@IT
atmarkIT

巨大 SIer のコンテナ・Kubernetes 活用事例

小林隆浩, 澤頭 毅, 海内映吾, 新井雅也,
野村総合研究所[著]

安定志向の NRI が
変化の激しい Kubernetes を推進する理由

さまざまな顧客のシステム開発や運用に関わる中で、NRI はコンテナ技術や Kubernetes に積極的に取り組んでいる。本連載の初回は NRI が抱える組織や文化の課題を整理し、Kubernetes に期待していることを紹介する。

小林隆浩，野村総合研究所（2020 年 09 月 24 日）

野村総合研究所（以後、NRI）のコーポレート・ステートメントは「未来創発」だ。顧客とともに新しいビジネスモデルを生み出そうとする姿勢を示している。これはシステム開発や運用などを提供する分野でも一貫しており、IT サービスの分野でも先進的な取り組みをいち早くプロジェクトに取り入れ、強い実行力をもって、存在しなかったようなサービスを生み出してきた。

しかし、NRI はシステムインテグレーター（SIer）と呼ばれ、慎重・安定などのキーワードを連想されることがある。例えば、こんな評判を聞くことがある。

> 大規模なプロジェクトを幾つも推進していそう

> 先進的な技術というよりは、安定した（枯れた）技術を得意としているように見える

> コスト効率を重視し、リスクの少ないプロジェクト管理が得意そうだ

これらの評価は間違っていないが、さまざまな技術を調査・評価してプロジェクトに適用し、顧客に価値を届けることも、SIer の仕事の醍醐味（だいごみ）の一つである。こうした考えから、現在注目度が非常に高い「コンテナ技術」や「Kubernetes」への取り組みも社内で積極的に行われている。

そこで今回は、NRI のさまざまなプロジェクトの形態と組織の特性をまず整理する。そして本連載第 2 回以降で、コンテナや Kubernetes を適用した事例を紹介する。

→目次に戻る

プロジェクトの分類と 3 つの事例

先述したように、NRI が扱うプロジェクトは多種多様であり、IT サービスに関連するものだけではない。筆者が所属する IT サービス関連部署でも、稼働しているプロジェクトはバイモーダルと定義されるように 2 つの種類に大きく分けられる。

バイモーダルとは

Gartner が提唱した、情報システムを「モード 1：変化が少なく、確実性、安定性を重視する領域」と「モード 2：開発・改善のスピードや “ 使いやすさ ” を重視する領域」に分類する考え方。モード 1 は基幹系業務である会計や生産管理・人事などを指し、モード 2 は収益の拡大を目指して顧客と直接接点を持つ業務などを指す。求められる組織体制・文化も異なり、モード 1 には「ITIL」など効率化を重視した厳密な規律、モード 2 には「DevOps」といわれる開発チームと運用チームの一体化によりアジリティを引き出す組織づくりが求められる。

NRI が得意としてきたのは、モード 1 の分野で、かつ大規模なプロジェクトである。しかし、こうしたプロジェクトは何より安定性が重視され、同時に導入・管理のコストを削減する意識が強く働くため、厳密な管理手法が用いられた結果として、新規技術の導入が阻害されているケースがあった。

このような先端技術への取り組みが難しいモード 1 のプロジェクトにおいても、これまで培ってきた設計、開発手法を生かしながら、Kubernetes を適用している。これは連載 4 回目で詳説する予定だ。

NRI ではモード 2 に当たるプロジェクトも多数の実績がある。それらの中からコンテナ技術をベースに顧客の新規ビジネスに必要なシステムを構築・運用した事例を連載の第 2 回で紹介する。同様のモード 2 のケースとして、社内で開発したプロジェクト支援ツールを Kubernetes で構築した事例を第 3 回で紹介する。

NRI における組織・開発文化の課題　Kubernetes の適用が目標とするもの

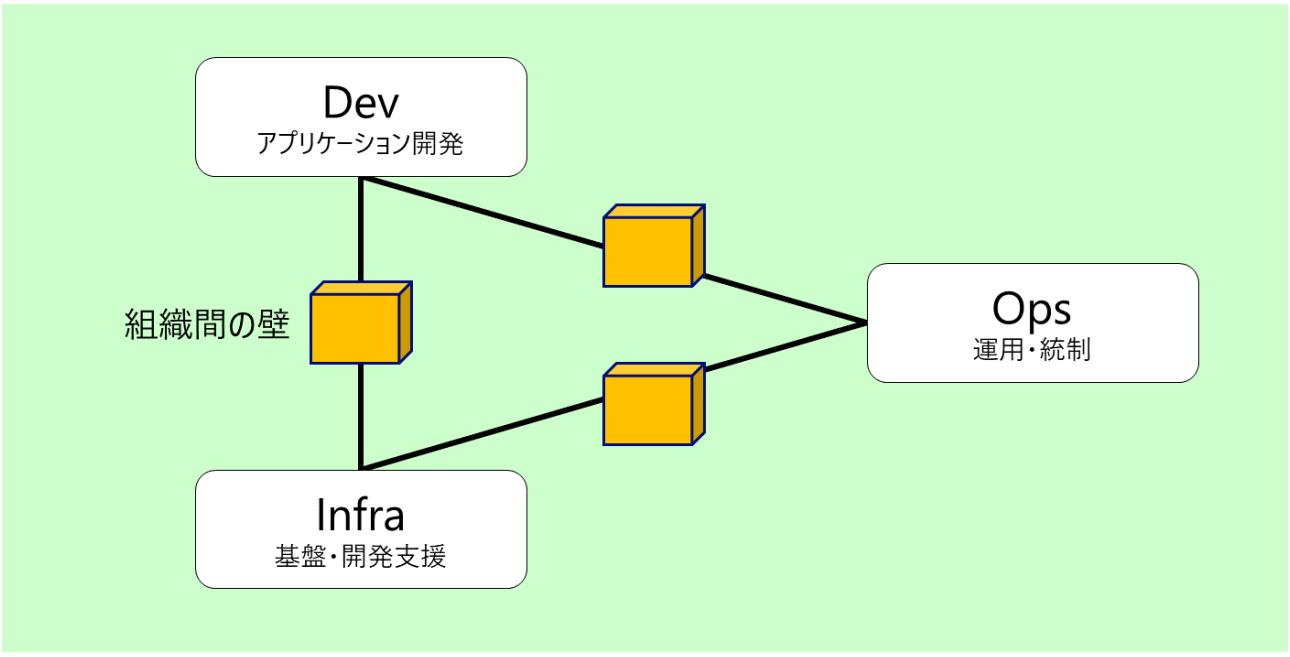
さて、説明してきたように、NRI が得意とするプロジェクトの形態はモード 1 であり、大規模な開発プロジェクトであっても標準化を通じて効率的に推進し、顧客ビジネスを支える安定したシステム運用につなげてきた。その中で NRI の組織は、長い時間をかけて得意とする大規模向けの開発・運用プロセスに最適化されてきたといえるだろう。

組織の最適化とはつまり、機能の分化である。システム開発・運用において必要な機能を、以下のように分けている。

1. Dev：アプリケーション開発
2. Infra：基盤・開発支援
3. Ops：運用・統制

それぞれが専門技術をプロジェクトに提供することで、多数の顧客から寄せられるさまざまなリクエストに応えることが可能となっている。例えば、Infra には「ハードウェア・ソフトウェアの発注を行い自社データセンター（DC）に組み上げるチーム」や「アプリケーションの開発を効率化するフレームワークを提供するチーム」が含まれ、内部で専門領域に応じて細分化されている。

こうした組織構成は NRI の固有のものではないが、そこには大きな課題が存在する。それが「組織間の壁」である。



組織間の壁

→目次に戻る

Infra の担当者はアプリケーションの設計を理解せず、同様に Dev の担当者はフレームワーク以下の仕組みに踏み込まないなど、本来必要なコミュニケーションが壁によって阻まれた好ましくない状況が発生し、課題となっていた。

