



PRÁCTICA 4: CONTENEDORES

RAÚL MATEUS SÁNCHEZ

COMPUTACIÓN EN LA NUBE
Escuela de Ingeniería Informática



Índice

Introducción	2
Objetivos	2
Desarrollo de la práctica	3
1. Crear un contenedor con Docker que tenga una aplicación que permita comprobar su funcionamiento (e.g. una página web).....	3
2. Crear un repositorio en ECR y subir el contenedor creado en el paso 1	6
3. Desplegar el contenedor usando ECS	9
4. Desplegar el contenedor usando Fargate y comparar la experiencia	17
Presupuesto y estimación de gasto de los recursos desplegados	23
Fuentes	24

Introducción

AWS es muy grande como para poder conocer todo lo que ofrece y todo su potencial en una sola asignatura. En este caso, la influencia y uso de herramientas o servicios que permiten la creación de contenedores para la ejecución de aplicaciones de forma portable ha incrementado a lo largo del tiempo en estos últimos años, siendo fundamental tener conocimiento sobre contenedores y su uso.

Objetivos

El objetivo de esta práctica es, principalmente, aprender a utilizar contenedores en AWS usando los servicios de ECS

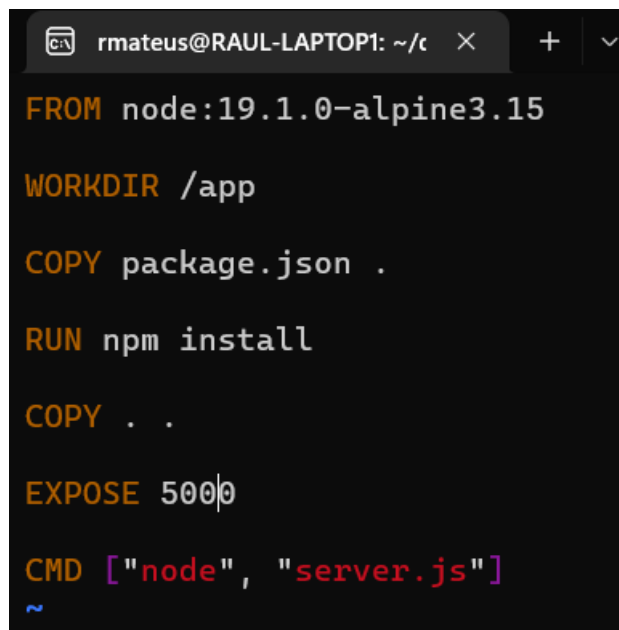
Desarrollo de la práctica

1. Crear un contenedor con Docker que tenga una aplicación que permita comprobar su funcionamiento (e.g. una página web)

Para desarrollar este apartado de la práctica, debido a los recursos y el tiempo, se desplegará en la versión de Docker Desktop para Windows junto con el subsistema Linux para Windows mediante Ubuntu.

Se instala Docker en Windows y se configura de forma que quede totalmente funcional, y se inicia Ubuntu, en donde se dará forma al contenedor. Para crearlo, se crean tres ficheros: Un Dockerfile, un archivo JSON y un fichero JavaScript, según las referencias de pie de página.¹²

La aplicación a desplegar será un servidor web muy básico, aunque en este caso en vez de usar *nginx*, emplearemos la imagen Alpine de Node.js, la cual reducirá al máximo el tamaño de la imagen, permitiendo reducir además la huella que deja y, por tanto, las vulnerabilidades.

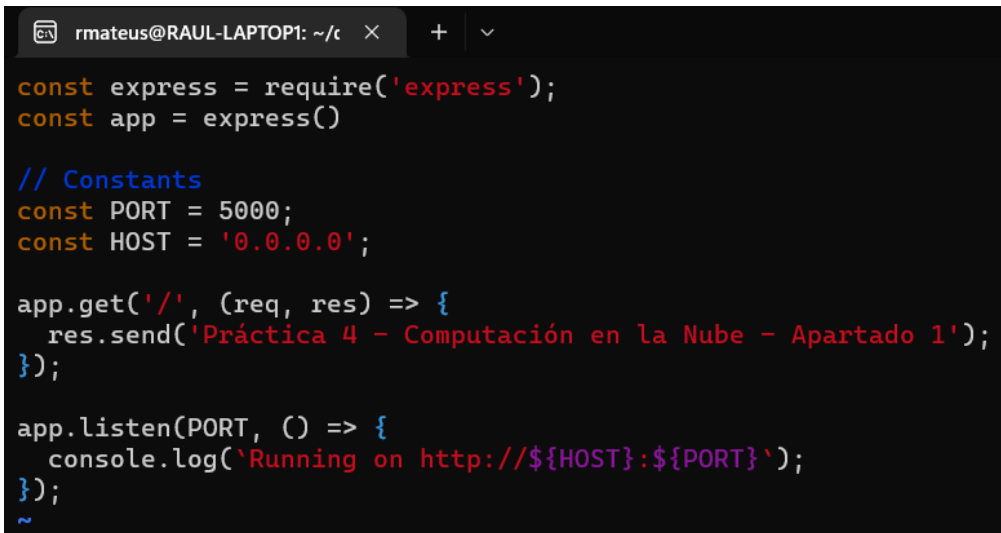


```
rmateus@RAUL-LAPTOP1: ~/c × + v
FROM node:19.1.0-alpine3.15
WORKDIR /app
COPY package.json .
RUN npm install
COPY . .
EXPOSE 5000
CMD ["node", "server.js"]
~
```

