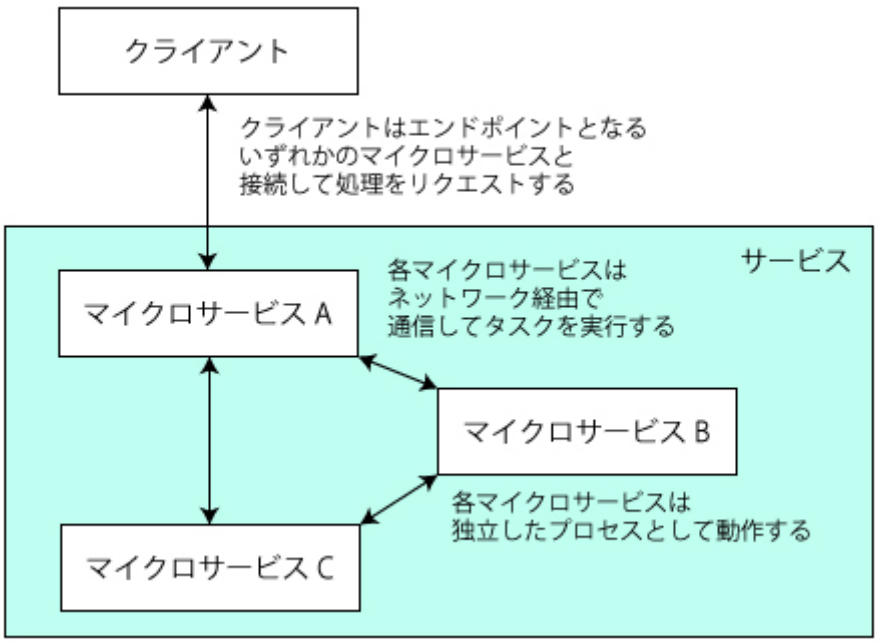
是否将当前网页翻译成中文 网页翻译 中日对照 关闭

マイクロサービスアーキテクチャとそれを支える技術

エンジニア向け 2018.12.06



ツイート シェア 22 225

最近では「マイクロサービス」と呼ばれる、機能毎に細かくサービスを分割して開発や運用を行うアーキテクチャの採用例が増えている。本記事ではこのマイクロサービスアーキテクチャや、それに使われる技術について紹介する。

是否将当前网页翻译成中文 网页翻译 中日对照 关闭

マイクロサービスとは

近年、ITシステムの開発・運用において「Microservice（マイクロサービス）」というアーキテクチャを採用する例が増えている。マイクロサービスアーキテクチャは、簡単に言えばサービスを構成する各要素を「マイクロサービス」と呼ばれる独立した小さなコンポーネントとして実装するという手法で、2011年ごろから提唱されているものだ。

マイクロサービスについては、2014年に公開された「Microservices」という文書が有名だ（有志による日本語訳）。また、さくらのナレッジでも2015年に紹介されている。マイクロサービスの詳しい思想についてはこれら記事を参照してほしいが、簡単にまとめると次のような特徴がある（**図1**）。

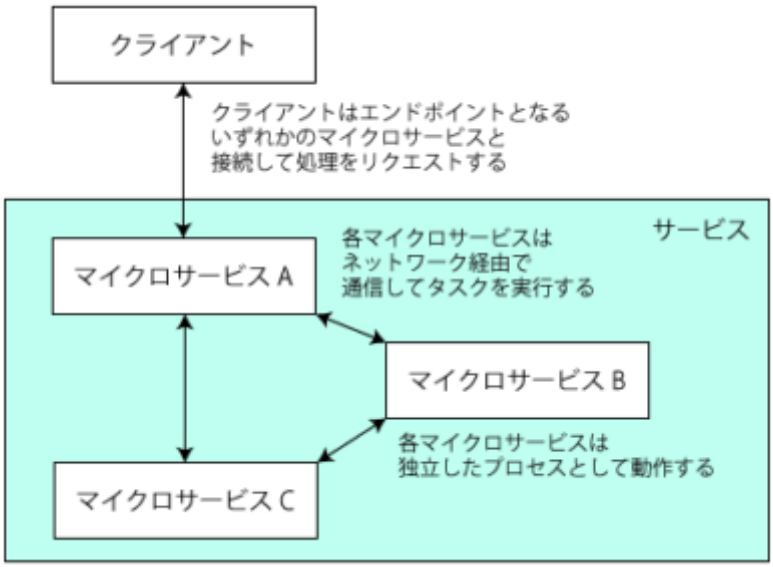


図1 マイクロサービスアーキテクチャのイメージ

1. 個々のマイクロサービスはそれぞれ独立したプロセスとして動作する
2. 各マイクロサービスは主にネットワーク経由で通信して所定のタスクを処理する
3. 各マイクロサービスはほかのマイクロサービスに依存せず起動でき、独立してデプロイやアップデートが可能

