



A REPORT ON

Deployment on AWS in Monolithic and Microservices Architectures

Submitted by:

DEEP KUMAR P KHANRA

Email: dkhanra605@gmail.com

TABLE OF CONTENTS

SL.NO	DESCRIPTION	PAGE NO.
1	INTRODUCTION	1
2	ARCHITECTURE OVERVIEW	2-3
3	DEPLOYMENT REQUIREMENTS	4-6
4	DEPLOYMENT STEPS	7-20
5	COMAPRISON OF ARCHITECTURES	21
6	CONCLUSION	22
7	REFERENCES	23

INTRODUCTION

With the growth of cloud computing, deploying applications in cloud environments has become essential for ensuring scalability, flexibility, and efficient resource management. Amazon Web Services (AWS) is a comprehensive and widely adopted cloud platform that provides a suite of on-demand computing services, including storage, computing power, databases, machine learning, analytics, and more. It is designed to help businesses and developers scale, innovate, and manage their technology needs with efficiency, reliability, and cost-effectiveness. AWS operates on a global scale, with data centers around the world, allowing users to deploy resources close to their customers for better performance.

This report focuses on deploying a WordPress application on Amazon Web Services (AWS), using two different architectural approaches: Monolithic and Microservices. WordPress, one of the most popular content management systems (CMS) worldwide, requires a web server and a database to function. AWS offers an ideal platform for deploying such applications due to its robust infrastructure, flexible configurations, and range of scalable services.

The goal of this deployment is to explore and compare the **Monolithic** and **Microservices** architectures. Each architecture will be implemented using EC2 instances, which provide scalable, resizable virtual machines in the cloud. By setting up the application in both monolithic and microservices styles, this report aims to outline the benefits and limitations of each approach, offering insights into which architecture best suits various deployment need

ARCHITECTURE OVERVIEW

The architecture of a web application defines how its components are structured, interact, and communicate to deliver functionality. This section provides an in-depth look at the two architectures used for deploying the WordPress application on AWS: **Monolithic Architecture** and **Microservices Architecture**. Each architecture has its own set of characteristics, benefits, and trade-offs, which influence the performance, scalability, and maintenance of the deployed application.

1. Monolithic Architecture

In the monolithic approach, all components of the application are bundled together within a single instance. This means that the WordPress application, web server, and MySQL database all run on the same EC2 instance. In this deployment:

- **Single EC2 Instance:** The entire application stack, including both WordPress (the application layer) and MySQL (the database layer), is installed on a single t2.micro EC2 instance.
- **Unified Resource Usage:** The resources (CPU, memory, and storage) of the instance are shared among all components, resulting in simpler management and setup.
- **Security Group Configuration:** One security group is set up to allow HTTP/HTTPS access (for WordPress) and MySQL access within the instance.

This structure is ideal for applications with low to moderate traffic and simple architectures. It provides a straightforward deployment model, ideal for quick setups or smaller projects..

2. Microservices Architecture

The **Microservices Architecture** model breaks the application into distinct, independently managed components, deploying each on its own EC2 instance. In this setup, WordPress and MySQL run on separate EC2 instance, representing a more modular approach.

- **Dedicated EC2 Instances:** The application layer (WordPress) and database layer (MySQL) are separated across two t2.micro EC2 instances. The WordPress instance handles HTTP requests, serves web pages, and processes user interactions, while the MySQL instance manages data storage, retrieval, and transactions.
- **Component Isolation:** This separation of concerns means each component is isolated, reducing the likelihood of one component affecting the performance or stability of the other.
- **Security Group Configuration:** Two security groups are set up — one for the WordPress instance (allowing HTTP, HTTPS, and SSH access) and one for the MySQL instance (restricting access to port 3306 from the WordPress instance only). This approach minimizes exposure and enhances security.

This modular structure is well-suited for applications that anticipate scaling needs or for production environments where uptime and performance are critical. It allows independent updates, maintenance, and scaling, making it an excellent choice for dynamic and evolving application ecosystems.

DEPLOYMENT REQUIREMENTS

To deploy the WordPress application using both Monolithic and Microservices architectures on AWS, several key requirements must be met. These include selecting appropriate EC2 instances, configuring security groups, setting up the necessary software environment, and ensuring secure communication between components.

1. Amazon EC2 Instances

- **Instance Type:** For both architectures, we'll use the t2.micro instance type, which is part of AWS's free tier. This instance provides sufficient CPU and memory resources for lightweight web and database applications, such as the WordPress setup.
- **Amazon Machine Image (AMI):** We use the Ubuntu AMI (ubuntu-*) as the base image for each instance. Ubuntu is commonly used for web applications making it ideal for WordPress and MySQL deployments. The AMI provides a stable environment and includes necessary packages for further customization.

2. Network Configuration

- **Virtual Private Cloud (VPC):** Each EC2 instance will be deployed within a VPC, which is AWS's default isolated network. This network ensures secure communication between components (in the microservices setup).
- **Subnet Configuration:** Both the Monolithic and Microservices setups require instances to be in a public subnet to allow internet access. This is

especially necessary for setting up and accessing WordPress from a web browser.

3. Security Groups

- **Port Configurations:** Security groups act as virtual firewalls, controlling traffic to each EC2 instance. Specific port configurations are needed to allow WordPress and MySQL traffic while restricting unauthorized access.
 - **HTTP (Port 80) and HTTPS (Port 443):** Open on WordPress instance to allow web traffic.
 - **SSH (Port 22):** Open on both instances for administrative access.
 - **MySQL (Port 3306):** Open only to the WordPress instance in the microservices setup to restrict database access.
- **Security Groups by Architecture:**
 - **Monolithic:** A single security group allows HTTP, HTTPS, and SSH access to the single instance.
 - **Microservices:** Separate security groups for each instance:
 - **WordPress Instance:** Allows HTTP, HTTPS, and SSH.
 - **MySQL Instance:** Restricts access on port 3306 to the WordPress instance IP only.

