

|  |  |
| --- | --- |
| Course / Programme: | BEng Software Engineering | BSc Computing |
| Module name and code: | Computing Infrastructure (SWE4202) |
| Student ID | 2116149 |
| Tutor: | Francis Morrissey | Dr Mohammed Benmubarak |
| Assignment Number: | 2 |
| Assignment Title: | Infrastructure development for Suretide |
| Weighting | 40% of overall module grade |
| Issue Date: | 31/03/2023 |
| Submission Deadline: | 05/05/2023@ 2355 |
| Learning Outcomes assessed:   1. Formulate Scripted Solutions to automate operating system services. 2. Implement practical solutions based on the client/server model to solve a given problem. | |

Contents

[Introduction 1](#_Toc134210016)

[Task 1 - Resource Monitoring Application 2](#_Toc134210017)

[I. Introduction 2](#_Toc134210018)

[II. Design 4](#_Toc134210019)

[III. Implementation 7](#_Toc134210020)

[a. Code Explanation and Justification 7](#_Toc134210021)

[b. Test Cases and Issues Faced 14](#_Toc134210022)

[c. Final script 15](#_Toc134210023)

[d. Running Script 16](#_Toc134210024)

[Task 2 – Content Management System Deploment 17](#_Toc134210025)

[I. Connecting Putty and Setting AWS Instances. 17](#_Toc134210026)

[a. AWS Configuration 17](#_Toc134210027)

[b. Putty Configuration 19](#_Toc134210028)

[II. Installing Updates and Upgrade Available Packages. 21](#_Toc134210029)

[III. Installing Nginx 21](#_Toc134210030)

[IV. Installing MySQL and Addons 22](#_Toc134210031)

[V. Installing and Configuring PHP 23](#_Toc134210032)

[VI. Installing WordPress 24](#_Toc134210033)

[VII. Configuration of Firewall 26](#_Toc134210034)

[VIII. Creating Database and User for WordPress 27](#_Toc134210035)

[IX. Reconnection with WordPress 33](#_Toc134210036)

[a. Reconnection 33](#_Toc134210037)

[b. Nginx Restart Failure 34](#_Toc134210038)

[X. Connection of MySQL Workbench 36](#_Toc134210039)

[XI. Test Results, Issues…. 39](#_Toc134210040)

[Test 1 – Testing the Complete WordPress Installation. 39](#_Toc134210041)

[Test 2 – Reconnection WordPress and Website. 39](#_Toc134210042)

[Issues Found 40](#_Toc134210043)

[XII. Landing Page 40](#_Toc134210044)

[Task 3 – Automated remote database backup 42](#_Toc134210045)

[I. Introduction 42](#_Toc134210046)

[II. Design 42](#_Toc134210047)

[a. Pseudocode 42](#_Toc134210048)

[b. Script Explanation 43](#_Toc134210049)

[III. Implementation 45](#_Toc134210050)

[Creating Backup Server and Dumping Key 45](#_Toc134210051)

[Bibliography 52](#_Toc134210052)

# Introduction

Suretide is an up-and-coming energy company in the UK with ambitions to dominate the tidal energy sector. The COVID-19 crisis and Russia's invasion of Ukraine have caused a surge in demand, which has led to skyrocketing petrol prices. As a result, the UK has been badly affected by previously unheard-of increases in energy costs at a crucial moment. The UK also relies excessively on fossil fuels, with 78% of houses using gas for central heating and gas accounting for 41.9% of all energy generated over the past year. These problems have been made worse by excessive heat loss, which has boosted energy usage. Homes in the UK lose heat three times more quickly than homes in Norway and Germany.

The UK government has promoted the usage of renewable energy in response to this scenario and has provided major subsidies to businesses in this industry. By establishing itself as the industry pioneer in tidal energy, Suretide hopes to take advantage of this potential. You are required to finish a number of tasks as a graduate intern in SysOps for Suretide in order to demonstrate your proficiency with shell scripting.

The selected applicant for the permanent post at Suretide should be competent in scripting, capable of setting and developing client/server applications, and knowledgeable of the operating system utilities. You will need to use built-in utilities like pipe, tail, grep, sort, awk, echo, and sed as well as the correct programming constructs, such as sequence, selection, and iteration, to complete the tasks assigned to assess your proficiency with shell scripting.

Your responsibility as a SysOps intern at Suretide is to assist the business in achieving its goal of dominating the tidal energy market. You will be in charge of making sure that the infrastructure and systems used by the business are secure, scalable, and reliable. You must exhibit a high level of technical proficiency and a comprehensive knowledge of the SysOps-related technologies and tools in order to succeed in this position.

This report will go into depth about how you handled the shell scripting tasks that Suretide gave you. You will offer evidence of your comprehension of basic programming concepts and built-in tools, as well as an explanation of how you used this understanding to address the issues given. You will also give a thorough analysis of the difficulties you had while completing the work and how you overcame them.

Overall, this report will act as a showcase for your technical proficiency and SysOps and shell scripting experience. Additionally, it will shed light on your capacity for critical thought, problem-solving, and technical challenge handling strategies.

# Task 1 - Resource Monitoring Application

## I. Introduction

The first objective given by Suretide is to create a custom application that can show real-time data pertaining to various operating system components with a 1-second refresh rate. The details include host name, kernel version, number of jobs executing, system uptime, total and available RAM, total and CPU use, total and available disc size, and process details including PID, user,%CPU,%RAM, and process command. The goal of this assignment is to deliver a better resource monitoring tool that is more user-friendly and targeted at Suretide's requirements. A BASH script that can retrieve important data from the system is anticipated to be the answer.

Suretide needed a custom application authored in BASH that would show current system information. In order to meet this requirement, a BASH script was made that extracts and shows the following information in real-time with a 1 second refresh rate:

1. Host name
2. Kernel version
3. Number of tasks running
4. System up-time
5. Total and available RAM
6. Total CPU usage
7. Total and available disk size
8. Process details including PID, user, %CPU, %RAM, and process command.

The script was created to be extremely practical yet simple to use and comprehend. To gather the necessary system data, it made use of several built-in OS functions, including hostname, uname, ps, uptime, free, mpstat, df, and ps. A number of programming constructs, including sequence, selection, iteration, and functions, were also used in the script.

Overall, Suretide now has a highly effective tool for tracking the system's performance and resource usage in real-time thanks to the custom application.

