

Práctica: Despliegue Continuo - Blue/Green Deployment en AWS

1. [Punto de partida](#)
2. [Infraestructura](#)
 - [Creación infraestructura mediante Cloudformation](#)
3. [Creación del Job para el despliegue de la aplicación](#)
4. [Modificar Jenkinsfile para añadir despliegue en las ramas release](#)
5. [Ejemplo de funcionamiento completo](#)
 - [Despliegue de release 1.3.0](#)
 - [Despliegue de release 1.4.0](#)
6. [Pruebas continuidad servicio con Jmeter](#)
7. [Referencias documentación AWS](#)

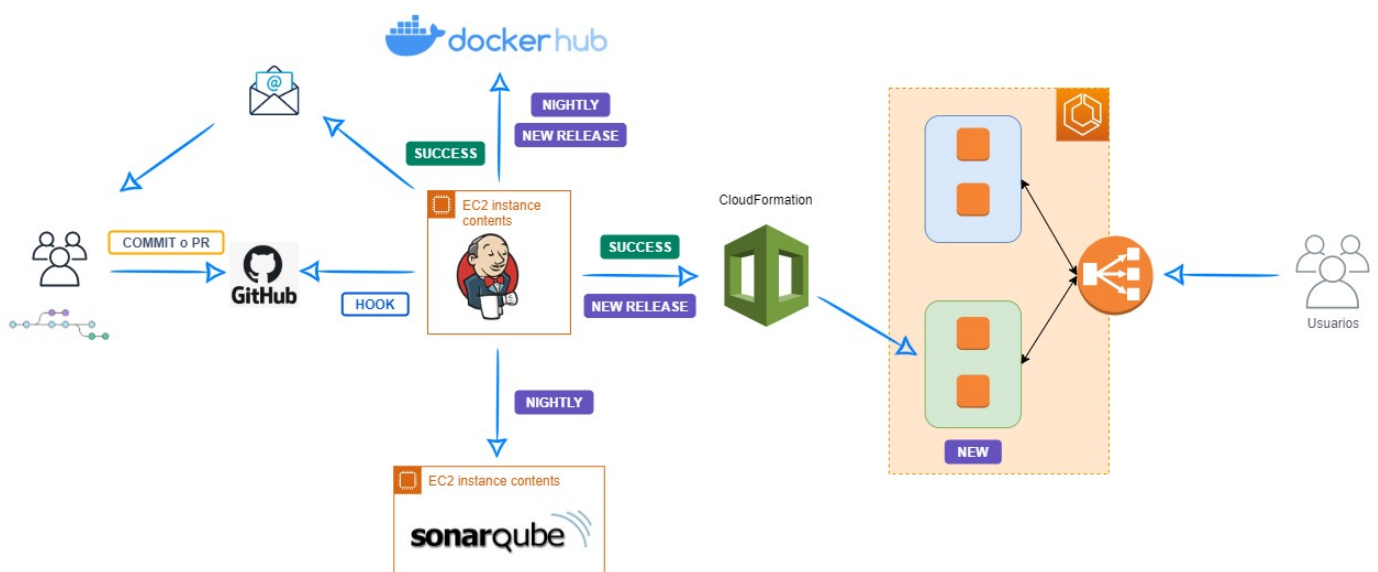
1. Punto de partida

Partimos de la aplicación de la práctica anterior (<https://github.com/mscarceller/practica-ci>), realizando algunos cambios:

- La base de datos MySQL ahora está desplegada mediante una RDS de AWS.
- Con cada commit en ramas de release se ejecuta el pipeline, pero ahora, además, se desplegará la imagen en la infraestructura creada en AWS para ello.

2. Infraestructura

El esquema final quedaría del siguiente modo:



2.1 Creación infraestructura mediante Cloudformation

Los elementos principales de la infraestructura que se ha añadido para publicar la web son:

- Un LoadBalancer que proporciona un endpoint público para acceder a la aplicación.
- Un ECS (Elastic Container Service) en el que se despliegan dos nodos (Services) con la aplicación (Tasks).

