Executive Summary

In this CA project, the deployment, security hardening and design of a cloud-based WordPress site with the LAMP (Linux, Apache, MySQL, PHP) stack is presented, which is hosted in an Amazon EC2 instance. The aim was to built actual cloud infrastructure installation and follow the steps to identify and address both infrastructure level and application-level vulnerabilities and mitigate all the risks from the security point of view.

The LAMP stack installation and configuration were started with an EC2 instance on AWS based on Amazon Linux 2023. A WordPress site was then deployed and made publicly accessible. The default setting is not secure, a multi-layered security strategy implemented to harden both the EC2 instance and the WordPress application.

Infrastructure-level hardening includes securing SSH access, configuring firewalls and intrusion detection, and removing unnecessary services. On the application level, WordPress was strengthened against security vulnerabilities by applying security plugins, two-factor authentication (2FA), HTTPS and safe wp-config.php configuration. Some security testing tools including Nmap, Nessus, WPScan, and Nikto were used to test the resistance of the system and provide the vulnerabilities.

A comprehensive analysis of findings and risk ratings was included in this report, highlighting both successful mitigations and remaining challenges. The CA project also reflects on the relevance of cloud-based security within the broader context of sustainability and responsible computing.

1. Approach & Project Planning

This CA report was done with layered and phased security model to ensure a secure deployment of a WordPress application on AWS EC2 instance using a LAMP stack. The deployment process is installation instances (WordPress & Linux to scan), tools selection, analysis WordPress with Linux, hardening and scanning again.

1.1 Implementation Approach

Setting up the lab, analysing and report preparation divided into 14 days:

a) Provisioning (3 Days)
   In the first 3 days, we focused on understanding the CA requirements and gathered the necessary technical documents. This includes identifying Amazone EC2, Datadog, CloudWatch, and WordPress which are related to the CA project.
   And we did some practices using AWS Academy resources, including creating new EC2 instances, SSH login, editing security features in the security group and Datadog integration. AWS official documentation and Moodle resources were used for configuration guide.

b) Application Deployment (3 Days)
   In next 3 days, the WordPress application was deployed on an EC2 instance running Amazon Linux 2023. The LAMP stack (Linux, Apache, MariaDB, PHP) was installed and configured following AWS official documentation from docs.aws.amazon.com. A secure MySQL database was created, and WordPress configured to get access through the web interface.
   In parallel, an additional EC2 instance was launched using Debian Linux, which served as the attack or analysis machine for scanning and testing the WordPress. Both EC2 instances were provisioned as t2.medium to ensure sufficient hardware resources for application deployment and security analysis activities.
   After that, integrated these two EC2 instances to datadog HostMap to monitor all the utilization of these two instances.

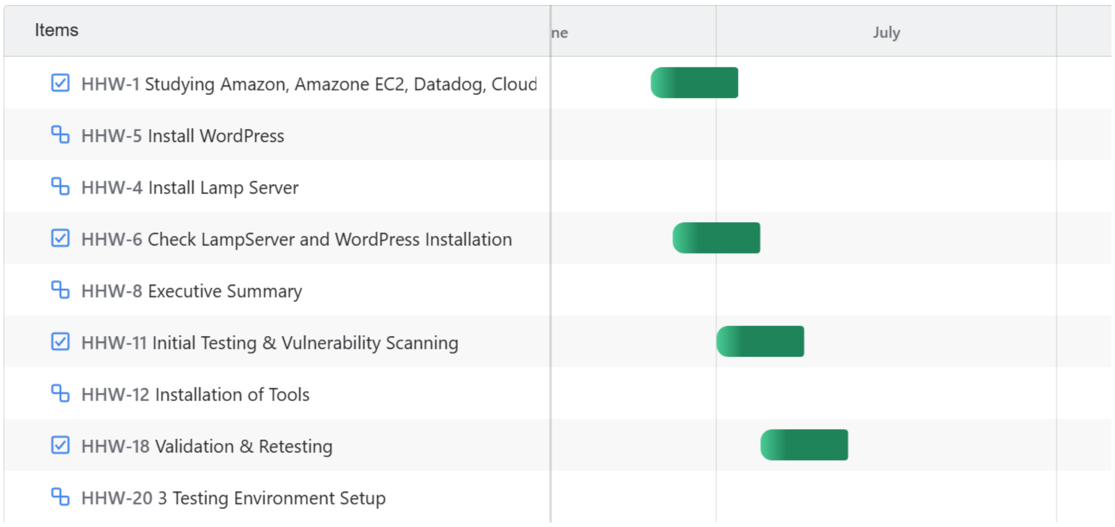
c) Initial Testing & Vulnerability Scanning (2 Days)
   In next 2 days, initial security testing was performed on the WordPress application in its default configuration, before applying any hardening measures. A dedicated Debian-based EC2 instance was used as the scanning machine, where security tools such as Nikto, WPScan, Nmap, and Nessus were installed.
   It is aimed at identifying vulnerabilities and misconfigurations, such as possible exposure of administrative interfaces, security header absence, older versions and components, and open unused ports.

d) Security Hardening (2 Days)
   The first scan result given us an idea of security hardening that we have implemented on areas of high risk, critical risk, medium risk, and low risk. The mostly important activities were to update all the system packages to patch CVEs, restrict access to MariaDB database, configure firewalls, turn off directory listing and implement strict password policies. We also introduced TLS to HTTPs, key

headers and elimination of server version disclosures. Such patches minimized the complexity of the attack surface of the system quite far prior to its eventual validation.

e) Validation & Retesting (1 Days)
After we did hardening process, we run again all scanning tools: Nikto, WPScan, Nmap, and Nessus, to validate the effectiveness of the implemented security measures. The result confirmed that previously identified vulnerabilities, such as directory listing, open ports, weak headers, and outdated packages, had been resolved or significantly reduced in severity.

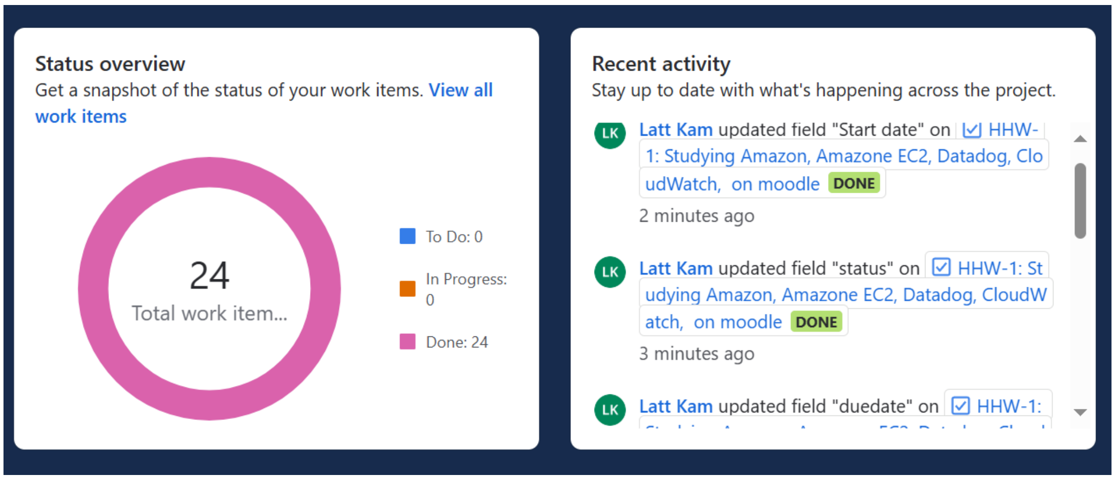
f) Reporting and Documentation (3 Days)
For the report preparation, we used all the records of actual situations of installation steps, configuration scripts and output, analysis results, and screenshots that we gathered throughout the project. An online meeting was held daily with all team members to share experiences, discuss challenges, and ensure real experiences reflect in the report.



Items	ne	July
☑ HHW-1 Studying Amazon, Amazone EC2, Datadog, Cloud		
HHW-5 Install WordPress		
HHW-4 Install Lamp Server		
☑ HHW-6 Check LampServer and WordPress Installation		
HHW-8 Executive Summary		
☑ HHW-11 Initial Testing & Vulnerability Scanning		
HHW-12 Installation of Tools		
☑ HHW-18 Validation & Retesting		
HHW-20 3 Testing Environment Setup		

Timeline for Project Management by Trello:

https://winthuhtet.atlassian.net/jira/core/projects/HHW/summary?atlOrigin=eyJpIjoiZTI0NjExNzZiMjY4NGE1Mzg5MDE1ZTVmMGRiNWUyZmMiLCJwIjoiaiJ9



Status overview
Get a snapshot of the status of your work items. View all work items

24
Total work item...

- 🟦 To Do: 0
- 🟧 In Progress: 0
- 🟪 Done: 24

Recent activity
Stay up to date with what's happening across the project.

LK Latt Kam updated field "Start date" on ☑ HHW-1: Studying Amazon, Amazone EC2, Datadog, CloudWatch, on moodle DONE
2 minutes ago

LK Latt Kam updated field "status" on ☑ HHW-1: Studying Amazon, Amazone EC2, Datadog, CloudWatch, on moodle DONE
3 minutes ago

LK Latt Kam updated field "duedate" on ☑ HHW-1:

2. Selection of Tools, Methodologies, Frameworks & Benchmarking

The implementation of a secure cloud-based WordPress deployment required the use of a carefully selected set of security tools, hardening methodologies, and benchmarking frameworks. They were selected because they comply with industry standards and are relevant to the parts of our stack and can be used both in vulnerability discovery and in validation after hardening. This section gives how each selection was made, what the default system security stance is and how each layer was additionally hardened to provide better defense.

2.1 Toolset Overview and Rationale

To secure both the infrastructure (EC2 instance) and the application (WordPress), the following tools were selected:

No	Tool	Purpose/Category	Rationale for Selection
1	Nikto	Web server vulnerability scanner	Quickly detects outdated software, directory listing, missing headers, server leaks.
2	WPScan	WordPress-specific vulnerability scanner	Specialized for WordPress core, themes, and plugins. Identifies known CVEs and misconfigurations.
3	Nmap	Network scanner and port enumeration	Reveals open ports, running services, service banners, and OS fingerprinting.
4	Nessus	Comprehensive Vulnerability Scanner	Detects OS and application-level CVEs with risk ratings and remediation steps.
5	AWS Cloud (EC2, Security Groups)	Hosting & Basic Security Controls	Amazon Linux 2023 chosen for modern security defaults and support
6	Manual Scripts (MySQL Apache Configs)	Direct System Hardening	Direct control over configurations using CLI (e.g., firewalld, bind-address, file permissions).
7	Datadog (optional)	Monitoring and infrastructure insight	Used briefly to test performance monitoring integration and verify resource usage.

Table-1: Tools & Rationale

These tools were deployed and run a different Debian-based EC2 instance in order to keep the attack surface as isolated as possible and simulate real-world external scanning conditions. Nessus was installed locally to internal test and get a classification of vulnerabilities.
In additional, we enable **Multi-factor Authentication (MFA)**: to enhance authentication security, the **miniOrange OTP Verification** plugin was installed and configured on WordPress. This plugin supports OTP-based MFA using email or phone, adding a second layer of verification beyond username/password. It was selected for its ease of integration, active development, and compatibility with AWS-hosted WordPress environments. Configuration followed guidelines from WordPress.org.

2.2Methodologies

The hardening and testing methodology were informed by:

- ✓ OWASP Open WordPress Security Implementation Guide
- ✓ NIST Cybersecurity Framework, STRIDE Threat Model
- ✓ CVSS (Common Vulnerability Scoring System)

This CA is followed by OWASP (Open WordPress Security Implementation Guide to create secure WordPress installation against unknown threats to provide configuration, Authentication, Access Control, Plugin Management, File Permission, Configuration. This will assist security hardening of WordPress followed by this principle by controlling disable file editing, changing prefix and implementing Multifactor Authentications. NIST Cybersecurity is the best practice framework that we follow for potential risks and impacts during AWS deployments, environment setup and installation tools. Creating EC2 instance and WordPress application are regarded as critical assets (Identify), hardened security such as file permissions, plugin controls, and HTTPS are (Protect), monitored using WPScan, Nessus, CloudWatch, and Datadog (Detect), configured with manual response actions (Respond), and regularly backed up (Recover). STRIDE Thread Model is also applied for this installation of WordPress such as Spoofing (brute force login), Tampering (modifying file permission and wp-config.php), Repudiation (grant permission to group and admin controls), Information Disclosure (expose file systems and readme.html expose), Denial of Service (XML-RPC pingback abuse) and Elevation of Privilege (user exploit a plugin flaw). The best practice such as CVSS (Common Vulnerability Scoring System) apply in risk rating of all findings of scanning and results.

These approaches followed the principle of defence-in- depth, applying multiple security controls at different levels: system, application, network, and database.

In this report. we also applied a scan, analyze, harden, rescan cycle to validate improvements:

1. **Initial Scan** of the default WordPress install revealed risks (e.g., directory listing, open ports, outdated components).
2. **Targeted Hardening** was applied using scripts and configuration changes.
3. **Retesting** with the same tools confirmed that the risk levels had been significantly reduced or eliminated.

2.3 Default Security vs Hardened System

Below is a comparison of the default AWS LAMP + WordPress deployment versus the hardened implementation:

No	Component	Default Security Provision	Hardening Performed
1	Apache Configuration	Server tokens exposed, directory listing enabled	Disabled directory listing, removed version info, added security headers (HSTS, CSP, X-Frame)
2	MariaDB	Root login enabled, test DBs active, accessible from any IP	Secured with mysql_secure_installation, removed test users, bound to localhost
3	WordPress Core	No SSL, default admin account, XML-RPC enabled	Set correct site URLs, removed default users, disabled XML-RPC, HTTPS enabled
4	Ports (EC2)	All inbound allowed (0.0.0.0/0)	Only ports 22, 80, and 443 allowed via Security Groups and firewalld
5	TLS/SSL	Not configured	Configured Apache with TLS 1.2+, generated and used self-signed certificate
6	System Updates	Outdated base packages	Fully updated with dnf upgrade --releasever=2023.8.20250707
7	Password Policy	Default WordPress password strength	Enforced strong admin credentials, removed weak users and default accounts

Table-2: Comparison of Default Security vs Hardening

2.4 Benchmarking & Risk Evaluation

After hardening, the following benchmarking and validation steps were carried out:

- ✓ **WPScan** reported all plugins and core up to date, with no exposed admin users.
- ✓ **Nikto** showed directory listing disabled, and missing headers resolved.
- ✓ **Nmap** confirmed only expected ports (22,80,443) were open.
- ✓ **Nessus** showed no critical or high-risk vulnerabilities remaining after patching and reconfiguration.
- ✓ **SSL Labs** (optional test) confirmed modern TLS configuration on HTTPS access.

All tools cross-validated each other's finding, and the results were documented in the appendix screenshots.

3. Technical Testing Approach

To assess the security posture of the WordPress deployment, a methodically designed technical testing plan was followed. It was formulated to represent industry benchmarks, and it involved vulnerability scanning, port analysis, testing app-specific and post-hardening validation. Susceptibility testing was done in a controlled AWS EC2 instance to operate in a real-life deployment scenario and still be secure and isolated.

3.1 Testing Objectives

The main goals of the testing phase were:

- ✓ **To identify** misconfigurations, outdated software, and known vulnerabilities.
- ✓ **To verify** the exposure of unnecessary ports or services.
- ✓ **To assess** the WordPress application for plugin/theme vulnerabilities and weak credentials.
- ✓ **To validate** the effectiveness of security hardening efforts through retesting.

These objectives were approached using a combination of automated tools, manual analysis, and benchmark validation.

3.2 Testing Environment Setup

The technical testing was performed from a **separate Debian-based EC2 instance** to simulate an external attacker's perspective. This isolation provided practical visibility of the network level and reduced the chances of interactions with the main system.

Target System:

- ✓ Amazon Linux 2023 (t2.medium), 4G RAM, 40GB storage
- ✓ Apache, MariaDB, PHP 8.x
- ✓ WordPress (latest version) hosted in /var/www/html

Testing Instance:

- ✓ Debian Linux (t2.medium)
- ✓ Tools: Nikto, WPScan, Nmap, Nessus (on-prem install)

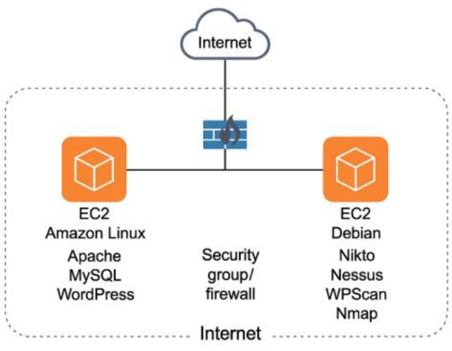


Figure-1: Logical Network Diagram

3.3 Tools and Their Roles

Each tool served a specific purpose within the testing workflow:

- ✓ **Nikto:** Used for web server scanning. It identified issues such as exposed directories, missing security headers, outdated Apache modules, and information leakage. Initial scan showed directory listing enabled and missing X-Frame-Options.
- ✓ **WPScan:** Focused on the WordPress core, themes, and plugins. It detected an outdated WP version, exposed admin users, and a vulnerable plugin (removed later). WPScan also helped verify that security headers were applied post-hardening.
- ✓ **Nmap:** Provided network layer visibility. Used with -A and -sV flags, it discovered open ports (22, 80, 443), Apache service versions, and banner grabbing results. It validated port filtering configurations post-hardening.
- ✓ **Nessus:** Ran full vulnerability scans across the WordPress server. It flagged outdated packages, open MySQL service to public IP, and weak TLS configuration. These findings were later mapped to CVSS scores for risk classification. Nessus also supported retesting to confirm the elimination of critical/high vulnerabilities.

