

# WordPress Deployment on AWS ECS Fargate with RDS Documentation

## Table of Contents

WordPress Deployment on AWS ECS Fargate with RDS Documentation.....	1
1. Prerequisites .....	1
2. Objective .....	2
3. Architecture Diagram General Workflow of the Infrastructure .....	3
4. Project Structure .....	4
5. Files, Purpose and Explanation.....	4
5.1 provider.tf.....	4
5.2 backend.tf.....	4
5.3 variables.tf.....	5
5.4 terraform.tfvars .....	5
5.5 outputs.tf .....	5
6. Module Explanations.....	5
6.1 VPC Module.....	5
6.2 Security Module .....	6
6.3 RDS Module.....	6
6.4 ECR Module.....	7
6.5 ECS Module .....	7
main.tf.....	8
7. Terraform Deployment Commands.....	8
Initialize providers and modules .....	8
Format code (optional but recommended).....	8
Validate configuration.....	8
View changes before applying .....	8
Deploy infrastructure.....	8
8. Testing and Validation .....	9
8.1 Verify AWS Resources in Console .....	9
8.2 Push WordPress Image to Amazon ECR .....	10
8.3 Validate ECS Fargate Deployment .....	11
8.4 Accessing the WordPress Frontend.....	11
8.5 Connecting WordPress to RDS .....	12
8.6 Post-Deployment Verification .....	13
9. Troubleshooting .....	13
Issue 1 : ECS Task Starts but WordPress Cannot Reach Database.....	13
Issue 2 : ECS Task Fails to Pull Image .....	14
Issue 3 : RDS Setup Fails with Password Error .....	14
10. Cleanup.....	14
Conclusion.....	15

## 1. Prerequisites

Before starting, ensure the following requirements are satisfied:

20 November 2025

- Terraform installed (v1.0+ recommended)
- AWS CLI configured with valid IAM credentials
- An S3 bucket **created manually** for Terraform remote backend
  - (Best practice for production to avoid storing state locally)
- Basic networking understanding (VPC, Subnets, Security Groups, IAM)
- Docker installed (only needed if pushing custom images to ECR)

## 2. Objective

Deploy a **production-style WordPress application using Terraform**, hosted on:

- **ECS Fargate** (fully serverless compute)
- **Amazon RDS MySQL** (managed database)
- **VPC with two public subnets**
- **Security groups allowing communication between services**
- **Ubuntu-based WordPress container from Docker Hub**  
(wordpress:latest)

The deployment goal:

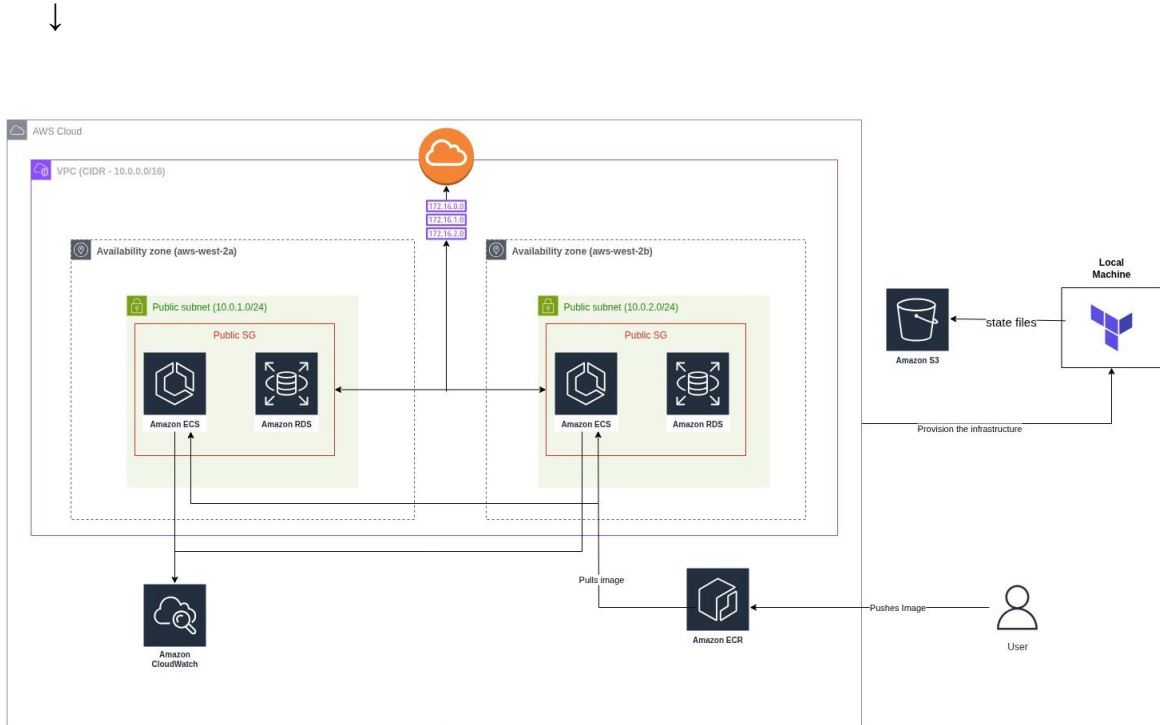
- ECS Fargate runs the WordPress container
- Application connects securely to the RDS MySQL instance
- Infrastructure follows modular Terraform design, similar to enterprise layouts

