(19)日本国特許庁(JP)

(12)特許公報(B2)

(11)特許番号
特許第7246381号
(P7246381)

(45)発行日　令和5年3月27日(2023.3.27)

(24)登録日　令和5年3月16日(2023.3.16)

(51)国際特許分類

		FI		
G06F	*8/61 (2018.01)*	G06F	8/61	
G06F	*9/455(2018.01)*	G06F	9/455	150

請求項の数　19　（全59頁）

(21)出願番号　特願2020-518526(P2020-518526)
(86)(22)出願日　平成30年9月28日(2018.9.28)
(65)公表番号　特表2020-536321(P2020-536321A)
(43)公表日　令和2年12月10日(2020.12.10)
(86)国際出願番号　PCT/US2018/053626
(87)国際公開番号　WO2019/068036
(87)国際公開日　平成31年4月4日(2019.4.4)
審査請求日　令和3年8月16日(2021.8.16)
(31)優先権主張番号　62/566,351
(32)優先日　平成29年9月30日(2017.9.30)
(33)優先権主張国・地域又は機関
米国(US)

(73)特許権者　502303739
オラクル・インターナショナル・コーポレイション
アメリカ合衆国カリフォルニア州９４０６５レッドウッド・シティー，オラクル・パークウェイ５００
(74)代理人　110001195
弁理士法人深見特許事務所
(72)発明者　カルダト，クラウディオ
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
(72)発明者　ショール，ボリス
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・

最終頁に続く

(54)【発明の名称】　環境要件に基づくコンテナのデプロイメント

(57)【特許請求の範囲】

【請求項１】

サービスのためのマイクロサービスコンテナを複数のコンピューティング環境に分配する方法であって、前記方法は、

コンテナプラットフォームにデプロイされる複数のコンテナ化されたマイクロサービスでビルドされたサービスを受けるステップを含み、前記コンテナプラットフォームは複数のコンピューティング環境を含み、前記方法はさらに、

前記コンテナプラットフォームに前記サービスをデプロイするためのデプロイメント基準を受けるステップと、

前記複数のコンピューティング環境の特徴にアクセスするステップと、

前記複数のコンテナ化されたマイクロサービスを、前記デプロイメント基準と前記複数のコンピューティング環境の特徴とに基づいて、前記複数のコンピューティング環境にデプロイするステップとを含み、

前記デプロイメント基準は、前記サービスのデプロイが前記コンテナ化されたマイクロサービスのコロケーションについて最適化されねばならないという表示を含む、方法。

【請求項２】

前記複数のコンピューティング環境はクラウドコンピューティング環境を含む、請求項１に記載の方法。

【請求項３】

前記複数のコンピューティング環境はオンプレミスコンピューティング環境を含む、請

求項１に記載の方法。
【請求項４】
前記複数のコンテナ化されたマイクロサービスのうちの少なくとも１つは、１つのコンテナポッド内の複数のコンテナを含む、請求項１から３のいずれか１項に記載の方法。
【請求項５】
前記複数のコンテナ化されたマイクロサービスのうちの少なくとも１つは、１つのコンテナポッド内の複数のマイクロサービスを含む１つのコンテナを含む、請求項１から３のいずれか１項に記載の方法。
【請求項６】
前記デプロイメント基準は、前記サービスのデプロイがセキュリティについて最適化されねばならないという表示を含む、請求項１から５のいずれか１項に記載の方法。
【請求項７】
前記デプロイメント基準は、前記サービスのデプロイが可用性について最適化されねばならないという表示を含む、請求項１から６のいずれか１項に記載の方法。
【請求項８】
前記デプロイメント基準は、前記サービスのデプロイがコストについて最適化されねばならないという表示を含む、請求項１から７のいずれか１項に記載の方法。
【請求項９】
前記デプロイメント基準は、前記サービスのデプロイがレイテンシについて最適化されねばならないという表示を含む、請求項１から８のいずれか１項に記載の方法。
【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プログラムであって、前記動作は、
コンテナプラットフォームにデプロイされる複数のコンテナ化されたマイクロサービスでビルドされたサービスを受けることを含み、前記コンテナプラットフォームは複数のコンピューティング環境を含み、前記動作はさらに、
前記コンテナプラットフォームに前記サービスをデプロイするためのデプロイメント基準を受けることと、
前記複数のコンピューティング環境の特徴にアクセスすることと、
前記複数のコンテナ化されたマイクロサービスを、前記デプロイメント基準と前記複数のコンピューティング環境の特徴とに基づいて、前記複数のコンピューティング環境にデプロイすることとを含み、
前記デプロイメント基準は、前記サービスのデプロイが前記コンテナ化されたマイクロサービスのコロケーションについて最適化されねばならないという表示を含む、プログラム。
【請求項１１】
前記複数のコンテナ化されたマイクロサービスの各々が、前記デプロイメント基準に対応する属性を含む、請求項１０に記載のプログラム。
【請求項１２】
前記マイクロサービスの属性の値を、前記複数のコンピューティング環境の特徴とマッチングさせる、請求項１１に記載のプログラム。
【請求項１３】
前記複数のコンテナ化されたマイクロサービスはコンテナプラットフォームスケジューラによってデプロイされる、請求項１０から１２のいずれか１項に記載のプログラム。
【請求項１４】
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、
コンテナプラットフォームにデプロイされる複数のコンテナ化されたマイクロサービスでビルドされたサービスを受けることを含み、前記コンテナプラットフォームは複数のコ

ンピューティング環境を含み、前記動作はさらに、
　前記コンテナプラットフォームに前記サービスをデプロイするためのデプロイメント基準を受けることと、
　前記複数のコンピューティング環境の特徴にアクセスすることと、
　前記複数のコンテナ化されたマイクロサービスを、前記デプロイメント基準と前記複数のコンピューティング環境の特徴とに基づいて、前記複数のコンピューティング環境にデプロイすることとを含み、
前記複数のコンテナ化されたマイクロサービスはＡＰＩレジストリによってデプロイされる、システム。
【請求項１５】
　前記ＡＰＩレジストリは、前記コンテナプラットフォーム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１４に記載のシステム。
【請求項１６】
　前記ＡＰＩレジストリは、
　統合開発環境（ＩＤＥ）において開発中のサービスと、
　前記コンテナプラットフォーム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１４または１５に記載のシステム。
【請求項１７】
　前記ＡＰＩレジストリは、前記複数のコンテナポッドのサービスエンドポイントを１つ以上のＡＰＩ関数にマッピングする、請求項１４から１６のいずれか１項に記載のシステム。
【請求項１８】
　前記ＡＰＩレジストリは、前記サービスをコールするために呼び出し側サービスのクライアントライブラリを自動的に生成する、請求項１４から１７のいずれか１項に記載のシステム。
【請求項１９】
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、
コンテナプラットフォームにデプロイされる複数のコンテナ化されたマイクロサービスでビルドされたサービスを受けることを含み、前記コンテナプラットフォームは複数のコンピューティング環境を含み、前記動作はさらに、
前記コンテナプラットフォームに前記サービスをデプロイするためのデプロイメント基準を受けることと、
前記複数のコンピューティング環境の特徴にアクセスすることと、
前記複数のコンテナ化されたマイクロサービスを、前記デプロイメント基準と前記複数のコンピューティング環境の特徴とに基づいて、前記複数のコンピューティング環境にデプロイすることとを含み、
前記デプロイメント基準は、前記サービスのデプロイが前記コンテナ化されたマイクロサービスのコロケーションについて最適化されねばならないという表示を含む、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 FOR

AUTOMATICALLY GENERATING CLIENT CODE LIBRARIESと題された米国特許出願第１６／１４７，３３４号（代理人整理番号０８８３２５－１０９０７４５）
・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ー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】

簡単な概要

いくつかの実施形態において、サービスのためのマイクロサービスコンテナを複数のコンピューティング環境に分配する方法は、コンテナプラットフォームにデプロイされる複

数のコンテナ化されたマイクロサービスでビルドされたサービスを受けるステップを含み得る。コンテナプラットフォームは複数のコンピューティング環境を含み得る。この方法はまた、コンテナプラットフォームにサービスをデプロイするためのデプロイメント基準を受けるステップと、複数のコンピューティング環境の特徴にアクセスするステップと、複数のコンテナ化されたマイクロサービスを、デプロイメント基準と複数のコンピューティング環境の特徴とに基づいて、複数のコンピューティング環境にデプロイするステップとを含み得る。

【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において、当該動作は、コンテナプラットフォームにデプロイされる複数のコンテナ化されたマイクロサービスでビルドされたサービスを受けることを含み得る。コンテナプラットフォームは複数のコンピューティング環境を含み得る。この動作はまた、コンテナプラットフォームにサービスをデプロイするためのデプロイメント基準を受けることと、複数のコンピューティング環境の特徴にアクセスすることと、複数のコンテナ化されたマイクロサービスを、デプロイメント基準と複数のコンピューティング環境の特徴とに基づいて、複数のコンピューティング環境にデプロイすることとを含み得る。

【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、コンテナプラットフォームにデプロイされる複数のコンテナ化されたマイクロサービスでビルドされたサービスを受けることを含む。コンテナプラットフォームは複数のコンピューティング環境を含み得る。この動作はまた、コンテナプラットフォームにサービスをデプロイするためのデプロイメント基準を受けることと、複数のコンピューティング環境の特徴にアクセスすることと、複数のコンテナ化されたマイクロサービスを、デプロイメント基準と複数のコンピューティング環境の特徴とに基づいて、複数のコンピューティング環境にデプロイすることとを含み得る。

【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。複数のコンピューティング環境はクラウドコンピューティング環境を含み得る。複数のコンピューティング環境はオンプレミスコンピューティング環境を含み得る。複数のコンテナ化されたマイクロサービスのうちの少なくとも１つは、１つのコンテナポッド内の複数のコンテナを含み得る。複数のコンテナ化されたマイクロサービスのうちの少なくとも１つは、１つのコンテナポッド内の複数のマイクロサービスを含む１つのコンテナを含み得る。デプロイメント基準は、サービスのデプロイがセキュリティについて最適化されねばならないという表示を含み得る。デプロイメント基準は、サービスのデプロイが可用性について最適化されねばならないという表示を含み得る。デプロイメント基準は、サービスのデプロイがコストについて最適化されねばならないという表示を含み得る。デプロイメント基準は、サービスのデプロイがレイテンシについて最適化されねばならないという表示を含み得る。デプロイメント基準は、サービスのデプロイがコンテナ化されたマイクロサービスのコロケーションについて最適化されねばならないという表示を含み得る。複数のコンテナ化されたマイクロサービスの各々が、デプロイメント基準に対応する属性を含み得る。マイクロサービスの属性の値を、複数のコンピューティング環境の特徴とマッチングさせてもよい。複数のコンテナ化されたマイクロサービスはコンテナプラットフォームスケジューラによってデプロイされてもよい。複数のコンテナ化されたマイクロサービスはＡＰＩレジストリによってデプロイされてもよい。ＡＰＩ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てもよい。ＡＰＩレジストリは、統合開発環境（ＩＤＥ）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、利用できるも

のであってもよい。ＡＰＩレジストリは、複数のコンテナポッドのサービスエンドポイントを１つ以上のＡＰＩ関数にマッピングしてもよい。ＡＰＩレジストリは、サービスをコールするために呼び出し側サービスのクライアントライブラリを自動的に生成してもよい。

【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、ＩＤＥ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ＡＰＩ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ＡＰＩ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、ＡＰＩ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、ＡＰＩ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の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、ＡＰＩ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、ＡＰＩレジストリに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。

【図８】いくつかの実施形態に係る、ＡＰＩレジストリに登録されたＡＰＩ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。

【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１５Ａ】いくつかの実施形態に係る、ＡＰＩプロパティをＡＰＩレジストリに与える方法を示す図である。

【図１５Ｂ】いくつかの実施形態に係る、如何にしてサービスがＡＰＩプロパティをＡＰＩレジストリに与えることができるかについてのハードウェア／ソフトウェア図を示す。

【図１６】いくつかの実施形態に係る、ＡＰＩレジストリがプロパティを用いて高可用性のサービスをデプロイする場合のハードウェア／ソフトウェア図を示す。

【図１７】いくつかの実施形態に係る、ＡＰＩレジストリを通じてエンドツーエンド暗号化を実施するプロパティのハードウェア／ソフトウェア図を示す。
【図１８】いくつかの実施形態に係る、ＡＰＩレジストリがサービス１８０８の使用ロギングを実現するためのプロパティを示す図である。
【図１９】いくつかの実施形態に係る、サービスの認証プロトコルを実施することが可能なプロパティのハードウェア／ソフトウェア図を示す。
【図２０】いくつかの実施形態に係る、サービスのランタイムインスタンス化を可能にするプロパティのハードウェア／ソフトウェア図を示す。
【図２１】いくつかの実施形態に係る、サービスのレート制限関数を実現するプロパティのハードウェア／ソフトウェア図を示す。
【図２２】いくつかの実施形態に係る、複数のマイクロサービスでビルドされたサービスの一例を示す図である。
【図２３】いくつかの実施形態に係る、サービスのデプロイメント環境を示す図である。
【図２４】いくつかの実施形態に係る、複数のコンピューティング環境へのサービスのデプロイメントを最適化する方法のフローチャートを示す図である。
【図２５】いくつかの実施形態に係る、サービス基準および環境／マイクロサービス属性の図を示す。
【図２６】いくつかの実施形態に係る、動作コストを最小にするように最適化されたサービスデプロイメントの一例を示す図である。
【図２７】いくつかの実施形態に係る、サービス可用性について最適化されたサービスデプロイメントの一例を示す図である。
【図２８】いくつかの実施形態に係る、セキュリティ問題について最適化されたサービスデプロイメントの一例を示す図である。
【図２９】いくつかの実施形態に係る、処理速度について最適化されたサービスデプロイメントの一例を示す図である。
【図３０】いくつかの実施形態に係る、同時に複数の基準に従って最適化されたサービスデプロイメントの一例を示す図である。
【図３１】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３２】実施形態のシステム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３３】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】
【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。

【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。

【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。

【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１１２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、ＩＤＥ１０２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、ＩＤＥ１０２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）、ＣｏｅＯＳ、Apache（登録商標） Mesosを、それ以外のものともに含み得る。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」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の、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。本開示よりも前において、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に、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った。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。本開示よりも前において、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可能性があるので、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。

【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。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。

【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。

【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。

【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。

【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。

【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使用できる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、ＩＤＥ１０２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。

【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、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定義ファイルを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。

【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。

【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。

【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。

【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１２０４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、ＡＰＩレジストリ４０４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、ＡＰＩレジストリ４０４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１１４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。

【００４７】

図１３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１３０２の実施形態を示す。クライアントライブラリ１３０２を用いた簡略化のために、クライアントライブラリ１３０２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、パラメー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１３０２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１３０２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１３０４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１３０４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ＨＴＴＰ　ＰＯＳＴ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。

【００４８】

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１３０２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、ＰＯＳＴコマンドはデータパケットをＲｅｓｕｌｔ（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１３０２は、Ｒｅｓｕｌｔ変数におけるデータフィールドをパースし、データをＲｅｓｕｌｔ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易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１３０６は、Ｒｅｓｕｌｔ変数からフィールドを抽出して使用することにより、ＡＰＩ関数から返される新たなＵｓｅｒオブジェクトを作成することができる。ＧＥＴコマンドを使用する別の例において、個々のＡＰＩ関数は、ＧＥＴコマンドからのＲｅｓｕｌｔ変数から異なるフィールドを抽出するＵｓｅｒクラスで生成することができる。たとえば、Ｕｓｅｒ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、Ｒｅｓｕｌｔ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。

【００４９】

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１３０２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１３０８を生成することができる。この例において、コード１３０８は、Ｒｅｓｕｌｔ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、ＰＯＳＴ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Ｕｓｅｒオブジェクトを返すことができる。Ｐｏｓｔ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。

【００５０】

図１４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１４０２を示す。図１３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Ｒｅｓｕｔ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】