Figura 1: Dockerfile para contenedor

¹ <https://nodejs.org/en/docs/guides/nodejs-docker-webapp/>

² <https://www.youtube.com/watch?v=YDNSItBN15w>



```

rmateus@RAUL-LAPTOP1: ~/c  X  +  v
const express = require('express');
const app = express()

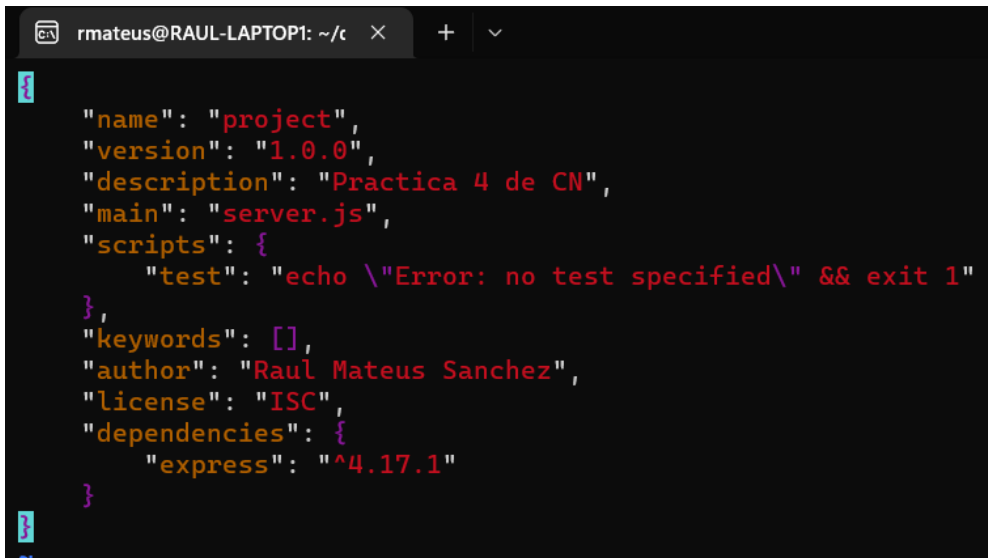
// Constants
const PORT = 5000;
const HOST = '0.0.0.0';

app.get('/', (req, res) => {
  res.send('Práctica 4 - Computación en la Nube - Apartado 1');
});

app.listen(PORT, () => {
  console.log(`Running on http://${HOST}:${PORT}`);
});
~

```

Figura 2: server.js



```

rmateus@RAUL-LAPTOP1: ~/c  X  +  v
{
  "name": "project",
  "version": "1.0.0",
  "description": "Practica 4 de CN",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "Raul Mateus Sanchez",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1"
  }
}
~

```

Figura 3: package.json

En primer lugar, se declara un Dockerfile siguiendo las recomendaciones de los tutoriales, donde se declara una serie de comandos que permite la creación de la imagen que se quiere. Se instalarán las dependencias necesarias declaradas en el JSON y se indica que se escuchará por el puerto 5000 para finalmente, ejecutar el server.js creado en el contenedor instanciado.

A continuación, un fichero JavaScript que describirá la aplicación que se desplegará. En este caso, siguiendo algunos tutoriales de YouTube o NodeJS referenciados anteriormente, se declara una aplicación en Express que funcionará como servidor web, escuchando por el puerto 5000 las peticiones, ya que el puerto 8080 presenta problemas, al menos en Windows.

Por último, un fichero JSON, en el cual se declara cierta información como las dependencias necesarias o la autoría del proyecto, aunque en nuestro caso no tendrá casi influencia.

Una vez se ejecutan los comandos correspondientes a la creación o despliegue del contenedor y su puesta en ejecución, se puede verificar su funcionamiento entrando en el servidor:

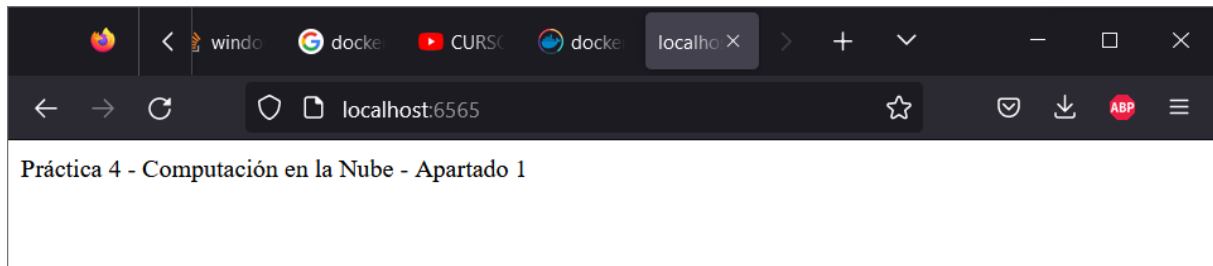


Figura 4: Servidor web en funcionamiento

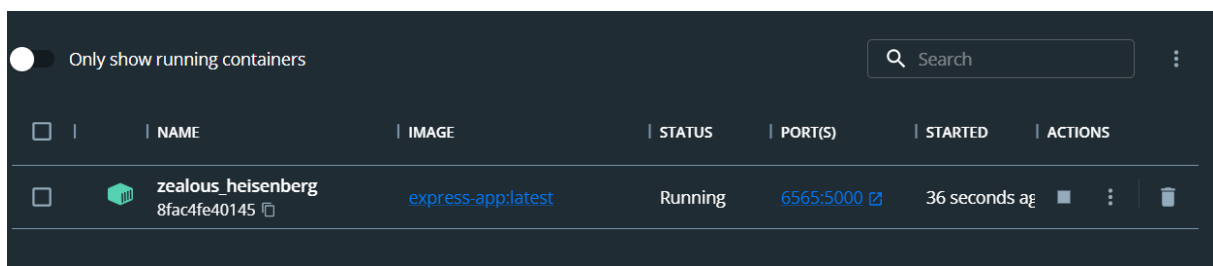


Figura 5: Declaración del contenedor en Docker Desktop

- Para despliegue del contenedor: `docker build -t express-app .`
- Para iniciar el contenedor: `docker run -p 6565:5000 express-app`

2. Crear un repositorio en ECR y subir el contenedor creado en el paso 1

Desde AWS, buscamos la sección de ECR y entramos. Una vez dentro, es bastante intuitivo el proceso, seleccionando la opción de crear repositorio. Durante este proceso, solo se debe tener en cuenta dos cosas: Establecer la visibilidad del repositorio a privada, con el fin de evitar errores de creación por falta de permisos, e introducir el nombre de nuestro nuevo repositorio.

