Reflective Learning Journal Template – BRG27-ISEA

Student Name: Josh Pecson
Student ID: CT0387224@kaplan.edu.sg
GitHub Repository Link: https://github.com/GR11M
Video Walkthrough Link: https://shorturl.at/AqJ71

I currently work as an Engineer in the marine industry. While my professional background is rooted in engineering and vessel operations, I have made the decision to pivot into cybersecurity, with a specific interest in Digital Forensics and Incident Response (DFIR).

Throughout this module on Introduction to Server Environments and Architecture, I approached each lab as someone preparing to investigate, secure, and analyse systems. This journal captures my learning journey, the mistakes I made, and the steps I took to correct them.

Session 1 (A) – Virtualisation & Linux Familiarisation

Lab Tasks:

- Created GitHub repository and cloned it locally

- Installed Ubuntu using VirtualBox

- Explored core Linux commands and system queries

This session introduced me to virtualisation tools and basic Linux navigation. Setting up the VM was smooth, but I ran into some confusion around resource allocation (RAM, disk size), and I also did not fully understand the networking mode differences. Once the VM was running, I got a better sense of how virtualisation works and how it lets us create testable environments, something that is crucial for hands-on learning and lab testing. Virtualisation is like turning one computer into many mini computers, each running separately on the same machine.

Commands like `uname -a`, `hostnamectl`, and `ls -lah` helped me view system information and track directories. These are very relevant for forensic investigations, especially for timestamp verification and system state identification.

Also learnt the `whatis` command which proved to be very useful moving forward being able to quickly tell me what the command roughly does.

`uname -a`

```
joshpecson@joshpc:~$ uname -a
Linux joshpc 6.11.0-29-generic #29~24.04.1-Ubuntu SMP PREEMPT_DYNAMIC Thu Jun 26 14:16:59 UTC 2 x86_64 x86_64 x86_64 GNU/Linux
joshpecson@joshpc:~$ whatis hostname
hostname (1)         - show or set the system's host name
hostname (5)         - Local hostname configuration file
hostname (7)         - hostname resolution description
```

`ls -lah`

```
joshpecson@joshpc:~$ ls -lah
total 152K
drwxr-x--- 17 joshpecson joshpecson 4.0K Jul  6 14:56 .
drwxr-xr-x  3 root       root       4.0K Jul  2 14:21 ..
```

`hostnamectl`

```
joshpecson@joshpc:~$ hostnamectl
 Static hostname: joshpc
       Icon name: computer-vm
         Chassis: vm
      Machine ID: 32440525fc0042efa5c72665834b6bb1
         Boot ID: c6c9200c55264bebaf9dbac01689bdb2
  Virtualization: oracle
Operating System: Ubuntu 24.04.2 LTS
          Kernel: Linux 6.11.0-29-generic
    Architecture: x86-64
 Hardware Vendor: innotek GmbH
  Hardware Model: VirtualBox
Firmware Version: VirtualBox
   Firmware Date: Fri 2006-12-01
    Firmware Age: 18y 7month 5d
```

Using `whois` and `nslookup` introduced me to DNS metadata. The interest leans towards, who made the request and where it came from, rather than what the actual IP address is. That would be valuable in DFIR work for tracing the source of attack or identifying anomalies.

Session 1 (B) – Apache2, Firewall, SSH, and Permissions

Lab Tasks:

- Installed Apache2 and served a webpage

- Used `nmap` to scan open ports

- Enabled `ufw` firewall and allowed SSH

- Set up SSH access and tested cross-VM connections

- Practised file compression and user permissions

Web Server (Apache2)

This part of the lab felt more like setting up a real-world system. Running `nmap` before and after starting services like Apache2 helped me understand how exposed ports reflect what is currently running.

`sudo systemctl status apache2`

```
ubuntu@ip-172-31-34-119:~$ sudo systemctl status apache2
● apache2.service - The Apache HTTP Server
     Loaded: loaded (/usr/lib/systemd/system/apache2.service; enabled; preset: enabled)
     Active: active (running) since Sun 2025-06-29 06:37:51 UTC; 38s ago
       Docs: https://httpd.apache.org/docs/2.4/
   Main PID: 2276 (apache2)
      Tasks: 55 (limit: 1124)
     Memory: 5.2M (peak: 5.5M)
        CPU: 37ms
     CGroup: /system.slice/apache2.service
             ├─2276 /usr/sbin/apache2 -k start
             ├─2279 /usr/sbin/apache2 -k start
             └─2280 /usr/sbin/apache2 -k start

Jun 29 06:37:51 ip-172-31-34-119 systemd[1]: Starting apache2.service - The Apache HTTP Server...
Jun 29 06:37:51 ip-172-31-34-119 systemd[1]: Started apache2.service - The Apache HTTP Server.
```

Ensuring that my apache2 service is active.

I served the web page to see if it is working correctly

This allowed me to verify that port 80 was open and actively listening for HTTP requests

`sudo nmap <target> -sV -p80`

```
joshpecson@joshpc:~$ sudo nmap 192.168.1.20 -sV -p80
[sudo] password for joshpecson:
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-07-06 15:38 UTC
Nmap scan report for 192.168.1.20 (192.168.1.20)
Host is up (0.00028s latency).

PORT   STATE    SERVICE VERSION
80/tcp filtered http
```

Shows that port 80 is filtered, could be due to a firewall.

`sudo nmap <target> -sV -p22`

```
joshpecson@joshpc:~$ sudo nmap 192.168.1.20 -sV -p22
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-07-06 15:43 UTC
Nmap scan report for 192.168.1.20 (192.168.1.20)
Host is up (0.00040s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH for_Windows_9.5 (protocol 2.0)
```

Shows that port 22 is open.

Enabling the `sudo ufw enable` firewall taught me how security starts with configuration where ports need to be allowed manually. SSH-ing into my neighbour's VM (my kali VM) gave me insight into remote system access and user isolation.

`ip a`

Identified neighbour's IP address.

```
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:f6:f8:e0 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.207/24 brd 192.168.1.255 scope global dynamic noprefixroute eth1
       valid_lft 14379sec preferred_lft 14379sec
    inet6 fe80::5282:eeb6:30a8:bef/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Lesson Learnt:

Initially, I forgot to open port 22 and spent time troubleshooting until I checked firewall rules. That taught me to always verify UFW status.

`sudo ufw allow 80/tcp`

`sudo ufw allow 22/tcp`

`sudo ufw statuts verbose`

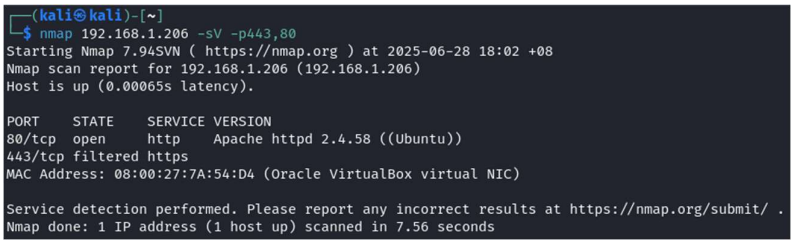
```
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), disabled (routed)
New profiles: skip