4. Software Installation

To set up WordPress and MySQL, several software components need to be installed and configured on the EC2 instances.

- **LAMP Stack (Linux, Apache, MySQL, PHP):** The base setup for running WordPress.
 - **Apache:** As the web server to serve WordPress

- **MySQL:** As the relational database for WordPress data storage.
- **PHP:** For executing WordPress's backend code.
- **WordPress:** Install WordPress after setting up the LAMP stack. The files can be downloaded from the official WordPress site and extracted into Apache's web directory (/var/www/html).
- **Configuration Files:**
 - **wp-config.php:** The primary configuration file for WordPress, which connects it to the MySQL database and defines database details like the name, user, and password.
 - **MySQL Database Setup:** Create a dedicated database and user for WordPress. In a microservices setup, configure MySQL to accept connections from the WordPress instance's IP.

5. Communication and Access Control

- **Database User Permissions:** In both architectures, the MySQL user for WordPress should have restricted permissions to prevent unauthorized access.
- **Firewall Rules:** Configure firewall rules to ensure that MySQL traffic is restricted only to the specific WordPress instance.

6. Testing and Validation

- **Accessing WordPress:** Verify that the WordPress homepage loads in the browser via the EC2 instance's public IP.
- **Database Connectivity:** In the microservices setup, ensure that WordPress can communicate with the MySQL instance by testing page loads and database interactions.

DEPLOYMENT STEPS

Monolithic Deployment Steps

1. Launch EC2 Instance:

- Log into the AWS Management Console and launch a new EC2 instance with the **Ubuntu AMI** (e.g., ubuntu-20.04), using the **t2.micro** instance type (free tier eligible).
- Choose a security group that allows HTTP (port 80) and SSH (port 22) access.

i-056ac6a34387daca9 (monolithic-wordpress-sg)

▼ Instance details [Info](#)

Platform
Ubuntu

Platform details
Linux/UNIX

Stop protection
Disabled

Instance auto-recovery
Default

AMI ID
[ami-0dee22c13ea7a9a67](#)

AMI name
[ubuntu/images/hvm-ssd-gp3/ubuntu-noble-24.04-amd64-server-20240927](#)

Launch time
[Mon Nov 04 2024 15:41:40 GMT+0530 \(India Standard Time\)](#) (about 3 hours)

Lifecycle
normal

Monitoring
disabled

Termination protection
Disabled

AMI location
[amazon/ubuntu/images/hvm-ssd-gp3/ubuntu-noble-24.04-amd64-server-20240927](#)

Stop-hibernate behavior
Disabled

Filter rules						< 1 >
Name	Security group rule ID	Port range	Protocol	Source	Security groups	
-	sgr-0a070cd341a85b0e5	80	TCP	0.0.0.0/0	launch-wizard-2	
-	sgr-0c97151a17e039908	3306	TCP	0.0.0.0/0	launch-wizard-2	
-	sgr-0870d7b414ab842ad	443	TCP	0.0.0.0/0	launch-wizard-2	
-	sgr-05b72b16b3d63888c	22	TCP	0.0.0.0/0	launch-wizard-2	

<input checked="" type="checkbox"/>	monolithic-wo...	i-056ac6a34387daca9	<input checked="" type="checkbox"/> Running	<input checked="" type="checkbox"/> t2.micro	<input checked="" type="checkbox"/> 2/2 checks passed
-------------------------------------	------------------	---------------------	---	--	---

2. Connect to the EC2 Instance:

- SSH into your EC2 instance using your SSH key pair.

▼ Instance summary [Info](#)

Instance ID

i-056ac6a34387daca9

IPv6 address

-

Public IPv4 address

13.201.230.57 | [open address](#)

Instance state

Running

Private IPv4 addresses

172.31.14.27

Public IPv4 DNS

ec2-13-201-230-57.ap-south-1.compute.amazonaws.com |

[open address](#)

3. Update the Instance:

- Once connected, update the system packages on your instance to ensure you have the latest versions of software and security patches.

4. Install LAMP Stack (Linux, Apache, MySQL, PHP):

- Install Apache web server, MySQL database server, and PHP along with necessary modules for WordPress to work. These components form the backbone of your WordPress environment.

5. Configure MySQL Database:

- Secure MySQL and create a database for WordPress. This database will store all WordPress data.
- Set up a user with necessary permissions for accessing the WordPress database.

```
sudo apt install mysql-server -y
```

6. Install WordPress:

- Download the latest version of WordPress, extract the files, and configure the WordPress settings to connect to the database you created earlier.

```
wget https://wordpress.org/latest.tar.gz
tar -xzf latest.tar.gz
sudo mv wordpress/* /var/www/html/
sudo chown -R www-data:www-data /var/www/html/
```

7. Configure Apache for WordPress:

- Set up an Apache virtual host to serve WordPress from the correct directory. This may involve modifying the Apache configuration files.

```
define('DB_NAME', 'wordpress_db');
define('DB_USER', 'wp_user');
define('DB_PASSWORD', 'password');
define('DB_HOST', 'localhost');
```

8. Set Permissions:

- Set the correct file permissions on the WordPress directories so that Apache can serve files and PHP can write to the database.

9. Install and Configure WordPress:

- Navigate to the IP address or domain name of your EC2 instance in a browser, and complete the WordPress setup process by providing the database connection details and admin credentials.

10. Create a Welcome Page:


- Once WordPress is installed, go into the admin panel, create a static page that will serve as the homepage, and make it the default page.