そうした壁を打ち壊し、NRI が得意とするモード 1 だけではなく、デジタルトランスフォーメーション（DX）の担い手としてモード 2 のプロジェクトへ積極的に取り組むために、社内でもさまざまな変革が行われている。その一つが Kubernetes に代表されるクラウドネイティブ技術の適用である。クラウドネイティブ技術の活用で、アプリケーション開発者はよりパワフルになり、運用は効率化され、結果として壁を取り除かれた組織全体が活力を取り戻す。そうしたゴールを目指し、各プロジェクトの模索が続いている。

今後の連載各回のテーマを、バイモーダルにおけるプロジェクトの分類、そしてプロジェクト推進体制の違いで生じる組織間の壁の有無で整理すると下表のようになる。

	連載タイトル	モード	“組織間の壁”の有無		
			Dev - Infra	Dev - Ops	Infra - Ops
第2回	Fintech系サービスにおける コンテナ適用の背景とポイント	2	なし	あり	あり
第3回	NRI発の開発支援ツールにおける Kubernetesの適用事例	2	なし	なし	なし
第4回	Kubernetesによるマルチテナンシーの実現と開発の効率化	1	あり	あり	あり

ここからは第 2 回以降で紹介するプロジェクトのアウトラインを少しだけ紹介していこう。

顧客の新規ビジネスを創り出すプロジェクト

NRI が多くの基幹システムを長年支えてきた結果、顧客の重要なビジネスパートナーとして認知され、新規ビジネスの立ち上げでも相談いただくことがある。最近はユーザー側の技術理解も非常に高く、コンテナ技術や Kubernetes のメリットを顧客と NRI の双方が理解した上でプロジェクトをスタートできる例も増えてきている。

このプロジェクトは、モード 2 として、新規システム構築で開発スピードと変化許容性を重視してコンテナ技術を採用し、少数精鋭の開発チームでサービスリリースまでを走り切ったケースだ。こうした小規模プロジェクトでは厳密なプロジェクト管理の必要性が薄く、NRI 内のさまざまなプロジェクトと比較して先端技術の導入が容易というメリットがある。

このプロジェクトではパブリッククラウドからコンテナ技術、プラットフォーム、アプリケーション開発と業務知識までを兼ね備えたエキスパートチームを組成して、前述の Dev と Infra の壁を乗り越えた。しかし、業界固有の規制要件により、Ops との間には壁が存在していた。これらをプロジェクトのアジリティを損なわずに、どのように回避したかについても焦点を当てる予定である。

社内サービスの開発における Kubernetes 適用

NRI はチームのコミュニケーションを促進するサービスとして「aslead」を展開しており、サブプロジェクトとして、開発チームに必要な継続的インテグレーションのパイプラインを提供する「aslead DevOps」がある。

aslead DevOps は、以前から抱えていたさまざまな課題の解消に向けて、各コンポーネントを適切に分割してコンテナ化し、マネージド Kubernetes サービスにデプロイして管理する構成にした。これにより「Pod」「Node」レベルのオートスケールも実現し、開発ピーク時に発生する密度の高いパイプラインのリクエストにも柔軟に対応が可能となっている。

こちらもモード 2 のプロジェクトであるが、自社向けのサービスとして Dev ／ Infra ／ Ops を全て統合した組織で管理し、Kubernetes がもたらすメリットの最大化を実現している。

また、若いエンジニアたちが組織の壁に縛られずに活動し、「CKA（Certified Kubernetes Administrator）」「CKAD（Certified Kubernetes Application Developer）」などの資格取得を通じて知識を強化するとともに、外部コミュニティーからベストプラクティスを持ち帰り、新規技術のプロジェクト適用を積極的に推進した。

この事例はコンテナ技術、Kubernetes 技術の獲得と社内共有を目的とした側面もある。実績をもとに他プロジェクトへ展開が可能となったのはもちろん、ステートフルなワークロードへの Kubernetes の適用可否やマルチクラスタ対応など “ その先 ” のノウハウ獲得に向けて、aslead は前進している。

大規模な基幹プロジェクトにおける Kubernetes 適用

NRI が得意とする大規模プロジェクトで Kubernetes を適用した事例だ。こちらは典型的なモード 1 であり、対応する組織構成も従来通りで壁が随所に見られる状況だった。それに加えて、前述の Infra 組織の中でも役割が分かれ、Kubernetes に関連するチームとして基盤チームと開発支援チームが生まれた。

基盤チームは、Kubernetes クラスタ管理者＝仮想サーバの上に Kubernetes のクラスタを構築・運用するという役割を担い、Kubernetes のノードやネットワーク、そして Namespace の設計・構築・テストを行った。もう一方の開発支援チームでは、Kubernetes の Namespace 内管理者という役割を担い、Kubernetes リソースをどのように使うか標準化し、必要であれば定義ファイル（YAML）のひな型を作り、Dev チームに展開した。そして、コンテナをビルドするパイプラインの構築・管理も開発管理チームが担当した。

では、アプリケーション開発チームではどのように開発を行ったのだろうか。そして、それは Kubernetes の導入により何か変化があったのか。

今回の対象プロジェクトでは生産性を重視し、アプリケーションの設計・開発は大きな変化なく行えることを方針とした。つまり、Dev と Infra 間の壁をそのまま残し、同時に Kubernetes などの技術理解の必要性も壁の内に閉じ込めた。こうしたアプローチは NRI で過去に蓄積されてきたアプリケーション設計やフレームワークを最大限に生かすためだったが、当然課題も存在した。それらの詳細は連載第 4 回で記す予定だ。

まとめ

NRI はモード 1 のような安定性が求められる大規模プロジェクトにおいて、これまで多くの成功をおさめ、顧客の信頼を勝ち得てきた。しかし現在、異なる形でのプロジェクトへの参与を望まれている。それは DX の潮流における革新の担い手として、モード 2 のプロジェクトに寄り添い、スピード感を持って歩むことだ。

その実現には NRI が長年培ってきた組織と開発文化の変革も不可欠だ。コンテナ技術や Kubernetes を変革の一助として捉え、組織間の壁を乗り越えようとする動きは今回紹介する 3 つの事例に限らず、今後も多くのプロジェクトに拡がっていくだろう。

NRI におけるコンテナ技術や Kubernetes の活用事例が、自らの組織・文化に同様の課題を抱える読者にとって、変革を始める一助になれば望外の喜びである。

なぜ金融系プロジェクトで先進のコンテナ技術を選択したのか

NRI のコンテナ・Kubernetes 活用事例について紹介する本連載。第 2 回は FinTech サービスをクラウドやコンテナで支援した事例を紹介する。

新井雅也 , 小林隆浩, 野村総合研究所（2020 年 10 月 23 日）

金融系サービスでも顧客体験を改善する迅速さは不可欠

「金融」と聞くと、勘定系処理や外部システムとの接続、バックオフィス業務などを思い浮かべる読者も少なくないだろう。これらのシステムでは、「求められるシステム品質が高く、ドキュメントは重厚に整備、管理され、大規模な工数が必要なプロジェクト」という点を想像するに難くない。野村総合研究所（以後、NRI）はインターネットバンキングや証券業の大規模共同利用型サービスを構築、運用しており、まさに NRI が得意とする領域でもある。

こうした大規模プロジェクトのみならず、NRI はクラウドネイティブ技術を活用した「FinTech サービス」の共創にも奮闘している。FinTech サービスといえば、家計簿アプリやキャッシュレス決済、資産運用アプリなど、スマートフォンアプリケーションを軸にしたサービスが主流だ。今や私たちの生活に溶け込み、日常の中で自然に利用されている。故に、利用時にストレスを与えないユーザー体験が求められる。利用者からのニーズや不満を日々くみ取り、素早く提供、改善することがサービスの価値を高める上で極めて重要だ。

