(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 830 438**

(51) Int. CI.:

G06F 8/76 (2008.01)
G06F 9/455 (2008.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(86) Fecha de presentación y número de la solicitud internacional: **28.04.2017 PCT/IB2017/052504**

(87) Fecha y número de publicación internacional: **01.11.2018 WO18197928**

(96) Fecha de presentación y número de la solicitud europea: **28.04.2017 E 17723765 (8)**

(97) Fecha y número de publicación de la concesión europea: **02.09.2020 EP 3411785**

(54) Título: **Despliegue de microservicios en contenedores basado en aplicaciones heredadas monolíticas**

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
03.06.2021

(73) Titular/es:

LZLABS GMBH (100.0%)
Richtiarkade 16
8304 Wallisellen, CH

(72) Inventor/es:

JAEGER, JAN;
DURAND, DIDIER;
DITSCHEID, PIERRE-JEAN y
VUATTOUX, JEAN-LUC

(74) Agente/Representante:

ELZABURU, S.L.P

DESCRIPCIÓN

Despliegue de microservicios en contenedores basado en aplicaciones heredadas monolíticas

Campo técnico

La presente invención se refiere a técnicas y sistemas para dividir aplicaciones heredadas monolíticas para su despliegue como microservicios que se ejecutan en un entorno operativo en contenedores, escalable y flexible.

Antecedentes

En los entornos informáticos de ordenador central heredados, es común encontrar aplicaciones monolíticas que incluyen miles e incluso decenas de miles de programas individuales que se ejecutan juntos en una estructura muy monolítica en un solo entorno operativo. Esta estructura monolítica de programas puede representar inversiones sustanciales de tiempo y recursos en el desarrollo de su código subyacente (hasta varios miles de personas por año), y la naturaleza interdependiente de los programas de software hace que traducir o migrar el código desde un entorno informático sea muy difícil.

Los archivos de programa heredados pueden ser compilados, ensamblados y vinculados con la restricción de ejecutarse solo en un procesador de una arquitectura y un conjunto de instrucciones específicos, a menudo denominados parte de un sistema heredado o plataforma heredada.

La figura 1A representa los elementos de una plataforma heredada (100) que utiliza virtualización con hipervisor. El hardware (10) del sistema puede incluir, por ejemplo, un ordenador central que ejecuta un hipervisor (30), a menudo como un monitor de máquina virtual (z/VM), para proporcionar un conjunto de máquinas virtuales (70) completamente aisladas, cada una con su propio sistema operativo (SO – Operating System, en inglés) invitado, en el que normalmente se ejecutan los programas. El hipervisor (30) proporciona una plataforma de gestión que divide los recursos de la máquina central en el conjunto de máquinas virtuales o invitadas (70) que son operables de manera independiente dentro del sistema heredado. Un sistema operativo invitado (40) o varios sistemas operativos invitados (40) están instalados en las máquinas virtuales. Un conjunto de binarios y de programas de librería (50) y una o más aplicaciones (60) se ejecutan en una máquina virtual determinada. Al igual que una máquina física, la máquina virtual tiene información de estado asociada, se puede realizar una copia de seguridad o puede ser restaurada, y se le pueden asignar recursos de sistema dedicados. La puesta en marcha y el desmontaje de una máquina virtual en un sistema con hipervisor requiere una sobrecarga considerable y, por esta razón, cuando se establecen, las máquinas virtuales suelen persistir durante tiempos de ejecución considerables.

La figura 1B representa un ejemplo de un sistema de gestión de contenedores (110). El hardware (15) del sistema de contenedores puede ser un servidor físico o una agrupación de servidores físicos, que pueden ser, por ejemplo, ordenadores basados en X86. El núcleo del sistema operativo central (25) del sistema, tal como Linux, es compartido por la plataforma, y un conjunto de contenedores (75) están habilitados a través de un sistema de gestión (35) de contenedores tal como Docker. En particular, la funcionalidad de espacio de nombre y de grupo de control (cgroup, en inglés) del núcleo de Linux se puede utilizar para la operación en contenedores. Los sistemas de gestión de contenedores pueden ser proporcionados como empaquetamientos alrededor de las funcionalidades del núcleo y permitir la gestión de contenedores, tal como el despliegue.

Se pueden utilizar otros sistemas de gestión de contenedores tales como Amazon ACS, Azure Container Service, Cloud Foundry Diego, CoreOS Fleet, Docker Swarm, Google Container Engine o Mesosphere Marathon, u otro sistema de gestión y orquestación de contenedores. El sistema de gestión (35) de contenedores y un conjunto de librerías compartidas del sistema operativo (85) proporcionan una plataforma en la que se puede ejecutar el conjunto de contenedores (75). Por ejemplo, algunas librerías del sistema operativo (35) de bajo nivel, tales como las que se utilizan para las funciones básicas de entrada/salida (E/S) de archivos, pueden ser compartidas por todos los contenedores a través del núcleo del sistema operativo o del sistema de gestión de contenedores, en lugar de residir en contenedores individuales.

Como en el caso de la máquina virtual, un conjunto de binarios y programas de librería (55) y una o más aplicaciones (65) se ejecutan en un conjunto de contenedores (75). A modo de ejemplo, una librería que proporcione servicios de acceso web, tal como el protocolo http, puede que solo sea necesaria en algunas aplicaciones y no en otras, por lo que se incluiría en los programas de la librería (55) cuando se requiera para un servicio de aplicación específico, pero se omitiría de los programas de la librería (55) de un contenedor solo con aplicaciones que nunca utilizan un servicio de acceso web.

En comparación con una máquina virtual, un contenedor es una construcción relativamente ligera y no se carga con la sobrecarga de su propio sistema operativo completo y toda la información de estado asociada con una máquina física o virtual. En consecuencia, la puesta en marcha y el desmontaje de un contenedor requiere poca sobrecarga, lo que hace que el despliegue y la terminación de contenedores sea una técnica eficaz para la actualización de aplicaciones, el equilibrio dinámico de carga y la asignación de recursos dentro de una agrupación.

En particular, las máquinas virtuales tienen su propio sistema operativo, sistema de archivos, procesador o

procesadores, adaptadores de red y volúmenes de almacenamiento asociados. El hecho de que ejecuten un sistema operativo invitado sobre un hipervisor convierte a las máquinas virtuales en un proceso pesado, con la sobrecarga de ejecutar dos sistemas operativos (hipervisor + sistema operativo invitado) uno encima del otro, que no pueden ser iniciados y terminados fácilmente para adaptarse a la cambiante demanda de servicios de aplicaciones. Los contenedores, por otro lado, comparten funciones centrales del sistema operativo a través del acceso directo al núcleo y a otros recursos físicos, incluidos los volúmenes de almacenamiento. Los volúmenes de almacenamiento suelen residir en unidades de disco fijas, pero también pueden residir en otro almacenamiento masivo, incluidas unidades flash, cintas u otros medios de almacenamiento fijos o extraíbles. Aunque el comportamiento de diferentes contenedores puede diferir en función de los programas binarios y de librería que se incorporan en la imagen cargada en esos contenedores particulares, la utilización de servicios de sistema operativo compartido reduce significativamente la sobrecarga asociada con cada instancia individual de un contenedor. Por esta razón, los contenedores son ligeros, en relación con las máquinas virtuales, lo que hace más factible la creación de instancias y la terminación de contenedores en respuesta a las demandas de las aplicaciones. De hecho, en el caso de, por ejemplo, el sistema de gestión de contenedores de Kubernetes que ejecuta Docker, un contenedor puede ser iniciado en una fracción de segundo. Por esa razón, los grandes despliegues pueden iniciar y terminar varios miles de esos contenedores cada segundo.

Los sistemas de gestión de contenedores también pueden incluir módulos. Un pod es una unidad de implementación en un sistema de contenedores que incluye uno o más contenedores que se implementan juntos en el mismo ordenador central o agrupación. En algunos sistemas de gestión de contenedores, tal como Kubernetes, los contenedores de un pod comparten el mismo espacio de nombre de red y espacio de puerto. Además, los volúmenes compartidos de almacenamiento que se adjuntan al pod pueden ser montados en uno o más de los contenedores del pod.

Una distribución estándar de Linux incluye decenas (incluso cientos) de miles de archivos individuales y, dependiendo de la aplicación para la que se utilice dicho sistema, puede ser combinada con miles de paquetes de sistema adicionales que agregan funcionalidad a la plataforma. Ejemplos de dichos paquetes incluyen el servidor web Apache, la máquina virtual Java, PostgreSQL u otros paquetes para proporcionar soporte para bases de datos o idiomas y similares. Estos paquetes incluyen código de programa y metadatos que describen los paquetes y las dependencias entre paquetes y otras librerías. Las librerías compartidas pueden ser utilizadas por paquetes vinculados dinámicamente para proporcionar una funcionalidad tremenda, pero pueden aumentar en gran medida la huella de la imagen de Linux y la complejidad de la gestión del sistema. Una instancia mínima de Linux que incorpore muy pocos paquetes puede ocupar solo unos pocos megabytes de memoria. Por otro lado, una instalación grande con muchos paquetes utilizados para soportar, por ejemplo, un servidor web de aplicaciones a gran escala con servicios avanzados de bases de datos, puede ocupar cientos de megabytes de almacenamiento, o incluso más. La gestión de plataformas basadas en Linux a menudo incluye la utilización de software de gestión de paquetes para gestionar las dependencias entre paquetes y librerías y las actualizaciones recurrentes de esas librerías y paquetes. Una imagen grande que atiende a múltiples objetivos a la vez es más compleja de gestionar que una simple.

Los microservicios son, normalmente, servicios pequeños y autónomos que pueden colaborar estrechamente para proporcionar la funcionalidad de una aplicación. La naturaleza autónoma de los microservicios permite que se implementen independientemente unos de otros como servicios aislados, que se pueden comunicar con otros servicios a través de llamadas de red. Un conjunto de microservicios o microservicios estrechamente relacionados que, en su funcionamiento, comparten el acceso a un volumen común, pueden ser desplegados dentro del mismo pod. Una arquitectura de microservicios ofrece importantes ventajas de capacidad de gestión, disponibilidad, escalabilidad y despliegue en sistemas en agrupación. Sin embargo, la naturaleza monolítica de muchas aplicaciones heredadas hace que traducir dichas aplicaciones monolíticas en conjuntos de microservicios mínimamente interdependientes sea una tarea difícil y que requiere mucha intervención manual. Para complicar aún más el problema, las aplicaciones monolíticas heredadas escritas en Cobol y compiladas para ser ejecutadas en arquitecturas heredadas tales como MVS o z/OS con sus API propietarias, en general, no pueden ser exportadas desde la arquitectura heredada y ejecutadas en Linux o en otro sistema operativo o agrupación, especialmente cuando están basadas en servidores x86, debido a diferencias en conjuntos de instrucciones y API.

De manera más general, los sistemas que traducen el código de la aplicación de un entorno operativo a otro, ya sea mediante emulación, compilación cruzada, transcodificación o un enfoque híbrido, pueden ser desplegadas para permitir la ejecución de un programa heredado compilado para que se ejecute en un sistema operativo invitado mediante una arquitectura subyacente diferente. Sin embargo, estos sistemas tienden a ser grandes programas que no se escalan fácilmente, lo que es particularmente problemático en el caso de ejecutar aplicaciones que realizan grandes volúmenes de transacción. Además, los sistemas de emulación o transcodificación se prestan a ser aplicaciones monolíticas porque, para ser útiles, el emulador o transcodificador debe ser capaz de ejecutar un subconjunto desconocido de las posibles instrucciones del entorno heredado en el entorno invitado.

Una técnica para migrar y ejecutar aplicaciones en un entorno operativo diferente se proporciona en el documento US 8.458.651 B2.

Compendio

La invención se expone en la reivindicación 1 del sistema independiente y en la reivindicación 10 del método

correspondiente.

La presente invención proporciona realizaciones de un sistema escalable basado en contenedores implementado en instrucciones de ordenador almacenadas en un medio no transitorio. El sistema incluye un depósito de códigos fuente que contiene el código fuente de una aplicación heredada monolítica que contiene una pluralidad de programas ejecutables en un entorno informático heredado para realizar una pluralidad de transacciones. El sistema también incluye un analizador de código fuente, operable para analizar el código fuente e identificar, para cada transacción en la pluralidad de transacciones, un vector de definición de transacción que identifica cada programa potencialmente llamado durante la transacción, para crear una pluralidad de vectores de definición de transacción. El sistema también incluye un depósito de definición de estado de transacción que puede ser operado para almacenar la pluralidad de vectores de definición de transacción. El sistema también incluye un analizador de registro de actividad puede ser operado para crear un depósito de definición dinámica que identifica qué programas son realmente utilizados por la aplicación heredada monolítica en la realización en, como mínimo, un subconjunto de la pluralidad de transacciones. El sistema también incluye un optimizador de definición de microservicio operable para comparar la pluralidad de vectores de definición de transacciones con el depósito de definición dinámica y eliminar programas no utilizados de los vectores de definición de transacciones para crear una pluralidad de vectores de definición de microservicios que definen una pluralidad de microservicios. El sistema también incluye un generador de imágenes de microservicio operable para, para cada vector de definición de microservicio de entre la pluralidad de vectores de definición de microservicio, localizar para cada programa identificado por el vector de definición de microservicio binarios de código fuente compilados para ser ejecutados en el entorno informático heredado para formar una pluralidad de imágenes de microservicio correspondientes a los vectores de definición de microservicio. El sistema también incluye un depósito de imágenes de microservicio operable para almacenar la pluralidad de imágenes de microservicio. El sistema también incluye un depósito de componentes complementarios operables para almacenar un conjunto de imágenes de binarios de elementos de emulador de un emulador heredado que, en conjunto, son menos que un emulador heredado completo, correspondiendo dichas imágenes a una pluralidad de funciones o conjuntos de funciones de dicho entorno informático heredado, y siendo dichas imágenes ejecutables en un entorno informático distinto caracterizado por un conjunto de instrucciones distinto del conjunto de instrucciones del entorno heredado. El sistema también incluye un generador de contenedor puede ser operado para formar una imagen de contenedor para cada microservicio o un conjunto de microservicios en la pluralidad de microservicios utilizando la imagen de microservicio correspondiente o imágenes del depósito de imágenes de microservicio y utilizando archivos de imagen del depósito de componentes complementarios para los elementos del emulador heredado correspondientes a funciones o conjuntos de funciones empleados por el microservicio o conjunto de microservicios cuando son ejecutados, según se identifican mediante firmas de llamadas en los binarios en el microservicio o conjunto de microservicios, para crear una pluralidad de imágenes de contenedor. El sistema también incluye un depósito de imágenes de contenedor puede ser operado para almacenar la pluralidad de imágenes de contenedores ejecutables en el entorno informático distinto. El sistema también incluye un sistema de gestión de contenedor puede ser operado para crear, como mínimo, un contenedor para su ejecución en el entorno informático distinto y para ejecutar, como mínimo, un microservicio almacenado en el depósito de imágenes del contenedor en el, como mínimo, un contenedor.