この3つの特徴は、それぞれ次のようなメリットを生む。まず1.の「個々のマイクロサービスが独立したプロセスとして動作する」だが、これによって各マイクロサービスをそれぞれ異なる言語で実装できるようになる。たとえば、あるマイクロサービスは既存システムとの互換性のためにPerlで実装し、別のマイクロサービスはより開発やメンテナンスが容易なPythonで実装する、といったことが可能になる。また、使用するライブラリやフレームワークなども独立して選定ができる。

2.の「各マイクロサービス間は主にネットワーク経由で通信して所定のタスクを処理する」では、これによってそれぞれのマイクロサービスを異なるマシン上で実行できるようになり、マイクロサービス間の依存性を小さくすることが可能になる。さらに、同一のマイクロサービスのプロセスを複数同時に実行させてリクエストを振り分けることで、冗長化や性能向上（スケーリング）を行える。

是否将当前网页翻译成中文 网页翻译 中日对照 关闭

3.の「独立してデプロイが可能」では、[illegible]うになるというメリットがある。これによ[illegible]る。また、各マイクロサービスの規模を小さくしてプログラムの複雑性を減らすことで、開発やメンテナンスに必要なコストの圧縮も見込める。

マイクロサービスアーキテクチャのこのようなメリットは、最近普及が進んでいる仮想化技術やクラウドサービスとの相性も良い。たとえば仮想化技術やクラウドサービスを使ったインフラストラクチャでは、高スペックなマシンを1台用意するよりも、比較的低スペックなマシンを複数台用意して組み合わせるクラスタ構成をとった方が低コストになるケースが多く、比較的小規模なプロセスを多数実行するマイクロサービスアーキテクチャはコスト面でも優位性が期待できる。

マイクロサービスアーキテクチャのデメリットと設計時の留意点

一方で、マイクロサービスアーキテクチャではコンポーネントを小さな粒度で分割するため、コストやパフォーマンスに影響するようなオーバーヘッドが発生しやすいというデメリットもある。たとえばそれぞれのマイクロサービスは別プロセスで動作するため、トータルのメモリ使用量は増える傾向がある。また、ネットワーク経由でのデータのやり取りにはある程度の遅延も発生する。

さらに、マイクロサービス同士の通信は原則として公開APIを通じてのみ行うことから、これらAPIの仕様を事前に十分に検討しておかなければならない。データベースについても設計時に注意が必要で、マイクロサービスアーキテクチャでは基本的にはマイクロサービスごとに固有のデータベースやストレージを持ち、基本的にほかのマイクロサービスが管理しているデータベースやストレージには直接アクセスできないよう設計するのが基本となっている（**図2**）。そのため、複数のマイクロサービスにまたがるトランザクションは実現が難しい。

