Lab 1/Sprint 1: System setup & Lab config

Presented by: **Hamza LAZAAR**

Chapter 1. VMware and Vagrant configuration

Downloaded `vmware` and `vagrant` with the vmware plugin and vagrant-vmware-utility :

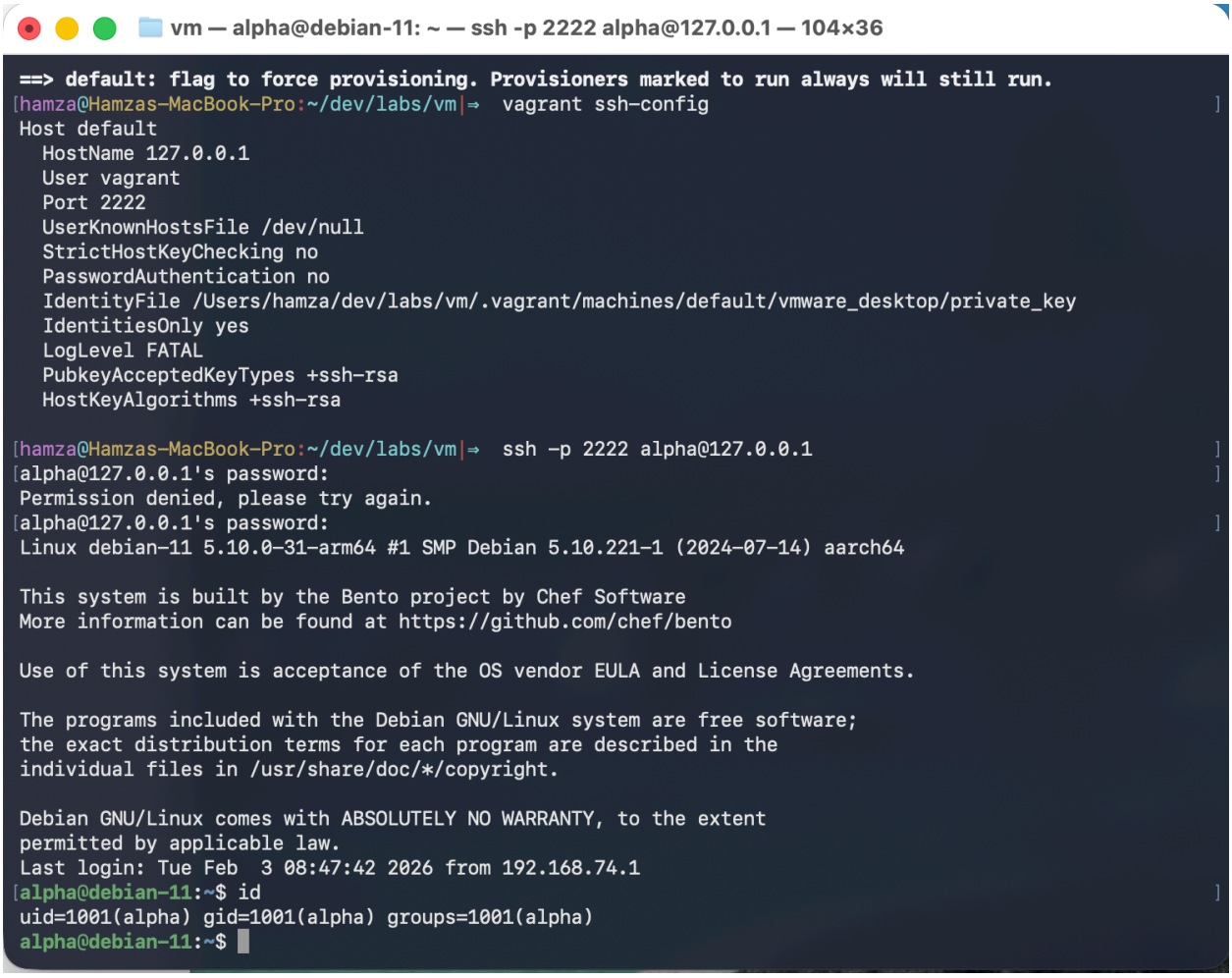
The Virtual Machine is now available. We'll move to chapter 2 to uncover SSH logging to this VM.

