

AWS Cloud Engineer
Trainee Assignment Documentation

Cirrusgo

Prepared by:
Mustafa Abed Alhakim Al-Shamali

Submission Date:
22/sep/2025

Supervised by:
Hamza Noweder

Introduction

This project demonstrates the design and deployment of a scalable, secure, and resilient WordPress application on AWS using a Three-Tier Architecture. The architecture is structured into three distinct layers — Web Tier, Application Tier, and Database Tier — to ensure separation of concerns, security, and scalability while following the AWS Well-Architected Framework.

The deployment is carried out inside a custom VPC spanning two Availability Zones (AZs) for high availability and fault tolerance. The network is divided into four subnets:

- Two Public Subnets (one in each AZ) hosting the Application Load Balancer (ALB) and NAT Gateways, enabling internet-facing access and outbound traffic for private resources.
- Two Private Subnets (one per AZ) hosting the EC2 instances running the WordPress application, an Amazon RDS database instance (in AZ1), and Amazon EFS for shared storage between EC2 instances.

To ensure scalability and resilience, the EC2 instances are managed by an Auto Scaling Group (ASG), which dynamically adjusts capacity based on traffic. Shared content (e.g., WordPress uploads) is persisted using Amazon EFS, ensuring consistency across multiple instances. For database persistence, Amazon RDS provides a managed relational database backend.

For secure connectivity, the project leverages AWS Instance Connect Endpoint (ICE) to access EC2 instances without exposing SSH ports to the internet, enhancing security by eliminating the need for public SSH access.

The deployment process is automated using user data scripts that configure the environment, mount the EFS filesystem, install required Apache/PHP dependencies, configure MySQL client, and set up WordPress with proper permissions.

Assumptions

In designing and deploying this solution, the following assumptions were made:

1. Database Selection

The application requires a relational database for WordPress. Amazon RDS (MySQL engine) was chosen for simplicity, managed scaling, and compatibility with WordPress. A single-AZ deployment was assumed instead of Multi-AZ to remain within budget, with the understanding that this slightly reduces availability compared to enterprise-grade deployments.

2. Cost and Budget Constraints

The project was implemented under the AWS Free Tier where possible. EC2 instances and RDS were selected from Free Tier-eligible instance types (e.g., t2.micro/t3.micro). Amazon EFS was provisioned with General Purpose storage class, assuming workload requirements fit within Free Tier usage limits. Multi-AZ RDS, provisioned IOPS storage, or enterprise-class EC2 families (e.g., M- or C-series) were not used due to cost constraints.

3. Traffic & Workload

The workload is assumed to be low and consistent, without significant traffic spikes. An Auto Scaling Group (ASG) was still configured to handle any unexpected increases in demand, even though it is not expected under normal operation.

4. Networking & Access

All resources are deployed within a single AWS region for simplicity. Connectivity to the application is assumed to be through HTTP via the Application Load Balancer. Administrative access to EC2 instances is assumed to be possible through AWS Instance Connect Endpoint (ICE), avoiding the need for public SSH access.

5. Storage & File Sharing

WordPress media uploads are assumed to require shared storage across EC2 instances, hence Amazon EFS was chosen. Storage requirements are assumed to be minimal for example within Free Tier or low-cost thresholds.

6. Scope Limitations

The solution is designed as a training and demonstration project, not an enterprise production environment features like WAF, CloudFront CDN, Route 53 DNS failover,

or backup/restore policies are considered out-of-scope for this initial deployment but can be added in future improvements.

Theoretical Design

1. Architecture Diagram



2. Framework Considerations

2.1 Operational Excellence

The system will monitor, maintained and improved over time by using different AWS resources, for example to monitor the health of the instances and provision them over the time we attached an ALB with a TG to the instances to check the health status of them, we also can connect to the instances using ICE, by doing this we make sure to get into the instances safely and check the logs of them, also the system could be improved a lot in the future due to ASG and the flexibility to adjust the instances and the number of them as needed.

2.2 Security

To make sure that the instances and the DB are secure, we made sure that they are inside private subnets and could be only accessed by ALB in a public instance with connecting to IG and ICE which is located inside private instance.

2.3 Reliability

We made sure that the system is reliable by having an ASG and distributing the system over 2 different availability zones A and C in Ireland region, the ASG will make sure to launch 2 instances as minimum if any of them become unhealthy, also the EFS are distributed in two different availability zones which will make the application more reliable.

1.1 Performance Efficiency

As a future plane, the ASG will be able to launch more instances once the demand increase by holding the AVG CPU usage under 70%, it will increase them as 3 maximums in favor to stay within the cost limit.