FinTech サービスに取り組む上で、システムインテグレーター（SIer）の立場においても、「お客さまのお客さま」（サービスを利用するエンドユーザー）に届ける体験価値をより強く意識しながら取り組むべきだと考えている。スコープを小さくして価値を素早く届けるという意味では、FinTech サービスは**継続的なアジリティを重視したアプリケーション開発が求められる**と言える（もちろん、FinTech サービスに限った話ではないが）。

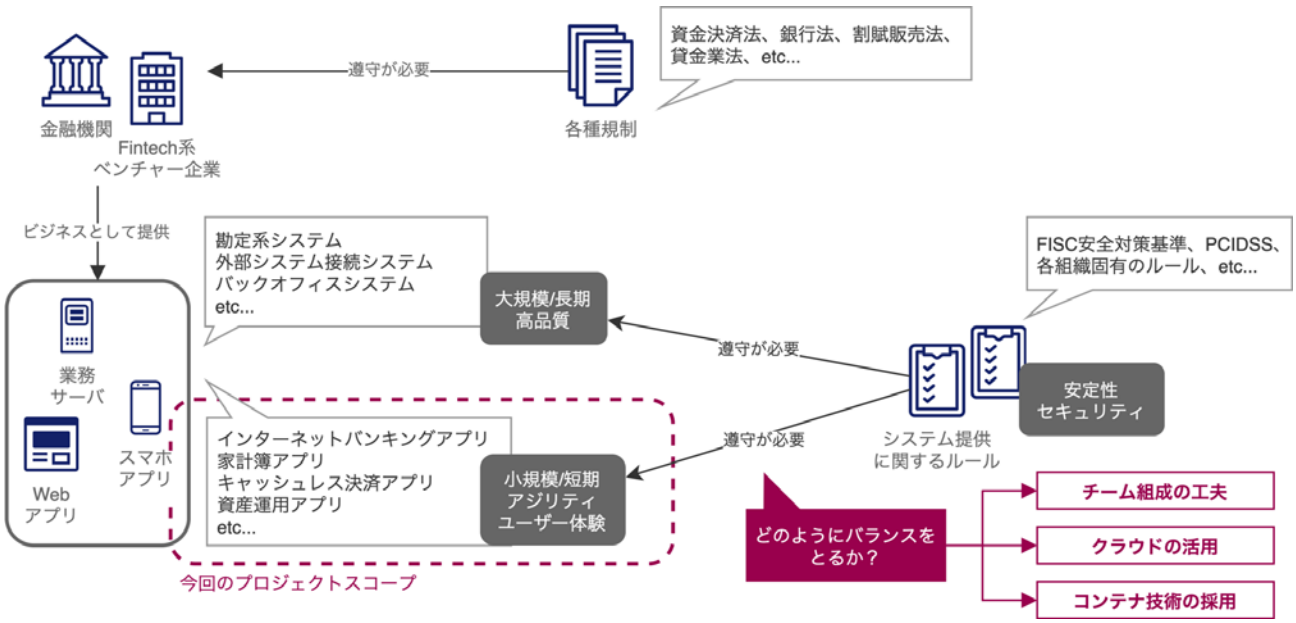
→目次に戻る

業界標準規格、監査、ガバナンスルール――金融系サービスの高いハードル

FinTech サービスのみならず、金融サービスは私たちの生活インフラを担っている。故にシステムの安定性や高いセキュリティ維持のためにさまざまなルールが存在する。

安全なシステム構築の指針として、金融情報システムセンター（FISC）は「金融機関等コンピュータシステムの安全対策基準（FISC 安全対策基準）」を提示している。クレジットカード事業に求められるグローバルなセキュリティ標準規格「Payment Card Industry Data Security Standard（PCI DSS）」への準拠も一つの例だ。他にも、金融機関や FinTech 企業が自組織内で定めたガバナンスルールへの準拠や、監査法人などを通した準拠状況の検査が求められるケースも珍しくない。「必要なドキュメントの整備」「開発ルールの策定」「ソフトウェアの適切な設定」「システムの実装を証明するための証跡提出」など、実施すべき項目は多岐にわたる。昨今では、組織ごとにリスクを定義、評価してそのリスクごとに対処する「リスクベースアプローチ」が採用されつつあるが、他の業界と比較して金融サービスは不正取引や個人情報漏えいなど重要なセキュリティリスクが多く内在するため、対応範囲が広くなる傾向が強い。

これらの内容を踏まえると、**「アジリティを重視した継続的なアプリケーション開発」と「コンプライアンス、ガバナンスの確実な準拠」のバランスをどのように実現するか**、という点が FinTech サービスの継続的な発展に必要なポイントだと捉えられるだろう。



金融系サービスの全体像

両者を両立させるべく、筆者のプロジェクトでは、「チーム組成の工夫」「クラウドの利用」「コンテナの活用」という 3 つの観点で取り組みを検討することにした。

チーム組成の工夫：Dev と Infra の融合

本連載の初回で紹介した通り、Dev・Infra・Ops の組織に分かれてそれぞれが責務を持ち、システムを作り上げていくことは NRI の得意とする開発スタイルであり、NRI で実施するプロジェクトのチーム組成における大半を占める。一方で、組織間の責務に対する曖昧さが生まれたり、責務の調整に関するオーバーヘッドが生まれたりすることも少なくない。これに起因して「アプリケーション開発のアジリティが最大化されないのではないか」という懸念もあった。

筆者が担当したプロジェクトでは、**少数のエンジニアが Dev と Infra の両方を一手に担う、もしくは Dev と Infra で 1 つの混成チームを結成する**ことで懸念の払拭（ふっしょく）を図った。小規模かつ短納期というプロジェクトの特性から、利用者に対して最も価値のある箇所を見定めてスコープを限定することで、必要となる技術スキルや学習コスト、開発ボリューム、失敗に対する影響を最小限にできる。

また、開発要件が変更された場合もチーム間の調整負荷も下がるため、変化に対して柔軟に振る舞えるだろうと考えた。

クラウドの利用：マネージドサービスの活用

2013 年に NRI は Amazon Web Services（AWS）のプレミアコンサルパートナーとして認定を受け、多岐にわたる業界のお客さま向けに **AWS を活用したシステム構築を支援してきた。これらの経験を生かし、AWS のマネージドサービスを上手に活用することでインフラ層の抽象度が上がり、Dev・Infra チームとしてアプリケーション開発寄りのスコープにより注力できる**と考え、AWS を利用することにした。

AWS は FISC 安全対策基準や PCI DSS の準拠に向けたガイドラインを提供しており、AWS 利用者がこれらの準拠に向けてお互いどのような責務で臨むべきかを整理している。マネージドサービスの活用により、**各種ルール準拠に必要なハードウェアから OS、ミドルウェアまで設計、運用の一部を AWS 側に委譲**できるため、よりアプリケーション開発に注力できると考えた。

コンテナ技術の採用

チーム組成とクラウド活用に加えて、本プロジェクトではコンテナ技術を積極的に活用する方針とした。コンテナ利用のメリットの一つは「可搬性」だ。開発環境でビルド、テストされた同一イメージを本番環境でも利用でき、従来の手動オペレーションによる各環境向けのビルド&リリースなどと比較して、稼働環境ごとの差異に伴う考慮やリスクを減らすことができる。

一方、アプリケーションデプロイやリソースの割り当て、負荷分散や可用性など非機能面や運用面を考えると、コンテナオーケストレーションが必要不可欠だ。オーケストレーションツールの選択肢である「Kubernetes」は、今やクラウドネイティブの中心的存在だ。クラウドネイティブ技術のオープンソースプロジェクトを推進する Cloud Native Computing Foundation（CNCF）の調査レポートによると、2018 年のコンテナオーケストレーションを利用する 83%のユーザーが Kubernetes を利用している、との結果を公表している。これに対し、フルマネージドで Docker コンテナの実行、管理を提供する「Amazon ECS」（Amazon Elastic Container Service）は全体の 24%だが、今回筆者のプロジェクトは Amazon ECS を選択した。その理由は幾つかあるが、本稿ではこの選択のプロセスをもう少し掘り下げて紹介する。