## 3. Architecture Diagram

20 November 2025

## General Workflow of the Infrastructure

You Push Image → ECR

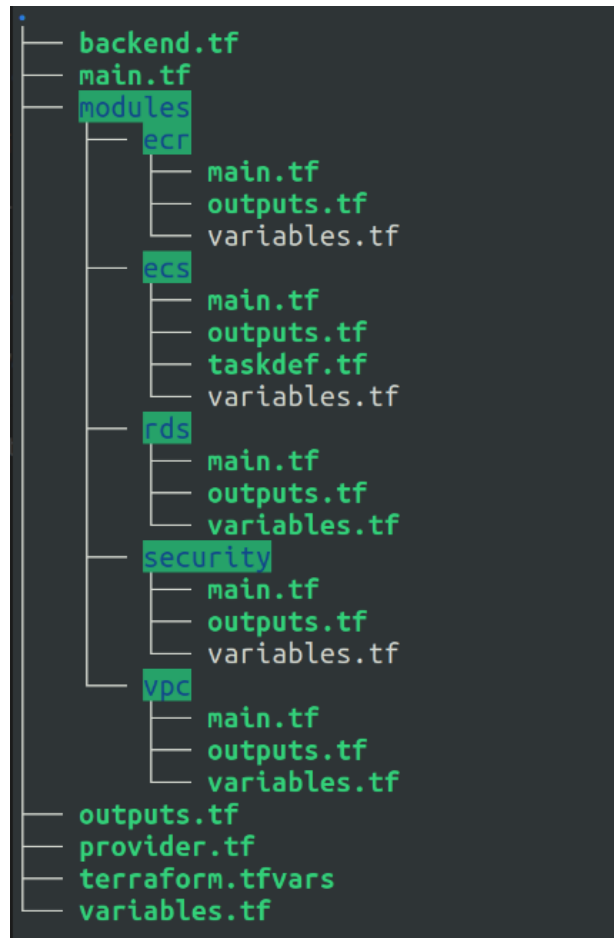


### ECS Pulls Image → ECS Task Starts (WordPress Container)

ECS Task Connects → RDS (Database)

You Access ECS Task IP → WordPress Frontend Loads in Browser

## 4. Project Structure



## 5. Files, Purpose and Explanation

### 5.1 provider.tf

- Configures the AWS provider pointing to us-west-2.
- Ensures Terraform knows which cloud provider and region to deploy to.

### 5.2 backend.tf

- Configures Terraform to store state remotely in S3.
- Enhances:
  - multi-user collaboration
  - state durability
- S3 bucket is created manually and referenced here.

20 November 2025

## 5.3 variables.tf

- Central location for input variables:
  - VPC CIDRs
  - Subnet lists
  - RDS DB username/password
  - WordPress image name (wordpress:latest)
  - Instance types / ECS configuration

## 5.4 terraform.tfvars

Defines real deployment values, such as:

- AWS region
- VPC CIDR
- Subnets
- DB credentials
- Container image name

Terraform loads these automatically.

