解説
Article

クラウドネイティブ時代に振り返るコンテナのこれまでとこれから

千葉立寛 | 日本アイ・ビー・エム（株） 東京基礎研究所

クラウドネイティブの潮流

皆さんはクラウドネイティブという言葉を聞いたことがあるだろうか．クラウドネイティブとは，一言で言うならば，クラウド上で実行されるサービスからアプリケーション，ミドルウェア，インフラストラクチャに至るまでのすべてのコンポーネントにおいて，クラウドの特性であるスケーラビリティとレジリエント性を持つようにそれらのコンポーネントを構成する手法や考え方の総称である．なかなかピンとこない言葉ではあるが，多くのアプリケーションがクラウド上で実行される時代においては，今後ますますこのような特性を考慮してシステムを構築するのは必然の流れとなっている．

クラウドが登場し，クラウドという言葉が広く認知されるようになってから 10 年ほどが経過した今，オンプレミスで自前で運用しているクラスタ環境の多くもクラウドへの移行が進んでいる．柔軟にその構成を変更することが容易に迅速に低コストでできるようになったためである．しかしながら，既存のシステムで動作していたアプリケーションをそのままクラウド上に移行したとしても，ただクラウド上で動作しているというだけであり，自動的にレジリエントかつスケーラブルなアプリケーションになることはない．アプリケーション自体もクラウドネイティブ指向なものへと変化していく必要があり，今まさにこのクラウドネイティブ化の流れが起きている．

ではなぜ，今この流れが巻き起こっているのだろうか？　そして，その背景には何があるのだろうか？　多くのコンポーネントがクラウドネイティブ化していく中，DevOps やアジャイル開発，マイクロサービスといったアプリケーションにおけるクラウドネイティブ化の流れの中心に存在するものが，コンテナ技術である．現代のモダンなクラウド上のアプリケーションは，コンテナ化されていることがほとんどであり，コンテナ化されていることはもはや必須である．

本稿では，コンテナのこれまでの歩みを振り返りつつ，そもそもコンテナとは何なのか，なぜコンテナが登場したのか，なぜコンテナが注目されているのかを解説していく．さらに，今後コンテナ化が進んでいった次の未来について展望する．

クラウドの普及とアプリケーションの変化

コンテナを構成する技術やアイディアそのものは以前から存在していたが，なぜここまで利用され始めているのだろうか？　仮想化というインフラの変化と，ビジネス上求められる特性を取り入れるためのアプリケーションモデルとソフトウェア開発手法の変化という 2 つの変化が，コンテナという形で合流して発展してきたのが現在であるからだと筆者は考えているが，この章では，仮想化技術がコンテナにもたらした影響と，アプリケーションおよび開発スタイルの変化がコンテナにもたらした影響の 2 つの側面から，コンテナの登場に至る流れを振り返っていきたい．

仮想化技術によるクラウド利用の広がり

仮想化の歴史は古く，その源流を辿ればメインフレームの時代にまで遡る．2000 年頃には，VMware

やXen，KVMの登場により，x86環境でもハードウェア仮想化によるバーチャルマシン(VM)が利用可能となったことから，多くのユーザにとって仮想化技術は身近になり，実際に手元のマシンに導入してVMを構築するなど，VMに触れる機会は多くなったと考えられる．1台のサーバ上に多数の仮想的なサーバを構築することで，サーバのハードウェアリソースを柔軟かつ効率的に利用することができるようになった．さらに，仮想マシンではサーバの環境をそのまま仮想ディスクイメージとして保存できることも特徴である．ディスクイメージをコピーして実行するだけで，必要なときに必要なだけサーバを構築できるので，VMの利用は急速に広まっていった．

ただ，ユーザの視点から見ると，所有する物理サーバとその上に構築される仮想サーバの2つのレイヤが存在しており，物理サーバのリソースを融通しあうことでクラスタ全体での効率化は進んだが，依然として物理サーバの管理も必要であり，日々発生するハードウェア故障などとも向き合わなければならなかった．そういった中で，Amazon EC2をはじめとする仮想サーバをネットワーク越しで利用可能にするIaaS型クラウドが登場したことで，ユーザは物理サーバの管理から解放され，マシンそのものをソフトウェア的に扱えるようになった．ユーザにとっては，この大きな変化により，マシンを含めたアプリケーション実行環境は自由にローカル環境からクラウド環境に必要に応じて移動可能であるという考え方が浸透していくきっかけとなった．