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	<input type="text"/>
Username	<input type="text" value="deep_kumar_52"/> <small>Username can have only alphanumeric characters, spaces, underscores, hyphens, periods, and the @ symbol.</small>
Password	<input type="password" value="ty5svBfZS@f(JHhkaA"/> <div>Strong</div> <div>Important: You will need this password to log in. Please store it in a secure location.</div>
Your Email	<input type="text" value="dkhanra605@gmail.com"/> <small>Double-check your email address before continuing.</small>

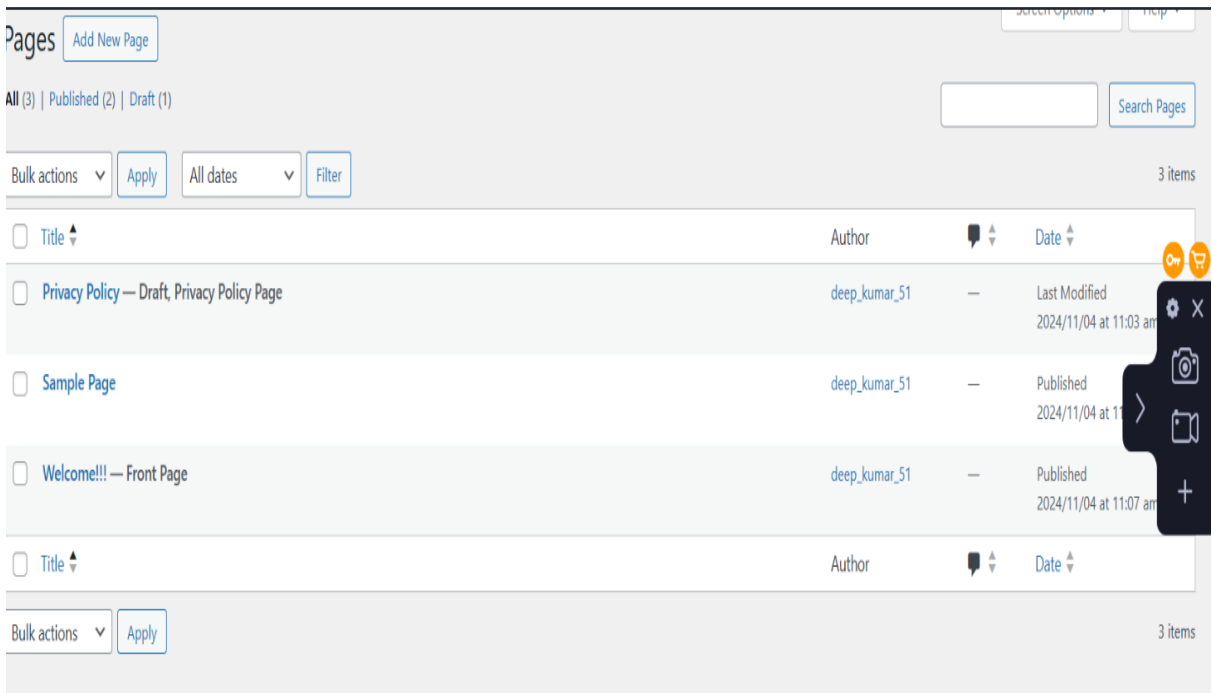
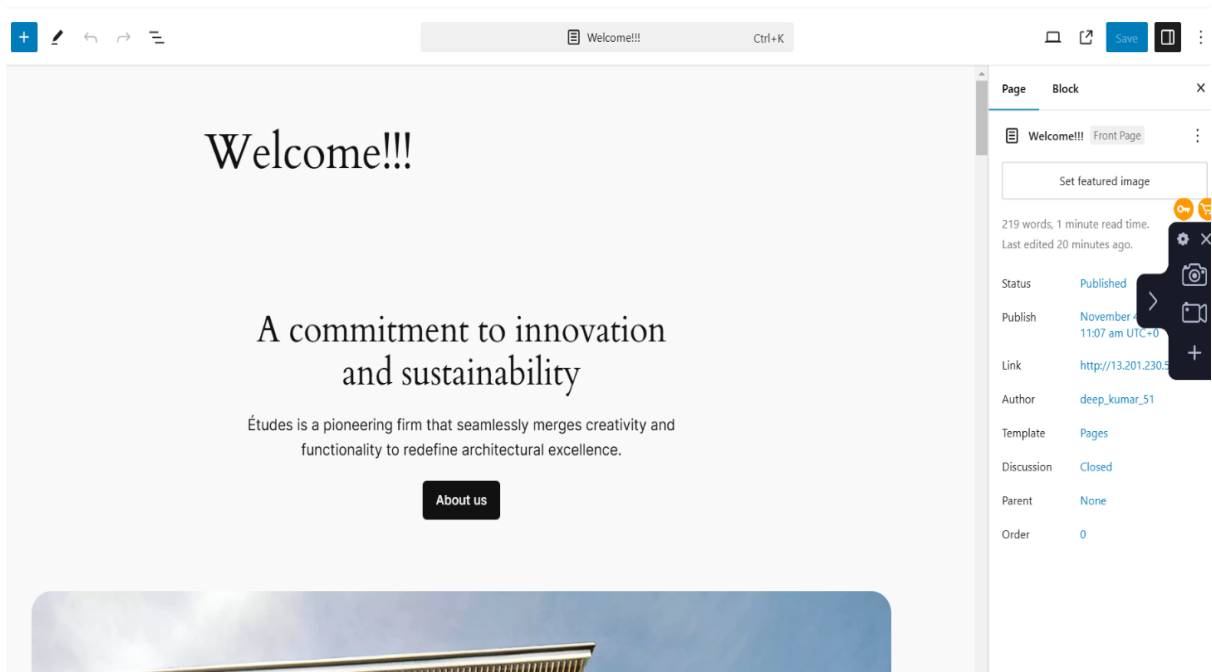