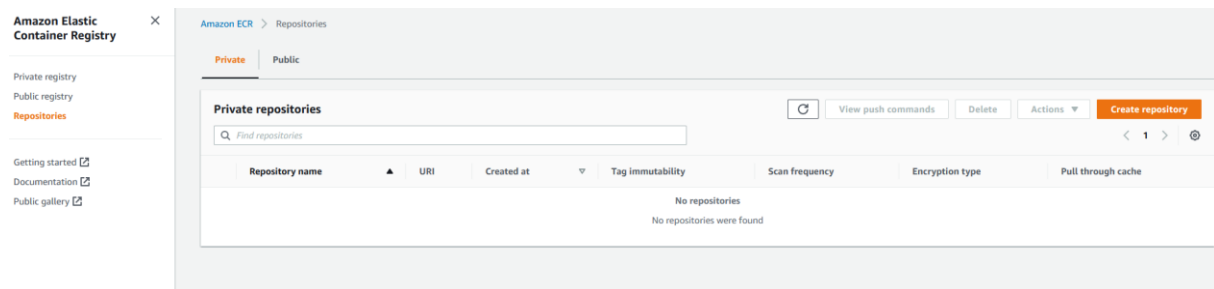


Figura 6: Interfaz de repositorios de ECR

Create repository

General settings

Visibility settings | [Info](#)
Choose the visibility setting for the repository.

☒ **Private**
Access is managed by IAM and repository policy permissions.

☐ **Public**
Publicly visible and accessible for image pulls.

Repository name
Provide a concise name. A developer should be able to identify the repository contents by the name.

340954026236.dkr.ecr.us-east-1.amazonaws.com/

13 out of 256 characters maximum (2 minimum). The name must start with a letter and can only contain lowercase letters, numbers, hyphens, underscores, periods and forward slashes.

Tag immutability | [Info](#)
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.

☐ **Disabled**

i Once a repository is created, the visibility setting of the repository can't be changed.

Figura 7: Creación de repositorio en ECR

Una vez creado el repositorio, se debe subir la imagen creada y probada al repositorio. Para ello, AWS suministra una serie de instrucciones y comandos para autenticar Docker con el

registro y poder subir la imagen. Antes de ejecutar estas instrucciones, debemos autenticarnos con nuestra cuenta de AWS.

Push commands for server1-p4-cn ✕

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-east-1 | docker login --username AWS --password-stdin 340954026236.dkr.ecr.us-east-1.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 server1-p4-cn .
```
3. After the build completes, tag your image so you can push the image to this repository:

```
docker tag server1-p4-cn:latest 340954026236.dkr.ecr.us-east-1.amazonaws.com/server1-p4-cn:latest
```
4. Run the following command to push this image to your newly created AWS repository:

```
docker push 340954026236.dkr.ecr.us-east-1.amazonaws.com/server1-p4-cn:latest
```

Close

Figura 8: Instrucciones de AWS para hacer "push" al repositorio

Antes de proceder a la autenticación, se descarga la última versión de AWS CLI, y mediante la orden `aws configure`, podemos hacer todo el proceso. La información suministrada a la orden se encuentra en el Learner Lab de la sesión.

```
rmateus@RAUL-LAPTOP1:~$ aws configure
AWS Access Key ID [None]: ASIAU6YTRSD6AC42ERSR
AWS Secret Access Key [None]: lDVTZWBD06QEG086hSlN0urMkbS5GQ329BAKrOX
Default region name [None]: us-east-1
Default output format [None]: json
```

Figura 9: `aws configure`


```
rmateus@RAUL-LAPTOP1:~$ aws configure set aws_session_token FwoGZXIvYXZELf////////wEaDCsvdoLyInV6IfJWJyLEAaNmYqMej+hk
uT9RDjE76W2dfcbW4f6WiYFj4E3/tYWuoEMJinbYpEvkvfGIBxu5BKTWahSxQuS8W/nGjkJoGVdFLbk1JpN2batB4rf+I52opnMo2I9Vu0SQtEnH9t/en6
XbTG0pLI9Yu65iajzTnbQpJg+WMX5Tyotde870mmDZ/k6520e50B2gVU5EP8VeTG0xEUbr0ccLQwtdqHZLLne+R3JBqS2gaSQmU8yXEkJz7hYIAcc0xvMqJ
hkTLceXZUAko922tnAYyLT9UqZeQLZpIc2ejUx7GBJYkIcz0tjVfL11sIMHuQFGIpu8x3fqc1nC1e3Icrw==
rmateus@RAUL-LAPTOP1:~$ aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 340
954026236.dkr.ecr.us-east-1.amazonaws.com
Login Succeeded
```

Figura 10: Input de aws_session_token

Ya autenticados, tanto nosotros en AWS como el cliente Docker, podemos seguir las instrucciones para subir la imagen.

```
rmateus@RAUL-LAPTOP1:~/server1-p4-cn$ docker build -t server1-p4-cn .
[+] Building 2.6s (10/10) FINISHED
=> [internal] load build definition from Dockerfile                                0.0s
=> => transferring dockerfile: 175B                                              0.0s
=> [internal] load .dockerignore                                                  0.0s
=> => transferring context: 2B                                                  0.0s
=> [internal] load metadata for docker.io/library/node:19.1.0-alpine3.15        2.4s
=> [1/5] FROM docker.io/library/node:19.1.0-alpine3.15@sha256:8d87f28bd6f8a6c717e0be1defc297b22672b79820c83a7087  0.0s
=> [internal] load build context                                                0.0s
=> => transferring context: 886B                                                0.0s
=> CACHED [2/5] WORKDIR /app                                                    0.0s
=> CACHED [3/5] COPY package.json .                                             0.0s
=> CACHED [4/5] RUN npm install                                                  0.0s
=> CACHED [5/5] COPY . .                                                        0.0s
=> exporting to image                                                            0.0s
=> => exporting layers                                                            0.0s
=> => writing image sha256:9e8c54583647fd5f454ad46631fe4ace73da0768a83d5fd374ae693255df2918 0.0s
=> => naming to docker.io/library/server1-p4-cn                                0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
rmateus@RAUL-LAPTOP1:~/server1-p4-cn$ docker images
REPOSITORY          TAG          IMAGE ID          CREATED          SIZE
server1-p4-cn       latest       9e8c54583647     2 days ago      175MB
docker/getting-started latest     e5be50c31cb9     7 days ago      29.8MB
rmateus@RAUL-LAPTOP1:~/server1-p4-cn$ docker tag server1-p4-cn:latest 340954026236.dkr.ecr.us-east-1.amazonaws.com/server1-p4-cn:latest
```

Figura 11: Build del contenedor y "tag" de la imagen