上述のように、いくつかの実施形態はまた、ＡＰＩ定義とともに一組のＡＰＩプロパティをＡＰＩレジストリに与えてもよい。図１５Ａは、いくつかの実施形態に係る、ＡＰＩプロパティをＡＰＩレジストリに与える方法を示す。この方法はまた、ＡＰＩ定義のアップロードを受けることを含み得る（１５０１）。この方法はまた、ＡＰＩプロパティのアップロードを受けることを含み得る（１５０３）。プロパティのアップロードはＡＰＩ定義のアップロードと同じ送信の一部であってもよい。いくつかの実施形態において、ＡＰＩプロパティはＡＰＩ定義の一部であってもよい。いくつかの実施形態において、ＡＰＩプロパティは、プロパティがＡＰＩレジストリによって設定されねばならないことを示すためにチェックされる１つ以上のフラグまたは予め定められたデータフィールドであってもよい。いくつかの実施形態において、ＡＰＩプロパティは、何らかの予め構築されたフォーマットに従う必要はないが、代わりに、認証、暗号化などのような以下で説明する特徴をＡＰＩレジストリに実現させる命令コードで表すことができる。ＡＰＩプロパティは、各サービスごとにＡＰＩ定義とともに格納することができる。

【００５２】

この方法はさらに、サービスとＡＰＩとの間のＡＰＩバインディングを作成することを含み得る（１５０５）。この動作は先に詳述したように実行されてもよい。加えて、この方法は、ＡＰＩプロパティを用いて、サービスに対応付けられた１つ以上の動作を実行することを含み得る（１５０７）。ＡＰＩプロパティは、サービスのライフサイクル中のさまざまなフェーズで使用されてもよい。一般的に、これは、サービスのデプロイメント中、サービスのクライアントライブラリを生成するとき、および／またはサービスをコールするときに、プロパティに対応付けられた関数をＡＰＩプロパティを用いて実現することであると、説明することができる。これらの関数各々の例を以下でより詳細に説明する。

【００５３】

図１５Ａ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プロパティをＡＰＩレジストリに与える特定の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１５Ａ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。

【００５４】

図１５Ｂは、いくつかの実施形態に係る、如何にしてサービスがＡＰＩプロパティをＡＰＩレジストリに与えることができるかについてのハードウェア／ソフトウェア図を示す。ＩＤＥ１０２においてサービスを開発しているとき、サービス開発者は、ＡＰＩ定義ファイル１５０２と１つ以上のプロパティ１５０４とをＡＰＩレジストリ４０４に与えることができる。ＡＰＩレジストリ４０４は、ＩＤＥ１０２でもコンテナプラットフォームで

も実行時にアクセスできるので、ＡＰＩレジストリ４０４は、プロパティ１５０４を格納しこれらのプロパティを使用することにより、開発シナリオ中およびランタイムシナリオ中いずれにおいても、如何にしてサービスがデプロイされ、コールされ、および／またはクライアントライブラリを生成するために使用されるかに、影響を与えることができる。

【００５５】

図１６は、いくつかの実施形態に係る、ＡＰＩレジストリがプロパティを用いて高可用性のサービスをデプロイする場合のハードウェア／ソフトウェア図を示す。ＡＰＩレジストリ４０４は、特定のサービスのＡＰＩ定義ファイル１５０５に加えて、サービスが高いレジリエンスまたは高可用性を有するようにデプロイされねばならないことを示すプロパティ１６０２を受けてもよい。このプロパティ１６０２を、サービスを高可用性を有するようにデプロイするためにＡＰＩレジストリ４０４が実行する一組の命令として受けてもよい。この選択肢により、開発者は、このサービスについて「高可用性」を有するということが何を意味するかを定義することができる。たとえば、プロパティ１６０２は、ＡＰＩレジストリ４０４に、サービスの複数のインスタンス６０２、６０４をコンテナプラットフォーム２１０にデプロイさせる命令を含み得る。ＡＰＩレジストリ４０４は、これらの命令を実行することにより、自身で決定または判断を行う必要はなくなるが、代わりに、プロパティ１６０２の一部として与えられたデプロイメントコードを実行するだけでよい。

【００５６】

プロパティ１６０２を、高可用性のサービスをデプロイするためにＡＰＩレジストリ４０４において既存の命令を実行するという選択肢をＡＰＩレジストリ４０４に対して示す、フラグまたは設定として受けることもできる。この選択肢の場合、ＡＰＩレジストリ４０４は、プロパティ１６０２として、実行すべき任意のコードを受ける必要はない。むしろ、ＡＰＩレジストリ４０４は、高可用性プロパティ１６０２を認識し、ＡＰＩレジストリ４０４で保持されているコードを実行することにより、サービスの複数のインスタンス６０２、６０４をデプロイすることができる。これにより、ＡＰＩレジストリ４０４は、ＡＰＩレジストリ４０４に登録された任意のサービスのデプロイメントについて、「高可用性」であるということは何を意味するのかを定義することができる。

【００５７】

ＡＰＩレジストリ４０４はコンテナプラットフォーム２１０のランタイム環境に接続されているので、ＡＰＩレジストリ４０４は、コンテナプラットフォーム２１０とやり取りすることにより、サービスのランタイム可用性を決定するインスタンス６０２、６０４をデプロイすることができる。なお、図１６に示されるサービスの２つのインスタンス６０２、６０４は一例として与えられているにすぎず、限定を意図している訳ではない。高可用性サービスは、コンテナプラットフォームにデプロイされているサービスの３つ以上の冗長インスタンスを含み得る。

【００５８】

いくつかの実施形態は、デフォルトとして実行することが可能なコードをＡＰＩレジストリ４０４において含み得る。高可用性が望ましいという簡単な表示のみをプロパティ１６０２が含んでいる場合、ＡＰＩレジストリ４０４は自身のコードを実行することができる。サービスをデプロイするためのデプロイメントコードをプロパティ１６０２が含んでいる場合、ＡＰＩレジストリ４０４は代わりにプロパティ１６０２のコードを実行することができる。いくつかの場合において、プロパティ１６０２は、高可用性のサービスをデプロイするのに必要なコードの一部のみを含み得る。ＡＰＩレジストリ４０４は、プロパティ１６０２から与えられたコードの一部を実行し、次に、プロパティ１６０２から与えられない任意のコードをＡＰＩレジストリ４０４におけるコードを用いてを実行することができる。これにより、開発者らは、ＡＰＩレジストリ４０４において高可用性等のプロパティを如何にして実行するかについての既存の定義を上書きしつつも、登録されたサービスによって使用されることができるプロパティを実行するための統一された定義をＡＰＩレジストリ４０４が提供できるようにすることができる。

【００５９】
いくつかの実施形態において、高可用性プロパティはまた、サービスの複数のインスタンス６０２、６０４に要求を分散させるロードバランシングサービス６０６を、コンテナプラットフォーム２１０にデプロイさせてもよい。ロードバランシングサービス６０６のエンドポイントを、ＡＰＩレジストリ４０４に登録し、他のサービスが利用できるようにすることができる。これに代えてまたはこれに加えて、サービス６０２、６０４の複数のインスタンス各々をサービスエンドポイントとしてＡＰＩレジストリ４０４に登録してもよい。
【００６０】
以下で説明する各々の例において、図１６に関して説明したものと同じ原理を適用することができる。たとえば、下記のいずれかのプロパティにコードが伴っていてもよく、このコードをＡＰＩレジストリ４０４が受け、そうでなければＡＰＩレジストリ４０４が実行するであろうコードを無効にするために使用することができる。本開示よりも前においては、プロパティを実行する一方で同時にサービス開発者が必要に応じてこれらのプロパティを無効にできるようにするための統一されたデフォルトを作成する方法は存在していなかった。よって、ＡＰＩレジストリ４０４は、デフォルトとしてＡＰＩレジストリ４０４でコードを実行できるようにしつつ、開発者から受けたプロパティ１６０２によってこのコードを無効にすることもできるようにすることで、技術的課題を克服する。
【００６１】
図１７は、いくつかの実施形態に係る、ＡＰＩレジストリを通じてエンドツーエンド暗号化を実施するプロパティのハードウェア／ソフトウェア図を示す。ＡＰＩレジストリ４０４は、ＡＰＩ定義ファイル１５０５とともに、サービス１７０８をコールするためにエンドツーエンド暗号化を生じさせるコードを示すまたは含むプロパティ１７０４を受けてもよい。開発中、サービス１７０８は、サービス１７０８が受けたパケットを復号しサービス１７０８が返すパケットを暗号化する、自身の復号／暗号化コード１７１０を含むことができる。本開示よりも前においては、サービス１７０８のユーザがサービス１７０８と互換性のある暗号化を提供する必要があることを示す仕様書を、開発者が提供しなければならなかった。本実施形態は、呼び出し側サービス１７０６においてクライアントライブラリが如何にして生成されるかをサービス１７０８が規定できるようにしてサービス１７０８の暗号化との互換性を保証することにより、技術的課題を解決する。
【００６２】
いくつかの実施形態において、サービス１７０８の開発者は、暗号化／復号コード１７１０をサービス１７０８に含める必要はない。代わりに、サービス１７０８のエンドツーエンド暗号化を実施するよう、プロパティ１７０４がＡＰＩレジストリ４０４に命令するだけでよい。サービス１７０８がコンテナプラットフォーム２１０にデプロイされるとき、ＡＰＩレジストリ４０４は、デプロイ時にサービス１７０８に暗号化／復号コード１７１０が挿入されるようにすることができる。これにより、開発者が、プロパティ１７０４に基づいて異なる暗号化制度からの選択を行うことができる、および／または、ＡＰＩレジストリ４０４が、デフォルトとして好ましい暗号化制度を選択することができる。
【００６３】
エンドツーエンド暗号化には、サービス１７０８がデプロイされるときまたはその開発中に暗号化／復号コード１７１０をサービス１７０８に挿入することが必要なだけでなく、互換性のある暗号化／復号コードを呼び出し側サービス１７０６が含むことも必要である。先に述べたように、呼び出し側サービス１７０６がサービス１７０８を使用する必要があるとき、ＡＰＩレジストリ４０４は、簡単で効率的なやり方でサービス１７０８とやり取りするのに必要なコードを完全に実現する１つ以上のクライアントライブラリ１７０２を生成することができる。このクライアントライブラリ１７０２が生成されるとき、ＡＰＩレジストリ４０４は、プロパティ１７０４を分析することにより、サービス１７０８が使用する暗号化制度を判断することができる。次に、このプロパティ１７０４に基づいて、ＡＰＩレジストリ４０４は、呼び出し側サービス１７０６のために互換性のある暗号

化／復号コードをクライアントライブラリ１７０２に追加することができる。よって、呼び出し側サービス１７０６が要求をサービス１７０８に送るとき、情報は呼び出し側サービス１７０６で暗号化されサービス１７０８がこの情報を受けたときに復号されてもよい。同様に、サービス１７０８は、呼び出し側サービス１７０６に送られる前のレスポンスを暗号化することができ、そうすると、呼び出し側サービス１７０６は、このレスポンスを、クライアントライブラリ１７０２の外部に送る前に復号することができる。これにより、この暗号化プロセス全体を呼び出し側サービス１７０６の開発者にとって完全にトランスペアレントなものにすることができる。サービス１７０６をコールするときに互換性のある暗号化／復号制度を実現することが要求されるのではなく、プロパティ１７０４は、ＡＰＩレジストリ４０４が互換性のある暗号化／復号コードをクライアントライブラリ１７０２に既に生成していることを保証し、エンドツーエンド暗号化プロパティを実現することができる。

【００６４】

図１８は、いくつかの実施形態に係る、ＡＰＩレジストリがサービス１８０８の使用ロギングを実現するためのプロパティ１８０４を示す。本開示よりも前においては、サービスに対する要求の頻度、ソース、成功率などをモニタリングしロギングするためには、サービス自身がこの情報をロギングする必要があった。これに代わるものとしては、コンテナ環境がサービスをモニタリングしその使用情報をロギングする必要があった。サービス１８０８自体において情報をロギングすることは、極めて非効率であり、このサービスが扱うすべての要求についてのスループットを減速させる。同様に、特定のサービスに対して行われるコールすべてをモニタリングしロギングすることをコンテナプラットフォームに求めるときのオーバーヘッドも、コンテナサービスのスケジューリングおよびオーケストレーションに対する極めて大きなオーバーヘッドを表している。本実施形態は、この技術的課題を、サービス１８０８をコールするサービスのためのクライアントライブラリにコードを直接挿入することによって解決する。これにより、メモリの使用またはＣＰＵの使用という点でサービス１８０８のパフォーマンスに全く影響を与えることなく、サービス１８０８の使用をロギングしモニタリングすることができる。

【００６５】

ＡＰＩレジストリ４０４は、ＡＰＩ定義ファイル１５０５に加えて、使用ロギング１８０４を実現するコードを示すまたは含むプロパティ１８０４を受けることができる。呼び出し側サービス１８０６の開発者が要求をサービス１８０８に出すことを所望するとき、ＡＰＩレジストリ４０４は、サービス１８０８に関するアクティビティをロギングするためのコードを含むクライアントライブラリ１８０２を自動的に生成することができる。先に述べたように、このコードは、ＡＰＩレジストリ４０４が管理し実行するデフォルトコードに基づいて生成することができる、または、プロパティ１８０４とともに受けてＡＰＩレジストリ４０４が実行するコードによって生成することができる。

【００６６】

クライアントライブラリ１８０２における、アクティビティをロギングするためのコードは、サービス１８０８がコールされるたびにインクリメントされるカウンターと、サービス１８０８がコールされたときにアクティビティをログファイルにロギングさせる関数と、サービス１８０８に送られた要求およびサービス１８０８から受けたレスポンスの特徴をモニタリングし記録するその他の関数とを、含み得る。特定の実施形態に応じて、このコードは、サービス１８０８に対して行われる要求に対応付けられた多数の異なる種類の特徴をモニタリングしてもよい。たとえば、いくつかの実施形態は、サービス１８０８に対して行われたコールの総数をロギングしてもよい。いくつかの実施形態は、サービス１８０８から受けたレスポンスの成功率をロギングしてもよい。いくつかの実施形態は、サービス１８０８に対する要求において送られたデータのタイプをロギングしてもよい。いくつかの実施形態は、サービス１８０８がコールされた時刻またはサービス１８０８がコールされたときについてのその他の外部情報をロギングしてもよい。いくつかの実施形態は、サービス１８０８をデバッグするために使用することが可能なサービス１８０８へ

の／からの入力および出力をロギングしてもよい。いくつかの実施形態は、限定されることなく、これらの特徴のうちのいずれかまたはすべてを、任意の組み合わせでロギングしてもよい。

【００６７】

図１９は、いくつかの実施形態に係る、サービス１９０８の認証プロトコルを実施することが可能なプロパティ１９０４のハードウェア／ソフトウェア図を示す。いくつかのサービスは、要求に応答する前に、ユーザアイデンティティが認証されることおよびユーザがサービスの使用を認可されることを必要とする場合がある。本開示よりも前においては、認証および認可の手続がサービス１９０８自体で行われるという技術的課題があった。これは、サービス１９０８が受けるすべてのコールについてメモリの使用およびＣＰＵの使用という点でオーバーヘッドを増加させ、レスポンスにおけるサービスのレイテンシを大きくした。その結果、スループットが減少し、所定の期間中にサービス１９０８が処理できる要求の数が制限された。これらの実施形態は、この技術的課題を、認証／認可コードを、ＡＰＩレジストリ１４０４が自動的に生成するクライアントライブラリ１９０２に移動させることによって解決する。