なぜ Kubernetes ではなく、Amazon ECS を採用したのか？

一つは AWS をプラットフォームとして作り上げていく際の考え方だ。Amazon ECS を含め、AWS では各サービスをブロックと見立てて組み合わせる思想があり、組み合わせたサービス間はスムーズな連携が可能だ。この点は、今後 AWS を主軸としてサービスを拡張する上で開発アジリティを高めていく際の重要なポイントになる。Kubernetes とその他多数の CNCF 関連ソフトウェアとの組み合わせと比較すると、オーケストレーション層における機能面の自由度は限定されるものの、今回のサービス要件では Amazon ECS で充足すると考えた。

Amazon ECS の採用は運用面でも大きなメリットがある。Kubernetes はおおよそ 3 カ月に 1 度マイナーバージョンがリリースされており、AWS でも同じ流れが踏襲されている。プロジェクト対応当時、マイナーバージョンは最初のリリースから約 9 カ月間サポートされていたため、定期的なバージョンアップ運用が必要だった（現在は約 12 カ月間サポートに延長されている）。Kubernetes のバージョンアップ後は稼働するソフトウェアやアプリケーションへの影響確認が不可欠だ。Amazon ECS ならこれらの運用を AWS 側が透過的に実施してくれる点でメリットが大きい。

また「サービスリリース後に誰が Kubernetes を運用するのか」という点でも課題があった。Kubernetes を選択した場合、Kubernetes レイヤーでの金融規制の準拠に必要な運用スコープが増えてしまう。NRI 内の Ops チームを巻き込むスコープが広くなり、運用の受け入れまで含めると、引き継ぎのリードタイムが生まれ、サービスインへの支障も懸念された。一方、Amazon ECS は Kubernetes と比較して AWS 利用者側の責務範囲を抑えることができる。故に運用スコープを小さくできると期待できたことから、Amazon ECS を採用するに至った。

コラム：Kubernetes を選択するメリット

今回選定しなかった Kubernetes にも良さがたくさんある。一例を挙げると、Kubernetes はコミュニティーグループの活動が非常に活発であり、IT エンジニア間での熱量も非常に大きい。Kubernetes をはじめとするクラウドネイティブ技術に関する勉強会が多数のコミュニティーを通じて日々行われ、モダンアーキテクチャに対する共有やディスカッションが日々なされている。所属している会社、組織の枠を超えて、IT エンジニアとしてのモチベーションが高く保てる技術の一つだと考えている。Kubernetes を利用していることが企業、サービスに対する一種のプレゼンスとなっている点もあるのではないかと思うこともある。

また CNCF を軸としたソフトウェアの活用により、インフラとしての自由度を高めることが可能だ。Amazon ECS は AWS の仕様範囲内の利用に限定されるが、Kubernetes は求められる要件ごとに CNCF 関連ソフトウェアを組み合わせることで、より柔軟なアーキテクチャを構成できる。オープンソースソフトウェア（OSS）が多く、オンプレミス環境や他のパブリッククラウド環境への移植性も高い。

今回紹介した事例は、「FinTech サービス × 規制ルールの準拠」というプロジェクトの特色を考慮し、Amazon ECS を採用する方針とした。一方で NRI においても Kubernetes を積極採用する事例が増えてきている。至極当たり前のことだが、Amazon ECS にせよ、Kubernetes にせよ、プロジェクトの特性と技術の特性を理解し、判断軸を設けて選択するプロセスを踏むことが重要だろう（※プロジェクト特性次第では、一人の IT エンジニアとして筆者も Kubernetes の世界に飛び込んでみたいと考えている）。

おわりに

短納期かつユーザーファーストな FinTech 領域の課題に先進のコンテナ技術を採用した NRI の事例を紹介した。今後も、組織組成の在り方に柔軟性を持たせたり、パブリッククラウドの積極的な活用や Cloud Native 技術の採用による成功事例を積み重ねていきたいと考えている。

本稿の内容が同じような課題に直面している読者のヒントにつながれば大変うれしい限りである。

なぜ Kubernetes を採用するのか　2000 人の開発者に提供するサービスで得られた知見と課題

NRI のコンテナ・Kubernetes 活用事例を紹介する本連載。第 3 回は開発者の QCD（Quality、Cost、Delivery）を向上させることを目的とした開発支援サービスに Kubernetes を適用した事例を紹介する。

海内映吾 , 小林隆浩,　野村総合研究所（2020 年 11 月 27 日）

Kubernetes 採用の背景とメリット

野村総合研究所（以後、NRI）では、ここ数年で Kubernetes を利用するプロジェクトが増えてきた。本記事で取り上げる自社サービス「aslead」や、社内のプライベートクラウドに構築した「OpenShift」でも Kubernetes を活用している。Amazon Web Services（AWS）が提供する「Amazon EKS（Amazon Elastic Kubernetes Service）」（以後、EKS）を本番環境としてアプリケーションをデプロイしているプロジェクトも存在する。

Kubernetes 採用の背景は 2 つある。一つはコンテナ実行が可能なインフラを構築することでアプリケーション開発やテストをするための環境をすぐに提供できるようにすること。もう一つは、アプリケーションをコンテナ化することで、インフラストラクチャの EOL（End Of Life）に伴うアップデートをアプリケーションに影響を与えることなく実現可能にすることだ。

コンテナの可搬性は、アプリケーションのデプロイ頻度を高める CI（継続的インテグレーション）／ CD（継続的デリバリー）との親和性が高い。Kubernetes を導入したプロジェクトが増えるにつれ、Kubernetes リソースを手軽にデプロイしたりテストしたりするための CI/CD の需要も高まりつつある。

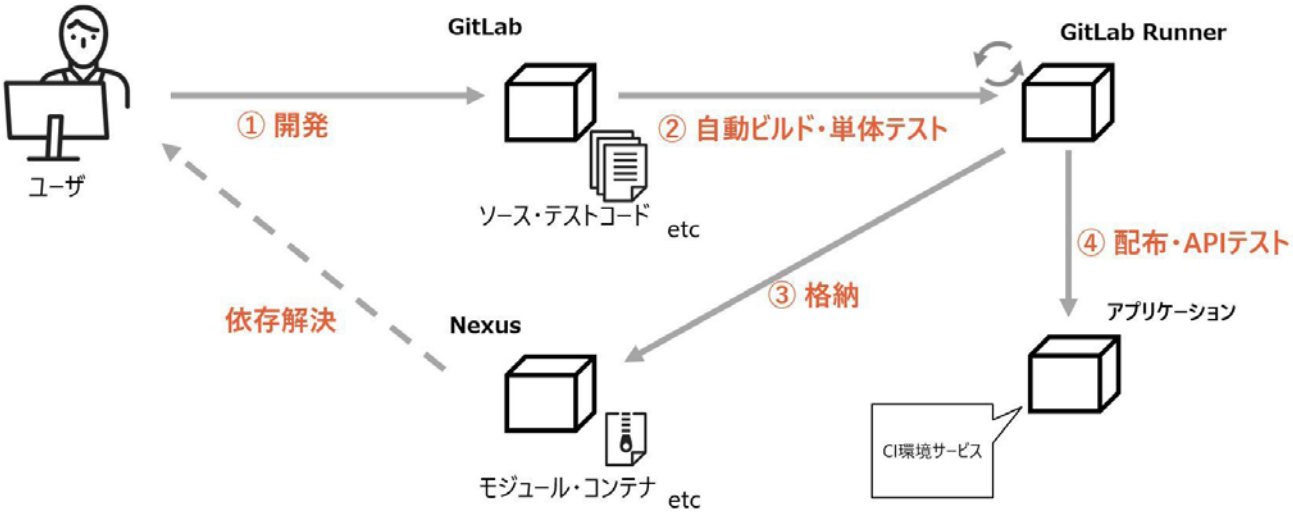
NRI 内のプロジェクトでは本番環境と CI 環境を別にする要件が多く、EKS の場合は NRI 内のセキュリティポリシーに準拠しつつ手軽に CI を実行可能な環境を用意することにしていた。その結果、Kubernetes のプロジェクト導入事例が増えるとともに、Kubernetes リソースをデプロイして手軽にテストするための CI 環境の需要が高まっていった。

