Sardar Noor Ul Hassan
Cloud Intern

WordPress Deployment on ECS Fargate + RDS MySQL Using Terraform (Modular + Public Subnets Only)

Contents

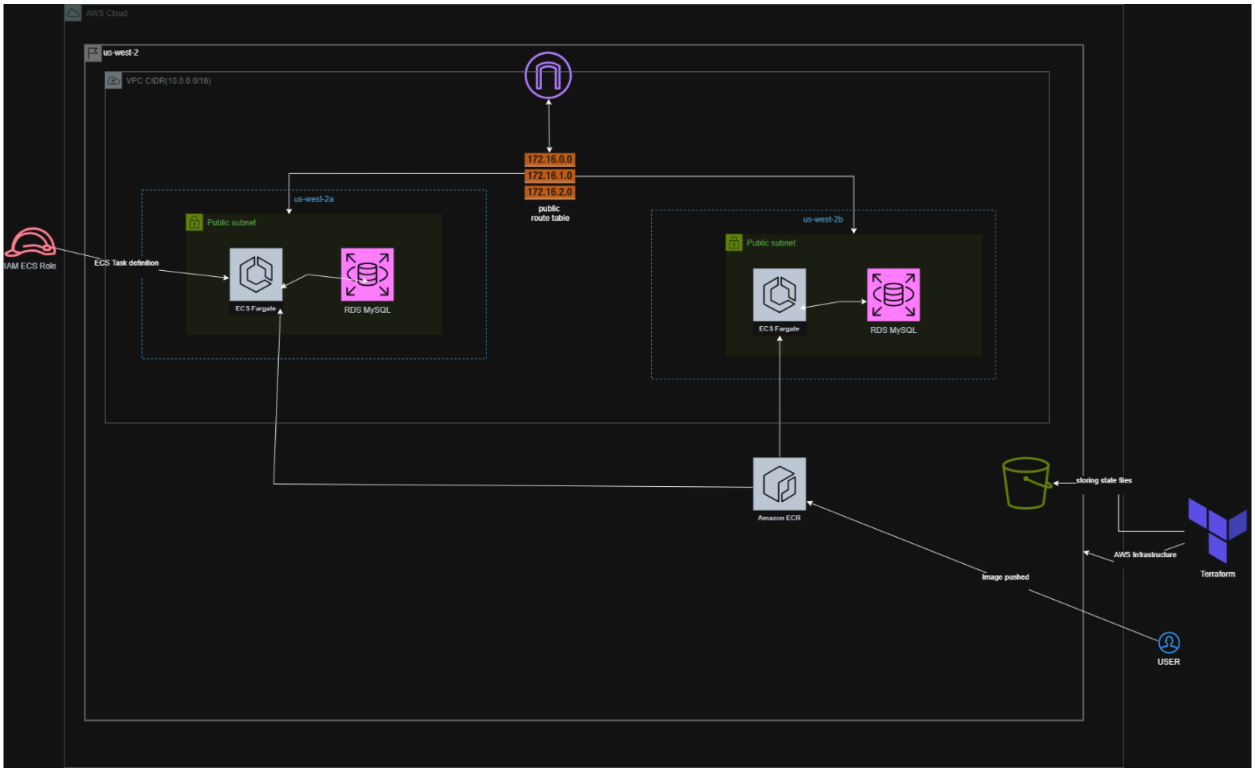
1. Project Overview

This project implements a complete cloud-native deployment of **WordPress** using:

- Amazon **ECS Fargate** (serverless containers)
- Amazon **RDS MySQL** as backend database
- A **Custom VPC** with only **public subnets**
- **Terraform Modular Architecture** for clean, reusable code
- **S3 Remote Backend** for Terraform state management

This entire setup was designed, implemented, and tested end-to-end, and the final result is a fully accessible **WordPress website**, deployed through Infrastructure as Code.

2. Architecture Diagram



The architecture consists of the following components:

Networking

- 1 VPC (CIDR: 10.0.0.0/16)
- 2 Public Subnets across 2 Availability Zones
- Internet Gateway
- Public Route Table → Default route to IGW
- No NAT Gateway (as per requirement)
- ECS tasks directly receive Public IPs

Security

- WordPress SG → Allow HTTP (80)
- RDS SG → Allow MySQL (3306) only from WordPress SG

Compute

- ECS Cluster
- Fargate Task Definition
- ECS Service running WordPress container

Database

- RDS MySQL 8.0 instance
- Stored in public subnets (training setup)
- DB subnet group
- Strong username & password

IAM

- Execution Role for ECS (pull images, write logs, authenticate)

Terraform

- Fully modular architecture
- S3 backend for state consistency

Task 1:Define a Custom VPC with Public Subnets

What was done:

I created a custom VPC using Terraform with the CIDR block **10.0.0.0/16**, which provides 65,536 IPs and enough room for future subnets.

Inside this VPC, I provisioned **two public subnets**, each located in a different Availability Zone, such as:

- us-west-2a → 10.0.1.0/24
- us-west-2b → 10.0.2.0/24

The subnets were marked **map_public_ip_on_launch = true**, ensuring any EC2/ECS resources launched inside automatically receive a public IP.

