

**Instituto Politecnico Nacional**

**ESCUELA SUPERIOR DE CÓMPUTO**

Aplicaciones Para Comunicaciones De Red

PRÁCTICA 3: Servidor Web

Profesor: Axel Moreno Cervantes

Alumnos: Regis Valero Jorge Jesus

Medina Martinez Miguel Angel.

GRUPO: 3CM7

**Introducción**

**Servidor Web**

Los servidores web (también conocidos como servidores HTTP) son un tipo de servidores utilizados para la distribución (entrega) de contenido web en redes internas o en Internet (“servidor” hace referencia al verbo “servir”). Como parte de una red de ordenadores, un servidor web transfiere documentos a los llamados clientes (del inglés clients), por ejemplo, una página web a un explorador.

Los servidores web sirven para almacenar contenidos de Internet y facilitar su disponibilidad de forma constante y segura. Cuando visitas una página web desde tu navegador, es en realidad un servidor web el que envía los componentes individuales de dicha página directamente a tu ordenador. Esto quiere decir que para que una página web sea accesible en cualquier momento, el servidor web debe estar permanentemente online.

**Historia:**

El desarrollo de los servidores web está estrechamente vinculado al físico e informático británico Tim Berners-Lee, quien en 1989 sugirió que el intercambio de información en el CERN (Organización Europea para la Investigación Nuclear) debería realizarse a través de un sistema de hipertexto más fácil y rápido. En 1990, junto con Robert Cailliau, presentó un proyecto a partir del cual y con el paso del tiempo, junto con el primer servidor web, llamado “CERN httpd”, también se creó el primer navegador web y otros elementos básicos de Internet como HTML y HTTP. Es así como Berners-Lee es hoy considerado el fundador de Internet.

**Tecnología De Servicios Web:**

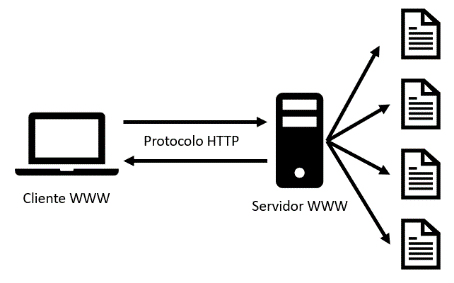
El protocolo utilizado para la transmisión es HTTP (o su variante cifrada HTTPS), que se basa, a su vez, en los protocolos de red IP y TCP (y muy rara vez en UDP). Un servidor web puede entregar los contenidos simultáneamente a varios ordenadores o navegadores web. La cantidad de solicitudes (requests) y la velocidad con la que pueden ser procesadas depende, entre otras cosas, del hardware y la carga (número de solicitudes) del host. Sin embargo, la complejidad del contenido también juega un papel importante: los contenidos web dinámicos necesitan más recursos que los contenidos estáticos.

Fig 1. Ilustración Protocolo Http

**Protocolo HTTP**

HTTP, de sus siglas en inglés: "Hypertext Transfer Protocol", es el nombre de un protocolo el cual nos permite realizar una petición de datos y recursos, como pueden ser documentos HTML. Es la base de cualquier intercambio de datos en la Web, y un protocolo de estructura cliente-servidor, esto quiere decir que una petición de datos es iniciada por el elemento que recibirá los datos (el cliente), normalmente un navegador Web.

Clientes y servidores se comunican intercambiando mensajes individuales (en contraposición a las comunicaciones que utilizan flujos continuos de datos). Los mensajes que envía el cliente, normalmente un navegador Web, se llaman peticiones, y los mensajes enviados por el servidor se llaman respuestas.

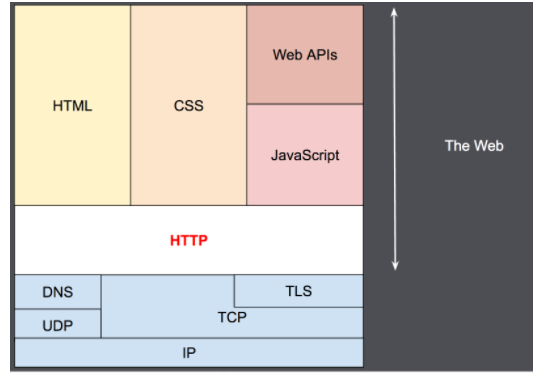


Fig 2. Interacción del protocolo http con aplicación.

**Características clave del protocolo HTTP**

* **Http es sencillo:** Incluso con el incremento de complejidad, que se produjo en el desarrollo de la versión del protocolo HTTP/2, en la que se encapsularon los mensajes, HTTP esta pensado y desarrollado para ser leído y fácilmente interpretado por las personas, haciendo de esta manera más facil la depuración de errores, y reduciendo la curva de aprendizaje para las personan que empieza a trabajar con él.
* **Http es extensible:** Presentadas en la versión HTTP/1.0, las cabeceras de HTTP, han hecho que este protocolo sea fácil de ampliar y de experimentar con él. Funcionalidades nuevas pueden desarrollarse, sin más que un cliente y su servidor, comprendan la misma semántica sobre las cabeceras de HTTP.
* **Http y conexiones:** Una conexión se gestiona al nivel de la capa de trasporte, y por tanto queda fuera del alcance del protocolo HTTP. Aún con este factor, HTTP no necesita que el protocolo que lo sustenta mantenga una conexión continua entre los participantes en la comunicación, solamente necesita que sea un protocolo fiable o que no pierda mensajes (como mínimo, en todo caso, un protocolo que sea capaz de detectar que se ha pedido un mensaje y reporte un error). De los dos protocolos más comunes en Internet, TCP es fiable, mientras que UDP, no lo es. Por lo tanto HTTP, se apoya en el uso del protocolo TCP, que está orientado a conexión, aunque una conexión continua no es necesaria siempre.

En la versión del protocolo HTTP/1.0, habría una conexión TCP por cada petición/respuesta intercambiada, presentando esto dos grandes inconvenientes: abrir y crear una conexión requiere varias rondas de mensajes y por lo tanto resultaba lento. Esto sería más eficiente si se mandaran varios mensajes.