2.4 Cost Optimization

To maintain the cost reasonable to the project, we made sure that the RDS is only deployed in on Az instead of multiple AZ's in favor to be within the cost limit, we also made sure to keep the instances and the EFS size small and within the free tier.

The infrastructure is managed via Auto Scaling Groups, 24/7 human monitoring is not assumed, also the instances needed to be accessed securely without being exposed to the public and that's why we used ICE, also the expected traffic is low and the ASG will do the job and finally we made sure to stay within the cost limit by not enabling RDS with multiple AZ's.

Practical Implementation

1.1 Overview of the 3-Tier Web Application

The solution was deployed using a Three-Tier Architecture:

1. Web Tier – Public-facing Application Load Balancer (ALB) distributes incoming HTTP/HTTPS requests across EC2 instances.
2. Application Tier – EC2 instances (managed by an Auto Scaling Group) host the WordPress application.

3. Database Tier – A managed Amazon RDS MySQL instance provides persistent data storage for WordPress.

To ensure consistency of uploaded content across multiple EC2 instances, Amazon EFS was mounted as a shared file system for WordPress.

The design is spread across two Availability Zones to improve resilience:

- Public Subnets (AZ1 & AZ2): ALB and NAT Gateways.
- Private Subnet (AZ1): EC2 instance(s) and RDS.
- Private Subnet (AZ2): EC2 instance(s) for redundancy.
- Amazon EFS: Accessible from both private subnets.

1.2 Functional Requirements Coverage

- Internet **Accessibility**: Achieved via the ALB in public subnets.
- Media **Upload & Retrieval**: WordPress uploads are stored on **EFS**, ensuring shared access across EC2 instances.
- Database **Persistence**: **Amazon RDS MySQL** stores WordPress data, including posts, users, and settings.

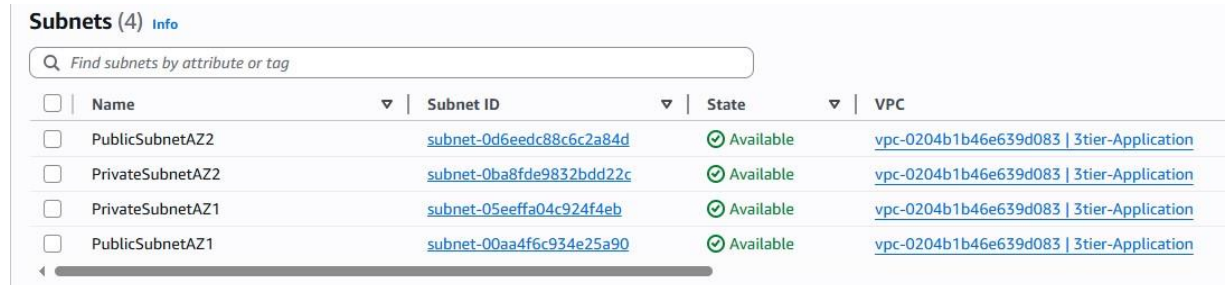
1.3 Non-Functional Requirements Coverage

- **Reliability**: Multi-AZ design with resources spread across AZ1 and AZ2; Auto Scaling ensures replacement of failed instances.
- **Security**:
 - No public SSH; access via AWS Instance Connect Endpoint (ICE).
 - Security Groups restrict traffic to required ports only (80/443 for ALB, 3306 restricted to private subnets).
- **Performance Efficiency**: ASG automatically scales EC2 capacity based on demand.
- **Operational Excellence**: Logging and monitoring can be enabled with CloudWatch for EC2, ALB, and RDS.
- **Cost Optimization**: Free Tier resources used where possible; RDS Multi-AZ and provisioned IOPS not enabled to save costs.

1.4 Deployment steps

1. VPC Setup

- Created VPC with 4 subnets (2 public, 2 private across 2 AZs).
- Configured Internet Gateway for outbound traffic and NAT Gateways in public subnets for private subnet egress.
- Created route tables associating public and private subnets appropriately.