De acuerdo con otras realizaciones, todas las cuales pueden ser combinadas con el sistema citado anteriormente y entre sí y con el sistema citado anteriormente en cualquier combinación, a menos que sean claramente excluyentes entre sí, la invención también proporciona:

i) el analizador de registro de actividad puede ser operado para crear una pluralidad de vectores de definición de transacciones dinámicas que corresponden, como mínimo, a una parte de la pluralidad de vectores de definición de transacciones, y en el que el optimizador de definición de microservicios compara cada vector de definición de transacción dinámica con cada vector de definición de transacción correspondiente para crear la pluralidad de vectores de definición de microservicios;

ii) el analizador de registro de actividad utiliza registros de actividad heredados de la aplicación heredada monolítica generados al ejecutar la aplicación heredada monolítica en el entorno informático heredado;

iii) el analizador de registro de actividad utiliza un emulador para ejecutar la aplicación heredada monolítica para generar archivos de registro y determinar qué programas utiliza la aplicación heredada monolítica durante la ejecución de transacciones;

iv) el analizador de código fuente puede ser utilizado para utilizar información del analizador de registro de actividades para identificar los vectores de definición de transacciones;

v) el analizador de código fuente puede ser operado, además, para crear una pluralidad de tablas de traducción;

vi) el optimizador de definición de microservicio es operativo para optimizar aún más los vectores de definición de microservicio;

vii) el optimizador de definición de microservicio puede ser operado para optimizar aún más los vectores de definición de microservicio creando vectores de definición de microservicio adicionales que contienen programas compartidos por más de una transacción en la pluralidad de transacciones;

viii) que comprende, además, un depósito de binarios operable para almacenar el código fuente compilado que contiene binarios compilados para ser ejecutados en el entorno informático heredado;

ix) el código fuente compilado en el depósito de binarios es compilado a partir del código fuente en el depósito de códigos fuente en archivos binarios;

x) el entorno informático heredado incluye un sistema informático de almacenamiento virtual múltiple (MVS – Múltiple Virtual Storage, en inglés) o z/OS;

xi) el depósito de componentes complementarios puede ser operado, además, para almacenar una pluralidad de imágenes de paquetes de software del sistema operativo utilizados por el emulador heredado, y en el que el generador de contenedores también coloca imágenes de cualquier paquete de software utilizado por un elemento particular del emulador heredado en una imagen de contenedor que contiene el elemento particular del emulador heredado.

xii) el generador de contenedores es además operable para sustituir las firmas de llamadas en los binarios en el microservicio o conjunto de microservicios por instrucciones para llamadas operables en el emulador heredado;

xiii) el sistema de gestión de contenedores puede ser operado para crear una pluralidad de contenedores;

xiv) se instancia un conjunto de imágenes complementarias en un contenedor separado dentro de un pod común;

xv) se activan más de una copia de, como mínimo, una imagen de contenedor en más de un contenedor separado;

xvi) el sistema de gestión de contenedores puede ser operado para variar el número de contenedores en la pluralidad de contenedores;

xvii) el sistema de gestión de contenedores puede ser utilizado para asignar diversos recursos a contenedores separados;

xviii) el sistema de gestión de contenedores puede ser operado para utilizar información del analizador de registro de actividad para determinar cómo colocar el número de copias de, como mínimo, una imagen de contenedor en más de un contenedor separado, para determinar el número de contenedores en la pluralidad de contenedores, y/o para determinar recursos para asignar a contenedores separados;

xix) el sistema de gestión de contenedores puede ser utilizado para utilizar la información a partir de la utilización del sistema escalable basado en contenedores para determinar cómo colocar la cantidad de copias de, como mínimo, una imagen de contenedor en más de un contenedor separado, para determinar la cantidad de contenedores en la pluralidad de contenedores y/o para determinar recursos para asignar a contenedores separados;

xx) el analizador de código fuente puede ser operado, además, para crear una o más bases de datos secundarias o grupos de bases de datos secundarias a partir de una base de datos de la aplicación monolítica heredada;

xxi) el generador de contenedores puede ser operado para colocar una o más bases de datos secundarias o grupos de bases de datos secundarias en uno o más contenedores; y

xxii) cuando se cambia el código fuente, el sistema basado en contenedores es operativo para actualizar automáticamente, como mínimo, una imagen de microservicio, como mínimo, una imagen de contenedor y como mínimo, un contenedor para contener un binario actualizado en base al cambio de código fuente.

La presente invención proporciona, además, realizaciones de un método para crear y operar un sistema escalable basado en contenedores. El método incluye analizar una aplicación heredada monolítica ejecutable en un entorno informático heredado y dividir sus archivos de programa para crear una pluralidad de vectores de definición de transacciones correspondientes a una pluralidad de transacciones ejecutables por parte de la aplicación heredada monolítica e identificar, para cada transacción, todos los programas llamados por esa transacción. El método incluye, además, almacenar la pluralidad de vectores de definición de transacciones en un depósito de estado de transacciones. El método incluye, además, para, como mínimo, una parte de la pluralidad de transacciones, la creación de un depósito de definición dinámica determinando qué programas se utilizan realmente cuando la transacción la realiza la aplicación heredada monolítica. El método incluye, además, comparar la pluralidad de vectores de definición de transacciones con el depósito de definición dinámica, y eliminar programas no utilizados en una transacción de su correspondiente vector de definición de transacciones para crear una pluralidad de vectores de definición de microservicios. El método incluye, además, para cada vector de definición de microservicio de entre la pluralidad de vectores de microservicio, situar el código fuente compilado correspondiente que contiene binarios compilados para ser ejecutado en el entorno informático heredado y crear una imagen de microservicio que contiene el código fuente compilado correspondiente para formar una pluralidad de imágenes de microservicio. El método incluye, además, almacenar la pluralidad de imágenes de microservicios en un depósito de imágenes de microservicios. El método incluye, además, almacenar, en un depósito de componentes complementarios, imágenes de una pluralidad de elementos de un emulador heredado operable para ejecutar programas en un entorno informático diferente al del entorno informático heredado, los elementos del emulador heredado correspondientes a una pluralidad de funciones o conjuntos de funciones de la aplicación monolítica heredada. El método incluye, además, formar una imagen de

contenedor para cada microservicio o un conjunto de microservicios en la pluralidad de microservicios utilizando la imagen de microservicio correspondiente o imágenes del depósito de imágenes de microservicio y utilizando archivos de imagen del depósito de componentes complementarios para los elementos del emulador heredado correspondiente a funciones o conjuntos de funciones empleadas por el microservicio o conjunto de microservicios cuando se ejecutan, identificadas por firmas de llamadas en los binarios en el microservicio o conjunto de microservicios, para crear una pluralidad de imágenes de contenedor. El método incluye, además, almacenar las imágenes del contenedor en un depósito de imágenes del contenedor. El método incluye, además, crear, como mínimo, un contenedor en el entorno informático diferente utilizando un sistema de gestión de contenedores y almacenar, como mínimo, una imagen del contenedor en el contenedor en una forma ejecutable en el entorno informático diferente.

El método incluye, además, ejecutar el microservicio o el conjunto de microservicios en el contenedor.

De acuerdo con otras realizaciones, todas las cuales pueden ser combinadas con el método citado anteriormente y entre sí y con el método citado anteriormente en cualquier combinación, a menos que sean claramente excluyentes entre sí, la invención también proporciona:

i) crear una pluralidad de vectores de definición de transacciones dinámicas que correspondan, como mínimo, a una parte de la pluralidad de vectores de definición de transacciones utilizando el analizador de registro de actividad y comparar cada vector de definición de transacciones dinámicas con cada vector de definición de transacciones correspondiente para crear la pluralidad de vectores de definición de microservicios utilizando el optimizador de definición de microservicios;

ii)que comprende el analizador de registros de actividad que utiliza registros de actividades heredados de la aplicación heredada monolítica generada al ejecutar la aplicación heredada monolítica en el entorno informático heredado;

iii) que comprende el analizador de registro de actividades que utiliza un emulador para ejecutar la aplicación heredada monolítica para generar archivos de registro y determinar qué programas utiliza la aplicación heredada monolítica durante la ejecución de transacciones;

iv) que comprende el analizador de código fuente que utiliza la información del analizador de registro de actividad para identificar los vectores de definición de transacciones;

v) crear una pluralidad de tablas de traducción utilizando el analizador de código fuente;

vi) optimizar aún más los vectores de definición de microservicio utilizando el optimizador de definición de microservicio;

vii) optimizar aún más los vectores de definición de microservicio utilizando el optimizador de definición de microservicio creando vectores de definición de microservicio adicionales que contienen programas compartidos por más de una transacción en la pluralidad de transacciones;

viii) almacenar el código fuente compilado que contiene binarios compilados para ser ejecutado en el entorno informático heredado en un depósito de binarios;

ix) compilar el código fuente en el depósito de binarios a partir del código fuente en el depósito de código fuente en archivos binarios;

x) el entorno informático heredado incluye un sistema informático de almacenamiento virtual múltiple (MVS) o z/OS.

xi) el depósito de componentes complementarios almacena una pluralidad de imágenes de paquetes de software del sistema operativo utilizados por el emulador heredado, y el generador de contenedores también coloca imágenes de cualquier paquete de software utilizado por un elemento particular del emulador heredado en una imagen de contenedor particular que contiene el elemento del emulador heredado.

xii) el generador de contenedores sustituye las firmas de llamadas en los binarios en el microservicio o conjunto de microservicios por instrucciones para llamadas operables en el emulador heredado;

xiii) clasificar una pluralidad de contenedores utilizando el sistema de gestión de contenedores;

ix) crear una instancia de un conjunto de imágenes complementarias en un contenedor separado dentro de un pod común;

x) activar más de una copia de, como mínimo, una imagen de contenedor en más de un contenedor separado;

xi) el sistema de gestión de contenedores varía el número de contenedores en la pluralidad de contenedores;

xii) el sistema de gestión de contenedores asigna diversos recursos a contenedores separados;

xiii) el sistema de gestión de contenedores utiliza la información del analizador de registro de actividades para determinar cómo colocar el número de copias de, como mínimo, una imagen de contenedor en más de un contenedor separado, para determinar el número de contenedores en la pluralidad de contenedores, y/o para determinar recursos

para asignar a contenedores separados.

xiv) el sistema de gestión de contenedores utiliza la información de la utilización del sistema escalable basado en contenedores para determinar cómo colocar el número de copias de, como mínimo, una imagen de contenedor en más de un contenedor separado, para determinar el número de contenedores en la pluralidad de contenedores y/o para determinar recursos para asignar a contenedores separados.

xv) el analizador de código fuente crea una o más bases de datos secundarias o grupos de bases de datos secundarias a partir de una base de datos de la aplicación monolítica heredada.

xvi) el generador de contenedores coloca una o más bases de datos secundarias o grupos de bases de datos secundarias en uno o más contenedores.

xvii) cuando se cambia el código fuente, actualizar automáticamente, como mínimo, una imagen de microservicio, como mínimo, una imagen de contenedor y, como mínimo, un contenedor para contener un binario actualizado en base al cambio de código fuente.

Breve descripción de los dibujos

Para una comprensión más completa de varias realizaciones de la presente invención y de sus características y ventajas, a continuación, se hace referencia a la siguiente descripción, tomada junto con los dibujos adjuntos, en los que:

la figura 1A es un diagrama esquemático de un entorno de máquina virtual basado en hipervisor de la técnica anterior.

la figura 1B es un diagrama esquemático de un entorno virtualizado basado en contenedores que puede ser modificado y utilizado junto con la presente invención.

la figura 2A es un diagrama esquemático de un conjunto de vectores de programa correspondientes a las transacciones de una aplicación.

la figura 2B es un diagrama esquemático de un conjunto de vectores de programa optimizados correspondientes a las transacciones de una aplicación.

la figura 3 es una descripción de los componentes de un sistema escalable basado en contenedores para la división de una aplicación heredada monolítica en microservicios.

la figura 4 es una descripción de los componentes de los árboles de llamadas para dos transacciones en una aplicación heredada monolítica.

la figura 5 es una representación de árboles de llamadas para las mismas dos transacciones de la figura 4 implementados como microservicios en un entorno escalable basado en contenedores.

la figura 6 es un diagrama de flujo que describe las etapas de un método para analizar una aplicación heredada monolítica para implementar microservicios en un entorno escalable basado en contenedores.

Descripción detallada

De acuerdo con un aspecto de la invención, se propone un sistema escalable basado en contenedores que puede dividir automáticamente una aplicación heredada monolítica en un conjunto de microservicios, y desplegar dichos microservicios, con elementos apropiados de un emulador heredado, en contenedores.

Los procesadores que tienen diferentes arquitecturas soportan diferentes conjuntos de instrucciones con diferentes representaciones binarias, con el resultado de que un programa ejecutable que incluye instrucciones de máquina de un conjunto de instrucciones (a menudo denominado “binario” o “imagen de binario”), en general, no se ejecutará en un procesador que tiene una arquitectura diferente y un conjunto de instrucciones correspondiente diferente. En consecuencia, una aplicación heredada monolítica diseñada para ser ejecutada en un procesador heredado con una arquitectura específica utilizando un conjunto de instrucciones de máquina específico en un entorno informático heredado, tal como un entorno informático de ordenador central heredado que incluye el procesador heredado, no puede ser ejecutada fácilmente en un tipo diferente de procesador en un entorno informático diferente. En particular, los sistemas escalables basados en contenedores, descritos en este documento, operan utilizando un procesador distinto, un conjunto de instrucciones distinto y un entorno informático distinto del entorno informático heredado en el que las aplicaciones heredadas monolíticas están diseñadas para ser ejecutadas. Por tanto, una aplicación heredada monolítica no se ejecutaría en el entorno informático distinto del sistema escalable basado en contenedores sin la modificación de la aplicación heredada monolítica y/o del entorno informático distinto, tales como los descritos en el presente documento.

Normalmente, para ejecutar la aplicación heredada monolítica en un entorno informático distinto que contiene un procesador distinto, la aplicación heredada monolítica se vuelve a compilar utilizando un compilador diseñado para la arquitectura distinta, sus instrucciones son transcodificadas para ser ejecutadas en la arquitectura distinta, o la

aplicación heredada monolítica es ejecutada en un traductor de arquitectura heredada (en adelante, emulador de aplicaciones heredadas), que puede ejecutar el programa ejecutable compilado para el entorno informático heredado en un entorno informático distinto que tiene una arquitectura distinta. Esto solo es posible cuando existe un compilador adecuado que puede compilar el código fuente heredado en el entorno informático distinto, o existe un transcodificador o emulador heredado adecuado.

En consecuencia, el sistema escalable basado en contenedores de la presente invención incluye, como mínimo, un elemento de un emulador heredado. Sin embargo, el sistema escalable basado en contenedores optimiza la utilización del emulador heredado colocando elementos del emulador, tales como imágenes de binarios de componentes funcionales, del emulador heredado en contenedores solo cuando los microservicios utilizan esos elementos, en lugar de requerir una imagen del emulador heredado completo en cada contenedor para realizar todas las tareas ejecutables por la aplicación heredada monolítica. Los elementos del emulador independientes soportan diferentes subconjuntos de funciones de las aplicaciones heredadas monolíticas.

Asimismo, un emulador heredado utiliza, normalmente, varias funcionalidades proporcionadas por un sistema operativo, tal como una funcionalidad de entrada/salida. En lugar de colocar una imagen de todo el sistema operativo en cada contenedor, el sistema escalable basado en contenedores también optimiza la utilización del sistema operativo colocando elementos del sistema operativo, tales como imágenes de binarios de componentes funcionales del sistema operativo, en un contenedor con microservicios y elementos de emuladores que utilizan eficazmente esos elementos del sistema operativo. Los elementos independientes del sistema operativo soportan diferentes subconjuntos de las funciones del emulador heredado y las funciones de aplicación heredadas monolíticas relacionadas.

El sistema escalable basado en contenedores puede identificar transacciones individuales que pueden ser realizadas utilizando la aplicación heredada monolítica, tales como crear un registro, realizar un pedido, realizar una consulta, etc. El sistema escalable basado en contenedores identifica los programas incluidos en cada transacción individual. Finalmente, el sistema escalable basado en contenedores crea microservicios que pueden ser utilizados o combinados para realizar la misma transacción fuera de la aplicación heredada monolítica. En algunos casos, los programas individuales que componen una transacción de la aplicación heredada monolítica pueden estar situados en microservicios distintos. En otros casos, un microservicio puede contener más de un programa de la aplicación heredada monolítica. Además, debido a que los microservicios pueden agrupar programas de cualquier manera para realizar transacciones de manera eficiente desde la aplicación monolítica heredada, cualquier programa de la aplicación monolítica heredada puede estar situado en un solo microservicio del sistema escalable basado en contenedores, o puede estar situado en múltiples microservicios distintos del sistema escalable basado en contenedores.