I attached an **Internet Gateway** to the VPC, and a public route table was created with:

0.0.0.0/0 → Internet Gateway

Each public subnet was associated with this route table so that ECS containers could reach the internet and users could reach the container.

Why this is important:

- ECS Fargate tasks need access to the internet to pull the WordPress Docker image.
- WordPress requires public access from the browser.
- RDS (training mode) was also deployed in public subnets.
- This design follows the project requirement of using **only public subnets** and avoiding NAT Gateways.

Task 2 :Create a Security Group to Allow HTTP (Port 80)

What was done:

I created the WordPress Security Group that:

- Allows inbound **HTTP (80)** from 0.0.0.0/0
- Allows outbound traffic to RDS MySQL port (3306)

This SG was attached to the ECS Service so the running WordPress container can serve web traffic.

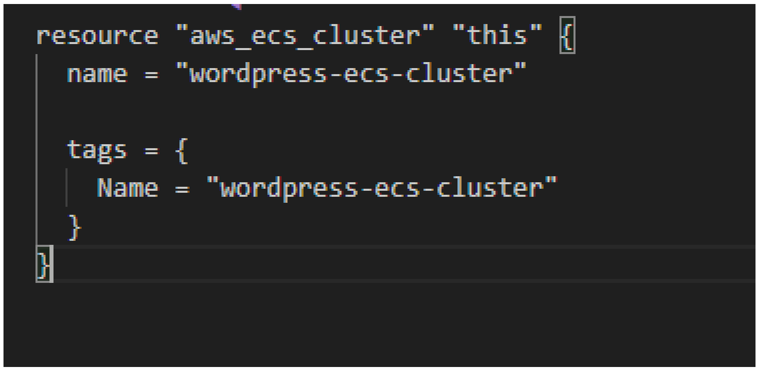
Why this matters:

- WordPress runs on Apache/PHP on port 80.
- Without this SG, users cannot reach the site.
- Outbound rules ensure WordPress can communicate with RDS correctly, which is mandatory for installation.

Task 3 :Create an ECS Cluster Using Terraform

What was done:

I created an **ECS Cluster** using Terraform with no EC2 capacity, because Fargate is serverless and does not require worker nodes.



```
resource "aws_ecs_cluster" "this" {
  name = "wordpress-ecs-cluster"

  tags = {
    Name = "wordpress-ecs-cluster"
  }
}
```

Why this matters:

- ECS cluster is the logical environment where tasks run.
- Fargate provides compute capacity automatically.
- It simplifies management since no EC2 nodes are required.

Task 4 :Write a Task Definition for the WordPress Container

What was done:

I defined a Task Definition that included:

- WordPress official Docker image: wordpress:latest

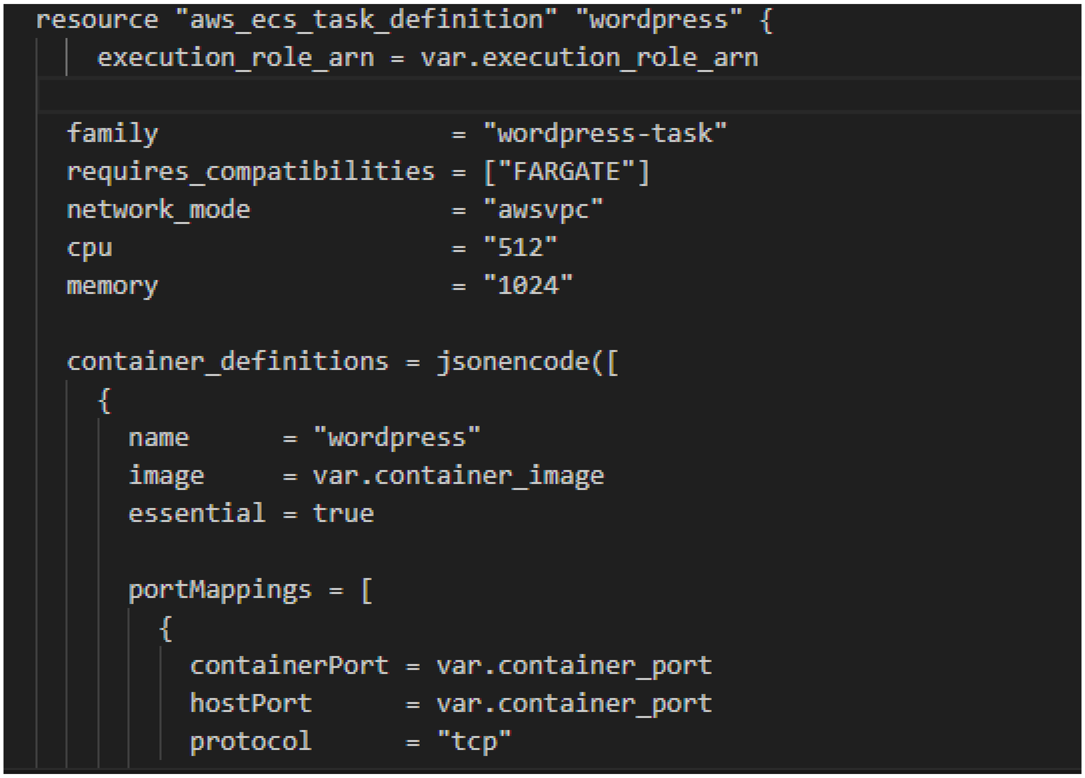
- Port mapping: 80:80
- Memory/CPU configuration
- Network mode: awsvpc (required for Fargate)
- Environment variables for DB connectivity:

