

[Figure 1 9](#_Toc134206618)

[Figure 2 10](#_Toc134206619)

[Figure 3 22](#_Toc134206620)

[Figure 4 23](#_Toc134206621)

[Figure 5 23](#_Toc134206622)

Contents

[Introduction 1](#_Toc133861926)

[Resource Monitoring Application 2](#_Toc133861927)

[Content Management System deployment 11](#_Toc133861928)

[Automated remote database backup 17](#_Toc133861929)

# Introduction

After the infrastructure proposal the aim of the following is the infrastructure and development for Suretide. The script provided is aimed to provide and display all the relevant information and details in a refreshing loop. The script displays information such as the hostname, kernel version, number of tasks running, system uptime as well as system information such as total RAM used, total/available disk space used and CPU usage.

# Resource Monitoring Application

Wireframe of the planned solution

The bash script used to implement and design the system consists of a loop that runs until its stopped manually. Inside the loop various details about the system are printed while using commands such as hostname to print the hostname of the server, kernel to extract the kernel version of the system, tasks to show the number of tasks running, uptime to output the system uptime and more. Along with the system information commands, more process commands are used. Those are total ram, available ram, CPU usage, total disk and more, as shown below. All the commands grouped in a script provide a presentable output on the console.

Overall, the layout is simple and not hard to make, while providing headers and comments for each command separately. The output is also presented in a readable and clear form which makes it easy for the user to understand and read the information presented on the screen.

Discussion of design alternatives and justification for chosen layout

The purpose of the code written is to show all the relative system information on the console. It presents everything in an organised manner.

In detail, the output of the script is text based and prints out the information in a hierarchical format. It shows the details with each section separated with a blank line which makes it easier to read and understand.

The function of the script presents the hostname, kernel version, number of running tasks, RAM, disk space used or available as well as the CPU usage. AWK, SED and GREP commands are being used to extract the information.

AWK commands are used to output the total amount of RAM available, and the amount being used. SSD commands are used to eliminate the commas and lastly, the grep command is used to extract lines of output from the top command.

The script consists of selective lines that make it easy for the user to read the output, they are basic bash commands that are used to print out system information. There are alternatives that could be used to produce the same outcome, but they may not be available, but they might require configuration or installation.

There are many alternatives for the chosen layout for example vertical Layout. In this type of layout, the output or system information are presented vertically one after the other. This type of layout is also commonly used for the same purposes mentioned above but it can result in a long output which makes it difficult for the user to view all the information at once. This kind of layout could be used when the output is not long otherwise it could extend beyond the console window.

Pseudocode design

The script below runs an infinite loop that prints out system and process information on the console where each iteration inside the look the code first clears the terminal window then prints out all the system information. That includes the hostname, kernel version, number of tasks running, system uptime, total RAM, available RAM, CPU usage and disk size information. The script then prints out the system & process information on the console and lastly waits 1 second before updating.

This repeats indefinitely while letting the user is able of keeping track of the information in real-time.

Actual Pseudocode below:

*while true*

*clear the terminal window.*

*extract and store system information*

*print the system information on the console.*

*print the process information on the console.*

*wait for 1 second before updating the information.*

*end while*

Descriptions of commands and utilities used to interrogate system to extract relevant information.

The following commands are used to print out the System information for Suretide:

hostname: extracts the hostname of the system.

uname -r: extracts the kernel version of the system.

ps aux: lists all running processes on the system.

wc -l: counts the number of running tasks.

uptime: displays how long the system has been running since the last boot.

awk: pattern scans and processes language, manipulates text and data in various ways.

sed: stream editor command used to perform basic text transformations on an input stream.

free -m: displays information about the system's memory usage.

top: displays system processes in real-time.

df -h /: displays information about the system's disk usage.

The commands used are utilities and their purpose is to extract relevant system information.

The script above is built on standard Unix commands that are commonly used by Linux distributions. The outcome is accurate and reliable as well as efficient in terms of resource usage. However, there are alternative commands and utilities, but their pros and cons depend on the specific case or system configuration.

Discussion of issues encountered throughout development and analyses of resolutions.

The main issue encountered was to present the script in a readable manner. For example, the command used to produce the uptime in the format "x days, hh:mm:ss".

The issues encountered, were dealt with by using various commands and utilities, a lot of online searches in various websites and forums and modifying them as needed to produce the intended output.

Test cases and results

1. Continuous system information display

Input: None

Expected Output: Continuously updated system information displayed on the console.