To                         Action      From
--                         ------      ----
80/tcp                     ALLOW IN    Anywhere
22/tcp                     ALLOW IN    Anywhere
```

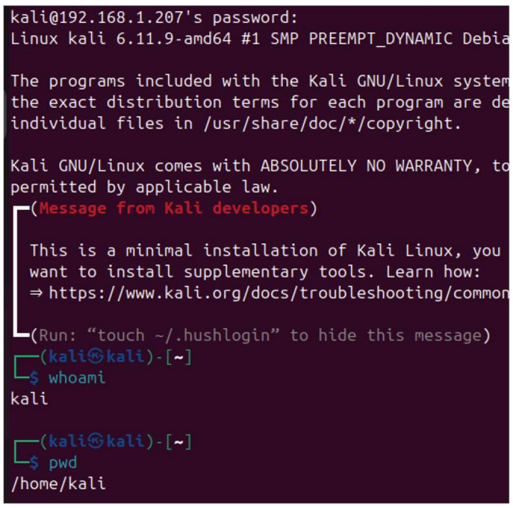
status → active

SSH Testing

From my neighbour's machine, tried to scan my machine to test if the ports are working.
`nmap <target_ip> -sV -p443,80`

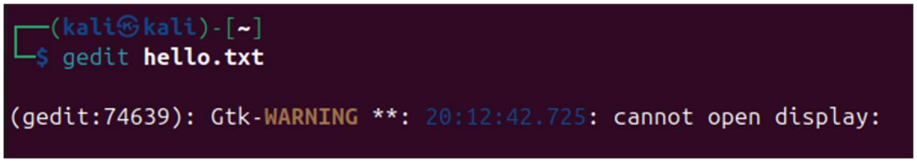


I was able to successfully initiate an SSH session from my machine (192.168.1.206) to my neighbour's machine (192.168.1.207), confirming that the SSH service is running and accessible.



It was interesting to experience controlling a remote machine through SSH. To test the connection, I created files from the logged-in user account and later verified them directly on my neighbour's machine. This helped reinforce the concept that SSH allows full access to remote systems as if I were using them locally, including the ability to create and modify files.

One of the lab activities was launching gedit while ssh'ed into my neighbour's machine.



Why was it unsuccessful?

SSH does not send GUI applications by default, so graphical programs like `gedit` cannot launch because there is no graphical display available over the SSH session.

Instead, text editors like `nano`, run entirely within the shell that do not require a graphical environment.

I went ahead to try the `scp` command. I always get confused with the syntax order of this command. The correct order should be:

`scp <local_file/filepath> <user@destination_adress>:/full/file/path`

```
joshpecson@joshpecson:~/Books$ scp Books.tar alice@192.168.1.207:/home/alice
alice@192.168.1.207's password:
Books.tar                                                                   100% 1120KB  55.0MB/s   00:00
```

-r flag used for recursive copying, to include all files in the directory

```
joshpecson@joshpecson:~$ scp -r /home/joshpecson/Books/ alice@192.168.1.207:/home/alice
alice@192.168.1.207's password:
12-0.txt                                                                    100%  173KB  30.5MB/s   00:00
76-0.txt                                                                    100%  577KB  33.3MB/s   00:00
36-0.txt                                                                    100%  358KB  45.2MB/s   00:00
Books.tar                                                                   100% 1120KB  33.7MB/s   00:00
12-0.txt                                                                    100%  173KB  44.4MB/s   00:00
76-0.txt                                                                    100%  577KB  31.6MB/s   00:00
36-0.txt                                                                    100%  358KB  47.3MB/s   00:00
```

File Compressions

Archiving is about grouping, while compression is about shrinking. Both tools work together to allow for clean backups and space efficiency.

`tar cf <archive_file_name> <file_to_archive>`

```
joshpecson@joshpecson:~/Books$ tar cf Top10Books.tar Top10Books
joshpecson@joshpecson:~/Books$ ls
12-0.txt  36-0.txt  76-0.txt  Books  Books.tar  Top10Books  Top10Books.tar
```

Using bzip2

```
joshpecson@joshpecson:~/Books$ bzip2 Top10Books.tar
joshpecson@joshpecson:~/Books$ ls
12-0.txt  36-0.txt  76-0.txt  Books  Books.tar  Top10Books  Top10Books.tar.bz2
```

Tar vs. Bzip2

`tar` is used to archive multiple files into a single file and is commonly applied to entire directories. On its own, tar does not compress the data, it groups the files. In contrast, `bzip2` is a compression tool meant to reduce the size of a single file. By first using tar to bundle the files, we create a single `.tar` file, which can then be effectively compressed using `bzip2` to produce a `.tar.bz2` file. This process makes it easier to store or transfer multiple files as one compressed file.

User Permissions

Created 3 new users
`sudo adduser <username>`

```
joshpecson@joshpecson:/home$ ls
alice  bob  joshpecson  mallory
```

To give alice read, write and execute permission
`sudo chown -R alice /home/shared`

```
joshpecson@joshpecson:/home$ sudo chown -R alice /home/shared
joshpecson@joshpecson:/home$ ls -l
total 20
drwxr-x---  2 alice      alice      4096 Jun 28 14:17 alice
```

Performed `ls -l` to check on user ownership of /home/shared directory

Created new group called sharedgroup for shared permissions with Bob since Bob can also read, write but NOT execute.

```
joshpecson@joshpecson:/home$ sudo groupadd sharedgroup
joshpecson@joshpecson:/home$
```

This is to show that `/home/shared` directory group owner is under `sharedgroup`
This is done to provide a secure and organised way to manage access.

```
joshpecson@joshpecson:/home$ sudo ls -l /home/shared
total 4
-rwxr-x--- 1 alice sharedgroup 11 Jun 28 17:14 file1
-rwxr-x--- 1 alice sharedgroup  0 Jun 28 14:24 file10
```

It supports the principles of segregation of duty and least privilege, which are important to secure environments with multiple users.

`groups <username>` to check which group(s) user is in

```
joshpecson@joshpecson:/home$ groups alice
alice : alice users sharedgroup
joshpecson@joshpecson:/home$ groups bob
bob : bob users sharedgroup
```

Ensure alice and bob belong in the same group called `sharedgroup`

`sudo chmod -R 750 /home/shared`