## II. Design

The wireframe is broken up into two sections: the header, which shows system data like hostname, kernel version, number of tasks running, system uptime, total and available RAM, total and CPU usage, total and available disc size, and the process details section, which shows PID, user, and%CPU usage for the top three processes by CPU usage. A table for showing process data, with columns for PID, USER, and%CPU utilisation, is included in the wireframe. The layout and structure of the script can be designed using this wireframe as a guide.

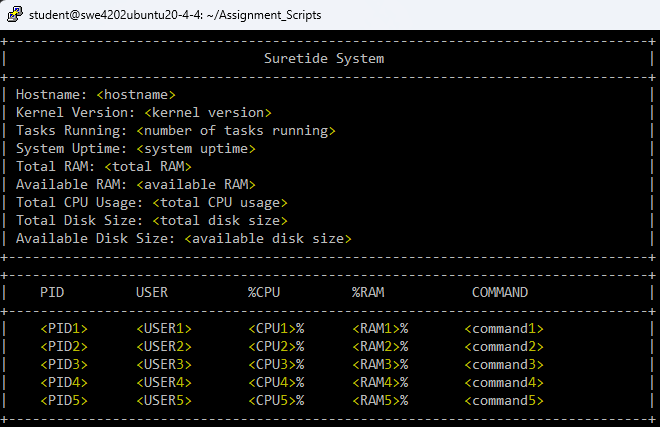


Figure . Wireframe of the planned solution

There are various design options to take into account while creating an application for real-time system monitoring. The data could also be shown in a tabular fashion, where each row corresponds to a distinct system metric and each column to a different time period. This would make it possible to quickly and simply compare the numbers over time.

However, for the purposes of this task, a vertical list format is the preferred layout option. This structure offers the data in a straightforward, readable way that is especially well-suited for the particular metrics being tracked. When monitoring multiple metrics in real-time, the information can be easily scanned thanks to the vertical list format.

A clear header identifying the system being monitored and each metric being shown is part of the layout that was chosen. The measures are simpler to differentiate from one another and are easier to read because to the usage of whitespace and vertical spacing. Furthermore, the use of colour coding assists in emphasising important metrics like total CPU usage and available disc size.

Overall, the selected layout is easy to use, intuitive, and effective for real-time monitoring of the specified system parameters.

Next, we can see the Pseudocode of the bash script:

1. Define a function called display\_system\_info() that does the following:

a. Print the system information header.

b. Print the hostname using the hostname command.

c. Print the kernel version using the uname -r command.

d. Print the number of running tasks using the ps -e | wc -l command.

e. Print the system uptime using the uptime command, formatting it with awk and sed commands.

f. Print the total RAM using the free -h command and awk command to extract the appropriate value.

g. Print the available RAM using the free -h command and awk command to extract the appropriate value.

h. Print the total CPU usage using the mpstat command and awk command to calculate the appropriate value.

i. Print the total disk size using the df -h / command and awk command to extract the appropriate value.

j. Print the available disk size using the df -h / command and awk command to extract the appropriate value.

2. Define a function called display\_process\_info() that does the following:

a. Print the process details header.

b. Print the process ID, user, CPU and RAM usage, and command name for the top 5 CPU-intensive processes using the ps command and awk command to extract and format the appropriate values.

3. Enter an infinite loop using the while statement that does the following:

a. Clear the screen using the clear command.

b. Call the display\_system\_info() function to display system information.

c. Print the process details header.

d. Call the display\_process\_info() function to display process information.

e. Wait for 1 second using the sleep command.

4. End the script.

## III. Implementation

### a. Code Explanation and Justification

The bash script I created makes use of a unique set of commands and tools to deliver the right output for the real-time system monitoring application. Here's a breakdown of the commands and utilities used:

1. “**display\_system\_info()**” First we define a function to display the system information. The display\_system\_info() keyword, which defines the function's name, comes first in the function. Any arguments passed to the function are enclosed in brackets following the name. The brackets are empty since the function in this instance doesn't require any arguments.

2. The “**echo**” command is used to print a message to the terminal. The message in this instance is a string of characters made up of 80 dashes enclosed in plus signs.

**echo "**+--------------------------------------------------------------------------------+**"**

This line of code has been used numerous times, resulting in a better table shape for the output.

This line's function is to print a horizontal line dividing the output into several pieces. The line is made up of a string of dashes, which are frequently used in command-line output as line separators. The plus signs frame the line and help it stand out more aesthetically.

The length of the line is determined by the number of dashes in the string, which in this case is 80. This length can be changed to suit the needs of the script or the user.

3. The next line of the code uses the same echo command to print the Title of the header of the system monitoring application.

**echo "| Suretide System |"**

4. The “**printf**” command is used to format and print a string. In this case, the string being printed is **"|%-80s|\n"**. The **%s** is a placeholder that will be replaced with the value passed as an argument after the string. The **-80** specifies that the value should be left-aligned within a field that is **80** characters wide.

The value being passed as an argument after the string is **"Hostname: $(hostname)"**. The **$()** syntax is used to execute a command and substitute its output into the string. The **hostname** command returns the hostname of the system, and that value is inserted into the string after the word **"Hostname:"**.

The result of this code will be to print a line that resembles a table row with a left-aligned label ("Hostname:") and a value (the result of the **hostname** command) within an 80-character wide field.

**“ printf "|%-80s|\n" "Hostname: $(hostname)" ”**

The same starting commands are used in the code's next 8 lines.

5. “ **printf "|%-80s|\n" "Kernel Version: $(uname -r)"** “. Prints the kernel version using **uname -r.**

To obtain the same information, we can use alternate instructions:

* uname -v
* hostnamectl | grep "Kernel"

6. “ **printf "|%-80s|\n" "Tasks Running: $(ps -e | wc -l)"** “. Prints the number of running processes using “ **ps -e | wc -l** “. In operating systems that resemble Unix, the **ps** command shows details about active processes. The **ps** **-e** command displays details about all the processes that are currently active on the machine.

The output of one command is passed as the input to another programme using the **|** symbol (pipe). In this instance, the **wc** command receives the output of the **ps -e** command.

A file or text's word, line, and character counts can be done with the **wc** command. The number of lines in the output of **ps -e**, which represents the number of active processes on the computer, is counted by the command **wc -l**.

Alternatives: “**systemctl status | grep "running" | wc -l**” , “**ps aux | wc -l**”.

7. “ **printf "|%-80s|\n" "System Uptime: $(uptime | awk '{print $3,$4}' | sed 's/,//')"** “. Prints the system uptime.

