

TASK 4

**ECS Fargate and RDS MySQL
WordPress Deployment with Terraform**

Umar Satti

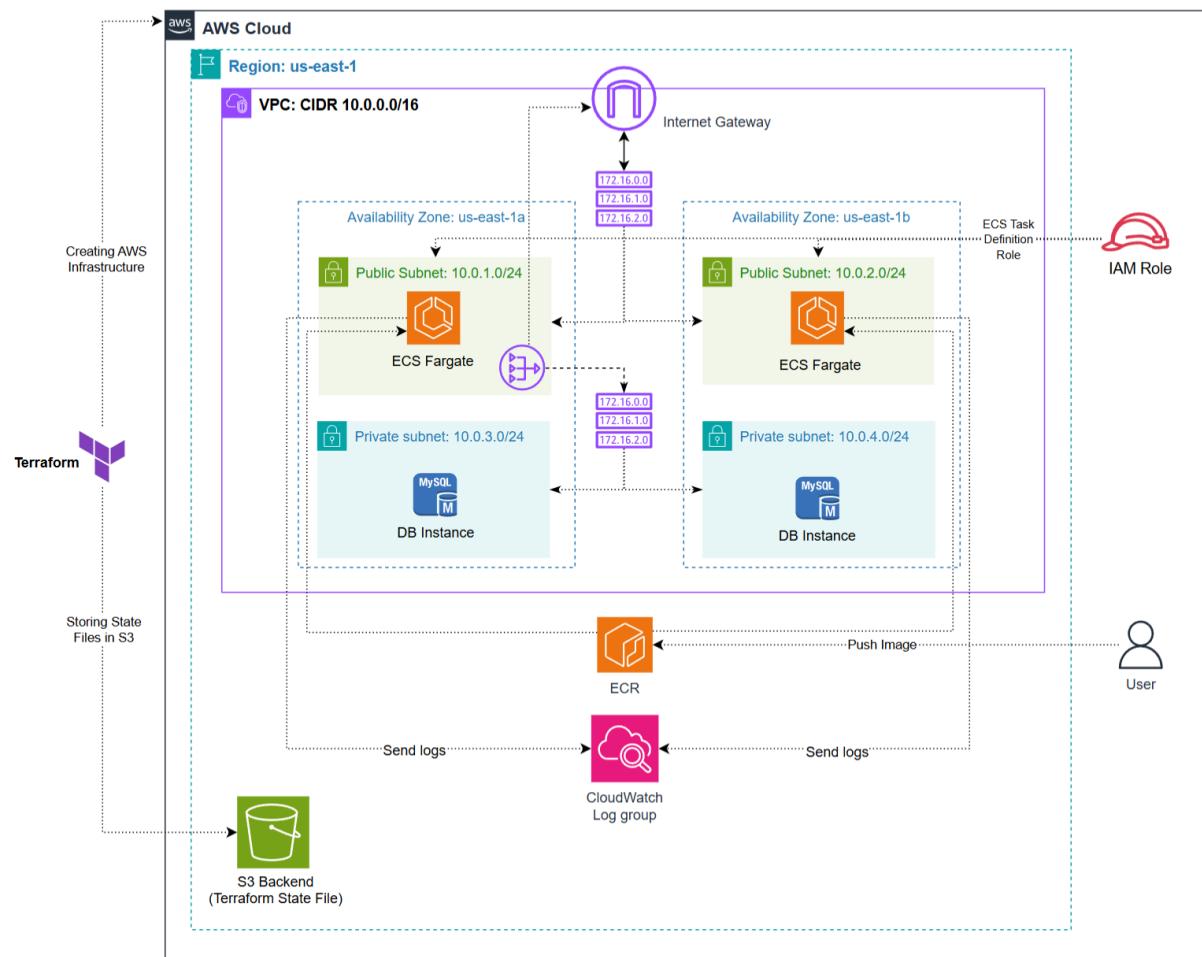
Table of Contents

Task Description	3
Architecture Diagram.....	3
Task 1.1: Create Terraform Project Structure	4
Task 1.2: Create S3 Bucket for Terraform Remote Backend	5
Task 1.3: Root Directory Files.....	7
Task 1.4: Configure VPC Module	12
Task 1.5: Configure Security Group Module.....	19
Task 1.6: Configure Elastic Container Registry (ECR) Module.....	22
Task 1.7: Configure Task Definition Module	24
Task 1.8: Configure Elastic Container Service Module.....	28
Task 1.9: Configure Relation Database Service (RDS) Module.....	30
Task 1.10: Running Terraform Commands to Deploy Infrastructure	32
Task 1.11: Validate Infrastructure Deployment and WordPress Installation.....	34
Task 1.11.1: Verify Deployment of AWS Infrastructure.....	34
Task 1.10.2: Push WordPress Image to ECR.....	45
Task 1.10.3: Access the WordPress Application	48
Task 1.10.4: Configure WordPress	49
Task 1.11: Clean Up	51
Task 1.12: Troubleshooting	52
Conclusion	53

Task Description

Set up a WordPress application running on Amazon ECS Fargate using Terraform. Create a VPC with public and private subnets, multiple security groups, task definition, ECR repository, ECS cluster, ECS service, and RDS MySQL with minimal configurations.

Architecture Diagram



Task 1.1: Create Terraform Project Structure

Steps:

1. Create a `terraform.tf` file in the root directory. This file defines the AWS provider, provider version, AWS region, and the S3 backend for storing Terraform state files.
2. Create an S3 bucket in the same AWS region to store Terraform state files (explained in Task 1.2 below).
3. Create a `main.tf` file in the root directory. This connects modules together, passes outputs from one module to another, and passes variables between them.
4. Create a `variables.tf` file in the root directory. This file defines variables by description and type as key-value pairs. These contain arguments for parameters such as VPC name, CIDR block, ECR URIs, and more.
5. Create an `outputs.tf` file in the root directory. This file exposes important module outputs such as VPC ID, ECR Private Registry URI for Wordpress after deployment.
6. Create a `modules` directory that contains 6 total modules including VPC, Security groups, Task definition, ECR, ECS, and RDS. Each module contains its own `main.tf`, `variables.tf`, and `outputs.tf` files.

The terraform project directory structure looks like this:

```
PS D:\Cloudelligent\Task-4-ECS-Fargate-RDS-WordPress-with-Terraform> tree /f
Folder PATH listing for volume New Volume
Volume serial number is 5A5C-D3FC
D:.
├── .terraform.lock.hcl
├── main.tf
├── outputs.tf
├── terraform.tf
└── terraform.tfvars
└── variables.tf

└── .terraform
    └── terraform.tfstate

└── modules
    └── modules.json

└── providers
    └── registry.terraform.io
        └── hashicorp
            ├── aws
            │   ├── 6.21.0
            │   │   └── windows_amd64
            │   │       └── LICENSE.txt
            │   │       └── terraform-provider-aws_v6.21.0_x5.exe
            └── null
                └── 3.2.4
                    └── windows_amd64
                        └── LICENSE.txt
                        └── terraform-provider-null_v3.2.4_x5.exe

└── modules
    ├── ecr
    │   ├── main.tf
    │   ├── outputs.tf
    │   └── variables.tf
    ├── ecs
    │   ├── main.tf
    │   ├── outputs.tf
    │   └── variables.tf
    ├── rds
    │   ├── main.tf
    │   ├── outputs.tf
    │   └── variables.tf
    ├── sg
    │   ├── main.tf
    │   ├── outputs.tf
    │   └── variables.tf
    ├── task_definition
    │   ├── main.tf
    │   ├── outputs.tf
    │   └── variables.tf
    └── vpc
        ├── main.tf
        ├── outputs.tf
        └── variables.tf
```

Task 1.2: Create S3 Bucket for Terraform Remote Backend

Steps:

1. Log in to the AWS Management Console. Navigate to S3 using the search bar at the top of the console page.
2. Click on **Create Bucket** button.
3. Choose **General Purpose**, add a globally unique bucket name, and make sure the AWS Region is the same as Terraform.
4. Click **Create Bucket**.
5. Update **terraform.tf** file in root directory to reference this S3 bucket in the backend block.

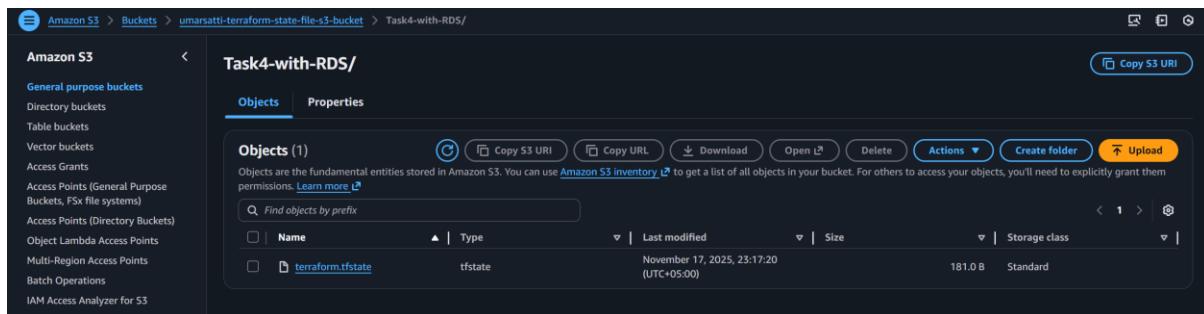
```
backend "s3" {
  bucket      = "umarsatti-terraform-state-file-s3-bucket"
  key         = "task4/terraform.tfstate"
  region      = "us-east-1"
  encrypt     = true
  use_lockfile = true
}
```

Once the S3 bucket is created in the AWS Management Console and referenced in the Terraform backend configuration, Terraform automatically begins storing and versioning state files in this bucket.

In this case, the S3 bucket named **umarsatti-terraform-state-file-s3-bucket** is used as the remote backend, as defined in the **provider.tf** file. The backend block ensures all state information is centralized, secure, and persistent across multiple users or workstations.

The screenshot shown below shows the exact file path inside the S3 bucket:

S3 > Buckets > umarsatti-terraform-state-file-s3-bucket > Task4-with-RDS > terraform.tfstate



This confirms that:

- Terraform successfully initialized the backend and wrote the state file to the S3 bucket.

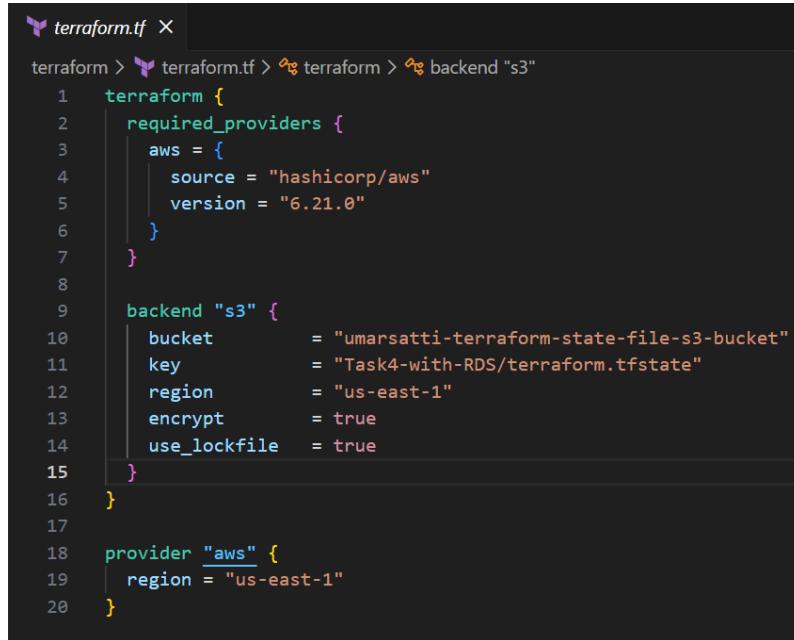
- The **terraform.tfstate** file contains metadata about all deployed AWS resources (VPC, Subnet, EC2, etc.).
- Every terraform plan, apply, or destroy operation reads and updates this file automatically.
- The locking mechanism (enabled by **use_lockfile = true**) ensures that no two processes modify the state simultaneously, preventing state corruption.

Note: The S3 bucket **umarsatti-terraform-state-file-s3-bucket** was already created in the prior projects. However, a new folder was created just for this project named **Task4-with-RDS repository** in which the **terraform.tfstate** file is stored.

Task 1.3: Root Directory Files

This section explains each Terraform configuration file in the root directory and how they work together to deploy the VPC, Security Groups, ECR Repository, RDS Database, ECS Task Definition, and ECS Fargate Service.

terraform.tf:



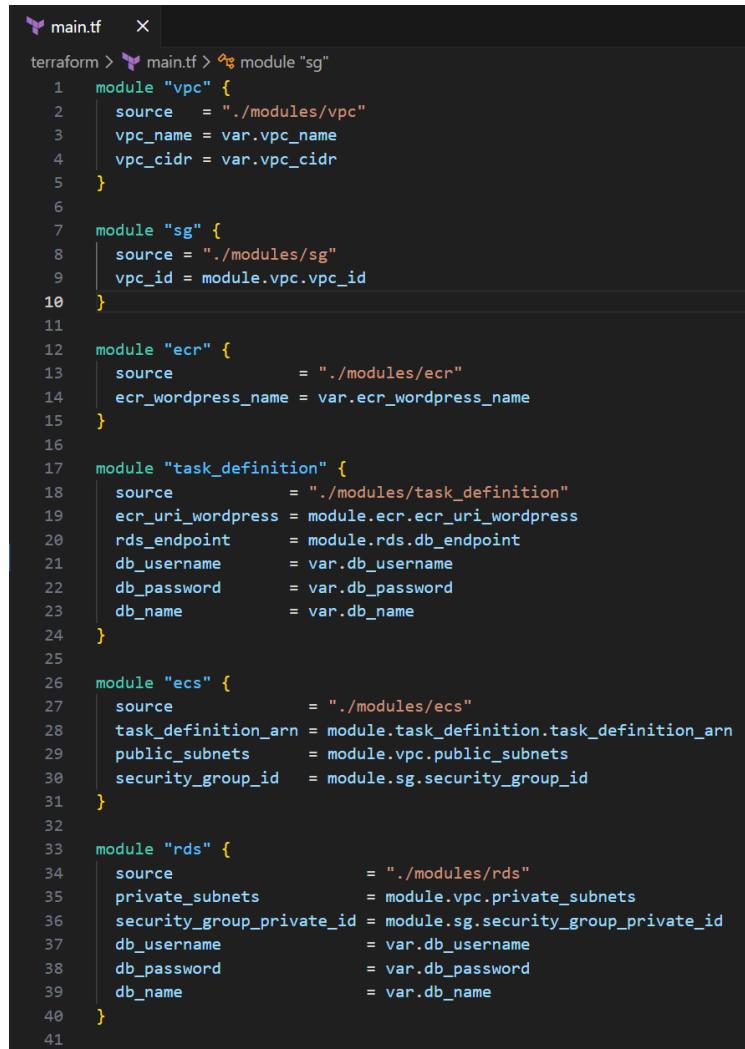
```
terraform > terraform.tf > terraform > backend "s3"
1  terraform {
2    required_providers {
3      aws = {
4        source  = "hashicorp/aws"
5        version = "6.21.0"
6      }
7    }
8
9    backend "s3" {
10      bucket     = "umarsatti-terraform-state-file-s3-bucket"
11      key        = "Task4-with-RDS/terraform.tfstate"
12      region     = "us-east-1"
13      encrypt    = true
14      use_lockfile = true
15    }
16  }
17
18  provider "aws" {
19    region = "us-east-1"
20  }
```

This file configures the AWS provider and sets up the remote backend used to store Terraform state securely in an S3 bucket.

