Informe Examen Técnico

Álvaro Santos Romero

Mayo del 2022

Índice

1. Exercise 1: Setup

Vagrantfile setting up four machines in the same net.

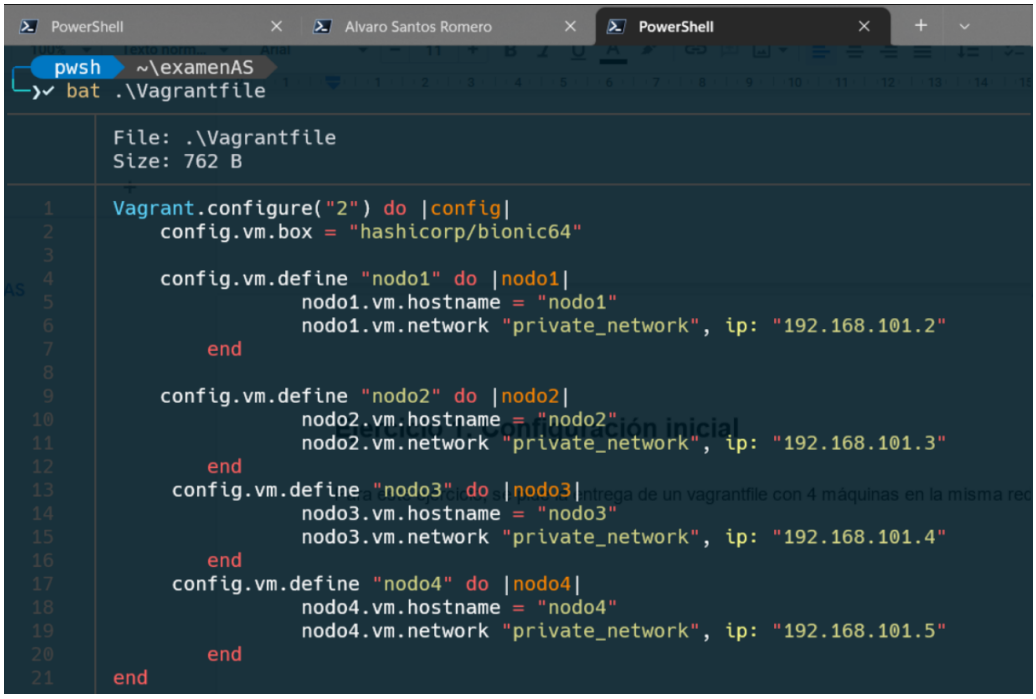

Figura 1: Vagrantfile

In this exercise, we are defining four machines in the same net, as we want them to be visible from each others.

2. Exercise 2: Firewall

We will use **iptables** to configure a simple firewall.

1. **Restrict input requests from HTTP, HTTPS and SSH**
   - iptables -P input DROP — default deny input requests rule
   - iptables -A INPUT tcp -dport 80 -j ACCEPT — accept input requests from port 80
   - iptables -A INPUT tcp -dport 443 -j ACCEPT — accept input requests from port 443
   - iptables -A INPUT tcp -dport 22 -j ACCEPT — accept input requests from port 22

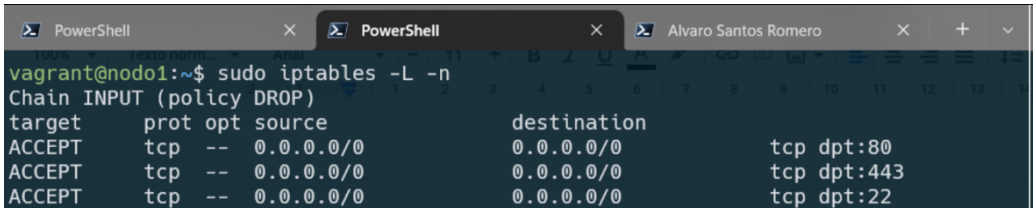

Figura 2: iptables

2. **Allow established and related connections**
   - iptables -A INPUT -m conntrack –ctstate ESTABLISHED -p all -j ACCEPT
   - iptables -A INPUT -m conntrack –ctstate RELATED -p all -j ACCEPT

   In this case, we are using a module called **conntrack** which is used to manage connections.

3. **Allow FTP connection to an external server (2.21.25.33)**
   - iptable -A INPUT -p tcp -dport 21 -s 2.21.25.33 -j ACCEPT

Figura 3: ftp

¡Caution!

Iptables rules are followed in order, so the rules must be set in this order to work propertly.