The **“uptime”** command is used to display the system uptime, along with the current time, number of logged-in users, and load average information.

“**awk '{print $3,$4}'** ” is used to extract the third and fourth fields from the output of the **uptime** command. These fields represent the system uptime in days and time since the last reboot, respectively.

“ **sed 's/,//'** ” is used to remove any commas from the output. Some versions of the **uptime** command include commas to separate large numbers, but this command removes them for readability.

Alternatives: “ **uptime | cut -d' ' -f3,4 | tr -d ','** ” , “**uptime | sed 's/^.\*load average: //' | sed 's/,//g'**”.

8. “**printf "|%-80s|\n" "Total RAM: $(free -h | awk '/^Mem:/{print $2}')"** ”. Prints the total RAM.

The command **free -h | awk '/^Mem:/{print $2}'** is a way to find out how much memory your computer has in a human-readable format.

The first part of the command, **free -h**, shows the amount of free and used memory in the system. The **-h** option just makes the output easier to read by showing the sizes in gigabytes, megabytes, and kilobytes instead of bytes.

The second part of the command, **awk '/^Mem:/{print $2}'**, searches for lines that start with the word "Mem:" and then prints out the second field on that line, which is the total amount of memory available in your system.

In order to provide you the total amount of memory in your system in a human-readable manner, such as "8 GB" or "16 GB," you must perform the whole command **free -h | awk '/Mem:/print $2'**.

Alternatives: “**grep 'MemTotal' /proc/meminfo | awk '{print $2}'**”

9. “ **printf "|%-80s|\n" "Available RAM: $(free -h | awk '/^Mem:/{print $NF}')"** ”. Prints the available RAM.

The command **free -h** displays information about the system's memory usage. The **awk** command is then used to extract a specific piece of information from this output. The expression **'/^Mem:/{print $NF}'** is a regular expression that matches lines that start with the string "**Mem**:". The **print $NF** command then prints the last field on that line, which corresponds to the amount of free memory on the system in this case.

Alternatives: “**vmstat -s | awk '/free memory/{print $1}'**”, “**top -bn1 | grep 'Mem:' | awk '{print $4}'**”.

10. “ **printf "|%-80s|\n" "Total CPU Usage: $(mpstat | awk '$12 ~ /[0-9.]+/ { print 100 - $12 }')"** ”.

Prints the total use of the CPU.

The system's CPU use is detailed by this command “**mpstat**”. It is used to compile and present information on CPU usage.

This symbol “**|**”, sometimes known as a pipe, is used to pass the output of one command into the input of another.

**Awk** is a text-processing utility tool that is frequently used for processing and scanning patterns.

**$12,** this stands for the input's 12th field, which in this case represents the CPU idle time.

This symbol “**~**”, is used in the context of a regular expression match in **awk**.

The regular expression pattern “**/[0-9.]+/**” can be used to match any run of one or more digits or decimal points.

The awk statement "**{print 100 - $12}**" outputs the percentage of CPU utilisation after subtracting the idle time from 100.

Alternative: “**top -bn1 | awk 'NR>7{s+=$9} END {print 100 - s/4}'**”.

11. **printf "|%-80s|\n" "Total Disk Size: $(df -h / | awk '/^\/dev/{print $2}')".** Prints the total disk size.

The command “**df -h /**”, is used to get details about the root file system's disc consumption. The **/** parameter indicates that we are interested in the root file system, while the **-h** option displays the sizes in a human-readable format.

This symbol “**|**”, sometimes known as a pipe, is used to pass the output of one command into the input of another.

**Awk** is used in this command to look for lines that start with **"/dev/"** and display the second column of those lines when it finds them (**{print $2}**). The overall size of the file system is what we are interested in, and it is displayed in the second column of the output of **df -h /**.

Alternative: “**du -sh /**”.

12. “**printf "|%-80s|\n" "Available Disk Size: $(df -h / | awk '/^\/dev/{print $4}')"**”. It prints the available size of the disk.

The exact same command that was used to determine the entire size of the disk has been used here. But in this instance, we'll print the fourth column (**{print$4}**), which corresponds to the size of the available disk space, rather than the second.

The next block of the script defines the function, **display\_process\_info() {}**, which is responsible for displaying process-related system information.

The first command in this function is: “**echo "+--------------------------------------------------------------------------------+"** ”: This command outputs a horizontal line of dashes and plus signs, which is used to separate the process information from the rest of the output. In the first block of the code, it is already explained with all the details this line of the code which has been used several times.

The second line of the function, “ **printf "|%-11s %-11s %-11s %-11s %-32s|\n" "PID" "USER" "%CPU" "%RAM" "COMMAND":** ”, The process ID, user, CPU use %, memory usage percentage, and command are all placeholders in a string that is formatted and printed by this command using **printf**. Formatting choices are shown by **%** marks after letters. For instance, **%s** stands for a text, **%d** for an integer, and **%f** for a floating-point value. The **-11s** option instructs the system to left-justify the appropriate value and add spaces to make it 11 characters wide. A newline is represented by the letter **\n**.

The third line is the same echo command which outputs a horizontal line of dashes. “ **echo "+**--------------------------------------------------------------------------------**+"** “.

“ **ps -eo pid,user,%cpu,%mem,cmd --sort=-%cpu** “. The **ps** tool is used by this command to show process information in a specific manner. The process ID (**pid**), user (**user**), CPU utilisation percentage (**%cpu**), memory usage percentage (**%**mem), and command (**cmd**) are all included in the format, which is specified by the **-eo** option. The result is sorted by CPU utilisation percentage in descending order using **the --sort=-%cpu** option.

“ **head -n 6** ”. The first six lines of the **ps** output are extracted by this command using the **head** tool. Six processes are the most that may be presented as a result.

The final line of the function is: “ **awk '{printf "|%-11s %-11s %-11s %-11s %-32s|\n", $1, $2, $3, $4, substr($0, index($0,$5))}'** ”. In order to prepare and output the process information in the same manner as the header, this programme makes use of **awk**. Similar to line 2's **printf** function, but utilising **$1**, **$2**, **$3**, **$4**, and **substr($0, index($0,$5))** to replace the placeholders with the relevant values taken from the **ps** output. The **substr** method finds the index of the fifth field (**$5**) in the full line (**$0**) and returns the substring beginning at that position to extract the command string.

The following line is used once again at the function's end to print the end of the table: “ **echo "+**--------------------------------------------------------------------------------**+"** “.

