



**Universidad**  
Internacional  
de Valencia

# Despliegue de aplicaciones en AWS

(Beanstalk, Automatización con Gitlab y Terraform)

**Titulación:**

Máster Universitario en  
Desarrollo de Aplicaciones y  
Servicios Web

**Curso académico:** 2022-2023

**Alumno/a:** Gómez Olivencia, Rubén

**D.N.I.:** 78910013-A

**Asignatura:** Computación en la nube

## Índice general

<b>1. Introducción</b>	<b>3</b>
<b>2. Base de datos RDS</b>	<b>3</b>
<b>3. Despliegue con AWS Elastic Beanstalk</b>	<b>4</b>
<b>4. Despliegue automatizado desde GitLab</b>	<b>6</b>
<b>5. IaC con Terraform</b>	<b>11</b>
<b>6. Conclusiones</b>	<b>15</b>

# 1. Introducción

A la hora de poner un proyecto en producción era habitual realizar el despliegue de manera manual, teniendo que realizar la instalación y la configuración de cada servicio que se vaya a necesitar.

A día de hoy el despliegue manual ha sufrido muchas mejoras gracias a los sistemas de contenedores, como ejemplo [Docker](#) o sistemas PaaS (plataformas como servicio) como [AWS Elastic Beanstalk](#).

Por otro lado, tenemos la posibilidad de realizar despliegues automatizados gracias a los nuevos sistemas de integración continua y despliegue continuo (CI/CD), muy utilizado en la metodología DevOps. De esta manera, una vez creado el *pipeline* de ejecución, el despliegue siempre se realizará de la misma manera.

Si al punto anterior le añadimos la creación de toda la infraestructura mediante código (IaC, *Infrastructure as code*), pasar de un proveedor de servicios en la nube a otro se convierte en una tarea mucho más sencilla.

En este documento usaremos la aplicación [Flarum](#) como ejemplo para realizar los distintos despliegues explicados previamente.

# 2. Base de datos RDS



Para la aplicación se va a necesitar una base de datos, por lo que va a ser lo primero que se va a desplegar.

AWS permite realizar el despliegue de distintos gestores de bases de datos a través de su servicio **RDS** (*Relational Database Service*). En este caso se va a desplegar un RDS de MySQL 5.X con las siguientes características:

- RDS de tipo db.t3.micro
- Almacenamiento 20Gib SSD de uso general (gp2).
- Cifrado en reposo habilitado.

- Nombre bd = DB\_flarum
- User = admFlarum
- Password = Viu2022Flarum
- Charset = utf8mb4
- Collation = utf8mb4\_unicode\_ci

Es importante añadirle un grupo de seguridad que permita todo el tráfico entrante procedente de la red del VPC de AWS (por defecto 172.31.0.0/16).

Configuración	Clase de instancia	Almacenamiento
ID de instancia de base de datos db-flarum	Clase de instancia db.t3.micro	Cifrado Habilitado
Versión del motor 5.7.40	vCPU 2	Clave de AWS KMS <a href="#">aws/rds</a> 
Nombre de base de datos DB_flarum	RAM 1 GB	Tipo de almacenamiento SSD de uso general (gp2)
License model General Public License	Disponibilidad	Almacenamiento 20 GiB
Grupos de opciones <a href="#">default:mysql-5-7</a>  En sincronización	Nombre de usuario maestro admFlarum	IOPS provisionadas -

Con esto ya tendríamos creada la base de datos.

### 3. Despliegue con AWS Elastic Beanstalk

[Amazon Elastic Beanstalk](#) nos permite crear y desplegar aplicaciones web de manera sencilla sin tener que realizar la tediosa configuración que hay detrás de ello, ya que lo hace de manera automática.

Para realizar el despliegue de Flarum se ha utilizado la siguiente configuración personalizada, a través del editor avanzado al crear el entorno de Beanstalk:


- **Nombre de la aplicación** = flarum
- **Nombre del entorno** = pro-rg-08masw
- Sin balanceador de carga.
- **Entorno:** PHP 8.0

- **Servidor Web** = apache
- **Raíz del documento:** /public
- **Mostrar errores:** On
- **Código fuente** de la aplicación obtenido del siguiente [enlace](#).