Chapter 2. SSH

Phase A: Identification

Used `vagrant ssh-config` to identify the local forwarding port (e.g., `2222`) and the private key location.

- **User:** `alpha`
- **Password:** `123456`



Phase B: Manual Key Installation

To understand the mechanics of SSH, we manually configured passwordless access:

- **Generated Key:** `ssh-keygen -t ed25519 -C "vm_lab" -f ~/.ssh/vm_lab_key` (Host machine).

```
[hamza@Hamzas-MacBook-Pro:~/dev/labs/vm|⇒  ssh-keygen -t ed25519 -C "vm_lab" -f ~/.ssh/vm_lab_key
Generating public/private ed25519 key pair.
Enter passphrase for "/Users/hamza/.ssh/vm_lab_key" (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/hamza/.ssh/vm_lab_key
Your public key has been saved in /Users/hamza/.ssh/vm_lab_key.pub
The key fingerprint is:
SHA256:4DE2Hq7Eb27CbpnZgKyDEKXmdpZRe18y82dcKiGugp0 vm_lab
The key's randomart image is:
+--[ED25519 256]--+
|                 |
|  .  .           |
| o  . O          |
|o. o * B = o   . |
|oo .= = S B o o  |
|.oo=.o   o o =   |
|+.o.+*+ .   +    |
|+  .BEo.         |
| . o.oo          |
+----[SHA256]-----+
hamza@Hamzas-MacBook-Pro:~/dev/labs/vm|⇒
```

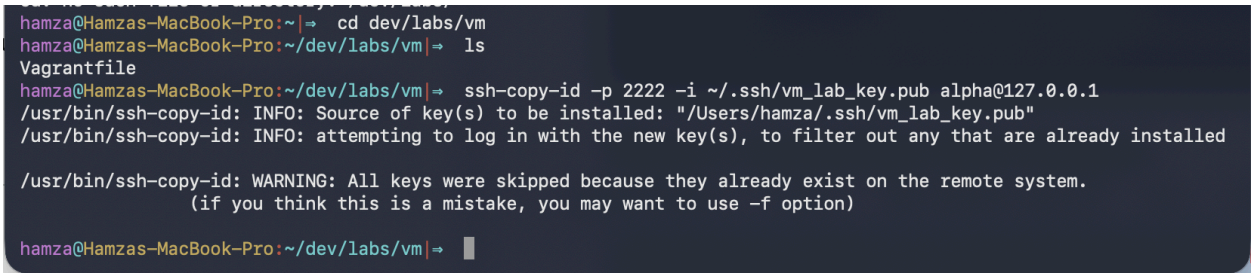
- **Transferred Key:** Used `scp` to copy the public key to `/tmp/` on the VM.
- **Installed Key:** Appended the key to `~/.ssh/authorized_keys` and set permissions (`chmod 700` for dir, `600` for file).

```
[hamza@Hamzas-MacBook-Pro:~/dev/labs/vm|⇒  scp -P 2222 ~/.ssh/vm_lab_key.pub alpha@127.0.0.1:/tmp/
[alpha@127.0.0.1's password:
vm_lab_key.pub                                100%   88   197.1KB/s   00:00
[hamza@Hamzas-MacBook-Pro:~/dev/labs/vm|⇒  ssh -p 2222 alpha@127.0.0.1
[alpha@127.0.0.1's password:
Linux debian-11 5.10.0-31-arm64 #1 SMP Debian 5.10.221-1 (2024-07-14) aarch64

This system is built by the Bento project by Chef Software
More information can be found at https://github.com/chef/bento

Use of this system is acceptance of the OS vendor EULA and License Agreements.

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Feb  3 08:49:53 2026 from 192.168.74.1
[alpha@debian-11:~$ mkdir -p ~/.ssh
[alpha@debian-11:~$ chmod 700 ~/.ssh
[alpha@debian-11:~$ cat /tmp/vm
vm_lab_key.pub              vmware-root_516-2998936442/
[alpha@debian-11:~$ cat /tmp/vm_lab_key.pub >> ~/.ssh/authorized_keys
[alpha@debian-11:~$ chmod 600 ~/.ssh/authorized_keys
[alpha@debian-11:~$ exit
```

Phase C: Automation (Best Practice)

Verified that `ssh-copy-id` can automate Phase B.

```
ssh-copy-id -p 2222 -i ~/.ssh/vm_lab_key.pub alpha@127.0.0.1
```



💡 Since the keys were already manually installed, the execution of the command above was skipped to avoid any conflicts.

