Modularización por virtualización AWS

Diego Alejandro Corredor Tolosa, Escuela Colombiana de Ingeniería

***Resumen*—El presente documento tiene como objetivo definir la arquitectura usada para el taller de introducción al servicio de Amazon Web Services, uniéndolo al proyecto de primer tercio. En este proyecto se cero un servidor web capaz de recibir solicitudes concurrentes y mostrar imágenes tipo "png" o archivos "html" de acuerdo a la solicitud hecha. También se creó un cliente concurrente capaz de leer la información de una URL en específica, el cliente enviara múltiples solicitudes paralelas a nuestro servidor concurrente web.**

***Abstract*—The purpose of this document is to define the architecture used for the Amazon Web Services introduction workshop, joining it to the first third Project. In that Project, we built a web server capable of receiving concurrent requests and displaying "png" images and "html" files according to the request made. Also, we built a concurrent client capable of Reading the information of a specific URL, the client will send multiple parallel requests to our concurrent web.**

I. Introducción

Este documento sirve de guía para ayudar a entender la implementación realizada, con un diseño de componentes que permite ver la relación entre estos, su funcionamiento y su comportamiento en el tiempo de ejecución todo esto tomando como base los requisitos planteados al momento de iniciar el proyecto.

II. DISEÑO

Las capas de diseño se implementaron con base a los requerimientos y/o requisitos y necesidades sobre la marcha de la implementación. Siempre buscando generar aspectos de extensibilidad y estabilidad. Dicho lo anterior las capas o módulos que componen la infraestructura de este proyecto son:

A. Servidor:

Esta capa construye la clase principal del proyecto que en este caso es "Server" desde aquí se procesa la solicitud del cliente de acuerdo a lo que consulte. También se inicializa el hasmap con los métodos que tienen la anotación @web para así ser accedidos cuando el cliente lo solicite por medio de la URL. También se encarga de enviarle al cliente todo lo necesario para poder visualizar las paginas e imágenes png. Server implementa la interfaz runnable de esta forma poder atender varias solicitudes en paralelo creando un hilo de ejecución por cada solicitud recibida.

B. framework:

En esta capa se encuentran todo lo que tiene que ver con la configuración de los pojos, la interfaz y las implementaciones encargas de invocar los métodos de estos, como también la interfaz para la anotación @web

C. app:

En esta capa se encuentran todo lo que tiene que ver con la creación de los pojos, aquí se utiliza la anotación @web para crear los métodos que se desea que se utilicen en la aplicación.

D. Resources:

En esta capa se encuentran todos los archivos que se quieren mostrar en la página, están todas las paginas html e imágenes .jpg.

E. Sockets:

Esta capa se añadió para desacoplarla de el server ya que anteriormente este se encargaba de manejar los sockets, de esta forma al realizar algún cambio en alguna de las capas no influya con la otra.

III. ESTRUCTURA DEL PROYECTO

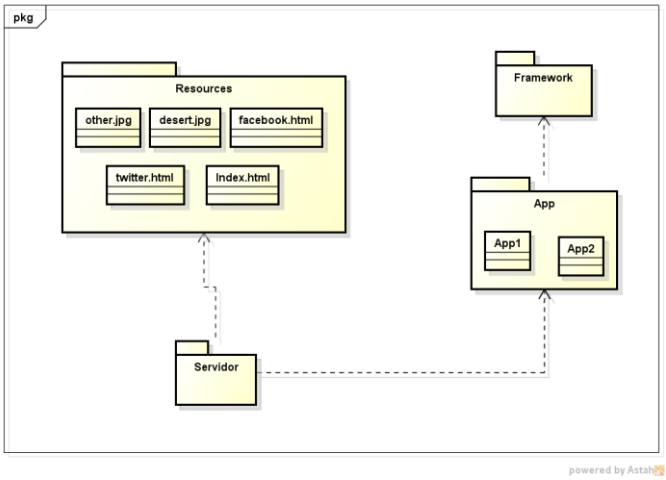

Figura 1. Estructura interna del proyecto

Esta compuesto por los diferentes recursos, los cuales se podrán acceder desde la página principal que es ("/index.html"). De aquí se generan todos los vínculos a las demás dependencias del servidor, Incluyendo tanto las páginas HTML, como las imágenes usadas para visualizarlas. Para acceder a la visualización de los diferentes métodos con notación web (Pojos) se debe de colocar la extensión del método a llamar y si este necesita de parámetros a

continuación se deberá colocar dos puntos y colocar los parámetros.