Tras crear el entorno, nos aparecerá a modo resumen en el interfaz lo siguiente:

Nombre del entorno ▲	Estado ▼	Nombre de la aplicación ▼	Fecha de creación ▼	Última modificación ▼	URL ▼	Versiones en ejecución ▼	Plataforma ▼	Estado de la plataforma
pro-rg-08masw	OK	flarum	24-01-2023 17:13:02 UTC+0100	24-01-2023 17:24:16 UTC+0100	pro-rg-08masw.us-east-1.elasticbeanstalk.com	flarum-source-1	PHP 8.0 running on 64bit Amazon Linux 2	Supported

Para poder acceder al nuevo entorno a través del navegador web, vemos cómo se nos ha creado una URL propia [pro-rg-08masw.us-east-1.elasticbeanstalk.com](https://pro-rg-08masw.us-east-1.elasticbeanstalk.com), que al acceder nos muestra el panel de configuración de Flarum:



## Install Flarum

Set up your forum by filling out your details below. If you have any trouble, get help on the [Flarum website](#).

Forum Title

Actividad2 RG 08MASW

MySQL Host

db-flarum.c92nioexause.us-east-1.rds.ama

MySQL Database

DB\_flarum

MySQL Username

admFlarum

MySQL Password

.....

Table Prefix

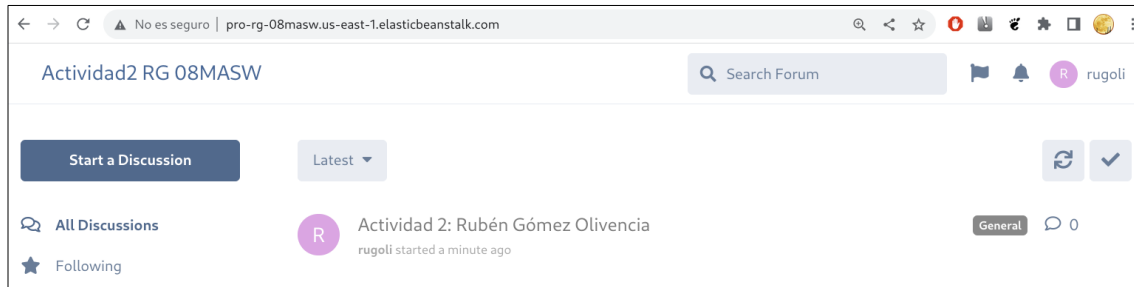
Admin Username

rugoli

Admin Email

rugoli@gmail.com

En este formulario nos aparecen las opciones que debemos rellenar para que se acceda a la base de datos creada previamente. Tras introducir los datos de manera correcta, la aplicación desplegará las tablas que necesita y ya podremos acceder a la aplicación.

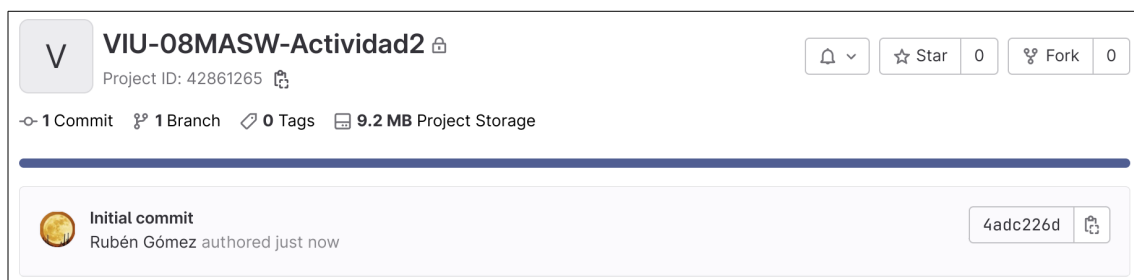


En la imagen superior se puede cómo se ha podido crear una nueva entrada en el foro recién desplegado.

## 4. Despliegue automatizado desde GitLab

Para este apartado vamos a utilizar [GitLab](#) para realizar el despliegue de manera automatizada a través de su sistema de integración continua y despliegue continuo (CI/CD).

