

(19)日本国特許庁(JP)

(12)公開特許公報(A)

(11)公開番号
特開2024-105342
(P2024-105342A)

(43)公開日 令和6年8月6日(2024.8.6)

(51)国際特許分類

G 0 6 F 8/60 (2018.01)

F I

G 0 6 F 8/60

審査請求 有 請求項の数 20 O L 外国語出願 (全43頁)			
(21)出願番号	特願2024-70583(P2024-70583)	(71)出願人	502303739
(22)出願日	令和6年4月24日(2024.4.24)		オラクル・インターナショナル・コーポレーション
(62)分割の表示	特願2022-199646(P2022-199646)の分割		アメリカ合衆国カリフォルニア州94065レッドウッド・シティ、オラクル・パークウェイ500
原出願日	平成30年9月28日(2018.9.28)	(74)代理人	110001195
(31)優先権主張番号	62/566,351		弁理士法人深見特許事務所
(32)優先日	平成29年9月30日(2017.9.30)	(72)発明者	カルダト, クラウディオ
(33)優先権主張国・地域又は機関	米国(US)		アメリカ合衆国、98074 ワシントン州、サマミッシュ、ノースイースト・トゥエンティス・ウェイ、21926
(特許庁注：以下のものは登録商標)		(72)発明者	ショール, ボリス
1. WINDOWS PHONE			アメリカ合衆国、98034 ワシントン州、カークランド、ノースイースト・
2. iOS			最終頁に続く

(54)【発明の名称】 APIレジストリにおけるバックエンドサービスエンドポイントからAPI関数へのバインディング

(57)【要約】 (修正有)

【課題】 コンテナ環境内の登録されたサービスエンドポイントに対するアプリケーションプログラミングインターフェイス (API) 関数を与える方法を提供する。

【解決手段】 方法は、APIレジストリにおいてAPI定義のアップロードを受けるステップを含む。API定義は、コンテナ環境にデプロイされたコンテナにカプセル化された第1のサービスのエンドポイントと、1つ以上のAPI関数と、を含む。方法はまた、APIレジストリが、1つ以上のAPI関数とサービスのエンドポイントとの間のバインディングを作成するステップと、APIレジストリが、第1のサービスの使用を求める要求を第2のサービスから受けるステップと、APIレジストリが、1つ以上のAPI関数を第2のサービスに与えるステップと、を含む。

【選択図】 図7A

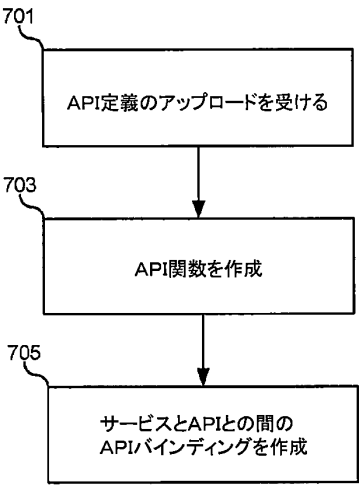


FIG. 7A

【特許請求の範囲】

【請求項 1】

コンテナ環境内の登録されたサービスエンドポイントに対するアプリケーションプログラミングインターフェイス（API）関数を与える方法であって、前記方法は、

APIレジストリにおいてAPI定義を受けるステップを含み、前記API定義は、

コンテナ環境にデプロイされたコンテナにカプセル化された第1のサービスのエンドポイントと、

1つ以上のAPI関数とを含み、前記方法はさらに、

前記APIレジストリが、前記1つ以上のAPI関数と前記サービスのエンドポイントとの間のバインディングを作成するステップと、

前記APIレジストリが、前記第1のサービスの使用を求める要求を第2のサービスから受けるステップと、

前記APIレジストリが、前記1つ以上のAPI関数を前記第2のサービスに与えるステップとを含む、方法。

【請求項 2】

前記1つ以上のAPI関数を含む前記第2のサービスのためのクライアントライブラリを生成するステップをさらに含む、請求項1に記載の方法。

【請求項 3】

前記クライアントライブラリは、前記API関数を実現するメンバ関数とともにオブジェクトクラスを含む、請求項2に記載の方法。

【請求項 4】

前記クライアントライブラリは、前記第1のサービスのエンドポイントに対するダイレクトコールを含む、請求項2に記載の方法。

【請求項 5】

前記クライアントライブラリは、前記1つ以上のAPI関数のうちの1つが前記第2のサービスによってコールされたときに前記第1のサービスのエンドポイントを実行時にアップデートするコードを含む、請求項2に記載の方法。

【請求項 6】

前記クライアントライブラリは、前記1つ以上のAPI関数の1つ以上のパラメータとして受けたデータを、前記第1のサービスのエンドポイントに対するHTTPコールに再フォーマットする、請求項2に記載の方法。

【請求項 7】

前記クライアントライブラリは、前記第1のサービスのエンドポイントの前記HTTPコールに加えるための追加情報を第3のサービスに要求する、請求項6に記載の方法。

【請求項 8】

前記クライアントライブラリは、前記第1のサービスに対する失敗コールをリトライするコードを含む、請求項2に記載の方法。

【請求項 9】

前記クライアントライブラリは、結果値を前記第1のサービスの結果セットから抽出し前記結果値を前記1つ以上のAPI関数に対するリターン値としてパッケージングするコードを含む、請求項2に記載の方法。

【請求項 10】

1つ以上のプロセッサによって実行されると前記1つ以上のプロセッサに動作を実行させる命令を含む非一時的なコンピュータ読取可能媒体であって、前記動作は、

APIレジストリにおいてAPI定義を受けることを含み、前記API定義は、

コンテナ環境にデプロイされたコンテナにカプセル化された第1のサービスのエンドポイントと、

1つ以上のAPI関数とを含み、前記動作はさらに、

前記APIレジストリが、前記1つ以上のAPI関数と前記サービスのエンドポイントとの間のバインディングを作成することと、

10

20

30

40

50

前記 A P I レジストリが、前記第 1 のサービスの使用を求める要求を第 2 のサービスから受けることと、

前記 A P I レジストリが、前記 1 つ以上の A P I 関数を前記第 2 のサービスに与えることとを含む、非一時的なコンピュータ読取可能媒体。

【請求項 1 1】

前記第 1 のサービスのエンドポイントは、

I P アドレスと、

ポート番号とを含む、請求項 1 0 に記載の非一時的なコンピュータ読取可能媒体。

【請求項 1 2】

前記コンテナ環境は、コンテナにカプセル化された複数のサービスを含む、請求項 1 0 に記載の非一時的なコンピュータ読取可能媒体。

【請求項 1 3】

前記 A P I レジストリは、前記コンテナ環境内のコンテナにカプセル化されたサービスとしてデプロイされる、請求項 1 0 に記載の非一時的なコンピュータ読取可能媒体。

【請求項 1 4】

前記 A P I レジストリは、

統合開発環境（I D E）において開発中のサービスと、

前記コンテナ環境において既にデプロイされているサービスとが、利用できる、請求項 1 0 に記載の非一時的なコンピュータ読取可能媒体。

【請求項 1 5】

前記動作はさらに、前記 A P I レジストリが、前記コンテナ環境にデプロイされているコンテナ化されたサービスの任意の利用できるエンドポイントを識別することを含む、請求項 1 0 に記載の非一時的なコンピュータ読取可能媒体。

【請求項 1 6】

1 つ以上のプロセッサと、

1 つ以上のメモリデバイスとを備えるシステムであって、前記 1 つ以上のメモリデバイスは、前記 1 つ以上のプロセッサによって実行されると前記 1 つ以上のプロセッサに動作を実行させる命令を含み、前記動作は、

A P I レジストリにおいて A P I 定義を受けることを含み、前記 A P I 定義は、

コンテナ環境にデプロイされたコンテナにカプセル化された第 1 のサービスのエンドポイントと、

1 つ以上の A P I 関数とを含み、前記動作はさらに、

前記 A P I レジストリが、前記 1 つ以上の A P I 関数と前記サービスのエンドポイントとの間のバインディングを作成することと、

前記 A P I レジストリが、前記第 1 のサービスの使用を求める要求を第 2 のサービスから受けることと、

前記 A P I レジストリが、前記 1 つ以上の A P I 関数を前記第 2 のサービスに与えることとを含む、システム。

【請求項 1 7】

前記動作はさらに、前記 A P I レジストリに登録されたすべてのサービスに対する A P I 関数のリストをユーザインターフェイスに表示させることを含む、請求項 1 6 に記載のシステム。

【請求項 1 8】

前記第 1 のサービスの使用を求める前記第 2 のサービスからの要求は、前記ユーザインターフェイスに表示された前記 1 つ以上の A P I 関数のうちの 1 つの選択を受けることによって受ける、請求項 1 7 に記載のシステム。

【請求項 1 9】

前記ユーザインターフェイスはコマンドラインインターフェイスを含む、請求項 1 7 に記載のシステム。

【請求項 2 0】

10

20

30

40

50

前記ユーザインターフェイスはグラフィカルユーザインターフェイスを含む、請求項 17 に記載のシステム。

【発明の詳細な説明】

【技術分野】

【0001】

関連出願の相互参照

本願は、本明細書に引用により援用する 2017 年 9 月 30 日出願の米国仮出願第 62 / 566 , 351 号に基づく利益を主張する。本願はまた、本願と同日に出願され本願と共通の譲受人に譲渡された以下の出願に関連し、これらの出願各々も本明細書に引用により援用する。

・ 2018 年 9 月 28 日に出願され API REGISTRY IN A CONTAINER PLATFORM PROVIDING PROPERTY-BASED API FUNCTIONALITY と題された米国特許出願第 16 / 147 , 305 号 (

代理人整理番号 088325-1090746)

・ 2018 年 9 月 28 日に出願され DYNAMIC NODE REBALANCING BETWEEN CONTAINER PLATFORMS と題された米国特許出願 16 / 147 , 343 号 (代理人整理番号 088325-1090747)

・ 2018 年 9 月 28 日に出願され OPTIMIZING REDEPLOYMENT OF FUNCTIONS AND SERVICES ACROSS MULTIPLE CONTAINER PLATFORMS AND INSTALLATIONS と題された米国特許出願第

16 / 147 , 332 号 (代理人整理番号 088325-1090748)

・ 2018 年 9 月 28 日に出願され REAL-TIME DEBUGGING INSTANCES IN A DEPLOYED CONTAINER PLATFORM と題された米国特許出願 16 / 147 , 351 号 (代理人整理番号 088325-1090753)

【背景技術】

【0002】

背景

理論上、任意の形態のコンテナは、情報のパッケージングおよび情報との対話のための標準化された方法を表している。コンテナは、互いに分離することが可能であり、相互汚染 (コンタミネーション) のいかなるリスクも伴うことなく並列に使用することが可能である。現代のソフトウェアの世界において、「コンテナ」という用語は固有の意味を獲得している。Docker (登録商標) コンテナのようなソフトウェアコンテナは、1つのソフトウェアを論理的にカプセル化し定義するソフトウェア構造である。コンテナにカプセル化される最も一般的なタイプのソフトウェアは、アプリケーション、サービス、またはマイクロサービスである。現代のコンテナはまた、オペレーティングシステム、ライブラリ、ストレージボリューム、構成ファイル、アプリケーションバイナリ、および、典型的なコンピューティング環境において見出されるであろうテクノロジスタックのその他の部分のような、アプリケーション / サービスが動作するのに必要なソフトウェアサポートすべてを含む。そのため、このコンテナ環境を使用することにより、各々が自身のサービスを任意の環境で実行する複数のコンテナを作成することができる。コンテナは、プロダクションデータセンター、オンプレミスデータセンター、クラウドコンピューティングプラットフォームなどにおいて、いかなる変更も伴うことなくデプロイすることができる。クラウド上にコンテナを立ち上げることは、ローカルワークステーション上にコンテナを立ち上げることと同一である。

【0003】

現代のサービス指向アーキテクチャおよびクラウドコンピューティングプラットフォームは、大きなタスクを多数の小さな特定のタスクに分割する。コンテナをインスタンス化することによって個々の特定のタスクに集中することができ、さらに、複数のコンテナが協働することによって高度なアプリケーションを実現することができる。これはマイクロサービスアーキテクチャと呼ばれることがあり、各コンテナは、独立してアップグレード

10

20

30

40

50

することが可能なさまざまなバージョンのプログラミング言語およびライブラリを使用することができる。コンテナ内の処理には分離されているという性質があるので、コンテナは、より大きくよりモノリシックなアーキテクチャに対して行われる変更と比較すると、手間またはリスクがほとんどない状態でアップグレードおよびリプレイスが可能である。コンテナプラットフォームを実行するために仮想マシンを使用できるが、このマイクロサービスアーキテクチャの実行において、コンテナプラットフォームは従来の仮想マシンよりも遥かに効率的である。

【発明の概要】

【課題を解決するための手段】

【0004】

10

簡単な概要

いくつかの実施形態において、コンテナ環境内の登録されたサービスエンドポイントに対するアプリケーションプログラミングインターフェイス (Application Programming Interface) (API) 関数を与える方法は、APIレジストリにおいて、コンテナ環境に

デプロイされたコンテナにカプセル化された第1のサービスのエンドポイントと、1つ以上のAPI関数とを含み得る、API定義を受けるステップを含み得る。この方法はまた、APIレジストリが、1つ以上のAPI関数とサービスのエンドポイントとの間のバインディングを作成するステップと、APIレジストリが、第1のサービスの使用を求める要求を第2のサービスから受けるステップと、APIレジストリが、1つ以上のAPI関数を第2のサービスに与えるステップとを含み得る。

20

【0005】

いくつかの実施形態において、1つ以上のプロセッサによって実行されると当該1つ以上のプロセッサに動作を実行させる命令を含み得る非一時的なコンピュータ読取可能媒体において、当該動作は、APIレジストリにおいて、コンテナ環境にデプロイされたコンテナにカプセル化された第1のサービスのエンドポイントと、1つ以上のAPI関数とを含み得る、API定義を受けることを含み得る。上記動作はまた、APIレジストリが、1つ以上のAPI関数とサービスのエンドポイントとの間のバインディングを作成することと、APIレジストリが、第1のサービスの使用を求める要求を第2のサービスから受けることと、APIレジストリが、1つ以上のAPI関数を第2のサービスに与えることとを含み得る。

30

【0006】

いくつかの実施形態において、システムは、1つ以上のプロセッサと、1つ以上のメモリデバイスとを含み得る。1つ以上のメモリデバイスは、1つ以上のプロセッサによって実行されると当該1つ以上のプロセッサに動作を実行させる命令を含み、当該動作は、APIレジストリにおいて、コンテナ環境にデプロイされたコンテナにカプセル化された第1のサービスのエンドポイントと、1つ以上のAPI関数とを含み得る、API定義を受けることを含む。上記動作はまた、APIレジストリが、1つ以上のAPI関数とサービスのエンドポイントとの間のバインディングを作成することと、APIレジストリが、第1のサービスの使用を求める要求を第2のサービスから受けることと、APIレジストリが、1つ以上のAPI関数を第2のサービスに与えることとを含み得る。

40

【0007】

いずれの実施形態にも、以下の特徴のうちのいずれかまたはすべてが、限定されることなく任意の組み合わせで含まれ得る。上記方法／動作はまた、1つ以上のAPI関数を含む第2のサービスのためのクライアントライブラリを生成することを含み得る。クライアントライブラリは、API関数を実現するメンバ関数とともにオブジェクトクラスを含み得る。クライアントライブラリは、第1のサービスのエンドポイントに対するダイレクトコールを含み得る。クライアントライブラリは、1つ以上のAPI関数のうちの1つが第2のサービスによってコールされたときに第1のサービスのエンドポイントを実行時にアップデートするコードを含み得る。クライアントライブラリは、1つ以上のAPI関数の

50

1つ以上のパラメータとして受けたデータを、第1のサービスのエンドポイントに対するH T T Pコールに再フォーマットし得る。クライアントライブラリは、第1のサービスのエンドポイントのH T T Pコールに加えるための追加情報を第3のサービスに要求し得る。クライアントライブラリは、第1のサービスに対する失敗コールをリトライするコードを含み得る。クライアントライブラリは、結果値を第1のサービスの結果セットから抽出し当該結果値を1つ以上のA P I関数に対するリターン値としてパッケージングするコードを含み得る。第1のサービスのエンドポイントは、I Pアドレスとポート番号とを含み得る。コンテナ環境は、コンテナにカプセル化された複数のサービスを含み得る。A P Iレジストリは、コンテナ環境内のコンテナにカプセル化されたサービスとしてデプロイされ得る。A P Iレジストリは、統合開発環境（Integrated Development Environm 10
ent）（I D E）において開発中のサービスと、コンテナ環境において既にデプロイされているサービスとが、利用できる。上記方法／動作はまた、A P Iレジストリが、コンテナ環境にデプロイされているコンテナ化されたサービスの任意の利用できるエンドポイントを識別することを含み得る。上記方法／動作はさらに、A P Iレジストリに登録されたすべてのサービスに対するA P I関数のリストをユーザインターフェイスに表示させることを含み得る。第1のサービスの使用を求める第2のサービスからの要求は、ユーザインターフェイスに表示された1つ以上のA P I関数のうちの1つの選択を受けることによって受けることができる。ユーザインターフェイスはコマンドラインインターフェイスを含み得る。ユーザインターフェイスはグラフィカルユーザインターフェイスを含み得る。

【0008】

本発明の性質および利点の一層の理解は、本明細書の残りの部分および図面を参照することによって実現するであろう。いくつかの図面で使用されている同様の参照番号は、同様の構成要素を示す。いくつかの例では、参照番号に添字を対応付けることにより、同様の複数の構成要素のうちの1つを示している。既存の添字を示すことなく参照番号に言及している場合は、そのような同様の複数の構成要素すべてに言及することを意図している。