Chapter 3. SSL

Objective

To analyze how clients (browsers/CLI) handle invalid SSL certificates by testing against specific misconfigured endpoints on `badssl.com`. The goal is to differentiate between **Expiry** and **Trust** errors.

Part 1: cURL Analysis

We tested two endpoints to observe how `curl` (which uses the system's strict CA store) responds to invalid certificates.

A. Expired Certificate

Command: `curl -v https://expired.badssl.com/`

> **Observation:**
> The server handshake was rejected because the current date is past the certificate's `notAfter` date (2020 in our case).

- **Error Code:** `(60) SSL certificate problem: certificate has expired`
- **Meaning:** The chain of trust is valid (issued by a real CA), but the "time validity" check failed.

```
[hamza@Hamzas-MacBook-Pro:~/dev/labs/vm|⇒  curl -v https://expired.badssl.com/
* Host expired.badssl.com:443 was resolved.
* IPv6: (none)
* IPv4: 104.154.89.105
*   Trying 104.154.89.105:443...
* Connected to expired.badssl.com (104.154.89.105) port 443
* ALPN: curl offers h2,http/1.1
* (304) (OUT), TLS handshake, Client hello (1):
*  CAfile: /etc/ssl/cert.pem
*  CApath: none
* (304) (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (OUT), TLS alert, certificate expired (557):
* SSL certificate problem: certificate has expired
* Closing connection
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (OUT), TLS alert, certificate expired (557):
curl: (60) SSL certificate problem: certificate has expired
More details here: https://curl.se/docs/sslcerts.html

curl failed to verify the legitimacy of the server and therefore could not
establish a secure connection to it. To learn more about this situation and
how to fix it, please visit the web page mentioned above.
```

B. Self-Signed Certificate

Command: `curl -v https://self-signed.badssl.com/`

> **Observation:**
> The handshake was rejected because the issuer is unknown.

- **Error Code:** `(60) SSL certificate problem: self signed certificate`
- **Meaning:** The chain of trust is broken. The certificate was signed by the server itself, not by a trusted Root CA recognized by the OS.

```
[hamza@Hamzas-MacBook-Pro:~/dev/labs/vm|⇒  curl -v https://self-signed.badssl.com/
* Host self-signed.badssl.com:443 was resolved.
* IPv6: (none)
* IPv4: 104.154.89.105
*   Trying 104.154.89.105:443...
* Connected to self-signed.badssl.com (104.154.89.105) port 443
* ALPN: curl offers h2,http/1.1
* (304) (OUT), TLS handshake, Client hello (1):
*  CAfile: /etc/ssl/cert.pem
*  CApath: none
* (304) (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (OUT), TLS alert, unknown CA (560):
* SSL certificate problem: self signed certificate
* Closing connection
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (OUT), TLS alert, unknown CA (560):
curl: (60) SSL certificate problem: self signed certificate
More details here: https://curl.se/docs/sslcerts.html

curl failed to verify the legitimacy of the server and therefore could not
establish a secure connection to it. To learn more about this situation and
how to fix it, please visit the web page mentioned above.
```

🔍 Part 2: OpenSSL Inspection

We used OpenSSL to view the raw certificate details that `curl` was complaining about.

Command: `openssl s_client -connect expired.badssl.com:443`

Key Findings

- ***Validity Period*:** The `Not After` field shows a date in the past (May 30, 2020).

```
  2 s:C=GB, ST=Greater Manchester, L=Salford, O=COMODO CA Limited, CN=COMODO RSA Certification Authority
    i:C=SE, O=AddTrust AB, OU=AddTrust External TTP Network, CN=AddTrust External CA Root
    a:PKEY: RSA, 4096 (bit); sigalg: sha384WithRSAEncryption
    v:NotBefore: May 30 10:48:38 2000 GMT; NotAfter: May 30 10:48:38 2020 GMT
---
```