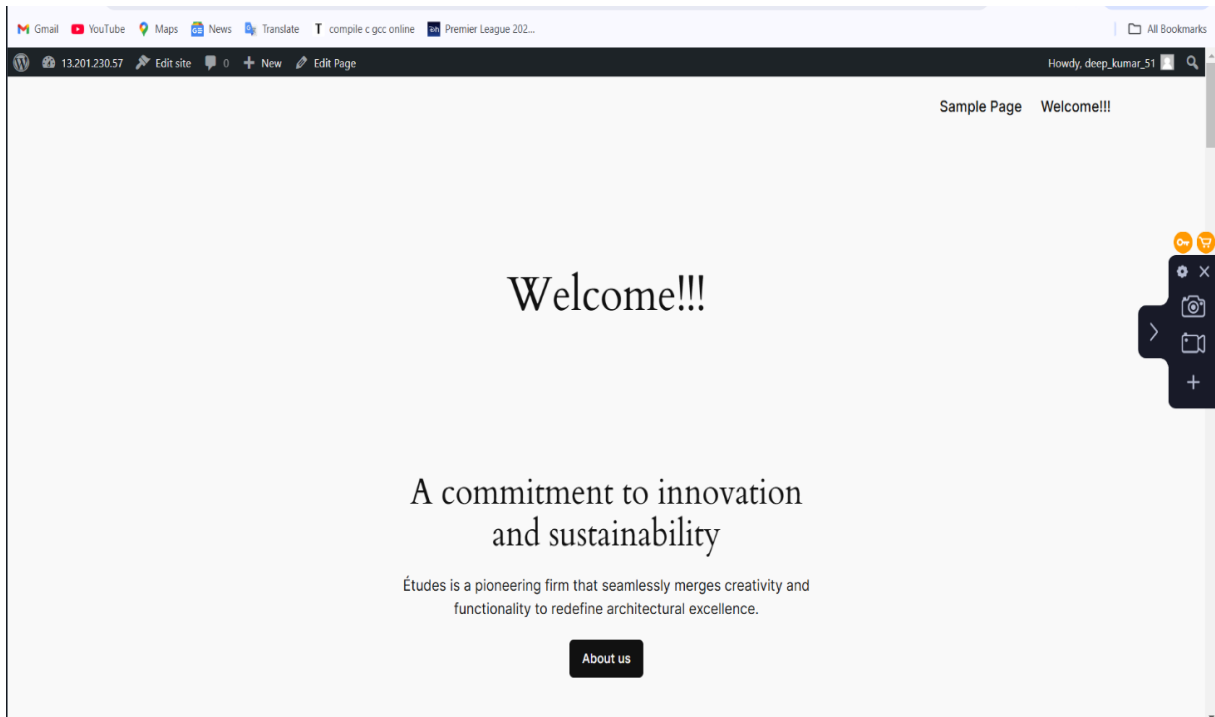
Success!

WordPress has been installed. Thank you, and enjoy!

Username	deep_kumar_52
Password	Your chosen password.

[Log In](#)

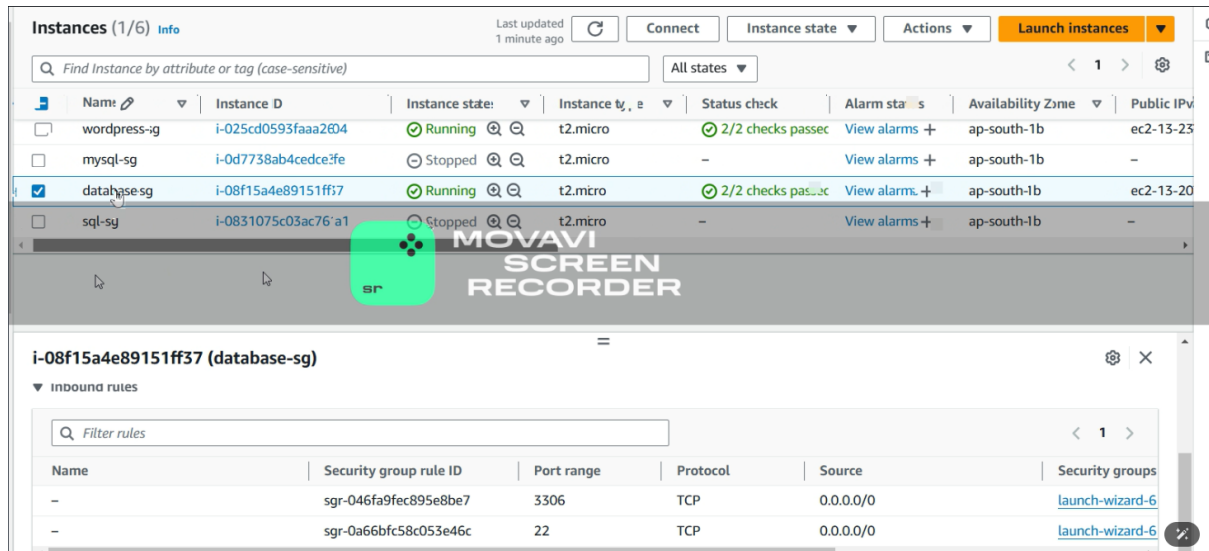




Microservices Deployment Steps

1. Launch EC2 Instances:

- Launch two EC2 instances (each with Ubuntu AMI and t2.micro instance type). One will host the WordPress application and the other will host the MySQL database.
- Set up security groups to allow HTTP access (port 80) for WordPress, and MySQL access (port 3306) between the two instances



2. Connect to Both EC2 Instances:

- SSH into both EC2 instances separately using your SSH key pair.

```
ubuntu@ip-172-31-4-81:~$ ssh -i "C:\Users\DEEPAKUMAR\Downloads\wordpresskey.pem"
```

3. Update the Instances:

- Update system packages on both instances to ensure they have the latest software and security updates.