【００６８】

呼び出し側サービス１９０６がサービス１９０８の使用を所望するとき、ＡＰＩレジストリ４０４は、認可および／認証を実行するためのコードを含むクライアントライブラリ１９０２を生成することができる。いくつかの実施形態において、これは、ユーザアイデンティティを特別に検証するおよび／またはユーザがサービス１９０８の使用を認可されるか否かを判断する外部認証／認可サービス１９２０にコンタクトすることを含み得る。外部認証／認可サービス１９２０は、アクセスマネージャ、ライトウェイトディレクトリアクセスプロトコル（Lightweight Directory Access Protocol）（ＬＤＡＰ）マネージャ、アクセスコントロールリスト（Access Control List）（ＡＣＬ）、ネットワーク認証プロトコルマネージャなどを、含み得る。そうすると、クライアントライブラリ１９０２内のコードは、認証／認可手続が成功したときにコールをサービス１９０８に送ることができる。

【００６９】

認証／認可の実施をＡＰＩレジストリ４０４およびクライアントライブラリ１９０２にオフロードすることにより、このコードをサービス１９０８から完全に除去することができる。外部認証／認可サービス１９２０とのやり取りには多大な遅延が伴うことが頻繁に生じ得るので、この遅延をサービス１９０８から取り除くことによってスループットを高めることができる。加えて、認証／認可の実施をサービス１９０８にハードコーディングするのではなく、サービス１９０８の開発者が、その代わりにＡＰＩレジストリ４０４に送られたプロパティ１９０４を用いて予め定められた認証／認可制度を選択するだけでよい。ＡＰＩレジストリ４０４は、認可／認証の予め定められたリストを、クライアントライブラリ１９０２の添付のインプリメンテーションコードとともに管理することができる。これはまた、認可および／または認証されることができないサービス１９０８に対して呼び出し側サービス１９０６が要求を送ることを防止する。代わりに、認証および／または認可ルーチンが失敗した場合はクライアントライブラリ１９０２でコールを中止してもよい。これにより、認証および／または認可された要求のみをサービス１９０８が受けることを保証する。

【００７０】

ＡＰＩレジストリ４０４が提供するもう１つの技術的改善は、登録されたいずれかのサービスのコードのうちのいずれかを変更することを要求されることなく、プロパティ１９０４が提供する機能のいずれかをアップグレードする機能である。たとえば、認証／認可コードは、ＡＰＩレジストリ１４０４が生成したクライアントライブラリ１９０２にオフロードされているので、クライアントライブラリ１９０２をアップデートすることにより、認証／認可制度を変更することが可能である。呼び出し側サービス１９０６またはサービス１９０８におけるコードのうちのいずれもモニタリングする必要はない。コードは１

つの場所でしか変更されないので、これは、そうでなければすべての個別サービスに送られる分散型パッチを伴うコード統合エラーの確率を大幅に減じる。

【００７１】

図２０は、いくつかの実施形態に係る、サービスのランタイムインスタンス化を可能にするプロパティ２００４のハードウェア／ソフトウェア図を示す。いくつかのサービスは、滅多に使用されない、または所定の期間中にしか使用されない場合がある。したがって、サービスをコンテナプラットフォームにデプロイすることが、常に、即時利用可能なサービスのインスタンスを実際にコンテナにインスタンス化することにつながる必要はない。仮想マシンとは異なり、コンテナは非常に素早くインスタンス化し作動させることが可能である。そのため、サービス開発者は、コールされたときにのみサービスをインスタンス化させることを所望する場合がある。サービス開発者はまた、所定の期間内においてのみサービスをインスタンス化させることを所望する場合がある。同じく、サービス開発者は、サービスインスタンスは所定の休止期間が過ぎると削除されねばならないと明示する場合がある。

【００７２】

ＡＰＩレジストリ４０４は、ＡＰＩ定義ファイル１５０５を受けることに加えて、ランタイムインスタンス化またはその他のインスタンス化パラメータを指定するプロパティ２００４を受けることができる。たとえば、このプロパティは、サービス２００８をデプロイ後にインスタンス化すべき１つ以上の期間の指定を含み得る。別の例において、このプロパティは、サービス２００８はオンデマンドでのみインスタンス化されねばならないという表示を含み得る。別の例において、このプロパティは、タイムアウト期間を指定してもよく、インスタンス化されたサービス２００８はこのタイムアウト期間後にコンテナプラットフォームから削除されねばならない。

【００７３】

呼び出し側サービス２００６がサービス２００８の使用を所望するとき、ＡＰＩレジストリ４０４は、サービス２００８のランタイムインスタンス化を処理するコードをクライアントライブラリ２００２に生成することができる。たとえば、クライアントライブラリ２００２のCreateInstance( )関数コールは、ＡＰＩレジストリ４０４に対するコールを作成することができる。そうすると、ＡＰＩレジストリはコンテナプラットフォーム２１０とやり取りすることにより、サービス２００８のオペレーティングインスタンスを利用できるか否かを判断することができる。利用できない場合、ＡＰＩレジストリ４０４は、コンテナプラットフォーム２０１０のコンテナにサービス２００８のインスタンスをインスタンス化するようコンテナプラットフォーム２１０に命令することができる。そうすると、コンテナプラットフォーム２１０は、エンドポイント（たとえばＩＰアドレスおよびポート番号）をＡＰＩレジストリ４０４に返すことができる。次に、ＡＰＩレジストリ４０４は、エンドポイントと、クライアントライブラリ２００２に作成されたＡＰＩ関数コールとの間のバインディングを作成することができる。次に、ＡＰＩレジストリ４０４は、このエンドポイントをクライアントライブラリ２００２に返すことができ、これを用いることにより、呼び出し側サービス２００６と新たにインスタンス化されたサービス２００８との間の直接接続を作成することができる。

【００７４】

所定の期間中のみインスタンス化されるべきサービスの場合、ＡＰＩレジストリ４０４は、特定のサービスのインスタンス化および削除時間のテーブルを構築してもよい。格納されたこれらのインスタンス化／削除時間に基づいて、ＡＰＩレジストリ４０４は、サービス２００８のインスタンスをインスタンス化または削除するようコンテナプラットフォーム２１０に命令することができる。ＡＰＩレジストリ４０４は、これらの所定期間中にインスタンス化すべきインスタンスの数を指定することもできる。たとえば、プロパティ２００４は、午後５時から午後１０時まで、サービス２００８の少なくとも１０のインスタンスがコンテナプラットフォーム２１０上でアクティブであることを明示してもよい。この期間になると、ＡＰＩレジストリ４０４は、追加のインスタンスを作成するようコン

テナプラットフォーム２１０に命令することができる。

【００７５】

図２１は、いくつかの実施形態に係る、サービス２１０８のレート制限関数を実現するプロパティ２１０４のハードウェア／ソフトウェア図を示す。いくつかのサービスは、要求を受けるレートを制限する必要がある場合がある。その他のサービスは、特定の送信者または特定のタイプのサービスからの要求を制限する必要がない場合がある。本開示よりも前においては、この関数は、サービス自体が、各要求のソースを判断し、このソースをホワイトリスト／ブラックリストと比較し、これらの要求をサービスするレートを絞ることにより、実行しなければならなかった。上記例のうちのほとんどと同様、このオーバーヘッドをサービス自体に課すと、サービスが使用するメモリおよびＣＰＵパワーの量が増し、サービスのスループットが制限される。これらの実施形態は、この技術的課題を、ＡＰＩレジストリが生成したクライアントライブラリにレート制限コードを自動的に生成することによって解決する。これにより、サービス２１０８がその関係するオーバーヘッドすべてを伴ってこの機能を実現しなくても、サービスはプロパティ２１０４によってレート制限を指定することができる。

【００７６】

呼び出し側サービス２１０６が要求をサービス２１０８に送ることを所望するとき、ＡＰＩレジストリ４０４は、レート制限コードを含むクライアントライブラリ２１０２を自動的に生成することができる。クライアントライブラリ２１０２が生成されると、ＡＰＩレジストリ４０４は、特定のサービス２１０６のレート制限が必要か否かを判断することができる。必要でなければ、クライアントライブラリ２１０２を通常通り生成することができる。ＡＰＩレジストリ４０４が、呼び出し側サービス２１０６をレート制限すべきであると判断した場合（たとえばホワイトリスト／ブラックリストとの比較によって）、ＡＰＩレジストリ４０４は、遅延を追加する、カウンターを追加する、および／またはそうでなければレート制限関数を実現するコードをクライアントライブラリ２１０２に挿入することにより、所定のレートに従って任意の所定期間において呼び出し側サービス２１０６が所定の最大数の要求を行うことを保証することができる。このコードは、その間はレート制限関数がアクティブである時間ウィンドウを実現してもよい。これにより、サービス２１０８は、高トラフィック期間中レート制限を自動的に実施することができる。

【００７７】

マイクロサービスアーキテクチャは、より大きなサービスを実現するために使用できる複数のマイクロサービスを含み得る。マイクロサービスアーキテクチャは、大きなタスクを多数のマイクロサービスと呼ばれる細分化された個々のタスクに分割することを中心とするソフトウェア設計技術を使用する。各マイクロサービスの、対応する実現すべきマイクロタスクは単純であり、プロトコルは一般的に軽量である。このアーキテクチャは、大きなアプリケーションを緩く結合された小さなサービスの集まりとして設計したサービス指向アーキテクチャ（service-oriented architecture）（ＳＯＡ）のアーキテクチャスタイルからの進化である。マイクロサービスアーキテクチャは、このコンセプトを、各サービスをさらに、単純で完結しており概ね独立しているタスクのセットに純化することによって改良し、これらのタスクは、単独で、または複雑な動作を実現するために他のマイクロサービスとの組み合わせとして、設計し、開発し、デプロイし、動作させることが可能である。あるアプリケーションをマイクロサービスに分解する利点のうちの１つは、結果として得られる、単純で理解し易いソフトウェア設計のモジュラリティである。このモジュラリティのおかげで、マイクロサービスアプリケーションは、より設計し易く、テストし易く、維持し易く、かつ老朽化により強い。マイクロサービスは、小規模の自律的なチームが並行して開発することができ、独立してデプロイすることができる。また、マイクロサービスは、より大きなアプリケーションまたはサービスフレームワークにおいてその外的振る舞いまたはその動作を変えることなく、継続的にリファクタリングおよび再構築することもできる。

【００７８】

本明細書における定義では、マイクロサービスを、通常のサービスまたはアプリケーションと、当該マイクロサービスによってカプセル化される処理の範囲によって対比させることができる。マイクロサービスは、ＨＴＴＰ等の、技術に依存しないプロトコルを用いてネットワーク上で通信することが多い。マイクロサービスは、独立してデプロイ可能であり、一般的にはその利用できるインターフェイスの外部のその他のマイクロサービスと共有する情報がほとんどない。ＳＯＡアーキテクチャのオーバーヘッドとは対照的に、マイクロサービスは、エンタープライズサービスバス（enterprise service bus）（ＥＳＢ）ではなく単純なメッセージングを用いることができる。マイクロサービスは、コンポーネントを共有するのではなく、依存性を最小にして１単位としてマイクロサービスコードとデータとを結合するコンテキスト境界を通じて共有を最小にすることを目指す。最後に、マイクロサービスの接頭辞である「マイクロ」は、その内部動作の小さな細分性を指している。マイクロサービスは一般的に専用サービスであり、できる限り効率的に１つのタスクを実行する。一般的なサービスは、通常より多くのビジネス機能を含み、完全なサブシステムとして実装されることが多い。

【００７９】

本開示のために、マイクロサービスはまた、コンテナ化された環境に非常に適している。なお、コンテナとマイクロサービスとは同じコンセプトではなく、コンテナは、マイクロサービスを組織し展開するのに良く適応したソフトウェアツールである。コンテナは、マイクロサービスを運用するのに必要なものすべてを提供しつつ、マイクロサービスを同じ環境のコンテナにおける他のソフトウェアから相対的に分離させる、ソフトウェア、ランタイム、システムツール、システムライブラリ、設定その他をすべてパッケージングする。これにより、マイクロサービスは、システムとのやり取りおよびオーバーヘッドではなく、そのタスク実行に集中することができる。加えて、コンテナプラットフォーム環境は特にマイクロサービスに良く適している。たとえば、コンテナプラットフォームは、同時に、同一タイプの複数のコンテナおよび多数の異なるタイプのコンテナを、容易にデプロイし管理することができる。実際のところ、コンテナプラットフォームは、コンテナ化されたマイクロサービスの割当およびデプロイメントを、プラットフォームにおける利用できるハードウェアに合ったものになるように、処理することができる。

【００８０】

図２２は、いくつかの実施形態に係る、複数のマイクロサービスでビルド（build）されたサービスの一例を示す。トップレベルサービス２２１６を用いることにより、標準のビジネスプロセスを実現してもよい。トップレベルサービス２２１６は、ビジネスプロセスにおける単一離散ソフトウェアステップを実現する複数のマイクロサービスで構成されていてもよい。マイクロサービスの各々は、独立型プロセスであり、サービス２２１６は、サービス２２１６を実現するのに必要なすべてのマイクロサービスを含む。この実施形態において、マイクロサービスは全体のサービス２２１６を実行する際の単一ステップを表しておりマイクロサービスはサービス２２１６内に論理的にカプセル化されていることから、サービス２２１６はマイクロサービスのいずれとも区別することができる。

【００８１】

このようなトップレベルサービス２２１６の一例は、顧客オーダーを処理するためのProcessOrder( )サービスを含み得る。サービス２２１６が実現するビジネスプロセスは、在庫調整、インボイス作成、ユーザアカウント課金、船積指図書の作成、台帳のアップデートその他のような、複数の離散ステップを含み得る。これらの離散ステップの各々は、サービス２２１６をビルドするマイクロサービスによって実現されてもよい。これらのマイクロサービスの各々はまた、サービス２２１６によって表されるポッド内のコンテナにデプロイされてもよい。たとえば、これらのマイクロサービスは、マイクロサービス２２０２、２２０４、２２０６、２２０８、２２１０、２２１２、および２２１４によって表すことができる、AdjustInventory( )、CreateShipper( )、Invoice( )、ChargeAccount( )、UpdateLedger( )、EmailReceipt( )、およびCreateUserAccount( )のような関数を含み得る。これらのマイクロサービスは各々、サービス２２１６内で互いにコール

することができる。サービス２２１６は、コンテナプラットフォーム内のノードまたはポッドで表されてもよく、また、プラットフォーム内の他のサービスのＨＴＴＰコールを通じて利用できるエンドポイント２２１８を含み得る。

【００８２】

場合によって、マイクロサービスは、単一のコンテナにカプセル化された単一のプロセスでビルドされてもよい。たとえば、図２２のマイクロサービス２２０２、２２０６、２２０８および２２１２は、単一のコンテナ内の単一のサービスからなる。しかしながら、他のマイクロサービスは、単一のコンテナにカプセル化された関数の集まりを含み得る。たとえば、マイクロサービス２２０４および２２１４は、単一のコンテナに複数の関数を含む。さらに、マイクロサービス２２１０のようないくつかのマイクロサービスは、単一のマイクロサービスにカプセル化された複数のコンテナの複数の関数を含み得る。これらのマイクロサービス設計は各々、本明細書で使用する「マイクロサービス」という用語に含まれる。加えて、各マイクロサービスは、図２２に示されるポッドにカプセル化されてもよい。しかしながら、他の実施形態に従うと、マイクロサービスは、コンテナによって表されポッドまたはコンテナノード内で集められてサービス２２１６を形成してもよい。

【００８３】

図２３は、いくつかの実施形態に係る、サービス２２１６のデプロイメント環境を示す。コンテナプラットフォーム２１０のいくつかの実施形態は、コンテナプラットフォームスケジューラ２３０６を利用することにより、多数のコンテナをコンテナプラットフォーム２１０全体にスケーリングしデプロイしてもよい。サービスがコンテナプラットフォーム２１０内の複数のホストシステムにスケールアウトされる場合、各ホストシステムを管理し基礎となるコンテナプラットフォームの複雑さを取り除く機能が有利になるであろう。「オーケストレーション」は、コンテナスケジューリング、ノードのクラスタの管理、およびコンテナ環境における追加ホストの可能なプロビジョニングを意味する、広い用語である。コンテナプラットフォームにおける「スケジューリング」は、サービスをホストシステムにロードしこの特定のコンテナを如何にして運用するかを定める機能を意味する場合がある。スケジューリングは特にサービス定義をロードするプロセスを意味するが、これは、ノードクラスタの動作のその他の面の管理を担うこともできる。ノードのクラスタを管理することは、コンピューティングホストのグループを制御するプロセスを含む。これはスケジューリングと密接に結びついている。なぜなら、スケジューラ２３０６は、先に定義したようにサービスをスケジューリングするためにはクラスタ内の各ノードにアクセスする必要がある場合があるからである。しかしながら、下記実施形態にとって最も重要なスケジューラ２３０６のプロセスは、ホストの選択を含む。この意味において、スケジューラ２２０２は、複数のマイクロサービスでビルドされた特定のサービスをデプロイし運用するコンテナ環境を自動的に選択するタスクが課されていてもよい。