【図面の簡単な説明】

【0009】

【図1】いくつかの実施形態に係る、コンテナプラットフォームにおけるサービスのための開発およびランタイム環境のソフトウェア構造および論理構成を示す図である。

【図2】本明細書に記載の実施形態を実行するために特別に設計された専用コンピュータハードウェアシステムを示す図である。

【図3】本明細書に記載の実施形態のうちのいくつかで使用されるコンテナプラットフォームに固有であってもよいデータ組織を示す図である。

【図4】いくつかの実施形態に係る、I D Eおよびプロダクション／ランタイム環境にデプロイすることができるA P Iレジストリを示す図である。

【図5】いくつかの実施形態に係る、実行時にコンテナプラットフォームとともに使用されるA P Iレジストリのデプロイを示す図である。

【図6A】いくつかの実施形態に係る、A P Iレジストリをデプロイする方法のフローチャートを示す図である。

【図6B】いくつかの実施形態に係る、A P Iレジストリが図6Aのフローチャートを用いてデプロイされるときコンテナプラットフォームのソフトウェア構造を示す図である。

【図7A】いくつかの実施形態に係る、A P Iレジストリにサービスを登録する方法のフローチャートを示す図である。

【図7B】いくつかの実施形態に係る、A P IレジストリにA P Iを登録するためのステップのハードウェア／ソフトウェア図を示す。

【図8】いくつかの実施形態に係る、A P Iレジストリに登録されたA P Iをブラウズおよび選択するためのグラフィカルインターフェイスおよびコマンドラインインターフェイスの例を示す図である。

【図 9】いくつかの実施形態に係る、APIレジストリに登録されたサービスおよびその対応する関数を使用する方法のフローチャートを示す図である。

【図 10】APIレジストリが如何にしてCreateUser()関数の選択をグラフィカルインターフェイスを介して受けることができるかを示す図である。

【図 11】いくつかの実施形態に係る、APIレジストリがあるサービスのために自動的に生成したクライアントライブラリの一例を示す図である。

【図 12】いくつかの実施形態に係る、サービスエンドポイントとAPI関数との間の動的バインディングを含むクライアントライブラリの実施形態を示す図である。

【図 13】いくつかの実施形態に係る、サービスコールのための入力データセットを完成させるために追加データを配置できるクライアントライブラリの実施形態を示す図である。

10

【図 14】いくつかの実施形態に係る、サービスをコールするときにリトライを処理することができるクライアントライブラリを示す図である。

【図 15】実施形態のうちのいくつかを実現するための分散型システムの簡略化されたブロック図を示す。

【図 16】実施形態の構成要素が提供するサービスをクラウドサービスとして提供し得るシステム環境の構成要素の簡略化されたブロック図を示す。

【図 17】各種実施形態を実現し得る具体例としてのコンピュータシステムを示す図である。

【発明を実施するための形態】

20

【0010】

詳細な説明

開発者が開発中にサービスを登録することを可能にするとともにデプロイ中およびデプロイ後双方においてこれらのサービスを他のサービスが利用できるようにする、統合開発環境（IDE）の一部であるアプリケーションプログラミングインターフェイス（API）レジストリの実施形態について説明する。APIレジストリは、コンテナプラットフォーム上のコンテナ化されたアプリケーションとして動作するオーケストレーションされたコンテナプラットフォームの一部としてデプロイすることが可能である。サービスまたはマイクロサービスが開発されコンテナプラットフォーム上のコンテナにデプロイされると、APIレジストリは、ディスカバリプロセスを実行することにより、利用できるサービスに対応するコンテナプラットフォーム内の利用できるエンドポイント（たとえばIPアドレスおよびポート番号）の場所を特定することができる。APIレジストリはまた、API定義ファイルのアップロードを受け入れることができ、API定義ファイルは、生のサービスエンドポイントを、APIレジストリを介して利用できるようにされるAPI関数にするために使用することができる。APIレジストリは、発見されたエンドポイントを、最新状態に保たれコンテナプラットフォーム内の他のサービスが利用できるようにされたAPI関数に動的にバインドすることができる。これは、API関数とサービスエンドポイントとの間のバインディングに対する任意の変更をAPIレジストリが管理している間、他のサービスが静的にコールできる安定したエンドポイントを提供する。これはまた、コンテナプラットフォーム内のサービスを使用するプロセスを簡略化する。HTTPコールに対するコードを記述する代わりに、新たなサービスは、APIインターフェイスを使用するだけで、登録されたサービスにアクセスすることができる。

30

【0011】

いくつかの実施形態において、IDEは、コンテナプラットフォーム内で利用することが可能でありAPIレジストリに登録されているサービスの場所を開発者が特定するための、ナビゲーション／ブラウズインターフェイスを提供することができる。開発中の新たなサービスのために、APIレジストリが既存のサービスにコールするとき、APIレジストリは、登録されているサービスとのやり取りのために必要なすべての機能を含む一組のクライアントライブラリを自動的に生成することができる。たとえば、いくつかの実施形態は、APIコールに対応するメンバ関数を含むオブジェクトクラスを生成することが

40

50

できる。開発中、新たなサービスは、これらのオブジェクトをインスタンス化するおよび／またはそれらのメンバ関数を使用するだけで、対応するAPIにコールすることができる。クライアントライブラリにおけるコードは、呼び出し側サービスと登録されたサービスのエンドポイントとの間の直接接続を支配し、このやり取りに必要なすべての機能を扱うコードを含み得る。たとえば、自動的に生成されたクライアントライブラリは、APIコールからのパラメータをサービスエンドポイントへのHTTPコールにパッケージングしフォーマットするためのコード、コールのためのパラメータセットを完成させるためにデータを配置するためのコード、情報を互換パケット（JSON、XMLなど）にパッケージングするためのコード、結果パケットを受けてパースするためのコード、リトライおよびエラー条件を扱うためのコードなどを含み得る。呼び出し側サービスの観点からすると、この機能すべてを扱うためのコードは、APIレジストリによって自動的に生成され、したがって、サービスコールの詳細を要約しカプセル化してクライアントライブラリオブジェクトにする。呼び出し側サービスに要求されるのは、APIレジストリが作成したクライアントライブラリオブジェクトのメンバ関数を実行することだけである。

10

【0012】

いくつかの実施形態において、APIレジストリは、登録されたサービスのランタイム実行を定義し得る一組のプロパティのアップロードを受け入れることもできる。この一組のプロパティは、開発中に、API定義ファイルとともにアップロードすることができる。これらのプロパティは、エンドツーエンド暗号化、使用／ロギング要件、ユーザ認証、オンデマンドサービスインスタンス化、高可用性のための複数のサービスデプロイメントインスタンス、レート／使用制限、およびその他のランタイム特徴等の、ランタイム特徴を定義することができる。APIレジストリは、開発中、デプロイ中、およびランタイム中に、コンテナ環境とやり取りすることによってこれらのプロパティが満たされることを保証できる。開発中、呼び出し側サービスのために自動的に生成されたクライアントライブラリは、暗号化コード、使用ロギングコード、および／またはユーザ認証サービスとのやり取り等の、これらのプロパティの実行に必要となり得るコードを含むことができる。登録されたサービスがデプロイされているとき、APIレジストリは、コンテナプラットフォームに対し、サービスおよび／または追加のロードバランシングモジュールの複数のインスタンスをインスタンス化することによってランタイム中のサービスの高い信頼性を保証するよう指示することができる。ランタイム中にサービスがコールされると、APIレジストリは、オンデマンドインスタンス化のためにサービスをインスタンス化させ、使用を抑えるために行うことができるAPIコールの数を制限し、その他のランタイム関数を実行することができる。

20

30

【0013】

図1は、いくつかの実施形態に係る、コンテナプラットフォームにおけるサービスのための開発およびランタイム環境のソフトウェア構造および論理構成を示す。この環境は、コンテナプラットフォーム上にデプロイされるサービスおよびマイクロサービスを開発するために使用し得るIDE102を含み得る。IDEは、新たなサービスの記述およびテストのためにサービス開発者が使用できる基本ツールすべてを統合し提供するソフトウェアスイートである。IDE102は、開発者がソースコード記述プロセスの記述、ナビゲート、統合、および視覚化を行うことを可能にする、グラフィカルユーザインターフェイス（GUI）、コード補完機能、およびナビゲート／ブラウズインターフェイスとともに、ソースコードエディタ106を含み得る。IDE102はまた、可変インターフェイス、即時可変インターフェイス、表現評価インターフェイス、メモリコンテンツインターフェイス、ブレークポイント可視化および機能、ならびにその他のデバッグ関数を含む、デバッガ110を含み得る。IDE102はまた、マシンコードをコンパイルしコンパイルしたマシンコードまたは解釈されたバイトコードを実行するためのコンパイラおよび／またはインタプリタ108を含み得る。コンパイラ／インタプリタ108は、開発者が別のビルド自動化構成のためのメイクファイル（makefiles）を使用／生成することを可能にするビルドツールを含み得る。IDE102のいくつかの実施形態は、コードライブラリ

40

50

1 1 2を含み得る。コードライブラリは、一般的なコード関数、オブジェクト、インターフェイス、および／または、開発中のサービスにリンクさせることができ複数の開発で再使用できるその他の構造を含む。

【0014】

サービスは、IDE 102内で、開発することができ、デプロイの準備が整うまで徹底的にテストすることができる。サービスはその後、プロダクション／デプロイメント環境104にデプロイすることができる。プロダクション／デプロイメント環境104は、専用ハードウェア、仮想マシン、およびコンテナ化されたプラットフォームを含む、多数の異なるハードウェアおよび／またはソフトウェア構造を含み得る。本開示よりも前において、サービス114がプロダクション／デプロイメント環境104にデプロイされたとき、サービス114は、IDE 102で使用されるツールの多くに対するランタイムアクセスを行うことができない。サービス114がプロダクション／デプロイメント環境104で実行するのに必要ないずれの機能も、コードライブラリ112からパッケージングしサービス114とともにプロダクション／デプロイメント環境104にデプロイする必要があった。加えて、一般的にサービス114は、デバッガ110の機能またはソースコードエディタ106からのソースコードのコピーのうちのいずれも伴うことなくデプロイされる。実際、サービス114は、ランタイム動作に必要な機能すべてとともにプロダクション／デプロイメント環境104にデプロイされるが、開発中にだけ使用された情報は取り除かれる。

【0015】

図2は、本明細書に記載の実施形態を実行するために特別に設計された専用コンピュータハードウェアシステムを示す。一例として、サービス114は、サービスとしてのインフラストラクチャ（Infrastructure as a Service）（IaaS）クラウドコンピューティング環境202にデプロイすることができる。これは、ネットワーク上に仮想化または共有コンピューティングリソースを提供するクラウドコンピューティングの一形態である。IaaSクラウドコンピューティング環境202はまた、サービスとしてのソフトウェア（Software as a Service）（SaaS）および／またはサービスとしてのプラットフォーム（Platform as a Service）（PaaS）アーキテクチャとして構成されたその他のクラウドコンピューティング環境を含み得る、または当該環境に結合し得る。この環境において、クラウドプロバイダは、従来オンプレミスデータセンターに存在していたハードウェアおよび／またはソフトウェアコンポーネントのインフラストラクチャをホストすることができる。このハードウェアは、サーバ、ストレージ、ネットワーキングハードウェア、ディスクアレイ、ソフトウェアライブラリ、および、ハイパーバイザレイヤのような仮想化ユーティリティを含み得る。IaaS環境202は、Oracle（登録商標）またはその他一般に利用できるクラウドプラットフォームのような商用ソースによって提供されることができる。IaaS環境202はまた、ハードウェアおよびソフトウェアのプライベートインフラストラクチャを用いてプライベートクラウドとしてデプロイすることもできる。

【0016】

クラウド環境のタイプとは関係なく、サービス114は、複数種類のハードウェア／ソフトウェアシステムにデプロイすることができる。たとえば、サービス114は、専用ハードウェア206にデプロイすることができる。専用ハードウェア206は、サービス114に特別に割り当てられたサーバ、ディスク、オペレーティングシステム、ソフトウェアパッケージなどのようなハードウェアリソースを含み得る。たとえば、特定のサーバが、サーバ114との間で流れるトラフィックを処理するように割り当てられていてもよい。

【0017】

別の例において、サービス114は、1つ以上の仮想マシン208として動作させるハードウェア／ソフトウェアにデプロイすることができる。仮想マシンは、専用コンピュータハードウェア206の機能を提供するコンピュータシステムをエミュレートしたもので

10

20

30

40

50

ある。しかしながら、特定の関数専用にする代わりに、物理ハードウェアを複数の異なる仮想マシンが共有してもよい。各仮想マシンは、完全なオペレーティングシステムを含む、実行する必要があるすべての機能を提供することができる。これにより、異なるオペレーティングシステムを有する複数の仮想マシンが、同一の物理ハードウェア上で実行することができ、複数のサービスが1つのハードウェアを共有することができる。

【0018】

別の例において、サービス114はコンテナプラットフォーム210にデプロイすることができる。コンテナプラットフォームは、仮想マシン208と、複数の重要な点において異なっている。第1に、以下図3で詳述するように、コンテナプラットフォーム210は、個々のサービスをコンテナにパッケージングする。各コンテナは、ホストオペレーティングシステムカーネルを共有するとともに、バイナリ、ライブラリ、およびその他の読取専用コンポーネントを共有する。これにより、コンテナを極めて軽くすることができ、わずか数メガバイトのサイズであることも多い。加えて、軽量コンテナは、仮想マシンのブートアップに数分を要するのに対して起動にわずか数秒しかかからず、非常に効率的である。また、コンテナは、オペレーティングシステム、および、コンテナプラットフォーム210内の一組のコンテナセットに対してともに管理できるその他のライブラリを共有することにより、管理オーバーヘッドを減じる。コンテナは同じオペレーティングシステムを共有するものの、オペレーティングシステムは分離のために仮想メモリサポートを提供するので、分離されたプラットフォームを提供する。コンテナ技術は、Docker（登録商標）コンテナ、Linux（登録商標）Libcontainer（登録商標）、オープンコンテナイニシアティブ（Open Container Initiative）（OCI）、Kubernetes（登録商標）、C o e

OS、Apache（登録商標）Mesosを、それ以外のものとともに含み得る。これらのコンテナは、本明細書では簡単に「コンテナプラットフォーム210」と呼ぶ場合があるコンテナオーケストレーションプラットフォームにデプロイすることができる。コンテナプラットフォームは、デプロイされたソフトウェアコンテナの自動化された構成、調整、および管理を維持する。コンテナプラットフォーム210は、サービスディスカバリ、ロードバランシング、ヘルスチェック、マルチデプロイメントなどを提供することができる。コンテナプラットフォーム210は、ノードおよびポッドで構成されたコンテナを実行する、Kubernetesのような一般に利用できるコンテナプラットフォームによって実現し得る。

【0019】

サービス114をデプロイするプラットフォーム206、208、210とは関係なく、プラットフォーム206、208、210は各々、サービス114をコールするためのパブリックアクセスを提供するサービスエンドポイント212、214、216を提供できる。一般的に、これらのエンドポイントには、HTTPコールを通してアクセスでき、これらのエンドポイントは、IPアドレスおよびポート番号に対応付けられる。正しいIPアドレスおよびポート番号に接続することにより、その他のサービスは、公的に利用できるようにされたときにプラットフォーム206、208、210のうちのいずれかにデプロイされるサービスをコールすることができる。サービス114等の各サービスは、サービスをコールするための自身のプロプライエタリフォーマットとデータ要件とを含み得る。同様に、各サービスは、フォーマットおよびデータタイプがそのサービス114に固有である結果を返すことができる。サービス固有の要件に加えて、特定のデプロイメントプラットフォーム206、208、210はまた、サービスと適切にやり取りするために準拠する必要がある、プログラミング言語、パッケージフォーマット（JSON、XMLなど）その他のような、サービス114とやり取りするための追加要件を含み得る。

【0020】

上述の例は、サービス114を上記プラットフォーム206、208、210のうちのいずれかにデプロイすることを可能にするが、本明細書に記載の実施形態は、上記コンテナプラットフォーム210のために特別に設計される。よって、「コンテナプラットフォーム

ーム」にデプロイされると具体的に記載された実施形態は、仮想マシンプラットフォームに、サーバもしくは専用ハードウェアプラットフォーム上に、または一般的にIaaS環境にデプロイされると具体的に記載された実施形態と区別することができる。

【0021】

図3は、本明細書に記載の実施形態のうちのいくつかが使用するコンテナプラットフォーム210に固有であってもよいデータ組織を示す。一般的に、コンテナプラットフォームへのサービスのいかなるデプロイメントもポッド304、306にデプロイされる。ポッドは、1つ以上のアプリケーションコンテナ（たとえばDockerまたはrkt）からなるグループを表す抽象概念である。ポッドはまた、当該ポッド内のすべてのコンテナが共通して利用できるいくつかの共有リソースを含み得る。たとえば、ポッド304はコンテナ310とコンテナ312とを含む。ポッド304はまた、共有リソース308を含む。このリソースは、ストレージボリューム、またはコンテナが如何にしてポッド304内で実行されるかまたは接続されるかに関するその他の情報を含み得る。ポッド304は、比較的密接に結合された異なるサービスコンテナ310、312を含むアプリケーション専用論理ホストをモデル化することができる。たとえば、コンテナ310内のサービス326は、リソース308を利用することができ、コンテナ312内のサービス320をコールすることができる。サービス320もサービス322をコールすることができ、サービス322もサービス324をコールすることができ、これらのサービスは各々コンテナ312にデプロイされている。サービス324の出力はネットワークIPアドレスおよびポート318に与えることができ、これはポッド304が共有する別の共通リソースである。このように、サービス320、322、324、326すべてが共有リソース308と協働することにより、他のコンテナで実行されるサービスがIPアドレスおよびポート番号318によってアクセスすることができる、1つのサービスを提供する。このサービスは、コンテナプラットフォームまたはIaaS環境の一部ではない、ワークステーション、ラップトップコンピュータ、スマートフォンまたはその他のコンピューティングデバイスのような、コンテナプラットフォームの外部のコンピュータシステムが、IPアドレスポート318を介してアクセスすることもできる。

