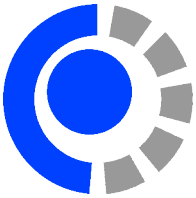
Universidad Nacional del Comahue

Facultad de Informática

Departamento de Ingeniería de Computadoras



Laboratorio de Programación Distribuida

Trabajo Obligatorio N° 1

Implementación de arquitectura de servicios utilizando sockets con NodeJS y Typescript

Amarante Carlos Adolfo, FAI-1922

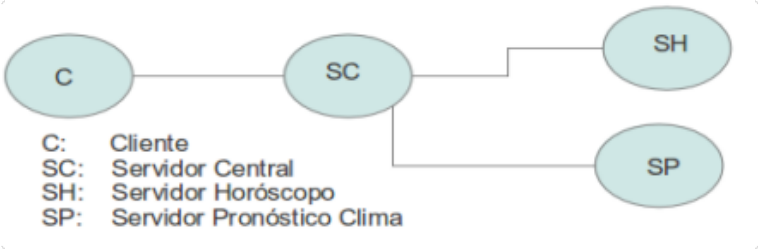
Abril 2021

1 Introducción

1.1 Problema

Se pide desarrollar un programa en lenguaje java utilizando las primitivas de programación sockets de tipo Stream.

El esquema general es el siguiente:



C realiza una consulta a SC pidiendo una predicción del Horóscopo para un signo determinado y el pronóstico del clima dada una fecha. SC utilizando el signo consultará a SH y con la fecha a SP. Las predicciones de SH y SP son random.
El trabajo puede realizarse en distintos niveles:

- Básico: Los servidores son simples y solo atienden una solicitud a la vez.
- Medio: Los servidores pueden responder múltiples consultas a la vez.
- Avanzado: Nivel Medio y las consultas repetidas obtienen el mismo resultado

El problema dado sugiere una arquitectura de servicios, donde un servidor central recibirá solicitudes de clientes, las redirigirá a otros servidores, de los cuales obtendrá su respuesta y las reenviara al cliente. Todo esto usando sockets tipo stream.

Se propone:

- Usar NodeJS con Typescript y la libreria net para construir la solución.
- Aprovechar la arquitectura orientada a eventos de node para permitir al servidor centrar recibir data y redirigirla donde corresponda.
- Utilizar la libreria child_process para resolver las solicitudes recibida por los servicios

2 Marco teórico

2.1 NodeJS

Ideado como un entorno de ejecución de JavaScript orientado a eventos asíncronos, Node.js está diseñado para crear aplicaciones network escalables

Node.js es similar en diseño y está influenciado por sistemas como Event Machine de Ruby y Twisted de Python. Pero Node.js lleva el modelo de eventos un poco más allá. Incluye un bucle de eventos como runtime de ejecución en lugar de una biblioteca. En otros sistemas siempre existe una llamada de bloqueo para iniciar el bucle de eventos. Por lo general, el comportamiento se define mediante devoluciones callbacks de llamada al iniciarse un script y al final se inicia un servidor a través de una llamada de bloqueo como EventMachine::run(). En Node.js, no existe como tal la llamada de inicio del evento de bucle o start-the-event-loop. Node.js simplemente entra en el bucle de eventos después de ejecutar el script de entrada y sale cuando no hay más devoluciones callbacks de llamada para realizar. Se comporta de una forma similar a JavaScript en el navegador - el bucle de eventos está oculto al usuario.[1]

[1] https://nodejs.org/es/about/

2.1.1 Arquitectura basada en eventos

La arquitectura impulsada por eventos es relativamente común hoy en día y las aplicaciones impulsadas por eventos producen, detectan y reaccionan a diferentes tipos de eventos.

Podría decirse que el núcleo de Node.js se basa en parte en eventos, ya que muchos módulos nativos, como el sistema de archivos (fs), y stream módulo están escritos como EventEmitters mismos.[2]

2.2 Typescript

Typescript es un superset de JavaScript. Decimos que una tecnología es un superset de un lenguaje de programación, cuando puede ejecutar programas de la tecnología, Typescript en este caso, y del lenguaje del que es el superset, JavaScript en este mismo ejemplo. En resumen, esto significa que los programas de JavaScript son programas válidos de TypeScript, a pesar de que TypeScript sea otro lenguaje de programación.[3]

2.3 Arquitectura de microservicios

Es un enfoque que consiste en crear una sola aplicación como una colección de pequeños servicios, cada uno de los cuales se ejecuta en su propio proceso, y un mecanismo de comunicación ligero, habitualmente es un recurso de una interfaz de programación de aplicaciones (API) sobre un protocolo HTTP.