Un microservicio en una sola imagen de contenedor se puede implementar en múltiples instancias paralelas, normalmente en contenedores separados, a través de un sistema escalable basado en contenedores. Un contenedor puede incluir más de un microservicio, así como otra información, según sea necesario, para permitir que el o los microservicios se ejecuten y funcionen. Los microservicios se pueden estructurar preferiblemente de manera que sean mínimamente interdependientes y/o para minimizar el número de microservicios que requieren cambios cuando se actualizan los programas. La imagen del contenedor de microservicio puede estar limitada a los binarios de la aplicación y, a continuación, ser asociada con contenedores de utilidades genéricas (registro de errores, registro de actividades, seguridad, etc.) para formar un pod.

El sistema escalable basado en contenedores es altamente flexible, lo que permite cambios en los propios microservicios, así como en el tipo y número de contenedores, en los microservicios agrupados en un contenedor o contenedores en particular, y en programas de soporte tales como los elementos de emuladores y elementos del OS incluidos en contenedores y los recursos dedicados a contenedores o pods particulares en base a cambios en las transacciones, programas, otra información o utilización de transacciones o microservicios, entre otros factores.

Además, el número total de microservicios creados a partir de una aplicación heredada monolítica, o de una porción de la misma, puede ser mayor que el número total de transacciones individuales en la aplicación heredada monolítica o en la porción de la misma.

La figura 3 ilustra un sistema escalable basado en contenedores (300). El sistema escalable basado en contenedores puede incluir un depósito de códigos fuente (305), que almacena el código fuente de la aplicación heredada monolítica. El código fuente de la aplicación heredada monolítica puede ser, por ejemplo, una aplicación COBOL monolítica que puede incluir docenas, cientos o incluso hasta decenas de miles de archivos de programa individuales diseñados para realizar de manera individual o en grupos cientos de transacciones distintas, T_1, T_2, ..., T_X. Ejemplos de dichas transacciones pueden incluir la creación, actualización, traslado o eliminación de registros de clientes, que pueden, por ejemplo, utilizar el Sistema de control de información del cliente (“CICS” - Customer Information Control System, en inglés) o el Sistema de gestión de información (“IMS” - Information Management System, en inglés) para realizar transacciones de bases de datos relacionales de base de datos 2 (“DB2” - DataBase 2, en inglés) o transacciones de bases de datos jerárquicas de interfaz de lenguaje de datos (“DL/I” - Data Language Interface, en inglés). Un compilador (310) compila el código fuente en un conjunto de uno o más binarios que son almacenados en un depósito de binarios (315).

De acuerdo con ciertas realizaciones, un analizador de código fuente (320), normalmente a través de un componente analizador de dependencias, analiza el código fuente y los archivos asociados en la aplicación heredada monolítica tal como se almacena en el depósito de códigos fuente (305), y genera un árbol de código que identifica interdependencias (llamante <> destinatario de la llamada) en el código fuente. Preferiblemente, el analizador de código fuente (320) itera a través de cada transacción de la aplicación heredada monolítica, tal como se define en los parámetros de configuración del sistema transaccional, tal como CICS, IMS, etc. En un ejemplo, el analizador de código fuente (320) recibe como entrada del depósito de códigos fuente (305), un archivo que identifica las definiciones de transacciones de CICS disponibles que pueden ser invocadas por los usuarios en sus interacciones con la aplicación heredada monolítica. Preferiblemente, este archivo identifica cada transacción y su raíz, o primer programa invocado al realizar la transacción. Esto puede incluir el programa raíz como el destinatario de la llamada de un EXEC CICS LINK, utilizado como en muchas de las transacciones. En este ejemplo, el programa raíz se refiere al primer programa llamado por el programa que maneja la interfaz (por ejemplo, hacer los MAPAS ENVIAR / RECIBIR cuando la interfaz es 3270 pero también otras API equivalentes cuando la interfaz es diferente). Se pueden utilizar otros archivos o formatos que identifiquen transacciones o contribuyan a sus servicios, por ejemplo, los archivos de generación adicionales pueden incluir archivos de definiciones para los recursos utilizados por una transacción, tales como colas de mensajes y fuentes de datos.

Adicionalmente, el analizador de código fuente (320) puede analizar todos los archivos de programa asociados con la aplicación heredada monolítica, para detectar relaciones de interdependencia (llamante <> destinatario de la llamada para programas o inclusión de recursos como cuadernos de copias) entre archivos de programa para todas las transacciones de la aplicación heredada monolítica. Un analizador de dependencia dentro del analizador de código fuente (320) identifica las relaciones entre el llamante y el destinatario de la llamada o las relaciones de inclusión entre los programas utilizados por una transacción. El analizador estático puede generar un árbol de llamadas o inclusión en forma de un vector o conjunto de vectores o un gráfico que identifica los programas o módulos que el código fuente puede invocar o incluir para una transacción particular.

Se desea una división de la aplicación heredada monolítica para dividir la aplicación en un conjunto de transacciones mínimamente interdependientes accesibles, por ejemplo, a través de SOAP o REST (con JSON u otro formato de datos). Cada una de las transacciones mínimamente interdependientes puede ser ejecutada en una instancia independiente del emulador heredado (325). Una salida del analizador de código fuente (320) puede ser una llamada a programa o un árbol de inclusión o un gráfico que identifique, para cada transacción, el conjunto completo de programas que se pueden invocar o utilizar para realizar cada transacción y las relaciones entre llamante, destinatario de la llamada o inclusión entre los programas. La figura 4 es un ejemplo de un árbol de llamadas de este tipo en el que una primera transacción, T1, comienza con un programa raíz A, que luego puede llamar al programa F o al programa D. Aún en la transacción T1, el programa D puede llamar al programa E. Una segunda transacción, T2, comienza con el programa raíz B, que luego puede llamar al programa C, o también al mismo programa D, que luego llama al programa E.

El árbol de llamadas se puede traducir en un conjunto de vectores, uno para cada transacción, o en un subconjunto definido de las posibles transacciones de la aplicación heredada monolítica, identificando los programas que pueden ser invocados para realizar una transacción. La figura 2A representa un ejemplo de un conjunto (200) de vectores de definición de transacciones, Ta (210), Tb (220), ..., Tc (230). En este ejemplo, un primer vector, tal como Ta (210), incluye el conjunto de programas <P1, P2, ..., Px> que son potencialmente llamados para realizar una primera transacción. Utilizando el ejemplo de la figura 4, la transacción podría ser T1 y este conjunto de programas incluiría los programas A, F, D y E. Un segundo vector ilustrativo, Tb (220), que incluye los programas <P2, P3, ..., Py> y el tercer vector ilustrativo Tc (230), que incluye los programas <P1, P6, ..., Pz> correspondientes a la segunda y tercera transacción, también se muestran. Diferentes números y combinaciones de programas pueden designar las diferentes transacciones de la aplicación monolítica heredada.

El analizador de código fuente (320) también puede, en base a la definición de interfaz del programa raíz, extraer o generar los tipos de datos, mensajes, formatos / enlaces de mensajes y conjuntos de entradas y salidas de mensajes, y definir direcciones y puntos finales de cada transacción, y traducir esta información en una estructura de mensaje para ser utilizada en la generación y definición de una interfaz para las transacciones cuando el mensaje es proporcionado al generador de contenedores (330) y/o al sistema de gestión de contenedores (335), por ejemplo de una imagen de microservicios. Además, el analizador de código fuente (320) también puede generar una interfaz de mensajes WSDL si se utiliza el protocolo SOAP. La interfaz de mensajes WSDL puede ser un documento formateado definido en un estándar W3C, que incluye una estructura para almacenar tipos de datos definidos, mensajes, tipos de puertos, enlaces, puertos e información de definición de servicios. El analizador de código fuente (320) también puede generar otras representaciones de los mensajes de interfaz si otros protocolos (REST, etc.) y representaciones (JSON) son preferibles para una situación dada. El analizador de código fuente también puede ser configurado para generar tablas de traducción de codificación de datos bidireccionales o procedimientos para convertir caracteres UTF en caracteres EBCDIC de 8 bits y viceversa (o entre diferentes conjuntos de caracteres, incluido ASCII), y esta traducción puede ser implementada generando una secuencia de comandos / programa que se utilizará con microservicios en función de las transacciones y en sus interfaces hacia el solicitante.

El conjunto (200) de vectores de definición de transacciones, la definición de interfaz de comunicación (WSDL, REST) y las directivas de traducción a través de la secuencia de comandos pueden ser almacenadas en un depósito de

definición de estado de transacción (340).

El analizador de código fuente (320) también puede incluir parte de una aplicación de transcodificación para presentar una ruta de transcodificación para la utilización de programas transcodificados en el sistema escalable basado en contenedores. De esta manera, el analizador de código fuente también se puede utilizar para apoyar la transición del código fuente de su idioma original, tal como Cobol, a un idioma diferente, tal como Java. Se podrían realizar otras traducciones de código fuente. Además, el analizador de código fuente (320) también se puede utilizar en forma de un programa autónomo que no forma parte de una aplicación de transcodificación.

Cada vector de definición de transacción (210), (220), (230) en el depósito de definición de estado de transacción (340) incluye un superconjunto de los programas que se invocan realmente en el curso de la realización de transacciones reales utilizando la aplicación heredada monolítica. Con frecuencia, las aplicaciones de transacciones contienen muchos programas que nunca son invocados. Esto puede surgir debido al diseño inicial de la aplicación de transacción, para diseñar cambios, cambiar casos de utilización, compartir programas y sus destinatarios de llamada en diferentes partes de la aplicación de transacción o en otra evolución de la aplicación de transacción. La inclusión de estos programas no utilizados en el código da como resultado una menor eficiencia de la aplicación basada en contenedores por varias razones, incluida la sobrecarga requerida para moverse en el almacenamiento permanente, cargar y descargar en la memoria central del ordenador programas que no son invocados, así como retrasos adicionales en la compilación, generación o transporte a través de una red de actualizaciones a los contenedores de transacciones. Para eliminar estos programas no utilizados de las imágenes de la aplicación de microservicio, el optimizador de definición de microservicio (345) extrae el vector de definición de transacción, la definición de interfaz y las tablas de traducción del depósito de definición de estado de la transacción (340) y aplica un vector de definición dinámica almacenado en el depósito dinámico de definición (350), para eliminar programas no utilizados incluidos en los vectores de definición de transacción (210), (220), (230) del depósito de definición de estado de la transacción (340) para llegar a los vectores de definición de microservicio (260) (270), (280) correspondientes, tal como se muestra en la figura 2B, que puede ser almacenado en un estado intermedio por el optimizador de definición de microservicio (345) pendiente de un mayor refinamiento y definición de los microservicios, o procesado por el generador de imágenes de microservicios (350) para crear imágenes de microservicios almacenadas en el depósito de imágenes de microservicios (355) . En una aplicación heredada de un sistema monolítico grande, normalmente habrá programas no utilizados que pueden ser eliminados de este modo. Sin embargo, para las transacciones que utilizan todos los programas identificados por el análisis estático del estado de la transacción, el vector de definición de microservicio será el mismo que el vector de definición de transacción inicial. Esto se ilustra mediante el vector de definición de transacción (220) en la figura 2A y el vector de definición de microservicio correspondiente (270) en la figura 2B.

El vector de definición dinámico se desarrolla por separado de los vectores de definición del estado de la transacción mediante un proceso de definición dinámico, que normalmente se ejecuta en un sistema diferente o utiliza registros de actividad heredados. El vector de definición dinámica puede existir previamente o puede desarrollarse en paralelo con los vectores de definición de la transacción.

En el proceso de definición dinámica, se ejecuta la aplicación heredada monolítica y se analiza cada transacción para determinar qué programas son llamados realmente y cuáles no. Cuando el sistema se ejecuta durante un período de tiempo suficiente (por ejemplo, semana, mes, trimestre, año, según la naturaleza de la aplicación) o utilizando conjuntos de datos que invocan todos los casos de utilización reales, entonces, el vector de definición dinámica identificará con mayor precisión los programas a los que se llama realmente al realizar una transacción.

Alternativamente, el vector de definición dinámica también se puede generar comenzando con el vector de definición del estado de la transacción estático, que puede incluir programas en exceso, y luego seleccionar solo aquellos programas que son invocados realmente. Por tanto, el vector de definición dinámica puede construirse a medida que se identifican los programas, o puede crearse eliminando programas innecesarios del vector de definición del estado de la transacción.

En algunos sistemas, el analizador de registros de actividad (365) utiliza los registros de actividad heredados (360) preexistentes de la aplicación heredada monolítica ejecutada en su entorno informático heredado para identificar programas que realmente son invocados por la ejecución de transacciones del mundo real y, por lo tanto, generar un vector de programa que indique qué programas se utilizan para cada transacción

En ciertos sistemas, la aplicación heredada monolítica es ejecutada en un emulador heredado (325) y un analizador de registro de actividad (365) analiza los datos de registro de actividad generados por el emulador para generar un vector de programa que indica qué programas se utilizan para cada transacción. En algunas realizaciones, el emulador heredado (325) ejecuta cada transacción durante un período de tiempo suficiente para conseguir la confianza de que se han encontrado todas las variantes reales de los casos de utilización para cada transacción. Alternativamente, se puede llevar a cabo un conjunto definido de transacciones de prueba diseñadas para ejercitar cada caso de utilización real, lo que permite que el analizador de registro de actividad (365) determine de manera similar qué programas son realmente utilizados por las transacciones en la aplicación heredada monolítica.

En algunos sistemas, el analizador de registros de actividades (365) puede utilizar la información de los registros de actividades heredados (360) y del emulador heredado (325) para determinar qué programas utilizan realmente las

transacciones en la aplicación heredada monolítica. Por ejemplo, si los registros de actividad heredados (360) no contienen ejemplos de un programa que se esté utilizando en una transacción determinada, se pueden consultar los registros del emulador heredado (325), o viceversa, antes de concluir que esa transacción no utiliza el programa. En otro ejemplo, las transacciones para las que hay una gran cantidad de datos heredados pueden ser evaluadas utilizando solo registros de actividad heredados (360), sin emulación adicional mediante el emulador heredado (325). En otro ejemplo más, los datos de registro heredados pueden ser utilizados como una pista inicial para la definición de microservicios.

La salida del analizador de registro de actividades se almacena en el depósito de definición dinámica (370), que almacena los vectores correspondientes a los programas realmente utilizados, para cada transacción.

Un módulo de carga se refiere a todo o a parte de un programa ejecutable, normalmente en el contexto de un entorno informático heredado de ordenador central. El emulador heredado (325) puede ser un emulador desarrollado para permitir la ejecución de una aplicación heredada compilada o de un módulo de carga desde un z/OS u otro entorno informático heredado para ser ejecutado en un entorno informático distinto, tal como una plataforma x86 con el sistema operativo Linux. El emulador heredado puede convertir cada instrucción nativa o llamada a servicio del sistema operativo nativo del programa ejecutable original en instrucciones equivalentes y llamadas a sistemas del entorno informático distinto. El emulador heredado (325) puede implementar un conjunto de API nativas para permitir la emulación de instrucciones heredadas individuales o llamadas a un servicio del sistema. El emulador heredado (325) puede ser una sola imagen de todo el emulador, o puede incluir imágenes divididas, tal como se describe más adelante en el presente documento. El emulador heredado (325) puede incluir, además, o tener acceso operable a un sistema operativo o a componentes del mismo realmente utilizados por el emulador heredado.

El optimizador de definición de microservicio (345) aplica vectores de transacción dinámica almacenados en el depósito de definición dinámica (370) a los vectores de definición de transacción almacenados en el depósito de definición del estado de la transacción (340) para llegar a vectores de definición de microservicio que pueden ser utilizados por el generador de imágenes de microservicio (350) para crear imágenes de microservicio. A continuación, estas imágenes son almacenadas en el depósito de imágenes de microservicio (355).