- The `terraform` block specifies the AWS provider version (6.21.0) sourced from HashiCorp.
- The `backend "s3"` block stores the Terraform state file in the bucket **umarsatti-terraform-state-file-s3-bucket** under the path **Task4-with-RDS/terraform.tfstate**.
- Encryption is enabled, and `use_lockfile = true` ensures that only one Terraform process can update the state at a time.
- The `provider "aws"` block defines **us-east-1** as the region where all AWS resources will be created.

Using an S3 backend allows centralized, versioned, and protected state management—ideal for team use or working across multiple machines.

main.tf:



```
1 module "vpc" {
2   source  = "./modules/vpc"
3   vpc_name = var.vpc_name
4   vpc_cidr = var.vpc_cidr
5 }
6
7 module "sg" {
8   source = "./modules/sg"
9   vpc_id = module.vpc.vpc_id
10 }
11
12 module "ecr" {
13   source      = "./modules/ecr"
14   ecr_wordpress_name = var.ecr_wordpress_name
15 }
16
17 module "task_definition" {
18   source      = "./modules/task_definition"
19   ecr_uri_wordpress = module.ecr.ecr_uri_wordpress
20   rds_endpoint = module.rds.db_endpoint
21   db_username  = var.db_username
22   db_password   = var.db_password
23   db_name       = var.db_name
24 }
25
26 module "ecs" {
27   source      = "./modules/ecs"
28   task_definition_arn = module.task_definition.task_definition_arn
29   public_subnets = module.vpc.public_subnets
30   security_group_id = module.sg.security_group_id
31 }
32
33 module "rds" {
34   source      = "./modules/rds"
35   private_subnets = module.vpc.private_subnets
36   security_group_private_id = module.sg.security_group_private_id
37   db_username  = var.db_username
38   db_password   = var.db_password
39   db_name       = var.db_name
40 }
41
```

This file defines how Terraform assembles the entire WordPress + RDS architecture by wiring together all modules.

- **VPC module (./modules/vpc)**
Creates all networking components including the VPC, public and private subnets, internet gateway, and route tables. It uses **vpc_name** and **vpc_cidr** from root variables.
- **Security Group module (./modules/sg)**
Builds both the public security group (for ECS) and private security group (for RDS). It receives the VPC ID from the VPC module to ensure both security groups are associated with the correct network.
- **ECR module (./modules/ecr)**
Creates the private ECR repository used to store the WordPress Docker image. The repository name is taken from the **ecr_wordpress_name** variable.

- **Task Definition module (`./modules/task_definition`)**

Defines the WordPress ECS task that connects to the RDS database. This module uses:

- The ECR WordPress image URI from the ECR module
- RDS endpoint from the RDS module
- Database username, password, and database name from root-level variables

- **ECS module (`./modules/ecs`)**

Deploys the ECS Cluster and ECS Fargate Service using:

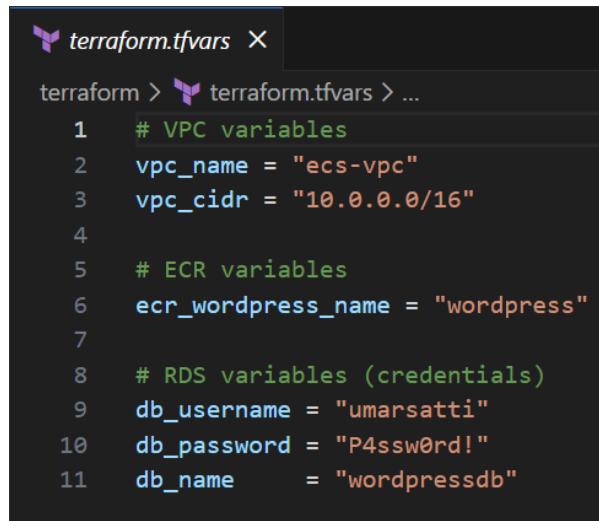
- The Task Definition ARN from the Task Definition module
- Public subnets from the VPC module
- Security group ID from the SG module

- **RDS module (`./modules/rds`)**

Provisions an RDS MySQL database inside the private subnets provided by the VPC module. It uses the private security group to restrict database access only to ECS tasks. Database credentials are passed through variables.

This modular design separates networking, database, security, compute, and container image handling into cleanly isolated units.

terraform.tfvars



A screenshot of a code editor window titled "terraform.tfvars". The code is written in Terraform configuration language. It includes comments and variable assignments for VPC, ECR, and RDS.

```
1 # VPC variables
2 vpc_name = "ecs-vpc"
3 vpc_cidr = "10.0.0.0/16"
4
5 # ECR variables
6 ecr_wordpress_name = "wordpress"
7
8 # RDS variables (credentials)
9 db_username = "umarsatti"
10 db_password = "P4ssw0rd!"
11 db_name      = "wordpressdb"
```

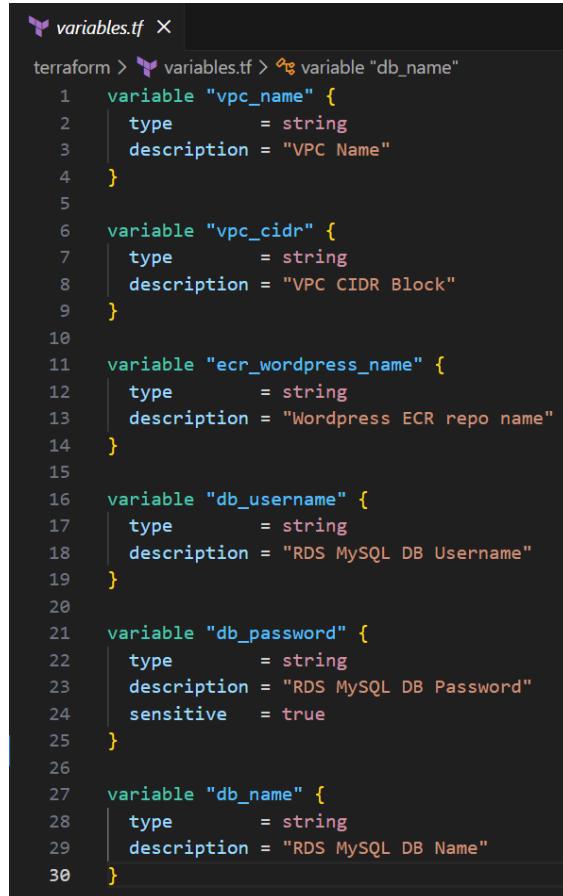
This file contains actual input values that Terraform uses during deployment.

- VPC settings (`vpc_name`, `vpc_cidr`) define the name and network range of the VPC.
- ECR repository name (`ecr_wordpress_name`) specifies the name of the WordPress image repository.

- RDS database credentials (**db_username**, **db_password**, **db_name**) configure the MySQL database that WordPress uses.

Storing values here keeps the configuration clean while allowing easy modification without touching the actual infrastructure code.

variables.tf:



```

variables.tf ×
terraform > variables.tf > variable "db_name"
1   variable "vpc_name" {
2     type      = string
3     description = "VPC Name"
4   }
5
6   variable "vpc_cidr" {
7     type      = string
8     description = "VPC CIDR Block"
9   }
10
11  variable "ecr_wordpress_name" {
12    type      = string
13    description = "Wordpress ECR repo name"
14  }
15
16  variable "db_username" {
17    type      = string
18    description = "RDS MySQL DB Username"
19  }
20
21  variable "db_password" {
22    type      = string
23    description = "RDS MySQL DB Password"
24    sensitive  = true
25  }
26
27  variable "db_name" {
28    type      = string
29    description = "RDS MySQL DB Name"
30  }

```

This file defines all the input variables required by the root module.

- VPC inputs (**vpc_name**, **vpc_cidr**) are used to build the networking layer.
- The **ecr_wordpress_name** variable allows the ECR module to create the WordPress repository dynamically.
- RDS variables (**db_username**, **db_password**, **db_name**) store database credentials. The password is marked **sensitive** so Terraform does not display it in logs or outputs.

This structure keeps the configuration modular and makes it clear what values the user must provide.

outputs.tf:



```
outputs.tf  X
terraform > outputs.tf > output "rds_endpoint"
1   output "vpc_id" {
2     value      = module.vpc.vpc_id
3     description = "VPC ID"
4   }
5
6   output "ecr_uri_wordpress" {
7     value      = module.ecr.ecr_uri_wordpress
8     description = "ECR Private Registry Wordpress URI"
9   }
10
11  output "rds_endpoint" {
12    value      = module.rds.db_endpoint
13    description = "RDS MySQL Database Endpoint"
14  }
```

This file exposes key outputs from the root module after applying Terraform.

- **vpc_id** returns the VPC ID created by the VPC module.
- **ecr_uri_wordpress** outputs the full private ECR URI for the WordPress image.
- **rds_endpoint** returns the RDS MySQL endpoint used by WordPress to connect to the database.

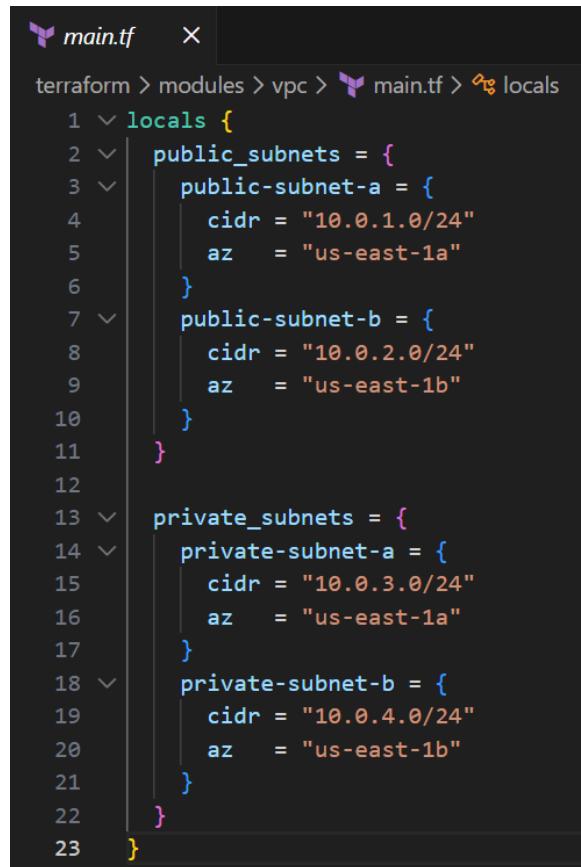
These outputs are helpful for verification and for referencing important resource details without manually navigating AWS.

Task 1.4: Configure VPC Module

This section explains each Terraform configuration file located in the VPC module (modules/vpc) and how they work together to build the networking layer used by the rest of the infrastructure.

main.tf:

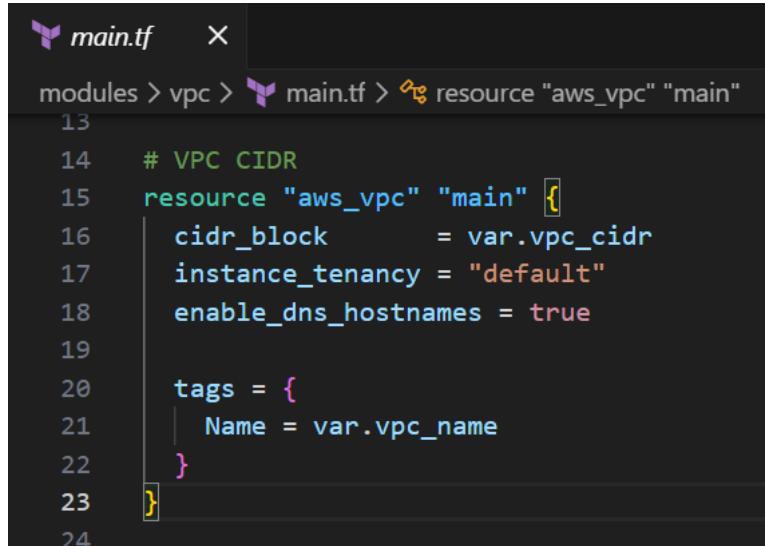
Local Variables



```
main.tf
 terraform > modules > vpc > main.tf > locals
 1 ~> locals {
 2 ~>   public_subnets = {
 3 ~>     public-subnet-a = {
 4 ~>       cidr = "10.0.1.0/24"
 5 ~>       az   = "us-east-1a"
 6 ~>     }
 7 ~>     public-subnet-b = {
 8 ~>       cidr = "10.0.2.0/24"
 9 ~>       az   = "us-east-1b"
10 ~>   }
11 ~>
12
13 ~>   private_subnets = {
14 ~>     private-subnet-a = {
15 ~>       cidr = "10.0.3.0/24"
16 ~>       az   = "us-east-1a"
17 ~>     }
18 ~>     private-subnet-b = {
19 ~>       cidr = "10.0.4.0/24"
20 ~>       az   = "us-east-1b"
21 ~>   }
22 ~>
23 }
```

- The locals block defines two maps:
 - public_subnets
 - private_subnets
- Each map contains subnet names paired with their CIDR blocks and Availability Zones.
- This method allows the module to create multiple subnets dynamically using **for_each**, reducing repetitive code.
- Adding additional subnets in the future simply requires adding another entry to the map as no new resource blocks are needed.

Virtual Private Cloud



```
modules > vpc > main.tf > resource "aws_vpc" "main"
  13
  14  # VPC CIDR
  15  resource "aws_vpc" "main" {
  16    cidr_block      = var.vpc_cidr
  17    instance_tenancy = "default"
  18    enable_dns_hostnames = true
  19
  20    tags = {
  21      Name = var.vpc_name
  22    }
  23  }
  24
```

This resource creates the core VPC that all other resources will rely on.

- **cidr_block** is passed from root variables, enabling customizable network ranges.
- **enable_dns_hostnames = true** ensures that resources inside the VPC receive DNS names automatically.
- The VPC is tagged using **var.vpc_name**, allowing easy identification in the AWS console.

Public Subnets



```
# Subnets

# Public Subnets
resource "aws_subnet" "public_subnet" {
  for_each = local.public_subnets

  vpc_id      = aws_vpc.main.id
  cidr_block   = each.value.cidr
  availability_zone = each.value.az

  tags = {
    Name = each.key
  }
}
```

This block creates public subnets using the map defined in **locals.public_subnets**.