After these two blocks of bash code, we have the last 4 lines of the entire script. First is an Infinite while loop, “ **while true : do** ”.

The next command is “**clear**”, this command allows to clear the shell. After this line, the bash script calls the previous two functions. “**display\_system\_info()**” and “**display\_process\_info()**”.

After this section, the next line to run is “**sleep 1**” which allows creating a 1-second delay between each iteration.

The majority of the tools and commands that went into creating this system monitoring programme were chosen above the other options since they are virtually all common and accessible on any machine. As a result, all Ubuntu versions are now extensively compatible with the programme.

### b. Test Cases and Issues Faced

The program's testing revealed two problems. The line **"printf "|%-80s|n" "Total CPU Usage: $(mpstat | awk '$12 /[0-9.]+/ print 100 - $12 ')”** was the first and most obvious.

The mistake was the lack of the pre-installed command "**mpstat**" on some devices. Even though the script was still executed, there was a problem with the line when we looked at the overall CPU consumption. The output that appears is "**line 12: mpstat: command not found**". The following resolution to this problem was discovered after considerable investigation:

The "**mpstat**" command is a component of the "**sysstat**" package. The answer was to manually install this package. We must use these instructions in order to proceed:

-> **sudo apt-get update && sudo apt-get install sysstat.**

Concerning the output design, there was a second issue. The output table's lines all had an asymmetrical form. Changing the command "**echo**" with the "**printf**" tool was the answer to this problem. With the use of these instructions, the output could be given a customised format, which improved the table's shape.

These answers were made possible through study and the "**man**" internal function. This is analogous to manual. To utilise this functionality, enter **man** and the command you need assistance with.

### c. Final script

The following script is the final script, with all the changes applied and fully working. It can be stopped by pressing “**CTRL + C**”.

Graphical user interface

Description automatically generated

Figure . Final Bash Script

### d. Running Script

Graphical user interface, text

Description automatically generated

Figure . Script Running

# Task 2 – Content Management System Deploment

Deploying a Content Management System (CMS) for Suretide is the challenge at hand. The objective is to provide a digital platform so that interested parties and stakeholders may keep informed about the company's actions. The CIO has recommended utilising WordPress, a well-liked content management system (CMS) that runs 43.2% of websites globally.

The LEMP architecture, which stands for Linux (Ubuntu), Nginx, MySQL Community Server, and PHP, is anticipated to be used for the deployment. Installing and configuring these components as well as hardening the system to lower the risk of security exploits are required for the assignment. Specifically, setting up a WordPress database, making a user with the right database permissions, installing required PHP extensions, installing and configuring the Nginx web server, downloading and installing WordPress, configuring the firewall, and setting up the first landing page for Suretide are all objectives.

## I. Connecting Putty and Setting AWS Instances.

### a. AWS Configuration

After launching Learner Lab and logging into the AWS account, we proceeded to start a new instance.

Ubuntu must be chosen as the operating system, and then the right key pair must be generated and named. Additionally, we must enable the setting that permits HTTP traffic to the server.

After ensuring that, the correct options have been enabled, we must launch the instance by pressing the orange button.

Graphical user interface, application, email, website

Description automatically generated

Figure . AWS New Instance

Graphical user interface, application

Description automatically generated

Figure . AWS Key Pair & HTTP

Graphical user interface, text, application

Description automatically generated

Figure . Instance Running

### b. Putty Configuration

The next step is connecting to the Ubuntu server using PUTTY Tool. First, we will copy the IP address of the AWS instance.

We must configure the Putty programme with the IP address we acquired from the AWS server.

Graphical user interface, application

Description automatically generated

Figure . PuTTY Configuration

The key that we downloaded while setting up the instance must be entered in Putty. And we proceed to open the putty terminal.

Graphical user interface, text, application

Description automatically generated

Figure . Auth Key Putty

The result will be putty connected to the AWS server.

Graphical user interface, application

Description automatically generated

Figure . Connection with AWS

## II. Installing Updates and Upgrade Available Packages.

The commands which are going to be used are the following ones:

“**sudo apt update && sudo apt upgrade -y**”

Graphical user interface, text, application, Word

Description automatically generated

The system will begin updating and upgrading all of the available updates and packages as soon as you run the command. The yes option can be pre-entered when updating using the "-y" option. Without it, the system would first offer a yes/no question before beginning the updates.

## III. Installing Nginx

The command which is going to be used to install this tool is:

“**sudo apt install nginx -y**”

Text

Description automatically generated with low confidence

Once the nginx tool is installed we can check the status of it with the next command:

“**sudo systemctl status nginx**”

Text

Description automatically generated

## IV. Installing MySQL and Addons

Using this command “**sudo apt install mysql-server -y**”, MySQL server will be installed successfully. And to check if the installation has been successful, the command “**sudo mysql**” will initiate mysql.

The following command must be typed in order to exit MySQL interface, “**exit**”.

Graphical user interface, text

Description automatically generated

The next commands will install add-ons. “**sudo apt install tree -y**”,“**sudo apt install software-properties-common -y**” and “**sudo apt install vsftpd -y**”

Text, letter

Description automatically generated

## V. Installing and Configuring PHP

Installing the appropriate repository is necessary in order to install PHP in the correct version.

To install the repository, the next command is inserted in the terminal, “**sudo add-apt-repository ppa:ondrej/php -y**”.

Text, letter

Description automatically generated

Now, we'll use three commands to install PHP in version 8.1 and a few add-ons, which will enable PHP to function properly.

“**sudo apt install php8.1 -y**”, “**sudo apt install php8.1-fpm -y**” and “**sudo apt install php8.1-mysql -y**”.

Text

Description automatically generated

We will check if nginx is correctly running once again before downloading the WordPress.

## VI. Installing WordPress

The first command will be the next one:

“**sudo wget -O /var/www/html/latest.tar.gz** [**https://wordpress.org/latest.tar.gz**](https://wordpress.org/latest.tar.gz)”. This command downloads the latest WordPress version.

“**sudo tar -xzf /var/www/html/latest.tar.gz -C /var/www/html/**”. These lines start the extracting progress of the **latest.tar.gz** version which we previously downloaded and save it in the **/var/www/html/** directory.

**sudo:** runs the command with administrative privileges.

**tar:** is a command-line tool for archiving files and directories into a single file (known as a "tarball") or extracting files from a tarball.

**-xzf:** is a combination of options that tells tar to extract the contents of the archive file (x), to use gzip compression (z), and to display verbose output (v) while doing so.