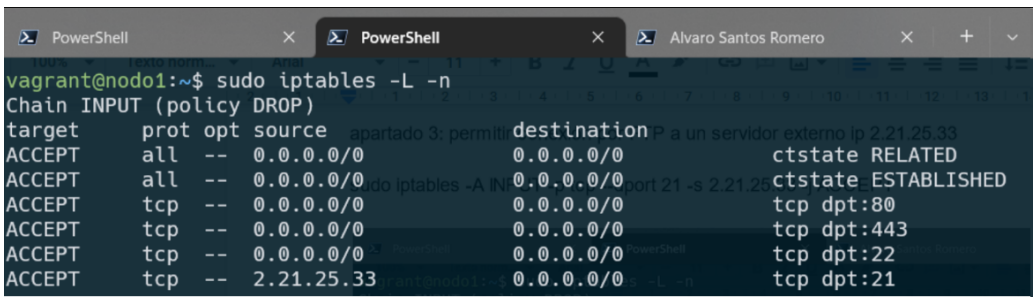

Figura 4: iptables rules

Rules order
Usually, the rules are order from most to less specific.

As we discussed earlier, iptables follows a secuencial order to execute the rules.

First of all, we should set the **established** and **related** ones first, as we dont want them to be overrided by the **restricted** ones.

The last rules should always be the most specific, such as acepting requests from a particular port or similar.

4. **Testing iptables rules**

   We will use **NetCat**, **Telnet** and **Nmap**.

