Disponible en línea en www.sciencedirect.com

ScienceDirect
ScienceDirect
Procedia Computer Science 00 (2019) 000 - 000
Disponible en línea en www.sciencedirect.com
Procedia Computer Science 00 (2019) 000 - 000
www.elsevier.com/locate/procedia
www.elsevier.com/locate/procedia
ScienceDirect

Procedia Computer Science 159 (2019) 1035-1044

Procedia
Computer Science

www.elsevier.com/locate/procedia

23ª Conferencia Internacional sobre Basada en el Conocimiento y la Información e Ingeniería Inteligente sistemas
23ª Conferencia Internacional sobre Basada en el Conocimiento y la Información e Ingeniería Inteligente sistemas

Microservicios dinámicas para crear tolerancia de fallos y escalable Arquitectura

Microservicios dinámicas para crear tolerancia de fallos y escalable Arquitectura

Mihai Baboi una, Adrian Iftene una,*, Daniela Gifu a B C †

una Facultad de Ciencias de la Computación, "Alexandru Ioan Cuza", el general Berthelot, 16, 700483, Rumania

si Instituto de Ciencias de la Computación, Academia Rumana - rama de Iasi, Bulevardul Carol I, 8, 700505, Rumania

c Cognos Business Consulting SRL, 7, Iuliu Maniu Blvd., 061072, Bucarest, Rumania
Mihai Baboi una, Adrian Iftene una,*, Daniela Gifu a B C †

Resumen

Una de las tendencias más importantes de la industria de la arquitectura empresarial se relaciona con el uso de microservicios, a la detrime nt de arquitecturas monolíticas, que están empezando ya no se utiliza. Debido a las arquitecturas de nube nativa el despliegue de sistemas de microservicios es más productivo, flexible y rentable. De todos modos, muchas empresas comenzaron a migrar de un tipo de arquitectura a otra, pero todavía está en la fase temprana. En este trabajo se abordan los retos que plantea la necesidad de desarrollar un sistema escalable y tolerancia a fallos basado en microservicios. En nuestros experimentos se consideran dos tipos de microservicios, simple y extendida y la solución propuesta resulta ser un innovador en especial uno basado en su comportamiento dinámico.
Resumen

© 2019 Los Autores. Publicado por Elsevier
Este es un artículo de acceso abierto bajo la licencia CC BY-NC-ND licencia (https://creativecommons.org/licenses/by-nc-nd/4.0/) Examen de los pares bajo la responsabilidad de KES Internacional.

palabras clave: microservicios; comportamiento dinámico; escalabilidad; Tolerancia a fallos
palabras clave: microservicios; comportamiento dinámico; escalabilidad; Tolerancia a fallos

1. Introducción

1. Introducción

La historia de los lenguajes de programación y paradigmas de la informática se ha caracterizado en las últimas décadas por una mayor atención a la distribución y la modularización para mejorar la reutilización y la robustez del código.

* Autor correspondiente. Tel .: +40 232 201 091.
Dirección de correo electrónico: adiftene@info.uaic.ro
† Autor correspondiente. Tel .: +40 232 201 771.
Dirección de correo electrónico: daniela.gifu@info.uaic.ro

1877-0509 © 2019 Los Autores. Publicado por Elsevier
Este es un artículo de acceso abierto bajo la licencia CC BY-NC-ND licencia (https://creativecommons.org/licenses/by-nc-nd/4.0/) Examen de los pares bajo la responsabilidad de KES Internacional.
10.1016 / j.procs.2019.09.271

Había una necesidad de aumentar la cantidad y calidad del software [1]. Uno de los factores clave para aclarar algunos malentendidos relacionados con la ingeniería innovadora es la forma de diferentes herramientas adecuadas con el fin de diseñar y desarrollar mejores sistemas de software [2]. El éxito principal en este proceso se ha demostrado recientemente por los sistemas basados en microservicios [3] siendo un paradigma de arquitectura orientada en diferentes aplicaciones (por ejemplo, personas con discapacidad) [3]. Bajo el término microService, existe un interés creciente en la arquitectura y el diseño. Calidad

atributos (por ejemplo, la escalabilidad, el rendimiento, y la tolerancia de error) o la elección de modelos como "contractual de servicio" [5]

o puerta de enlace API ya no se viola el principio YAGNI (" *Yagni* "), Que sufre un tipo de error BDUF (" *Gran Diseño Up Front* "). La pregunta de investigación principal de este documento tiene la intención de responder *cómo podríamos desarrollar un sistema basado en microservicios tan fácil como el desarrollo de un sistema monolítico?* A partir de esta preocupación, *la forma en que se podría crear un entorno que permita la distribución dinámica de la potencia de cálculo entre los clientes?* Nuestra hipótesis de investigación propone el uso de la arquitectura de un sistema cliente-servidor que combina la computación distribuida y microservicios para resolver estos desafíos.

El documento está estructurado de la siguiente manera: La sección 2 presenta un breve resumen de la literatura actual en la clarificación de la importancia de los microservicios, incluyendo dos servicios ofrecidos por conocidos Azure, mientras que la sección 3 se analiza la arquitectura propuesta. La sección 4 se analiza la evaluación de este sistema antes de sacar algunas conclusiones en la última sección.