Subnets (4) [Info](#)

Find subnets by attribute or tag

<input type="checkbox"/>	Name	Subnet ID	State	VPC
<input type="checkbox"/>	PublicSubnetAZ2	subnet-0d6eedc88c6c2a84d	Available	vpc-0204b1b46e639d083 3tier-Application
<input type="checkbox"/>	PrivateSubnetAZ2	subnet-0ba8fde98332bdd22c	Available	vpc-0204b1b46e639d083 3tier-Application
<input type="checkbox"/>	PrivateSubnetAZ1	subnet-05eeffa04c924f4eb	Available	vpc-0204b1b46e639d083 3tier-Application
<input type="checkbox"/>	PublicSubnetAZ1	subnet-00aa4f6c934e25a90	Available	vpc-0204b1b46e639d083 3tier-Application

Figure 1: VPC Setup

2. Networking & Security

- Security Group for ALB (allow HTTP/HTTPS from 0.0.0.0/0).
- Security Groups for EC2 (allow inbound only from ALB, outbound to RDS).
- Security Group for RDS (allow inbound only from EC2 Security Group).
- Security Group for ICE (allow outbound to VPC CIDR).
- Security Group for EFS (allow inbound from SSH-SG, EFS-SG and WEB-SG)
- Security Group for SSH (allow inbound SSH from ICE-SG)

3. Compute Layer (EC2 + ASG)

- Launch Template with **User Data** to install Apache, PHP, MySQL client, configure EFS mount, and deploy WordPress.
- Configured **Auto Scaling Group** to span private subnets in AZ1 and AZ2.
- Attached ASG to ALB for traffic distribution.

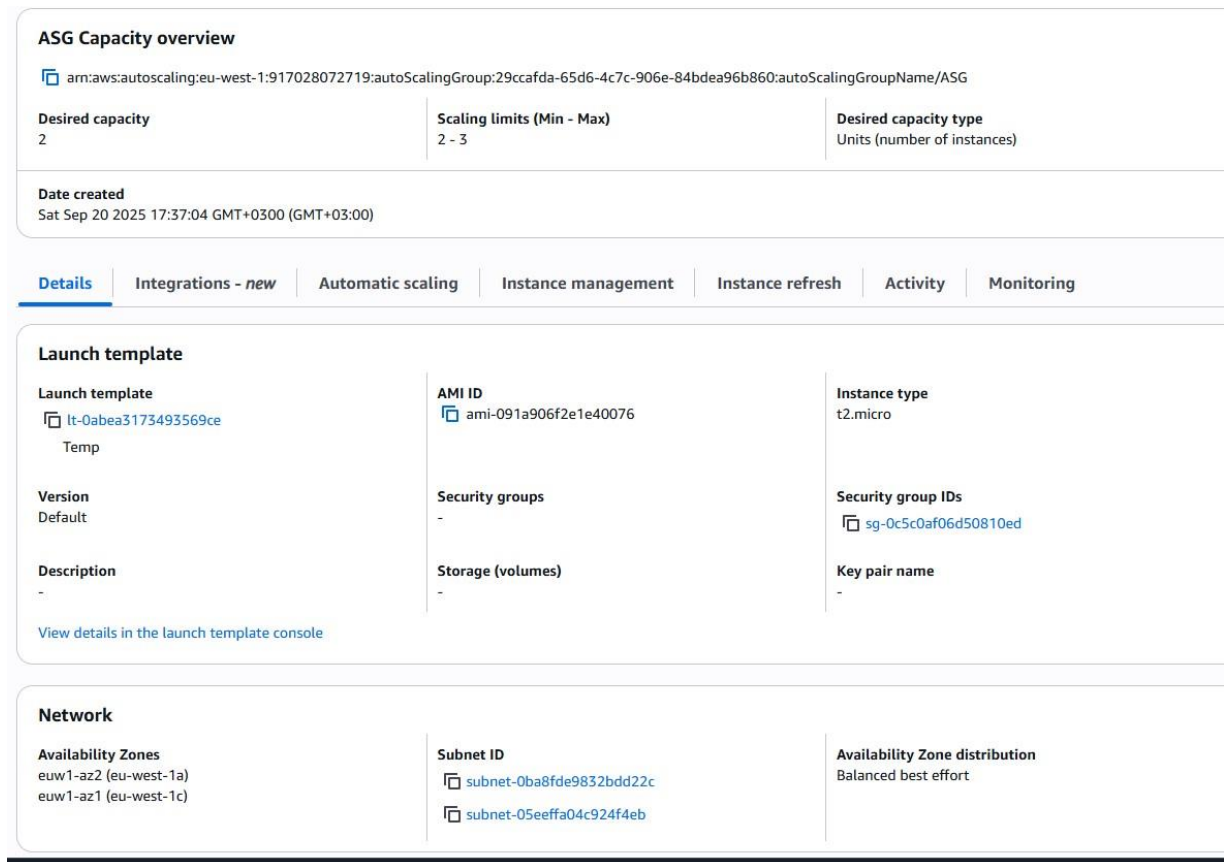


Figure 2: Auto Scaling Group

4. Storage Layer (EFS + RDS)

- Created Amazon EFS and mounted it on EC2 instances at /var/www/html.
- Created Amazon RDS MySQL in private subnet (single AZ).