La figura 2B representa un ejemplo de un conjunto de vectores (250) de definición de microservicio, MSa (260), MSb (270), ..., MSc (280). En este ejemplo, un primer vector de definición de microservicio, MSa (260), incluye el vector optimizado elaborado a partir del conjunto de programas <P1b, ..., Px-qb> que son llamados cuando se realiza la primera transacción Ta. En este ejemplo, el programa P2 no se utiliza realmente en la transacción Ta y, por lo tanto, se elimina del vector de definición de microservicio. Un segundo vector de definición de microservicio, MSb (270), ilustrativo, incluye los programas <P2, P3, ..., Py>. En este ejemplo, se utilizan todos los programas que componen el vector de definición de transacciones y, por lo tanto, se conservan en el vector de definición de microservicios. Un tercer vector de definición de microservicio, MSc (280), ilustrativo, incluye los programas <P1, P6, ..., Pz-y>. La arquitectura resultante incluye un conjunto de transacciones Tx, cada una definida por el menor número de programas. Cualquiera de las transacciones Tx de la aplicación heredada monolítica se puede definir como un microservicio MSx que puede ser llamado de manera independiente, tanto en la operación traducida de la aplicación heredada monolítica citada anteriormente como en aplicaciones mejoradas o modificadas que pueden invocar los microservicios MSx definidos.

Cualquiera de las transacciones Tx también se puede definir como un conjunto de microservicios que pueden ser llamados de manera independiente. Para el conjunto total de transacciones Tx de una aplicación heredada monolítica, algún subconjunto puede estar definido por un microservicio por cada transacción, mientras que otro subconjunto puede estar definido por un conjunto de microservicios por cada transacción. Por ejemplo, tal como se ilustra en la figura 5, si las transacciones T1 y T2 utilizan programas comunes D y E, cuando estas transacciones son traducidas en microservicios por el optimizador de definición de microservicio (345), esos programas comunes pueden ser agrupados como un microservicio independiente, MS3, que puede ser llamado por el MS1, que contiene los otros programas de T1, o llamado por el MS2, que contiene los otros programas de T2.

El optimizador de definición de microservicio (345) puede almacenar los vectores de imagen de microservicio o los vectores de imagen de microservicio intermedios que luego cambia u optimiza. Por ejemplo, el optimizador de definición de microservicio (345), cuando se presenta con vectores de definición de transacciones para las transacciones de la figura 4, puede crear primero vectores de definición de microservicio intermedios, MS1 y MS2, los cuales contienen los programas también situados en los vectores de definición de transacciones. El optimizador de definición de microservicio (345), puede reconocer el componente común de estos vectores de definición de microservicio MS1 y MS2, tal como se indica mediante los elementos D y E de la figura 4 y extraer el componente común de los dos primeros vectores de definición de microservicio. Tal como se muestra en la figura 5, además del primer y segundo microservicio, MS1 y MS2, los elementos comunes D y E se utilizan para crear un tercer vector de definición de microservicio, MS3, que contiene estos componentes comunes y que puede ser llamado por el MS1 o el MS2. Estos vectores de definición de microservicios optimizados, MS1, MS2 y MS3, son proporcionados a continuación al generador de imágenes de microservicios (350).

Alternativamente, los vectores de definición de microservicios intermedios pueden ser almacenados en una ubicación distinta al optimizador de definición de microservicios (345), tal como en un depósito intermedio (no mostrado). En

ciertas realizaciones, los vectores de definición de microservicios intermedios pueden ser almacenados en el generador de imágenes de microservicios (350) o como imágenes intermedias en el depósito de imágenes de microservicios (355), y, a continuación, acceder a ellos y/o ser sustituidos por vectores de definición de microservicios optimizados o por imágenes de microservicios.

El compilador (310), compila el código fuente en el depósito de código fuente (305) para producir binarios en el depósito de binarios (315). El compilador (310) genera binarios para un entorno informático heredado, tal como un sistema de ordenador central 390 o z/OS. De esta manera, los binarios utilizados para generar imágenes de microservicio en el sistema escalable basado en contenedores descrito en el presente documento pueden ser los mismos que los binarios que se ejecutan en el entorno informático heredado, lo que facilita la interoperabilidad y la migración gradual de la aplicación heredada monolítica del entorno informático heredado al sistema escalable basado en contenedores.

El generador de imágenes de microservicio (350) recupera los binarios compilados del depósito de binarios (315) que corresponden a los programas identificados en los vectores de definición de microservicio o en los vectores de definición de microservicio optimizados, según corresponda, y combina los binarios para generar una imagen para cada microservicio que incluye imágenes de binarios para cada programa en el vector de definición de microservicio. Las imágenes de microservicio también pueden incluir información y artefactos asociados, tales como definiciones de recursos compartidos, etc. recuperados por el generador de imágenes de microservicio (350). Estas imágenes de microservicios se almacenan en el depósito de imágenes de microservicios (355).

El generador de contenedores (375) genera imágenes de contenedores combinando las imágenes de binarios asociadas con un microservicio específico almacenado en el depósito de imágenes de microservicio (355) con imágenes de binarios almacenadas en el depósito de componentes complementarios (380). El depósito de componentes complementarios (380) puede almacenar un conjunto de archivos de imagen de los elementos del emulador que, juntos, forman un emulador heredado, que normalmente es el mismo que el emulador heredado (325) utilizado de otra manera por el sistema escalable basado en contenedores.

El emulador heredado puede estar dividido por funciones o subconjuntos de funciones para formar elementos heredados, lo que proporciona ventajas para la implementación del emulador heredado en el sistema basado en contenedores descrito en el presente documento. Por ejemplo, el soporte para subconjuntos de instrucciones en interfaces compatibles con el emulador heredado puede ser separado. Además, el soporte en el emulador heredado para operaciones por lotes, para servicios de transacciones CICS, DB2 u otros servicios de bases de datos relacionales, los servicios IMS, seguridad, registro u otras capacidades, pueden ser divididos. De esta manera, solo un elemento heredado individual o un conjunto de elementos del emulador heredado utilizado por los microservicios en un contenedor puede ser ejecutado dentro de un contenedor determinado. Además, ciertos elementos heredados utilizados por los contenedores en un pod pueden ser almacenados en contenedores separados, y a continuación, los microservicios pueden acceder a ellos en otros contenedores del pod. Elementos heredados adecuados incluyen funciones de seguimiento y registro del entorno de ejecución del emulador. Dicha configuración puede mejorar el rendimiento y/o la seguridad.

El depósito de componentes complementarios (380) también puede almacenar paquetes de software del sistema operativo que puede utilizar el emulador heredado, que se pueden denominar elementos del OS. Por ejemplo, los componentes individuales de la API del sistema también pueden ser almacenados individualmente como imágenes independientes. En algunos ejemplos, los paquetes individuales y los archivos de la librería se pueden combinar en tiempo de ejecución para aumentar la funcionalidad ofrecida por Linux u otro sistema operativo, y los binarios pueden ser almacenados en el depósito de componentes complementarios (380).

El generador de contenedores (375) puede incorporar de manera selectiva elementos de emulador y/o elementos del OS para proporcionar funcionalidades asociadas con un microservicio o un conjunto de microservicios en la imagen del contenedor que contiene ese microservicio o conjunto de microservicios. De esta manera, el tamaño total de la imagen para cada contenedor puede ser menor que si se incluyera la imagen completa del emulador heredado o una imagen completa del sistema operativo en cada contenedor.

La imagen de un emulador heredado puede tener, en algunos casos, varios cientos de megabytes. Los elementos del emulador que ejecutan una función específica, tal como un proceso por lotes específico o una transacción de base de datos específica, por otro lado, pueden ser solo unas pocas decenas de megabytes. De manera similar, una imagen de un sistema operativo completo puede ser muchas veces mayor que las imágenes de los componentes reales utilizados por un elemento de emulador.

En consecuencia, la división del emulador heredado en elementos del emulador y la inclusión de menos de todos estos elementos en un contenedor, o en un contenedor en un pod, puede reducir la memoria utilizada para alojar el contenedor o el pod de cinco a siete veces más en comparación con un contenedor o pod que, de otro modo, sería idéntico que contiene una imagen del emulador heredado completo, o elementos del emulador no utilizados por microservicios en el contenedor o pod.

La inclusión de menos de todos los elementos del sistema operativo en un contenedor, o en un contenedor en un pod, puede reducir de manera similar la memoria utilizada para alojar el contenedor o el pod de cinco a siete veces en

comparación con un contenedor o pod idéntico que contiene una imagen del OS completo, o elementos del OS no utilizados por microservicios y/o elementos del emulador en el contenedor o pod.

Mediante la inclusión de menos de todos los elementos del emulador y menos de los elementos del sistema operativo en un contenedor, o en un contenedor en un pod, la memoria utilizada para alojar el contenedor o pod también se puede reducir de cinco a siete veces en comparación con un contenedor o pod, por lo demás idéntico, que contiene una imagen del emulador heredado completo, o elementos del emulador no utilizados por microservicios en el contenedor o pod, y una imagen del sistema operativo completo, o elementos del sistema operativo no utilizados por microservicios y/o elementos del emulador en el contenedor o pod. En este caso, las contribuciones relativas de la reducción del tamaño del emulador heredado y el tamaño del sistema operativo a la reducción de la memoria utilizada para alojar la combinación de los dos pueden depender de los tamaños generales relativos del emulador heredado y el sistema operativo y el grado de división de ambos. Por ejemplo, en el caso de un emulador heredado de 200 MB dividido en aproximadamente diez elementos y un sistema operativo de 50 MB dividido en aproximadamente cincuenta elementos, las contribuciones de eliminar elementos del emulador normalmente superarán las contribuciones de eliminar elementos del sistema operativo.

El emulador heredado puede ser dividido en elementos de emulador que se correspondan con las necesidades probables de los microservicios. Por ejemplo, es probable que los microservicios no necesiten ciertas funcionalidades, tales como la consola de gestión y las funcionalidades de la interfaz de usuario, o el sistema de gestión de contenedores las puede proporcionar de forma nativa, en una forma más adecuada para esta arquitectura (385) y, por lo tanto, pueden estar separadas de los demás elementos del emulador, e incluso pueden estar omitidas del depósito de componentes complementarios (380). Otros elementos del emulador, tales como los elementos de seguridad, pueden ser divididos específicamente para que puedan ser dispuestos en contenedores separados de otros elementos del emulador y microservicios, o incluso ser sustituidos por servicios similares proporcionados por el nuevo sistema.

El emulador heredado también puede ser dividido para colocar las funcionalidades centrales, en las que se basan otros componentes del emulador heredado, en un elemento del emulador central. Dicho elemento puede estar incluido en la mayoría, si no en todos los contenedores o pods. A menudo, este elemento principal del emulador será una proporción mayor del tamaño total del emulador heredado que otros elementos del emulador. Por ejemplo, un elemento de emulador central puede tener entre el 30 % y el 40 % del tamaño del emulador heredado total.

El emulador heredado puede ser dividido, además, para colocar las funcionalidades que probablemente se utilizarán, en general, en uno o en varios contenedores en un pod, pero no en todos los contenedores, tales como las funcionalidades de seguridad, en un elemento separado, tal como un elemento del emulador de seguridad.

Utilizando un emulador transaccional como ejemplo, los elementos adecuados del emulador también pueden incluir un elemento de emulador en línea / de comunicaciones (tal como uno que contenga subproductos para CICS e IMS-TM para servicios transaccionales), un elemento de emulador relacional (tal como uno para DB2), un elemento de emulador de base de datos jerárquica (tal como uno para IMS-DB), un elemento de emulador de gestión de conjuntos de datos / fecha (tal como uno para archivos VSAM y archivos secuenciales), un elemento de emulador de servicios por lotes, y/o un elemento de emulador de idiomas (tal como uno con subproductos para Cobol y PL/1), un elemento de emulador de seguridad y un elemento de emulador de interfaz de usuario / consola de gestión.

Los subproductos pueden estar excluidos de la imagen del elemento de emulador incorporado realmente en un contenedor. Por ejemplo, un elemento de emulador en línea / de comunicaciones puede contener solo imágenes de binarios para CICS y no para IMS-TM.

Los elementos del emulador pueden variar en tamaño en comparación con el emulador heredado total, pero, normalmente, los elementos del emulador que no son del núcleo pueden estar entre el 1 % y el 20 %, más particularmente entre el 3 % y el 15 % del tamaño total del emulador heredado. El tamaño de un elemento de emulador en comparación con el emulador heredado total, junto con otros factores tales como la probabilidad de utilización conjunta, se pueden utilizar para determinar qué funcionalidades están separadas en diferentes elementos del emulador.

Los elementos del sistema operativo pueden estar en forma de paquetes disponibles, tal como diversos paquetes de Linux como PostgresSQL, LLVM, node.js, etc.

El tamaño de los elementos del sistema operativo que acompañan a los elementos del emulador también se puede utilizar para determinar qué funcionalidades del emulador heredado están separadas en diferentes elementos del emulador.

En algunos sistemas escalables basados en contenedores, el generador de contenedores (375) incluye un compilador de módulo de carga, que recibe como entrada los archivos binarios, tales como los archivos de imagen ejecutables System 390 o z/OS, almacenados en el depósito de imágenes de microservicios (355). El compilador del módulo de carga detecta todas las firmas en los archivos binarios de las llamadas a programas, servicios o funciones del entorno informático heredado mediante la aplicación heredada monolítica, tal como una serie de instrucciones de ensamblador. El compilador del módulo de carga puede utilizar esta información para determinar las funciones del emulador heredado que utiliza el microservicio o el conjunto de microservicios. El generador de contenedores (375) puede situar los elementos del emulador capaces de realizar estas funciones entre los elementos del emulador en el depósito de componentes complementarios (380) y colocar los elementos del emulador, junto con cualquier elemento del OS

asociado del depósito de componentes complementarios (380) con las imágenes del microservicio o conjunto de imágenes del microservicio en una imagen del contenedor. Alternativamente, el generador de contenedores (375) colocará las imágenes de los elementos del emulador y los elementos del OS en una imagen de contenedor asociada con una imagen de contenedor de la imagen de microservicios o de un conjunto de imágenes, de modo que ambas imágenes del contenedor serán colocadas en un pod.

Además, el compilador del módulo de carga puede sustituir la firma o firmas en los binarios por instrucciones para llamar a la misma función o funciones llamadas en el entorno informático heredado en el emulador heredado en su lugar, formando de este modo una imagen de microservicio optimizada del emulador heredado que puede ser almacenada en la imagen del contenedor. Las firmas pueden ser identificadas y las instrucciones de sustitución pueden ser situadas utilizando una base de datos preexistente creada para la aplicación heredada monolítica o el entorno informático heredado y el emulador heredado o el entorno informático distintivo del sistema escalable basado en contenedores. Además, el generador de contenedores (375) puede sustituir las llamadas a funciones heredadas identificadas por llamadas a API nativas del emulador heredado, y generar una imagen o imágenes modificadas.

Durante o después de cualquier optimización o modificación de imágenes de microservicio o imágenes de contenedor tal como se ha descrito en el presente documento, el generador de contenedores (375) realiza un almacenamiento, a continuación, en el depósito de imágenes de contenedor (390). Posteriormente, las imágenes del contenedor en el depósito de imágenes de contenedor (390) son ejecutadas en contenedores (395) gestionados por el sistema de gestión de contenedores (385).

De acuerdo con ciertas realizaciones, el depósito de imágenes de contenedor (390) puede ser un depósito de Docker, similar en estructura al Docker Hub público. El sistema de gestión de contenedores (385) soporta, por lo tanto, preferiblemente, contenedores Docker, y permite su ejecución optimizada.