   - **Listening port 80**

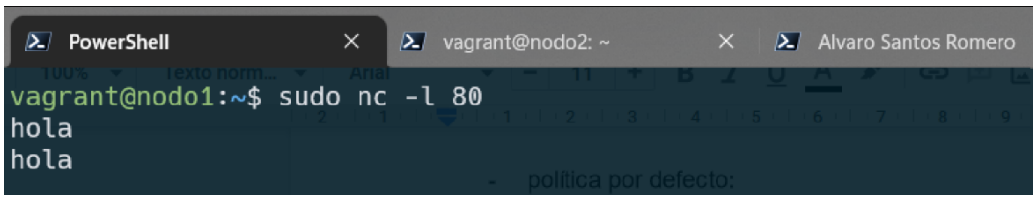

Figura 5: HTTP connection

   - **Listening port 443**

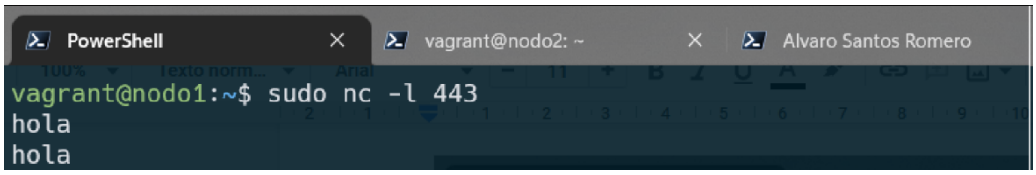

Figura 6: HTTPS connection

- **SSH connections**

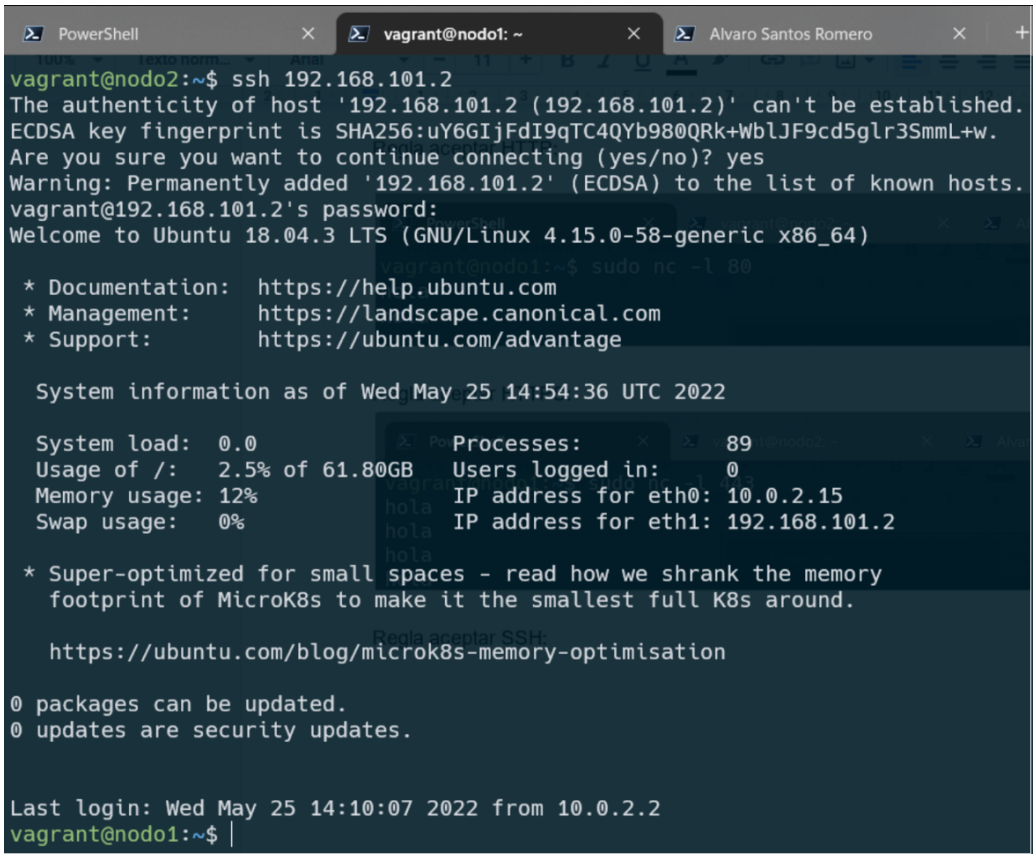

Figura 7: SSH connection

In this case, even if we decided to create a new rule denying connections from SSH, we will not be affected as we have a ESTABLISHED rule affecting this protocol.

- **Nmap scan**

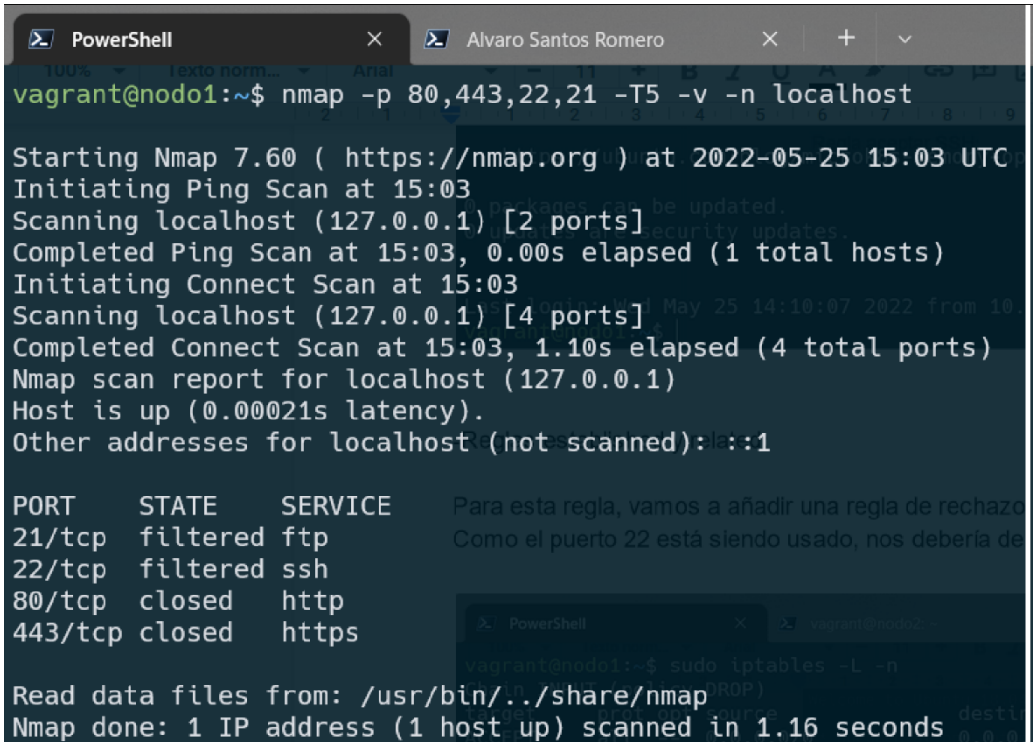

Figura 8: Nmap scan

