

# Servicios con contenedores (Docker+ECR+ECS)

Autor: Alberto Herrera Poza  
Github: <https://github.com/AlbertoHP>

## *Introducción*

El presente documento tiene la finalidad de implementar un servicio con Amazon Web Services (AWS) y Docker, el cual tendrá en su kernel, una imagen creada con docker build, la cual será guardada en el servicio Elastic Container Registry (ECR) de AWS, y servido a través de ECS con auto escalamiento y balanceo de carga.

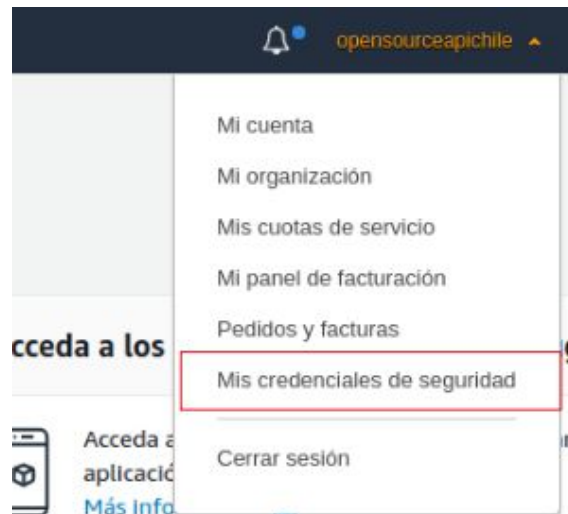
### *1. Configuración e instalación de AWS CLI*

El servicio ECR de AWS, es un repositorio de imágenes **Docker** tal como lo es Docker Hub mantenida por Docker. AWS ofrece este servicio principalmente, porque este, está servido dentro de la red de Amazon, y es mucho más fácil integrarlo y obtener un mejor rendimiento en nuestros servicios con contenedores. Para poder publicar una imagen Docker en un ECR, primero es necesario tener configurado AWS CLI, que es una API de consola bash, este permite la creación de una amplia gama de recursos dentro de AWS. En este específico caso de uso, se utilizará el CLI para poder subir las imagenes Docker al repositorio ECR.

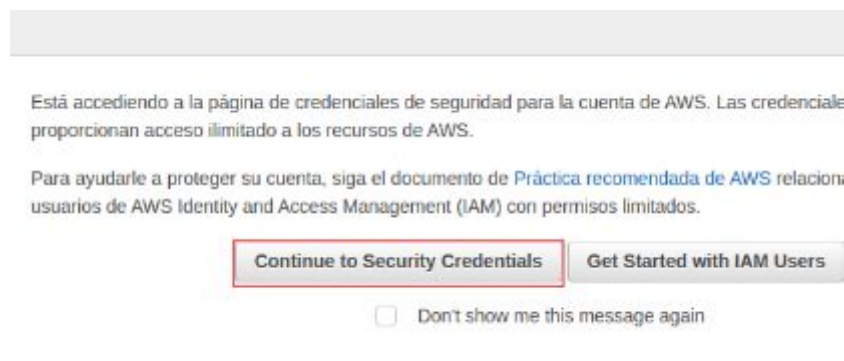
Como recomendación, crear un directorio llamado **programs** o **programas**, en el cual guardar todos aquellos archivos binarios útiles para agregarlos posteriormente a la variable de entorno **PATH** del sistema.

```
cd
mkdir programas
cd programas
curl "https://s3.amazonaws.com/aws-cli/awscli-bundle.zip" -o
"awscli-bundle.zip"
unzip awscli-bundle.zip
sudo ./awscli-bundle/install -i /usr/local/aws -b /usr/local/bin/aws
```

Con lo anterior, el CLI ya se encuentra en la máquina local. Para agregar un perfil (usuario) a nuestro CLI, es necesario en primera instancia obtener las credenciales (ID and Secret keys) y la región donde se trabajara. Para esto nos dirigimos a la consola de AWS



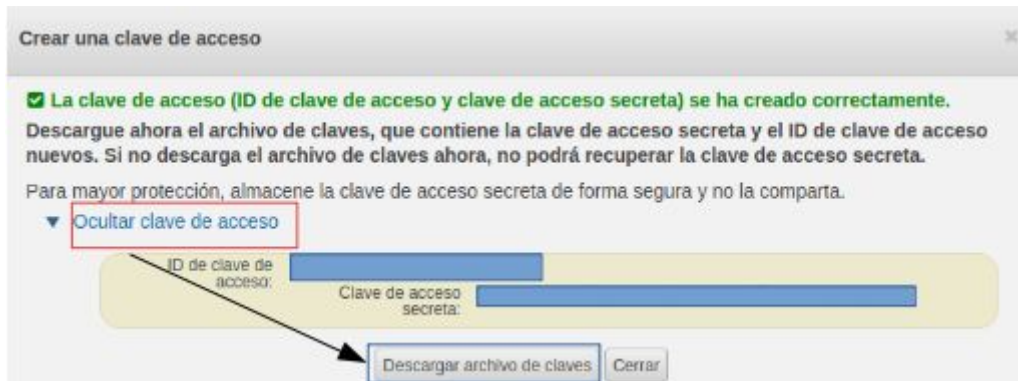
Aparecerá una advertencia con respecto al uso de estas credenciales, que por cierto se recomienda leer al respecto. Ya que el filtrado de un par de estas credenciales puede traducirse en un problema de mayor envergadura, como el levantamiento ilegítimo de instancias para minar bitcoins, robo de información o bases de datos hospedadas en AWS como RDS.