El sistema de gestión de contenedores (385) puede combinar las funciones de programar la creación de instancias de contenedores, ejecutar contenedores, asignarles una cantidad controlada de recursos informáticos / de almacenamiento / de redes, actualizándolos, y/o puede realizar funciones adicionales de registro y gestión para rastrear y gestionar la salud del sistema. De acuerdo con determinadas realizaciones, el sistema de gestión de contenedores (385) puede ser el sistema de gestión de contenedores de Kubernetes para contenedores Docker. Pero se podría utilizar otro sistema de gestión de contenedores tal como Amazon ACS, Azure Container Service, Cloud Foundry Diego, CoreOS Fleet, Docker Swarm, Google Container Engine o el sistema de gestión de contenedores Mesosphere Marathon, u otros sistemas de orquestación de contenedores. El sistema de gestión de contenedores (385) puede ser similar al descrito en la figura 1B, con modificaciones y adiciones, tal como se ha descrito en el presente documento. La asignación selectiva de recursos por parte del sistema de gestión de contenedores (385) se puede realizar mediante la utilización de grupos de control, cuando los contenedores están basados en Docker.

Un proxy inteligente (no mostrado) delante del sistema de gestión de contenedores (385) puede mantener una conexión de TCP permanente con el emulador del terminal del usuario final o con cualquier otra interfaz de cliente que requiera una conexión permanente. A continuación, este proxy escaneará las solicitudes en la conexión permanente y las convertirá en las solicitudes de servicio apropiadas que, a continuación, son enrutadas por Kubernetes hacia el microservicio apropiado. Los contenedores ad hoc en el proxy inteligente y en los microservicios permiten la encapsulación del tráfico 3270 o de cualquier otro tráfico específico en solicitudes y respuestas de microservicios.

Los contenedores (395) y el sistema de gestión de contenedores (385) pueden residir en el subsistema (400). El subsistema (400) puede estar físicamente separado del resto del sistema escalable basado en contenedores (300) y puede operar en un sistema independiente que es capaz de conseguir los mismos beneficios disponibles cuando se utiliza un sistema escalable basado en contenedores (300). Por ejemplo, el subsistema (400) puede realizar funciones de asignación de recursos y de gestión de contenedores, tal como se ha descrito en el presente documento. Particularmente, si el subsistema (400) también incluye un depósito de imágenes de contenedores (390), el sistema de gestión de contenedores (385) también puede crear contenedores adicionales o duplicados utilizando imágenes de contenedores. El subsistema (400) todavía puede beneficiarse de la división de la aplicación heredada monolítica en microservicios y de la inclusión solo de los elementos del emulador y los elementos del sistema operativo necesarios en las imágenes del contenedor. Sin embargo, debido a que el subsistema (400) carece de los elementos del sistema escalable basado en contenedores (300) dedicado a crear vectores de definición de microservicio e imágenes de contenedor, no puede actualizar automáticamente sus imágenes de contenedores y contenedores. En su lugar, puede recibir imágenes de contenedores actualizadas que el sistema de gestión de contenedores (385) aplica a los contenedores (395), o que están almacenadas en el depósito de imágenes de contenedores (390), si está presente.

Otro subsistema, no ilustrado, puede incluir contenedores (395), sistema de gestión de contenedores (385), depósito de imágenes de contenedores (390), generador de contenedores (375) y depósito de componentes complementarios (380). Dicho subsistema puede estar físicamente separado del resto del sistema escalable basado en contenedores (300), y puede conseguir muchos de los beneficios descritos en conexión con el sistema (300). El subsistema tiene la capacidad de actualizar las imágenes de contenedores cuando se le proporcionan nuevas imágenes del microservicio. Dicho subsistema puede contener, además, un depósito de imágenes de microservicio (355) y/o (emulador de aplicaciones heredado (325), pero carece de componentes responsables de desarrollar nuevos vectores de definición de microservicio y/o imágenes de microservicio, inicialmente o cuando se actualiza el código fuente monolítico. Dicho

subsistema también puede incluir un emulador de aplicaciones heredado (325).

Muchas aplicaciones heredadas basadas en bases de datos relacionales están estructuradas de acuerdo con la teoría relacional de Tedd Codd publicada inicialmente en su artículo “A Relational Model of Data for Large Shared Data Banks” CACM 13, Nº 6, junio de 1970. Esas bases de datos heredadas se han diseñado pensando en una redundancia mínima: su estructura ha sido normalizada, habitualmente, en la medida de lo posible. La quinta forma normal (5NF – Fifth Normal Form, en inglés) fue el objetivo de diseño inicial para la mayoría de ellas, incluso si la vida real ha alterado esta forma ideal a lo largo de los años. El resultado de un alto grado de normalización es una gran interdependencia entre diversas secciones de los datos utilizados por una aplicación heredada monolítica.

Esta arquitectura de datos entrelazados crea interdependencias indirectas entre grupos de programas en la aplicación heredada monolítica que comparten los mismos datos, ya sea directamente (solicitudes sql que acceden a las mismas tablas) o indirectamente (tablas a las que accede el programa X modificadas por restricciones de integridad referencial en tablas actualizadas por el programa Y).

Pero, en la mayoría de los casos, una aplicación heredada monolítica grande normal todavía tiene grupos de datos independientes en su gran base de datos compuesta por miles de tablas. En un sistema escalable basado en contenedores, estas agrupaciones deberían, para mejorar varias capacidades del sistema, estar separadas en bases de datos secundarias independientes, cada una utilizada por un conjunto independiente de microservicios. Estas bases de datos secundarias pueden ser aisladas, por ejemplo, en servidores de bases de datos independientes, y pueden ser gestionadas de forma independiente entre sí. Esto aumenta la flexibilidad y la agilidad del sistema global, porque los cambios en la estructura de datos locales son más simples de ejecutar desde un punto de vista operativo que los globales. Esta separación de bases de datos en bases de datos secundarias también aumenta la disponibilidad global del sistema escalable basado en contenedores, porque un problema con una base de datos secundaria o su mantenimiento no afecta a las otras bases de datos y microservicios que las utilizan.

De manera similar a la identificación de las dependencias del programa, los datos pueden ser divididos de acuerdo con la arquitectura de microservicios mediante la creación de árboles de dependencia que identifican los grupos de datos mediante su utilización en las transacciones o conjuntos de transacciones correspondientes. Esta identificación puede ser realizada por el analizador de código fuente (320), y, particularmente, por su analizador de dependencia, ya que analiza la aplicación heredada monolítica para producir bases de datos secundarias y grupos de bases de datos secundarias, normalmente en forma de vectores o tablas, que pueden ser separados entre sí para conseguir, como mínimo, algunos de los beneficios descritos anteriormente.

Diversas imágenes de microservicios pueden compartir un acceso similar a las mismas bases de datos secundarias. En particular, las transacciones de servicios de bases de datos relacionales pueden ser empaquetadas de manera separada de las transacciones para otras funcionalidades del emulador heredado, de modo que, por ejemplo, los servicios de procesamiento y los servicios de bases de datos se definan en última instancia en microservicios separados.

La base de datos completa o las bases de datos secundarias pueden ser compartidas entre varios microservicios. La base de datos completa o las bases de datos secundarias pueden estar situadas en contenedores de base de datos de larga duración separados, a los que un contenedor de procesamiento de menor duración puede acceder de manera remota. Normalmente, los contenedores con microservicios de procesamiento pueden estar en un pod con uno o más contenedores que alojan los servicios de bases de datos relacionales y las bases de datos secundarias utilizados por los microservicios de procesamiento.

En tipos similares de estructuras, el soporte para objetos compartidos entre transacciones en la aplicación heredada monolítica puede ser implementado detectando los objetos compartidos utilizando el analizador de código fuente y, a continuación, reuniendo objetos de soporte en contenedores de recursos especializados utilizando el generador de contenedores según lo informado por el analizador de código fuente. Por ejemplo, las colas de TS de CICS compartidas entre programas presentes en varios microservicios pueden residir en un contenedor de recursos de larga duración que los aloja. Se puede acceder a estos objetos compartidos (por ejemplo, sesiones de memoria, colas de mensajes, objetos de datos compartidos) de manera remota pero transparente a través de las funciones de acceso remoto del emulador heredado, desarrolladas inicialmente con el propósito de replicar las funciones de acceso remoto del entorno informático heredado. En el caso del entorno heredado de CICS, esas funciones son las versiones emuladas de funciones heredadas tales como MRO, IPIC, etc. Se pueden detectar zonas de memoria compartida (CSA, CICS CWA, CICS TCTUA, etc. en el caso de un sistema z/OS), colocadas en una memoria caché compartida distribuida y a los que acceden de manera remota las mismas funciones de acceso remoto en los contenedores de recursos específicos cuando son compartidos entre diversos microservicios.

En otro tipo de estructura similar, para maximizar la separación de datos, se pueden construir transacciones que abarquen varios microservicios que se llamen entre sí de manera síncrona en cascada después de la solicitud de servicio inicial a Kubernetes. Esta realización introduce la complejidad adicional de compartir conexiones de bases de datos y transacciones distribuidas con problemas relacionados con el compromiso distribuido de 2 fases.

El sistema basado en contenedores descrito en el presente documento presenta un panorama cambiado desde el punto de vista de la generación, proporcionando un proceso de generación integrado y adaptable que se acopla de

manera flexible al entorno de producción. Cuando se realizan modificaciones al código fuente almacenado en el depósito de códigos fuente (305), el compilador (310) lo compila y lo almacena en el depósito de binarios (315), el analizador de código fuente (320), el depósito de definición de estado de transacción (340), el optimizador de definición microservicio (345) y el generador de imágenes de microservicio (350) se pueden utilizar para generar una imagen de microservicio actualizada o un conjunto de imágenes de microservicio para el microservicio o microservicios correspondientes solo a las transacciones afectadas por los cambios. A continuación, el generador de contenedores (375) puede activar procedimientos de generación, definidos y configurados de manera automática y óptima en base a los vectores de definición de microservicios extraídos previamente por el generador de contenedores, imágenes de contenedores para los microservicios actualizados, que, a continuación, pueden ser implementados por el sistema de gestión de contenedores (385). Las imágenes del contenedor pueden incluir simplemente imágenes actualizadas para un microservicio o conjunto de microservicios, pero también pueden incluir cambios, si es necesario, en las imágenes del depósito de componentes complementarios (380). En el caso de cambios más extremos o múltiples en el código fuente, los vectores de definición de microservicios pueden ser cambiados, de modo que se cree un microservicio o conjunto de microservicios diferente. Por ejemplo, si el código fuente es cambiado para proporcionar un gran número de transacciones que utilizan un conjunto común de programas, entonces ese conjunto común de programas puede ser colocado nuevamente en un microservicio separado, de manera similar a MS3 en la figura 5, y los vectores de definición de microservicios nuevos y existentes para otros microservicios son modificados o creados en consecuencia.

El proceso completo de actualización está, preferiblemente, automatizado, pero la implementación de microservicios actualizados también se puede colocar bajo el control de una consola de gestión administrativa (no mostrada). De manera similar, cuando hay cambios en otra información, tal como datos (por ejemplo, cuadernos, archivos sql, etc.), las dependencias del cambio pueden ser identificadas y propagadas para adaptar automáticamente los procedimientos de compilación.

Para ilustrarlo, las etapas automáticas del proceso de actualización pueden incluir: (1) estructura de código fuente colocada en el depósito de códigos fuente (310); (2) definición de trabajo de compilación de Jenkins (u otro sistema de compilación de DevOps); (3) generación de imágenes de Docker mediante la agrupación adecuada de binarios del ordenador central; y (4) parámetros de gestión de Kubernetes.

La estructura de microservicios del sistema escalable basado en contenedores también ofrece ventajas en términos de la cantidad de cambios necesarios para actualizar y del tiempo consumido para hacerlo. Por ejemplo, tal como se ilustra en la figura 5, los cambios en el programa D o E solo deben ser realizados en la compilación del microservicio MS3, en lugar de en dos compilaciones de microservicio independientes, MS1 y MS2, para las transacciones T1 y T2. El alto nivel de granularidad que presenta una gran cantidad de microservicios independientes lo permite, y, preferiblemente, opera bajo una automatización completa.

La formación de dichos microservicios puede mejorar la capacidad de administración general del sistema, ya que las actualizaciones o cambios en el código de la aplicación que cambian el árbol secundario solo necesitan provocar actualizaciones en los contenedores correspondientes para el microservicio interno, y no para todos los microservicios que lo invocan.

Dada la facilidad con la que se pueden construir los contenedores y el tiempo reducido para cargar una imagen de contenedor en un contenedor si es más pequeño, el optimizador de definición de microservicio (345) en muchos sistemas escalables basados en contenedores puede implementar instrucciones para crear múltiples vectores de definición de microservicio por cada vector de definición de transacción, particularmente donde, tal como se ilustra en la figura 4 y la figura 5, las transacciones utilizan programas o conjuntos de programas comunes que pueden ser colocados en un microservicio separado. Por ejemplo, las transacciones T se pueden convertir fácilmente en microservicios P, donde P es el número de programas y T es el número de puntos de entrada para transacciones compatibles con la aplicación heredada monolítica, si la necesidad de puntos de entrada ya no es el programa raíz de cada transacción existente, sino cualquier programa invocable (a través de LINK, por ejemplo, bajo CICS) dentro de la aplicación.

El hecho de que un determinado sistema escalable basado en contenedores sea implementado utilizando pods o solo contenedores puede informar aún más de cómo son creados y definidos los microservicios. Por ejemplo, un mayor análisis de transacciones en microservicios y más vectores de definición de microservicio mínimos pueden ser posibles en un sistema escalable basado en contenedores diseñado para utilizar pods que en uno no diseñado de esta manera.

En algunos casos, los únicos límites en la cantidad de microservicios separados definidos pueden ser la cantidad de programas separados en la aplicación heredada monolítica y/o la memoria disponible en el sistema escalable basado en contenedores para alojar el depósito de imágenes de microservicio (355) y/o de contenedores. (395).

Además, debido a que una imagen de contenedor determinada puede ser colocada en cualquier número de contenedores activos, el sistema escalable basado en contenedores permite la verificación y la implementación gradual de actualizaciones, ejecutando algunos contenedores versiones antiguas de un microservicio o conjunto de microservicios, ejecutando los contenedores más nuevos el microservicio o el conjunto de microservicios actualizados. Esto permite que las actualizaciones sean verificadas y probadas en busca de fallos, mientras se mantiene la capacidad de realizar una transacción utilizando una versión anterior del microservicio o un conjunto de microservicios,

si es necesario. Los contenedores que ejecutan una versión antigua de microservicios pueden ser eliminados automáticamente (o eliminados según una instrucción del usuario) una vez que la actualización se haya verificado suficientemente.

Además, debido a que los contenedores se pueden generar y eliminar fácilmente, si una transacción se está ejecutando en algunos contenedores, se pueden generar nuevos contenedores con actualizaciones para realizar nuevas solicitudes para esa transacción, mientras termina en contenedores existentes que carecen de la actualización, pudiendo entonces ser eliminados automáticamente cuando completen la transacción que están ejecutando inmediatamente. Así, por ejemplo, si diez contenedores C1 a C10 están ejecutando la transacción T1, cuando se produce una actualización al correspondiente MS1, el sistema de gestión de contenedores (385) puede crear automáticamente un nuevo contenedor, C11, cuando se recibe una nueva solicitud de transacción. El contenedor C11 incluye una imagen del microservicio actualizado, MS1'. Cuando el contenedor C1 completa la transacción que se está ejecutando, no se asignan nuevas transacciones al contenedor C1, y se elimina. Un nuevo contenedor con el microservicio actualizado MS1' puede ser generado inmediatamente para sustituir a C1, o puede ser generado cuando entra una nueva solicitud para la transacción T1, dependiendo de los parámetros aplicados por el sistema de gestión de contenedores (385) para crear y gestionar contenedores.

