

Práctica 4 - Contenedores en AWS con ECS

Asignatura: Computación en la Nube

Fecha: [Fecha de entrega]

Autor: Francisco Javier López-Dufour Morales



Índice

- Práctica 4 - Contenedores en AWS con ECS
 - Índice
 - 1. Introducción
 - 2. Objetivos
 - 3. Descripción de Actividades
 - 3.1. Creación de un contenedor Docker con una aplicación
 - 3.2. Creación de un repositorio en ECR y subida del contenedor
 - 3.3. Despliegue del contenedor usando ECS
 - 3.4. Despliegue del contenedor usando Fargate y comparación
 - 3.5. Actividad Extra 1 (Opcional): Automatización con CloudFormation
 - 3.6. Actividad Extra 2: Despliegue de un cluster con varios contenedores
 - 4. Conclusiones
 - 5. Referencias
 - 6. Anexos
 - Anexo 1: Docker Compose para WordPress y MariaDB
 - Anexo 2: Implementación de la tarea en ECS (Punto 3.1)
 - Anexo 3: Implementación de la tarea en Fargate

1. Introducción

En esta práctica exploraremos el uso de contenedores en AWS utilizando el servicio Amazon Elastic Container Service (ECS). El objetivo es familiarizarse con la creación y despliegue de contenedores Docker en AWS, gestionando repositorios en ECR y comparando diferentes métodos de despliegue, incluyendo Fargate.

2. Objetivos

- Crear un contenedor Docker con una aplicación sencilla para comprobar su funcionamiento.
- Configurar un repositorio en Amazon ECR y subir el contenedor creado.
- Desplegar el contenedor usando Amazon ECS.
- Desplegar el contenedor usando AWS Fargate y comparar la experiencia con el despliegue anterior.
- (Opcional) Automatizar el procedimiento utilizando AWS CloudFormation y scripts para minimizar el uso del cliente web de AWS.
- (Opcional) Extender el despliegue para implementar un cluster con varios contenedores distintos.

3. Descripción de Actividades

3.1. Creación de un contenedor Docker con una aplicación

Descripción: Crear un contenedor Docker que contenga una aplicación que permita comprobar su funcionamiento, como por ejemplo una página web sencilla.

Conceptos Clave:

- **ECS** (*Elastic Container Service*): Servicio de orquestación de contenedores que permite ejecutar y escalar aplicaciones en contenedores.
- **Clúster ECS**: Conjunto lógico de recursos en el que se ejecutan las tareas y servicios de ECS.
- **Definición de Tarea** (*Task Definition*): Plantilla que describe uno o más contenedores (por ejemplo, imagen de contenedor, variables de entorno, puertos, volúmenes).
- **Servicio ECS**: Gestiona la ejecución de tareas y asegura que se mantenga el número deseado de tareas en ejecución.

Pasos a seguir:

1. Instalar Docker:

- Asegurarse de tener Docker instalado en el sistema local.
- [Guía de instalación de Docker](#)

2. Crear la aplicación:

- Vamos a utilizar una aplicación existente, WordPress, para simplificar la configuración.
- Descargamos la imagen oficial de WordPress desde Docker Hub: `docker pull wordpress:latest`
- Descargar también la imagen de MariaDB para la base de datos: `docker pull mariadb:latest`

3. Escribir el Dockerfile:

- Crear un archivo `docker-compose.yml` que defina los servicios de WordPress y MariaDB.
- Ejemplo de `docker-compose.yml`: [Anexo 1](#)

4. Construir y ejecutar los contenedores:

- Ejecutar el comando para construir y ejecutar los contenedores:

```
docker-compose up
```

5. Probar el contenedor localmente:

- Acceder a `http://localhost` para comprobar que la aplicación de WordPress funciona.



Hola

¡Este es el famoso proceso de instalación de WordPress en cinco minutos! Simplemente completa la información siguiente y estarás a punto de usar la más enriquecedora y potente plataforma de publicación personal del mundo.

Información necesaria

Por favor, proporciona la siguiente información. No te preocupes, siempre podrás cambiar estos ajustes más tarde.

Título del sitio

Nombre de usuario

Los nombres de usuario pueden tener únicamente caracteres alfanuméricos, espacios, guiones bajos, guiones medios, puntos y el símbolo @.

Contraseña [Mostrar](#)

Fuerte

Importante: Necesitas esta contraseña para acceder. Por favor, guárdala en un lugar seguro.

Tu correo electrónico

Comprueba bien tu dirección de correo electrónico antes de continuar.

Visibilidad en los motores de búsqueda ☐ Pedir a los motores de búsqueda que no indexen este sitio
Depende de los motores de búsqueda atender esta petición o no.

3.2. Creación de un repositorio en ECR y subida del contenedor

Descripción: Configurar un repositorio en Amazon Elastic Container Registry (ECR) y subir la imagen del contenedor creado en el paso anterior.

Pasos:

1. Creación del repositorio en ECR:

- Accedemos a la consola de AWS ECR.
- Creamos un nuevo repositorio privado.
- Anotar el URI del repositorio: `[account].dkr.ecr.[region].amazonaws.com/my-app`.
 - En nuestro caso, el URI del repositorio quedaría: `491250998585.dkr.ecr.us-east-1.amazonaws.com/wordpress`
- Mutabilidad de la imagen: Seleccionamos "Mutable" para poder sobrescribir la imagen.
- Configuración de cifrado: Seleccionamos "KMS" y dejamos la configuración por defecto.
- Análisis de imágenes: habilitamos la opción "Análisis de imágenes" para poder escanear las imágenes en busca de vulnerabilidades automáticamente al subirlas al repositorio.

2. Configuración de las credenciales de AWS:

- Instalamos y configuramos AWS CLI en el sistema local.
- [Guía de instalación de AWS CLI](#)
- Configuramos las credenciales con `aws configure`.