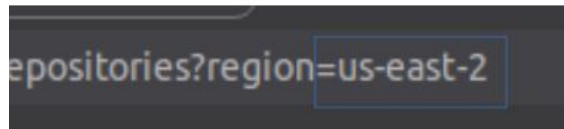
Luego dar en **Claves de acceso (ID de clave de acceso y clave de acceso secreta)** y luego dar en **Crear una clave de acceso**.



Si no hay problemas en la creación, entonces aparecerá un pop-up o modal, el cual mostrará información relevante con respecto a estas nuevas llaves, para verlas dar en **Mostrar la clave de acceso**. Además, es posible descargarlas en formato **csv** dando en **Descargar archivos de claves**.



Por último, se hace necesario saber la región con la que se trabajara en el CLI, para esto fijarse en la barra de direcciones como se muestra en la imagen, en este caso se trabajara con la región **us-east-2**



Anotadas estas últimas **dos claves** de acceso, y la **zona** en la cual se trabajara, ya es posible configurar el CLI a través de los siguientes comandos.

```
sudo chown -R $USER:$USER $HOME/.aws
sudo chown -R $USER:$USER $HOME/.docker
sudo usermod -aG docker $USER
aws configure --profile "opensourceapi"
```

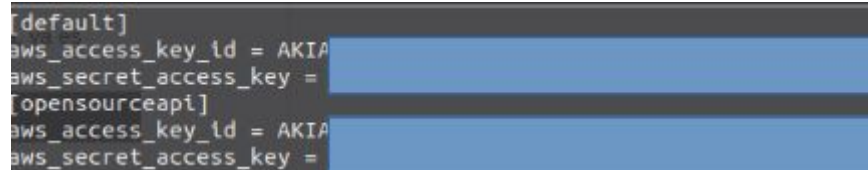
Esto último preguntara por estos tres elementos, teniendo que anotar primero la **ID de clave de acceso** (Access Key ID en inglés), luego la **Clave de acceso secreta** (Secret Access Key en inglés) y por último la **región**.

```
→ Proyectos sudo aws configure --profile opensourceapi
[sudo] contraseña para bebinski:
AWS Access Key ID [None]:
AWS Secret Access Key [None]:
Default region name [None]: us-east-2
Default output format [None]:
```

AWS CLI, guarda en el path **\$HOME/.aws** dos archivos relevantes para la configuración, **config** y **credentials**. Credentials tiene la configuración de las claves de acceso como se ve en la siguiente imagen, asignados a un nodo padre con el tag **[opensourceapi]**. Para abrir el archivo utilizar

```
vim $HOME/.aws/credentials
```

Con esto se abrirá el editor de texto **vim** y podremos ver lo mencionado anteriormente.

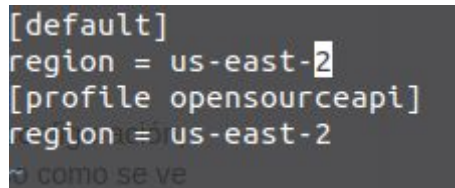


```
[default]
aws_access_key_id = AKIA[redacted]
aws_secret_access_key = [redacted]
[opensourceapi]
aws_access_key_id = AKIA[redacted]
aws_secret_access_key = [redacted]
```

Para salir del editor, dar en la tecla **ESC** y posteriormente escribir **:q!** y dar en la tecla **ENTER**. Esto indica con **q** que deseamos salir (quit) y con el **!** señalamos que esto sea forzado. Por otro lado, tenemos el fichero **config** el cual almacena la configuración regional de los perfiles, para verlo utilizar el comando.

```
vim $HOME/.aws/config
```

Con esto se abrirá el editor de texto **vim** y podremos ver lo mencionado anteriormente.



```
[default]
region = us-east-2
[profile opensourceapi]
region = us-east-2
como se ve
```

Para salir del editor, dar en la tecla **ESC** y posteriormente escribir **:q!** y dar en la tecla **ENTER**. Cabe destacar que el CLI se ejecutó sin **sudo**, por lo que el ambiente de dicho comando es el configurado, si el CLI se ejecutará con **sudo** entonces las credenciales no serían reconocidas. Por último, para reflejar todos los cambios (incluidos los de docker), reiniciar la máquina local con el comando.