4. Install LAMP Stack on WordPress Instance:

- Install Apache, PHP, and necessary PHP extensions on the WordPress instance.

5. Install and Configure MySQL on the MySQL Instance:

- Install MySQL on the second EC2 instance and secure it.
- Create a WordPress database and user for the WordPress instance to connect to.

```
All done!
ubuntu@ip-172-31-4-81:~$ sudo mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 10
Server version: 8.0.39-0ubuntu0.22.04.1 (Ubuntu)

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql> create database wordpress_db;
Query OK, 1 row affected (0.00 sec)

mysql> create user 'wordpress_user'@ '%' identified by 'Qwerty123!';
Query OK, 0 rows affected (0.02 sec)

mysql> grant all privileges on wordpress_db.* to 'wordpress_user'@ '%';
Query OK, 0 rows affected (0.00 sec)

mysql> flush privilege|
```

6. Configure Networking Between EC2 Instances:

- Make sure the security group of the WordPress instance allows inbound connections to MySQL on port 3306 from the MySQL instance's private IP.


```
ubuntu@ip-172-31-4-81:~$ sudo nano /etc/mysql/conf.d/mysqld.cnf
ubuntu@ip-172-31-4-81:~$ sudo nano /etc/mysql/mysql.conf.d/mysqld.cnf
ubuntu@ip-172-31-4-81:~$ sudo systemctl restart mysql
```

7. Install WordPress on the WordPress Instance:

- Download and extract WordPress on the WordPress EC2 instance.
- Configure the wp-config.php file to connect to the MySQL database hosted on the second EC2 instance.

8. Install Apache on WordPress Instance:

- Set up Apache on the WordPress EC2 instance to serve WordPress content, configure virtual hosts, and ensure it is pointing to the correct directory where WordPress is installed.

```
No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@ip-172-31-4-81:~$ sudo systemctl start apache2
ubuntu@ip-172-31-4-81:~$ sudo systemctl enable apache2
Synchronizing state of apache2.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable apache2
ubuntu@ip-172-31-4-81:~$ cd /tmp
ubuntu@ip-172-31-4-81:/tmp$ wget https://wordpress.org/latest.tar.gz
--2024-11-04 12:29:21-- 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: 24640061 (23M) [application/octet-stream]
Saving to: 'latest.tar.gz'
```

9. Set Permissions on WordPress Instance:

- Set the appropriate file and directory permissions on the WordPress instance for Apache and PHP to access and modify files.

```

ubuntu@ip-172-31-4-81:/tmp$ sudo mv wordpress/* /var/www/html/
ubuntu@ip-172-31-4-81:/tmp$ sudo chown -R www-data:www-data /var/www/html/
ubuntu@ip-172-31-4-81:/tmp$ sudo chmod -R 755 /var/www/html/
ubuntu@ip-172-31-4-81:/tmp$ sudo cp /var/www/html/wp-config-sample.php /var/www/html/wp-config.php

```

10. Configure WordPress:

- Navigate to the WordPress instance's IP address or domain in the browser and complete the WordPress installation process.
- Provide database details, set up the admin account, and finalize the installation.

```

*
* This file contains the following configurations:
*
* * Database settings
* * Secret keys
* * Database table prefix
* * ABSPATH
*
* @link https://developer.wordpress.org/advanced-administration/wordpress/wp-config/
*
* @package WordPress
*/

/** Database settings - You can get this info from your web host ** */
/** The name of the database for WordPress */
define( 'DB_NAME', 'wordpress_db' );

/** Database username */
define( 'DB_USER', 'wordpress_user' );

/** Database password */
define( 'DB_PASSWORD', 'Qwerty123!' );

/** Database hostname */
define( 'DB_HOST', 'localhost' );

```

MOVAVI SCREEN RECORDER

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location M-U Undo M-A Set Mark
 ^X Exit ^R Read File ^N Replace ^U Paste ^J Justify ^_ Go To Line M-E Redo M-G Copy

```

/** Database hostname */
define( 'DB_HOST', '172.31.4.81' );

```

```
ubuntu@ip-172-31-11-116:/tmp$ mysql -u wordpress_user -p -h 172.31.4.81
Enter password:
ERROR 1045 (28000): Access denied for user 'wordpress_user'@'ip-172-31-11-116.ap-south-1.compute.internal' (using password: NO)
ubuntu@ip-172-31-11-116:/tmp$ mysql -u wordpress_user -p -h 172.31.4.81
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 8.0.39-0ubuntu0.22.04.1 (Ubuntu)

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql> exit;
Bye
ubuntu@ip-172-31-11-116:/tmp$ sudo systemctl start apache2
ubuntu@ip-172-31-11-116:/tmp$ ls /var/www/html
index.html  readme.html  wp-blog-header.php  wp-config.php  wp-includes  wp-login.php  wp-signup.php
index.php   wp-activate.php  wp-comments-post.php  wp-content  wp-links-opml.php  wp-mail.php  wp-trackback.php
license.txt  wp-admin        wp-config-sample.php  wp-cron.php  wp-load.php      wp-settings.php  xmlrpc.php
```