Look at figure 3 below.

database-1

Summary

DB identifier database-1	Status Available	Role Instance	Engine MySQL Community
CPU 3.26%	Class db.t3.micro	Current activity 0 Connections	Region & AZ eu-west-1a

[Connectivity & security](#) | [Monitoring](#) | [Logs & events](#) | [Configuration](#) | [Zero-ETL integrations](#) | [Maintenance & backups](#) | [Data migrations - new](#)

Connectivity & security

Endpoint & port	Networking	Security
Endpoint database-1.cfcqc0k0ww8j.eu-west-1.rds.amazonaws.com	Availability Zone eu-west-1a	VPC security groups rds-ec2-1 (sg-0092b4214cd9f1e94) Active default (sg-0ecbcf5fcf964d0b1) Active
Port 3306	VPC 3tier-Application (vpc-0204b1b46e639d083)	Publicly accessible No
	Subnet group db subnet	Certificate authority Info rds-ca-rsa2048-g1
	Subnets subnet-0ba8fde9832bdd22c subnet-05eeffa04c924f4eb	Certificate authority date May 20, 2061, 20:49 (UTC+03:00)
	Network type IPv4	DB instance certificate expiration date September 19, 2026, 22:15 (UTC+03:00)

Figure 3: RDS

5. WordPress Deployment

- WordPress downloaded and extracted via script.
- Configured wp-config.php to connect to RDS.
- Permissions applied to ensure Apache and EC2-user access.

1.5 Key Script Used (User Data)

The following script was used in the EC2 Launch Template to automate the setup of Apache, PHP, MySQL client, and WordPress:

```
#!/bin/bash
# update the software packages on the ec2 instance
sudo yum update -y
# install the apache web server, enable it to start on boot, and then start the server immediately
sudo yum install -y git httpd
sudo systemctl enable httpd
sudo systemctl start httpd
# install php 8 along with several necessary extensions for wordpress to run
sudo yum install -y \
php \
php-cli \
php-cgi \
php-curl \
php-mbstring \
php-gd \
php-mysqlnd \
php-gettext \
php-json \
php-xml \
php-fpm \
php-intl \
php-zip \
php-bcmath \
php-ctype \
php-fileinfo \
php-openssl \
php-pdo \
php-tokenizer
# Install Mysql-Client
sudo wget https://dev.mysql.com/get/mysql80-community-release-el9-1.noarch.rpm
sudo dnf install mysql80-community-release-el9-1.noarch.rpm -y
sudo rpm --import https://repo.mysql.com/RPM-GPG-KEY-mysql-2023
sudo dnf repolist enabled | grep "mysql.*-community.*"
sudo dnf install -y mysql-community-server
# start and enable the mysql server
sudo systemctl start mysqld
sudo systemctl enable mysqld
# environment variable
EFS_DNS_NAME=fs-0010eb9b6121b2db3.efs.eu-west-2.amazonaws.com
# mount the efs to the html directory
echo "$EFS_DNS_NAME:/var/www/html nfs4 nfsvers=4.1,rsize=1048576,wsz=1048576,hard,timeo=600,retrans=2 0 0" | sudo tee -a /etc/fstab
sudo mount -a
# set permissions
sudo chown apache:apache -R /var/www/html
sudo chmod 755 -R /var/www/html
# restart the webserver
sudo systemctl restart httpd
sudo systemctl restart php-fpm
```

Figure 4: User Data

Key Design Decisions

Several design choices were made to balance cost, scalability, security, and resilience while meeting the requirements of the project. The key decisions include:

1. Database Selection (Amazon RDS – MySQL)

RDS (MySQL engine) was selected instead of a self-managed database on EC2 to reduce operational overhead (backups, patching, maintenance). A single-AZ RDS

instance was chosen due to cost constraints, accepting reduced redundancy for this demo/training environment.

2. Compute Layer (EC2 + Auto Scaling Group)

EC2 was chosen over serverless (e.g., Lambda) because WordPress is a stateful, PHP/MySQL-based application that benefits from traditional server hosting.

An Auto Scaling Group (ASG) was implemented to provide elasticity and redundancy across multiple Availability Zones.

3. File Storage (Amazon EFS)

EFS was selected to provide shared storage for WordPress uploads, ensuring consistency across multiple EC2 instances.

This prevents issues with media files being tied to a single instance's local storage.

