail Services

For this part we will need an infrastructure with just two machines

Machines can be with or without graphical environment

Network settings shown on the picture reflect the ones, used during the demonstration. You should adjust them according to your setup

### Sendmail

Log on to the **M1** machine

#### Installation

Install the required package

**sudo apt-get update**

**sudo apt-get install postfix**

When asked, select Ok and then **Internet Site** and confirm with **Ok**

Confirm the **system mail name** (it should match **m1.lsaa.lab**)

Once it is installed, the service will be started automatically

Check its status

**systemctl status postfix**

Now, check open TCP ports and what is listening on them

**sudo ss -ltnp**

#### Configuration

Let us configure **postfix** to listen on all interfaces

First get current value and then change it (it should be configured correctly already)

**postconf inet\_interfaces**

**sudo postconf -e "inet\_interfaces = all"**

Examine and set the **inet\_protocols** parameter

**postconf inet\_protocols**

**sudo postconf -e "inet\_protocols = ipv4"**

Examine and set if needed (it should be set already to the correct value) the **myhostname** parameter (it should be configured correctly already)

**postconf myhostname**

**sudo postconf -e "myhostname = m1.lsaa.lab"**

Examine and set if needed (it should be set already to the correct value) the **mydomain** parameter (it should be configured correctly already)

**postconf mydomain**

**sudo postconf -e "mydomain = lsaa.lab"**

All above changes could have been done with direct changes in **/etc/postfix/main.cf** file

We can list the available parameters and their default values with

**postconf -d**

We can check how many parameters are there

**postconf -d | wc -l**

Wow, we have more than 900 parameters

Alternatively, we can get the list of explicitly set parameters

**postconf -n**

We can check the configuration for any errors with

**sudo postfix check**

Restart the **postfix** service

**sudo systemctl restart postfix**

Alternatively, we can ask the daemon to reload its configuration with

**sudo systemctl reload postfix**

Check again the open port and listening processes with

**sudo ss -ltnp**

Open the **smtp** service in the firewall if needed

**sudo ufw allow "Postfix"**

#### Aliases

Let us explore the alias list

**cat /etc/aliases**

Now, add an alias redirecting mail from **root** to the current user

**sudo vi /etc/aliases**

Go to the end of the file and add

**root: <current-user>**

Save and close the file

Now, we must apply the changes by executing either the **sendmail** emulation command

**sudo newaliases**

Or the native one

**sudo postalias /etc/aliases**

And then restart the service

**sudo systemctl restart postfix**

#### Message and Mailbox Limits

Message size limit we can check and change via the following parameter

**postconf -p message\_size\_limit**

And the mailbox size limit, via the following

**postconf -p mailbox\_size\_limit**

The second should be bigger than the first one

#### Testing Locally

Install a command line mail client

**sudo apt-get update**

**sudo apt-get install mailutils**

Let us create a simple message with subject **test** for our current user

**mailx -s test <current-user>**

Then enter some text, for example

**This is a test message**

Once done, on a new line press **Ctrl+D**

We can compose and send another one with

**mailx -s 'Test message' postmaster**

Enter something for the body of the message and press **Ctrl+D** to send it

Then enter the mail program with

**mailx**

We can see a list of the messages in our mailbox

In order to read a message, we must enter its number and hit **Enter**

If you hit **Enter** again, the next message will be displayed

Should you want to see the list of messages, execute **header**

List of all commands can be invoked with **list**

In order to leave the application, you must execute **quit**

We can explore the log as well

**sudo tail /var/log/mail.log**

#### Sending Between Servers

Log on to **M2** machine

Install **postfix** and do a basic configuration

**sudo apt-get update**

**sudo apt-get install postfix mailutils**

When asked, select Ok and then **Internet Site** and confirm with **Ok**

Confirm the **system mail name** (it should match **m2.lsaa.lab**)

Once it is installed, the service will be started automatically

**sudo postconf -e "inet\_interfaces = all"**

**sudo postconf -e "inet\_protocols = ipv4"**

Restart the service

**sudo systemctl restart postfix**

Open the **smtp** service in the firewall if needed

**sudo ufw allow "Postfix"**

Now, we are ready to try to send an email from **M2** to **M1**

**mailx -s 'From M2 to M1 #1' user@m1.lsaa.lab**

Enter some body text and send the message

Ask for the message queue with

**postqueue -p**

It appears to be empty. Check the log

**sudo tail /var/log/mail.log**

It appears that the message was delivered

Log on to the **M1** machine and check with **mailx**

Indeed, the message has arrived

#### TLS with Postfix

Return to **M2** machine and create a self-signed certificate

**openssl req -x509 -nodes -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365 -subj "/CN=m2.lsaa.lab"**

Alternatively, we can use the ones generated during **postfix** installation (go with this one)

We can explore the certificate with

**openssl x509 -in /etc/ssl/certs/ssl-cert-snakeoil.pem -text**

Now, let us check a set of parameters and their values

**postconf -p smtpd\_use\_tls**

**postconf -p smtpd\_enforce\_tls**

**postconf -p smtpd\_tls\_loglevel**

**postconf -p smtpd\_tls\_key\_file**

**postconf -p smtpd\_tls\_cert\_file**

If **smptd\_use\_tls** is set to **no**, then change it to **yes**. In addition, set the **smtpd\_tls\_loglevel** to **1** and restart the service