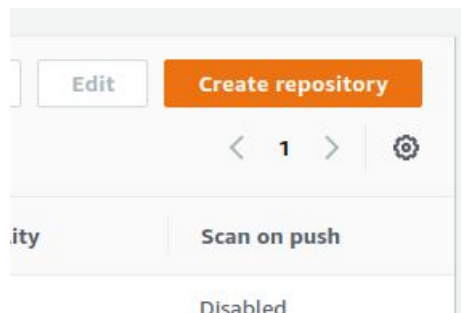
```
reboot
```

## 2. Creación de ECR y subida de imagen Docker

Una vez configurado el AWS CLI, es necesario crear una instancia del servicio ECR de AWS. Para esto ir a la consola, y en servicios dirigirse a ECR.



Una vez ahí, dar en el botón **Create repository**.



AWS permite configurar este repositorio, asignándole un **nombre**, además de ofrecer la opción de **inmutabilidad de tags**, lo que permite versionar y sobrescribir versiones pertenecientes a un mismo tag, y evitar redundancia de imágenes inútiles en nuestro repo. En la siguiente imagen se muestra como nuestro repositorio tendrá el nombre de **opensourceapirepo** y además estará con la **inmutabilidad de tags** activada como se muestra en la imagen.

A screenshot of the 'Create repository' form in the AWS Management Console. The form has a title 'Create repository' and a section 'Repository configuration'. Under 'Repository name', the text '862314791426.dkr.ecr.us-east-2.amazonaws.com/' is followed by a text input field containing 'opensourceapirepo'. Below this, there's a note: 'A namespace can be included with your repository name (e.g. namespace/repo-name)'. Under 'Tag Immutability', there's a description: 'Enable tag immutability to prevent image tags from being overwritten by subsequent image pushes using the same tag. Disable tag immutability to allow image tags to be overwritten.' and a radio button labeled 'Enabled' is selected. Under 'Scan on push', there's a description: 'Enable scan on push to have each image automatically scanned after being pushed to a repository. If disabled, each image scan must be manually started to get scan results.' and a radio button labeled 'Disabled' is selected. At the bottom right, there are two buttons: 'Cancel' and 'Create repository'.

Dar en **Create repository**, y la página será redirigida al dashboard que mostrar nuestro nuevo ECR. Ahora que la instancia de ECR ya se encuentra configurada, es necesario conseguir una imagen Docker para utilizar de ejemplo, y subirla a través del AWS CLI (Se asume que el docker daemon está instalado en la máquina local).

(UPDATE: Este fragmento ha sido modificado debido a la licencia privada del contenedor que se utilizaba en versiones anteriores del documento my-node-container es equivalente de aquí en adelante a gantt-exporter)

```
docker pull node
```

Si no hay problemas, debería ser visible la imagen con el comando.

```
docker image ls
```

Ahora con el AWS CLI podemos publicarla en nuestro **opensourceapirepo ECR**.

```
$(aws ecr get-login --no-include-email --region us-east-2 --profile
opensourceapi)
docker tag node:latest
862314791426.dkr.ecr.us-east-2.amazonaws.com/opensourceapirepo:my-node-c
ontainer
docker push
862314791426.dkr.ecr.us-east-2.amazonaws.com/opensourceapirepo:my-node-c
ontainer
```

Si todo resulta correcto, la imagen subida por docker a AWS ECR será visible desde el dashboard como se muestra en la imagen, donde en **Image URI** tenemos la referencia necesaria para configurar nuestra instancia de AWS ECS.

Images (1)

Find images

↻

Delete



Scan

<

1

>

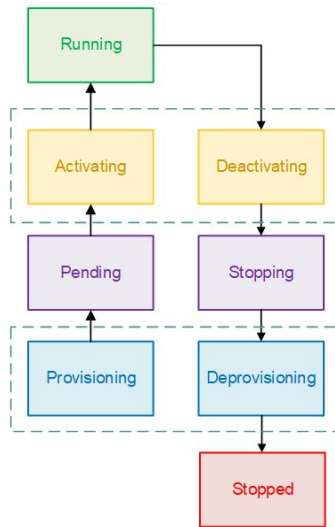
⚙

<input type="checkbox"/>	Image tag	Image URI	Pushed at ▼	Digest	Size (MB) ▼	Scan status	Vulnerabilities
<input type="checkbox"/>	lpsum-gantt-exporter	 862314791426.dkr.ecr.us-east-2.amazonaws.com/opensourceapirepo:lpsum-gantt-exporter	01/08/20, 12:41:19 PM	 sha256:438d826b7...	322.62	-	-

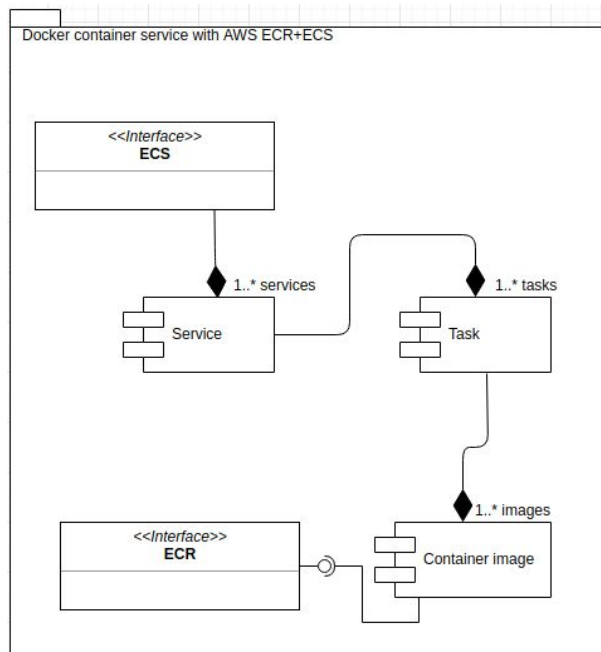
### 3. Creación ECS y configuración para despliegue (deploy) del contenedor