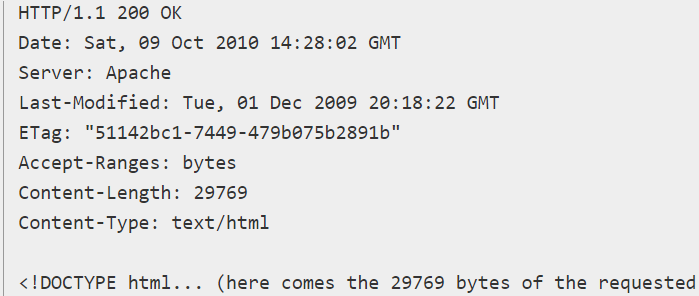
Para atenuar estos inconvenientes, la versión del protocolo HTTP/1.1 presentó el 'pipelining' y las conexiones persistentes: el protocolo TCP que lo transmitía en la capa inferior se podía controlar parcialmente, mediante la cabecera 'Connection'. La versión del protocolo HTTP/2 fue más allá y usa multiplexación de mensajes sobre una única conexión, siendo así una comunicación más eficiente.

**Flujo de vida del protocolo http:**

1. Abre una conexión TCP: la conexión TCP se usará para hacer una petición, o varias, y recibir la respuesta. El cliente pude abrir una conexión nueva, reusar una existente, o abrir varias a la vez hacia el servidor.
2. Hacer una petición HTTP: Los mensajes HTTP (previos a HTTP/2) son legibles en texto plano. A partir de la versión del protocolo HTTP/2, los mensajes se encapsulan en franjas, haciendo que no sean directamente interpretables, aunque el principio de operación es el mismo.

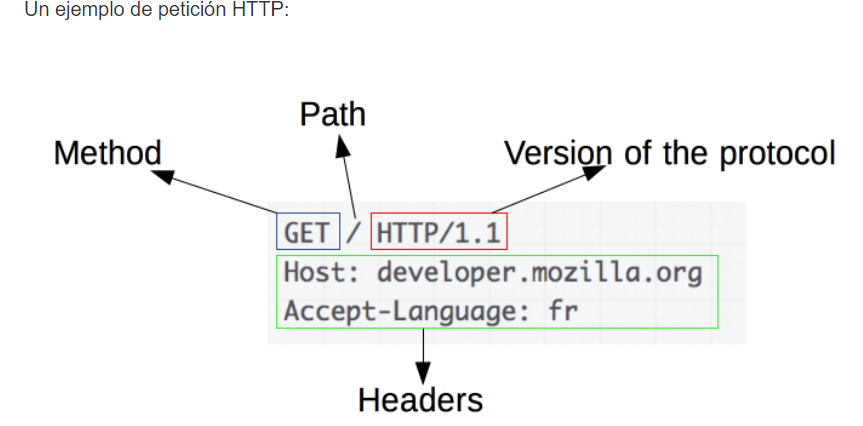


1. Leer la respuesta enviada por el servidor:



1. Cierre o reusó de la conexión para futuras peticiones.

**Peticiones:**

****

Una petición de HTTP, está formado por los siguientes campos:

* Un método HTTP, normalmente pueden ser un verbo, como: GET, POST o un nombre como: OPTIONS o HEAD, que defina la operación que el cliente quiera realizar. El objetivo de un cliente, suele ser una petición de recursos, usando GET, o presentar un valor de un formulario HTML, usando POST, aunque en otras ocasiones puede hacer otros tipos de peticiones.
* La dirección del recurso pedido; la URL del recurso, sin los elementos obvios por el contexto, como pueden ser: sin el protocolo (http://), el dominio (aquí developer.mozilla.org), o el puerto TCP (aquí el 80).
* La versión del protocolo HTTP.
* Cabeceras HTTP opcionales, que pueden aportar información adicional a los servidores.
* O un cuerpo de mensaje, en algún método, como puede ser POST, en el cual envía la información para el servidor.

**Respuestas:**

Las respuestas están formadas por los siguentes campos:

* La versión del protocolo HTTP que están usando.
* Un código de estado, indicando si la petición ha sido exitosa, o no, y debido a que.
* Un mensaje de estado, una breve descripción del código de estado.
* Cabeceras HTTP, como las de las peticiones.
* Opcionalmente, el recurso que se ha pedido.