- ***Verification Code:*** The bottom of the output confirms `Verify error:num=10:(certificate has expired)`.

```
hamza@Hamzas-MacBook-Pro:~/dev/labs/vm|⇒  openssl s_client -connect expired.badssl.com:443

Connecting to 104.154.89.105
CONNECTED(00000005)
depth=2 C=GB, ST=Greater Manchester, L=Salford, O=COMODO CA Limited, CN=COMODO RSA Certification Authority
verify return:1
depth=1 C=GB, ST=Greater Manchester, L=Salford, O=COMODO CA Limited, CN=COMODO RSA Domain Validation Secure Server
CA
verify return:1
depth=0 OU=Domain Control Validated, OU=PositiveSSL Wildcard, CN=*.badssl.com
verify error:num=10:certificate has expired
notAfter=Apr 12 23:59:59 2015 GMT
verify return:1
depth=0 OU=Domain Control Validated, OU=PositiveSSL Wildcard, CN=*.badssl.com
notAfter=Apr 12 23:59:59 2015 GMT
verify return:1
```

🛠️ Part 3: Solutions & Fixes

Answer to the lab question: "How would you fix these issues?"

Problem 1: Expired Certificate

Fix: Renew the certificate.

1. Generate a new **CSR** (Certificate Signing Request) on the server.
2. Submit it to the Certificate Authority (e.g., DigiCert, Let's Encrypt ...).
3. Install the newly issued certificate with a valid `notAfter` date.

Problem 2: Self-Signed Certificate

Scenario A: Public Website (Production)

- **Fix:** Replace the self-signed cert with one from a trusted Public CA (like Let's Encrypt). This ensures all browsers trust it automatically.

Scenario B: Internal Tool (Dev/Intranet)

- **Fix:** Add the self-signed certificate to the client's **Trust Store**.
- *Example:* On Linux, copy the `.crt` file to `/usr/local/share/ca-certificates/` and update the store. This tells the OS to explicitly trust this specific custom certificate.

📝 Conclusion

- **Expiry** is a maintenance failure (forgetting to renew).
- **Self-Signed** is a configuration choice that requires **manual trust**.

- Tools like `curl` are strict by default, whereas `openssl` allows for deep debugging of the handshake properties.

Chapter 4: Cronjob

1. 🎯 Objective

To automate system maintenance by scheduling a daily script that deletes temporary files older than 7 days.

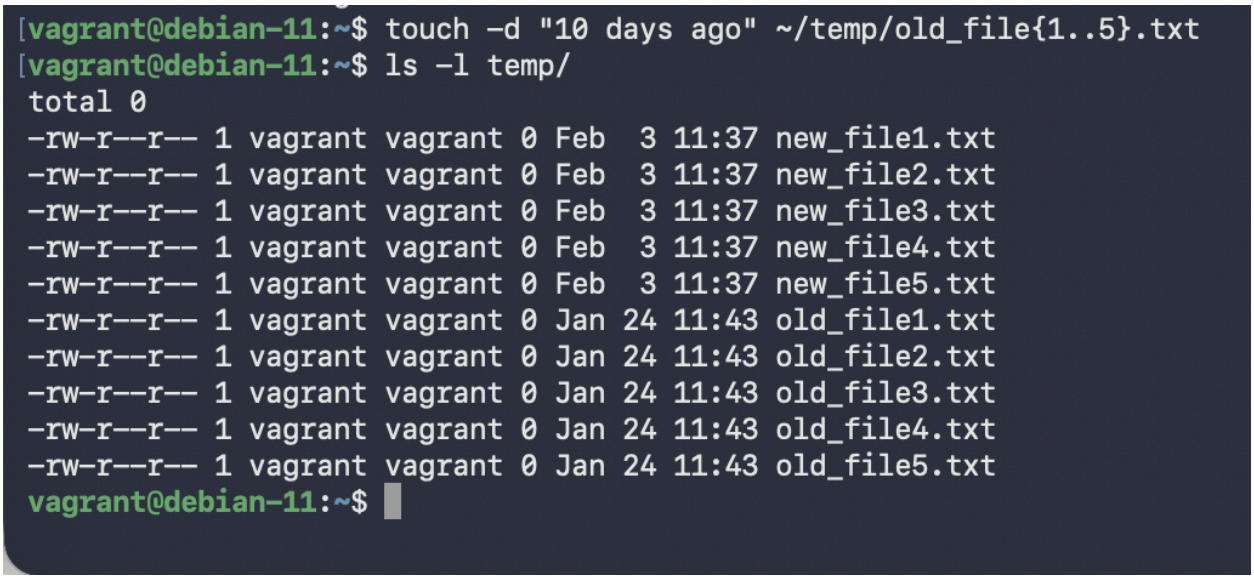
2. ⚙️ Implementation

A. The Setup

We created a testing environment with mixed file ages to verify the logic.