2. Revisión de la literatura sobre microservicios

Debido las arquitecturas en la nube nativo el despliegue de sistemas microservicios es más productivo, flexible y rentable [6]. Sin embargo, observa que Zimmermann microservicios son un tema delicado, especialmente investigado por la academia [7] y la industria. Por primera vez, microservicios término fue discutido en un taller de arquitectos de software en Italia en mayo de 2011 con el fin de describir lo que los participantes veían como un estilo arquitectónico común recientemente explorado por muchos de ellos. Un año más tarde, el mismo grupo confirmó que microservicios término es el nombre correcto. De hecho, los microservicios se han desarrollado como respuesta a los problemas en las aplicaciones monolíticas o arquitecturas orientadas a servicios que han puesto en dificultades por parte de la escalabilidad, la complejidad de la pieza, y por parte de las dependencias de la aplicación en construcción, todos los que utilizan mecanismos de comunicación ligeros [8-9]. Dado que el monolito es una aplicación de software cuyos módulos no se pueden ejecutar de manera independiente, la solución basada en microservicios debe ser considerado como el único capaz de ejecutar instrucciones independientes el uno del otro [10-11]. Estas grandes monolitos se convierten en el tiempo difícil de mantener y evalúan con dificultad debido a su complejidad, pero una desventaja importante es que limitan la escalabilidad del producto. Otro problema viene del hecho de que no ofrece resistencia de la falta, y no hay posibilidad de que un componente del sistema para trabajar mientras que otro componente no funciona, lo que es posible con las arquitecturas orientadas MICROSERVICE. Dado que el monolito es una aplicación de software cuyos módulos no se pueden ejecutar de manera independiente, la solución basada en microservicios debe ser considerado como el único capaz de ejecutar instrucciones independientes el uno del otro [10-11]. Estas grandes monolitos se convierten en el tiempo difícil de mantener y evalúan con dificultad debido a su complejidad, pero una desventaja importante es que limitan la escalabilidad del producto. Otro problema viene del hecho de que no ofrece resistencia de la falta, y no hay posibilidad de que un componente del sistema para trabajar mientras que otro componente no funciona, lo que es posible con las arquitecturas orientadas MICROSERVICE. Dado que el mon

En SOA (Service Oriented Architecture) los principales servicios son coordinados utilizando dos métodos: la orquestación (donde hay una microService central que envíe solicitudes a otros servicios y supervisar todo el proceso mediante el envío y la recepción de las respuestas) y la coreografía (lo que no supone ningún centralización, pero cada servicio sabe de antemano lo que tiene que hacer) [1]. Al igual que con las arquitecturas monolíticas y arquitecturas SOA, la cuestión más difícil sigue siendo la descomposición del sistema en los servicios [12]. Además, en absoluto no debe ser dejado de lado la cuestión de proporcionar información confidencial a través de la propagación incontrolada de los servicios [13].

Nuestros combina la arquitectura de computación distribuidos con microservicios para crear un ambiente que permita la distribución dinámica de la informática entre los clientes. Por la computación distribuida, entendemos la disponibilidad de procesamiento y almacenamiento de grandes cantidades de datos en la nube, un elemento clave en la industria de hoy, dentro y fuera del área de TI. sistemas de almacenamiento distribuidos están diseñados para cumplir los requisitos de aplicaciones extendidas distribuidos y computacionalmente con amplia aplicabilidad, la escalabilidad y alto rendimiento. Una solución conocida es MapReduce [14]. Una solución conocida es MapReduce que orquesta el procesamiento por clasificación de servidores distribuidos, mediante la gestión de las diferentes tareas en paralelo, todas las comunicaciones y las transferencias de datos entre las partes del sistema que proporciona redundancia y tolerancia a fallos.

Otro modelo de programación utilizado para ejecutar de manera eficiente con las aplicaciones informáticas en paralelo o en gran escala, sin necesidad de configuración manual o gestión de la infraestructura, la más potente de computación de alto rendimiento (HPC) racimos estén disponibles en lotes Azure [15]. Para ilustrar estas ideas, recordamos SaaS (Software como servicio) o aplicaciones de cliente donde es necesario ampliamente ejecución [16]. De hecho, diferentes empresas TIC son

exhibiendo un mayor interés en SaaS, siendo atractiva para reducir sus costos operativos y, en consecuencia, aumentar su agilidad de la empresa [17]. Otro de los servicios ofrecidos por los principales proveedores de servicios de nube Azure es Funciones que permiten la ejecución bajo demanda sin la infraestructura que tiene que ser proporcionado o administrado [18] de forma explícita.

También da un mayor interés a las aplicaciones a ejecutar fácilmente las pequeñas piezas de código o "funciones" en la nube. El interés creciente de Internet-de-objetos (IO) hace que las funciones de Azure [19] para convertirse en una solución excelente para el procesamiento de datos, la integración del sistema y la construcción de APIs y microservicios simples.

3. Metodología

Estructural, el sistema propuesto puede ser dividido en 3 zonas diferentes: (1) el cliente - el que va a ejecutar las tareas asignadas por el servidor; (2) del servidor - la interfaz con el cliente, el cerebro de la aplicación monolitos (3) el área de gestión de relaciones cliente-servidor que encapsula todos los detalles conectados por la transferencia de la ejecución de servidor a cliente. Toda la información enviada a través de la red entre el cliente y el servidor está cifrada utilizando el algoritmo DES (Data Encryption Standard) y la tecla se cambia a través del protocolo Diffie-Hellman [20], que, aunque es vulnerable bajo ciertas condiciones, sigue siendo aplicados en los distintos Las soluciones de seguridad de Internet.

3.1. Arquitectura del sistema

Nuestro sistema respeta en gran medida basado en la arquitectura de un sistema microservicios dinámicos. La arquitectura se basa en un tipo de servidor-cliente en el que un servidor corresponde a más clientes. Tanto el servidor como el cliente de gestión microservicios web, siendo el protocolo de comunicación HTTP, el formato de datos está en JSON. Esta arquitectura es útil en la distribución y redistribución de los recursos de forma dinámica entre los clientes. Tal un modelo arquitectónico se utiliza para construir aplicaciones grandes, complejas y escalables en horizontal que consiste en procesos pequeños, independientes y desacopladas se comunican entre sí utilizando las interfaces de programación de aplicaciones (API) [21].

En la Fig. 1 es describe cómo el servidor distribuye paquetes con todas las funciones a sus clientes. Dependiendo del número de clientes, puede haber instrucciones que no están asignados a cualquier cliente o el mismo conjunto de instrucciones asignadas a varios clientes.

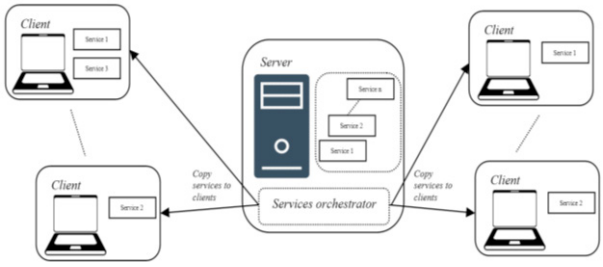

Fig. 1. La distribución de servicios a clientes.