**/var/www/html/latest.tar.gz**: is the path to the archive file that is being extracted.

**-C**: tells tar to change to the specified directory before extracting the files.

**/var/www/html/**: is the directory where the extracted files will be placed.

“**sudo mv /var/www/html/index.nginx-debian.html /var/www/html/index.html**”. This command renames the **file /var/www/html/index.nginx-debian.html** to **/var/www/html/index.html** using the **mv** command. The **sudo** command is used to execute the **mv** command with superuser privileges, which allows the command to make changes to system files in the **/var** directory.

This command is frequently used to switch out the default Nginx welcome page with a customised web page when a web server powered by a Linux distribution based on Debian is involved. When a user accesses the server's root URL, the Nginx web server will provide the customised web page rather than the standard welcome page by renaming **index.nginx-debian.html** to **index.html**.

A screenshot of a computer

Description automatically generated

Figure . Nginx Welcome Page

“**sudo cp /var/www/html/wordpress/wp-config-sample.php /var/www/html/wordpress/wp-config.php**”.

To move files or directories from one place to another, use the **cp** command.

This particular command copies the **wp-config-sample.php** file from **the /var/www/html/wordpress/** directory. to a brand-new document called **wp-config.php** in the same **/var/www/html/wordpress/** directory.

Since it contains configuration settings like database login information and security keys, the **wp-config.php** file is crucial to a WordPress installation.

The user can then change the **wp-config.php** file to include their own settings and credentials by copying the sample configuration file to the actual configuration file name.

“**ls /var/www/html/wordpress**”. Allows to check in the directory the files already created.

## VII. Configuration of Firewall

To start with this configuration, we will check the status of the firewall. The command to enter is the following, “**sudo ufw status verbose**”. We will extract the app list in the firewall by entering this command “**sudo ufw app list**”. After that we can just allow the **ssh** and the **nginx** service. With these commands:

“**sudo ufw allow ssh**” and “**sudo ufw allow ‘NGINX HTTP’**”

After allowing the respective services, we will enable the firewall with this command “**sudo ufw enable**”.

Now we will check the status again and it must show the rules applied and the firewall running.

Table

Description automatically generated with medium confidence

## VIII. Creating Database and User for WordPress

First, we will check the status of nginx, “**sudo systemctl status nginx**”. Once got the green signal from the nginx status, we forward to create the database “**suretidewordpress**” and the user “**suretidewordpressuser**”.

To get that we must edit the wp-config file. “**sudo nano /var/www/html/wordpress/wp-config.php**”.

This command will open an editor where three lines and an authentication key must be changed. Each user has a different key.

The three first lines to be modified are:

“define( 'DB\_NAME', 'database\_name\_here' );”. Here we have to put the database name upper indicated.

“define( 'DB\_USER', 'username\_here' );” In this line we must write suretidewordpressuser as indicated in the task 2 instructions.

“define( 'DB\_PASSWORD', 'password\_here' );” We can give any password of choice.

Text

Description automatically generated with medium confidence

Now we need to get the unique Auth key by entering to this link [**https://api.wordpress.org/secret-key/1.1/salt/**](https://api.wordpress.org/secret-key/1.1/salt/)

**A screenshot of a computer screen

Description automatically generated with medium confidence**

The next step is replacing this key in the same file we are editing.

Text

Description automatically generated

To save the changes we have to press “**CTRL + X**”, then press the key “**y**” and finally press “**enter**”.

Proceeding to the next steps by checking the status of nginx… After confirming nginx is running correctly we have to edit the next file.

“**sudo nano /etc/nginx/sites-available/default**”.

This command opens the configuration file for the default Nginx server block using the Nano text editor. We will set the WordPress service as default.

Once entered the nano editor, the following lines must be edited:

1. “**root /var/www/html;**” with “**root/var/www/html/wordpress;**”

This command sets NGINX web server to open the WordPress website as default.

2. “**index index.html index.htm index.nginx-debian.html;**” with “**index index.php index.html index.htm index.nginx-debian.html;**”.

We add the “**index.php**” because we are using PHP. In case we were using Apache2 instead of nginx these steps won’t be necessary.

3. “**server\_name \_**;” with “**server\_name localhost;**”.

Next, we have to replace the following lines:

#location ~ \.php$ {

        #       include snippets/fastcgi-php.conf;

        #

        #       # With php-fpm (or other unix sockets):

        #       fastcgi\_pass unix:/var/run/php/php7.4-fpm.sock;

        #       # With php-cgi (or other tcp sockets):

        #       fastcgi\_pass 127.0.0.1:9000;

        #}

With

location ~ \.php$ {

               include snippets/fastcgi-php.conf;

               # With php-fpm (or other unix sockets):

               fastcgi\_pass unix:/var/run/php/php8.1-fpm.sock;

               # With php-cgi (or other tcp sockets):

        #       fastcgi\_pass 127.0.0.1:9000;

        }

Graphical user interface, text, application, email

Description automatically generated

We must save this file by pressing "**CTRL + X**" after finishing all the replacements, then "**y**" and "**enter**" after that.

The rest of the process needs to be done in mysql-server. Before starting this process we must check if nginx is running correctly once again.

If yes, we continue with the next command which starts MySQL interface. “**sudo mysql**”.

Text

Description automatically generated

The next commands must be entered to finish creating the database and the user.

“**create database suretidewordpress default character set utf8 collate utf8\_unicode\_ci;”**

This line creates a new database with the name "suretidewordpress" and sets the default character set to "utf8" and collation to "utf8\_unicode\_ci".

“**create user 'suretidewordpressuser'@'localhost' identified by 'Password123!';**”

Creates the user called suretidewordpressuser and sets the password “Password123!”.

“**grant all on suretidewordpress.\* to 'suretidewordpressuser'@'localhost';**”

This command will allow all the privileges on the database to the user. To confirm these privileges the command “**flush privileges**” is required. And then enter “**exit**” to quit the mysql interface.

The last command which will finish this process successfully is:

“**sudo chown -R www-data:www-data /var/www**”. After this, we must restart nginx and then reload the localhost page. The result will be the login page of WordPress.

Graphical user interface, text, application

Description automatically generated

Figure . WordPress Registration

This site will be opened when we put in the url of the navigator the IP address of the AWS server. **This Ip address, changes each time the instance is restarted.**

Fill the details asked, and then press install WordPress button. The login page will be opened. Enter the login details which you used to register the user.

After completing the registration of the WordPress account, we will have the next interface.