Bash

```
# Simulating files from 10 days ago
touch -d "10 days ago" ~/temp/old_file{1.5}.txt
```



B. The Script (`clean_temp.sh`)

Bash

```
#!/bin/bash
# Find files in 'temp' older than 7 days and delete them

find /home/vagrant/temp -type f -mtime +7 -delete
```

```
[vagrant@debian-11:~$ mkdir -p ~/temp
[vagrant@debian-11:~$ touch ~/temp/new_file{1..5}.txt
[vagrant@debian-11:~$ touch -d "10 days ago" ~/temp/old_file{1..5}.txt
[vagrant@debian-11:~$ nano ~/clean_temp.sh
[vagrant@debian-11:~$ pwd
 /home/vagrant
[vagrant@debian-11:~$ chmod +x ~/clean_temp.sh
[vagrant@debian-11:~$ crontab -e
```

new and old files creation + the script for the cronjob.

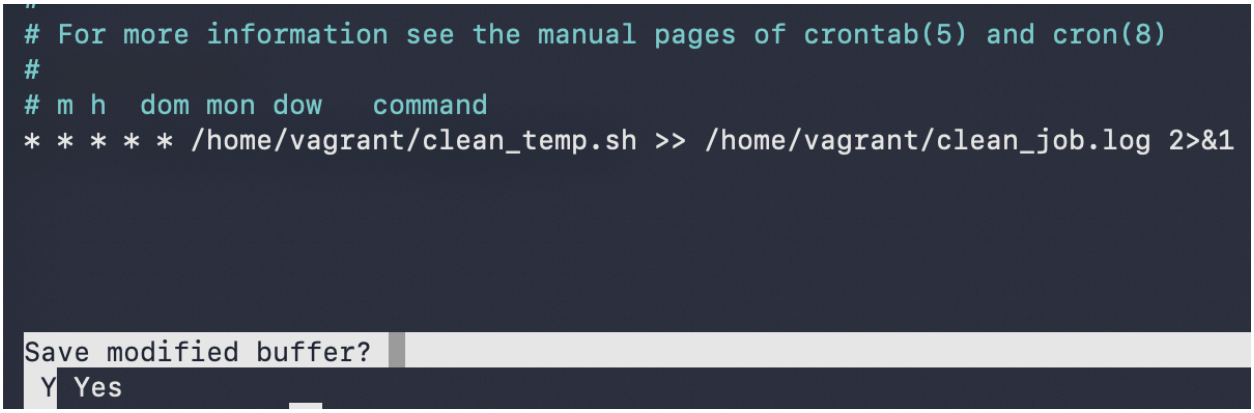
The script's content

C. The Schedule

We used `crontab -e` to schedule the job for **Midnight (00:00)** daily.

```
0 0 * * * /home/vagrant/clean_temp.sh >> /home/vagrant/clean_job.log 2>&1
```

- For testing reasons, I changed the cron time to each minute (* * * * *) :



3. 🔍 Analysis & Q&A

Q1: What happens if the script is not executable?

Answer: The Cron daemon will attempt to run the file but will fail with a `Permission denied` error. The job will not execute, and the failure is usually reported in the local mail spool or system logs.

Q2: How do you verify the cronjob is running?

Answer:

1. **System Logs:** Check `/var/log/syslog` (Debian/Ubuntu) or `/var/log/cron` (RHEL/CentOS).
    - Command: `grep CRON /var/log/syslog` (sudo can be needed)
2. **Journalctl:** `journalctl -u cron`

Q3: How do you capture errors (Logging)?

Answer: By using I/O redirection in the crontab line.

- `>>` appends the output to a file instead of discarding it.
- `2>&1` ensures that **Error** messages (Stream 2) are sent to the same file as **Output** messages (Stream 1).