IV. ESTRUCTURA DEL CLIENTE CONCURRENTE

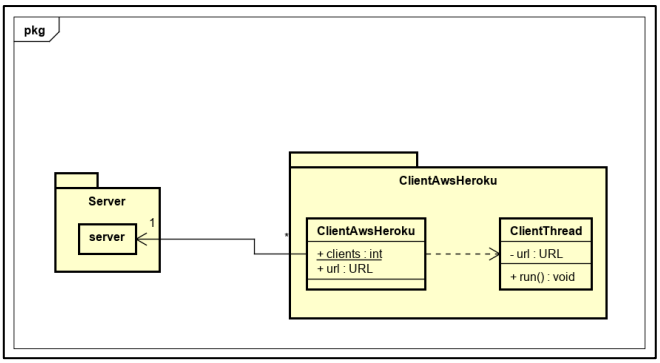

Figura 2. Estructura interna del cliente

El cliente este compuesto por dos clases, ClientAwsHeroku y ClientThread. ClientAwsHeroku es la encarga de administrar la cantidad de solicitudes que se realizaran en paralelo como también de a que URL se realizaran dichas solicitudes. ClientThread por su parte es el encargado de leer los datos de una URL en específico (su código fuente) y mostrarlos en pantalla

V. ARQUITECTURA EN AMAZON WEB SERVICES

El cliente se desplegará en una maquina virtual creada en Amazon Web Services y de allí realizará las respectivas solicitudes a nuestro servidor desplegado en heroku.

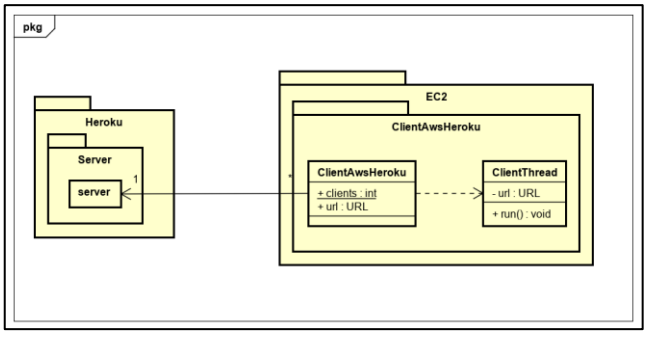

Figura 3. Cliente desplegado en AWS

Como se puede observar en la figura 3 nuestro cliente realizará las solicitudes alojado en una máquina virtual de Amazon Web Services, hará múltiples solicitudes en paralelo al aplicativo desplegado en heroku el cual estará en la capacidad de recibir cada una de estas.

VI. EXPERIMENTOS

Para comprobar el funcionamiento de nuestro aplicativo se realizaron las siguientes pruebas funcionales:

Cliente concurrente vs Servidor no concurrente

Primero se verifico, el comportamiento del servidor al recibir múltiples solicitudes de forma paralela. A lo cual como es de esperarse logra atender solo una de las solicitudes y las demás resultan en un error 503 de disponibilidad.

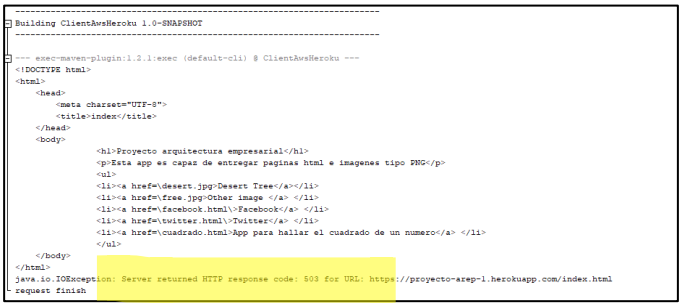

Figura 4. 503 http response.

Para esta prueba el cliente está localizado en mi maquina local.

Cliente concurrente vs Servidor concurrente

Para esta prueba se sitúa el cliente en una maquina Linux de AWS, y realizamos la ejecución para verificar que al ser concurrente nuestro servidor puede atender las múltiples solicitudes del cliente.

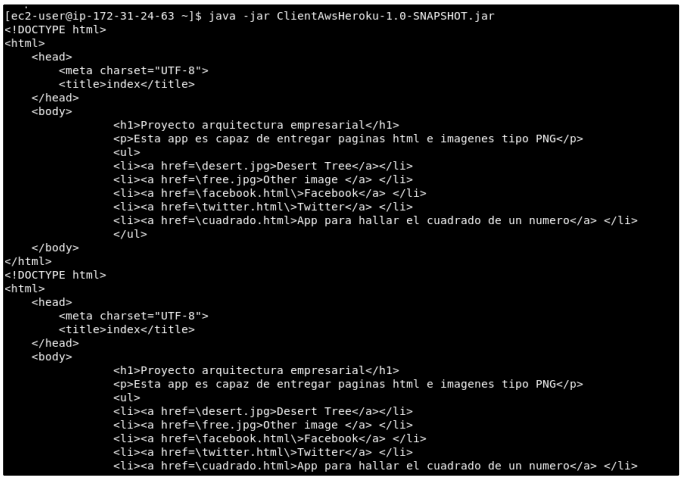

Figura 5. Cliente desplegado en AWS

Como se puede observar en la figura 5, todas las solicitudes son atendidas por el servidor hosteado en heroku.

Diego Alejandro Corredor Tolosa nació en Zipaquirá, Cundinamarca, Colombia en 1997. Graduado de el Colegio Gimnasio Campestre Santa Sofia en el 2014, inicio sus estudios de ingeniería de sistemas en la escuela colombiana de ingeniería Julio Garavito