```
joshpecson@joshpecson:/home$ sudo chmod -R 750 /home/shared
```

One of the lab challenges was to set group permissions that allow read and execute access only, while preventing write access for group members. Calling the `750` argument applying a format where:

- The owner has full access (7 → read, write, execute),

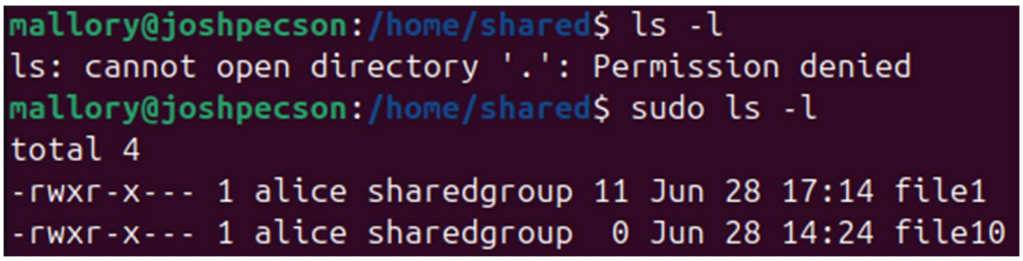
- The group has read and execute permissions only (5 → no write),
- Others have no access (0).

Here are some commonly used permission modes:

Symbolic	Octal	Owner	Group	Others
rwxrwxrwx	777	Read, write, execute	Read, write, execute	Read, write, execute
rwxr-xr-x	755	Read, write, execute	Read, execute	Read, execute
rwxr-x---	750	Read, write, execute	Read, execute	None
rwx------	700	Read, write, execute	None	None
rw-rw-r--	664	Read, write	Read, write	Read
rw-r--r--	644	Read, write	Read	Read
rw-------	600	Read, write	None	None

Extra Challenge

One of the challenges involved adding a user named Mallory to the `sudoers` group to grant her administrative privileges. This was done by running `sudo usermod -aG mallory`, followed by a test using `su – mallory` and executing a command with `sudo`. Once confirmed, Mallory was able to run commands with elevated privileges. However, she still could not access /home/shared directly. This highlighted an important concept.



The screenshot shows that Mallory cannot list files in /home/shared due to permission restrictions. However, using `sudo` runs the command as the root user, allowing her to bypass those restrictions and view the contents. This confirms that file permissions control access by default, and `sudo` grants temporary access through root privileges which will supersede permissions modes.

Session 2 (A) – Total Cost of Ownership (TCO)

Lab Task:

- Compared 5-year TCO of cloud server and physical server

It was not surprising to see that the cloud server was much more cost-effective than the physical server. From the electricity to power up the servers, to the air conditioning

cooling the environment. These are one of many contributing factors that would rack up the electricity bill.

Decisions like whether to run a physical server or a cloud server require TCO-style thinking. This includes considering not just upfront costs, but also maintenance and consumables required in the long run.

Cost Component	Cloud (Monthly)	Cloud (Yearly)	Cloud (5-Years)	On-Prem (Monthly)	On-Prem (Yearly)	On-Prem (5-Years)
Hardware	/	/	/	$1,200	$14,400	$72,000
Software	$150	$1,800	$9,000	$100	$1,200	$6,000
Licensing	$200	$2,400	$12,000	$150	$1,800	$9,000
Maintenance/Support	$100	$1,200	$6,000	$250	$3,000	$15,000
Power & Facilities	/	/	/	$500	$6,000	$30,000
Total (per Year)	**$450**	**$5,400**	**$27,000**	**$2,200**	**$26,400**	**$132,000**

A simple breakdown of the two comparisons

Session 2 (B) – Cloud Server Setup & Bash Scripting

Lab Tasks:

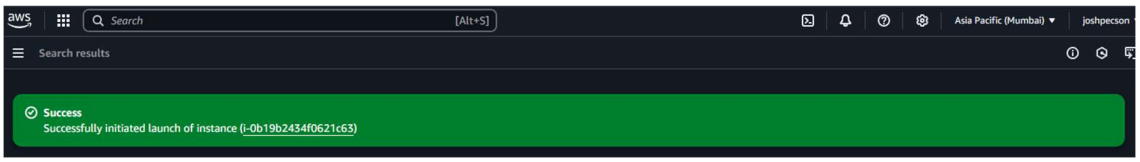
- Launched an EC2 Ubuntu server

- Installed Apache2 and hosted a webpage

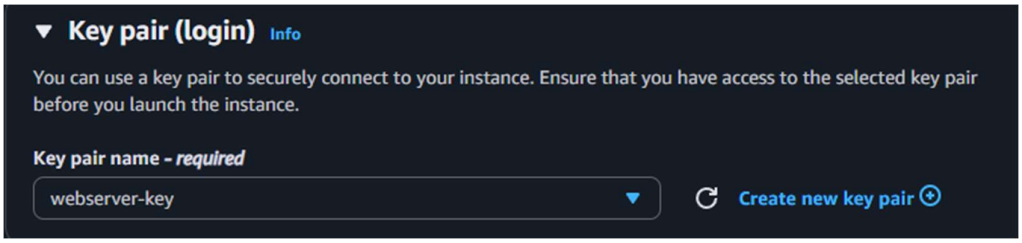
- Simple Bash Scripting

AWS EC2 Instance

Launching an EC2 instance felt like taking a step into the professional IT world. Compared to VirtualBox, AWS required more care with key pairs, security groups, and cost tracking.



The private key reinforces how authentication works in cloud environments, using key pairs to verify identity. It also highlights how this method provides a more secure way to control access and reduce the risk of unauthorized users gaining entry.



Upon successful login, I did the usual `sudo apt update` and installed apache2 and other useful tools to continue tinkering about using the EC2 instance.

Configured my security group as well to be able to listen to for requests from the following ports:

sgr-07ea608d1880e646d	IPv4	HTTP	TCP	80
sgr-09902d1851c88d9c7	IPv4	All ICMP - IPv4	ICMP	All
sgr-020a8bec7b7ce8e7e	IPv4	SSH	TCP	22

Allow ICMP rule was added to allow ping command to work for some latency tests to be done.

Tried pinging two difference sources: murdoch.edu.au and google.com
`ping murdoch.edu.au`

```
ubuntu@ip-172-31-34-119:~$ ping murdoch.edu.au
PING murdoch.edu.au (20.43.99.27) 56(84) bytes of data.
64 bytes from 20.43.99.27: icmp_seq=1 ttl=110 time=139 ms
64 bytes from 20.43.99.27: icmp_seq=2 ttl=110 time=139 ms
64 bytes from 20.43.99.27: icmp_seq=3 ttl=110 time=139 ms
64 bytes from 20.43.99.27: icmp_seq=4 ttl=110 time=139 ms
^C
--- murdoch.edu.au ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 139.286/139.311/139.345/0.021 ms
```

139ms average latency– Not too slow, but not fast either.

`ping google.com`

```
ubuntu@ip-172-31-34-119:~$ ping google.com
PING google.com (142.250.70.78) 56(84) bytes of data.
64 bytes from pnbomb-ab-in-f14.1e100.net (142.250.70.78): icmp_seq=1 ttl=117 time=2.23 ms
64 bytes from pnbomb-ab-in-f14.1e100.net (142.250.70.78): icmp_seq=2 ttl=117 time=2.21 ms
64 bytes from pnbomb-ab-in-f14.1e100.net (142.250.70.78): icmp_seq=3 ttl=117 time=2.27 ms
64 bytes from pnbomb-ab-in-f14.1e100.net (142.250.70.78): icmp_seq=4 ttl=117 time=2.25 ms
64 bytes from pnbomb-ab-in-f14.1e100.net (142.250.70.78): icmp_seq=5 ttl=117 time=2.26 ms
64 bytes from pnbomb-ab-in-f14.1e100.net (142.250.70.78): icmp_seq=6 ttl=117 time=2.23 ms
^C
--- google.com ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5007ms
rtt min/avg/max/mdev = 2.205/2.241/2.269/0.021 ms
```

2ms average latency – High speed

This is why location of the physical server matters. The closer it is to the user, the faster the data travels. Choosing a region near your target user helps reduce latency and improve performance. My EC2 instance benefits from proximity to Google infrastructure, which explains the low latency.

There are many alternative cloud servers including Google Cloud Platform, Microsoft Azure, Oracle Cloud, and many others.

Exploring HTML

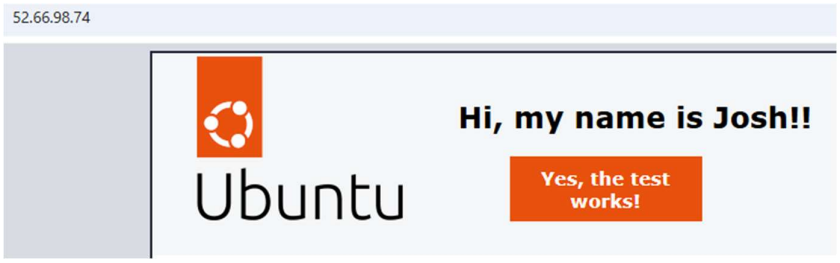
Testing HTML code reminded me of the early blogging days, when everyone was figuring out how to embed an auto-play media player or an archive section with a drop box feature, felt like works of web development. This lab brought that same curiosity back, now in a more technical angle.

`sudo nano /var/www/html/index.html`