社内向け開発支援ツールの提供

NRI は 2018 年から社内向けの生産性向上サービスとしてコミュニケーションツールやファイル共有ツールなどを提供する aslead を内製している。利用者は独自で環境構築をする必要がなく、サービスによってセキュリティが担保される。aslead の中でも開発を支援する「aslead DevOps」は 2020 年 10 月時点で利用者が 2000 人を超えている。

aslead DevOps とは

NRI の開発現場から生まれた開発管理製品。ソースコードを管理する「GitLab」や資材を管理する「Nexus」、ソースコードを静的解析する「SonarQube」に NRI 独自のプラグインを加えた機能を持つ。アプリケーションの Docker イメージを資産として格納し、それを Kubernetes リソースとしてテストするための環境も提供している。開発方式をモダナイズしたいという開発者の要望に応え開発され、利用が広まっている。



aslead DevOps による開発イメージ

aslead DevOps 展開までの過程で、社内の開発工程をモダナイズさせるためにどのようなプロセスで構築が進められたのかを説明する。

→目次に戻る

Dev、Ops、Infra の結束

NRI における組織、開発文化の課題として Dev（アプリケーション開発）、Ops（運用）、Infra（基盤）に組織間の壁があることは第 1 回に述べた通りだ。近年のクラウドネイティブに関する動向を見ても「まずは組織を変えることが重要だ」と述べられているケースが多い。これはまさにその通りで、企業特有の組織文化やコミュニケーションを手軽に取れない環境が、サービス展開のスピードを高めることが難しくさせてしまう現実がある。

そこで aslead では Dev、Ops、Infra で組織間の壁が存在せず同じチームとして動いている。aslead の開発過程は NRI の他プロジェクトとは毛色が異なり、Dev は OSS（オープンソースソフトウェア）に導入する NRI 独自のプラグイン開発やデプロイを担当し、Ops は障害対応や QA 対応、Infra はプライベートクラウド環境での必要リソースの構築、管理を担当した。

最初の組織体制では社内のセキュリティポリシーを満たすプライベートクラウド環境を別チームに依頼して提供してもらっていたが、後に統合する形となった。この組織構成となったのはもともと人数の問題があったためであり、覚える業務が多くなるというデメリットもあるが、提供するプロダクトを全員が正しく理解できるというメリットもある。

Kubernetes 採用に至るまで

サービス開始当初は Kubernetes を利用せずに運用をしていたが、サービスの運用コストが課題として挙がるようになった。そこで、運用コスト削減を第一の目標とし、副次的にリソース管理の負担軽減や自己回復性の担保を実施すべく、Kubernetes を採用した。Kubernetes は自動復旧やリソース割り当ての容易さに加え、多くの便利な機能が OSS として提供されるエコシステムが確立していることも採用の背景にある。

所属チームでは「Cloud Native Computing Foundation」（CNCF）の動向に注目しており、GitOps を実現する「Argo CD」や Chaos Engineering を実現する「Litmus」などを内部で調査検証している。顧客を技術支援する際に効率良く展開できるようキャッチアップを行い、勉強会や Wiki などで共有している。このような OSS をすぐに利用できるのも CNCF で I/F 仕様の標準化が進められ、プラガブルな利用が可能な Kubernetes ならではといえる。

Kubernetes は開発スピードが早いため、それに追随することでさまざまな機能を利用できる。また、オンプレミスに Kubernetes クラスタを構築するか、各種クラウドサービスのマネージド Kubernetes サービスを利用するかなど自分たちで構築場所を決められる自由度もある。

EKS での運用

当プロジェクトの Kubernetes 環境には EKS を採用している。運用コスト削減を目的としていたため、AWS が提供するマネージド Kubernetes サービスが最適と判断した。テスト環境を提供する上で、Kubernetes のコントロールプレーンに負荷がかかることも想定されていたため、高可用性や AWS リソースとの親和性も総合的に考慮した。

EKS では Kubernetes のマイナーバージョンのリリースが 3 カ月ごとに行われ、最新 3 バージョンがサポート対象となる。Kubernetes について「バージョンアップに追随しなくてはいけない」ことがマイナスポイントとして挙げられることもあるが、Kubernetes のバージョンアップに合わせてエコシステムは開発が進み、機能も増えていくためそれに追従できる環境を構築することが大切だと考えている。

ここからは EKS で運用するに当たって得られた恩恵を紹介する。

シングルクラスタ構成によるコスト削減

以前は利用プロジェクトごとに Application Load Balancer（ALB）で窓口を用意するようにしていたが、コスト削減のために 1 つのクラスタにつき ALB が 1 つで済むようにリバースプロキシを適切に配置したり、複数のプロジェクトがある場合は Namespace で区切ったりすることで、EKS のクラスタを無駄に増やさずコスト効率を高められた。

オートスケールによる可用性向上

EKS では「Cluster Autoscaler」や「Prometheus Adapter」という機能により Node と Pod をオートスケールさせることができる。Node のオートスケール条件は決まっているが、Pod に関しては自由に設定できるのでアプリケーションごとに設定することで適切なスケーリングができる。

適切なマニフェスト整備によるリソース管理の負荷軽減

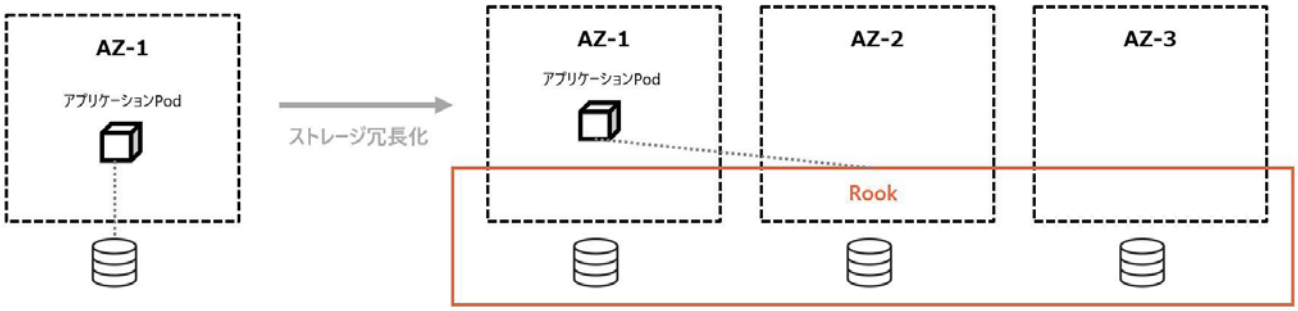
アプリケーション単位で「Helm Chart」を持っておくことで、複数のコンテナが 1 つのアプリケーションに含まれる場合でもリソース設定を容易にしている。利用者にはテスト環境を提供しており、デフォルトのリソース設定では不足に思う利用者もいた。Helm Chart の設定ファイルを編集してデプロイまでにかかる時間はわずかなため、利用者の意思で設定変更を素早くできるようになった。

また、EKS と AWS の製品との親和性もここ数年で上がってきているのも実感している。以前は「IAM Roles for Service Account」や「AWS Timestream」が存在していなかったため、それらを補うために自分たちで実装する場面があった。「CloudWatch Container Insights」の機能も充実してきており監視ツールもマネージドサービスに任せることが可能になってきている。