A screenshot of a computer

Description automatically generated

Figure . WordPress Dashboard

## IX. Reconnection with WordPress

### a. Reconnection

To reconnect to the database and Wordpress, we need to run the instance and copy the new Ip address. This new IP address need to be replaced in putty configuration “**ubuntu@ipaddress**”.

Once putty running and connected to the instance, we will open mysql interface.

“**sudo mysql**”

We will enter the next command to check the databases created.

“**show databases;**” This will display all the databases created. (ours is **suretidewordpress**). To use this database, we will insert the following command “**use suretidewordpress;**”.

Now we have to select from that database the wordpress options table and update the previous old Ip address saved.

Table

Description automatically generated with medium confidence

“**select \* from `wp\_options` where `option\_name` in ('siteurl', 'home');**”

“**update `wp\_options` set `option\_value` = 'http://newipaddress' where `option\_name` = 'siteurl';**”

“**update `wp\_options` set `option\_value` = 'http://newipaddress' where `option\_name` = 'home';**”

Then we will exit the database, “**exit**”.

Now we will check the status of nginx with this command, “**sudo systemctl status nginx**”. If the nginx service is inactive, we will restart the service using the next command “**sudo systemctl restart nginx**”.

### b. Nginx Restart Failure

In case that the nginx service gives an error while restarting it and it doesn’t work, then we must follow the next process to fix this issue. This issue happens when the database and nginx server is not closed in the correct way.

Graphical user interface, text

Description automatically generated

First, we will need to install the net-tools, “**sudo apt install net-tools**”. The next command will be “**sudo netstat -tulpn**”. This command will show the ports blocked by apache2. Next to this information this tool will provide the PID of apache2. The following command must be run, “**sudo kill -2 <PID apache2>**”

Then we can use additionally the next two commands to clean the ports, “**sudo fuser -k 80/tcp**” and “**sudo fuser -k 443/tcp**”. Then we will proceed to restart nginx “**sudo service nginx restart**”.

Text

Description automatically generated

Table

Description automatically generated with medium confidence

Text, letter

Description automatically generated

This will solve the issue and nginx will be running successfully. The last step to is to enter this command “**sudo chown -R www-data:www-data /var/www**”. Now we can access back to the WordPress and the website.

To access to the WordPress dashboard, we need to use this link: **ipaddress/wp-admin.**

And to access only to the website, simply insert the **IP address** of the instance in a web browser.

## X. Connection of MySQL Workbench

To start the connection with the Workbench, first we need to allow the tcp forwarding. To get that, we need to install the tool “**ssh**”. “**sudo apt-get install ssh**”. Next, we will edit the config file of this tool, “**sudo nano /etc/ssh/sshd\_config**”. Entered the nano text editor, we need to look for the option “**# AllowTcpForwarding yes**”. We need to change it to “**AllowTcpForwarding yes**”.

Now we must restart the service, “**sudo systemctl restart sshd**”.

Now we need to open the Workbench and click on new connection.

In connection method, we will choose “**Standard TCP/IP over SSH**”. The following data need to be inserted in each please to get it working.

Ssh hostname, here we need to enter the **ipaddress** of the server and the port **22**.

Ssh username: “**ubuntu**”

Ssh password: in case we have password with the key, then we need to click on store in vault and insert the password.

On MySQL hostname: “**localhost**”

MySQL Server Port: “**3306**”

Username: “**suretidewordpressuser**”, if we have any other user, we can use it too.

Password: Here we need to input the password which we selected while we create the user.

Graphical user interface, application

Description automatically generated

Figure . MySQL Workbench Connection

After filling all the details, we need to press on test the connection and when its successful proceed with saving the connection.

Graphical user interface, application

Description automatically generated

Figure . MySQL Workbench Connection 2

Final connection made:

Graphical user interface, text, application, email

Description automatically generated

Figure . MySQL Connected

## XI. Test Results, Issues….

### Test 1 – Testing the Complete WordPress Installation.

While following the steps from the beginning, to check if the process is being done correctly, the command to check the Nginx Service status, and MySQL service has been done after each step. After receiving green light and confirming the welcome page of nginx is appearing, we forwarded to the installation of WordPress. To test and check if the installation was successful and working, we updated the page URL and the page which was showing was the WordPress registration.

### Test 2 – Reconnection WordPress and Website.

While doing the reconnection part the following issues where found. First the Ip address of the server changed, so we had to find a way to solve this issue. The solution was changing the IP addresses saved in the mysql database “**suretidewordpress**”. This process is done in the reconnecting section upper written.

To automatize this process, we can create a bash script, which will do all the steps to get the IP address updated. And additionally, we will add the script to CRON, which will automatically run the script every minute.

Text

Description automatically generated with low confidence

### Issues Found

The Nginx service error was the biggest problem I experienced. Cleansing ports 80 and 443 resolved the issue. Additionally, by removing any apache2 services from the port that can conflict with nginx. The inability of nginx to restart was caused by these problems.

## XII. Landing Page

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

# Task 3 – Automated remote database backup

## I. Introduction

Any organization's disaster recovery and business continuity plans must include database backups. In order to reduce risks that could have an impact on its business continuity, Suretide, a firm that depends on a MySQL database to support its operations, understands the value of routine backups and wants to make sure that backups are carried out at a set time each day. In order to complete this process automatically, we will write a scripted solution that runs a MySQL dump, adds the current date and time to the backup filename, transfers the backup file to a remote server, and adds the script to Cron to ensure automation with the necessary credentials. By stopping and restarting particular services as necessary, we will also handle any service dependencies that may have an influence on the backup procedure.

## II. Design

### a. Pseudocode

- backup\_file = "backupfile" + current date and time in the format of "%Y%m%d\_%H:%M:%S"

- execute the command "mysqldump -u suretidewordpressuser -pP@kistan786 suretidewordpress > backup\_file"

- if sftp command with the arguments "put backup\_file", "-o StrictHostKeyChecking=no", "-i backupkey.pem", "ubuntu@54.242.39.80:/script" succeeds:

- append the current date and time and "SFTP Transfer Successful" to the Backup\_Records.txt file

else:

- append the current date and time and "SFTP Transfer Unsuccessful" to the Backup\_Records.txt file

- delete the backup file

- log = "Backup Time " + current date and time in the format of "%H:%M:%S %Y-%m-%d"

- append log and "Database Backup Complete" to the Backup\_Records.txt file

- if the script is not in the crontab: # check if the script is already in the crontab

- add the script to the crontab with the argument "0 0 \* \* \* /home/ubuntu/script/scriptbackup1.sh"

- print "Script added to crontab"

### b. Script Explanation

“**backup\_file="backupfile$(date +"%Y%m%d\_%H:%M:%S").sql**". This command creates a variable called “**backup\_file**”. This variable will have the name of “**backupfile**”, followed by the “**date**” and after “**\_**”, it will have the full-time including seconds.