Tecnologías tales como Docker y Kubernetes se han diseñado para funcionar a escala web y, en consecuencia, para permitir un crecimiento muy rápido de las cargas de trabajo que se pueden distribuir en cada vez más máquinas x86 agregadas a medida que llegan más solicitudes. Ese es exactamente el propósito de un orquestador tal como Kubernetes. Puesto que las transacciones de los clientes en línea requieren cada vez más responder a un número cada vez mayor de consultas antes de completar una transacción, las demandas del comercio en línea introducen problemas de escalabilidad en la expansión de los entornos informáticos heredados en el mercado en línea. La escalabilidad de un sistema basado en contenedores tal como se describe en el presente documento es particularmente ventajosa para aumentar la escalabilidad de dichos entornos informáticos heredados, omitiendo la proliferación de contenedores dedicados a estas aplicaciones de consulta intensivas para el consumidor. Además, debido a que cada imagen de contenedor, o en algunos casos cada pod, contiene algunos elementos del OS y algunos elementos del emulador, se puede duplicar o mover fácilmente de un fragmento de hardware a otra, siempre que el entorno informático distinto, tal como la utilización de un sistema operativo Linux se conserve.

El aislamiento que proporcionan los contenedores aislados también proporciona un enfoque mucho más sofisticado en la gestión del nivel de servicio. A cada contenedor se le puede asignar una cantidad diferente de recursos para prestar un mejor servicio a algunos microservicios (correspondientes o utilizados por transacciones heredadas específicas) que a otros. Un sistema escalable basado en contenedores tal como se describe en el presente documento puede detectar y rastrear automáticamente la utilización de recursos por contenedor y dedicar más o menos recursos según la utilización. Además, o alternativamente, el sistema de gestión de contenedores puede escalar el número de contenedores dedicados a un microservicio o conjunto de microservicios particular en base a la utilización. Las prioridades definidas por el usuario también pueden ser incluidas en los cálculos para la asignación de recursos o del número de contenedores correspondientes a una transacción o microservicio. Este ajuste, definido por el usuario, de los recursos disponibles para una transacción determinada, no es posible en la aplicación heredada monolítica.

En algunas variaciones, el despliegue inicial de imágenes de contenedor que contienen microservicios o conjuntos de microservicios en contenedores o pods puede estar basado, como mínimo en parte, en la actividad de la transacción cuando la aplicación heredada monolítica es ejecutada en un entorno informático heredado, o en una emulación del mismo. Dicha información se puede derivar de un emulador heredado, tal como un emulador heredado (325) tal como se ilustra en la figura 3. Dicha información también puede ser derivada de registros de actividad heredados, tales como registros de actividad heredados (360) o un analizador de registros de actividad, tal como el analizador de registros de actividad (365) (no ilustrado en la figura 3).

Por ejemplo, el consumo de recursos para una transacción determinada cuando se utiliza una aplicación heredada monolítica a menudo se monitoriza con precisión. Los números de recursos se pueden extraer y utilizar, después de la transposición a números de recursos similares en el entorno informático distinto del sistema escalable basado en contenedores, como base para los parámetros de definición de implementación del sistema escalable basado en contenedores, en particular el sistema de gestión de contenedores (385).

Además, al ejecutar la seguridad y las API individuales o las características de soporte del servicio de transacciones en contenedores discretos, el sistema escalable basado en contenedores aumenta la seguridad, al limitar el acceso a datos y recursos protegidos según sea necesario. Adicionalmente, las características de seguridad de la aplicación heredada inicial son trasladadas al conjunto de microservicios disponibles, y pueden ser identificadas e incluidas específicamente con microservicios (345) por el optimizador de definición de microservicios.

Los contenedores en un sistema escalable basado en contenedores, tal como el de tipo general representado en la figura 1B puede funcionar sin un hipervisor, lo que permite que el sistema escalable basado en contenedores funcione de manera más eficiente que un sistema, tal como una máquina virtual como el tipo representado en la figura 1A, en el que también deben operar componentes adicionales, tales como un hipervisor o múltiples copias del OS.

Un sistema, de acuerdo con la descripción anterior, puede implementarse en instrucciones informáticas almacenadas

en un medio no transitorio, tal como un medio de almacenamiento informático en un servidor o grupo de servidores, o en un conjunto de agrupaciones de servidores. Las instrucciones del ordenador pueden ser almacenadas en un medio de almacenamiento fijo o extraíble no volátil para su instalación en dicho sistema. En una realización, el depósito de código fuente (310), el depósito de definición de estado de transacción (340) y el depósito de definición dinámica (440) se almacenan en un sistema de depósito común, mientras que el depósito de binarios (330), el depósito de imágenes de transacción (360), el depósito de componentes complementarios (450) y el depósito de imágenes de contenedor (370) se almacenan en un sistema de depósito de imágenes de binarios común. En otra realización, el depósito de imágenes del contenedor (370) es instanciado en una plataforma separada. Dependiendo de la escala y de las necesidades del sistema, se pueden utilizar diferentes números de sistemas de depósito, y los depósitos de fuentes y de binarios pueden ser compartidos o estar separados en distintos sistemas de depósito.

Las instrucciones y/o los datos pueden ser almacenados de una manera habitual. Por ejemplo, las imágenes de binarios pueden ser almacenadas en el disco en la estructura jerárquica habitual de un sistema de archivos estándar. Los datos de la aplicación pueden ser almacenados en archivos regulares y/o en una base de datos estructurada (relacional, jerárquica, etc.).

De acuerdo con otro aspecto de la invención, se da a conocer un método para producir y/o mantener un sistema escalable basado en contenedores que realiza las operaciones de una aplicación heredada monolítica. La figura 6 es un diagrama de flujo de ciertas etapas de dicho método. Sin embargo, cualquier función descrita anteriormente en conexión con el sistema escalable basado en contenedores también puede estar incluida en el método. Además, aunque el método no se limita a su utilización con ningún sistema particular, se puede implementar en el sistema escalable basado en contenedores descrito anteriormente.

El método 600 incluye la etapa 605, en la que se analiza sintácticamente una aplicación heredada monolítica y los archivos de programa se dividen automáticamente. En la etapa 610, se identifican los programas raíz de transacciones. En la etapa 615, que puede ocurrir antes o después de la etapa 610, se identifican las interdependencias entre programas. Las etapas 610 y 615 pueden ocurrir simultáneamente para diferentes transacciones en una pluralidad de transacciones.

A continuación, en la etapa 620, se identifican una pluralidad de árboles de llamadas a transacciones. Preferiblemente, esta pluralidad de árboles de llamadas a transacciones representan todas las transacciones posibles en la aplicación heredada monolítica o todas las transacciones posibles en una parte secundaria definida de la aplicación heredada monolítica.

En la etapa 625, la pluralidad de árboles de llamadas a transacciones se utiliza para crear una pluralidad de vectores de definición de transacciones que se almacenan, por ejemplo, en un depósito de definición de estados de transacciones.

En la etapa 650, un analizador de registro de actividades determina qué programas se utilizan realmente en todas las transacciones posibles en la aplicación heredada monolítica, o en todas las transacciones posibles en una parte secundaria definida de la aplicación heredada monolítica. Si solo se utiliza una parte secundaria definida de la aplicación monolítica heredada, normalmente será la misma, que incluirá la totalidad o se superpondrá, como mínimo parcialmente, con la parte secundaria de la etapa 625. El analizador de registros de actividad puede utilizar registros de actividad heredados de la aplicación heredada monolítica tal como se ejecuta en su entorno original para determinar qué programas se utilizan realmente en las transacciones. El analizador de registro de actividades puede utilizar alternativamente un emulador para ejecutar la aplicación heredada monolítica con el fin de determinar qué programas se utilizan realmente en las transacciones. En algunos métodos, los mismos o diferentes analizadores de registro de actividad pueden utilizar tanto registros de actividad heredados como un emulador para determinar qué programas se utilizan realmente en las transacciones. En base a los resultados, se crea un depósito de definiciones dinámicas. El depósito de definiciones dinámicas contiene un registro de programas utilizados para cada transacción en una pluralidad de transacciones. En algunas realizaciones, este registro puede incluir una pluralidad de vectores de definición dinámica. El depósito de definiciones dinámicas puede estar definido con respecto al depósito de definiciones de estado de la transacción, o puede estar creado de manera independiente.

En la etapa 630, la pluralidad de vectores de definición de transacción de la etapa 625 son comparados con el depósito de definición dinámica de la etapa 650 mediante un optimizador de definición de microservicio, y los programas que no se utilizan realmente en una transacción son eliminados de cada vector de definición de transacción para crear una pluralidad de vectores de definición de microservicio correspondientes a la pluralidad de transacciones.

En la etapa 635, el optimizador de definición de microservicio determina si se producirá una optimización adicional. Si se va a producir una optimización adicional, entonces en la etapa 640, como mínimo, uno de entre la pluralidad de vectores de definición de microservicio se optimiza aún más y, a continuación, en la etapa 645, es proporcionado a un generador de imágenes de microservicio. Si no se va a producir una optimización adicional para ninguno de entre la pluralidad de vectores de definición de microservicio, entonces en la etapa 645, el vector de definición de microservicio es proporcionado a un generador de imágenes de microservicio. Independientemente de si se produce la optimización para cualquiera de los vectores de definición de microservicio, la pluralidad de vectores de definición de microservicio derivados de la pluralidad de vectores de transacción son proporcionados al generador de imágenes de microservicio

en la etapa 645.

En la etapa 655, el generador de imágenes de microservicio toma cada vector de definición de microservicio de entre la pluralidad de vectores de definición de microservicio y localiza el código fuente compilado correspondiente, compilado para ser ejecutado en el entorno informático heredado desde un depósito de binarios para formar una imagen de microservicio en un depósito de imágenes de microservicio. La imagen del microservicio también puede contener más información y artefactos utilizados por los programas que contiene. Después de que se completa la etapa 655, el depósito de imágenes de microservicio contiene preferiblemente una pluralidad de imágenes de microservicio correspondientes a cada una de una pluralidad de transacciones posibles en la aplicación monolítica heredada o en una parte secundaria definida de la misma.

En la etapa 660, se crea un depósito de componentes complementarios a partir de imágenes separadas de elementos de un emulador heredado. Los elementos separados corresponden a diferentes funciones del emulador heredado. Las imágenes de los elementos del OS asociados con el emulador heredado también se pueden almacenar en el depósito de componentes complementarios.

En la etapa 665, un generador de contenedores forma una imagen de contenedor para cada microservicio o un conjunto de microservicios utilizando imágenes del depósito de imágenes de microservicio junto con imágenes del depósito de componentes complementarios de elementos de emulador del emulador heredado utilizado para ejecutar el o los microservicios. Otras imágenes del depósito de componentes complementarios, tales como las imágenes de elementos del sistema operativo asociados con los elementos del emulador heredado, también se pueden colocar en la imagen del contenedor. Los elementos del emulador pueden ser seleccionados identificando firmas de llamadas a funciones o programas en los binarios de la imagen del microservicio y que incluyen elementos del emulador capaces de realizar las funciones llamadas u operar con los programas llamados. En ciertas realizaciones, como mínimo, un binario en, como mínimo, una imagen de microservicio en cada imagen de contenedor puede ser alterado para formar una imagen de microservicio optimizada de un emulador heredado, en la que la firma de una llamada en la imagen del binario del microservicio es sustituida por instrucciones para llamar a la misma función o funciones en el emulador heredado.

En la etapa 670, la pluralidad de imágenes de contenedores es almacenada en un depósito de imágenes de contenedores.

En la etapa 675, un sistema de gestión de contenedores almacena en un contenedor, como mínimo, una imagen de contenedor en el depósito de imágenes de contenedor. El sistema de gestión de contenedores puede utilizar la información de un analizador de registros de actividad, así como las propias imágenes de microservicio. Preferiblemente, cada imagen de contenedor se activa en, como mínimo, un contenedor. A cada imagen de contenedor se le puede asignar una asignación de recursos que se refleja en los recursos asignados al contenedor o contenedores en los que está contenida.

En la etapa 680, se ejecuta, como mínimo, un microservicio en un contenedor en el sistema de gestión de contenedores.

REIVINDICACIONES

1. Un sistema escalable basado en contenedores implementado en instrucciones informáticas almacenadas en un medio no transitorio, comprendiendo el sistema:

un depósito de códigos fuente, que contiene el código fuente de una aplicación heredada monolítica que contiene una pluralidad de programas ejecutables en un entorno informático heredado, para realizar una pluralidad de transacciones;

un analizador de código fuente, operable para analizar el código fuente e identificar, para cada transacción en la pluralidad de transacciones, un vector de definición de transacción que identifica cada programa potencialmente llamado durante la transacción, para crear una pluralidad de vectores de definición de transacción;

un depósito de definición de estado de la transacción, operable para almacenar la pluralidad de vectores de definición de transacción;

un analizador de registro de actividades, operable para crear un depósito de definición dinámica que identifica qué programas son utilizados realmente por la aplicación heredada monolítica cuando es realizada en, como mínimo, un subconjunto de la pluralidad de transacciones;

un optimizador de definición de microservicio, operable para comparar la pluralidad de vectores de definición de transacciones con el depósito de definición dinámica y eliminar programas no utilizados de la pluralidad de vectores de definición de transacciones para llegar a una pluralidad correspondiente de vectores de definición de microservicios que definen una pluralidad de microservicios;

un generador de imágenes de microservicio, operable para, para cada vector de definición de microservicio de entre la pluralidad de vectores de definición de microservicio, localizar para cada programa identificado por el vector de definición de microservicio, binarios de código fuente compilados, compilados para ser ejecutados en el entorno informático heredado y para ser combinados, para cada vector de definición de microservicio de entre la pluralidad de vectores de definición de microservicio, binarios compilados correspondientes para generar una imagen de microservicio para formar una pluralidad de imágenes de microservicio;

un depósito de imágenes de microservicio, operable para almacenar la pluralidad de imágenes de microservicio;

un depósito de componentes complementarios, operable para almacenar un conjunto de imágenes de binarios de elementos de emulador de un emulador heredado que, en conjunto, son menos que un emulador heredado completo, correspondiendo dichas imágenes a una pluralidad de funciones o conjuntos de funciones de dicho entorno informático heredado, y siendo ejecutables dichas imágenes en un entorno informático distinto caracterizado por un conjunto de instrucciones distinto del conjunto de instrucciones del entorno heredado;

un generador de contenedores, operable para formar una imagen de contenedor para cada microservicio o un conjunto de microservicios en la pluralidad de microservicios, utilizando la imagen de microservicio correspondiente o las imágenes del depósito de imágenes de microservicio, y utilizando archivos de imagen del depósito de componentes complementarios para los elementos de emulador del emulador heredado correspondiente a funciones o conjuntos de funciones empleadas por el microservicio o el conjunto de microservicios cuando se ejecutan, identificadas por firmas de llamadas en los binarios en el microservicio o conjunto de microservicios, para crear una pluralidad de imágenes de contenedor;

un depósito de imágenes de contenedores, operable para almacenar la pluralidad de imágenes de contenedores ejecutables en el entorno informático distinto; y

un sistema de gestión de contenedores, operable para crear, como mínimo, un contenedor para su ejecución en el entorno informático distinto y para ejecutar, como mínimo, una imagen de contenedor almacenada en el depósito de imágenes de contenedor en el, como mínimo, un contenedor.

2. El sistema escalable basado en contenedores de la reivindicación 1, en el que el analizador de registro de actividad puede ser operado para crear una pluralidad de vectores de definición de transacciones dinámicas que corresponden, como mínimo, a una parte de la pluralidad de vectores de definición de transacciones, y en el que el optimizador de definición de microservicio compara cada vector de definición dinámica de transacción con cada vector de definición de transacción correspondiente, para crear la pluralidad de vectores de definición de microservicio; y/o

en el que el analizador de registro de actividad utiliza registros de actividad heredados de la aplicación heredada monolítica generada ejecutando la aplicación heredada monolítica en el entorno informático heredado.