Toda la infraestructura se despliega mediante un fichero coludformation.yaml, mediante el cual además, se desplegarán los nuevos servicios. A continuación, se explican los elementos incluidos:

- Cluster ECS (Elastic Container Service): dentro se desplegarán nuestros servicios. Mas adelante elegiremos ejecutar los clústeres de ECS con AWS Fargate. Fargate elimina la necesidad de aprovisionar y administrar servidores:

```
Cluster:
  Type: AWS::ECS::Cluster
  Properties:
    ClusterName: mcapRACTICAD-cluster
```

- LogGroup: para agrupar los logs de los servicios desplegados en ECS y poder monitorizar los eventos de las aplicaciones con CloudWatch agrupados:

```
LogGroup:
  Type: AWS::Logs::LogGroup
  Properties:
    LogGroupName: mcapRACTICAD-log-group
```

- Rol para poder ejecutar Task dentro del cluster ECS:

```
ExecutionRole:
  Type: AWS::IAM::Role
  Properties:
    RoleName: mcapRACTICAD--role
    AssumeRolePolicyDocument:
      Statement:
        - Effect: Allow
          Principal:
            Service: ecs-tasks.amazonaws.com
          Action: sts:AssumeRole
    ManagedPolicyArns:
      - arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy
```

- Grupo de seguridad, para para controlar el tráfico entrante y saliente:

```
SecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: !Sub ${AWS::StackName}-alb
    SecurityGroupIngress:
      - CidrIp: "0.0.0.0/0"
        IpProtocol: "TCP"
        FromPort: 8080
        ToPort: 8080
      - IpProtocol: tcp
        FromPort: 3306
        ToPort: 3306
        CidrIp: 0.0.0.0/0
    VpcId: !Ref VpcId
```

- Rol para los servicios dentro del ECS, este rol (AmazonEC2ContainerServiceRole) permitirá al load balancer registrar y desregistrar instancias de contenedores ECS:

```
ECSServiceRole:
  Type: AWS::IAM::Role
  Properties:
    Path: /
    AssumeRolePolicyDocument:
      Statement:
        - Effect: Allow
          Principal:
            Service: [ecs.amazonaws.com]
          Action: ['sts:AssumeRole']
    ManagedPolicyArns:
      - arn:aws:iam::aws:policy/service-role/AmazonEC2ContainerServiceRole
```

- El LoadBalancer que repartirá el tráfico entre los servicios desplegados en el cluster ECS, en el que se define, además, el puerto en el que estará escuchando el balanceador:

LoadBalancer:

Type: `AWS::ElasticLoadBalancingV2::LoadBalancer`

Properties:

Subnets: `!Ref Subnets`

SecurityGroups:

- `!Ref SecurityGroup`

LoadBalancerListener:

Type: `AWS::ElasticLoadBalancingV2::Listener`

Properties:

LoadBalancerArn: `!Ref LoadBalancer`

Port: `8080`

Protocol: `HTTP`

DefaultActions:

- Type: `forward`

TargetGroupArn: `!Ref TargetGroup`

- ♦ TargetGroup: se utiliza por el LoadBalancer para redireccionar solicitudes a uno o varios destinos registrados. Los principales parámetros que se especifican son:
 - Configuración de direccionamiento: se especifica el protocolo y el puerto que se debe redireccionar. En este caso seleccionamos HTTP y puerto 8080, y el tipo de destino IP (que realmente será un rango de IPs)
 - **deregistration_delay.timeout_seconds**: tiempo que Elastic Load Balancing espera antes de anular el registro de un destino.
 - Se indican los siguientes parámetros de HealthCheck para que se apliquen en el LoadBalancer:
 - **IntervalSeconds**: el intervalo en segundos entre cada solicitud enviada desde el LoadBalancer para comprobar si el servicio está disponible.
 - **HealthCheckPath**: el path al que se enviarán las solicitudes para verificar que el servicio está disponible. En nuestro caso seleccionamos "/", ya que la web del blog responde en esa ruta con un código http 200 si todo está OK.
 - **HealthCheckProtocol**: el protocolo que el LoadBalancer usará para realizar las comprobaciones sobre el servicio. En este caso HTTP.
 - **HealthCheckTimeoutSeconds**: el número de segundos durante los cuales una respuesta negativa desde el servicio significa que no está disponible.
 - **HealthyThresholdCount**: el número consecutivo de respuestas válidas desde un servicio para que se considere que está disponible.