【００８４】

スケジューラ２３０６に加えて、コンテナプラットフォームは、先に詳述したＡＰＩレジストリ４０４を含み得る。ＡＰＩレジストリ４０４は、ＩＤＥの開発環境からサービス２２１６を受け、スケジューラ２３０６と協働することにより、サービス２２１６をコンテナプラットフォーム２１０にデプロイしてもよい。先に述べたように、サービス２１６は、アクセシビリティ、セキュリティ、暗号化、認証その他のような、ＡＰＩレジストリ４０４に与えられた１つ以上のプロパティに基づいてデプロイすることができる。これらのプロパティは、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に、呼び出し側サービスまたはサービス２２１６自体のいずれかに追加される追加コードを決定することができる。しかしながら、上記プロパティは、一般的に、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にサービス２２１６の場所を考慮する必要はない。

【００８５】

本開示よりも前において、サービスのデプロイメントに基づく技術には技術的課題があった。サービスは独立型機能ブロックになることを意図しているので、サービスの内的作業はまとめて１つのコンピューティング環境内のポッド／ノードにデプロイされていた。

たとえば、サービス２２１６がコンテナプラットフォーム２１０にデプロイされるとき、マイクロサービス２２０２、２２０４、２２０６、２２０８、２２１０、２２１２、および２２１４はすべてまとめて第１の環境２３０２にデプロイされる。マイクロサービス２２０２、２２０４、２２０６、２２０８、２２１０、２２１２、および２２１４は論理および機能ブロックとしてともに動作することが想定されているので、サービス２２１６は常に、その内部コンポーネントのうちのいずれも他の環境に分離することなく、１つの環境にデプロイされる。

【００８６】

本明細書で使用する「環境」という用語はコンピューティング環境を含み得る。２つの異なる環境が、機能的に考慮され、物理的に互いに分離されている。一例として、第１の環境は、コンテナ化されたサービスおよびマイクロサービスを受けることができるクラウドプラットフォームを含み得る。コンテナプラットフォームにおける第２の環境２３０４は、第１の環境２３０２のクラウドプラットフォームからは物理的にも機能的にも分離されているオンプレミス環境を含み得る。第１の環境２３０２および第２の環境２３０４は、１マイルを超える距離、物理的に離れていてもよく、別々の異なるプロセッサ、メモリデバイス、ネットワークインターフェイスやその他のコンピュータハードウェアを用いて動作してもよい。第１の環境２３０２および第２の環境２３０４は、異なるオペレーティングシステムを用いて、または同一タイプのオペレーティングシステムの異なるインスタンスを用いて動作してもよい。加えて、第１の環境２３０２および第２の環境２３０４は、下記実施形態が考慮する異なる特徴を有していてもよい。具体的には、これらの環境２３０２、２３０４は、コスト、可用性、セキュリティ、レイテンシ、および、コンピューティング環境に対応付けられたその他の一般的な特徴に対応付けられた、異なる動作特徴を有していてもよい。

【００８７】

サービス２２１６は第１の環境２３０２のような単一の環境にデプロイされるので、本開示より前に機能していたシステムでは、サービス２２１６のデプロイメントを最適化して第２の環境２３０４を含む別々の物理的環境にデプロイすることはできなかった。これらの異なる環境２３０２、２３０４は異なる特徴を有するので、マイクロサービス２２０２、２２０４、２２０６、２２０８、２２１０、２２１２、および２２１４のうちのいくつかが、これらのマイクロサービスのうちのいくつかを異なる環境でより効率的に機能させる特徴を有していてもよい。しかしながら、サービスは別々の環境に分離されることはなかった。さらに、サービス２２１６の総合的なパフォーマンスを最適化するためにどのマイクロサービスをどの環境に配置すべきかを判断するのに利用できる方法もプロセスもなかった。最後に、サービス自体２２１６を、コスト、可用性、レイテンシ、セキュリティ、場所その他のようなデプロイメント基準に対応付けて、デプロイメント中に最適化されるべきサービス２２１６の所望の特徴を指し示すことができる。しかしながら、コンテナプラットフォームは、デプロイメント属性に基づいてプロセスを認識しサービスデプロイメントを最適化することはできなかった。これは、レイテンシの増加、非効率的なメモリ使用、および第１の環境２３０２のようなホスト環境の処理要件の増加を含む、非効率的なサービス動作をもたらしていた。

【００８８】

本明細書に記載の実施形態は、デプロイメントによって最適化すべきサービス２２１６の動作側面を指定する、サービス２２１６のデプロイメントに対応付けられた１つ以上の基準を受けることにより、上記およびその他の技術的課題を解決し、クラウドプラットフォーム２１０の全体的な機能を改善する。スケジューラ２３０６および／またはＡＰＩレジストリ４０４は、複数の環境の各々に対応付けられた属性にアクセスし、サービス２２１６のデプロイメント基準を最適に満たすためには、どのようにしてサービス２２１６を構成するマイクロサービスを、利用できる環境のいずれかにデプロイするのが最適化かを、判断することができる。これは、サービス２２１６自体の動作を最適化するだけでなく、コンテナプラットフォームリソースの使用も最適化する。たとえば、レイテンシを最適

化するためのデプロイメント基準は、環境と環境との間の環境間コールの数を低減し、サービス自体のレイテンシを最小にする。セキュリティを最適化するためのデプロイメント基準は、マイクロサービスに、最も安全な環境に対する最も厳しいセキュリティ要件を課すが、それほど厳しくないセキュリティ要件を有するマイクロサービスをより効率的なサーバに配置することにより、それを必要としないマイクロサービスのセキュリティ関数に対応付けられた処理およびリソースのオーバーヘッドを減じる。

【００８９】

図２４は、いくつかの実施形態に係る、複数のコンピューティング環境へのサービスデプロイメントを最適化する方法のフローチャートを示す。この方法は、コンテナ化された複数のマイクロサービスを含むサービスを受けることを含み得る（２４０１）。マイクロサービスのサービスは、図２２～図２３について先に述べたように配置されてもよい。これらのサービスは、スケジューラ、オーケストレーションプラットフォーム、および／またはＡＰＩレジストリがデプロイメントのために受けてもよい。この方法はまた、複数の環境にサービスをデプロイするためのデプロイメント基準を受けることを含み得る（２４０３）。先に述べたように、複数の環境は、物理的に分離された環境であってもよく、異なる動作特徴を有していてもよい。したがって、この方法はまた、複数の環境の特徴にアクセスすることを含み得る（２４０５）。この方法はまた、複数の環境の特徴とサービスのデプロイメント基準とに基づいて、複数のマイクロサービスを複数の環境にデプロイすることを含み得る（２４０７）。複数の環境に複数のマイクロサービスを如何にしてデプロイできるかについての例を、以下でより詳細に説明する。

【００９０】

図２５は、いくつかの実施形態に係る、サービス基準および環境／マイクロサービス属性の図を示す。サービス２２１６は、先に述べたように複数のマイクロサービスを含み得る。加えて、サービス２２１６は、１つ以上のデプロイメント基準２５０２に対応付けられていてもよい。いくつかの実施形態において、デプロイメント基準２５０２は選択肢のリストを含み得る。これは、コンテナプラットフォームにおける各種デプロイメント戦略を通して最適化することができる、サービス２２１６を動作させる際の異なる側面を含む。たとえば、図２５において、基準２５０２は、サービス２２１６がデプロイされる前に開発者が選択できる選択肢のチェックボックスのリストを含む。次に、スケジューラおよび／またはＡＰＩレジストリは、選択された基準２５０２に従ってサービス２２１６のデプロイメントを実行することができる、格納されている予め定められたコードシーケンスにアクセスすることができる。いくつかの実施形態において、基準２５０２から複数の選択を行うことができ、全体的な最適化プロセスによりデプロイメントを最適化して、選択された１つ以上の基準２５０２のバランスを取ることができる。

【００９１】

サービス２２１６の基準全体２５０２に加えて、サービス２２１６の各マイクロサービスは、サービス２２１６に対して指定された基準２５０２各々に対応付けられた値を提供する１つ以上の属性２５１０を有し得る。これらの属性２５１０各々に対応付けられた値は、対応するマイクロサービスを基準２５０２に従って如何にして評価すべきかを特徴付けてもよい。たとえば、サービス２２１６のデプロイメントはセキュリティ問題について最適化されねばならないと指定する基準の場合、マイクロサービスのセキュリティ属性は、サービス２２１６の他のマイクロサービスとの比較において、このマイクロサービスの相対セキュリティ要件を特徴付ける値を含み得る。別の例において、サービス２２１６のデプロイメントはレイテンシについて最適化されねばならないと指定する基準の場合、マイクロサービスのレイテンシ属性は、サービス２２１６の他のマイクロサービスとの関係において、対応するマイクロサービスの実行レイテンシを特徴付けてもよい。別の例において、サービス２２１６のデプロイメントは同じコンピューティング環境に対するコロケーション（co-location）について最適化されねばならないと指定する基準の場合、マイクロサービスのコロケーション属性の値は、コールするまたはこのマイクロサービスからコールされる他のサービスの数または識別のような、他のマイクロサービスとのサービス

間接続を特徴付けていてもよい。別の例において、サービス２２１６のデプロイメントは可用性について最適化されねばならないと指定する基準の場合、マイクロサービスの可用性属性の値は、マイクロサービスが如何にしてバックエンド処理に対するフロンドエンド要求に露出されるかを特徴付けることができる。これらの例に加えて、如何にしてサービスのデプロイメントを最適化できるかを指定するその他任意の基準が、サービスの各マイクロサービスの属性に対応していてもよく、この属性には、このマイクロサービスを各種環境に配置し基準に従ってデプロイメントを最適化できる方法を特徴付ける値が対応付けられている。

【００９２】

先に述べたように、コンテナプラットフォームは、第１の環境２３０２および第２の環境２３０４のような複数の環境を含み得る。なお、これら２つの環境の使用は、一例として提供しているだけであって、限定を意図したものではない。他の実施形態は、クラウドデプロイメントおよびオンプレミスデプロイメントを超えて拡張されるハードウェアプラットフォームとともに、任意の数の任意のタイプのコンピューティング環境を含み得る。環境の数に関わらず、各環境は、デプロイメントを最適化するために対応する基準に対して環境がどのように関連しているかを特徴付ける値を伴う一組の属性２５０４、２５０６を含み得る。たとえば、コストを最小にする基準の場合、第１の環境２３０２の対応するコスト属性は、マイクロサービスを他の環境にデプロイすることを基準として、マイクロサービスを第１の環境２３０２にデプロイするコストを、特徴付けることができる。別の例において、サービスの可用性を最大にすることを指定する基準の場合、第１の環境２３０２の対応する可用性属性は、コンテナプラットフォームにおける他の環境を基準として第１の環境２３０２の可用性を特徴付けることができる。その他同様の例は、環境に対して動作するマイクロサービスのレイテンシを特徴付ける属性、環境のセキュリティレベル、およびコンピューティング環境のその他任意の動作特徴を含み得る。

【００９３】

ＡＰＩレジストリ４０４および／またはスケジューラ２３０６は、サービス２２１６をデプロイするための基準２５０２、サービス２２１６の各マイクロサービスの属性２５１０、および／またはコンテナプラットフォームにおける動作環境２３０２、２３０４その他の各々の属性を受けてもよい。スケジューラ２３０４および／またはＡＰＩレジストリ４０４は次に、基準２５０２が指定する属性をマッチングさせることにより、マイクロサービスをコンピューティング環境とマッチングさせることができる。たとえば、セキュリティの最適化を指定する基準の場合、セキュリティ要求値が高いマイクロサービスは、セキュリティ評価値が高い環境とマッチングさせればよく、セキュリティ要求値が低いマイクロサービスは、セキュリティ評価値が低いコンピューティング環境とマッチングさせればよい。デプロイ中、最適化の複数の基準２５０２が指定されている場合、ＡＰＩレジストリ４０４および／またはコンテナプラットフォーム２３０６は、基準２５０２各々をランク付けし、値の重み付けした組み合わせを用いることにより、デプロイする各マイクロサービスの場所を表す変数で、コスト式を最小にすることができる。コスト式を最小にすることにより、マイクロサービスごとに選択されたコンピューティング環境を指定する、サービス２２１６の各マイクロサービスに対し、最適化されたデプロイメントマップをもたらすことができる。このデプロイメント最適化を実際如何にして計算できるかについての具体例を以下で詳細に説明する。

【００９４】

図２６は、いくつかの実施形態に係る、動作コストを最小にするように最適化されたサービスデプロイメントの一例を示す。スケジューラ２３０６および／またはＡＰＩレジストリがサービス２２１６をデプロイするとき、コスト基準を選択することができ、マイクロサービス各々の属性を分析することができる。次に、コンピューティング環境２３０２、２３０４各々の属性も分析することができ、それに応じてコンピューティング環境２３０２、２３０４に対するマイクロサービスのデプロイメントを最適化することができる。たとえば、マイクロサービスの属性を解析するときに、マイクロサービス２２１４および

マイクロサービス２２１２の属性の値は低コスト値を示し得ると判断する場合がある。これは、これらのマイクロサービス２２１４、２２１２のメモリおよび／または処理フットプリントが小さく第１の環境２３０２のクラウドプラットフォームにデプロイするときのコストが低い可能性があることを、示し得る。なお、他の例は、受けた要求の数、帯域幅要件などのような他のメトリクスを用いてコストを判断してもよい。これらのサービスは、クラウド環境にデプロイするためのコストが相対的に低いので、ＡＰＩレジストリ４０４および／またはスケジューラ２３０６は、マイクロサービス２２１４、２２１２に対して第１の環境２３０２を選択することができる。

【００９５】

対照的に、マイクロサービス２２１０、２２０６、２２０２、２２０８、および２２０４のコスト属性の値は、クラウドプラットフォームに対するデプロイメントのコストの値が相対的に高いことを示し得る。したがって、ＡＰＩレジストリ４０４および／またはスケジューラ２３０６は、これらの高コスト値を、オンプレミスコンピューティングプラットフォームに対応する第２の環境２３０４の属性とマッチングさせることができる。第２の環境２３０４は、そのコスト属性の値が高く、高コストのマイクロサービスをこの環境にデプロイすべきであることを示し得る。

【００９６】

指定されたデプロイメント基準に対応する属性値に基づいてマイクロサービスをコンピューティング環境とマッチングさせた後に、ＡＰＩレジストリ４０４および／またはスケジューラ２３０６は、個々のマイクロサービスを選択されたコンピューティング環境にデプロイしサービス全体にアクセスするためのエンドポイント２２１８を提供することができる。先に述べたように、ＡＰＩレジストリ４０４は、サービス全体に対する安定したエンドポイントを、ＡＰＩレジストリ４０４にアップロードされたＡＰＩ定義ファイルが規定する１つ以上のＡＰＩ関数のセットにこのエンドポイントをバインディングすることにより、提供することができる。ＡＰＩレジストリ４０４がない実施形態では、スケジューラ２３０６が同様のやり方でマイクロサービスをデプロイしＨＴＴＰサービスコールが利用できるようにエンドポイント２２１８を提供することができる。