## 5.5 outputs.tf

Displays useful post-deployment information, such as:

- ECS Load Balancer or Service endpoint (if configured)
- RDS connection endpoint
- VPC ID and subnet IDs
- ECS cluster name

# 6. Module Explanations

## 6.1 VPC Module

Files:

```
modules/vpc/main.tf
modules/vpc/variables.tf
modules/vpc/outputs.tf
```

Function:

20 November 2025

- Creates:
  - AWS VPC
  - Two public subnets (as per requirements)
  - Internet Gateway
  - Route tables associating subnets with the IGW

Reason:

- ECS tasks must reach the internet to pull Docker images.
- RDS deployed in public subnet for simplicity in this minimal setup.

Outputs usually include:

- `vpc_id`
- `public_subnet_ids`

## 6.2 Security Module

Files:

```
modules/security/main.tf
modules/security/variables.tf
modules/security/outputs.tf
```

Creates:

- Security group for WordPress ECS tasks:
  - Allows HTTP/HTTPS inbound
  - Allows outbound to RDS
- Security group for RDS:
  - Only accepts inbound MySQL (3306) from ECS SG
  - Everything else denied

This ensures controlled network access.

## 6.3 RDS Module

Files:

```
modules/rds/main.tf
modules/rds/variables.tf
modules/rds/outputs.tf
```

Responsible for:

- Deploying an Amazon RDS MySQL instance (`db.t3.micro`)

20 November 2025

- Using credentials provided via variables
- Placed in a VPC subnet
- Security group applied

Outputs include:

- RDS endpoint  
Used later by ECS task definition.

## 6.4 ECR Module

Files:

```
modules/ecr/main.tf
modules/ecr/variables.tf
modules/ecr/outputs.tf
```

Creates:

- An AWS Elastic Container Registry repository
- Allows pushing custom WordPress images if needed

If using Docker Hub image (`wordpress:latest`) – module still exists but may be unused.

## 6.5 ECS Module

Files:

```
modules/ecs/main.tf
modules/ecs/variables.tf
modules/ecs/outputs.tf
modules/ecs/taskdef.tf
```

Responsible for:

- Creating the ECS Cluster
- Defining an ECS Service running Fargate
- Creating an IAM Execution Role so the task can pull images and write logs
- Creating an ECS Task Definition (`taskdef.tf`) including:
  - WordPress container
  - Environment variables:
    - `WORDPRESS_DB_HOST`
    - `WORDPRESS_DB_USER`
    - `WORDPRESS_DB_PASSWORD`
    - `WORDPRESS_DB_NAME`

20 November 2025

- Log configuration
- CPU / Memory sizing

This is the core of the application deployment.

The ECS service ensures:

- The container is continuously running
- If stopped, AWS restarts it automatically

## **main.tf**

This file ties all modules together:

- Calls VPC module
- Calls security groups module
- Deploys RDS with the correct DB credentials
- Deploys ECS with the correct container configuration
- Passes outputs between modules (example: RDS endpoint → ECS task definition)

It is the central orchestration file.

# **7. Terraform Deployment Commands**

## **Initialize providers and modules**

```
terraform init
```

## **Format code (optional but recommended)**

```
terraform fmt
```

## **Validate configuration**

```
terraform validate
```

## **View changes before applying**

```
terraform plan
```

## **Deploy infrastructure**

```
terraform apply
```

Once complete, Terraform prints outputs, including:

- RDS endpoint

20 November 2025

- ECS service details

```
Apply complete! Resources: 3 added, 0 changed, 0 destroyed.
```

#### Outputs:

```
ecr_repository_url = "504649076991.dkr.ecr.us-west-2.amazonaws.com/imag  
ecs_cluster_name = "wordpress-cluster"  
ecs_service_name = "wordpress-cluster-svc"  
public_subnet_ids = [  
  "subnet-01f610e4dd58c1686",  
  "subnet-0f9434faaf60a6bfa",  
]  
rds_endpoint = "wp-mysql.cuoj6ci9wmw9.us-west-2.rds.amazonaws.com"  
vpc_id = "vpc-08634b8f433e73b10"
```

## 8. Testing and Validation

After deploying the Terraform configuration, the following steps were performed to verify that the infrastructure and WordPress deployment were working correctly.