Con la imagen ya en el repositorio de AWS ECR, podemos entonces configurar un nuevo servicio ECS para desplegar nuestro contenedor en cuestión. Para esto es necesario antes, entender un par de conceptos del funcionamiento de ECS. A la fecha de creación de este documento **ECS** cuenta con **dos tipos de lanzamiento**, uno provisto por un servidor (**EC2**) y otro provisto de manera independiente (serverless **Fargate**).

Dentro de **ECS** existe el concepto de **Tarea (Task)**, la cual posee un ciclo de vida propio para servir los contenedores que estén incluidos (Cabe destacar que una tarea puede estar compuesta por varios contenedores ubicados dentro de la misma red desasistida)

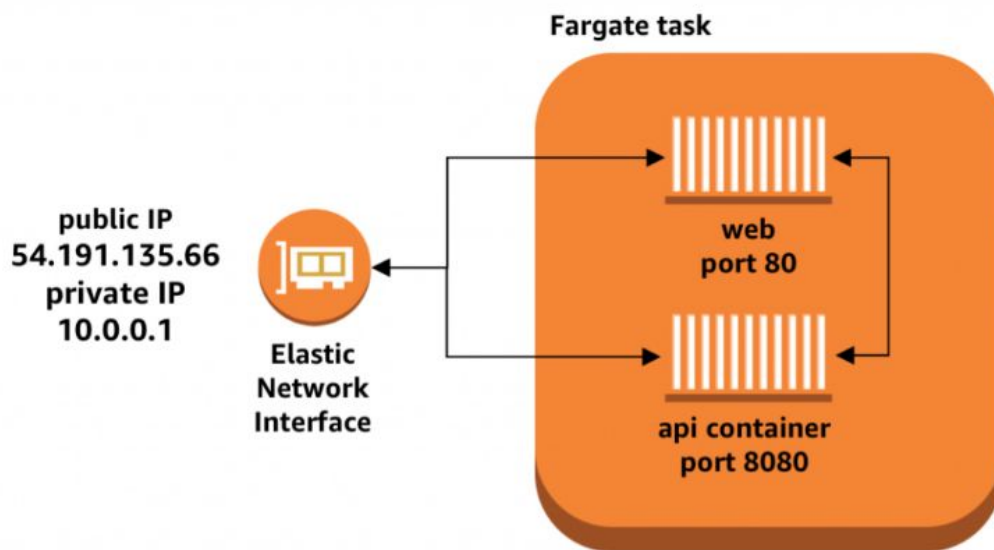


**ECS** utiliza **clusters** en el cual hospeda **servicios**, estos últimos están asociados a una o varias **tareas**, las cuales a su vez pueden estar compuestas por uno o varios **contenedores**.



Este documento trabajara la versión de ECS más reciente, **Fargate**, la cual permite configurar balanceo de carga y auto escalamiento. Además de tener un coste menor debido a la característica **serverless** lo que se traduce en cómputo muerto mientras no hayan peticiones.

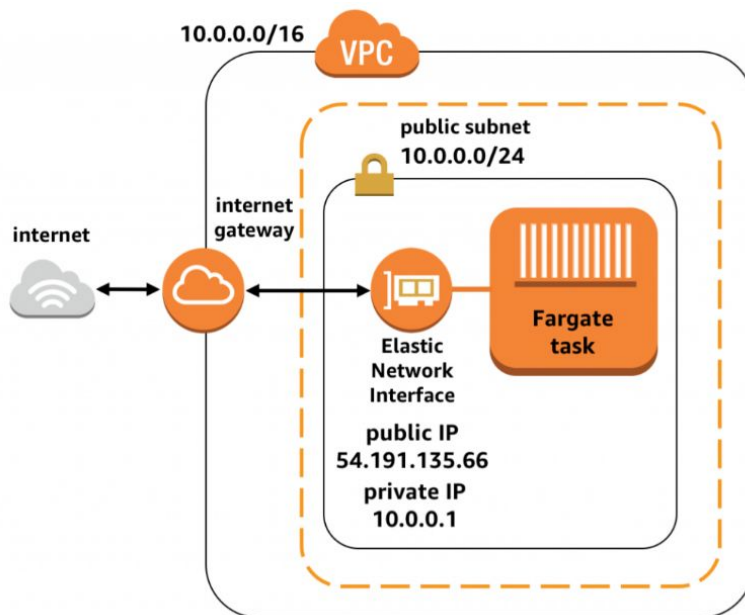
Existen varias configuraciones para desplegar una aplicación de este tipo, pero todas cumplen el mismo estándar, cada tarea configurada, al ser consultada, e iniciar su ciclo de vida, requiere de una **Elastic Network Interface (ENI)**, la cual provee a la tarea de una IP pública y privada. Además de permitirle disponer de persistencia (sólo durante el ciclo de vida de la tarea, cuando este último termina, la persistencia también) como es el caso de los **Docker Volume**. Imaginemos un caso en el que una tarea se compone por dos contenedores, uno en el puerto **80** y otro en el puerto **8080**, estos podrán comunicarse mediante el hostname **localhost** de manera interna y muy eficiente, mientras que a través del **ENI**, esta tarea tendrá acceso de manera externa.