11. Create a Welcome Page in WordPress:

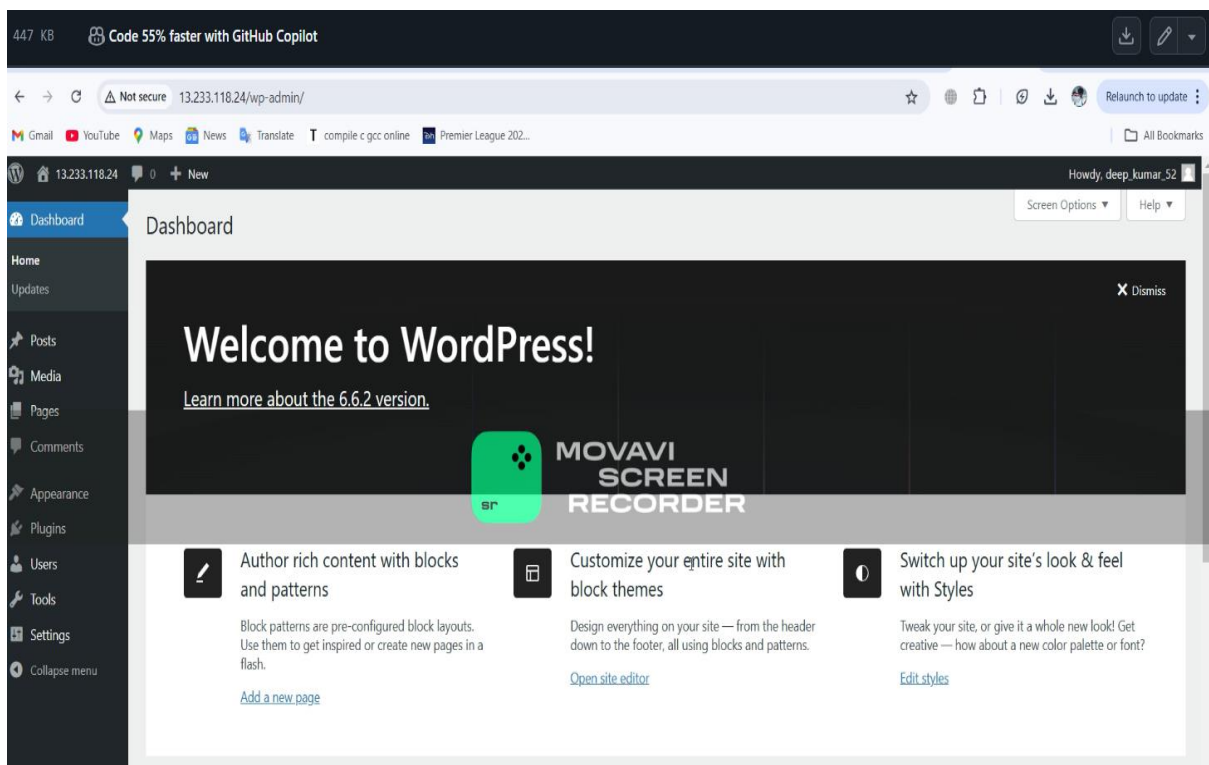
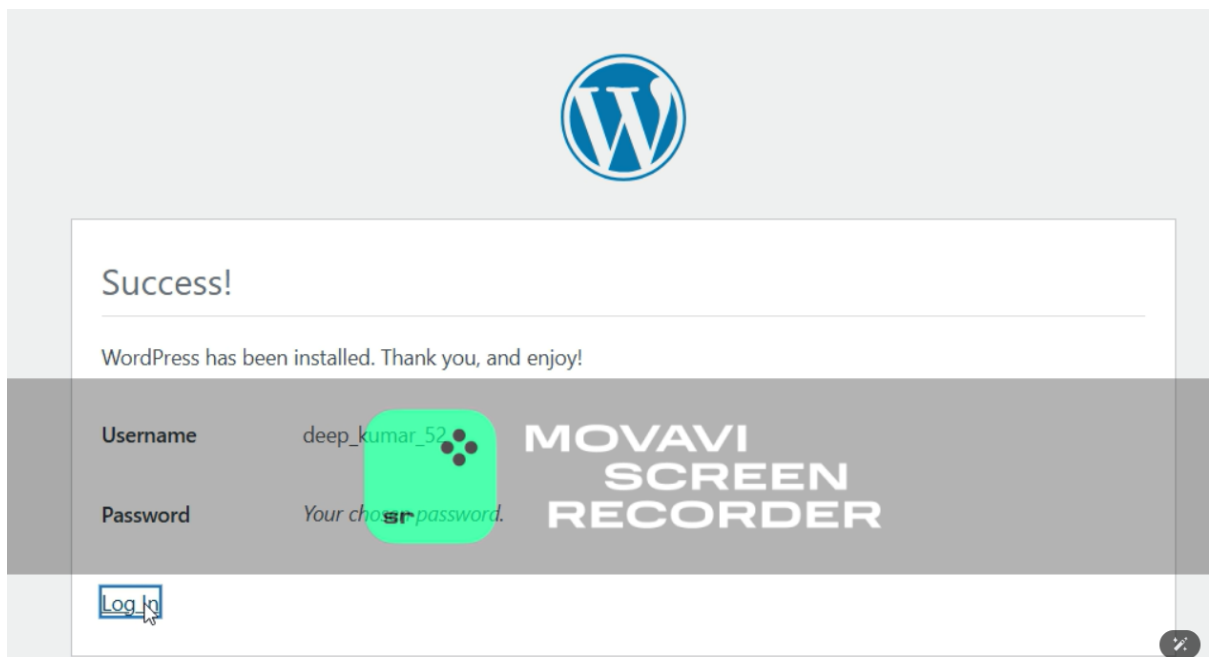
In the WordPress admin panel, create a static welcome page and set it as the homepage of your website.