【0022】

最も単純なデプロイメントの場合、各コンテナは1つのサービスを含むことができ、各ポッドはサービスをカプセル化する1つのコンテナを含むことができる。たとえば、ポッド306は1つのサービス328のみを有する1つのコンテナ314のみを含む。この1つのサービスは、ポッド306のIPアドレスおよびポート番号316を通してアクセスできる。典型的に、サービスがコンテナプラットフォームにデプロイされるとき、コンテナおよびポッドがインスタンス化されてこのサービスを保持する。複数の異なるポッドをコンテナノード302にデプロイすることができる。一般的に、ポッドはノード内で実行される。ノードはコンテナプラットフォーム内のワーカーマシン（仮想または物理いずれか）を表す。各ノードは、各ノード内のスケジューリングポッドを自動的に扱う「マスタ」によって管理される。各ノードは、マスタとノードとの間の通信を担い、かつ当該ノードによって表されるマシン上のコンテナ内でポッドを管理するためのプロセスを実行することができる。各ノードはまた、レジストリからコンテナ画像を引き出すこと、コンテナを解凍すること、およびサービスを実行することを担うコンテナランタイムを含み得る。

【0023】

図4は、いくつかの実施形態に係る、IDE102およびプロダクション／デプロイメント環境104にデプロイすることができるAPIレジストリ404を示す。上述のように、サービス114がIDE102からプロダクション／デプロイメント環境104にデプロイされるとき、IDE102内で独占的に利用できる情報へのランタイムアクセスをサービス114が失うという、技術的課題がある。APIレジストリ404に対し、サービス114は、プロダクション／デプロイメント環境104にデプロイされこの環境でランタイム中に動作している間、アクセスすることができる。開発関数がランタイム関数から分離されるという過去の技術的課題は、開発中にサービスをAPIレジストリ404に

登録しAPI定義および／またはAPIプロパティをAPIレジストリ404に与えることにより、APIレジストリ404によって克服される。APIを定義する情報は、IDE102内の開発中の新たなサービス、および、プロダクション／デプロイメント環境104に既にデプロイされているサービスが、使用できる。この登録プロセスの完了後、サービス114はクライアントライブラリを用いて動作することができ、クライアントライブラリは、ランタイム中にAPIレジストリ404にアクセスすることにより、API関数が、対応するサービスの現在のIPアドレスおよびポート番号に正しくバインドされていることを保証する。APIレジストリ404は、これらの技術的課題を解決するために特別に設計された新たなデータ構造および処理ユニットを表す。

【0024】

当該技術に存在していたもう1つの技術的課題は、サービスプロパティがプロダクション／デプロイメント環境104にデプロイされるとき、サービスプロパティの実現であった。たとえば、サービスを高可用性を伴ってデプロイしようとする場合、開発者は、コンテナプラットフォーム内でサービスの複数のインスタンスを特別にインスタンス化したコンテナデプロイメントファイルと、バランスが取れたトラフィックとを、サービスが常に利用できるように構築する必要がある。サービス開発者は必ずしもこの専門知識を有していた訳ではなく、それらのサービスのデプロイメントを大抵は管理できた訳でもない。上述のように、APIレジストリ404により、サービスは、APIレジストリ404が自動的に実現できる高可用性のようなプロパティを単純に選択することができる。この技術的解決策が可能な理由は、APIレジストリ404がIDE102とプロダクション／デプロイメント環境104との間の隙間の架け橋となるからである。

【0025】

図5は、いくつかの実施形態に係る、実行時にコンテナプラットフォーム210とともに使用されるAPIレジストリ404のデプロイメントを示す。APIレジストリ404が提供する、既存の技術に対する技術的解決策および改善のうちの1つは、サービスコールのための安定したエンドポイントの維持、ならびにサービスコールへのアクセスの簡略化およびサービスコールへのアクセスのための自動コード生成である。本開示よりも前に、サービス間のコールは、たとえばIPアドレスおよびポート番号に対するHTTPコールを用いるポイントツーポイント接続であった。サービスがアップデート、リプレイス、リロケート、およびコンテナプラットフォーム210に再デプロイされるとき、IPアドレスおよびポート番号が頻繁に変わる可能性がある。この場合、アップデートされたサービスをコールしたすべてのサービスが、そのサービスをコールした実際のコードのIPアドレスおよびポート番号をアップデートする必要がある。APIレジストリ404は、この技術的課題を、サービスのIPアドレスおよびポート番号と、APIレジストリを介して利用できるようにされるAPI関数との間の動的バインディングを提供することによって解決する。APIレジストリ404が自動的に生成するクライアントライブラリは、APIレジストリ404にアクセスすることにより特定のサービスの現在のIPアドレスおよびポート番号を取り出すおよび／または検証する関数を含み得る。したがって、第2のサービスに接続する第1のサービスは、クライアントライブラリを一度生成するだけで、第2のサービスへの、存続期間にわたって安定した接続を提供する。

【0026】

APIレジストリ404が解決する別の技術的課題は、クライアントライブラリの自動生成である。本開示よりも前に、第2のサービスにアクセスする第1のサービスは、第2のサービスにアクセスするためのカスタムコードを開発者が記述することを必要としていた。このコードは時間の経過に伴って変化するので、どちらもアップデートを必要とする第1のサービスと第2のサービスとの間に非互換性が生じることになる。APIレジストリ404は、この技術的課題を、サービスをコールするためにクライアントライブラリを自動的に生成するのに使用されるAPI定義ファイルをアップロードすることによって解決する。したがって、サービスは、その他いずれかのサービスにおけるコーリングコードが如何にして動作すべきかを具体的に指定することができ、これが互

10

20

30

40

50

換性を保証する。これらのクライアントライブラリはまた、サービスをコールするためのコードを大幅に簡略化しカプセル化する。以下で述べるように、IPアドレスおよびポート番号を使用する複雑なHTTPコールは、呼び出し側サービスに固有の言語（たとえばJava（登録商標）、C#など）の単純なメンバ関数に置き換えることができる。これにより、呼び出し側サービスは、APIレジストリ404からAPI関数を選択することができ、関数で実現するコードは、クライアントライブラリとして呼び出し側サービスダウンロードすることができる。

【0027】

図6Aは、いくつかの実施形態に係る、APIレジストリ404をデプロイする方法のフローチャートを示す。この方法は、APIレジストリサービスをコンテナ環境にデプロイすることを含み得る（601）。APIレジストリは、コンテナ環境においてコンテナ内で動作するサービスとして実現することができる。よって、APIレジストリは、実行時にアクセスされることができるよう、コンテナ環境内にサービスがデプロイされた後で能動的に実行することができる。APIレジストリは、上記既存のIDEにリンクさせることもできる。この方法はさらに、コンテナプラットフォーム内の利用できるサービスのためのポートを発見することを含み得る（603）。サービスがコンテナプラットフォームにデプロイされると、APIレジストリは、コンテナプラットフォームにデプロイされたサービス各々を逐次的にトラバースするディスカバリプロセスを開始することができる。サービスごとに、APIレジストリはIPアドレスおよびポート番号を検出して記録することができる。このプロセスによって発見されたIPアドレスおよびポート番号の一覧表は、APIレジストリに対応付けられたテーブル等のデータ構造に格納することができる。また、各IPアドレスおよびポート番号は、そのサービスの名称とともに、または、コンテナプラットフォーム上のサービスを一意に識別するその他の識別子とともに、格納することができる。図6Aのフローチャートに示されるこれらの初期ステップは、APIレジストリがコンテナプラットフォームのランタイム環境内で動作を開始するため、かつ、IDE内で開発中のサービスがAPIレジストリを利用できるようにするための、出発点を提供する。

【0028】

図6Bは、いくつかの実施形態に係る、図6Aのフローチャートを用いてAPIレジストリをデプロイするときのコンテナプラットフォーム210のソフトウェア構造を示す。先に述べたように、APIレジストリ404はコンテナプラットフォーム210内のコンテナ620にデプロイすることができる。先に図3で説明したように、コンテナ620は、あるノードにおいて1つ以上のポッド内で動作することができる。APIレジストリ404を、コンテナプラットフォーム210内の他のコンテナのうちのいずれかが非公式に利用できるようにすることができる。いくつかの実施形態において、APIレジストリ404を、コンテナプラットフォーム210の一部ではない他のデバイスが公的に利用できるようにすることもできる。コンテナ化されたサービスとして、APIレジストリ404は他のサービスが利用できるIPアドレスおよびポート番号を有することができる。しかしながら、APIレジストリ404のIPアドレスおよびポート番号は、クライアントライブラリで自動的に生成されたコードのみによって使用されるので、いくつかの実施形態はAPIレジストリ404のIPアドレスおよびポート番号を公開する必要はない。その代わりに、IDE自身のクライアントライブラリが、他のサービスの開発、デプロイ、および実行中にコンタクトできるよう、APIレジストリ404のIPアドレスおよびポート番号の最新一覧表を管理することができる。

【0029】

APIレジストリ404は、コンテナ620にデプロイされた後に、ディスカバリプロセスを実行することができる。ディスカバリプロセスは、コンテナプラットフォームにおけるノードのディレクトリ一覧表を用いることにより、IPアドレスおよびポート番号で、サービスを実現するポッドを特定することができる。そうすると、APIレジストリ404は、利用できる各サービスの番号または名称等の固有識別子にアクセスし、識別子を

10

20

30

40

50

各IPアドレスおよびポート番号とともにコンテナプラットフォーム210に格納することができる。このディスカバリプロセスを周期的に実行することにより、コンテナプラットフォーム210に追加された新たなサービスを検出することができるとともに、コンテナプラットフォーム210から削除された既存のサービスを特定することができる。以下で述べるように、このディスカバリプロセスを用いることにより、既存のサービスについてIPアドレスおよびポート番号がいつ変化したかを検出することもできる。たとえば、APIレジストリ404は、エンドポイント602、604、606、608を有するサービスを発見することができる。下記プロセスにおいて、APIレジストリ404は、これらのエンドポイント602、604、606、608各々を、APIレジストリ404に登録されたAPI関数にバインドすることができる。この最初の発見後のある時点で、エンドポイント602のIPアドレスおよび／またはポート番号が、エンドポイント602に対応付けられたサービスがリプレイス、アップデート、またはリバイズされたときに、変更される場合がある。APIレジストリ404は、このエンドポイント602に対する変更を検出し、APIレジストリ404が提供する既存のAPI関数へのバインディングをアップデートすることができる。

10

【0030】

同様に、APIレジストリ404は、ディスカバリプロセスを用いることにより、いつエンドポイントが利用できなくなったかを検出し、その後、サービスに対応付けられたAPI関数を削除することができる。いくつかの実施形態において、サービスはAPIレジストリ404に登録されているものの、対応するAPI関数が現在有効なエンドポイントにバインドされていないとき、APIレジストリ404は、対応するAPI関数をコールしているいずれかのサービスにモックレスポンス(mock response)を与えることができる。

20

たとえば、エンドポイント604に対応するサービスについてAPIが登録されているものの、エンドポイント604は現在利用できない場合、APIレジストリ404は、エンドポイント604に対するコールをインターセプトしそれに応じてデフォルトまたはダミーデータを与えることができる。これにより、エンドポイント604に対応付けられているサービスをコールするサービスは、機能を維持する、および／またはこの特定のサービスに対する接続を「切断する」ことなく設計プロセスを続けることができる。モック／テストデータシナリオについては以下でより詳細に説明する。

30

【0031】

図7Aは、いくつかの実施形態に係る、APIレジストリ404にサービスを登録する方法のフローチャートを示す。この方法は、API定義のアップロードを受けることを含み得る(701)。API定義は、データパケット、ファイル、または情報リポジトリに対するリンクの形態で提供し得る。API定義は、サービスに対応付けられたエンドポイントにバインドすべきAPI関数を特定し定義するために使用できる任意の情報を含み得る。たとえば、API定義のいくつかの実施形態は、以下のデータを、すなわち、サービス名またはその他の固有識別子、サービスエンドポイントおよびコールに対応する関数名、対応する記述およびデータタイプでサービスにコールするのに必要なデータ入力、結果データフォーマットおよびデータタイプ、現在のIPアドレスおよび／またはポート番号、エンドポイントに対応付けられることになるAPI関数の機能を記述する文書、モック／テストシナリオ中に返すべきデフォルトまたはダミーデータ値、ならびに、APIレジストリ404が、エンドポイントが受けたHTTP要求を、クラスデータオブジェクトのAPI関数コールを使用するクライアントライブラリに変換するために使用し得るその他任意の情報を、含み得る。

40

【0032】

この方法はまた、アップロードされたAPI定義に基づいて対応するAPI関数を作成することを含み得る(703)。これらのAPI関数はAPI定義に基づいて自動的に生成することができる。サービスの各エンドポイントは複数の異なるAPI関数に対応付けることができる。たとえば、RESTfulインターフェイスを実現するエンドポイント

50

は、同一のIPアドレスおよびポート番号のPOST、GET、PUT、およびDELETE関数に対するHTTPコールを受けることができる。これは結果としてたとえば異なるAPI関数に対するものになり得る。たとえば、このインターフェイスがユーザのリストを表す場合、これは、GetUser()、AddUser()、RemoveUser()、およびUpdateUser()のような、少なくとも4つの異なるAPI関数に対応し得る。加えて、各API関数は、UpdateUser(id)、UpdateUser(name)、UpdateUser(firstname, lastname)などのような、

複数の異なるパラメータリストを含み得る。これらのAPI関数を生成し、生成されたAPI関数をAPIレジストリを介して他のサービスが利用できるようにすることができる。以下でより詳細に説明するように、これらの関数をAPIレジストリを介してコールするためにサービスは必要ではないことに注意する必要がある。代わりに、これらの関数は、APIレジストリにおいてブラウズするのに利用できるようにされ、選択されると、APIレジストリは、呼び出し側サービスにおいてこれらの関数を実現するクライアントライブラリを生成することができる。

10

【0033】

この方法はさらに、APIレジストリにおいて、API関数と、対応するサービスのエンドポイントとの間のバインディングを作成することを含み得る(705)。上記ディスカバリプロセスおよびステップ701の登録プロセスに基づいて、APIレジストリは、コンテナプラットフォームにおけるサービスのエンドポイントと、APIレジストリが作成したAPI関数との間の動的バインディングを作成することができる。利用できるエンドポイントおよびサービスを発見したときに形成される上記データ構造に、APIレジストリは、各エンドポイントの対応する関数または一組の関数を格納することができる。上述のように、このバインディングは、サービスがいつアップデート、移動、リプレイス、またはコンテナプラットフォームに追加されるかがディスカバリプロセスによって判断されたときに、常にアップデートすることができる。これにより、呼び出し側サービスにおいて作成されたクライアントライブラリは、先ずAPIレジストリを調べることにより、サービスの現在のIPアドレスおよびポート番号を検証するまたは受けることができる。

20

【0034】

図7Bは、いくつかの実施形態に係る、APIレジストリ404にAPIを登録するためのステップのハードウェア／ソフトウェア図を示す。上述のように、APIレジストリ404をインスタンス化しコンテナプラットフォーム210におけるコンテナ620内で実行することができる。コンテナプラットフォーム210はプロダクション／デプロイメント環境を表しているが、APIレジストリ404はそれでもなお、サービスを開発するために使用されるIDE102からアクセスすることができる。よって、IDE102は、API定義ファイル702をAPIレジストリ404にアップロードするメカニズムを提供することができる。具体的には、IDE102のユーザインターフェイスは、開発者がAPI定義ファイル702のフィールドを定義するおよび／またはフィールドにポピュレートすることを可能にするウィンドウまたはインターフェイスを含み得る。上述のこの情報は、関数名、パラメータリスト、データタイプ、フィールド長、オブジェクトクラス定義、IPアドレスおよびポート番号、サービス名またはその他の固有識別子などを、含み得る。この情報は、APIレジストリ404にアップロードすることができ、動的バインディングによって、エンドポイント602の特定のIPアドレスおよびポート番号にリンクさせることができる。最後に、APIレジストリ404は、APIレジストリ404を介して利用できるようにすることができる1つ以上のAPI関数704を生成することができる。

30

40

【0035】

サービスがAPIレジストリ404に登録され1つ以上のAPI関数が生成された後に、APIレジストリは、開発者がサービスを設計する際にこれらの関数を利用できるようにすることができる。図8は、いくつかの実施形態に係る、APIレジストリ404に登録されたAPIをブラウズし選択するためのグラフィカルインターフェイス802および

50

コマンドラインインターフェイス 804 の例を示す。コンテナプラットフォームに対して新たなサービスをプログラミングおよび開発するときに、開発者は、グラフィカルインターフェイス 802 にアクセスすることにより、それらのサービスで利用できる API 関数をブラウズし選択することができる。このグラフィカルインターフェイス 802 は、一例にすぎず、API 関数をブラウズし選択するために使用できるグラフィカルインターフェイスのタイプを限定することを意図している訳ではない。