3. El sistema escalable basado en contenedores de cualquiera de las reivindicaciones anteriores, en el que el analizador de registro de actividad utiliza un emulador para ejecutar la aplicación heredada monolítica para generar archivos de registro y para determinar qué programas son utilizados por la aplicación heredada monolítica durante la ejecución de transacciones; y/o

en el que el analizador de código fuente puede ser utilizado para utilizar información del analizador de registro de actividades para identificar los vectores de definición de transacciones; y/o

en el que el analizador de código fuente puede ser operado, además, para crear una pluralidad de tablas de traducción.

4. El sistema escalable basado en contenedores de cualquiera de las reivindicaciones anteriores, en el que el optimizador de definición de microservicio puede ser operado para optimizar aún más los vectores de definición de microservicio, y

en el que, opcionalmente, el optimizador de definición de microservicio puede ser operado para optimizar aún más los vectores de definición de microservicio creando vectores de definición de microservicio adicionales que contienen programas compartidos por más de una transacción en la pluralidad de transacciones.

5. El sistema escalable basado en contenedores de cualquiera de las reivindicaciones anteriores, que comprende, además, un depósito de binarios operable para almacenar el código fuente compilado que contiene binarios compilados para ser ejecutados en el entorno informático heredado, y

en el que, opcionalmente, el código fuente compilado en el depósito de binarios es compilado a partir del código fuente en el depósito de códigos fuente en archivos binarios.

6. El sistema escalable basado en contenedores de cualquiera de las reivindicaciones anteriores, en el que el entorno informático heredado comprende un sistema informático de almacenamiento virtual múltiple (MVS) o z/OS; y/o

en el que el depósito de componentes complementarios puede ser operado, además, para almacenar una pluralidad de imágenes de paquetes de software del sistema operativo utilizados por el emulador heredado, y en el que el generador de contenedores también coloca imágenes de cualquier paquete de software utilizado por un elemento particular del emulador heredado en una imagen de contenedor particular que contiene el elemento particular del emulador heredado; y/o

en el que el generador de contenedores puede ser operado, además, para sustituir las firmas de llamadas en los binarios en el microservicio o conjunto de microservicios por instrucciones para llamadas operables en el emulador heredado.

7. El sistema escalable basado en contenedores de cualquiera de las reivindicaciones anteriores, en el que el sistema de gestión de contenedores puede ser operado para crear una pluralidad de contenedores

en el que, opcionalmente, se instancia un conjunto de imágenes complementarias en un contenedor separado dentro de un pod común.

8. El sistema escalable basado en contenedores de la reivindicación 7, en el que más de una copia de, como mínimo, una imagen de contenedor, se activan en más de un contenedor separado; y/o

en el que el sistema de gestión de contenedores puede ser operado para variar el número de contenedores en la pluralidad de contenedores; y/o

en el que el sistema de gestión de contenedores puede ser operado para asignar diversos recursos a contenedores separados; y/o

en el que el sistema de gestión de contenedores puede ser operado para utilizar información del analizador de registro de actividad para determinar cómo se colocarán el número de copias de, como mínimo, una imagen de contenedor en más de un contenedor separado, para determinar el número de contenedores en la pluralidad de contenedores, y/o para determinar recursos para asignar a contenedores separados; y/o

en el que el sistema de gestión de contenedores puede ser operado para utilizar información de la utilización del sistema escalable basado en contenedores para determinar cómo se colocará el número de copias de, como mínimo, una imagen de contenedor en más de un contenedor separado, para determinar el número de contenedores en la pluralidad de contenedores y/o para determinar recursos para asignar a contenedores separados.

9. El sistema escalable basado en contenedores de cualquiera de las reivindicaciones anteriores, en el que el analizador de código fuente puede ser operado, además, para crear una o más bases de datos secundarias o grupos de bases de datos secundarias a partir de una base de datos de la aplicación heredada monolítica; y/o

en el que el generador de contenedores puede ser operado para colocar una o más bases de datos secundarias o grupos de bases de datos secundarias en uno o más contenedores; y/o

en el que, cuando se cambia el código fuente, el sistema basado en contenedores puede ser operado para actualizar automáticamente, como mínimo, una imagen de microservicio, como mínimo, una imagen de contenedor y, como mínimo, un contenedor para contener un binario actualizado en base al cambio de código fuente.

10. Un método para crear y operar un sistema escalable basado en contenedores, comprendiendo el método:

analizar una aplicación heredada monolítica ejecutable en un entorno informático heredado y dividir sus archivos de programa para crear una pluralidad de vectores de definición de transacciones correspondientes a una pluralidad de transacciones ejecutables por parte de la aplicación heredada monolítica e identificar, para cada transacción, todos los programas llamados por esa transacción;

almacenar la pluralidad de vectores de definición de transacciones en un depósito de estado de las transacciones;

para, como mínimo, una parte de la pluralidad de transacciones, crear un depósito de definición dinámica determinando qué programas se utilizan realmente cuando la transacción la realiza la aplicación heredada monolítica;

comparar la pluralidad de vectores de definición de transacciones con el depósito de definiciones dinámicas y eliminar programas no utilizados en una transacción de su vector de definición de transacciones correspondiente para llegar a un vector de definición de microservicio correspondiente para crear una pluralidad de vectores de definición de microservicios;

para cada vector de definición de microservicio de entre la pluralidad de vectores de definición de microservicio, localizar los correspondientes binarios de código fuente compilados, compilados para ser ejecutados en el entorno informático heredado y combinar dichos binarios para generar una imagen de microservicio que contiene el código fuente compilado correspondiente para formar una pluralidad de imágenes de microservicio;

almacenar la pluralidad de imágenes de microservicios en un depósito de imágenes de microservicios;

almacenar, en un depósito de componentes complementarios, imágenes de una pluralidad de elementos de un emulador heredado operable para ejecutar programas en un entorno informático diferente al entorno informático heredado, correspondiendo los elementos del emulador heredado a una pluralidad de funciones o conjuntos de funciones de la aplicación heredada monolítica;

formar una imagen de contenedor para cada microservicio o un conjunto de microservicios en la pluralidad de microservicios utilizando la imagen o imágenes de microservicio correspondientes del depósito de imágenes de microservicio y utilizando archivos de imagen del depósito de componentes complementarios para los elementos del emulador heredado correspondientes a funciones o conjuntos de funciones empleadas por el microservicio o conjunto de microservicios cuando son ejecutadas, identificadas por firmas de llamadas en los binarios en el microservicio o conjunto de microservicios, para crear una pluralidad de imágenes de contenedor;

almacenar las imágenes de contenedor en un depósito de imágenes de contenedor;

crear, como mínimo, un contenedor en el entorno informático diferente utilizando un sistema de gestión de contenedores y almacenar, como mínimo, una imagen del contenedor en el contenedor en una forma ejecutable en el entorno informático diferente; y

ejecutar el, como mínimo, un contenedor en el contenedor.

11. El método de la reivindicación 10, que comprende:

crear una pluralidad de vectores de definición de transacciones dinámicas que corresponden, como mínimo, a una parte de la pluralidad de vectores de definición de transacciones, utilizando el analizador de registro de actividades, y

comparar cada vector de definición de transacción dinámica con cada vector de definición de transacción correspondiente para crear la pluralidad de vectores de definición de microservicio utilizando el optimizador de definición de microservicio; y/o

que comprende que el analizador de registro de actividad utiliza registros de actividad heredados de la aplicación heredada monolítica generada al ejecutar la aplicación heredada monolítica en el entorno informático heredado.

12. El método de la reivindicación 10 u 11, que comprende que el analizador de registro de actividad utiliza un emulador para ejecutar la aplicación heredada monolítica para generar archivos de registro y determinar qué programas son utilizados por la aplicación heredada monolítica durante la ejecución de transacciones; y/o

que comprende que el analizador de código fuente utiliza información del analizador de registro de actividades para identificar los vectores de definición de transacciones; y/o

crear una pluralidad de tablas de traducción utilizando el analizador de código fuente.

13. El método de una cualquiera de las reivindicaciones 10 a 12, que comprende optimizar, además, los vectores de definición de microservicio utilizando el optimizador de definición de microservicio, y

que comprende, opcionalmente, optimizar aún más los vectores de definición de microservicio utilizando el optimizador de definición de microservicio creando vectores de definición de microservicio adicionales que contienen programas

compartidos por más de una transacción en la pluralidad de transacciones.

14. El método de cualquiera de las reivindicaciones 10 a 13, que comprende, además, almacenar el código fuente compilado que contiene binarios compilados para ser ejecutados en el entorno informático heredado en un depósito de binarios, y

que comprende, opcionalmente, compilar el código fuente en el depósito de binarios desde el código fuente en el depósito de códigos fuente en archivos binarios.

15. El método de una cualquiera de las reivindicaciones 10 a 14, en el que el entorno informático heredado comprende un sistema informático de almacenamiento virtual múltiple (MVS) o z/OS; y/o

que comprende el depósito de componentes complementarios, que almacena una pluralidad de imágenes de paquetes de software del sistema operativo utilizados por el emulador heredado, y el generador de contenedores también coloca imágenes de cualquier paquete de software utilizado por un elemento particular del emulador heredado en una imagen de contenedor particular que contiene el elemento particular del emulador heredado; y/o

que comprende el generador de contenedores que sustituye las firmas de las llamadas en los binarios en el microservicio o el conjunto de microservicios por instrucciones para las llamadas operables en el emulador heredado.

16. El método de una cualquiera de las reivindicaciones 10 a 15, que comprende crear una pluralidad de contenedores utilizando el sistema de gestión de contenedores, y

que comprende, opcionalmente, la creación de instancias de un conjunto de imágenes complementarias en un contenedor separado dentro de un pod común.

17. El método de la reivindicación 16, que comprende activar más de una copia de, como mínimo, una imagen de contenedor en más de un contenedor separado; y/o

que comprende el sistema de gestión de contenedores que varía el número de contenedores en la pluralidad de contenedores; y/o

que comprende el sistema de gestión de contenedores que asigna diversos recursos a contenedores separados; y/o

que comprende el sistema de gestión de contenedores, que utiliza la información del analizador de registro de actividad para determinar cómo se colocarán el número de copias de, como mínimo, una imagen de contenedor en más de un contenedor separado, para determinar el número de contenedores en la pluralidad de contenedores, y/o para determinar recursos para asignar a contenedores separados; y/o

que comprende el sistema de gestión de contenedores, que utiliza la información de utilización del sistema escalable basado en contenedores para determinar cómo se colocarán el número de copias de, como mínimo, una imagen de contenedor en más de un contenedor separado, para determinar el número de contenedores en la pluralidad de contenedores y/o para determinar recursos para asignar a contenedores separados.

18. El método de una cualquiera de las reivindicaciones 10 a 17, que comprende que el analizador de código fuente crea una o más bases de datos secundarias o grupos de bases de datos secundarias a partir de una base de datos de la aplicación monolítica heredada; y/o

que comprende que el generador de contenedores coloque una o más bases de datos secundarias o grupos de bases de datos secundarias en uno o más contenedores; y/o

que comprende, cuando se cambia el código fuente, actualizar automáticamente, como mínimo, una imagen de microservicio, como mínimo, una imagen de contenedor, y, como mínimo, un contenedor para contener un binario actualizado en base al cambio de código fuente.

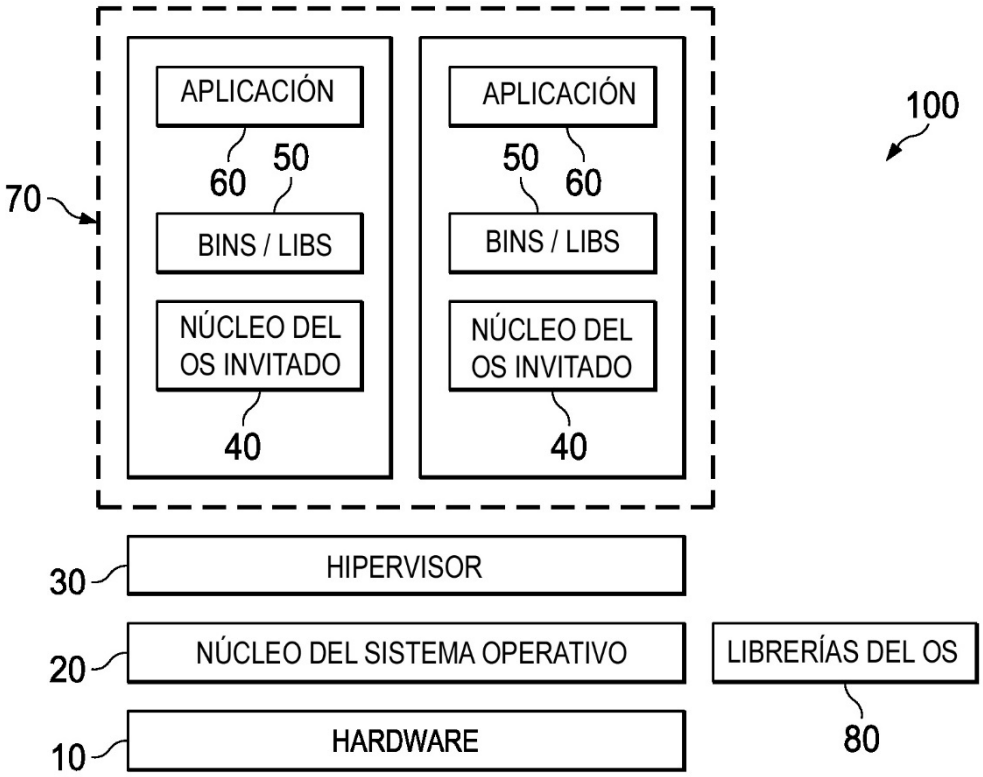

APLICACIÓN
APLICACIÓN
100
70
60
50
50
60
BINS / LIBS
BINS / LIBS
NÚCLEO DEL OS INVITADO
NÚCLEO DEL OS INVITADO
40
40
30
HIPERVISOR
20
NÚCLEO DEL SISTEMA OPERATIVO
LIBRERÍAS DEL OS
10
HARDWARE
80

FIG. 1A

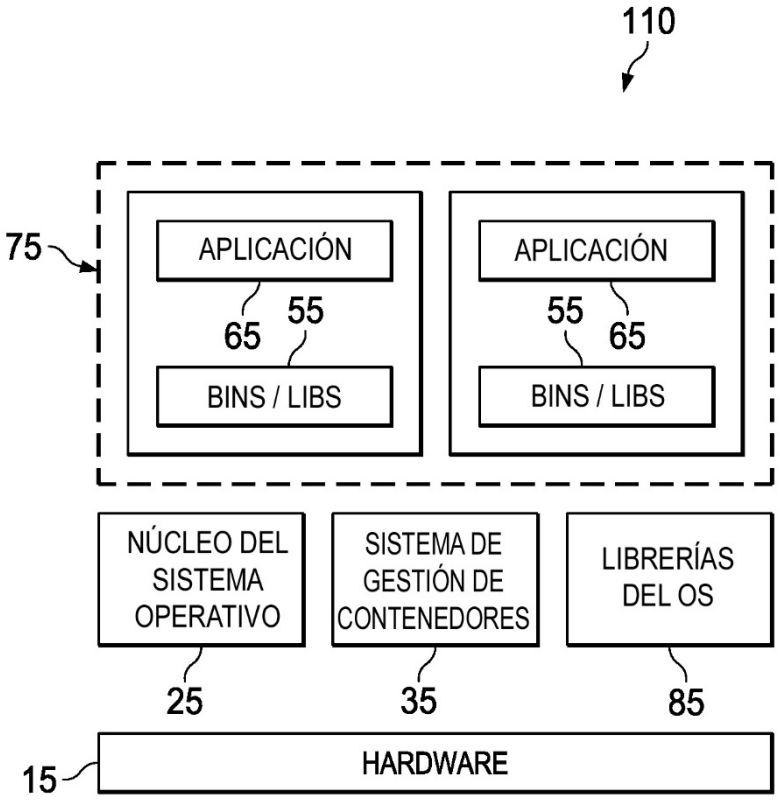

110
75
APLICACIÓN
APLICACIÓN
65
55
55
65
BINS / LIBS
BINS / LIBS
NÚCLEO DEL SISTEMA OPERATIVO
SISTEMA DE GESTIÓN DE CONTENEDORES
LIBRERÍAS DEL OS
25
35
85
15
HARDWARE