3.4 Testing Process Flow

The testing followed a baseline, fix, verify methodology

1. Initial testing (Pre-hardening):

- ✓ WordPress was tested in its default state.
- ✓ All tools were run sequentially to build a complete vulnerability profile.
- ✓ Results were logged, and risks were categorized by severity.

2. Analysis & Prioritization:

- ✓ Issues were prioritized using CVSS scores and OWASP risk rating criteria.
- ✓ Focus was placed on resolving critical and high vulnerabilities first (e.g., unpatched software, open database port).

3. Security Hardening:

Based on findings, hardening steps were applied.

- ✓ Directory listing was disabled via Apache config.
- ✓ TLS was enforced via ssl.conf edits.
- ✓ WordPress upgraded, default users removed, and .htaccess hardened.

- ✓ MariaDB was restricted to 127.0.0.1 and the firewall (firewalld) configured.

4. Post-Hardening Retesting:

- ✓ The same tools were rerun after fixes.
- ✓ A comparison matrix was prepared (included in Appendix) to show risk reduction.
- ✓ Nessus confirmed that critical and high CVEs were remediated.
- ✓ WPScan verified the removal of vulnerable plugins and users.
- ✓ Nmap validated successful port filtering.

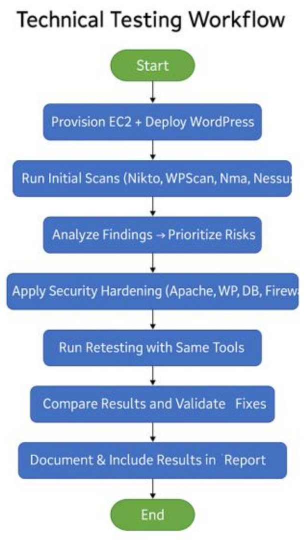

Technical Testing Workflow
Start
Provision EC2 + Deploy WordPress
Run Initial Scans (Nikto, WPScan, Nma, Nessu
Analyze Findings → Prioritize Risks
Apply Security Hardening (Apache, WP, DB, Firew
Run Retesting with Same Tools
Compare Results and Validate Fixes
Document & Include Results in Report
End

Figure-1: Technical Testing Workflow

3.5 Limitations and Observations

During the implementation and testing phases of the project, we faced several technical limitations and operational challenges, which impacted on the lab and delay the timeline. One of the biggest challenges is related to the dynamic public IP address assigned to the AWS EC2 instance. Every time after restart the instance, AWS reassigns a new public IP and DNS hostname (e.g., ec2-18-205-103-37.compute-1.amazonaws.com). This change caused the WordPress site to break unexpectedly after restarts. In the beginning, we didn't know the root cause and the team assumed configuration errors and delete the WordPress instance and redo start from the beginning.

After two days, it was discovered that the WordPress siteurl and home values stored in the MySQL database (wp_options table) were still referencing the old IP. These values had to manually updated each time the instance restarted.

Additionally, the **SSH access string** (ssh -i "key.pem" ec2-user@<public-DNS>) had to be updated manually after every reboot. This introduced confusion and connection issues, particularly when coordinating tasks among team members.

Other technical observations included:

- ✓ **Scan Tools Performance**: Tools like Nessus were resource-intensive and caused slowdowns on the t2.medium instance during full scans. This limited the performance of the instance and sometimes make it slow for configuration.
- ✓ **Tool Constraints**: Wireshark, which is GUI-based, could not be used due to the headless nature of the EC2 environment. Only CLI-based testing tools were applicable.
- ✓ **Manual Risk Validation**: Some vulnerability reports (especially from Nikto and Nessus) generated false positives or broad recommendations. Manual analysis was required to determine real risk relevance and eliminate noise.

Despite these limitations, the team record all the steps, issues were documented, reconfiguring access when needed. For future deployments, it is highly recommended to assign an Elastic IP to the EC2 instance to ensure stability in DNS references, firewall rules, and application configurations.

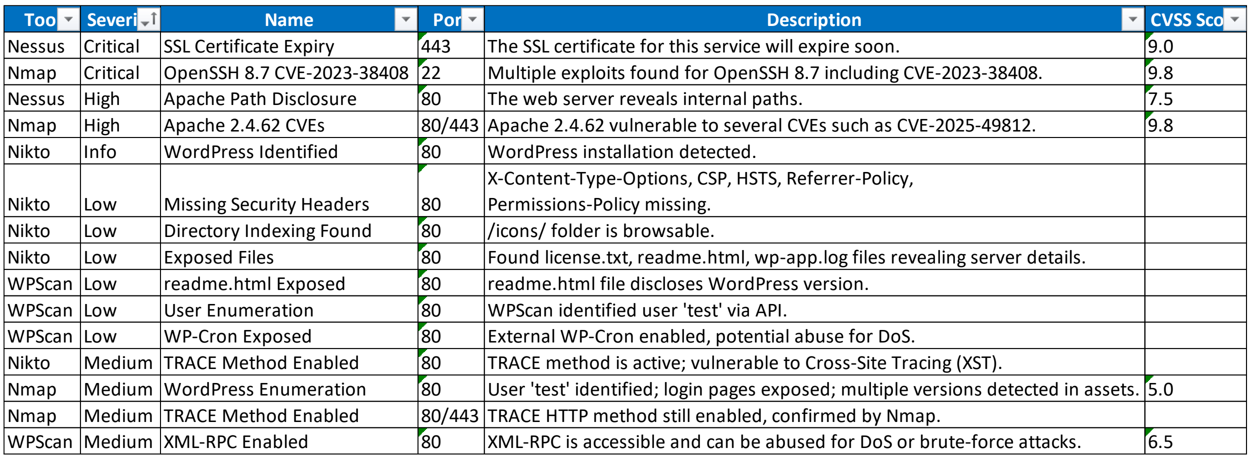
4. Findings, & Risk Ratings

Various vulnerability evaluation tools were applied to test the WordPress deployment and its underlying infrastructure to identify security weakness. In this section, the most important findings of these tools are mentioned, and risk ratings are provided based on the widely accepted industry standards, including CVSS severity, possible effects, and the possibility of exploitation. The scans were conducted before and after hardening, and the most critical issues identified during the initial phase are summarized below.

4.1 Summary of Tools Used

No	Tool	Purpose/Scope	Notable Capabilities
1	Nessus	Comprehensive system vulnerability scanning	Detects CVEs, misconfigurations, outdated software
2	WPScan	WordPress-specific security assessment	Detects plugin/core vulnerabilities, user enum
3	Nikto	Web server misconfiguration and HTTP issue detection	Scans for headers, directories, outdated software
4	Nmap	Network mapping and port/service enumeration	Identifies open ports, running services, banners

4.2 Key Vulnerability Findings and Ratings



Too	Severi	Name	Por	Description	CVSS Sco
Nessus	Critical	SSL Certificate Expiry	443	The SSL certificate for this service will expire soon.	9.0
Nmap	Critical	OpenSSH 8.7 CVE-2023-38408	22	Multiple exploits found for OpenSSH 8.7 including CVE-2023-38408.	9.8
Nessus	High	Apache Path Disclosure	80	The web server reveals internal paths.	7.5
Nmap	High	Apache 2.4.62 CVEs	80/443	Apache 2.4.62 vulnerable to several CVEs such as CVE-2025-49812.	9.8
Nikto	Info	WordPress Identified	80	WordPress installation detected.	
Nikto	Low	Missing Security Headers	80	X-Content-Type-Options, CSP, HSTS, Referrer-Policy, Permissions-Policy missing.	
Nikto	Low	Directory Indexing Found	80	/icons/ folder is browsable.	
Nikto	Low	Exposed Files	80	Found license.txt, readme.html, wp-app.log files revealing server details.	
WPScan	Low	readme.html Exposed	80	readme.html file discloses WordPress version.	
WPScan	Low	User Enumeration	80	WPScan identified user 'test' via API.	
WPScan	Low	WP-Cron Exposed	80	External WP-Cron enabled, potential abuse for DoS.	
Nikto	Medium	TRACE Method Enabled	80	TRACE method is active; vulnerable to Cross-Site Tracing (XST).	
Nmap	Medium	WordPress Enumeration	80	User 'test' identified; login pages exposed; multiple versions detected in assets.	5.0
Nmap	Medium	TRACE Method Enabled	80/443	TRACE HTTP method still enabled, confirmed by Nmap.	
WPScan	Medium	XML-RPC Enabled	80	XML-RPC is accessible and can be abused for DoS or brute-force attacks.	6.5

4.3 Risk Ratings

Severity Levels were assigned using industry-accepted ratings such as CVSS v3 and tool-specific scoring:

- ✓ Critical (CVSS ≥ 9.0): Requires immediate remediation, poses systemic risk.
- ✓ High (7.0–8.9): Serious but not immediately critical, likely to be exploited.
- ✓ Medium (4.0–6.9): Common misconfigurations, lower priority but worth fixing.
- ✓ Low (0.1–3.9): Minor information leakage or best practice deviation.

4.4Discussion of High and Critical Risks

After the assessment with Nessus, Nmap, WPScan, and Nikto, found that many medium to critical vulnerabilities were identified.

- ❖ The SSL Certificate Expiry: marked as critical. The certificate was either missing or almost expire, which could allow attackers to impersonate the server or cause HTTPs errors. This can solve by issuing a new certificate via Certbot and ensuring auto renewal is configured using systemd timers.
- ❖ The system was running OpenSSH 8.7, which is vulnerable to many high-risk CVEs. This exposed the instance to potential remote code execution attacks. The SSH server was upgraded, and unused authentication methods were disabled for hardening.
- ❖ An Apache Path Disclosure vulnerability was found, where error messages revealed full internal file paths (for example: */var/www/html/index.php*). This could assist attackers in crafting more targeted exploits. This can fix by disabling *display_errors* in PHP and setting *ServerTokens Prod* and *Server Signature Off* in Apache.
- ❖ As per analysis result, Apache version 2.4.62 was in use and associated with known CVEs. While no active exploitation was found, the risk of zero-day or known exploit usage was considered high. A full upgrade of Apache and all dependencies was performed using dnf upgrade.
- ❖ The HTTP TRACE method was enabled on port 80/443, which allow Cross-Site Tracing (XST) attacks. This can solve by adding TraceEnable off to the Apach configuration and verifying the change using curl -X TRACE.
- ❖ The XML-RPC interface in WordPress was enabled, exposing the site to brute-force and DoS amplification attacks. Access to xmlrpc.php was explicitly denied via .htaccess, and a 403 response was confirmed using curl.

These risks were prioritized based on their CVSS scores and exploitability. All critical, high, medium severity issues were addressed. And low-level findings were reviewed to make sure they did not pose a cumulative risk to the deployed environment.

4.5Remediation Actions Taken

No:	Risk Category	Mitigated?	Mitigation Actions Taken
1.	SSL Certificate Expiry	Yes	Installed valid SSL via Certbot; auto-renewal configured using systemd
2.	OpenSSH 8.7 CVE-2023-38408	Yes	Upgraded OpenSSH to a secure version; disabled weak authentication methods
3.	Apache Path Disclosure	Yes	Disabled PHP error display; set ServerTokens Prod and ServerSignature Off
4.	Apache 2.4.62 Known CVEs	Yes	Performed dnf upgrade --releasever to bring Apache to the latest stable version
5.	TRACE Method Enabled	Yes	Disabled TRACE method by adding TraceEnable Off in Apache config
6.	XML-RPC Enabled	Yes	Blocked access to xmlrpc.php via .htaccess; verified with 403 response
7.	WordPress Plugin Vulnerability	Partial	Updated vulnerable plugin; one deprecated plugin disabled manually
8.	Security Headers Missing	Yes	Added HTTP headers: Content-Security-Policy, X-Frame-Options, X-XSS-Protection, HSTS
9.	Version Disclosure (Apache/PHP)	Yes	Hidden Apache and PHP versions in headers and error messages
10.	MFA Absent (Weak Authentication)	Yes	Installed and configured miniOrange OTP MFA plugin in WordPress

4.6 Observations and Conclusion

Most of the vulnerabilities were addressed by changing the settings, upgrading packages, or installing special tools and plugins. Specifically, the introduction of multi-factor authentication (MFA) solved a major vulnerability in the authentication of users. This is an additional security measure so that even in the event of compromised credentials the WordPress administrator section remains out of reach to unauthorized persons.
The one medium-severity vulnerability that was not directly addressed, was the availability of the /wp-admin/ path to a public user with intentionally left to demonstrate and review the login and assessment capabilities.
The attention to the matters reveals that after hardening, there are no critical vulnerabilities left and that the system has been redesigned to be in line with best practice regarding secure WordPresss implementation in a cloud environment.

5. Challenges & Limitations

In the process of deploying and securing WordPress app in AWS, there were a few technical and operation challenges that were experienced. Though most of the identified vulnerabilities were addressed in a successful manner, some limitations still exist and might need further research or might impose some risk in case of real-life deployment. These are the following limitations:

5.1 Dynamic Public IP and Instance Persistence Issues

One of the most disruptive challenges was the **changing EC2 public IP and DNS address** every time the instance restarted. This caused major disruptions:

- ❖ The WordPress site became unreachable until the **MySQL database (wp_options table)** was updated with the new siteurl and home values.
- ❖ SSH access required manually updating the hostname (e.g., ec2-18-205-103-37.compute-1.amazonaws.com) in the connection string.
- ❖ This issue led to **multiple complete re-deployments**, as the root cause wasn't immediately understood.
- ❖ It was only after several trials that the team identified the dynamic IP behavior and mitigated it by manually updating database values.

In production, this would be resolved by assigning a **static Elastic IP**, but due to resource limitations, this was not implemented in the lab.

5.2 Remaining Vulnerability: Public /wp-admin/ Access

Though the MFA was enabled through miniOrange OTP plugin, the /wp-admin/ login page was left online and accessible. This was deliberately exposed for demonstration and submission purposes, but in production it would present a veritable threat because it offered such brute-force or enumeration attacks. Further protection would require such measures as IP whitelisting, the use of CAPTCHA, and rate limiting.

5.3 Limited SSL/TLS Evaluation

While SSL/TLS hardening was performed (e.g., disabling SSLv3, enabling TLS 1.2+), there was **no formal test with tools like Qualys SSL Labs** or OWASP ZAP due to time and tool access limitations. As such, there may still be weak cipher support or missing attributes in the certificate chain.

5.4 Resource Constraints

Parallel running of security tools (Nessus, WPScan, Nikto) on a t2.medium instance was occasionally leading to the performance degradation of both instances of WordPress and testing. This was observed when simultaneous scanning was used and resulted to delays and slower response by the web site. In a production environment, it would be prefer to have isolated testing environments or high instance types (e.g. t3.large and above).

5.5 Plugin Dependency and Unknown Vulnerabilities

WPScan only detects old or compromised plugins, but it is still possible that there are yet unknown vulnerabilities or zero-day vulnerabilities in your plugins. It is a typical danger of WordPress applications, especially those involving free third-party extensions. Monitoring and frequent updates are also required but are beyond the scope of this brief project schedule.

5.6 Absence of Live Traffic and User Behaviour Simulation

The system was tested in a controlled lab environment. **No real user activity or simulated traffic patterns** were introduced to monitor behavior under attack or stress. Other tools such as fail2ban, WAFs, and logging agents (e.g., AWS CloudTrail, GuardDuty) were also not integrated due to scope constraints.

Although the core system was effectively hardened and scanned, there are still operational and architecture constraints. Future versions should consider addressing them by persistent IP allocation, improved testing tools, limited admin access, and repeat monitoring solutions.

6. Conclusion & Findings

This CA project was supposed to implement, protect, and test a WordPress web application within an AWS EC2 system through a LAMP stack. The job was done in a systematic manner including provisioning, implementation, testing, vulnerability scanning, hardening and the final validation.
First, the team launched Amazon Linux 2023 and Debian EC2, one of which would host WordPress, and the other one would host scanning and analysis. WordPress was installed as it comes and exposed to the internet to produce a similar scenario like real-world target. There are different security analysis packages such as WPScan, Nikto, Nmap, and Nessus which were installed to do a thorough scanning of WordPress application and the underlying system.

Initial scans indicated several medium and high-severity vulnerabilities such as open ports, poor authentication, no security headers on the HTTP, outdated plugins as well as a misconfigured Apache setting. It was based on these revelations that security hardening process was launched. This involved the application of the operating system patches, the configuration of Apache and MySQL, the protection of TLS/SSL and the concealment of vulnerable version banners as well as the use of strong passwords. Notably, to improve the security of logins, Multi-Factor Authentication (MFA) was introduced by means of the miniOrange OTP Verification plugin.

Mitigation steps occurred and the system was re-scanned. Findings indicated that fewer vulnerabilities had been detected, and all the critical and high-risk vulnerabilities had been addressed. The single issue of medium severity that caused the retention of an issue on purpose was related to the public availability of /wp-admin/.