**Métodos de Peticiones Htttp:**

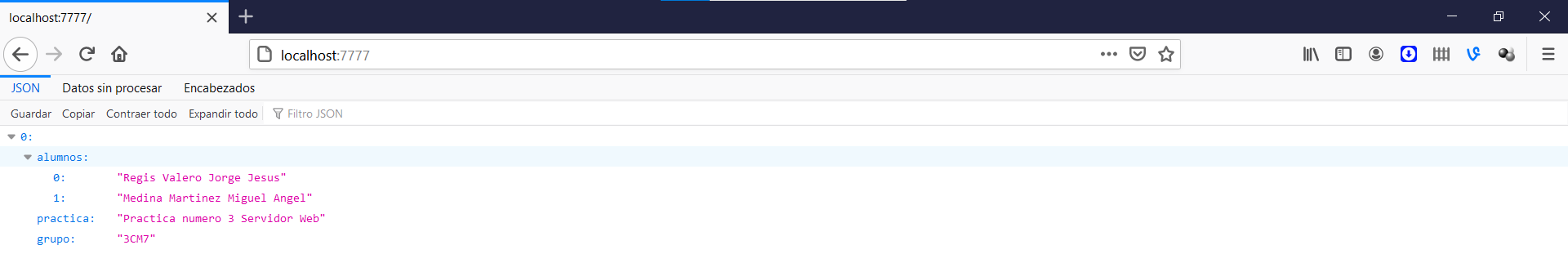
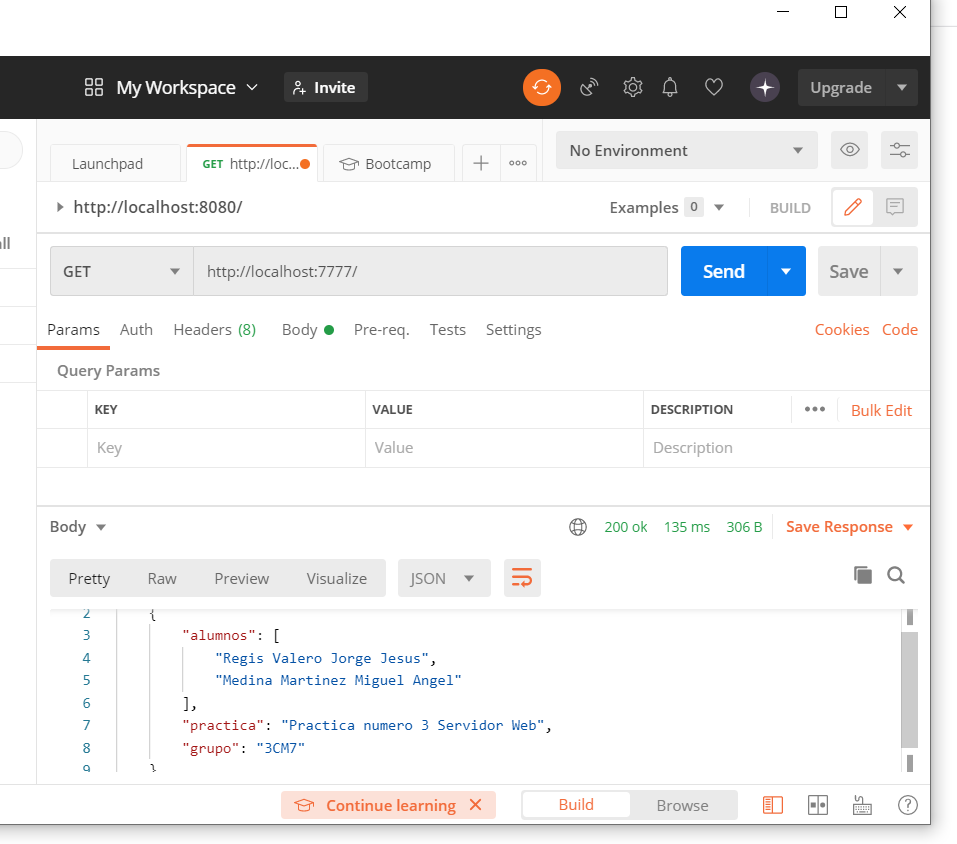
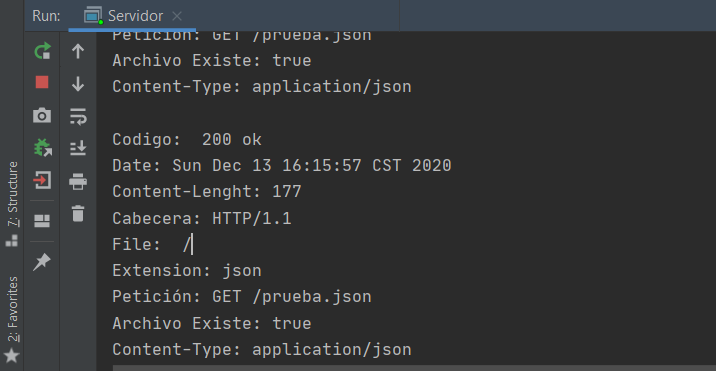
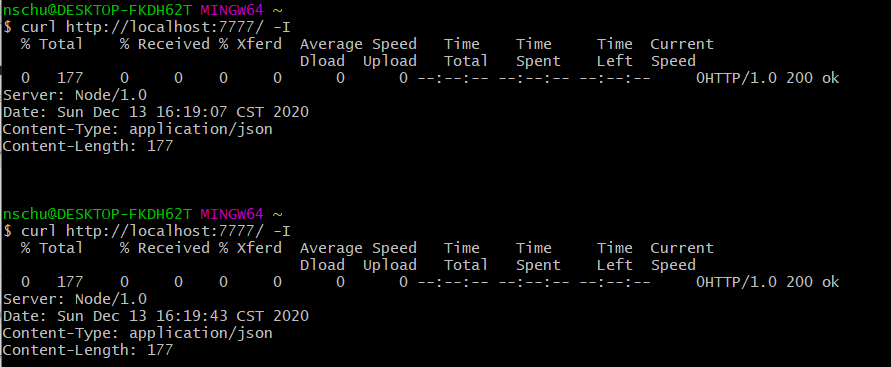
HTTP define un conjunto de métodos de petición para indicar la acción que se desea realizar para un recurso determinado. Aunque estos también pueden ser sustantivos, estos métodos de solicitud a veces son llamados HTTP verbs. Cada uno de ellos implementan una semántica diferente, pero algunas características similares son compartidas por un grupo de ellos.

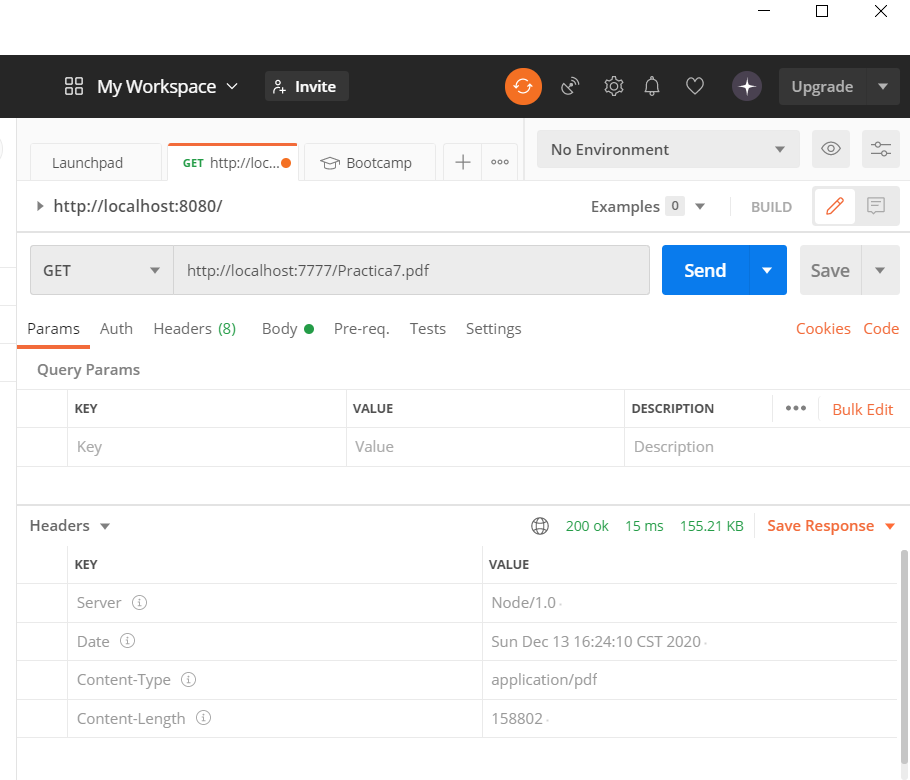
* GET : El método GET solicita una representación de un recurso específico. Las peticiones que usan el método GET sólo deben recuperar datos.
* HEAD: El método HEAD pide una respuesta idéntica a la de una petición GET, pero sin el cuerpo de la respuesta.
* POST: El método POST se utiliza para enviar una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor.
* PUT: El modo PUT reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.
* DELETE: El método DELETE borra un recurso en específico.