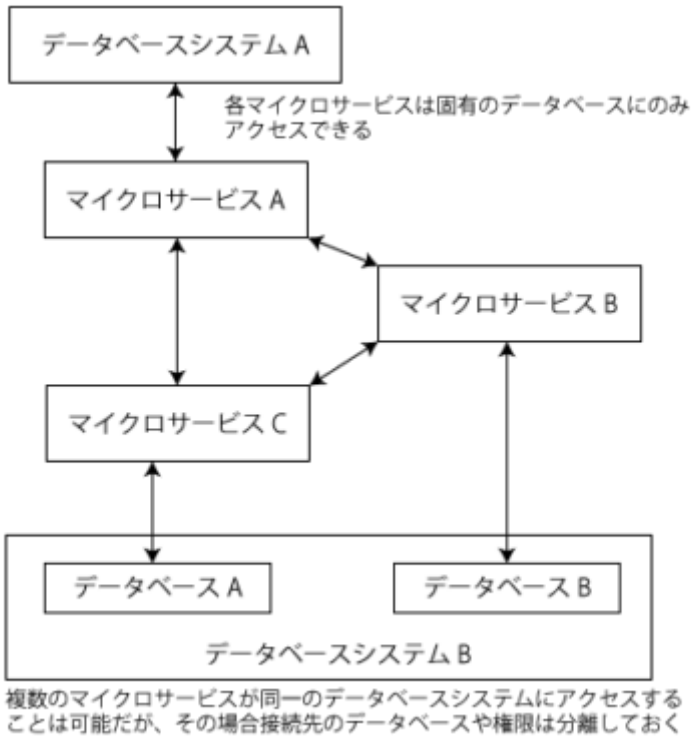


図2 マイクロサービスアーキテクチャにおけるデータベース設計

是否将当前网页翻译成中文 网页翻译 中日対照 关闭

そのため、マイクロサービスアーキテク…ロサービスに割り当てるかが重要となる。…、逆に分割粒度が大きいとマイクロサービスのメリットは少なくなってしまう。

また、運用時の課題として監視対象が増えることによりサービスの死活監視やログの管理が煩雑になるといった問題もある。マイクロサービスアーキテクチャを選択する場合、このような問題を認識した上で、設計時には十分な検討が必要となる。

マイクロサービスで使われる技術

さて、マイクロサービスの概要についてはすでに多くの記事や文献などで紹介されているので、アーキテクチャに関する話についてはこのくらいにとどめておき、続いてはマイクロサービスを構築する際の課題と、それを解決するために現時点で使われる技術について紹介しておこう。

データベースの選択と設計

マイクロサービスアーキテクチャにおいても、データをデータベースに格納して管理するという点はほかのアーキテクチャと変わらない。マイクロサービスアーキテクチャだからと言って特別なアクセスの仕方をする訳ではなく、そのため使われるデータベースもOracleやMySQL、PostgreSQLといったリレーショナルデータベースやMongoDB、RedisといったKey-Valueストア/ドキュメントデータベースなどさまざまだ。

マイクロサービスアーキテクチャでは各マイクロサービスがそれぞれ独立してデータを管理するため、各データベースに格納するデータが比較的シンプルになることも多い。そのため、データベースシステム自体の管理やチューニング作業などが不要なクラウド型のデータベースサービスを活用しやすい。たとえば大量のデータを操作したり、複雑なクエリを処理したりするようなマイクロサービスは独自にデータベースサーバーを用意し、そうでないマイクロサービスについてはクラウド型のデータベースを利用する、といった住み分けも可能だ。

ただし、データベースの設計には注意が必要だ。前述の通り、マイクロサービスアーキテクチャでは各マイクロサービスごとにデータベースを管理し、各データベースは1つのマイクロサービスのみからアクセスできるようにするという原則がある。このように設計することで、マイクロサービスごとの独立性が向上し、ロックやトランザクションなどにまつわるトラブルやパフォーマンス低下などを防ぐことが期待できる。また、マイクロサービスごとに適切なデータベースエンジンを採用できるというメリットや、万が一データベース構造を大きく変更せざるを得ない状況になった場合でも、それによる影響範囲を抑えられるというメリットもある。

一方でこのような設計のために発生するデメリットもある。まず、ほかのマイクロサービスが管理するデータベース内のデータが必要となる場合にそのマイクロサービスを経由してデータを取得しなければならないという点だ（**図3**）。

是否将当前网页翻译成中文 网页翻译 中日对照 关闭

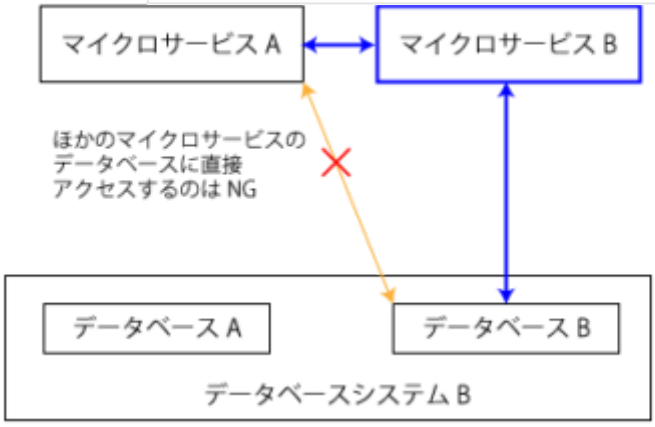


図3 ほかのマイクロサービスが管理するデータにはそのマイクロサービス経由でアクセスする

そのため、こういったアクセスが必要な場合はデータをやり取りするためのAPIを用意する必要がある。また、複数のマイクロサービスにわたるトランザクション処理を実装するのは難度が高い。トランザクションが必要となる場合、1つのマイクロサービス内で完結するように設計するべきだろう。