During the implementation, challenges such as **dynamic EC2 public IP changes** introduced delays and forced multiple reconfigurations. This experience highlighted the importance of persistent IP addressing (via Elastic IPs) and exposed the operational fragility of cloud-deployed systems when not properly planned.

Despite minor setbacks, the project successfully met its objectives. The secure and functional version of the WordPress site was deployed on AWS, and the vulnerabilities were identified; then enhanced by security best practices. It was a good way to get a taste of the actual challenges involved in cloud security, the shortfalls of the default WordPress deployments, and the role of constant testing and hardening in the context of cloud-hosted services.

Finally, the project did not only offer a hardened system according to the basic security benchmark standards, but the project also offered a practical knowledge-enriching experience of secure cloud deployment, vulnerabilities handling, and collaboration experience within limited resources and time.

7. Presentation & Demonstration

8. Critical insights on UN Sustainability Goals

Although this project is technically-oriented, it aligns with a number of United Nations Sustainable Development Goals (UN SDGs) especially in the aspects of innovation, security, and digital infrastructure..

Goal 9: Industry, Innovation, and Infrastructure

This project will result in more secure and resilient digital infrastructure and adoption of secure and scalable technologies by the deployment and securing of a WordPress platform based in the cloud on AWS. It shows how cloud computing should be utilized in a responsible way and how this will help support secure digital services that are critical in education, healthcare, and other small business settings..

Goal 16: Peace, Justice, and Strong Institutions

The adoption of cyber-security controls that include vulnerability scanning, patch management, and multi-factor authentication implies a resolve to good digital institutions. By securing systems and data, the misuse of data is avoided, trust established in internet resources, and privacy and rights of the users ensured.

Goal 4: Quality Education

The project aligns with the goals of education as well, to build professional cybersecurity skills, to support the responsible utilisation of technology and recognise ethical implementation and monitoring of systems. These go in line with the SDG, which emphasizes on life-long, inclusive and equitable quality education in digital literacy and cybersecurity consciousness.

Guidance for Future Alignment

Future versions may perform energy-efficient set-ups, efficient presence on cloud resources (right-sized instances), and use tracking tools related to code sustainability based on open source, to monitor the environmental impact. The design and documentation of the platform could also include the education of users on the aspects of digital sustainability and responsible IT practices.

9. Individual Grading Score/Team Member Score

This project was a work done by a team of 3 members and they were Win Thu, Htet Eindra Wai, and Htet Htet Maung. The hands on deployment, testing, analysis and reporting phases were enhanced by the

active involvement of each team members. Firstly, our lab setup was done individually so that each of us experiences various issues and develops a better image of the environment.
Having tested the systems individually, we combined all results, talked about vulnerabilities, and identified the best methods of hardening. To coordinate fixes, to divide documentation tasks and guarantee their understandability in all configurations, online team meetings were organized.

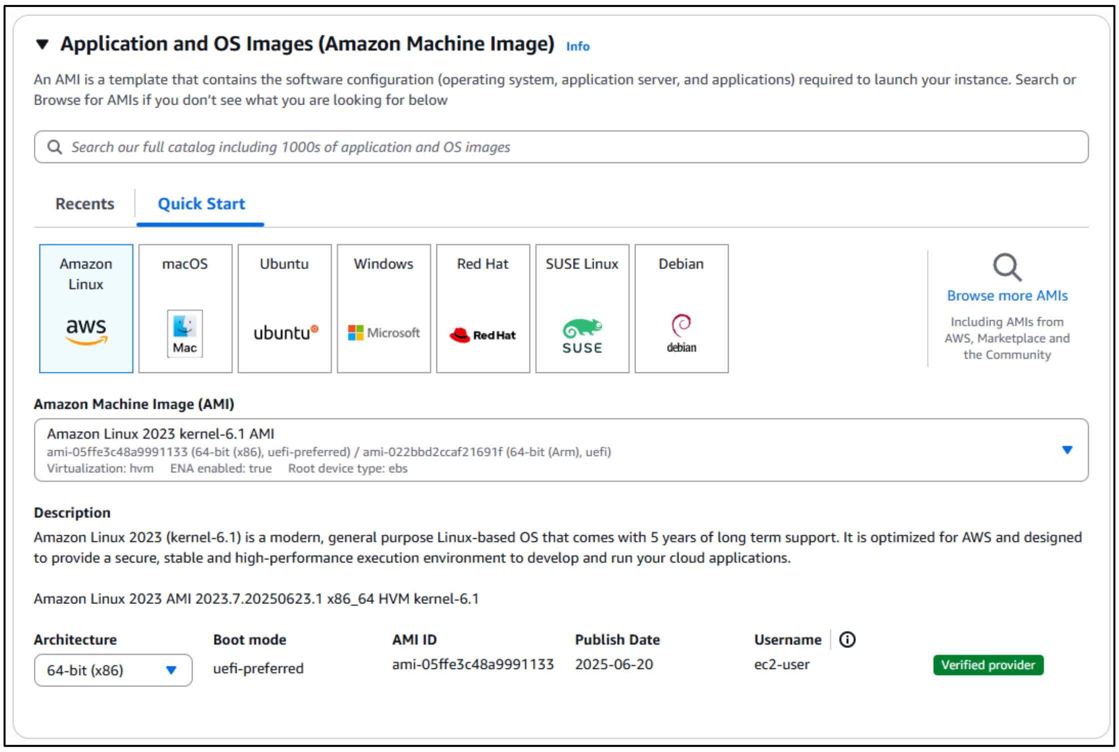
Notable contributions include:

- ❖ **Win Thu** led the report writing, coordinated documentation, and managed firewall configuration and WordPress hardening.
- ❖ **Htet Eindra Wai** identified the root cause of the WordPress instance failure after IP changes and proposed the **MySQL siteurl update fix**, which was a turning point in project stability. Check details script in Appendix 10.4.
- ❖ **Htet Htet Maung** focused on security testing with Nessus, Nikto, and WPScan, and helped document the vulnerabilities and corresponding risk ratings.

Final grading was decided mutually based on each member's effort and contribution. All members contributed equally to the analysis, solution development, and final report. The collaborative approach helped us learn from each other and produce a more resilient and secure system.

10.Appendix: Screenshots, configs, logs

10.1 Initial Setup



▼ Application and OS Images (Amazon Machine Image) Info

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Search our full catalog including 1000s of application and OS images

Amazon Machine Image (AMI)

Amazon Linux 2023 kernel-6.1 AMI
ami-05ffe3c48a9991133 (64-bit (x86), uefi-preferred) / ami-022bbd2ccaf21691f (64-bit (Arm), uefi)
Virtualization: hvm   ENA enabled: true   Root device type: ebs

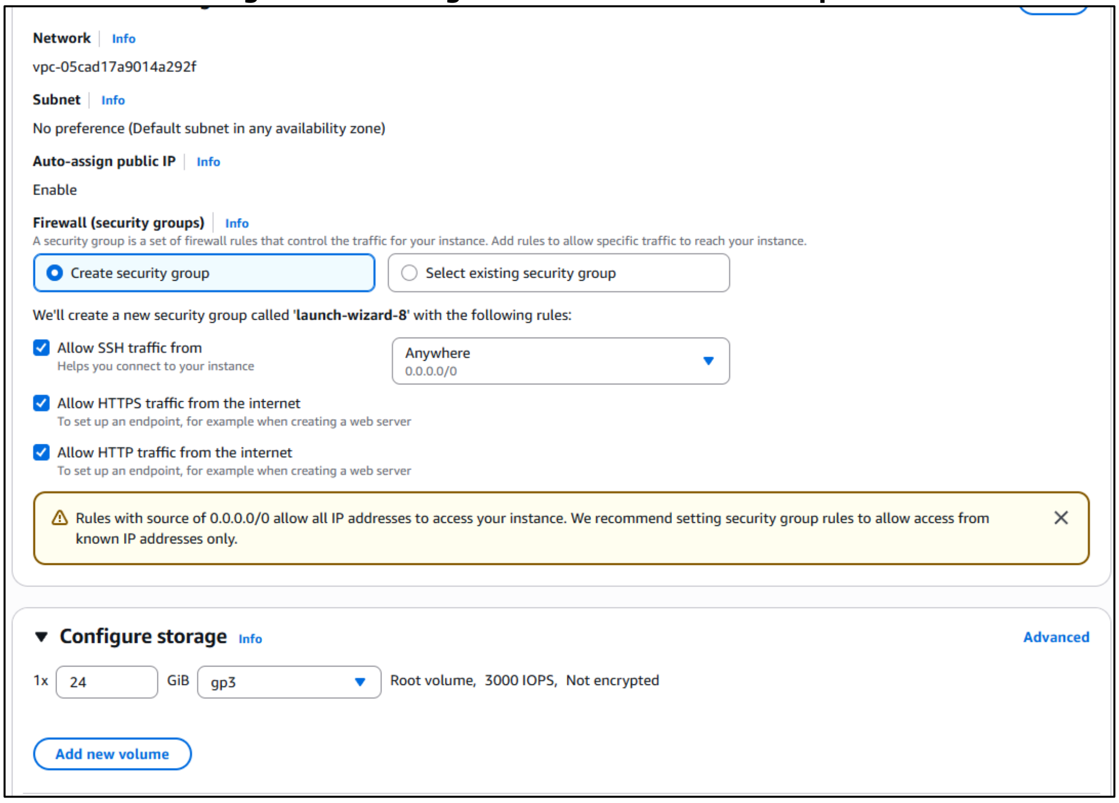
Description

Amazon Linux 2023 (kernel-6.1) is a modern, general purpose Linux-based OS that comes with 5 years of long term support. It is optimized for AWS and designed to provide a secure, stable and high-performance execution environment to develop and run your cloud applications.

Amazon Linux 2023 AMI 2023.7.20250623.1 x86_64 HVM kernel-6.1

Architecture	Boot mode	AMI ID	Publish Date	Username	
64-bit (x86)	uefi-preferred	ami-05ffe3c48a9991133	2025-06-20	ec2-user	Verified provider

Figure 1: Creating the EC2 instance for wordpress



Network Info
vpc-05cad17a9014a292f

Subnet Info
No preference (Default subnet in any availability zone)

Auto-assign public IP Info
Enable

Firewall (security groups) Info
A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

[✓ Create security group] Select existing security group

We'll create a new security group called '**launch-wizard-8**' with the following rules:

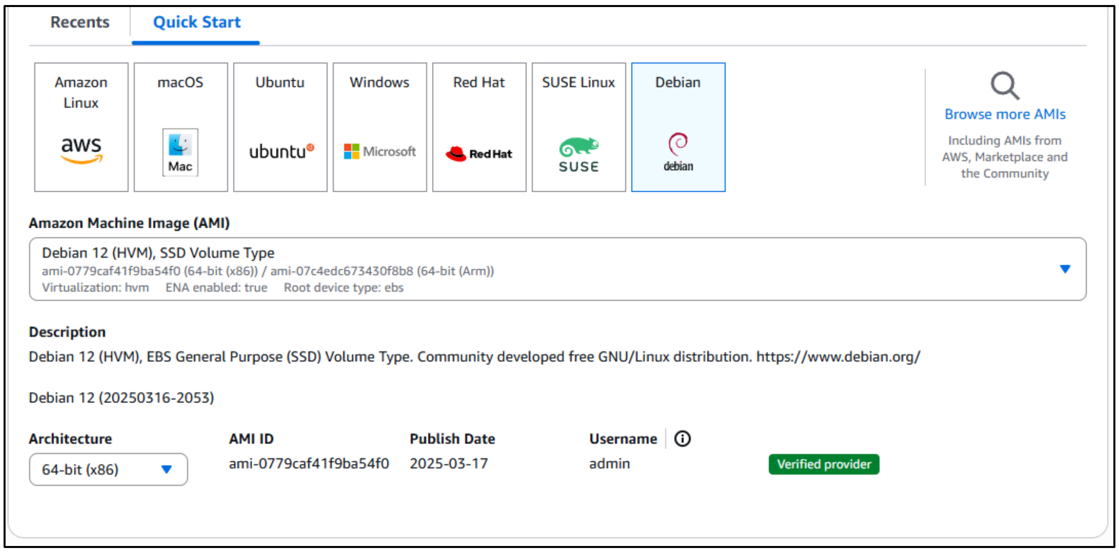
- [✓] Allow SSH traffic from — Anywhere 0.0.0.0/0
  Helps you connect to your instance
- [✓] Allow HTTPS traffic from the internet
  To set up an endpoint, for example when creating a web server
- [✓] Allow HTTP traffic from the internet
  To set up an endpoint, for example when creating a web server

⚠ Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

▼ **Configure storage** Info   Advanced

1x 24 GiB gp3   Root volume, 3000 IOPS, Not encrypted

Figure 2: Create Security Group and Configure storage



Amazon Machine Image (AMI)
Debian 12 (HVM), SSD Volume Type
ami-0779caf41f9ba54f0 (64-bit (x86)) / ami-07c4edc673430f8b8 (64-bit (Arm))
Virtualization: hvm ENA enabled: true Root device type: ebs

Description
Debian 12 (HVM), EBS General Purpose (SSD) Volume Type. Community developed free GNU/Linux distribution. https://www.debian.org/

Debian 12 (20250316-2053)

Architecture	AMI ID	Publish Date	Username	
64-bit (x86)	ami-0779caf41f9ba54f0	2025-03-17	admin	Verified provider

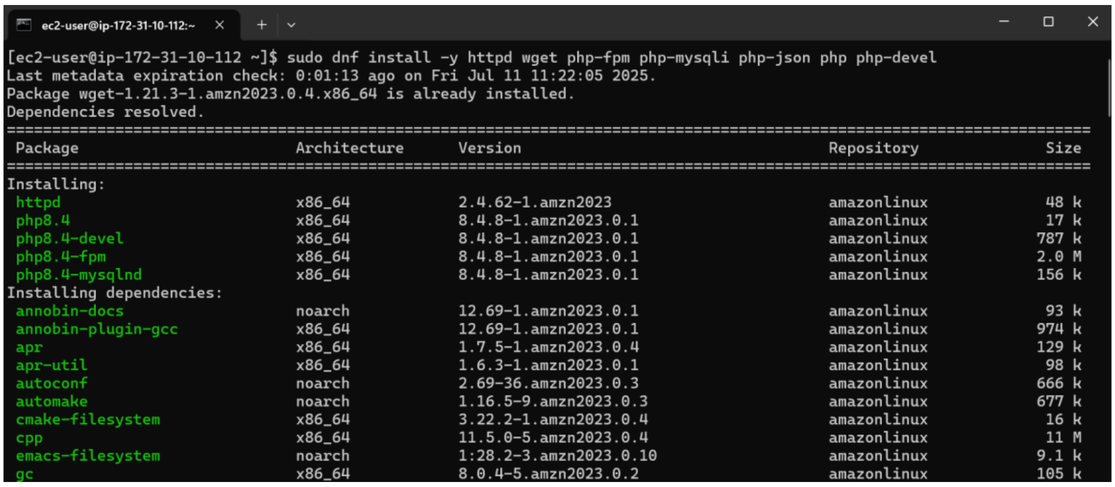
Figure 3: Create Attacker instance for testing

Next step is we have to login to instance using the following command

ssh -i Downloads\next.pem ec2-user@ec2-54-205-150-89.compute-1.amazonaws.com

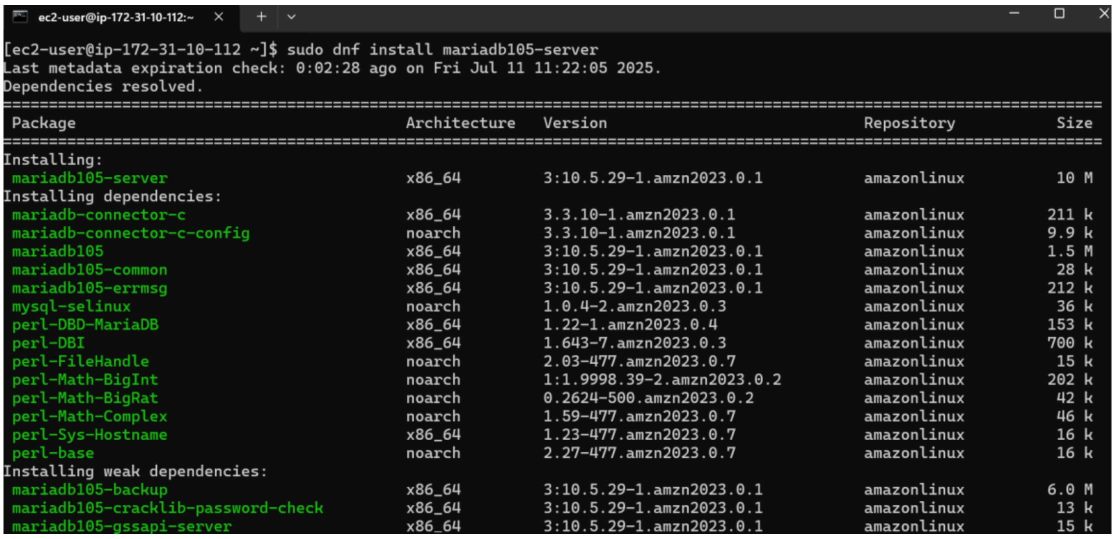
CREATE USER 'wtm'@'localhost' IDENTIFIED BY 'P@ssw0rd123!@#';