fuelle: <https://aws.amazon.com/es/blogs/compute/task-networking-in-aws-fargate/>

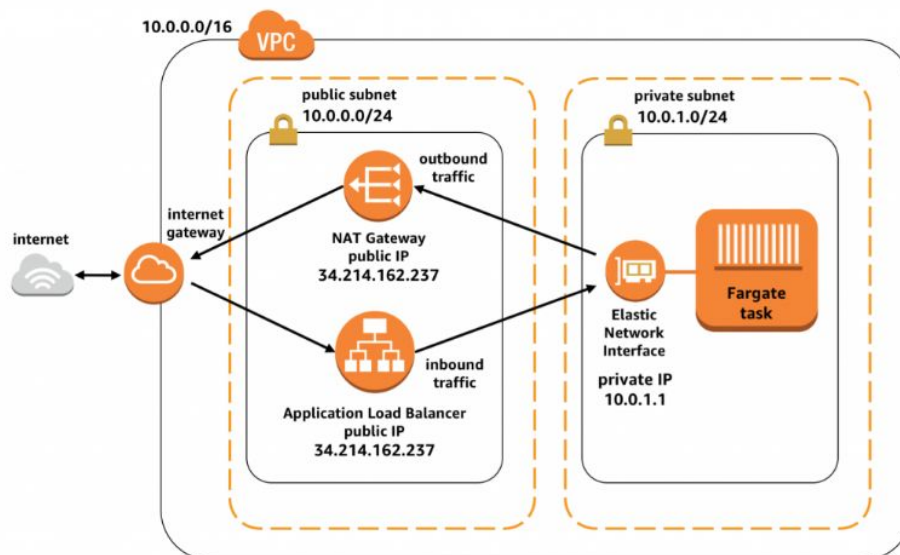
Ahora bien, este es el caso de uso más primitivo de esta feature en **Fargate**, como se comentaba anteriormente, el **ENI** es servido en el inicio del ciclo de vida de la tarea, y matado cuando este concluye. ¿Como entonces disponemos de un endpoint que nos permita de manera elástica (Dirección DNS o IP estática) consultar nuestros servicios, y que por debajo, AWS ECS con **Fargate** decida como reaccionar?. Para esto es necesario en primera instancia contar con un **Amazon Virtual Private Cloud (VPC)** para poder separar del resto de la red **cloud**, con esta nueva subred que se generará para poder levantar las tareas que se definan y sean necesarias.





fuelle: <https://aws.amazon.com/es/blogs/compute/task-networking-in-aws-fargate/>

Por último, es necesario un **Application Load Balancer (ALB)** para poder definir un DNS estático a nuestras tareas.



fuelle: <https://aws.amazon.com/es/blogs/compute/task-networking-in-aws-fargate/>

Como se muestra en la imagen anterior, básicamente será necesario montar un **VPC**, dentro del cual se **crean dos subredes**, la primera encargada del **manejo de las peticiones** (y que nos permite además tener un recurso estático para consultar la API) y la otra en la cual están definidas las **tareas** con sus **ENI** manejadas por su ciclo de vida (de la tarea no del ENI).

Actualmente, AWS ECS ofrece una sección asistida para configurar esto, de todas maneras se hace fundamental entender que es lo que pasa debajo de esta configuración automática, ya que eventualmente puede ser necesario modificar cada una de sus configuraciones (**ya**

sea **Cluster**, **ENI definitions**, **Task definitions**, **Service definitions**, etc.) que de hecho, en el último inciso del presente documento se hace para configurar la persistencia. Para empezar esta configuración ir a la sección de servicios y dar en **ECS**.



Una vez ahí, dar en **Get Started**.

## Clusters

An Amazon ECS cluster is a regional c  
instance type.

For more information, see the [ECS do](#)

Create Cluster

Get Started

View

list

card

Aquí, en **Container definition**, dar en el botón **Edit**.



En **image\*** pegar la dirección del **ECR** configurado en el inciso anterior, y una memoria límite de **2048** mb, en modo **Hard limit** que matara la tarea si este flag es excedido por una petición. Con respecto a los puertos, depende básicamente del contenedor en cuestión, por ejemplo si en el **Dockerfile** se escribe el comando **EXPOSE** cierto puerto, entonces es necesario aca también especificarlo. En este caso, el contenedor trabaja en el puerto **80**, así que no es necesario hacer nada.