3. Iniciar sesión en ECR:

- Ejecutar el comando de inicio de sesión en ECR:

```
aws ecr get-login-password --region [region] | docker login --username AWS --password-stdin [account].dkr.ecr.[region].amazonaws.com
```

En nuestro caso:

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 491250998585.dkr.ecr.us-east-1.amazonaws.com
Login Succeeded
```

4. Etiquetar la imagen para ECR:

- Etiquetar la imagen local para que coincida con el repositorio de ECR:

```
docker tag my-app:latest [account].dkr.ecr.[region].amazonaws.com/my-app:latest
```

En nuestro caso:

```
docker tag wordpress:latest 491250998585.dkr.ecr.us-east-1.amazonaws.com/wordpress:latest
docker tag mariadb:11.0.2 491250998585.dkr.ecr.us-east-1.amazonaws.com/mariadb:latest
```

5. Subir la imagen a ECR:

- Ejecutar el comando para subir la imagen:

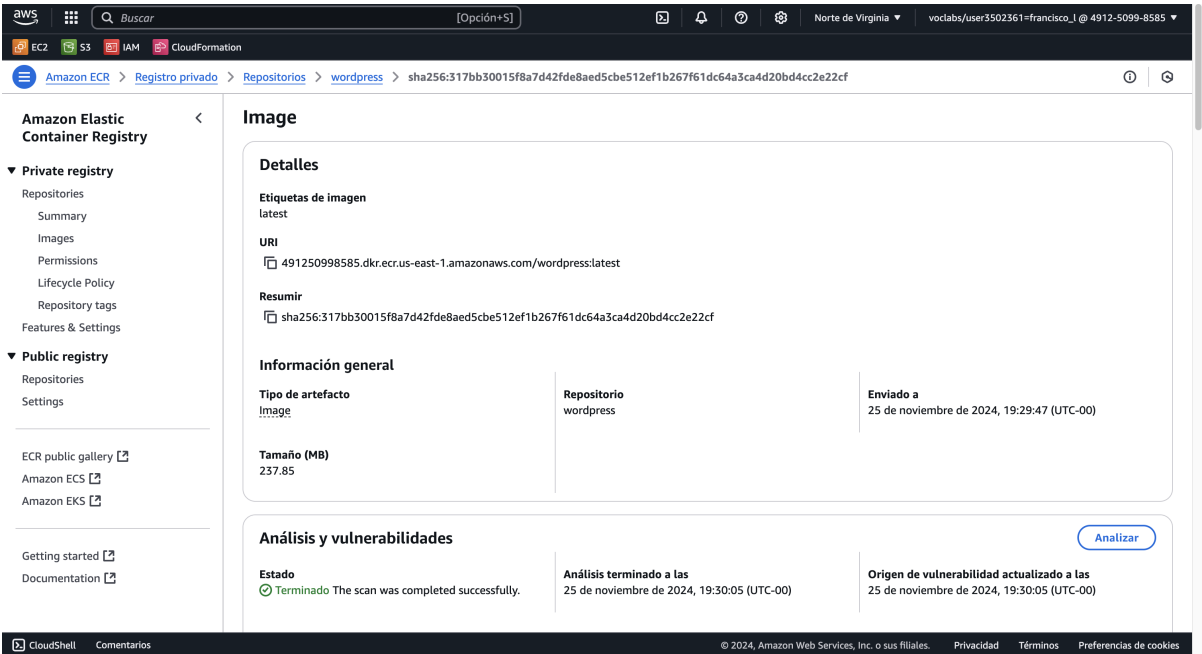
```
docker push [account].dkr.ecr.[region].amazonaws.com/my-app:latest
```

En nuestro caso:

```
docker push 491250998585.dkr.ecr.us-east-1.amazonaws.com/wordpress:latest
docker push 491250998585.dkr.ecr.us-east-1.amazonaws.com/mariadb:latest
```

6. Verificar la imagen en ECR:

- Acceder a la consola de ECR y verificar que la imagen se ha subido correctamente.



3.3. Despliegue del contenedor usando ECS

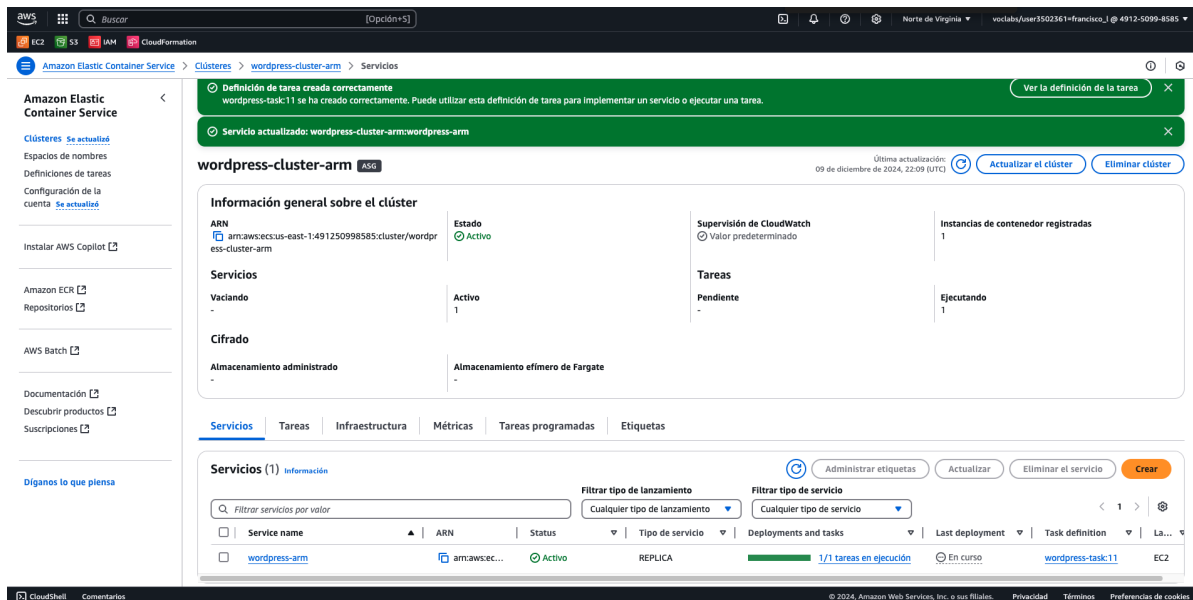
Descripción: Ahora crearemos un clúster en ECS usando instancias EC2, definiendo la tarea y el servicio. Esto proporciona más control sobre la infraestructura subyacente, permitiendo por ejemplo el uso de tipos de instancia específicos y estrategias de escalado personalizadas.

Pasos para crear un Clúster de ECS con Instancias EC2:

1. Crear un clúster de ECS:

- Acceder a la consola de Amazon ECS.
- Crear un nuevo clúster (por ejemplo, con EC2 Linux + Networking).
 - Nombre: **wordpress-cluster**
- Configuración de la Infraestructura:
 - Instancias EC2
 - Grupo de Auto Scaling: Crear un nuevo grupo de Auto Scaling.
 - Modelo de aprovisionamiento: **On-Demand**.
 - Imagen de la máquina EC2: **Amazon Linux 2 (kernel 5.10/arm64)**.
 - Es importante seleccionar una imagen compatible con ARM si vamos a utilizar instancias ARM.
 - Tipo de instancias EC2: **t2.medium** para soportar la carga de WordPress y la base de datos. Ajustar según las necesidades de la aplicación.
 - **Nota:** Ajustar el tipo de instancia según los requisitos de la aplicación.
 - Rol de instancia: Crear un nuevo rol de IAM o seleccionar uno existente. (**LabInstanceProfile** en nuestro caso).
 - Capacidad deseada:
 - Número de instancias: **1**.
 - Capacidad máxima: **2**.
 - Par de claves: Seleccionar un par de claves existente o crear uno nuevo. (**vockey** en nuestro caso).
 - Tamaño del volumen raíz: **30 GiB**.
- Configuración de la red:
 - VPC: Seleccionamos la VPC por defecto.
 - Subredes: Seleccionamos todas las subredes.
 - Grupo de seguridad: Crear un nuevo grupo de seguridad o seleccionar uno existente.
 - Debemos asegurarnos de que permite tráfico en los puertos necesarios (80 para HTTP y 3306 para MariaDB).
 - Balanceador de carga: No seleccionar balanceador de carga.
 - Asignar automáticamente direcciones IP públicas: **Activado**.
- Monitoreo, Cifrado y Etiquetas: Mantenemos la configuración por defecto.

Resumen de la configuración:



Nota: Debemos asegurarnos de que las imágenes de los contenedores soporten la arquitectura elegida.

2. Definir una tarea (*Task Definition*):

Como estamos ejecutando dos contenedores (WordPress y MariaDB) que necesitan comunicarse entre sí, usaremos una definición de tarea que incluye ambos contenedores.

- Accedemos a **Definiciones de Tareas** en la consola de ECS.
 - Familia de la tarea: **wordpress-task**.
 - Requisitos de infraestructura:
 - Tipo de Lanzamiento: **EC2**.
 - Sistema operativo: **Linux/ARM64**.
 - **Nota:** Si estamos utilizando instancias ARM, debemos asegurarnos de que las imágenes de los contenedores son compatibles con ARM.
 - Modo de red: **awsvpc**.
 - CPU: **1 vCPU**.
 - Memoria: **1 GB**.
 - **Nota:** Ajustar la memoria y CPU según los requisitos de la aplicación.
 - Rol de tarea: Crear un nuevo rol de IAM o seleccionar uno existente. (**LabRole** en nuestro caso).
 - Rol de ejecución de tareas: Crear un nuevo rol de IAM o seleccionar uno existente. (**LabRole** en nuestro caso).
 - Contenedor 1: **mariadb**
 - Nombre: **mariadb**.
 - Contenedor esencial: **Si**.
 - URI de la imagen: **491250998585.dkr.ecr.us-east-1.amazonaws.com/mariadb:latest**.
 - Mapeo de puertos:
 - Puerto de anfitrión: **0**. (mapeo dinámico al host).
 - Puerto de contenedor: **3306**.
 - Protocolo: **tcp**.
 - Nombre: **mariadb-3306-tcp**.
 - Variables de entorno:

- **MYSQL_ROOT_PASSWORD:** root
- **MYSQL_DATABASE:** wordpress
- **MYSQL_USER:** wordpress
- **MYSQL_PASSWORD:** wordpress
- Contenedor 2: **wordpress**
 - Nombre: **wordpress**.
 - URI de la imagen: **491250998585.dkr.ecr.us-east-1.amazonaws.com/wordpress:latest**.
 - Contenedor esencial: **Si**.
 - Mapeo de puertos:
 - Puerto de anfitrión: **80**.
 - Puerto de contenedor: **80**.
 - Protocolo: **tcp**.
 - Nombre: **wordpress-80-tcp**.
 - Protocolo de la aplicación: **http**.
 - Ordenación de la dependencia de inicio:
 - Nombre del contenedor: **mariadb**. Condiciones: **Start**. (Para asegurar que MariaDB se inicia antes de WordPress).
 - Variables de entorno:
 - **WORDPRESS_DB_HOST:** mariadb:3306
 - **WORDPRESS_DB_USER:** wordpress
 - **WORDPRESS_DB_PASSWORD:** wordpress
 - **WORDPRESS_DB_NAME:** wordpress
 - Configuración de red de contenedores:
 - Enlaces: **mariadb**. Alias: **mariadb**. (Como estamos utilizando el modo bridge, necesitamos enlazar los contenedores para que puedan comunicarse).

Resumen de la configuración:

The screenshot displays the AWS Management Console for Amazon Elastic Container Service (ECS). The main view shows the configuration for a task definition named 'wordpress-task:10'. The interface includes a sidebar with navigation links for Clusters, Namespaces, Task Definitions, and Account Configuration. The main content area is divided into several sections:

- Definition Created Successfully:** A green banner at the top indicates that the task definition 'wordpress-task:10' has been created successfully.
- Task Definition Summary:** A section titled 'Información general' provides key details:
 - ARN:** am:aws:ecs:us-east-1:491250998585:task-definition:wordpress-task:10
 - Estado:** ACTIVE
 - Horas de creación:** 09 de diciembre de 2024, 21:55 (UTC)
 - Entorno de la aplicación:** EC2
 - Rol de tarea:** LabRole
 - Rol de ejecución de tareas:** LabRole
 - Sistema operativo/arquitectura:** Linux/ARM64
 - Modo de red:** bridge
- Task Size (Tamaño de la tarea):** A section showing resource allocation:
 - CPU de tareas:** 1024 unidades (1 vCPU)
 - Memoria de tareas:** 1024 MiB (1 GB)
 - Asignación máxima de la CPU de la tarea para los contenedores:** A bar chart showing the maximum CPU allocation for the task.
 - Asignación máxima de memoria de tarea para la reserva de memoria del contenedor:** A bar chart showing the maximum memory allocation for the task.
- Containers (Contenedores):** A table at the bottom lists the containers defined in the task definition:

Nombre del contenedor	Imagen	Registro privado	Esencial	CPU	Límite invariable/flexibil...	GPU
mariadb						
wordpress						

Esta configuración centralizada en la **Task Definition** proporciona una plantilla reutilizable, facilitando futuras actualizaciones y despliegues.

3. Crear un servicio de ECS:

- Acceder a Servicios:
 - En la consola de ECS, seleccionamos **Clusters**, luego seleccionamos nuestro clúster **wordpress-cluster** y procedemos a crear un servicio.
- Configuración el Servicio:
 - Entorno:
 - Tipo de lanzamiento: **EC2**.
 - Configuración de la implementación:
 - Tipo de implementación: **Servicio**.
 - Familia de la tarea: **wordpress-task**. (la definición de tarea que creamos anteriormente).
 - Nombre del servicio: **wordpress-service**.
 - Tipo de servicio: **REPLICA**. (para mantener un número fijo de tareas en ejecución).
 - Tareas deseadas: **1**.
- Mantenemos las opciones por defecto en el resto de las secciones.

Resumen de la configuración:

The screenshot shows the Amazon ECS console for the 'wordpress-arm' service. The service is active and has 1 desired task. The task definition is 'wordpress-task:10'. The task is running on an EC2 instance with 1 vCPU and 1 GB of memory. The task is named '78271f...' and is in the 'Ejecutando' state. The task is running on a container named 'wordpress' with the image '81462cf4069c75d...' and is in the 'Running' state. The task is running on a container named 'mariadb' with the image 'da77797735273...' and is in the 'Running' state.

Tarea	Último est...	Estado de...	Definició...	Estado	Iniciado a ...	Instancias de co...	Tipo de lanz...	Versión de L...	CPU	Memoria
78271f...	Ejecutando	Ejecutando	wordpress-...	Desconocid	hace 1 minuto	0241a5f88dc04528...	EC2	-	1 vCPU	1 GB

Nombre del contenedor	ID de tiempo de eje...	URI de imag...	Resumen de...	Estado	Estado	CPU	Límite invariable/flexible de mem...
wordpress	81462cf4069c75d...	49125099...	sha256:31...	Running	Desconocido	0	- / -
mariadb	da77797735273...	49125099...	sha256:35...	Running	Desconocido	0	- / -

El servicio mantiene el número de tareas especificado en ejecución, lo que aporta resiliencia ante fallos. La creación del servicio permite escalar el número de contenedores de forma dinámica y asegura alta disponibilidad (si se configura con múltiples tareas y balanceador de carga).

4. Probar el despliegue:

- Comprobar que las instancias de tarea están en ejecución.

Amazon Elastic Container Service

Clústeres [Se actualizó](#)

Espacios de nombres

Definiciones de tareas

Configuración de la cuenta [Se actualizó](#)

Instalar AWS Copilot [🔗](#)

Amazon ECR [🔗](#)

Repositorios [🔗](#)

AWS Batch [🔗](#)

Documentación [🔗](#)

Descubrir productos [🔗](#)

Suscripciones [🔗](#)

[Díganos lo que piensa](#)

Amazon Elastic Container Service

Clústeres > wordpress-cluste... > Infraestructura > Instancias de conte... > 0241a5f88dc0452880e9b12c... > Tareas > 35e8f4ee6bb24618899ccab... > Configuración

CPU | Memoria
1 vCPU | 1 GB

Versión de la plataforma
-

Tipo de lanzamiento
EC2 | [i-0f66bb523f9ec0eba](#)

ID de instancias de contenedor
[0241a5f88dc0452880e9b12c8a83e2b81](#)

Definición de tareas: revisión
wordpress-task:11

Grupo de tareas
wordpress

Modo de red
bridge

ID de subred
[subnet-05c938cea73e9f1e4](#)

IP privada
[172.31.28.177](#)

Dirección MAC
[0a:ff:fa:cd:2:c5](#)

Contenedores (2)

Filtrar contenedores

Nombre del conten...	ID de tiempo de eje...	URI de imag...	Resumen de...	Estado	Estado	CPU	Límite invariable/flexible de ...
wordpress	d2262039862a14...	49125099...	sha256:31...	Running	Desconocido	0	- / -
mysql	5441061c64c25b...	49125099...	sha256:35...	Running	Desconocido	0	- / -

Detalles del contenedor para wordpress

Detalles

Configuración de registros

Política de reinicio

Enlaces de red

Etiquetas y hosts de Docker

Archivos y variables de entorno

Configuración de volumen

URI de imagen
[491250998585.dkr.ecr.us-east-1.amazonaws.com/wordpresslatest](#)

Esencial
Sí

Comando
-