Ports **22** and **21** are tagged as **filtered** as Nmap can not determine its states. (port 21 denies all input requests and port 22 only accepts from 2.21.25.33).

3. Exercise 3: Apache

Setup

- New folder to contain the website at **var/www**
- Create site configurations for in **/etc/apache2/sites-avaliable**

3.1. Indexes

We disable **indexes option** for all folder except for src.

```
<Directory "/var/www/asr">
        Options -Indexes
</Directory>
<Directory "/var/www/asr/src">
        Options +Indexes
</Directory>
```

Figura 9: Site config

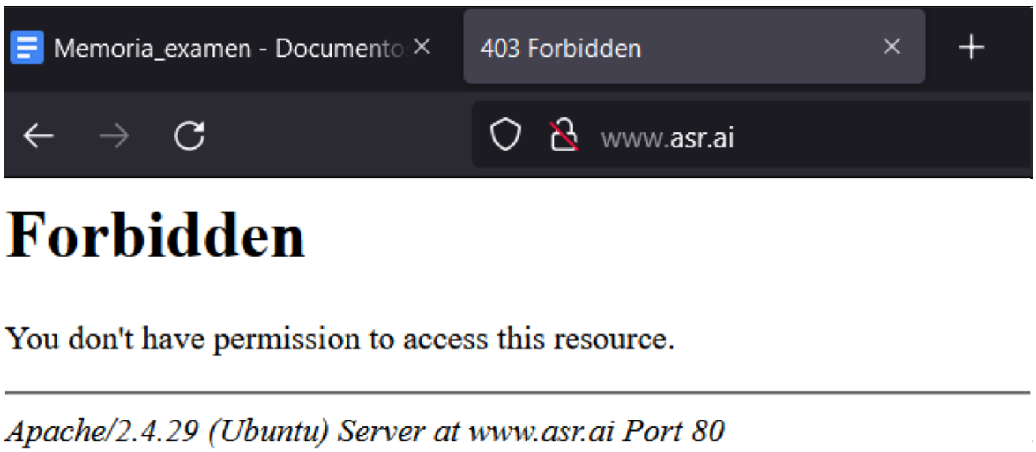

Figura 10: home

As we see, we can not access the content in the main domain.