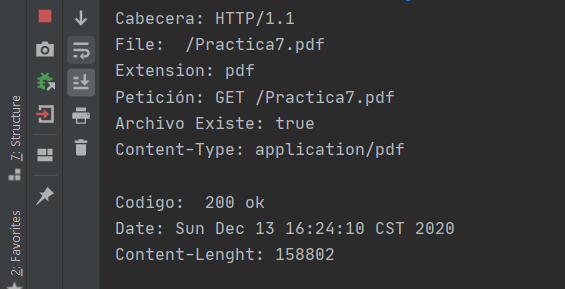
**Desarrollo de la practica:**

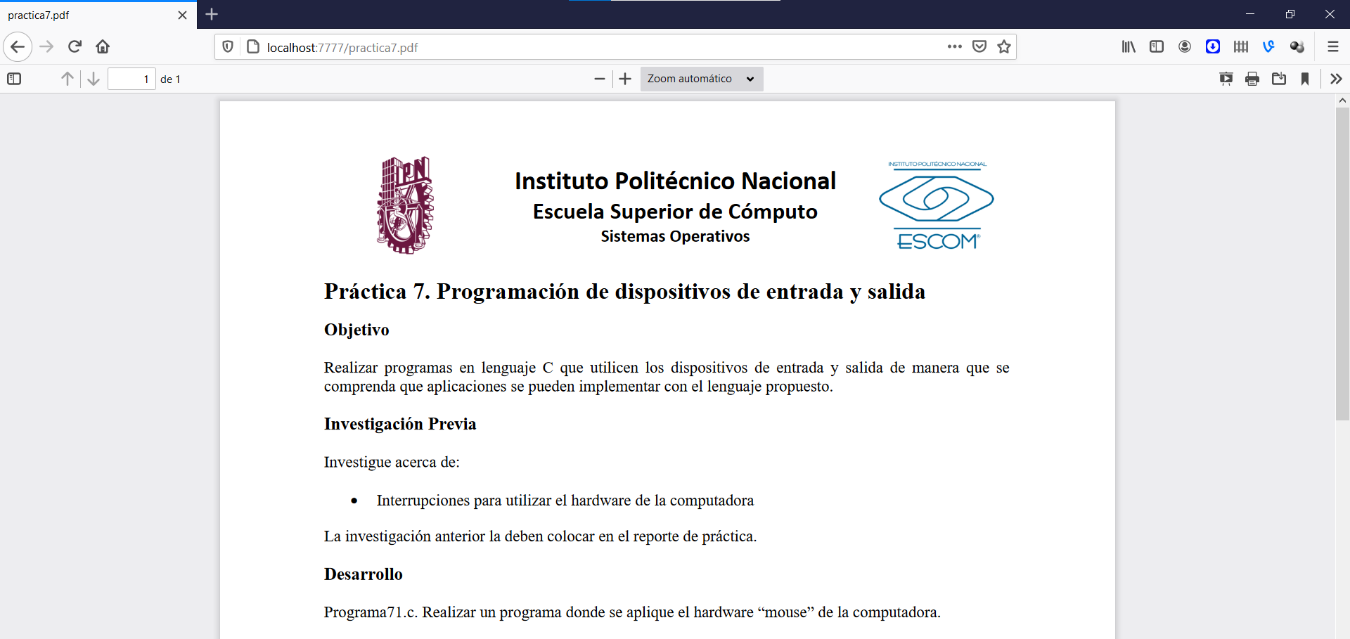
**Capturas de funcionamiento:**

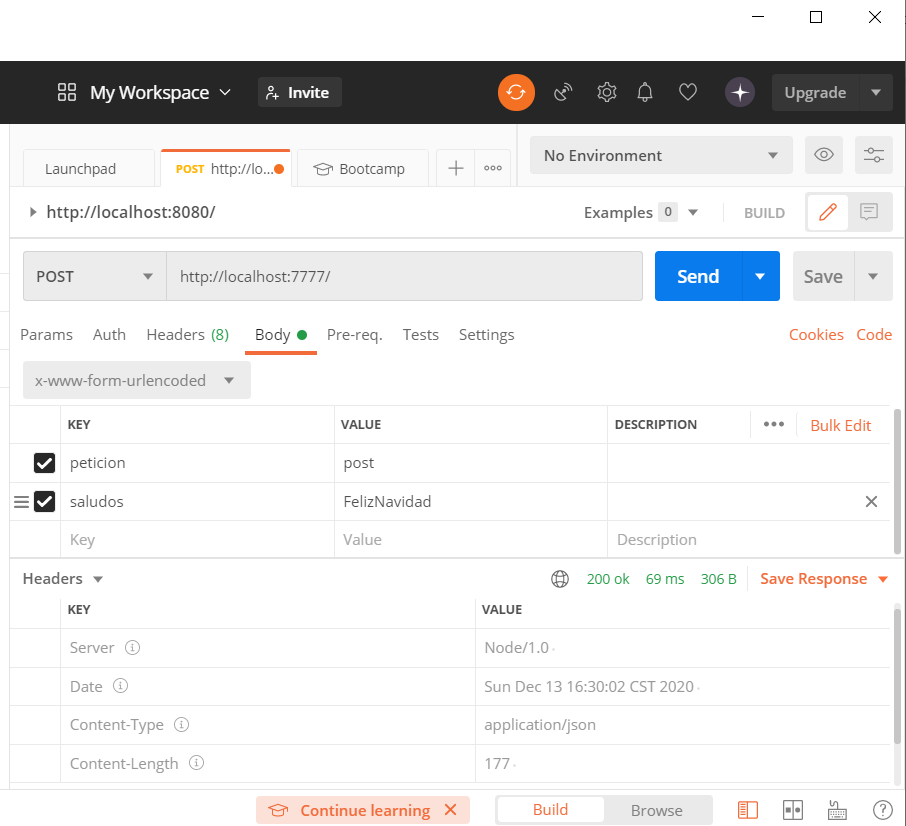
Petición GET a localhost sin solicitar ningún recurso mediante postman , neavegador y consola