```
  <span style="margin-top: 1.5em;" class="floating_element">
    Hi, my name is Josh!!
  </span>
</div>
<div class="banner">
  <div id="about"></div>
  Yes, the test works!
</div>
```

`index.html` located at `/var/www/html/` is the default web page served by Apache2

Checked to see if the changes were reflected on the web page



Web page header successfully modified.

Bash Scripting for System Monitoring

Imagine writing a series of commands and executing it sequentially, line by line. Bash scripting allows you to automate these commands in a single file. Common use cases would be automating updates, file management or even backups. This provides more consistent and time-efficient results.

Simple script of system monitoring

```
#!/bin/bash

echo "System Resource Monitoring"

# Number of iterations
read -p "How many times would you like to monitor the system? " iterations

# Loop based on user input
for ((i=1; i<=iterations; i++))
do
    echo "----- Monitoring $i -----"

    # Display CPU usage
    echo "CPU Usage:"
    top -b -n1 | grep "Cpu(s)"

    # Display Memory usage
    echo "Memory Usage:"
    free -h

    # Display Disk usage
    echo "Disk Usage:"
    df -h | grep "^/dev"

    echo "-----------------------"
```

`./resource_monitor.sh`

```
ubuntu@ip-172-31-36-187:~/LabFiles$ ./resource_monitor.sh
System Resource Monitoring
How many times would you like to monitor the system? 2
----- Monitoring 1 -----
CPU Usage:
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
Memory Usage:
               total        used        free      shared  buff/cache   available
Mem:           957Mi       401Mi       138Mi       1.4Mi       611Mi       556Mi
Swap:             0B          0B          0B
Disk Usage:
/dev/root       6.8G  2.5G  4.3G  37% /
/dev/xvda16     881M  148M  672M  18% /boot
/dev/xvda15     105M  6.2M   99M   6% /boot/efi
-----------------------
----- Monitoring 2 -----
CPU Usage:
%Cpu(s):  9.1 us,  0.0 sy,  0.0 ni, 90.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
Memory Usage:
               total        used        free      shared  buff/cache   available
Mem:           957Mi       401Mi       138Mi       1.4Mi       611Mi       556Mi
Swap:             0B          0B          0B
Disk Usage:
/dev/root       6.8G  2.5G  4.3G  37% /
/dev/xvda16     881M  148M  672M  18% /boot
/dev/xvda15     105M  6.2M   99M   6% /boot/efi
-----------------------
Monitoring complete.
```

By simply executing the script, I was able to extract multiple types of information at on go. This highlights how automation through scripting can streamline daily tasks and reduce manual effort.

Lesson Learnt:

I kept forgetting to change permissions on my script. Just learnt that when using commands like touch, nano or echo permissions usually default to `644`. Newly created files do not have default execute permissions.

Default permissions when a new file is created.

```
-rw-rw-r-- 1 ubuntu ubuntu  30 Jul  2 14:40 notes.txt
```

`chmod +x notes.txt`

```
~/LabFiles$ chmod +x notes.txt
```

Permissions shows that users, groups and others can read and execute.

```
-rwxrwxr-x 1 ubuntu ubuntu  30 Jul  2 14:40 notes.txt
```

Only users and groups can write, but not others.

Going back to my reflection on permission modes from the earlier session, I can also say that `chmod +x <file.name>` similar to `chmod 755 <file.name>` both give the same permission modes. The main difference is that `chmod +x` will add execute on top of the existing permission set while `chmod 755` will overwrite the permission set to exactly `rwxr-xr-x`.

Session 3 (A) –

Lab Tasks:

- Configured A record pointing to public IP

- Verified domain with dig, nslookup, and ping

DNS Configuration

DNS (Domain Naming System) is what allows us to be able to read human-readable forms instead of IP addresses. In this lab, I learned how to link a domain name to my cloud server using DuckDNS. This involved creating an A record that pointed my domain to my EC2 instance's public IP.

Domain name: joshict171.duckdns.org

Checked to see if the domain is linked to the cloud server correctly.

`nslookup <domain_name>`

```
Non-authoritative answer:
Name:   joshict171.duckdns.org
Address: 43.204.25.73
```

Non-authoritative indicates that the DNS is coming from a cache as opposed to the direct DuckDNS server. It is still valid and confirms that the domain resolves to the correct IP address.

Using the `dig` command (domain information groper) to check if the domain name is pointing to the correct server.

`dig joshict171.duckdns.org`

```
ubuntu@ip-172-31-36-187:~$ dig joshict171.duckdns.org

; <<>> DiG 9.18.30-0ubuntu0.24.04.2-Ubuntu <<>> joshict171.duckdns.org
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 34939
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;joshict171.duckdns.org.                IN      A

;; ANSWER SECTION:
joshict171.duckdns.org. 60      IN      A       43.204.25.73

;; Query time: 191 msec
;; SERVER: 127.0.0.53#53(127.0.0.53) (UDP)
;; WHEN: Mon Jun 30 16:28:51 UTC 2025
;; MSG SIZE  rcvd: 67
```

The A record in the answer section confirms that my domain is correctly pointing to my cloud server's public IP address

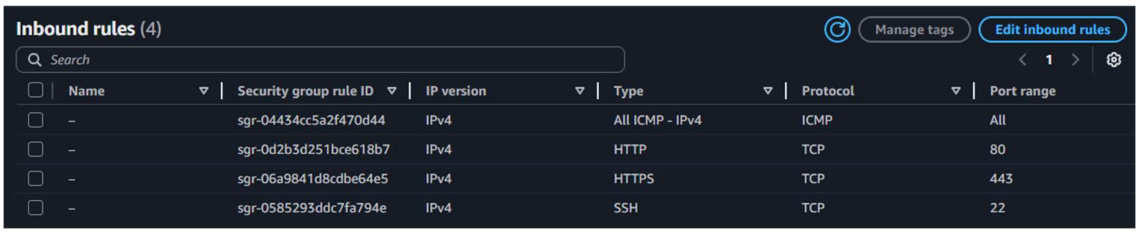
We can also use the `ping` command for further validation.

`ping <domain_name>`