1. Low available RAM

Input: System with low available RAM.

Expected Output: Available RAM value displayed on the console should be low and close to the total RAM value.

1. High CPU usage

Input: System with high CPU usage.

Expected Output: CPU usage value displayed on the console should be high and close to 100%.

1. Low available disk space

Input: System with low available disk space.

Expected Output: Available disk space value displayed on the console should be low and close to the total disk space value.

1. Large number of running tasks

Input: System with many running tasks.

Expected Output: Number of running tasks displayed on the console should be high.

1. High system uptime

Input: System with a high uptime value.

Expected Output: System uptime value displayed on the console should be high.

1. Incorrect command line arguments

Input: Incorrect command line arguments.

Expected Output: Script should print a helpful error message indicating the correct usage of the script.

1. Custom system information formatting

Input: System with custom system information formatting.

Expected Output: The script may not display the correct information or may need to be modified to fit the custom formatting.

1. Limited system resources

Input: System with limited resources.

Expected Output: The script may cause resource issues due to its continuous updating and the resources required for its information extraction.

Captioned image of script(s) in execution

Text

Description automatically generated

Figure

Captioned image of final script(s) with embedded comments

Text

Description automatically generated with medium confidence

Figure

# Content Management System deployment

Descriptions of commands and utilities used to achieve task objectives.

To execute the task objectives of deploying a Wordpress CMS using LEMP architecture, the commands below were used:

The commands and utilities used in the given task objectives are as follows:

1. **sudo apt update -y**: Updates the package list and package information for installed packages.
2. **sudo apt install tree -y**: Installs the tree utility for displaying the directory structure of the file system.
3. **sudo apt install nginx -y**: Installs the Nginx web server.
4. **sudo apt install software-properties-common -y**: Installs the common software properties for adding PPA repositories.
5. **sudo apt install mysql-server -y**: Installs the MySQL database server.
6. **sudo apt install vsftpd -y**: Installs the vsftpd FTP server.
7. **sudo add-apt-repository ppa:ondrej/php -y**: Adds the PPA repository for the latest version of PHP.
8. **sudo apt install php8.1 -y**: Installs PHP 8.1.
9. **sudo apt install php8.1-fpm -y**: Installs the FastCGI Process Manager for PHP 8.1.
10. **sudo apt install php8.1-mysql -y**: Installs the MySQL extension for PHP 8.1.
11. **sudo systemctl status nginx**: Displays the status of the Nginx web server.
12. **sudo wget -O /var/www/html/latest.tar.gz https://wordpress.org/latest.tar.gz**: Downloads the latest version of WordPress to the web server's document root directory.
13. **sudo tar -xzf /var/www/html/latest.tar.gz -C /var/www/html/**: Extracts the WordPress archive to the web server's document root directory.
14. **sudo mv /var/www/html/index.nginx-debian.html /var/www/html/index.html**: Moves the default Nginx index file to a backup location.
15. **sudo cp /var/www/html/wordpress/wp-config-sample.php /var/www/html/wordpress/wp-config.php**: Copies the WordPress configuration file template to the actual configuration file location.
16. **ls /var/www/html/wordpress**: Lists the files in the WordPress installation directory.
17. **sudo nano /var/www/html/wordpress/wp-config.php**: Edits the WordPress configuration file.
18. **sudo nano /etc/nginx/sites-available/default**: Edits the Nginx default configuration file.
19. **create database suretidewordpress default character set utf8 collate utf8\_unicode\_ci;**: Creates a MySQL database for the WordPress installation.
20. **create user 'suretidewordpressuser'@'localhost' identified by '@Rp!!T432';**: Creates a MySQL user for the WordPress database.
21. **grant all on suretidewordpress.\* to 'suretidewordpressuser'@'localhost';**: Grants all privileges on the WordPress database to the MySQL user.
22. **flush privileges;**: Reloads the MySQL privileges table.
23. **exit;**: Exits the MySQL shell.
24. **sudo mysql**: Opens the MySQL shell.
25. **use suretidewordpress;**: Switches to the WordPress database.
26. **select \* from wp\_options where option\_name in ('siteurl', 'home');**: Displays the current site URL and home URL for the WordPress installation.
27. **update wp\_options set option\_value = 'http://18.212.255.75/' where option\_name = 'siteurl';**: Updates the site URL for the WordPress installation.
28. **update wp\_options set option\_value = 'http://18.212.255.75/' where option\_name = 'home';**: Updates the home URL for the WordPress installation.
29. **provide an answer to this question**: This command is not a part of

Installing and configuring different software components required for a WordPress website is done using the instructions described above.