- Obtener la dirección IP o el balanceador de carga para acceder a la aplicación.
 - En nuestro caso, la dirección IP pública de la instancia EC2 es **ec2-54-144-88-19.compute-1.amazonaws.com**

← → ↻ No es seguro ec2-54-144-88-19.compute-1.amazonaws.com/wp-admin/install.php?step=1

Hola

¡Este es el famoso proceso de instalación de WordPress en cinco minutos! Simplemente completa la información siguiente y estarás a punto de usar la más enriquecedora y potente plataforma de publicación personal del mundo.

Información necesaria

Por favor, proporciona la siguiente información. No te preocupes, siempre podrás cambiar estos ajustes más tarde.

Título del sitio

Nombre de usuario

Los nombres de usuario pueden tener únicamente caracteres alfanuméricos, espacios, guiones bajos, guiones medios, puntos y el símbolo @.

Contraseña

Fuerte

Importante: Necesitas esta contraseña para acceder. Por favor, guárdala en un lugar seguro.

Tu correo electrónico

Comprueba bien tu dirección de correo electrónico antes de continuar.

Visibilidad en los motores de búsqueda ☐ Pedir a los motores de búsqueda que no indexen este sitio

Depende de los motores de búsqueda atender esta petición o no.

← → ↻ No es seguro ec2-54-144-88-19.compute-1.amazonaws.com/wp-admin/

Mi página personal 1 0 + Añadir


Escritorio

Hola, fran

Opciones de pantalla Ayuda


¡Te damos la bienvenida a WordPress!

[Aprende más sobre la versión 6.7.1.](#)

 **Crea contenido rico con bloques y patrones**


Los patrones de bloques son diseños de bloques preconfigurados. Úsalos para inspirarte o crear nuevas páginas en un instante.

[Añadir una nueva página](#)

 **Personaliza todo tu sitio con temas de bloques**

Diseña todo en tu sitio — Desde la cabecera hasta el pie de página. Todo usando bloques y patrones.

[Abrir el editor del sitio](#)

 **Cambia la apariencia de tu sitio con los estilos**

¡Retoca tu sitio o dale un aspecto completamente nuevo! Sé creativo — ¿Qué tal una nueva paleta de color o una nueva fuente?

[Editar estilos](#)

Estado de salud del sitio

Las pruebas de salud del sitio se ejecutarán automáticamente de forma periódica para obtener información sobre tu sitio. También puedes [visitar ahora la pantalla de salud del sitio](#) para obtener información sobre tu sitio.

Aún no hay información...

Borrador rápido

Título

Contenido

¿En qué estás pensando?

De un vistazo

1 entrada 1 página

1 comentario

WordPress 6.7.1 está funcionando con el tema [Twenty Twenty-Eight](#).

Eventos y noticias de WordPress

Asiste a un próximo evento cerca de ti.

[Seleccionar la ubicación](#)

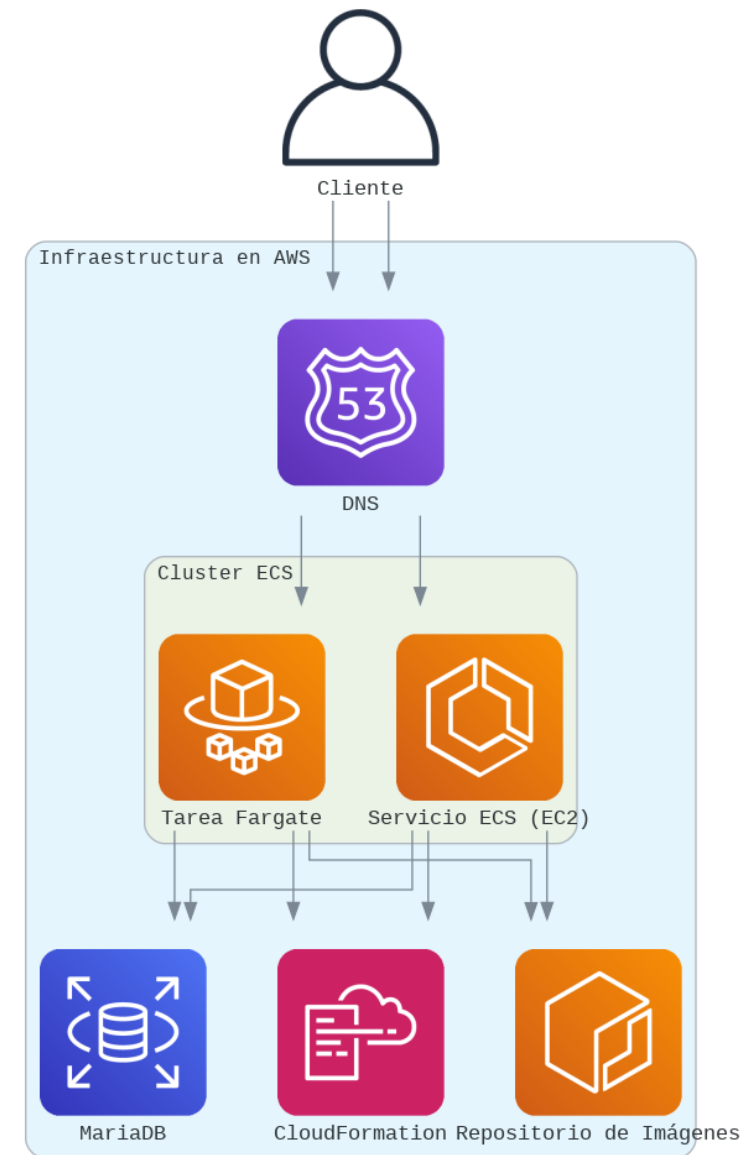
Arrastra aquí las cajas

Arrastra aquí las cajas

Notas:

- Debemos asegurarnos de que el rol de ejecución de tareas de ECS (**ECS Task Execution Role**) tenga permisos suficientes para acceder a ECR.
- Verificar que el grupo de seguridad permita tráfico en el puerto **80**.
- Ajustar recursos según las necesidades reales, evitando sobrecostos o subdimensionamiento.

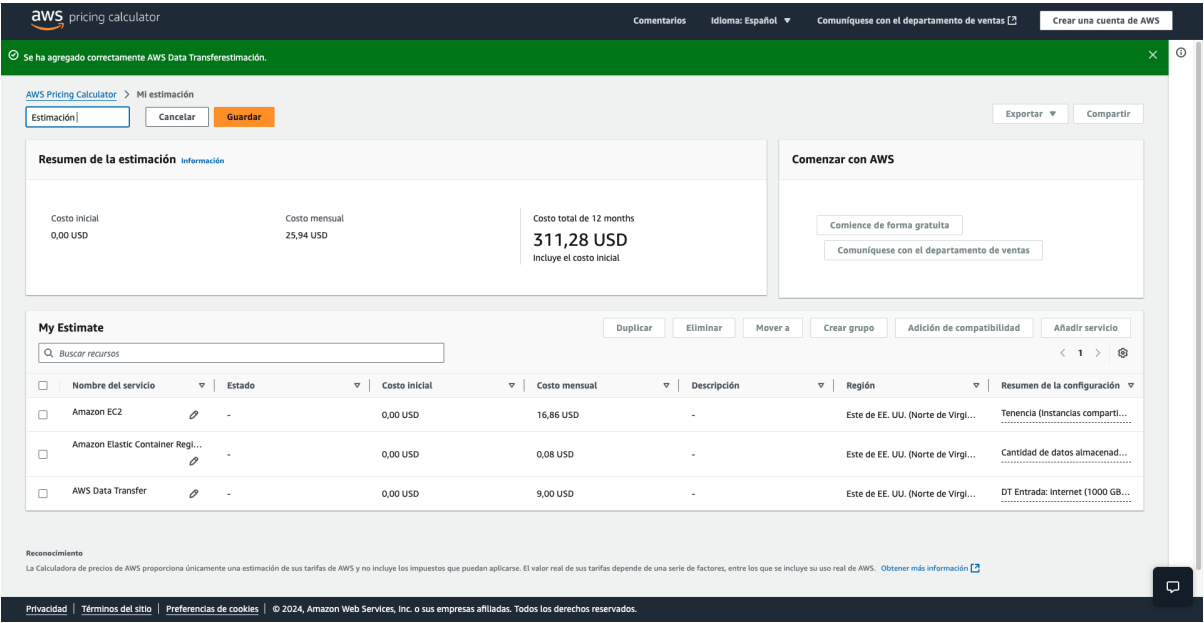
5. Arquitectura Final:



Arquitectura Técnica General - Práctica AWS ECS

La arquitectura final incluye un clúster de ECS con instancias EC2, una definición de tarea con dos contenedores (WordPress y MariaDB) y un servicio que mantiene una tarea en ejecución. La comunicación entre los contenedores se realiza a través de la red interna de ECS.

6. Análisis de Costos:



Los costos principales se dividen entre instancias EC2 (\$16.86 USD/mes), almacenamiento de imágenes en ECR (\$0.08 USD/mes), y transferencia de datos (\$9.00 USD/mes). Los costos adicionales opcionales incluyen balanceadores de carga (ELB), almacenamiento adicional (EBS) y monitoreo (CloudWatch).

Servicio	Costo Mensual (USD)	Costo Anual (USD)
Amazon EC2	\$16.86	\$202.32
Amazon Elastic Container Registry (ECR)	\$0.08	\$0.96
AWS Data Transfer	\$9.00	\$108.00
Elastic Load Balancer (ELB) (opcional)	~\$16.20	~\$194.40
Almacenamiento EBS (opcional)	\$1.00	\$12.00
AWS CloudWatch (opcional)	~\$1.10	~\$13.20
Total Estimado	\$27.94 - \$44.24	\$335.28 - \$530.88

3.4. Despliegue del contenedor usando Fargate y comparación

Descripción: Desplegar la misma aplicación (WordPress y MariaDB) en AWS ECS, pero empleando el tipo de lanzamiento Fargate en lugar de EC2. Esto facilitará la comparación entre ambas opciones, resaltando las diferencias en la gestión de infraestructura, costos, escalabilidad y simplicidad de la operación.

Consideraciones Previas:

- Arquitectura sin servidores (Serverless): Fargate es un servicio serverless que permite ejecutar contenedores sin necesidad de gestionar servidores, clústeres ni instancias EC2 subyacentes. Esto simplifica el mantenimiento pero puede tener un impacto en el costo.

Compatibilidad y Requerimientos:

- Asegurar que las imágenes y configuraciones empleadas anteriormente (WordPress, MariaDB) sean compatibles con Fargate. Normalmente lo son, pero es importante recordar:
 - Fargate no soporta el modo de red **bridge**, así que en la **Task Definition** hay que emplear **awsvpc**.
 - Cada contenedor debe contar con puertos definidos en el modo awsvpc.

