Modularización con Virtualización

Aplicación Web

Amalia Inés Alfonso Campuzano - amalia.alfonso@mail.escuelaing.edu.co

Profesor Luis Daniel Benavides Navarro

Escuela Colombiana de Ingeniería Julio Garavito

# Introducción

Este documento muestra cómo realizar una aplicación web Cliente servidor, que permite responder a múltiples peticiones concurrentemente sin dañar la modularización utilizando virtualización. La virtualización permite ejecutar varios sistemas operativos dentro de un mismo servidor físico.

Para realizar las peticiones se utilizó Amazon, que es una plataforma de computación en la nube que ofrece servicios de infraestructura para servicios web.

# ARQUITECTURA DE SERVIDOR

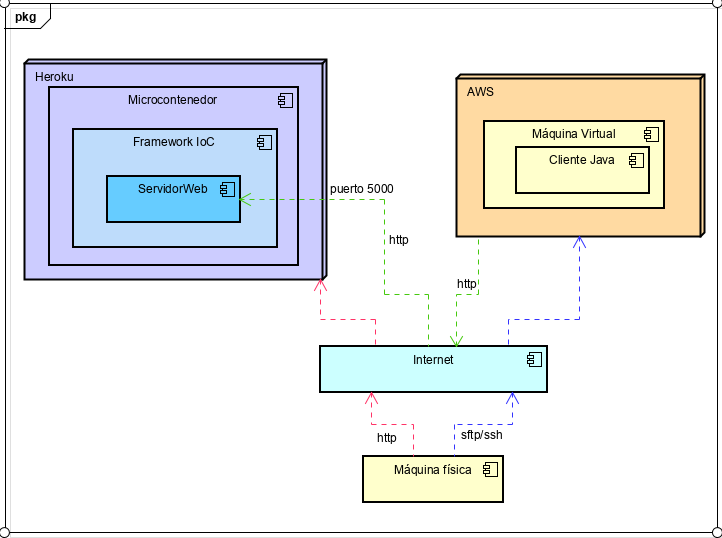
## 

Para permitir comunicación entre el cliente y el servidor, se implementó un socket para cada uno, los cuales establecen los puntos finales del enlace de comunicación y están vinculados a un puerto. El socket del servidor está vinculado al puerto de heroku (5000), y por este espera las solicitudes del cliente, ya que allí está desplegada la aplicación.

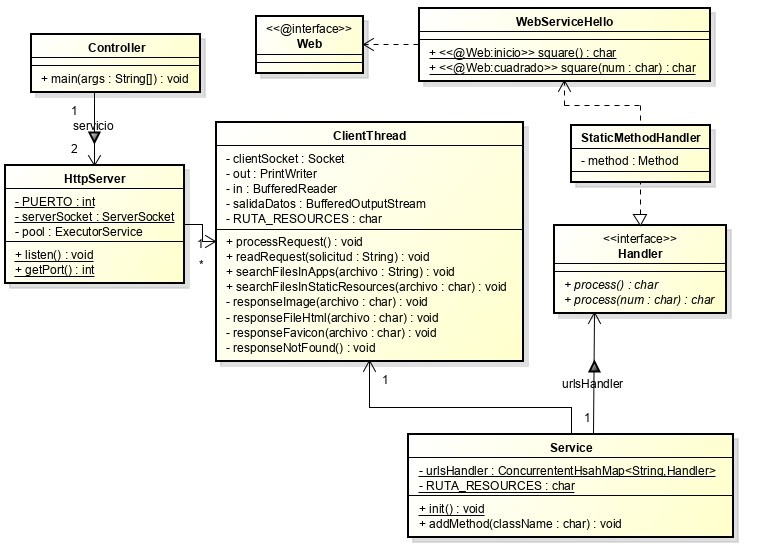
En la aplicación web se procesarán las peticiones del cliente para dar una respuesta. Dependiendo el tipo de petición, buscará en apps o en los recursos estáticos. Una vez se tiene respuesta, el servidor responde al cliente.