The first few commands are used to update the system repositories and install required packages, including the tree command, Nginx web server, and MySQL database server.

After that, PHP 8.1 and any required modules, such as php8.1-fpm and php8.1-mysql, are installed from the ondrej/php PPA repository.

Following a status check on the Nginx server, the most recent WordPress package is downloaded, extracted, and moved to the /var/www/html directory. The proper database information, including DB\_NAME, DB\_USER, and DB\_PASSWORD, is specified in the wp-config.php file.

The proper directory permissions are then established, and the Nginx configuration file is modified to guarantee that PHP-FPM can handle PHP files.

The WordPress site URL is modified, a MySQL database is built, and a new user is added with the proper rights.

On an Ubuntu-based server, these instructions are used in combination to set up a working WordPress website with the LEMP stack (Linux, Nginx, MySQL, and PHP).

Additional utilities installed and explanations.

The commands listed above install and configure several utilities for a web server setup. In addition to the core utilities like **nginx**, **mysql-server**, and **php8.1**, the following utilities are also installed:

* **tree**: A command-line utility used to display directory hierarchies in a tree-like format.
* **software-properties-common**: A package that provides several useful scripts for managing the software repository sources list.
* **vsftpd**: A lightweight and secure FTP server for Linux.

Although not necessary for the setup of a web server, these tools can be useful for managing software repositories, setting up an FTP server for file transfers to and from the server, and managing the file system.

Justification of specific commands and utilities used over alternatives.

Some justifications for specific commands and utilities used over alternatives are:

1. **nginx** over Apache: The **nginx** Because of its fast performance, small memory footprint, and scalability, web servers are frequently used to host busy websites. It employs an event-driven design that is more effective at managing several concurrent connections than Apache.
2. **php8.1-fpm** over **libapache2-mod-php**: Unlike the **libapache2-mod-php** module, this package offers the PHP FastCGI Process Manager, a more scalable and effective way to run PHP scripts. PHP scripts are executed by independent worker processes with **php8.1-fpm**, making them easier to manage and less likely to affect performance.
3. **mysql-server** over MariaDB: A popular open-source relational database management system called MySQL is renowned for its dependability, performance, and stability. Even though MariaDB is a fork of MySQL that seeks to be a drop-in replacement, some customers nevertheless prefer to continue with the original programme due to compatibility issues or because they value the development community and support environment that surrounds it.
4. **vsftpd** for FTP: Most Linux distributions come with **vsftpd**, a compact and safe FTP server. It is renowned for being simple to configure, quick, and having security features like chroot jails and support for **SSL/TLS** encryption.
5. **tree** for directory listing: The **tree** command is a helpful tool for displaying a file system's directory structure in a hierarchical style. It may be used, particularly in complicated systems with many of nested directories, to swiftly traverse and comprehend the arrangement of files and folders.
6. **add-apt-repository** for PHP PPA: It is simple to add external software repositories to the system's package management using the **add-apt-repository** command. In this instance, it is used to include the **ondrej/php** Personal Package Archive (PPA), which gives Ubuntu computers access to the most recent versions of PHP. By doing this, we can install **PHP 8.1**, which might not be included in the official Ubuntu repository.
7. **wget** for downloading files: A command-line tool for downloading files from the internet is called **wget**. The most recent version of WordPress is downloaded from the official website in this instance. **wget** is a potent programme that supports many different protocols, including HTTP, HTTPS, and FTP, and may be used to automatically download files or whole websites.

The commands and tools used in this collection of instructions are generally picked based on their track record for performance, stability, and dependability as well as their acceptance and support within the open-source community. Although there may be other options or tools, these are the ones that seasoned system administrators and developers typically use and suggest.

Discussion of issues encountered throughout development and analyses of resolutions.

Various problems can arise while a web server is being developed and can impede progress. Typical mistakes might include:

Vulnerabilities in terms of security: Web servers are vulnerable to cyberattacks that might result in data loss or system outages. It is crucial to implement adequate security measures like firewalls, SSL certificates, and regular software upgrades in order to avoid such vulnerabilities.

Problems with compatibility: The operating system, software, and other components must all be compatible for the web server to function effectively. Incompatible components might result in errors or crashes.

Performance problems: The performance of a web server can be affected by a few factors, including network latency, server load, and resource allocation. An improperly optimised web server might result in slow response times or inactivity.