La arquitectura de la aplicación fue construida utilizando el marco ASP.NET MVC de Microsoft. En la parte central, podemos ver microservicios del lado del servidor en el servidor, y en la derecha e izquierda, más clientes que están esperando para ejecutar tareas desde el servidor. componente de servicio Orchestrator garantiza por un lado, la comunicación entre el servidor y los clientes, el envío de tareas desde el servidor a los clientes, y por otro lado se supervisa el estado de estas solicitudes.

Esta arquitectura ofrece la posibilidad de que un microService a llamar a otro microService (obtención de una microService extendida) o llamar entre sí, lo que puede conducir a una dependencia circular, lo que se debe impedir a nivel de usuario. El protocolo de comunicación cliente-servidor se lleva a cabo en las siguientes etapas:

1) El cliente se conecta al servidor y se inicia el protocolo de intercambio de claves. También dará al servidor y al puerto que corresponderá a.
2) El servidor notifica al cliente con la siguiente tarea a ejecutar (una tarea está representado por un par (microService, los datos de entrada)).

3) El cliente recibe la tarea y luego notifica al servidor que la transmisión y la carga de la misma ha tenido éxito o no éxito.

4) Una vez establecida la conexión entre las dos entidades, el servidor envía los datos en formato JSON, cifrado con DES para el cliente para su procesamiento.

5) El cliente recibe los datos y devuelve la respuesta (los datos de salida obtenidos a partir de la microService que se ejecuta en los datos de entrada) también en formato JSON cifrado con DES.

6) Para cada minuto que el cliente notifica al servidor que esté disponible o no para recibir nuevas tareas.

7) El servidor puede inicializar cualquier momento del procedimiento para restablecer el cliente, en cuyo caso se pondrá a cero y será capaz de aceptar nuevas tareas.

Un caso especial de esta comunicación es el escenario en el que un cliente se ejecuta una tarea necesita la respuesta de otro cliente para completar la ejecución. En este caso, se evaluaron dos posibilidades existentes: orquestación y coreografía.

En el caso de la coreografía, hemos identificado varios obstáculos: (a) la lista de clientes disponibles para la ejecución de la tarea externa tenía que ser enviado por el servidor al cliente, y mantener esta lista de valores actualizados ejercería presión sobre la red de frecuencia intercambiando información; (B) la conexión entre los dos clientes era vulnerable a los ataques. Las dos situaciones se resolvieron mediante el uso de la orquestación. Básicamente, el servidor tiene todos los problemas de manejar, los clientes son sólo entidades simples que son fáciles de manejar.

Por el lado microService extendida, las fases de comunicación de cliente a cliente son:

1) El cliente inicializa una llamada al servidor solicitando el resultado de una tarea en particular. Se genera un único contraseña para ser usado una vez. Esta contraseña se cifra con todo el paquete de datos enviado utilizando el algoritmo DES.

2) El servidor recibe la llamada, comprueba si hay un cliente disponible. Si existe, está llamando de vuelta, si lo hace No, no va a devolver nada, y la tarea será recuperado para su ejecución por el cliente que inicia la llamada. Recibe, además del paquete con instrucciones directas, paquetes adicionales (dependencias, etc.).

3) Si el servidor interroga a un cliente para un microService extendida, sino que también recibirá el único desechable contraseña (usada como la optimización de encriptación de cifrado) creado por el cliente original, que puede cifrar el resultado con esta clave.

4) Una vez que se recibe el resultado, el servidor sólo volverá a dirigir al primer cliente.

5) El cliente descifra el resultado con la contraseña desechable y continúa la ejecución.

3.2. Solicitud

Para la prueba y la evaluación de esta arquitectura, hemos implementado varias microservicios que llamamos en lo que queríamos hacer la salida al mismo tiempo.

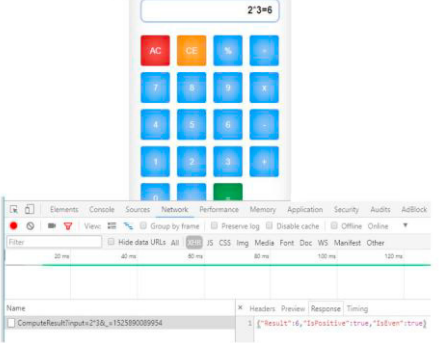

Fig. 2. La interfaz.

En la primera serie de experimentos se utilizó 3 microservicios como sigue: (1) un microService que realiza una operación matemática entre dos números (usando *LibraryMath), (*2) un microService que nos dice si un número es positivo-prim (*MasterOfNumbers)* y (3) un microService extendida que llamará a la primera microService cuando

recibe dos números, el resultado va a enviar a la segunda microService para averiguar información acerca de este número (*UniverseOfSuperMath).*

La Figura 2 muestra cómo se logra el cálculo matemático a través de los micro servicios presentados. A nivel de interfaz, sólo se muestra el resultado de la operación matemática, el resto de la información se puede ver en el resultado recibido desde el servidor después de la llamada AJAX puesto en marcha pulsando la tecla igual (el resultado es positivo y el resultado es positivo).

A continuación, vamos a ver una funcionalidad básica de la aplicación que se centra en *lo que sucede cuando hay una, dos o más clientes conectados?* En la figura 3, podemos ver cómo en los experimentos que hicimos empezamos a más clientes en el equipo local, utilizando diferentes puertos para cada uno de ellos.