Installing LAMP Sever on EC2 instance



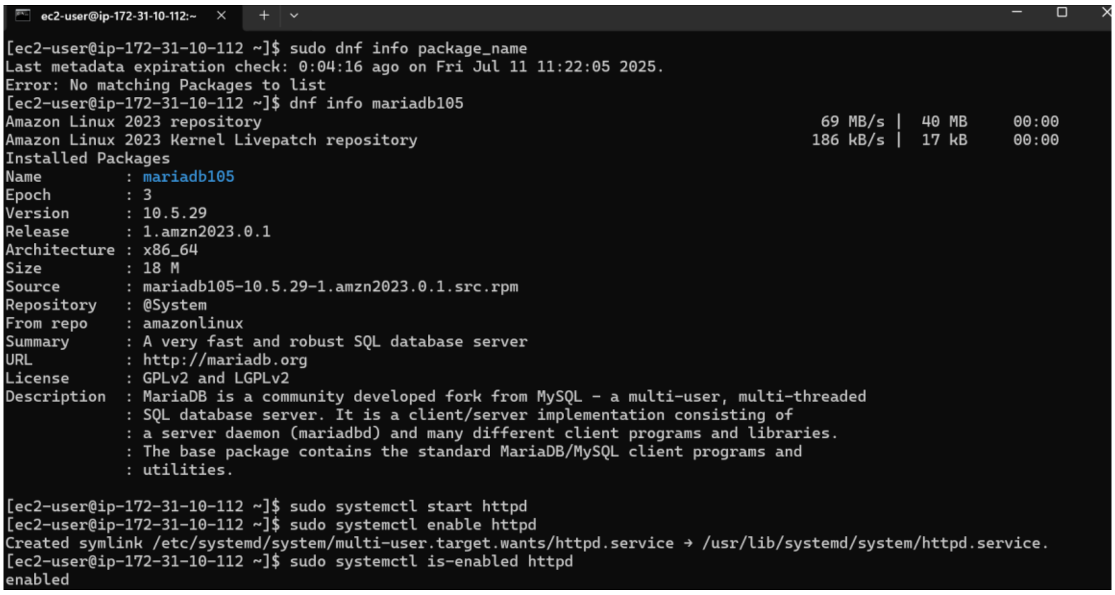
```
[ec2-user@ip-172-31-10-112 ~]$ sudo dnf install -y httpd wget php-fpm php-mysqli php-json php php-devel
Last metadata expiration check: 0:01:13 ago on Fri Jul 11 11:22:05 2025.
Package wget-1.21.3-1.amzn2023.0.4.x86_64 is already installed.
Dependencies resolved.
========================================================================================================================
 Package                        Architecture      Version                                  Repository             Size
========================================================================================================================
Installing:
 httpd                          x86_64            2.4.62-1.amzn2023                        amazonlinux            48 k
 php8.4                         x86_64            8.4.8-1.amzn2023.0.1                     amazonlinux            17 k
 php8.4-devel                   x86_64            8.4.8-1.amzn2023.0.1                     amazonlinux           787 k
 php8.4-fpm                     x86_64            8.4.8-1.amzn2023.0.1                     amazonlinux           2.0 M
 php8.4-mysqlnd                 x86_64            8.4.8-1.amzn2023.0.1                     amazonlinux           156 k
Installing dependencies:
 annobin-docs                   noarch            12.69-1.amzn2023.0.1                     amazonlinux            93 k
 annobin-plugin-gcc             x86_64            12.69-1.amzn2023.0.1                     amazonlinux           974 k
 apr                            x86_64            1.7.5-1.amzn2023.0.4                     amazonlinux           129 k
 apr-util                       x86_64            1.6.3-1.amzn2023.0.1                     amazonlinux            98 k
 autoconf                       noarch            2.69-36.amzn2023.0.3                     amazonlinux           666 k
 automake                       noarch            1.16.5-9.amzn2023.0.3                    amazonlinux           677 k
 cmake-filesystem               x86_64            3.22.2-1.amzn2023.0.4                    amazonlinux            16 k
 cpp                            x86_64            11.5.0-5.amzn2023.0.4                    amazonlinux            11 M
 emacs-filesystem               noarch            1:28.2-3.amzn2023.0.10                   amazonlinux           9.1 k
 gc                             x86_64            8.0.4-5.amzn2023.0.2                     amazonlinux           105 k
```

Figure 4: Installing the latest versions of Apache web server and PHP packages for AL2023.



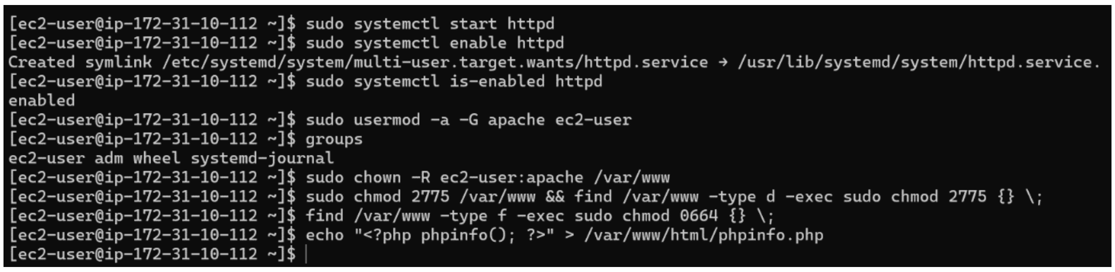
```
[ec2-user@ip-172-31-10-112 ~]$ sudo dnf install mariadb105-server
Last metadata expiration check: 0:02:28 ago on Fri Jul 11 11:22:05 2025.
Dependencies resolved.
================================================================================================================================
 Package                                       Architecture   Version                                Repository          Size
================================================================================================================================
Installing:
 mariadb105-server                             x86_64         3:10.5.29-1.amzn2023.0.1               amazonlinux         10 M
Installing dependencies:
 mariadb-connector-c                           x86_64         3.3.10-1.amzn2023.0.1                  amazonlinux        211 k
 mariadb-connector-c-config                    noarch         3.3.10-1.amzn2023.0.1                  amazonlinux        9.9 k
 mariadb105                                    x86_64         3:10.5.29-1.amzn2023.0.1               amazonlinux        1.5 M
 mariadb105-common                             x86_64         3:10.5.29-1.amzn2023.0.1               amazonlinux         28 k
 mariadb105-errmsg                             x86_64         3:10.5.29-1.amzn2023.0.1               amazonlinux        212 k
 mysql-selinux                                 noarch         1.0.4-2.amzn2023.0.3                   amazonlinux         36 k
 perl-DBD-MariaDB                              x86_64         1.22-1.amzn2023.0.4                    amazonlinux        153 k
 perl-DBI                                      x86_64         1.643-7.amzn2023.0.3                   amazonlinux        700 k
 perl-FileHandle                               noarch         2.03-477.amzn2023.0.7                  amazonlinux         15 k
 perl-Math-BigInt                              noarch         1:1.9998.39-2.amzn2023.0.2             amazonlinux        202 k
 perl-Math-BigRat                              noarch         0.2624-500.amzn2023.0.2                amazonlinux         42 k
 perl-Math-Complex                             noarch         1.59-477.amzn2023.0.7                  amazonlinux         46 k
 perl-Sys-Hostname                             x86_64         1.23-477.amzn2023.0.7                  amazonlinux         16 k
 perl-base                                     noarch         2.27-477.amzn2023.0.7                  amazonlinux         16 k
Installing weak dependencies:
 mariadb105-backup                             x86_64         3:10.5.29-1.amzn2023.0.1               amazonlinux        6.0 M
 mariadb105-cracklib-password-check            x86_64         3:10.5.29-1.amzn2023.0.1               amazonlinux         13 k
 mariadb105-gssapi-server                      x86_64         3:10.5.29-1.amzn2023.0.1               amazonlinux         15 k
```

Figure 5: Installing the mariadb databse server



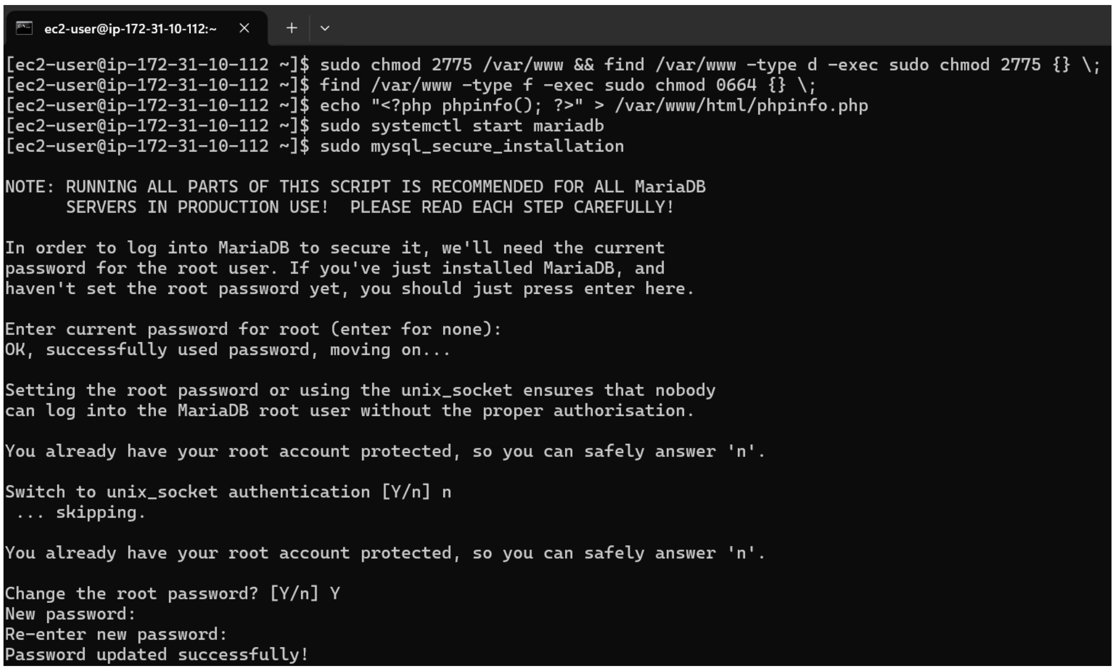
```
[ec2-user@ip-172-31-10-112 ~]$ sudo dnf info package_name
Last metadata expiration check: 0:04:16 ago on Fri Jul 11 11:22:05 2025.
Error: No matching Packages to list
[ec2-user@ip-172-31-10-112 ~]$ dnf info mariadb105
Amazon Linux 2023 repository                                                         69 MB/s |  40 MB     00:00
Amazon Linux 2023 Kernel Livepatch repository                                       186 kB/s |  17 kB     00:00
Installed Packages
Name         : mariadb105
Epoch        : 3
Version      : 10.5.29
Release      : 1.amzn2023.0.1
Architecture : x86_64
Size         : 18 M
Source       : mariadb105-10.5.29-1.amzn2023.0.1.src.rpm
Repository   : @System
From repo    : amazonlinux
Summary      : A very fast and robust SQL database server
URL          : http://mariadb.org
License      : GPLv2 and LGPLv2
Description  : MariaDB is a community developed fork from MySQL - a multi-user, multi-threaded
             : SQL database server. It is a client/server implementation consisting of
             : a server daemon (mariadbd) and many different client programs and libraries.
             : The base package contains the standard MariaDB/MySQL client programs and
             : utilities.

[ec2-user@ip-172-31-10-112 ~]$ sudo systemctl start httpd
[ec2-user@ip-172-31-10-112 ~]$ sudo systemctl enable httpd
Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service → /usr/lib/systemd/system/httpd.service.
[ec2-user@ip-172-31-10-112 ~]$ sudo systemctl is-enabled httpd
enabled
```

Figure 6: Checking the current version of the package name



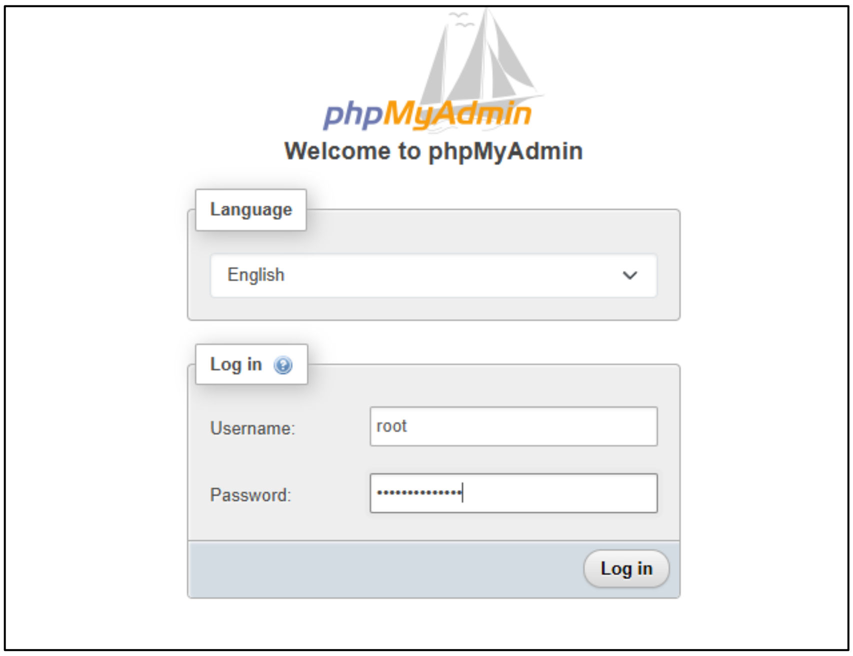
```
[ec2-user@ip-172-31-10-112 ~]$ sudo systemctl start httpd
[ec2-user@ip-172-31-10-112 ~]$ sudo systemctl enable httpd
Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service → /usr/lib/systemd/system/httpd.service.
[ec2-user@ip-172-31-10-112 ~]$ sudo systemctl is-enabled httpd
enabled
[ec2-user@ip-172-31-10-112 ~]$ sudo usermod -a -G apache ec2-user
[ec2-user@ip-172-31-10-112 ~]$ groups
ec2-user adm wheel systemd-journal
[ec2-user@ip-172-31-10-112 ~]$ sudo chown -R ec2-user:apache /var/www
[ec2-user@ip-172-31-10-112 ~]$ sudo chmod 2775 /var/www && find /var/www -type d -exec sudo chmod 2775 {} \;
[ec2-user@ip-172-31-10-112 ~]$ find /var/www -type f -exec sudo chmod 0664 {} \;
[ec2-user@ip-172-31-10-112 ~]$ echo "<?php phpinfo(); ?>" > /var/www/html/phpinfo.php
[ec2-user@ip-172-31-10-112 ~]$ |
```

Figure 7: Start the Apache web server



```
[ec2-user@ip-172-31-10-112 ~]$ sudo chmod 2775 /var/www && find /var/www -type d -exec sudo chmod 2775 {} \;
[ec2-user@ip-172-31-10-112 ~]$ find /var/www -type f -exec sudo chmod 0664 {} \;
[ec2-user@ip-172-31-10-112 ~]$ echo "<?php phpinfo(); ?>" > /var/www/html/phpinfo.php
[ec2-user@ip-172-31-10-112 ~]$ sudo systemctl start mariadb
[ec2-user@ip-172-31-10-112 ~]$ sudo mysql_secure_installation

NOTE: RUNNING ALL PARTS OF THIS SCRIPT IS RECOMMENDED FOR ALL MariaDB
      SERVERS IN PRODUCTION USE!  PLEASE READ EACH STEP CAREFULLY!

In order to log into MariaDB to secure it, we'll need the current
password for the root user. If you've just installed MariaDB, and
haven't set the root password yet, you should just press enter here.

Enter current password for root (enter for none):
OK, successfully used password, moving on...

Setting the root password or using the unix_socket ensures that nobody
can log into the MariaDB root user without the proper authorisation.

You already have your root account protected, so you can safely answer 'n'.

Switch to unix_socket authentication [Y/n] n
 ... skipping.

You already have your root account protected, so you can safely answer 'n'.

Change the root password? [Y/n] Y
New password:
Re-enter new password:
Password updated successfully!
```

Figure 8: Giving the permissions

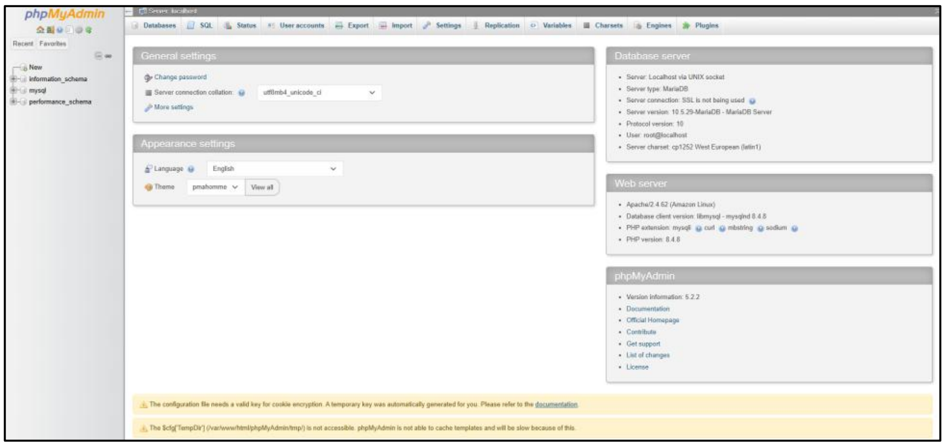


Language: English

Username: root

Password: ••••••••••••••

Figure 9: phpMyAdmin



Server connection collation: utf8mb4_unicode_ci

Language: English

Theme: pmahomme

Database server
- Server: Localhost via UNIX socket
- Server type: MariaDB
- Server connection: SSL is not being used
- Server version: 10.5.29-MariaDB - MariaDB Server
- Protocol version: 10
- User: root@localhost
- Server charset: cp1252 West European (latin1)