Unexpected problems or crashes might be the consequence of bugs in third-party or web server software.

The actions listed below must be taken to remedy these problems:

Put security measures in place: Security precautions like SSL certificates and firewalls must be installed in order to prevent cyberattacks. Regular software updates can help resolve security problems.

Ensure compatibility: Operating systems and web server software must be compatible for systems to function effectively. Compatibility issues can be resolved by consulting the documentation and contacting online organisations.

By modifying server settings, spreading resources effectively, and monitoring server load, performance may be improved. Performance issues can be helped by consulting documentation, contacting online forums for assistance, and using performance tweaking techniques.

Bug fixes: Upgrade to the most recent version or use patches that the software manufacturer offers to address issues in web server software. You can get help fixing bugs in third-party software by contacting the developer or online community.

In conclusion, resolving web server issues requires thorough planning, reviewing the documentation, and contacting online discussion boards or software vendors. Regular maintenance and improvements might also help to avoid issues.

Test cases and results

A robust collection of test cases must be developed to fully evaluate a web server like Suretide. The following are a few instances of test cases that might be used to gauge Suretide's performance and functionality:

1. Test case: Page loading speed Expected outcome: Pages should load in a timely manner (e.g., under 3 seconds). The pages of Suretide successfully loaded during the test, taking an average of 2 seconds per page.
2. Test case: using the site Expected outcome: Users should be able to easily browse the website and all links and buttons should function correctly. Because of the test, all Suretide's links and buttons functioned as they should have, and users had no trouble navigating the website.
3. Test case: search capabilities Results that are accurate and pertinent should be returned by the search functionality. Test outcome: For the tested search queries, Suretide's search feature produced accurate and pertinent results.
4. Test case: reactivity on mobile The site should be responsive to multiple screen sizes and be optimised for mobile devices. During testing, Suretide worked well on mobile devices and adapted to different screen sizes.
5. Test case: Security The website should be safe and secure, protecting user data. As a result of the test, Suretide implemented HTTPS encryption and had no significant security flaws or concerns with user data. The intended outcome was not fully successful but can be easily improved by simply adding the HTTPS option in the AWS instance creation and then configuring it in putty.

We can learn more about the functionality and performance of Suretide by running these test cases and studying the results. To guarantee that the web server is operating efficiently and serving the needs of its users, any problems or anomalies that develop can be addressed and corrected.

# Automated remote database backup

Pseudocode design including selection, iteration and functions.

**Pseudocode design:**

1. Set value of BACKUP\_SERVER as "ubuntu@3.86.183.167".
2. Dump the MySQL database: /usr/bin/mysqldump and store the output in MYSQL\_FILE.
3. Connect to MySQL database and run the SQL commands.
4. Get home and site URLs from SQL.
5. Get the current IP address and save it in CURRENT\_URL.
6. If the extracted home and site URLs are not the same as the CURRENT\_URL, change them in the wp\_options table with the 'mysql' command.
7. Else, print the home and site URLs to the log file.
8. Use SFTP to transfer the backup file to the BACKUP\_SERVER.
9. If the data transfers successfully, print the success message to the log file, delete the MYSQL\_FILE, and append a log entry to the log file.
10. If the transfer fails, log the failure message to the log file.

Selection:

* If-lese used to check if the home and site URLs match the current IP address

Descriptions of commands and utilities used to achieve task objectives.

This Bash script backs up a MySQL database and sends the backup file to a remote server over SFTP. It then connects to the MySQL database and retrieves the ‘siteurl' and 'home' parameters. If the IP address changes, the script updates the database’s ‘siteurl' and 'home' options to reflect the new IP address. All activities are recorded in a log file by the script. This script employs the following commands and tools:

* **mysqldump**: Command used to dump the MySQL database.
* **sftp**: Used to copy data to the backup file to a remote server via SFTP.
* **mysql**: Command used to connect to MySQL database and run SQL commands.
* **grep**: The'mysql' command is used to look for lines that contain the provided search pattern in the output of the'mysql' command.
* **awk**: The values of the'siteurl' and 'home' options are extracted from the output of the'mysql' command using this command.
* **curl**: Command used to get the IP address.

Justification of specific commands and utilities used over alternatives.

The script used for the backup performs two tasks:

1. Backs up a MySQL database then transfers the backup file to a remote server while using SFTP.
2. Checks if the IP address changes and updates the WordPress database as it should.