ClientToken	Date	IP	Port	Function	Success
68ca8385-a	08:44:29	127.0.0.1	8390	UniverseOfMath.UniverseOfSuperMath	True
6f5972eb-e	08:44:28	127.0.0.1	8984	LibraryMasterOfNumbers.MasterOfNumbers_extendedService	True
5822f786-c	08:44:28	127.0.0.1	8827	ClassLibraryMath.LibraryMath_extendedService	True
6f5972eb-e	08:43:38	127.0.0.1	8984	LibraryMasterOfNumbers.MasterOfNumbers	True
5822f786-c	08:43:38	127.0.0.1	8827	ClassLibraryMath.LibraryMath	True
localhost	08:43:38			UniverseOfMath.UniverseOfSuperMath	True
localhost	08:42:53			LibraryMasterOfNumbers.MasterOfNumbers	True
5822f786-c	08:42:53	127.0.0.1	8827	ClassLibraryMath.LibraryMath	True
localhost	08:42:53			UniverseOfMath.UniverseOfSuperMath	True
localhost	08:38:21			ClassLibraryMath.LibraryMath	True

una)

ClientToken	Date	IP	Port	Function	Success
6f5972eb-e	08:47:14	127.0.0.1	8984	LibraryMasterOfNumbers.MasterOfNumbers	True
localhost	08:47:14			UniverseOfMath.UniverseOfSuperMath	True
localhost	08:47:14			ClassLibraryMath.LibraryMath	True
6f5972eb-e	08:46:26	127.0.0.1	8984	LibraryMasterOfNumbers.MasterOfNumbers	True
5822f786-c	08:46:25	127.0.0.1	8827	ClassLibraryMath.LibraryMath	False
localhost	08:46:25			ClassLibraryMath.LibraryMath	True
68ca8385-a	08:46:23	127.0.0.1	8390	UniverseOfMath.UniverseOfSuperMath	False
localhost	08:46:23			UniverseOfMath.UniverseOfSuperMath	True

si)

Fig. 3. El interfaz.

Contamos con 6 campos: *ClientToken* - un token único asociado a cada cliente (cuando la llamada es local y tiene el valor localhost); *Fecha* - el momento en que se presentó la solicitud; *IP y puerto* = el cliente ' s IP y el puerto a través del cual se hace la comunicación; *Función* - el nombre de la función llamada; *Éxito* - un mensaje verdadero o falso si la llamada fue o no finalizó con éxito. Por ejemplo, se observa que en la primera llamada (h: 08:38:21, ningún cliente se conecta al servidor, el proceso que está siendo ejecutado por el servidor). En la segunda llamada, nos damos cuenta el comportamiento dinámico del sistema, una de las tareas que se están ejecutando por uno de los clientes, mientras que los otros dos son ejecutados por el servidor. concretamente, *UniverserOfSuperMath* se llama (local - no hay ningún cliente disponible para esta tarea) que a su vez llama a otros dos microservicios, uno local y uno a través del cliente delegado de usar la instrucción específica, etc.

Tolerancia a fallos

Otra funcionalidad que tuvo en cuenta cuando construí esta arquitectura se relaciona con la tolerancia a fallos del sistema. Partiendo del escenario anterior, podemos ver lo que sucede si uno o más clientes quieren dejar el sistema.

En la Figura 3 a la derecha, la llamada a las 8:46 AM muestra este escenario. Los clientes de los puertos 8390 y 8827 tienen un problema local o de red, o simplemente cerraron la conexión con el servidor, y el servidor no se notifica a tiempo para eliminarlos de la lista. El servidor intentará ponerse en contacto con los clientes y lanzar comandos, pero si ellos no responden de manera oportuna, el servidor se hace cargo de sus tareas y devuelve el resultado requerido. Los clientes se les pide una vez más para la verificación por un tiempo, y si siguen sin responder, serán eliminados de la lista de clientes disponibles. La siguiente llamada (8:47) ya no innecesariamente consultar los clientes que ya no están disponibles, y las tareas que no son detectados por los clientes disponibles será ejecutado por el servidor.

Ventajas / desventajas de la solución ofrecida

Los beneficios de esta arquitectura son evidentes: los bajos costos de alojamiento, microservicios ofrecen en red distribuida, siendo dinámica y escalable automático (cuando los clientes también ofrecen la potencia de cálculo como su número aumenta, la potencia de cálculo de los aumentos del sistema).

Igualmente, las limitaciones deben ser enfatizados: cuando la curva de potencia de cálculo no se sigue la de los clientes. También tenemos una restricción de la capacidad para ejecutar esta aplicación en cualquier sistema operativo. Para esto, decidimos transformar la solución disponible en .NET en Java. Sin embargo, esta solución tiene algunos inconvenientes a la solución inicial (Java ofrece menor velocidad de procesamiento de datos y la transmisión de paquetes menos dinámico que tenemos en .NET). Utilizamos actualmente

esta solución porque .Net Core ofrecido por Microsoft para funcionar en múltiples plataformas todavía no está maduro y no ofrece todas las funcionalidades de la plataforma .NET estándar).

3.3. componentes cliente-servidor

3.3.1. Cliente

En esta arquitectura, el cliente es una aplicación de escritorio de WPF (Windows Presentation Foundation) especialmente construido para comunicarse con el servidor y ejecutar diversas tareas recibidas de él. Dado que la aplicación es un archivo ejecutable que no requiere instalación, el sistema operativo debe ejecutar .Net Framework. Básicamente, un microService web se comunicará con otra microService web.

Inicialmente, el cliente inicia un planificador de tareas en un hilo paralelo que cada minuto intentará notificar al servidor con su presencia. La tarea puede tener dos estados: (1) o bien tiene un trabajo para ejecutar (inicializar un paquete de código ya hecho) - en cuyo caso sólo se notifica al servidor con su presencia; (2) o requiere una inicialización con el servidor.

Inicialización con el servidor implica, en primer lugar, la elección arbitraria de un código y de puerto que el servidor se ejecutará que son enviados a ella por un protocolo de intercambio de claves de Diffie-Hellman (IKE). Una vez que se establece el enlace entre las dos entidades, el servidor notificará al cliente con el paquete de instrucciones para ser instalado. La función principal de un cliente va a recibir un paquete de instrucciones del servidor, cargarlo en la memoria, procesar la información recibida del servidor y luego devolver el resultado obtenido de la ejecución de ese paquete de instrucciones. El primer paso ejecutado por el cliente es ponerse en contacto con el servidor con el fin de recibir el paquete de instrucciones. Este paquete de instrucciones se presenta en forma de un archivo ZIP.