Web server
- Apache/2.4.62 (Amazon Linux)
- Database client version: libmysql - mysqlnd 8.4.8
- PHP extension: mysqli curl mbstring sodium
- PHP version: 8.4.8

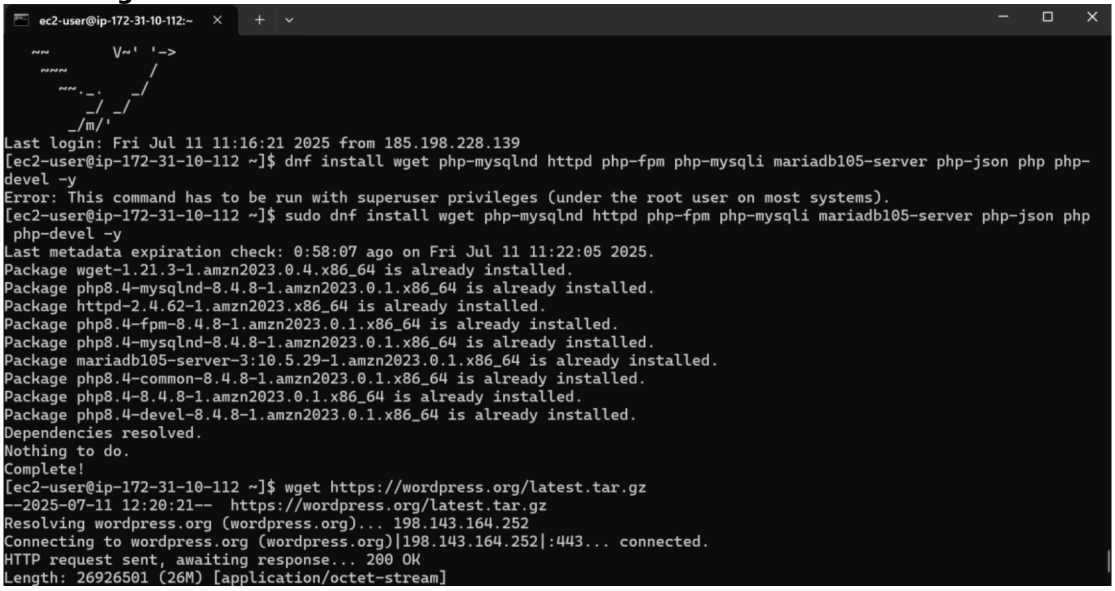
phpMyAdmin
- Version information: 5.2.2
- Documentation
- Official Homepage
- Contribute
- Get support
- List of changes
- License

The configuration file needs a valid key for cookie encryption. A temporary key was automatically generated for you. Please refer to the documentation.

The $cfg['TempDir'] (/var/www/html/phpMyAdmin/tmp/) is not accessible. phpMyAdmin is not able to cache templates and will be slow because of this.

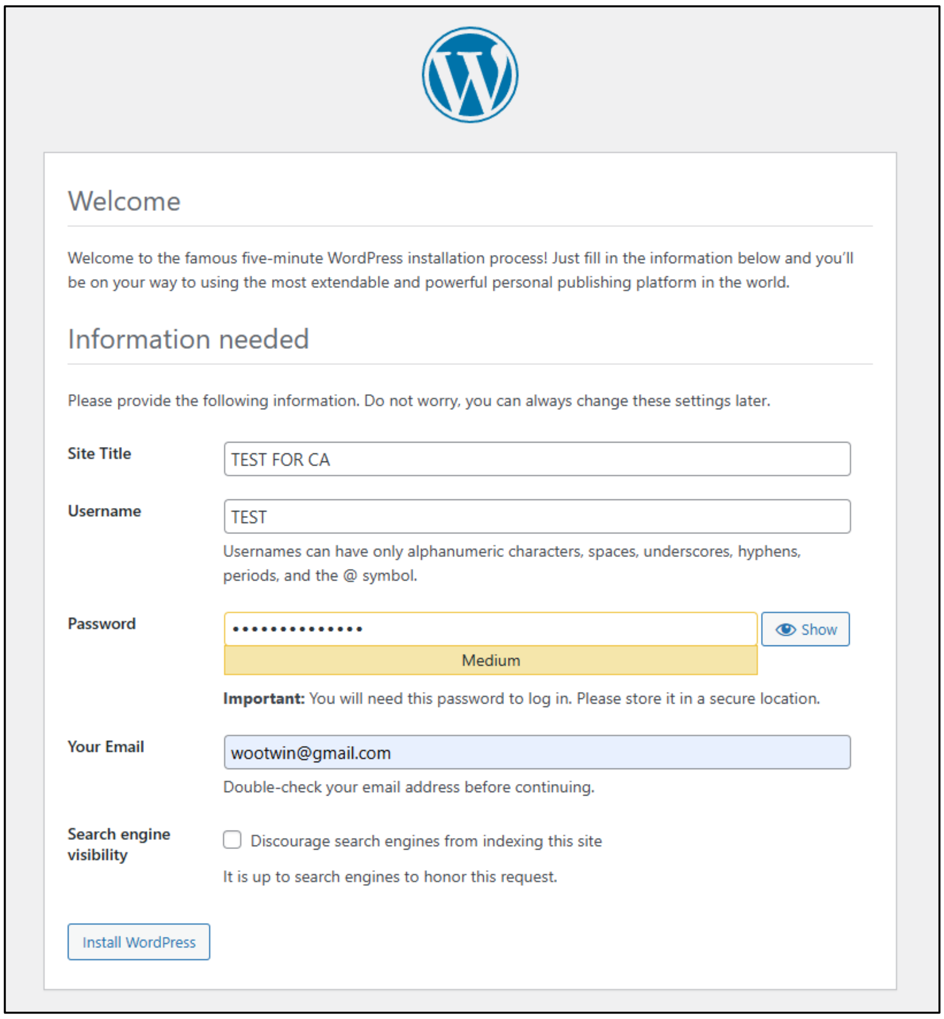
Figure 10: phpMyAdmin Dashboard

Installing WordPress



```
  ~~       V~' '->
   ~~~         /
     ~~._.   _/
        _/ _/
      _/m/'
Last login: Fri Jul 11 11:16:21 2025 from 185.198.228.139
[ec2-user@ip-172-31-10-112 ~]$ dnf install wget php-mysqlnd httpd php-fpm php-mysqli mariadb105-server php-json php php-
devel -y
Error: This command has to be run with superuser privileges (under the root user on most systems).
[ec2-user@ip-172-31-10-112 ~]$ sudo dnf install wget php-mysqlnd httpd php-fpm php-mysqli mariadb105-server php-json php
 php-devel -y
Last metadata expiration check: 0:58:07 ago on Fri Jul 11 11:22:05 2025.
Package wget-1.21.3-1.amzn2023.0.4.x86_64 is already installed.
Package php8.4-mysqlnd-8.4.8-1.amzn2023.0.1.x86_64 is already installed.
Package httpd-2.4.62-1.amzn2023.x86_64 is already installed.
Package php8.4-fpm-8.4.8-1.amzn2023.0.1.x86_64 is already installed.
Package php8.4-mysqlnd-8.4.8-1.amzn2023.0.1.x86_64 is already installed.
Package mariadb105-server-3:10.5.29-1.amzn2023.0.1.x86_64 is already installed.
Package php8.4-common-8.4.8-1.amzn2023.0.1.x86_64 is already installed.
Package php8.4-8.4.8-1.amzn2023.0.1.x86_64 is already installed.
Package php8.4-devel-8.4.8-1.amzn2023.0.1.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[ec2-user@ip-172-31-10-112 ~]$ wget https://wordpress.org/latest.tar.gz
--2025-07-11 12:20:21--  https://wordpress.org/latest.tar.gz
Resolving wordpress.org (wordpress.org)... 198.143.164.252
Connecting to wordpress.org (wordpress.org)|198.143.164.252|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 26926501 (26M) [application/octet-stream]
```

Figure 11: Installing WordPress



Welcome

Welcome to the famous five-minute WordPress installation process! Just fill in the information below and you'll be on your way to using the most extendable and powerful personal publishing platform in the world.

Information needed

Please provide the following information. Do not worry, you can always change these settings later.

Site Title	TEST FOR CA
Username	TEST Usernames can have only alphanumeric characters, spaces, underscores, hyphens, periods, and the @ symbol.
Password	•••••••••••••• Show Medium **Important:** You will need this password to log in. Please store it in a secure location.
Your Email	wootwin@gmail.com Double-check your email address before continuing.
Search engine visibility	☐ Discourage search engines from indexing this site It is up to search engines to honor this request.

Figure 12: WordPress installation

10.1.1 CloudWatch Alarm Create

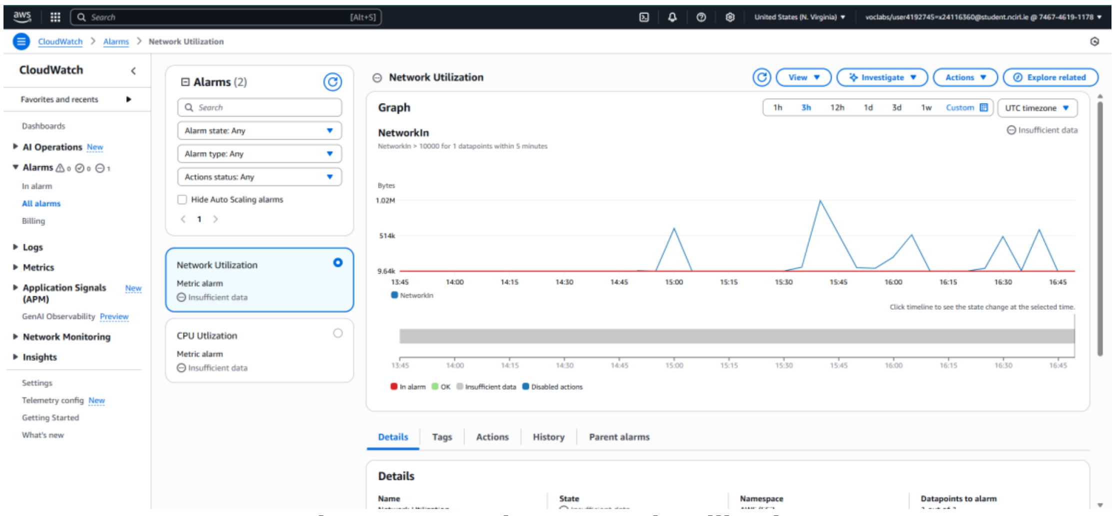
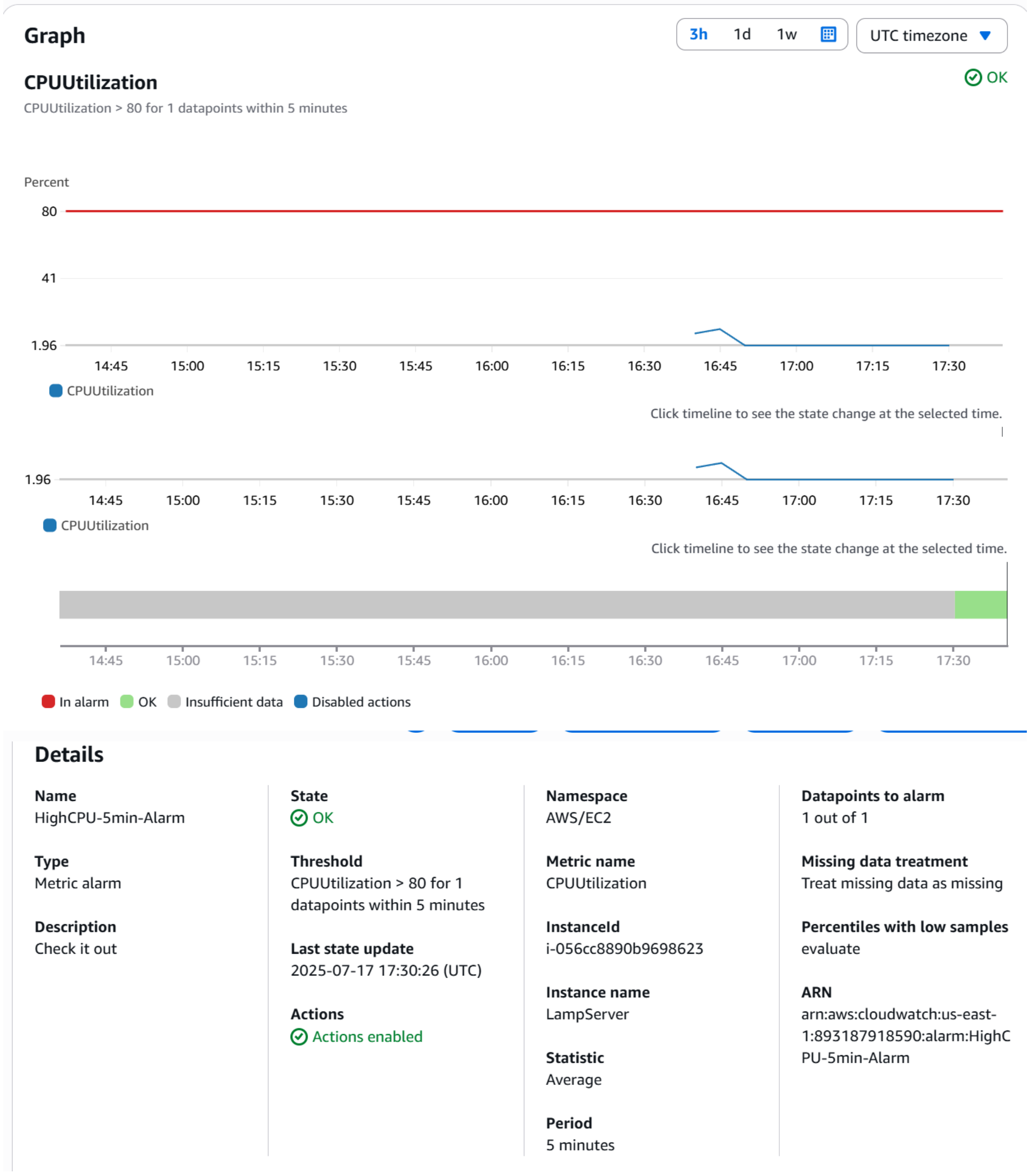


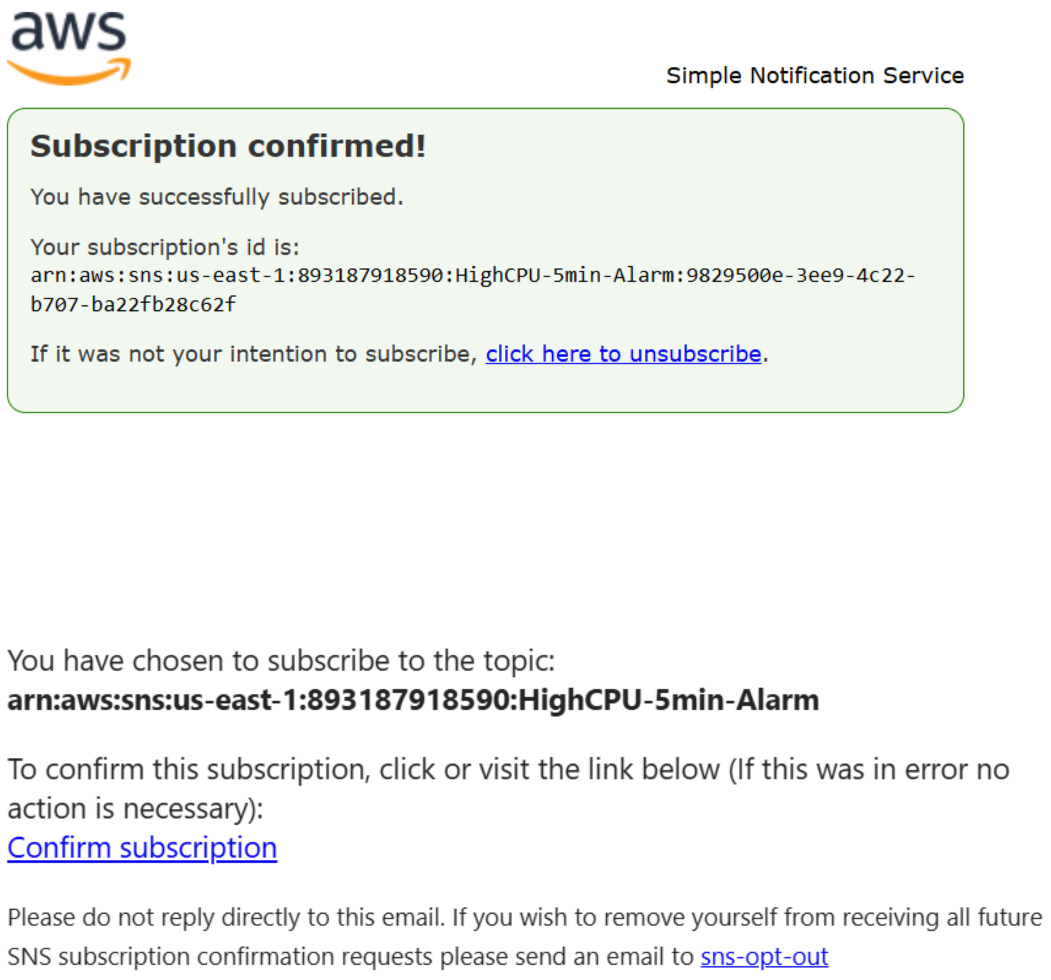
Figure 13: Setting Network Utilization



Details

Name HighCPU-5min-Alarm	**State** ✓ OK	**Namespace** AWS/EC2	**Datapoints to alarm** 1 out of 1
Type Metric alarm	**Threshold** CPUUtilization > 80 for 1 datapoints within 5 minutes	**Metric name** CPUUtilization	**Missing data treatment** Treat missing data as missing
Description Check it out	**Last state update** 2025-07-17 17:30:26 (UTC)	**InstanceId** i-056cc8890b9698623	**Percentiles with low samples** evaluate
	Actions ✓ Actions enabled	**Instance name** LampServer	**ARN** arn:aws:cloudwatch:us-east-1:893187918590:alarm:HighCPU-5min-Alarm
		Statistic Average	
		Period 5 minutes	

Figure 14: Setting 5 min alarm for 80% above CPU utilization



Subscription confirmed!

You have successfully subscribed.

Your subscription's id is:
`arn:aws:sns:us-east-1:893187918590:HighCPU-5min-Alarm:9829500e-3ee9-4c22-b707-ba22fb28c62f`

If it was not your intention to subscribe, click here to unsubscribe.

You have chosen to subscribe to the topic:
arn:aws:sns:us-east-1:893187918590:HighCPU-5min-Alarm

To confirm this subscription, click or visit the link below (If this was in error no action is necessary):
Confirm subscription

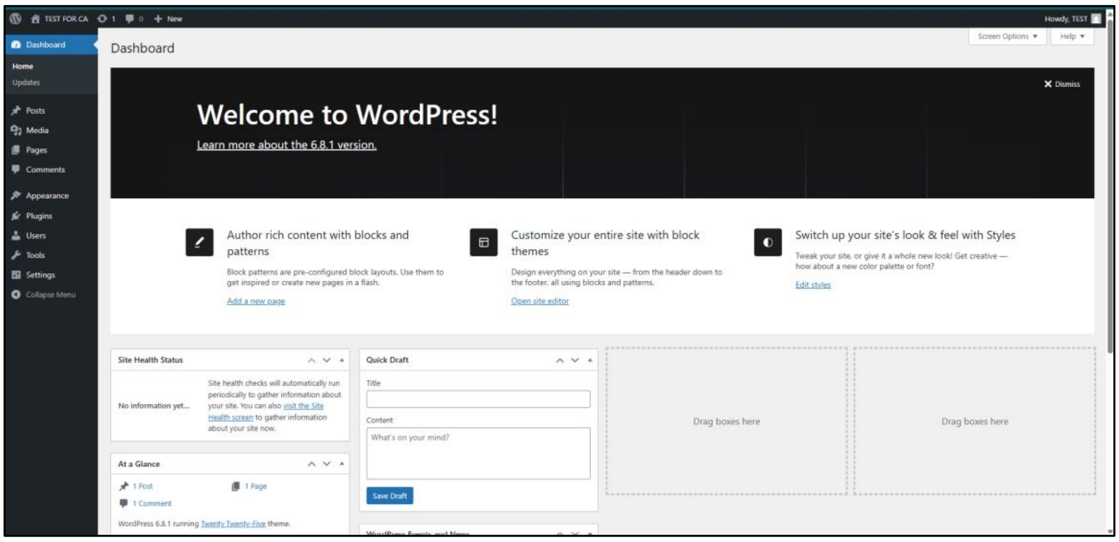
Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to sns-opt-out

Figure 15: Email Notification sent for subscription

10.2 WordPress Before Hardening

http://3.83.167.114/wp-login.php?redirect_to=http%3A%2F%2F3.83.167.114%2Fwp-admin%2F&reauth=1

http://3.83.167.114/wp-admin/



Dashboard

Welcome to WordPress!
Learn more about the 6.8.1 version.

Author rich content with blocks and patterns
Block patterns are pre-configured block layouts. Use them to get inspired or create new pages in a flash.

Customize your entire site with block themes
Design everything on your site — from the header down to the footer, all using blocks and patterns.

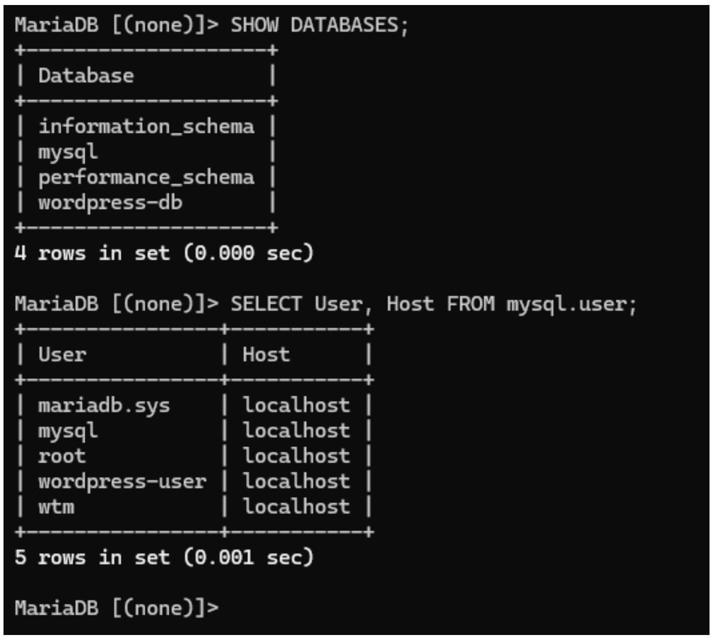
Switch up your site's look & feel with Styles
Tweak your site, or give it a whole new look! Get creative — how about a new color palette or font?

Site Health Status
No information yet...
Site health checks will automatically run periodically to gather information about your site. You can also visit the Site Health screen to gather information about your site now.

Quick Draft
Title
Content
What's on your mind?

At a Glance
1 Post
1 Page
1 Comment
WordPress 6.8.1 running Twenty Twenty-Five theme.

Figure 16: WordPress Dashboard

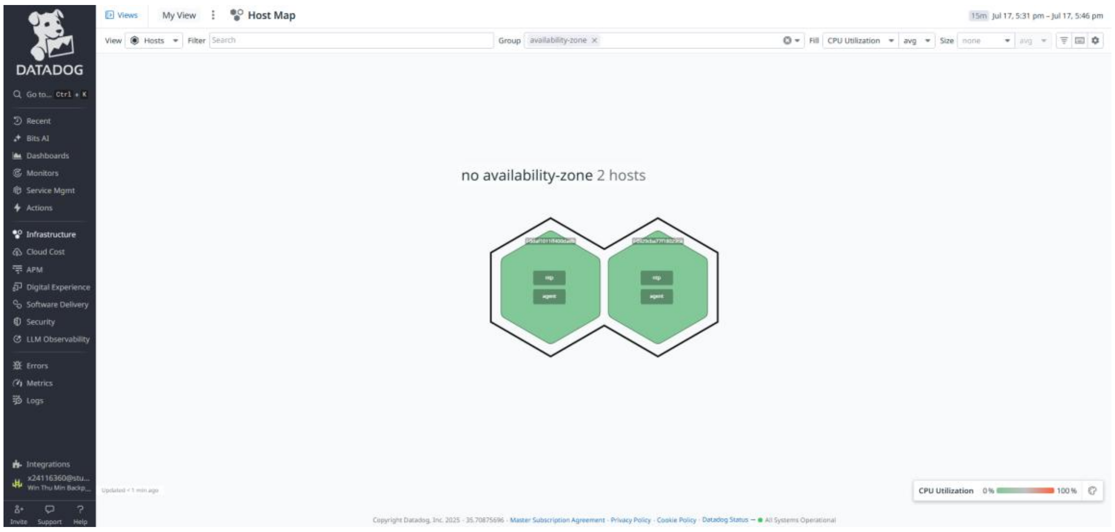