【００９７】

図２７は、いくつかの実施形態に係る、サービス可用性について最適化されたサービスデプロイメントの一例を示す。図２６に関する例において先に述べたように、ＡＰＩレジストリ４０４および／またはスケジューラ２３０６は、サービスの最大可用性のためにデプロイメントが最適化されねばならないことを指定する、サービスのデプロイメントについての選択された１つ以上の基準を受けることができる。ＡＰＩレジストリ４０４および／またはスケジューラ２３０６は次に、コンピューティング環境２３０２、２３０４における属性の値と、各マイクロサービスの可用性属性とをマッチングさせることができる。この例において、マイクロサービス２２１４は、可用性の値が相対的に高いフロントエンドインターフェイスを実現し得る。これは第２のコンピューティング環境２３０４のオンプレミスシステムとマッチングさせることができる。第２のコンピューティング環境も、第２の環境２３０４の可用性が高いことを示す高可用性値を有し得る。

【００９８】

対照的に、他のマイクロサービス２２０２、２２０４、２２０６、２２０８、２２１０、および２２１２は、マイクロサービス２２１４と比較して相対的に低い属性値を有し得る。これは、これらのマイクロサービスにとって高可用性はサービス全体の可用性を高くするのに不可欠ではないことを示している。この属性値は、第２の環境２３０４のような高い可用性評価を有していないかもしれないクラウドプラットフォームの第２のコンピューティング環境２３０２とマッチングさせることができる。次に、ＡＰＩレジストリ４０４および／またはスケジューラ２３０６は、マイクロサービス各々を、選択された対応するコンピューティング環境にデプロイし、サービス全体に対するエンドポイント２２１８を提供することができる。

【００９９】

図２８は、いくつかの実施形態に係る、セキュリティ問題について最適化されたサービスデプロイメントの一例を示す。この例において、ＡＰＩレジストリ４０４および／またはスケジューラ２３０６は、デプロイメントされるサービス全体を定めるマイクロサービスを、各マイクロサービスのセキュリティ要件に対してコンピューティング環境のセキュリティをマッチングさせるようにデプロイメントが最適化されねばならないことを指定する基準とともに、受けてもよい。このデプロイメントの最適化を実現するために、ＡＰＩレジストリ４０４および／またはスケジューラ２３０６は、各マイクロサービスのセキュリティ属性の値を、対応するコンピューティング環境２３０２、２３０４各々のセキュリティ評価値とマッチングさせてもよい。この例において、クラウドプラットフォームの第１のコンピューティング環境２３０２は、セキュリティ属性について相対的に高い評価値を有し得る。このことは、クラウドプラットフォームは他のコンピューティング環境よりも優れたセキュリティを提供できることを示す。同様に、マイクロサービス２２０２および２２０４も、セキュリティ属性について相対的に高い要求値を有し得る。このことは、これらのマイクロサービスが高レベルのセキュリティ保護を要求する機密データを処理することを示す。ＡＰＩレジストリ４０４および／またはスケジューラ２３０６は、これらの相対的に高いセキュリティ属性値をマッチングし、マイクロサービス２２０２、２２０４のために第１のコンピューティング環境２３０２を選択することができる。

【０１００】

同様に、オンプレミスプラットフォームの第２のコンピューティング環境２３０４は、第１のコンピューティング環境２３０２よりも低いセキュリティ評価を有し得る。よって、マイクロサービス２２０６、２２０８、２２１０、２２１２、および２２１４は、マイクロサービス２２０２、２２０４の要件よりも相対的に低い要件値を有するのであれば、第２のコンピューティング環境２３０４とマッチングさせることができる。これらの判断を行った後に、ＡＰＩレジストリ４０４および／またはスケジューラ２３０６は、マイクロサービス各々を、それぞれの選択されたコンピューティング環境にデプロイし、サービス全体のための安定したエンドポイント２２１８を提供することができる。

【０１０１】

図２９は、いくつかの実施形態に係る、処理速度について最適化されたサービスデプロイメントの一例を示す。速度基準はサービスのレイテンシ基準と同様であってもよい。ＡＰＩレジストリ４０４および／またはスケジューラ２３０６は、サービス全体を既定するマイクロサービスを、デプロイメントが処理速度について最適化されねばならないことを指定する基準とともに、受けることができる。第１のコンピューティング環境２３０２の速度属性は、第１のコンピューティング環境２３０２が、コンテナプラットフォームの他の環境よりも利用できるコンピューティングリソースが少ないことを示し得る。そのため、第１のコンピューティング環境２３０２は、その速度属性の値が相対的に低く効率的に動作するために相対的に多くの量の処理パワーを必要としないこと示す、相対的に低い値を有するマイクロサービス２２０２、２２０４、および２２０８と、マッチングさせることができる。

【０１０２】

加えて、ＡＰＩレジストリ４０４および／またはスケジューラ２３０６は、第２のコンピューティング環境２３０４の速度属性は相対的に高いと判断することができる。このことは、第２のコンピューティング環境２３０４は、コンテナプラットフォームの他の環境よりも、利用できる処理リソースが多いことを示す。そのため、第２のコンピューティング環境２３０４は、相対的に量が多い処理リソースを要することを示す速度属性値を有するマイクロサービス２２０６、２２１０、２２１２、および２２１４とマッチングさせることができる。次に、ＡＰＩレジストリ４０４および／またはスケジューラ２３０６によってマイクロサービス各々をデプロイすることができ、サービス全体に対するエンドポイント２２１８を提供することができる。

【０１０３】

図３０は、いくつかの実施形態に係る、同時に複数の基準に従って最適化されたサービ

スデプロイメントの一例を示す。この例において、サービスに付随する基準は、異なるコンピューティング環境間で行われるコールを最小にしかつ処理速度を最大にするようにデプロイメントが最適化されねばならないことを示し得る。通常、コンピューティング環境間のマイクロサービスコールの最小化は、すべてのマイクロサービスを１つのコンピューティング環境に置くことによって最も上手く実現される。しかしながら、追加基準を指定することにより、デプロイメントは、マイクロサービスの処理要件を、コンピューティング環境の処理能力とマッチングさせてもよい。

【０１０４】

たとえば、上述のように属性値をマッチングさせることにより、ＡＰＩレジストリ４０４および／またはスケジューラ２３０６は、マイクロサービス２２０２、２２０４、および２２０８の処理要件を、第１のコンピューティング環境２３０２の処理能力とマッチングさせることができる。同様に、ＡＰＩレジストリ４０４および／またはスケジューラ２３０６は、マイクロサービス２２０６、２２１０、２２１２、および２２１４の処理要件を、第２のコンピューティング環境２３０４の処理能力とマッチングさせることができる。しかしながら、これらのマイクロサービスを選択されたコンピューティング環境にデプロイする代わりに、グループ化することができるサービスコールパターンを有するマイクロサービスを同じ場所に配置する（コロケーション、co-location)ためにさらなる最適化を行うことができる。たとえば、マイクロサービス２２０２、２２０４、２２０６、および２２０８はすべて複数のサービス間コールを互いにやり取りする。環境を跨るサービスコールの数を最小にするために、マイクロサービス２２０６を第２のコンピューティング環境２３０４から第１のコンピューティング環境２３０２に移すことができる。このタイプのデプロイメントは、計算速度を最大にしつつ環境を跨るサービスコールを最小にするデプロイメント基準のバランスを取る。この例はまた、上記コスト最小化関数においてこの属性に適用する重みを高くすることにより、環境を跨るサービスコールを最小にすることをわずかに優先する。

【０１０５】

上記例において、マイクロサービスデプロイメントの最適化は、複数のコンピューティング環境に対するサービスの最初のデプロイメントの間に実行されている。しかしながら、これと同じ最初のデプロイメントを最適化するプロセスを用いることにより、次のデプロイメントを最適化することができる。たとえば、最初のデプロイメントの後で、サービスの基準は、開発者、アドミニストレータ、または、コンテナプラットフォーム上で作業する自動プロセスによって、変更される場合がある。この基準変更が行われると、上記と同じプロセスをもう一度実行することにより、環境の値を新たな基準選択のためにマイクロサービス属性とマッチングさせることができる。次に、新たな基準に従ってサービスのマイクロサービスを再度デプロイすることができる。

【０１０６】

同様に、コンピューティング環境の属性の値は時間の経過に伴って変化する場合がある。たとえば、第１の環境２３０２は、利用できる処理リソースの量を増加させるハードウェアアップグレードを受ける場合がある。そうすると、第１の環境２３０２において利用できるコンピューティングリソースの量を特徴付ける属性が対応して増加する場合がある。そこで、ＡＰＩレジストリ４０４および／またはスケジューラ２３０６は、変更された属性が、サービスデプロイメントの基準のうちの１つに相当するか否かを判断することができる。属性が、確かにデプロイメント基準に相当する場合、コンピューティング環境の変更された属性値に基づいて、サービスを再度デプロイして最適化することができる。

【０１０７】

コンピューティング環境の属性値の変更に加えて、サービスのマイクロサービスの属性も時間の経過に伴って変化する場合がある。コンテナ化されたマイクロサービスを使用する利点の１つは、これらのマイクロサービスは、独立して、時間の経過に応じてアップグレード、リプレイス、および再デプロイできることである。サービスの１つ以上のマイクロサービスがアップグレードされた場合、これらのマイクロサービスは、アップグレード

／再デプロイメントに従ってマイクロサービスのパフォーマンスまたは要件が変化した場合に対応する属性に割り当てられた新たな値を有し得る。先に述べたように、ＡＰＩレジストリ４０４および／またはスケジューラ２３０６は、変化した値を有する属性がサービス全体に対するデプロイメント基準に対応するか否かを判断することができる。もしそうであれば、ＡＰＩレジストリ４０４および／またはスケジューラ２３０６が、少なくとも影響を受けたマイクロサービスを再度デプロイすることにより、変化した属性値に従ってデプロイメントを最適化することができる。

【０１０８】

上記例ではコンピューティング環境およびマイクロサービス双方の属性値が相対的方法で互いに比較されている。たとえば、第１の環境２３０２が第２の環境２３０４よりもセキュリティ評価値が高い場合、他のマイクロサービスよりもセキュリティ要件が相対的に高いマイクロサービスを第１の環境２３０２に配置すればよい。セキュリティ属性の値で表されるセキュリティ要件を、サービスにおけるマイクロサービス間で相対比較し、要件がより高いマイクロサービスを第１の環境にデプロイすればよい。この相対比較は絶対しきい値を用いることによって行うことができる。たとえば、予め定められたしきい値を用いることにより、任意の属性の値をさまざまなグループに分けることができる。いくつかの実施形態において、マイクロサービスは、コンピューティング環境の数に対応する数のグループにクラスタ化することができる。図３０を一例として用いると、マイクロサービス２２０２、２２０４、２２０６、２２０８を、属性値によって１つのクラスタにすることにより第１のグループを形成し、マイクロサービス２２１０、２２１２、および２２１４を、属性によって１つのクラスタにすることにより、第２のグループを形成することができる。クラスタリングアルゴリズムは、属性値に自然な分離を見出してこれらのグループを形成することができ、グループの数は、利用できるコンピューティング環境の数以下であればよい。次に、これらのクラス他グループの各々を、属性値によってランク付けされたコンピューティング環境とマッチングさせることができる。

【０１０９】

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、これらの入力は各々、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行ってもよい。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。

【０１１０】

図３１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３１００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３１００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３１０２、３１０４、３１０６、および３１０８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３１１０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３１１２が、ネットワーク３１１０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３１０２、３１０４、３１０６、および３１０８に対して通信可能に結合されていてもよい。

【０１１１】

各種実施形態において、サーバ３１１２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３１０２、３１０４、３１０６、および／または３１０８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３１０２、３１０４、３１０６、および／または３１０８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３１１２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。

【０１１２】

図面に示される構成において、システム３１００のソフトウェアコンポーネント３１１８、３１２０および３１２２は、サーバ３１１２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３１００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３１０２、３１０４、３１０６、および／または３１０８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３１００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。

【０１１３】

クライアントコンピューティングデバイス３１０２、３１０４、３１０６、および／または３１０８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。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３１０２、３１０４、３１０６、および３１０８は、ネットワーク３１１０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。

【０１１４】

具体例としての分散型システム３１００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３１１２とやり取りしてもよい。

【０１１５】

分散型システム３１００のネットワーク３１１０は、限定されないがＴＣＰ／ＩＰ（transmission control protocol/Internet protocol）、ＳＮＡ（systems network architecture）、ＩＰＸ（Internet packet exchange）、AppleTalk（登録商標）などを含む、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。一例にすぎないが、ネットワーク３１１０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３１１０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。

【０１１６】

サーバ３１１２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３１１２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３１１２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。

【０１１７】

サーバ３１１２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３１１２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。

【０１１８】

いくつかの実装例において、サーバ３１１２は、クライアントコンピューティングデバイス３１０２、３１０４、３１０６、および３１０８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３１１２は、クライアントコンピューティングデバイス３１０２、３１０４、３１０６、および３１０８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。

【０１１９】

分散型システム３１００は、１つ以上のデータベース３１１４および３１１６も含み得る。データベース３１１４および３１１６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３１１４および３１１６のうちの１つ以上は、サーバ３１１２に対してローカルな場所にある（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３１１４および３１１６は、サーバ３１１２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３１１２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３１１４および３１１６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３１１２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３１１２にローカルにおよび／またはサーバ３１１２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３１１４および３１１６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。

【０１２０】

図３２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３２００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３２００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３２０２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３２０４、３２０６、および３２０８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３２０２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３２０２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。

【０１２１】

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３２０２は示されているも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３２０２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。

【０１２２】

クライアントコンピューティングデバイス３２０４、３２０６、および３２０８は、３１０２、３１０４、３１０６、および３１０８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。

【０１２３】

具体例としてのシステム環境３２００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３２０２とやり取りしてもよい。

【０１２４】

ネットワーク３２１０は、クライアント３２０４、３２０６、および３２０８とクラウドインフラストラクチャシステム３２０２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３１１０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。

【０１２５】

クラウドインフラストラクチャシステム３２０２は、１つ以上のコンピュータ、および／またはサーバ３１１２について先に述べたものを含み得るサーバを含み得る。

【０１２６】

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。

【０１２７】

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。

【０１２８】

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３２０２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。

【０１２９】

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３２０２は、クラウドインフラストラクチャシステム３２０２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３２０２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３２０２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３２０２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３２０２およびクラウドインフラストラクチャシステム３２０２が提供するサービスを関連するコミュニティー内のいくつかの組織が共有するコミュニティー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。

【０１３０】

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３２０２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３２０２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３２０２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。

【０１３１】

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３２０２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。

【０１３２】

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。

【０１３３】

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、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 Service）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。

【０１３４】

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。

【０１３５】

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３２０２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３２３０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３２３０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、およびネットワーキング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。

【０１３６】

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３２０２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３２０２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。

【０１３７】

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３２０２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３２０２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３２３２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。

【０１３８】

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３２０２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３２０２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。

【０１３９】

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３２２０、オーダーオーケストレーションモジュール３２２２、オーダープロビジョニングモジュール３２２４、オーダー管理およびモニタリングモジュール３２２６、ならびにアイデンティティ管理モジュール３２２８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、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。

【０１４０】

具体例としての動作３２３４において、クライアントデバイス３２０４、３２０６または３２０８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３２０２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３２０２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３２０２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　３２１２、クラウドＵＩ　３２１４および／またはクラウドＵＩ　３２１６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３２０２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３２０２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。

【０１４１】

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　３２１２、３２１４および／または３２１６を介して受信される。

【０１４２】

動作３２３６において、オーダーはオーダーデータベース３２１８に格納される。オーダーデータベース３２１８は、クラウドインフラストラクチャシステム３２０２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。

【０１４３】

動作３２３８において、オーダー情報はオーダー管理モジュール３２２０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３２２０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。

【０１４４】

動作３２４０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３２２２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３２２２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調整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３２２２は、リソースのプロビジョニングを調整することにより、オーダープロビジョニングモジュール３２２４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。

【０１４５】

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３２２２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３２４２

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３２２２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３２２４に送る。オーダープロビジョニングモジュール３２２４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３２２４は、クラウドインフラストラクチャシステム３２０２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３２２２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。