Figura 11: src rute

However, we can access the content of src as we enabled the indexes option.

3.2. UTF-8

First of all, we need to import the module **libapache2-mod-php** to be able to use PHP in Apache.
In order to use special characters, we need to edit the file **charset.conf** in **/etc/apache2/conf-available**.

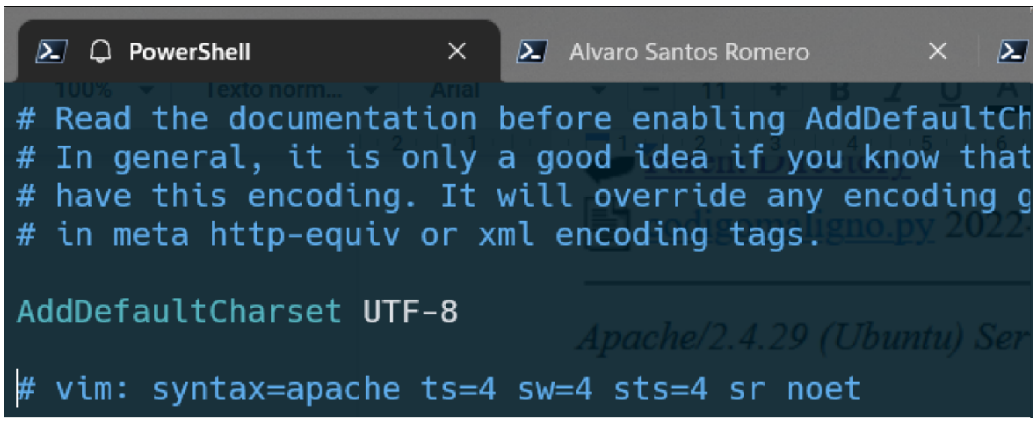

Figura 12: enabling UTF-8

We can check if the configuration is working by using the tool **curl**.

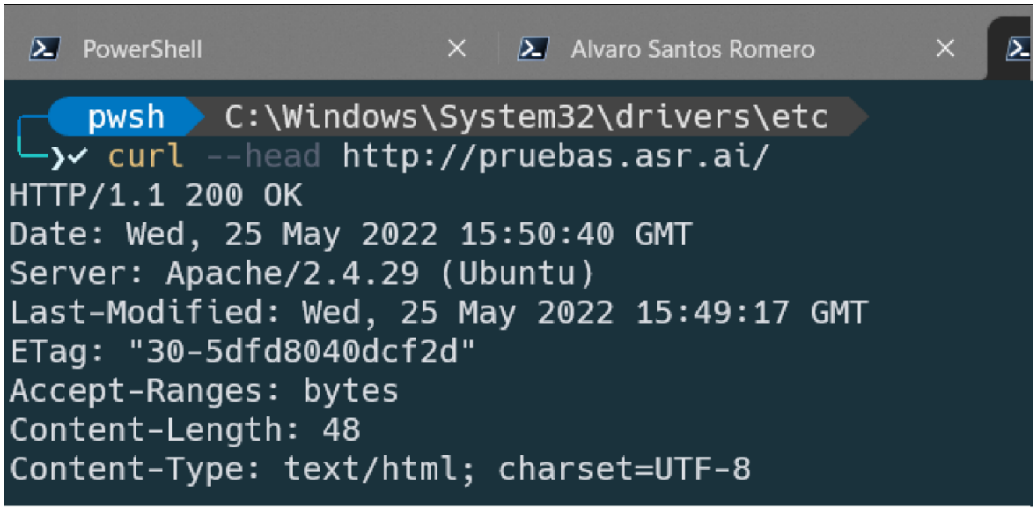

Figura 13: curl

3.3. Apache Engine

In order to rewrite rules, we need to enable the rewrite module.