FIG. 1B

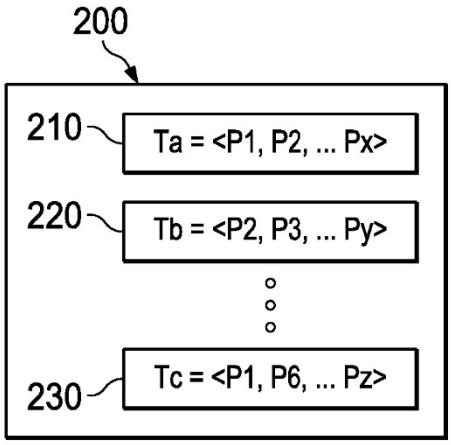


FIG. 2A

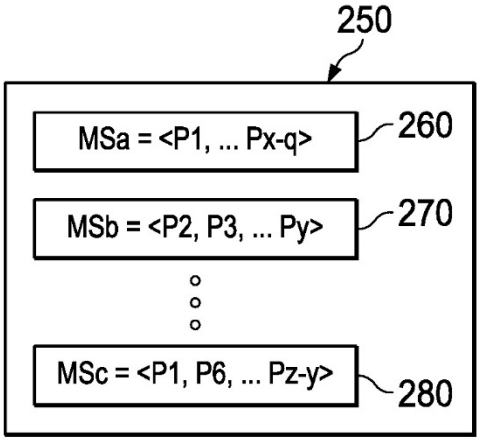


FIG. 2B

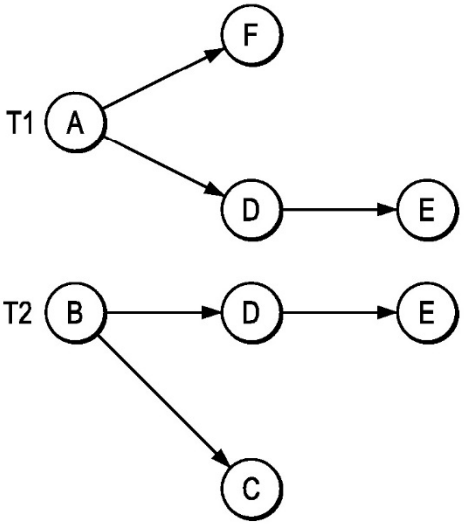


FIG. 4

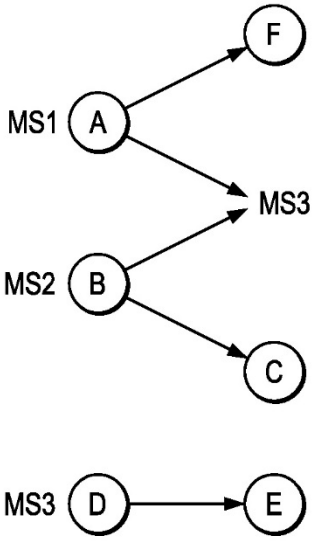


FIG. 5

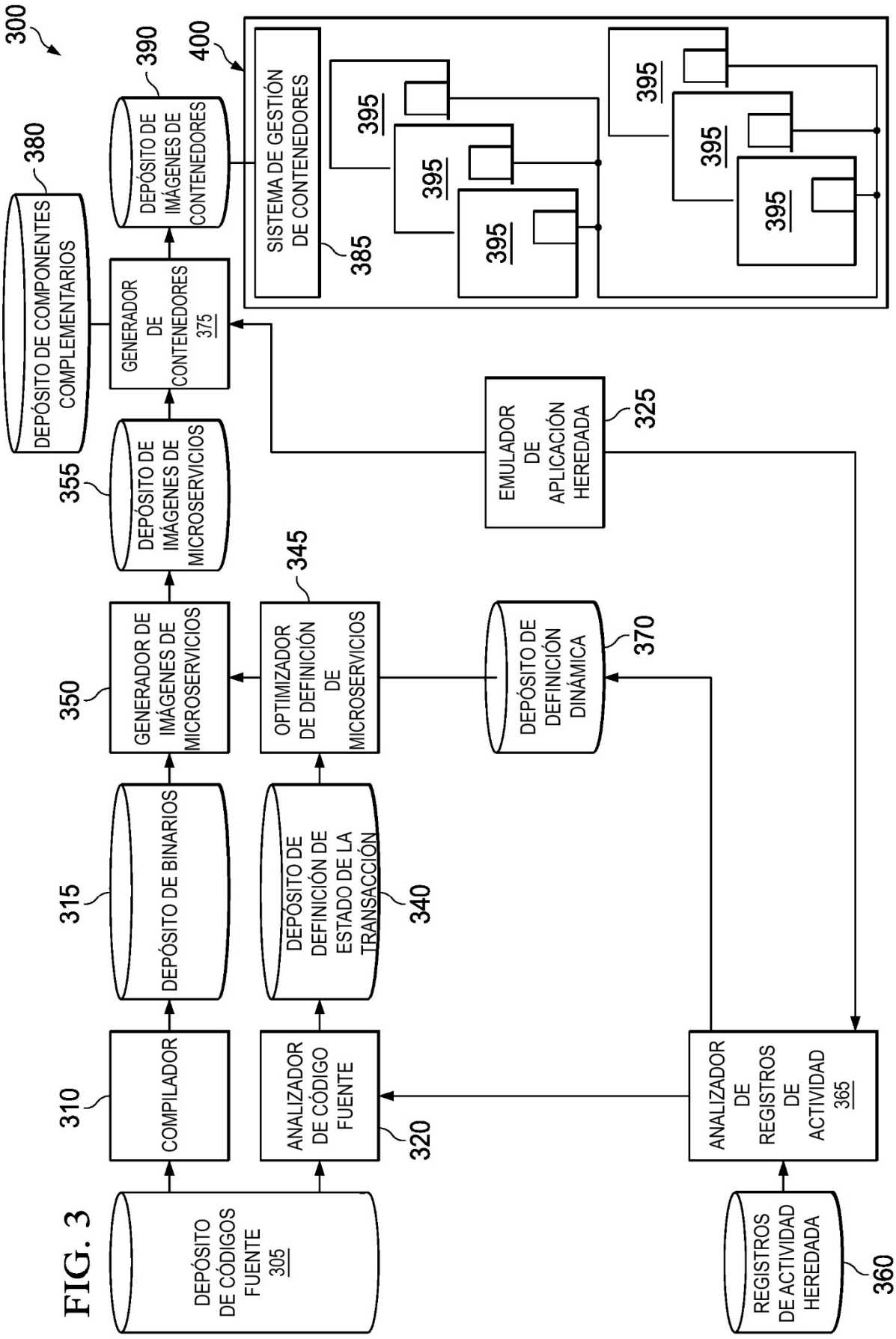
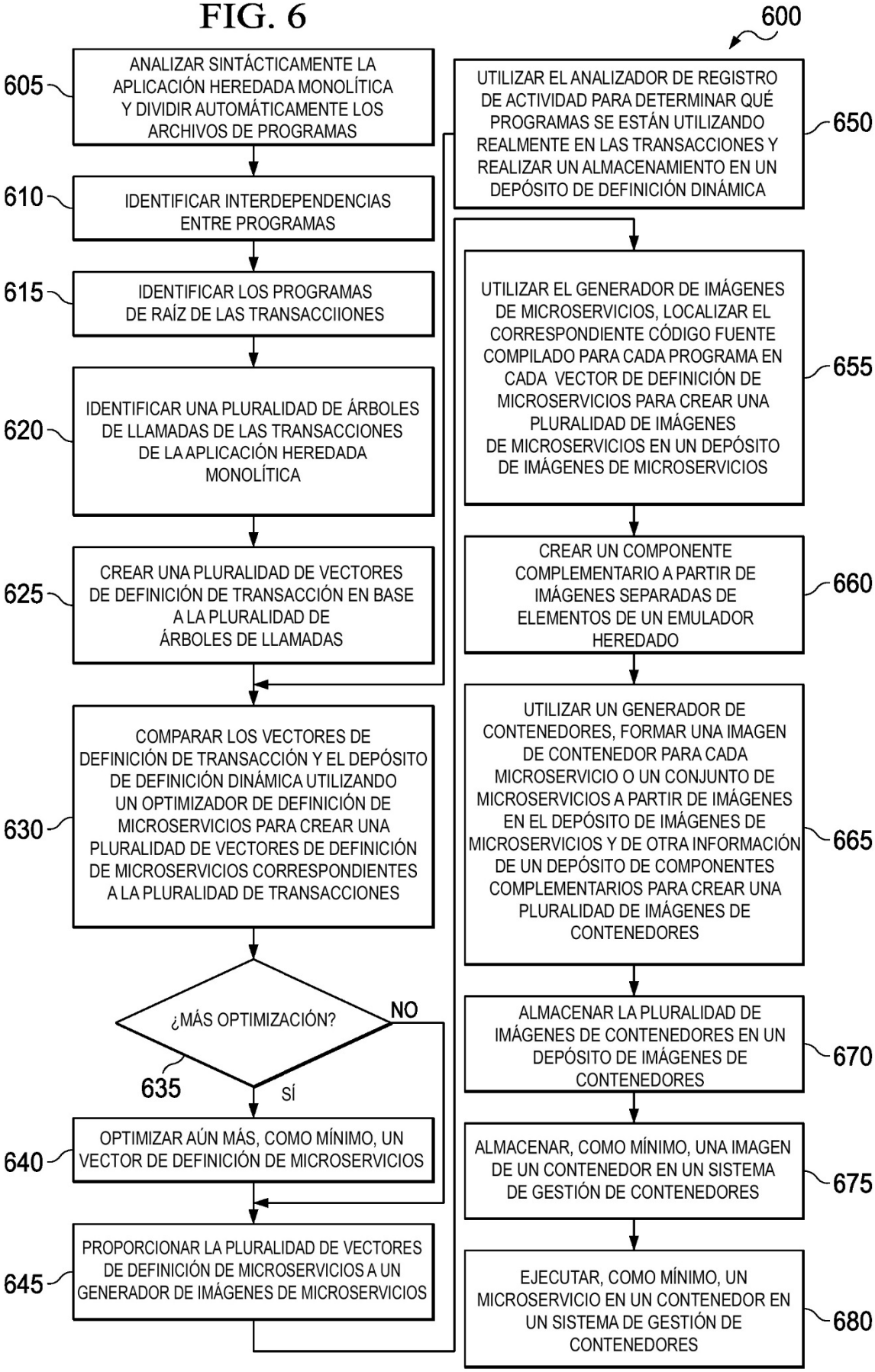

300
DEPÓSITO DE CÓDIGOS FUENTE 305
COMPILADOR 310
DEPÓSITO DE BINARIOS 315
GENERADOR DE IMÁGENES DE MICROSERVICIOS 350
DEPÓSITO DE IMÁGENES DE MICROSERVICIOS 355
GENERADOR DE CONTENEDORES 375
DEPÓSITO DE IMÁGENES DE CONTENEDORES 390
DEPÓSITO DE COMPONENTES COMPLEMENTARIOS 380
400
SISTEMA DE GESTIÓN DE CONTENEDORES 385
395
395
395
395
395
395
ANALIZADOR DE CÓDIGO FUENTE 320
OPTIMIZADOR DE DEFINICIÓN DE MICROSERVICIOS 345
DEPÓSITO DE DEFINICIÓN DE ESTADO DE LA TRANSACCIÓN 340
DEPÓSITO DE DEFINICIÓN DINÁMICA 370
EMULADOR DE APLICACIÓN HEREDADA 325
ANALIZADOR DE REGISTROS DE ACTIVIDAD 365
REGISTROS DE ACTIVIDAD HEREDADA 360

FIG. 3

FIG. 6


600
605
ANALIZAR SINTÁCTICAMENTE LA APLICACIÓN HEREDADA MONOLÍTICA Y DIVIDIR AUTOMÁTICAMENTE LOS ARCHIVOS DE PROGRAMAS
610
IDENTIFICAR INTERDEPENDENCIAS ENTRE PROGRAMAS
615
IDENTIFICAR LOS PROGRAMAS DE RAÍZ DE LAS TRANSACCIIONES
620
IDENTIFICAR UNA PLURALIDAD DE ÁRBOLES DE LLAMADAS DE LAS TRANSACCIONES DE LA APLICACIÓN HEREDADA MONOLÍTICA
625
CREAR UNA PLURALIDAD DE VECTORES DE DEFINICIÓN DE TRANSACCIÓN EN BASE A LA PLURALIDAD DE ÁRBOLES DE LLAMADAS
630
COMPARAR LOS VECTORES DE DEFINICIÓN DE TRANSACCIÓN Y EL DEPÓSITO DE DEFINICIÓN DINÁMICA UTILIZANDO UN OPTIMIZADOR DE DEFINICIÓN DE MICROSERVICIOS PARA CREAR UNA PLURALIDAD DE VECTORES DE DEFINICIÓN DE MICROSERVICIOS CORRESPONDIENTES A LA PLURALIDAD DE TRANSACCIONES
¿MÁS OPTIMIZACIÓN?
NO
635
SÍ
640
OPTIMIZAR AÚN MÁS, COMO MÍNIMO, UN VECTOR DE DEFINICIÓN DE MICROSERVICIOS
645
PROPORCIONAR LA PLURALIDAD DE VECTORES DE DEFINICIÓN DE MICROSERVICIOS A UN GENERADOR DE IMÁGENES DE MICROSERVICIOS
650
UTILIZAR EL ANALIZADOR DE REGISTRO DE ACTIVIDAD PARA DETERMINAR QUÉ PROGRAMAS SE ESTÁN UTILIZANDO REALMENTE EN LAS TRANSACCIONES Y REALIZAR UN ALMACENAMIENTO EN UN DEPÓSITO DE DEFINICIÓN DINÁMICA
655
UTILIZAR EL GENERADOR DE IMÁGENES DE MICROSERVICIOS, LOCALIZAR EL CORRESPONDIENTE CÓDIGO FUENTE COMPILADO PARA CADA PROGRAMA EN CADA VECTOR DE DEFINICIÓN DE MICROSERVICIOS PARA CREAR UNA PLURALIDAD DE IMÁGENES DE MICROSERVICIOS EN UN DEPÓSITO DE IMÁGENES DE MICROSERVICIOS
660
CREAR UN COMPONENTE COMPLEMENTARIO A PARTIR DE IMÁGENES SEPARADAS DE ELEMENTOS DE UN EMULADOR HEREDADO
665
UTILIZAR UN GENERADOR DE CONTENEDORES, FORMAR UNA IMAGEN DE CONTENEDOR PARA CADA MICROSERVICIO O UN CONJUNTO DE MICROSERVICIOS A PARTIR DE IMÁGENES EN EL DEPÓSITO DE IMÁGENES DE MICROSERVICIOS Y DE OTRA INFORMACIÓN DE UN DEPÓSITO DE COMPONENTES COMPLEMENTARIOS PARA CREAR UNA PLURALIDAD DE IMÁGENES DE CONTENEDORES
670
ALMACENAR LA PLURALIDAD DE IMÁGENES DE CONTENEDORES EN UN DEPÓSITO DE IMÁGENES DE CONTENEDORES
675
ALMACENAR, COMO MÍNIMO, UNA IMAGEN DE UN CONTENEDOR EN UN SISTEMA DE GESTIÓN DE CONTENEDORES
680
EJECUTAR, COMO MÍNIMO, UN MICROSERVICIO EN UN CONTENEDOR EN UN SISTEMA DE GESTIÓN DE CONTENEDORES