【0036】

この実施形態において、IDE 102 は、API レジストリに登録されている API のリストを提供するようグラフィカルインターフェイス 802 に命じることができる。この実施形態において、API はエンドポイントに基づいてカテゴライズされる。たとえば、あるサービスに対応する 1 つのエンドポイントは、ユーザ記録を格納するための RESTful インターフェイスを提供し得る（たとえば「UserStorage」）。グラフィカルイン

10

ターフェイス 802 は、選択されたエンドポイントを介して利用できるすべての API 関数（たとえば「CreateUser」、「DeleteUser」、「UpdateUser」など）を表示することができる。その他の実施形態は、サービスが複数のエンドポイントを提供する場合、サービス全体に基づいて関数をグループ分けすることができる。グラフィカルインターフェイス 802 は、呼び出し側サービスで使用される 1 つ以上の API 関数の選択を受けることができる。API レジストリは次に、必要なパラメータおよびリターン値を含む、API 関数の使用方法を示す文書を提供することができる。当業者は、コマンドラインインターフェイス 804 がグラフィカルインターフェイス 802 と同様の情報を提供することができかつ同様の入力を受けることができることを、理解するであろう。

20

【0037】

図 8 に示されるインターフェイス 802、804 は複数の技術的利点を提供する。第 1 に、これらのインターフェイス 802、804 は、API レジストリに登録されているすべての API の最新一覧表を提供する。これは、コンテナプラットフォームで現在利用できるすべてのサービスのリストに相当する。文書を調べることで、サービス開発者にコンタクトすること、および／または利用できるサービスのリストの場所を特定するためのその他の非効率的なタスクを実行することが要求されるのではなく、サービス開発者はこの情報をリアルタイムで取り出して表示することができる。加えて、サービスがアップデートされると、API 定義ファイルに対応するやり方でアップデートすることができる。これにより、次に図 8 に示されるディスプレイがアップデートされて各 API 関数の最新の可用性情報を提供する。

30

【0038】

図 9 は、いくつかの実施形態に係る、API レジストリに登録されたサービスおよびその対応する関数を使用する方法のフローチャートを示す。この方法は、登録された API の一覧表を提供することを含み得る（901）。所望のサービスが既にわかっている場合はこのステップを省略してもよい。しかしながら、一般的に、サービスは、上述の図 8 のインターフェイスを用いたブラウズおよびナビゲーションのために表示することができる。この方法はまた、API 関数の選択を受けることを含み得る（903）。この選択は、API レジストリがサービスの開発者から受けることができるものである。たとえば、開発者は、上記 CreateUser() 関数を用いてユーザ記録のデータベースをアップデートすると決定する場合がある。図 10 は、API レジストリが如何にして CreateUser() 関数の選択 1002 をグラフィカルインターフェイス 802 を介して受けることができるかを示す。他の実施形態は、この選択を、コマンドラインインターフェイスを介してまたは IDE が提供するその他の入力方法を介して受けることができる。

40

【0039】

再び図 9 を参照して、API レジストリは、API 関数の選択を受けると、呼び出し側サービスのための 1 つ以上のクライアントライブラリを生成することができる（905）。クライアントライブラリを生成することにより、呼び出し側サービスに、API 関数に

50

動的にバインドされたサービスエンドポイントを提供することができる。具体的には、IDEは、コンテナプラットフォームにおけるサービスエンドポイントと直接インターフェイスするのに必要な機能をカプセル化する一組のクラスオブジェクトをIDEに生成することができる。いくつかの実施形態において、クライアントライブラリは、サービスとの通信に必要なコードを実施するメンバ関数をコールするためにインスタンス化または使用することができるオブジェクトクラスを含み得る。これらのクライアントライブラリの例については以下でより詳細に説明する。

【0040】

この方法はさらに、テストデータを提供することを含み得る(907)。サービスは、APIレジストリに登録されるとき、完全である必要はない。代わりに、サービスは、呼び出し側サービスに対する機能的レスポンスを提供する準備がまだ整っていないことをAPIレジストリに対して示すことができる。いくつかの実施形態において、APIレジストリにアップロードされたAPI定義ファイルは、サービスが機能的になる前に返す必要がある情報のタイプの明細を含み得る。呼び出し側サービスがAPI関数をコールすると、APIレジストリが生成したクライアントライブラリは、サービスエンドポイントではなくAPIレジストリに要求をルーティングすることができる。そうすると、APIレジストリは、ダミー、ヌル、またはデフォルト値を用いてレスポンスを提供することができる。これに代えて、クライアントライブラリ自身の中にあるコードが、呼び出し側サービスに返されるデフォルトデータを生成してもよい。

【0041】

図9に示される具体的なステップが本発明の各種実施形態に係るAPIレジストリの特定の使用方法を提供することが理解されるはずである。代替実施形態に従いステップの他のシーケンスを実行することもできる。たとえば、本発明の代替実施形態は、概要を述べた上記ステップを異なる順序で実行し得る。加えて、図9に示される個々のステップは、個々のステップに適したさまざまなシーケンスで実行し得る複数のサブステップを含み得る。さらに、特定のアプリケーションに応じてその他のステップを追加または削除する場合もある。当業者は、多数の変形、修正、および代替形を認識するであろう。

【0042】

図11は、いくつかの実施形態に係る、APIレジストリがあるサービスのために自動的に生成したクライアントライブラリの一例を示す。このクライアントライブラリ1102は、ユーザ記録を格納するサービスに対応し得る。このクライアントライブラリ1102ならびに対応するクラスおよびサービスは、例示のために提供されているにすぎず、限定を意図している訳ではない。先に述べたように、各API関数およびサービスは、APIレジストリにアップロードされたAPI定義ファイルによって如何にしてクライアントライブラリが生成されるべきかを指定することができる。したがって、「User(ユーザ)」サービスに関して以下で述べる原理は、その他のサービスにも適用し得る。

【0043】

Userサービスを表すために、APIレジストリはUserのクラスを生成することができる。呼び出し側サービスは、APIレジストリによるクライアントライブラリの生成を要求するとき、呼び出し側サービスが使用しているプログラミング言語を指定することができる。たとえば、呼び出し側サービスがJavaでIDEに記述されている場合、APIレジストリはJavaプログラミング言語でクラスライブラリを生成することができる。これに代えて、呼び出し側サービスがC#で記述されている場合、APIレジストリはC#プログラミング言語でクラスライブラリを生成することができる。Userクラスは、サービスエンドポイントを介して実行することができる異なる動作に対応するメンバ関数を有するように生成することができる。これらのメンバ関数は、スタティックメンバ関数にすることにより、Userクラスのインスタンス化されたインスタンスを必要としないようにすることができる、または、インスタンス化されたUserオブジェクトとともに使用されてもよい。

【0044】

この例において、User サービスは、RESTful インターフェイスを使用することにより、サービスによって格納された個々のユーザ記録を編集することができる。たとえば、API レジストリは、CreateUser() 関数を生成することにより、User サービスに対する POST コールを実現することができる。クラスライブラリが実行できる機能のうちの1つは、サービスに対してデータパケットとして直接送られる API 関数に対してパラメータとして提供されるデータを、パースし、フィルタリングし、フォーマットすることである。この例において、CreateUser() 関数は、呼び出し側サービスの便宜のためにフォーマットされたパラメータを受け入れることができる。たとえば、呼び出し側サービスは、ユーザの名 (first name) およびユーザの姓 (last name) のストリングを別

10

々に格納することができる。しかしながら、POST コマンドは、名と姓との連結ストリングを必要とする場合がある。ユーザーフレンドリーなパラメータセットを受け入れるために、クライアントライブラリ 1102 は、関数に対するパラメータとして受けたデータを、サービスエンドポイントと互換性のあるフォーマットにフォーマットする一組の動作を実行することができる。これは、ヘッダ情報の生成、特定のデータフィールドのフォーマットの変更、データフィールドの連結、その他のソースからの追加データを要求すること、計算またはデータ変換の実行などを含み得る。これはまた、再フォーマットされたパラメータを、JSON、XML などのようなフォーマットにパッケージングすることを含み得る。

20

【0045】

パラメータがサービスエンドポイントのためのパッケージに正しくフォーマットされると、クライアントライブラリ 1102 は、サービスに対する POST コールを扱うこともできる。クライアントライブラリが生成されるときに、サービスの IP アドレスおよびポート番号を、サービスに対する HTTP 要求で使用される CreateUser() 関数に挿入することができる。なお、HTTP 要求の詳細は、CreateUser() 関数にカプセル化される。呼び出し側サービスのための開発者が、サービスによって利用できるようにされた POST 関数の使用を所望する場合、コードをライブラリ 1102 自身に書き込む代わりに、API レジストリから User サービスを選択することができる。そうすると、API レジストリは、User クラスを含むクライアントライブラリ 1102 を自動的に生成する。次に、POST 関数を使用するために、サービス開発者は、User.CreateUser("John", "Smith", 2112) 関数を使用するだけで、ユーザである John Smith をサービスに追加することができる。

30

【0046】

図 12 は、いくつかの実施形態に係る、サービスエンドポイントと API 関数との間の動的バインディングを含むクライアントライブラリ 1202 の実施形態を示す。この例において、API レジストリがクライアントライブラリ 1202 を生成するとき、CreateUser() 関数は、サービスの IP アドレスおよびポート番号を動的に取り出すコード 120

4 を含み得る。呼び出し側サービス 114 は、呼び出し側サービス 114 がコンテナプラットフォームのようなプロダクション／デプロイメント環境 104 で動作しているときに、GetIPPort() 関数を用いることにより実行時に要求を API レジストリ 404 に送ることができる。API レジストリ 404 は、API 関数とサービスエンドポイントとの間の最新バインディングを維持するために常にアップデートされるその内部テーブルにアクセスすることができる。次に、API レジストリ 404 は、現在の IP アドレスおよびポート番号を呼び出し側サービス 114 に返すことができる。次に、クライアントライブラリ 1202 は、IP アドレスおよびポート番号を、サービスに接続する HTTP POST コードに挿入することができる。API レジストリ 404 に対しては、コンテナプラットフォームにおける任意の呼び出し側サービスが実行時にアクセスできるので、コールされているサービスのポート番号についての IP アドレスが変化したとき、これらのサービスのうちのいずれのサービスも、アップデートまたはパッチは不要である。代わりに、A

40

50

P I レジストリ 4 0 4 は、サービスがコールされるたびに最新情報を提供することができる。いくつかの実施形態において、GetIPPort()関数は、1 時間に 1 度、1 日に 1 度、1 週間に 1 度など、A P I レジストリ 4 0 4 をコールするだけで、サービスエンドポイントはプロダクション環境において頻繁に変わるものではないという仮定のもとでサービス 1 1 4 についてコンテナの外部で行われる関数コールの数を最小にすることができる。

【0 0 4 7】

図 1 3 は、いくつかの実施形態に係る、サービスコールのための入力データセットを完成させる追加データを配置できるクライアントライブラリ 1 3 0 2 の実施形態を示す。クライアントライブラリ 1 3 0 2 を用いた簡略化のために、クライアントライブラリ 1 3 0 2 はサービス開発者に要求されるパラメータの数を最小にすることができる。サービスコールを行うために必要であろう追加データは、他のソースから取り出すことができ、したがって、パラメタリストから省略してもよい。代わりに、これらの追加パラメータは、クライアントライブラリ 1 3 0 2 がこれらの他のソースから直接取り出してもよい。たとえば、新たなユーザの作成は、このユーザのユーザロールを指定することを含み得る。パラメータのうちの 1 つとしてユーザロールを提供することをサービス開発者に要求する代わりに、クライアントライブラリ 1 3 0 2 は、他のいずれかのソースからユーザのロールを自動的に取り出すコード 1 3 0 4 を含み得る。この例において、ユーザロールは、データベースから、コンテナプラットフォーム内の別のサービスから、または呼び出し側サービス内のユーザロールを格納する別のクラスから、取り出すことができる。これらの場合のうちのいずれの場合でも、コード 1 3 0 4 は、自動的にユーザロールを取り出し、これを、サービスに送られる H T T P P O S T コマンドのための入力データの一部としてパッケージングすることができる。

【0 0 4 8】

サービスに対する入力のためにデータを配置しフォーマットすることに加えて、クライアントライブラリ 1 3 0 2 は、サービスから受けたデータをパースして返し、エラー条件を処理することができる。この例において、P O S T コマンドはデータパケットを R e s u l t (結果) 変数に返すことができる。しばしば、サービスは、呼び出し側サービスが必要とする情報よりも多くの情報を含むデータパケットを返すことができる。したがって、クライアントライブラリ 1 3 0 2 は、R e s u l t 変数におけるデータフィールドをパースし、データを R e s u l t 変数から抽出し、ユーザクラスによって予想されるより使いやすいフォーマットにフォーマットしパッケージングすることができる。この例において、コード 1 3 0 6 は、R e s u l t 変数からフィールドを抽出して使用することにより、A P I 関数から返される新たな U s e r オブジェクトを作成することができる。G E T コマンドを使用する別の例において、個々の A P I 関数は、G E T コマンドからの R e s u l t 変数から異なるフィールドを抽出する U s e r クラスで生成することができる。たとえば、U s e r クラスは、GetFirstName(id)関数、GetLastName(id)関数、GetRole(id)関数などを提供することができる。これらの関数は各々、R e s u l t 変数から異なるフィールドを返す間、非常によく似たコードを含み得る。

【0 0 4 9】

結果をパースすることに加えて、クライアントライブラリ 1 3 0 2 はまた、サービスの使用に対応付けられたエラー条件を処理するコード 1 3 0 8 を生成することができる。この例において、コード 1 3 0 8 は、R e s u l t 変数におけるステータス (status) フィールドをテストすることにより、P O S T コマンドが成功したか否かを判断することができる。コマンドが成功していた場合、CreateUser()関数は新たな U s e r オブジェクト

を返すことができる。P o s t コマンドが失敗した場合、関数は、代わりにヌルオブジェクトを返すことおよび／またはサービス対するコールをリトライすることが可能である。

【0 0 5 0】

図 1 4 は、いくつかの実施形態に係る、サービスをコールするときにリトライを処理することができるクライアントライブラリ 1 4 0 2 を示す。図 1 3 の例と同様に、クライア

ントライブラリ1402は、POST HTTPコールによってポピュレートされたResult変数におけるステータスを使用することにより、このコールが成功したか否かを判断する。その結果が不成功である間、クライアントライブラリ1402は、コールが成功するまでリトライを続けることができる。いくつかの実施形態は、カウンターまたはその他のメカニズムを使用することにより、リトライの数を制限するまたはリトライとリトライとの間に待ち時間を追加することができる。

【0051】

本明細書に記載の各方法は、専用コンピュータシステムによって実現することができる。これらの方法の各ステップは、当該コンピュータシステムが自動的に実行してもよく、および／またはユーザを必要とする入力／出力が与えられてもよい。たとえば、ユーザが方法の各ステップに対して入力を与えてもよく、これらの入力はいずれも、このような入力を要求する、コンピュータシステムが生成した特定の出力に応じて与えられてもよい。各入力は、対応する要求出力に応じて受けられてもよい。さらに、入力は、ユーザから受けられてもよく、別のデータシステムからデータストリームとして受けられてもよく、メモリロケーションから取り出されてもよく、ネットワークを介して取り出されてもよく、ウェブサービスから要求されてもよく、および／またはその他であってもよい。同様に、出力は、ユーザに与えられてもよく、データストリームとして別のコンピュータシステムに与えられてもよく、メモリロケーションに保存されてもよく、ネットワークを介して送られてもよく、ウェブサービスに与えられてもよく、および／またはその他であってもよい。すなわち、本明細書に記載の方法の各ステップは、コンピュータシステムが実行し得るものであり、かつ、ユーザを必要とするもしくは必要としないかもしれないコンピュータシステムに対する、任意の数の入力、当該コンピュータシステムからの出力、および／または当該コンピュータシステムへの／からの要求を必要とし得るものである。ユーザを必要としないステップは、人の介入なしでコンピュータシステムが自動的に実行し得ると言える。したがって、本開示に照らして、本明細書に記載の各方法の各ステップは、ユーザへのおよびユーザからの入力および出力を含むように変更されてもよく、または、プロセッサが何らかの判断を行う場合は人の介入なしでコンピュータシステムが自動的に実行してもよい。さらに、本明細書に記載の各方法のいくつかの実施形態は、有形の非一時的な記憶媒体に格納されて有形のソフトウェアプロダクトを形成する一組の命令として実現されてもよい。

【0052】

図15は、上記実施形態のうちのいずれかとのやり取りが可能な分散型システム1500の簡略図を示す。示されている実施形態において、分散型システム1500は1つ以上のクライアントコンピューティングデバイス1502、1504、1506、および1508を含み、これらのクライアントコンピューティングデバイスは、1つ以上のネットワーク1510を介してウェブブラウザ、専用クライアント（たとえばOracle Forms）などのようなクライアントアプリケーションを実行し操作するように構成されている。サーバ1512が、ネットワーク1510を介してリモートクライアントコンピューティングデバイス1502、1504、1506、および1508に対して通信可能に結合されている。

【0053】

各種実施形態において、サーバ1512は、このシステムのコンポーネントのうちの1つ以上が提供する1つ以上のサービスまたはソフトウェアアプリケーションを実行するようにされてもよい。いくつかの実施形態において、これらのサービスは、クライアントコンピューティングデバイス1502、1504、1506、および／または1508のユーザに対し、ウェブベースのもしくはクラウドサービスとして、またはサービスとしてのソフトウェア（SaaS）モデルのもとで、提供されてもよい。そうすると、クライアントコンピューティングデバイス1502、1504、1506、および／または1508を操作しているユーザは、1つ以上のクライアントアプリケーションを利用してサーバ1512とやり取りすることにより、これらのコンポーネントが提供するサービスを利用す