コンポーネント間通信技術の選定

マイクロサービスアーキテクチャでは、各マイクロサービスがほかのマイクロサービスと通信して処理を実行する。この通信方法の選定もマイクロサービスアーキテクチャを利用するに当たって重要なポイントとなる。

マイクロサービス同士がやり取りする情報は「メッセージ」と呼ばれる。メッセージのやり取りには大きく分けると同期的な手法と非同期的な手法の2種類があり、同期的なメッセージ通信では送信元と送信先が同期して処理を行う（**図4**）。

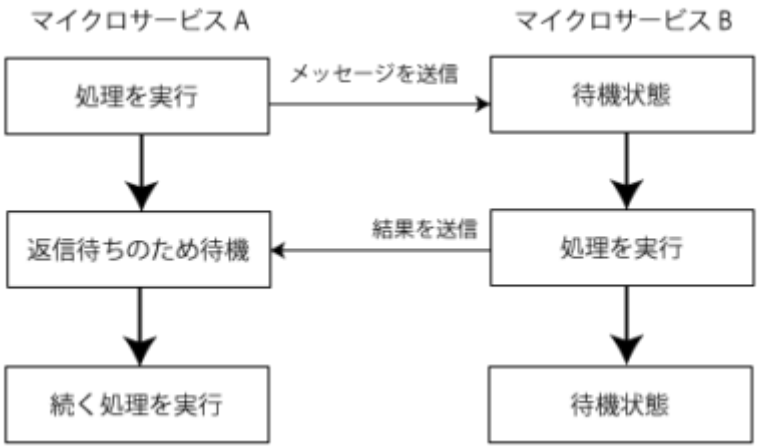


図4 同期的なメッセージ通信

そのため、メッセージの送信元は送信先がメッセージを受け取って何らかの処理を実行してその結果を送り返すまで、処理を一時停止して待機することとなる。もちろん実際の実装ではマルチスレッドやイベントキューなどを使

って複数の処理を並行して実行するため送信しない限り次の処理には進めない。

是否将当前网页翻译成中文 网页翻译 中日对照 关闭

一方、非同期的なメッセージ通信では、マイクロサービスが送信したメッセージはメッセージキューに保存され、送信後に即座に次の処理を実行できるという特徴がある（**図5**）。

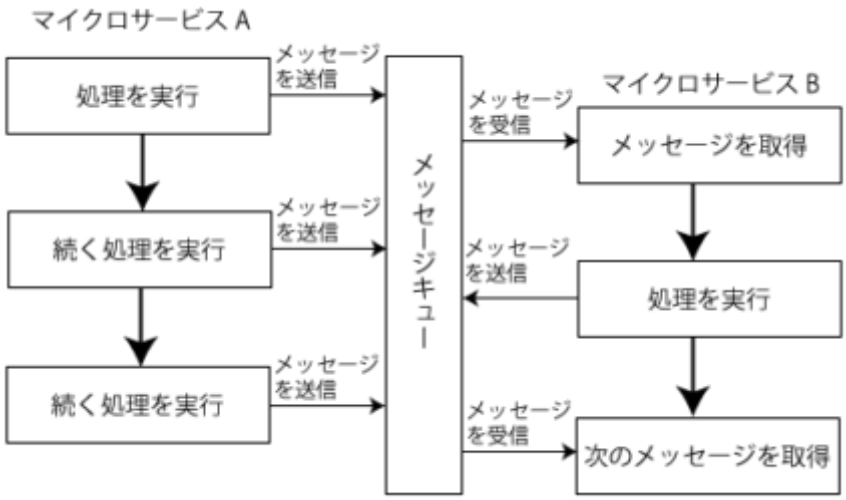


図5 非同期的なメッセージ通信

送信されたメッセージを処理するマイクロサービス側では、メッセージキューに格納されたメッセージを順次取り出し、対応する処理を実行する。そのため非同期的なメッセージ通信ではリアルタイム性は保証されないが、通信先マイクロサービスの状況に左右されずに処理を完了させることができるというメリットがある。