WORDPRESS_DB_HOST = <RDS endpoint>

WORDPRESS_DB_USER = wpuser

WORDPRESS_DB_PASSWORD = <password>

WORDPRESS_DB_NAME = wordpress

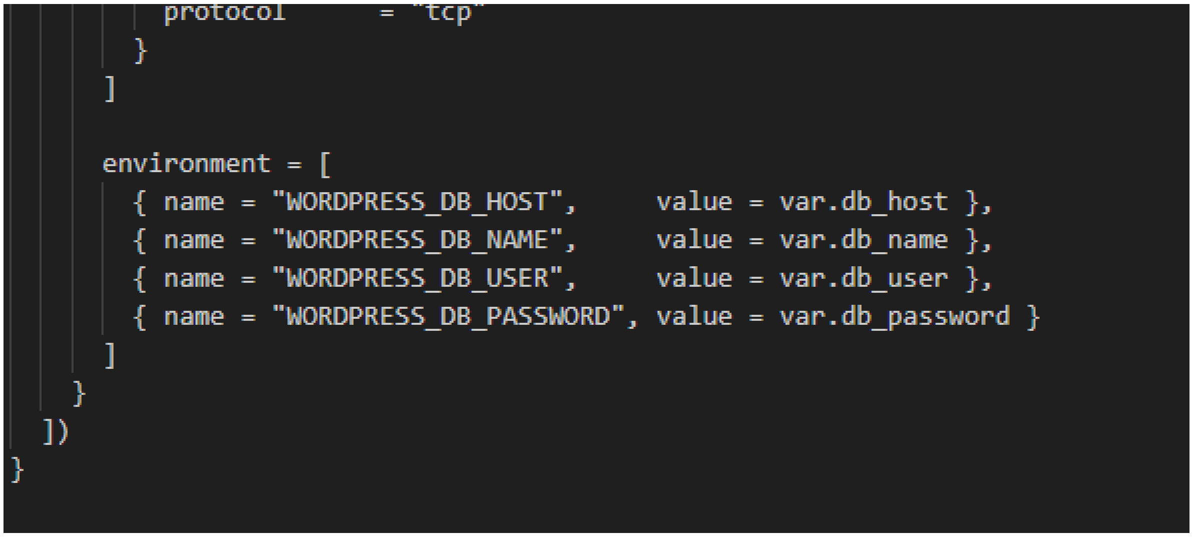


```
resource "aws_ecs_task_definition" "wordpress" {
    execution_role_arn = var.execution_role_arn

  family                   = "wordpress-task"
  requires_compatibilities = ["FARGATE"]
  network_mode             = "awsvpc"
  cpu                      = "512"
  memory                   = "1024"

  container_definitions = jsonencode([
    {
      name      = "wordpress"
      image     = var.container_image
      essential = true

      portMappings = [
        {
          containerPort = var.container_port
          hostPort      = var.container_port
          protocol      = "tcp"
```



```
          protocol      = "tcp"
        }
      ]

      environment = [
        { name = "WORDPRESS_DB_HOST",     value = var.db_host },
        { name = "WORDPRESS_DB_NAME",     value = var.db_name },
        { name = "WORDPRESS_DB_USER",     value = var.db_user },
        { name = "WORDPRESS_DB_PASSWORD", value = var.db_password }
      ]
    }
  ])
}
```

Why this matters:

WordPress does not run unless it can connect to a database.
Passing environment variables ensures:

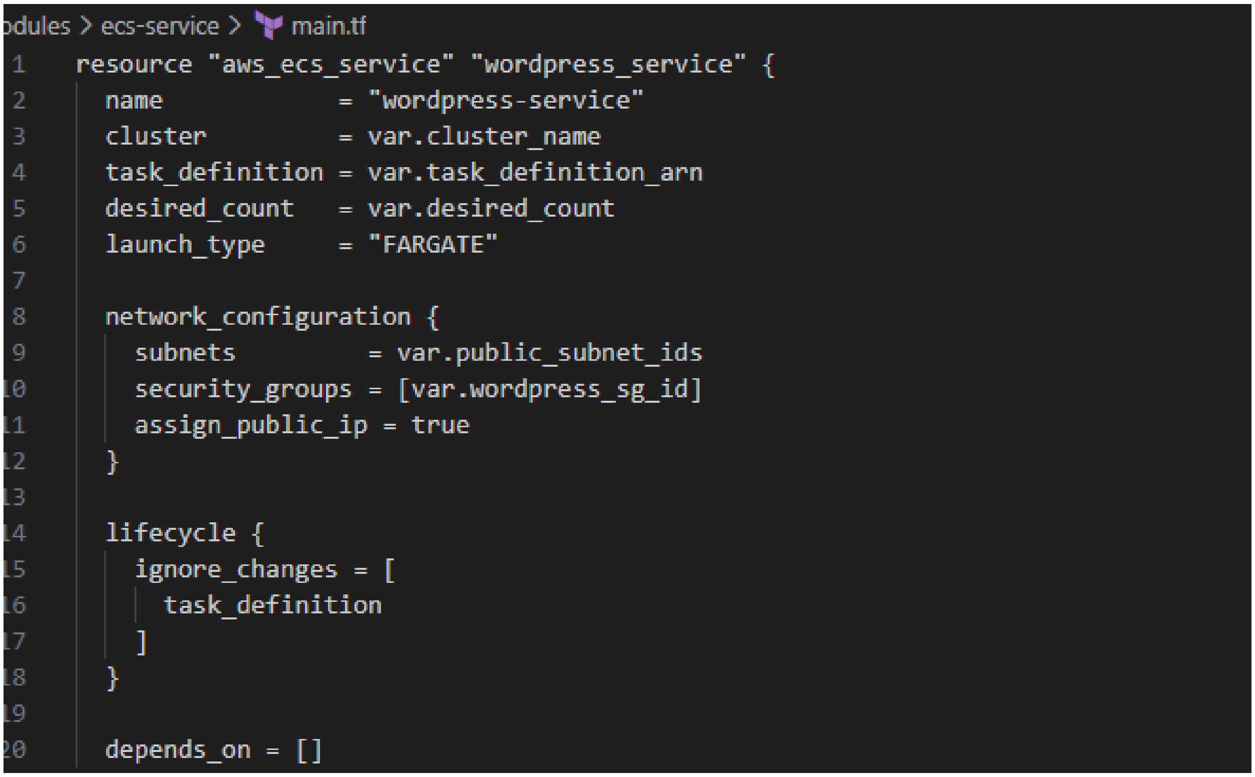
- WordPress knows where the DB is
- The ECS container connects at boot time
- No manual configuration is needed inside the container

Task 5 :Configure an ECS Service Using Fargate Launch Type

What was done:

I created an ECS Service that:

- Runs **1 task** of WordPress
- Uses Fargate as the launch type
- Assigns **Public IP automatically**
- Uses both public subnets
- Attaches WordPress Security Group
- Uses task definition created earlier



odules > ecs-service > main.tf

```
resource "aws_ecs_service" "wordpress_service" {
  name            = "wordpress-service"
  cluster         = var.cluster_name
  task_definition = var.task_definition_arn
  desired_count   = var.desired_count
  launch_type     = "FARGATE"

  network_configuration {
    subnets          = var.public_subnet_ids
    security_groups = [var.wordpress_sg_id]
    assign_public_ip = true
  }

  lifecycle {
    ignore_changes = [
      task_definition
    ]
  }

  depends_on = []
```

Why this matters:

- ECS Service keeps WordPress running even if container fails
- Ensures high availability

- Provides automatic restarts
- Public IP enables browser access directly

Task 6 :Set Up IAM Roles and Execution Policies for ECS Task

What was done:

Created an IAM Role:

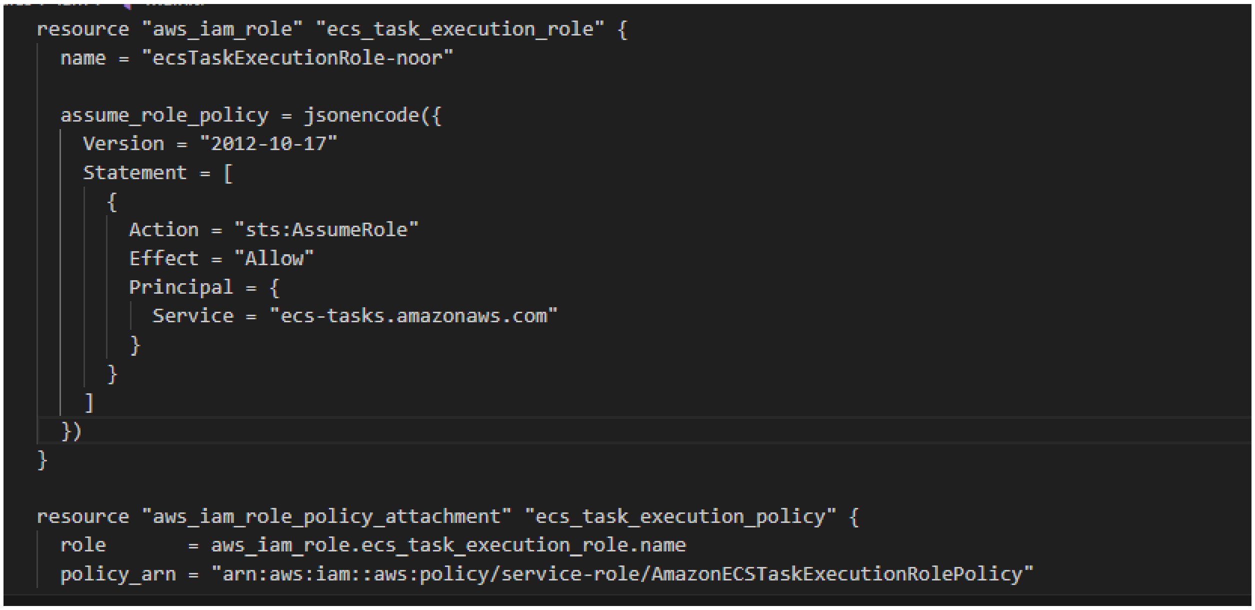
ecsTaskExecutionRole-noor

Attached the policy:

AmazonECSTaskExecutionRolePolicy

This role allows:

- Pulling Docker images from public ECR
- Writing container logs to CloudWatch
- Managing task metadata



```
resource "aws_iam_role" "ecs_task_execution_role" {
  name = "ecsTaskExecutionRole-noor"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Action = "sts:AssumeRole"
        Effect = "Allow"
        Principal = {
          Service = "ecs-tasks.amazonaws.com"
        }
      }
    ]
  })
}

resource "aws_iam_role_policy_attachment" "ecs_task_execution_policy" {
  role       = aws_iam_role.ecs_task_execution_role.name
  policy_arn = "arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy"
```

Why this matters:

Without this IAM role, ECS cannot:

- Download WordPress image

- Run task
- Authenticate
- Register tasks

This role is mandatory for Fargate tasks.

Task 7 :Attach Security Groups to the ECS Service

What was done:

Attached two security groups:

- WordPress SG → Inbound HTTP allowed
- RDS SG → Allows MySQL only from WordPress SG

Why this matters:

This enforces strict communication:

- The public can reach WordPress
- Only WordPress can reach the RDS database
- Database remains protected (no public DB access)

This follows AWS best practices.

Task 8 :Deploy WordPress Container Using Terraform

What was done:

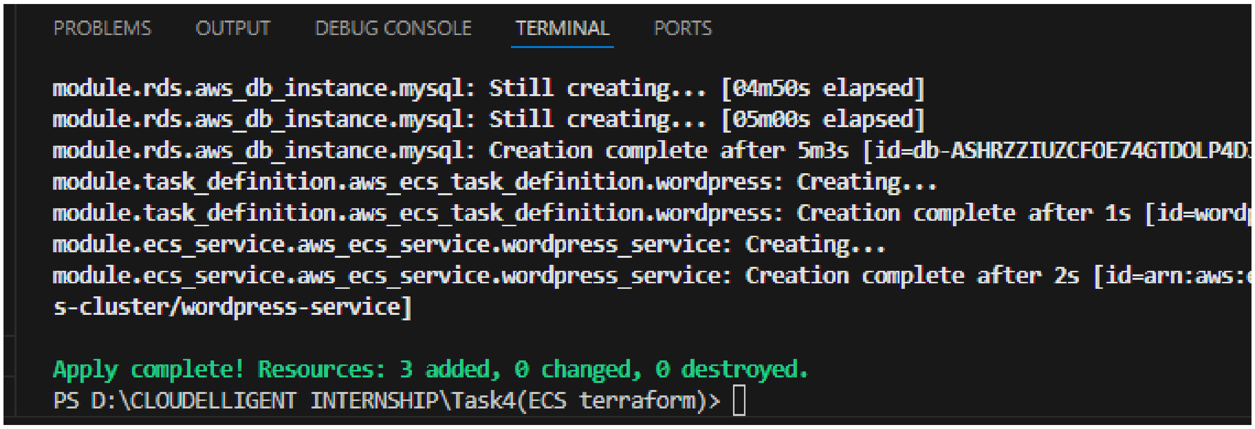
Ran these commands:

```
terraform init

terraform plan

terraform apply
```

Terraform deployed every component in a modular, organized manner.



```
module.rds.aws_db_instance.mysql: Still creating... [04m50s elapsed]
module.rds.aws_db_instance.mysql: Still creating... [05m00s elapsed]
module.rds.aws_db_instance.mysql: Creation complete after 5m3s [id=db-ASHRZZIUZCFOE74GTDOLP4D
module.task_definition.aws_ecs_task_definition.wordpress: Creating...
module.task_definition.aws_ecs_task_definition.wordpress: Creation complete after 1s [id=word
module.ecs_service.aws_ecs_service.wordpress_service: Creating...
module.ecs_service.aws_ecs_service.wordpress_service: Creation complete after 2s [id=arn:aws:
s-cluster/wordpress-service]

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.
PS D:\CLOUDELLIGENT INTERNSHIP\Task4(ECS terraform)>
```

Why this matters:

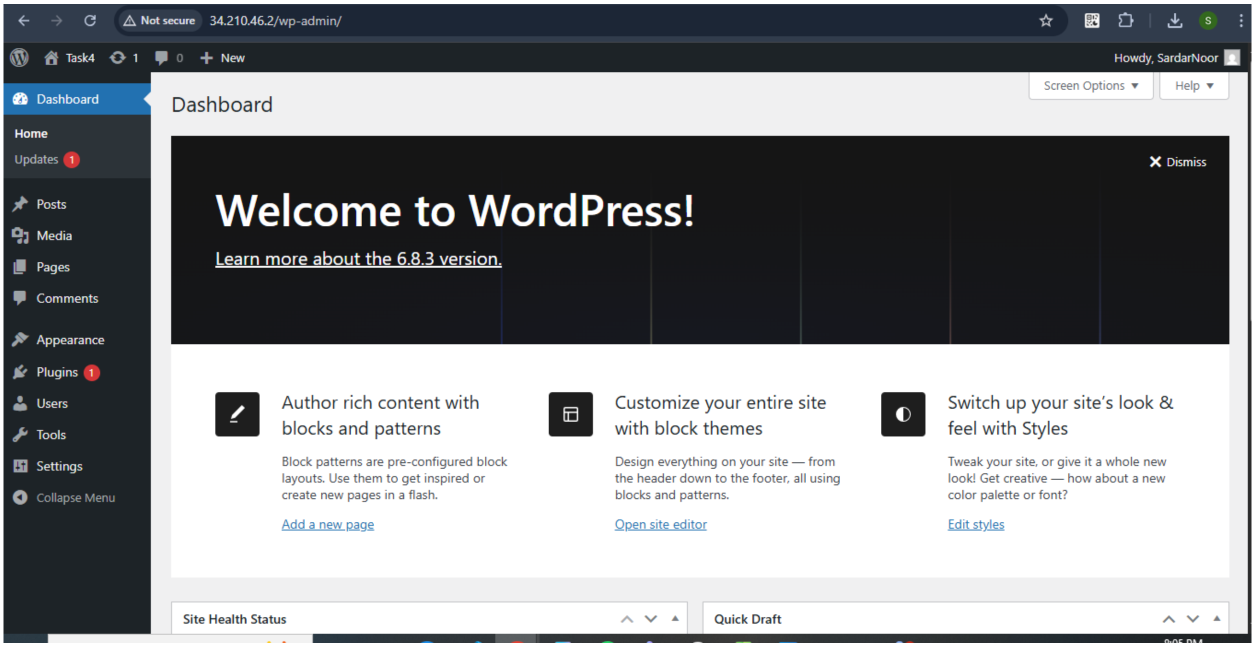
Infrastructure as Code ensures:

- Fully automated provisioning
- Predictable configuration
- Easy replication
- Minimal human error
- Version-controlled infrastructure

Task 9 :Verify WordPress Server is Running and Accessible

What was done:

- Navigated to ECS Task public IP
- WordPress installation page loaded
- Completed setup wizard
- Logged into WordPress admin dashboard



Dashboard

Welcome to WordPress!

Learn more about the 6.8.3 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

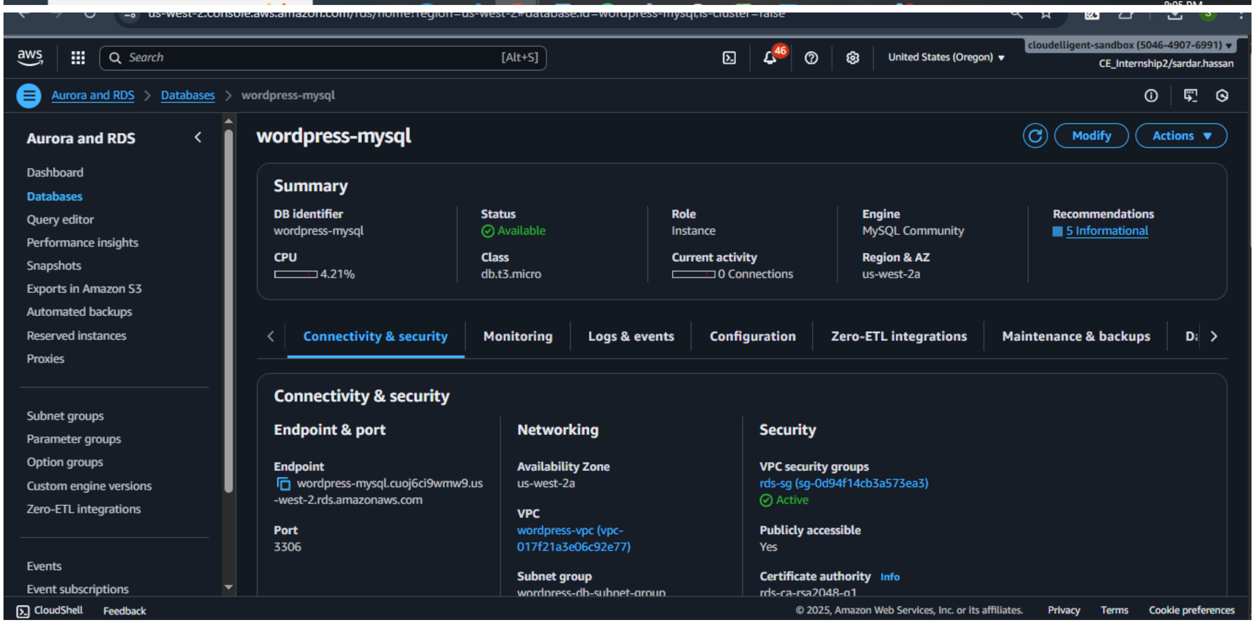
Switch up your site's look & feel with Styles