Lo primero que tenemos que hacer es crear un nuevo repositorio con el código obtenido de este [enlace](#).



Tras subir el código hay que realizar las siguientes modificaciones:

- Añadir el fichero `.gitlab-ci.yml` : Es el encargado de lanzar el *pipeline*, que son las instrucciones para realizar el despliegue automatizado que ejecuta GitLab al recibir un commit del código en la rama **main**.


## </> Automatización con .gitlab-ci.yml

```
image: "php:8.0"
stages:
  - build
  - deploy
cache:
  key: cache-key
  paths:
    - vendor/
    - node_modules/
before_script:
  - apt-get update
  - apt-get install -y python3 python3-pip wget zip unzip git
  - pip -V
.build:
  script:
    - echo "Building app"
    - apt-get install -y libpng-dev libxml2-dev libzip-dev sudo nodejs npm
    - wget https://composer.github.io/installer.sig -O - -q | tr -d '\n' > installer.sig
    - php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
    - php composer-setup.php --install-dir=/usr/local/bin -filename=composer --version=2.0.14
    - php -r "unlink('composer-setup.php'); unlink('installer.sig');"
    # Extra PHP libraries
    - docker-php-ext-install gd
    - docker-php-ext-install soap
    - docker-php-ext-install zip
    - zip ../myapp.zip -r * .[^.]* >/dev/null
    - mv ../myapp.zip .
artifacts:
  paths:
    - myapp.zip
  expire_in: 1 week
.deploy:
  script:
    - echo "Deploy in server"
    - pwd
    - ls -la
    # AWScli de ElasticBeanstalk
    - pip install awsebcli==3.18
    - export PATH=~/.local/bin:$PATH
    # Credentials AWS (Profile)
    - set -x
    - set -e
    - mkdir ~/.aws
    - touch ~/.aws/config
    - chmod 600 ~/.aws/config
    - echo "[profile eb-cli]" > ~/.aws/config
    - echo "aws_access_key_id=$AWS_ACCESS_KEY_ID" >> ~/.aws/config
    - echo "aws_secret_access_key=$AWS_SECRET_ACCESS_KEY" >> ~/.aws/config
```

```
- echo "aws_session_token=$AWS_SESSION_TOKEN_KEY" >> ~/.aws/config
# Deploy
- cp $CONFIG_FILE_BEAN_STALK_AWS .elasticbeanstalk/config.yml
- eb --version
- eb status --verbose $ENV_BEAN_STALK_AWS
- eb use $ENV_BEAN_STALK_AWS
- eb deploy

1_build_pro:
  extends: .build
  stage: build
  only:
    - main

2_deploy_pro:
  extends: .deploy
  stage: deploy
  variables:
    ENV_BEAN_STALK_AWS: "pro-rg-08masw"
    CONFIG_FILE_BEAN_STALK_AWS: ".elasticbeanstalk/config.pro.yml"
  only:
    - main
```

- Modificar el fichero  `.elasticbeanstalk/config.pro.yml` para que el *pipeline* anterior lo utilice para realizar el despliegue. El despliegue se realizará sobre el mismo Beanstalk creado previamente.

#### Fichero de configuración para Beanstalk

```
global:
  application_name: flarum
  default_ec2_keyname: null
  default_region: us-east-1
  include_git_submodules: true
  instance_profile: null
  platform_name: null
  platform_version: null
  profile: eb-cli
  sc: git
  workspace_type: Application
deploy:
  artifact: ./myapp.zip
```



- Añadir **variables** de entorno dentro del repositorio GitLab (en *Settings* → *CI/CD* → *Variables*). El valor de estas variables será sustituido dentro del pipeline. De esta manera no introducimos credenciales en el código fuente del repositorio.

Type	↑ Key	Value	Options	Environments
Variable	AWS_ACCESS_KEY_ID	*****	Masked, Expanded	All (default)
Variable	AWS_SECRET_ACCESS_KEY	*****	Masked, Expanded	All (default)
Variable	AWS_SESSION_TOKEN_KEY	*****	Masked, Expanded	All (default)

Una vez realizadas las modificaciones, el *pipeline* se ejecutará, y tras esperar a que termine los dos pasos del despliegue, en GitLab veremos un resumen:

passed Pipeline #756535892 triggered 6 minutes ago by Rubén Gómez Delete

### Arreglado error por falta de directorio

🕒 2 jobs for `main` in 3 minutes and 56 seconds

🚩 latest

🔗 68f30d5c

🔗 No related merge requests found.