同期的なメッセージ通信の選択肢

同期的なメッセージ通信の実現手法として一般的なのが、HTTPベースでメッセージのやり取りを行う手法だ。HTTPは広く普及しており、クライアント/サーバーを実装したライブラリも多数選択肢があるため利用しやすい。ただし、やり取りしたいデータやエンドポイント（接続先URL）をどのような形で表現するかについては複数の手法がある。そのなかでも現在広く使われているのが「REST」と「RPC（RPC over HTTP）」だ（**図6**）。

是否将当前网页翻译成中文 网页翻译 中日对照 关闭

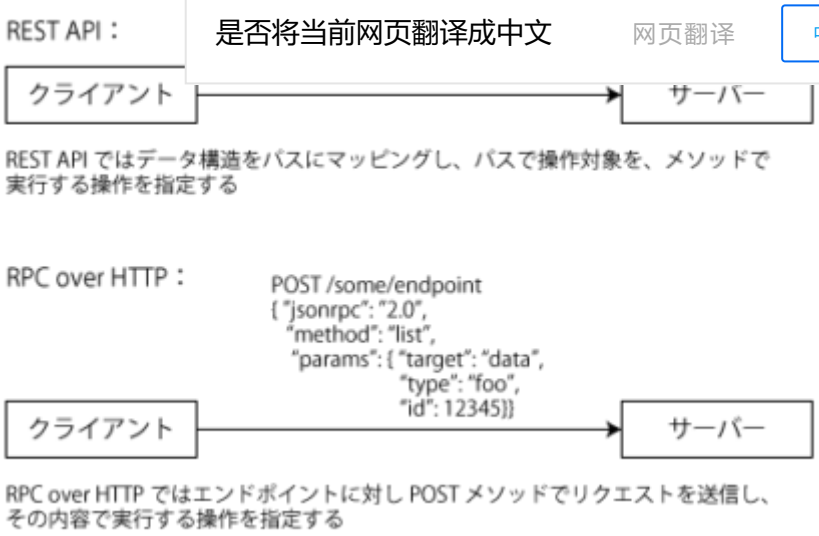


図6 RESTとRPCの違い

RESTは、操作対象とするリソースをディレクトリ構造（木構造）で表現しそれをURLにマッピングして指定する。また、実行する処理は「GET」や「POST」、「PUT」、「PATCH」、「DELETE」といったHTTPリクエストメソッドによって指定する。GETメソッドがリソースの取得、POSTメソッドがリソースの作成、PUTメソッドおよびPATCHメソッドがリソースの更新、DELETEメソッドが削除に対応する。

POSTやPUT、PATCHといったメソッドの場合、クライアントは何らかの形で送信するデータを構造化しなければならないが、データの構造化にはXMLやJSONが使われる。また、RESTを利用したシステムの中でもCookieなどを使ったセッション管理を用いず、セッションに依存しないような実装は「RESTful」と呼ばれる。

一方のRPCは、URLではなく送信するメッセージ内で操作対象や実行する処理（メソッド）を指定する点がRESTと異なる。RPCはHTTPだけでなくさまざまなプロトコルで利用できるが、今日ではHTTPをベースにすることが多い。また、送信するデータフォーマットもさまざまなものが使われるが、HTTPベースの場合XMLもしくはJSONが使われるのが一般的だ。

RESTでは処理対象のリソースに応じてURLが動的に生成される一方、RPCではエンドポイントとなるURLは1つもしくは少数となるためURLの設計や変更がしやすいというメリットがある。いっぽうで、URLだけでは実行する処理を判断できないというデメリットもある。

HTTPベースではないRPCとして最近では「gRPC」も注目されている。gRPCはGoogleが開発を進めるRPC実装で、効率的にデータをやり取りできることや、HTTP/2ベースでの双方向通信がサポートされている点などが特徴となっている。C++やPython、Java、Rubyなどの言語向けのライブラリが提供されているほか、「protocol buffers」というフォーマットでメソッドの名前や受け付けるデータ形式、戻り値となるデータ形式を記述し、そこから対応するコードを自動生成するツールなども提供されている。これによって、異なる言語で実装したシステム間でも整合性の取れた実装が可能になる。

そのほか、同期的なメッセージやり取りに使える技術としてはApache Hadoopで使われる「Apache Avro」やFacebookが開発した「Apache Thrift」といったフレームワークもある。