**sudo postconf -e 'smtpd\_use\_tls = yes'**

**sudo postconf -e 'smtpd\_tls\_loglevel = 1'**

**sudo systemctl restart postfix**

Now, let us test the **TLS/SSL** communication to our **SMTP** server with

**openssl s\_client -connect m2.lsaa.lab:25 -starttls smtp**

We can browse the information here, in the client

We can type the following to receive information from the server

**ehlo m2.lsaa.lab**

Let us exit with **quit** and check the logs

**sudo tail /var/log/mail.log**

Here, we can see that an anonymous **TLS** connection had been established

It appears that we have **TLS** working out of the box without significant changes

Furthermore, now we are using just one port - **25/tcp** for both plain and **TLS** connections. We can check

**sudo ss -ltnp**

Let us see how we can switch or expose the **TLS** service on a separate port

Open for editing the **/etc/postfix/master.cf** file

**sudo vi /etc/postfix/master.cf**

And find and uncomment the **smtps** section (lines 29 to 39)

Save and close the file

Check the configuration with

**sudo postfix check**

Add the following to the end of the **/etc/postfix/main.cf** file especially if they are not defined (you see warnings when use **postconf -n** for example)

**smtpd\_restriction\_classes = mua\_sender\_restrictions, mua\_client\_restrictions, mua\_helo\_restrictions**

**mua\_client\_restrictions = permit\_sasl\_authenticated, reject**

**mua\_sender\_restrictions = permit\_sasl\_authenticated, reject**

**mua\_helo\_restrictions = permit\_mynetworks, reject\_non\_fqdn\_hostname, reject\_invalid\_hostname, permit**

Add the following to enable the **SASL** (**Simple Authentication and Security Layer**) authentication

**smtpd\_sasl\_auth\_enable = yes**

**broken\_sasl\_auth\_clients = yes**

**smtpd\_recipient\_restrictions = permit\_sasl\_authenticated, permit\_mynetworks, reject\_unauth\_destination**

Save and close the file

Check the configuration again with

**sudo postfix check**

Install the **saslauthd** service

**sudo apt-get update**

**sudo apt-get install sasl2-bin**

Check the status of the **saslauthd** service (it should be automatically started)

**systemctl status saslauthd**

Restart the **postfix** service

**sudo systemctl restart postfix**

If we check again the open ports and the listening processes

**sudo ss -ltnp**

We will see that we have the **postfix master** process listening on port **465**

Let us check again by establishing an interactive session

**openssl s\_client -connect m2.lsaa.lab:465**

We can type the following to receive information from the server

**ehlo m2.lsaa.lab**

And the type **quit** to exit

### Dovecot

Continue on **M2** machine

#### Preparation

In order to be able to send emails from (desktop) clients, we must enable the **submission** service

Open for editing the **/etc/postfix/master.cf** file

**sudo vi /etc/postfix/master.cf**

And uncomment lines **17** to **28**

Save and close the file

Check the values and change them if needed

**postconf -p smtpd\_sasl\_type**

**sudo postconf -e 'smtpd\_sasl\_type = dovecot'**

**postconf -p smtpd\_sasl\_path**

**sudo postconf -e 'smtpd\_sasl\_path = private/auth'**

Restart the **postfix** service

**sudo systemctl restart postfix**

And check the list of open ports and listening processes

**sudo ss -tlpn**

#### Installation

Let us install the required packages

**sudo apt-get update**

**sudo apt-get install dovecot-core dovecot-imapd dovecot-pop3d**

The service will auto start. Check its status for the protocols loaded

**systemctl status dovecot**

Check the **mail\_spool\_directory** parameter

**postconf -p mail\_spool\_directory**

It will return **/var/mail**

Now, open the **/etc/dovecot/conf.d/10-mail.conf** file for editing

**sudo vi /etc/dovecot/conf.d/10-mail.conf**

Copy the text from line **25** and set it on line **30** (if not set already)

On line **114** set **mail\_privileged\_group = mail** (if not set already)

Save and close the file

Add the **dovecot** user to the **mail** group

**sudo usermod -a -G mail dovecot**

Next, adjust the authentication mechanisms

Open for editing the **/etc/dovecot/conf.d/10-auth.conf** file

**sudo vi /etc/dovecot/conf.d/10-auth.conf**

Adjust any authentication related settings if needed

Save and close the file

Open for editing the **/etc/dovecot/conf.d/10-master.conf** file

**sudo vi /etc/dovecot/conf.d/10-master.conf**

Uncomment lines **101** to **103** and add bellow the **mode** line

**user = postfix**

**group = postfix**

Save and close the file

Open the file **/etc/dovecot/conf.d/15-mailboxes.conf** for editing

**sudo vi /etc/dovecot/conf.d/15-mailboxes.conf**

Add **auto = create** to every mailbox definition you like

Save and close the file

Restart the **dovecot** service

**sudo systemctl restart dovecot**

Restart the **postfix** service

**sudo systemctl restart postfix**

Open the appropriate ports in the firewall if needed

**sudo ufw allow "Dovecot IMAP"**

**sudo ufw allow "Dovecot Secure IMAP"**

**sudo ufw allow "Dovecot POP3"**

**sudo ufw allow "Dovecot Secure POP3"**

#### Remote Testing

Log on to the **M1** machine

Install terminal mail client

**sudo apt-get update**

**sudo apt-get install mutt**

Create a settings file

**vi .muttrc**

With the following content

**set folder="imap://user@m2.lsaa.lab"**

**set spoolfile="imap://user@m2.lsaa.lab/INBOX"**

Save and close the file

Start the client with **mutt**

You will be asked if you accept the certificate. Confirm with **a**

Then you will be asked for a password

Now, if everything is okay, you will see the content of your mailbox

Try to send an email