- Created dynamically by iterating over **local.public_subnets**.
- Each subnet's CIDR block and AZ come from **each.value**.

- All subnets are automatically attached to the VPC through **vpc_id**.
- Each subnet is tagged using each.key, ensuring meaningful naming (e.g., *public-subnet-a*).

Private Subnets

```
# Private Subnets
resource "aws_subnet" "private_subnet" {
  for_each = local.private_subnets

  vpc_id          = aws_vpc.main.id
  cidr_block      = each.value.cidr
  availability_zone = each.value.az

  tags = {
    Name = each.key
  }
}
```

- Built the same way as public subnets but using **local.private_subnets**.
- These subnets will host private resources such as the RDS database.
- They do not have direct internet access and rely on the NAT Gateway for outbound connectivity.

Internet Gateway (IGW)

```
37 # Internet Gateway
38 resource "aws_internet_gateway" "igw" [
39   vpc_id = aws_vpc.main.id
40
41   tags = {
42     Name = "ecs-igw"
43   }
44 ]
45
```

- Creates an Internet Gateway and attaches it to the VPC.
- Allows public subnets to send and receive internet traffic.
- Tagged as "**ecs-igw**" for consistent naming and visibility.

Elastic IP (EIP) for NAT

```
# EIP
resource "aws_eip" "nat_gateway_eip" {
  domain = "vpc"

  tags = {
    Name = "nat-gateway-eip"
  }

  depends_on = [aws_internet_gateway.igw]
}
```

- Allocates a static public IP address for the NAT Gateway.
- It depends on the IGW to ensure the VPC has internet connectivity before NAT is deployed.

NAT Gateway

```
# NAT Gateway
resource "aws_nat_gateway" "nat_gateway" {
  allocation_id = aws_eip.nat_gateway_eip.id
  subnet_id     = aws_subnet.public_subnet["public-subnet-a"].id

  tags = {
    Name = "nat-gateway"
  }

  depends_on = [aws_internet_gateway.igw]
}
```

- The NAT Gateway is deployed in **public-subnet-a**.
- Enables private subnets to reach the internet for updates (e.g., package downloads) without becoming publicly accessible.
- Depends on the Internet Gateway to ensure correct routing setup.

VPC Route Tables

Public Route Table:

```
# Route Tables

# Public Route Table
resource "aws_route_table" "public_rt" {
  vpc_id = aws_vpc.main.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.igw.id
  }

  tags = {
    Name = "Public-Route-Table"
  }
}
```

- Routes all outbound traffic (0.0.0.0/0) from public subnets to the IGW.
- Ensures ECS services placed in public subnets can reach the internet.

Private Route Table

```
# Private Route Table
resource "aws_route_table" "private_rt" {
  vpc_id = aws_vpc.main.id

  route {
    cidr_block = "0.0.0.0/0"
    nat_gateway_id = aws_nat_gateway.nat_gateway.id
  }

  tags = {
    Name = "Private-Route-Table"
  }
}
```

- Routes outbound traffic from private subnets through the NAT Gateway.
- Ensures RDS and other backend resources can reach the internet **without being exposed**.

Route Table Associations

Public Associations

```
# Public Route Association
resource "aws_route_table_association" "public" {
  for_each = aws_subnet.public_subnet

  subnet_id      = each.value.id
  route_table_id = aws_route_table.public_rt.id
}
```

- Each public subnet is linked to the Public Route Table using an iterative for_each.
- Ensures public subnets use IGW for internet access.

Private Associations

```
# Private Route Association
resource "aws_route_table_association" "private" {
  for_each = aws_subnet.private_subnet

  subnet_id      = each.value.id
  route_table_id = aws_route_table.private_rt.id
}
```

- Each private subnet is associated with the Private Route Table.
- Ensures private subnets route outbound traffic through the NAT gateway only.

variables.tf:

```
variables.tf
 terraform > modules > vpc > variables.tf >
 1   variable "vpc_name" {
 2     |   type = string
 3   }
 4
 5   variable "vpc_cidr" {
 6     |   type = string
 7 }
```

- **vpc_name** defines the name tag applied to the VPC.
- **vpc_cidr** allows the VPC's CIDR block to be passed in dynamically from the root module.

These variables provide flexibility and reusability across different environments (dev, test, prod).

outputs.tf:

```
outputs.tf
 terraform > modules > vpc > outputs.tf > output "private_subnets"
 1 < output "igw_name" {
 2   |   description = "Internet Gateway"
 3   |   value       = aws_internet_gateway.igw.id
 4 }
 5
 6 < output "vpc_id" {
 7   |   description = "VPC ID"
 8   |   value       = aws_vpc.main.id
 9 }
10
11 < output "public_subnets" {
12   |   description = "VPC Public Subnets"
13   |   value       = [for s in aws_subnet.public_subnet : s.id]
14 }
15
16 < output "private_subnets" {
17   |   description = "VPC Private Subnets"
18   |   value       = [for s in aws_subnet.private_subnet : s.id]
19 }
```

igw_name

- Outputs the Internet Gateway ID for reference by other modules if required.

vpc_id

- Provides the VPC ID to the root module so other services can attach resources inside this network.

public_subnets

- Returns a list of all public subnet IDs.
- Used by modules like ECS, which require public subnets for Fargate tasks.

private_subnets

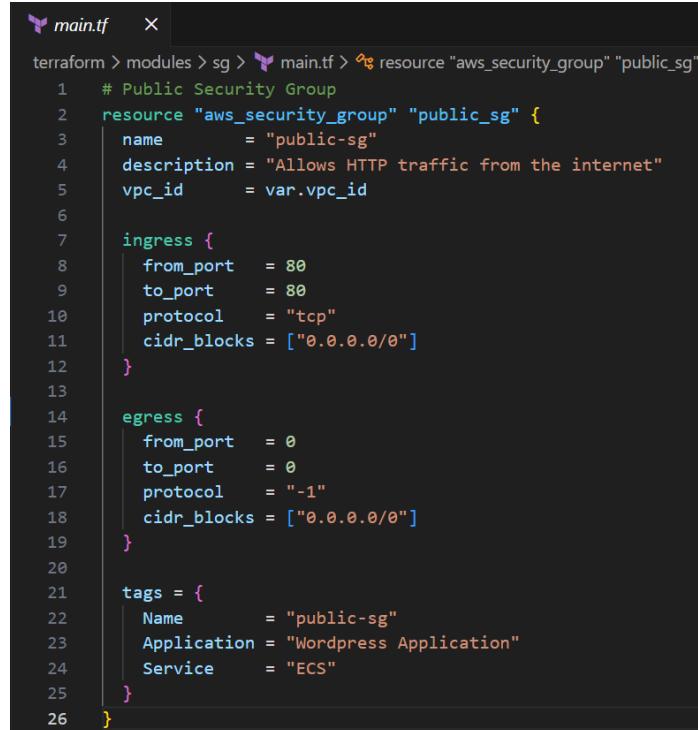
- Returns a list of private subnet IDs.
- Used by the RDS module to deploy the database securely.

Task 1.5: Configure Security Group Module

This section explains each Terraform configuration file inside the Security Group module (modules/sg) and how they manage traffic flow between the ECS service and the RDS database.

main.tf:

Public Security Group



```
main.tf
 terraform > modules > sg > main.tf > resource "aws_security_group" "public_sg"
 1  # Public Security Group
 2  resource "aws_security_group" "public_sg" {
 3    name        = "public-sg"
 4    description = "Allows HTTP traffic from the internet"
 5    vpc_id      = var.vpc_id
 6
 7    ingress {
 8      from_port  = 80
 9      to_port    = 80
10      protocol   = "tcp"
11      cidr_blocks = ["0.0.0.0/0"]
12    }
13
14    egress {
15      from_port  = 0
16      to_port    = 0
17      protocol   = "-1"
18      cidr_blocks = ["0.0.0.0/0"]
19    }
20
21    tags = {
22      Name        = "public-sg"
23      Application = "Wordpress Application"
24      Service     = "ECS"
25    }
26 }
```

- This resource creates a security group inside the VPC (`vpc_id` passed from the root module).
- The purpose of this security group is to allow external HTTP traffic to reach the ECS Fargate service running WordPress.
- Ingress Rule:
 - Allows inbound TCP traffic on port 80 from anywhere (0.0.0.0/0).
 - This makes the ECS service publicly accessible for web requests.
- Egress Rule:
 - Allows all outbound traffic to any destination.
 - Ensures ECS tasks can reach external services such as ECR, package repositories, or the NAT Gateway.
- Tagged for clarity using Name, Application, and Service, making the resource easy to identify in the AWS Console.

Private Security Group (for RDS MySQL)

```
# Private Security Group used for RDS
resource "aws_security_group" "private_sg" {
  name      = "private-sg"
  description = "Allows MySQL traffic from ECS (public-sg) only"
  vpc_id    = var.vpc_id

  ingress {
    from_port    = 3306
    to_port     = 3306
    protocol    = "tcp"
    security_groups = [aws_security_group.public_sg.id]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol   = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name      = "private-sg"
    Application = "RDS MySQL Database"
  }
}
```

- This security group is dedicated to the RDS MySQL database.
- It restricts incoming MySQL connections to only the ECS application by referencing the public security group.
- Ingress Rule:
 - Allows inbound traffic on port 3306 (MySQL) only from the **public_sg** security group.
 - This ensures the database is not exposed publicly and only accepts traffic from trusted ECS tasks.
- Egress Rule:
 - Allows all outbound traffic to support tasks like DNS resolution and updates.
- Tags clearly indicate that this SG belongs to the RDS database layer.

This setup ensures proper separation of concerns:

- The ECS service is accessible to the internet.
- The RDS database is isolated inside private subnets and only reachable by ECS.

variables.tf:



```
variables.tf ×
terraform > modules > sg > variables.tf > variable "vpc_id"
1   variable "vpc_id" {
2     type = string
3 }
```

- **vpc_id** is the only required input variable.
- It ensures both security groups are created inside the correct VPC, making the module fully reusable across environments.

outputs.tf



```
outputs.tf ×
terraform > modules > sg > outputs.tf > output "security_group_private_id"
1   output "security_group_id" {
2     value      = aws_security_group.public_sg.id
3     description = "Security group ID for public resources"
4   }
5
6   output "security_group_private_id" {
7     value      = aws_security_group.private_sg.id
8     description = "Security group ID for private resources (RDS)"
9 }
```

security_group_id

- Outputs the public security group ID.
- Used by the ECS module to attach this SG to the Fargate service.

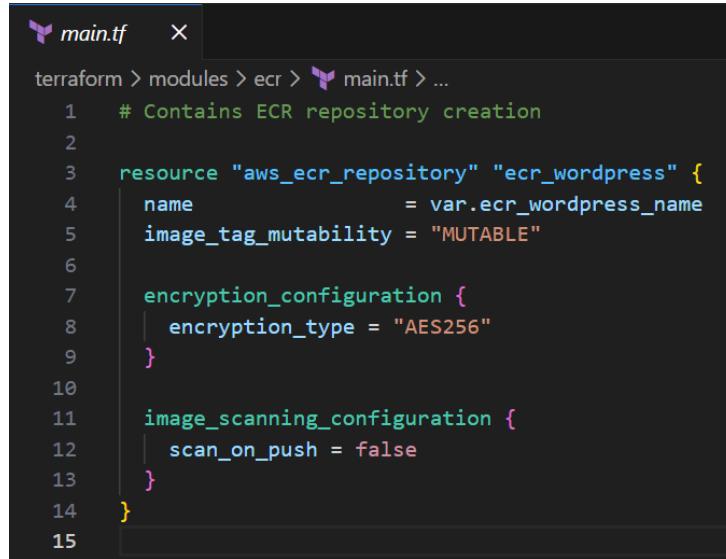
security_group_private_id

- Outputs the private RDS security group ID.
- Used by the RDS module to secure the database and restrict access to only the ECS application.

Task 1.6: Configure Elastic Container Registry (ECR) Module

This section explains the Terraform configuration files inside the **ECR module (modules/ecr)**, responsible for creating the private container registry used by ECS.

main.tf

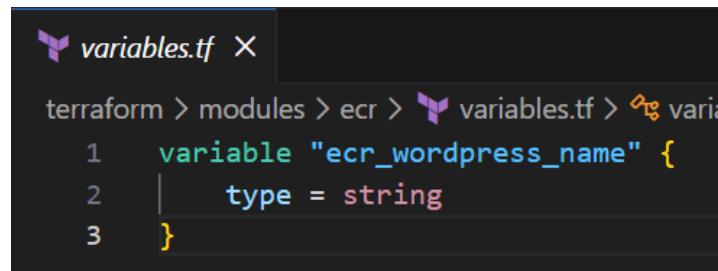


```
main.tf
 terraform > modules > ecr > main.tf > ...
 1  # Contains ECR repository creation
 2
 3  resource "aws_ecr_repository" "ecr_wordpress" {
 4      name          = var.ecr_wordpress_name
 5      image_tag_mutability = "MUTABLE"
 6
 7      encryption_configuration {
 8          encryption_type = "AES256"
 9      }
10
11      image_scanning_configuration {
12          scan_on_push = false
13      }
14  }
15
```

- This module creates an Amazon ECR (Elastic Container Registry) repository to store the WordPress Docker image.
- name is dynamically passed from the root variable **ecr_wordpress_name**, allowing flexible naming across environments.
- **image_tag_mutability = "MUTABLE"** allows updating tags (e.g., overwriting latest), which simplifies redeployments.
- The repository is encrypted using **AES256** for secure storage.
- Image scanning on push is disabled (**scan_on_push = false**), though it can be enabled in production environments.

This registry will store the container image that is later pulled by ECS Fargate during application deployment.

variables.tf:



```
variables.tf
 terraform > modules > ecr > variables.tf > ...
 1  variable "ecr_wordpress_name" {
 2      type = string
 3  }
```

- Defines the variable **ecr_wordpress_name**, allowing the repository name to be supplied from **terraform.tfvars**.
- Keeping this variable external ensures the module can be reused for any application image in the future.

outputs.tf:

```
outputs.tf  X
terraform > modules > ecr > outputs.tf > output "ecr_uri_wordpress"
1   output "ecr_uri_wordpress" {
2     value      = aws_ecr_repository.ecr_wordpress.repository_url
3     description = "Wordpress ECR Repository URI"
4 }
```

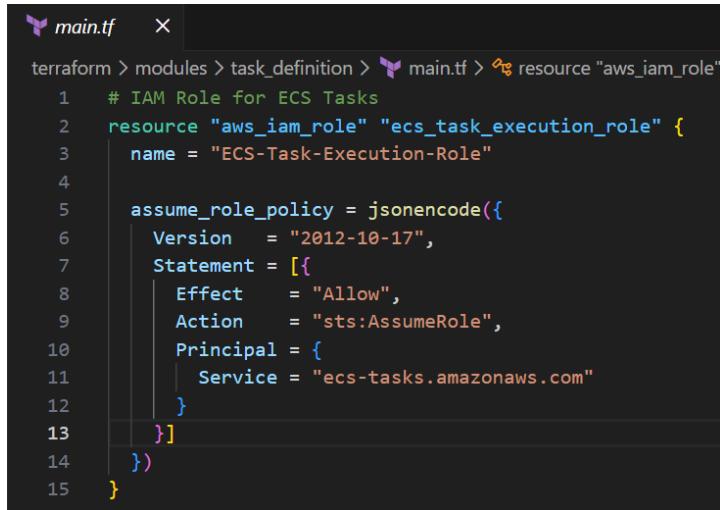
- Outputs **ecr_uri_wordpress**, which exposes the full repository URL (e.g., 123456789012.dkr.ecr.us-east-1.amazonaws.com/wordpress).
- This URL is passed to the Task Definition module so ECS knows where to pull the WordPress image from.