【０１４６】

動作３２４４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３２０２のオーダープロビジョニングモジュール３２２４によってクライアントデバイス３２０４、３２０６および／または３２０８の顧客に送信されてもよい。

【０１４７】

動作３２４６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３２２６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３２２６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。

【０１４８】

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３２０２はアイデンティティ管理モジュール３２２８を含み得る。アイデンティティ管理モジュール３２２８は、クラウドインフラストラクチャシステム３２０２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３２２８は、クラウドインフラストラクチャシステム３２０２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３２２８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。

【０１４９】

図３３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３３００を示す。システム３３００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３３００は、バスサブシステム３３０２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３３０４を含む。これらの周辺サブシステムは、処理加速ユニット３３０６と、入出力サブシステム３３０８と、記憶サブシステム３３１８と、通信サブシステム３３２４とを含み得る。記憶サブシステム３３１８は、有形のコンピュータ読取可能記憶媒体３３２２とシステムメモリ３３１０とを含み得る。

【０１５０】

バスサブシステム３３０２は、コンピュータシステム３３００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３３０２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３３０２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、ＩＥＥＥ　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。

【０１５１】

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３３０４は、コンピュータシステム３３００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３３０４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３３０４は、１つ以上の独立した処理ユニット３３３２および／または３３３４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３３０４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ッドコア処理ユニットとして実現されてもよい。

【０１５２】

各種実施形態において、処理ユニット３３０４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、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３３０４および／または記憶サブシステム３３１８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３３０４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３３００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３３０６を含み得る。

【０１５３】

入出力サブシステム３３０８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。

【０１５４】

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。

【０１５５】

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３３００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。

【０１５６】

コンピュータシステム３３００は、現在はシステムメモリ３３１０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３３１８を含み得る。システムメモリ３３１０は、処理ユニット３３０４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。

【０１５７】

コンピュータシステム３３００の構成および種類に応じて、システムメモリ３３１０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３３０４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３３１０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３３００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３３１０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３３１２と、プログラムデータ３３１４と、オペレーティングシステム３３１６とを示している。例として、オペレーティングシステム３３１６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。

【０１５８】

ストレージサブシステム３３１８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３３１８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３３０４によって実行されてもよい。ストレージサブシステムｙ３３１８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。

【０１５９】

ストレージサブシステム３３１８はまた、コンピュータ読取可能記憶媒体３３２２にさ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３３２０を含み得る。システムメモリ３３１０とともに、また、任意でシステムメモリ３３１０と組み合わされて、コンピュータ読取可能記憶媒体３３２２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。

【０１６０】

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３３２２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ティングシステム３３００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。

【０１６１】

例として、コンピュータ読取可能記憶媒体３３２２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３３２２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３３２２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３３００のためのその他のデータの不揮発性ストレージを提供することができる。

【０１６２】

通信サブシステム３３２４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３３２４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３３００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３３２４は、コンピュータシステム３３００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３３２４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３３２４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。

【０１６３】

いくつかの実施形態において、通信サブシステム３３２４は、コンピュータシステム３３００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３３２６、イベントストリーム３３２８、イベントアップデート３３３０などの形態の入力通信を受けることもできる。

【０１６４】

例として、通信サブシステム３３２４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３３２６をリアルタイムで受信するように構成されてもよい。

【０１６５】

加えて、通信サブシステム３３２４は、明確な終わりがない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３３２８および／またはイベントアップデート３３３０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。

【０１６６】

通信サブシステム３３２４はまた、構造化および／または非構造化データフィード３３２６、イベントストリーム３３２８、イベントアップデート３３３０などを、コンピュータシステム３３００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。

【０１６７】

コンピュータシステム３３００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。

【０１６８】

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３３００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。

【０１６９】

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。

【０１７０】

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。

【０１７１】

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、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。