今後の展望

Kubernetes を運用して恩恵が得られた部分は複数あるが、改善の余地も同様に複数出てきた。

まず、ステートフルアプリケーションのスケーラビリティをどう確保するかが挙げられる。AWS の場合、PersistentVolume に「Amazon EBS（Amazon Elastic Block Store）」を利用すると ReadWriteOnce の制約によりマウントさせる Pod は 1 つだけに限定されマルチ AZ で冗長化できない。これを解決する方法として「Amazon EFS（Amazon Elastic File System）」などの共有ストレージの利用やストレージオーケストレーター「Rook」の利用が考えられる。もちろん、ただ冗長化すればよいものではなく性能など観点を広げて考えるべきである。当プロジェクトでは Amazon EBS を使用しており、Rook を用いた冗長化を検討している。



Rook を用いたストレージ冗長化のイメージ

次に、Kubernetes のエコシステムを柔軟に取り入れられていない点がある。本番環境には Helm を用いてデプロイしているが、まだ GitOps を導入できていないために本番環境での手作業が必要になっている。GitOps の導入はセキュリティ強化、視覚化、複数クラスタ管理などあらゆる点でメリットがあるので、Argo CD をはじめとするエコシステムを利用してこの問題を解決することを検討している。

このように、Kubernetes の運用を自分たちで実施することで Kubernetes を用いた技術支援をする際も知見として生かせられるようになった。また、若手メンバーは「CKA（Certified Kubernetes Administrator）」や「CKAD（Certified Kubernetes Application Developer）」といった外部資格を取得したり外部コミュニティーの勉強会に積極的に参加したりして、サービス展開に役立てている。

まとめ

NRI での Kubernetes 適用事例の一つとして、aslead DevOps への導入事例を紹介した。Kubernetes を採用したことでサービスの管理コスト削減や可用性の向上を実施できたと同時に改善の余地も判明してきている。今後は aslead DevOps の一般提供に向けて注力し、Kubernetes を推進する存在を目指して活動していく所存だ。

本記事が Kubernetes の採用を検討している読者の皆さまの参考になれば幸いである。

大規模基幹システムを「マルチテナンシーKubernetes」で構築、そのメリットと悩ましい課題

NRI のコンテナ、Kubernetes 活用事例を紹介する本連載。最終回はマルチテナンシー構成を活用した大規模な基幹システム開発の事例を紹介する。

澤頭 毅 , 小林隆浩, 野村総合研究所（2021 年 01 月 12 日）

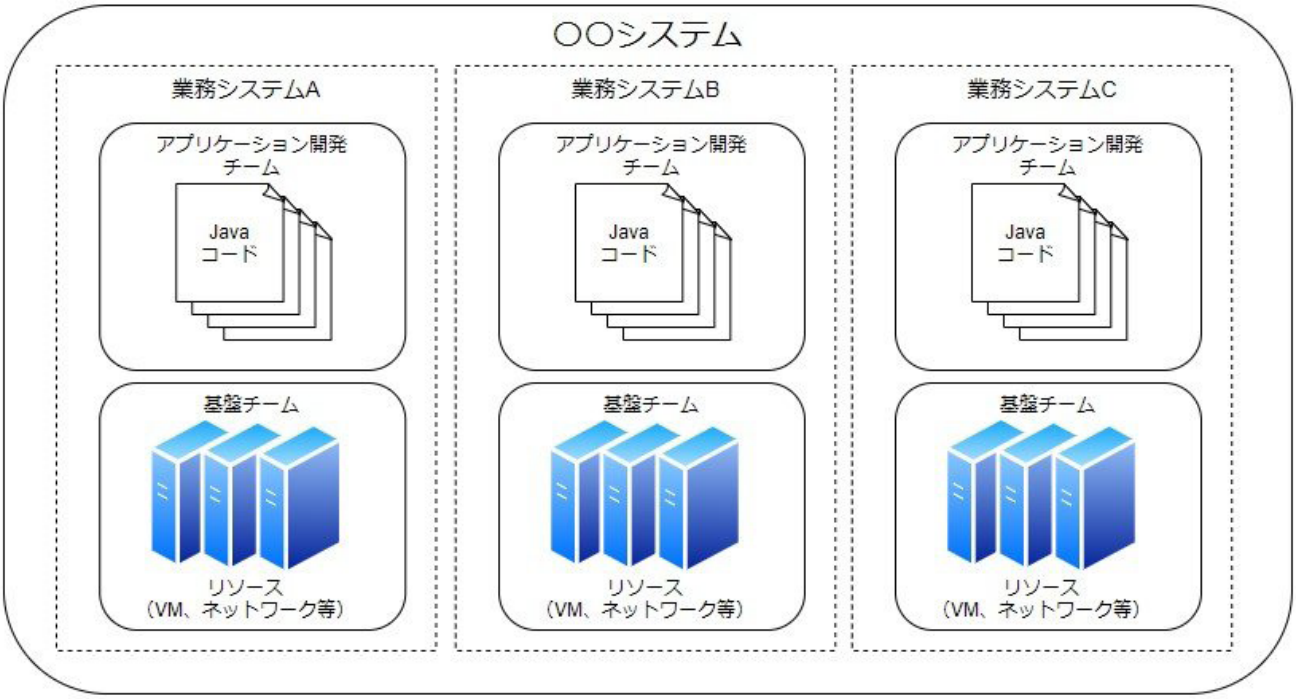
年々増加する設計コストが課題に

2020 年現在、ソフトウェアやサービスの開発において、開発速度の向上やリリースサイクルの短期化はより強く求められるようになっている。野村総合研究所（以後、NRI）でも「アジャイル」や「DevOps」といった開発手法に加えて「マイクロサービスアーキテクチャ」といった設計思想が現場になじみつつある。本連載のテーマでもある、コンテナを効率的に運用するためのオーケストレーターである「Kubernetes」の普及も進んでいる。

第 2 回と第 3 回で述べたように、小規模チームによるコンテナ活用や DevOps チームによる Kubernetes の利用事例はあるが、大規模な基幹システム開発において Kubernetes を導入する事例は少ない。今回は NRI のクラウド環境における「大規模基幹システムをマルチテナンシー Kubernetes に導入した事例」を紹介する。

大規模な基幹システムの開発においては、業務知識やアプリケーション開発のスペシャリストである「アプリケーション開発チーム」と、ネットワークの設計やコンピュートリソースの整備を担う「基盤チーム」に分かれてプロジェクトを進めることが一般的だ。大規模である分、業務が多岐にわたっているため、アプリケーション開発チームはそれぞれの担当に分かれており、独立した開発体制を持っている。

従来のシステム開発におけるアーキテクチャの設計は、システムごとにそれぞれのチームが個別に検討、対応することが多く大きな負担になっていた。自社のクラウド環境でシステムを構築する場合、縮退運用や一時的なアクセス増加に対処するための余剰リソースを常に用意しておく必要もある。リスクを十分に考慮した上で、システムごとの余剰リソースをいかに有効活用するかが重要な設計要素になった結果、設計や見積もりにかかるコストも高まっていた。



こうした状況を改善するには、システム構築を進める上で下記要件を満たす環境が必要だと考えた。

- 自社クラウド環境で、コンピュートリソースの共有をより柔軟に実現する
- 各アプリケーション開発チームの実行環境が高い分離性を持つ

この要件を満たす一つの解として NRI が検討したのが、マルチテナンシー Kubernetes の導入だった。

→目次に戻る

マルチテナンシー Kubernetes をどう実現？　NRI の場合

Kubernetes には、同一の物理クラスタ上に複数の仮想クラスタを構築できる「Namespace」と呼ばれる機能が存在する。基盤チームは Namespace を活用することで、Kubernetes クラスタを 1 つ構築して、アプリケーション開発チームごとに分離された実行環境を提供できると考えた。