- **url conversion**

```
RewriteEngine On
RewriteRule "^catalogo/(.+)$" "prodList.php?id=$1" [QSA]
```

Figura 14: url conversion

- **PHP file to get the variable**

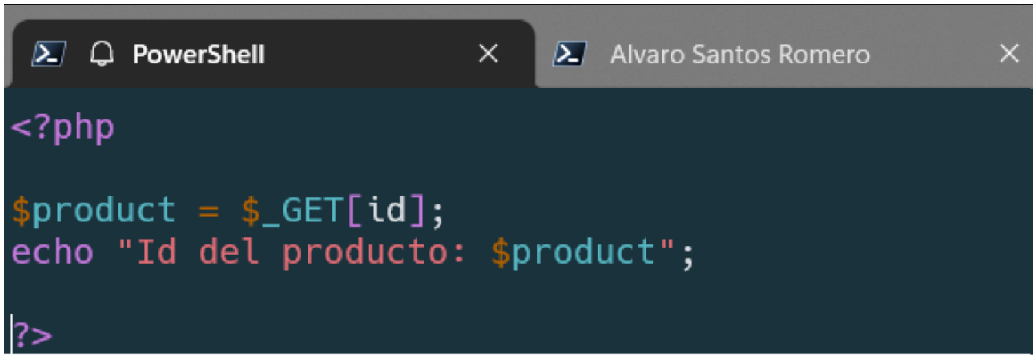

Figura 15: php variable

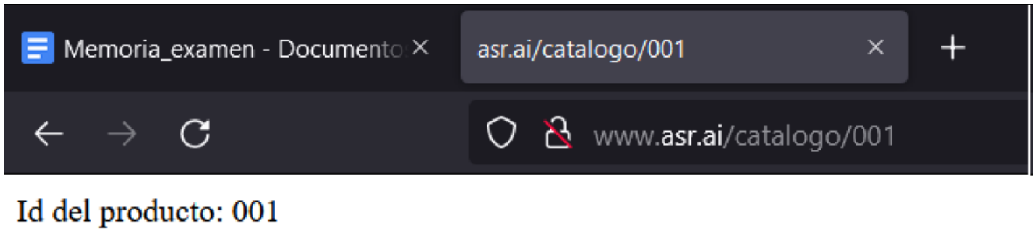

Figura 16: php file output

- **Subdomains**

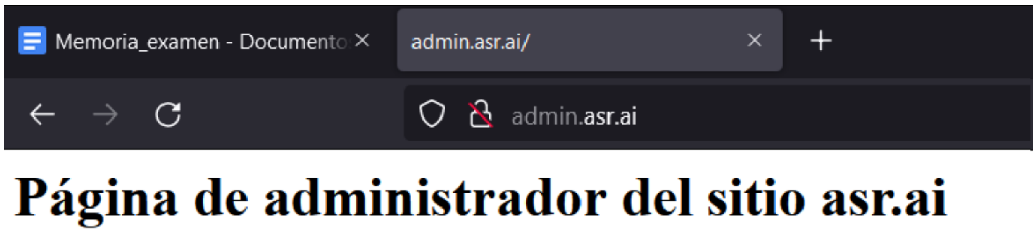

Figura 17: admin page

Figura 18: pruebas page

- **Zones**

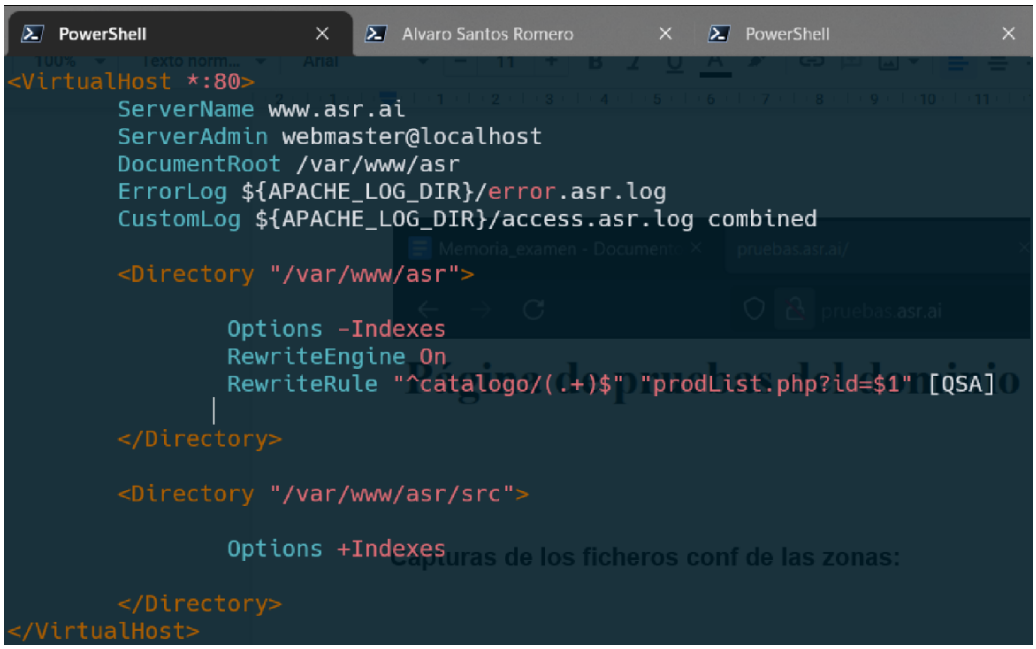

Figura 19: www.asr.ai

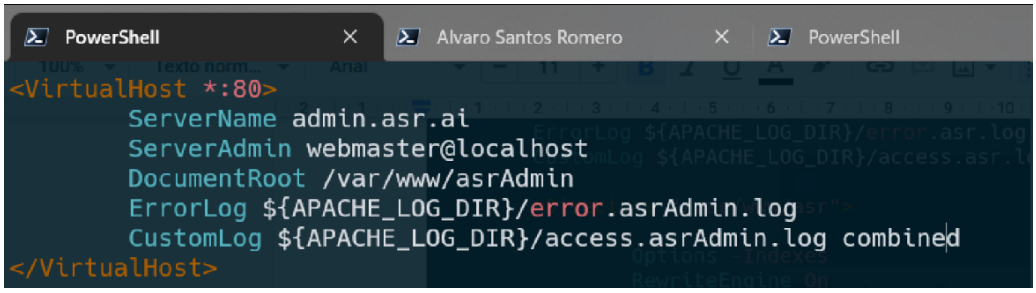

Figura 20: admin.asr.ai

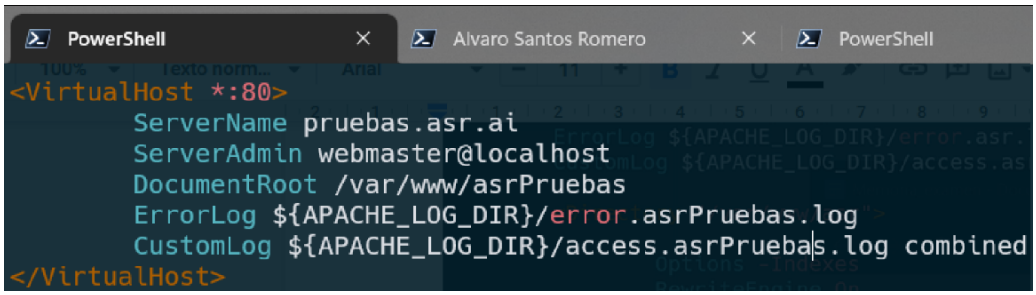

Figura 21: pruebas.asr.ai

4. Exercise 4: DNS config

Before hand, we will install **Bind9** in order to use the DNS.

4.1. Direct Zones

We can create the zones in **named.conf.local**.