La definición quedaría:

```
TargetGroup:
  Type: AWS::ElasticLoadBalancingV2::TargetGroup
  Properties:
    VpcId: !Ref VpcId
    Port: 8080
    TargetType: ip
    Protocol: HTTP
    Matcher:
      HttpCode: 200-299
    HealthCheckIntervalSeconds: 90
    HealthCheckPath: /
    HealthCheckProtocol: HTTP
    HealthCheckTimeoutSeconds: 60
    HealthyThresholdCount: 2
    TargetGroupAttributes:
      - Key: deregistration_delay.timeout_seconds
        Value: 30
  DependsOn:
    - LoadBalancer
```

- ♦ TaskDefinition y Service, definen los contenedores que se despliegan dentro del cluster y el servicio que expone cada uno de ellos. Se especifica qué imágenes de Docker utilizar, en este caso se le pasa como parámetro, los recursos necesarios (CPU, memoria...) y otras configuraciones relacionadas con el lanzamiento de la definición de tarea a través de un servicio o tarea de Amazon ECS (grupo de logs...).

Cabe destacar el tipo de lanzamiento: Fargate, permite ejecutar aplicaciones containerizadas sin necesidad de aprovisionar más infraestructura.

```
TaskDefinition:
  Type: AWS::ECS::TaskDefinition
  Properties:
    Family: mcapRACTICacd-task
    Cpu: 256
    Memory: 2048
    NetworkMode: awsvpc
    ExecutionRoleArn: !Ref ExecutionRole # Fargate requires task definition to
have execution role ARN to support log driver awslogs
    ContainerDefinitions:
      - Name: mcapRACTICacdV1
        Image: !Ref DockerImage
        PortMappings:
          - ContainerPort: 8080
        LogConfiguration:
          LogDriver: awslogs
        Options:
          awslogs-region: !Ref AWS::Region
          awslogs-group: !Ref LogGroup
          awslogs-stream-prefix: ecs
    RequiresCompatibilities:
      - EC2
      - FARGATE

Service:
  Type: AWS::ECS::Service
  Properties:
    ServiceName: !Ref ServiceName
    Cluster: !Ref Cluster
    TaskDefinition: !Ref TaskDefinition
    LaunchType: FARGATE
    DesiredCount: 2
    LoadBalancers:
      - ContainerName: mcapRACTICacdV1
        ContainerPort: 8080
        TargetGroupArn: !Ref TargetGroup
    NetworkConfiguration:
      AwsVpcConfiguration:
        AssignPublicIp: ENABLED # Private subnet with NAT gateway
        SecurityGroups:
          - !GetAtt SecurityGroup.GroupId
        Subnets: !Ref Subnets
    DependsOn:
      - TaskDefinition
      - ECSServiceRole
```

- ♦ Las salidas que necesitamos para tener información de que todo ha ido bien. La más importante es ServiceUrl, ya que será la url dónde esté disponible nuestro servicio.

Outputs:

ClusterName:

Value: `!Ref Cluster`

ServiceUrl:

Description: URL of the load balancer for the application.

Value: `!Sub http://${LoadBalancer.DNSName}`


DNSName:

Value: `!GetAtt LoadBalancer.DNSName`

NOTA: se podía haber añadido además una ElasticIp + Gateway, pero no era necesario para poder realizar las pruebas, al no tratarse de una aplicación real que necesite mantener siempre la misma URL.

3. Creación del Job para el despliegue de la aplicación

- Creamos un proyecto de tipo "estilo libre":

**Crear un proyecto de estilo libre**

Esta es la característica principal de Jenkins, la de ejecutar el proyecto combinando cualquier tipo de repositorio de software (SCM) con cualquier modo de construcción o ejecución (make, ant, mvn, rake, script ...). Por tanto se podrá tanto compilar y empaquetar software, como ejecutar cualquier proceso que requiera monitorización.

- Indicamos los parámetros que debe recibir:
 - El nombre de la nueva imagen que debe desplegarse.
 - El nombre del nuevo servicio que debe crearse en el cluster.
 - El fichero de cloudformation que debe usarse para el despliegue.

☒ Esta ejecución debe parametrizarse

Parámetro de fichero

Localizacióncloudformation.yaml

Descripción

[Plain text] Visualizar

Parámetro de texto

NameDockerImage

Default Valuemscarceller/mcapracticacd:1.1.0

DescriptionDockerImage

[Plain text] Visualizar

Parámetro de texto

NameServiceName

Default Valuebuild_0

Description

[Plain text] Visualizar

- Definimos las variables de entorno que se necesitan para la autenticación en AWS, que previamente hemos dado de alta en las "credentials" de Jenkins:

Entorno de ejecución

☐ Delete workspace before build starts

☒ Use secret text(s) or file(s)

Bindings

Secret text

Variable:

Credentials: ☒ Specific credentials ☐ Parameter expression

Add

Secret text

Variable:

Credentials: ☒ Specific credentials ☐ Parameter expression

Add

- Finalmente, el comando a ejecutar:

Ejecutar

☒ Ejecutar línea de comandos (shell)

Comando:

Visualizar [la lista de variables de entorno disponibles](#)

Avanzado...

```
aws cloudformation deploy --template-file cloudformation.yaml --region us-east-1 --capabilities CAPABILITY_IAM --stack-name practicacd --parameter-overrides DockerImage=$DockerImage ServiceName=$ServiceName
```

4. Modificar Jenkinsfile para añadir despliegue en las ramas release

Modificamos el pipeline de jenkins, para incluir la llamada a este job si todo ha ido correctamente y si se trata de una rama de release:

```
```shell
def cloudformation_file = new File("${WORKSPACE}/cloudformation.yaml")
build(job: "DeployAWS", parameters: [
 string(name: "DockerImage", value: "$registry:${GIT_BRANCH}"),
 string(name: "ServiceName", value: "build_${GIT_COMMIT}"),
 new FileParameterValue('cloudformation.yaml', cloudformation_file,
'original_cloudformation')
])
```
```

5. Ejemplo de funcionamiento completo

Una vez desplegada la aplicación, inicialmente mediante el template cloudformation desde la consola de Amazon web, vemos que tenemos lo siguiente:

- Cluster de ECS creado y con un servicio con dos tasks (containers):

Clusters

An Amazon ECS cluster is a regional grouping of one or more container instances on which you can run task requests. Each a Amazon EC2 instance type.

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

Create Cluster

Get Started

View

list

card

mcapRACTICacd-cluster >

CloudWatch monitoring

✓ Default Monitoring

FARGATE

1

Services

2

Running tasks

0

Pending tasks

• Detalles del servicio:

Cluster : mcapRACTICAcD-cluster

Get a detailed view of the resources on your cluster.

Cluster ARN

arn:aws:ecs:us-east-1:282044502281:cluster/mcapRACTICAcD-cluster

Status

ACTIVE

Registered container instances

0

Pending tasks count

0 Fargate, 0 EC2

Running tasks count

2 Fargate, 0 EC2

Active service count

1 Fargate, 0 EC2

Draining service count

0 Fargate, 0 EC2

Services

Tasks

ECS Instances

Metrics

Scheduled Tasks

Tags

Capacity Providers

Create

Update

Delete

Actions

Filter In this page

Launch type

ALL

Service type

ALL

| <input type="checkbox"/> | Service Name | Status | Service type |
|--------------------------|--------------|--------|--------------|
| <input type="checkbox"/> | build_0 | ACTIVE | REPLICA |

Detalles de los containers dentro del servicio:

Services

Tasks

ECS Instances

Metrics

Scheduled Tasks

Tags

Capacity Providers

Run new Task

Stop

Stop All

Actions

Desired task status:

Running

Stopped

Filter In this page

Launch type

ALL

| <input type="checkbox"/> | Task | Task definition | Container instan... | Last status | Desired status |
|--------------------------|------------------------------|-----------------------|---------------------|-------------|----------------|
| <input type="checkbox"/> | a3d43c38-6715-4ef7-b87e-8... | mcapRACTICAcD-task:49 | -- | RUNNING | RUNNING |
| <input type="checkbox"/> | eb00f504-d0e6-46db-b665-1... | mcapRACTICAcD-task:49 | -- | RUNNING | RUNNING |

La url publicada por el LoadBalancer:

| | | |
|------------|---|---|
| ServiceUrl | http://pract-LoadB-1NHMH67NVGIDO-1627061796.us-east-1.elb.amazonaws.com | URL of the load balancer for the application. |
|------------|---|---|

Nuestro blog publicado en dicha URL:



5.1 Despliegue de release 1.3.0

Para hacer el despliegue de la nueva aplicación bastará con hacer publicar una rama de release. En este caso vamos a publicar la versión 1.3.0 de nuestro blog, introduciendo un cambio en la web para mostrar la versión que esta publicada:

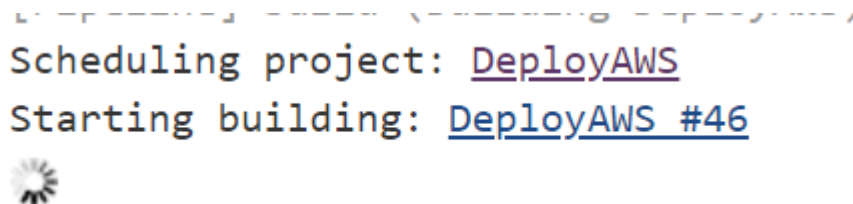
```
....  
<h1>Blog </h1>(v1.3.0)  
....
```