```
rmateus@RAUL-LAPTOP1:~/server1-p4-cn$ docker push 340954026236.dkr.ecr.us-east-1.amazonaws.com/server1-p4-cn:latest
The push refers to repository [340954026236.dkr.ecr.us-east-1.amazonaws.com/server1-p4-cn]
a1410b832cbe: Pushed
9523010346c7: Pushed
5e4baebefe82: Pushed
b0ee63f421c0: Pushed
a76198018474: Pushed
c8537d005f84: Pushed
078e8c4e1d44: Pushed
34d5ebaa5410: Pushed
latest: digest: sha256:ae388beb8681f496408a780b935d8a4f63899d42d4254e084ef30820e0d614b7 size: 1990
rmateus@RAUL-LAPTOP1:~/server1-p4-cn$
```

Figura 12: Push de la imagen al repositorio

Una vez subida, basta con comprobar el repositorio para ver la imagen en el mismo.

server1-p4-cn									
Images (1)									
Find images									
<input type="checkbox"/>	Image tag	Artifact type	Pushed at	Size (MB)	Image URI	Digest	Scan status	Vulnerabilities	
<input type="checkbox"/>	latest	Image	03 de diciembre de 2022, 14:25:07 (UTC-00)	54.56	Copy URI	sha256:ae388beb8681f496408a780b935d8a4f63899d42d4254e084ef30820e0d614b7	-	-	

Figura 13: Repositorio con imagen subida

3. Desplegar el contenedor usando ECS

Para desplegar un contenedor usando ECS, se requiere el empleo de un clúster, así como una “Task Definition” para ello. En este caso, el clúster será la infraestructura donde se ejecutará el contenedor, mientras que la tarea contendrá la creación del contenedor a partir de la imagen subida al repositorio de forma, abstrayendo al usuario de este proceso.

En este apartado, se crea un clúster usando ECS:

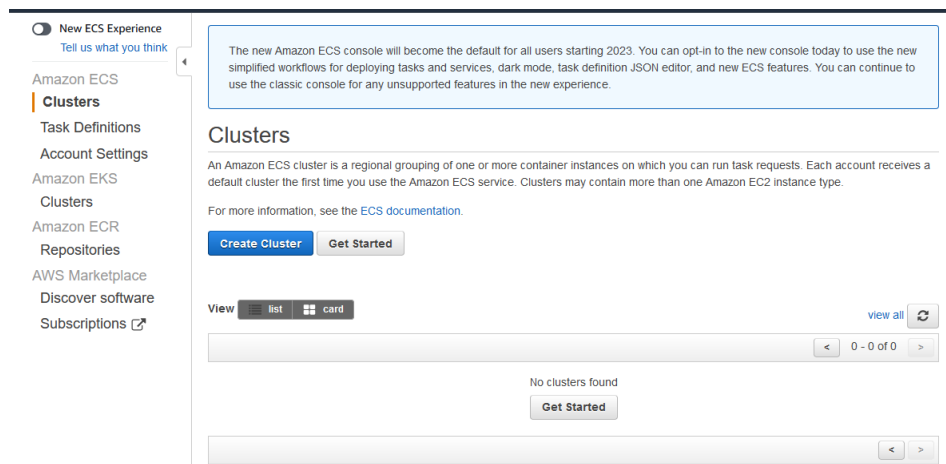


Figura 14: Interfaz de ECS

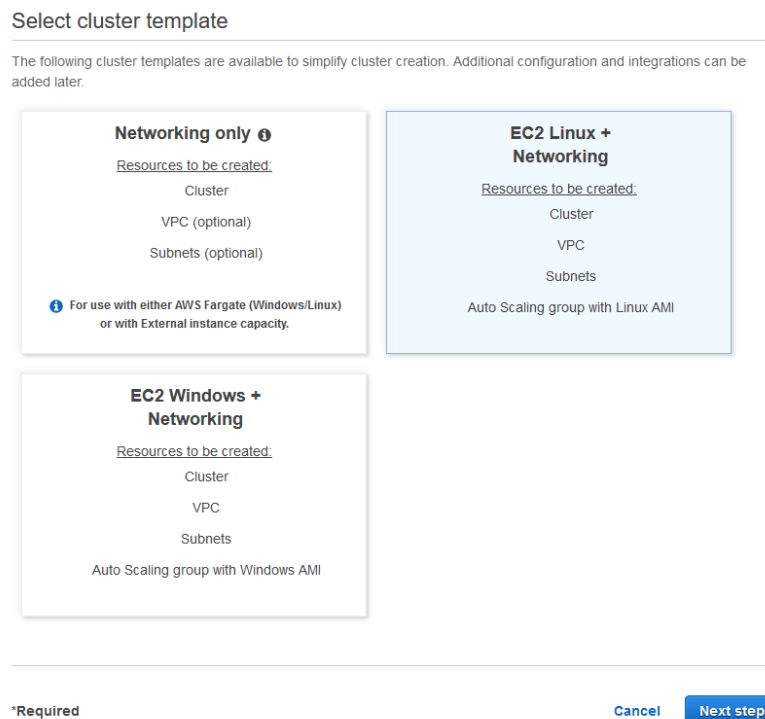


Figura 15: Selección del tipo de clúster

En este caso, se escoge la opción EC2 Linux + Networking, ya que permite el uso de una instancia externa, aunque la primera opción también funcionaría.

Configure cluster

Cluster name* ECSCluster-CN ⓘ

☐ Create an empty cluster

Instance configuration

Provisioning Model ☒ On-Demand Instance

With On-Demand Instances, you pay for compute capacity by the hour, with no long-term commitments or upfront payments.

☐ Spot

Amazon EC2 Spot Instances let you take advantage of unused EC2 capacity in the AWS cloud. Spot Instances are available at up to a 90% discount compared to On-Demand prices. [Learn more](#)

EC2 instance type* t3.micro ⓘ

☐ Manually enter desired instance type

Number of instances* 1 ⓘ

EC2 AMI ID* Amazon Linux 2 AMI [ami-0fe5f...] ⓘ

Root EBS Volume Size (GiB) 30 ⓘ

Key pair SSHGate-key ⓘ

You will not be able to SSH into your EC2

Figura 16: Configuración del clúster

En este apartado no hay mucho que hablar, pues simplemente se asigna un nombre al clúster, así como el tipo de instancia y un par de claves, similar a cuando creamos una simple instancia en EC2.

EC2 > Security Groups > sg-0eb7b329fcb920ab6 - cluster-security-group

sg-0eb7b329fcb920ab6 - cluster-security-group

Actions

Details

Security group name cluster-security-group	Security group ID sg-0eb7b329fcb920ab6	Description SG for Clusters - P4 - CN	VPC ID vpc-088feb7f494a9d6b
Owner 340954026236	Inbound rules count 3 Permission entries	Outbound rules count 1 Permission entry	

Inbound rules
Outbound rules
Tags

Inbound rules (1/3)
Manage tags
Edit inbound rules

Name	Security group rule...	IP version	Type	Protocol	Port range	Source	Description
-	sgr-086df9a55cd40eb93	-	All traffic	All	All	sg-06e5d4782f9d606c...	-
-	sgr-06e8c0f4e82a51f48	IPv6	Custom TCP	TCP	5000	::/0	Cualquier IPv6 puede ...
-	sgr-06a225047f7403dc2	IPv4	Custom TCP	TCP	5000	0.0.0.0/0	Cualquier IPv4 puede ...

Figura 17: Creación de un SG para el clúster

Networking

Configure the VPC for your container instances to use. A VPC is an isolated portion of the AWS cloud populated by AWS objects, such as Amazon EC2 instances. You can choose an existing VPC, or create a new one with this wizard.

VPC
vpc-088feb7f494a9d6b...

Check the structure for [vpc-088feb7f494a9d6b](#) in the Amazon EC2 console.

Subnets

subnet-067218daac8437c2d
(172.31.16.0/20) - us-east-1c
assign ipv6 on creation: Disabled

Select a subnet...

Auto assign public IP
Use subnet setting

Security group
sg-0eb7b329fcb920ab6...

Rules for [sg-0eb7b329fcb920ab6](#) in the EC2 Console.

Figura 18: Configuración de red para el clúster