非同期的メッセージ通信の選択肢

是否将当前网页翻译成中文 网页翻译 中日対照 关闭

非同期にメッセージをやり取りする手法[illegible]ざまなものがあるが、大きく「ブローカー[illegible]ー」は直訳すると「仲介者」という意味で、その名の通り仲介者となるプログラムを介してメッセージをやり取りするアーキテクチャを「ブローカード」、仲介者なしで直接送信元と送信先がメッセージをやり取りするアーキテクチャを「ブローカーレス」と呼ぶ（**図7**）。

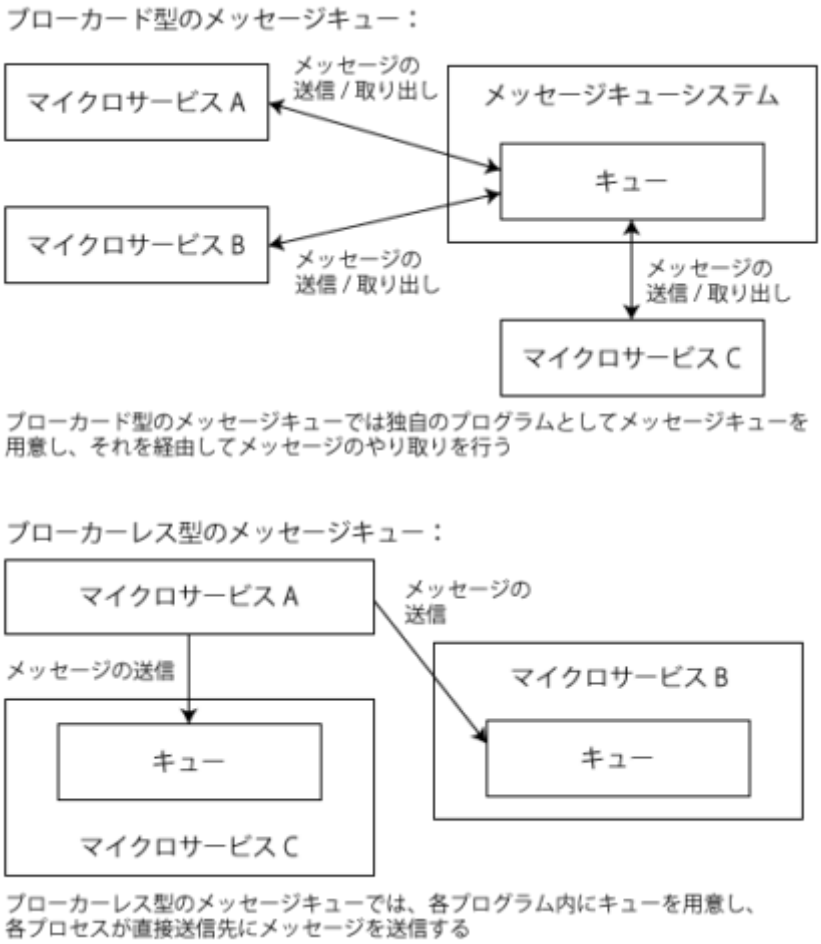


図7 メッセージキューの実装手法

ブローカードのメッセージキューとしては「RabbitMQ」や「Apache ActiveMQ」などが有名だ。これらはAMQP（Advanced Message Queuing Protocol」というプロトコルのオープンソース実装で、利用の際はブローカーとなるプログラムを立ち上げておく必要がある。

特徴としては、メッセージを受け取るプログラムがクラッシュするなどして動作していない場合でも、メッセージ送信元はブローカーさえ動作していれば問題なくメッセージ送信が行える点がある。この場合、送信されたメッセージはブローカー内に保存され、メッセージ送信先が再度動き出したときに順次処理される。ただし、ブローカー自体がクラッシュした場合は送信元でなんらかの対処を行わなければならない。

一方、ブローカーレスのメッセージキューでは、メッセージを受け取るプログラムがそれぞれにキューを持ち、送信元のプログラムが送信先のプロセスに直接メッセージを送信する。そのため、遅延が少なく高速に処理できるという傾向がある。ただしプログラムにメッセージキューの機能を組み込む必要性があるため、プログラムの複雑性や使用するリソースが増えるというデメリットもある。

是否将当前网页翻译成中文 网页翻译 中日对照 关闭

ブローカーレスのメッセージキューの代
ラムにライブラリとして組み込むことで
てやり取りできる。主要な言語向けのバインディングも提供されており、さまざまなプログラミング言語と組み合わせての利用が可能だ。

また、データベースをキュー代わりに利用するという手法もある。これは広義にはブローカード型のメッセージキューとなる。多くのデータベースにはトランザクションやロックといったデータの一貫性を保つ機能があり、これを利用してメッセージキューのようなものを独自に実装できる。