```
ubuntu@ip-172-31-36-187:~$ ping joshict171.duckdns.org
PING joshict171.duckdns.org (43.204.25.73) 56(84) bytes of data.
^C
--- joshict171.duckdns.org ping statistics ---
8 packets transmitted, 0 received, 100% packet loss, time 7146ms
```

No response

Added new inbound rule to allow ICMP traffic on my cloud server.



This will allow my cloud server to listen for ICMP requests.

`ping joshict171.duckdns.org`

```
ubuntu@ip-172-31-36-187:~$ ping joshict171.duckdns.org
PING joshict171.duckdns.org (43.204.25.73) 56(84) bytes of data.
64 bytes from ec2-43-204-25-73.ap-south-1.compute.amazonaws.com (43.204.25.73): icmp_seq=1 ttl=63 time=0.337 ms
64 bytes from ec2-43-204-25-73.ap-south-1.compute.amazonaws.com (43.204.25.73): icmp_seq=2 ttl=63 time=0.435 ms
64 bytes from ec2-43-204-25-73.ap-south-1.compute.amazonaws.com (43.204.25.73): icmp_seq=3 ttl=63 time=0.362 ms
64 bytes from ec2-43-204-25-73.ap-south-1.compute.amazonaws.com (43.204.25.73): icmp_seq=4 ttl=63 time=0.465 ms
^C
--- joshict171.duckdns.org ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3076ms
rtt min/avg/max/mdev = 0.337/0.399/0.465/0.052 ms
```

This shows that the domain name is resolving to the correct address, and that it is active and responsive over the network

Session 3 (B)

Lab Tasks:

- Installed Certbot and requested certificate via DNS challenge

- Updated Apache config with Let's Encrypt cert

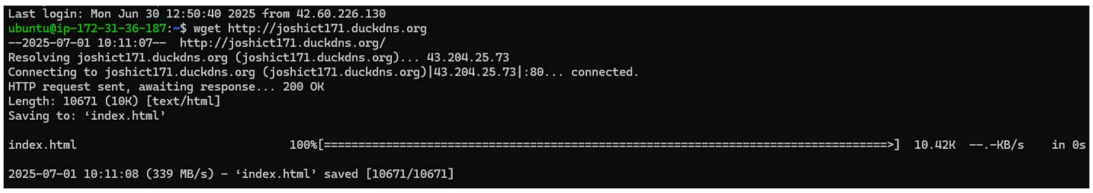
- Verified with curl and browser

- Improved backup script and integrated `cron` +` scp`

HTTPS & Certbot

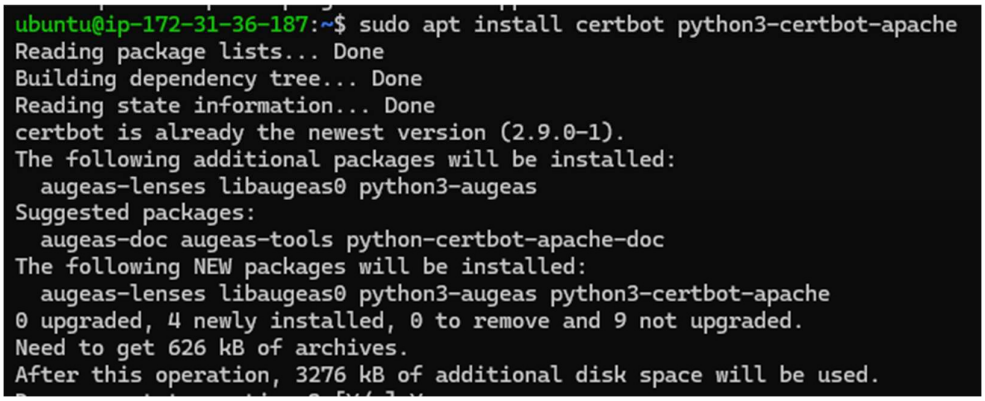
HTTPS encrypts data between user's browsers and the web page to protect against malicious interference. I used `certbot` with Let's Encrypt to generate an SSL certificate to harden my web page. Setting up HTTPS is crucial, especially for public facing web services that are more prone targets for potential threat actors.

`wget http://joshict171.duckdns.org`



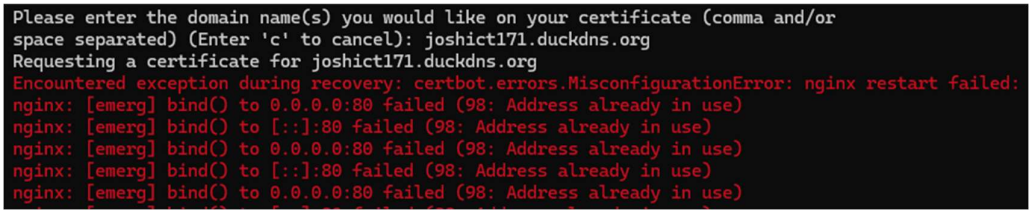
A quick test before proceeding to ensure my web server is up and running. If `index.html` was not downloaded successfully, it could mean that the web server is not running.

`sudo apt install certbox python3-certbot-apache`
`sudo certbox --nginx`



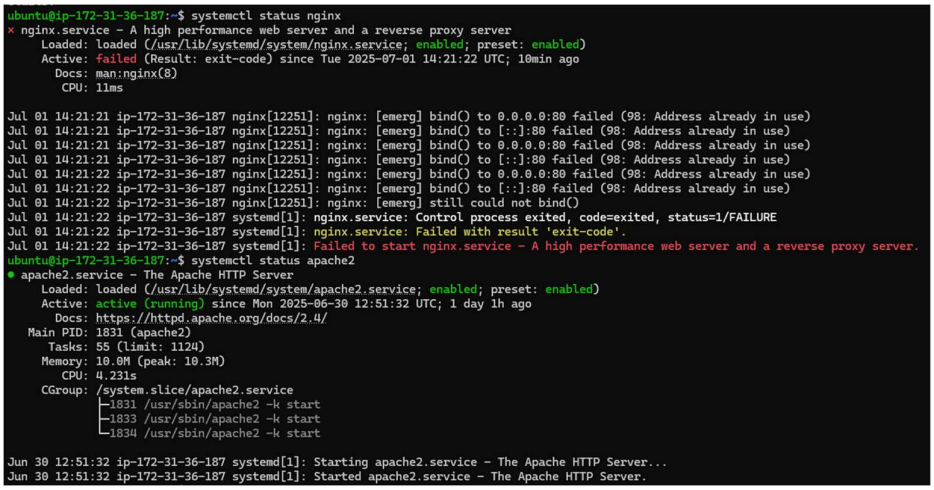
I had a brain fog moment and ran `certbot` for Nginx service, instead of the intended Apache2 service

A bunch of error messages flooded my terminal. This is due to my Apache2 service already running live on port 80.



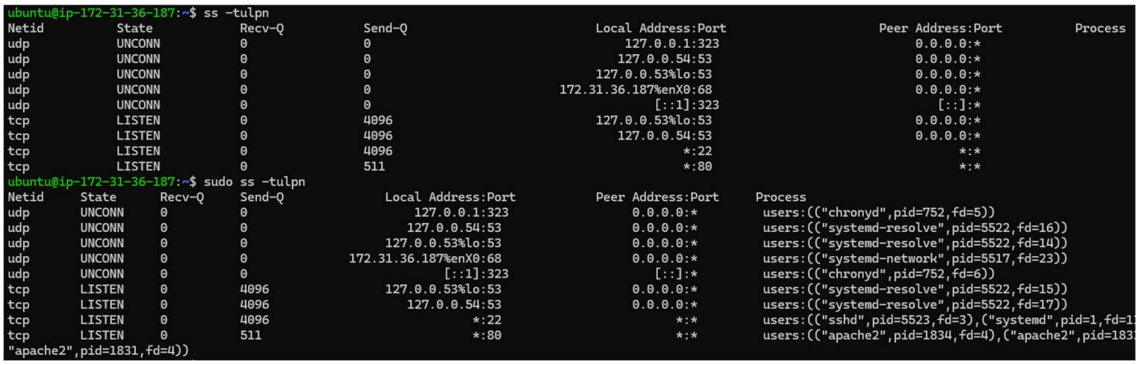
Since only one service can listen on a specific port at a time, Certbot could not start the Nginx plugin for validation. This conflict is a common issue when multiple web services are installed on the same machine.

Shows that both Nginx and Apache2 are enabled, but only Apache2 was active.



Active: active (running)

`sudo ss -tulpn`



This further validates that Apache2 service is running on port 80. Since only one service can bind to a specific port, it is likely that Apache2 claimed port 80 first because it was installed and running before Nginx.

`sudo certbot --apache`

Failed.

Lesson Learnt:

Let's Encrypt failed to validate the domain using Apache's HTTP-01 challenge because DuckDNS does not provide CAA records. Without CAA support, DuckDNS cannot

respond to Let's Encrypt's CAA checks during domain validation, which caused the challenge to fail.

As an alternative, I switched to the DNS-01 method. This approach does not rely on CAA checks and will allow the certificate request to succeed.

DNS-01 Challenge

The main difference lies within how domain ownership is verified. HTTP-01 verifies domain ownership using a token file served from the web server.

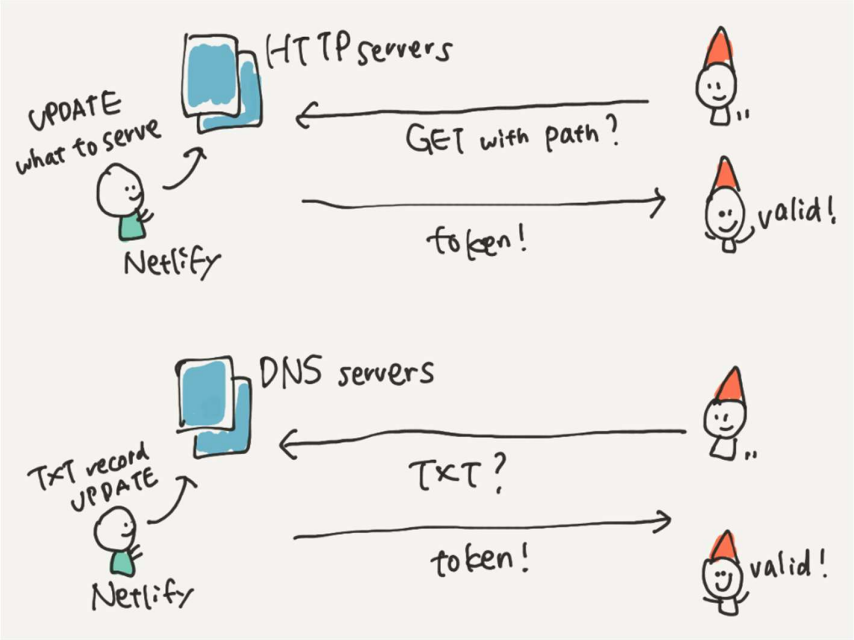


Image credit: Netlify Blog – Enabling Free Wildcard Domain Certificates with Let's Encrypt

DNS-01 uses a DNS TXT record, with the token typically provided through an API key stored in a .ini file.

DuckDNS API token found on the web page's dashboard

Created a credential file by inserting the token into `.ini` text file.