Para la configuración de red, es necesario asignar un grupo de seguridad. En este caso, se opta por crear un nuevo grupo de seguridad específico para este clúster, así como el que se creará posteriormente, con reglas específicas para permitir la conexión al puerto 5000.

Container instance IAM role

The Amazon ECS container agent makes calls to the Amazon ECS API actions on your behalf, so container instances that run the agent require the `ecsInstanceRole` IAM policy and role for the service to know that the agent belongs to you. If you do not have the `ecsInstanceRole` already, we can create one for you.

Container instance IAM role LabRole ⓘ

For container instances to receive the new ARN and resource ID format, the root user needs to opt in for the container instance IAM role. Opt in and try again.

Tags

Key	Value
<input type="text" value="Add key"/>	<input type="text" value="Add value"/>

CloudWatch Container Insights

CloudWatch Container Insights is a monitoring and troubleshooting solution for containerized applications and microservices. It collects, aggregates, and summarizes compute utilization such as CPU, memory, disk, and network; and diagnostic information such as container restart failures to help you isolate issues with your clusters and resolve them quickly. [Learn more](#)

CloudWatch Container Insights ☐ Enable Container Insights

Figura 19: IAM Role

[Back](#) [View Cluster](#)

ECS status - 3 of 3 complete **ECSCluster-CN**

✓

ECS cluster
ECS Cluster ECSCluster-CN successfully created

✓

ECS Instance IAM Policy
IAM Policy for the role LabInstanceProfile successfully attached

✓

CloudFormation Stack
CloudFormation stack EC2ContainerService-ECSCluster-CN and its resources successfully created

Cluster Resources

Instance type	t3.micro
Desired number of instances	1
Key pair	SSHGate-key
ECS AMI ID	ami-0e5f356c-083f5ca
VPC	vpc-088b0f71a94a95b
Subnets	subnet-067215daac8437c2d
VPC Availability Zones	us-east-1a, us-east-1b, us-east-1c, us-east-1d, us-east-1e, us-east-1f
Security group	sg-0eb7b329kb920ab6
Launch configuration	EC2ContainerService-ECSCluster-CN-EcsInstanceLC-S88wepoSD2s5
Auto Scaling group	EC2ContainerService-ECSCluster-CN-EcsInstanceAG-1SR2RG096C8R

Figura 20: Clúster creado exitosamente

Ya configurado el clúster, se procede a su creación. En caso de éxito, deberá aparecer como en la *Figura 20: Clúster creado exitosamente*. A partir de aquí, ya podemos crear la tarea correspondiente para poder desplegar el contenedor.

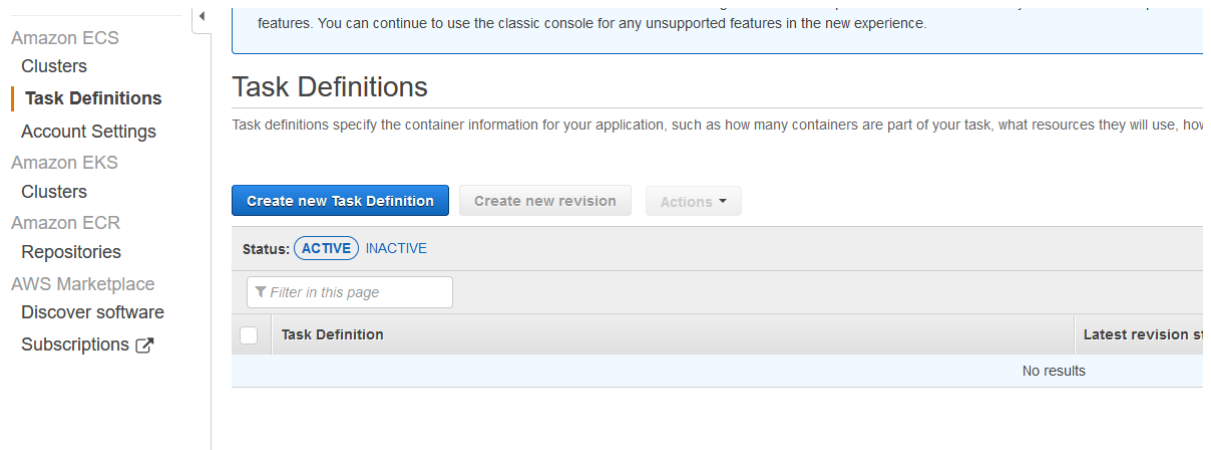


Figura 21: Interfaz de "Task Definitions"

Create new Task Definition


Step 1: Select launch type compatibility

Step 2: Configure task and container definitions

Select launch type compatibility


Select which launch type you want your task definition to be compatible with based on where you want to launch your task.

FARGATE




Price based on task size
Requires network mode awsvpc
AWS-managed infrastructure, no Amazon EC2 instances to manage

EC2



Price based on resource usage
Multiple network modes available
Self-managed infrastructure using Amazon EC2 instances

EXTERNAL



Price based on instance-hours and additional charges for other AWS services used
Self-managed on-premise infrastructure with ECS
Anywhere

Figura 22: Creación de nueva "Task Definition"

En este punto, se elige "EC2" como tipo de despliegue de acuerdo con el clúster creado. A continuación, se configura la tarea, así como la definición del contenedor.

Configure task and container definitions

A task definition specifies which containers are included in your task and how they interact with each other. You can also specify data volumes for your containers to use. [Learn more](#)

Task definition name* t1-p4 ⓘ

Requires compatibilities* EC2

Task role None ⓘ
Optional IAM role that tasks can use to make API requests to authorized AWS services. Create an Amazon Elastic Container Service Task Role in the [IAM Console](#) ⓘ

Network mode <default> ⓘ
If you choose <default>, ECS will start your container using Docker's default networking mode, which is Bridge on Linux and NAT on Windows. Windows tasks support the <default> and awsvpc network modes.

Task execution IAM role

This role is required by tasks to pull container images and publish container logs to Amazon CloudWatch on your behalf. If you do not have the ecsTaskExecutionRole already, we can create one for you.