4. 📸 Verification

```
[vagrant@debian-11:~$ ls -l ~/temp/
total 0
-rw-r--r-- 1 vagrant vagrant 0 Feb  3 11:37 new_file1.txt
-rw-r--r-- 1 vagrant vagrant 0 Feb  3 11:37 new_file2.txt
-rw-r--r-- 1 vagrant vagrant 0 Feb  3 11:37 new_file3.txt
-rw-r--r-- 1 vagrant vagrant 0 Feb  3 11:37 new_file4.txt
-rw-r--r-- 1 vagrant vagrant 0 Feb  3 11:37 new_file5.txt
vagrant@debian-11:~$
```

We can see that only the new files are left.

```
[vagrant@debian-11:~$ sudo grep CRON /var/log/syslog                                ]
Feb  3 09:34:24 debian-11 cron[508]: (CRON) INFO (pidfile fd = 3)
Feb  3 09:34:24 debian-11 cron[508]: (CRON) INFO (Running @reboot jobs)
Feb  3 09:17:01 debian-11 CRON[12486]: (root) CMD (   cd / && run-parts --report /etc/cron.hourly)
Feb  3 10:17:01 debian-11 CRON[12550]: (root) CMD (   cd / && run-parts --report /etc/cron.hourly)
Feb  3 11:42:01 debian-11 CRON[12748]: (vagrant) CMD (/home/vagrant/clean_temp.sh >> /home/vagrant/cle
an_job.log 2>&1)
Feb  3 11:43:01 debian-11 CRON[12770]: (vagrant) CMD (/home/vagrant/clean_temp.sh >> /home/vagrant/cle
an_job.log 2>&1)
Feb  3 11:53:01 debian-11 CRON[12834]: (vagrant) CMD (/home/vagrant/clean_temp.sh >> /home/vagrant/cle
an_job.log 2>&1)
vagrant@debian-11:~$
```

🔐 Chapter 5: Permissions & Access Control

1. 🛡️ Sudoers Configuration

Objective: Allow `alpha` to run network diagnostics without full root access.

- **Tool Installed:** `net-tools` (provides `ifconfig`).

```
vagrant@debian-11:~$ sudo apt update && sudo apt install -y net-tools
Hit:1 http://security.debian.org/debian-security bullseye-security InRelease
Hit:2 http://httpredir.debian.org/debian bullseye InRelease
Hit:3 http://httpredir.debian.org/debian bullseye-updates InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
98 packages can be upgraded. Run 'apt list --upgradable' to see them.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages will be upgraded:
  net-tools
1 upgraded, 0 newly installed, 0 to remove and 97 not upgraded.
Need to get 248 kB of archives.
After this operation, 1024 B of additional disk space will be used.
```

- **Configuration:** Edited `/etc/sudoers` via `visudo` , since it's more secure than directly editing in the sudoers file.
- **Rule Added:** `alpha ALL=(ALL) NOPASSWD: /usr/sbin/ifconfig`

```
  GNU nano 5.4                          /etc/sudoers.tmp *
# User privilege specification
root    ALL=(ALL:ALL) ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL

# See sudoers(5) for more information on "@include" directives:

@includedir /etc/sudoers.d

alpha ALL=(ALL) NOPASSWD: /usr/sbin/ifconfig
```

Verification:

> `su - alpha` -> `sudo ifconfig` -> **Works!**

```
vagrant@debian-11:~$ su - alpha
Password:
alpha@debian-11:~$ sudo ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.74.128  netmask 255.255.255.0  broadcast 192.168.74.255
        inet6 fe80::20c:29ff:fe3f:f089  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:3f:f0:89  txqueuelen 1000  (Ethernet)
        RX packets 28976  bytes 31941984 (30.4 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 9257  bytes 1021682 (997.7 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 45  memory 0x3fe00000-3fe20000

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.33.10  netmask 255.255.255.0  broadcast 192.168.33.255
        inet6 fe80::20c:29ff:fe3f:f093  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:3f:f0:93  txqueuelen 1000  (Ethernet)
        RX packets 187  bytes 20022 (19.5 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 32  bytes 3301 (3.2 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 49  memory 0x3ee00000-3ee20000
```

> `sudo apt update` -> **Permission Denied (Correct).**

```
alpha@debian-11:~$ sudo apt update

We trust you have received the usual lecture from the local System
Administrator. It usually boils down to these three things:

    #1) Respect the privacy of others.
    #2) Think before you type.
    #3) With great power comes great responsibility.

[sudo] password for alpha:
Sorry, user alpha is not allowed to execute '/usr/bin/apt update' as root on debian-11.10-a
arch64.
alpha@debian-11:~$
```