### 8.1 Verify AWS Resources in Console

Once `terraform apply` completed successfully, the first step was to confirm that all major AWS resources were deployed as expected:

- VPC
- Public subnets
- Security groups
- RDS MySQL instance
- ECS Cluster
- ECS Service and running Task
- ECR Repository (if using ECR)

#### Validation Steps

1. Log in to the AWS Console
2. Open **VPC Console** → confirm VPC CIDR and subnets
3. Check **RDS Console** → ensure DB instance is available
4. Open **ECS Console** → Cluster → verify task is running
5. Ensure **ECS Service targets are healthy**
6. **If using ECR: open ECR Console and verify the repository exists**

## 8.2 Push WordPress Image to Amazon ECR

To make the WordPress container available to ECS, the image was pushed to ECR.

- Go to AWS Console for ECR, select your private registry and goto images.
- Click on View push commands on the right and paste them into the aws configured cli.

### Push commands for image

macOS / Linux

Windows

Make sure that you have the latest version of the AWS CLI and Docker installed. For more information, see [Getting Started with Amazon ECR](#).

Use the following steps to authenticate and push an image to your repository. For additional registry authentication methods, including the Amazon ECR credential helper, see [Registry Authentication](#).

1. Retrieve an authentication token and authenticate your Docker client to your registry. Use the AWS CLI:  

```
aws ecr get-login-password --region us-west-2 | docker login --username AWS --password-stdin 504649076991.dkr.ecr.us-west-2.amazonaws.com
```

Note: If you receive an error using the AWS CLI, make sure that you have the latest version of the AWS CLI and Docker installed.
2. Build your Docker image using the following command. For information on building a Docker file from scratch see the instructions [here](#). You can skip this step if your image is already built:  

```
docker build -t image .
```
3. After the build completes, tag your image so you can push the image to this repository:  

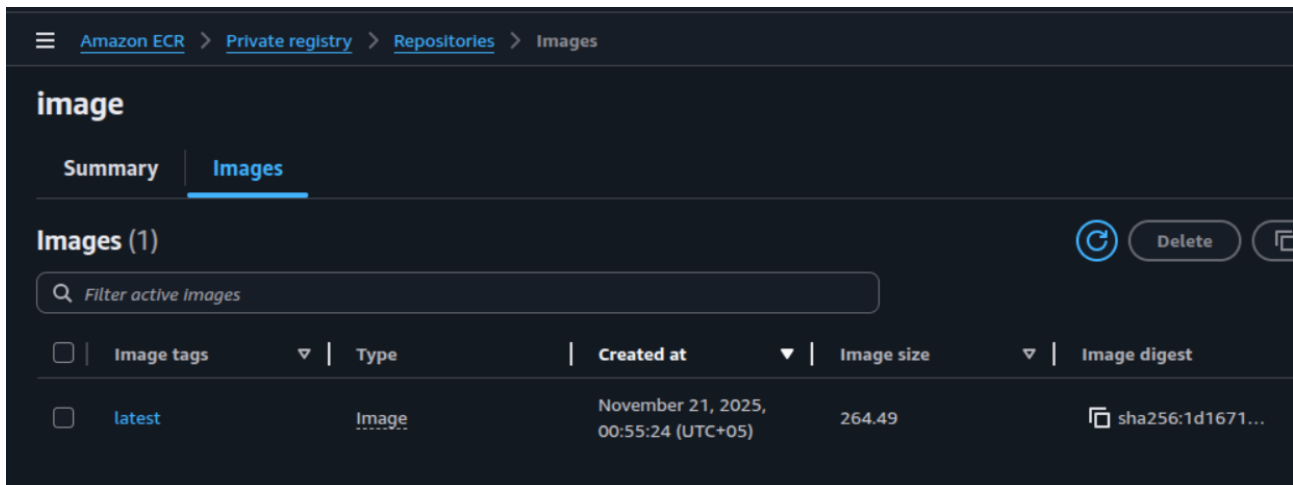
```
docker tag image:latest 504649076991.dkr.ecr.us-west-2.amazonaws.com/image:latest
```
4. Run the following command to push this image to your newly created AWS repository:  

```
docker push 504649076991.dkr.ecr.us-west-2.amazonaws.com/image:latest
```

Close

Then confirm the image has been pushed:

20 November 2025



## 8.3 Validate ECS Fargate Deployment

Once the image was pushed, the ECS service automatically pulled the latest image and started a task in the cluster.