```
MariaDB [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| wordpress-db       |
+--------------------+
4 rows in set (0.000 sec)

MariaDB [(none)]> SELECT User, Host FROM mysql.user;
+----------------+-----------+
| User           | Host      |
+----------------+-----------+
| mariadb.sys    | localhost |
| mysql          | localhost |
| root           | localhost |
| wordpress-user | localhost |
| wtm            | localhost |
+----------------+-----------+
5 rows in set (0.001 sec)

MariaDB [(none)]>
```

Figure 17: Existing databases and username

10.3 Install Datadog and connect via following command

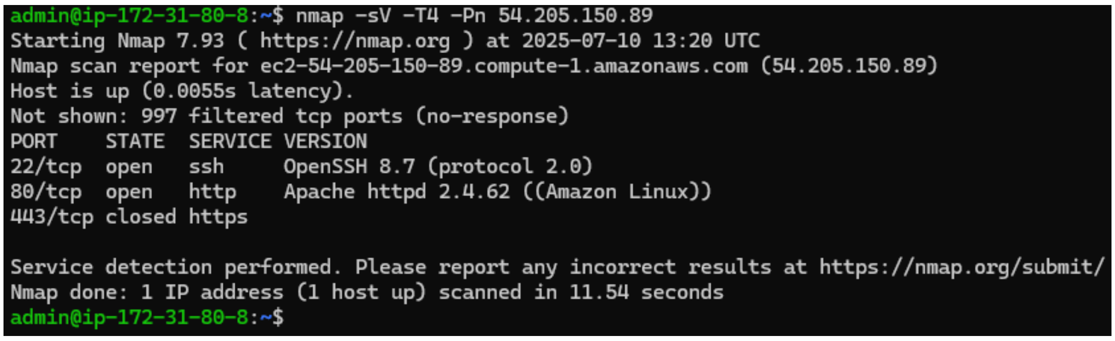
DD_AGENT_MAJOR_VERSION=7 DD_API_KEY=65e5da7b1be78be6f6f8cfbbf8c79585 DD_SITE="us5.datadoghq.com" bash -c "$(curl -L https://s3.amazonaws.com/dd-agent/scripts/install_script.sh)"



10.4 Run scanning tools to test the vulnerabilities

10.4.1 Nmap scan

sudo apt update && sudo apt install nmap -y
nmap -sV -T4 -Pn 54.205.150.89

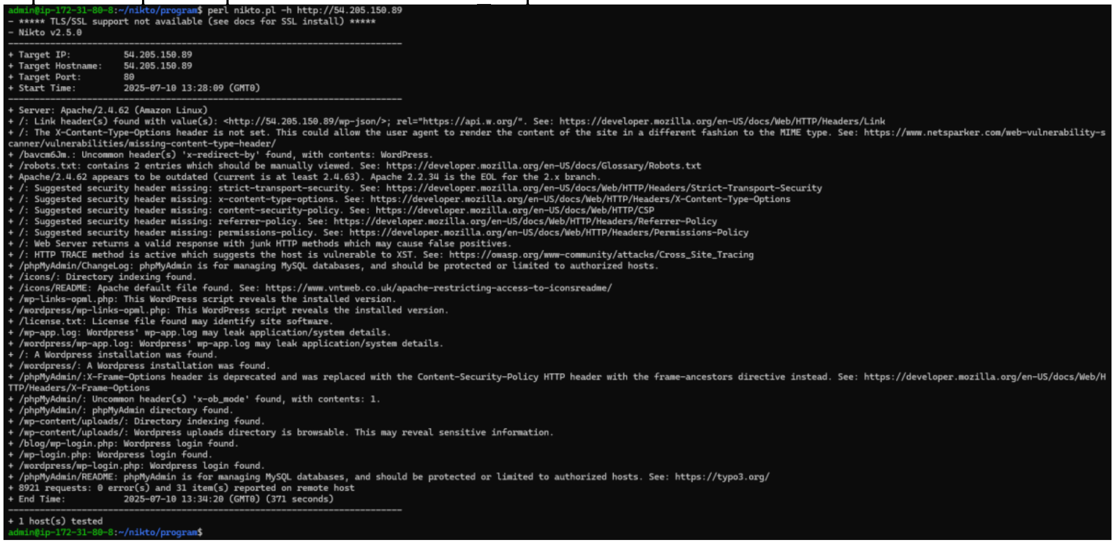


```
admin@ip-172-31-80-8:~$ nmap -sV -T4 -Pn 54.205.150.89
Starting Nmap 7.93 ( https://nmap.org ) at 2025-07-10 13:20 UTC
Nmap scan report for ec2-54-205-150-89.compute-1.amazonaws.com (54.205.150.89)
Host is up (0.0055s latency).
Not shown: 997 filtered tcp ports (no-response)
PORT    STATE  SERVICE VERSION
22/tcp  open   ssh     OpenSSH 8.7 (protocol 2.0)
80/tcp  open   http    Apache httpd 2.4.62 ((Amazon Linux))
443/tcp closed https

Service detection performed. Please report any incorrect results at https://nmap.org/submit/
Nmap done: 1 IP address (1 host up) scanned in 11.54 seconds
admin@ip-172-31-80-8:~$
```

Figure 18: Nmap scan for WordPress site

10.4.2 Nikto (Web Server Vulnerability Scanner)

Scanning with Nikto Step by Step
1. sudo apt update
2. sudo apt install git perl libnet-ssleay-perl openssl libwhisker2-perl -y
3. git clone https://github.com/sullo/nikto.git
4. cd nikto/program
5. perl nikto.pl -h http://3.86.83.103
6. perl nikto.pl -h http://3.86.83.103 -o nikto_output.txt



```
admin@ip-172-31-80-8:~/nikto/program$ perl nikto.pl -h http://54.205.150.89
- ***** TLS/SSL support not available (see docs for SSL install) *****
- Nikto v2.5.0
---------------------------------------------------------------------------
+ Target IP:          54.205.150.89
+ Target Hostname:    54.205.150.89
+ Target Port:        80
+ Start Time:         2025-07-10 13:28:09 (GMT0)
---------------------------------------------------------------------------
+ Server: Apache/2.4.62 (Amazon Linux)
+ /: Link header(s) found with value(s): <http://54.205.150.89/wp-json/>; rel="https://api.w.org/". See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Link
+ /: The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type. See: https://www.netsparker.com/web-vulnerability-s
canner/vulnerabilities/missing-content-type-header/
+ /bavcm6Jm.: Uncommon header(s) 'x-redirect-by' found, with contents: WordPress.
+ /robots.txt: contains 2 entries which should be manually viewed. See: https://developer.mozilla.org/en-US/docs/Glossary/Robots.txt
+ Apache/2.4.62 appears to be outdated (current is at least 2.4.63). Apache 2.2.34 is the EOL for the 2.x branch.
+ /: Suggested security header missing: strict-transport-security. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security
+ /: Suggested security header missing: x-content-type-options. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Content-Type-Options
+ /: Suggested security header missing: content-security-policy. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP
+ /: Suggested security header missing: referrer-policy. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referrer-Policy
+ /: Suggested security header missing: permissions-policy. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Permissions-Policy
+ /: Web Server returns a valid response with junk HTTP methods which may cause false positives.
+ /: HTTP TRACE method is active which suggests the host is vulnerable to XST. See: https://owasp.org/www-community/attacks/Cross_Site_Tracing
+ /phpMyAdmin/ChangeLog: phpMyAdmin is for managing MySQL databases, and should be protected or limited to authorized hosts.
+ /icons/: Directory indexing found.
+ /icons/README: Apache default file found. See: https://www.vntweb.co.uk/apache-restricting-access-to-iconsreadme/
+ /wp-links-opml.php: This WordPress script reveals the installed version.
+ /wordpress/wp-links-opml.php: This WordPress script reveals the installed version.
+ /license.txt: License file found may identify site software.
+ /wp-app.log: Wordpress' wp-app.log may leak application/system details.
+ /wordpress/wp-app.log: Wordpress' wp-app.log may leak application/system details.
+ /: A Wordpress installation was found.
+ /wordpress/: A Wordpress installation was found.
+ /phpMyAdmin/:X-Frame-Options header is deprecated and was replaced with the Content-Security-Policy HTTP header with the frame-ancestors directive instead. See: https://developer.mozilla.org/en-US/docs/Web/H
TTP/Headers/X-Frame-Options
+ /phpMyAdmin/: Uncommon header(s) 'x-ob_mode' found, with contents: 1.
+ /phpMyAdmin/: phpMyAdmin directory found.
+ /wp-content/uploads/: Directory indexing found.
+ /wp-content/uploads/: Wordpress uploads directory is browsable. This may reveal sensitive information.
+ /blog/wp-login.php: Wordpress login found.
+ /wp-login.php: Wordpress login found.
+ /wordpress/wp-login.php: Wordpress login found.
+ /phpMyAdmin/README: phpMyAdmin is for managing MySQL databases, and should be protected or limited to authorized hosts. See: https://typo3.org/
+ 8921 requests: 0 error(s) and 31 item(s) reported on remote host
+ End Time:           2025-07-10 13:34:20 (GMT0) (371 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
admin@ip-172-31-80-8:~/nikto/program$
```

Figure 19: Nikto Scan

10.4.3 WPScan

sudo apt install wpscan -y
admin@ip-172-31-80-8:~$ wpscan --url http://54.205.19.201

```
_______________________________________________________________
         __          _______   _____
         \ \        / /  __ \ / ____|
          \ \  /\  / /| |__) | (___   ___  __ _ _ __ ®
           \ \/  \/ / |  ___/ \___ \ / __|/ _` | '_ \
            \  /\  /  | |     ____) | (__| (_| | | | |
             \/  \/   |_|    |_____/ \___|\__,_|_| |_|

         WordPress Security Scanner by the WPScan Team
                         Version 3.8.28

       @_WPScan_, @ethicalhack3r, @erwan_lr, @firefart
_______________________________________________________________

[i] Updating the Database ...
[i] Update completed.