```
  GNU nano 7.2
dns_duckdns_token = 48291c96-6e02-4008-bf48-910f8e1c1cd6
```

The .ini file is a plain text configuration file that now stores my DuckDNS API token. Certbot reads this token to authenticate with DuckDNS and automatically create a TXT record, used to verify domain ownership.

I used Certbot along with the DuckDNS plugin and newly created credential file to request an SSL certificate for my domain.

```
Requesting a certificate for joshict171.duckdns.org

Successfully received certificate.
Certificate is saved at: /etc/letsencrypt/live/joshict171.duckdns.org/fullchain.pem
Key is saved at:         /etc/letsencrypt/live/joshict171.duckdns.org/privkey.pem
This certificate expires on 2025-09-29.
These files will be updated when the certificate renews.
Certbot has set up a scheduled task to automatically renew this certificate in the background.
```

Certificates are found at `/etc/letsencrypt/live/joshict171.duckdns.org/`

SSL Certificate and SSL Private Key

`sudo a2enmod ssl`

```
ubuntu@ip-172-31-36-187:~$ sudo a2enmod ssl
Considering dependency mime for ssl:
Module mime already enabled
Considering dependency socache_shmcb for ssl:
Enabling module socache_shmcb.
Enabling module ssl.
See /usr/share/doc/apache2/README.Debian.gz on how to configure SSL and create self-signed certificates.
To activate the new configuration, you need to run:
  systemctl restart apache2
```

`sudo a2ensite default-ssl.conf`

`sudo systemctl restart apache2`

```
ubuntu@ip-172-31-36-187:~$ sudo a2ensite default-ssl.conf
Enabling site default-ssl.
To activate the new configuration, you need to run:
  systemctl reload apache2
ubuntu@ip-172-31-36-187:~$ sudo systemctl restart apache2
```

I updated the SSLCertificateFile and SSLCertificateKeyFile directives in my Apache config to point to the correct Let's Encrypt certificate and private key.

```
SSLCertificateFile      /etc/letsencrypt/live/joshict171.duckdns.org/fullchain.pem
SSLCertificateKeyFile   /etc/letsencrypt/live/joshict171.duckdns.org/privkey.pem
```

This change would enable my server to serve HTTPS traffic securely with a trusted certificate issued for my domain.

Lesson Learnt:

ServerName was incorrectly set as webmaster@localhost, causing: an error.
`curl -I <domain_name>`