Pipeline
Needs
**Jobs** 2
Tests 0

Status	Job	Stage	Name	Duration	Coverage
<span>passed</span>	#3654655996 🔗 main → 68f30d5c	deploy	2_deploy_pro	🕒 00:01:36 🕒 2 minutes ago	
<span>passed</span>	#3654655994 🔗 main → 68f30d5c	build	1_build_pro	🕒 00:02:19 🕒 3 minutes ago	

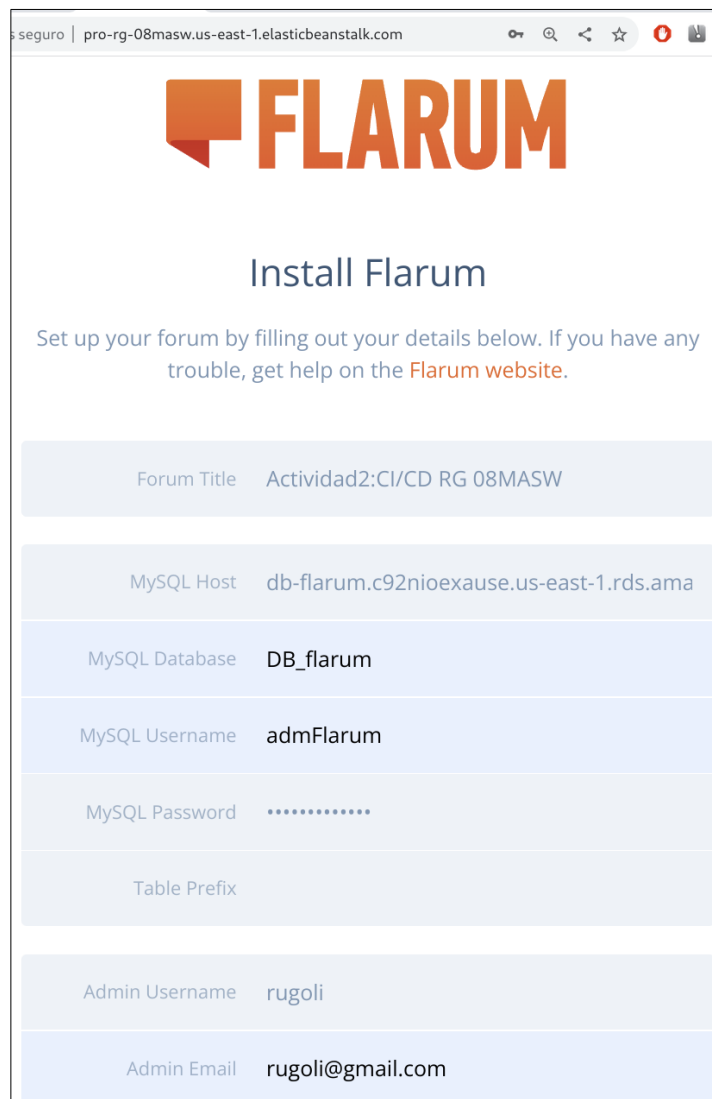
Beanstalk necesita del código fuente de la aplicación para realizar el despliegue. Este se ha subido a través del **artifact** que se ha creado en el *pipeline* (“myapp.zip”), que en AWS se añade dentro de un bucket S3 creado específicamente para Beanstalk.

A continuación se puede ver cómo se han subido varios ficheros tras realizar distintos despliegues:

Nombre	Tipo	Última modificación	Tamaño
 <a href="#">app-f5c8-230124_195103.zip</a>	zip	24 Jan 2023 8:51:04 PM CET	20.5 MB
 <a href="#">app-0a2f-230124_200618.zip</a>	zip	24 Jan 2023 9:06:19 PM CET	20.5 MB
 <a href="#">app-68f3-230124_201313.zip</a>	zip	24 Jan 2023 9:13:15 PM CET	20.5 MB

Dado que es un sistema automatizado, deberíamos también controlar el número de ficheros que se tienen en el bucket S3 para no incurrir en gastos innecesarios.