Se disparará de forma automática el job de jenkins sobre la nueva rama, y se sucederán los siguientes pasos:

- Job de jenkins sobre la release:



- Si todo va OK, y la release pasa los test se ejecuta el job de despliegue:



- Que lanza el comando para actualizar la aplicación:

```
Started by upstream project "Miguel Soriano Carceller/practica-cd/release%2F1.3.0" build number 2  
originally caused by:  
  Started by user Miguel Soriano Carceller  
Running as SYSTEM  
Building in workspace /var/lib/jenkins/workspace/DeployAWS  
Copying file to cloudformation.yaml  
[DeployAWS] $ /bin/sh -xe /tmp/jenkins18315861927731214833.sh  
+ aws cloudformation deploy --template-file cloudformation.yaml --region us-east-1 --capabilities CAPABILITY_NAM  
DockerImage=mscarceller/mcapracticacd:1.3.0 ServiceName=build_c9e5371  
  
Waiting for changeset to be created..  
Waiting for stack create/update to complete
```

- Podemos ver en la consola de AWS como llega la solicitud de Update:

Logical ID	Status	Status reason
Service	 UPDATE_IN_PROGRESS	Resource creation Initiated
Service	 UPDATE_IN_PROGRESS	Requested update requires the creation of a new physical resource; hence creating one.

- Se crea un service nuevo:

Services

Tasks

ECS Instances

Metrics

Scheduled Tasks

Tags

Capacity Providers

Create

Update

Delete

Actions

Last updated on June

Filter in this page

Launch typeALL

Service typeALL

	Service Name	Status	Service type	Task Definition	Desired tasks	Running tasks	La
	build_c9e5371	ACTIVE	REPLICA	mcapRACTICAcad-tas...	2	0	FA
	build_0	ACTIVE	REPLICA	mcapRACTICAcad-tas...	2	2	FA

- Se crean las tareas dentro del nuevo nodo:

Services

Tasks

ECS Instances

Metrics

Scheduled Tasks

Tags

Capacity Providers

Run new Task

Stop

Stop All

Actions

Desired task status:

Running

Stopped

Filter in this page

Launch type

ALL

<input type="checkbox"/>	Task	Task definition	Container Instanc...	Last status	Desired status	Started By
<input type="checkbox"/>	1d61685e-1268-494...	mcpracticacd-task:50	--	RUNNING	RUNNING	ecs-svc/640
<input type="checkbox"/>	415b9ec7-786e-4d2...	mcpracticacd-task:50	--	PENDING	RUNNING	ecs-svc/640
<input type="checkbox"/>	a3d43c38-6715-4ef...	mcpracticacd-task:49	--	RUNNING	RUNNING	ecs-svc/382
<input type="checkbox"/>	eb00f504-d0e6-46d...	mcpracticacd-task:49	--	RUNNING	RUNNING	ecs-svc/382

- Una vez que las dos nuevas están operativas se eliminan las antiguas:

Services | **Tasks** | ECS Instances | Metrics | Scheduled Tasks | Tags | Capacity Providers

Run new Task | Stop | Stop All | Actions ▾

Desired task status: **Running** Stopped

Filter In this page Launch type ALL ▾

<input type="checkbox"/>	Task	Task definition	Container instanc...	Last status
<input type="checkbox"/>	1d61685e-1268-49...	mcapRACTICacd-task:50	--	RUNNING
<input type="checkbox"/>	415b9ec7-786e-4d...	mcapRACTICacd-task:50	--	RUNNING

- Y se eliminan tambien el nodo antiguo:

Services | **Tasks** | ECS Instances | Metrics | Scheduled Tasks | Tags | Capacity Providers

Create | Update | Delete | Actions ▾

Filter In this page Launch type ALL ▾ Service type ALL ▾

<input type="checkbox"/>	Service Name	Status	Service type	Task Definition
<input type="checkbox"/>	build_c9e5371	ACTIVE	REPLICA	mcapRACTICacd-tas...