ることができる。

【0054】

図面に示される構成において、システム1500のソフトウェアコンポーネント1518、1520および1522は、サーバ1512上に実装されたものとして示されている。他の実施形態において、システム1500のコンポーネントおよび／またはこれらのコンポーネントが提供するサービスのうちの1つ以上が、クライアントコンピューティングデバイス1502、1504、1506、および／または1508のうちの1つ以上によって実装されてもよい。そうすると、クライアントコンピューティングデバイス进行操作しているユーザは、1つ以上のクライアントアプリケーションを利用することにより、これらのコンポーネントが提供するサービスを使用することができる。これらのコンポーネントは、ハードウェア、ファームウェア、ソフトウェア、またはその組み合わせで実装されてもよい。分散型システム1500とは異なり得るさまざまな異なるシステム構成が可能であることが理解されるはずである。図面に示される実施形態はしたがって、実施形態のシステムを実装するための分散型システムの一例であって、限定を意図したものではない。

10

【0055】

クライアントコンピューティングデバイス1502、1504、1506、および／または1508は、Microsoft Windows Mobile（登録商標）等のソフトウェア、および／またはiOS、Windows Phone、Android、BlackBerry 10、Palm OSその他等のさまざまなモバイルオペレーティングシステムを実行し、インターネット、電子メール、ショートメッセージサービス（SMS）、Blackberry（登録商標）、またはその他の通信プロトコルに接続可能な、ポータブルハンドヘルドデバイス（たとえばiPhone（登録商標）、携帯電話、iPad（登録商標）、コンピューティングタブレット、携帯情報端末（PDA））またはウェアラブルデバイス（たとえばGoogle Glass（登録商標）ヘッドマウントディスプレイ）であってもよい。クライアントコンピューティングデバイスは、例として、さまざまなバージョンのMicrosoft Windows（登録商標）、Apple Macintosh（登録商標）、および／またはLinux（登録商標）オペレーティングシステムを実行するパーソナルコンピュータおよび／またはラップトップコンピュータを含む汎用パーソナルコンピュータであってもよい。

20

クライアントコンピューティングデバイスは、限定されないがたとえばGoogle Chrome OS等のさまざまなGNU/Linux（登録商標）オペレーティングシステムを含む市場で入手可能な多様なUNIX（登録商標）またはUNIX（登録商標）系オペレーティングシステムのうちのいずれかを実行するワークステーションコンピュータであってもよい。これに代えてまたはこれに加えて、クライアントコンピューティングデバイス1502、1504、1506、および1508は、ネットワーク1510を介して通信可能な、シンクライアントコンピュータ、インターネット接続可能なゲームシステム（たとえばKinect（登録商標）ジェスチャー入力デバイスを有するまたは有しないMicrosoft Xboxゲームコンソール）、および／またはパーソナルメッセージングデバイス等の、電子デバイスであってもよい。

30

クライアントコンピューティングデバイスは、限定されないがたとえばGoogle Chrome OS等のさまざまなGNU/Linux（登録商標）オペレーティングシステムを含む市場で入手可能な多様なUNIX（登録商標）またはUNIX（登録商標）系オペレーティングシステムのうちのいずれかを実行するワークステーションコンピュータであってもよい。これに代えてまたはこれに加えて、クライアントコンピューティングデバイス1502、1504、1506、および1508は、ネットワーク1510を介して通信可能な、シンクライアントコンピュータ、インターネット接続可能なゲームシステム（たとえばKinect（登録商標）ジェスチャー入力デバイスを有するまたは有しないMicrosoft Xboxゲームコンソール）、および／またはパーソナルメッセージングデバイス等の、電子デバイスであってもよい。

40

【0056】

具体例としての分散型システム1500が4つのクライアントコンピューティングデバイスとともに示されているが、任意の数のクライアントコンピューティングデバイスをサポートすることができる。センサなどを有するデバイスのようなその他のデバイスがサーバ1512とやり取りしてもよい。

【0057】

分散型システム1500のネットワーク1510は、限定されないがTCP/IP（transmission control protocol/Internet protocol）、SNA（systems network architecture）、IPX（Internet packet exchange）、AppleTalk（登録商

50

標) などを含み、市場で入手可能なさまざまなプロトコルのうちのいずれかを用いてデータ通信をサポートできる、当業者によく知られた任意のタイプのネットワークであってもよい。一例にすぎないが、ネットワーク 1510 は、たとえばイーサネット (登録商標)、トークンリングおよび／または同様のものに基づく、ローカルエリアネットワーク (LAN) であってもよい。ネットワーク 1510 は、広域ネットワークおよびインターネットであってもよい。これは、限定されないが仮想プライベートネットワーク (VPN)、イントラネット、エクストラネット、公衆交換電話網 (PSTN)、赤外線ネットワーク、無線ネットワーク (たとえば Institute of Electrical and Electronics (IEEE) 802.11 プロト

コルスイート、Bluetooth (登録商標)、および／または任意の他の無線プロトコルのうちのいずれかの下で動作するネットワーク)、および／または上記および／またはその他のネットワークの任意の組み合わせを含む、仮想ネットワークを含み得る。

10

【0058】

サーバ 1512 は、1つ以上の汎用コンピュータ、専用サーバコンピュータ (一例として PC (パーソナルコンピュータ) サーバ、UNIX (登録商標) サーバ、ミッドレンジサーバ、メインフレームコンピュータ、ラックマウントサーバなどを含む)、サーバファーム、サーバクラスタ、または、その他任意の適切な構成および／または組み合わせからなるものであってもよい。各種実施形態において、サーバ 1512 は、上の開示で説明した1つ以上のサービスまたはソフトウェアアプリケーションを実行するようにされていてもよい。たとえば、サーバ 1512 は、本開示の実施形態に係る上記処理を実行するためのサーバに対応していてもよい。

20

【0059】

サーバ 1512 は、上記オペレーティングシステムのうちのいずれかおよび市場で入手可能なサーバオペレーティングシステムを含むオペレーティングシステムを実行し得る。また、サーバ 1512 は、HTTP (hypertext transport protocol) サーバ、FTP (file transfer protocol) サーバ、CGI (common gateway interface) サーバ、JAVA (登録商標) サーバ、データベースサーバなどを含む、さまざまな付加的なサーバアプリケーションおよび／またはミッドティアアプリケーションのうちのいずれかを実行し得る。具体例としてのデータベースサーバは、Oracle、Microsoft、Sybase、IBM (International Business Machines) などから市販されているものを含むが、これらに限定されない。

30

【0060】

いくつかの実装例において、サーバ 1512 は、クライアントコンピューティングデバイス 1502、1504、1506、および 1508 のユーザから受信したデータフィードおよび／またはイベントアップデートを分析し統合するための1つ以上のアプリケーションを含み得る。一例として、データフィードおよび／またはイベントアップデートは、限定されないが、1つ以上の第三者情報源および連続データストリームから受信したTwitter (登録商標) フィード、Facebook (登録商標) アップデートまたはリアルタイム

アップデートを含み得る。これらはセンサデータアプリケーション、金融ティッカー、ネットワーク性能測定ツール (たとえばネットワークモニタリングおよびトラフィック管理アプリケーション)、クリックストリーム分析ツール、自動車交通監視などに関連するリアルタイムイベントを含み得る。また、サーバ 1512 は、クライアントコンピューティングデバイス 1502、1504、1506、および 1508 の1つ以上のディスプレイデバイスを介してデータフィードおよび／またはリアルタイムイベントを表示するための1つ以上のアプリケーションを含み得る。

40

【0061】

分散型システム 1500 は、1つ以上のデータベース 1514 および 1516 も含み得る。データベース 1514 および 1516 はさまざまな場所に存在し得る。一例として、データベース 1514 および 1516 のうちの1つ以上は、サーバ 1512 に対してロー

50

カルな場所にある（および／またはサーバ内にある）非一時的な記憶媒体上にあってもよい。これに代えて、データベース1514および1516は、サーバ1512から遠隔の場所に位置してネットワークベースのまたは専用接続を介してサーバ1512と通信してもよい。一組の実施形態において、データベース1514および1516は、ストレージエリアネットワーク（SAN）内にあってもよい。同様に、サーバ1512に帰する機能を実行するために必要な任意のファイルを、適宜、サーバ1512にローカルにおよび／またはサーバ1512から遠隔の場所に格納してもよい。一組の実施形態において、データベース1514および1516は、SQLフォーマットのコマンドに応答してデータを格納し、アップデートし、取り出すようにされた、Oracleが提供するデータベース等のリレーショナルデータベースを含み得る。

10

【0062】

図16は、本開示の実施形態に係る、実施形態のシステムの1つ以上のコンポーネントがサービスをクラウドサービスとして提供し得るシステム環境1600の1つ以上のコンポーネントの簡略化されたブロック図である。示されている実施形態において、システム環境1600は、クラウドサービスを提供するクラウドインフラストラクチャシステム1602とやり取りするためにユーザが使用し得る1つ以上のクライアントコンピューティングデバイス1604、1606、および1608を含む。クライアントコンピューティングデバイスは、クライアントコンピューティングデバイスのユーザがクラウドインフラストラクチャシステム1602とやり取りすることによってクラウドインフラストラクチャシステム1602が提供するサービスを使用するために用いることができる、ウェブブラウザ、専用クライアントアプリケーション（たとえばOracle Forms）、またはその他何らかのアプリケーション等のクライアントアプリケーションを操作するように構成されてもよい。

20

【0063】

図面に示されているクラウドインフラストラクチャシステム1602は示されているものの以外のコンポーネントを有し得ることが理解されるはずである。さらに、図面に示されている実施形態は、本発明の実施形態を組み込むことができるクラウドインフラストラクチャシステムの一例にすぎない。他のいくつかの実施形態において、クラウドインフラストラクチャシステム1602は、図示されているものよりも多いまたは少ないコンポーネントを有していてもよく、2つ以上のコンポーネントを組み合わせてもよく、またはコンポーネントの異なる構成または配置を有していてもよい。

30

【0064】

クライアントコンピューティングデバイス1604、1606、および1608は、1502、1504、1506、および1508について先に述べたものと同様のデバイスであってもよい。

【0065】

具体例としてのシステム環境1600は3つのクライアントコンピューティングデバイスとともに示されているが、任意の数のクライアントコンピューティングデバイスをサポートすることができる。センサなどを有するデバイスのようなその他のデバイスがクラウドインフラストラクチャシステム1602とやり取りしてもよい。

40

【0066】

ネットワーク1610は、クライアント1604、1606、および1608とクラウドインフラストラクチャシステム1602との間におけるデータの通信および交換を容易にすることができる。各ネットワークは、ネットワーク1510について先に述べたものを含む市場で入手可能なさまざまなプロトコルのうちのいずれかを用いてデータ通信をサポートできる、当業者によく知られた任意のタイプのネットワークであってもよい。

【0067】

クラウドインフラストラクチャシステム1602は、1つ以上のコンピュータ、および／またはサーバ1512について先に述べたものを含み得るサーバを含み得る。

【0068】

50

特定の実施形態において、クラウドインフラストラクチャシステムが提供するサービスは、オンラインデータストレージおよびバックアップソリューション、ウェブベースの電子メールサービス、ホストされているオフィススイートおよびドキュメントコラボレーションサービス、データベース処理、管理された技術サポートサービスなどのような、クラウドインフラストラクチャシステムのユーザがオンデマンドで利用できるようにされる多数のサービスを含み得る。クラウドインフラストラクチャシステムが提供するサービスは、そのユーザのニーズに合わせて動的にスケーリングすることができる。クラウドインフラストラクチャシステムが提供するサービスを具体的にインスタンス化したものを本明細書では「サービスインスタンス」と呼ぶ。一般的に、クラウドサービスプロバイダのシステムからの、インターネットのような通信ネットワークを介してユーザが利用できるようにされる任意のサービスを、「クラウドサービス」と呼ぶ。典型的に、パブリッククラウド環境において、クラウドサービスプロバイダのシステムを構成するサーバおよびシステムは、顧客自身のオンプレミスサーバおよびシステムとは異なる。たとえば、クラウドサービスプロバイダのシステムはアプリケーションをホストすることができ、ユーザは、インターネットのような通信ネットワークを介して、オンデマンドでこのアプリケーションをオーダーし使用することができる。

10

【0069】

いくつかの例において、コンピュータネットワーククラウドインフラストラクチャにおけるサービスは、クラウドベンダーによってまたは当該技術において周知の他のやり方でユーザに提供される、ストレージ、ホストされているデータベース、ホストされているウェブサーバ、ソフトウェアアプリケーション、またはその他のサービスに対する保護されたコンピュータネットワークアクセスを含み得る。たとえば、サービスは、インターネットを介した、クラウド上の遠隔ストレージに対するパスワードで保護されたアクセスを含むことができる。別の例として、サービスは、ネットワーク化された開発者の私的使用のための、ウェブサービスベースのホストされているリレーショナルデータベースおよびスクリプト言語ミドルウェアエンジンを含むことができる。別の例として、サービスは、クラウドベンダーのウェブサイト上でホストされている電子メールソフトウェアアプリケーションに対するアクセスを含むことができる。

20

【0070】

特定の実施形態において、クラウドインフラストラクチャシステム1602は、セルフサービスで、サブスクリプションベースで、弾力的にスケーラブルで、信頼性が高く、可用性が高く、かつ安全なやり方で顧客に与えられる、アプリケーション、ミドルウェア、およびデータベースサービス提供物一式を含み得る。そのようなクラウドインフラストラクチャシステムの一例は、本願の譲受人が提供するOracle Public Cloudである。

30

【0071】

各種実施形態において、クラウドインフラストラクチャシステム1602は、クラウドインフラストラクチャシステム1602が提供するサービスに対する顧客のサブスクリプションを自動的にプロビジョニングし、管理し、追跡するようにされていてもよい。クラウドインフラストラクチャシステム1602は、異なるデプロイメントモデルを介してクラウドサービスを提供することができる。たとえば、（例としてOracle所有の）クラウドサービスを販売する組織がクラウドインフラストラクチャシステム1602を所有しており一般の人々または異なる産業企業がサービスを利用できるようにされるパブリッククラウドモデルの下で、サービスが提供されてもよい。別の例として、クラウドインフラストラクチャシステム1602が1つの組織のためにのみ運営されこの組織内の1つ以上のエンティティのためにサービスを提供し得るプライベートクラウドモデルの下で、サービスが提供されてもよい。また、クラウドインフラストラクチャシステム1602およびクラウドインフラストラクチャシステム1602が提供するサービスを関連するコミュニティ内のいくつかの組織が共有するコミュニティクラウドモデルの下で、クラウドサービスが提供されてもよい。また、2つ以上の異なるモデルの組み合わせであるハイブリッドクラウドモデルの下で、クラウドサービスが提供されてもよい。

40

50

【0072】

いくつかの実施形態において、クラウドインフラストラクチャシステム1602が提供するサービスは、サービスとしてのソフトウェア（SaaS）カテゴリ、サービスとしてのプラットフォーム（PaaS）カテゴリ、サービスとしてのインフラストラクチャ（IaaS）カテゴリ、またはハイブリッドサービスを含むサービスの他のカテゴリの下で提供される、1つ以上のサービスを含み得る。顧客は、クラウドインフラストラクチャシステム1602が提供する1つ以上のサービスを、サブスクリプションオーダーを通じてオーダーすることができる。そうすると、クラウドインフラストラクチャシステム1602は、この顧客のサブスクリプションオーダーのサービスを提供するために処理を実行する。

10

【0073】

いくつかの実施形態において、クラウドインフラストラクチャシステム1602が提供するサービスは、限定されないが、アプリケーションサービス、プラットフォームサービスおよびインフラストラクチャサービスを含み得る。いくつかの例において、アプリケーションサービスは、クラウドインフラストラクチャシステムがSaaSプラットフォームを介して提供することができる。SaaSプラットフォームは、SaaSカテゴリに含まれるクラウドサービスを提供するように構成されてもよい。たとえば、SaaSプラットフォームは、統合開発およびデプロイメントプラットフォーム上でオンデマンドアプリケーション一式を構築し配信する機能を提供することができる。SaaSプラットフォームは、SaaSサービスを提供するための基礎となるソフトウェアおよびインフラストラクチャを管理し制御することができる。SaaSプラットフォームが提供するサービスを利用することにより、顧客は、クラウドインフラストラクチャシステム上で実行されるアプリケーションを利用することが可能である。顧客は、顧客が別々のライセンスおよびサポートを購入しなくても、アプリケーションサービスを得ることが可能である。さまざまな異なるSaaSサービスを提供することができる。例は、限定されないが、大組織向けの販売実績管理、企業統合、およびビジネスフレキシビリティのためのソリューションを提供するサービスを含む。

20

【0074】

いくつかの実施形態において、プラットフォームサービスは、クラウドインフラストラクチャシステムがPaaSプラットフォームを介して提供することができる。PaaSプラットフォームは、PaaSカテゴリに含まれるクラウドサービスを提供するように構成し得る。プラットフォームサービスの例は、限定されないが、共有されている共通アーキテクチャ上の既存のアプリケーションを組織（Oracle等）が統合することを可能にするサービスと、当該プラットフォームが提供する共有サービスを推進する新たなアプリケーションを構築する機能とを含み得る。PaaSプラットフォームは、PaaSサービスを提供するための基礎となるソフトウェアおよびインフラストラクチャを管理し制御することができる。顧客は、顧客が別々のライセンスおよびサポートを購入しなくても、クラウドインフラストラクチャシステムが提供するPaaSサービスを得ることが可能である。プラットフォームサービスの例は、限定されないが、Oracle Java Cloud Service（JCS）、Oracle Database Cloud Service（DBCS）その他を含む。