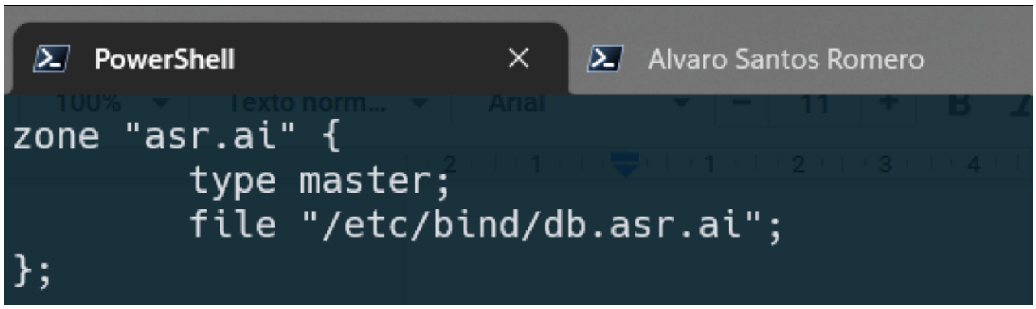

Figura 22: Direct zone asr.ai

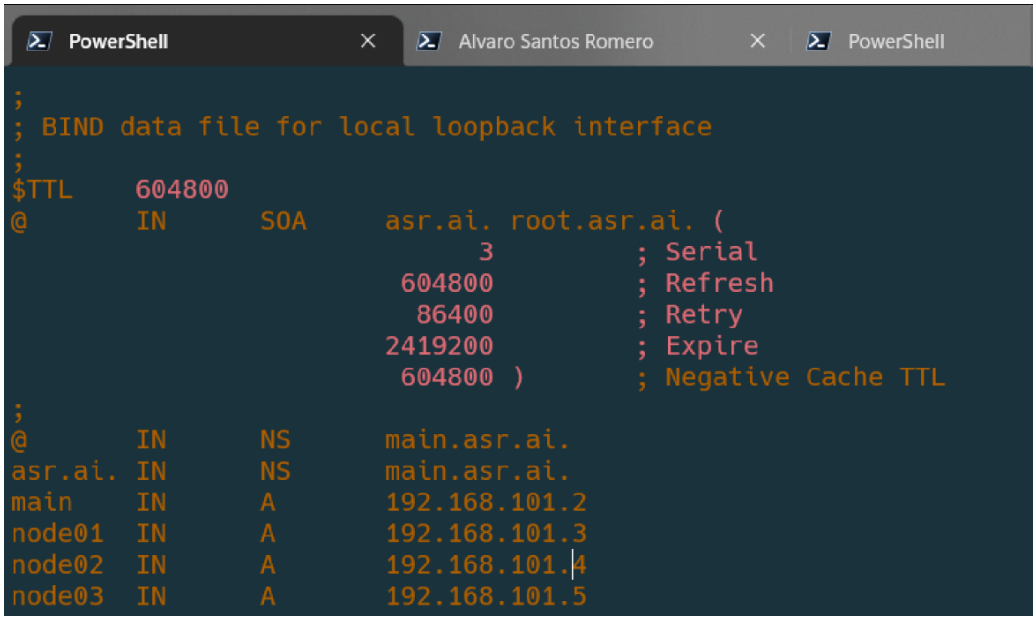

Figura 23: DB file asr.ai

This file is used to store the domains and IPs of the different sites, so the DNS can associate the domain with the IP.

4.2. Inverse Zones

As we did with the direct zones, we can create them in **named.conf.local**.

```
zone "101.168.192.in-addr.arpa" {
        type master;
        file "/etc/bind/db.192.168.101";
};
```

Figura 24: Inverse zone 101.168.192

```
;
; BIND reverse data file for local loopback interface
;
$TTL    604800
@       IN      SOA     main.asr.ai. root.asr.ai. (
                              5         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
;
@       IN      NS      main.
2       IN      PTR     main.asr.ai.
3       IN      PTR     node01.asr.ai.
4       IN      PTR     node02.asr.ai.
5       IN      PTR     node03.asr.ai.
```

Figura 25: db file 101.168.192

We can check if the zones are loaded by using:

```
name-checkconf -z named.conf.local
```

```
vagrant@nodo1:/etc/bind$ sudo named-checkconf -z named.conf.local
zone asr.ai/IN: loaded serial 3
zone 101.168.192.in-addr.arpa/IN: loaded serial 2
vagrant@nodo1:/etc/bind$
```

Figura 26: zones check

5. Load Balancer

Before hand, we have to load these modules:

- **proxy**
- **proxy_http**
- **lbmethod_byrequests**

We will need to have an operative DNS installed in order to use the proxy.
We will use the default config file **000-default.conf** as we will not include any additional features to the proxy.

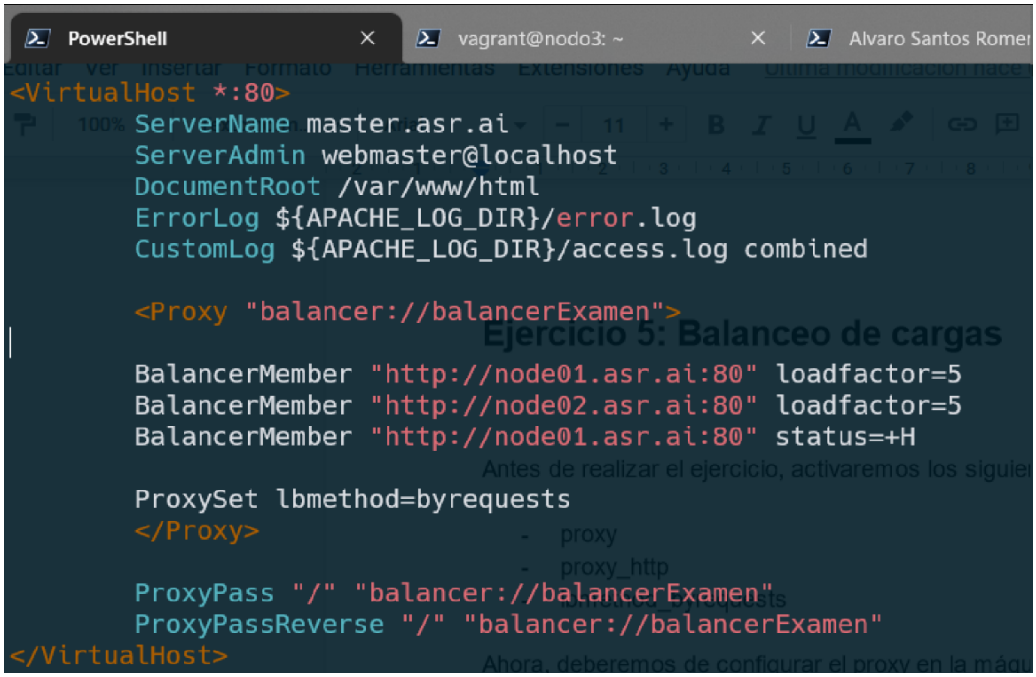

Figura 27: 000-default.conf

- Balance Members: node01, node02 and node03.
- Nodes 01 and 02 respond to half of the requests.
- Node03 is acting as a backup in case any of the other nodes fails.
- Load method is by requests.
- ProxyPass and ProxyPassReverse is used to get and send the information between the nodes.