Antes de la extracción de este paquete, elimine el pre directorio vious con las instrucciones de la carpeta “proceso” (si existe), a continuación, extraiga el nuevo contenido de esta carpeta y cargarlo en la memoria. Esta carga en la memoria se ejecuta una vez, no importa cuántos golpes que el cliente va a tener. Esto es posible porque las tres propiedades se mantienen estáticos en la sesión: el montaje, *MethodInfo* y el tipo. La propiedad montaje almacena la referencia a la DLL cargado, la *MethodInfo* la propiedad tiene en cuenta el método llamado de la DLL, y el tipo se describe el tipo de la DLL. El archivo install.zip es el paquete de instrucción recibida desde el servidor que contiene (DLL, XMLs, imágenes, archivos de configuración, etc.) y todo el código compilado para ser ejecutado en el proceso futuro.

Esta etapa marca el comienzo de la relación entre el cliente y el servidor con el fin de ejecutar una tarea en particular. Una vez que el cliente se ha inicializado correctamente con una determinada tarea a ejecutar, el servidor sólo enviará el paquete de datos que necesita ser procesada de forma encriptada y esperará a la respuesta de forma encriptada.

Mediante la ejecución de la co de recibido del servidor, un “atados” sistema no se forma, el cliente es capaz de conectarse a bases de datos, para llamar a otras API, siendo un caso especial para llamar a otros clientes que ejecutan las mismas u otras instrucciones. Esta conexión se realiza en un sistema de orquestación, donde el servidor busca el próximo cliente disponible, es interrogado por el resultado, y su respuesta es redirigido de vuelta al cliente por el servidor. este micros ervicio orquestación se llama “ *ExtendedService* ” y la única diferencia en el nivel de cliente de esto es que hay una optimización de cifrado.

Un problema técnico encontrado era para reiniciar el cliente con otro paquete de instrucciones a ejecutar. Debido a que la carga de memoria es estático en un contexto especial (el servidor web), esto sólo era posible mediante el reinicio de todo el proceso para hacer frente a las DLL cargadas en la memoria. Para lograr esto, hemos creado eventos de Windows que se activa con la aplicación web que se ejecuta en una aplicación de escritorio. Esto es necesario porque se trata de dos contextos diferentes en dos hilos de ejecución diferentes.

3.3.2. Servidor

El micro-servicio integrado tiene una *ILibraryMath* interfaz que proporciona el *SimpleMath* método, y la implementación de la interfaz se realiza por el *LibraryMath* clase. los *LibraryMath* clase extiende la clase abstracta genérica *MicroCore* <I, O> que tiene dos parámetros de adaptación de la entrada y salida. La extensión de esta clase abstracta, la *ProcessTask* método debe ser implementado en todo el código que se ejecuta está escrito, y la función Ejecutar en la clase abstracta extendida se llama para ejecutar este código en el *SimpleMath* método. De esta manera, es posible definir interfaces y métodos sin verse limitados por cualquier nombre particular, pero haciendo pasar el código a través de la

clase abstracta vamos a obtener el control total del código que se puede distribuir a diferentes clientes. Dentro de esta clase podemos tener más funciones, bibliotecas importados, siendo ningún problema, siempre y cuando se agrupan en el mismo paquete.

El siguiente paso consistió en grabar en esta interfaz *SimpleInjector,* una biblioteca que facilita el despliegue de patrón de inyección de dependencia con los componentes libremente acoplados. Además de la grabación de clase intercalado en el contenedor del inyector simple para romper la dependencia entre los niveles de aplicación (inyección de dependencias del patrón), tenemos que registrar la clase en un recipiente de almacenamiento microService que luego se ampliará por la aplicación. Después de este paso, vamos a ser capaces de utilizar la función proporcionada por la interfaz con el propósito de crear.

Servicio1 implementa IService1 y extiende la *MicroCore* clase abstracta, y luego se ha registrado *MicroContainer.RegisterMicro* en ese contenedor. Vale la pena mencionar la existencia de APIs disponibles en *localohst / DynamicMicros / {} Servicio,* a través del cual los clientes se comunican con el servidor, las acciones importantes que se ponen a disposición por esta API: el cliente se conecta, el cliente notifica al servidor sobre su actividad, microservicios extendidos, y así sucesivamente. A continuación, vamos a introducir el *MicroCore* y *Microcontainer* clases, que juntas forman el marco de nuestra aplicación.

los *MicroCore* clase es una clase abstracta, genérica y es responsable de la llamada, el código de la *ProcessTask* método virtual. Esto se realiza mediante una llamada al método Run, que a su vez llama a un método público denominado *Administrador de tareas.* Mencionamos que el microService llamará a su vez también este método. Cuando el paquete postal se envía al cliente para ser cargado en la memoria y ejecutado, se envía con todos sus dependenc IES incluyendo esta clase a través del cual el cliente microService gestión se lleva a cabo. gestión de la ejecución consiste en de-serialización / serializar el paquete de datos a enviar, llamar al código mismo, llamando a otras API, etc.

Volviendo al lado del servidor, la gestión de la ejecución de código consta de los siguientes pasos:

1) Si se trata de una *ExtendedService* llama, el servidor será llamado por la respuesta.
2) Si hay un cliente disponible para la consulta, que será enviado a la misma para procesar el resultado; en el caso negativo, el servidor va a procesar los datos.
3) Consultamos el cliente para el procesamiento de datos.
4) Si el cliente es encuentran problemas, consultamos una vez más para confirmar la disponibilidad, pero enviamos el servidor de respuesta (para evitar tiempos muertos y tiempos de espera elevados).
5) Registramos la actividad en curso. los *Microcontainer* clase es el espacio de gestión de todo el microsistema integrado. Aquí, los clientes que se conectan a la aplicación (el servidor) conectar y aquí vienen las llamadas de funciones que amplían la *MicroCore* clase abstracta para la parte de "servicio extendido". Esta es una clase estática en la lista de tareas a ejecutar en los microservicios, la lista de clientes conectados y la lista de tareas del cliente tareas que se están ejecutando las tareas se almacenan en un diccionario.