30

40

【0075】

PaaSプラットフォームが提供するサービスを利用することにより、顧客は、クラウドインフラストラクチャシステムがサポートするプログラミング言語およびツールを採用することができる、デプロイされたサービスを制御することもできる。いくつかの実施形態において、クラウドインフラストラクチャシステムが提供するプラットフォームサービスは、データベースクラウドサービス、ミドルウェアクラウドサービス（たとえばOracle Fusion Middlewareサービス）、およびJavaクラウドサービスを含み得る。一実施形態において、データベースクラウドサービスは、組織がデータベースリソースをプールしデータベースクラウドの形態でサービスとしてのデータベース（Database as a S

50

ervice)

を顧客に提供することを可能にする共有サービスデプロイメントモデルをサポートすることができる。ミドルウェアクラウドサービスは、顧客がさまざまなビジネスアプリケーションを開発しデプロイするためのプラットフォームを提供してもよく、Javaクラウドサービスは、クラウドインフラストラクチャシステムにおいて顧客がJavaアプリケーションをデプロイするためのプラットフォームを提供してもよい。

【0076】

さまざまな異なるインフラストラクチャサービスは、クラウドインフラストラクチャシステムにおいてIaaSプラットフォームが提供してもよい。インフラストラクチャサービスは、SaaSプラットフォームおよびPaaSプラットフォームが提供するサービスを利用する顧客のためのストレージ、ネットワーク、および他の基本的なコンピューティングリソース等の、基礎となるコンピューティングリソースの管理および制御を容易にする。

10

【0077】

特定の実施形態において、クラウドインフラストラクチャシステム1602はまた、クラウドインフラストラクチャシステムの顧客にさまざまなサービスを提供するために用いられるリソースを提供するためのインフラストラクチャリソース1630を含み得る。一実施形態において、インフラストラクチャリソース1630は、PaaSプラットフォームおよびSaaSプラットフォームが提供するサービスを実行するためのサーバ、ストレージ、およびネットワークリソース等のハードウェアの、予め統合し最適化した組み合わせを含み得る。

20

【0078】

いくつかの実施形態において、クラウドインフラストラクチャシステム1602におけるリソースは、複数のユーザによって共有され要求ごとに動的に再割り当てされてもよい。加えて、リソースは、異なるタイムゾーンのユーザに割り当てられてもよい。たとえば、クラウドインフラストラクチャシステム1602は、第1のタイムゾーンの第1組のユーザが指定された時間数だけクラウドインフラストラクチャシステムのリソースを利用することを可能にし、それから、異なるタイムゾーンにいる別の1組のユーザに対して同じリソースを再割り当てすることにより、リソースの利用を最大化することができる。

【0079】

特定の実施形態において、クラウドインフラストラクチャシステム1602の異なるコンポーネントまたはモジュールによって共有され、クラウドインフラストラクチャシステム1602が提供するサービスによって共有される、複数の内部共有サービス1632を提供することができる。これらの内部共有サービスは、限定されないが、セキュリティおよびアイデンティティサービス、統合サービス、企業リポジトリサービス、企業マネージャーサービス、ウィルススキャンおよびホワイトリストサービス、高可用性、バックアップおよびリカバリサービス、クラウドサポートを可能にするためのサービス、電子メールサービス、通知サービス、ファイル転送サービスなどを含み得る。

30

【0080】

特定の実施形態において、クラウドインフラストラクチャシステム1602は、クラウドインフラストラクチャシステムにおけるクラウドサービス（たとえばSaaS、PaaS、およびIaaSサービス）の包括的管理を提供し得る。一実施形態において、クラウド管理機能は、クラウドインフラストラクチャシステム1602が受けた顧客のサブスクリプションをプロビジョニング、管理、および追跡する機能を含み得る。

40

【0081】

一実施形態において、図面に示されるように、クラウド管理機能は、オーダー管理モジュール1620、オーダーオーケストレーションモジュール1622、オーダープロビジョニングモジュール1624、オーダー管理およびモニタリングモジュール1626、ならびにアイデンティティ管理モジュール1628等の、1つ以上のモジュールによって提供されてもよい。これらのモジュールは、汎用コンピュータ、専用サーバコンピュータ、

50

サーバファーム、サーバクラスタ、またはその他任意の適切な構成および／または組み合わせであってもよい、1つ以上のコンピュータおよび／またはサーバを含み得る、または、これらを用いて提供し得る。

【0082】

具体例としての動作1634において、クライアントデバイス1604、1606または1608等のクライアントデバイスを用いる顧客は、クラウドインフラストラクチャシステム1602が提供する1つ以上のサービスを要求すること、および、クラウドインフラストラクチャシステム1602が提示する1つ以上のサービスのサブスクリプションをオーダーすることにより、クラウドインフラストラクチャシステム1602とやり取りしてもよい。特定の実施形態において、顧客は、クラウドユーザインターフェイス（UI）、クラウドUI 1612、クラウドUI 1614および／またはクラウドUI 1616にアクセスしこれらのUIを介してサブスクリプションオーダーを行ってもよい。顧客がオーダーを行ったことに応じてクラウドインフラストラクチャシステム1602が受けるオーダー情報は、顧客を特定する情報と、クラウドインフラストラクチャシステム1602が提供する、顧客がサブスクライブする予定の1つ以上のサービスとを含み得る。

10

【0083】

顧客がオーダーを行った後に、オーダー情報は、クラウドUI 1612、1614および／または1616を介して受信される。

【0084】

動作1636において、オーダーはオーダーデータベース1618に格納される。オーダーデータベース1618は、クラウドインフラストラクチャシステム1602によって運営され他のシステム要素とともに運営されるいくつかのデータベースのうちの1つであってもよい。

20

【0085】

動作1638において、オーダー情報はオーダー管理モジュール1620に転送されてもよい。いくつかの例において、オーダー管理モジュール1620は、オーダーを確認し確認後にオーダーを記入する等の、オーダーに関連する課金および会計機能を行うように構成されてもよい。

【0086】

動作1640において、オーダーに関する情報は、オーダーオーケストレーションモジュール1622に伝えられる。オーダーオーケストレーションモジュール1622は、オーダー情報を利用することにより、顧客によって行われたオーダーのためのサービスおよびリソースのプロビジョニングをオーケストレーションするように構成されてもよい。いくつかの例において、オーダーオーケストレーションモジュール1622は、リソースのプロビジョニングをオーケストレーションすることにより、オーダープロビジョニングモジュール1624のサービスを使用するサブスクライブされたサービスを、サポートしてもよい。

30

【0087】

特定の実施形態において、オーダーオーケストレーションモジュール1622は、各オーダーに関連付けられたビジネスプロセスの管理を可能にし、ビジネスロジックを適用することにより、オーダーがプロビジョニングに進むべきか否かを判断する。動作1642において、新規サブスクリプションのオーダーを受けると、オーダーオーケストレーションモジュール1622は、サブスクリプションオーダーを遂行するのに必要なリソースを割り当てて構成することを求める要求を、オーダープロビジョニングモジュール1624に送る。オーダープロビジョニングモジュール1624は、顧客がオーダーしたサービスのためのリソースの割当を可能にする。オーダープロビジョニングモジュール1624は、クラウドインフラストラクチャシステム1602が提供するクラウドサービスと、要求されたサービスを提供するためのリソースのプロビジョニングに使用される物理実装層との間の抽象化レベルを提供する。このようにして、オーダーオーケストレーションモジュール1622を、サービスおよびリソースが実際にオンザフライでプロビジョニングされ

40

50

るか否か、または、予めプロビジョニングされており要求後に割り当て／アサインされるか否か等の、実装の詳細から切り離すことができる。

【0088】

動作1644において、サービスおよびリソースがプロビジョニングされると、提供されるサービスの通知が、クラウドインフラストラクチャシステム1602のオーダープロビジョニングモジュール1624によってクライアントデバイス1604、1606および／または1608の顧客に送信されてもよい。

【0089】

動作1646において、顧客のサブスクリプションオーダーは、オーダー管理およびモニタリングモジュール1626によって管理および追跡されてもよい。いくつかの例において、オーダー管理およびモニタリングモジュール1626は、ストレージ使用量、転送データ量、ユーザ数、ならびにシステムアップタイムおよびシステムダウンタイムの量等の、サブスクリプションオーダーにおけるサービスの使用統計を収集するように構成されてもよい。

【0090】

特定の実施形態において、クラウドインフラストラクチャシステム1602はアイデンティティ管理モジュール1628を含み得る。アイデンティティ管理モジュール1628は、クラウドインフラストラクチャシステム1602におけるアクセス管理および認可サービス等のアイデンティティサービスを提供するように構成されてもよい。いくつかの実施形態において、アイデンティティ管理モジュール1628は、クラウドインフラストラクチャシステム1602が提供するサービスの利用を所望する顧客に関する情報を管理してもよい。そのような情報は、このような顧客のアイデンティティを認証する情報と、さまざまなシステムリソース（たとえばファイル、ディレクトリ、アプリケーション、通信ポート、メモリセグメントなど）に関してこれらの顧客が実行を認可されるアクションを記述する情報とを含み得る。アイデンティティ管理モジュール1628は、各顧客に関し、かつ、誰が如何にしてこの記述情報にアクセスし修正することができるかに関する記述情報の管理も含み得る。

【0091】

図17は、本発明の各種実施形態を実現し得る具体例としてのコンピュータシステム1700を示す。システム1700を用いることにより、上記コンピュータシステムのうちのいずれかを実現することができる。図面に示されるように、コンピュータシステム1700は、バスサブシステム1702を介して複数の周辺サブシステムと通信する処理ユニット1704を含む。これらの周辺サブシステムは、処理加速ユニット1706と、入出力サブシステム1708と、記憶サブシステム1718と、通信サブシステム1724とを含み得る。記憶サブシステム1718は、有形のコンピュータ読取可能記憶媒体1722とシステムメモリ1710とを含み得る。

【0092】

バスサブシステム1702は、コンピュータシステム1700の各種コンポーネントおよびサブシステムを目的に合わせて互いに通信させるためのメカニズムを提供する。バスサブシステム1702は1つのバスとして概略的に示されているが、バスサブシステムの代替実施形態は複数のバスを利用し得る。バスサブシステム1702は、さまざまなバスアーキテクチャのうちのいずれかを用いる、メモリバスまたはメモリコントローラ、周辺バス、およびローカルバスを含むいくつかのタイプのバス構造のうちのいずれかであってもよい。たとえば、そのようなアーキテクチャは、Industry Standard Architecture (ISA) バス、Micro Channel Architecture (MCA) バス、Enhanced ISA (EISA) バス、Video Electronics Standards Association (VESA) ローカルバス、および、I

EEE P1386. 1規格に準拠して製造されたMezzanineバスとして実装することができるPeripheral Component Interconnect (PCI) バスを含み得る。

【0093】

1つ以上の集積回路（たとえば従来のマイクロプロセッサまたはマイクロコントローラ）として実現することができる処理ユニット1704は、コンピュータシステム1700の動作を制御する。1つ以上のプロセッサが処理ユニット1704に含まれていてもよい。これらのプロセッサはシングルコアまたはマルチコアプロセッサを含み得る。特定の実施形態において、処理ユニット1704は、1つ以上の独立した処理ユニット1732および／または1734として、各処理ユニットに含まれるシングルまたはマルチコアプロセッサとともに実現されてもよい。他の実施形態において、処理ユニット1704はまた、2つのデュアルコアプロセッサを1つのチップに統合することによって形成されるクアドコア処理ユニットとして実現されてもよい。

【0094】

各種実施形態において、処理ユニット1704は、プログラムコードに応じてさまざまなプログラムを実行することができる、かつ、同時に実行している複数のプログラムまたはプロセスを管理することができる。ある所定の時点で、実行すべきプログラムコードのうちの一部またはすべてが、プロセッサ1704および／または記憶サブシステム1718にあってもよい。適切なプログラミングを通して、プロセッサ1704は上記各種機能を提供することができる。コンピュータシステム1700はさらに、デジタル信号プロセッサ（DSP）、専用プロセッサ、および／またはその他同様のものを含むことができる、処理加速ユニット1706を含み得る。

【0095】

入出力サブシステム1708は、ユーザインターフェイス入力デバイスと、ユーザインターフェイス出力デバイスとを含み得る。ユーザインターフェイス入力デバイスは、キーボード、マウスまたはトラックボール等のポインティングデバイス、ディスプレイに組み込まれたタッチパッドまたはタッチスクリーン、スクロールホイール、クリックホイール、ダイヤル、ボタン、スイッチ、キーパッド、音声コマンド認識システム付きの音声入力デバイス、マイク、およびその他のタイプの入力デバイスを含み得る。ユーザインターフェイス入力デバイスは、たとえば、ジェスチャーおよび発話コマンドを使用するナチュラルユーザインターフェイスを通してユーザが入力デバイスを制御し入力デバイスとやり取りすることを可能にする、Microsoft Xbox（登録商標）360ゲームコントローラ等のMicrosoft Kinect（登録商標）モーションセンサのような、モーション検知および／またはジェスチャー認識デバイスを含み得る。ユーザインターフェイス入力デバイスはまた、ユーザの目の活動（たとえば写真撮影中および／またはメニュー選択中の「まばたき」）を検出し目のジェスチャーを入力デバイス（たとえばGoogle Glass（登録商標））への入力

として変換する、Google Glass（登録商標）まばたき検出器等のアイジェスチャー認識デバイスを含み得る。加えて、ユーザインターフェイス入力デバイスは、ユーザが音声コマンドを通して音声認識システム（たとえばSiri（登録商標）ナビゲータ）とやり取りすることを可能にする音声認識検知デバイスを含み得る。

【0096】

ユーザインターフェイス入力デバイスはまた、限定されないが、3次元（3D）マウス、ジョイスティックまたはポインティングスティック、ゲームパッドおよびグラフィックタブレット、およびスピーカ等のオーディオ／ビジュアルデバイス、デジタルカメラ、デジタルカムコーダー、ポータブルメディアプレイヤー、ウェブカメラ、イメージスキャナ、指紋スキャナ、バーコードリーダー3Dスキャナ、3Dプリンタ、レーザ測距装置、および視線追跡デバイスを含み得る。加えて、ユーザインターフェイス入力デバイスは、たとえば、コンピュータ断層撮影装置、磁気共鳴撮像装置、ポジトロン断層撮影装置、医療用超音波検査装置等の医療用撮像入力デバイスを含み得る。ユーザインターフェイス入力デバイスはまた、たとえば、MIDIキーボード、デジタル楽器などのような音声入力装置を含み得る。

【0097】

ユーザインターフェイス出力デバイスは、ディスプレイサブシステム、表示灯、または

10

20

30

40

50

音声出力装置等の非視覚的ディスプレイなどを含み得る。ディスプレイサブシステムは、陰極線管（CRT）、液晶ディスプレイ（LCD）またはプラズマディスプレイを用いるもの等のフラットパネルデバイス、投影デバイス、タッチスクリーンなどであってもよい。一般的に、「出力デバイス」という用語を使用する場合は、コンピュータシステム1700からユーザまたは他のコンピュータに情報を出力するための可能なすべてのタイプのデバイスおよびメカニズムを含むことを意図している。たとえば、ユーザインターフェイス出力デバイスは、限定されないが、モニタ、プリンタ、スピーカ、ヘッドホン、カーナビゲーションシステム、プロッタ、音声出力デバイス、およびモデム等の、テキスト、図形、およびオーディオ／ビデオ情報を視覚的に伝えるさまざまなディスプレイデバイスを含み得る。

10

【0098】

コンピュータシステム1700は、現在はシステムメモリ1710内にあるものとして示されているソフトウェア要素を含むストレージサブシステム1718を含み得る。システムメモリ1710は、処理ユニット1704上にローディング可能であり処理ユニット上で実行可能なプログラム命令と、これらのプログラムの実行中に生成されたデータとを格納することができる。

【0099】

コンピュータシステム1700の構成および種類に応じて、システムメモリ1710は、揮発性（たとえばランダムアクセスメモリ（RAM））であってもよく、および／または不揮発性（たとえば読出専用メモリ（ROM）、フラッシュメモリなど）であってもよい。典型的に、RAMは、処理ユニット1704が直ちにアクセス可能でありおよび／または現在操作し実行している、データおよび／またはプログラムモジュールを含む。いくつかの実装例において、システムメモリ1710は、スタティックランダムアクセスメモリ（SRAM）またはダイナミックランダムアクセスメモリ（DRAM）等の複数の異なる種類のメモリを含み得る。いくつかの実装例において、たとえば起動中のコンピュータシステム1700内の要素間の情報の転送を支援する基本ルーチンを含む基本入出力システム（BIOS）は、典型的にはROMに格納することができる。例として、限定されないが、システムメモリ1710はまた、クライアントアプリケーション、ウェブブラウザ、ミッドティアアプリケーション、リレーショナルデータベース管理システム（RDBMS）などを含み得るアプリケーションプログラム1712と、プログラムデータ1714と、オペレーティングシステム1716とを示している。例として、オペレーティングシステム1716は、さまざまなバージョンのMicrosoft Windows（登録商標）、Apple Macintosh（登録商標）、および／またはLinux（登録商標）オペレーティングシステム、市

20

30

場で入手可能な多様なUNIX（登録商標）またはUNIX（登録商標）系オペレーティングシステム（限定されないが多様なGNU/Linux（登録商標）オペレーティングシステム、Google Chrome（登録商標）OSなどを含む）、および／またはiOS、Windows（登録商標）Phone、Android（登録商標）OS、BlackBerry（登録商標）10 OS、およびPalm（登録商標）OSオペレーティングシステム等のモバイルオペレーティングシステムを含み得る。