【０１７２】

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。

【０１７３】

「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。

【０１７４】

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。

【０１７５】

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。

【０１７６】

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】

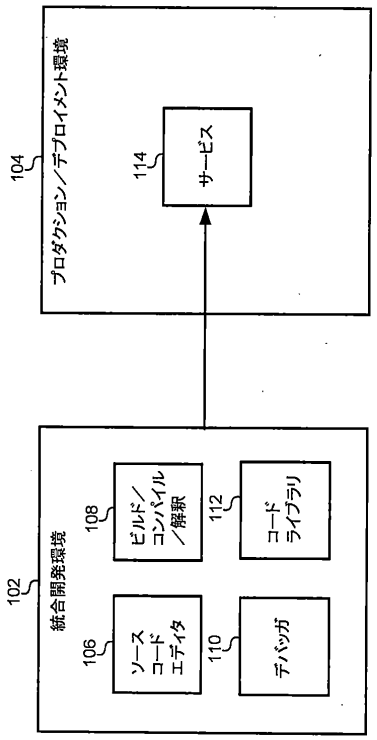

FIG. 1

【図２】

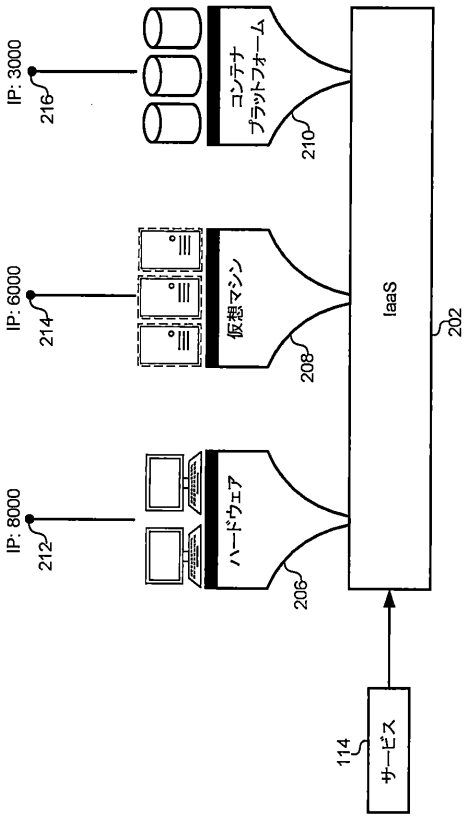

FIG. 2

【図３】

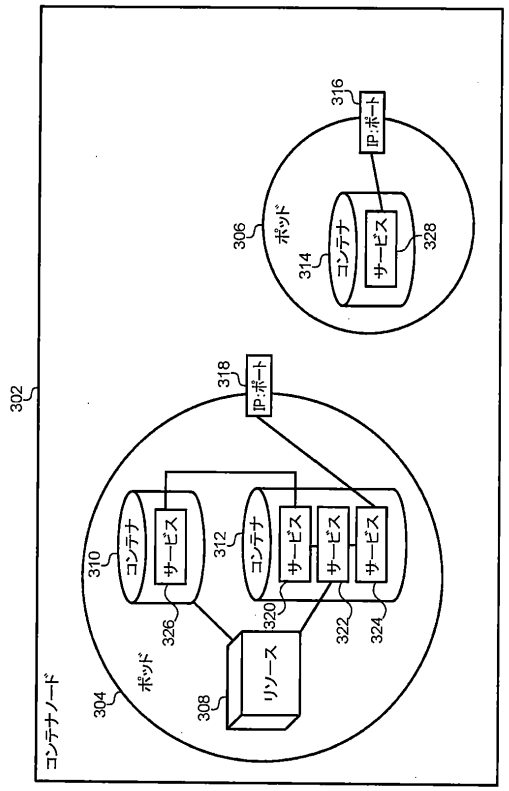

FIG. 3

【図４】

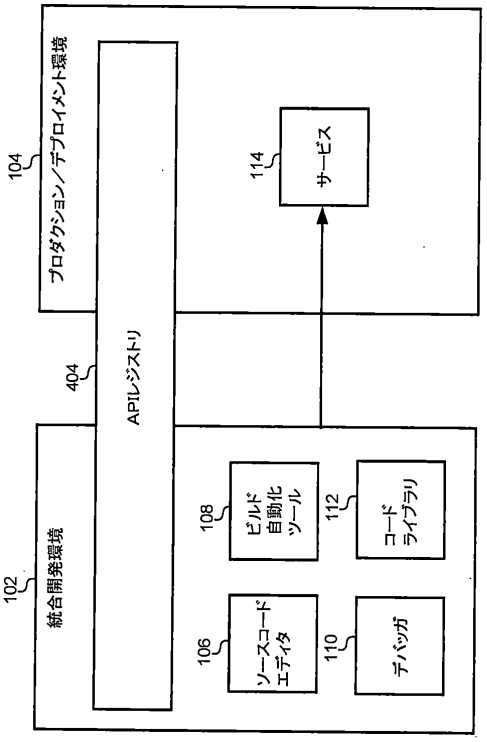

FIG. 4

【図５】

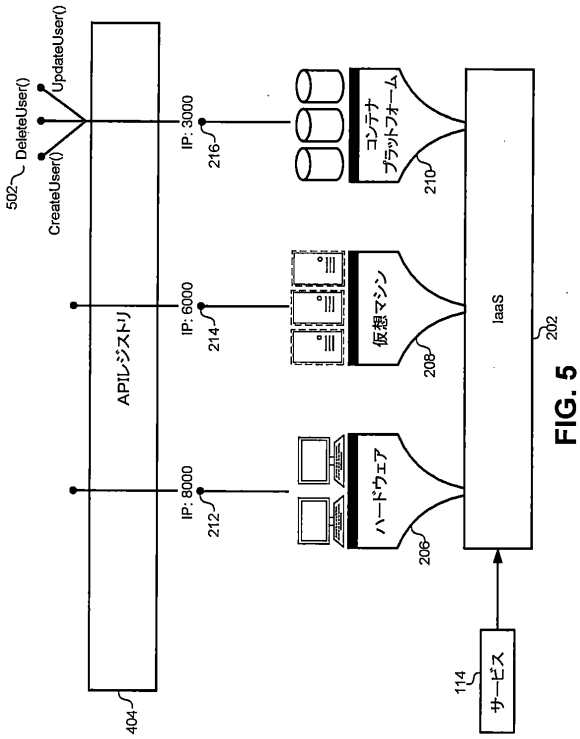

FIG. 5

【図６Ａ】

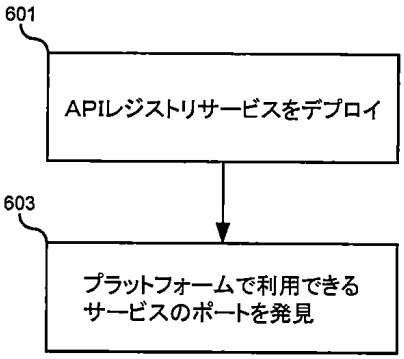

FIG. 6A

【図６Ｂ】

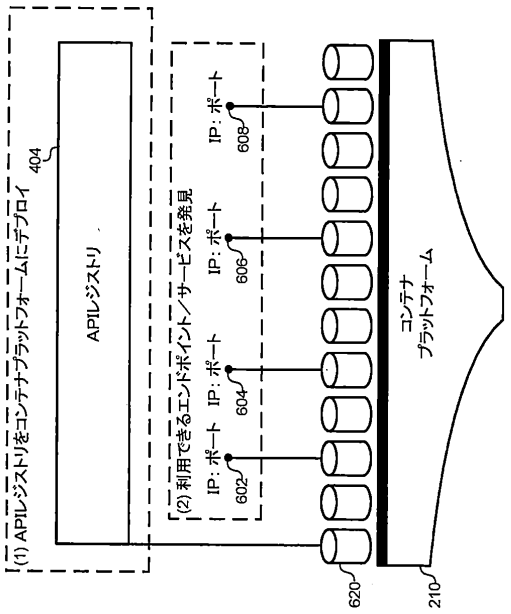

FIG. 6B

【図７Ａ】

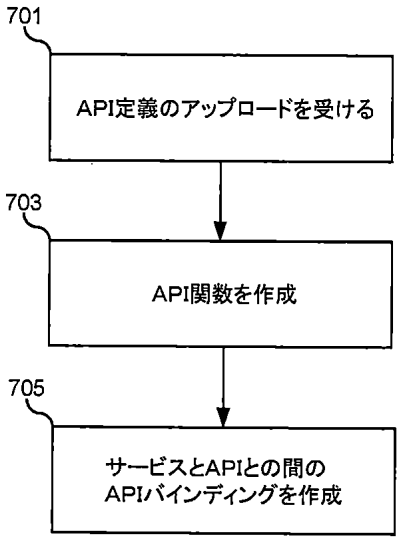

FIG. 7A

【図７Ｂ】

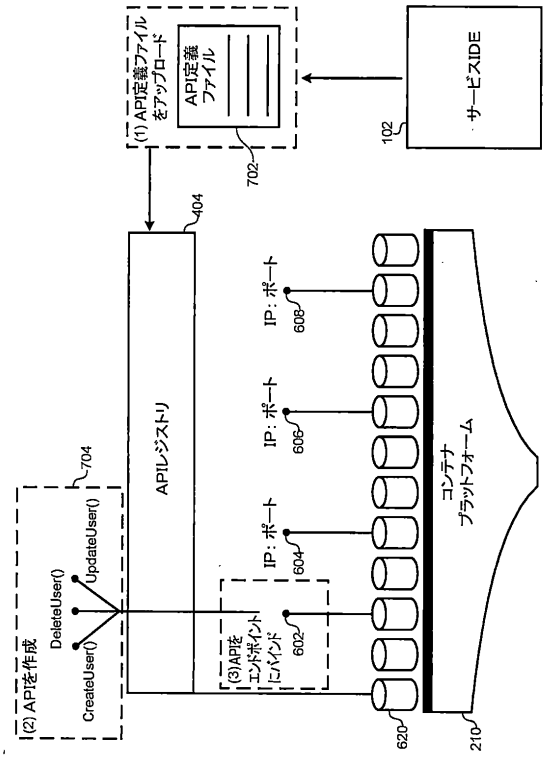

FIG. 7B

【図８】

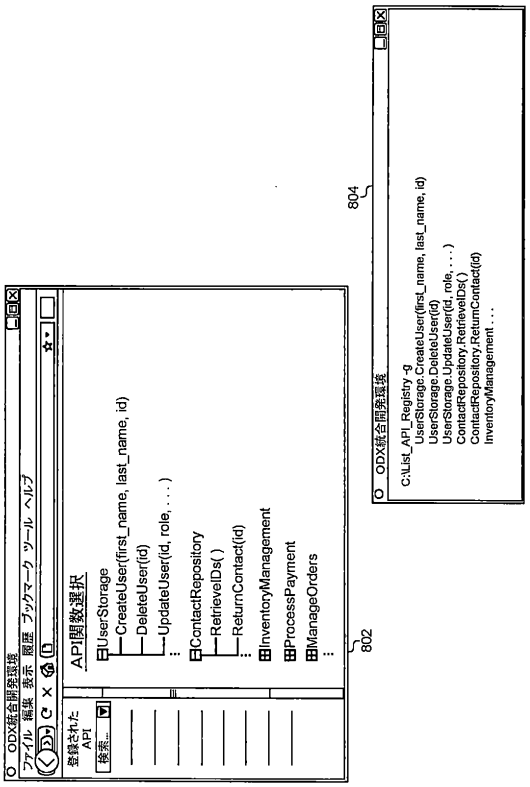

FIG. 8

【図９】

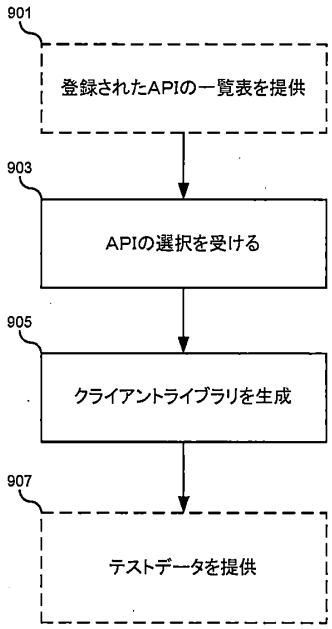

FIG. 9

【図１０】

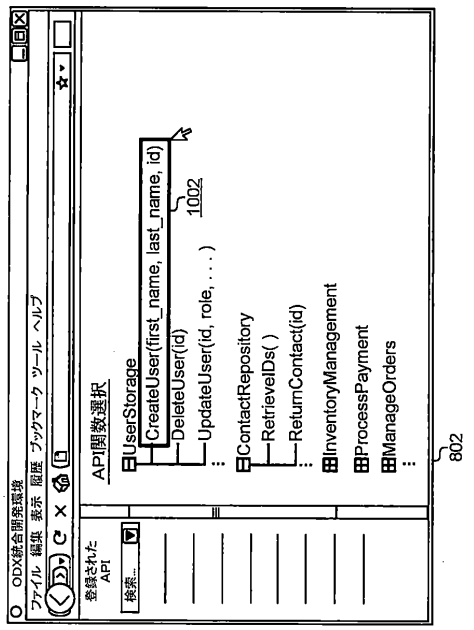

FIG. 10

【図１１】

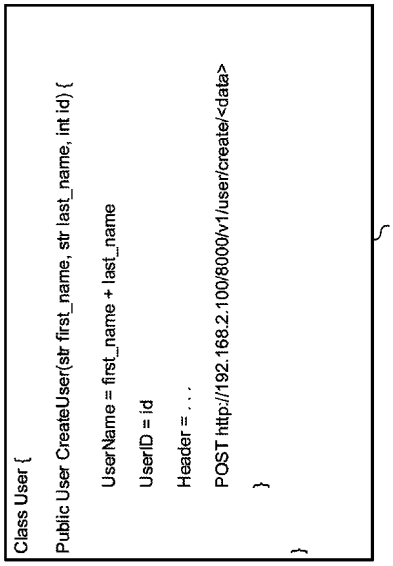

FIG. 11

【図１２】

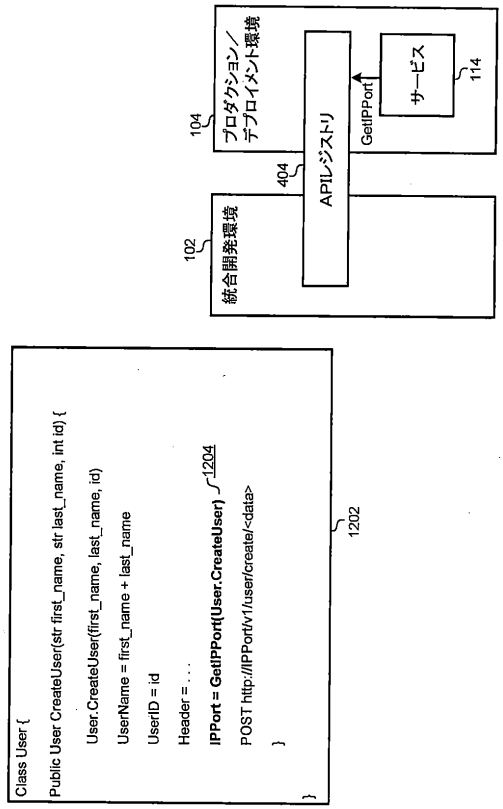

FIG. 12

【図１３】

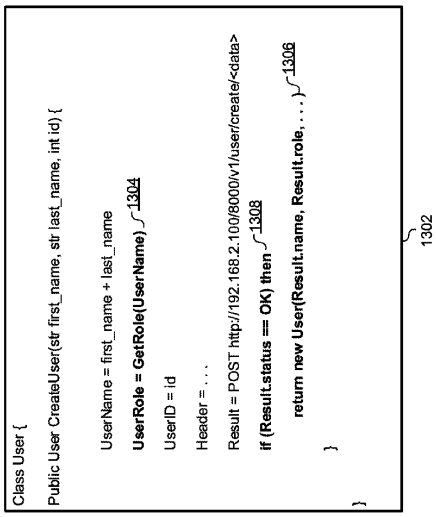

FIG. 13

【図１４】

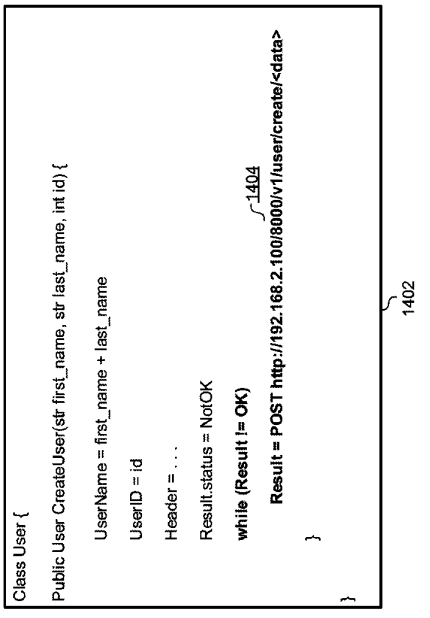

FIG. 14

【図１５Ａ】

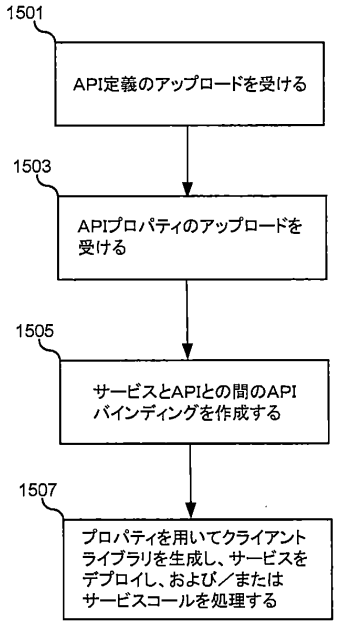

FIG. 15A

【図１５Ｂ】

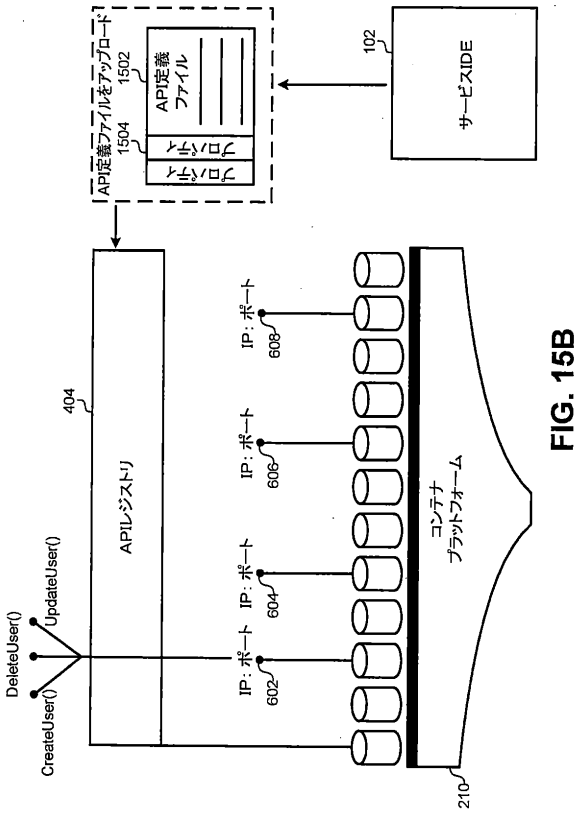

FIG. 15B

【図１６】