しかし、Namespace を分割するだけでは、特定のコンテナのせいでコンピュートリソースが枯渇したり、誤操作で他の Namespace にアクセスしてしまったりする可能性がある。そのため、複数のアプリケーション開発チームで構成される大規模システムを運用するには十分な機能とはいえない。Namespace で環境を分割する際、考慮すべきポイントになった。

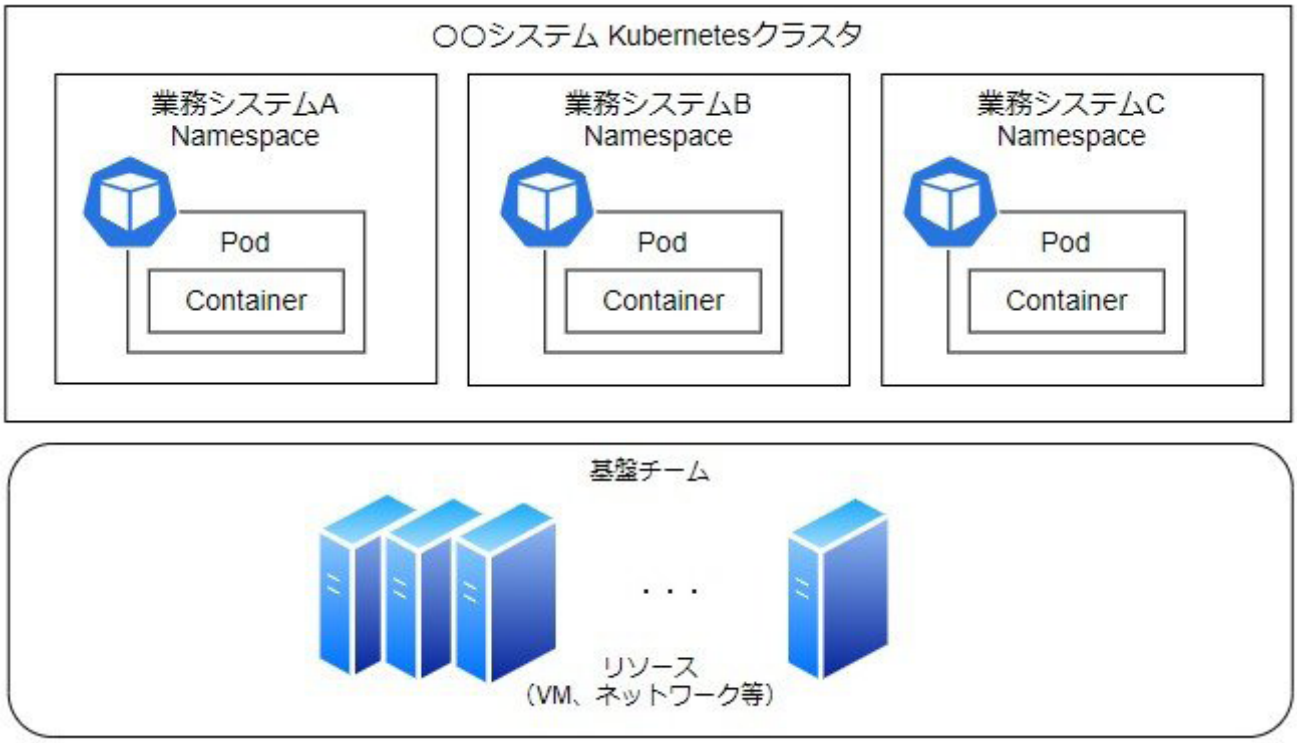
また、大規模開発においては開発者のスキルレベルもさまざまであり、コンテナに詳しいメンバーばかりとはいえない。コンテナが稼働するマシン（Node）に不要なアクセスを禁止するなど、コンテナ実行時の権限制御も不可欠だ。

ネットワークも同様に、クラスタ内で他チームが開発する API へのアクセスを禁止し、外部アクセスの際には特定のエンドポイントのみを許可するといった制御も必要だ。

従って、下記を要件としたマルチテナンシー Kubernetes を構築した (※)。

- 開発者の Kubernetes API 操作権限の制御
- Namespace ごとのコンピュートリソースの制御
- 各 Node で稼働するコンテナの権限制御
- ネットワークの通信制御

(※) これは単一のクラスタを異なる会社やクライアントに分割して提供するハードマルチテナンシーを想定したものではない。あくまで同一組織内の信頼できるチームによるソフトマルチテナンシーを前提とした。



マルチテナンシー Kubernetes の実装ポイント

開発者の Kubernetes API の操作権限の制御

Kubernetes には、「ServiceAccount」と通常の「UserAccount」の 2 種類のユーザーがある。

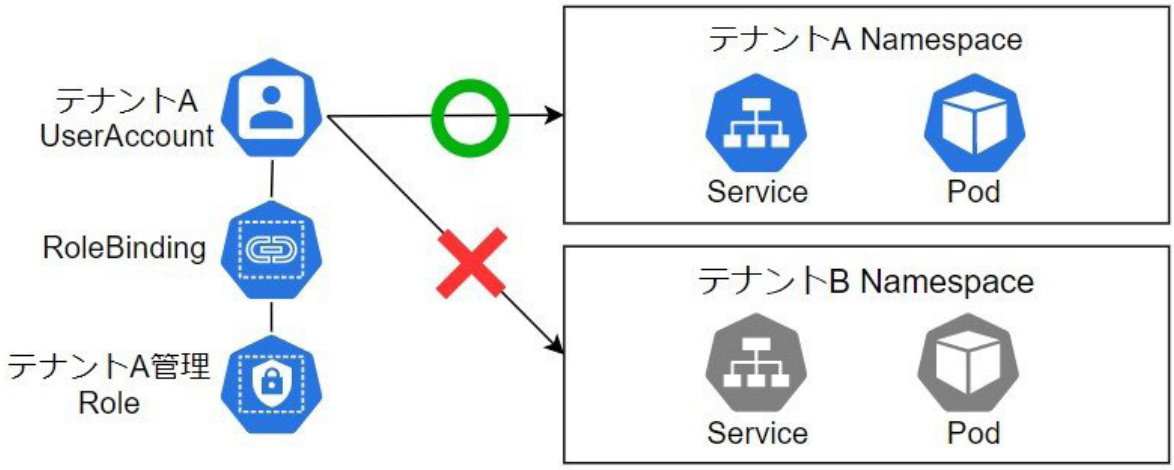
Kubernetes にアプリケーションをデプロイする場合、Kubernetes API を介して実行する。それぞれのユーザーに対して Kubernetes API のアクセス制御を実現する機能として、Kubernetes は「Role Base Access Control」（RBAC）を提供している。この機能を利用することで「どのユーザーが」「どの Kubernetes のリソースにアクセスできるか」を制御できる。マルチテナンシーの Kubernetes クラスタを構築する中で、このアクセス制御は特に重要だ。

ユーザーによる Kubernetes API のアクセス制御の実装方針として、各テナントの Namespace とテナント管理用 UserAccount は基盤チームが作成し、権限を付与した上でテナントのアプリケーション開発者に提供する方式とした。

テナント管理用 UserAccount には「どのリソースにどういった操作ができるか」を表した「Role」と、「誰に Role を与えるか」を表した「RoleBinding」の機能を用いて Namespace 内に閉じた権限が与えられている。

これで、各テナント管理者は Namespace 外のリソースや他 Namespace のリソースにアクセスできない状態となる。これにより、誤った Manifest の適用や誤操作による他テナントへの影響をなくすことができる。

ServiceAccount は Kubernetes で稼働するアプリケーションなどを表すものである。各テナントで稼働するコンテナには Kubernetes API のアクセス要件はないため、ServiceAccount へ Kubernetes API のアクセス権限は付与しないこととした。