En el inicio, se registrará la clase para ser integrado en un microService con *RegisterMicro.* Esto ocurrirá sólo una vez durante la inicialización. los *AddNewClien* método nos proporcionará el registro de un nuevo cliente, el intercambio de claves, el registro de la dirección IP del servidor y el puerto se ejecutará. El token recibida por el nuevo cliente se comprobará antes de insertarlo en la lista de clientes para verificar su singularidad. Una vez establecida la conexión con un cliente, el servidor va a llamar a la *InstallService* método que los paquetes de los datos, lo envía, y después de t que el cliente de respuesta, que se añadirá al diccionario para esa tarea. El tiempo de servicio que se asignará a cada cliente depende de la estrategia utilizada. En el funcionamiento de la *MicroCore* microService abstracta, llamada tanto en el servidor y el cliente (con *ExtendedService),* la consulta de los clientes disponibles para la tarea solicitada con el *GetNextClient* función. Esta operación será ejecutada con mucha frecuencia y su complejidad tendrá un impacto directo sobre el tiempo de respuesta de la aplicación. Es por eso que nuestro enfoque era escoge al azar un cliente. Esto se realiza de forma rápida y de nuestros experimentos asegura una distribución uniforme de llamadas.

Otra opción era implementar una lista circular, una solución que viene con el inconveniente de que, en el caso de un gran flujo de entrada de clientes / salida, la actualización de la lista circular requerirá más tiempo y la complejidad, las cosas que han intentado evitamos ellos. los *RecordClientError* método se llama cuando un cliente no responde a una consulta recibida. A raíz de la respuesta a este punto, se decidió retener o eliminar ese cliente. Excepcionalmente, los clientes se identifican por un código de identificador enviado por el cliente en la inicialización, y cada microService se identifica por un nombre de espacio de nombres y clase. La gestión de todos los recursos (clientes, códigos) se realiza a través de este bloque unitario que proporciona soporte para las operaciones requeridas.

En cuanto a la seguridad del sistema, se han tomado medidas para prevenir los ataques, intercepciones y garantizar la protección de datos. Todos los mensajes enviados entre el servidor y los clientes se cifran con la clave simétrica DES

algoritmo, y un intercambio de claves Diffie-Hellman entre el cliente y el servidor, que tiene lugar en la inicialización de un cliente. clientes disponibles y los programas en ejecución se almacenan en la memoria del servidor. Hemos elegido este solución, ya que representa la mejor opción en nuestra opinión, ya que proporciona una alta velocidad de acceso a los datos, la información puede cambiar con mucha frecuencia, y el área de memoria es muy difícil atacar por los atacantes cibernéticos.

3.4. el comportamiento dinámico del sistema microservicios

En primer lugar, todos los equipos que ejecute los clientes pueden estar situados en la misma red o en redes diferentes. Dos elementos están dirigidos: (a) el tiempo pasado con la transferencia de datos; y (b) la sobrecarga añadida por un sistema para gestionar los datos (por ejemplo, se encuentra cliente, encripta, desencripta, trata los errores, etc.). Principalmente, estábamos interesados en el comportamiento de nuestro sistema de LAN y WAN (Fig. 4).

Task name	Logs	Logs
ClassLibraryMath.LibraryMath	(22:38:03 ,193) (22:38:09 ,36) (22:38:12 ,27) (22:38:15 ,36) (22:38:18 ,37) (22:38:23 ,27)	(22:19:29 ,124) (22:19:35 ,129) (22:19:39 ,137) (22:19:44 ,148) (22:19:56 ,17) (22:23:25 ,45)
LibraryMasterOfNumbers.MasterOfNumbers	(22:38:03 ,58) (22:38:09 ,27) (22:38:12 ,21) (22:38:15 ,16) (22:38:19 ,27) (22:38:23 ,11)	(22:19:29 ,47) (22:19:35 ,41) (22:19:39 ,68) (22:19:44 ,38) (22:19:56 ,29) (22:23:25 ,50)
UniverseOfMath.UniverseOfSuperMath	(22:38:03 ,655) (22:38:09 ,112) (22:38:12 ,84) (22:38:15 ,101) (22:38:19 ,100) (22:38:23 ,72)	(22:19:29 ,534) (22:19:35 ,384) (22:19:39 ,299) (22:19:44 ,305) (22:19:56 ,147) (22:23:25 ,349)
WriteLogToDb.ClientWriteLog	(22:38:35 ,6405) (22:38:39 ,473) (22:38:42 ,465) (22:39:24 ,480) (22:39:26 ,476)	(22:24:21 ,6571) (22:24:24 ,486) (22:24:27 ,490) (22:27:00 ,517) (22:27:02 ,489)
PdfDiploma.GenerateDiploma	(22:38:49 ,1211) (22:38:54 ,1081) (22:38:56 ,452) (22:39:31 ,1394) (22:39:34 ,481)	(22:24:36 ,1404) (22:24:43 ,501) (22:25:11 ,544) (22:26:48 ,750) (22:26:52 ,514)

Fig. 4. grabación el sistema que ejecuta en la LAN (primera columna de troncos) y WAN (segunda columna de troncos).

El nombre de tareas columna contiene todos los registros realizados por la llamada del cliente para cada tarea y los registros de las columnas, la hora y la duración en ms para cada tarea de procesamiento (izquierda en la LAN y WAN en la derecha). Observamos que las tareas tienen un tiempo de respuesta más alta a la primera llamada, después de lo cual disminuye. Naturalmente, porque todas las cargas de memoria, retención de dirección, etc. se hacen generalmente en la primera llamada. Las tres primeras tareas son operaciones matemáticas simples que por lo general se ejecutan en una milésima de segundo, tiempo que también es necesario para nuestro sistema.

Para LAN, tenemos un promedio de 20-30 milisegundos por puesto de trabajo, lo que resulta en el cifrado, el registro y transferencia de la red (incluso si es local). Este modelo de comunicación LAN también se utiliza en la nube, donde se encuentran los equipos en la misma ubicación (Centros de Datos), y la comunicación entre ellos se lleva a cabo por la fibra óptica, la latencia en la red que tiene valores mínimos. Los resultados se muestran en la columna de la Fig. 4 troncos a la izquierda.