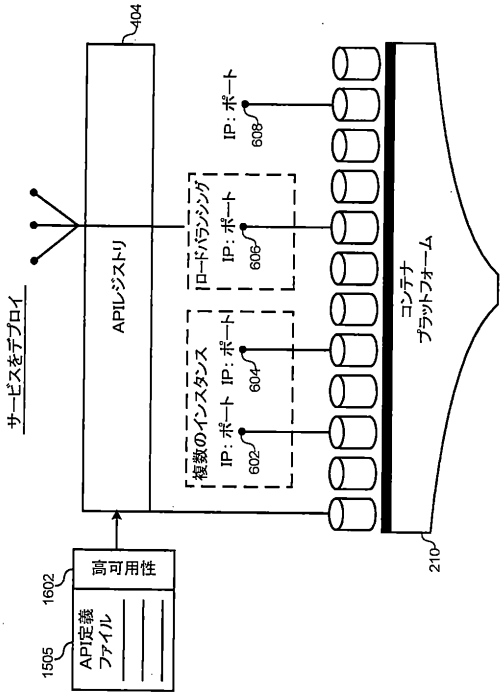

FIG. 16

【図１７】

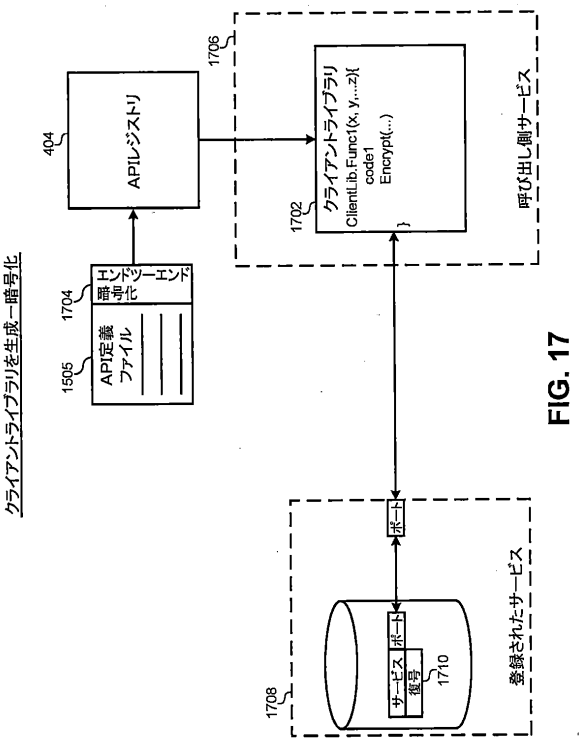

FIG. 17

【図１８】

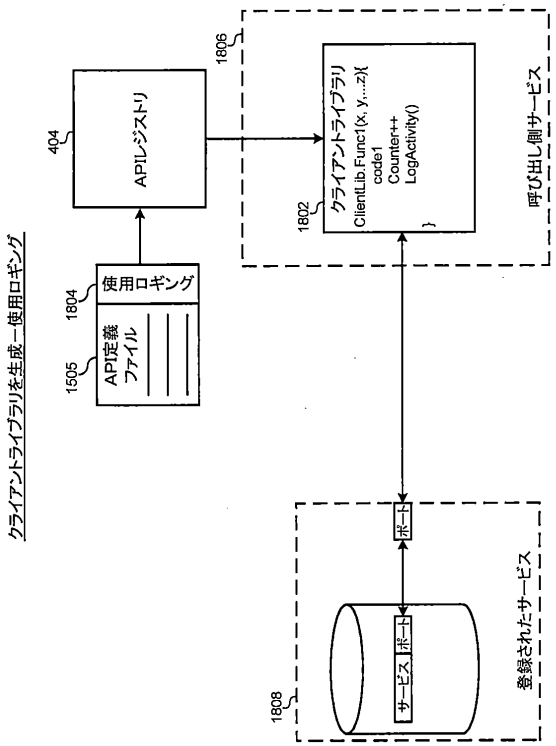

FIG. 18

【図１９】

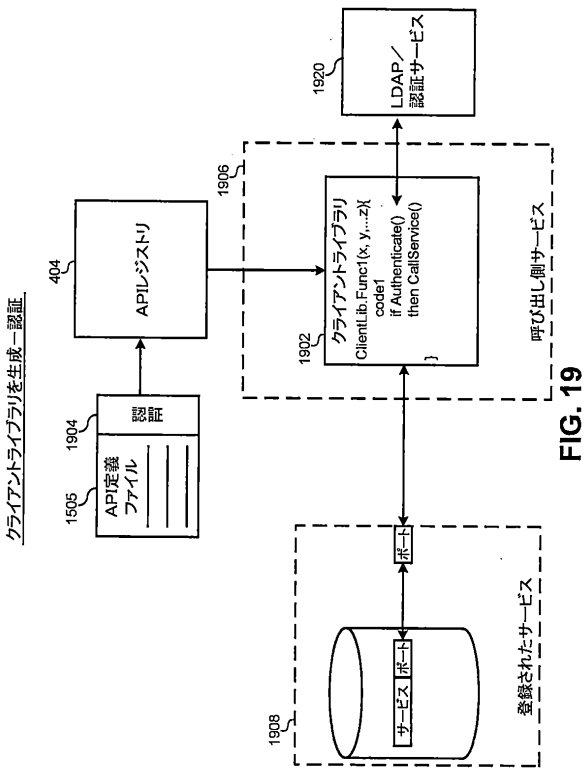

FIG. 19

【図２０】

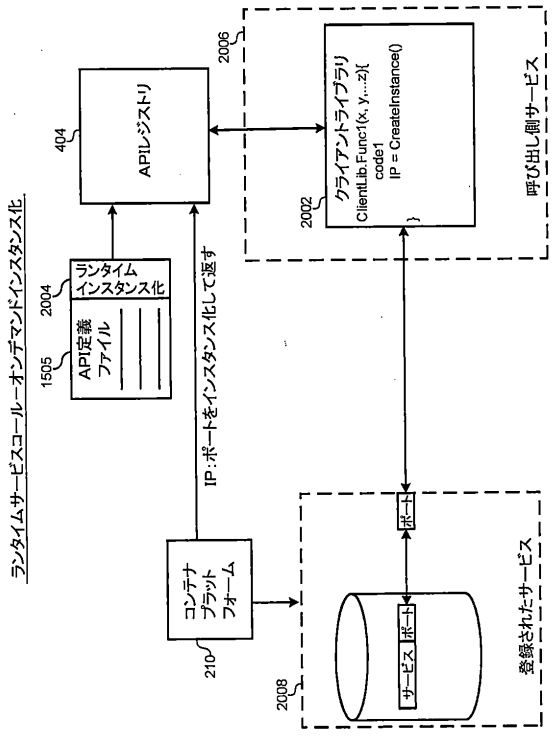

FIG. 20

【図２１】

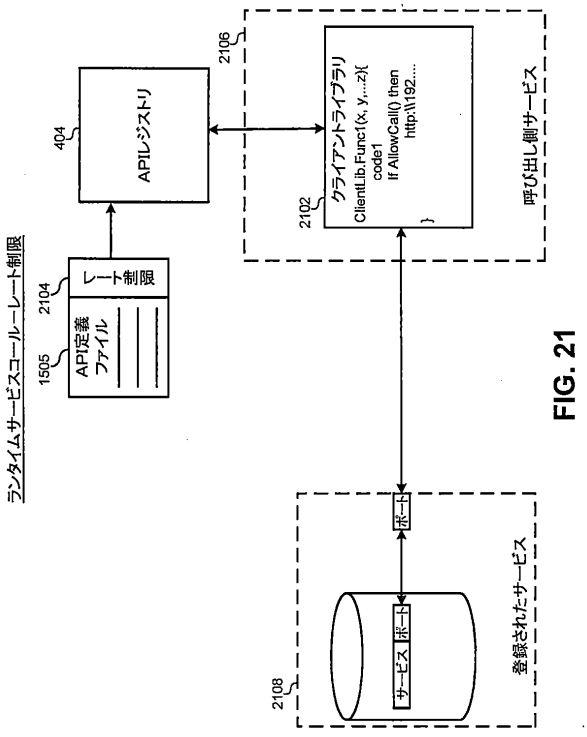

FIG. 21

【図２２】

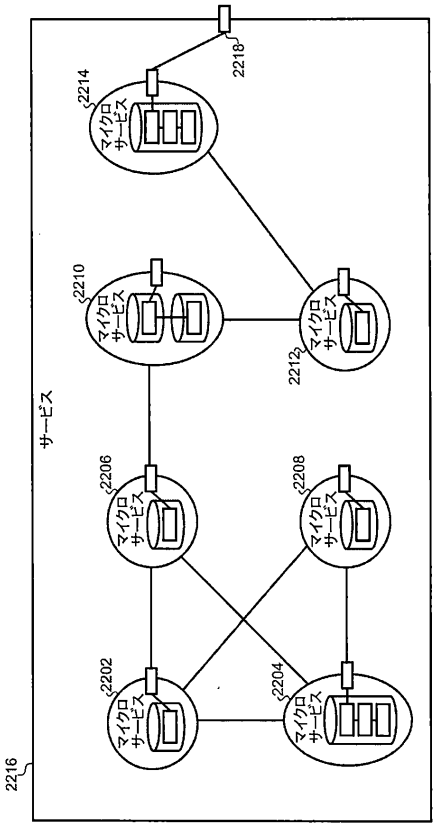

FIG. 22

【図２３】

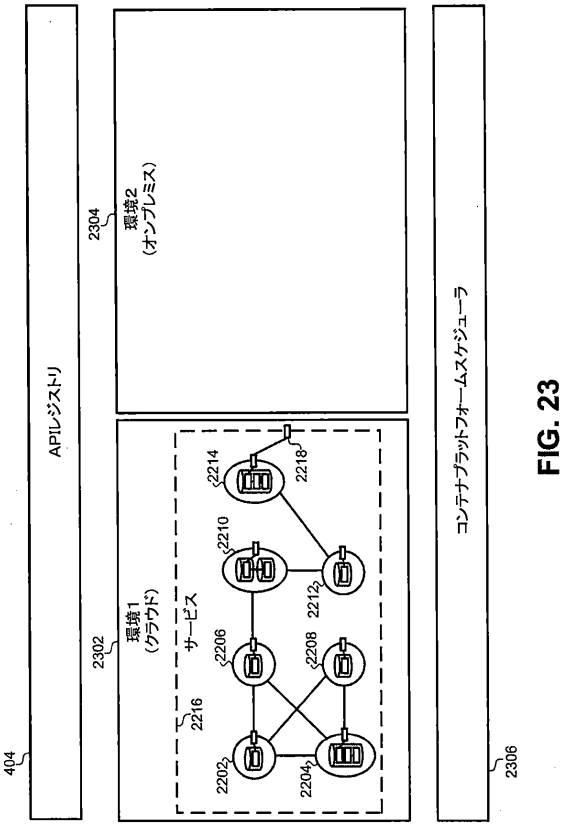

FIG. 23

【図２４】

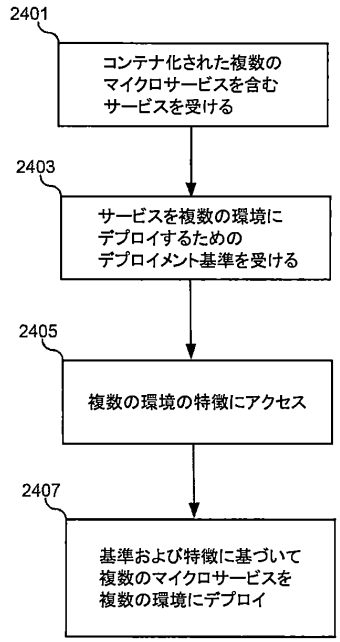

FIG. 24

【図２５】

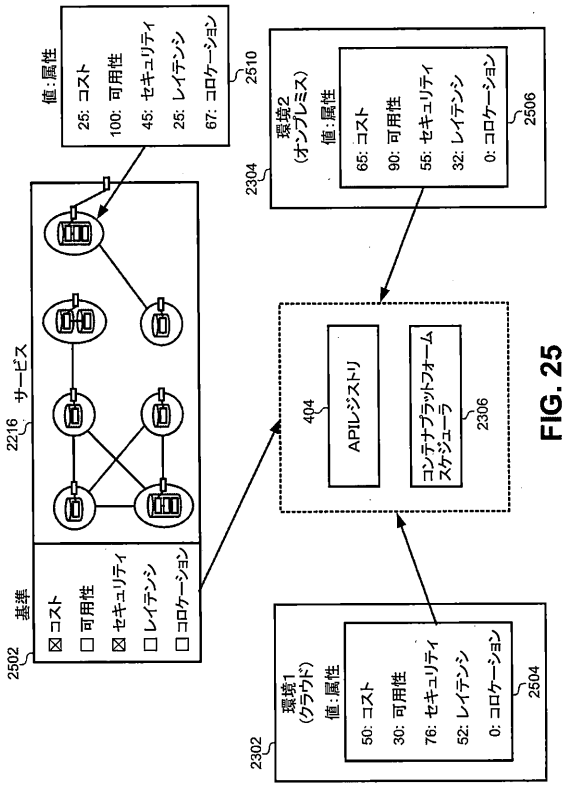

FIG. 25

【図２６】

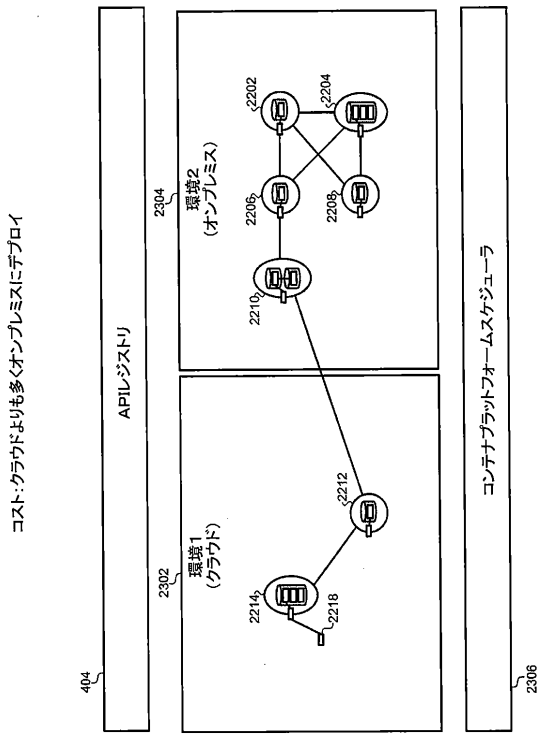

FIG. 26

【図２７】

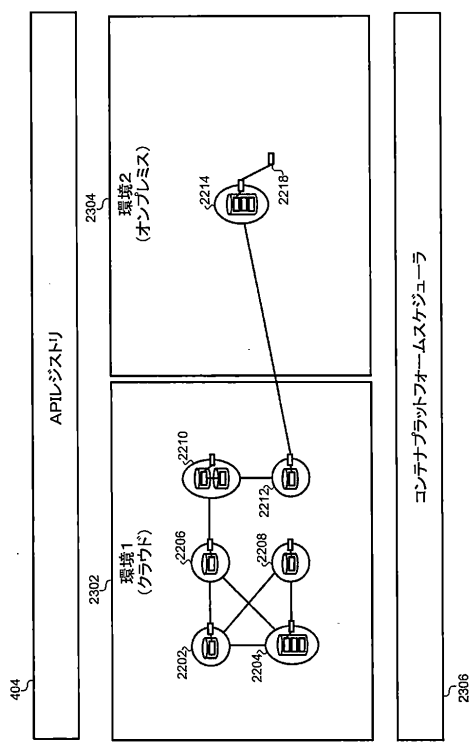

FIG. 27

【図２８】

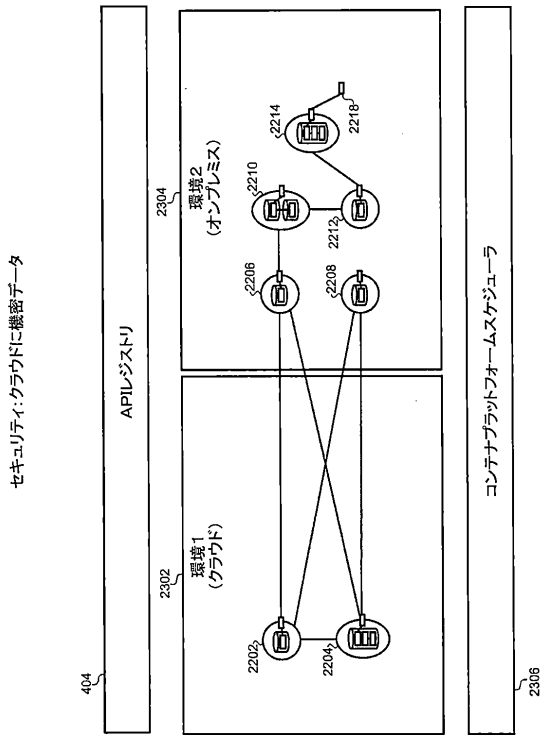

FIG. 28

【図２９】

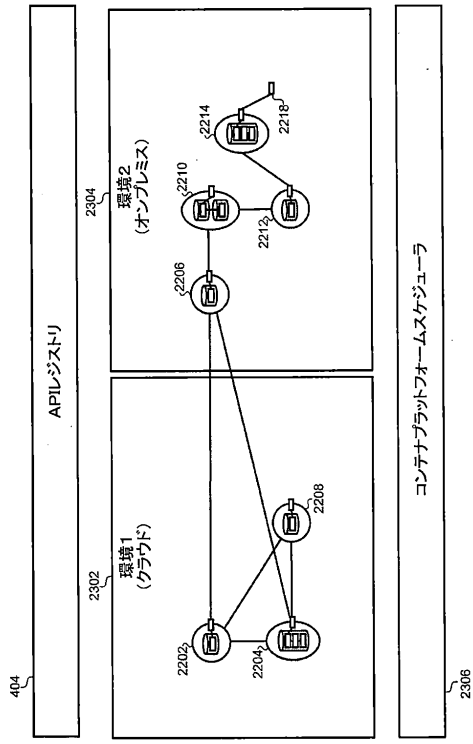

FIG. 29

【図３０】

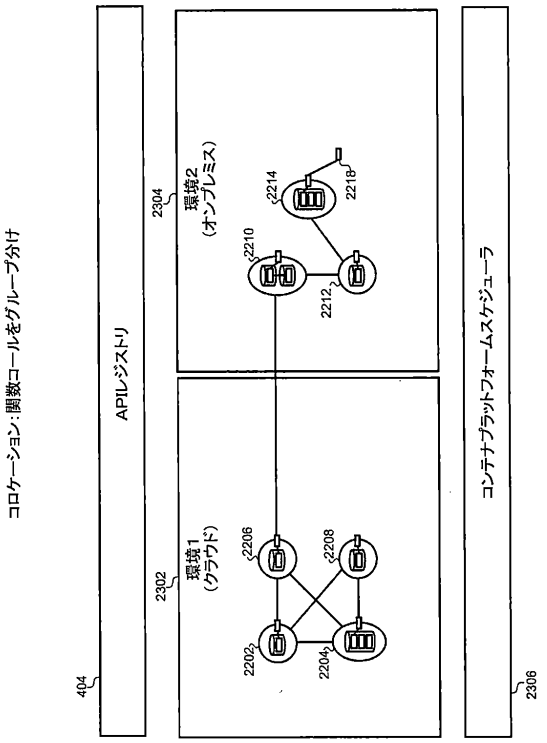

FIG. 30

【図３１】

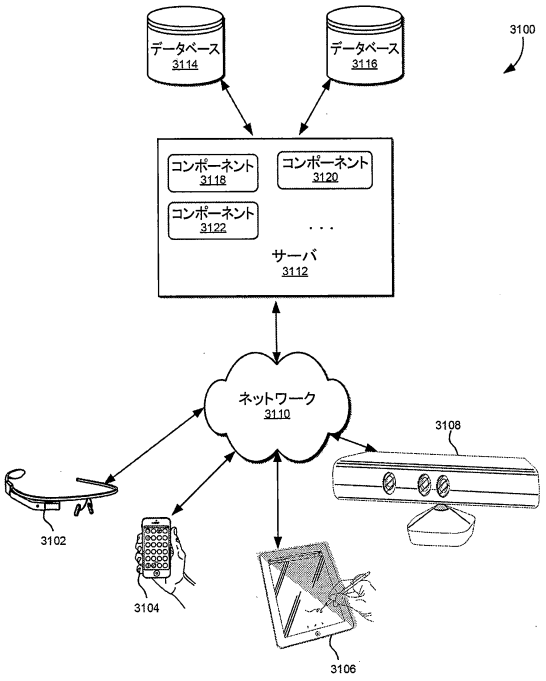

FIG. 31

【図３２】

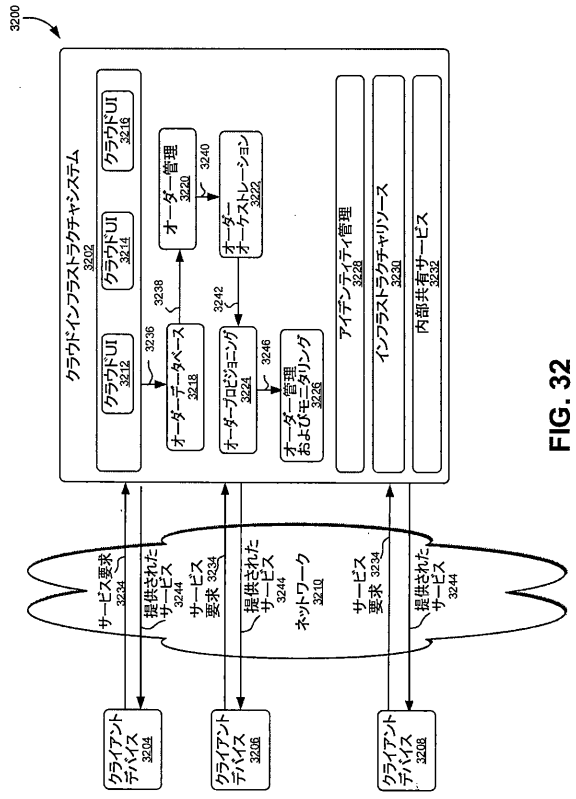

FIG. 32

【図３３】

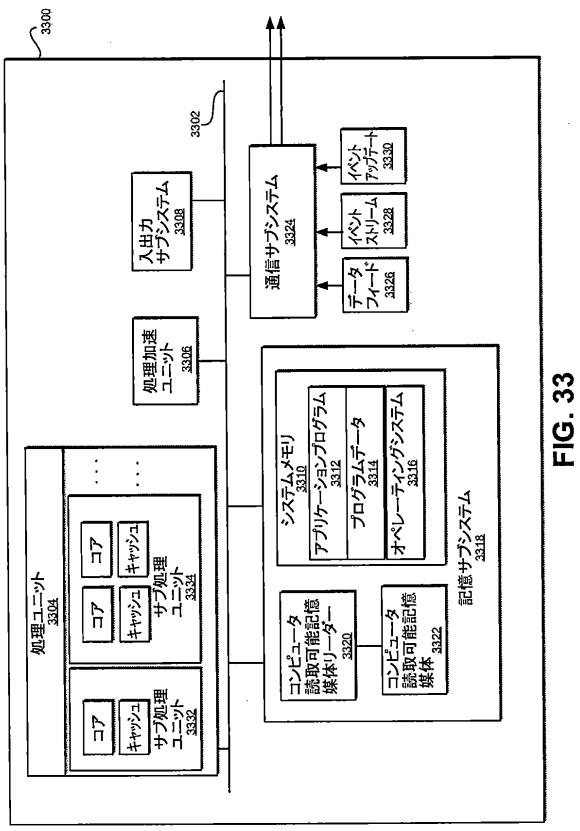

FIG. 33

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、８５３０

審査官　山本 俊介

(56)参考文献　米国特許出願公開第２０１４／０２０１２１８（ＵＳ，Ａ１）
特表２０１４－５３０４１５（ＪＰ，Ａ）
米国特許出願公開第２０１７／００５２７７１（ＵＳ，Ａ１）
阿佐志保ほか，プログラマのためのＧｏｏｇｌｅ Ｃｌｏｕｄ Ｐｌａｔｆｏｒｍ入門，第1版，日本，株式会社翔泳社，2017年06月01日，pp.145-173

(58)調査した分野　(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　８／００－８／７７
Ｇ０６Ｆ　９／００－９／５４