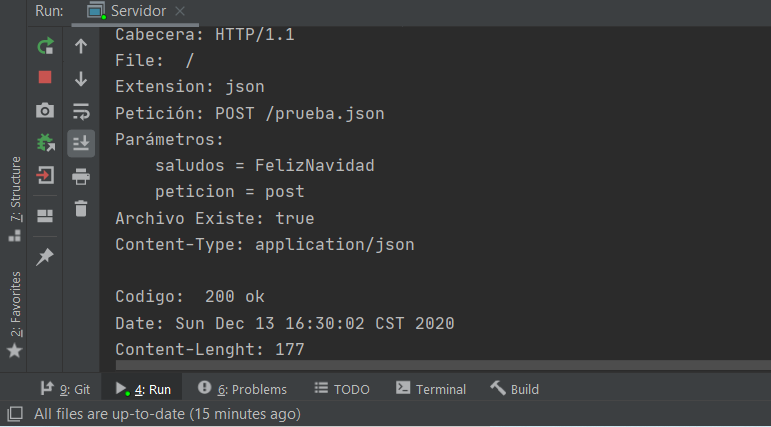
****Peticion GET solicitando un recurso en postman y navegador:

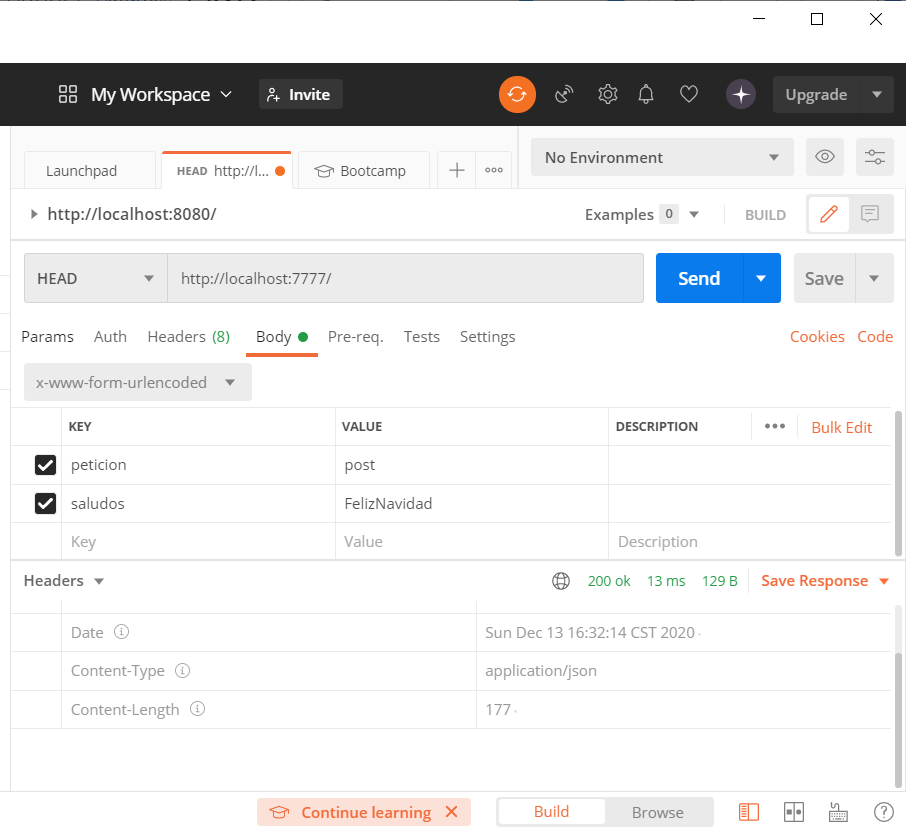
****

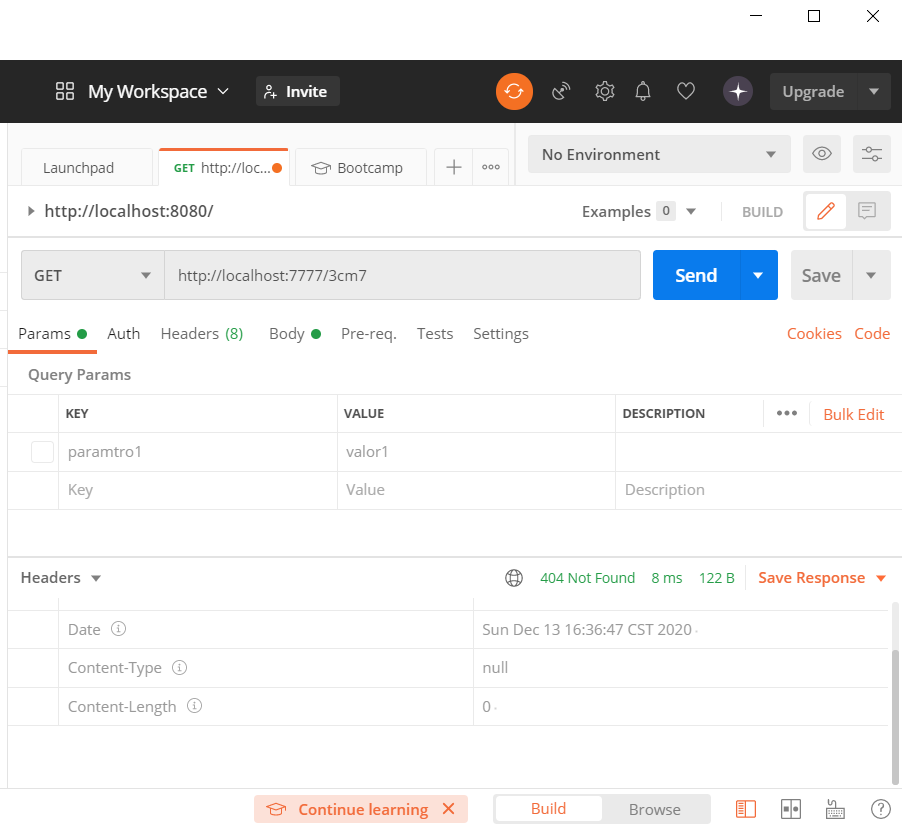