Task 1.7: Configure Task Definition Module

This section explains the Terraform files inside the **Task Definition module (modules/task_definition)**, which defines how the WordPress container should run inside ECS Fargate.

main.tf:

ECS Task Execution Role



```
main.tf
 terraform > modules > task_definition > main.tf > resource "aws_iam_role" "ecs_task_execution_role" {
  name = "ECS-Task-Execution-Role"

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

- Creates an IAM Role that ECS tasks assume during execution.
- The **assume_role_policy** allows ECS to use this role.
- This role allows the task to:
 - Pull images from ECR
 - Send logs to CloudWatch
 - Access necessary AWS APIs during runtime

IAM Policy Attachment



```
# IAM Policy
resource "aws_iam_role_policy_attachment" "ecs_exec_policy" {
  role        = aws_iam_role.ecs_task_execution_role.name
  policy_arn = "arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy"
}
```

- Attaches the AWS-managed policy **AmazonECSTaskExecutionRolePolicy**.
- This policy automatically provides:
 - ECR read access
 - CloudWatch Logs permissions
 - Basic ECS runtime permissions

This avoids creating custom IAM policies unless needed.

CloudWatch Log Group

```
# CloudWatch Log Group
resource "aws_cloudwatch_log_group" "ecs_logs" {
    name          = "/ecs/wordpress-app"
    retention_in_days = 7
}
```

- Creates a CloudWatch Logs group named /ecs/wordpress-app.
- All container logs from ECS will be stored here.
- Retention is set to 7 days, preventing unnecessary storage costs.

ECS Task Definition

```
# ECS Task Definition
resource "aws_ecs_task_definition" "task_definition" {
    family           = "wordpress-app"
    network_mode     = "awsvpc"
    requires_compatibilities = ["FARGATE"]
    cpu              = "512"
    memory           = "1024"

    execution_role_arn = aws_iam_role.ecs_task_execution_role.arn
    task_role_arn      = aws_iam_role.ecs_task_execution_role.arn

    container_definitions = jsonencode([
        {
            name      = "wordpress"
            image     = "${var.ecr_uri_wordpress}:latest"
            essential = true
            cpu       = 200
            memory    = 512

            environment = [
                { name = "WORDPRESS_DB_HOST",      value = "${var.rds_endpoint}:3306" },
                { name = "WORDPRESS_DB_USER",      value = var.db_username },
                { name = "WORDPRESS_DB_PASSWORD", value = var.db_password },
                { name = "WORDPRESS_DB_NAME",     value = var.db_name }
            ]
        }

        portMappings = [
            {
                containerPort = 80
                protocol     = "tcp"
            }
        ]

        logConfiguration = {
            logDriver = "awslogs"
            options = {
                awslogs-group      = aws_cloudwatch_log_group.ecs_logs.name
                awslogs-region     = "us-east-1"
                awslogs-stream-prefix = "wordpress"
            }
        }
    ])
}
```

This resource defines how the WordPress container runs on ECS Fargate. Key settings include:

General Settings

- family = "wordpress-app" groups revisions of the same task definition.
- network_mode = "awsvpc" is required for Fargate.
- CPU and memory are allocated (512 CPU units, 1024 MB memory).

Task and Execution Role

- Both execution_role_arn and task_role_arn use the ECS Task Execution Role created earlier.

Container Definitions

A single container is defined:

- Name: wordpress
- Image pulled from the ECR repository (`${var.ecr_uri_wordpress}:latest`)
- CPU and memory are defined for the container's share of task resources.

Environment Variables

These variables provide WordPress with the required database configuration:

- **WORDPRESS_DB_HOST**: RDS endpoint
- **WORDPRESS_DB_USER**: Database username
- **WORDPRESS_DB_PASSWORD**: Database password
- **WORDPRESS_DB_NAME**: Database name

This connects WordPress running on ECS to the MySQL database running on RDS.

Port Mapping

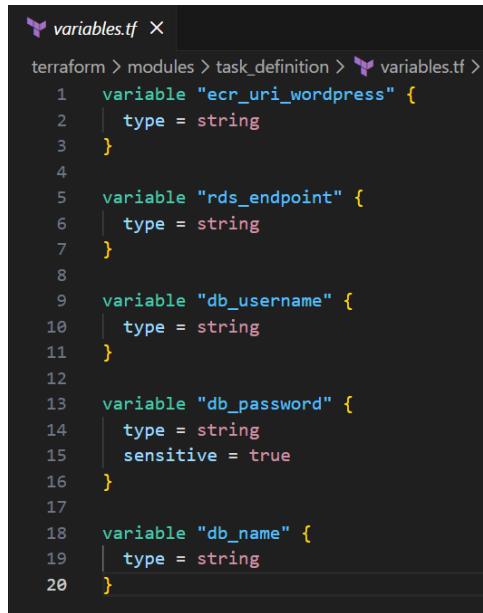
- Maps container port 80, allowing ECS to route traffic to the WordPress application.

Log Configuration

- Sends all logs to CloudWatch using:
 - log group: /ecs/wordpress-app
 - region: us-east-1
 - stream prefix: wordpress

This ensures observability for debugging and monitoring.

variables.tf:



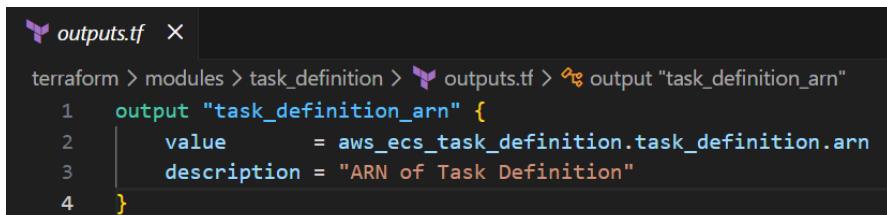
```
variables.tf ×
terraform > modules > task_definition > variables.tf >
1   variable "ecr_uri_wordpress" {
2     type = string
3   }
4
5   variable "rds_endpoint" {
6     type = string
7   }
8
9   variable "db_username" {
10    type = string
11  }
12
13  variable "db_password" {
14    type = string
15    sensitive = true
16  }
17
18  variable "db_name" {
19    type = string
20  }
```

Defines inputs needed by the task definition:

- **ecr_uri_wordpress** is the Image location in ECR
- **rds_endpoint** is the Private RDS MySQL database endpoint
- **db_username**, **db_password**, **db_name** define credentials and DB name used by WordPress

The **db_password** variable is marked as sensitive, preventing accidental logging of secrets.

outputs.tf:



```
outputs.tf ×
terraform > modules > task_definition > outputs.tf > output "task_definition_arn"
1   output "task_definition_arn" {
2     value      = aws_ecs_task_definition.task_definition.arn
3     description = "ARN of Task Definition"
4   }
```

task_definition_arn

- Exposes the ARN of the ECS Task Definition.
- This ARN is required by the ECS module to deploy the service.

Task 1.8: Configure Elastic Container Service Module

This module provisions the ECS cluster and ECS service that run the WordPress container on AWS Fargate.

main.tf:

ECS Cluster

```
# Cluster
resource "aws_ecs_cluster" "ecs_cluster" {
  name = "wordpress-ecs-cluster"

  setting {
    name  = "containerInsights"
    value = "enhanced"
  }

  configuration {
    execute_command_configuration {
      logging = "DEFAULT"
    }
  }
}
```

- Creates an ECS cluster named **wordpress-ecs-cluster**.
- Enables Container Insights (enhanced) for improved performance monitoring.
- Configures ECS Exec, allowing secure command execution directly into running containers for debugging.

Cluster Capacity Providers

```
resource "aws_ecs_cluster_capacity_providers" "example" {
  cluster_name = aws_ecs_cluster.ecs_cluster.name

  capacity_providers = ["FARGATE"]

  default_capacity_provider_strategy {
    base          = 1
    weight        = 100
    capacity_provider = "FARGATE"
  }
}
```

- Uses **FARGATE** as the capacity provider.
- Sets Fargate as the default with weight=100, ensuring all tasks run using serverless Fargate capacity.

ECS Service

```

resource "aws_ecs_service" "wordpress-service" {
  name          = "wordpress-service"
  cluster       = aws_ecs_cluster.ecs_cluster.id
  task_definition = var.task_definition_arn
  desired_count    = 2
  launch_type     = "FARGATE"
  platform_version = "LATEST"
  scheduling_strategy = "REPLICA"
  enable_execute_command = true

  deployment_configuration {
    strategy = "ROLLING"
  }

  network_configuration {
    assign_public_ip = true
    security_groups  = [var.security_group_id]
    subnets          = var.public_subnets
  }
}

```

- Creates the service **wordpress-service** running on Fargate.
- Uses the task definition passed in from the task definition module (**var.task_definition_arn**).
- Runs 2 containers for redundancy and high availability (`desired_count = 2`).
- Uses REPLICA scheduling strategy to evenly place tasks.
- Enables ECS Exec for troubleshooting.

Networking

- Uses:
 - **public_subnets** for task placement
 - **security_group_id** to control inbound/outbound traffic
 - public IP assignment enabled (**assign_public_ip = true**)

This ensures the WordPress application is reachable externally.

Deployment

- Uses a rolling deployment strategy, ensuring zero downtime when updating the service.

variables.tf:

Defines variables used to configure the service:

- **task_definition_arn:** The task definition to run
- **public_subnets:** List of subnets for Fargate tasks
- **security_group_id:** ECS service security group

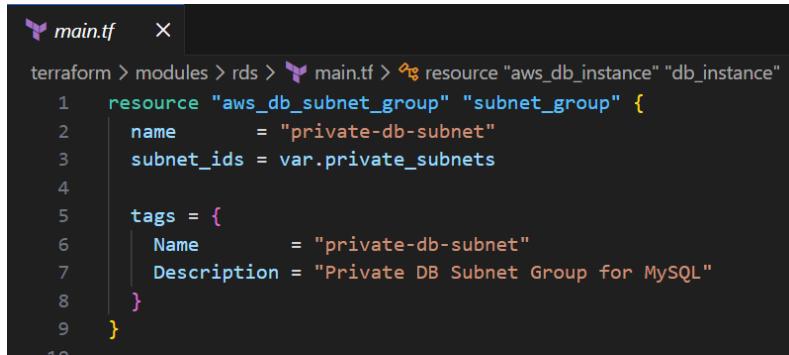
These inputs maintain modularity and allow you to plug in values from other modules.

Task 1.9: Configure Relation Database Service (RDS) Module

This module provisions the MySQL RDS database used by the WordPress ECS application. It creates a private subnet group for the DB and deploys the RDS instance inside secure, non-public subnets.

main.tf:

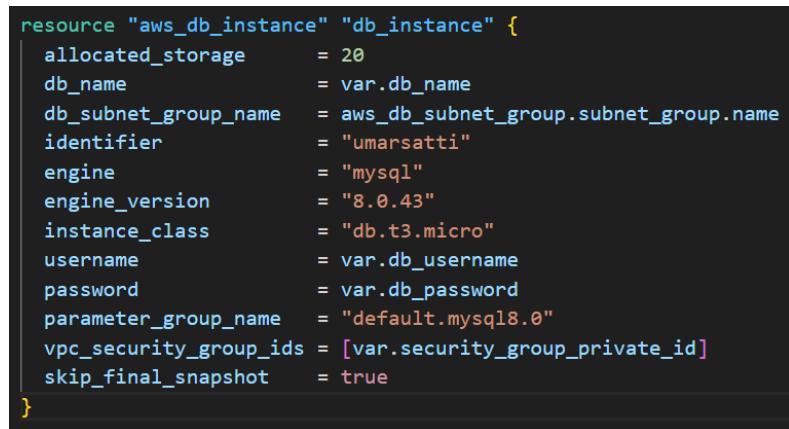
DB Subnet Group



```
main.tf
  terraform > modules > rds > main.tf > resource "aws_db_subnet_group" "subnet_group" {
    1   name      = "private-db-subnet"
    2   subnet_ids = var.private_subnets
    3
    4   tags = {
    5     Name      = "private-db-subnet"
    6     Description = "Private DB Subnet Group for MySQL"
    7   }
    8
    9 }
```

- Creates a subnet group named **private-db-subnet**.
- Uses the **private subnets** passed from the VPC module.
- Ensures the RDS instance is deployed in isolated subnets, not accessible from the internet.

RDS MySQL Instance



```
resource "aws_db_instance" "db_instance" {
  allocated_storage      = 20
  db_name                = var.db_name
  db_subnet_group_name   = aws_db_subnet_group.subnet_group.name
  identifier              = "umarsatti"
  engine                  = "mysql"
  engine_version          = "8.0.43"
  instance_class           = "db.t3.micro"
  username                = var.db_username
  password                = var.db_password
  parameter_group_name    = "default.mysql8.0"
  vpc_security_group_ids  = [var.security_group_private_id]
  skip_final_snapshot      = true
}
```

- Provisions a **MySQL 8.0** database using a small instance type (db.t3.micro) and 20GB storage.
- Uses credentials and database name passed in from variables.
- Associates the RDS instance with the **private security group** so only ECS containers (via public-sg → private-sg rule) can access it.
- `skip_final_snapshot = true` makes deletion faster during development.

variables.tf:

```
variables.tf
variable "private_subnets" {
  type = any
}
variable "security_group_private_id" {
  type = string
}
variable "db_username" {
  type = string
}
variable "db_password" {
  type = string
  sensitive = true
}
variable "db_name" {
  type = string
}
```

Defines inputs required to deploy RDS:

- **private_subnets**: Ensures DB is placed in private network tiers
- **security_group_private_id**: Restricts DB access to ECS tasks only
- **db_username, db_password**: Credentials for MySQL
- **db_name**: Name of the WordPress database

These values are passed down securely from the root module.

outputs.tf:

```
outputs.tf
output "db_endpoint" {
  value = aws_db_instance.db_instance.address
  description = "RDS endpoint address"
}
output "db_username" {
  value = aws_db_instance.db_instance.username
  description = "RDS username"
}
output "db_name" {
  value = aws_db_instance.db_instance.db_name
  description = "RDS DB name"
}
output "db_password" {
  value = var.db_password
  sensitive = true
  description = "RDS DB password (sensitive input forwarded)"
}
```

Exports key information for downstream modules:

- **db_endpoint** is the hostname used by the WordPress container
- **db_username, db_name** is used in task definition environment variables
- **db_password** is marked sensitive and forwarded securely

These outputs allow the ECS task definition module to connect WordPress to the database.

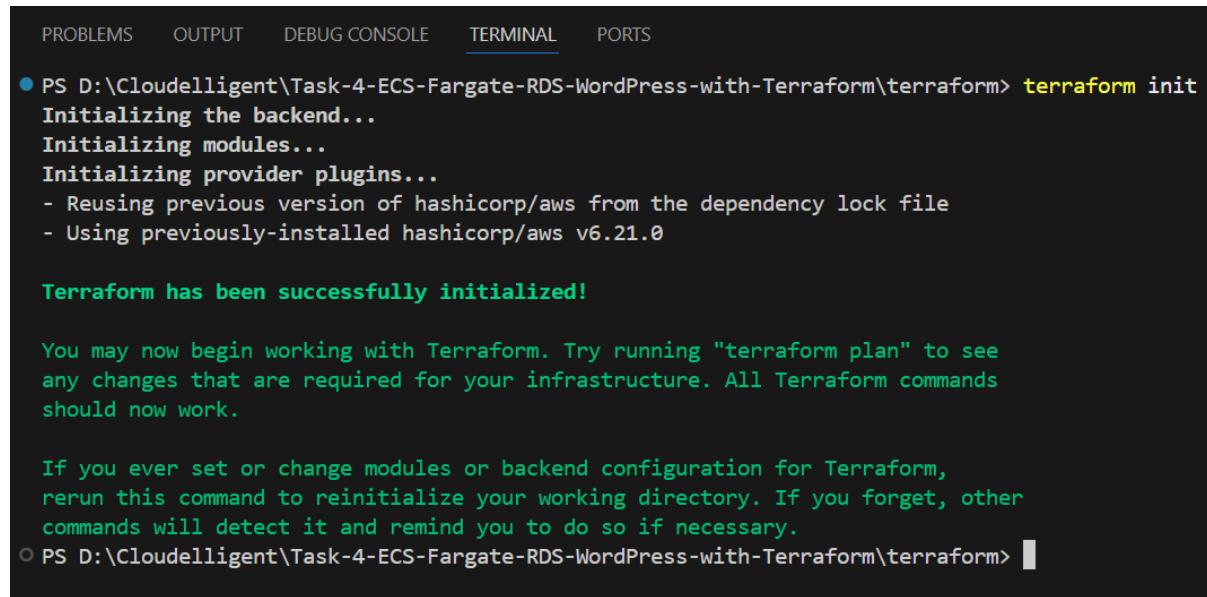
Task 1.10: Running Terraform Commands to Deploy Infrastructure

This section documents the series of Terraform CLI commands executed to deploy, validate, and destroy the WordPress environment.

Note: To perform this task, the user must be in the root directory of Terraform project where the provider.tf and the root main.tf files are stored.

Step 1: terraform init

Initializes the working directory by downloading the required provider plugins and connecting them to the configured backend (S3).



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

● PS D:\Cloudelligent\Task-4-ECS-Fargate-RDS-WordPress-with-Terraform\terraform> terraform init
Initializing the backend...
Initializing modules...
Initializing provider plugins...
  - Reusing previous version of hashicorp/aws from the dependency lock file
  - Using previously-installed hashicorp/aws v6.21.0

Terraform has been successfully initialized!

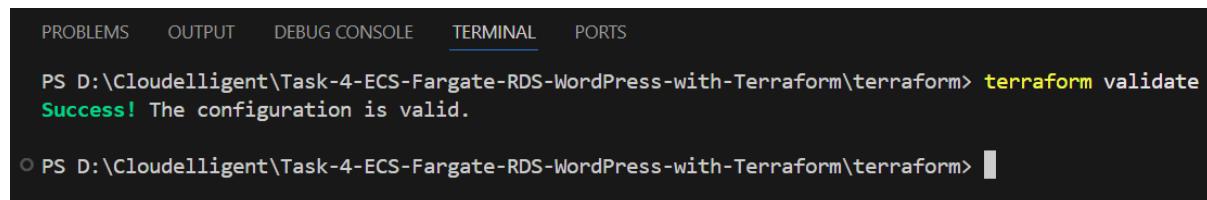
You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
○ PS D:\Cloudelligent\Task-4-ECS-Fargate-RDS-WordPress-with-Terraform\terraform>
```

By running this command, Terraform confirms initialization, backend setup, and provider readiness.

Step 2: terraform validate

Performs a syntax and logic check on all configuration files in the directory. Outputs an error if the logic or syntax is incorrect.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS D:\Cloudelligent\Task-4-ECS-Fargate-RDS-WordPress-with-Terraform\terraform> terraform validate
Success! The configuration is valid.
○ PS D:\Cloudelligent\Task-4-ECS-Fargate-RDS-WordPress-with-Terraform\terraform>
```

Step 3: terraform plan

Generates an execution plan showing all actions Terraform will perform to reach the desired state (resource creation, updates, or deletions).

```
PS D:\Cloudelligent\Task-4-ECS-Fargate-RDS-WordPress-with-Terraform\terraform> terraform plan
    }
    + tags_all
      + "Name" = "ecs-vpc"
    }
}

Plan: 26 to add, 0 to change, 0 to destroy.

Changes to Outputs:
+ ecr_uri_wordpress = (known after apply)
+ rds_endpoint      = (known after apply)
+ vpc_id            = (known after apply)

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.
Releasing state lock. This may take a few moments...
○ PS D:\Cloudelligent\Task-4-ECS-Fargate-RDS-WordPress-with-Terraform\terraform>
```

Plan shows **26 to add, 0 to change, 0 to destroy**, confirming all required AWS resources are queued for creation.

Step 4: terraform apply –auto-approve (terraform apply)

Executes the plan and provisions the resources defined in the Terraform configuration without manual confirmation.

```
PS D:\Cloudelligent\Task-4-ECS Fargate Nginx Server Deployment with Terraform\terraform> terraform apply --auto-approve
module.vpc.aws_vpc.main: Creation complete after 15s [id=vpc-0c31353aa529415a9]
module.vpc.aws_internet_gateway.igw: Creating...
module.vpc.aws_subnet.public_subnet["public-subnet-a"]: Creating...
module.vpc.aws_subnet.public_subnet["public-subnet-b"]: Creating...
module.sg.aws_security_group.public_sg: Creating...
module.vpc.aws_internet_gateway.igw: Creation complete after 2s [id=igw-072efb59d2c674070]
module.vpc.aws_route_table.public_rt: Creating...
module.vpc.aws_subnet.public_subnet["public-subnet-a"]: Creation complete after 2s [id=subnet-0f1fb10d9e90663ce]
module.vpc.aws_subnet.public_subnet["public-subnet-b"]: Creation complete after 2s [id=subnet-03b536cb1b6ccba17]
module.vpc.aws_route_table.public_rt: Creation complete after 2s [id=rtb-0236706e829242711]
module.vpc.aws_route_table_association.public_a: Creating...
module.vpc.aws_route_table_association.public_b: Creating...
module.vpc.aws_route_table_association.public_a: Creation complete after 1s [id=rtbassoc-07325e7a4756f4da2]
module.vpc.aws_route_table_association.public_b: Creation complete after 1s [id=rtbassoc-0dfe482d0a71da778]
module.sg.aws_security_group.public_sg: Creation complete after 6s [id=sg-00c9e769c3dca928d]
module.ecs.aws_ecs_service.wordpress-service: Creating...
module.ecs.aws_ecs_cluster_capacity_providers.example: Still creating... [0m10s elapsed]
module.ecs.aws_ecs_service.wordpress-service: Creation complete after 2s [id=arn:aws:ecs:us-east-1:730335208305:service/wordpress-ecs-cluster/wordpress-service]
module.ecs.aws_ecs_cluster_capacity_providers.example: Creation complete after 11s [id=wordpress-ecs-cluster]
Releasing state lock. This may take a few moments...

Apply complete! Resources: 17 added, 0 changed, 0 destroyed.

Outputs:

ecr_uri_mysql = "730335208305.dkr.ecr.us-east-1.amazonaws.com/mysql"
ecr_uri_wordpress = "730335208305.dkr.ecr.us-east-1.amazonaws.com/wordpress"
vpc_id = "vpc-0c31353aa529415a9"
○ PS D:\Cloudelligent\Task-4-ECS Fargate Nginx Server Deployment with Terraform\terraform>
```

Resources (VPC, Subnets, Route Tables, Security Group, ECR, ECS, RDS etc.) are created successfully, followed by VPC ID, ECR URI for WordPress, and RDS Endpoint.

Step 5: terraform destroy --auto-approve (or terraform destroy)

Destroys all resources previously created by Terraform, cleaning up the AWS environment.

```
PS D:\Cloudelligent\Task-4-ECS-Fargate-RDS-WordPress-with-Terraform> terraform destroy --auto-approve
module.sg.aws_security_group.private_sg: Destroying... [id=sg-055550042037788d0]
module.rds.aws_db_subnet_group.subnet_group: Destruction complete after 1s
module.vpc.aws_subnet.private_subnet["private-subnet-a"]: Destroying... [id=subnet-0ff1749746b583d5a]
module.vpc.aws_subnet.private_subnet["private-subnet-b"]: Destroying... [id=subnet-046c577b01ca4b010]
module.vpc.aws_subnet.private_subnet["private-subnet-b"]: Destruction complete after 1s
module.vpc.aws_subnet.private_subnet["private-subnet-a"]: Destruction complete after 1s
module.vpc.aws_subnet.private_subnet["private-subnet-a"]: Destruction complete after 2s
○ module.sg.aws_security_group.public_sg: Destroying... [id=sg-06e2b01e21cf91100]
module.sg.aws_security_group.public_sg: Destruction complete after 1s
module.vpc.aws_vpc.main: Destroying... [id=vpc-0e6ddc9cedfbc3776]
module.vpc.aws_vpc.main: Destruction complete after 1s
Releasing state lock. This may take a few moments...