2. 📂 Directory Permissions (/opt)

- **Issue:** `curl` failed writing to `/opt/bat.zip`.

```
alpha@debian-11:~$ exit
logout
vagrant@debian-11:~$ curl -o /opt/bat.zip https://github.com/sharkdp/bat/archive/refs/tags
/v0.24.0.zip
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
Warning: Failed to create the file /opt/bat.zip: Permission denied
```

- **Reason:** `/opt` is owned by `root:root`. Standard users cannot write to it.
- **Solution:** We changed ownership of the directory to the current user *before* downloading.

`sudo chown vagrant:vagrant /opt`

`curl -L -o /opt/bat.zip https://...`

```
vagrant@debian-11:~$ sudo chown vagrant:vagrant /opt
vagrant@debian-11:~$ curl -o /opt/bat.zip https://github.com/sharkdp/bat/archive/refs/tags
/v0.24.0.zip
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
vagrant@debian-11:~$ curl -L -o /opt/bat.zip https://github.com/sharkdp/bat/archive/refs/t
ags/v0.24.0.zip
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 2971k    0 2971k    0     0  2462k      0 --:--:--  0:00:01 --:--:-- 4509k
```

3. 📦 Extraction Challenge

Scenario: The tool `unzip` was missing.

Solution: I installed the tool via `sudo apt install unzip`.

Here is the content of the unzipped bat directory

```
vagrant@debian-11:~$ cd /opt
vagrant@debian-11:/opt$ sudo ls
bat-0.24.0  bat.zip
vagrant@debian-11:/opt$ cd bat-0.24.0/
vagrant@debian-11:/opt/bat-0.24.0$ ls
CHANGELOG.md      Cargo.toml      NOTICE      build.rs      examples      tests
CONTRIBUTING.md   LICENSE-APACHE  README.md   diagnostics   rustfmt.toml
Cargo.lock        LICENSE-MIT     assets      doc           src
```

🌐 Chapter 6: Web Server & Process Debugging

1. 🎯 Objective

To install the Apache web server and troubleshoot a startup failure caused by a "zombie" process or conflicting service.

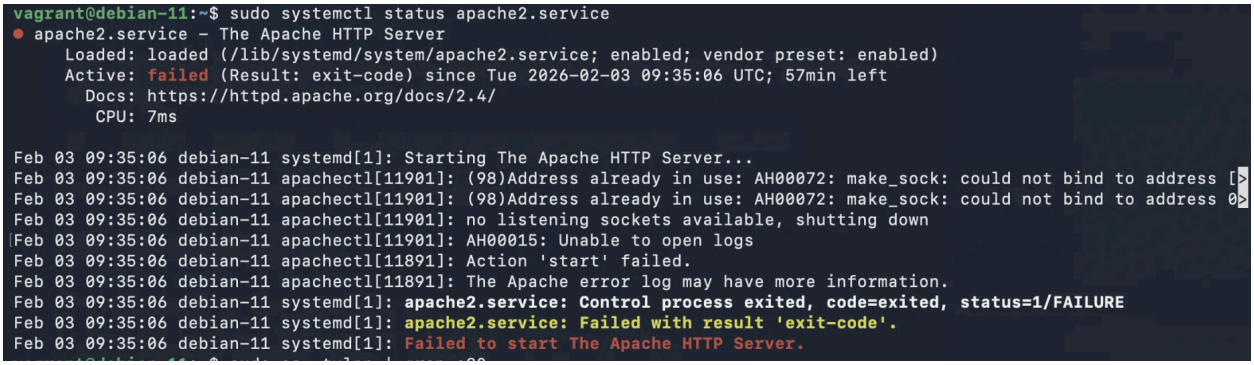
2. 🕵️ Diagnosis

The Error

After installing `apache2`, the service failed to start.