Costos y Dimensionamiento:

- Con Fargate se paga por tiempo de cómputo y recursos asignados (vCPU y RAM) a las tareas, no por la infraestructura persistente como en EC2. Ajustar adecuadamente la CPU y la memoria asignadas a la **Task Definition** evitará sobre costos.

Pasos para el Despliegue:

1. Crear un nuevo Clúster de ECS:

- Acceder a la consola de ECS.
- Configuración del Cluster
 - Nombre: **wordpress-fargate-cluster**.
- Infraestructura
 - Tipo de lanzamiento: **FARGATE**.
- Monitoreo, Cifrado y Etiquetas: Mantenemos la configuración por defecto.

2. Configurar la Task Definition para Fargate:

Creamos una nueva definición de tarea para Fargate, basada en la definición anterior pero con algunas modificaciones:

- Familia de la tarea: **wordpress-fargate-task**.
- Requisitos de infraestructura:
 - Tipo de Lanzamiento: **FARGATE**.
 - Sistema operativo/arquitectura: **Linux/ARM64**.
 - **Nota:** Asegurarse de que las imágenes de los contenedores sean compatibles con ARM.
 - CPU: **0.5 vCPU**.
 - Memoria: **1 GB**.

- Rol de tarea: **LabRole**. (mismo rol que en el despliegue anterior).
- Rol de ejecución de tareas: **LabRole**.
- Red: **awsvpc**.
- Contenedor 1: **mariadb**
 - URI de la imagen: **491250998585.dkr.ecr.us-east-1.amazonaws.com/mariadb:latest**.
 - Mapeo de puertos:
 - Puerto de contenedor: **3306**.
 - Protocolo: **tcp**.
 - Variables de entorno:
 - **MYSQL_ROOT_PASSWORD**: root
 - **MYSQL_DATABASE**: wordpress
 - **MYSQL_USER**: wordpress
 - **MYSQL_PASSWORD**: wordpress
- Contenedor 2: **wordpress**
 - URI de la imagen: **491250998585.dkr.ecr.us-east-1.amazonaws.com/wordpress:latest**.
 - Mapeo de puertos:
 - Puerto de contenedor: **80**.
 - Protocolo: **tcp**.
 - Variables de entorno:
 - **WORDPRESS_DB_HOST**: mariadb:3306
 - **WORDPRESS_DB_USER**: wordpress
 - **WORDPRESS_DB_PASSWORD**: wordpress
 - **WORDPRESS_DB_NAME**: wordpress
 - Ordenación de la dependencia de inicio:
 - Nombre del contenedor: **mariadb**. Condiciones: **Start**.

Cambios en la configuración de la **Task Definition**:

- No es necesario mapear a un puerto de host (en Fargate el puerto del host se asigna automáticamente).
- Retirar dependencias de tipo links ya que con awsvpc cada contenedor obtiene su propia interfaz de red y se comunican a través de la dirección interna asignada por la VPC (usando el nombre del contenedor como referencia en las variables de entorno).

3. Crear el Servicio en Modo Fargate:

En la consola de ECS, ir a la sección "Servicios" dentro del clúster existente o crear uno nuevo si se desea separar ambientes.

- Entorno:
 - Clúster: **wordpress-fargate-cluster**.
 - Estrategia de proveedor de capacidad: **FARGATE**.
- Configuración de la implementación:
 - Tipo de aplicación: **Servicio**.
 - Nombre del servicio: **wordpress-fargate-service**.
 - Tipo de servicio: **REPLICA**.

- Tareas deseadas: 1.
- Redes:
 - VPC: Seleccionar la VPC y subredes configuradas para Fargate.
 - Asignar automáticamente direcciones IP públicas: **Activado**.
 - Grupo de seguridad: Seleccionar un grupo de seguridad que permita tráfico en los puertos necesarios (80 y 3306).

The screenshot shows the AWS Elastic Container Service console. The breadcrumb trail is: **Amazon Elastic Container Service** > **Clústeres** > **wordpress-fargate-cluster** > **Servicios** > **wordpress-fargate-service** > **Estado**.

Descripción general del servicio

- Estado:** Activo
- Tareas (1 deseadas):** 0 pendiente/s | 1 en ejecución
- Definición de tarea:** [revisión wordpress-fargate-task:16](#)
- Estado de despliegue:** Realizado correctamente

Estado y métricas

- Estado:** Información
- Nombre del servicio:** [wordpress-fargate-service](#)
- ARN del servicio:** [arn:aws:ecs:us-east-1:4912509:98585:service/wordpress-fargate-cluster/wordpress-fargate-service](#)
- Estado actual de las implementaciones:** 1 tarea completada
- Creado a las:** 11 de diciembre de 2024, 19:29 (UTC)

Estado

- Utilización de CPU:** 31.7%
- Utilización de memoria:** 10.2%

4. Desplegar y Probar:

- Crear el servicio y esperar a que las tareas se inicien.

The screenshot shows the AWS Elastic Container Service console. The breadcrumb trail is: **Amazon Elastic Container Service** > **Clústeres** > **wordpress-fargate-cluster** > **Servicios** > **wordpress-fargate-service** > **Tareas** > **b24ec48240ca40308aab4...** > **Configuración**.

Configuración

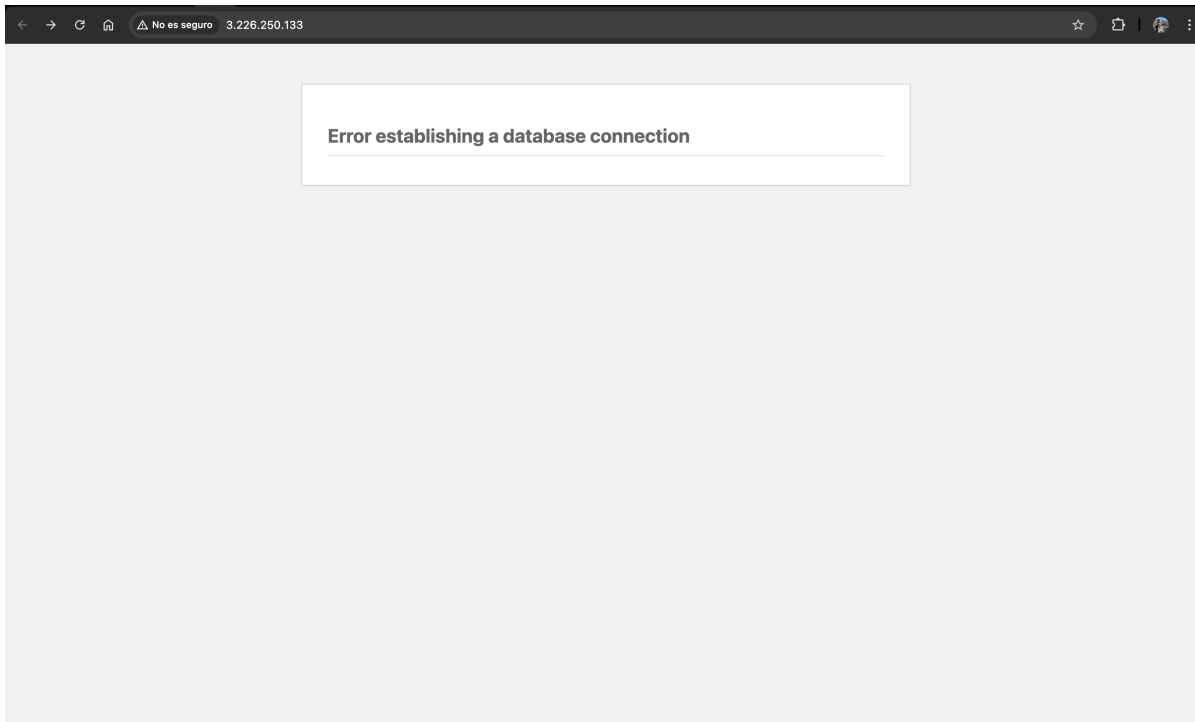
- Sistema operativo/arquitectura:** Linux/ARM64
- CPU | Memoria:** 1 vCPU | 2 GB
- Versión de la plataforma:** 1.4.0
- Proveedor de capacidad:** FARGATE
- Tipo de lanzamiento:** FARGATE
- ID de ENI:** [eni-0c44977a9c3da9f4a](#)
- Modo de red:** awsvpc
- ID de subred:** [subnet-01bc191d5db1cf47a](#)
- IP pública:** [54.81.71.39](#) | [dirección abierta](#)
- IP privada:** [172.31.32.59](#)
- Dirección MAC:** [0e:64:bd:3d:25:55](#)

Contenedores (3)

Nombre del contenedor	ID de tiempo de ejecución	URI de imagen	Resumen de...	Estado	Estado
wordpress	b24ec48240ca40...	49125099...	sha256:31...	Running	Desconocido
mariadb	b24ec48240ca40...	49125099...	sha256:35...	Running	Desconocido