Estos servicios están impulsados por oportunidades comerciales y pueden desplegarse de forma independiente e implementarse automatizada. Estos servicios tienen una gestión centralizada mínima, pueden estar escritos en diferentes lenguajes de programación y utilizan diferentes tecnologías de comunicación y almacenamiento de datos. (Lewis, 2014)

En nuestro caso es el servidor central el encargado de la gestión centralizada y los servicios son relacionados al horóscopo y el pronóstico del clima.

2.4 Modulo net

Según la documentación oficial de NodeJS, el módulo net proporciona una API de red asíncrona para crear servidores TCP o IPC basados en Stream (net.createServer()) y clientes (net.createConnection()).

2.5 Libreria child_process

Según la documentación oficial de NodeJS, el módulo child_process ofrece la posibilidad de generar subprocesos de forma similar, aunque no idéntica, a popen(3).

Esto permite delegar tareas a otro proceso y poder comunicarlos entre sí.

Cabe aclarar que esto es costos en cuanto a recursos computacionales pues no estamos hablando de hilos. En nuestro caso, los servicios serán los encargados de usar procesos para manejar las solicitudes del servidor central, pero dado a que la tarea asignada (devolver información de una pseudo base de datos) es tan ligera sería mejor simplemente dejar que el proceso principal haga todo o bien agrupar varias solicitudes para después ser resueltas por otro proceso.

3 Desarrollo

3.1 Cliente

El cliente consistirá en un servidor que permitirá la interacción por consola con el servidor central.

Dada la arquitectura orientada a eventos, al lanzar el servidor cliente se deberá:

1. Crear una conexión (socket) con el servidor central
2. Pedir al usuario elegir entre un conjunto de opciones para hacer solicitudes al servidor central mediante el socket creado previamente
3. La conexión creada con el servidor central debe estar al tanto de un evento del tipo data para volver a preguntar al usuario que desea hacer
4. En caso de decidir salir del programa, se debe cerrar la conexión con el servidor central

[2] https://pharos.sh/manejo-de-eventos-en-node-js-con-eventemitter/
[3] Fuente

3.2 Servidor central

El servidor central actuara de manejador de solicitudes y respuesta, esto requiere:

1. Crear 2 conexiones via socket, una con el servicio de clima y otra con el servicio de horóscopo.
2. Estar al tanto de conexiones entrantes
3. En caso de una conexión entrante al puerto designado al servidor central, esta debe ser una solicitud de un cliente y deberá redirigirla al socket que corresponda, sea de clima u horosocopo, previamente guardando en una estructura de tipo hash la conexión entrante desencadenante de toda esta acción.
4. Cuando los sockets correspondientes a servicios reciban data, esto quiere decir que una solicitud de un cliente ha sido atendida, el servidor deberá redirigirla al cliente correspondiente usando la estructura del tipo hash mencionada en el item anterior

3.3 Servicio horoscopo o clima

Se crea un servidor para cada uno de estos servicios, estos deberan

1. Aceptar conexiones
2. Al recibir data están recibiendo solicitudes, deben crear procesos hijos para resolverlas
3. Una vez procesada una solicitud, el proceso hijo debe enviar esta data procesada al proceso padre
4. El proceso padre mata al proceso hijo del cual recibio la data y reenvia la respuesta al servidor central.

3.4 Solicitud de un cliente

Un resumen del tratamiento del la solicitud del cliente puede observarse en la imagen adjunta a este trabajo llamada "Diagrama-solicitud-cliente.png".

4 Conclusión

El servidor central es capaz de responder a las solicitudes de los clientes como a la respuesta de los servicios aprovechando la arquitectura basada en eventos de node, cumpliendo de esta manera con los requerimientos para este elemento planteados en el problema.

Los servidores de servicios crean procesos para resolver las solicitudes recibidas y luego enviar sus respuestas, esto permite la respuesta asincronica de las solicitudes, cumpliendo de esta manera con los requerimientos para estos elementos planteados en el problema.

En cuanto a que las consultas repetidas obtengan el mismo resultado en base a la misma entrada, surgió el problema de que se obtenía una fecha distinta al solicitar la fecha actual porque cambiaba la hora. Esto fue solucionado normalizando la fecha en los servidores de servicios.

De esta forma se cumplió con todos los requerimientos planteados

References

Lewis, J. (2014). *Microservices.* Retrieved from `https://martinfowler.com/articles/microservices.html`