Tras realizar el despliegue, al ir a la URL proporcionada por Beanstalk, nos aparecerá el asistente de configuración de Flarum, en el que debemos volver a añadir la configuración necesaria. Si utilizamos la misma base de datos del despliegue anterior dará un error, por lo que es importante acordarnos de borrar las tablas existentes.



seguro | pro-rg-08masw.us-east-1.elasticbeanstalk.com

# FLARUM

## Install Flarum

Set up your forum by filling out your details below. If you have any trouble, get help on the [Flarum website](#).

Forum Title

MySQL Host

MySQL Database

MySQL Username

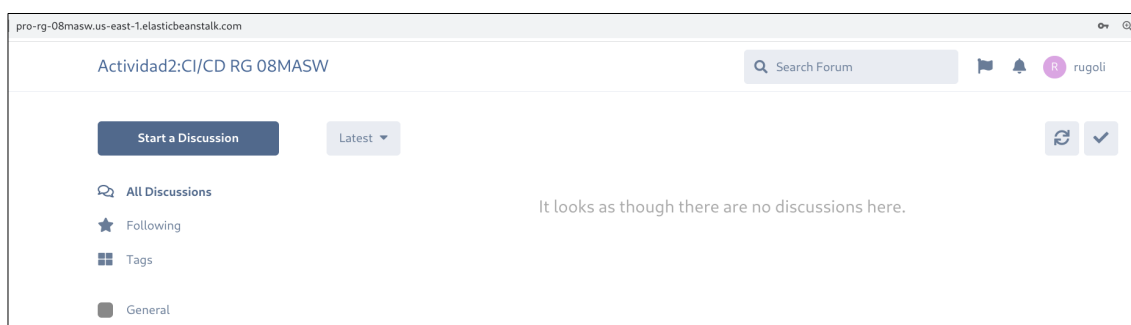
MySQL Password

Table Prefix

Admin Username

Admin Email

Después de la creación de las tablas y el despliegue inicial, ya podremos utilizar la aplicación:



## 5. IaC con Terraform

**Terraform** es una aplicación que nos permite la creación de toda la infraestructura dentro de una nube pública mediante código (IaC, *Infrastructure as code*). De esta manera, podremos realizar despliegues automatizados para distintos entornos (desarrollo, test, producción...).

Para poder utilizar Terraform, una vez **descargado el binario**, tenemos que crear unas variables de entorno (con datos obtenidos de nuestra cuenta AWS) para que la conexión se realice de manera correcta y autenticada.

**</> Seteamos las variables de entorno**

```
ruben@vega:~$ export AWS_ACCESS_KEY_ID=ASIAUSUHSKXGC...
ruben@vega:~$ export AWS_SECRET_ACCESS_KEY=iaC2DxU6Gg...
```

El siguiente paso es decidir qué queremos desplegar. Para esta prueba se va a desplegar una instancia RDS de MySQL similar a la del primer capítulo. Las características son:

- RDS MySQL versión 5.7.X
- **Identificador RDS:** actividad2-terra **Nombre bd** = DB\_terra **User** = admTerra **Password** = Viu2022Terra
- **Tipo de instancia:** T3.micro
- **Tamaño:** 15GB

- **Periodo de retención backups:** 15 días
- **Hora de realización backups:** 03:00 UTC.
- **Ventana de mantenimiento:** 05:00 UTC
- **Cifrado activo**

Para realizar el despliegue tenemos que crear un fichero `main.tf` que contendrá las directivas que Terraform utilizará para conectarse a AWS y realizar el despliegue con las características previas.