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

Quick Draft



wordpress-mysql

Summary

DB identifier	Status	Role	Engine	Recommendations
wordpress-mysql	✓ Available	Instance	MySQL Community	5 Informational
CPU	Class	Current activity	Region & AZ	
4.21%	db.t3.micro	0 Connections	us-west-2a	

Connectivity & security

Endpoint & port

Endpoint
wordpress-mysql.cuoj6ci9wmw9.us-west-2.rds.amazonaws.com

Port
3306

Networking

Availability Zone
us-west-2a

VPC
wordpress-vpc (vpc-017f21a3e06c92e77)

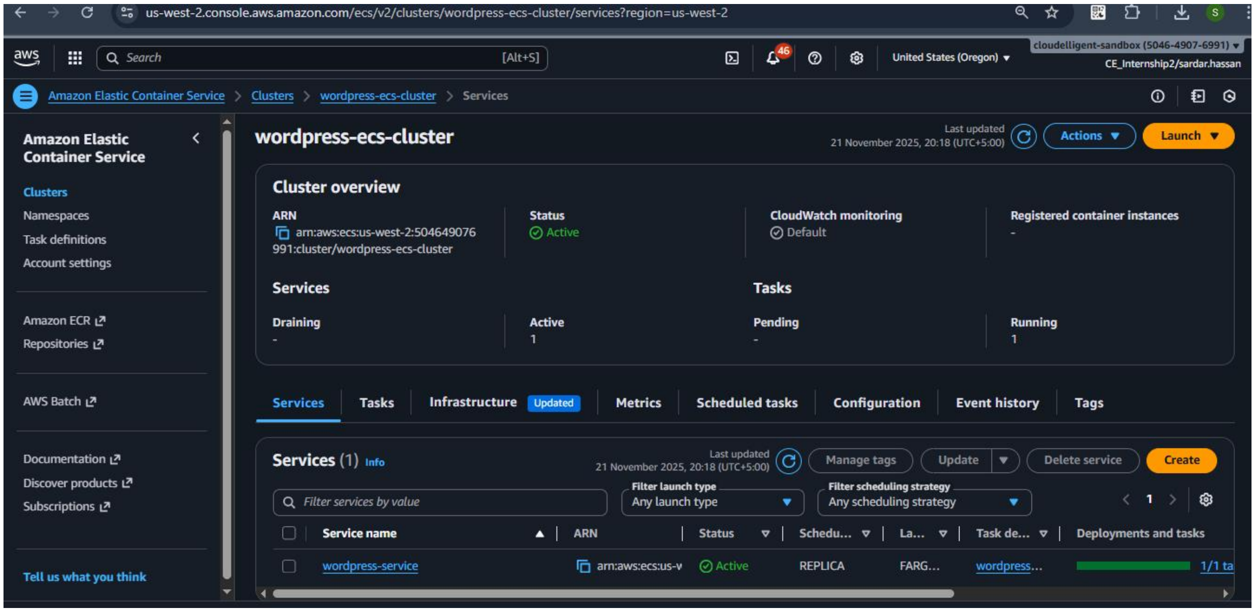
Subnet group
wordpress-db-subnet-group

Security

VPC security groups
rds-sg (sg-0d94f14cb3a573ea3)
✓ Active

Publicly accessible
Yes

Certificate authority Info
rds-ca-rsa2048-g1



wordpress-ecs-cluster

Last updated
21 November 2025, 20:18 (UTC+5:00)

Cluster overview

ARN: arn:aws:ecs:us-west-2:504649076991:cluster/wordpress-ecs-cluster
Status: Active
CloudWatch monitoring: Default
Registered container instances: -