Task execution role None ⓘ

Figura 23: Configuración de la tarea

▼ Standard

Container name* container-p4-ecs ⓘ

Image* 340954026236.dkr.ecr.us-east-1.amazonaws.com/server1-p4-cn ⓘ

Private repository authentication* ☐ ⓘ

Memory Limits (MiB)* Hard limit 128 ⓘ

➕ 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	Host port	Container port	Protocol
	5000	5000	tcp ⓘ

➕ Add port mapping

Figura 24: Declaración del contenedor

Task size



The task size allows you to specify a fixed size for your task. Task size is required for tasks using the Fargate launch type and is optional for the EC2 or External launch type. Container level memory settings are optional when task size is set. Task size is not supported for Windows containers.

Task memory (MiB)

The amount of memory (in MiB) used by the task. It can be expressed as an integer using MiB, for example 1024, or as a string using GB, for example '1GB' or '1 gb'.

Task CPU (unit)

The number of CPU units used by the task. It can be expressed as an integer using CPU units, for example 1024, or as a string using vCPUs, for example '1 vCPU' or '1 vcpu'.

Task memory maximum allocation for container memory reservation



0 500 shared of 500 MiB

Task CPU maximum allocation for containers



0 1024 shared of 1024 CPU units

Container definitions



Add container



Container ...	Image	Hard/Soft ...	CPU Units	GPU	Inference ...	Essential	
 contain...	3409540262...	--/--				true	

Figura 25: Límites de CPU y memoria

En la configuración de la tarea se destacan algunas cosas. En primer lugar, se le asigna un nombre y se asocia la imagen subida anteriormente al contenedor definido por la tarea mediante la URI que nos da ECR, así como se especifica el mapeo de puertos, donde se sitúa el puerto 5000.

Por otra parte, se establece un límite de memoria y CPU para la tarea de forma que no acapare más recursos de lo que necesita y se quiere.

Finalmente, una vez creada la definición de la tarea y del contenedor, podemos desplegar el contenedor mediante la opción "Run Task".

Run Task

Select the cluster to run your task definition on and the number of copies of that task to run. To apply container overrides or target particular container instances, click Advanced Options.

Launch type ☐ FARGATE ☒ EC2 ☐ EXTERNAL ⓘ

[Switch to capacity provider strategy](#) ⓘ

Task Definition Family
t1-p4

Revision
1 (latest)

Cluster ECSCluster-CN ▾

Number of tasks 1

Task Group ⓘ

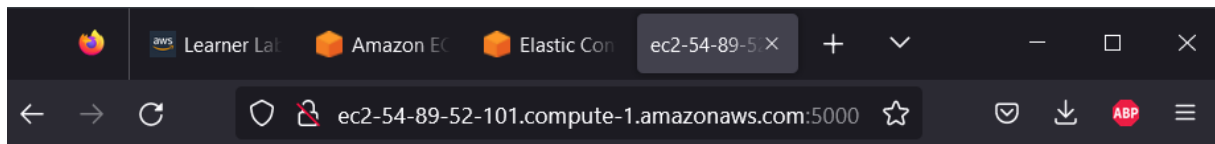
VPC and security groups

VPC and security groups are configurable when your task definition uses the awsvpc network mode.

Task Placement

Figura 26: Run Task

Se procede a elegir la definición de la tarea creada, la cual será lanzada mediante una instancia EC2 que quedará asociada al clúster por el momento. Ya ejecutado, podemos comprobar el correcto funcionamiento mediante la dirección pública que nos suministra la tarea o la instancia, accesible mediante la opción Tasks o EC2 Instances dentro del clúster.



Práctica 4 - Computación en la Nube - Apartado 1

Figura 27: Contenedor en ECS desplegado correctamente

4. Desplegar el contenedor usando Fargate y comparar la experiencia

Este apartado es similar al anterior, pero haciendo el despliegue mediante Fargate así como crear una definición de tarea de tipo Fargate.

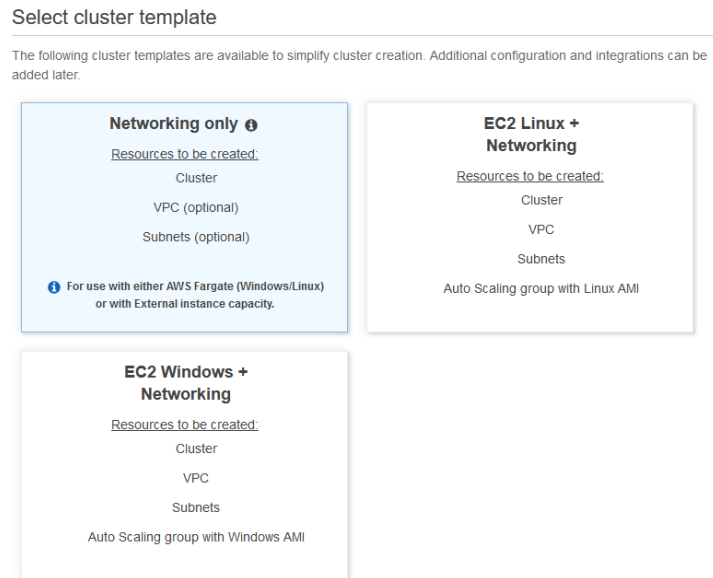


Figura 28: Selección de tipo de clúster

Configure cluster

Cluster name* FARGATE-Cluster-CN ⓘ

Networking

Create a new VPC for your cluster to use. A VPC is an isolated portion of the AWS Cloud populated by AWS objects, such as Fargate tasks.

Create VPC ☐ Create a new VPC for this cluster

Tags

Key	Value
name	Fargate Cluster
Add key	Add value

CloudWatch Container Insights

CloudWatch Container Insights is a monitoring and troubleshooting solution for containerized applications and microservices. It collects, aggregates, and summarizes compute utilization such as CPU, memory, disk, and network; and diagnostic information such as container restart failures to help you isolate issues with your clusters and resolve them quickly. [Learn more](#)

CloudWatch Container Insights ☐ Enable Container Insights

*Required

Cancel

Previous

Create

Figura 29: Configuración del clúster – Fargate