[+] URL: http://54.205.19.201/ [54.205.19.201]
[+] Started: Thu Jul 10 14:16:24 2025
```

```
Interesting Finding(s):

[+] Headers
 | Interesting Entry: Server: Apache/2.4.62 (Amazon Linux)
 | Found By: Headers (Passive Detection)
 | Confidence: 100%

[+] robots.txt found: http://54.205.19.201/robots.txt
 | Interesting Entries:
 |  - /wp-admin/
 |  - /wp-admin/admin-ajax.php
 | Found By: Robots Txt (Aggressive Detection)
 | Confidence: 100%

[+] XML-RPC seems to be enabled: http://54.205.19.201/xmlrpc.php
 | Found By: Direct Access (Aggressive Detection)
 | Confidence: 100%
 | References:
 |  - http://codex.wordpress.org/XML-RPC_Pingback_API
 |  -
https://www.rapid7.com/db/modules/auxiliary/scanner/http/wordpress_ghost_scanner/
 |  - https://www.rapid7.com/db/modules/auxiliary/dos/http/wordpress_xmlrpc_dos/
 |  - https://www.rapid7.com/db/modules/auxiliary/scanner/http/wordpress_xmlrpc_login/
 |  -
https://www.rapid7.com/db/modules/auxiliary/scanner/http/wordpress_pingback_access/

[+] WordPress readme found: http://54.205.19.201/readme.html
 | Found By: Direct Access (Aggressive Detection)
 | Confidence: 100%

[+] Upload directory has listing enabled: http://54.205.19.201/wp-content/uploads/
 | Found By: Direct Access (Aggressive Detection)
 | Confidence: 100%

[+] The external WP-Cron seems to be enabled: http://54.205.19.201/wp-cron.php
 | Found By: Direct Access (Aggressive Detection)
 | Confidence: 60%
 | References:
 |  - https://www.iplocation.net/defend-wordpress-from-ddos
 |  - https://github.com/wpscanteam/wpscan/issues/1299

[+] WordPress version 6.8.1 identified (Latest, released on 2025-04-30).
 | Found By: Emoji Settings (Passive Detection)
 |  - http://54.205.19.201/, Match: 'wp-includes\/js\/wp-emoji-release.min.js?ver=6.8.1'
 | Confirmed By: Meta Generator (Passive Detection)
 |  - http://54.205.19.201/, Match: 'WordPress 6.8.1'

[+] WordPress theme in use: twentytwentyfive
 | Location: http://54.205.19.201/wp-content/themes/twentytwentyfive/
 | Latest Version: 1.2 (up to date)
 | Last Updated: 2025-04-15T00:00:00.000Z
 | Readme: http://54.205.19.201/wp-content/themes/twentytwentyfive/readme.txt
 | [!] Directory listing is enabled
 | Style URL: http://54.205.19.201/wp-content/themes/twentytwentyfive/style.css
 | Style Name: Twenty Twenty-Five
 | Style URI: https://wordpress.org/themes/twentytwentyfive/
 | Description: Twenty Twenty-Five emphasizes simplicity and adaptability. It offers flexible
design options, suppor...
 | Author: the WordPress team
 | Author URI: https://wordpress.org
 |
 | Found By: Urls In Homepage (Passive Detection)
 | Confirmed By: Urls In 404 Page (Passive Detection)
 |
 | Version: 1.2 (80% confidence)
 | Found By: Style (Passive Detection)
 |  - http://54.205.19.201/wp-content/themes/twentytwentyfive/style.css, Match:
'Version: 1.2'

[+] Enumerating All Plugins (via Passive Methods)
[+] Checking Plugin Versions (via Passive and Aggressive Methods)

[i] Plugin(s) Identified:

[+] *
 | Location: http://54.205.19.201/wp-content/plugins/*/
 |
 | Found By: Urls In Homepage (Passive Detection)
 | Confirmed By: Urls In 404 Page (Passive Detection)
 |
 | The version could not be determined.

[+] Enumerating Config Backups (via Passive and Aggressive Methods)
 Checking Config Backups - Time: 00:00:00
<=============================================================
=============================================================
===================> (137 / 137) 100.00% Time: 00:00:00

[i] No Config Backups Found.

[!] No WPScan API Token given, as a result vulnerability data has not been output.
[!] You can get a free API token with 25 daily requests by registering at
https://wpscan.com/register

[+] Finished: Thu Jul 10 14:16:59 2025
[+] Requests Done: 190
[+] Cached Requests: 7
[+] Data Sent: 46.215 KB
[+] Data Received: 22.482 MB
[+] Memory used: 275.672 MB
[+] Elapsed time: 00:00:35
```

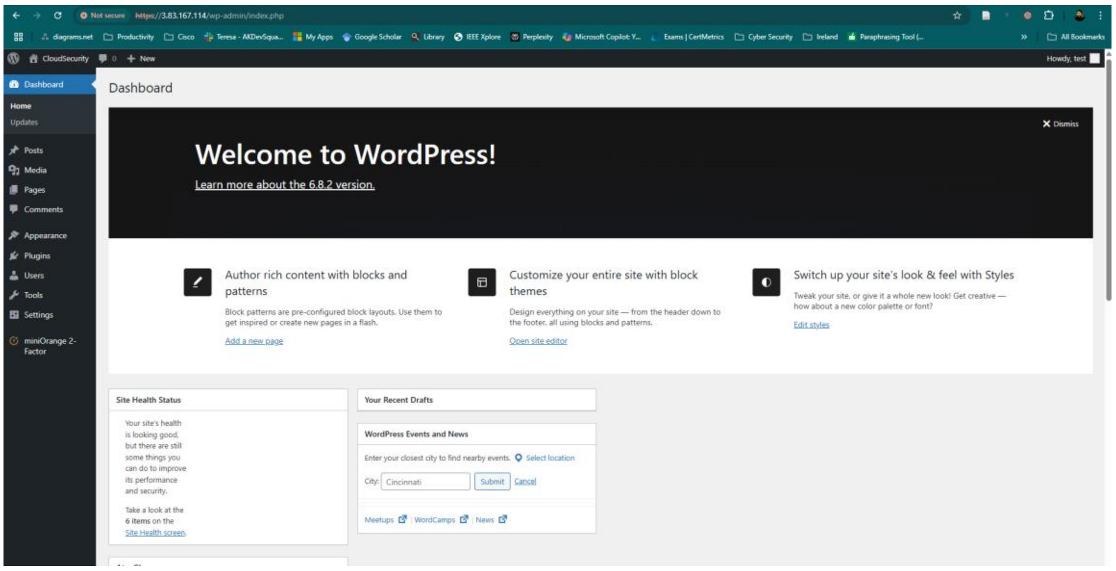
10.4.4 Nessus Scan

We also used Nessus scan to get the more information.
Run Nessus
upload to the instance

```
PS C:\Users\ThinkPad> scp -i Downloads\next.pem Downloads\Nessus-10.9.1-
debian10_amd64.deb admin@ec2-54-161-129-163.compute-1.amazonaws.com:~
sudo dpkg -i Nessus-10.9.1-debian10_amd64.deb
sudo apt-get install -f -y
sudo systemctl start nessusd
sudo systemctl enable nessusd
```

10.4.5 WordPress after hardening

https://3.83.167.114/wp-admin/index.php



Dashboard

Welcome to WordPress!

Learn more about the 6.8.2 version.

Author rich content with blocks and patterns

Block patterns are pre-configured block layouts. Use them to get inspired or create new pages in a flash.

Add a new page

Customize your entire site with block themes

Design everything on your site — from the header down to the footer, all using blocks and patterns.

Open site editor

Switch up your site's look & feel with Styles

Tweak your site, or give it a whole new look! Get creative — how about a new color palette or font?

Edit styles

Site Health Status

Your site's health is looking good, but there are still some things you can do to improve its performance and security.

Take a look at the **6 items** on the Site Health screen.

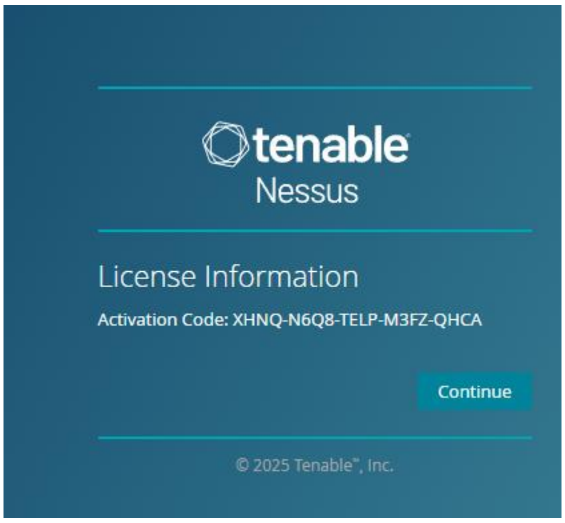
Your Recent Drafts

WordPress Events and News

Enter your closest city to find nearby events. Select location

City: Cincinnati Submit Cancel

Meetups | WordCamps | News

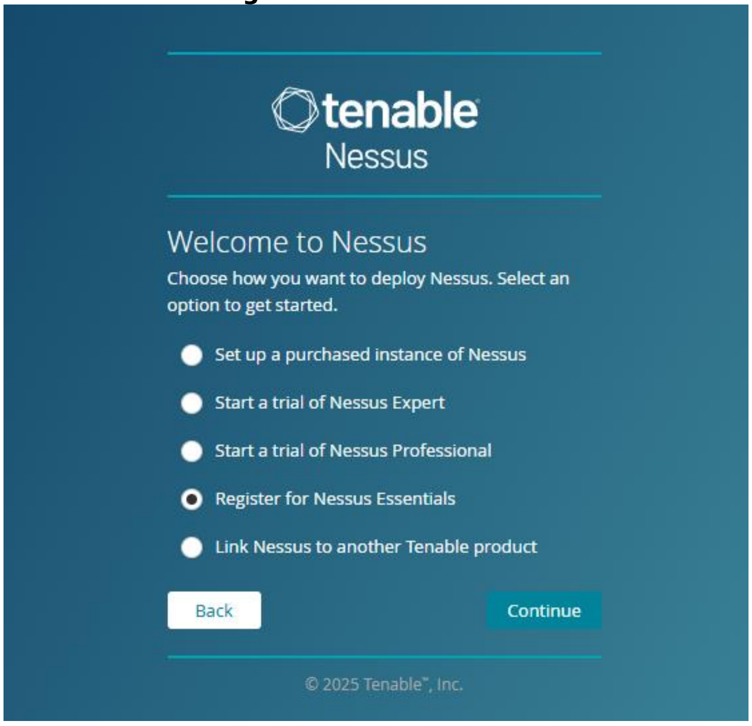


License Information

Activation Code: XHNQ-N6Q8-TELP-M3FZ-QHCA

© 2025 Tenable™, Inc.

Figure 20: Nessus Scan



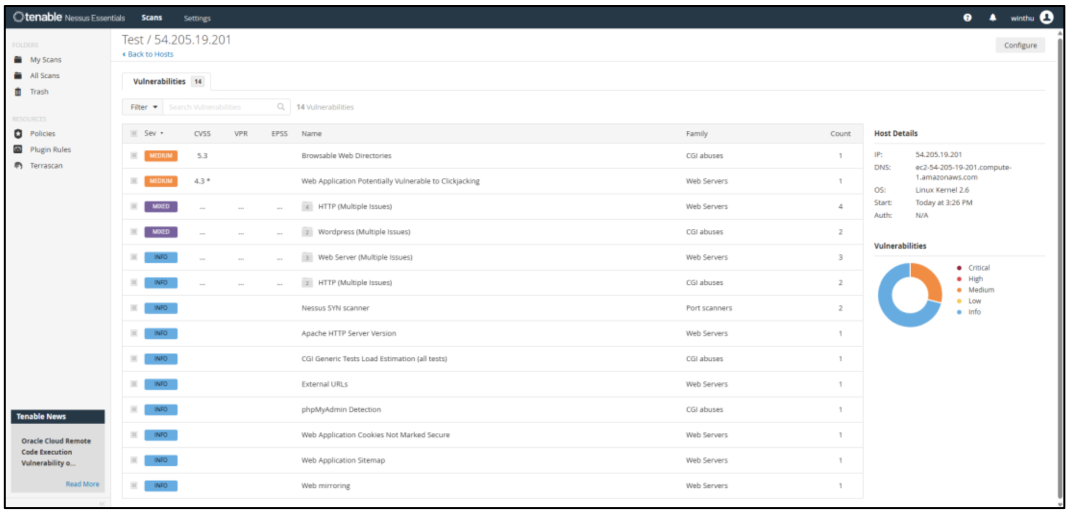
Welcome to Nessus

Choose how you want to deploy Nessus. Select an option to get started.

- Set up a purchased instance of Nessus
- Start a trial of Nessus Expert
- Start a trial of Nessus Professional
- [✓ Register for Nessus Essentials]
- Link Nessus to another Tenable product

© 2025 Tenable™, Inc.

Figure 21: Nessus Scan



Test / 54.205.19.201

14 Vulnerabilities

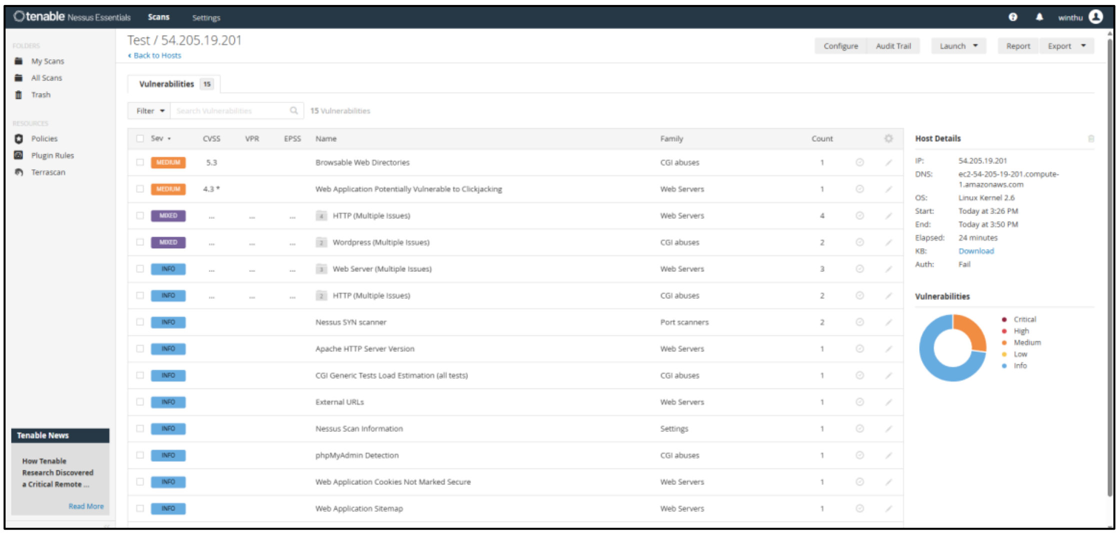
Sev	CVSS	VPR	EPSS	Name	Family	Count
MEDIUM	5.3			Browsable Web Directories	CGI abuses	1
MEDIUM	4.3 *			Web Application Potentially Vulnerable to Clickjacking	Web Servers	1
MIXED	...	...	...	4 HTTP (Multiple Issues)	Web Servers	4
MIXED	...	...	...	2 Wordpress (Multiple Issues)	CGI abuses	2
INFO	...	...	...	3 Web Server (Multiple Issues)	Web Servers	3
INFO	...	...	...	2 HTTP (Multiple Issues)	CGI abuses	2
INFO				Nessus SYN scanner	Port scanners	2
INFO				Apache HTTP Server Version	Web Servers	1
INFO				CGI Generic Tests Load Estimation (all tests)	CGI abuses	1
INFO				External URLs	Web Servers	1
INFO				phpMyAdmin Detection	CGI abuses	1
INFO				Web Application Cookies Not Marked Secure	Web Servers	1
INFO				Web Application Sitemap	Web Servers	1
INFO				Web mirroring	Web Servers	1

Host Details

IP:	54.205.19.201
DNS:	ec2-54-205-19-201.compute-1.amazonaws.com
OS:	Linux Kernel 2.6
Start:	Today at 3:26 PM
Auth:	N/A

Vulnerabilities

Figure 22: Nessus Scan result



Test / 54.205.19.201

15 Vulnerabilities

Sev	CVSS	VPR	EPSS	Name	Family	Count
MEDIUM	5.3			Browsable Web Directories	CGI abuses	1
MEDIUM	4.3 *			Web Application Potentially Vulnerable to Clickjacking	Web Servers	1
MIXED	...	...	...	4 HTTP (Multiple Issues)	Web Servers	4
MIXED	...	...	...	2 Wordpress (Multiple Issues)	CGI abuses	2
INFO	...	...	...	3 Web Server (Multiple Issues)	Web Servers	3
INFO	...	...	...	2 HTTP (Multiple Issues)	CGI abuses	2
INFO				Nessus SYN scanner	Port scanners	2
INFO				Apache HTTP Server Version	Web Servers	1
INFO				CGI Generic Tests Load Estimation (all tests)	CGI abuses	1
INFO				External URLs	Web Servers	1
INFO				Nessus Scan Information	Settings	1
INFO				phpMyAdmin Detection	CGI abuses	1
INFO				Web Application Cookies Not Marked Secure	Web Servers	1
INFO				Web Application Sitemap	Web Servers	1

Host Details

IP:	54.205.19.201
DNS:	ec2-54-205-19-201.compute-1.amazonaws.com
OS:	Linux Kernel 2.6
Start:	Today at 3:26 PM
End:	Today at 3:50 PM
Elapsed:	24 minutes
KB:	Download
Auth:	Fail

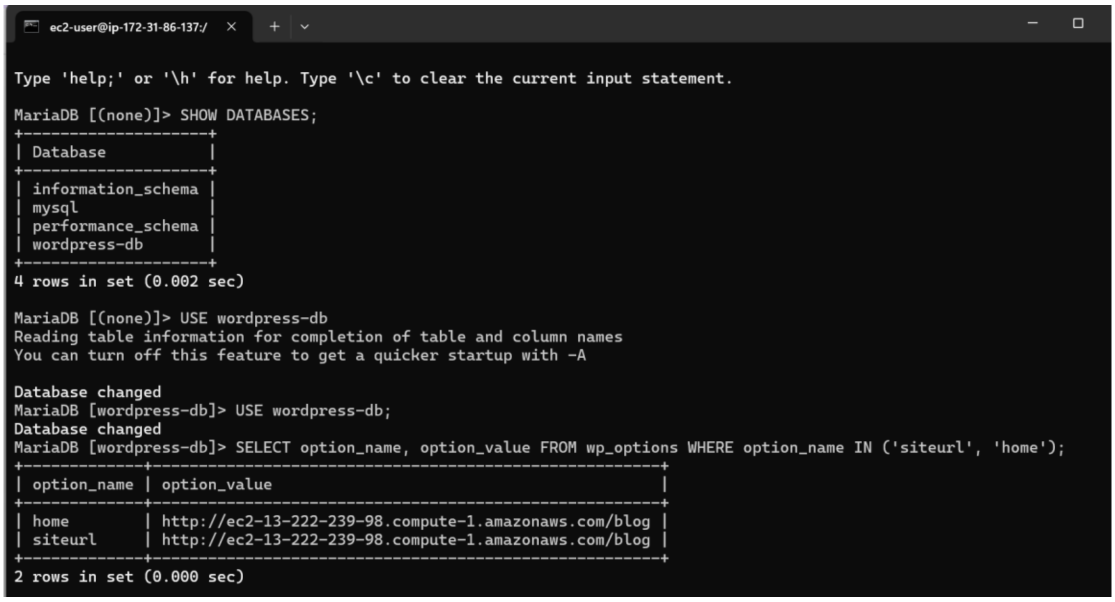
Vulnerabilities

Figure 23: Nessus Scan result

10.5 Solution for changing public DNS and blog is broken

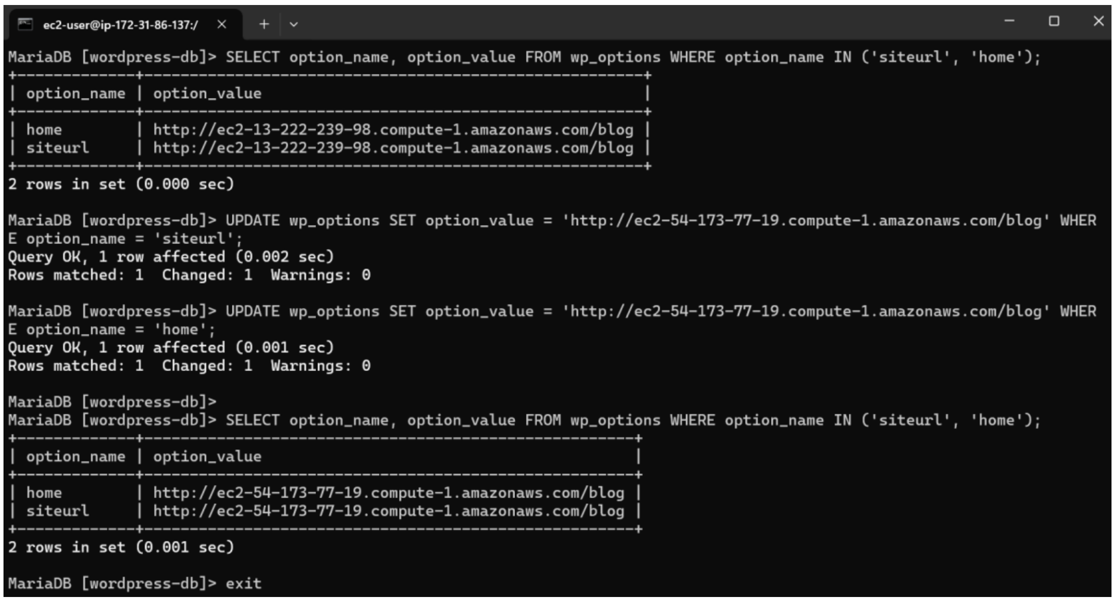
WordPress installation is automatically configured using with the public DNS. If we stop our instances and start again, the public DNS is changed and blog is not working. To solve this problem, we use the following command.

1. sudo mysql -u root -p
2. SHOW DATABASES;
3. USE wordpress-db;
4. SELECT option_name, option_value FROM wp_options WHERE option_name IN ('siteurl', 'home');
5. UPDATE wp_options SET option_value = 'http://3.86.83.103' WHERE option_name = 'siteurl';
   UPDATE wp_options SET option_value = 'http://3.86.83.103' WHERE option_name = 'home';
6. EXIT;
7. sudo systemctl restart httpd



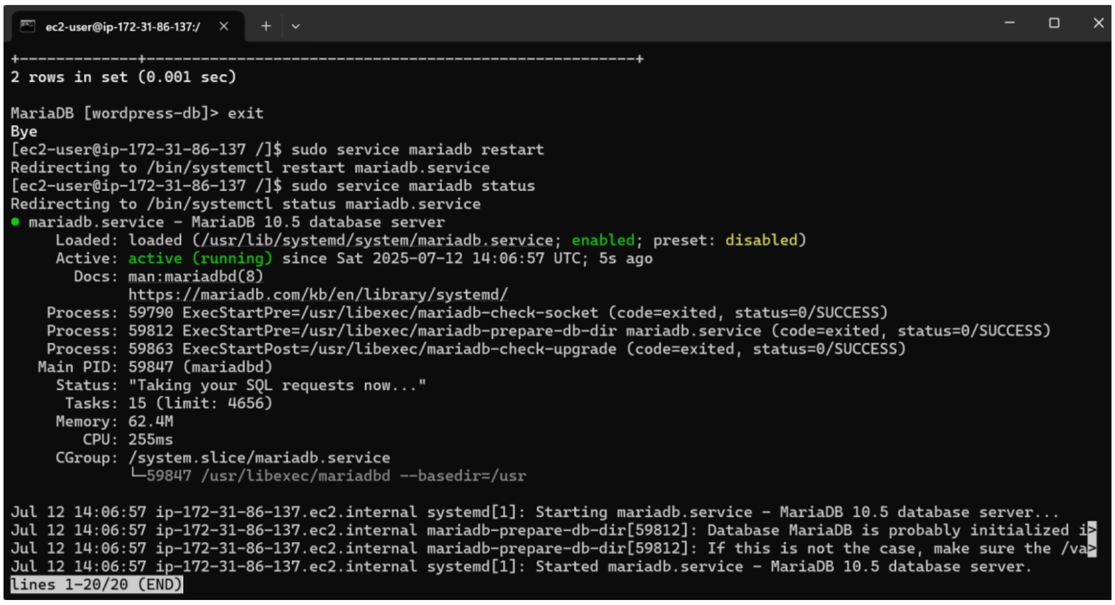
```
ec2-user@ip-172-31-86-137:/