▼ Standard

---

Container name\*

Image\*

Private repository authentication\* ☐

Memory Limits (MiB) Hard limit ▼

[+ Add Soft limit](#)

Define hard and/or soft memory limits in MiB for your container. Hard and soft limits correspond to the `memory` and `memoryReservation` parameters, respectively, in task definitions.  
ECS recommends 300-500 MiB as a starting point for web applications.

Port mappings

Container port	Protocol
<input type="text" value="80"/>	<span>tcp ▼</span>

[+ Add port mapping](#)

Es necesario además configurar en **Advanced container configuration**, el script de inicio que viene configurado y eliminarlo.

ENVIRONMENT

CPU units

GPUs

Essential ☒

Entry point

Command



ENVIRONMENT

CPU units

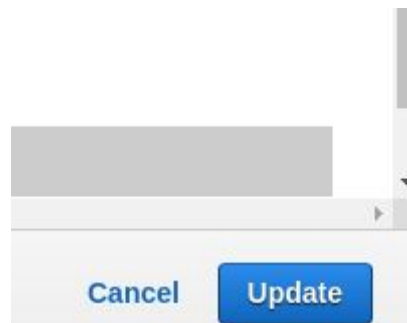
GPUs

Essential ☒

Entry point

Command

Si bien la persistencia no es algo que **Fargate** tenga disponible a fecha de creación de este documento, si la tarea sigue viva, podremos acceder a los elementos (archivos de cualquier tipo) que el contenedor produzca, repito, **solo si esta la tarea viva**, ya que cuando muera, dicha persistencia morirá con ella. Pero antes de esto, continuemos con la creación del AWS ECS, dar en **Update**.



Luego, dar en **edit** de la sección **Task definition**. Ahí, definir un nombre y aumentar a una capacidad razonable (**cada tarea debe ser muy bien definida para identificar la capacidad de cómputo que requiere**) los tamaños manejados por la tarea como se muestra en la siguiente imagen.

Configure task definition: gantt-exporter-ecs-task

---

### Task definition details

Task definition name*	<input type="text" value="gantt-exporter-ecs-task"/>	<a href="#">i</a>
Network mode*	awsvpc	<a href="#">i</a>
Task execution role	<input type="text" value="ecsTaskExecutionRole"/>	<a href="#">i</a>
Compatibilities*	FARGATE <a href="#">Learn more about compatibilities</a>	<a href="#">i</a>

---

### Task size

Task size allows you to size at the task level and optionally set container-specific CPU and memory sizes. You are billed for the task me

Task memory*	<input type="text" value="2GB (2048)"/>
Task CPU*	<input type="text" value="0.5 vCPU (512)"/>

Luego dar en **Save** y **Next**. En la sección de definición del servicio, implementar un balanceador de carga en el puerto **80** como se muestra en la imagen. Luego dar en **Next**.

Define your service Edit

Service allows you to run and maintain a specified number (the "desired count") of simultaneous instances of a task definition in an Amazon ECS cluster.

Service name `gantt-exporter-container-service`

Number of desired tasks `1`

Security group `Automatically create new`

Two security groups are created to secure your service: An Application Load Balancer security group that allows all traffic on the Application Load Balancer port and an Amazon ECS security group that allows all traffic ONLY from the Application Load Balancer security group. You can further configure security groups and network access outside of this wizard.

Load balancer type ☐ None ☒ Application Load Balancer

Load balancer listener port `80`

Load balancer listener protocol `HTTP`

Required Cancel Previous Next

En la sección de **cluster** solo definiremos un nombre y damos en **Next**.

Configure your cluster

The infrastructure in a Fargate cluster is fully managed by AWS. Your containers run without you managing and configuring individual Amazon EC2 instances.

To see key differences between Fargate and standard ECS clusters, see the [Amazon ECS documentation](#).

Cluster name

Cluster names are unique per account per region. Up to 255 letters (uppercase and lowercase), numbers, and hyphens are allowed.

VPC ID `Automatically create new` ⓘ

Subnets `Automatically create new` ⓘ

\*Required Cancel Previous Next

Se mostrará un resumen de toda la configuración, dar en **Create** para iniciar la creación del AWS ECS. Si todo está correcto, deberían desplegarse cada uno de los ítems sin problemas (que por cierto contiene cada uno de los elementos mencionados al principio de este inciso), como se muestra en la siguiente imagen.

## Getting Started with Amazon Elastic Container Service (Amazon ECS) using Fargate

### Launch Status

We are creating resources for your service. This may take up to 10 minutes. When we're complete, you can view your service.