### Validation Steps

- Navigate to:

ECS → Clusters → <cluster-name> → Tasks

- Confirm that:
  - Task status is **RUNNING**
  - Desired count equals running count
  - Task logs show no startup errors

20 November 2025

The screenshot displays the AWS Management Console for the 'wordpress-ecs-cluster'. The top navigation bar shows 'Clusters > wordpress-ecs-cluster > Services'. The cluster's ARN is 'arn:aws:ecs:us-west-2:50464907:6991:cluster/wordpress-ecs-cluster'. The status is 'Active'. CloudWatch monitoring is 'Default'. There are no registered container instances. The 'Services' section shows 'Draining' at 0, 'Active' at 1, 'Pending' at 0, and 'Running' at 1. The 'Tasks' section shows 'Pending' at 0 and 'Running' at 1. The 'Services (1/1)' table lists the 'wordpress-service' with an ARN of 'arn:aws:ecs:us-v', status 'Active', and a deployment of 1/1.

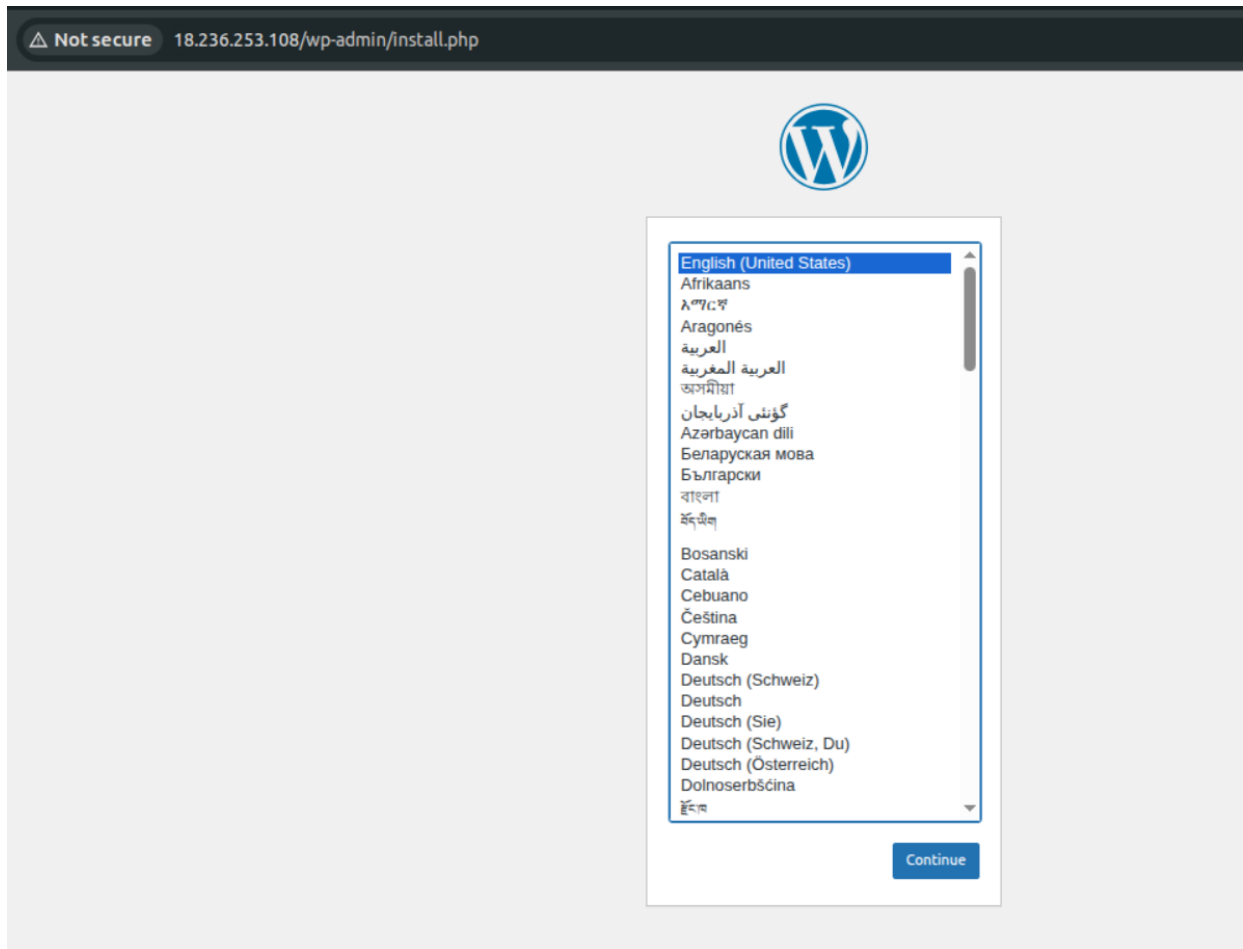
Service name	ARN	Status	Schedu...	La...	Task de...	Deployments and tasks
wordpress-service	arn:aws:ecs:us-v	Active	REPLICA	FAR...	wordpress...	1/1