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| wordpress-db       |
+--------------------+
4 rows in set (0.002 sec)

MariaDB [(none)]> USE wordpress-db
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [wordpress-db]> USE wordpress-db;
Database changed
MariaDB [wordpress-db]> SELECT option_name, option_value FROM wp_options WHERE option_name IN ('siteurl', 'home');
+-------------+---------------------------------------------------------+
| option_name | option_value                                            |
+-------------+---------------------------------------------------------+
| home        | http://ec2-13-222-239-98.compute-1.amazonaws.com/blog |
| siteurl     | http://ec2-13-222-239-98.compute-1.amazonaws.com/blog |
+-------------+---------------------------------------------------------+
2 rows in set (0.000 sec)
```

Figure 24: Checking the current database value for WordPress site



```
ec2-user@ip-172-31-86-137:/
MariaDB [wordpress-db]> SELECT option_name, option_value FROM wp_options WHERE option_name IN ('siteurl', 'home');
+-------------+---------------------------------------------------------+
| option_name | option_value                                            |
+-------------+---------------------------------------------------------+
| home        | http://ec2-13-222-239-98.compute-1.amazonaws.com/blog |
| siteurl     | http://ec2-13-222-239-98.compute-1.amazonaws.com/blog |
+-------------+---------------------------------------------------------+
2 rows in set (0.000 sec)

MariaDB [wordpress-db]> UPDATE wp_options SET option_value = 'http://ec2-54-173-77-19.compute-1.amazonaws.com/blog' WHER
E option_name = 'siteurl';
Query OK, 1 row affected (0.002 sec)
Rows matched: 1  Changed: 1  Warnings: 0

MariaDB [wordpress-db]> UPDATE wp_options SET option_value = 'http://ec2-54-173-77-19.compute-1.amazonaws.com/blog' WHER
E option_name = 'home';
Query OK, 1 row affected (0.001 sec)
Rows matched: 1  Changed: 1  Warnings: 0

MariaDB [wordpress-db]>
MariaDB [wordpress-db]> SELECT option_name, option_value FROM wp_options WHERE option_name IN ('siteurl', 'home');
+-------------+--------------------------------------------------------+
| option_name | option_value                                           |
+-------------+--------------------------------------------------------+
| home        | http://ec2-54-173-77-19.compute-1.amazonaws.com/blog |
| siteurl     | http://ec2-54-173-77-19.compute-1.amazonaws.com/blog |
+-------------+--------------------------------------------------------+
2 rows in set (0.001 sec)

MariaDB [wordpress-db]> exit
```

Figure 25: Adding new value to the database



```
+------------+-------------------------------------------------------+
2 rows in set (0.001 sec)

MariaDB [wordpress-db]> exit
Bye
[ec2-user@ip-172-31-86-137 /]$ sudo service mariadb restart
Redirecting to /bin/systemctl restart mariadb.service
[ec2-user@ip-172-31-86-137 /]$ sudo service mariadb status
Redirecting to /bin/systemctl status mariadb.service
● mariadb.service - MariaDB 10.5 database server
     Loaded: loaded (/usr/lib/systemd/system/mariadb.service; enabled; preset: disabled)
     Active: active (running) since Sat 2025-07-12 14:06:57 UTC; 5s ago
       Docs: man:mariadbd(8)
             https://mariadb.com/kb/en/library/systemd/
    Process: 59790 ExecStartPre=/usr/libexec/mariadb-check-socket (code=exited, status=0/SUCCESS)
    Process: 59812 ExecStartPre=/usr/libexec/mariadb-prepare-db-dir mariadb.service (code=exited, status=0/SUCCESS)
    Process: 59863 ExecStartPost=/usr/libexec/mariadb-check-upgrade (code=exited, status=0/SUCCESS)
   Main PID: 59847 (mariadbd)
     Status: "Taking your SQL requests now..."
      Tasks: 15 (limit: 4656)
     Memory: 62.4M
        CPU: 255ms
     CGroup: /system.slice/mariadb.service
             └─59847 /usr/libexec/mariadbd --basedir=/usr

Jul 12 14:06:57 ip-172-31-86-137.ec2.internal systemd[1]: Starting mariadb.service - MariaDB 10.5 database server...
Jul 12 14:06:57 ip-172-31-86-137.ec2.internal mariadb-prepare-db-dir[59812]: Database MariaDB is probably initialized i>
Jul 12 14:06:57 ip-172-31-86-137.ec2.internal mariadb-prepare-db-dir[59812]: If this is not the case, make sure the /va>
Jul 12 14:06:57 ip-172-31-86-137.ec2.internal systemd[1]: Started mariadb.service - MariaDB 10.5 database server.
lines 1-20/20 (END)
```

Figure 26: Restart the database

10.6 Reset Config to default Apache + WordPress

```
# 1. Reinstall core packages
sudo dnf install -y httpd php php-mysqlnd php-fpm mod_ssl

# 2. Reset ownership and permissions
sudo chown -R apache:apache /var/www/html
sudo find /var/www/html -type d -exec chmod 755 {} \;
sudo find /var/www/html -type f -exec chmod 644 {} \;

# 3. Reset .htaccess (overwrite it)
cat <<EOF | sudo tee /var/www/html/.htaccess
<IfModule mod_rewrite.c>
RewriteEngine On
RewriteBase /
RewriteRule ^index\.php$ - [L]
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule . /index.php [L]
</IfModule>
EOF

# 4. Enable PHP-FPM correctly
sudo systemctl enable php-fpm
sudo systemctl start php-fpm

# 5. Enable and restart Apache
sudo systemctl enable httpd
sudo systemctl restart httpd

# 6. Allow port 80 in firewall (again just to be sure)
sudo firewall-cmd --add-service=http --permanent
sudo firewall-cmd --reload
```

10.7 Vulnerability_Report and Script to fix

No	Too	Severi	Name	Por	Description	CVSS Score	Script to fix
1	Nessus	Critical	SSL Certificate Expiry	443	The SSL certificate for this service will expire soon.	9.0	
2	Nmap	Critical	OpenSSH 8.7 CVE-2023-38408	22	Multiple exploits found for OpenSSH 8.7 including CVE-2023-38408.	9.8	sudo nano /etc/ssh/sshd_config AllowAgentForwarding no
3	Nessus	High	Apache Path Disclosure	80	The web server reveals internal paths.	7.5	
4	Nmap	High	Apache 2.4.62 CVEs	80/443	Apache 2.4.62 vulnerable to several CVEs such as CVE-2025-49812.	9.8	#sudo nano /etc/httpd/conf.d/security.conf #<IfModule mod_headers.c> Header always set X-Content-Type-Options "nosniff" Header always set X-Frame-Options "SAMEORIGIN" Header always set X-XSS-Protection "1; mode=block" Header always set Referrer-Policy "strict-origin-when-cross-origin" Header always set Content-Security-Policy "default-src 'self'" </IfModule> #sudo systemctl restart httpd #curl -I http://localhost
5	Nikto	Info	WordPress Identified	80	WordPress installation detected.		
6	Nikto	Low	Missing Security Headers	80	X-Content-Type-Options, CSP, HSTS, Referrer-Policy, Permissions-Policy missing.		
7	Nikto	Low	Directory Indexing Found	80	/icons/ folder is browsable.		
8	Nikto	Low	Exposed Files	80	Found license.txt, readme.html, wp-app.log files revealing server details.		
9	WPScan	Low	readme.html Exposed	80	readme.html file discloses WordPress version.		
10	WPScan	Low	User Enumeration	80	WPScan identified user 'test' via API.		
11	WPScan	Low	WP-Cron Exposed	80	External WP-Cron enabled, potential abuse for DoS.		
12	Nikto	Medium	TRACE Method Enabled	80	TRACE method is active; vulnerable to Cross-Site Tracing (XST).		#sudo nano /etc/httpd/conf/httpd.conf #<IfModule mod_rewrite.c> RewriteEngine On RewriteCond %{REQUEST_METHOD} ^TRACE RewriteRule .* - [F] </IfModule> #curl -X TRACE http://<your-public-ip> -i
13	Nmap	Medium	WordPress Enumeration	80	User 'test' identified; login pages exposed; multiple versions detected in assets.	5.0	
14	Nmap	Medium	TRACE Method Enabled	80/443	TRACE HTTP method still enabled, confirmed by Nmap.		#sudo nano /etc/httpd/conf/httpd.conf #TraceEnable off #curl -X TRACE http://your-ip/
15	WPScan	Medium	XML-RPC Enabled	80	XML-RPC is accessible and can be abused for DoS or brute-force attacks.	6.5	#sudo nano /var/www/html/.htaccess #<Files xmlrpc.php> Order allow,deny Deny from all </Files> #curl -I http://your-ip/xmlrpc.php