Destroy complete! Resources: 26 destroyed.
PS D:\Cloudelligent\Task-4-ECS-Fargate-RDS-WordPress-with-Terraform>
```

Displays **Plan: 0 to add, 0 to change, 26 to destroy** and confirm complete deletion of all deployed resources.

The state file in S3 will also be updated automatically to reflect the destroyed state.

Task 1.11: Validate Infrastructure Deployment and WordPress Installation

This task demonstrates the successful deployment and verification of all AWS and application-level components created using Terraform. It includes screenshots of the AWS infrastructure created such as VPC, Security groups, ECS, ECR etc.

Task 1.11.1: Verify Deployment of AWS Infrastructure

After running **terraform apply** command, Terraform provisions all the required networking and compute resources. In this step, confirm that these resources exist and are configured correctly.

Verify VPC Creation

1. Sign in to the AWS Management Console.
2. Navigate to **VPC** service using the search bar at the top and then click **Your VPCs**.
3. Verify that a VPC named **ecs-vpc** has been created.
 - CIDR Block: 10.0.0.0/16
 - DNS Hostnames: Enabled
 - This VPC acts as the isolated network for the WordPress application environment.

Your VPCs (1/3) [Info](#)

Last updated less than a minute ago [Actions](#) [Create VPC](#)

Find VPCs by attribute or tag

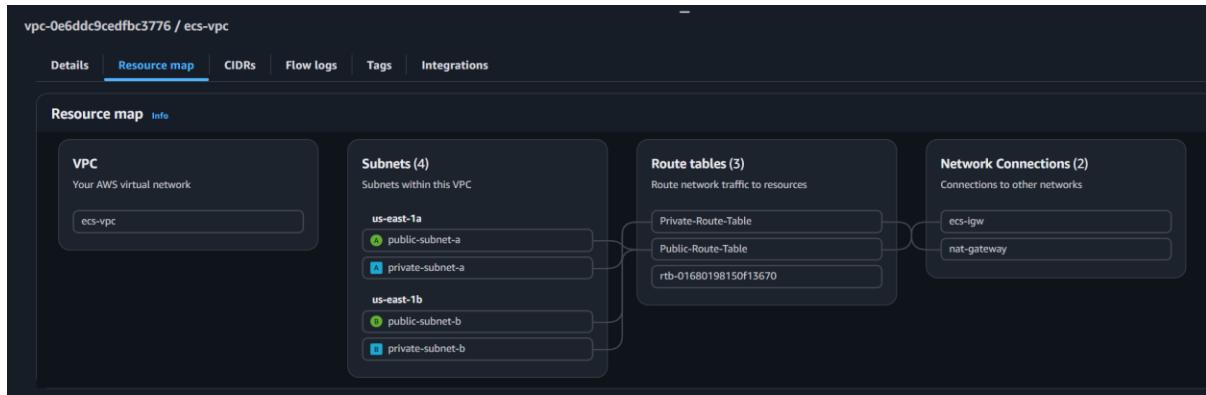
Name	VPC ID	State	Block Public...	IPv4 CIDR	IPv6 CIDR	DHCP option set
<input checked="" type="checkbox"/> ecs-vpc	vpc-0e6ddc9cedfbc3776	Available	<input type="radio"/> Off	10.0.0.0/16	-	dopt-088ee356fa06392dc
<input type="checkbox"/> Default VPC	vpc-040f5671e02b7f149	Available	<input type="radio"/> Off	172.31.0.0/16	-	dopt-088ee356fa06392dc
<input type="checkbox"/> nginx-vpc	vpc-0e9378abac897865	Available	<input type="radio"/> Off	10.0.0.0/16	-	dopt-088ee356fa06392dc

vpc-0e6ddc9cedfbc3776 / ecs-vpc

[Details](#) [Resource map](#) [CIDRs](#) [Flow logs](#) [Tags](#) [Integrations](#)

Details

VPC ID <input type="checkbox"/> vpc-0e6ddc9cedfbc3776	State Available	Block Public Access <input type="radio"/> Off	DNS hostnames Enabled
DNS resolution Enabled	Tenancy default	DHCP option set dopt-088ee356fa06392dc	Main route table rtb-01680198150f13670
Main network ACL acl-06f13e9538496cae	Default VPC No	IPv4 CIDR 10.0.0.0/16	IPv6 pool -
IPv6 CIDR (Network border group) -	Network Address Usage metrics Disabled	Route 53 Resolver DNS Firewall rule groups -	Owner ID <input type="checkbox"/> 730335208305



Verify Internet Gateway (IGW) Creation

1. In the VPC console, choose **Internet Gateways**.
2. Confirm that an Internet Gateway named **ecs-igw** exists and is attached to your VPC. This gateway allows the ECS service to communicate with the internet.

Internet gateways (1/3) [Info](#)

Last updated less than a minute ago [Actions](#) [Create internet gateway](#)

Find internet gateways by attribute or tag

Name	Internet gateway ID	State	VPC ID	Owner
<input type="checkbox"/> nginx-igw	igw-0140c1e822ab8c8d2	Attached	vpc-0e9378abac897865 nginx-vpc	730335208305
<input checked="" type="checkbox"/> ecs-igw	igw-094a4675e1602cedf	Attached	vpc-0e6ddc9cedfbc3776 ecs-vpc	730335208305
<input type="checkbox"/> -	igw-0cfb41de121fe763e	Attached	vpc-040f5671e02b7f149 Default VPC	730335208305

igw-094a4675e1602cedf / ecs-igw

[Details](#) [Tags](#)

Details

Internet gateway ID <input type="checkbox"/> igw-094a4675e1602cedf	State Attached	VPC ID vpc-0e6ddc9cedfbc3776 ecs-vpc	Owner <input type="checkbox"/> 730335208305
---	--------------------------------	---	--

Verify Subnet Creation

Subnets (4) Info						
Last updated 3 minutes ago Actions ▾ Create subnet						
<input type="text" value="Find subnets by attribute or tag"/> Clear filters						
Name	Subnet ID	State	VPC	Block Public Access	IPv4 CIDR	IPv4 CIDR
public-subnet-a	subnet-0ef79a97783673750	Available	vpc-0e6ddc9cedfb3776 ecs-vpc	Off	10.0.1.0/24	10.0.1.0/24
public-subnet-b	subnet-0d6be7088204f3258	Available	vpc-0e6ddc9cedfb3776 ecs-vpc	Off	10.0.2.0/24	10.0.2.0/24
private-subnet-a	subnet-0ff1749746b583d5a	Available	vpc-0e6ddc9cedfb3776 ecs-vpc	Off	10.0.3.0/24	10.0.3.0/24
private-subnet-b	subnet-046c577b01ca4b010	Available	vpc-0e6ddc9cedfb3776 ecs-vpc	Off	10.0.4.0/24	10.0.4.0/24

Public Subnets:

subnet-0ef79a97783673750 / public-subnet-a

[Details](#) [Flow logs](#) [Route table](#) [Network ACL](#) [CIDR reservations](#) [Sharing](#) [Tags](#)

Details	Subnet ARN arn:aws:ec2:us-east-1:730335208305:subnet/subnet-0ef79a97783673750	State Available	Block Public Access Off
IPv4 CIDR	10.0.1.0/24	IPv6 CIDR	IPv6 CIDR association ID
Availability Zone	use1-az2 (us-east-1a)	VPC	Route table
Network ACL	ad-06f13e595384496cae	Auto-assign public IPv4 address	rtb-06e5a5abdbff090d4b Public-Route-Table
Auto-assign customer-owned IPv4 address	No	Outpost ID	Auto-assign IPv6 address
IPv6 CIDR reservations	-	Hostname type	IPv4 CIDR reservations
Resource name DNS AAAA record	Disabled	IP name	Resource name DNS A record
DNS64	Disabled	Owner	Disabled

subnet-0d6be7088204f3258 / public-subnet-b

[Details](#) [Flow logs](#) [Route table](#) [Network ACL](#) [CIDR reservations](#) [Sharing](#) [Tags](#)

Details	Subnet ARN arn:aws:ec2:us-east-1:730335208305:subnet/subnet-0d6be7088204f3258	State Available	Block Public Access Off
IPv4 CIDR	10.0.2.0/24	IPv6 CIDR	IPv6 CIDR association ID
Availability Zone	use1-az4 (us-east-1b)	VPC	Route table
Network ACL	ad-06f13e595384496cae	Auto-assign public IPv4 address	rtb-06e5a5abdbff090d4b Public-Route-Table
Auto-assign customer-owned IPv4 address	No	Outpost ID	Auto-assign IPv6 address
IPv6 CIDR reservations	-	Hostname type	IPv4 CIDR reservations
Resource name DNS AAAA record	Disabled	IP name	Resource name DNS A record
DNS64	Disabled	Owner	Disabled

- In the VPC console, choose **Subnets** located on the left navigation panel.
- Confirm that two subnets named **public-subnet-a** and **public-subnet-b** exist.
- The subnets should have an IPv4 CIDR **10.0.1.0/24** and **10.0.2.0/24**.

Private Subnets:

- In the VPC console, choose **Subnets** located on the left navigation panel.
- Confirm that two subnets named **private-subnet-a** and **private-subnet-b** exist.
- The subnets should have an IPv4 CIDR **10.0.3.0/24** and **10.0.4.0/24**.

NAT Gateway and Elastic IP

1. In the VPC console, choose **Elastic IPs** located on the left navigation panel.
2. Select the Elastic IP named **nat-gateway-eip**.
3. Confirm that it is allocated and has a Public IPv4 address.

Elastic IP addresses (1/1) Info						
Actions Allocate Elastic IP address						
Name	Allocated IPv4 address	Type	Allocation ID	Reverse DNS record	Associated instance ID	...
nat-gateway-eip	3.213.88.58	Public IP	eipalloc-0dde0c8626c1fad34	-	-	...

3.213.88.58	
View IP address usage and recommendations to release unused IPs with Public IP insights 	
Summary Tags	

Summary

Allocated IPv4 address 3.213.88.58	Type Public IP	Allocation ID eipalloc-0dde0c8626c1fad34	Reverse DNS record -
Association ID eipassoc-0ce94f0f6f20f8c79	Scope VPC	Associated instance ID -	Private IP address 10.0.1.93
Network interface ID eni-0f8e458596bc866b3	Network interface owner account ID 730335208305	Public DNS ec2-3-213-88-58.compute-1.amazonaws.com	NAT Gateway ID nat-0e3faff75953db2dd (nat-gateway)
Address pool Amazon	Network border group us-east-1	Service managed -	

4. Scroll down and click **NAT gateways**.
5. Select the NAT Gateway named **nat-gateway**.
6. Confirm that it belongs to the same VPC (ID: **vpc-0e6ddc9cedfbc3776**) and has the same **Primary public IPv4 address** allocated to it (e.g. **3.213.88.58**).

NAT gateways (1/1) Info						
Actions Create NAT gateway						
Name	NAT gateway ID	Connectivity...	State	State ...	Primary public I...	Primary private ...
nat-gateway	nat-0e3faff75953db2dd	Public	Available	-	3.213.88.58	10.0.1.93

nat-0e3faff75953db2dd / nat-gateway							
Details Secondary IPv4 addresses Monitoring Tags							
<p>Details</p> <table border="0"> <tr> <td style="width: 50%;"> NAT gateway ID nat-0e3faff75953db2dd </td> <td style="width: 50%;"> Connectivity type Public </td> </tr> <tr> <td> NAT gateway ARN arn:aws:ec2:us-east-1:730335208305:natgateway/nat-0e3faff75953db2dd </td> <td> Primary public IPv4 address 3.213.88.58 </td> </tr> <tr> <td> VPC vpc-0e6ddc9cedfbc3776 / ecs-vpc </td> <td> Subnet subnet-0ef79a97783673750 / public-subnet-a </td> </tr> </table>		NAT gateway ID nat-0e3faff75953db2dd	Connectivity type Public	NAT gateway ARN arn:aws:ec2:us-east-1:730335208305:natgateway/nat-0e3faff75953db2dd	Primary public IPv4 address 3.213.88.58	VPC vpc-0e6ddc9cedfbc3776 / ecs-vpc	Subnet subnet-0ef79a97783673750 / public-subnet-a
NAT gateway ID nat-0e3faff75953db2dd	Connectivity type Public						
NAT gateway ARN arn:aws:ec2:us-east-1:730335208305:natgateway/nat-0e3faff75953db2dd	Primary public IPv4 address 3.213.88.58						
VPC vpc-0e6ddc9cedfbc3776 / ecs-vpc	Subnet subnet-0ef79a97783673750 / public-subnet-a						
State Available	State message -						
Primary private IPv4 address 10.0.1.93	Primary network interface ID eni-0f8e458596bc866b3						
Created Tuesday, November 18, 2025 at 14:41:28 GM T+5	Deleted -						

Verify Public Route Table and Routes Creation

1. In the VPC console, choose **Route tables** located on the left navigation panel.
2. Select the route table named **Public-Route-Table**.
3. Check that it contains a route to the Internet Gateway with the destination **0.0.0.0/0**.
4. Under **Subnet Associations**, verify that **public-subnet-a** and **public-subnet-b** are associated with this route table as shown under **Explicit subnet associations (2)**.

The screenshot shows two route tables:

- rtb-0236706e829242711 / Public-Route-Table** (selected):
 - Explicit subnet associations (2)**:

Name	Subnet ID	IPv4 CIDR	IPv6 CIDR
public-subnet-a	subnet-0f1fb10d9e90663ce	10.0.1.0/24	-
public-subnet-b	subnet-03b536cb1b6ccba17	10.0.2.0/24	-
- rtb-06e3a5abddf090d4b / Public-Route-Table**:
 - Routes (2)**:

Destination	Target	Status	Propagated	Route Origin
0.0.0.0/0	igw-094a4675e1602cedf	Active	No	Create Route
10.0.0.16	local	Active	No	Create Route Table

Verify Public Route Table and Routes Creation

1. In the VPC console, choose **Route tables** located on the left navigation panel.
2. Select the route table named **Private-Route-Table**.
3. Check that it contains a route to the NAT gateway with the destination **0.0.0.0/0**.
4. Under **Subnet Associations**, verify that **private-subnet-a** and **private-subnet-b** are associated with this route table as shown under **Explicit subnet associations (2)**.

The screenshot shows three route tables:

- rtb-0e11792e436b08268 / Private-Route-Table** (selected):
 - Routes (2)**:

Destination	Target	Status	Propagated	Route Origin
0.0.0.0/0	nat-0e3faaff75953db2dd	Active	No	Create Route
10.0.0.16	local	Active	No	Create Route Table
- rtb-0e63a5abddf090d4b / Public-Route-Table**:
 - Routes (2)**:

Destination	Target	Status	Propagated	Route Origin
0.0.0.0/0	igw-094a4675e1602cedf	Active	No	Create Route
10.0.0.16	local	Active	No	Create Route Table
- rtb-0e11792e436b08268 / Private-Route-Table**:
 - Routes (2)**:

Destination	Target	Status	Propagated	Route Origin
0.0.0.0/0	nat-0e3faaff75953db2dd	Active	No	Create Route
10.0.0.16	local	Active	No	Create Route Table

Route tables (1/5) Info

Last updated 16 minutes ago Actions Create route table

Find route tables by attribute or tag

Name	Route table ID	Explicit subnet associations	Edge associations	Main	VPC
Private-Route-Table	rtb-0e11792e436b08268	2 subnets	-	No	vpc-0e6ddc9cedfbc3776
-	rtb-05eb2b7ef1f93b8a8	-	-	Yes	vpc-0e9378abac897865
-	rtb-09622852a7c433c5b	6 subnets	-	Yes	vpc-040ff5671e02b7f149
Public-Route-Table	rtb-06e3a5abdf090d4b	2 subnets	-	No	vpc-0e6ddc9cedfbc3776

rtb-0e11792e436b08268 / Private-Route-Table

Details Routes Subnet associations Edge associations Route propagation Tags

Explicit subnet associations (2)

Edit subnet associations

Name	Subnet ID	IPv4 CIDR	IPv6 CIDR
private-subnet-b	subnet-046c577b01ca4b010	10.0.4.0/24	-
private-subnet-a	subnet-0ff1749746b583d5a	10.0.3.0/24	-

Verify Security Group Creation

Public Security Group

- In the VPC console, choose **Security groups** located on the left navigation panel.
- Locate the **public-sg** security group
- It should have the following inbound rules:

Type	Protocol	Port range	Source
HTTP	TCP	80	0.0.0.0/0

- Outbound rules should allow all traffic.

Security Groups (1/3) Info

Actions Export security groups to CSV Create security group

Find security groups by attribute or tag

VPC ID = vpc-0e6ddc9cedfbc3776

Clear filters

Name	Security group ID	Security group name	VPC ID	Description
-	sg-0b7540e290f45ae1	default	vpc-0e6ddc9cedfbc3776	default VPC security group
public-sg	sg-06e2b01e21cf91100	public-sg	vpc-0e6ddc9cedfbc3776	Allows HTTP traffic from the internet
private-sg	sg-055550042037788d0	private-sg	vpc-0e6ddc9cedfbc3776	Allows MySQL traffic from ECS (public-sg)

sg-06e2b01e21cf91100 - public-sg

Details Inbound rules Outbound rules Sharing - new VPC associations - new Tags

Inbound rules (1)

Manage tags Edit inbound rules

Search

Name	Security group rule ID	IP version	Type	Protocol	Port range	Source
-	sgr-06ec63ec8a50694d	IPv4	HTTP	TCP	80	0.0.0.0/0

Private Security Group

- In the VPC console, choose **Security groups** located on the left navigation panel.
- Locate the **public-sg** security group
- It should have the following inbound rules:

Type	Protocol	Port range	Source
MySQL/Aurora	TCP	3306	public-sg

4. Outbound rules should allow all traffic.

The screenshot shows the AWS Security Groups Inbound rules page for a security group named 'sg-055550042037788d0 - private-sg'. The 'Inbound rules' tab is selected. There is one rule listed:

Name	Security group rule ID	IP version	Type	Protocol	Port range	Source
-	sgr-0ba6058dcca74c3e5	-	MySQL/Aurora	TCP	3306	sg-06e2b01e21cf91100 / public-sg

At this point, VPC, Internet Gateway, NAT gateway, elastic IP, subnets, route tables, routes and security groups are confirmed as deployed and functional.

Verify ECR Creation

1. Sign in to the AWS Management Console.
2. Navigate to **ECR** service using the search bar at the top and then click **Repositories** under **Private registry** located in the left navigation bar.
3. Verify that the **wordpress** repository was created with a URI that matches the account ID.

The screenshot shows the AWS ECR Private registry Repositories page. The left sidebar shows 'Amazon Elastic Container Registry' with 'Private registry' expanded, showing 'Repositories'. The main area shows 'Private repositories (1)'. One repository is listed:

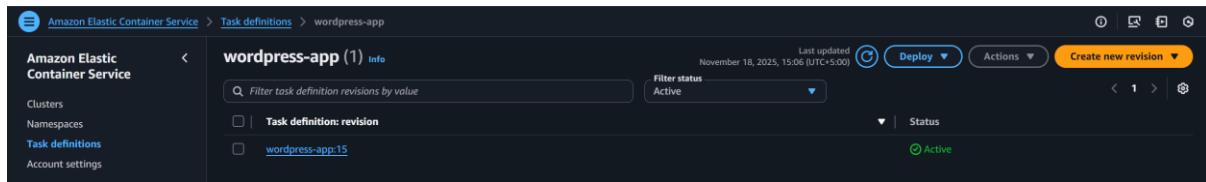
Repository name	URI	Created at	Tag immutability	Encryption type
wordpress	730335208305.dkr.ecr.us-east-1.amazonaws.com/wordpress	November 18, 2025, 14:41:09 (UTC+05)	Mutable	AES-256

Verify Task Definition Creation

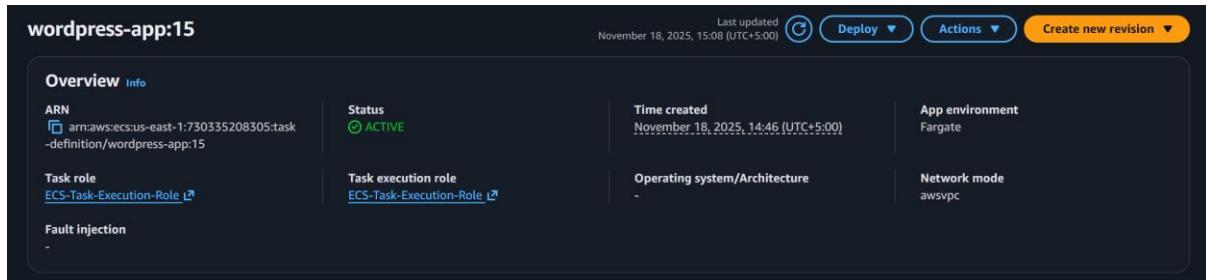
1. Sign in to the AWS Management Console.
2. Navigate to **ECS** service using the search bar at the top and then click **Task definitions** located in the left navigation bar.
3. Verify that a task definition named **wordpress-app** is created.

The screenshot shows the AWS Elastic Container Service Task definitions page. The left sidebar shows 'Amazon Elastic Container Service' with 'Task definitions' selected. The main area shows 'Task definitions (1)'. One task definition is listed:

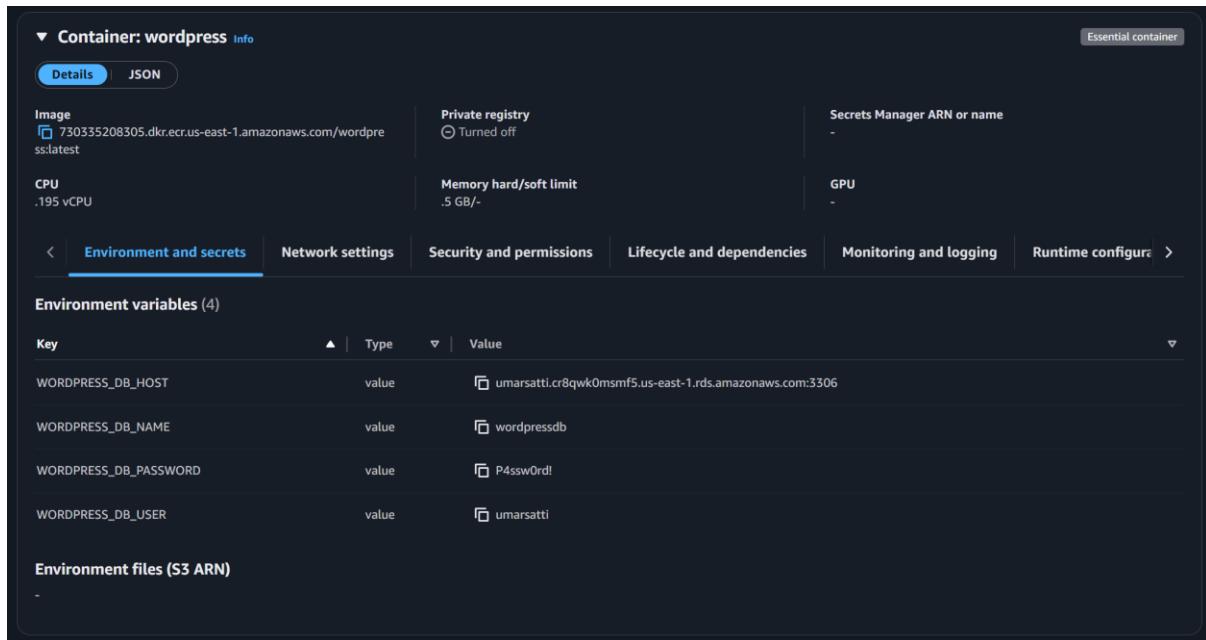
Task definition	Status of last revision
wordpress-app	Active



- Click on **wordpress-app** then and then click on **wordpress-app** again to view details.



- Confirm that it has an **Active** status.
- Click **Containers** and scroll down to confirm that **wordpress** container has been defined with proper **Environment Variables** as shown below.



- As shown in the image above, the environment variables value (as key-value pairs) correctly corresponds to RDS endpoint, database name, username, and password.

Verify ECS Cluster Creation

- Sign in to the AWS Management Console.
- Navigate to **ECS** service using the search bar at the top.
- Click **Clusters** located in the left navigation bar.

The screenshot shows the 'Clusters' section of the AWS ECS console. A single cluster named 'wordpress-ecs-cluster' is listed. The cluster has 1 service, 2 pending tasks, and 0 running tasks across 0 EC2 instances. It is associated with 'Container Insights with enhanced observability' and is using the Fargate provider strategy.

4. Verify that a cluster named **wordpress-ecs-cluster** is already created.
5. Click on this **wordpress-ecs-cluster**.

The screenshot shows the 'Cluster overview' page for the 'wordpress-ecs-cluster'. It displays the ARN (arn:aws:ecs:us-east-1:730335208305:cluster/wordpress-ecs-cluster), status (Active), CloudWatch monitoring (Container Insights with enhanced observability), and registered container instances. Under 'Services', there is one active service named 'wordpress-service'. Under 'Tasks', there are 1 pending task and 0 running tasks.

Services

Service name	ARN	Status	Task definition	Deployments and tasks
wordpress-service	arn:aws:ecs:us-e... (Active)	REPLICA	FARGATE	wordpress-app:15 (0/2 Tasks running)

6. There should be a **Service** running with an **Active** status and **Deployments and tasks** showing **Tasks** running.
7. Click on **Tasks** tab. It should show Tasks in a **Stopped** state. This is because the **Tasks** are failing as there are **NO** images that have been **Pushed** to the ECR repositories (e.g. wordpress image).

The screenshot shows the 'Service overview' page for the 'wordpress-service' within the 'wordpress-ecs-cluster'. The service is active with 2 desired tasks, 2 pending tasks, and 0 running tasks. The task definition is 'wordpress-app:15' and the deployment status is 'In progress'. The 'Tasks' tab is selected, showing a list of 12 tasks. All tasks are in a 'Stopped' state, with their last status also being 'Stopped'. The tasks were created 14 seconds ago and started by 'ecs-svc/99559860856...'.

Task	Last status	Desired st...	Task definition	Health sta...	Created at	Started by
e9de8b929a234b7887fecb0b866fae0f	Provisioning	Running	wordpress-app:15	Unknown	14 seconds ago	ecs-svc/99559860856...
f06e8e8296784f879f30d2c4ce674ca4	Provisioning	Running	wordpress-app:15	Unknown	14 seconds ago	ecs-svc/99559860856...
1165ac28fdbcb46559b672313a95d5ff	Stopped ...	Stopped	wordpress-app:15	Unknown	18 minutes ago	ecs-svc/99559860856...
4111584351124831b22ee340eadc044	Stopped ...	Stopped	wordpress-app:15	Unknown	7 minutes ago	ecs-svc/99559860856...

Verify RDS DB Subnet and Instance Creation

1. Sign in to the AWS Management Console.
2. Navigate to **RDS** service using the search bar at the top.
3. Click **Subnet groups** located in the left navigation bar.

4. There should be a DB subnet group present that is named **private-db-subnet**.

The screenshot shows the 'Subnet groups' section of the Aurora and RDS console. A single subnet group named 'private-db-subnet' is listed. The group is managed by Terraform, has a status of 'Complete', and is associated with VPC ID 'vpc-0e6ddc9cedfbc3776'. The interface includes a search bar, edit, delete, and create buttons.

Name	Description	Status	VPC
private-db-subnet	Managed by Terraform	Complete	vpc-0e6ddc9cedfbc3776

5. Click on this DB subnet to view details.
6. It should have **private-subnet-a** and **private-subnet-b** as its subnets.

The screenshot shows the details of the 'private-db-subnet'. It lists the VPC ID, ARN, supported network types (IPv4), and a description managed by Terraform. Below this, a table shows the two subnets: 'private-subnet-b' in 'us-east-1b' with CIDR block '10.0.4.0/24' and 'private-subnet-a' in 'us-east-1a' with CIDR block '10.0.3.0/24'.

Availability zone	Subnet name	Subnet ID	CIDR block
us-east-1b	private-subnet-b	subnet-046c577b01ca4b010	10.0.4.0/24
us-east-1a	private-subnet-a	subnet-0ff1749746b583d5a	10.0.3.0/24

7. Go back to the RDS console and click **Databases**.
8. There should be an RDS DB Instance present that is named **umarsatti**.

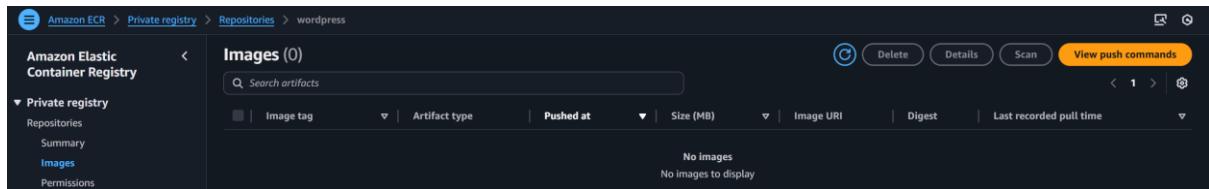
The screenshot shows the 'Databases' section of the Aurora and RDS console. A single database instance named 'umarsatti' is listed. The instance is available, running MySQL Community engine, and is associated with the 'db.t3.micro' provisioned plan. The interface includes a search bar, group resources, modify, actions, and create database buttons.

DB identifier	Status	Role	Engine	Region ...	Size	Recommendations
umarsatti	Available	Instance	MySQL Community	us-east-1a	db.t3.micro	

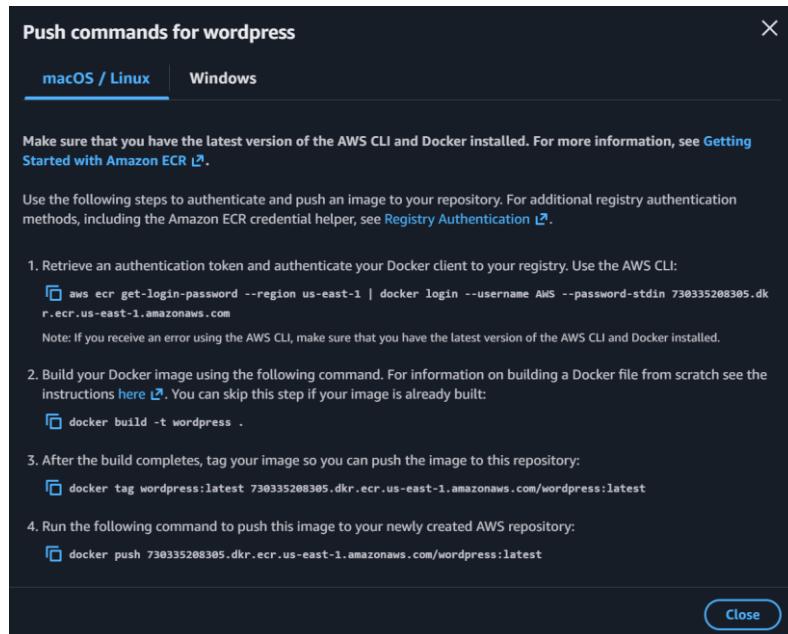
Task 1.10.2: Push WordPress Image to ECR

Push the images to ECR

- From the AWS Management Console, navigate to the **ECR** service using the search bar at the top
- Click **Repositories** located under **Private registry** in the left navigation bar.
- Perform the following:
 - Click **wordpress** repository (it should be empty).



- Click on **View push commands** button on the top right.
- Run the commands shown.



Note: Make sure Docker Desktop is installed and running for these commands to work. Additionally for step 2, run **docker pull wordpress** instead of **docker build -t wordpress**. This pulls the official WordPress image from DockerHub.