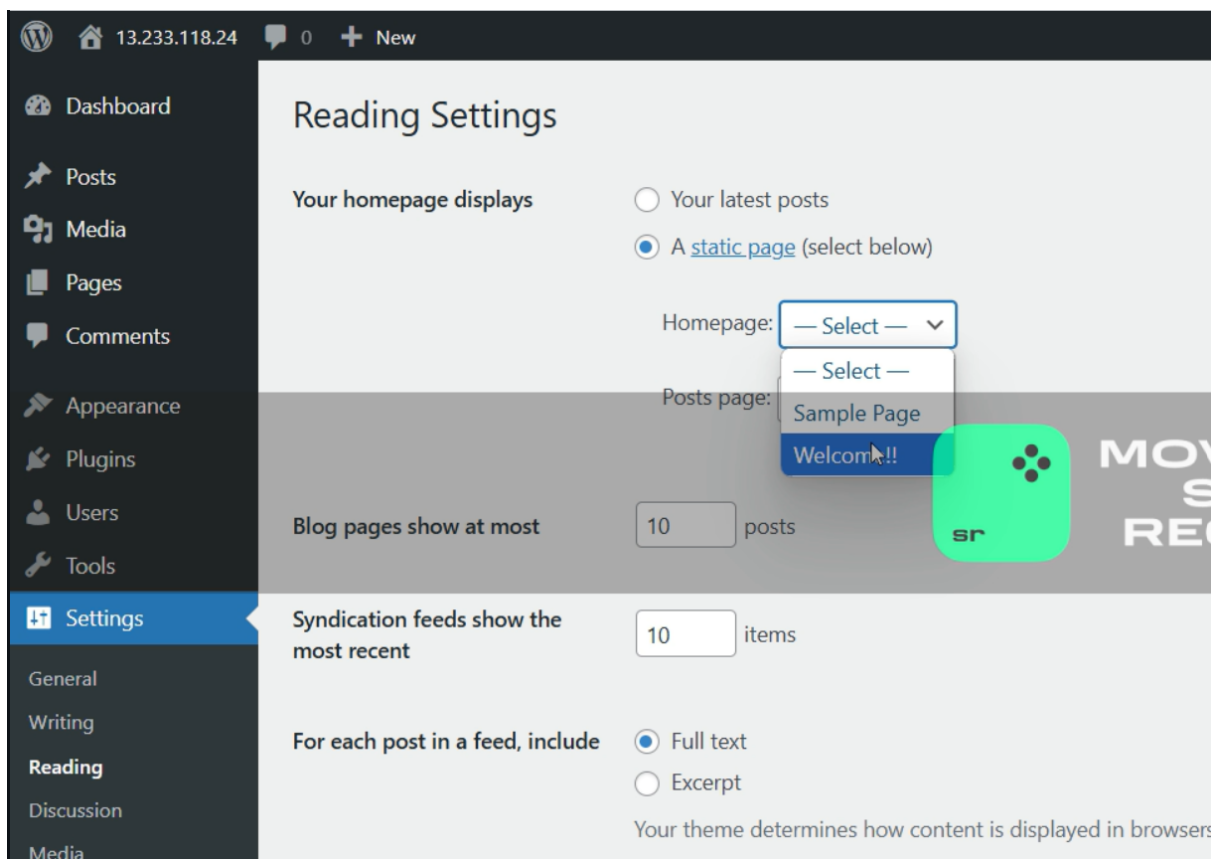
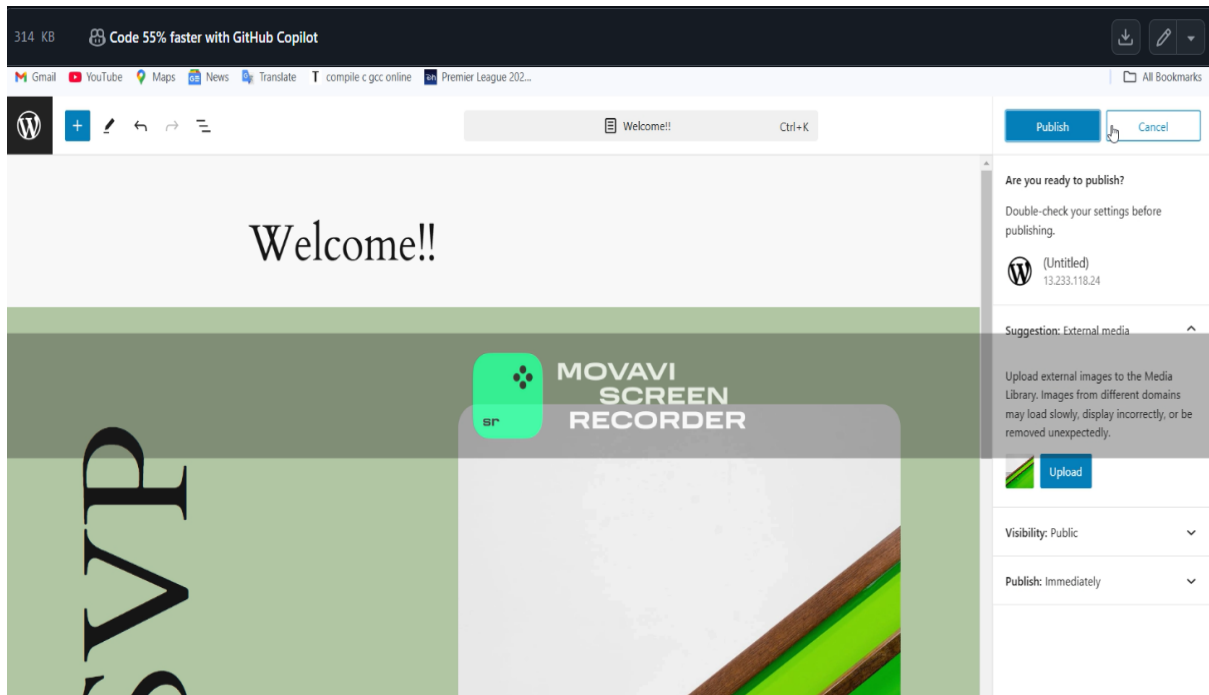
Welcome to the famous five-minute WordPress installation process! Just fill in the information below and you'll be on your way to using the most extendable and powerful personal publishing platform in the world.