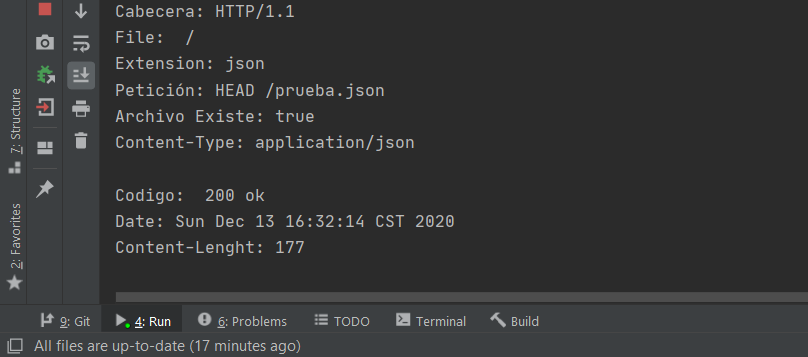


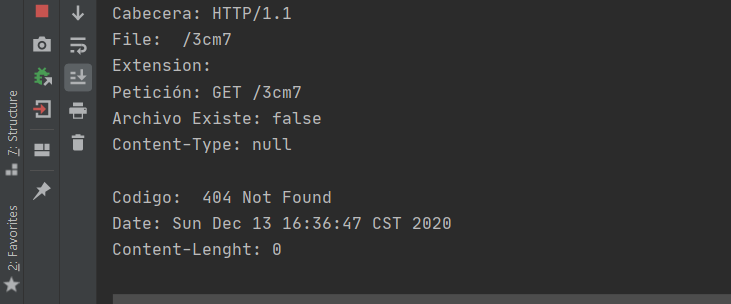
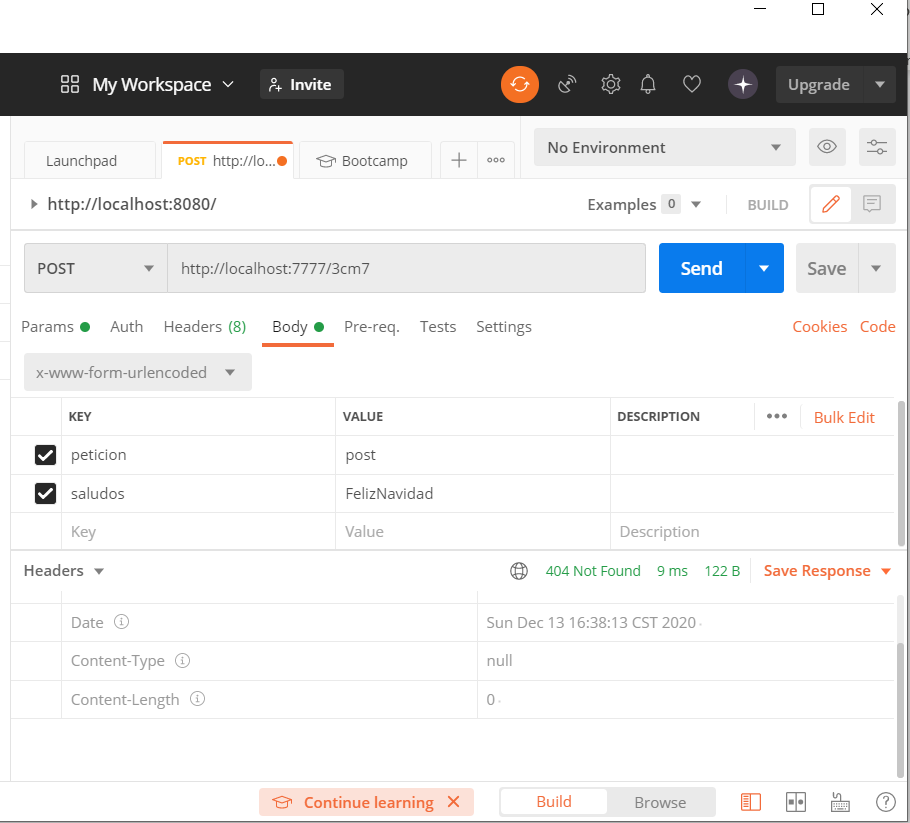
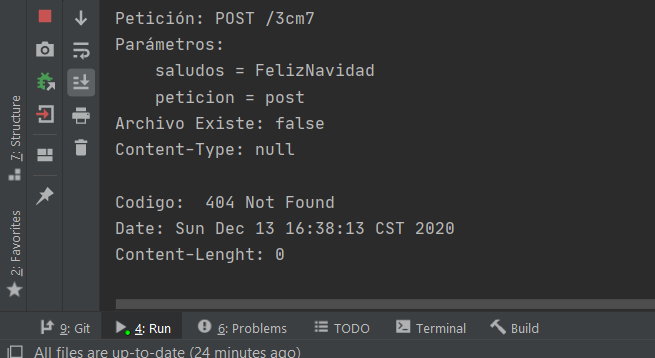
****