## 8.4 Accessing the WordPress Frontend

With the ECS task running successfully, the next step was to access WordPress from a browser.

- Go to EC2 Console and scroll down to Network Interfaces under Network and Security.
- Locate your ENI that belong to the public security group.
- Copy its IPv4 address and access it via browser. You will see the wordpress setup page.

20 November 2025



User can proceed with:

- Site title
- Admin user credentials
- Database configuration

## 8.5 Connecting WordPress to RDS

During setup, the following values were entered:

Field	Value
Database Name	cloud
Username	wajahat
Password	(secure password used in tfvars)
Database Host	<rds-endpoint>

These values match the Terraform-provisioned RDS configuration.

If the configuration is correct:

- WordPress connects to RDS
- The installation completes successfully

20 November 2025

## 8.6 Post-Deployment Verification

Final checks included:

- Confirming tables created in RDS:

```
SHOW DATABASES;  
USE cloud;  
SHOW TABLES;
```

- Confirming WordPress is reachable after a browser refresh
- Verifying logs in CloudWatch (if configured)

## 9. Troubleshooting

This section covers common errors encountered during deployment and how to resolve them.

### Issue 1 : ECS Task Starts but WordPress Cannot Reach Database

#### Symptom:

WordPress setup page reports:

```
Error establishing a database connection
```

#### Cause:

The ECS Task Definition does not contain the required database environment variables, such as:

- WORDPRESS\_DB\_HOST
- WORDPRESS\_DB\_NAME
- WORDPRESS\_DB\_USER
- WORDPRESS\_DB\_PASSWORD

Without these, the container starts but cannot locate the RDS instance.

#### Resolution:

1. Open `modules/ecs/taskdef.tf`
2. Ensure the container definition includes:

```
environment = [  
  { name = "WORDPRESS_DB_HOST", value = module.rds.db_endpoint },  
  { name = "WORDPRESS_DB_NAME", value = var.db_name },  
  { name = "WORDPRESS_DB_USER", value = var.db_username },  
  { name = "WORDPRESS_DB_PASSWORD", value = var.db_password }  
]
```

3. Re-run:

```
terraform apply
```

4. Wait for the ECS service to redeploy the task.

20 November 2025

## Issue 2 : ECS Task Fails to Pull Image

### Symptom:

Task status in ECS → “Stopped” with reason:

```
CannotPullContainerError: pull access denied
```

### Possible Causes & Fixes:

- **Image not pushed to ECR**

- Push again using:

```
docker push <aws_account_id>.dkr.ecr.<region>.amazonaws.com/image:latest
```

- **ECS Task IAM role missing ECR permissions**

- Ensure the task execution role includes:

- AmazonECSTaskExecutionRolePolicy

## Issue 3 : RDS Setup Fails with Password Error

### Symptom:

```
InvalidParameterValue: MasterUserPassword is not valid
```

### Cause:

RDS requires passwords with:

- Minimum length: 8 characters
- Must contain allowed character types

### Resolution:

Update `db_password` in `terraform.tfvars` accordingly:

```
db_password = "StrongPass123"
```

Then re-run:

```
terraform apply
```

# 10. Cleanup

To remove all AWS resources created by Terraform:

1. From the project root:

```
terraform destroy
```

2. Confirm destruction when prompted.

3. After destruction, manually delete the remote state bucket (if using S3) to avoid accidental future usage.

20 November 2025

For security:

- Remove leftover ECR images (optional)
- Clear CloudWatch logs
- Ensure no orphan RDS snapshots remain

## Conclusion

Testing confirmed that:

- Terraform successfully provisioned AWS networking, ECS, and RDS resources
- ECS pulled the WordPress image from ECR
- The application frontend loaded correctly
- WordPress successfully connected to RDS
- Installation completed through the browser interface