アプリケーションデザインと開発スタイルの変化

次に，ソフトウェア開発の潮流という側面からコンテナに至る流れを追っていく．時代のニーズに合わせて，インフラが仮想化へ変化していったように，アプリケーションでも同様に大きな変化が起こっていく．さまざまなサービスがインターネットを通じて提供される今日において，いかに迅速にサービスを構築し，その価値をユーザにいち早く提供するかがビジネスを決定づける重要な鍵となっている．一度作ったサービスやアプリケーションを変更せずに使い続けるのではなく，変化に対してアプリケーションも追従していく必要があるが，こうした変化に柔軟に対応するには，それまでの開発スタイルでは難しかった．

その原因の1つは，モノリシックに構築されたアプリケーションである．モノリシックなアプリケーションでは，小さな変更であっても，全体をビルドしなおす必要があったりするため，アプリケーションそのものの構成を見直すことが求められた．この背景により登場したのが，マイクロサービスで構成されたアプリケーションである．マイクロサービスでは，個々の機能ごとにアプリケーションを分割し，RESTなどのAPIを通じて個々のコンポーネントを疎結合で構成して，サービス全体を構成するアプローチであり，これにより機能コンポーネント単位での更新も容易になっていった．マイクロサービス的な考え方や構成手法自体は，オブジェクト指向やSOA(サービス指向アーキテクチャ)などでも目指していた部分ではあるが，RESTやgRPCといったプロトコルの成功とともに，クラウド上でのアプリケーション構成手法としてマイクロサービスは広く受け入れられていった．

これを後押しするかのように，今では当たり前のように使われているGitを始めとするソースコード管理システムやJenkinsやTravisのようなCI/CDシステム，および，ChefやAnsibleのような構成管理ツールが広く普及した．その結果，ソースコードの変更に応じて，アプリケーションをビルド・テストし，新たな環境やVM上に即座にデプロイするという仕組みが構築されていった．これらの手法は，2011年頃にはクラウド上でのアプリケーションのデザインパターンとなるThe Twelve Factorsとしてまとめられ，PaaSやSaaS型クラウドでアプリケーションを構築するベストプラクティスとして注目を集めた．

コンテナ登場前夜

VMによるインフラの柔軟性とアプリケーション構築の迅速性により，従来に比べて圧倒的に速いスケール

でシステムやサービスが構築可能になったが，アプリケーションを確実に，かつ，素早く動作可能にするという観点では，いささか無駄や改善すべき部分も多かった．

1つ目は，VMという単位で管理することに起因するオーバヘッドである．VMという単位で管理することで，ディスクイメージから容易に環境を複製したりすることはできるが，アプリケーションを複製してスケールさせるという観点から見ると，ディスクイメージも決して小さなサイズではないため，VMの起動に数分から数十分は要する．さらに，VM上のOSやネットワークを含めたシステムの構成や管理は依然としてユーザの責任で行う必要がある．

2つ目は，アプリケーション実行の確実性である．Javaにおける JVMのように，一度ビルドしたアプリケーションが確実に実行できるランタイムがあればよいが，異なるVMやテスト環境，本番環境，クラウド環境でのアプリケーションの実行には，さまざまな依存関係やライブラリの違いにより，しばしば困難がつきまとうことも多かった．

コンテナの登場と普及

これらの問題を解決するアプローチとして，2013年にアプリケーションをコンテナ化するためのプラットフォームが登場した．今や最も広く利用されているDockerである．VMのようなハイパーバイザ型の仮想化との比較で，しばしばコンテナ型仮想化とも呼ばれることも多いが，正確にはコンテナは仮想化技術ではない．ユーザから見た場合の抽象レイヤとしては，あながち間違いではないが，コンテナは，本来，プロセスに対して隔離と制限を行うための仕組みである．この隔離と制限を行うために使われている機能が，OSのカーネルで実装されているcgroupsとnamespaceである．cgroupsでは，プロセス自体に対して利用可能なハードウェアリソースを制限・分離する機能を提供し，namespaceでは，プロセス間でのソフトウェアリソースの分離・共有する機能を提供している．これら2つの機能を組み合わせることで，コンテナは実現されている．コンテナ化されたアプリケーションは，いわばホストOS上で動作する1プロセスであり，仮想化オーバヘッドが存在しないため，VMよりもはるかに小さいフットプリントで高速に実行できるのである．

これら2つの機能はDockerの登場以前からカーネルに存在し，LXC（Linux Containers）のように，それらの機能を使ってプロセスをコンテナ化して実行することも行われていた．では，なぜここまで急速にDocker を用いたコンテナ化が広く普及したのだろうか？　その要因としては，Dockerがプロセスを隔離するだけではなく，プロセスが利用するファイルやライブラリを組み込んだ再利用可能なイメージとしてパッケージしたことで，さまざまな環境で確実に動作するような可搬性を用意したことが大きな要因であった．