[Back](#)[View service](#)

Additional features that you can add to your service after creation

#### Scale based on metrics

You can configure scaling rules based on CloudWatch metrics

Preparing service : 10 of 10 complete

ECS resource creation	complete
Cluster gant-exporter-ecs-cluster	complete
Task definition gant-exporter-ecs-task:1	complete
Service gant-exporter-container-service	complete
Additional AWS service integrations	complete
Log group /ecs/gant-exporter-ecs-task	complete
CloudFormation stack EC2ContainerService-gant-exporter-ecs-cluster	complete
VPC vpc-03967a83394e2f590	complete
Subnet 1 subnet-046896a9b3506e033	complete
Subnet 2 subnet-0b2cd474795d5e52	complete
Security group sg-01cc11bd5ddef38a8	complete
Load balancer arn:aws:elasticloadbalancing:us-east-2:862314791426:loadbalancer/app/EC2Co-EcsEI-1CPMOB9FQ7KRG/17c6fb57e643cd3	complete

Dar en el botón **View Service**, Aquí ir al balanceador de carga.

[Details](#) [Tasks](#) [Events](#) [Auto Scaling](#) [Deployments](#) [Metrics](#) [Tags](#)

### Load Balancing

Target Group Name	Container Name	Container Port
<a href="#">EC2Co-Defau-1V5RNM08NO30S</a>	gant-exporter-container	80

Esto nos llevará al grupo de red del balanceador, necesitamos ir al balanceador de carga (específico, no su grupo).

1V5RNM08NO30S/96641cb200ee46e

Protocol	HTTP
Port	80
Target type	ip
VPC	<a href="#">vpc-03967a83394e2f590</a>
Load balancer	<a href="#">EC2Co-EcsEI-1CPMOB9FQ7KRG</a>

Aquí, buscar el endpoint (DNS) para consultar el cluster recién creado.

### uration

Name	EC2Co-EcsEI-1CPMOB9FQ7KRG
ARN	arn:aws:elasticloadbalancing:us-east-2:862314791426:loadbalancer/app/EC2Co-Ecs
DNS name	<a href="#">EC2Co-EcsEI-1CPMOB9FQ7KRG-1538500506.us-east-2.elb.amazonaws.com</a>

Es finalmente en esta dirección donde se deben realizar las peticiones.

#### 4. Configuraciones extra (Necesarias)

Ir al cluster recientemente creado, ir al servicio creado y dar en **Update**. Aquí dar en **Next Step**, e ingresar la siguiente configuración en **Health check grace period**.

##### Health check grace period

If your service's tasks take a while to start and respond to ELB health checks, you can specify a health check grace period in seconds during which the ECS service scheduler will ignore ELB health check status. This grace period prevents the ECS service scheduler from marking tasks as unhealthy and stopping them before they have time to come up. Your service is configured to use a load balancer.

Health check grace period

2147483647



Dar en **Next step**, aquí configurar el auto escalamiento de la siguiente manera.

Set Auto Scaling (optional)

##### Service Auto Scaling (optional)

Automatically adjust your service's desired count up and down within a specified range in response to CloudWatch alarms. You can modify your Service Auto Scaling configuration at any time to meet the needs of your application.

- Service Auto Scaling
- ☐ Do not adjust the service's desired count
  - ☒ Configure Service Auto Scaling to adjust your service's desired count

Minimum number of tasks

0



Automatic task scaling policies you set cannot reduce the number of tasks below this number.

Desired number of tasks

1



Maximum number of tasks

10



Automatic task scaling policies you set cannot increase the number of tasks above this number.

IAM role for Service Auto Scaling

ecsAutoscaleRole



##### Automatic task scaling policies

Add scaling policy

Policy name\*

No results

\*Required

Cancel

Previous

Next step

Dar en **Next Step**, se desplegará un resumen, dar en **Update Service**.

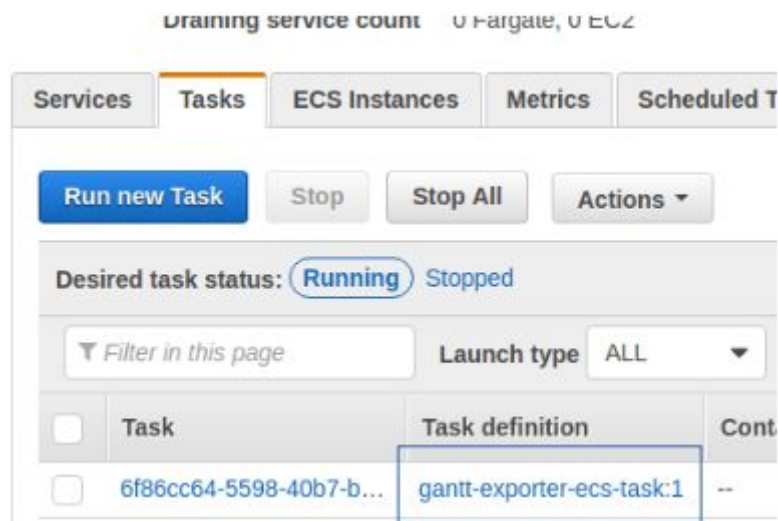
Previous

Update Service



Con esto, nuestro AWS **ECS** ya cuenta con una política de auto escalamiento definida arbitrariamente, y además cuenta con el **Health** check desactivado para evitar matar tareas de manera accidental.