```
ubuntu@ip-172-31-36-187:~$ curl -I https://joshict171.duckdns.org
curl: (60) SSL: no alternative certificate subject name matches target host name 'joshict171.duckdns.org
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.
```

Changed the hostname to my domain name in my Apache2 config file.

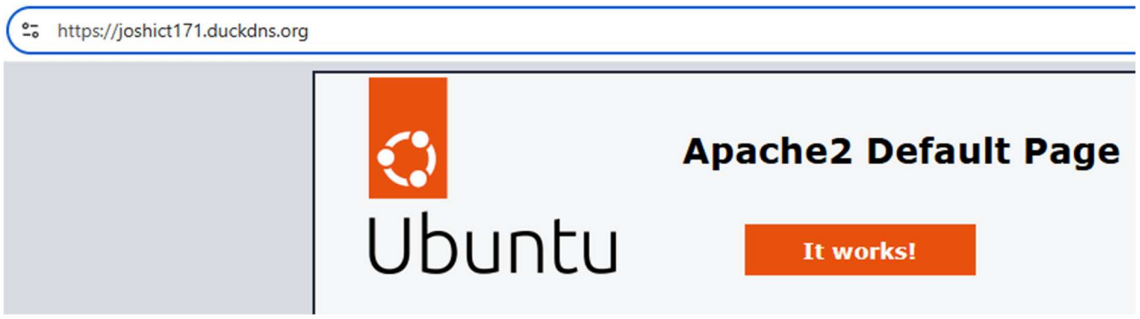
```
<VirtualHost *:443>
        ServerName joshict171.duckdns.org
```

Configuration file located at `/etc/apache2/sites-available/default-ssl.conf`

HTTP/1.1 200 OK = Success

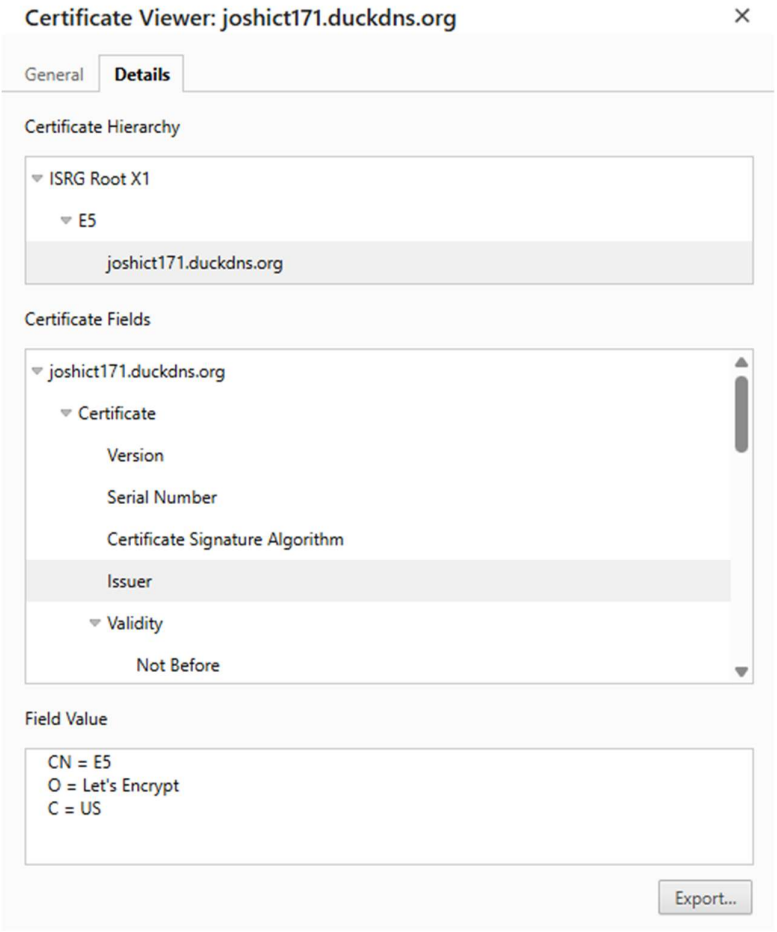
```
ubuntu@ip-172-31-36-187:~$ sudo systemctl restart apache2
ubuntu@ip-172-31-36-187:~$ curl -I https://joshict171.duckdns.org
HTTP/1.1 200 OK
Date: Tue, 01 Jul 2025 15:59:02 GMT
Server: Apache/2.4.58 (Ubuntu)
Last-Modified: Mon, 30 Jun 2025 12:51:30 GMT
ETag: "29af-638c97c96e5e8"
Accept-Ranges: bytes
Content-Length: 10671
Vary: Accept-Encoding
Content-Type: text/html
```

https://joshict171.duckdns.org



Shows that I am now able to connect to the HTTPS service.

Certificate successfully issued. This shows that the website is publicly available and securely encrypted.



Issuer: Let's Encrypt

Going through the steps that I did, I realized redirecting HTTP to HTTPS is a low effort yet highly impactful approach to contribute to a safer and a more trusted web experience for users. This is very relevant in the world of cybersecurity where secure communications should be a fundamental expectation.

Automation and Monitoring

In this final part of the lab, I focused on scripting and automation to reduce manual intervention in routine server tasks. I wrote a Bash script to archive files, made it executable, and scheduled it using `cron`. To track its execution, I logged the output to a file and monitored system activity using `tail` and other commands.

```
 GNU nano 7.2                                    /home/joshpecson/t
#!/bin/bash

echo "Backup script initiated"

### Add timestamped named files
now=$(date+"%d_%m_%y")
echo "Today's date is $now"

### Perform backup of files and compress
cp -R /home/joshpecson/Documents /home/joshpecson/backup/
zip -r /home/joshpecson/$now.zip /home/joshpecson/backup/

### Rename files dynamically
mv /home/joshpecson/$now.zip /home/joshpecson/backup_$now.zip
```

This gave me practical experience in setting up automated tasks and validating that they run as intended. This is an essential skill for managing server environments in a more time-efficient, accurate and consistent manner.

Recalling back to one of my lesson learnt moments from the previous sessions, I now remembered that I need to set the right permissions, in order to execute my script.

```
joshpecson@joshpecson:~$ chmod +x /home/joshpecson/testscript.sh
joshpecson@joshpecson:~$ ./testscript.sh
```

However, upon executing I ran into some syntax errors pointing towards line 6. Troubleshooting the script to ensure that it will run without errors.

```
joshpecson@joshpecson:~$ chmod +x /home/joshpecson/testscript.sh
joshpecson@joshpecson:~$ ./testscript.sh
Backup script initiated
./testscript.sh: line 6: date+%d_%m_%y: command not found
```

`date` is a command used to display the current system date and time. The `+` symbol is used to specify a custom format for how the date should be displayed.

```
 GNU nano 7.2
#!/bin/bash

echo "Backup script initiated"

### Add timestamped named files
now=$(date+"%d_%m_%y")
```

`+` flag is used to specify a custom format

`date +“<string_format>”` OR `date “+<string_format>”`

```
now=$(date +"%d_%m_%y")
```

Bash treats both as a single argument

Upon succesfull recification of the error, I needed to make the script globally executable. All the CLI leads to a directory that contains all the scripts to carry out the relevant tasks each command is suppose to execute. This can be found in the system's `/usr/bin/` directory.

```
joshpecson@joshpecson:/$ chmod +x /usr/bin/testscript
joshpecson@joshpecson:/$ testscript
Backup script initiated
Today's date is 05_07_25
```

Essentially, what I did was add my script into `usr/bin/` so that it becomes availble as a single string command as opposed to having to run the script.

As good practice, I logged a timestamp after each export to confirm the backup completed successfully. This helps with traceability and quick troubleshooting.