インフラストラクチャの選択とサービスメッシュ

マイクロサービスアーキテクチャでは、多数の独立したサービスを協調動作させて所定のサービスを提供することから、デプロイしなければならないコンポーネントの数も増えることになる。そのため、仮想化やクラウドインフラストラクチャを使ってサービスを運用することが多い。

各マイクロサービスは「独立したプロセス」である必要があるが、必ずしも各マイクロサービスを別のマシンで実行させる必要は無い。ただ、ライブラリの依存性などの問題から、各プロセスはそれぞれ独立したマシン（もしくは仮想マシン、コンテナ）上で実行させるのが一般的だ。

インフラストラクチャの選定についてはマイクロサービスアーキテクチャとはやや外れた話題になるためここでは触れないが、クラウドインフラストラクチャやコンテナなどの場合、各マイクロサービスが使用するIPアドレスが動的に変動する。そのため、こういった環境ではほかのマイクロサービスとメッセージをやり取りする際、目的のマイクロサービスに割り当てられているIPアドレスを何らかの方法で特定しなければならない。また、冗長性確保やスケーリングのために同じマイクロサービスを並行して複数稼動させることも多いが、この場合ロードバランサーのように動的に接続先を管理する仕組みが必要となる。こういった、サービスやそのネットワーク接続を管理するシステムを「サービスメッシュ」と呼ぶ。

たとえばクラスタインフラ管理機能を提供する「Kubernetes」ではシンプルなサービスメッシュ機能が組み込まれている。また、独自のサービスメッシュ機能を提供するクラウドインフラストラクチャもある。「Istio」や「Linkerd」といった、インフラストラクチャに依存せずにサービスメッシュ機能を提供するソフトウェアもある。これらは独自のプロキシを利用してマイクロサービス間の通信を管理する仕組みになっており、適切な接続の転送に加えて、ロードバランサーやネットワークゲートウェイ、認証、モニタリングなどの機能も備えている。

なお、LinkerdはKubernetesと組み合わせて利用することが前提となっているが、IstioはKubernetesを使ったインフラだけでなく、仮想マシンベースのインフラにも対応しているのも特徴だ。そのため、さくらのクラウドのようなIaaS型のクラウドサービスでも利用できる。

デプロイとアップデート

マイクロサービスアーキテクチャにおいては、サービスを構成するマイクロサービスの数が多くなる。そうすると、デプロイメント（デプロイ）やアップデート作業の自動化も必要になる。手作業でいちいち個別にこういった作業を行うのは手間がかかり、ミスなども発生しやすくなるからだ。

クラウド型のインフラストラクチャを利用する場合、ある程度はインフラストラクチャ側でデプロイやアップデートのための機能が提供される。たとえばKubernetesでは「deployment」という、デプロイのための機能が用意

是否将当前网页翻译成中文 网页翻译 中日对照 关闭

されている。また、多くのクラウドサー
ったサービスメッシュ支援ツールにもテ

こういったデプロイ機能やデプロイツールを利用するメリットの1つとして、サービスを停止させずに各マイクロサービスのソフトウェアをアップデートしたり、トラブルを少なくするようなアップデート手法を容易に利用できる点がある。

たとえばシステムのアップデート手法の1つとして、古いバージョンのソフトウェアを稼動させたまま新たに新しいバージョンのソフトウェアを使ってプロセスを立ち上げ、その後接続先を切り替える、というやり方がある。これは「ブルーグリーン（Blue-Green）デプロイメント」と呼ばれるものの一種で、もしアップデート後に新バージョンのソフトウェアに不具合が見つかった場合、接続先を戻すだけでロールバックを速やかに実行できるというメリットがある（**図8**）。