Now, we will duplicate the nodes in order to have the "same"server accepting the client´s requests.

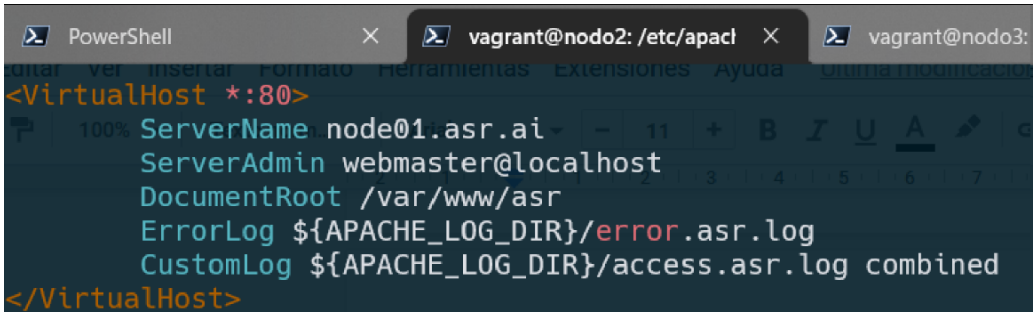

Figura 28: node server

Now, whenever a client sends a request to the server, one of the two nodes will answer.

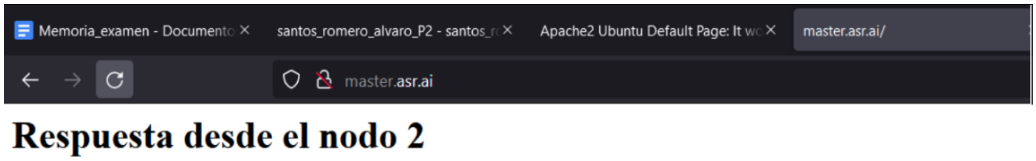

Figura 29: balancer working

If we reload the page, we will see that the answers would come from node01 and node02.
Node03 will answer only if both node01 and node02 are down.

5.1. Load Balancer with DNS

In general, **proxy_mod** works with IPs by default.
However, we can provide a DNS to be able to use the domain names instead of the IPs.

In order to use our DNS, we have to edit both the **hosts** and **resolv.conf** files.

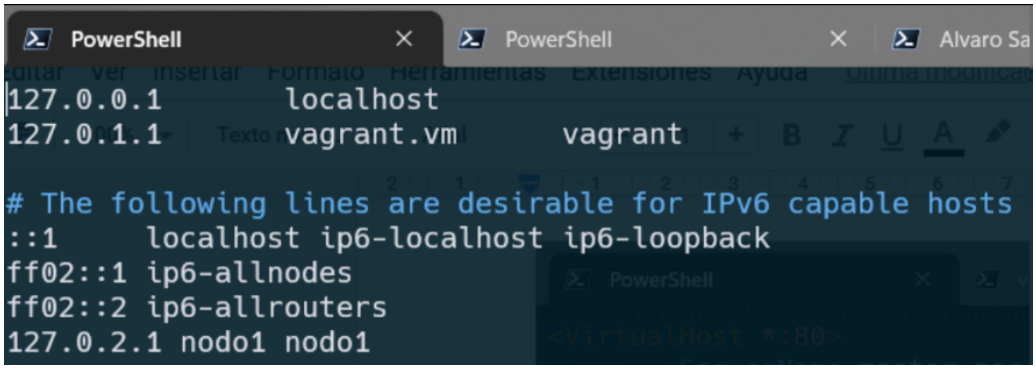

Figura 30: hosts file

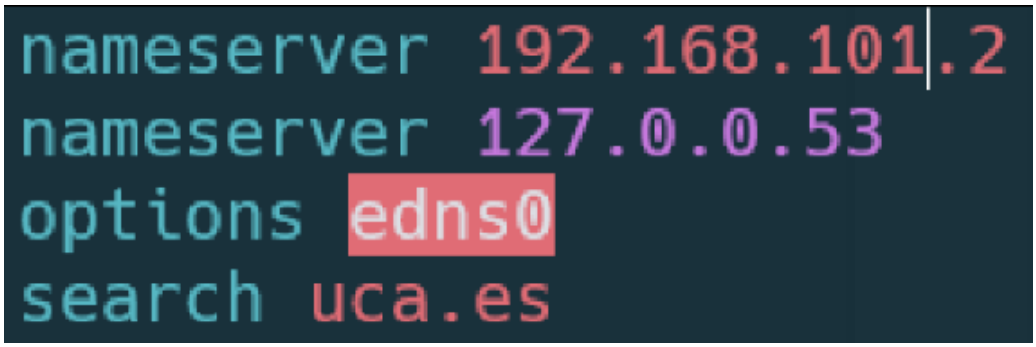

Figura 31: resolv.conf file