La creación de este clúster es bastante más sencilla que el apartado anterior, pues solo necesita un nombre y nada más.

Lo siguiente sería crear una nueva “Task Definition” compatible con Fargate:

Create new Task Definition

Step 1: Select launch type compatibility

Step 2: Configure task and container definitions

Select launch type compatibility

Select which launch type you want your task definition to be compatible with based on where you want to launch your task.




FARGATE  Price based on task size Requires network mode awsvpc AWS-managed infrastructure, no Amazon EC2 instances to manage	EC2  Price based on resource usage Multiple network modes available Self-managed infrastructure using Amazon EC2 instances
EXTERNAL  Price based on instance-hours and additional charges for other AWS services used Self-managed on-premise infrastructure with ECS Anywhere	

Figura 30: Creación de una nueva “Task Definition”

Configure task and container definitions

A task definition specifies which containers are included in your task and how they interact with each other. You can also specify data volumes for your containers to use. [Learn more](#)

Task definition name* ⓘ

Requires compatibilities* FARGATE

Task role ⓘ
Optional IAM role that tasks can use to make API requests to authorized AWS services. Create an Amazon Elastic Container Service Task Role in the [IAM Console](#) ⓘ

Network mode ⓘ
If you choose <default>, ECS will start your container using Docker's default networking mode, which is Bridge on Linux and NAT on Windows. Windows tasks support the <default> and awsvpc network modes.

Operating system family ⓘ

Figura 31: Configuración de la tarea

Standard

Container name* ⓘ

Image* ⓘ

Private repository authentication* ☐ ⓘ

Memory Limits (MiB) ⓘ
[+ Add Hard 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="5000"/>	<input type="text" value="tcp"/>

[+ Add port mapping](#) ⓘ

Figura 32: Configuración del contenedor

Task execution IAM role

This role is required by tasks to pull container images and publish container logs to Amazon CloudWatch on your behalf. If you do not have the `ecsTaskExecutionRole` already, we can create one for you.

Task execution role

LabRole

Task size

The task size allows you to specify a fixed size for your task. Task size is required for tasks using the Fargate launch type and is optional for the EC2 or External launch type. Container level memory settings are optional when task size is set. Task size is not supported for Windows containers.

Task memory (GB)

0.5GB

The valid memory range for 0.25 vCPU is: 0.5GB - 2GB.

Task CPU (vCPU)

0.25 vCPU

The valid CPU for 0.5 GB memory is: 0.25 vCPU

Task memory maximum allocation for container memory reservation

0

512 shared of 512 MiB

Task CPU maximum allocation for containers

0

256 shared of 256 CPU units

Container definitions

Add container

Container ...	Image	Hard/Soft ...	CPU Units	GPU	Inference ...	Essential	
contain...	3409540262...	--/--				true	

Figura 33: Límites de memoria y CPU

Esta nueva definición de tarea de tipo Fargate es prácticamente igual a la creada de tipo EC2. Entre las diferencias notorias, encontramos en la configuración de la tarea algunos campos distintos, donde en “Task Role” se selecciona LabRole. Por otra parte, en los límites de CPU se establece a 0.25 vCPU.

Ya creada, podemos lanzar esta nueva tarea con Fargate, similar al procedimiento empleado en el apartado anterior.

Run Task

Select the cluster to run your task definition on and the number of copies of that task to run. To apply container o

Launch type ☒ FARGATE ☐ EC2 ☐ EXTERNAL ⓘ

AWS Fargate is migrating service quotas from the current Amazon ECS task count-based quotas to vCPU-based quotas. To learn more, refer to the [AWS Fargate FAQs](#).

[Switch to capacity provider strategy](#) ⓘ

Operating system family

Task Definition Family

Revision

Platform version ⓘ

Cluster

Number of tasks

Task Group ⓘ

Figura 34: Detalles de la ejecución de la tarea con Fargate

VPC and security groups

VPC and security groups are configurable when your task definition uses the awsvpc network mode.

Cluster VPC* ⓘ

Subnets* ⓘ

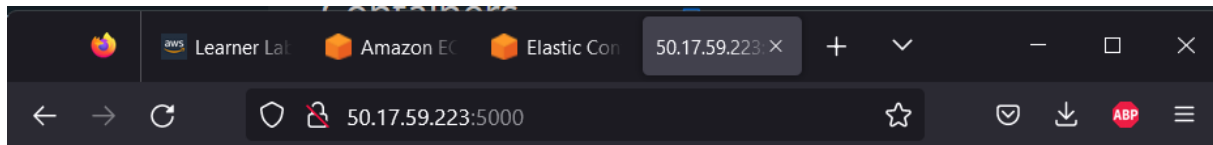
Security groups* ⓘ

Auto-assign public IP ⓘ

▶ Advanced Options