```
### Log the outout of the backup script into text file
echo "Backup completed at $(date)" >> /home/joshpecson/task.log
```

Automating with Cron

Scheduling with cron gave me a better understanding of cron formatting. The objective was to ensure that the script ran on an hourly basis. However, for the sake of testing, we will be setting it to a minutely interval instead.

Cron format

```
# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# |  |  |  |  |
# *  *  *  *  * user-name command to be executed
```

Added: * * * * * joshpecson /usr/bin/testscript

```
# *  *  *  *  * user-name command to be exe
17 *    * * *   root    cd / && run-parts -
25 6    * * *   root    test -x /usr/sbin/a
47 6    * * 7   root    test -x /usr/sbin/a
52 6    1 * *   root    test -x /usr/sbin/a
* * * * * joshpecson /usr/bin/testscript
```

This is to satisfy the testing conditions of running the script every minute

Checking on the latest entry to see if cron is executing the newly added job accordingly `tail -f /var/log/syslog`

```
ly)
2025-07-05T16:17:01.107668+00:00 joshpecson CRON[5413]: (joshpecson) CMD (/usr/bin/testscript)
2025-07-05T16:17:01.128311+00:00 joshpecson CRON[5409]: (CRON) info (No MTA installed, discarding output)
2025-07-05T16:17:01.163275+00:00 joshpecson tracker-miner-fs-3[5420]: (tracker-extract-3:5420): GLib-GIO-WARN
ING **: 16:17:01.162: Error creating IO channel for /proc/self/mountinfo: Invalid argument (g-io-error-quark,
 13)
2025-07-05T16:18:01.142944+00:00 joshpecson CRON[5427]: (joshpecson) CMD (/usr/bin/testscript)
2025-07-05T16:18:01.163279+00:00 joshpecson CRON[5426]: (CRON) info (No MTA installed, discarding output)
2025-07-05T16:18:01.202315+00:00 joshpecson tracker-miner-fs-3[5434]: (tracker-extract-3:5434): GLib-GIO-WARN
ING **: 16:18:01.201: Error creating IO channel for /proc/self/mountinfo: Invalid argument (g-io-error-quark,
 13)
2025-07-05T16:18:26.487651+00:00 joshpecson kernel: [UFW BLOCK] IN=enp0s3 OUT= MAC=01:00:5e:00:00:01:f8:79:28
:9d:66:30:08:00 SRC=192.168.1.254 DST=224.0.0.1 LEN=32 TOS=0x00 PREC=0x00 TTL=1 ID=46735 DF PROTO=2
2025-07-05T16:18:26.487716+00:00 joshpecson kernel: [UFW BLOCK] IN=enp0s3 OUT= MAC=01:00:5e:00:00:01:f8:79:28
:9d:66:30:08:00 SRC=192.168.1.254 DST=224.0.0.1 LEN=32 TOS=0x00 PREC=0x00 TTL=1 ID=46735 DF PROTO=2
2025-07-05T16:18:30.156015+00:00 joshpecson kernel: [UFW BLOCK] IN=enp0s3 OUT= MAC=01:00:5e:00:00:fb:70:d8:c2
:14:4c:13:08:00 SRC=192.168.1.20 DST=224.0.0.251 LEN=32 TOS=0x00 PREC=0x00 TTL=1 ID=48619 PROTO=2
2025-07-05T16:19:01.176976+00:00 joshpecson CRON[5441]: (joshpecson) CMD (/usr/bin/testscript)
```

The log confirmed that /usr/bin/testscript was executed every minute by the user `joshpecson`. The minute-based timestamps validate the cron schedule set earlier was working.

Exporting Backup Files to Cloud Server

I tested exporting my backup archive to my cloud server using `scp`. This step shows how transferring backups to external locations provides redundancy. In real cybersecurity scenarios, keeping backups off-site protects against risks like local hardware failure, accidental deletion, natural disasters and many others.

`scp -i ./<private_key.pem> /home/joshpecson/$now.zip ubuntu@<my_cloud_server_ip>:/home/ubuntu/`

```
  adding: home/joshpecson/backup/testfolder/file11 (stored 0%)
  adding: home/joshpecson/backup/file5 (stored 0%)
scp: stat local "/home/joshpecson/05_07_25.zip": No such file or directory
```

Syntax error. `$now.zip` does not exist, but rather it should be `backup_$now.zip`

```
### Upload the backup to cloud server
scp -i /home/joshpecson/webserver-key.pem /home/joshpecson/$now.zip ubuntu@43.204.25.73:/home/ubuntu/
```

```
backup_$now.zip
```

Retried exporting the backup file again

What went wrong here?

```
Permissions 0664 for '/home/joshpecson/webserver-key.pem' are too open.
It is required that your private key files are NOT accessible by others.
This private key will be ignored.
Load key "/home/joshpecson/webserver-key.pem": bad permissions
ubuntu@43.204.25.73: Permission denied (publickey).
scp: Connection closed
```

The private key file was not accessbile, likely a permission issue.

`chmod 600 webserver-key.pem` and tried to run the script once again

```
  adding: home/joshpecson/backup/file5 (stored 0%)
backup_05_07_25.zip                                           100% 5520    76.5KB/s   00:00
```

Back up successful.

Counter checking to see if the backup files were correctly exported to my cloud server.

```
ubuntu@ip-172-31-36-187:~$ whoami
ubuntu
ubuntu@ip-172-31-36-187:~$ ls
LabFiles  backup_05_07_25.zip  duckdns.ini  index.html  linux_serverfunctions  snap  testscript.sh
```

Success! Backup files were found in my cloud server.

Use `chmod 600 <private_key.pem>` right after downloading or creating a .pem file. SSH checks the permissions before using the key. If it is too accessible, it assumes someone else could have copied the key, which defeats the purpose of a private key.

Extra Challenges

`sudo crontab -e`

Added `@reboot /usr/bin/testscript`

```
# For more information see th
#
# m h  dom mon dow   command

@reboot /usr/bin/testscript
```

This is done to the root crontab to ensure that testscript runs automatically every time the system starts.

`echo "figlet Welcome" >> ~/.bashrc`

`echo "neofetch" >> ~/.bashrc`

As a small enhancement, I experimented with `figlet` and `neofetch` to customise the terminal output. `figlet` allowed me to display large stylised text, while `neofetch` showed system information on login.

Appendix

Ubuntu Command Line Cheat Sheet. Ubuntu Tutorials. Retrieved from: https://ubuntu.com/tutorials/command-line-for-beginners#1-overview

Tar vs Zip – What's the Difference? GeeksForGeeks. Retrieved from: https://www.geeksforgeeks.org/difference-between-tar-and-zip/

Understanding Linux File Permissions. Red Hat Enable Sysadmin. Retrieved from: https://www.redhat.com/sysadmin/linux-file-permissions

Chmod Calculator. chmod-calculator.com. Retrieved from: https://chmod-calculator.com/

What is Total Cost of Ownership (TCO)? TechTarget – SearchCIO. Retrieved from: https://www.techtarget.com/searchcio/definition/total-cost-of-ownership

AWS TCO Calculator. Amazon Web Services. Retrieved from: https://calculator.aws.amazon.com/tco/

Let's Encrypt Challenge Types. Let's Encrypt Documentation. Retrieved from: https://letsencrypt.org/docs/challenge-types/

Wildcard Certificates and DNS-01 Challenges. Netlify Blog. Retrieved from: https://www.netlify.com/blog/2018/08/20/enabling-free-wildcard-domain-certificates-with-lets-encrypt/

How HTTPS Works. Cloudflare Learning Center. Retrieved from: https://www.cloudflare.com/learning/ssl/how-https-works/

Certbot Documentation. Electronic Frontier Foundation. Retrieved from: https://eff-certbot.readthedocs.io/en/stable/