コンテナのイメージはUnionFSを用いて差分管理されたCopy-On-Writeのレイヤで構成されており，コンテナイメージサイズのフットプリントの小ささもさることながら，アプリケーションが変更された場合でもその差分のレイヤだけを重ねるだけでコンテナが利用するファイルを構成することが可能なため，コードやサービスを頻繁に更新するようなクラウド型アプリケーションにとってデプロイまでの時間が短縮できた．

また，コンテナのイメージそのものを簡単に作成することができ，さらにその公開されたコンテナイメージから自分のコンテナイメージを作成できるレジストリサービスが誰でも利用できる形として公開された．このアプリケーションのコンテナ化とパッケージングされたイメージの提供の2つのエコシステムがコンテナの利用が爆発的に普及していった大きな理由であった．

これまでのVMベースのデプロイから，コンテナベースのデプロイへ急速に進んでいったのは，コンテナ起動の迅速性とコンテナ化による可搬性の2つが，アプリケーション実行という観点においては圧倒的に優れていたからである．

そんな中，2014年にGoogleが公開したコンテナに関する資料[1]の中で，Google内ではすべてがコンテナ

化されており，1 週間に 20 億以上のコンテナが起動しているという発表があった．Docker によるコンテナ化が一般に注目され始めたタイミングで，すでに Google 内ではコンテナベースのアーキテクチャで動作していることが明らかになり，よりいっそうコンテナが目指す方向性がクラウドスケールのアーキテクチャとして正しいという認識が広がっていくきっかけになった．

コンテナ管理

Docker によってコンテナの利用が急速に広まり，多くのアプリケーションがコンテナ化されていった．先にも述べた通り，ピュアなコンテナ技術は OS 上でプロセスをほかから分離するための技術であるが，その次に必要になるのは，多数のノードで構成されたクラスタ上において，どのコンテナをどこのノードで動かすのかというスケジューラやコンテナマネージャとなるのは必然の流れであった．

そういった中，2014 年に Google が Kubernetes をオープンソースとして公開した．Kubernetes とは，クラスタ上でのコンテナのスケジューリング，負荷に応じたロードバランスやコンテナのオートスケーリング，停止したコンテナのリスタートなど，コンテナシステム全体を管理するための仕組みを有するコンテナマネジメントシステムである．Google 内で使われていた Borg というクラスタマネージャをベースにして作られているが，Borg を運用していく中で得られた多くの知見と，同様に Google 内で使われていたクラスタマネージャの Omega の思想を取り入れて，Kubernetes は開発された．Kubernetes がいかにして開発されていったのかという成り立ちについては，文献 2) が詳しいので，ぜひ一読すべきである．この Kubernetes の登場により，実システムでのコンテナの利用が一気に実用レベルまで引き上げられていったのである．

クラウドネイティブの時代へ

Kubernetes v1.0 の登場とともに，Cloud Native Computing Foundation (CNCF) が設立された．CNCF は，クラウドネイティブを普及・推進するための組織であり，クラウドネイティブに必要かつ重要なオープンソース・ソフトウェアの多くを支援している．Kubernetes はもとより，コンテナランタイム，モニタリング，ロギング，プロトコルなど 20 以上のプロジェクトが参画している．

ハイパーバイザ型仮想化やアプリケーション環境の変化を経て，コンテナ，コンテナマネージャというクラウドネイティブに至る流れをこれまでの章で追ってきたが，ここで少し，この流れの背景を補完するようなデータを，Xen，Docker，Kubernetes という 3 つのプロジェクトを用いて紹介しようと思う．

図 -1 は，2004 年頃から現在に至るまでのそれぞれのプロジェクトの検索トレンドを Google トレンドを用いて相対的に比較したものである．また図 -2 は，それぞれのプロジェクトの月ごとのコミット数からどれだけ活発に開発されているかを示したものである．Amazon EC2 は，Xen を用いて実現されているが，2006 年頃にそれに呼応するように Xen の開発も活発に行われていたことが分かる．検索トレンドは，Docker を基準とした相対値であるが，Xen に対する関心度は，2007 年頃をピークに減少していき，コンテナが普及し始めた 2015 年頃，すなわち Docker v1.0 がリリースされたあたりを境にその人気度は逆転している．Kubernetes においても，v1.0 のリリースを境に検索トレンドは上昇していっていることが分かる．また，プロジェクトのコミット数を見てみると，コンテナマネジメントは現在進行系で活発な開発が行われているが，コンテナランタイムの開発は 2017 年頃から安定してきたようにも思える．そういう意味では，コンテナはある意味で成熟し，実際の運用に使われるレベルに達しているといえるだろう．