Figura 35: Configuración de red

En la configuración de red, se selecciona el mismo grupo de seguridad que en el apartado 3, pues es justamente lo que se requiere para que funcione. Ejecutada la tarea, se puede comprobar el estado de la misma en la información de las tareas del clúster, en el apartado de Tasks. En la información de la tarea podemos encontrar la IP pública asignada por AWS, la cual se puede acceder para verificar el correcto funcionamiento y despliegue del contenedor mediante Fargate.



Práctica 4 - Computación en la Nube - Apartado 1

Figura 36: Servidor web desplegado mediante Fargate

Comparando la experiencia entre el despliegue del contenedor mediante Fargate y ECS, se hace notoria la mayor rapidez de despliegue mediante Fargate, además de ser más intuitivo. Si bien, al haber hecho el despliegue mediante Fargate en segundo lugar puede influir ya que hemos adquirido unos conocimientos previos, comparando la configuración a realizar, el proceso y la complejidad, Fargate es más simple y eficaz.

Presupuesto y estimación de gasto de los recursos desplegados

Para el clúster en ECS, la instancia desplegada es de tipo t3.micro, cuyo precio es de 0.0104\$/hora. De forma mensual, AWS calcula que el precio mensual de esta instancia sería de 11,27\$ en su modalidad de precio más barata, de forma que esta se ejecute de forma permanente un mes.

Configurar Amazon EC2 Información

Instance name	Memoria	vCPU	Network Perf...	Almacenami...	On-Demand ...	CurrentGeneration
t3.micro	1 GiB	2	Up to 5 Gigabit	EBS only	0.0104	Yes

Estrategia de precios Información

Modelo de precios

☐ EC2 Instance Savings Plans
EC2 Instance Savings Plans proporciona un descuento importante cuando se compromete a un gasto por hora en una familia de instancias de una región determinada. Una estimación podría ser una combinación de EC2 Instance Savings Plans y precios bajo demanda para obtener el mejor valor y rendimiento.

☐ Bajo demanda y reservada
Una combinación de precios reservados y bajo demanda para el mejor valor y desempeño.

☐ Bajo demanda
Pague por la capacidad de cómputo por hora o por segundo, según las instancias que ejecute.

☐ Compute Savings Plans
Los Compute Savings Plans proporcionan un descuento en cualquier instancia EC2 cuando se compromete a realizar un gasto por hora. Una estimación podría ser una combinación de Compute Savings Plans y precios bajo demanda para obtener el mejor rendimiento y valor.

☐ Todas reservadas
Las instancias reservadas proporcionan un descuento importante (de hasta el 75 %) en comparación con los precios de las instancias bajo demanda.

☒ Spot
Las instancias de spot de Amazon EC2 le permiten solicitar capacidad de cómputo adicional de Amazon EC2 con descuentos de hasta el 90 % del precio bajo demanda.

El descuento promedio histórico para t3.micro es 70%.

Costo total de Amazon Elastic Block Storage (EBS) (Mensual): 8,99 USD
Costo de instancias bajo demanda de Amazon EC2 (Mensual): 2,28 USD

Figura 37: Estimación de AWS Pricing Calculator

En caso de emplear un clúster para desplegar el contenedor mediante Fargate, AWS nos deja la siguiente información respecto a su precio:

Precios de Fargate Spot para Amazon ECS

Fargate Spot permite a los clientes ejecutar tareas de Amazon ECS tolerantes a las interrupciones* en la capacidad de reserva con un descuento de hasta el 70 % sobre el precio normal de Fargate. Con Fargate Spot, se paga el precio de Spot que esté en vigor durante el periodo de tiempo en que se ejecuten las tareas de Amazon ECS. AWS Fargate define los precios de Fargate Spot y estos se ajustan gradualmente en función de las tendencias a largo plazo de la oferta y la demanda de capacidad de Fargate Spot. La siguiente tabla muestra el precio actual de Spot por vCPU-hora y GB-hora de cada región.

Región:	EE.UU. Este (Norte de Virginia) +
	Precio
por CPU virtual por hora	0,0138844 USD
por GB por hora	0,00152461 USD

Figura 38: Precios para Spots de Fargate

En este caso, se emplea 0.5GB de memoria, así como 0.25vCPU, por lo que el precio mensual será de 3,05\$

Fuentes

- <https://nodejs.org/en/docs/guides/nodejs-docker-webapp/>
- <https://www.youtube.com/watch?v=YDNSItBN15w>
- <https://aws.amazon.com/es/fargate/pricing/>
- <https://aws.amazon.com/es/ec2/pricing/on-demand/>