- La actualización termina correctamente:

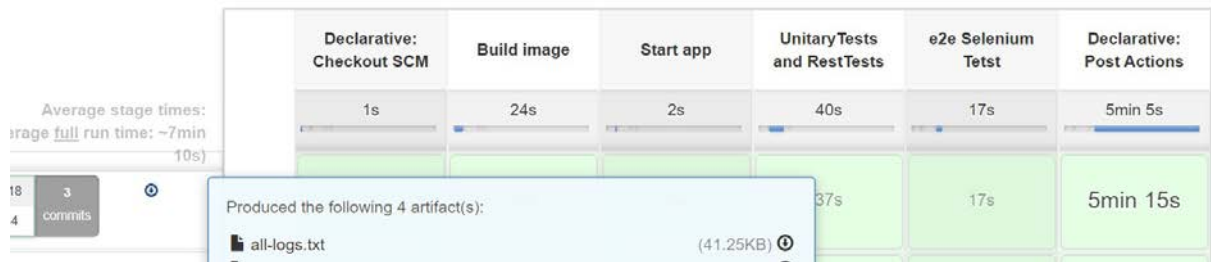
Logical ID	Status	Status reason
practicacd	UPDATE_COMPLETE	-

- En el job de despliegue se recibe el Success:

```
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/DeployAWS
Copying file to cloudformation.yaml
[DeployAWS] $ /bin/sh -xe /tmp/jenkins18315861927731214833.sh
+ aws cloudformation deploy --template-file cloudformation.yaml --region us-east-1 --c
DockerImage=mscарceller/mcapRACTICacd:1.3.0 ServiceName=build_c9e5371

Waiting for changeset to be created..
Waiting for stack create/update to complete
Successfully created/updated stack - practicacd
Finished: SUCCESS
```

- Y el job que se disparó al crear la release finaliza correctamente:



- La web ya dispone de la actualización introducida:

← → ↻ 🏠 ⓘ No es seguro | pract-loadb-1nhmh67nvgido-1627061796.us-east-1.elb.amazonaws.com:8080

Blog

(v1.3.0)

[Nuevo post](#)

5.2 Despliegue de release 1.4.0

De igual forma que hemos hecho con la release 1.3.0 hacemos para publicar la 1.4.0:

```
....
<h1>Blog </h1>(v1.4.0)
....
```

Se podrían introducir más cambios, pero este es suficiente para comprobar que el proceso funciona.

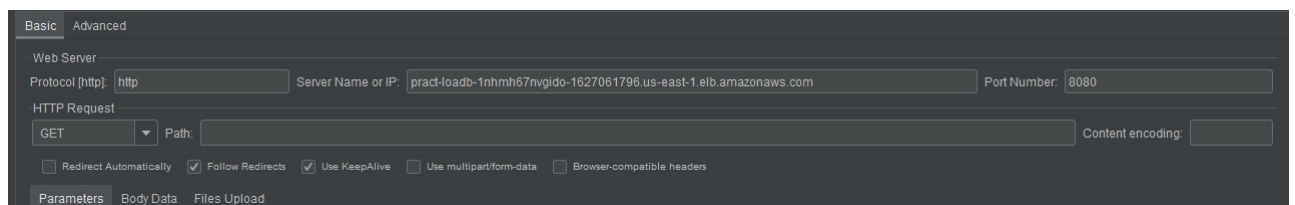
6. Pruebas continuidad servicio con Jmeter

Durante el despliegue de la versión 1.4 he hecho una pequeña prueba para comprobar si hay perdida de servicio. Para ello he usado jmeter, para hacer peticiones get a la url del blog.

- ♦ Configuración de JMeter:



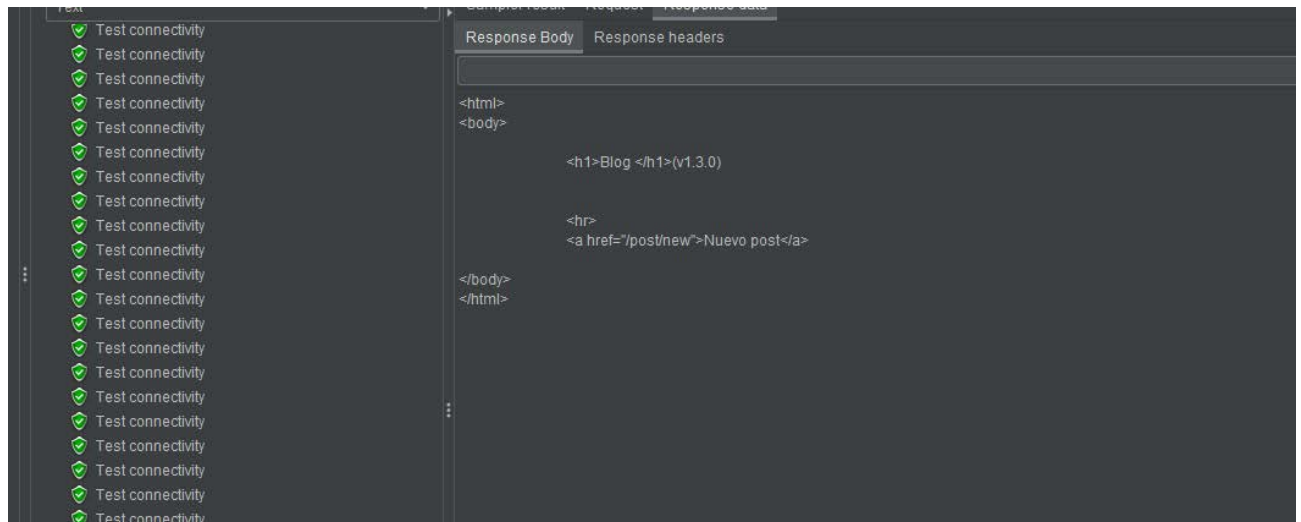
The screenshot shows the 'Thread Properties' window in JMeter. It contains three input fields: 'Number of Threads (users):' with the value '20', 'Ramp-up period (seconds):' with the value '5', and 'Loop Count:' with a checked checkbox for 'Infinite' and an empty text field.



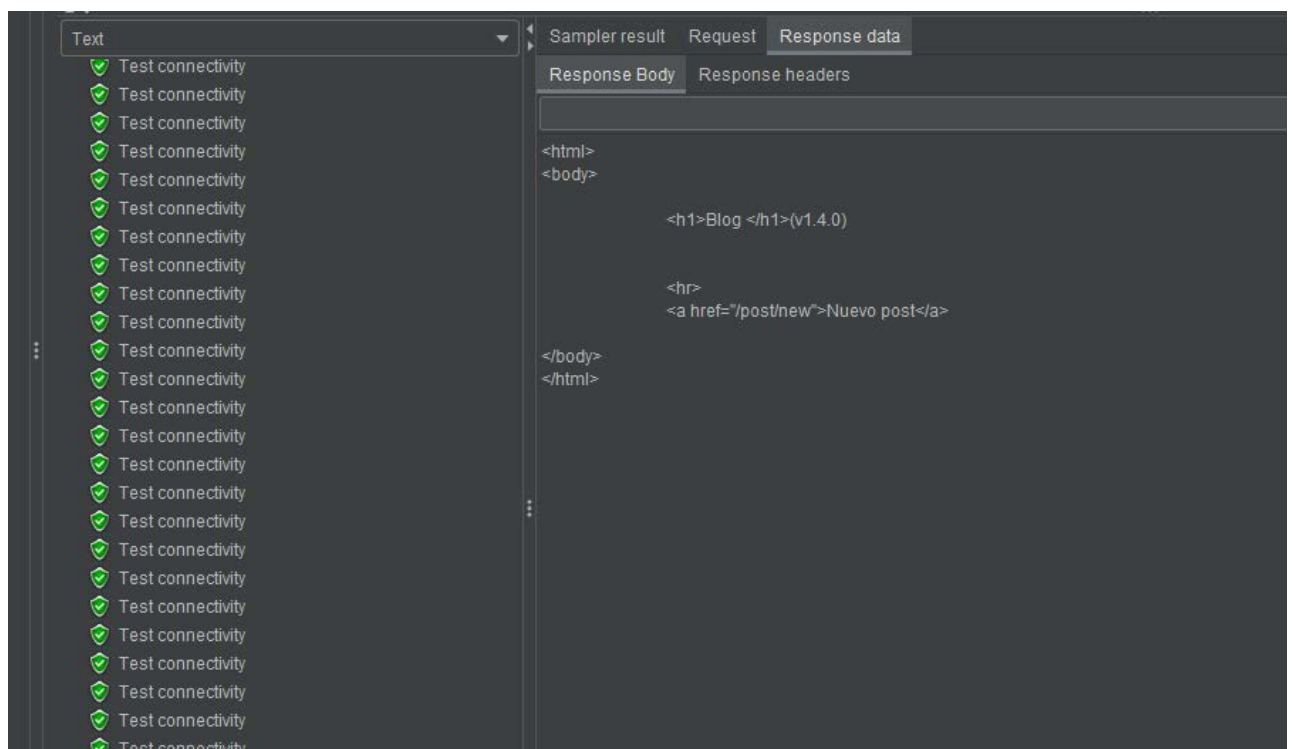
The screenshot shows the 'Basic' tab of the JMeter configuration window. It includes the following fields and options:

- Web Server:**
 - Protocol (http): http
 - Server Name or IP: pract-loadb-1nhmh67nvgido-1627061796.us-east-1.elb.amazonaws.com
 - Port Number: 8080
- HTTP Request:**
 - Method: GET (selected from a dropdown)
 - Path: (empty text field)
 - Content encoding: (empty text field)
- Options:**
 - ☐ Redirect Automatically
 - ☒ Follow Redirects
 - ☒ Use KeepAlive
 - ☐ Use multipart/form-data
 - ☐ Browser-compatible headers
- Tabs:** Parameters, Body Data, Files Upload

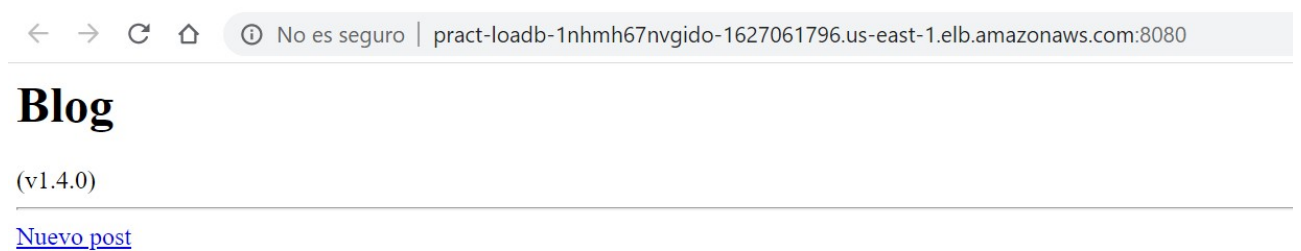
- Vemos las primeras peticiones devuelven el blog con la versión 1.3.0:



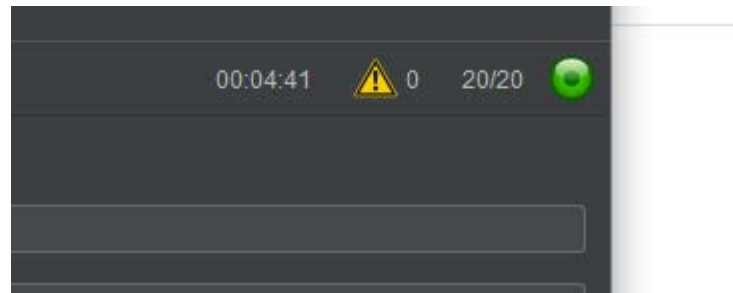
- Vemos las últimas peticiones devuelven el blog con la versión 1.4.0:



- Vemos como se ha actualizado la web:



- ♦ No hay errores en Jmeter:



No vemos ninguna petición rechazada.

7. Referencias documentación AWS

- Amazon ECS: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs_managed_policies.html#AmazonEC2ContainerServiceRole
- EC2 security groups: <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-security-groups.html>
- ECS Task Execution IAM Role: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task_execution_IAM_role.html
- Log Groups: <https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/Working-with-log-groups-and-streams.html>
- Target Groups: <https://docs.aws.amazon.com/elasticloadbalancing/latest/application/load-balancer-target-groups.html>
- Routing Configuration: <https://docs.aws.amazon.com/elasticloadbalancing/latest/application/load-balancer-target-groups.html#target-group-routing-configuration>
- Health Checks Target Groups: <https://docs.aws.amazon.com/elasticloadbalancing/latest/application/target-group-health-checks.html>
- AWS::ECS::TaskDefinition: <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-ecs-taskdefinition.html#cfn-ecs-taskdefinition-requirescompatibilities>
- AWS::ECS::Service: <https://docs.aws.amazon.com/es-es/AWSCloudFormation/latest/UserGuide/aws-resource-ecs-service.html>
- Amazon ECS Launch Types https://docs.aws.amazon.com/AmazonECS/latest/developerguide/launch_types.html
- Amazon ECS on AWS Fargate: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/AWS_Fargate.html