今後の課題と展望

Docker によるコンテナ化と Kubernetes によるコンテナマネジメントが広く普及してきたことで，クラウドネイティブな世界を構築する準備と環境は整ってきており，実際，多くの企業でもこれらの仕組みでシステムが構築されはじめている．しかしながら，コンテナ化の流れは始まったばかりであり，依然としてまだ解決すべき問題もいくつか存在する．ここでは，コンテナを中心とした場合に，近い将来起こる流れと課題を整理しつつ，今後のクラウドネイティブの展望を考える．

アプリケーションコンテナのクラウドネイティブ化

コンテナの延長線上で VM が捉えられることが多いため，既存の VM で動いているアプリケーションをそのままコンテナに移してしまうケースがある．これは，パッケージング化する単位が小さくなって取り回しやすくなった以上の意味はなく，本質的な部分では，クラウドネイティブ化はされていない．クラウドネイティブ化とは，コンテナにパッケージングすることではなく，コンテナ化した上で，アプリケーション自体に柔軟性と自律性を持たせることである．たとえば，キーバリューストアのような単純な状態を保存する機構をできるだけコンテナの外部におき，コンテナ自身をステートレスに保つことで，いつコンテナが停止しても自律的に復旧できるようにしたりすることである．クラウドネイティブなアプリケーションは，Declarative に宣言されたあるべき姿・状態（たとえば，10 個のコンテナを常に立ち上げておく，など）になるように，自動的にスケールアウトできるようにするべきであるが，このようなモデルが多くのアプリケーションに適用されるのは，まだ時間がかかると思われる．このアプリケーションのクラウドネイティブ化の流れは，今後数年をかけて変わっていくポイントの 1 つであるだろう．