# ARQUITECTURA DE DESPLIEGUE

En el diagrama de despliegue se muestra que la aplicación está desplegada en heroku usando el puerto 5000. En la aplicación web se encuentra el framework, el servidor y la aplicación que genera las respuestas para el cliente. El servidor recibe múltiples peticiones concurrentemente y las escucha por medio del puerto de heroku, las procesa en la aplicación y retorna la respuesta a cada petición. Puede recibir solicitudes de archivos dinámicos (apps) o estáticos (archivo html o una imagen). Esta respuesta es dada dependiendo el requestURI, ya que se puede especificar que se busque en apps o en recursos estáticos.



# Diagrama de clases

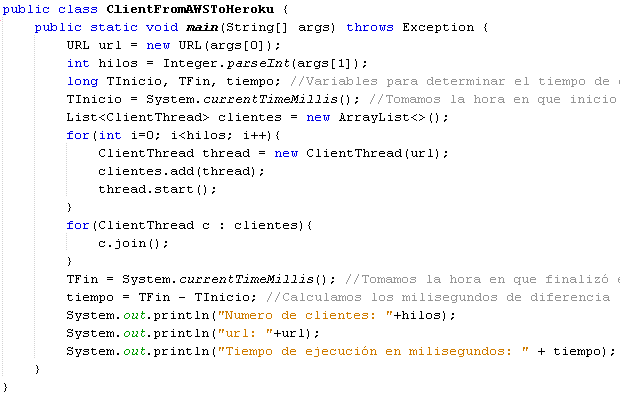


En este diagrama podemos ver la estructura del sistema, en el cual se muestran las clases con atributos y métodos. La clase en al cual se presta el servicio de http es httpServer la cual permite múltiples conexiones con clientes. ClientThread procesa y resuelve las peticiones de un cliente.

La aplicación tiene como clase principal Controller, la cual inicializa el servidor.

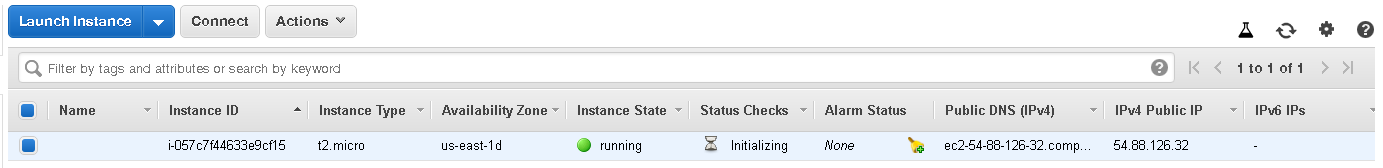
# desarrollo

Primero se realizó un cliente que se conecta a una url e imprime el contenido de la url. Este cliente envía múltiples peticiones concurrentes a la página web haciendo uso de hilos.



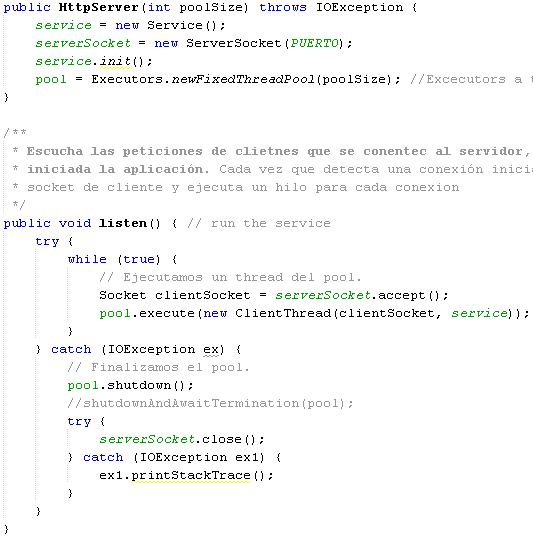
Al final, se muestra cuantas peticiones realizó, la url a la que se conectó y el tiempo que se demoró el servidor en dar respuesta a todas las peticiones.

Después se creó una máquina virtual de Linux en AWS.