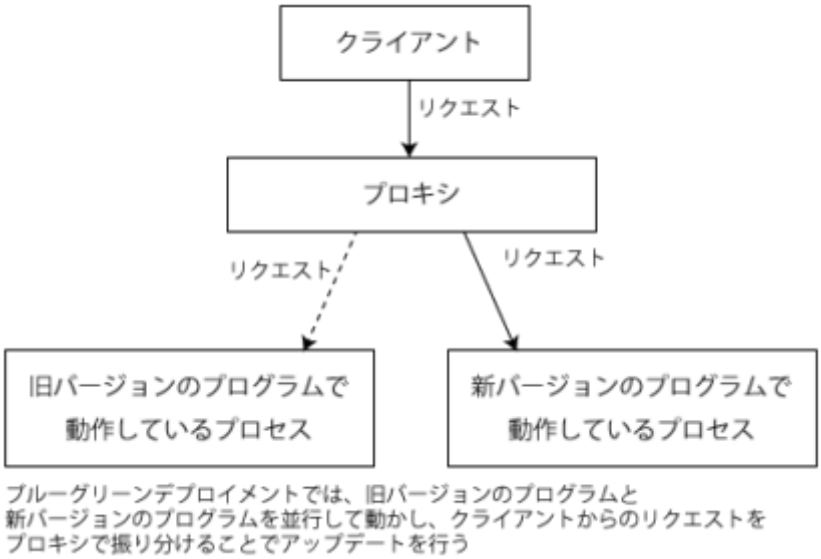


図8 ブルーグリーンアップデートの概要

ブルーグリーンデプロイメントは、元々はデプロイ先の本番環境を2つ用意しておき、デプロイごとに使用する環境を切り替えるという手法であり、クラウド型インフラストラクチャではない環境でも利用できたが、クラウド型インフラストラクチャではこういった冗長な構成を容易かつ比較的低コストで実現できるため普及が進んでいる。

また、マイクロサービスアーキテクチャにおいては冗長性を持たせるため、1つのマイクロサービスを複数のプロセスで並行動作させておくことが多い。そのため、本番環境を一気に切り替えるのではなく順次切り替えていく「ローリングアップデート」と呼ばれるやり方も使われる。

ブルーグリーンデプロイメントを発展させた「カナリアリリース」という手法も考案されている。カナリアリリースはブルーグリーンデプロイメントと似ているが、複数稼動させているノードのうちまずは少数のノードだけを新しいバージョンのものに入れ替えて様子を見て、問題がなければその後残りのノードを入れ替えていく、という手法だ。この手法は、かつて炭鉱での作業時に有毒ガスを検知するためにカナリアを持ち込んでいた（もし有毒ガスが発生したら人間より先にカナリアが異常をきたす）という話に由来して「カナリアリリース」と名付けられている。カナリアリリースを応用した手法として、管理者や一部のユーザーのみに限定して新バージョンを使わせるといった手法もある。

是否将当前网页翻译成中文 网页翻译 中日对照 关闭

クラウド型インフラストラクチャとマイ の マイクロサービスのみを入れ替えること 一部のプロセスのみをあえて不具合を起こすようなものに入れ替えて、そういった場合でも適切にエラー処理が行われシステム全体として適切に動作するかを確認する「フォールトインジェクション」といった手法があり、サービスメッシュ管理ソフトウェアによってはこのような機能を提供するものもある。

ソフトウェアやインフラストラクチャの進化によって実用的になったマイクロサービスアーキテクチャ

このように、マイクロサービスアーキテクチャではクラウドインフラストラクチャや仮想化、そしてさまざまな周辺ツールが活用されている。そのため、各マイクロサービスの規模は小さくなるが、その分開発にあたるエンジニアは同期/非同期通信やデータベース、インフラに関するものまで、より幅広い知識が求められる。また、特に機能毎のサービス分割に関しては十分な検討が必要となる。そのため、開発の際には「とりあえず流行っているから」などの理由でマイクロサービスアーキテクチャを採用するのではなく、事前に十分に検討を行い、対象システムがマイクロサービスアーキテクチャでの構築に本当に適しているのかを考える必要があるだろう。

なお、プロセス間通信やコンポーネントの分離、デプロイ手法など、マイクロサービスアーキテクチャの考え方はそれ以外のアーキテクチャでも参考になる点も多い。たとえばメインの処理は非マイクロアーキテクチャで構築し、一部のみをマイクロアーキテクチャ風の形で構築する、といった形も考えられる。既存システムのアップデートなどでは、こういった手法も十分実用的だろう。

さて、マイクロサービスアーキテクチャの採用例が増えた一因としては、クラウドインフラストラクチャの普及や関連ツールの登場が大きい。次記事ではマイクロサービスの構築に使われるサービスメッシュについて、実際の環境構築も含めた解説を行う予定だ。

クラウド , サーバー管理 , 運用管理 , オープンソース , マイクロサービス

ツイート シェア 22 225

是否将当前网页翻译成中文 网页翻译 中日对照 关闭

© SAKURA internet Inc.