ecs-service-connect-y...	-	-	-	Running	Estado correcto

- Una vez en estado **RUNNING**, obtener la dirección IP pública asignada a la tarea.
- Acceder desde el navegador a la URL y verificar que WordPress funcione correctamente.

Resultado del Despliegue



Tras intentar desplegar los contenedores (WordPress y MariaDB) en ECS utilizando Fargate, encontramos un problema: **los contenedores dentro de la misma tarea no lograron establecer comunicación entre sí.**

Análisis del Problema

Según la documentación oficial de AWS ECS:

"Containers that belong to the same task can also communicate over the localhost interface."
(Fuente: [AWS Docs](#))

En teoría, al usar el modo de red **awsvpc**, los contenedores de una misma tarea pueden comunicarse utilizando el nombre del contenedor como DNS. Sin embargo, en la práctica, el contenedor de WordPress no pudo resolver correctamente el nombre del contenedor de MariaDB (**mariadb**) ni conectarse al puerto **3306**.

Posibles Causas

- Problemas con la configuración de DNS en Fargate para la resolución de nombres entre contenedores.
- La dependencia **dependsOn** no garantiza que MariaDB esté listo antes de que WordPress intente conectarse.
- Limitaciones intrínsecas del diseño de Fargate para este caso específico de comunicación directa entre contenedores.

Propuestas de Solución

1. Dividir los contenedores en dos servicios ECS separados:

- Crear un servicio para MariaDB y otro para WordPress, conectándolos mediante:

- Un endpoint público para MariaDB (con acceso restringido a la IP del servicio de WordPress).
- Un balanceador de carga interno (opcional).
- Ventaja: Esto desacopla los contenedores, lo que permite una configuración más robusta y escalable.
- Desventaja: Aumenta la complejidad de la configuración.

2. Usar un motor de bases de datos gestionado (RDS):

- Reemplazar MariaDB en contenedor con un servicio RDS de Amazon (MariaDB o MySQL).
- Ventaja: Simplifica la configuración, mejora la disponibilidad y permite un escalado más eficiente.
- Desventaja: Puede ser más costoso en escenarios pequeños.

3.5. Actividad Extra 1 (Opcional): Automatización con CloudFormation

3.6. Actividad Extra 2: Despliegue de un cluster con varios contenedores

En esta actividad se realizó el despliegue de un clúster con múltiples contenedores utilizando AWS ECS en dos configuraciones principales:

1. ECS con instancias EC2:

- Se desplegó un clúster ECS utilizando instancias EC2 como capacidad subyacente.
- Los contenedores de WordPress y MariaDB se ejecutaron en una sola tarea, aprovechando el modo de red **bridge** para facilitar la comunicación entre ellos.
- Este despliegue permitió una mayor personalización en las configuraciones de red y almacenamiento, pero requirió la gestión manual de la infraestructura subyacente.

2. ECS con Fargate:

- Se intentó realizar el despliegue de los mismos contenedores (WordPress y MariaDB) en un clúster ECS utilizando Fargate, eliminando la necesidad de gestionar servidores.
- Aunque Fargate ofrece una solución más simplificada y serverless, encontramos problemas con la conectividad interna entre los contenedores en la misma tarea.
- Según la documentación oficial, los contenedores deberían comunicarse mediante el localhost o utilizando el nombre del contenedor como DNS. Sin embargo, en la práctica, WordPress no logró conectarse a MariaDB debido a posibles limitaciones o errores en la configuración del modo de red **awsvpc**.

4. Conclusiones

En esta práctica se ha demostrado el proceso completo de empaquetar una aplicación en contenedores utilizando Docker, alojar las imágenes resultantes en el registro privado de Amazon ECR y desplegarlas en AWS ECS. La experiencia ha permitido comparar dos modos de despliegue distintos: el uso de instancias EC2 para el clúster y la opción serverless que ofrece Fargate.

Como resultado, se han obtenido las siguientes conclusiones básicas:

- **Flujo de trabajo consolidado:** La práctica brinda una visión global del ciclo de vida de los contenedores en la nube, desde la construcción local de la imagen Docker hasta su almacenamiento en ECR y posterior despliegue en ECS.
- **Facilidad con ECS y ECR:** AWS ECS simplifica el proceso de orquestación y escalado de contenedores al permitir definir tareas, servicios y clústeres de forma centralizada. ECR, por su parte, ofrece un repositorio privado y seguro para las imágenes.

Comparación entre EC2 y Fargate:

- **EC2:** Requiere gestionar la infraestructura subyacente (instancias, escalado, mantenimiento), pero ofrece mayor flexibilidad y control sobre los recursos.
- **Fargate:** Facilita el despliegue sin gestionar servidores, permitiendo centrarse exclusivamente en la aplicación. Sin embargo, puede presentar dificultades en la comunicación entre contenedores dentro de la misma tarea y, en ocasiones, resultar más costoso dependiendo de la carga de trabajo.

5. Referencias

- [Amazon ECS Developer Guide](#)
- [Amazon ECR User Guide](#)
- [AWS Fargate Documentation](#)
- [Docker Documentation](#)
- [AWS CLI Command Reference](#)

6. Anexos

Anexo 1: Docker Compose para WordPress y MariaDB