40

【0100】

ストレージサブシステム1718はまた、いくつかの実施形態の機能を提供する基本的なプログラミングおよびデータ構造を格納するための有形のコンピュータ読取可能記憶媒体を提供することができる。プロセッサによって実行されると上記機能を実行するソフトウェア（プログラム、コードモジュール、命令）は、ストレージサブシステム1718に格納することができる。これらのソフトウェアモジュールまたは命令は、処理ユニット1704によって実行されてもよい。ストレージサブシステム1718はまた、本発明に従い使用されるデータを格納するためのリポジトリを提供することができる。

【0101】

ストレージサブシステム1718はまた、コンピュータ読取可能記憶媒体1722にさ

50

らに接続することが可能なコンピュータ読取可能記憶媒体リーダー1720を含み得る。システムメモリ1710とともに、また、任意でシステムメモリ1710と組み合わせられて、コンピュータ読取可能記憶媒体1722は、コンピュータ読取可能情報を一時的におよび／またはより恒久的に含む、格納する、送信する、および取り出すための、リモート、ローカル、固定、および／またはリムーバブル記憶装置プラス記憶媒体を包括的に表すことができる。

【0102】

コードまたはコードの一部を含むコンピュータ読取可能記憶媒体1722はまた、限定されないが、情報の記憶および／または送信のための任意の方法または技術で実現される、揮発性および不揮発性、リムーバブルおよび非リムーバブル媒体等の、記憶媒体および通信媒体を含む、当該技術において周知のまたは使用されている適切な任意の媒体を含み得る。これは、RAM、ROM、電子的に消去可能プログラム可能なROM（EEPROM）、フラッシュメモリまたはその他のメモリ技術、CD-ROM、デジタル多目的ディスク（DVD）、もしくはその他の光ストレージ、磁気カセット、磁気テープ、磁気ディスクストレージもしくはその他の磁気記憶装置、または、他の有形のコンピュータ読取可能媒体等の、有形のコンピュータ読取可能記憶媒体を含み得る。これはまた、所望の情報を送信するために使用することができコンピュータシステム1700がアクセスすることができる、データ信号、データ送信、またはその他任意の媒体等の、非有形コンピュータ読取可能媒体を含み得る。

【0103】

例として、コンピュータ読取可能記憶媒体1722は、非リムーバブル不揮発性磁気媒体からの読取およびこの媒体への書込を行うハードディスクドライブ、リムーバブル不揮発性磁気ディスクからの読取およびこのディスクへの書込を行う磁気ディスクドライブ、ならびに、CD-ROM、DVD、Blu-Ray（登録商標）ディスク、またはその他の光学媒体等のリムーバブル不揮発性光ディスクからの読取およびこのディスクへの書込を行う光ディスクドライブを含み得る。コンピュータ読取可能記憶媒体1722は、限定されないが、Zip（登録商標）ドライブ、フラッシュメモリカード、ユニバーサルシリアルバス（USB）フラッシュドライブ、セキュアデジタル（SD）カード、DVDディスク、デジタルビデオテープなどを含み得る。コンピュータ読取可能記憶媒体1722はまた、フラッシュメモリベースのSSD、エンタープライズフラッシュドライブ、ソリッドステートROMなどのような、不揮発性メモリベースのソリッドステートドライブ（SSD）、ソリッドステートRAM、ダイナミックRAM、スタティックRAM等の揮発性メモリベースのSSD、DRAMベースのSSD、磁気抵抗RAM（MRAM）SSD、ならびにDRAMおよびフラッシュメモリベースのSSDの組み合わせを用いるハイブリッドSSDを、含み得る。ディスクドライブおよびこれらに対応付けられたコンピュータ読取可能媒体は、コンピュータ読取可能命令、データ構造、プログラムモジュール、およびコンピュータシステム1700のためのその他のデータの揮発性ストレージを提供することができる。

【0104】

通信サブシステム1724は、他のコンピュータシステムおよびネットワークに対するインターフェイスを提供する。通信サブシステム1724は、他のシステムからデータを受信し、コンピュータシステム1700から他のシステムにデータを送信するためのインターフェイスの役割を果たす。たとえば、通信サブシステム1724は、コンピュータシステム1700がインターネットを介して1つ以上のデバイスに接続することを可能にする。いくつかの実施形態において、通信サブシステム1724は、（たとえば携帯電話技術、3G、4GもしくはEDGE（enhanced data rates for global evolution）等の高度データネットワーク技術、WiFi（登録商標）（IEEE 802.11系規格、または他の移動通信技術、またはそれらの任意の組み合わせを用いて）無線音声および／またはデータネットワークにアクセスするための無線周波数（RF）トランシーバーコンポーネント、グローバルポジショニングシステム（GPS）受信機コンポーネント、

および／またはその他のコンポーネントを含み得る。いくつかの実施形態において、通信サブシステム1724は、無線インターフェイスに加えてまたはその代わりに、有線ネットワークコネクティビティ（たとえばイーサネット（登録商標））を提供することができる。

【0105】

いくつかの実施形態において、通信サブシステム1724は、コンピュータシステム1700を使用し得る1人以上のユーザの代わりに、構造化および／または非構造化データフィード1726、イベントストリーム1728、イベントアップデート1730などの形態の入力通信を受けることもできる。

【0106】

例として、通信サブシステム1724は、Twitter（登録商標）フィード、Facebook（登録商標）アップデート、Rich Site Summary（RSS）フィード等のウェブフィード、

および／または1つ以上の第3者情報源からのリアルタイムアップデート等の、ソーシャルネットワークおよび／または他の通信サービスのユーザからのデータフィード1726をリアルタイムで受信するように構成されてもよい。

【0107】

加えて、通信サブシステム1724は、明確な終わりがなく本質的に連続しているまたは無限である可能性があるリアルタイムイベントのイベントストリーム1728および／またはイベントアップデート1730を含み得る、連続データストリームの形態のデータを受信するように構成されてもよい。連続データを生成するアプリケーションの例は、たとえば、センサデータアプリケーション、金融ティッカー、ネットワーク性能測定ツール（たとえばネットワークモニタリングおよびトラフィック管理アプリケーション）、クリックストリーム分析ツール、自動車交通監視などを含み得る。

【0108】

通信サブシステム1724はまた、構造化および／または非構造化データフィード1726、イベントストリーム1728、イベントアップデート1730などを、コンピュータシステム1700に結合された1つ以上のストリーミングデータソースコンピュータと通信し得る1つ以上のデータベースに出力するように構成されてもよい。

【0109】

コンピュータシステム1700は、ハンドヘルドポータブルデバイス（たとえばiPhone（登録商標）携帯電話、iPad（登録商標）コンピューティングタブレット、PDA）、ウェアラブルデバイス（たとえばGoogle Glass（登録商標）ヘッドマウントディスプレイ）、PC、ワークステーション、メインフレーム、キオスク、サーバラック、またはその他任意のデータ処理システムを含む、さまざまなタイプのうちの1つであってもよい。

【0110】

コンピュータおよびネットワークには常に変化しているという性質があるので、図面に示されるコンピュータシステム1700の説明は、具体的な一例を意図しているにすぎない。図面に示されるシステムよりも多くのまたは少ないコンポーネントを有するその他多数の構成が可能である。たとえば、カスタマイズされたハードウェアが使用されてもよく、および／または特定の要素がハードウェア、ファームウェア、ソフトウェア（アプレットを含む）、または組み合わせで実現されてもよい。さらに、ネットワーク入出力デバイスのようなその他のコンピューティングデバイスに対する接続を用いてもよい。本明細書で提供される開示および教示に基づいて、当業者は、各種実施形態を実現するためのその他のやり方および／または方法を理解するであろう。

【0111】

上述の記載では、説明のために、本発明の各種実施形態の十分な理解が得られるよう、数多くの具体的な詳細を述べている。しかしながら、当業者には、本発明の実施形態はこ

10

20

30

40

50

これらの具体的な詳細のうちの一部がなくても実施し得ることが明らかであろう。その他の例では、周知の構造およびデバイスはブロック図の形態で示されている。

【0112】

上述の記載は、具体例としての実施形態を提供しているだけであって、本開示の範囲、利用可能性、または構成を限定することを意図しているのではない。むしろ、具体例としての実施形態の上述の記載は、具体例としての実施形態を実現することを可能にする説明を当業者に提供するであろう。以下の請求項に記載の本発明の精神および範囲から逸脱することなく、要素の機能および構成に各種変更を加え得ることが理解されるはずである。

【0113】

上述の記載において、具体的な詳細は、実施形態の十分な理解を得るために提供されている。しかしながら、当業者は、実施形態はこれらの具体的な詳細がなくても実現し得ることを理解するであろう。たとえば、回路、システム、ネットワーク、プロセス、およびその他のコンポーネントは、実施形態を不必要な詳細で不明瞭にすることがないように、ブロック図の形態のコンポーネントとして示されている場合がある。その他の例において、周知の回路、プロセス、アルゴリズム、構造、および技術は、実施形態を不明瞭にすることを避けるために不必要な詳細なしで示されている場合がある。

【0114】

また、個々の実施形態は、フローチャート、フロー図、データフロー図、構造図、またはブロック図として示されるプロセスとして記載されている場合があることが注目される。フローチャートは動作を逐次プロセスとして説明している場合があるが、動作のうちの多くは並列または同時に実行することができる。加えて、動作の順序は構成し直してもよい。プロセスは、その動作が完了すると終了するが、図面に含まれていない追加のステップを有する可能性がある。プロセスは、方法、関数、手順、サブルーチン、サブプログラムなどに対応し得る。プロセスが関数に対応する場合、その終了は、この関数が呼び出し側関数またはメイン関数に戻ることに相当し得る。

【0115】

「コンピュータ読取可能媒体」という用語は、携帯型または固定記憶装置、光記憶装置、無線チャネル、ならびに、命令および／またはデータを格納する、含む、または保持することが可能なその他の各種媒体を含むが、これらに限定される訳ではない。コードセグメントまたはマシン実行可能命令は、手順、関数、サブプログラム、プログラム、ルーチン、サブルーチン、モジュール、ソフトウェアパッケージ、クラス、または、命令、データ構造、もしくはプログラムステートメントの任意の組み合わせを表し得る。コードセグメントは、情報、データ、引数、パラメータ、またはメモリコンテンツを送ることおよび／または受けることにより、別のコードセグメントまたはハードウェア回路に結合されてもよい。情報、引数、パラメータ、データなどは、メモリ共有、メッセージパッシング、トークンパッシング、ネットワーク送信などを含む任意の適切な手段を介して、送る、転送する、または送信することができる。

【0116】

さらに、実施形態は、ハードウェア、ソフトウェア、ファームウェア、ミドルウェア、マイクロコード、ハードウェア記述言語、またはその任意の組み合わせによって実現することができる。ソフトウェア、ファームウェア、ミドルウェアまたはマイクロコードで実現するとき、必要なタスクを実行するためのプログラムコードまたはコードセグメントは、マシン読取可能媒体に格納されていてもよい。プロセッサ（複数のプロセッサ）が必要なタスクを実行してもよい。

【0117】

上述の明細書では、本発明の局面を、その具体的な実施形態を参照しながら説明しているが、本発明がこれらの実施形態に限定される訳ではないことを当業者は認識するであろう。上記発明の各種特徴および局面は、個別に使用されてもよく、またはともに使用されてもよい。さらに、実施形態は、本明細書のより広い精神および範囲から逸脱することなく、本明細書に記載の環境およびアプリケーションを超える、任意の数の環境およびアプ

10

20

30

40

50

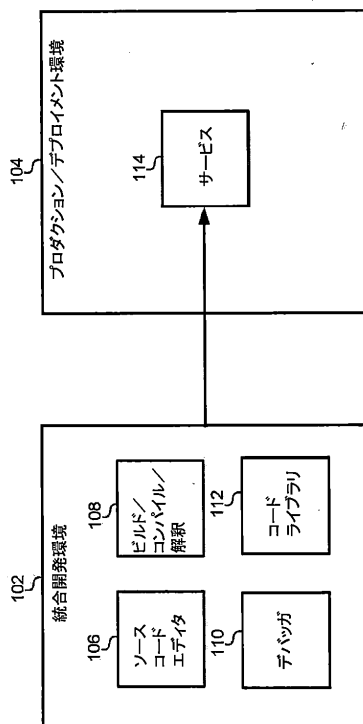
リケーションで利用することができる。したがって、本明細書および図面は、限定的なものではなく例示的なものであるとみなされねばならない。

【0118】

加えて、説明のために、方法は特定の順序で記載されている。代替実施形態においてこれらの方法は記載されている順序と異なる順序で実行され得ることが理解されるはずである。上記方法は、ハードウェアコンポーネントによって実行されてもよく、またはマシン実行可能命令のシーケンスで実施されてもよいことも、理解されるはずである。マシン実行可能命令は、当該命令でプログラミングされた汎用もしくは専用プロセッサまたは論理回路のようなマシンに当該方法を実行させるために使用することができる。これらのマシン実行可能命令は、CD-ROMもしくはその他のタイプの光ディスク、フロッピー（登録商標）ディスク、ROM、RAM、EPROM、EEPROM、磁気もしくは光カード、フラッシュメモリ、または、電子命令を格納するのに適したその他のタイプのマシン読取可能媒体等の、1つ以上のマシン読取可能媒体に、格納されてもよい。これに代えて、当該方法はハードウェアとソフトウェアとの組み合わせによって実行されてもよい。