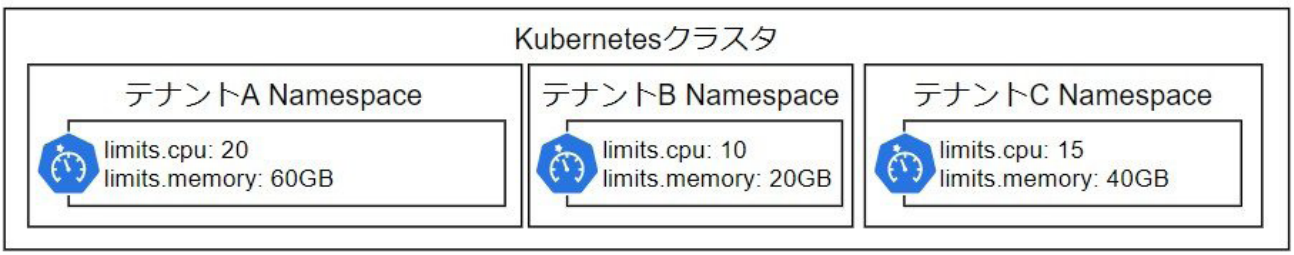
Namespace ごとのコンピュートリソースの制御

特定のアプリケーションによるコンピュートリソースの枯渇を防ぐため、Kubernetes には各 Namespace に使用可能なコンピュートリソースの制限をかけられる「ResourceQuota」という機能がある。

ResourceQuota によってテナントごとに利用可能なコンピュートリソースを制限し、指定以上のコンピュートリソースが利用されそうになった場合は Pod のスケジューリングをさせないことで、他テナントへの影響を抑えられる。

Kubernetes は、Namespace 内での Pod やコンテナへのコンピュートリソース割り当てを制御する「LimitRange」と呼ばれる機能を提供している。Pod やコンテナへのコンピュートリソース割り当ては、Deployment の Manifest のテンプレートにて設定するため LimitRange による制限は実施していない。こちらも要件と照らし合わせて利用可否を決めるとよい。

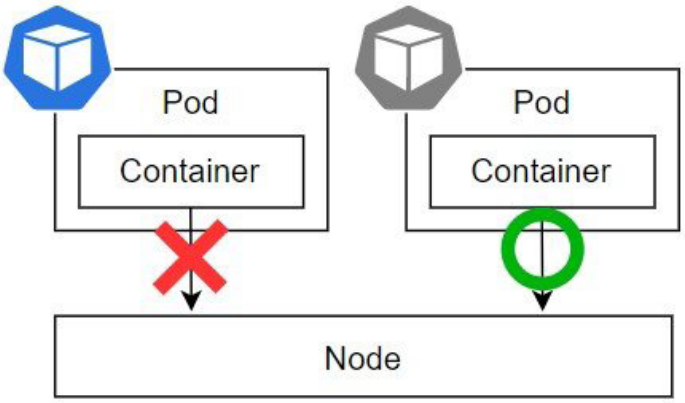
なお、コンピュートリソースを制限していないベストエフォートなコンテナがある場合、同じ Node 上で稼働する他のコンテナに影響を与える可能性がある。そのテナントで稼働するシステムの要件も踏まえ LimitRange を利用してコンピュートリソースをクラスタ管理者側で制御するか、（リソース集約の観点ではあまり推奨されないが）特定の Node にベストエフォートなコンテナが含まれる Pod を寄せてしまう方法を採ることもできる。



各 Node で稼働するコンテナの権限

セキュアな Kubernetes を構築するには、コンテナから Kubernetes の Node に不要なアクセスができないように制御する必要がある。Kubernetes には「PodSecurityPolicy」と呼ばれる機能があり、「root ユーザーでの動作を想定したコンテナの実行」や「ホストネットワークの使用」「hostPath の使用」など Node に影響を与え得るコンテナの稼働を制限できる。

実装方針として、上記に述べたような Node への影響を与える可能性のあるコンテナはデフォルトで実行できないように設定した。root 実行の許可や Node へのアクセスを許可する PodSecurityPolicy の作成と ServiceAccount へのひも付けを基盤チームが実施し、要件に応じて各アプリケーション開発チームに提供している。



ネットワークの通信制御

Kubernetes で通信を制御しない場合、A という Namespace の Pod から B という Namespace の Pod にアクセス可能だ。マルチテナンシーを考慮する場合は通信も制御すべきポイントになる。

Kubernetes には「NetworkPolicy」と呼ばれる Pod 間通信やクラスタ外部とのエンドポイント通信を制御するためのリソースが用意されている。対応した SDN プラグインを利用することで Pod の通信制御が可能だ。

各アプリケーション開発チームの API は一部を除いて互いに実行できないよう制御できること、外部エンドポイントへのアクセスも同様に制御できることが要件にあったため、NetworkPolicy を導入し通信を制御するようにした。

上記以外にも、Kubernetes の「Multi-tenancy SIG（Special Interest Group）」によって「Hierarchical Namespace Controller」といったマルチテナンシーな Kubernetes のための機能開発が進められている。

マルチテナント実現の課題

マルチテナンシー Kubernetes を実現するメリットは複数あるが、課題もある。

まず、Namespace 外のリソースをテナントにどう提供するかだ。アプリケーション開発チームには Namespace 内に閉じた権限を付与しているため、Namespace 外のリソースが作成できない。

実例として、急きょストレージが必要になった際に、アプリケーション開発チームで Namespace 外のリソースである PersistentVolume を作成できなかったため、クラスタ管理者側がリソースを作成して対処したことがある。頻度や作業量が増えた場合は細かいアクセスコントロールの検討が必要になるだろう。

また、Kubernetes の機能を拡張するための考え方として「Operator Pattern」と呼ばれるものがある。Operator Pattern では「CustomResourceDefinition」（CRD）と呼ばれるリソースを用いて、独自のリソース定義を追加するが、この CRD も Namespace 外のリソースになる。

現状、各テナントによる Operator の利用要件はないが、Operator Pattern が一般的になり、CRD の適用が増えていくと考えられる。Namespace 外のリソースをどのように提供していくかは今後検討すべきポイントになるだろう。

今回紹介した内容はソフトマルチテナンシーであったため、共通化可能な部分は集約したが、クラスタ上に構築するシステムの要件によっては個別対応が必要なケースも考えられる。どこまでを共通化し、どこから個別対応とするかの見極め、各テナントの開発者が利用しやすいようにスコープを決めることも難易度が高い。

本記事では大規模な基幹システム開発における問題を解決する手段として、NRI が採用したマルチテナンシー Kubernetes を紹介した。だが、システムの特性、要件によってはクラスタ自体を分けてしまった方が良い場合もあるだろう。

まとめ

本連載では、NRI 内に存在する組織間の壁を乗り越える一手段として、Kubernetes をはじめとするクラウドネイティブ技術の取り組みを紹介してきた。

第 2 回、第 3 回では NRI が得意とする金融分野のプロジェクトでの例、生産性向上のサービスそのものを Kubernetes 上に構築した事例を紹介した。今回は、大規模なプロジェクトにおいて、従来の仮想化技術によるプロビジョニングではなく、Kubernetes を利用したマルチテナンシー構成を採用した例を紹介した。

もちろん、こうした対応で組織に遍在する課題を全て解決したとはいえない。それでも小規模なスピード重視のプロジェクトなど、企業としての対応力を高め、「顧客とともに栄える」べく、社内での挑戦は続いている。NRI の取り組みが同様の課題に取り組み、コンテナや Kubernetes の技術採用で悩む読者の背中を押すものとなれば望外の喜びである。

筆者紹介

小林隆浩

野村総合研究所でデータベースを中心としたインフラ設計を担当。

新井雅也

野村総合研究所で金融業界を中心としたアプリケーション・インフラ設計・開発全般を担当。

Twitter：@msy78

海内映吾

野村総合研究所でシステム開発ソリューションサービス「aslead」のインフラ設計を担当。

澤頭 毅

野村総合研究所で Kubernetes 導入支援や周辺のインフラ設計を担当。

編集：@IT 編集部
発行：アイティメディア株式会社