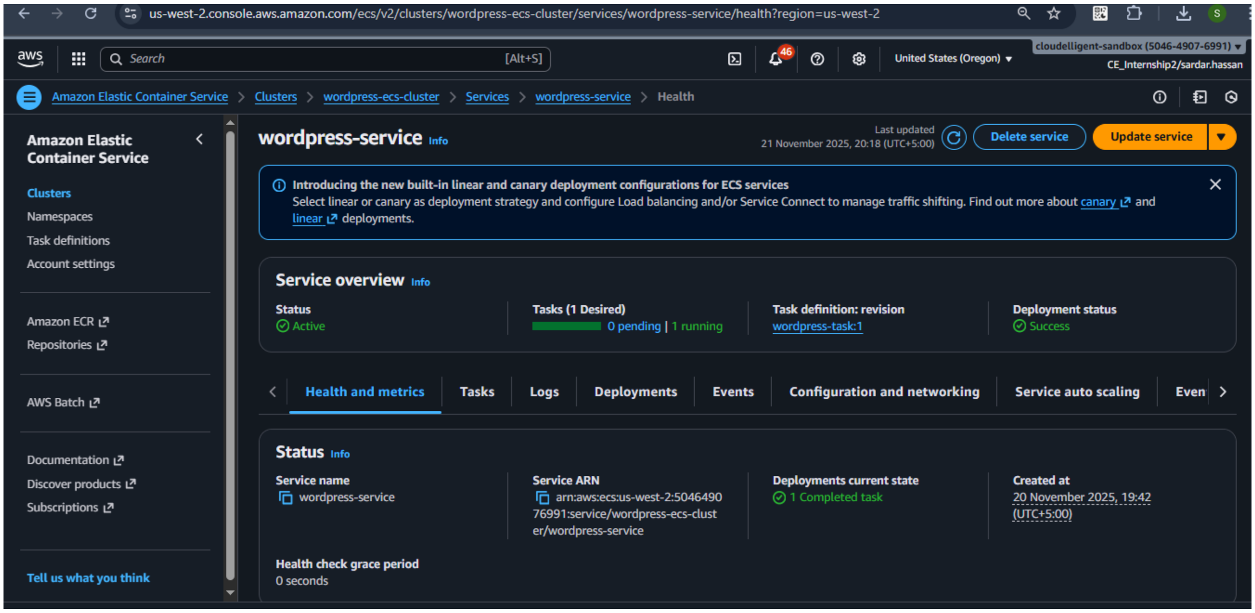
Services — Draining: - | Active: 1
Tasks — Pending: - | Running: 1

Services (1) Info

Last updated 21 November 2025, 20:18 (UTC+5:00)

Filter services by value
Filter launch type: Any launch type
Filter scheduling strategy: Any scheduling strategy

Service name	ARN	Status	Schedu...	La...	Task de...	Deployments and tasks
wordpress-service	arn:aws:ecs:us-v	Active	REPLICA	FARG...	wordpress...	1/1 ta


us-west-2.console.aws.amazon.com/ecs/v2/clusters/wordpress-ecs-cluster/services/wordpress-service/health?region=us-west-2
aws
Search
[Alt+S]
46
United States (Oregon) ▾
cloudelligent-sandbox (5046-4907-6991) ▾
CE_Internship2/sardar.hassan
Amazon Elastic Container Service > Clusters > wordpress-ecs-cluster > Services > wordpress-service > Health
Amazon Elastic Container Service
Clusters
Namespaces
Task definitions
Account settings
Amazon ECR
Repositories
AWS Batch
Documentation
Discover products
Subscriptions
Tell us what you think
Delete service
Update service
Health and metrics
Tasks
Logs
Deployments
Events
Configuration and networking
Service auto scaling
Even

wordpress-service Info

Last updated 21 November 2025, 20:18 (UTC+5:00)

Introducing the new built-in linear and canary deployment configurations for ECS services
Select linear or canary as deployment strategy and configure Load balancing and/or Service Connect to manage traffic shifting. Find out more about canary and linear deployments.

Service overview Info

Status: Active
Tasks (1 Desired): 0 pending | 1 running
Task definition: revision: wordpress-task:1
Deployment status: Success

Status Info

Service name: wordpress-service
Service ARN: arn:aws:ecs:us-west-2:504649076991:service/wordpress-ecs-cluster/wordpress-service
Deployments current state: 1 Completed task
Created at: 20 November 2025, 19:42 (UTC+5:00)
Health check grace period: 0 seconds

Why this proves success:

If WordPress installation appears:

✓ ECS Container running
✓ Port 80 open
✓ Network routing working
✓ RDS DB connected
✓ All environment variables correct
✓ IAM role working

✓ Terraform modules correct

✓ WordPress able to store data in RDS

This is full end-to-end validation.

Challenges Faced

Challenge 1 RDS Permission Denied

Internship1 SSO role did not include RDS creation permissions.

Solution:
Switched to CE_Internship2 identity, which had required permissions.

Challenge 2 Duplicate DB Subnet Group

Terraform threw error "DB Subnet Group Already Exists".

Reason:
RDS subnet group created earlier.

Solution:
Updated Terraform to reference the existing subnet group.

Challenge 3 IAM Role Already Exists

Execution role existed from previous deployments.

Solution:
Renamed role or reused existing role.

Challenge 4 ECS Logs Not Coming

ECS task logs did not appear in console.

Reason:
awslogs driver was not configured.

Solution:
Log verification done through WordPress dashboard + RDS logs.

Conclusion

This project successfully deployed a production-style WordPress system using AWS managed services and Terraform modular IaC structure.
The system is fully functional, publicly accessible, and demonstrates solid cloud engineering practices.