【図面】

【図1】



【図2】

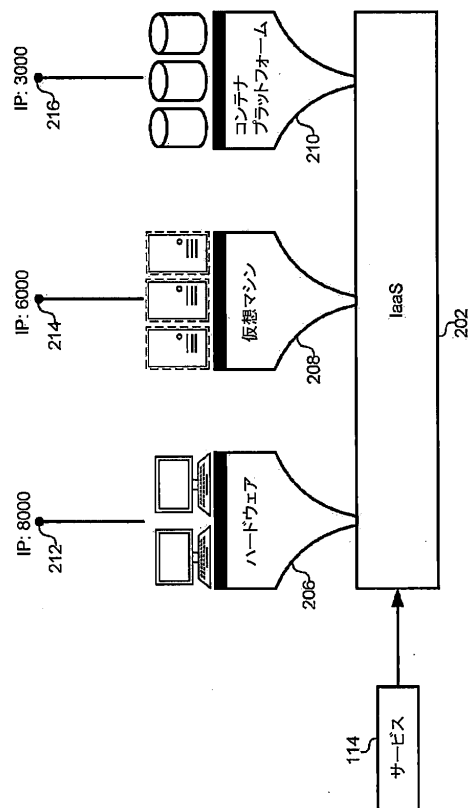


FIG. 1

FIG. 2

10

20

30

40

50

【図 6 B】

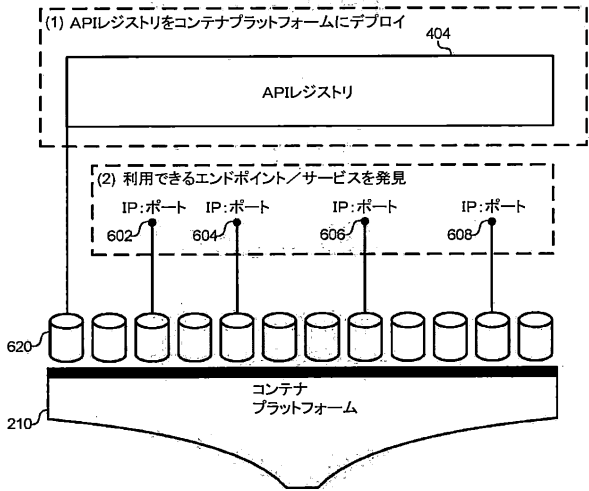


FIG. 6B

【図 7 A】

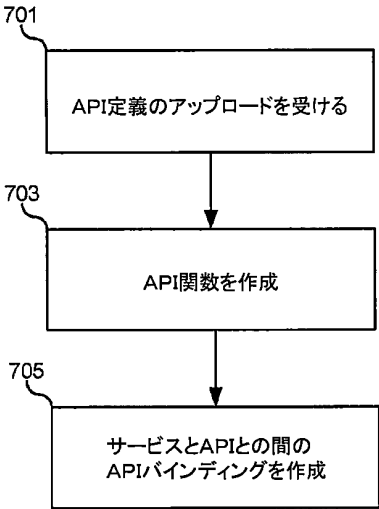


FIG. 7A

【図 7 B】

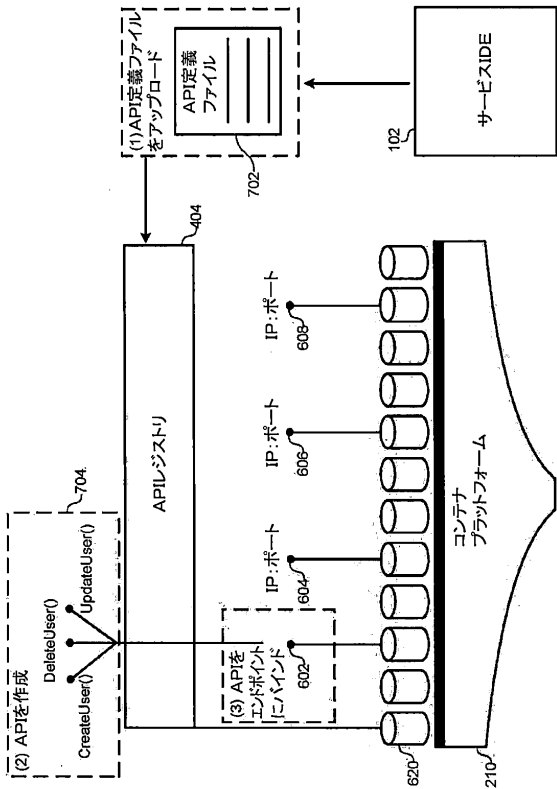


FIG. 7B

【図 8】

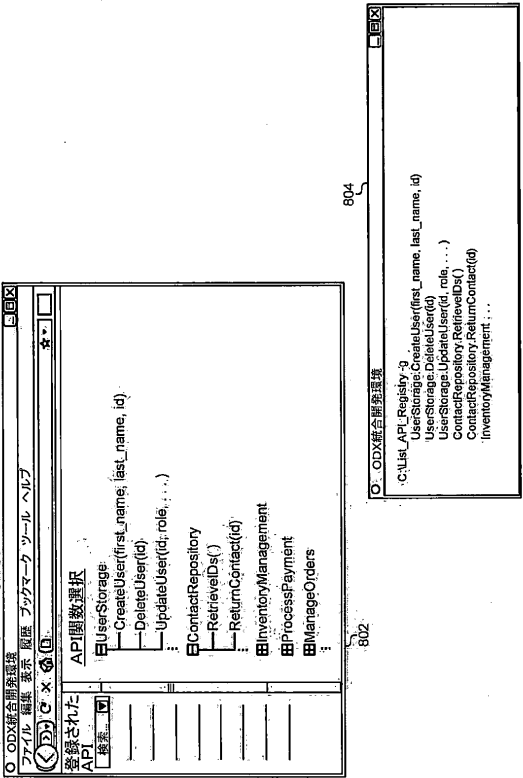


FIG. 8

【図 9】

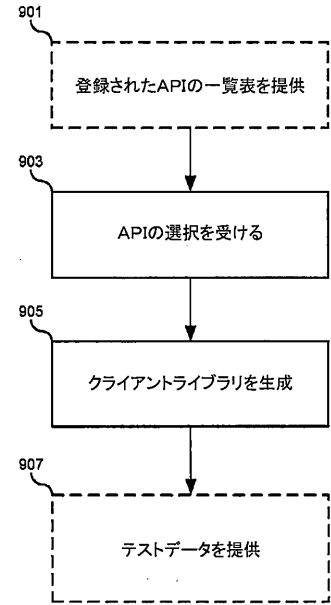


FIG. 9

【図 10】

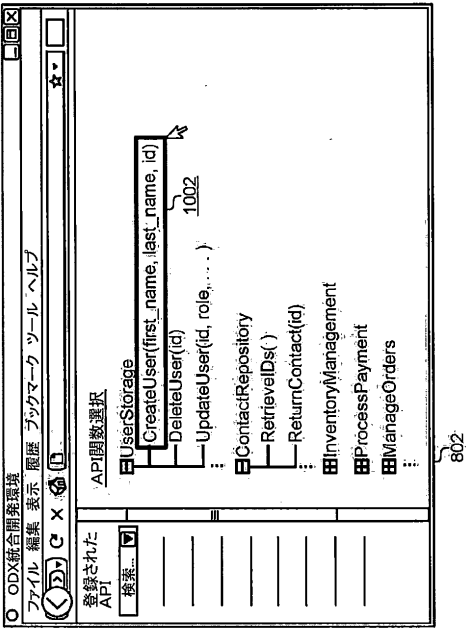


FIG. 10

【図 11】

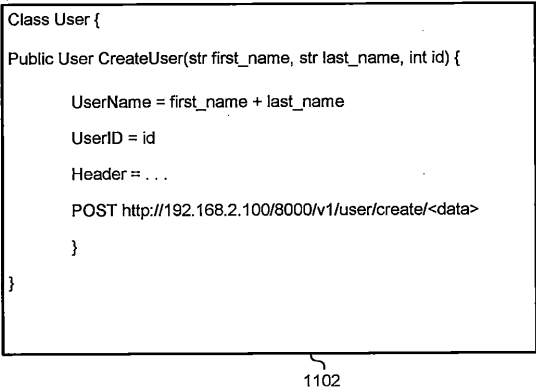


FIG. 11

【図 12】

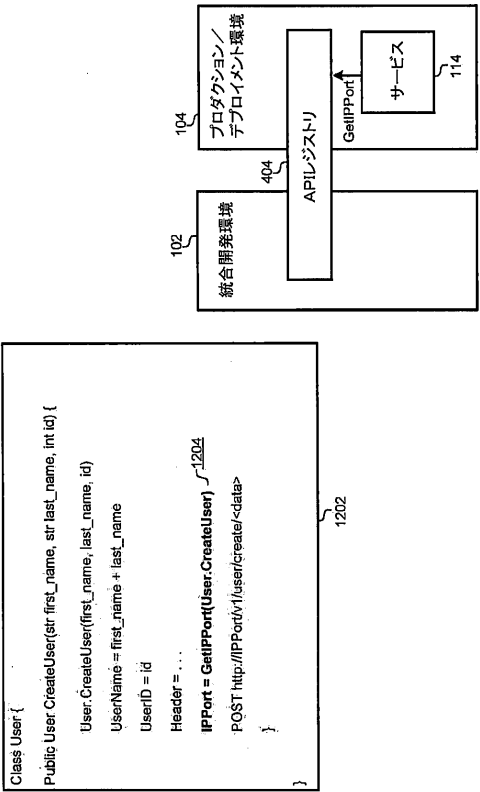


FIG. 12

【図 1 3】

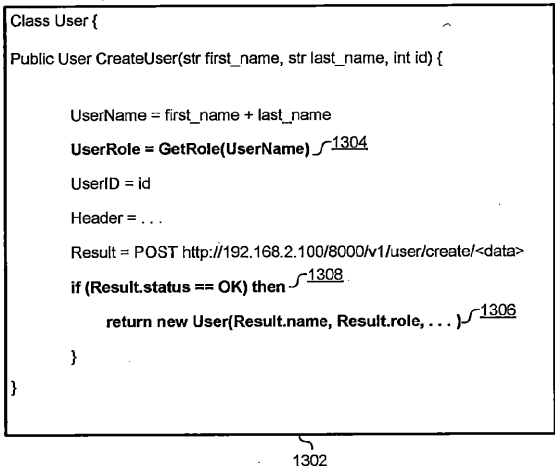


FIG. 13

【図 1 4】

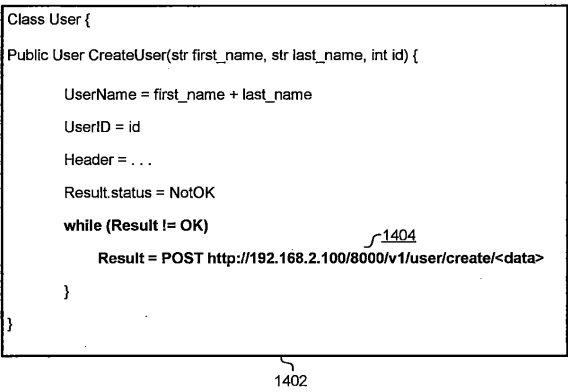


FIG. 14

【図 1 5】

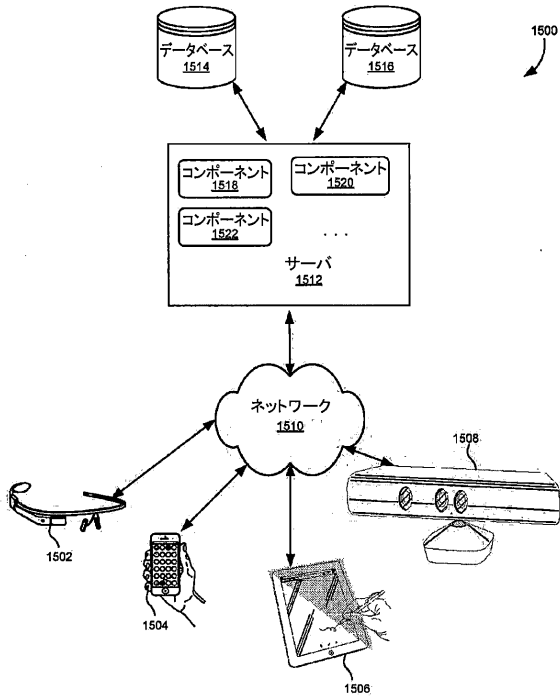


FIG. 15

【図 1 6】

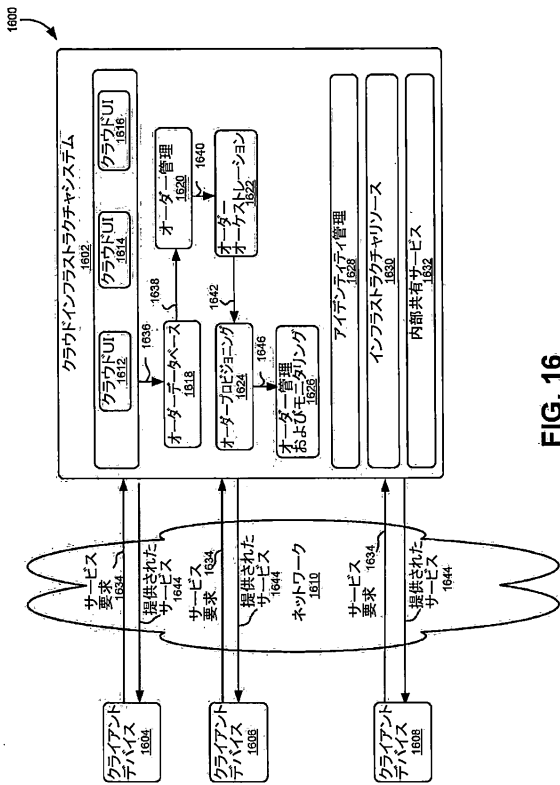


FIG. 16

【図 17】

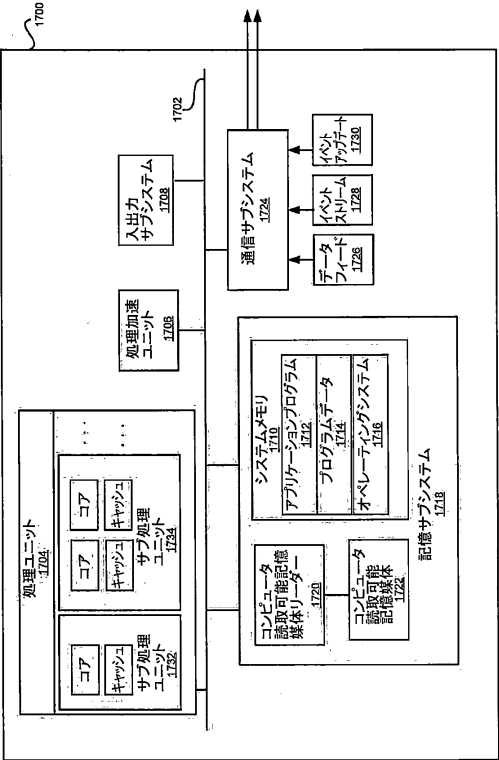


FIG. 17

10

20

30

40

50

【手続補正書】

【提出日】令和6年5月17日(2024.5.17)

【手続補正1】

【補正対象書類名】特許請求の範囲

【補正対象項目名】全文

【補正方法】変更

【補正の内容】

【特許請求の範囲】

【請求項1】

コンテナ環境内の登録されたサービスエンドポイントに対するアプリケーションプログラミングインターフェイス（API）関数を与える方法であって、前記方法は、 10

1つ以上のプロセッサが、APIレジストリにおいて、コンテナ環境における第1のサービスの使用を求める要求を第2のサービスから受けるステップを含み、前記APIレジストリは、前記第1のサービスをコールする1つ以上のAPI関数のリストを有し、前記方法はさらに、

1つ以上のプロセッサが、前記APIレジストリを用いて、前記第2のサービスから前記コンテナ環境における前記第1のサービスへのコールを実行するための前記1つ以上のAPI関数を含む、前記第2のサービスのためのクライアントライブラリを、生成するステップを含む、方法。

【請求項2】

1つ以上のプロセッサが、前記APIレジストリにおいて、前記第1のサービスのエンドポイントに対する前記1つ以上のAPI関数からの要求を受けるステップと、 20

1つ以上のプロセッサが、前記APIレジストリを用いて、前記第2のサービスにおける前記1つ以上のAPI関数が前記コンテナ環境における前記第1のサービスの前記エンドポイントへのコールを実行できるように、前記第1のサービスの前記エンドポイントを与えるステップとをさらに含む、請求項1に記載の方法。

【請求項3】

1つ以上のプロセッサが、前記第1のサービスの使用を求める前記第2のサービスからの前記要求を、開発環境において受ける、請求項1または2に記載の方法。

【請求項4】

1つ以上のプロセッサが、前記第1のサービスの前記エンドポイントに対する前記1つ以上のAPI関数からの前記要求を、プロダクション環境において受ける、請求項2に記載の方法。 30

【請求項5】

1つ以上のプロセッサが、前記APIレジストリにおいてAPI定義を受けるステップをさらに含み、前記API定義は、

前記コンテナ環境にデプロイされたコンテナにカプセル化された前記第1のサービスのエンドポイントと、

前記1つ以上のAPI関数とを含み、前記方法はさらに、

1つ以上のプロセッサが、前記APIレジストリを用いて、前記1つ以上のAPI関数と前記第1のサービスの前記エンドポイントとの間のバインディングを作成するステップを含む、請求項1～4のいずれかに記載の方法。 40

【請求項6】

1つ以上のプロセッサが、前記クライアントライブラリを前記第2のサービスに与えた後に、前記第1のサービスのエンドポイントに対する変更を受けるステップと、

1つ以上のプロセッサが、前記1つ以上のAPI関数からの要求を受ける前に、前記エンドポイントに対する前記変更後の前記第1のサービスと前記エンドポイントとの間の新たなバインディングを作成するステップとをさらに含む、請求項1～5のいずれかに記載の方法。

【請求項7】

50

前記クライアントライブラリは、前記 A P I 関数を実現するメンバ関数とともにオブジェクトクラスを含む、請求項 1 ～ 6 のいずれかに記載の方法。

【請求項 8】

前記クライアントライブラリは、前記第 1 のサービスのエンドポイントに対するダイレクトコールを含む、請求項 1 ～ 7 のいずれかに記載の方法。

【請求項 9】

1 つ以上のプロセッサが、前記クライアントライブラリを用いて、前記 1 つ以上の A P I 関数の 1 つ以上のパラメータとして受けたデータを、前記第 1 のサービスのエンドポイントに対する H T T P コールに再フォーマットする、請求項 1 ～ 8 のいずれかに記載の方法。

10

【請求項 10】

1 つ以上のプロセッサが、前記クライアントライブラリを用いて、前記第 1 のサービスのエンドポイントの前記 H T T P コールに加えるための追加情報を第 3 のサービスに要求する、請求項 9 に記載の方法。

【請求項 11】

前記クライアントライブラリは、前記第 1 のサービスに対する失敗コールをリトライするコードを含む、請求項 1 ～ 10 のいずれかに記載の方法。

【請求項 12】

前記クライアントライブラリは、結果値を前記第 1 のサービスの結果セットから抽出し前記結果値を前記 1 つ以上の A P I 関数に対するリターン値としてパッケージングするコードを含む、請求項 1 ～ 11 のいずれかに記載の方法。

20

【請求項 13】

1 つ以上のプロセッサに請求項 1 ～ 12 のいずれかに記載の方法を実行させるプログラム。

【請求項 14】

前記第 1 のサービスのエンドポイントは、
I P アドレスと、
ポート番号とを含む、請求項 13 に記載のプログラム。

【請求項 15】

前記コンテナ環境は、コンテナにカプセル化された複数のサービスを含む、請求項 13 または 14 に記載のプログラム。

30

【請求項 16】

前記 A P I レジストリは、前記コンテナ環境内のコンテナにカプセル化されたサービスとしてデプロイされる、請求項 13 ～ 15 のいずれかに記載のプログラム。

【請求項 17】

前記方法は、1 つ以上のプロセッサが、前記 A P I レジストリを用いて、前記コンテナ環境にデプロイされているコンテナ化されたサービスの利用できるエンドポイントを識別することをさらに含む、請求項 13 ～ 16 のいずれかに記載のプログラム。

【請求項 18】

1 つ以上のプロセッサと、

1 つ以上のメモリデバイスとを備えるシステムであって、前記 1 つ以上のメモリデバイスは、前記 1 つ以上のプロセッサによって実行されると前記 1 つ以上のプロセッサに動作を実行させる命令を含み、前記動作は、

40

A P I レジストリにおいて、コンテナ環境における第 1 のサービスの使用を求める要求を第 2 のサービスから受けることを含み、前記 A P I レジストリは、前記第 1 のサービスをコールする 1 つ以上の A P I 関数のリストを有し、前記動作はさらに、

前記 A P I レジストリが、前記第 2 のサービスから前記コンテナ環境における前記第 1 のサービスへのコールを実行するための前記 1 つ以上の A P I 関数を含む、前記第 2 のサービスのためのクライアントライブラリを、生成することを含む、システム。

【請求項 19】

50

前記動作は、前記APIレジストリに登録されたすべてのサービスに対するAPI関数のリストをユーザインターフェイスに表示させることをさらに含む、請求項18に記載のシステム。

【請求項20】

前記第1のサービスの使用を求める前記要求を前記第2のサービスから受けることは、当該要求を前記ユーザインターフェイスに表示された前記1つ以上のAPI関数のうちの1つの選択を受付けることによって受けることを含む、請求項19に記載のシステム。

【外国語明細書】

2024105342000021.pdf

10

20

30

40

50

フロントページの続き

ワンハンドレッドアンドトゥエンティエイス・ストリート、 8 5 3 0