4. Networking & Security

VPC with four subnets (2 public, 2 private) was designed to isolate components.

Public subnets host ALB and NAT Gateways, while private subnets host EC2, RDS, and EFS, reducing exposure to the internet.

AWS Instance Connect Endpoint (ICE) was used instead of SSH to access EC2 instances, eliminating the need to open port 22 to the public internet.

5. Cost Optimization

Free Tier-eligible EC2 (t2.micro/t3.micro) and RDS instances were selected to minimize cost. Single-AZ RDS and General Purpose EFS storage class were chosen instead of more expensive options (Multi-AZ, Provisioned IOPS, or One-Zone EFS).

6. Load Balancing (ALB)

An Application Load Balancer (ALB) was chosen instead of a Network Load Balancer (NLB) because WordPress requires HTTP/HTTPS layer-7 routing rather than just TCP forwarding.

7. Automation (User Data Scripts)

EC2 setup was automated with User Data scripts, ensuring that instances scale consistently and are immediately ready to serve WordPress traffic after launch. This reduces manual setup effort and ensures reproducibility.

Challenges Faced

During the implementation, several challenges were encountered and resolved as follows:

1. 502 Bad Gateway Error (Response Timeout)

- **Issue:** The ALB returned a 502 Bad Gateway error because the EC2 instances could not properly connect to the RDS database. This was caused by incorrect Security Group configuration on the RDS instance.
- **Resolution:** The RDS Security Group was updated to allow inbound traffic on port 3306 (MySQL) from the EC2 instances' Security Group. Once corrected, WordPress was able to connect to the database successfully, and the 502 error was resolved.

2. EFS Not Mounting Properly

- **Issue:** Initially, **Amazon EFS** was not mounting correctly to the `/var/www/html` directory. This was due to missing or misconfigured mount options and required utilities.
- **Resolution:** Installed the required `amazon-efs-utils` / `nfs-utils` package and used the correct DNS name and mount options in the User Data script. After this change, the EFS filesystem mounted successfully on all EC2 instances.

3. EFS Mounted but Showing No Data

- **Issue:** After successfully mounting EFS, the WordPress files did not appear in the mounted directory. The EC2 instance was pointing to an empty EFS volume while WordPress files were placed on the local instance storage.
- **Resolution:** WordPress files were re-downloaded and extracted directly into the mounted EFS directory (`/var/www/html`) to ensure all instances share the same content.

4. Security Group Adjustments for ALB–EC2 Communication

- **Issue:** Even after resolving database connectivity, the application was not fully loading because the EC2 instances' Security Group only allowed HTTP traffic from the ALB. Some WordPress assets and responses were being blocked.

- **Resolution:** The EC2 Security Group inbound rules were updated to allow both **HTTP (port 80)** and **HTTPS (port 443)** traffic coming from the ALB's Security Group. This ensured proper communication and content delivery.

Demo & Access Details

The demo will be shown separately throughout PowerPoint Presentation, the website could be accessed throughout ALB Dns: ALB-930736603.eu-west-1.elb.amazonaws.com.
also the instances could be access throughout the AWS with ICE.

Future Work & Conclusion

This project successfully demonstrated the deployment of a scalable, secure, and resilient WordPress application on AWS using a three-tier architecture. By leveraging services such as VPC, ALB, EC2 with Auto Scaling, RDS, and EFS, the solution met the functional and non-functional requirements while staying within budget constraints. The environment is fault-tolerant across two Availability Zones, supports dynamic scaling, and ensures that application data is persistent and consistent.

While the current implementation fulfills the project requirements, several enhancements can be introduced in the future to further improve reliability, security, and operational excellence:

1. Auto Scaling Notifications

- Integrate the Auto Scaling Group with Amazon SNS to send email notifications when instances are launched or terminated. This will improve operational visibility.

2. Database High Availability

- Enable Multi-AZ deployment for RDS to improve availability and durability in case of an Availability Zone failure.

3. Global Content Delivery and Security

- Add Amazon CloudFront as a Content Delivery Network (CDN) to cache content closer to users and reduce latency.
- Integrate AWS Route 53 for DNS management, SSL/TLS termination (HTTPS), and to improve resilience against DDoS attacks.
-

4. **Enhanced Security**

- Incorporate AWS WAF (Web Application Firewall) with the ALB to protect against common web exploits.
- Use AWS Secrets Manager or SSM Parameter Store for secure database credential management instead of manual configuration.

By implementing these future improvements, the WordPress deployment will evolve into a more production-ready environment with stronger reliability, security, and global performance.