Command: `sudo systemctl start apache2`

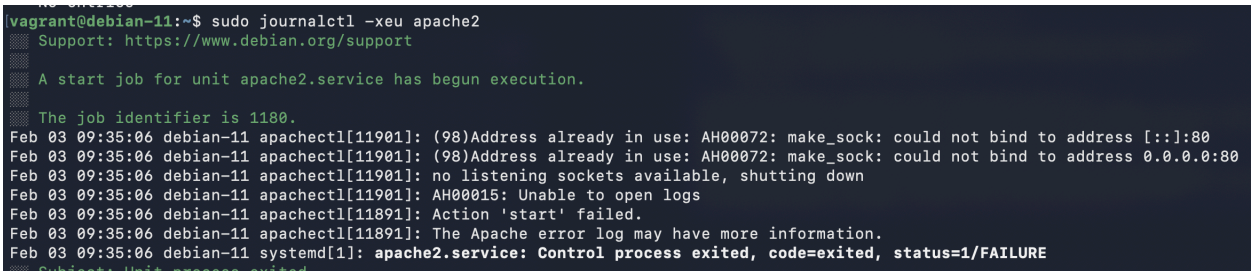
Output: `Job for apache2.service failed...`



Log Analysis (`journalctl`)

We used `journalctl -xeu apache2` to inspect the error logs.

> **Log Error:** `(98)Address already in use: AH00072: make_sock: could not bind to address [::]:80`
>
> *Analysis: This confirms that Port 80 is already occupied by another process.*



Process Identification (`ss`)

We used `ss -tulpn` to identify the conflicting process:

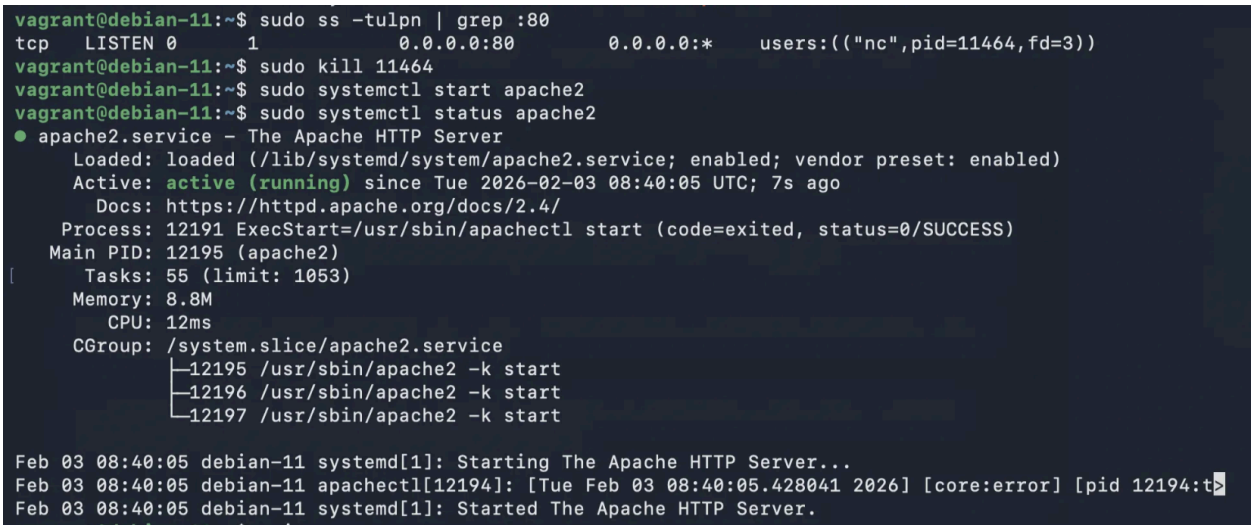
`ss -tulpn | grep :80`

Culprit Found: The utility `nc` (Netcat) was listening on port 80.

3. 🛠️ Resolution

We terminated the conflicting process and successfully started the web server.

```
# 1. Kill the Netcat process
sudo kill <PID>

# 2. Start Apache
sudo systemctl start apache2

# 3. Verify Status
sudo systemctl status apache2
```



4. 🌍 Verification

Accessed the web server via the browser.

In order to find the IP Address and port used by the web server, I ran this command in the mac terminal:

`vagrant ssh -c "hostname -I”`

```
[hamza@Hamzas-MacBook-Pro:~/dev/labs/vm|⇒  vagrant ssh -c "hostname -I"
192.168.74.128 192.168.33.10
hamza@Hamzas-MacBook-Pro:~/dev/labs/vm|⇒
```

The first address is the host machine's address. the second is for the guest machine.

Let's go to `192.168.74.128` :