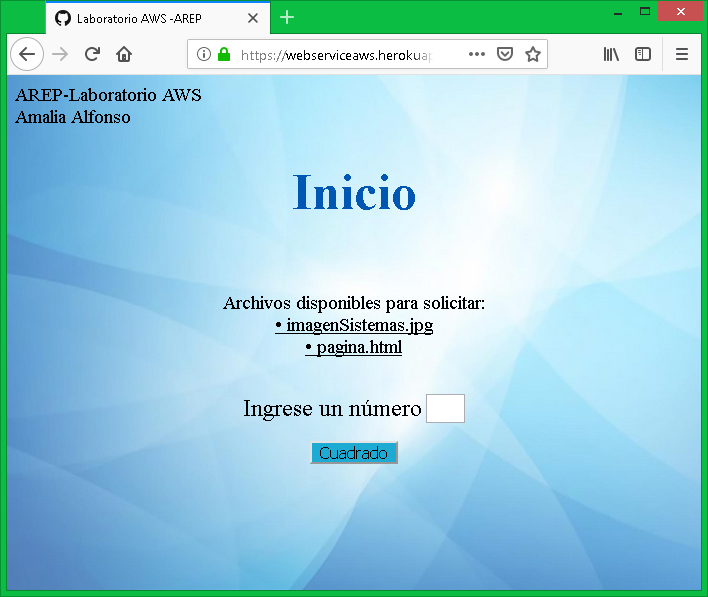


Para subir el cliente antes creado a la máquina virtual se usó el protocolo sftp que funciona sobre ssh para conectarse de forma segura a la maquina remota.

Se tomó la aplicación del repositorio <https://github.com/acai-bjca/Proyecto1-AREP.git> , y se modificó para que recibiera múltiples peticiones concurrentemente usando un pool de hilos, en el que cada hilo procesa un cliente.



Por medio del browser se aseguró que funcionara la aplicación corriendo en heroku.



Para ejecutar el cliente se conecta a la máquina AWS por medio de ssh. Ese cliente se conecta a la aplicación de heroku.

Para conectarse con el cliente, se ejecuta el jar y se usa http indicando la url y el número de peticiones deseadas:

A continuación se realizaron unos experimentos para ver el desempeño de la aplicación web desplegada en heroku, cambiando el número de peticiones enviadas por el cliente y solicitando archivos dinámicos y estáticos.

Dinámico - Calcula el cuadrado de un número:

<https://webserviceaws.herokuapp.com/apps/cuadrado?numero=5>

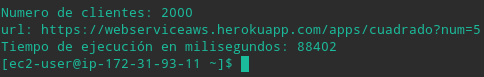
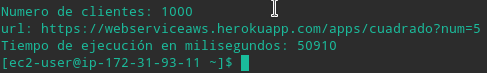
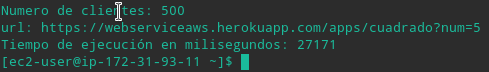
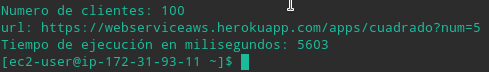
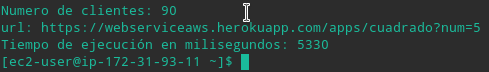
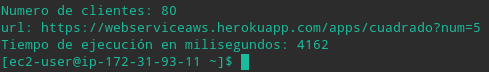
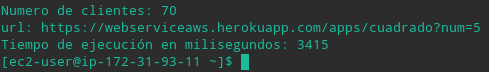
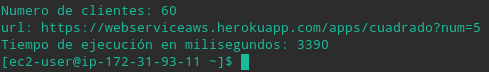
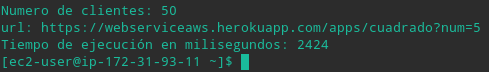
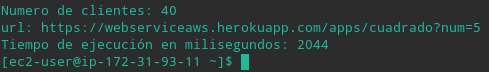
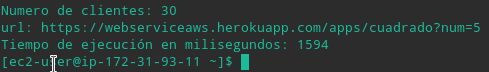
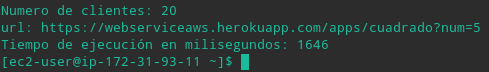
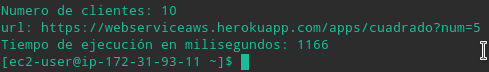
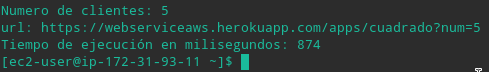
Estáticos:

• Imagen: <https://webserviceaws.herokuapp.com/imagenSistemas.jpg>

• Archivo html: <https://webserviceaws.herokuapp.com/pagina.html>

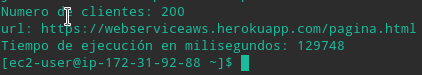
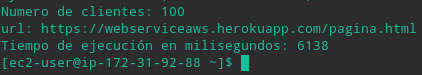
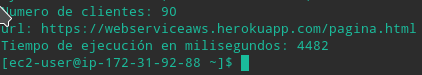
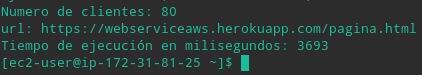
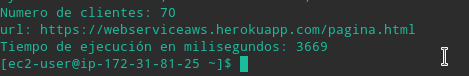
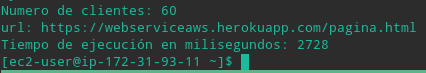
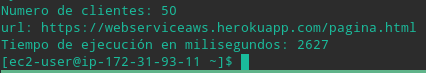
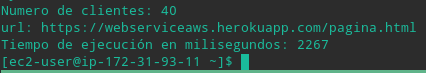
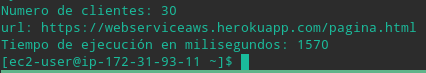
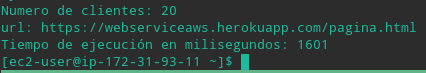
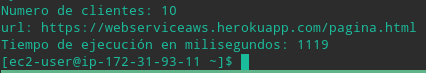
# Pruebas

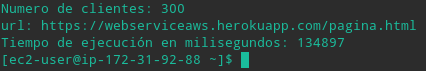
Dinámico: html

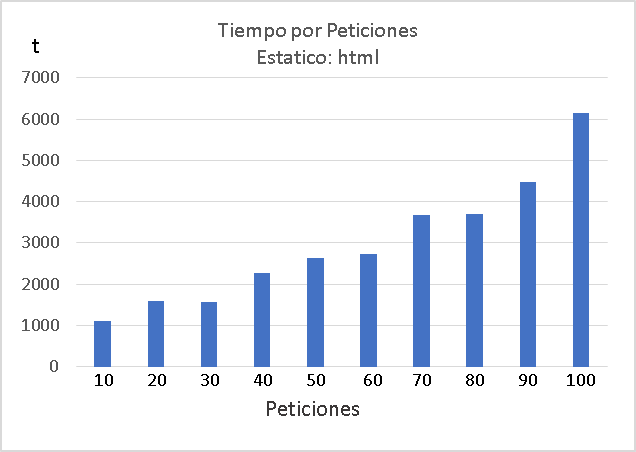


E:\Amalia\Downloads\AWS\pantallazos\apps-5000.png

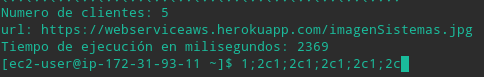
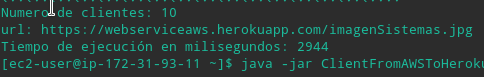
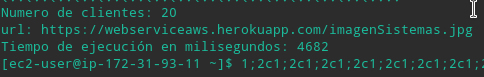
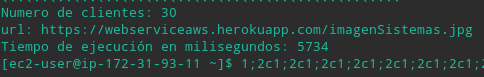
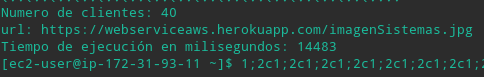
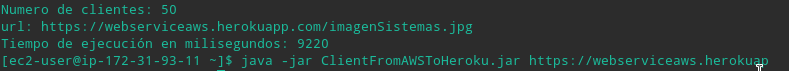
Estático: html