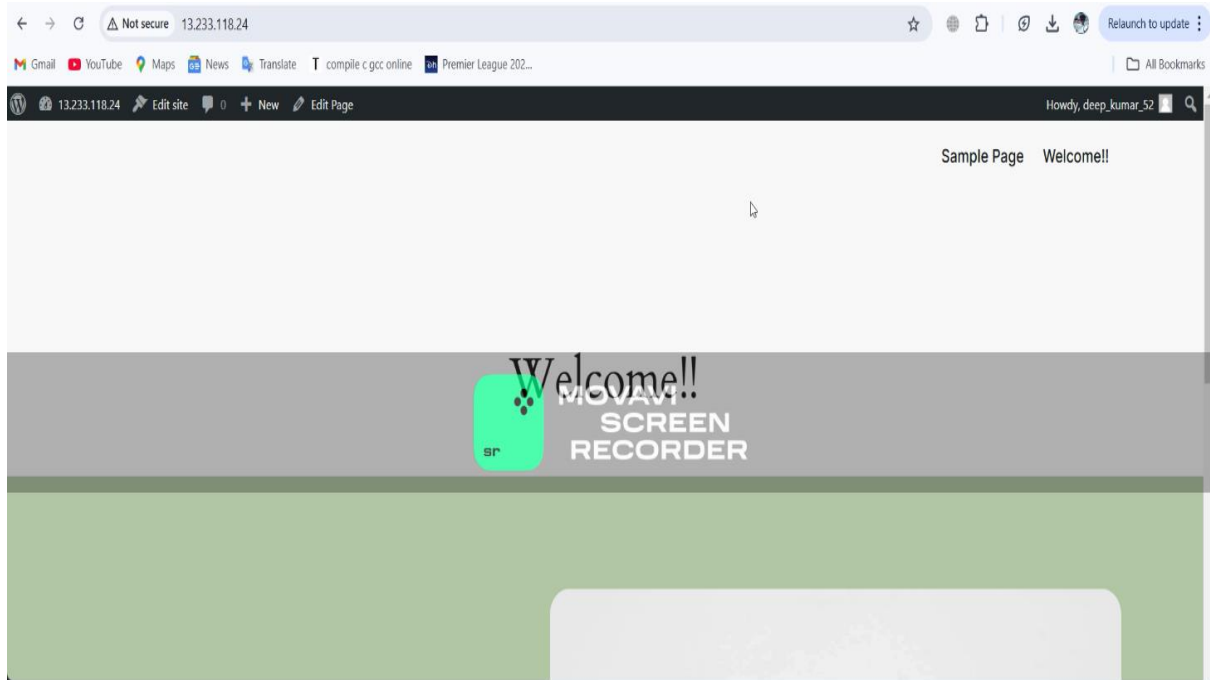
Information needed

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

Site Title	<input type="text"/>
Username	<input type="text" value="deep_kumar_52"/> <small>Username can have only alphanumeric characters, spaces, underscores, hyphens, periods, and the @ symbol.</small>
Password	<input type="password" value="ty5svBfZS@f(JHhkaA"/> <div>Strong</div> <div>Important: You will need this password to log in. Please store it in a secure location.</div>
Your Email	<input type="text" value="dkhanra605@gmail.com"/> <small>Double-check your email address before continuing.</small>







COMPARISON OF ARCHITECTURE

FACTOR	MONOLITHIC	MICROSERVICES
Design	Combined on one instance	Separate instances for each service
Scalability	Limited, entire instance must be scaled	Independent, each service scaled separately
Fault Tolerance	Low, failure affects all services	High, failures isolated to specific services
Performance	Good under moderate load, bottlenecks likely	Better in high-load, optimized per service
Complexity	Lower, easier setup	Higher, more setup and networking required
Security	Basic, limited isolation	Enhanced, specific settings per service
Maintenance	Simple, downtime affects all	Flexible, targeted updates possible

CONCLUSION

In conclusion, choosing between monolithic and microservices architecture depends on the project's requirements for scalability, flexibility, and complexity management. Monolithic architecture is straightforward and effective for small to moderate applications, where WordPress and MySQL are deployed on a single server. This setup simplifies the deployment and maintenance processes as it consolidates all services into one instance, which can be advantageous for straightforward applications that don't anticipate high growth or frequent modifications.

The inability to scale individual components independently, combined with shared resources for WordPress and MySQL, can create performance bottlenecks and limit fault tolerance. In scenarios where traffic spikes or specific services require individual optimization, the microservices approach provides a clear advantage. By separating WordPress and MySQL onto different instances, microservices architecture enables each service to function autonomously, allowing for independent scaling, dedicated resources, and reduced risk of a single point of failure affecting the entire application.

The microservices approach is generally more complex to configure and manage, as it requires more careful attention to security, networking, and resource allocation. However, this investment in complexity yields significant benefits in terms of flexibility, reliability, and performance under higher loads. In summary, the monolithic approach is better suited for simpler applications with low-to-moderate growth expectations, while microservices offer scalability and robustness for applications requiring a higher degree of resilience and independence among components.

REFERENCES

1. AWS Documentation on EC2 and WordPress Deployment

<https://aws.amazon.com/getting-started/hands-on/launch-a-wordpress-website/>

2. Using Amazon RDS and EC2 for Scalable Databases (for Microservices)

https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_GettingStarted.html

3. "How to Deploy WordPress on AWS EC2" by Tech With Sam

<https://www.youtube.com/watch?v=3fUpn2WS1XY>

4. "Monolithic vs Microservices Architecture Explained with Real Life Examples" by Code With Harry

https://www.youtube.com/watch?v=UIgSDQ_jM8w

5. "Microservices Best Practices for Beginners" by AWS Online Tech Talks

<https://www.youtube.com/watch?v=n4wk7Rl6elQ>