コンテナのセキュリティ

現在広く普及しているコンテナランタイムは，多数のコンテナ間でホスト OS のカーネルを共有したモデル

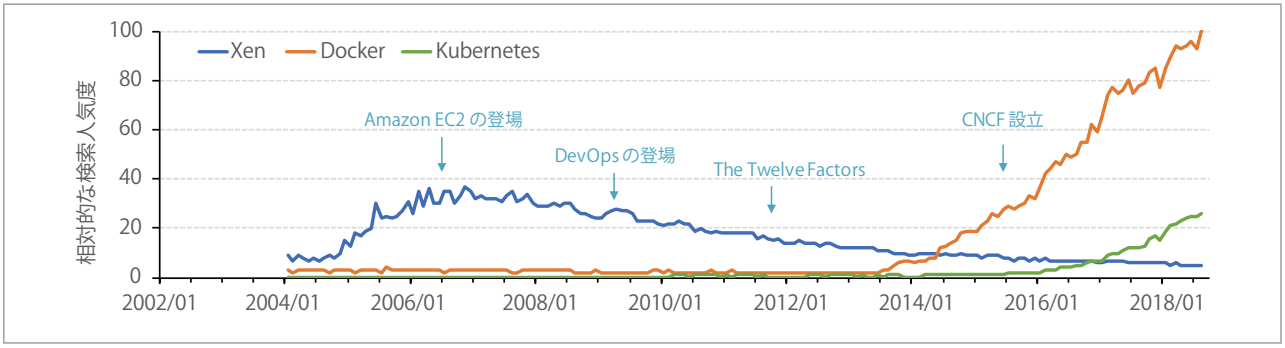

図 -1　Google トレンドでの各プロジェクトの相対的な検索人気度の推移

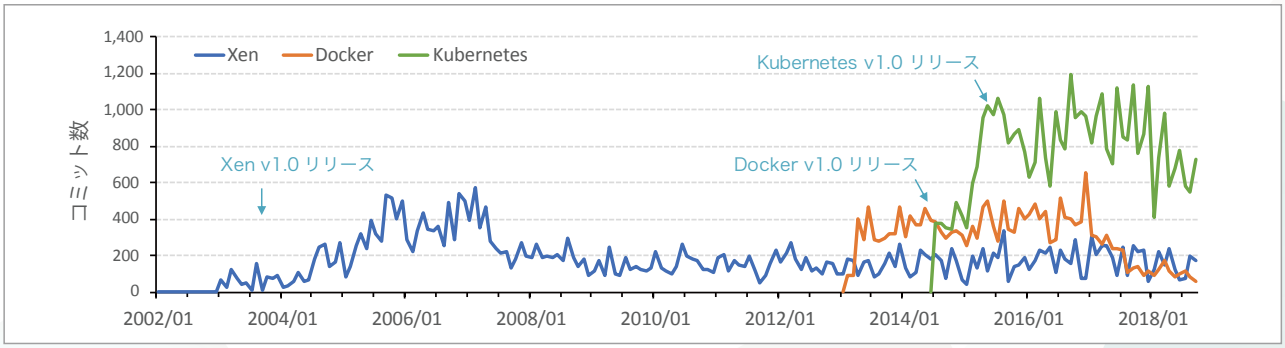

図 -2　各プロジェクトにおける月ごとのコミット数の推移

で動作しており，OS カーネルに対する脆弱性が影響することはもちろんのこと，悪意のあるコンテナがホスト OS カーネル経由でほかのコンテナに対してアクセスする可能性がある．VM であればハイパーバイザレベルでの仮想化によりこのような影響は発生しないため，セキュリティ面では VM に軍配が上がっていた．しかしながら，コンテナについてもセキュリティに対するニーズの高まりにより，セキュリティリスクをなくすためのさまざまなアプローチで実装されたコンテナランタイムが登場してきている．

ユーザ空間に隔離されたサンドボックス内でユーザのシステムコールをトラップして安全なシステムコールに変換するサンドボックスをコンテナに提供するアプローチ (Google の gVisor) や，セキュリティやパフォーマンス向上のために OS そのものをライブラリ化してアプリケーションに組み込むことでセキュリティやパフォーマンス向上を可能にするユニカーネル型のアプローチ (Nabla Containers) や，さらには，VM のように仮想化レイヤをより軽量化した形でコンテナに用意してコンテナ間の分離をハイパーバイザレベルで実現するアプローチ (Kata Containers) などである．セキュリティを担保しつつコンテナの性能を落とさないようにするコンテナランタイムは，今後大きく発展するエリアの 1 つであると考えられる．

サーバレス

ユーザから見えるコンピューティングリソースの抽象化がコンテナよりもさらに一歩進んだ先に，サーバレスコンピューティングが存在する．コンテナというプロセスそのものの運用すらもクラウド側に任せて，ユーザは純粋にサービスを実現するために必要な関数のみを実装する方式である．それらの関数は，たとえば，あるイベントの発生をトリガとして，必要に応じた回数だけ呼び出され，結果を生成することでサービスを構築する．実装する関数からは，もはや CPU やメモリ，プロセスといったものを意識する必要はない．もちろんここまで抽象化が進むと，実現できることにも大きな制約がかかるため，すべてのアプリケーションがサーバレスに移行することはない．アプリケーションやサービスを構築する 1 つのパーツとしてサーバレスを用いることで，ユーザはより迅速にサービスを立ち上げたりさらなるコスト最適化が可能となる．その一方で，サーバレスで実現可能なこと不可能なことを判断し，細かく分割されたマイクロサービスをコンテナを含めてどのように構築すればよいかについてを見極めることは，容易なタスクではない．クラウドネイティブの考え方やこのようなアプリケーション構築方法が一般に浸透していくにはまだまだ時間が必要である．

コンテナが広がっていく世界

本稿では，クラウドネイティブを体現する最も重要なピースの 1 つであるコンテナを，コンテナが登場してきた成り立ちや，コンテナ化されたクラウドが目指す世界，そして今後のコンテナに必要な技術的課題と将来展望という観点でまとめていった．クラウド上で実行される Web アプリケーションとしてのコンテナにフォーカスが当たることが多いが，実際には，クラウド上での実行いかんを問わず，今や多くのアプリケーションがコンテナ化されている．この流れは，クラウドに閉じたものだけではなく，エッジコンピューティング・IoT の世界や，HPC アプリケーションや基幹システムまで，さまざまな領域にコンテナ化の流れが浸透していくことが考えられる．コンテナとそれを取り巻く環境は，今まさに花盛りであり，今後も目が離せないエリアである．

参考文献

1) Beda, J. : Containers At Scale - At Google, the Google Cloud Platform and Beyond, GlueCon 2014 (2014).
2) Burns, B., Grant, B., Oppenheimer, D., Brewer, E. and Wilkes, J. : Borg, Omega, and Kubernetes, ACM Queue, Vol.14, pp.70-93 (online), available from, http://queue.acm.org/detail.cfm?id=2898444 (2016).

(2018 年 8 月 28 日受付)

■千葉立寛（正会員） CHIBA@jp.ibm.com

2011 年東京工業大学情報理工学研究科数理・計算科学専攻博士課程修了．博士（理学）．同年，日本アイ・ビー・エム（株）入社．東京基礎研究所にて，並列分散処理基盤や並列分散プログラミング言語の研究開発に従事．