“**mysqldump -u suretidewordpressuser -pP@kistan786 suretidewordpress > "$backup\_file"**”, this command performs the mysql dumb using the user “**suretidewordpressuser**” with his credentials of the database “**suretidewordpress**”. And it saves the backup into the backup\_file variable.

“**if echo -e "put $backup\_file\nexit" | sftp -o StrictHostKeyChecking=no -i backupkey.pem ubuntu@54.242.39.80 :/script** “. This line will send using the sftp method, the backup file created to the backup server, using the “**backupkey.pem**” key and the ipaddress of the backup server. “**/script**”. Indicates the location of the key.pem file.

“**echo "$(date +"%Y-%m-%d %H:%M:%S") - SFTP Transfer Successful" >> Backup\_Records.txt**”   
If the connection has been made correctly and the sftp transfer has been completed, the system will print STFP transfer successful with the date and time. Is will save the message in the Backup\_Records.txt file also.

“**echo "$(date +"%Y-%m-%d %H:%M:%S") - SFTP Transfer Unsuccessful" >> Backup\_Records.txt**”. And if the connection has failed, the system will print an unsuccessful transfer of the backup. And it will save the message in the txt file.

“**sudo rm "$backup\_file"**”. This will clear the variable backup\_file.

“**log=$(date +"Backup Time %H:%M:%S %Y-%m-%d")**” , this will create a log and with the backup time and date.

“**echo "$log - Database Backup Complete" >> Backup\_Records.txt**” this will print and save the previous log created in the records file. If this file is not created it will create it by itself.

“**if ! crontab -l | grep -q "/home/ubuntu/script/scriptbackup1.sh"; then**” This command will check if the script has already been added in the cron, by using “**crontab -l**” function. If the user has the file in another location he must insert the correct one.

“**(crontab -l 2>/dev/null; echo " 0 0 \* \* \* /home/ubuntu/script/scriptbackup1.sh") | crontab –** “. If the scripts wasn’t added previously, the system will add the script to cron, and it will be runned each day at midnight.

# The 0 0 \* \* \* part of the line specifies that the script should run at midnight every day.

“**echo "Script added to crontab"**” # prints script added to crontab

## III. Implementation

### a. Creating Backup Server and Dumping Key

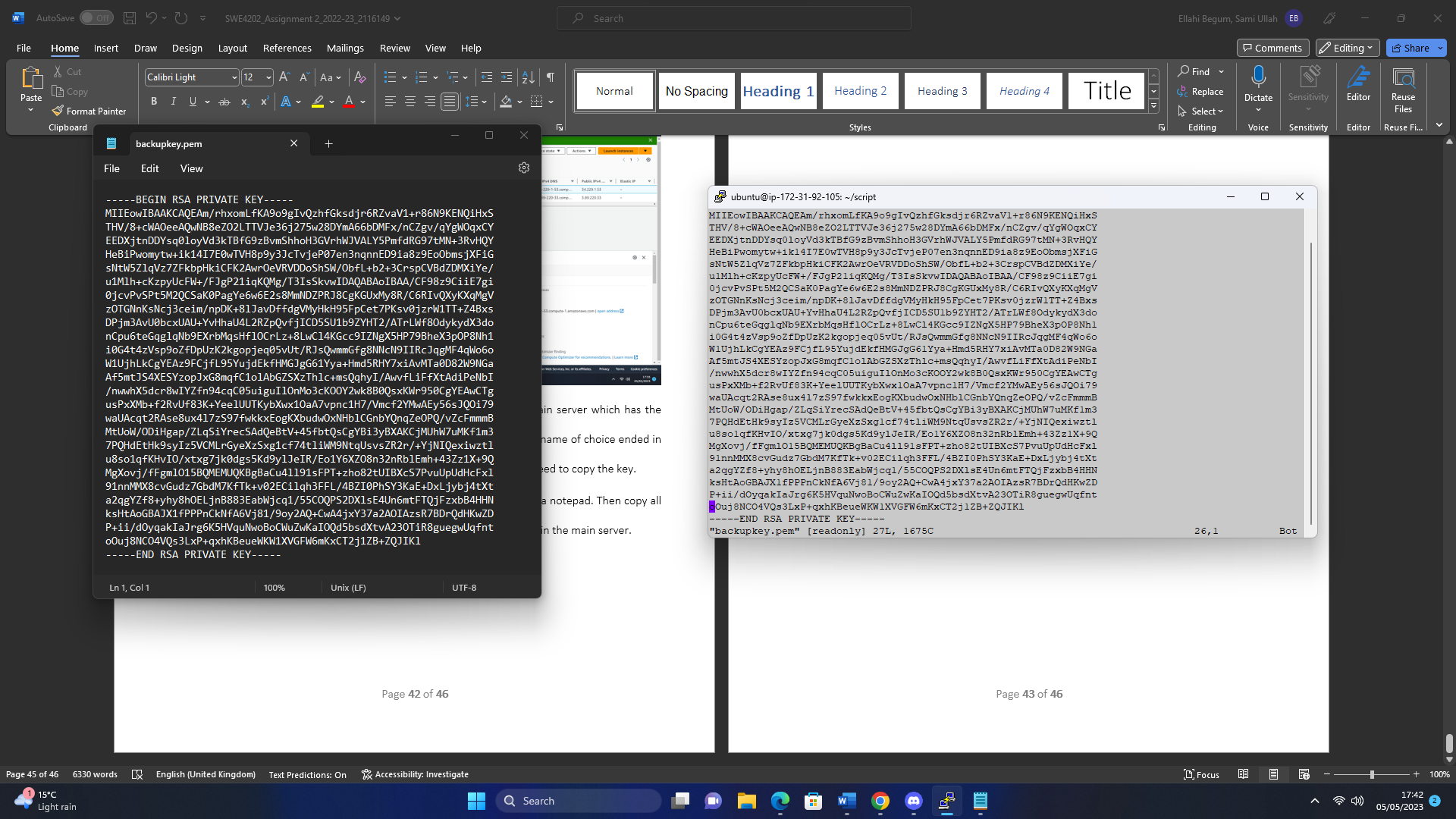
To create a backup, first we will create a new Instance (server) in AWS, and we will set a key in “**.pem**” format. The HTTP traffic option must be enabled. The process of opening an instance can be found in the task 2 section.