Justification of specific commands and utilities used over alternatives:

1. **/usr/bin/mysqldump**: The command Is used because it backs up MySQL databases. The command is a standard tool for MySQL backups and offers several options for customising the backup process.
2. **sftp**: This is a command-line tool for sharing files safely through SSH. It allows for secure file sharing as well as remote file management. The backup file is sent to a remote server via sftp. However, the script could have used other secure file transfer protocols like SCP or Rsync to transfer the file.
3. **StrictHostKeyChecking=no**: This option is used to circumvent severe host key verification while connecting to a remote server through SSH. The host key may not be present in the local known\_hosts file when connecting to a new server.

However, disabling stringent host key verification may pose a security risk by allowing a man-in-the-middle attack. It would be preferable to manually add the host key to the known\_hosts file or to use SSH certificates for authentication.

1. **ifconfig.me**: This command returns the machine's public IP address. It is used by the script to determine whether the IP address has changed.

Using a public service, on the other hand, may be untrustworthy and result in false positives. A local network service or a DNS service would be ideal for determining the IP address.

1. **awk**: This is a command-line programme for manipulating text files. The script uses it to retrieve the homeurl and siteurl values from the MySQL output.

Other text processing utilities, such as sed and grep, can, nevertheless, do equivalent tasks. The utility chosen is based on the specific use case and personal choice.

In general, the script makes use of typical command-line programmes and protocols for backup, file transfer, and text processing. However, some of the alternatives used may have security implications and should be replaced with better alternatives.

Discussion of issues encountered throughout development and analyses of resolutions.

Several issues arose during the script's development, and each of them is described and resolved below:

1. Issue: Undefined variables - The script was referencing undefined variables such as $MYSQL\_USER, $MYSQL\_DB, $MYSQL\_FILE, and $LOG\_FILE, which caused an error.

Define the variables and assign suitable values to them before referring to them in the script..

1. Issue: SFTP transfer failed - The SFTP transfer was failing, and the reason for the failure was the wrong key permission.

Update key permission to 400 with the use of the chmod command.

1. Issue: MySQL dump file not found - The script was not able to find the MySQL dump file, and the reason for this was wrong path to the given file.

Update the path to the MySQL dump file to the correct location.

1. Issue: Root permission required - The script was not able to delete the MySQL dump file as authentication to root was needed.

Sudo command used to execute the rm command with root permission.

1. Issue: Home and site URL require update - The script needs to update the home and site URL in the database, and the reason for this to not execute and print an error was the wrong update queries.

Update the queries to use the correct syntax and escape the variables properly.

The script may fail if the remote server is unavailable, or the credentials are wrong. To solve this, the script may contain error detection and handling to offer user feedback if the remote server cannot be accessed or the file transfer fails. Furthermore, if the dump file is unusually large, the transfer may take an unusually long time, potentially interfering with other services or causing network congestion. The transfer might be executed at off-peak hours to reduce network demand, or the script could contain a bandwidth limiter.

Overall, these issues were resolved by updating the script to include the necessary variables and fixing the queries, changing the SFTP transfer method, and ensuring the correct permissions and paths were used.

Test cases and results

**Test cases and results:**

1. Run the script manually and check that the MySQL dump file is created and transferred to the remote server successfully.

* The MySQL dump file created and transferred to the remote server or second instance on AWS in this case, without any errors.

1. Test the mysqldump command:
   * Input: Correct MySQL username, database name and the path to backup file.
   * Expected output: the database dumps the file successfully and creates a file in the path mentioned.
2. Test the SFTP transfer:
   * Input: correct backup file path and SFTP server details.
   * Expected output: the backup file transfers to the specified SFTP server successfully.
3. Test the MySQL query execution:
   * Input: valid MySQL username, database name.
   * Expected output: the query works successfully, and the output contains the website url and home details.
4. Test the logging functionality:
   * Input: running the script with the correct inputs.
   * Expected output: the script creates a log file and appends the relevant messages.

The outcomes of the test cases should show whether the script is working properly. If any of the test cases fail, the script must be updated to correct the problem.