**Peticion POST**

****

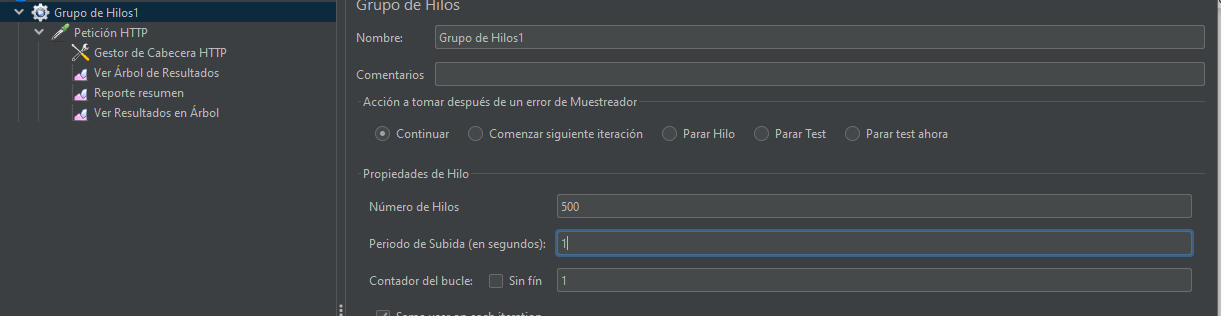
**Peticion Head:**

**Petición a un recurso no disponible:**

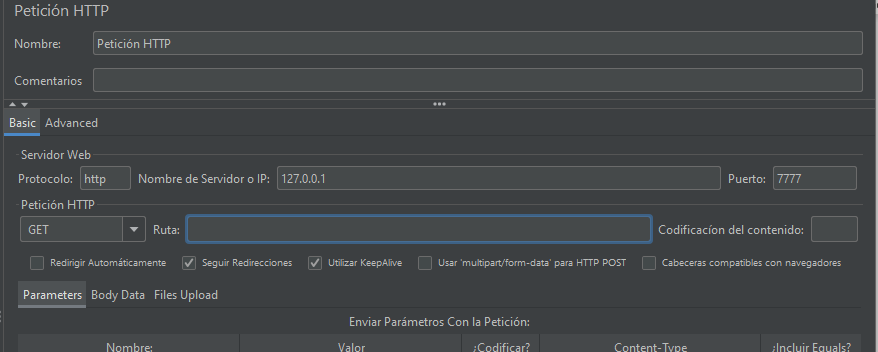
****

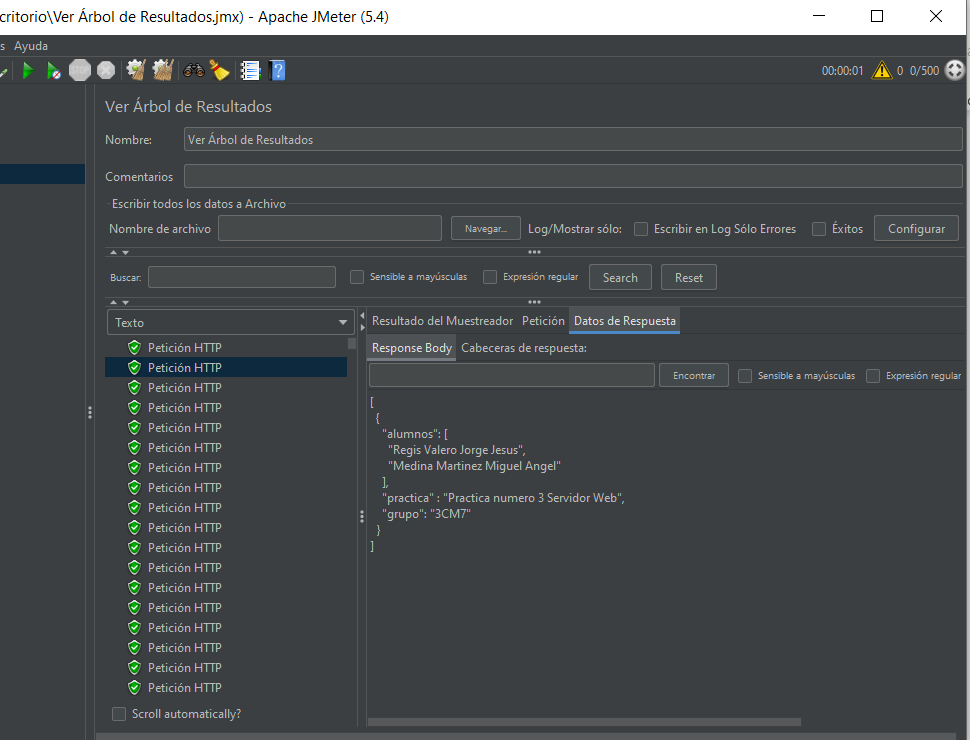
**Múltiples Peticiones a nuestro servidor web Probado nuestro Pool de Hilos**

Con la herramienta JMeter al cual es una herramienta creada por Apache la cual ayuda a testear servicios web probaremos múltiples peticiones 500 peticiones en un segundo

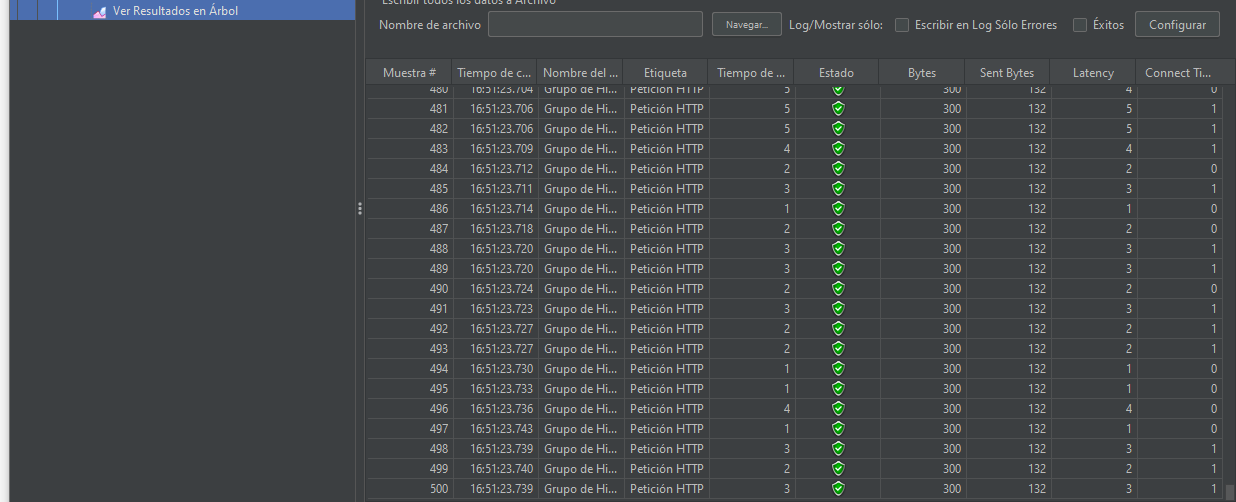
****Seleccionamos los numero de peticiones que deseamos simular en este caso 500 peticiones en un periodo de un segundo. El contador de bucle es la cantidad de peticiones que realiza cada hilo en un segundo, es decir cada hilo va hacer 1 petición es un segundo.