```
umars@Umar-Satti MINGW64 /d/Cloudelligent/Task-4-ECS-Fargate-RDS-WordPress-with-Terraform
● $ aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 730335208305.dkr.ecr.us-east-1.amazonaws.com
Login Succeeded

umars@Umar-Satti MINGW64 /d/Cloudelligent/Task-4-ECS-Fargate-RDS-WordPress-with-Terraform/terraform
● $ docker pull wordpress:latest
latest: Pulling from library/wordpress
0e4bc2bd6656: Pull complete
46605a656915: Pull complete
f7c9fe45c578: Pull complete
8924dcffec96: Pull complete
6a307b08e6b5: Pull complete
bf66ebb0c9a9: Pull complete
e53b88f1a2a7: Pull complete
6067d09e211b: Pull complete
b59bc243be75: Pull complete
86a7cc0b1b7d: Pull complete
be2e241aa1de: Pull complete
02189dee7fe6: Pull complete
b5f62298dabc: Pull complete
e86e5731331c: Pull complete
4f4fb700ef54: Pull complete
4a2bea9f290c: Pull complete
d8c4dd297ff: Pull complete
f4a7c4b52246: Pull complete
2c420b5256e3: Pull complete
557c18e28a88: Pull complete
db73bbb2ae8f: Pull complete
a5be4a15e86b: Pull complete
172fdd945699: Pull complete
4beba7a66238: Pull complete
Digest: sha256:5dfda843a925a1da229eadfc5099cf9ce3d5481c6dd872fc7cdcf920951e663d
Status: Downloaded newer image for wordpress:latest
docker.io/library/wordpress:latest

What's next:
  View a summary of image vulnerabilities and recommendations → docker scout quickview wordpress:latest

umars@Umar-Satti MINGW64 /d/Cloudelligent/Task-4-ECS-Fargate-RDS-WordPress-with-Terraform/terraform
○ $
```

```
umars@Umar-Satti MINGW64 /d/Cloudelligent/Task-4-ECS-Fargate-RDS-WordPress-with-Terraform/terraform
● $ docker tag wordpress:latest 730335208305.dkr.ecr.us-east-1.amazonaws.com/wordpress:latest
```