#### Fichero main.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.16"
    }
  }
  required_version = ">= 1.2.0"
}

provider "aws" {
  region = "us-east-1"
}

resource "aws_db_instance" "actividad2-terra" {
  identifier            = "actividad2-terra"
  allocated_storage     = 15
  db_name               = "DB_terra"
  engine                = "mysql"
  engine_version        = "5.7"
  instance_class        = "db.t3.micro"
  username              = "admTerra"
  password              = "Viu2022Terra"
  parameter_group_name  = "default.mysql5.7"
  skip_final_snapshot   = true
}
```

```
    backup_retention_period = 15
    backup_window           = "03:00-03:30"
    maintenance_window      = "Sat:05:00-Sat:05:30"
    storage_encrypted       = true
}
```

Para asegurarnos que el fichero está bien creado, podemos ejecutar `>_ terraform validate`, después inicializar el entorno y aplicar los cambios que se van a realizar ya en AWS.

#### Validar la configuración, inicializar y aplicar cambios

```
ruben@vega:~$ terraform validate
Success! The configuration is valid.

ruben@vega:~$ terraform init
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 4.16"...
- Installing hashicorp/aws v4.52.0...
- Installed hashicorp/aws v4.52.0 (signed by HashiCorp)
...
Terraform has been successfully initialized!

ruben@vega:~$ terraform apply
Terraform used the selected providers to generate the following
execution plan. Resource actions are indicated with the following
symbols:
+ create

Terraform will perform the following actions:
# aws_db_instance.actividad2-terra will be created
+ resource "aws_db_instance" "actividad2-terra" {
    + address                        = (known after apply)
    + allocated_storage            = 15
    ...
```

```


}


Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.
  Enter a value: yes

aws_db_instance.actividad2-terra: Creating...
aws_db_instance.actividad2-terra: Still creating... [10s elapsed]
...
aws_db_instance.actividad2-terra: Creation complete
  after 9m15s [id=actividad2-terra]
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

```

Tras esperar unos minutos, el proceso termina, y desde el interfaz web ya podemos ver que nos aparece la instancia, y al ir a mirar su configuración podemos ver los detalles configurados en el fichero  **main.tf**:

Configuración	Clase de instancia	Almacenamiento
ID de instancia de base de datos actividad2-terra	Clase de instancia db.t3.micro	Cifrado Habilitado
Versión del motor 5.7.38	vCPU 2	Clave de AWS KMS <a href="#">aws/rds</a>
Nombre de base de datos DB_terra	RAM 1 GB	Tipo de almacenamiento SSD de uso general (gp2)
License model General Public License	Disponibilidad	Almacenamiento 15 GiB
Grupos de opciones <a href="#">default:mysql-5-7</a>  En sincronización	Nombre de usuario maestro admTerra	IOPS provisionadas -
Nombre de recurso de Amazon (ARN) arn:aws:rds:us-east-1:314890671564:db:actividad2-terra	Contraseña maestra *****	Rendimiento de almacenamiento -
ID de recurso db-MLPFTVQWYMGZM6S3MXLYEVISSI	Autenticación de base de datos de IAM No habilitado	Escalado automático de almacenamiento Deshabilitado
Hora de creación February 02, 2023, 17:51 (UTC+01:00)	Multi-AZ No	
	Zona secundaria	

En la siguiente imagen se puede comprobar cómo la instancia está funcionando:

Identificador de base de datos ▲	Rol ▼	Motor ▼	Región y AZ ▼	Tamaño ▼	Estado ▼	Actions ▼	CPU
actividad2-terra	Instancia	MySQL Community	us-east-1f	db.t3.micro	Disponible	1 Action	4.46%

Si nos interesase destruir la instancia de base de datos recién levantada, podríamos hacerlo con `>_ terraform destroy`, y comenzaría con el proceso de parada y eliminación.

## 6. Conclusiones

La automatización del despliegue de aplicaciones, a través de sistemas CI/CD como el que ofrece GitLab, nos va a permitir que una vez creado el proceso podamos replicarlo tantas veces como sea necesario. De esta manera, podremos utilizarlo para realizar pases a producción, evitando tener que repetir cada vez el proceso y así evitar posibles olvidos o fallos durante el proceso.

Si esto mismo lo utilizamos para realizar despliegues a nivel de infraestructura con Terraform, podremos realizar despliegues enteros para distintos entornos asegurando que no existen diferencias entre ellos.

Tal como se puede ver, cuando vayamos a utilizar una infraestructura *Cloud*, es obligatorio hacer uso de las herramientas que nos van a permitir realizar un trabajo repetitivo de manera más eficiente y segura.