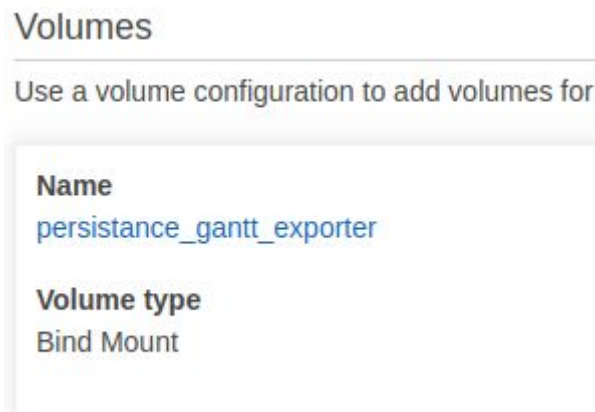
Por último, es necesario configurar un **Docker Volume** y un **Mount Point** a los contenedores. Para esto ir al **cluster** creado, ir a la pestaña de **Task**, y ahí dar en la definición de la tarea.



Luego, dar en **Create new revision**.

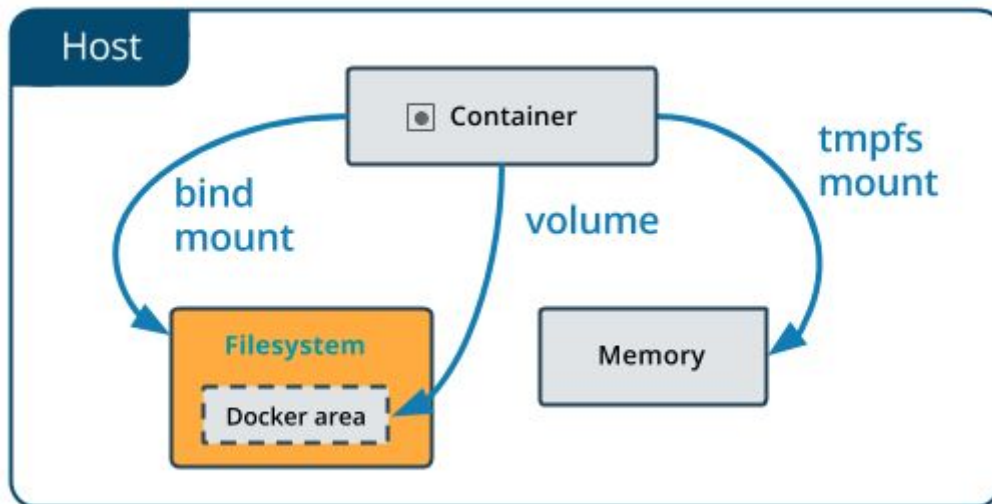


Aquí, en la sección **Volumes** crear el siguiente.





**Bind Mount** es un tipo de Docker volume, en el cual entre el **Host** y el **Contenedor**, se crea un recurso de persistencia, donde el contenido es hospedado.



fuelle: <https://docs.docker.com/storage/bind-mounts/>

Dar en **Create**.



A este **volume** estará asignado un **Mount Point** o punto de montura, mediante el cual podemos especificar a qué **volumen** queremos **dirigir el contenido** de algún **directorio** dentro de nuestro **contenedor**.

Para configurar esto, dar nuevamente en **Create new revision**.



Luego ir a la sección de **Container Definitions** y dar click en el creado.



Se abrirá un pop-up, aquí buscar la sección **STORAGE AND LOGGING**. Aquí, se puede crear el **punto de montura**, en este caso apuntando al **Docker volume** creado anteriormente. Y además, se especifica la **dirección** del directorio dentro del **contenedor**, del cual serán servidos (sacados literalmente para ponerlos en el **Docker volume**) que en este caso es **/sites/export-service**.

**STORAGE AND LOGGING**

Read only root file system ☐

Mount points

Source volume

persistence\_gantt\_exporter

Container path

/sites/export-service

Read only

☐

+ Add mount point

Finalmente dar en **Update** para cerrar la configuración del contenedor, y luego dar en **Create** para crear la nueva definición de tarea.

Con esto, solo queda ir al **cluster**, luego al **servicio**, aquí dar en **Update**, y en la sección **Task Definition** actualizar a la versión creada recientemente, y además dar en **Force new deployment** para forzar el nuevo levantamiento del servicio.

### Configure service

A service lets you specify how many copies of your task definition to run and maintain in a cluster. You can optionally use an Elastic Load Balancing load balancer to distribute incoming traffic to containers in your service. Amazon ECS maintains that number of tasks and coordinates task scheduling with the load balancer. You can also optionally use Service Auto Scaling to adjust the number of tasks in your service.

Task Definition

Family

gantt-exporter-ecs-task

Revision

3 (latest)

Launch type

FARGATE

Platform version

LATEST

Force new deployment

☒

Enter a value

Luego dar en **Skip to review**, y luego en **Update Service**.

Con esto el servicio **Docker AWS ECR+ECS** con **balanceador de carga** y **autoescalamiento** está configurado correctamente y accesible desde el **DNS** del balanceador de carga asociado.