Para probar nuestra aplicación WAN, hemos configurado el router para dirigir la llamada desde el puerto 80 a: http://192.168.1.160/ (dirección de red), y con IIS (Internet Information Services) que ejecuta la aplicación, que sea accesible desde en cualquier lugar fuera de la red local. Con el fin de ejecutar la aplicación en el nivel de cliente, el derecho a utilizar se requirió 9000 puertos (puertos arbitrarios): los 8000. Los clientes se encuentran en cualquier lugar, la conexión se conformó con la IP pública con la API: https://api.ipify.org/. Los resultados se muestran en la Fig. La columna 4 logs a la derecha.

En los resultados presentados en la Fig. 4, los valores de la columna de la derecha log en comparación con los valores de la columna de registro de la izquierda son 16-17% más alto para las tres primeras tareas (no se comunican con otros microservicios) y ± 10% para microservicios que descargar documentos desde Internet o interactuar con una base de datos en un servidor determinado.

4. Evaluación

En este estudio hemos seguido el comportamiento del sistema, tanto en la LAN (conexión inalámbrica a través de 5 computadoras) y WAN (utilizando el espacio de nombres http://mihaidm.ddns.net/), la comparación de nuestro sistema a un sistema monolítico, estas operaciones siendo ejecutado en el mismo equipo (véase la Tabla 1).

Tabla 1. Evaluación del sistema para redes.

	Cálculo numérico (ms)	Escribir en BD (ms)	Generar PDF (ms)
localhost	1	4,458	15.449
LAN	25	4.408	16.415
PÁLIDO	54	4,826	29.309

Las pruebas se efectuaron sucesivamente en el mismo dispositivo con 5 clientes conectados a la red para las pruebas. Cada tarea se ejecutó 100 veces, la evaluación del número total de milisegundos en todas las llamadas.

En caso de cálculo numérico, se realiza el producto de dos números. Microservice no interactúa con otros microservicios, la cantidad de información transmitida en la red es pequeña y la complejidad se reduce al mínimo estrictamente para estudiar el tiempo dedicado a las tareas de servidor, cliente y gestión de la red. Si el cálculo es ejecutado por el servidor (localhost), compruebe primero si hay un cliente disponible, y dado que no hay ningún cliente conectado, el servidor procesa el resultado. Para el siguiente caso, la existencia de clientes de la LAN muestra la ejecución de la tarea, mientras que la creación de redes es muy rápido, y el lado de procesamiento es de cifrado / descifrado, la búsqueda de la respuesta del cliente. Para las 100 ejecuciones, el tiempo medio necesario para completar la operación fue de 25 ms, el valor prometedores considerando la relación flexibilidad / velocidad. En el caso de la WAN,

Otra de las tareas que hemos estado siguiendo escribía en una base de datos. Específicamente, una palabra que será writte n en la base de datos se recibe como un parámetro. Estamos interesados en la rapidez con la que el cliente se comunica con la base de datos, que se encuentra fuera del área local (para este estudio, la base de datos estaba alojado en https://www.my.gearhost.com/). Tenga en cuenta que los valores de tiempo de ejecución en LAN y localhost están cerca. En la WAN, la diferencia es notable, ya que no es gran parte de la
el tiempo añadido al procesamiento, el cliente y la gestión de datos, pero con la banda del cliente que se conecta a la base de datos de
insertar un valor.

Última tarea realizada en este estudio, fue la generación de un archivo PDF, nuestra atención se centró en evaluar el tiempo de transferencia de datos dentro del sistema. Para ello se descarga un archivo PDF a partir de https://www.pdf-archive.com/2018/05/14/diploma/ diploma.pdf que se carga en memoria. El sistema escribirá un nombre a una posición específica y devolver el resultado (en forma de vectores de bytes) de vuelta al servidor. Para localhost y LAN, la diferencia de casi 1.000 ms representa el tiempo necesario para el cifrado y el transporte local de los archivos PDF. Para WAN, el valor registrado es mayor debido a que el costo de la transferencia byte vector es muy alta.

5. conclusiones y trabajo futuro

La naturaleza genérica y abstracta de la arquitectura del sistema se presenta en este trabajo en el lado del servidor ha hecho que dificulta el diseño debido a que el mismo código es ejecutado por el servidor y el cliente. Podemos afirmar que la arquitectura actual es compacto, sencillo y fácil de entender y ampliar; el cliente puede ejecutar las tareas asignadas por el servidor, siendo el servidor del monolito y la interfaz del cliente.

La arquitectura propuesta permite la creación de nuevas microservicios de una manera muy sencilla, que se integran automáticamente en el sistema integrado. Los elementos innovadores de esta arquitectura son: se puede escalar muy fácilmente, cada nuevo cliente se le asigna una tarea por el servidor de acuerdo con la estrategia seguida (las tareas más caros, los más comunes, y una combinación de los dos previamente en la lista o pura y simplemente arbitrario). Básicamente, tenemos un monolito que tiene la flexibilidad de un sistema basado microService. El servidor se encarga de la distribución dinámica de tareas entre los clientes, proporcionando un escalado dinámico basado en una serie de parámetros (el número de llamadas a una tarea, su tiempo de ejecución, o combinaciones de los mismos).

Una de las direcciones futuras tiene en cuenta que este sistema puede integrarse con éxito en un sitio web o sistema API con un carácter aplicativo pronunciada. La arquitectura propuesta puede mejorarse y ampliarse en cualquier momento, con disponibilidad para múltiples plataformas (por ejemplo, teléfonos móviles).

Otra dirección futura que estamos mirando es considerado como extremadamente atractivo hoy en día es que el usuario proporciona la potencia de procesamiento a cambio de una remuneración (por ejemplo, el sistema Bitcoin), nuestra aplicación está desarrollando para ejecutar microservicios de algunos ordenadores.