```
umars@Umar-Satti MINGW64 /d/Cloudelligent/Task-4-ECS-Fargate-RDS-WordPress-with-Terraform/terraform
● $ docker push 730335208305.dkr.ecr.us-east-1.amazonaws.com/wordpress:latest
The push refers to repository [730335208305.dkr.ecr.us-east-1.amazonaws.com/wordpress]
93895b9c0c35: Pushed
7853a0439798: Pushed
fe8eca22a016: Pushed
07713e4f529e: Pushed
9780eeff63d5: Pushed
a40b04fb48e0: Pushed
3d10e3005795: Pushed
63035e698d41: Pushed
24fabd4e2fd9: Pushed
5f70bf18a086: Pushed
d57b785a1d6d: Pushed
98fc0bf9532f: Pushed
60193afa7423: Pushed
971f66171313: Pushed
c19583d1051e: Pushed
26b21ea4e330: Pushed
3a5c6176315b: Pushed
5220220157ff: Pushed
72dbab29b306: Pushed
efd9cff39906: Pushed
6c9577238c27: Pushed
2feb3f62c88e: Pushed
953c38baa70a: Pushed
70a290c5e58b: Pushed
latest: digest: sha256:1d167155a5e94c2a8ae7de9d419acd6ffa74a97a8201d7dde7b07726e7b6a580 size: 5330
```

Verify the images have been pushed

1. Click the **wordpress** repository.
2. A **wordpress** image should be available with the **latest** tag.

Images (1)

Search artifacts

Image tag | Artifact type | Pushed at | Size (MB) | Image URI | Digest | Last recorded pull time

Latest Image November 18, 2025, 15:21:49 (UTC+05) 264.49 Copy URI sha256:1d167155a5e94c...

Verify the Service and Tasks are running

1. Navigate to **ECS** service using the search bar at the top and then click **Clusters** located in the left navigation bar.
2. Under the **Tasks** tab, it should display two tasks running successfully.

Clusters (1) Info

Search clusters

Last updated November 18, 2025, 15:22 (UTC+5:00) Create cluster

Cluster Services Tasks Container instances CloudWatch monitoring Capacity provider strat

wordpress-ecs-cluster 1 0 Pending | 2 Running 0 EC2 Container Insights with enhanced observability Fargate

3. Click the **wordpress-ecs-cluster** and under **Services** tab, it should display again that 2/2 tasks are running successfully.

wordpress-ecs-cluster Fargate

Last updated November 18, 2025, 15:23 (UTC+5:00) Actions Launch

Cluster overview

ARN arn:aws:ecs:us-east-1:730335208305:cluster/wordpress-ecs-cluster Status Active CloudWatch monitoring Container Insights with enhanced observability View in CloudWatch Registered container instances

Services Tasks Infrastructure Updated Metrics Scheduled tasks Configuration Event history Tags

Services (1/1) Info Last updated November 18, 2025, 15:22 (UTC+5:00) Manage tags Update Delete service Create

Filter services by value Any launch type Any scheduling strategy

Service name	ARN	Status	Scheduling ...	Launch type	Task definition	Deployments and tasks
wordpress-service	arn:aws:ecs:us-e...	Active	REPLICA	FARGATE	wordpress-app:15	2/2 Tasks running

4. Optionally, explore this console for additional information such as Tasks, Logs, Health checks, Metrics etc.

Task 1.10.3: Access the WordPress Application

Elastic Network Interface

- From the AWS Management Console, navigate to the **EC2** service using the search bar at the top.
- Scroll down and click **Network Interfaces** under **Network & Security**
- Two **ENIs** should be deployed with a **Public IPv4 address** which belong to **public-sg** security group and one **ENI** which belongs to **private-sg** security group.

Name	Network interface ID	Subnet ID	VPC ID	Availability Zone	Security group n...	Security group IDs
eni-0927ff6f01e83c647	subnet-0ff1749746b583d5a	vpc-0e6ddc9edfb3776	us-east-1a	private-sg	sg-0555500420377...	
eni-0f8e458596bc866b3	subnet-0ef79a97783673750	vpc-0e6ddc9edfb3776	us-east-1a	-	-	
eni-04222b3ff4cd34d3e	subnet-0ef79a97783673750	vpc-0e6ddc9edfb3776	us-east-1a	public-sg	sg-06e2b01e21cf91...	
eni-069955117ce21a9b2	subnet-0d6be7088204f5258	vpc-0e6ddc9edfb3776	us-east-1b	public-sg	sg-06e2b01e21cf91...	

- Copy the IPv4 address from one of these **public-sg** ENIs.
- Open the application using this Public IP **http://<eni-public-IP>**
 - Example: **http://54.226.219.4**
 - This takes the user to the URL: **/wp-admin/install.php**
- Confirm that the WordPress setup page is opening successfully as shown below.

English (United States)

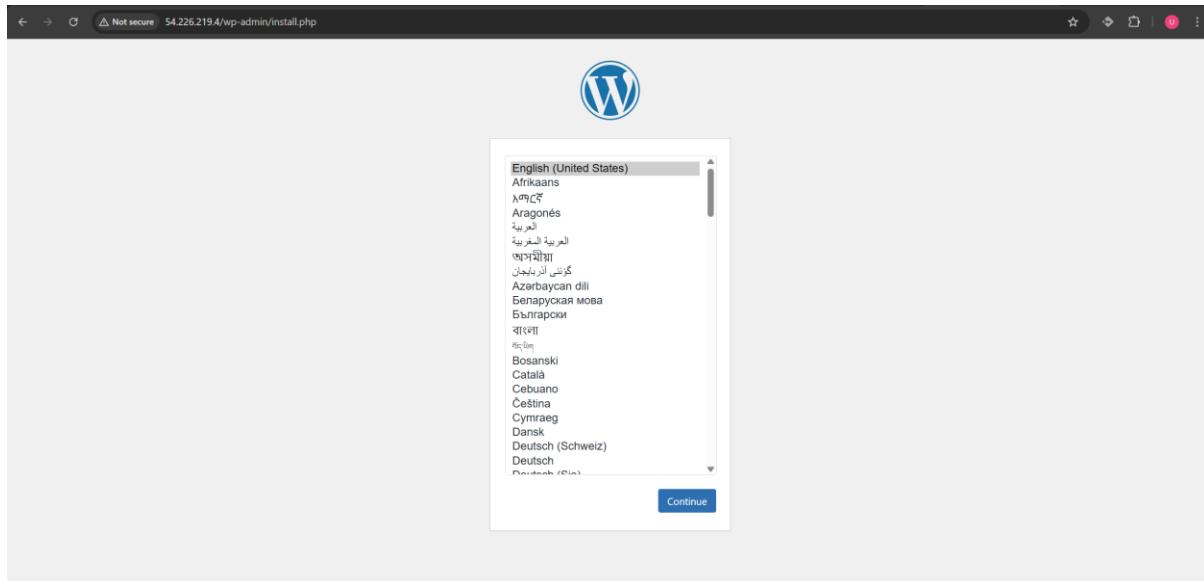
Afrikaans
العربية
العربية المغربية
অসমীয়া
گۈئىشلىپارچىلەن
Azərbaycan dil
Беларуская мова
Български
বাংলা
Bosanski
Català
Cebuano
Čeština
Cymraeg
Dansk
Deutsch (Schweiz)
Deutsch (Österreich)
Deutsch (Deutschland)

Continue

Task 1.10.4: Configure WordPress

Step 1: Open WordPress application

1. In browser, go to **http://<your-ec2-public-ip>/wp-admin/install.php**
 - Example: **http://54.226.219.4/wp-admin/install.php**



Step 2: Install 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 Umar Satti

Username umarsatti

Password 3RPsA1aTkBW3VnXbu Hide

Your Email umar123@example.com

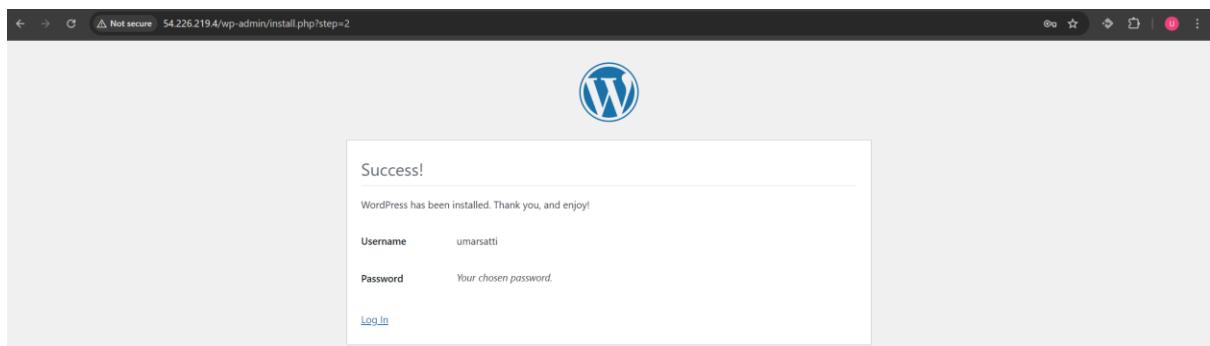
Search engine visibility Discourage search engines from indexing this site
It is up to search engines to honor this request.

Important: You will need this password to log in. Please store it in a secure location.

Double-check your email address before continuing.

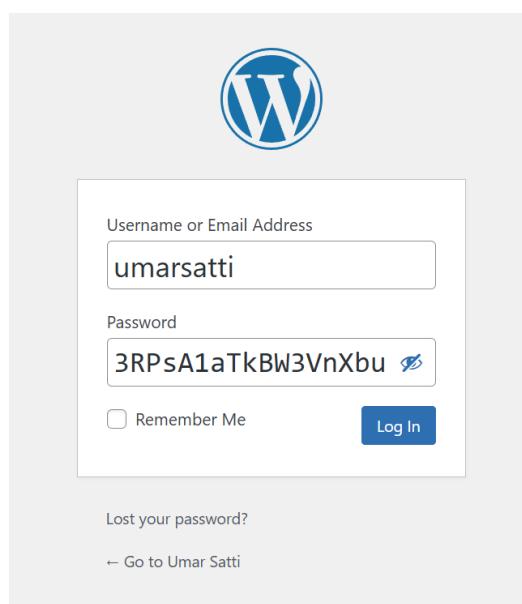
Install WordPress

- Once WordPress verifies database connection, it will display the installation page. To complete installation, fill out the required fields:
 - Site Title:** Umar Satti
 - Username:** umarsatti
 - Password:** 3RPsA1aTkBW3VnXbu)
 - Email:** umar123@example.com
- Click **Install WordPress**.
- After a few seconds, a success message is displayed:
 - “Success! WordPress has been installed. Thank you and enjoy!”

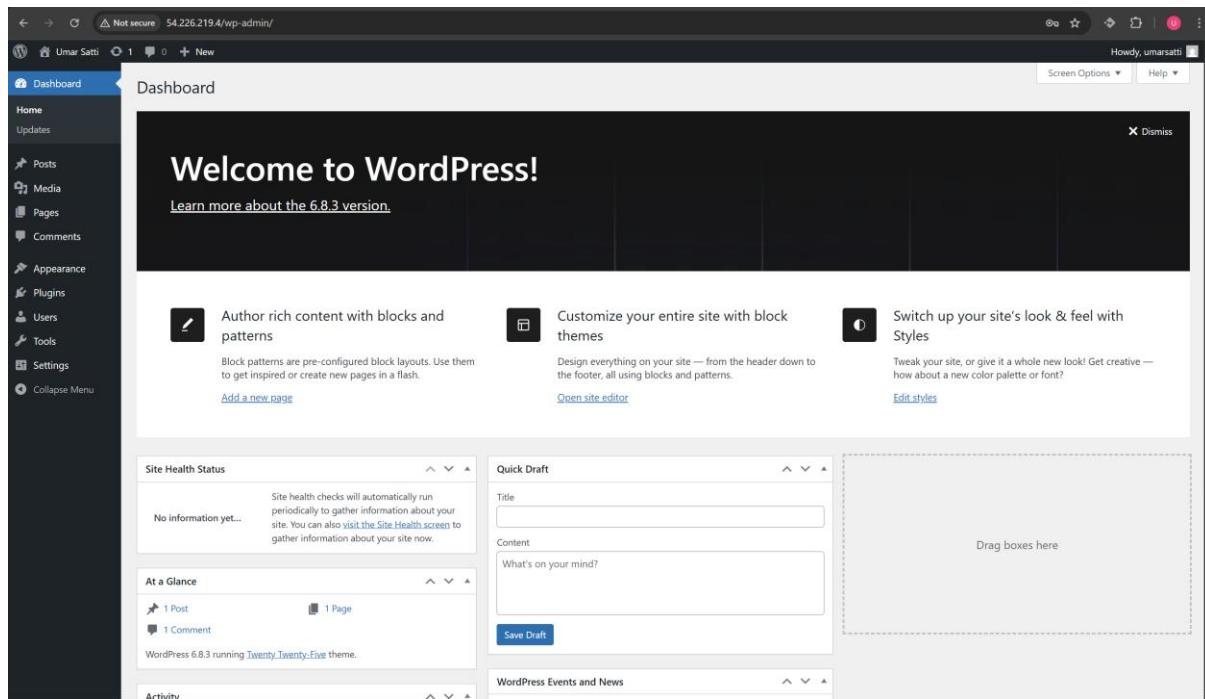


Step 3: Log In to the WordPress Admin Dashboard

- After installation is completed, click **Log In** and use the credentials provided earlier.
- Go to **http://<your-ec2-public-ip>/wp-login.php**
 - Example: http://13.220.176.179/wp-login.php
- Enter the username (umarsatti) and password (P@ssw0rd).
- Click **Log In**.



The page should automatically redirect to the WordPress Admin Dashboard, confirming that the website is now fully functional.



The AWS infrastructure, provisioned by Terraform, is now complete and verified. The deployment and initial configuration of the WordPress application were successful, meaning the application is healthy and accessible. The components defined in Terraform (VPC, Security groups, Task definition, ECR, ECS, and RDS) are used to create a live WordPress environment. This showcases how infrastructure-as-code simplifies complex deployments.

Task 1.11: Clean Up

Note: Make sure to manually delete images that were pushed to the ECR wordpress and mysql repositories by going into the ECR console before performing **terraform destroy** command.

To delete the resources, run **terraform destroy** or **terraform destroy --auto-approve**

Task 1.12: Troubleshooting

Issue 1: Difficulty Referencing the RDS Endpoint Correctly

Problem Description:

In a similar ECS-based project using Amazon RDS for MySQL, the RDS endpoint was not being referenced properly inside Terraform. This caused failures when ECS containers attempted to connect to the database.

Root Cause:

There was confusion about which RDS attribute exposed the actual endpoint address. Attempting to use incorrect attributes or incomplete references resulted in connection errors.

Solution:

Using the Terraform AWS provider documentation clarified that the correct attribute to expose the endpoint is `aws_db_instance.db_instance.address`. This value was then exported through an output variable, making it available to other modules such as the ECS Task Definition.

Issue 2: Incorrect Use of RDS Endpoint Inside the ECS Task Definition

Problem Description:

Even after correctly exporting the RDS endpoint, the value was incorrectly used within the ECS container environment variables. The port number (3306) was missing, causing WordPress to fail database connections.

Root Cause:

The `WORDPRESS_DB_HOST` environment variable must include both the RDS endpoint and the port number (default MySQL port is 3306). Using only `${var.rds_endpoint}` results in an incomplete host string.

Solution:

The environment variable was updated to append the port properly using:

- `${var.rds_endpoint}:3306`

This ensured the application had the complete database host information required for a successful MySQL connection.

Conclusion

This project successfully demonstrated how to automate infrastructure provisioning and WordPress deployment on AWS using ECS Fargate and RDS with Terraform. By following this documentation, users can create the same setup to launch WordPress using AWS.