```
version: '3.8'

services:
  db:
    image: mariadb:11.0.2
    restart: always
    ports:
      - '3306:3306'
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
  wordpress:
    image: wordpress:latest
    restart: always
    ports:
      - '80:80'
    depends_on:
      - db
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress
      WORDPRESS_CONFIG_EXTRA: |
        /** disable wp core auto update */
        define('WP_AUTO_UPDATE_CORE', false);
```

Anexo 2: Implementación de la tarea en ECS (Punto 3.1)

```
{
  "family": "wordpress-task",
  "containerDefinitions": [
    {
      "name": "mariadb",
      "image": "491250998585.dkr.ecr.us-east-1.amazonaws.com/mariadb:latest",
      "cpu": 0,
      "portMappings": [
        {
          "name": "mariadb-3306-tcp",
          "containerPort": 3306,
          "hostPort": 0,
          "protocol": "tcp"
```

```

    }
  ],
  "essential": true,
  "environment": [
    {
      "name": "MYSQL_DATABASE",
      "value": "wordpress"
    },
    {
      "name": "MYSQL_PASSWORD",
      "value": "wordpress"
    },
    {
      "name": "MYSQL_ROOT_PASSWORD",
      "value": "root"
    },
    {
      "name": "MYSQL_USER",
      "value": "wordpress"
    }
  ],
  "environmentFiles": [],
  "mountPoints": [],
  "volumesFrom": [],
  "ulimits": [],
  "systemControls": []
},
{
  "name": "wordpress",
  "image": "491250998585.dkr.ecr.us-east-1.amazonaws.com/wordpress:latest",
  "cpu": 0,
  "links": ["mariadb:mariadb"],
  "portMappings": [
    {
      "name": "wordpress-80-tcp",
      "containerPort": 80,
      "hostPort": 80,
      "protocol": "tcp",
      "appProtocol": "http"
    }
  ],
  "essential": true,
  "environment": [
    {
      "name": "WORDPRESS_DB_USER",
      "value": "wordpress"
    },
    {
      "name": "WORDPRESS_DB_HOST",
      "value": "mariadb:3306"
    },
    {
      "name": "WORDPRESS_DB_PASSWORD",

```

```

        "value": "wordpress"
    },
    {
        "name": "WORDPRESS_DB_NAME",
        "value": "wordpress"
    }
],
"environmentFiles": [],
"mountPoints": [],
"volumesFrom": [],
"dependsOn": [
    {
        "containerName": "mariadb",
        "condition": "START"
    }
],
"systemControls": []
}
],
"taskRoleArn": "arn:aws:iam::491250998585:role/LabRole",
"executionRoleArn": "arn:aws:iam::491250998585:role/LabRole",
"networkMode": "bridge",
"volumes": [],
"placementConstraints": [],
"requiresCompatibilities": ["EC2"],
"cpu": "1024",
"memory": "1024",
"runtimePlatform": {
    "cpuArchitecture": "ARM64",
    "operatingSystemFamily": "LINUX"
}
}

```

Anexo 3: Implementación de la tarea en Fargate

```

{
    "family": "wordpress-fargate-task",
    "containerDefinitions": [
        {
            "name": "mariadb",
            "image": "491250998585.dkr.ecr.us-east-1.amazonaws.com/mariadb:latest",
            "cpu": 0,
            "portMappings": [
                {
                    "name": "mariadb-3306-tcp",
                    "containerPort": 3306,
                    "hostPort": 3306,
                    "protocol": "tcp"
                }
            ],

```

```
"essential": false,
"environment": [
  {
    "name": "MYSQL_DATABASE",
    "value": "wordpress"
  },
  {
    "name": "MYSQL_PASSWORD",
    "value": "wordpress"
  },
  {
    "name": "MYSQL_ROOT_PASSWORD",
    "value": "root"
  },
  {
    "name": "MYSQL_USER",
    "value": "wordpress"
  }
],
"mountPoints": [],
"volumesFrom": [],
"systemControls": []
},
{
  "name": "wordpress",
  "image": "491250998585.dkr.ecr.us-east-1.amazonaws.com/wordpress:latest",
  "cpu": 0,
  "portMappings": [
    {
      "name": "wordpress-80-tcp",
      "containerPort": 80,
      "hostPort": 80,
      "protocol": "tcp",
      "appProtocol": "http"
    }
  ],
  "essential": true,
  "environment": [
    {
      "name": "WORDPRESS_DB_USER",
      "value": "wordpress"
    },
    {
      "name": "WORDPRESS_DB_HOST",
      "value": "localhost:3306"
    },
    {
      "name": "WORDPRESS_DB_PASSWORD",
      "value": "wordpress"
    },
    {
      "name": "WORDPRESS_DB_NAME",
      "value": "wordpress"
    }
  ]
}
```

```
    }
  ],
  "mountPoints": [],
  "volumesFrom": [],
  "dependsOn": [
    {
      "containerName": "mariadb",
      "condition": "START"
    }
  ],
  "systemControls": []
}
],
"taskRoleArn": "arn:aws:iam::491250998585:role/LabRole",
"executionRoleArn": "arn:aws:iam::491250998585:role/LabRole",
"networkMode": "awsvpc",
"volumes": [],
"placementConstraints": [],
"requiresCompatibilities": ["FARGATE"],
"cpu": "1024",
"memory": "2048",
"runtimePlatform": {
  "cpuArchitecture": "ARM64",
  "operatingSystemFamily": "LINUX"
},
"enableFaultInjection": false
}
```