Agradecimientos

Esta encuesta se publicó con el apoyo por el programa POC-A1-A1.2.3-G-2015, como parte de la PrivateSky proyecto (P_40_371 / 13 / 01.09.2016) y por el proyecto README “Inte aplicación ractivos e innovadoras para evaluar la legibilidad de los textos en idioma rumano y para la mejora de los usuarios estilos de escritura”, sin contraer. 114 / 09.15.2017, MySMIS 2014 código 119286.

referencias

[1] Dragoni, N., Giallorenzo, S., Lluch-Lafuente, AL, Mazzara, M., Montesi, F., Mustafin, R. (2017a) "microservicios: Ayer, Hoy, y Mañana ". Mazzara M., Meyer B. (eds.), *Presente y Ulterior Ingeniería de Software.* Saltador. [2] Mazzara, M., Khanda, K., Mustafin, R., Rivera, V., Safina, L. y Silitti, A. (2018) " Microservicios Ciencia e Ingeniería ". En p. Ciancarini, S. Litvinov, A. Messina, A., Sillitti, G. Succi (eds.) *Actas del 5º Congreso Internacional en Ingeniería de Software para aplicaciones de defensa, SEDA 2016,* Springer, 10-20.

[3] Dragoni, N., Lanese, I., Larsen, ST, Mazzara, M., Mustafin, R., y Safina, L. (2017B) " Microservicios: cómo hacer que su aplicación Escala ". En: Petrenko A., Voronkov A. (eds.) *Perspectivas de Informática del sistema. PSI 2017.* Lecture Notes in Computer Science, **10742.** Springer, Cham.

[4] Melis, A., Mirri, S., Prandi, C., Prandini, M., Salomoni, P., y Callegati, F. (2016) " Un caso de uso Microservice Arquitectura para las personas con discapacidades ". En el *2ª Conferencia Internacional sobre EAI objetos inteligentes y tecnologías para el bien social,* DOI: 10.1007 / 978-3-319-61949- 1_5.

[5] Zimmermann, O. (2017) " Microservicios Principios: enfoque ágil para el desarrollo y despliegue del servicio, Ciencias de la Computación - Investigación y Desarrollo ", **32 (3-4):** 301-310.

[6] Xia, C., Zhang, Y., Wang, L, Coleman, S., y Liu, Y. (2018) " sistema de robótica basada en la nube para el espacio inteligente Microservice ". En: *robótica y sistemas autónomos* **110,** DOI: 10.1016 / j.robot.2018.10.001. [7] Bogner, J., Fritzsch, J., Wagner, S., y Zimmermann, A. (2019) " Microservicios en la Industria: Miradas en torno a las tecnologías, características y Calidad del software ". UNA t la *2019 Conferencia Internacional IEEE sobre Talleres de arquitectura de software (ICSAW)* A: Hamburgo, Alemania. [8] Akentev, E., Tchitchigin, A., Safina, L., y Mzzara, M. (2017) " Verificado comprobador de tipos de lenguaje de programación Jolie ", https: // arXiv.org/pdf/1703.05186.pdf. [9] Černý, T., Donahoo, MJ, y Trnka, M. (2018) " comprensión contextual de la arquitectura microService: presente y futuro ". *ACM SIGAPP Informática Aplicada Revisión* **17 (4):** 29-45, DOI: 10.1145 / 3.183.628,3183631. [10] Larucces, X., Santamaria, I., Colomo-Palacios, R., y Ebert, C. (2018) " microservicios ". En: *IEEE Software,* **35/3:** 96-100. [11] Kalske, M. (2017) " La transformación de la arquitectura monolítica hacia la arquitectura microService ". M.Sc. Tesis, *Univ. de Helsinki.*

[12] Lenarduzzi, V., y Taibi, D. (2016) " MVP Explicación: Un estudio de mapeo sistemático en las definiciones de producto viable mínimo ". En el *Conferencia 42th Euromicro de Ingeniería del Software y Aplicaciones Avanzadas (SEAA),* 112-119. [13] Taibi, D., Lenarduzzi, V., Janes, A., Liukkunen, K., y Ahmad, MO (2017) " Comparando Requisitos de descomposición dentro del scrum, Scrum con Kanban, XP y los procesos de desarrollo de plátano ". En: Baumeister H., Lichter H., Riebisch M. (eds.) *Los procesos ágiles en ingeniería de software y la programación extrema.* Lecture Notes in Business Information Processing, **283.** Springer, Cham. [14] Gómez, A., Benelallam, A., y Tisi, M. (2015) " Persistencia modelo descentralizado para la computación distribuida ". En el *tercero BigMDE Taller,* L'Aquila, Italia.

[15] Kandave, KR (2018) " Computación de alto rendimiento en Azur". Nanette Ray (ed.), *AzureCAT, Microsoft Corporation.*

[16] Sreenivas, V., Sriharsha, S., y Narasimham, C. (2012) " Un modelo de nube para implementar SaaS ". En: *Materiales de Investigación Avanzada* **341-342,** Trans Tech Publications, Suiza, 499-503. [17] Badidi, E. (2013) " Un marco para el software como un servicio--Selección y Provisión ". En: *International Journal of Computer Networks & Comunicaciones (IJCNC),* **5 (3):** 189-200.

[18] Lynn, T., Rosati, P., Lejeune, A., y Emeakaroha, V. (2017) " Un examen preliminar de la Empresa sin servidor Cloud Computing (automático-as-a-Service) Plataformas ". En el 2017 *3ª Conferencia Internacional IEEE sobre Cloud Computing Tecnología y Ciencia,* 162-169. [19] Adzic, G. y Chatley, R. (2017) " Sin servidor Computing: impacto económico y arquitectónico ". A: *ESEC / FSE'17,* 4-8 de septiembre de 2017, los Paderborn, Alemania, ACM. [20] Diffie, W. y Hellman, M. (1976) "Nuevas direcciones en la criptografía". En: *IEEE Transactions on, Teoría de la Información,* **22 (6):** 644 - 654. [21] Kratzke, N. (2015) " Sobre microservicios, contenedores y su Impacto en el rendimiento de la red Underestimated ". En el *NUBE Comput. 2015,* **180** https://arxiv.org/abs/1710.04049.