A screenshot of a computer

Description automatically generated

Now the key, which is paired with the instance, must be copied in the main server which has the database. We will create a new file with the “**touch**” tool and give the file a name of choice ended in .pem. Then we need to open the created .pem file with VI editor. Here we need to copy the key.

We need to open the key downloaded while creating the backup server, on a notepad. Then copy all the text which is the key and paste it in the .pem file which we have created in the main server.



After saving the file we need to make the file read-only. In case we skip this step, the system will detect the copied key as a public key and the backup dump will be failed.

To make the file read-only we need to use the chmod command, “**sudo chmod 400 filename.pem**”. now we can check the file permissions by using **ls** with “**-ltr**” option.

Text, letter

Description automatically generated

### b. Creating Script File

Once the key has been created and given the correct permission, we will create a .sh file using the touch tool. Next, we will open the file using vi or nano editor and paste all the commands previously created. These commands have been explained in the design section.

We need to check the ipaddress of the backup server and if it does not match we need to update it. Also we need to check the location and the name of the key.pem file. Another important thing is to check and insert correctly the username, the password and the name of the database which we have been using while creating the landing page. Or we can add a new user and grant him all the privileges.

We will have to use the following commands on mysql interface. First, we will open MySQL with

“sudo mysql”.

**CREATE USER 'username'@'localhost' IDENTIFIED BY 'password';**

**GRANT ALL PRIVILEGES ON \* . \* TO 'username'@'localhost';**

Once finished editing the script and saving it, we will give it the all the permissions needed.

“**sudo chmod 777 filename.sh**”. This command will change the permission of the file.

**Graphical user interface, text, application

Description automatically generated**

**Graphical user interface, application

Description automatically generated with medium confidence**

### c. Running Script and Checking Backup

Now we will proceed to run the script to create the mysql dump. Using any of the following commands we will run the script.

“**sudo bash filename.sh**”

“**sudo sh filename.sh**”

“**./filename.sh**”

**Graphical user interface, text, application

Description automatically generated**

After the script has run successfully, we will be able to see the output messages and we can check if the log file has been created by using “**ls**”.

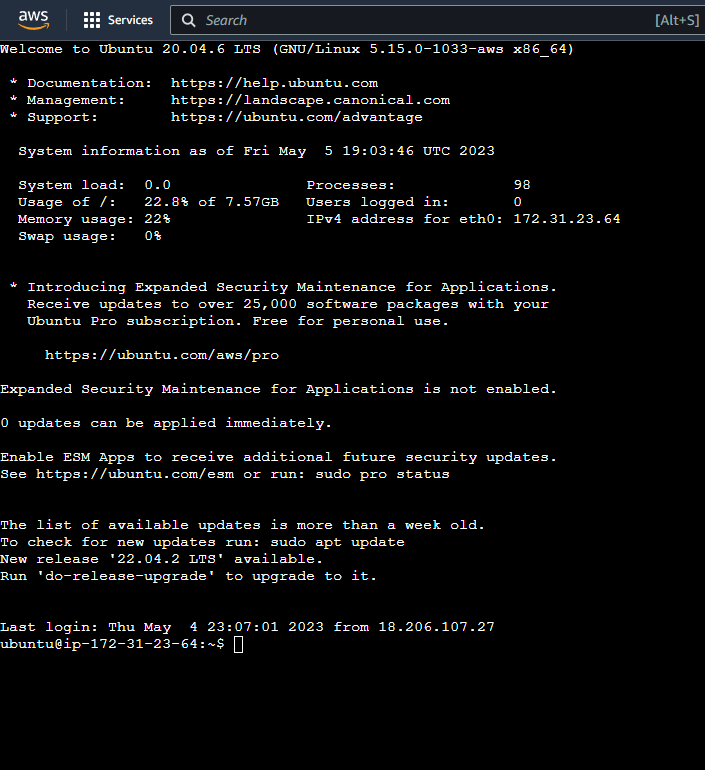
**A picture containing waterfall chart

Description automatically generated**

The last step is to check if the backup has been transferred correctly in the backup server. So, we will go the AWS dashboard, select the backup server, and connect to it via PUTTY or in the platform of AWS which is much faster.

**A screenshot of a computer

Description automatically generated**

****

In the backup server we will use again the “**ls**” command.

**Text

Description automatically generated**

We can observe that the transfer has completed correctly.

The script also has been added into CRON with the options to run each day at midnight. The following commands has been used to insert this feature:

# Check if the script is already in the crontab

**if ! crontab -l | grep -q "/home/ubuntu/script/scriptbackup1.sh"; then**

# here we need to put the location of your script and change it in the next line too.

# Add the script to the crontab

**(crontab -l 2>/dev/null; echo " 0 0 \* \* \* /home/ubuntu/script/scriptbackup1.sh") | crontab –**

# The 0 0 \* \* \* part of the line specifies that the script should run at midnight every day.

**echo "Script added to crontab"**  # prints script added to crontab

**fi**

## IV. Test Cases and Issues.

### Test 1

In the first test, we didn’t change the key file to read-only. And we got error of public key cannot be connected. This error was solved with the command “**sudo chmod 400 filename.pem**”, which changed the file permission to read-only.

### Test 2

While doing the second test after correcting the “public key” issue, we run the script and got the error of permission denied to the user. This error occurred due to the “**suretidewordpressuser**” did not had all the permission required to create a mysql dump.

Now to solve this issue the following command was used:

**GRANT ALL PRIVILEGES ON \* . \* TO 'username'@'localhost';**

After running this command on mysql, suretidewordpressuser got all the needed permissions, and the script finally run correctly.

## V. Final Script

## A picture containing text Description automatically generated

## VI. Script Running

### a. Before

### Text Description automatically generated

We only have one dump previously done. Once we run the script one more time, the log will be updated with another entry.

Also, we can see that there is only one file in the backup server:

Text

Description automatically generated

### b. After

After the script has been run, we have the following output and the log file updated.

Text, letter

Description automatically generated

Checking the number of backups transferred in the backup server will also show two backups.

Text

Description automatically generated

# Bibliography

https://stackoverflow.com/questions/35868976/nginx-service-failed-because-the-control-process-exited