Insertamos el protocolo que usamos, la dirección ip , el número de puerto y el método o verbo http que se va utilizar en este caso GET. Si queremos hacer la petición al home Page o alguna ruta en especifica.

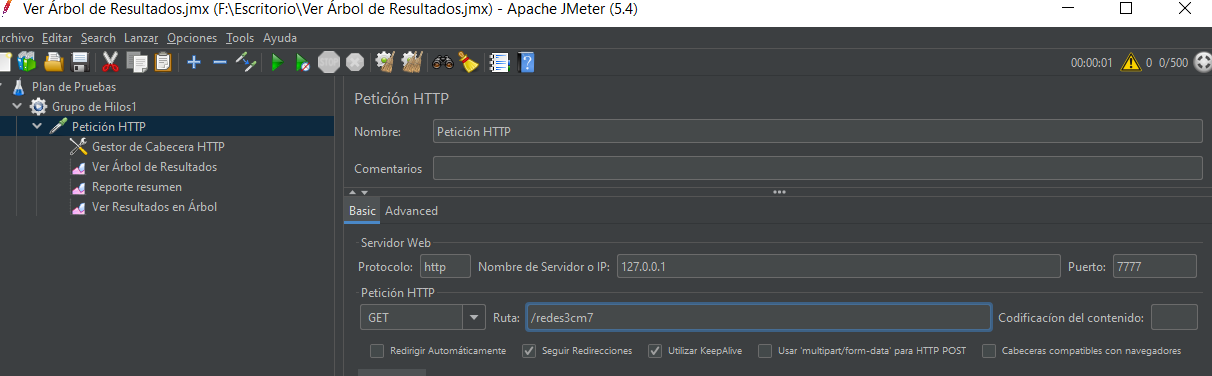


****Damos click en el botón iniciar para ejecutar las peticiones y podemos ver el resultado si la petición fue aceptada o rechazada. En caso de ser aceptada podemos ver los datos de respuesta como podemos observar nos regresa nuestro JSON.

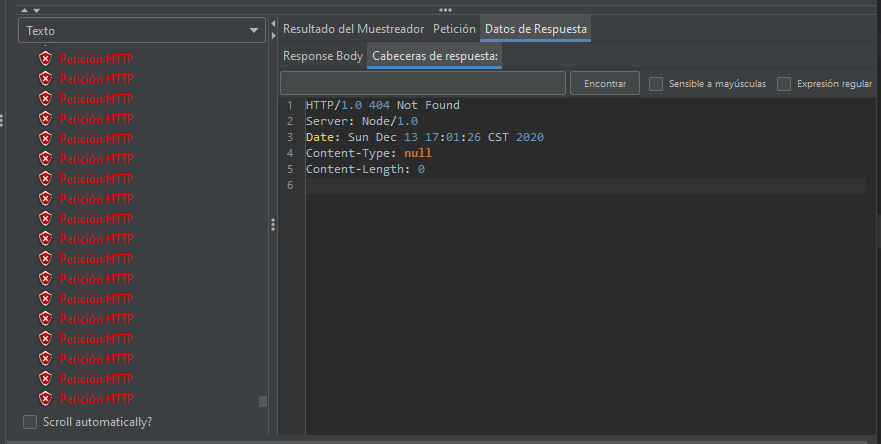
Podemos ver datos más específicos nos muestra las 500 peticiones su estado en este caso 200 ok que significa que la petición fue aceptada , el tamaño y la latencia cuando se conectó el cliente con el servidor y finalizo

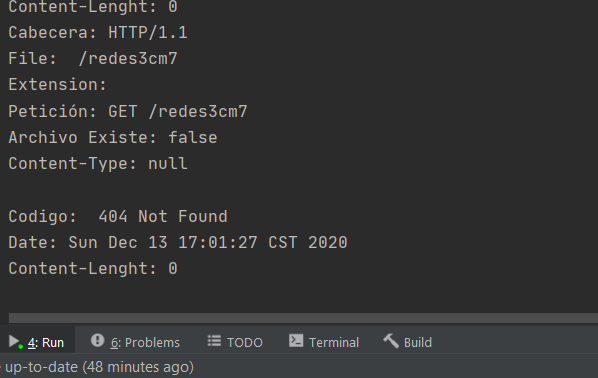
****

Para comprobar solicitaremos 500 peticiones a un recurso no disponible

****

Resultado:

****

****

Conclusiones:

**Regis Valero Jorge Jesus:**

El manejo de hilos resulta muy útil como herramienta de trabajo al ejecutar diversas tareas o acciones paralelamente dentro de un mismo proceso y como recurso, ya que, se ocupa una cantidad considerablemente menor de CPU para ejecutar tareas a través de hilos. Aunque no todo son enteramente ventajas, una de las particularidades de los hilos es que no son independientes a su proceso padre, por lo tanto, cuando el proceso padre finaliza, los hilos se finalizan también por default. En el caso de nuestro servidor web nos ayuda para que el servidor web tenga la capacidad te tener múltiples clientes o peticiones lográndolo a java que es capaz de crear multihilos.

**Referencias:**