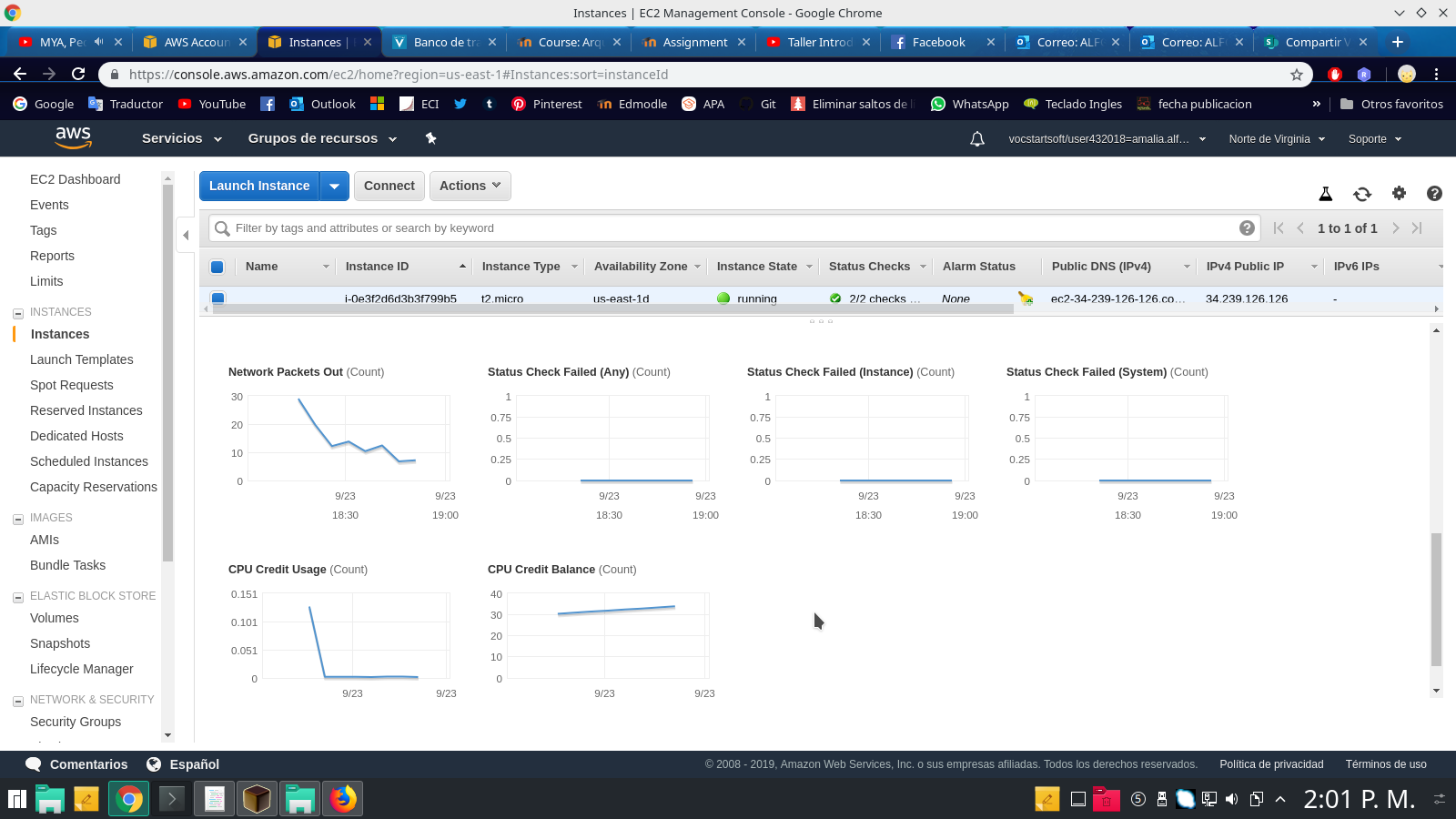
Estático: imagen

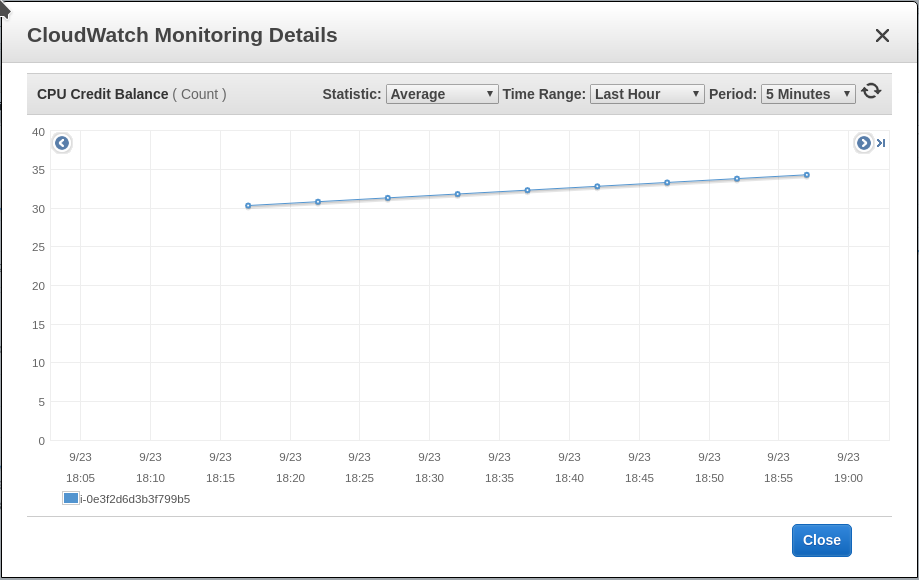
     

E:\Amalia\Downloads\AWS\pantallazos\img-100.png E:\Amalia\Downloads\AWS\pantallazos\img500.png

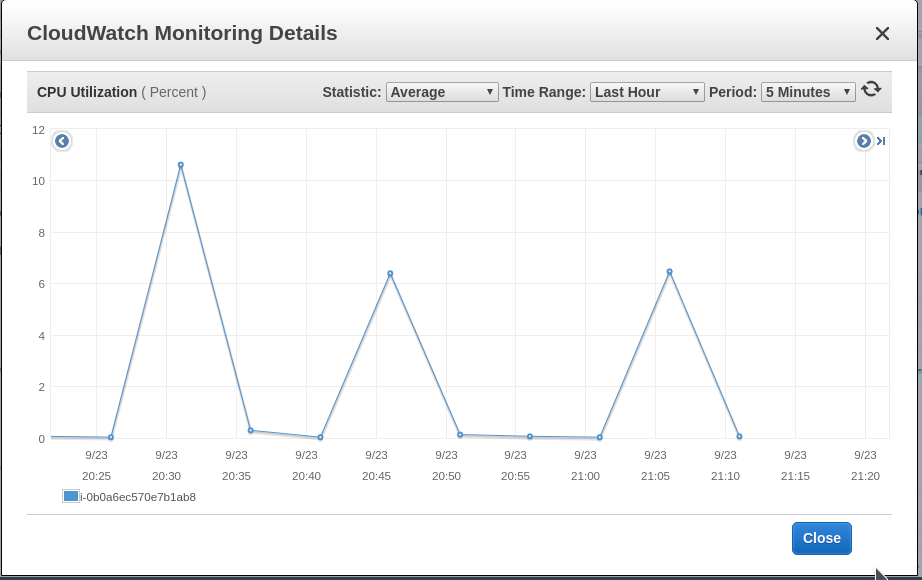
Las imágenes son más pesadas, por eso se tomaba más tiempo en responder el servidor, y por eso con menos peticiones que un archivo html comienza a caerse el servidor. Par poder realizar el experimento, se tenía que esperar a que se recuperara en cada cambio de peticiones.

Al realizar muchas peticiones el servidor no responde porque el cliente agota los créditos usados. Éstos créditos son dados al registrarse en AWS educate con una cuenta de la Escuela. Si la máquina virtual de AWS tuviese más capacidad, podrían enviarse más peticiones. Para solucionar esto se debe crear una instancia con más capacidad, pero los créditos dados no lo permiten, Cuando se mandan muchas peticiones la máquina de AWS no puede seguir escalando y el servidor deja de responder, por esta razón se deben interrumpir las peticiones, y esperar para que se recupere el servidor y se liberen créditos en AWS.





La CPU alcanzo los picos más altos cuando se usa en su totalidad al recibir muchas peticiones.



# Conclusiones

AWS es una excelente plataforma para realizar servicios web. Permite crear máquinas virtuales para probar el correcto funcionamiento de aplicaciones web y monitorear su funcionamiento.

Al tener modularización con virtualización es más fácil controlar y disminuir posibles falas al tener múltiples peticiones.

El buen uso de hilos permiten la disponibilidad de servidores.

# REFERENCIAS

• «https://aws.amazon.com/es/education/awseducate/,» AWS. [En línea].

• L. D. Benavides, « <https://www.youtube.com/watch?v=xmZFAAnvKIQ&t=2212s>» 2016.

• J. P. Maestras, «Programación Concurrente con Java,» 24 Enero 2017. [En línea]. Available: <http://grasia.fdi.ucm.es/jpavon/docencia/dso/programacionconcurrentejava.pdf>.