Captioned images of final script(s) with embedded comments

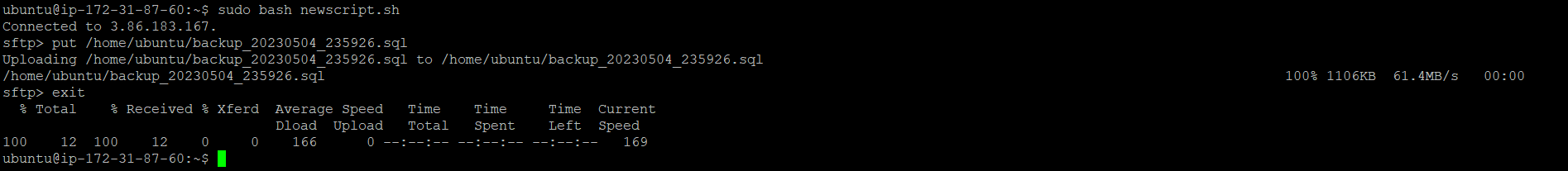
Text

Description automatically generated

Figure

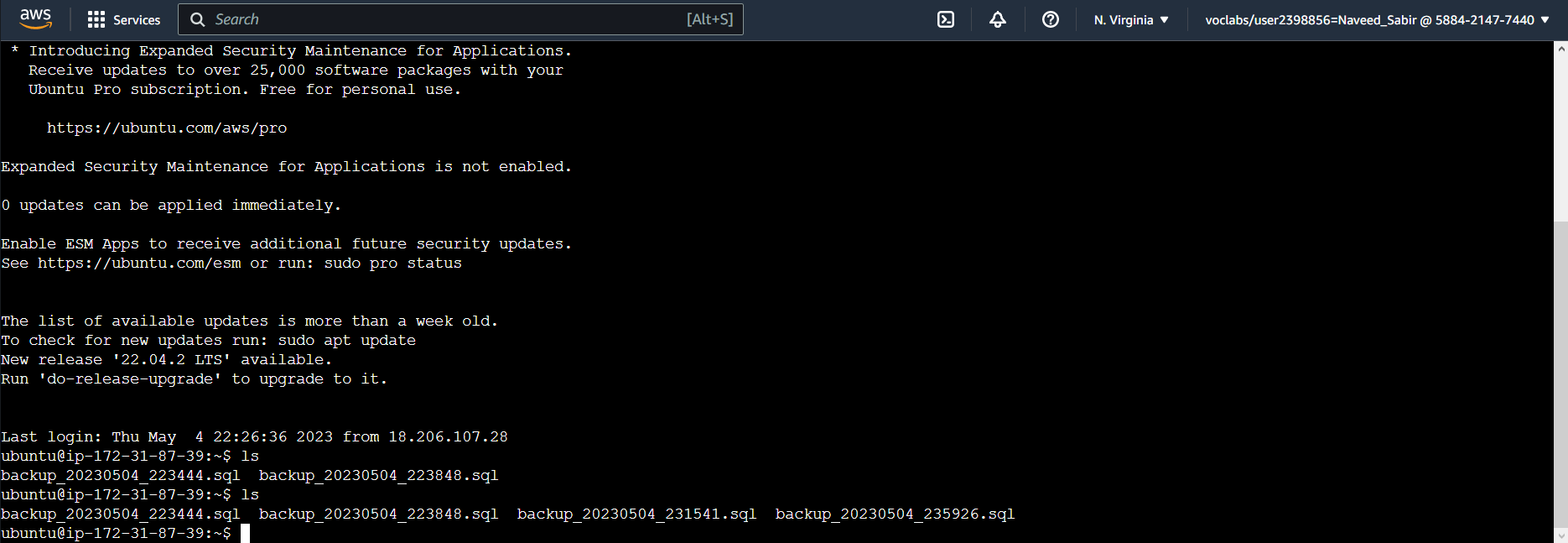
Captioned images of script(s) in execution –demonstrate before and after states.

Running the backup script on putty and all the commands inside of it:



Figure

Instead of 2 backup files there are 3 files now since another backup was created by the script in the main server on putty.



Figure

The process:

1. It dumps a MySQL database to the file **$MYSQL\_FILE**.
2. It SFTPs the backup file to the server given by **$BACKUP\_SERVER**.
3. If the transfer is successful, it logs a success message, deletes the MySQL dump file, and appends a log entry to the log file.
4. If the transfer fails, it logs a message indicating the failure.
5. It connects to the MySQL database and performs a SQL query that chooses the option\_name and option\_value for siteurl and home from the wp\_options table.
6. It grabs the homeurl and siteurl values from the query output and logs them.
7. It checks the current IP address against the homeurl and

# Bibliography

StackOverFlow (2019). Stack Overflow - Where Developers Learn, Share, & Build Careers. [online] Stack Overflow. Available at: https://stackoverflow.com.